709

Honeywell Bull

EXECUTIVE

SYSTEM 700

OS/700

SOFTWARE

Ref. : 61 A. 2 - AR 22, Rev. 0

SECTION I
INTRODUCTION

OS/700 is an operating system oriented toward satisfying real-time, multiprogramming, and data communications requirements. The system allows dynamic loading and simultaneous online execution of multiple user programs on a priority basis. It controls the operation of peripheral devices and data communications equipment, and facilitates the creation and manipulation of disk files. Language processors, utility housekeeping programs and other programs provided in the OS/700 software package can be executed online concurrently with user application programs under control of the system. This manual describes the operation and use of the OS/700 system executive, which monitors and coordinates all the online functions performed by the system. Figure 1-1 shows the relationship of the executive to the system as a whole.

FUNCTION OF THE OS/700 EXECUTIVE

The OS/700 executive is an integrated set of software routines that reside in the computer's main memory during the running of the system. The executive occupies a fixed area of memory, known as the system area, and the remaining part of memory - the user area - is free to contain user or Honeywell-supplied online programs (known as activities). Operation of the system is started by loading the executive into main memory from some external medium such as disk or magnetic tape, and initiating its execution. Once the executive is running, the operator can enter commands through the operator's console - an ASR or KSR device - to cause it to execute an activity. The activity can be already in main memory, having been loaded along with the executive, or the executive itself can load the activity into memory from disk or magnetic tape. Alternatively, the user can configure the system so that the executive automatically starts the execution of one or more activities after it has been started up.

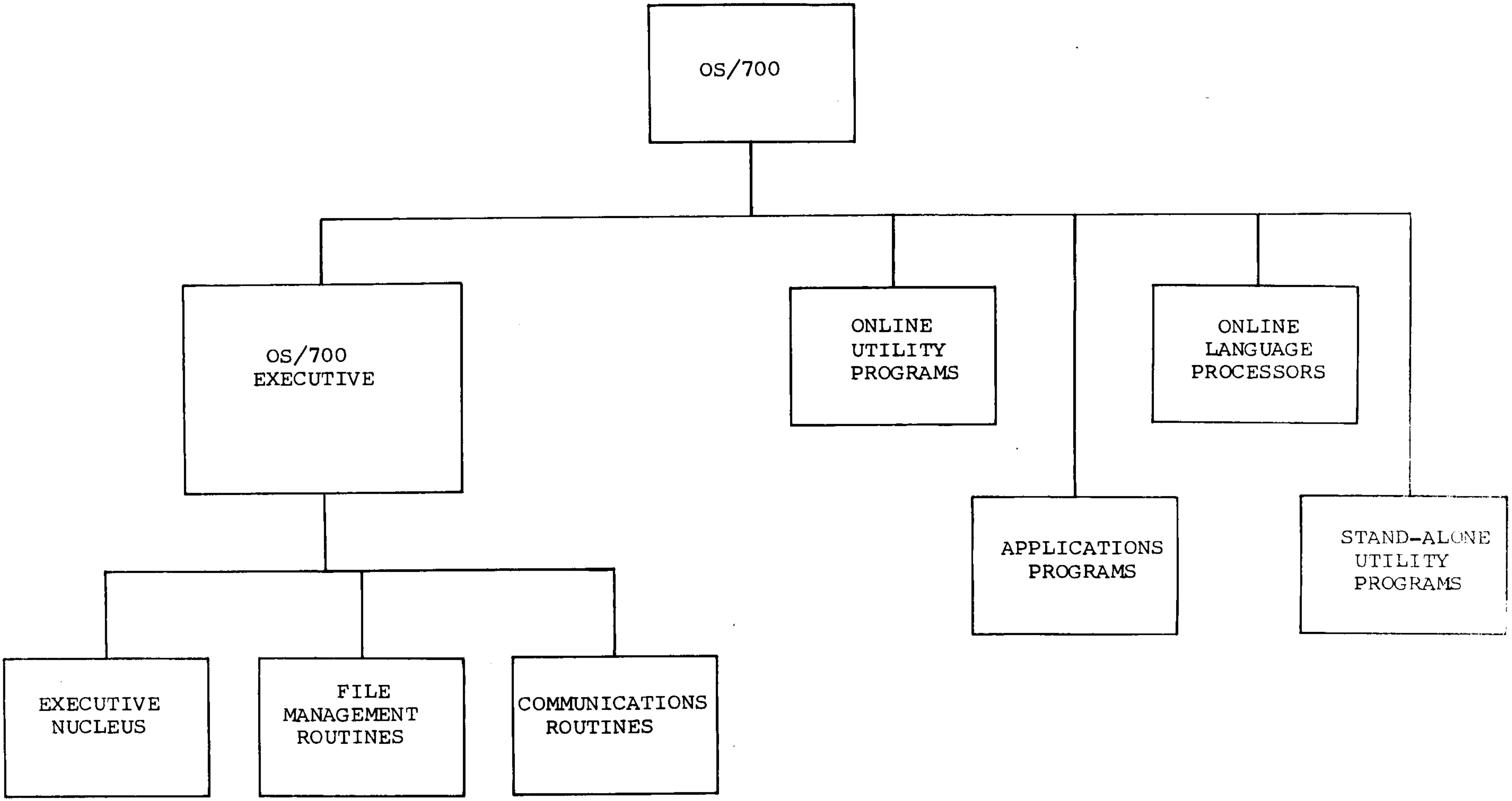


Figure 1-1. OS/700 Components

The activities can then proceed by calling on the executive to perform any of a number of individual functions, known as executive functions. An executive function may consist, for example, of inputting a card from the card reader, creating a disk file, or starting another activity. The executive continues to allocate central processor time to activities, allowing interleaved execution according to priority, while simultaneously monitoring the real-time clock and peripheral device interrupts. The executive performs all peripheral device input and output through device driver routines. The operator maintains control throughout the system's operation by communicating with the executive through the operator's console.

Executive Nucleus

As shown in Figure 1-1, the executive is for documentation purposes divided into the executive nucleus, the file management routines and the communications routines. The executive nucleus performs all executive functions except those included in the other two headings.

OPERATING SYSTEM TYPES

Systems configured under OS/700 are divided into two general but distinct types:

- Configurations which include a system disk. In this type of system, portions of the executive together with system and user activities are allowed to be disk-resident and are automatically loaded into main memory as they are required.
- Configurations which do not include a system disk. In this type of configuration, the entire executive must be main memory-resident; system and user activities can reside on external media (other than disk), but must be explicitly loaded by the operator.

Systems configured with a system disk are called Disk Operating Systems (DOS), while those systems configured without a system disk are called Core Operating Systems (COS). Note that a COS configuration can include disk storage as well as other secondary storage devices, but the disk is not used to dynamically load portions of the executive during normal system operation. Disk usage in a COS configuration is restricted to user physical I/O operations.

FACILITIES PROVIDED BY THE OS/700 EXECUTIVE

- Resource Management - The executive manages the allocation of central processor time, main memory, and I/O devices.
- File Management Capability (DOS only) - The executive provides a file management capability in addition to a physical I/O interface.
- Priority Assignment - The executive permits the user to assign priority levels to the execution of activities and individual routines (tasks).
- Foreground-Background Operation - Because of its priority assignment capability, the executive provides foreground-background operation. Normally, the foreground is used for high-priority operations; the background is used for low-priority operations.

- Clock-Initated Program Execution - The executive allows the user to execute programs at a specified time interval on a cyclic or noncyclic basis.
- Queued Execution Requests - Requests for the execution of executive functions are queued and initiated according to their assigned priorities.
- System Integrity (DOS only) - Activities can be run in restricted mode, in which they are closely monitored by the system; if they perform an illegal operation, they are terminated (aborted) by the system, which also keeps track of and returns all system resources used by these activities. Restricted activities can also be aborted by operator command.
- Systems Greater Than 32K (DOS only) - OS/700 supports up to 64K memory.
- Command Input Mode - Commands and responses normally typed on the operator's console may be read from a disk file.

SCOPE OF THIS MANUAL

This manual describes the structure and operation of the executive, and its interface with the user program. It describes the nature and purpose of all the executive nucleus functions available to the user. The manual explains how a user program can make use of the wide range of executive facilities listed in the previous subsection, with the exception of file management, communications functions and the command input facility. These are described separately and respectively in the OS/700 File Management manual, the OS/700 Communications manual, and the OS/700 Operators Guide.

Section II of this manual describes the structure and function of the system executive, and basic system principles. The operation of OS/700 centers around the concept of activities and tasks. Their definitions and relationships must be understood before the overall system operation can be grasped. These concepts are explained in Section II. This section also describes the interface between a user program and OS/700.

Section III goes on to explain in detail how the user can schedule tasks, and Section IV does the same for activities. Section V describes the management of queues and free memory. Section VI describes how the user performs physical I/O under OS/700, and Section VII describes the interface between the user program and the operator. Section VIII describes other miscellaneous functions which do not fall into any of the above categories.

The appendices list error codes and physical I/O device and other information needed by the OS/700 programmer.

SECTION II

GENERAL DESCRIPTION

SYSTEM EXECUTIVE OVERVIEW

The OS/700 executive performs a number of well-defined basic operations which monitor and control the computer hardware. These include operations such as:

- Responding to interrupts
- Driving peripheral devices
- Monitoring the real-time clock
- Allocating work areas in main memory
- Controlling the order in which routines are executed within the system.

Together, the interaction of these basic system operations controls the overall running of the system.

The user program, however, sees the system executive in terms of what it can do for him. The user program sees the executive as another collection of functions called executive functions, equally as well defined as (but in general different from) the basic executive operations. These executive functions can be called individually by a user program by means of an executive function macro. The OS/700 supplied software includes a macro library, MACLIB, which defines these executive function macros for use by a user program. Each of these executive functions is handled by an executive function action routine, and the execution of this routine normally requires several of the basic executive operations described above. The system executive itself can also call on an executive function in the same way as a user can. Each set of related action routines, such as the set which performs disk file operations, or the set which handles operator messages, can be grouped together into an individual functional component of the executive.

The system executive consists of the functional components shown in Figure 2-1. This diagram shows the two main sets of components: the basic system components and the executive function action routines.

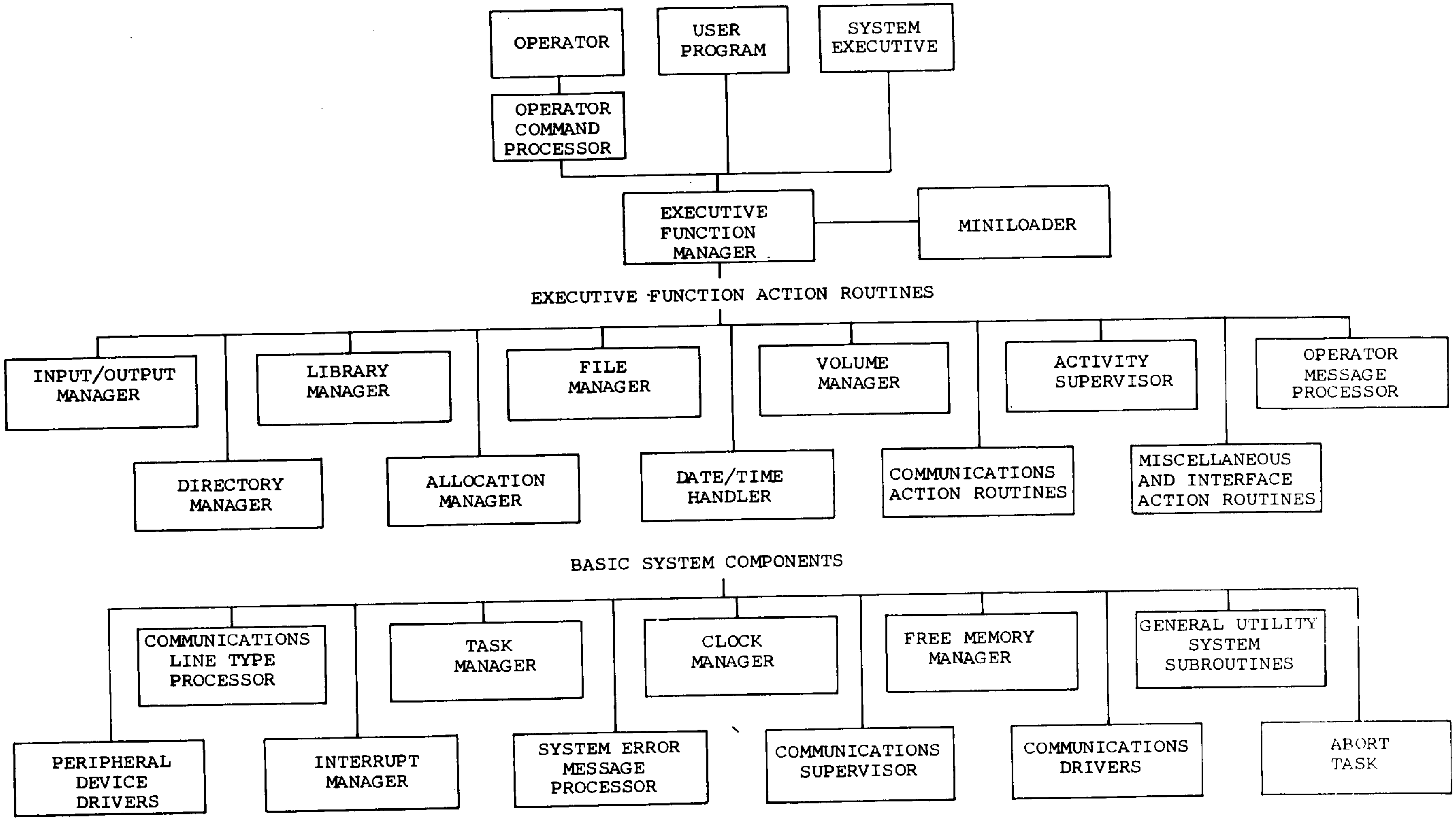

OPERATOR
USER PROGRAM
SYSTEM EXECUTIVE
OPERATOR COMMAND PROCESSOR
EXECUTIVE FUNCTION MANAGER
MINILOADER
EXECUTIVE FUNCTION ACTION ROUTINES
INPUT/OUTPUT MANAGER
LIBRARY MANAGER
FILE MANAGER
VOLUME MANAGER
ACTIVITY SUPERVISOR
OPERATOR MESSAGE PROCESSOR
DIRECTORY MANAGER
ALLOCATION MANAGER
DATE/TIME HANDLER
COMMUNICATIONS ACTION ROUTINES
MISCELLANEOUS AND INTERFACE ACTION ROUTINES
BASIC SYSTEM COMPONENTS
COMMUNICATIONS LINE TYPE PROCESSOR
TASK MANAGER
CLOCK MANAGER
FREE MEMORY MANAGER
GENERAL UTILITY SYSTEM SUBROUTINES
PERIPHERAL DEVICE DRIVERS
INTERRUPT MANAGER
SYSTEM ERROR MESSAGE PROCESSOR
COMMUNICATIONS SUPERVISOR
COMMUNICATIONS DRIVERS
ABORT TASK

Figure 2-1. OS/700 System Executive-Functional Diagram

Executive Component Interaction

The diagram makes no attempt to show the numerous control paths between the various executive components. In general the action routines make large numbers of calls on the basic executive components to set up or perform operations, whereas there is less interaction between individual action routines and individual basic components. Executive components request the execution of other executive components in one of five ways:

- By a direct subroutine call (or jump).
- By scheduling a task. This enables two routines to execute in parallel on a multiprogramming basis. (Tasks are explained later in this section.)
- By scheduling an activity. This enables a complete program such as a device driver, which may be disk-resident, to be brought into main memory for execution.
- By calling an executive function. Executive functions can also be disk-resident, and this method causes the system to perform the necessary input of overlays; also the calling routine enjoys the same conveniences that a user program has on an executive function call, such as having certain registers preserved through the call.
- By connecting an interrupt. This enables a routine to be executing when a peripheral device has completed some operation, and interrupts.

Executive Function Action Routines

Action routines receive executive function calls and parameters through a standard interface. Arguments and data for them to operate on are set up in standard memory locations when an executive function call is made. This interface is described from the user's point of view in "Executive Function Calls," later in this section.

Most of the action routine components shown in Figure 2-1 can be disk-resident in a DOS configuration, although the user can specify at configuration-time that individual routines are to be main-memory-resident to increase the speed of operation of the system. A portion of the system area of main memory – the system overlay area – is reserved in a DOS configuration for disk-resident action routines. The system reads routines into the overlay area from the disk automatically as they are required. In general, the system overlay area has room for one of the components shown in the diagram at any one time, although two or more of the smaller components can fit into an overlay, whereas one or two of the larger components are split into two or more overlays.

Executive function action routines can call on other executive functions: for example, the file manager can make a call on the allocation manager to obtain a work area on the disk.

EXECUTIVE FUNCTION MANAGER

This routine is memory-resident in every OS/700 system. It acts as the interface for all executive function calls. It performs the necessary saving of registers through calls, locates the required action routine and transfers control to it, communicating to it the caller's parameters. All action routines return to the caller through the function manager, which must set up the hardware in a suitable state for return to the caller and communicate error information returned to the caller

Since executive function action routines can call on other executive functions, the function manager must handle recursive calls on itself several levels in depth, and preserve registers through these multilevel calls.

MINILOADER

This routine loads action routines into the system overlay area in a DOS configuration. It queues concurrent requests for the use of the system overlay area according to priority.

INPUT/OUTPUT MANAGER

The I/O manager consists of routines for allocating peripheral devices and for initiating I/O operations to be carried out by peripheral device drivers.

LIBRARY MANAGER

This routine manipulates libraries (named groups of files) on disk in a DOS configuration.

FILE MANAGER

This routine performs the creation and manipulation of disk files in a DOS configuration.

VOLUME MANAGER

This routine manages the connection and disconnection of disk packs to the system in a DOS configuration and automatically sends commands to the operator for the loading and unloading of removable disk packs.

ACTIVITY SUPERVISOR

This routine manages the execution of programs in memory and performs the loading of disk-resident programs into main memory in a DOS configuration.

OPERATOR MESSAGE PROCESSOR

This routine enables a program to type a message on the operator's console and optionally receive a response from the operator.

DIRECTORY MANAGER

This routine manipulates disk directories, which in a DOS configuration record the names and characteristics of files, libraries, and disk-resident activities.

ALLOCATION MANAGER

This routine controls the sharing of disk space for various purposes by the system or the user in a DOS configuration.

DATE/TIME HANDLER

This routine, after initial setting-up, records the passage of the time of day.

COMMUNICATIONS ACTION ROUTINES

These routines enable the user to perform specific communications functions such as connecting to a station and sending and receiving data.

MISCELLANEOUS AND INTERFACE ACTION ROUTINES

This category covers a group of functions, such as scheduling a task, performed by the basic system components, which cannot be called directly by a user program. Interface action routines translate data presented by the user in the executive function call into a suitable form for a direct call on the basic components. It also covers assorted minor functions such as queue handling and the supply of information about the system configuration (e.g., which disk is the system disk).

Executive Function Requests

Executive function requests through the function manager, as shown in the diagram, come from three sources:

- The operator. The operator command processor routine enables the operator to type in basic requests, such as a request to start an activity, through the console.
- A user program.
- The system executive itself.

Basic System Components

These components in general can be accessed by the user only through action routines. They are always memory-resident, and must all be present in an OS/700 system (except for the communications components, which are optional). Calls to these components, from action routines or from each other, are usually in the form of subroutine calls.

INPUT/OUTPUT MANAGER

The I/O manager consists of routines to initiate I/O operations to be carried out by peripheral drivers and routines for allocating peripheral devices.

PERIPHERAL DEVICE DRIVERS

These are the routines that perform the actual hardware instructions to drive peripheral devices. There is a different driver for each different type of device. Drivers can be memory-resident, in which case they reside with the system executive, or in a DOS configuration they can be in the form of disk-resident activities which are brought into the user area of main memory as required. The driver for the operator's console and (for obvious reasons) the system disk driver in a DOS configuration must, however, be memory-resident.

INTERRUPT MANAGER

This routine saves the state of the computer when an interrupt occurs and passes control to the appropriate interrupt response routine.

TASK MANAGER

The most basic part of every OS/700 system, this routine controls the execution of all routines that are not interrupt-initiated. Routines are scheduled through the task manager, which causes them to be executed according to priority.

SYSTEM ERROR MESSAGE PROCESSOR

This routine enables the system to output error messages to the operator's console.

CLOCK MANAGER

This routine responds to interrupts made regularly by the real-time clock. Routines can be scheduled for execution at a specific interval or after a specified delay, and the clock manager sees that these are dispatched at the appropriate time. For example, if a peripheral device has not responded to a driver command after a reasonable period of time, the clock manager will call a driver routine to check for an error situation.

FREE MEMORY MANAGER

This routine controls the allocation of data areas in main memory.

COMMUNICATIONS SUPERVISOR

This routine oversees the operation of communications devices.

COMMUNICATIONS DRIVERS

These routines perform the necessary hardware manipulations for the sending and receiving of data through a communications controller.

COMMUNICATIONS LINE TYPE PROCESSORS

These routines cause data to be exchanged on communications links according to particular protocols.

GENERAL SYSTEM UTILITY SUBROUTINES

These subroutines perform utility operations such as the saving of registers and the manipulation of queues.

ABORT TASK

This routine terminates the operation of certain types of activities, known as restricted activites. These activities can be aborted if they malfunction, if the system is overloaded, or on a command from the operator. The abort task is responsible for cleaning up and returning all system resources used by these activities.

Operator Interface Processor (OIP)

This name is given to the collection of components comprising the operator command processor, the operator message processor, and the system error message processor. OIP controls all messages passed between the system and the operator's console.

FREE MEMORY

Certain areas of main memory in an OS/700 system are divided into free memory blocks of various sizes: all block sizes are powers of 2, ranging from 4 to 4096 words. The available sizes and the number of blocks of each size are defined by the user at the time the system is configured. These blocks are used as data space in reentrant system routines. The Free Memory Manager, a routine within the OS/700 executive, controls the allocation of these blocks. A routine can call upon the Free Memory Manager to request the allocation of a block of a particular size: it is the responsibility of the routine using the block to return the block to the appropriate pool of available blocks when it has finished with it. User programs can also use free memory blocks; executive function macros are provided to enable the user to call upon the Free Memory Manager to fetch or return a block. The user is recommended to use free memory blocks only when necessary; for example, in reentrant coding or to pass parameters from one user program to another. When a user program requires data space for an I/O buffer, for example, it is simpler to use an area of memory within the user program where this can reasonably be done instead of fetching a free memory block; this course of action also ensures the maximum supply of free memory blocks for the use of the system.

TASKS

On a 716 computer, program instructions are executed one at a time, in sequence. The sequence is controlled by the instructions themselves, and will normally include jumps and loops, but once execution of a set of instructions has begun, it will continue until an interrupt occurs or the computer is halted. An interrupt causes a new sequence of instructions to be started.

The online execution of program instructions (code) under OS/700 is controlled by a system routine known as the dispatcher. Program instructions executed under OS/700 fall into two groups: interrupt response code and task code. A set of interrupt response code is a sequence of instructions whose execution is started when a hardware interrupt occurs. A set of task code is a sequence of instructions whose execution is started by the dispatcher, upon request from a user or from the system itself. A task is a single instance of execution of a set of task code. The term "task" is also used to denote the set of task code itself.

Task code is contained both within the system and within user programs, whereas interrupt response code is contained only within the system, since it is the responsibility of the system to handle interrupts from peripheral devices and other sources. A user program (known as an activity) contains at least one task, and can contain several. All tasks and interrupt response code terminate ultimately by jumping to the dispatcher, which then decides which task to execute next. An executive function macro is provided to enable a user task to terminate itself.

Task Scheduling

The act of requesting a task to be executed is known as scheduling the task. Task scheduling and dispatching is performed by a system component known as the task manager (of which the dispatcher itself is a part). A task may be scheduled by interrupt response code or by another task (either a user task or a system task. Scheduling is performed by making a call upon the task manager; executive functions are provided to enable a user program to do this.

When scheduling a task, the caller assigns a priority level which indicates the relative importance of its execution. OS/700 allows the specification of up to 16 priority levels - three system levels for system tasks only, and a maximum of 13 user levels for either user tasks or system tasks. The priority level of a task is indicated by a number (0 through 15); the lower the number, the higher the priority level of the task.

The system keeps a separate queue of tasks scheduled for each priority level. When the user schedules a task, the task manager places the request at the end of the scheduling queue for the appropriate level.

Task Dispatching

A task is dispatched when the dispatcher removes it from the scheduling queue and initiates execution of the task code. A task is resumed when the dispatcher continues execution of the task code after it has been suspended; e.g., by an interrupt. Tasks are dispatched and resumed on the basis of priority level. A task or interrupt response code terminates by jumping to the dispatcher; this routine then dispatches previously scheduled tasks, or resumes previously suspended tasks, according to their priority.

A task must run to completion; i.e., jump to the dispatcher, before another task can be executed on the same priority level. A task which is suspended on a particular level, therefore, will be resumed by the dispatcher and allowed to run to completion before any other task scheduled on that level is dispatched.

Whenever control is returned to the dispatcher, after a task has been terminated or an interrupt serviced, either a new task is dispatched, or a previously interrupted task is resumed. If all tasks have been terminated and no scheduling requests are in the queues, the dispatcher enters an idle loop, waiting for an interrupt.

Task Suspension

A task becomes suspended when its operation is temporarily interrupted; e.g., by an interrupt from a peripheral device. When a task is suspended, the system saves the registers and the current state of the hardware, so that the task can be resumed in exactly the state it had prior to the suspension. When a task at a particular priority level is suspended, no other task at that level can be dispatched until the suspended task has been resumed, and has terminated. Tasks at other priority levels may, however, be dispatched or resumed.

A task may also suspend itself voluntarily, by calling on the task manager; the dispatcher is tehn entered, and dispatches or resumes the highest-priority task of those waiting to be executed. If no tasks of higher priority than the voluntarily suspended task are waiting, the dispatcher will execute a task of lower priority, if there is one. If no other tasks are waiting to be executed, the dispatcher will resume the voluntarily suspended task. Otherwise, the suspended task will be resumed later when the dispatcher is entered at a time when no higher-priority tasks are waiting.

An executive function is provided to enable a user task to suspend itself.

Task Control Block (TCB)

A task control block (TCB) is an 8-word memory block containing information about a task to be scheduled. It is generated by the routine which schedules the task. The TCB performs two functions. It defines to the task manager the characteristics of the task – the address at which execution is to begin, the priority level, and the user activity (if any) of which the task forms a part. The TCB is placed on the priority level queue when the task is scheduled. Once the task has been dispatched, the contents of the TCB are of no further interest to the task manager. The second function of the TCB is to enable the scheduling routine to pass parameters to the task. When a task is dispatched, the address of its TCB is in the X-register; the task can thus access information placed in the TCB by the scheduling routine.

Multitasking

A simple example of multitasking is shown in Figure 2-2 to illustrate the operations of task scheduling and dispatching for the user who is unfamiliar with these concepts. The solid line marked with arrows on the diagram shows the actual sequence in which computer instructions are executed; broken lines indicate where an interrupt has caused the sequence of instruction execution to be broken off and a new sequence to be started. The entire sequence can be followed continuously from point A to point F.

Initially, the computer is executing a small loop of instructions in the dispatcher at point A; there is no task currently suspended or scheduled, and it is in the idle state. An interrupt then occurs. This causes the interrupt response routine I_1 to be executed. At some stage, the routine I_1 schedules a task T_1 at priority level 4. (Calls to the TCB manager to schedule tasks are not shown on the diagram.) Routine I_1 terminates by going to the dispatcher, which then finds that task T_1 is scheduled for execution, and dispatches it. Execution of T_1 continues until a second interrupt occurs, at point B; this causes the interrupt response routine I_2 to be entered. I_2 saves the hardware state and registers at the point where execution of T_1 as broken off, so that it can be continued later. I_2 then schedules a second task, T_2,

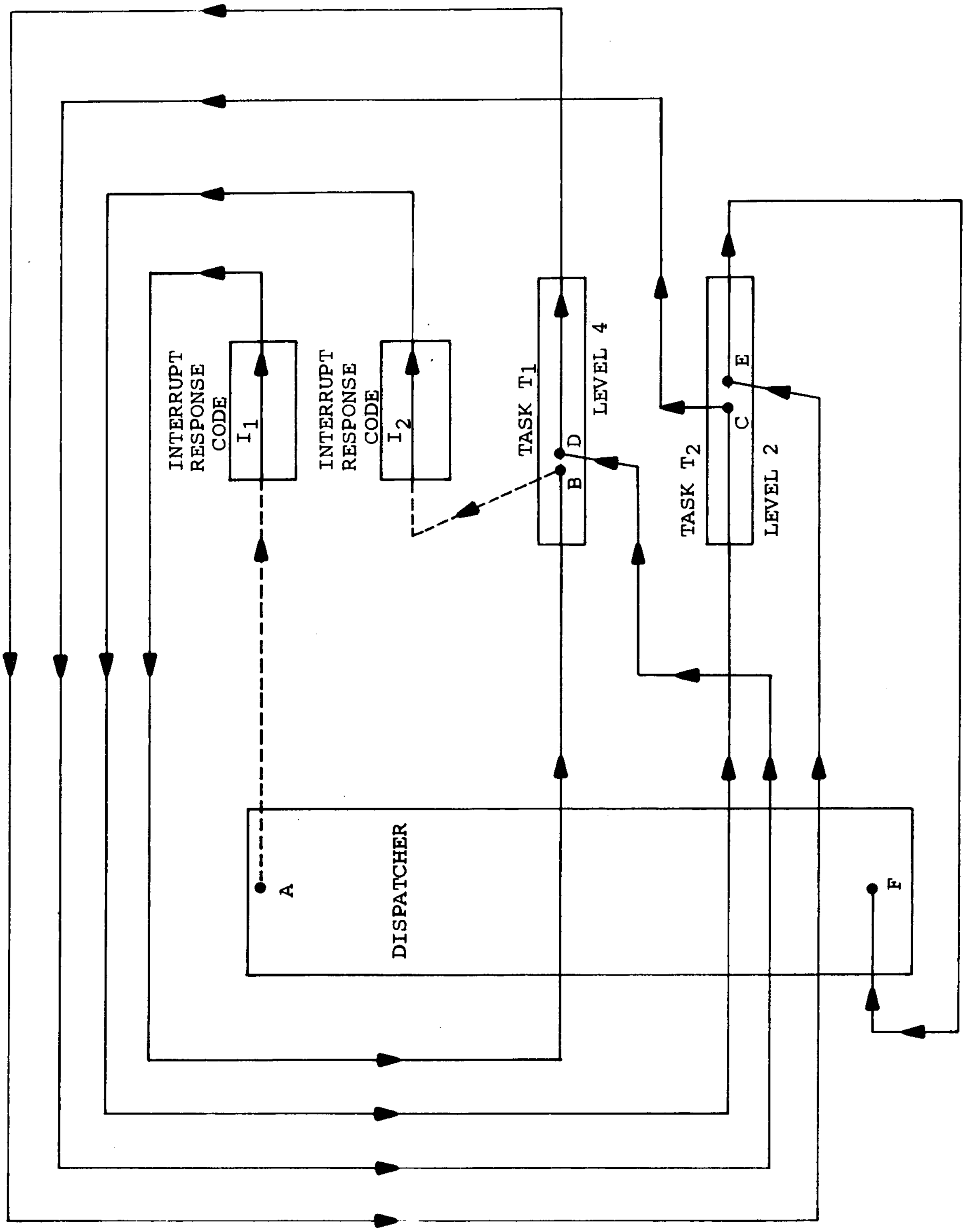


Figure 2-2. Multitasking

at level 2, and utimately returns to the dispatcher. The dispatcher now finds two tasks waiting for execution: T_1 at level 4, suspended, and T_2, newly scheduled at level 2. T_2, being at the higher priority level, is dispatched. At some point C, T_2 finds that it cannot continue because it must wait for task T_1 to complete some operation. T_2 therefore voluntarily suspends itself by calling on the task manager, to allow T_1 to run. The dispatcher, finding no other task waiting except for T_1, restores the hardware and registers to the state they were in at point B, and resumes T_1 at its next instruction D. T_1 finally terminates, at which time the only task waiting for execution is the suspended T_2; so the dispatcher restores T_2's hardware state and resumes its execution at point E. Finally T_2 terminates and enters the dispatcher, which, finding no other work to do, idles at point F. (This is the same instruction loop as was being executed at point A.)

This is a very simple example. In practice, the operations that occur under OS/700 are much more complex: many more priority levels can be involved, tasks can schedule other tasks, and interrupt response code can itself be interrupted by a privileged interrupt (e.g., power failure). However, the example does serve to illustrate that at some point such as C or D in the diagram, the tasks T_1 and T_2 can be seen from an oeverall viewpoint as being executed concurrently (multitasking), whereas only one continuous execution path is in fact followed, and at any one instant of time only one task, interrupt response routine, or executive routine is being executed. It is necessary to point out that the tasks T_1 and T_2 could in fact be the same set of instructions being in the course of execution at the same time at two different priority levels. A set of task code can be scheduled any number of times at any number of different priority levels, or at the same priority level. It can also be in the course of execution at several different priority levels concurrently, though not more than once at the same priority level, since only one task can be running on each priority level at any one time. Task code which can be executed in this way is said to be reentrant.

Task Code Classifications

Task code can be classified as reentrant, reusable or nonreusable. Reentrant code can be executed at two or more priority levels concurrently. Reusable code can be executed at only one priority level at any given time; however, it can be executed without the need to reload it into memory after each execution. Nonreusable code must be reloaded into memory each time it is executed.

The classification of a set of task code depends on how the code itself operates. If a set of task code contains a number of memory locations which must be set to some initial value in order for the code to operate correctly, and if the routine changes these values in the course of its execution and does not reinitialize them upon recommencement of the task, then the task code must be reloaded into memory before each execution to initialize the values. Such task code is nonreusable, and also (clearly) nonreentrant. Routines which overwrite their own instructions by using the locations for data space are also nonreusable (unless the overwritten instructions are executed only on the first dispatching of the task).

Task code that reinitializes itself as necessary for each execution is reusable. Task code is reentrant if it operates properly even when one task executing within this code is suspended and another task enters the code. To guarantee proper operation under these conditions, each task must store volatile data (data which varies from one task to another) in a separate area. That is, each instance of execution of the task code requires a distinct memory area for its volatile data. This area may be provided by the routine which scheduled the task. For instance, a single reentrant body of task code might be used for logging errors on a console. Each task which scheduled it would provide its own area for control blocks, parameter lists, and buffers. Alternatively, the logging task may obtain a data area for each request from the system free memory pools using the Get Storage Block (GBL$) executive function, and return it using the Return Storage Block (RBL$) function.

The hardware registers A, B, X, and S are equivalent to such data areas because the system maintains for each priority level, a separate area in which the registers are saved whenever an interrupt occurs. When the task is resumed, the dispatcher reloads the registers with the saved values. This property of the registers is what makes reentrant code possible. Even tasks that allocate memory areas for volatile data must make use of the registers to be reentrant, since the address of a volatile data area is itself volatile data, and must be stored somewhere. The registers, which have fixed locations, do not present this problem. To make reentrant coding easier, six pseudo-registers, ZCR1 through ZCR6, are provided. These are fixed memory locations which are also saved and restored by the system, and are therefore equivalent to the hardware registers for the purpose of reentrant coding.

A task is nonreentrant if it keeps data in any other fixed memory locations at a time when it can be interrupted. Sometimes, typically in a subroutine, it is convenient to inhibit interrupts for a short time rather than use the data area provided for reentrancy. While inhibited, local memory areas can be used. The maximum length of time during which interrupts may be inhibited depends upon the system's interrupt response time requirements, but, to be safe, should not exceed 100 microseconds. The return address deposited by a JST instruction is volatile data, and must be saved in a register or special data area unless interrupts are inhibited for the duration of the subroutine. The execution of a JST instruction delays controlled (nonprivileged) interrupts for the next two instructions. During this time, the subroutine may save the return address or inhibit interrupts to ensure reentrancy.

Sometimes several tasks (executing within the same reentrant task code or in separate bodies of code) must access or alter shared volatile data (such as a counter) or a shared volatile data structure (such as a queue). To ensure the integrity of the data, any operation performed must continue to completion before another such operation can start. If the operation can be performed in one instruction, no special precautions are necessary. For instance, a counter can be incremented by an IRS instruction. If the operation requires a small number of instructions (e.g., three instructions to decrement a counter: LDA, SUB, STA), those instructions should be executed with interrupts inhibited. Longer operations such as reading and updating a disk-resident data base require a mechanism such as a request queue. Tasks wishing to access the data place entries representing their requested operations on a queue, and a single service task removes the entries and performs the operations one by one.

Similar considerations of reusability and reentrancy apply to complete user programs (activities), and are discussed later.

ACTIVITIES

Programs executed under the control of OS/700 are referred to as activites. An activity is a set of one or more tasks whose task code shares a number of common attributes and occupies a common, continuous area of main memory — an activity area — during their execution. Activity areas are all located in a continuous area of main memory called the user area. Figure 2-3 illustrates a typical memory layout for OS/700. An activity is identified by a name of one to six alphanumeric characters, the first character of which must be a letter.

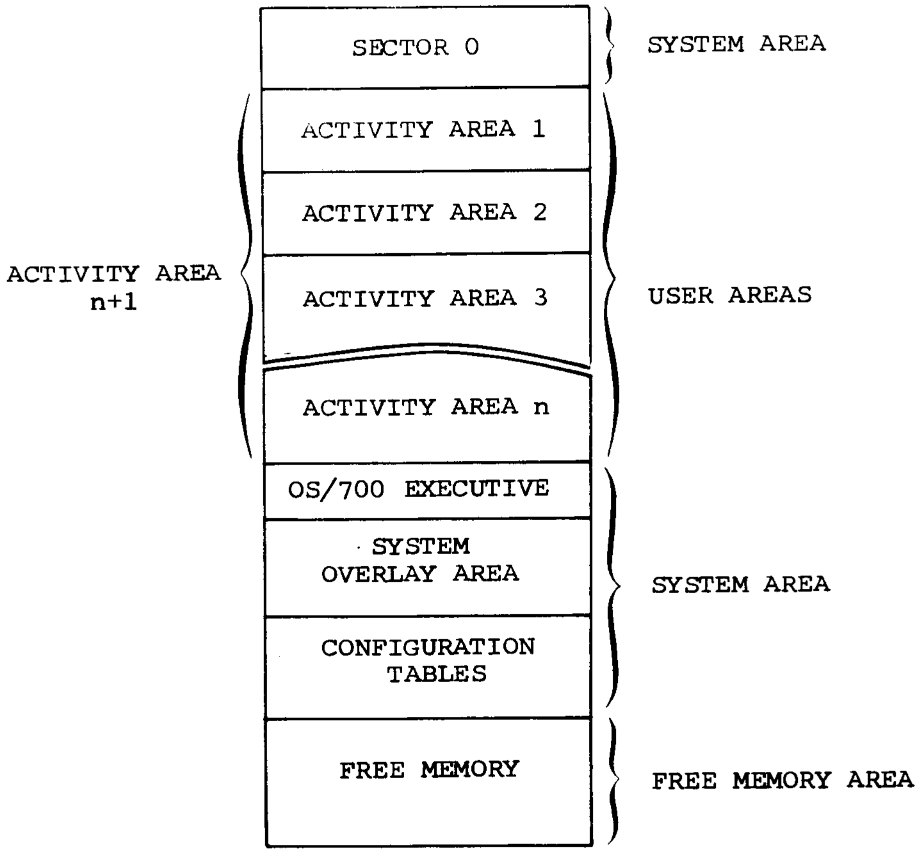


Figure 2-3. Sample Memory Layout for OS/700 (Minimum System – DOS Configuration)

Activity Residency

Activity code may or may not be main-memory-resident. Main-memory-resident activities are specified when the system is configured and are always loaded into main memory when the system is initialized. There are two categories of main-memory-resident activities:

- Permanent - This category of activity remains in main memory throughout system operation.
- Temporary - This category of activity is present when the system is initialized, but can be overlaid during system operation.

Activites that are not main-memory-resident are of two types: external and disk-resident. External activities (COS only) can reside on any input device, and are loaded into main memory as the result of an explicit operator request. Disk-resident activities (DOS only) are stored on the system disk and brought into main memory only when their execution is required. The execution of a disk-resident activity can be requested explicitly by the operator, by an executive function call from a user program, or by the system itself, and the loading and execution of the activity will then be performed automatically by the system. The user can create new disk-resident activities, or delete old ones, on the system disk at any time during the running of the system.

Activity Areas

The user area of main memory (see Figure 2-3) is divided into activity areas. There may be any number of activity areas (subject to the amount of main memory available) for permanent main-memory-resident activities. In a DOS configuration there may be, in addition, up to 17 activity areas for disk-resident activities, and these may overlap. Permanent main-memory-resident activities must reside in non-overlapping activity areas; these areas cannot be used for the loading of new activities. In a COS configuration, no activity areas can overlap.

A disk-resident or external activity is always loaded into the activity area for which it is linked. More than one activity can be assigned to a single activity area (as long as the activity area is not reserved for a permanent main-memory-resident activity); however, an activity cannot be loaded into an activity area if the area, or any portion of it, is currently being used by another activity. This means that in a COS configuration an external activity can be loaded into memory only when the activity area for which it is linked is free; and in a DOS configuration, where disk-resident activity areas can overlap, a disk-resident activity can be loaded into memory only if no portion of the activity area for which it is linked is in use.

ACTIVITY AREA QUEUES

The system maintains a queue for each defined activity area. If a requested activity requires part or all of an activity area which is currently in use, the requested activity is placed in the queue for that activity area. When the currently executing activity terminates, the system loads the next activity queued for that area.

Activity Management

The execution of activities under OS/700 is controlled by a system executive component known as the activity supervisor. Activities are requested by making a call on the activity supervisor to schedule the activity. An executive function is provided to enable a user program to do this.

Scheduling an activity causes two distinct functions to be performed: (1) if the activity is disk-resident, it is loaded into main memory (with appropriate queuing procedures if the activity area is already in use); and (2) execution of the activity is started. An activity may contain more than one task, but every activity has a lead task, which is scheduled by the activity supervisor to start execution of the activity. Other tasks within the activity must be scheduled by the lead task, by other activities, or by the system itself.

An activity can also schedule another activity without having the activity supervisor schedule the lead task automatically. In this case, the activity supervisor simply loads the scheduled activity into main memory; the scheduling activity can then schedule any task in the second activity, and monitor its execution.

When execution of an activity is finished, the activity is terminated by calling on the activity supervisor. An executive function is provided to allow the user to do this. An activity may terminate itself (in which case the call on the executive also terminates the task which performed the call); alternatively, an activity monitoring another activity may terminate the second activity while the monitoring activity itself continues to run.

Activity Code Classifications

Activities, like tasks, can be classified as reentrant, reusable, or nonreusable, depending upon the type of task code they contain and the way in which the tasks interact. The significance of these activity classifications is the same as that of the corresponding task classifications. A reentrant activity may be dispatched at any time, even if it is already executing, but a reusable activity must not be dispatched a second time until it has finished executing. A nonreusable activity must be reloaded into memory every time it is executed.

If the lead task of an activity — the first task to be dispatched — is nonreentrant, the entire activity is nonreentrant. If the lead task is nonreusable, the entire activity is nonreusable. However, the classification of the activity as a whole is further defined by the nature of the other tasks (if any) within the activity. If all the tasks within an activity are reentrant, the activity as a whole is reentrant. If the lead task is reentrant, but the activity contains some nonreentrant tasks, the activity as a whole can still be reeentrant if it operates in such a way that the nonreentrant tasks are protected from being scheduled on two or more levels concurrently. Similarly, if the lead task is reusable or reentrant, but the activity contains some nonreusable tasks, the activity as a whole can still be reusable or reentrant if it operates in such a way that the nonreusable tasks are executed only once each time the activity is brought into memory.

An important difference from the user's point of view between reusability or reentrancy in tasks and in activities is that the system is aware of the characteristics of an activity, but not those of a task. A task is not defined to the system until it is scheduled. At this time the scheduling routine specifies to the TCB manager the priority level of the task and the address at which execution is to start; these, together with the instructions that compose the task and any parameters passed, define the task, and the TCB manager knows nothing of the reentrancy or reusability of the task. Consequently, it is the responsibility of the scheduling routine to ensure that a nonreentrant task has finished executing, or that a nonreusable task has been loaded into memory correctly, before it schedules the task.

An activity, on the other hand, is defined to the system when the activity is created. The user gives the activity a name, and states to the system which activity area it occupies and other characteristics. The system knows which activities are resident in memory, and which are actually running, at any one time. When an activity is scheduled, the activity supervisor protects it from improper usage by queuing the request until it knows that the activity is ready to be executed.

Activity Control Block (ACB)

Every activity currently resident in main memory has an Activity Control Block (ACB), which is a 16-word block containing information about the activity and its current status. The ACB is generated by the activity supervisor in a free memory block at the time the activity is first scheduled (unlike a TCB, which must be generated by the scheduling routine and presented to the task manager). The ACB remains in existence until the activity ceases to be memory-resident; it is returned to free memory by the activity supervisor when the activity terminates (if it is nonreusable), or when it is overlaid by another activity (if it is reusable or reentrant).

The user must specify whether an activity is reentrant, reusable, or nonreusable when it is defined to the system. This specification allows the activity supervisor to schedule activities as follows:

- If a nonreusable activity, currently being executed, is scheduled again, the activity supervisor reloads the activity into main memory after its execution is terminated, and then reschedules the activity (DOS only).
- If a reentrant activity, currently being executed, is scheduled again, the activity supervisor immediately places the scheduling request into the appropriate scheduling queue. Concurrent executions of the same activity, or any task within it, are possible.
- If a reusable activity, currently being executed, is scheduled again, the activity supervisor delays further scheduling of the activity until the current execution is terminated. If the activity has been terminated, and is still in main memory (has not been overlaid by another activity), the activity supervisor is able to dispatch the lead task of the activity at any time. If the activity is not in memory, the activity supervisor must reload the activity from disk and then schedule it (DOS only).

If two activities must reside in main memory at the same time, the programmer must ensure that they are assigned to separate, nonoverlapping activity areas. He must configure main memory with regard to the activities that are to be loaded. Note that the entire activity is loaded into memory, not the currently executing task only.

Activity Generation

The way in which an activity is generated depends upon whether the activity is permanently memory-resident, disk-resident, or external.

A permanent or temporary memory-resident activity in a DOS or COS configuration is loaded into memory along with the system at the time the system is built; the user must define the name and characteristics of the activity when the system is configured, by means of a configuration macro provided for the purpose. The macro generates a permanent ACB containing the activity name and characteristics.

A disk-resident activity in a DOS is defined and created by using the online utility program Load Activity, either when the system disk is built, or later when the system is running. The Load Activity utility allows the user to define the name and characteristics of the activity, and translates the link text directly into a memory image of the activity on the disk. The utility places the activity name and characteristics into a disk directory for later reference by the system.

An external activity for use by a COS is generated by using the Activity Memory Image Text Generator stand-alone utility program. This program translates the link text of the activity into a memory image of the activity on the external medium, e.g., magnetic tape, and precedes this image with a record defining the activity name and characteristics supplied by the user.

Activity Scheduling

Any task may schedule an activity. Activities are generally scheduled in any of the following circumstances:

- Automatically by the system at system initialization time (if specified when the system was configured).
- By the system in response to an operator command input through the console. (This is the way in which most online operations under OS/700 are initiated.)
- By the system in response to a command in a system command file read from the disk. (The operator initiates the reading of the command file.)
- By an executive function call from a user program.
- By the system on behalf of a user program. (Disk-resident peripheral device drivers are activities scheduled by the system automatically when a user program requests I/O to these devices).

When a user schedules an activity, the activity supervisor performs one or more of the following basic operations:

- If a reentrant activity is already in main memory, the activity supervisor schedules it immediately.
- If a reusable activity is already in main memory, and has been terminated, the activity supervisor schedules it immediately.
- If the activity is already in main memory, is currently being executed, and is reusable, the activity supervisor reschedules the activity as soon as its current execution is terminated.
- If the activity is nonreusable or not in memory and if the activity area into which it is to be load is not currently being used, the activity supervisor loads the activity into memory and schedules it (DOS only).
- If an activity must be loaded into memory and part or all of the activity area assigned to the activity is currently being used by another activity, the activity supervisor delays loading the activity until the entire activity area becomes available (DOS only).

The logical progression of these operations is illustrated in Figure 2-4.

Operation of Activities

The system keeps track of the activity currently executing. When the system schedules the lead task of an activity, it generates from free memory a TCB, known as the primary TCB. It is the responsibility of the activity to return this TCB to free memory; it can do this explicitly, or it can specify that the TCB is to be returned by the system when the activity, or any task within the activity, is terminated. When the system dispatches the lead task, it records the identity of the activity associated with the task. This enables the system to set up the correct hardware state, such as the relocated base sector, when resuming the task after suspension, or returning to the task after it has made a call on a system function. If the task schedules another task within the activity, the system can thus record that the second task, and all other tasks scheduled, are also associated with the activity.

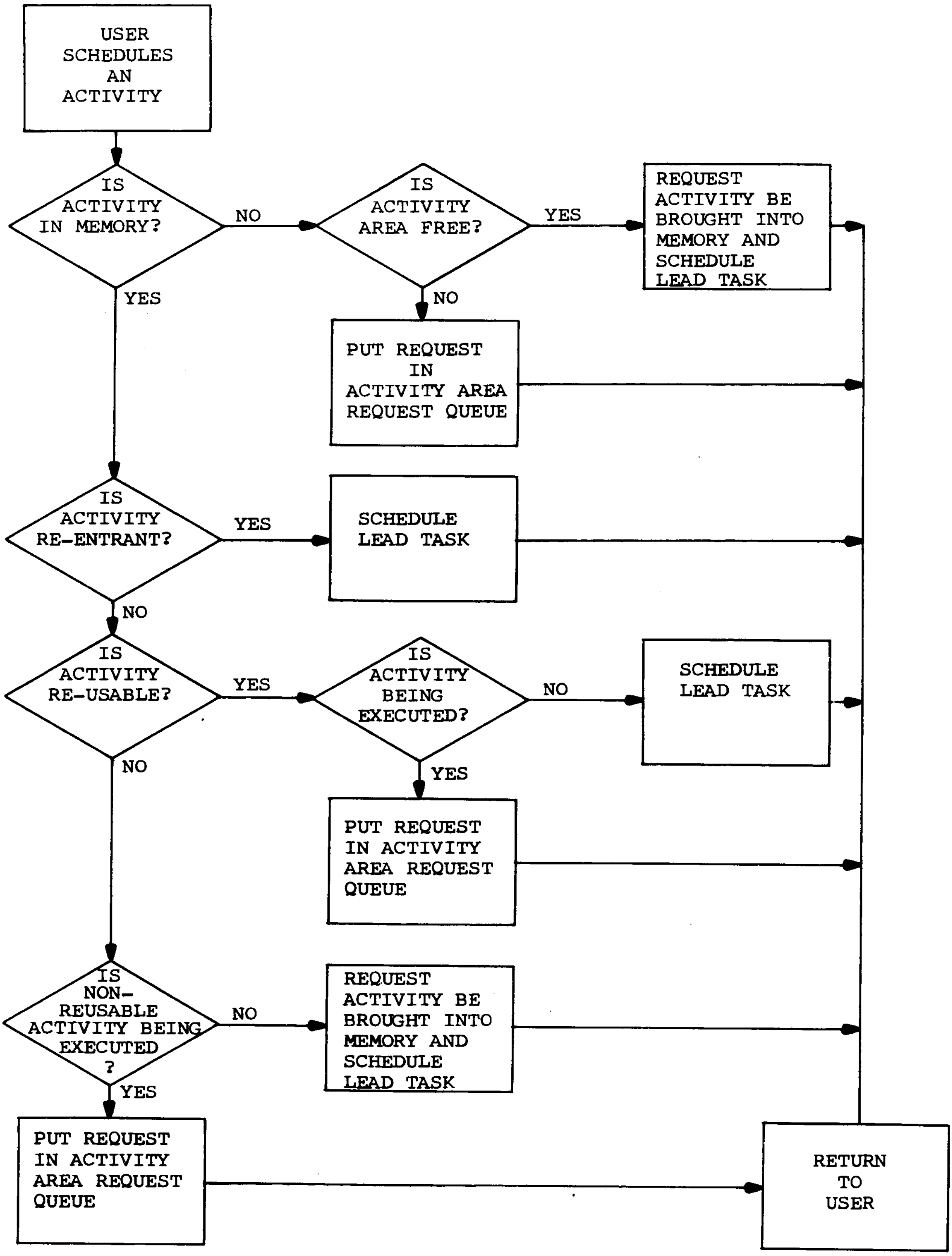

USER SCHEDULES AN ACTIVITY
IS ACTIVITY IN MEMORY?
NO
IS ACTIVITY AREA FREE?
YES
REQUEST ACTIVITY BE BROUGHT INTO MEMORY AND SCHEDULE LEAD TASK
YES
NO
PUT REQUEST IN ACTIVITY AREA REQUEST QUEUE
IS ACTIVITY RE-ENTRANT?
YES
SCHEDULE LEAD TASK
NO
IS ACTIVITY RE-USABLE?
YES
IS ACTIVITY BEING EXECUTED?
NO
SCHEDULE LEAD TASK
YES
PUT REQUEST IN ACTIVITY AREA REQUEST QUEUE
NO
IS NON-REUSABLE ACTIVITY BEING EXECUTED ?
NO
REQUEST ACTIVITY BE BROUGHT INTO MEMORY AND SCHEDULE LEAD TASK
YES
PUT REQUEST IN ACTIVITY AREA REQUEST QUEUE
RETURN TO USER

Figure 2-4. Logical Progression of Activity Scheduling (DOS)

When an activity schedules a second activity and wishes to monitor its running, the system schedules a secondary task within the monitoring activity when the second activity is ready to be executed. This method is known as scheduling an activity with a secondary TCB. When the monitoring activity schedules the lead task (or any other task) within the second activity, it must specify to the system the identity of the activity associated with the task, so that the system can record the fact that the task being scheduled is associated with the monitored, and not the monitoring, activity.

The system provides a standard technique to enable parameters to be passed from one activity to another. An activity can generate a parameter block (which may or may not reside in free memory), and specify the block at the time it schedules a second activity. The system then places the address of the parameter block in the scheduled activity's primary TCB, thus enabling it to access the parameters.

Restricted Activities

Activities may be either restricted or nonrestricted; this characteristic is defined by the user at the time he creates the activity using the Load Activity utility. The system subjects restricted activities to more stringent control. A restricted activity can be run only on a DOS configuration with the system integrity option, on a computer with the memory lockout option, whereas nonrestricted activities can be run on any OS/700 configuration (DOS or COS) on any suitable 716 computer.

Restricted activities are supervised by the system in the following manner:

- They run in restricted mode (see the System 700 Programmers' Reference Manual), and so cannot perform I/O instructions, inhibit interrupts, or do any other hardware operation which could interfere with the proper running of the system.
- They run with the memory lockout mask set up (see the System 700 Programmers' Reference Manual) so that they cannot overwrite any memory locations except those in their own activity area; thus the system and other activities are protected from being corrupted by improper operation of a restricted activity.
- Executive function calls made by a restricted activity, and the parameters of those calls, are rigorously checked by the system for legality, to ensure that the activity cannot interfere with the running of the system by requesting an illegal operation.
- Certain requests for system operations, permitted to a nonrestricted activity, are prohibited to a restricted activity, to guard against interference with the system by improper usage of these operations.
- A restricted activity which attempts to perform any of the above illegal operations is aborted (forcibly terminated) by the system, and an error diagnostic giving the reason for the abort is printed on the operator console.

- System resources used – peripheral devices reserved, volumes connected, files and libraries opened – by a restricted activity are recorded by the system. When a restricted activity terminates or is aborted, the system reclaims all such resources which ave not been explicitly returned by the activity, so that they cannot be prevented indefinitely from use by the system or by other activities. Files being created by the activity and left open when the activity terminates can be either preserved or deleted; this option is specified to the load activity utility when the activity is created. A restricted activity cannot explicitly fetch or use free memory blocks, although the system does use free memory blocks on behalf of a restricted activity.
- A restricted activity can be aborted by a specific request from the operator, or from a nonrestricted activity.
- If the system's supply of free memory runs low, all restricted activities are automatically aborted to increase the supply of free memory for the system and nonrestricted activities to run.
- The priority level at which restricted activities can run is limited, so that a restricted activity cannot lock out the running of the system or other activities of higher priority.

Systems with the system integrith option configured, then, are particularly suited to the following applications:

- Activities whose running is of vital importance can be run as nonrestricted activities at a high priority level, while other, less important activities can be run as restricted. Not only will the running of the essential activities take precedence over that of the restricted activities, but also the restricted activities will be prevented from causing a system malfunction through improper operation. Also, if the system is overloaded and the supply of free memory runs low, the essential activities will be permitted to continue running at the expense of the restricted activities.
- Activities being checked out can be run as restricted activities. Program errors cannot then cause a system crash; improper operation will be caught at an earlier stage by the system, and this, together with the error diagnostic printed out, will help the investigation of the error.
- Activities started erroneously by the operator, activities taking longer to run or using more system resources than expected, or activities which have encountered some error in the course of their running, can be cleanly terminated by use of the operator abort command.

The chief operations that are prohibited to a restricted activity, but permitted to a nonrestricted activity, are as follows:

- Fetching and use of free memory blocks. (A restricted activity may, however, examine the contents of free memory blocks such as its own ACB, except in certain special circumstances in a 64K system.)
- Scheduling a nonrestricted activity.
- Scheduling an activity with a secondary TCB.
- Scheduling a task within another activity.
- Scheduling a clock activity or clock task.
- Aborting another activity.

- Communications operations and executive function calls.
- Storing data outside of its activity area, including in the pseudoregisters ZCR1 through ZCR6, or executing a privileged instruction. Therefore, bank switching, changing to 64K indexing, or inhibiting interrupts cannot be done by a restricted activity, because these operations require the execution of a privileged instruction.
- Concurrent execution of more than one task within the activity. (A restricted activity can consist of more than one task, and can schedule these tasks at any time, but the system will not dispatch a new task within a restricted activity until the previous task has terminated.)
- Reentrant operation. (Tasks within a restricted activity are executed sequentially; the activity itself can be reusable or nonreusable, but not reentrant.)
- Utilization of physical sector 0 as the base sector. (A restricted activity's base sector must be relocated to a sector within its activity area.)

For the convenience of the operator wishing to abort lengthy operations, the four language processor components of OS/700 — the DAP assembler, the linkage editor, the test editor and the FORTRAN translator — are capable of being run as restricted activities. This also enables them to be run as background activities with a foreground activity in progress, and to be aborted automatically to preserve the running of the foreground activity if free memory runs low.

CLOCK-RELATED ACTIVITY AND TASK SCHEDULING

The executive allows activities and tasks to be scheduled at any specified time interval. Intervals may be specified in units of quarter-milliseconds, half-seconds, seconds, or minutes. The clock manager maintains a separate queue for each of the four relative time units. Scheduling may occur once (noncyclic) or at specified time intervals (cyclic).

The clock scheduling of an activity or task is identical to the normal scheduling of activities and tasks. However, the user also specifies:

- The number of quarter-milliseconds, half-seconds, seconds, or minutes
- Whether scheduling is to be noncyclic (once) or cyclic (repeated)

Each time a clock activity or clock task is requested, an entry is put in the appropriate clock timer queue. At regular intervals (as specified at system configuration time) the entries in each clock timer queue are examined.

First, entries in the quarter-millisecond queue are scanned. Any entries that have reached their specified time interval are scheduled. Cyclic entries are reset and left in the queue, while noncyclic entries are removed after they have reached their specified time interval.

Next, the clock manager checks if a half-second interval has been reached. If it has, entries in the half-second queue are scanned and processed as described above. If the second interval has been reached, entries in the second queue are scanned and processed. When the minute interval is reached, a similar operation is performed. After all entries whose time intervals have expired are scheduled, the executive returns to normal execution.

INTERRUPT PROCESSING

A number of I/O or processor-generated interrupts may occur during the processing of a task.

The System 700 central processor recognizes two basic types of interrupts: privileged and nonprivileged. When either occurs, the effect of the interrupts on the current task depends on the interrupt mode under which the processor is currently running. Interrupt modes include:

- Inhibited interrupt mode - In this mode, execution of the current task can be interrupted only by privileged interrupts. The privileged interrupts are: memory lockout, power failure, watchdog timer, trace, and stack under/overflow.
- Enabled interrupt mode - In this mode, execution of the current task can be interrupted by any interrupt.

When an interrupt which will suspend current execution of a task occurs, the executive performs the following actions:

1. Saves the status of the interrrupted task.
2. Identifies the type of interrupt, and transfers control to the response code that will process that interrupt.
3. Resumes execution of the interrupted task when the processing of the interrupt is completed, unless a task that has a higher priority than the interrupted task has been scheduled in the meantime.

The registers and status of the interrupted task are saved in an area of main memory referred to as a suspended save area. There is one such area for each priority level; hence the fact that only one task can run on each level at any one time.

There is a special register save area associated with each privileged interrupt. This is necessary because a privileged interrupt can occur before execution is resumed after a nonprivileged interrupt, in which case the suspended save area will already be in use.

QUEUE MANAGEMENT

OS/700 allows dynamic creation and manipulation of queues. Specifically, the user can create a queue, add an item either at the beginning or at the end of the queue, or remove an item from the beginning of the queue.

DISK FILE MANAGEMENT

OS/700 provides management of files on the system disk and on other disks attached to the system — fixed-head, moving-head or cartridge disks. These logical I/O capabilities are in addition to the capability of performing direct physical I/O to a disk on behalf of a user.

The user can create and manipulate any number of named disk files, and treat the data contained within the files as groups of logical records. If desired, the user can organize his files into named groups known as libraries. OS/700 provides the following features:

- Files may have either fixed or variable length records.
- Fixed length record files may be accessed either sequentially or directly.
- Variable length record files may be accessed sequentially only.
- Disk-resident file and library directories.
- Password protection at both file and library level.
- Files can reside on removable disk volumes, which can be interchanged while the system is online.
- File sizes do not have to be defined on creation; they can be expanded automatically when data is added.
- Fast transfers of file data through overlapped disk I/O and multiple buffering, with anticipatory buffer loading.

Logical I/O is described in the OS/700 File Management manual.

PHYSICAL I/O

In addition to the preceding file management capabilities, OS/700 provides a physical I/O interface through which a user is allowed to directly control I/O operations, and transfer data between main memory and any I/O device in the system. Physical I/O operations can be performed on the following devices:

- ASR-33, ASR-35, KSR-33
- Paper tape reader or punch
- Card reader, card punch, and card reader/punch
- Removable fixed-head, and cartridge disk
- 7- and 9-track magnetic tape equipment
- Line printer
- Cassette tape

Device Control Block (DCB)

Operations affecting I/O requests are controlled by the data previously stored in a device control block (DCB). Before starting physical I/O operations, the user must have previously created a DCB containing:

- Type of I/O device
- Logical unit number
- Data mode (ASCII, binary, etc.)
- User identification
- Location of the status block
- Location for return after I/O operation completion

Once the DCB has been created, the desired device must be reserved. When the device becomes available to the user, he can then request physical I/O operations for the device. All I/O requests for each device are placed in separate queues and are acted upon according to their priority.

At the end of the physical I/O operations for the reserved device, the user releases the I/O device so that is is available to satisfy other requests that may have been placed in the reserve queue.

Disk Resource Management

A disk volume must have a user-visible name (unique at an installation); it may contain a label record. Full resource management is provided only on labeled volumes. There are two levels of resource management available:

- Disk space allocation and deallocation on labeled volumes
- Location of a currently mounted named volume or location of a unit available for mounting another volume (labeled or unlabeled)

Physical records on a disk volume are grouped into logical work areas that are allocated and deallocated upon request. A map of current disk work area usage is maintained on the volume.

A request to be connected to a particular named volume involves a search of currently mounted volumes for the desired name, and if necessary, a search for an unused physical unit on which to mount the desired volume.

When necessary, the operator is told which volume to mount on which physical unit (removable disk only). When the volume is mounted, the contents of the label record, if present, are read and verified.

Both levels of disk management are available on labeled volumes to the physical I/O user. Both levels are utilized on behalf of the logical I/O user, although this management is not visible to the user. Logical I/O is supported only on labeled volumes.

OPERATOR INTERFACE

The user has the capability of sending messages to the operator during execution of task code and, when required, of accepting keyboard input from the operator in response to such messages. Thus, the user can include online interaction with the operator as part of a task code. When two or more messages are sent to the operator, the operator has the capability of identifying the message to which he is responding by preceding his response with the message number.

Special system commands (a dollar sign followed by a 2-letter mnemonic) can be issued by the operator at any time. These commands are:

- $SA - Schedule an activity. In COS configurations, the activity must be main memory-resident before the $SA command is issued.
- $LA - Load an external activity into main memory from a specified (nondisk) input medium. This command is supported only in COS systems.
- $AB - Abort a restricted activity. This command is supported only in DOS comfigurations with the system integrity option.
- $CI - Start executing a system command file. This command is supported only in DOS configurations with the system command input option. Additional system commands that may be included in a system command file are $CO (Consonant Command) and $OD (Output Device Command).
- $TR - Terminate execution of a system command file. This command is supported only in DOS configuration with the system command input option.

For more specific details concerning operator interface, refer to Section III in the OS/700 Operator's Guide.

COMMUNICATIONS

A full range of communications capabilities is available under OS/700. Programs may send and receive data in the form of messages to and from communication terminals, control the acknowledgement of received data, obtain and change the status of terminals and lines, and receive information concerning the validity of data and the integrity of the communications hardware in the OS/700 system.

Programs utilizing the communications functions of OS/700 are run as communications tasks, and request the services of the OS/700 Communications Supervisor through communications macro routines. The specific characteristics of communication lines and terminals are established at system configuration time, but can be modified to some extent by user communications tasks. Direct communication with and control of remote terminals are normally handled by the Communications Supervisor; this includes polling (requesting a terminal to send a message), terminal selection (determining if a terminal is ready to receive a message), code conversion, error detection and correction, and maintaining the proper line procedure for the terminals.

For a full description of the communications facilities available under OS/700, the user should refer to the OS/700 Communications manual.

EXECUTIVE FUNCTION CALLS

The following describes the interface between the system and the user – executive function calls – and their use in requesting executive functions.

OS/700 User-System Interface

The user program can interface with OS/700 only through executive function calls. OS/700 interacts with user activities and tasks only when requested to do so by a executive function call. When the user is coding a program and wants the program to call on an executive function, he codes into the program an executive function macro call. Executive function macros are contained in the System Macro Library, MACLIB. When the program is assembled, the assembler expands the macro call into the set of instructions required to call the executive function.

To simplify the writing of reentrant coding, the system supplies the user with six pseudoregisters – ZCR1 through ZCR6 – which are treated by the system in a manner similar to the hardware registers. These registers are located in physical sector zero, and must be referenced indirectly through address constants (DAC's) if the user program runs with relocated base sector.

When the user program makes an executive function call from which a return to his task is to be made, the status of the various registers on entry to the executive is as follows:

- Pseudoregisters ZCR1, ZCR2, and ZCR3 are saved.
- Pseudoregisters ZCR4, ZCR5, and ZCR6 are not saved.
- Hardware registers B and S are saved.
- The A-register contains the function call return address.
- The X-register contains the address of the function call parameter list.

There are three types of executive function macros:

- Those that request executive functions. These macros generate executable code.
- Those that define data for executive functions. They are: LCB$, VCB$, FCB$, DCB$, and TCBS. The TCB$ macro is a special form of a data defining executive macro which defines symbolic names for accessing specific words within a task control block. These macros do not generate executable code.

- The executive function macro – LNK$ – that provides the interface or linkage between the user activity and the OS/700 Executive Function Manager.

The executable code required in a user program to perform an executive function call is as follows:

- The X-register must be loaded with a pointer to a list of parameters that define the arguments (devices, buffers etc.) to be operated on by the executive function. Pseudoregisters may not be used as a parameter list for an executive function call.
- A call (JST) is made to the subroutine ZALINK, which provides the linkage between the user program and the system.
- An additional parameter word is provided to define which executive function is to be executed.

Each executive function macro called by a user program expands into the code described above. The user can optionally request the macro to generate the parameter list itself (an inline parameter list) by supplying the parameters in the macro call. Alternatively, he can code the parameter list itself in some other part of his program (an outline parameter list), and load the pointer into the X-register himself; in this case, the executive function macro call will expand into only the second and third of the three items described above.

The choice between inline and outline parameter lists is up to the user, although there is no alternative to the use of outline parameter lists in re-entrant coding (see the following section). Inline parameter lists are quicker to code, and the macro itself performs some checks on the validity of the parameters when it is expanded; thus inline parameter lists can reduce the number of initial coding errors when a user program is first generated. Outline parameter lists, however, save code, since the macro must generate an extra word (a jump around the parameter list) if an inline list is used; also with care it is often possible to use the same outline parameter list for two or more executive function calls. Outline lists also help the readability of a program listing, since each parameter can be documented with a suitable comment. An outline list should be used if any of the parameters in the list has to be changed or set during program execution, although the user interface has been designed to avoid the necessity of doing this to a large extent.

ZALINK is a 16-word subroutine which forms part of the user program. It is generated by a call on the LNK$ macro. Every module of the user program (i.e., each assembled segment of code having an END statement) which contains an executive function macro call must have access to the subroutine ZALINK. If the user program contains only one module, it should contain one call on LNK$. If the user program has two or more modules containing executive function macro calls one module should contain a call on LNK$, and be headed by the statement ENT ZALINK; all other modules containing executive function macro calls should contain the statement EXT ZALINK. Alternatively, a LNK$ call can be coded into each module. The subroutine ZALINK performs some register operations, and then transfers control to the OS/700 Executive function manager, either by means of a direct subroutine call, or (in the case of a restricted activity, which operates in restricted mode) by generating a memory lockout violation interrupt.

The addresses of the six pseudoregisters ZCR1 through ZCR6 are also defined by the call on LNK$. Every module of a user program which references these pseudoregisters must have access to their definitions. If the user program comprises more than one module, the module containing the LNK$ call can be headed by the statements ENT ZCR1, ENT ZCR2 etc., and modules referencing the pseudoregisters can contain the statements EXT ZCR1, EXT ZCR2 etc. Note that all these pseudoregisters are located in physical sector 0, and that if a user program runs with the base sector relocated to some other sector, it cannot reference the pseudoregisters through a direct memory reference instruction; it must reference them indirectly through a series of links of the form DAC ZC$1, DAC ZCR2 etc.

Restricted activities, which cannot alter the contents of locations outside their activity area, cannot use the pseudoregisters. Activities running in a 64K system (see "Memory Management in 64K Systems" later in this section) can access the pseudoregisters only when the A-bank is set to bank 0.

In many executive function requests, the parameter list points to data structures generated by the user, or specifies words and buffers within the user program into which the system is to transfer information. When a restricted activity makes an executive function call, the system checks that all such words, buffers and data structures actually do lie within the user's activity area; if they do not, the activity is aborted. In the case of a restricted activity running in a 64K system, all such user-generated data structures must lie in the B-space of the activity area. (Refer to "Memory Management in 64K Systems." later in this section.)

Executive Function Macro Calls

The user can request certain executive functions by means of executive function macro calls. An executive function macro call consists of (1) a macro name ending in a dollar sign ($) and (2) a parameter set. An example of an executive function macro call is illustrated below:

Operation	Operand
MAC$	parameter 1,[parameter 2],...,[parameter n]

MAC$ - The executive function macro name

parameter 1, parameter 2 ,..., parameter n - The macro call parameters, separated by commas. Optional parameters are enclosed by brackets. (In this example, parameter 1 is required).

The parameters are positional (i.e., if a parameter is omitted, its position must be preserved by a comma).

Another example of the macro format is shown below:

Operation	Operand
MAC$	buffer address, range

buffer address - The address of the buffer

range - The integer giving the maximum number of words in the buffer

If the macro call is used as defined in this example, it could be written as follows:

```
       MAC$    BUFF,13       BUFF AND 13 ARE DEFINED PARAMETERS
        .
        .
BUFF   BSZ     13
```

Returns From an Executive Function Call

After an executive function call has been processed successfully and a normal return is taken, the next instruction after the executive function call is executed. If the executive function call is not successfully processed, an error return is taken to the address of a routine, specified in the parameter list, which is to process the error. In such cases, the A-register will contain a code which indicates the type of error.

When control returns to a user task from an executive function call, the status of the various registers is as follows:

- Pseudoregisters ZCR1, ZCR2 and ZCR3 are restored.
- Pseudoregisters ZCR4, ZCR5 and ZCR6 are not restored.
- Hardware registers B and S are restored.
- The A-register contains the error code if the error return is taken. Otherwise, the contents of the A-register are undefined.
- If an executive function provides a return parameter it will appear in the A- or X-register. Otherwise, the contents of these registers are undefined.

- The keys, banks, and indexing mode are restored, extended addressing mode set, interrupts enabled, and the J-base set to the J-base value of the activity in which the task resides.

  NOTE: The contents of the pseudoregisters and hardware registers are not guaranteed, when a Wait For I/O function (WIO$) is used in which the I/O status block address pointer parameter is omitted.

Outline Parameter Lists and Reentrant Coding

When a variable parameter is used in a reentrant activity or task, the standard inline macro parameters cannot follow the macro call name. Instead, the parameters must be stored in an outline parameter list in the data section. In addition, the user must dynamically store his variable data in blocks obtained from free memory and may use three of the system pseudoregisters — ZCR1, ZCR2, and ZCR3 — to save the addresses of these blocks.

An example of an outline parameter list is shown below:

```
        LDX    LIST    ADDRESS OF PARAMETER LIST PUT IN X-REGISTER
        MAC$   (X)     EXECUTIVE FUNCTION MACRO CALL
        ---            NORMAL RETURN
         .
         .
         .
BUFF    BSZ    13      THIRTEEN WORDS RESERVED FOR BUFFER
LIST    DAC    **1     ADDRESS OF OUTLINE PARAMETER LIST
        DAC    BUFF    BUFFER ADDRESS
        DEC    13      RANGE
```

NOTE: If any of the optional parameters is omitted, a BSZ 1 entry should replace the appropriate DAC entry in the outline parameter list.

A macro call with an outline parameter list must include an X enclosed in parentheses (X) in the operand field as illustrated in the above example.

The description of each executive function macro call below includes the format required for the outline parameter list as well as an example of it.

If a reentrant program utilizes executive function calls containing variable parameters, it must first issue a GBL$ function call, which has fixed parameters stored in memory; the GBL$ function call requests a fixed-size memory block. This block can then be used to store the outline parameters needed to make reentrant calls to other executive functions, including requests for other variable size memory blocks. When the RBL$ function call is used to return the memory block obtained through the GBL$ function call, the outline parameters in the RBL$ function call can be stored by the user in the block being returned. Refer to the TMA$ function examples for an illustration of this technique.

When free memory blocks are used for outline parameter lists or when any outline list is reused, optional parameters which are to assume the default value must be set to the default value.

User and System Domains

Execution of code under control of OS/700 can take place in either the system domain or the user domain. The system domain is defined as:

- No relocation of base sector
- Extended addressing mode only
- Single precision only
- Stack interrupt disabled
- Trace interrupt disabled
- Bank 0 is always selected
- Normal (nonrestricted) mode with no memory protect.

Depending on the program requirements, the user can select any of the following programming capabilities to define a user domain:

- Relocation of the base sector
- Extended addressing mode only
- Single- or double-precision numbers
- Stack interrupt enabled
- Trace interrupt enabled
- RESTRICT mode and memory protect (restricted activities only)

When control returns to the user program after an executive function call, the status of the above options in the user program is restored to what it was before the function call was issued (except that interrupts are enabled and the extended addressing mode is set).

MEMORY MANAGEMENT IN 64K SYSTEMS

OS/700 supports the running of activities in up to 64K of main memory. The management of activities in over 32K of memory requires special considerations, which are discussed in this section.

Support of over 32K of memory must be specified by the user at the time the system is configured; this support is provided only in DOS configurations. For reasons which will become apparent, the layout of the system components in memory is modified for systems supporting over 32K (known for convenience as 64K systems), in order to locate the free memory area at the bottom of memory. The system layout shown in Figure 2-3 is thus replaced by the layout shown in Figure 2-5 for 64K systems.

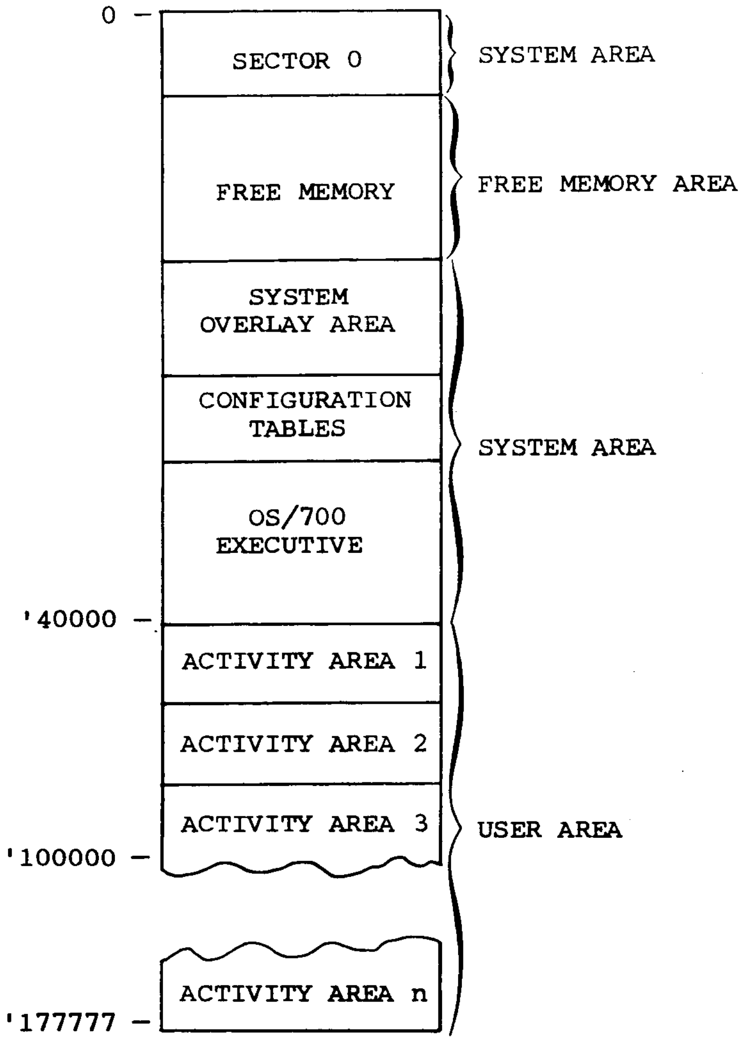


Figure 2-5. Sample Memory Layout for 64K DOS Configuration

Hardware Operation in 64K

Physical addressing in machines with more than 32K of memory is a natural extension of 32K addressing. Locations within the first 32K are addressed from 0 to '777777, and locations beyond the first 32K are addressed from '100000 upwards, up to a maximum of '177777. However, because the machine interprets only 15 bits of an (indirect) address word as an address, only 32K of memory is program addressable at any one time. For programming convenience, therefore, the memory is a machine of over 32K capacity is divided into banks of 16K each, numbered from zero onward. Bank 0 thus comprises locations 0 through '37777, and so on up to bank 3 (if present), which comprises locations 140000 through '177777. The machine hardware contains a bank register, which permits the program to select which particular two banks (32K) of memory are to be program addressable at a particular time. The two selected banks are known as the A-bank and the B-bank.

LOGICAL AND PHYSICAL ADDRESSING

Addressing works as follows. A program, by means of direct or indirect addressing (with or without indexing), references an address between 0 and '77777. This address is known as the logical address. If the logical address lies between 0 and '37777, the hardware interprets it as a physical address in the A-bank; if the logical address lies between '40000 and '77777, the hardware interprets it as a physical address in the B-bank. For example, if a program is running with the contents of the bank register as (1, 3); i.e., with bank 1 as its A-bank and bank 3 as its B-bank, then any logical address from 0 through '37777 referenced by the program will be mapped onto physical bank 1 (physical locations '40000 through '77777), and any logical address from '40000 through '77777 referenced by the program will be mapped onto physical bank 3 (physical locations '140000 through '177777). Thus if the program addresses logical location '1000, the hardware will interpret this as a reference to physical location '50000, and if the program addresses logical location '65000, the hardware will interpret it as a reference to physical location '165000. (See Figure 2-6.)

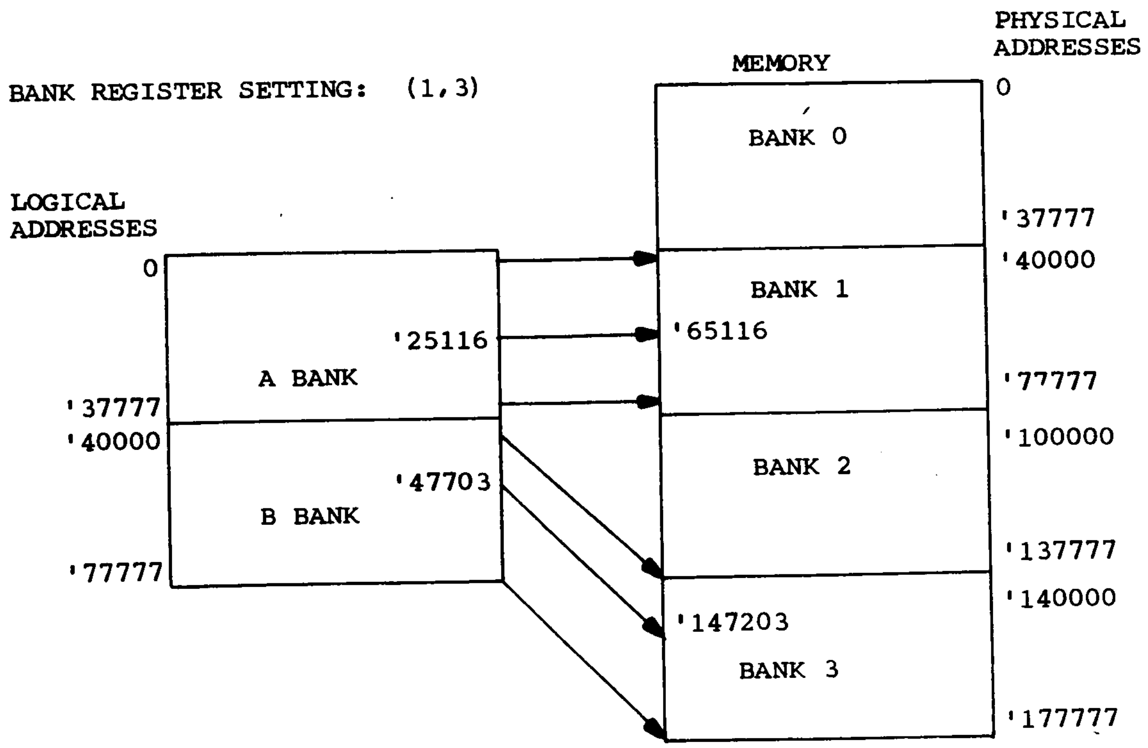


Figure 2-6. Mapping of Logical Addresses Onto Physical Memory Locations

Operation of Activities in 64K

Clearly, to operate conveniently in a given bank N, a program must have either its A-bank or its B-bank set to N. In a 64K OS/700 system, the system executive resides in bank 0, and runs at all times with its A-bank set to zero. User activities may reside in any of the four banks 0 through 3, although the executive usually occupies all of bank 0. Device drivers, which may be memory-resident (thus lying contiguous with the system executive in memory) or disk-resident (thus lying in an activity area), may reside in bank 0 or bank 1. An incoming interrupt causes the hardware to reset the bank register to (0, 1), and as both the system executive and the drivers must be capable of handling interrupts, the above scheme is convenient.

When an activity makes a call on the system executive, the executive must be able to communicate with the activity in order to fetch and store parameters of the function call. If, as is usually the case, the call is being made from a bank other than bank 0, the system accesses the parameters by running with its B-bank set to the bank from which the function call is being made. Address parameters passed to the executive, such as the address from which the function call is being made, the error return address, and parameter poknters, are communicated as logical addresses. For the system to be able to access these easily, parameter addresses must be seen by the system as B-bank addresses; i.e., logical addresses between '40000 and '77777. Therefore, the activity must be linked so that all executive function calls and parameters of those calls, and the subroutine ZALINK, lie in the activity's B-bank. This means that if an activity lies wholly within one bank (other than bank 0), this must be its B-bank to allow it to make executive function calls, and the activity must be linked to run in logical addresses between '40000 and '77777.

Free memory blocks, which must be program addressable by the system executive when running with banks (0, B) set up (where B might be any other bank), are constrained to reset in bank 0.

An activity area may cross a bank boundary. For simplicity of handling, activity areas must occupy contiguous areas of memory, so the two banks in which an activity runs will always be adjacent banks, never disjoint; the lower bank will be the activity's A-bank, and the upper bank will be its B-bank. All executive function calls and their parameters must lie in the upper bank, unless the lower bank is bank 0. One exception to this rule is input/output buffers, which are permitted to lie in either the A-bank or the B-bank. This is because DMA and DMC reference memory locations by physical and not logical addresses; consequently the system must in any case translate logical addresses of I/O buffers to physical addresses. The user can optionally specify I/O buffer addresses to the executive as physical instead of logical addresses. An activity which crosses a bank boundary will normally run with banks (A, B) set up, and when the system is called, banks will be switched to (0, B). An activity which does not cross a bank boundary will normally run with banks (0, B) set up (unless it lies in bank 0, in which case, it will run with banks (0, 1) set up).

The term "B-space" denotes that portion of an activity area which may contain executive function calls and parameters; i.e., that portion of an activity area which is program addressable when running with banks (0, B) set up, where B is the activity's B-bank. Thus an activity's B-space is that portion of its activity area which lies within its B-bank, unless its A-bank is bank 0, in which case its B-space consists of all of its activity area lying in banks 0 and 1 (in this case its B-bank must be bank 1, since its A- and B-banks must be contiguous).

An activity's base sector (J-base), if other than physical sector 0, must be accessible by the system executive when starting or resuming a task within the activity, in order to set the relocated X-register correctly. This means that an activity's J-base must also lie in its B-space.

The Load Activity utility sets into the activity directory on disk the banks that are to be set up when the activity is started by the system. If the activity resides entirely within one bank, bank 3 say, the default banks (0, 3) will be set up. If the activity resides in two banks, banks 1 and 2 say, the default banks (1, 2) will be set up. In specialized applications, a nonrestricted activity can perform its own bank switching, and a user may require an activity to run with some A-bank other than the default one set up. For example, an activity residing wholly within bank 2 (default banks (0, 2)) could be required to communicate directly with another activity residing in bank 1. The Load Activity utility allows the user to specify an A-bank other than the default one, and the system records the banks set up at the time an activity makes an executive function call or is suspended, so that if the activity itself has switched banks or entered the 74K indexing mode, this state is preserved when execution of the activity is resumed.

Occasionally, it may be necessary for an activity to reside entirely within its A-bank. For example, an activity residing within bank 2, and running with banks (1, 2) set up (as in the example above), may wish to communicate with an activity residing wholly within bank 1. To give consistency of meaning to the logical addresses used by the first activity and those used by the second activity, it is desirable for the second activity to run with bank 1 as its A-bank. Such an activity cannot be started by the system, nor can it make executive function calls; however, it can be scheduled, started and terminated by the first activity. The Load Activity utility makes provision for the creation of these specialized activities.

Because the system pseudoregisters and free memory lie in bank 0, they cannot be accessed by an activity running in normal indexing mode with its A-bank set to a bank other than bank 0. A nonrestricted activity residing in two banks other than bank 0, however, can switch banks from the initial setting provided by the system on initiation of the activity, or enter 64K indexing mode, in order to use these facilities. A restricted activity cannot perform bank switching or change of indexing mode, but in any case, it cannot change the contents of the pseudoregisters or free memory blocks, owing to the memory lockout protection.

SECTION III

SCHEDULING TASKS

TASK EXECUTIVE FUNCTIONS

The eight task functions — Task Control Block (TCB$), Schedule Task (STS$), Suspend Task (SUS$), Terminate Task (TMT$), Create a Task Control Block (CTC$), Schedule a Task Control Block (STC$), Connect Clock Task (CCL$), and Disconnect Clock Task (DC$) — are used to:

- Define a series of mnemonic offset symbols for accessing the individual words of a TCB.
- Schedule a task during execution of the current task.
- Suspend execution of the current task so that any other tasks with higher priorities can be dispatched.
- Indicate that a task is terminated.
- Create a task control block (TCB) without scheduling the task.
- Schedule a task for which a TCB has already been created.
- Schedule a clock task to be executed after a specified delay or at regular intervals (cyclically).
- Discontinue execution of a cyclic clock task.

User and System Priority Levels

The OS/700 executive dispatches tasks on a priority basis. The executive provides for a maximum of 16 absolute priority levels; the number of levels in a given system is determined at system configuration time. The absolute priority levels are divided into two types: system levels and user levels. The three system levels, always configured, are the highest priority absolute levels and are reserved for use by executive tasks. The user levels are below the system levels in priority and are the levels on which user tasks and activities are executed. A maximum of 13 user levels can be specified at system configuration time.

The absolute priority levels are assigned numbers from 0 to 15; 0 being the highest priority level and 15 the lowest priority level. The system level numbers are the first three absolute level numbers (0 through 2). The user levels are assigned under level numbers 1 through S (S is the number of user levels specified at system configuration time). User level numbers are used by the STS$, STC$, CCL$, SAC$ and CCA$ executive function action routines when scheduling tasks or activities. In these action routines, the priority level number specified in the function call parameter list is always biased by the number of system levels to determine the absolute level on which the task or activity will be scheduled. The user is prevented from running on any level reserved for the executive.

For executive functions that provide for a user level to be specified, a user level number of zero has the same effect as if this parameter were omitted: a default priority level number is used. Usually the default level for the task is the default level for the activity in which the task resides. If a user level number is specified which is greater than (lower priority) the lowest priority user level in the system, the lowest priority user level will be used.

Task Control Blocks

Associated with each scheduled task is a task control block (TCB). The TCB contains task-related data including: a status word, task entry address, task absolute priority level number, task parameters 1 and 2, and the address of the ACB associated with the activity in which the task resides. If the TCB is a secondary TCB, it has the address of the lead task's TCB. Normally, the user need not concern himself with the detailed contents of the TCB or with the generation of TCB's. When tasks are scheduled by the STS$ action routine and activities are scheduled by the SAC$ action routine, a TCB is automatically generated using the call parameters and scheduled. The user can also generate a TCB via the CTC$ executive function and subsequently schedule the TCB using the STC$ executive function.

All TCB's generated by the system on scheduling an activity, or created by use of the CTC$ or STS$ executive functions, are in free memory blocks. In a 64K system, all such blocks reside in physical bank 0. In order to access the contents of a TCB in a 64K system, e.g., to fetch parameters from the TCB, a user task must have the A-bank set to physical bank 0, or must access the TCB in 64K indexing mode.

Scheduling Tasks

When scheduling tasks, the user must remember that tasks differ from activities in two ways. First, tasks must be resident in main memory before they can be scheduled; this is not the case with activities. Second, the user must not schedule a reusable or nonreusable task before the previous execution has terminated. Further, a nonreusable task must not be rescheduled without being reloaded by rescheduling the nonreusable activity in which it resides. These rescheduling restrictions do not apply to DOS activities because the executive automatically handles activity loading and dispatching. The user must make sure that these task restrictions are considered when he issues STS$ and STC$ function calls to schedule tasks.

NOTE: When a task is scheduled it enters a priority level schedule queue from which it is later dispatched. Tasks are dispatched by the executive after it processes interrupts or when the previous task terminates or suspends. Normally, at least one interrupt (the clock) occurs frequently enough so that the executive dispatches at other times than when a task terminates or suspends. When a task schedules another task on a higher level, the scheduled task normally has not been dispatched when the scheduling task regains control, unless the return to the scheduling task was interrupted. If it is desired to dispatch a task scheduled on a higher level immediately, the scheduling function call (STS$ or STC$) should be followed by a suspend function call (SUS$).

Task Entry

When a given task's TCB is at the beginning of a priority level scheduling queue that is the highest priority level of any tasks scheduled, that task is dispatched by the executive. A dispatched task gains control at the task entry address. When control is transferred to the task, the X-register contains the address (TCB address) of the first word of the task's TCB. It is the user's responsibility to save this TCB address so that the TCB may be returned to free memory; this can be done by giving the TCB address as a parameter in a TMT$ or TMA$ function call when the task or activity is terminated. A pseudoregister (ZCR1, ZCR2, or ZCR3) can be used to store the TCB address if desired.

The TCB address is also needed to give the task access to the two parameters (parameter 1 and parameter 2) specified when the task was scheduled. The user gains access to these parameter words by having the address of the TCB in the X-register and performing a LDA ZTCBP1,1 and a LDA ZTCBP2,1 operation. ZTCBP1 and ZTCBP2 are defined by a TCB$ macro call as the relative displacements within the TCB where parameter 1 and parameter 2 are located.

It is the task to which the parameters are passed that determines the significance of the parameters just as it is the subroutine and not the subroutine caller that determines the significance of the argument list. If more than two parameters are needed to be passed to a task, it is suggested that the Free Memory Parameter Passing Technique described in Appendix G be used. This parameter passing technique must be used by any lead task of an activity that is scheduled with parameters by a FORTRAN program or by the Utilities.

The secondary task of a monitoring activity needs the TCB address so that it may determine if the second activity was successfully loaded into main memory. It is also needed to schedule the lead task of the second activity. The secondary task gets the TCB status word by having the secondary TCB address in the X-register and performing a LDA ZTCBST,1 operation (see the description of the SAC$ executive function for the significance of the status word). The TCB address of the lead task is obtained by having the address of the secondary TCB in the X-register and then performing a LDA ZTCBPT,1 operation. The lead task TCB address can then be used in a subsequent STC$ function call to schedule the lead task of the second activity.

Suspending Tasks

A task can be suspended (1) by an interrupt and (2) by a voluntary suspension. A user task has no control over interrupts. Voluntary suspension involves executing a SUS$ function call. Although the executive handles both suspensions in the same manner, there is a distinction.

Each time an interrupt suspends a task, the executive allocates the system to higher priority tasks. If no higher priority tasks are scheduled, execution of the interrupted task is resumed after the interrupt is processed.

When a task voluntarily suspends, the scheduling queue for the level from which the task was dispatched is "locked" (other tasks in that scheduling queue cannot be dispatched). Tasks can be dispatched from any other scheduling queue (level); the system is not necessarily idle during the voluntary suspension of a task. But if a task is in a continuously suspending loop, system resources can be wasted if no tasks are scheduled on other priority levels and tasks are scheduled on the priority level of the suspended task.

The SUS$ action routine also ensures that a task scheduled on a higher priority level by a currently executing task will be immediately dispatched. To do so, the STS$ or STC$ function call in the currently executing task must be followed by a SUS$ function call.

Terminating Tasks

Unlike activities, tasks can terminate only themselves. When a task terminates by using a TMT$ or TMA$ function call, it allows the executive to dispatch any other scheduled task.

Because a task does not regain control after it terminates, it must replace all allocated system resources that have not been passed to other tasks in the system. This requires that before terminating, the task must make sure all I/O operations initiated by the task are completed, release all reserved I/O devices, close all opened files, close all opened libraries, return all blocks obtained from free memory, and disconnect all explicitly connected volumes. One block of free memory allocated to that task is the TCB generated when the task was scheduled. This block may be released by giving the TCB address as a parameter in the TMT$ or TMA$ function call. Other ways of returning the TCB to free memory are by making an explicit RBL$ (return block) function call, or by converting the contents of the TCB and using it in a STC$ call to schedule a further task, whose responsibility it then becomes to return the TCB.

Normally, a task terminates by issuing a TMT$ function call but if it is the last task of an activity, the task and activity are terminated by a TMA$ function call. When an activity terminates, it must also make sure that any clock task is disconnected as well as releasing all allocated system resources. Clock tasks must be disconnected because the task will be resident only as long as the activity is.

Clock Tasks

The two clock task functions are used to initiate the scheduling of clock requested tasks. The functions are Connect Clock Task (CCL$) and Disconnect Clock Task (DCL$).

Clock tasks must remain resident in main memory as long as those tasks are connected to the clock. Care should be taken to ensure that the activity containing the clock tasks does not monopolize an activity area to the exclusion of other activities which use the same, or an overlapping area.

Clock tasks can be scheduled to be executed once after a specified delay (noncyclic task), or repeatedly at regular intervals (cyclic task). A cyclic task must be disconnected when no furtehr executions of the task are required; a noncyclic task does not have to be disconnected.

Care must be taken to ensure that a clock task is not scheduled cyclically at too short an interval. Otherwise, if the task is scheduled a second time before it has finished executing the first time, scheduling requests may mount up until the system runs short of free memory blocks for TCB's. Alternatively, if the task calls an executive function which performs an implicit terminate task operation, the second scheduling of the clock task could be dispatched, which would lead to trouble if the clock task were not reentrant. The length of the chosen interval must obviously depend on the nature of the work to be done by the task.

Canned TCB's

TCB's are normally fetched from free memory when required. All TCB's generated by the system executive for scheduling a user task - primary and secondary TCB's for scheduling an activity, and TCB's created by the Create TCB (CTC$) and Schedule Task (STS$) executive functions - come from free memory, and it is the responsibility of the task to ensure that these are returned to free memory. It is possible, however, for the user to generate a TCB himself within his activity area - a canned TCB - and use this to schedule a task by means of the Schedule TCB (STC$) executive function. Normally it is more convenient to avoid this. If the user wishes to use this method, however, a number of rules must be observed.

- The method should not be used in a 64K system, unless the user can guarantee that the canned TCB will lie within physical bank 0 or 1. This is because the TCB manager, which examines and services queues of TCB's, operates always with its A-bank set to bank 0, and its B-bank set to bank 1.
- The user must ensure that the TCB is not returned to free memory by the scheduled task. It must not be specified as a parameter of a TMT$ or TMA$ call.
- The TCB control word must be set up correctly, according to the layout specified in Appendix H, Table H-5. Bits 5 through 8 must contain the priority level with the correct bias added to convert the user priority level to the corresponding system level; i.e., with the value 2 added. Bit 12 must be set to indicate that this is a user task. All other bits in this word must be zero.
- The ACB address of the associated activity must be explicitly set in word 7 of the TCB (see Appendix H, Figure H-7).

Tasks Within Restricted Activities

A restricted activity can schedule tasks within itself. However, special considerations apply to tasks within restricted activities. Until the time a restricted activity terminates or is aborted, one and only one task can be executing within the activity at any one time. A restricted activity can use only the STS$ executive function to schedule a task; it will be aborted if it attempts to use the CTC$ or STC$ executive functions. When a restricted activity calls the STS$ executive function, the action routine does not schedule the task immediately; instead it queues the task on a separate queue, and the task is scheduled only when the current task has terminated by calling the TMT$ executive function.

The priority level specified for a task within a restricted activity has a special significance. First, the priority level at which a task within a restricted activity can run is limited by the system. If a restricted activity attempts to schedule a task at a level above the highest allowed priority level, the system will automatically adjust the level down to the highest allowed level. The task is not scheduled immediately; instead it is placed on a separate queue of tasks waiting to be executed within that activity, and scheduled later when the current task has terminated. The waiting tasks are queued in priority order. If two or more tasks are requested at the same priority level, the system schedules them in the order in which the calls on the STS$ executive function were made.

The following rules must be observed for tasks within a restricted activity:

- The user must bear in mind that any task scheduled will not be executed until the current task has terminated. If a task schedules a second task and expects to wait until the second task has run, it will be very disappointed.

- At least one task must be currently executing up until the time the activity terminates. The last task must therefore terminate with a Terminate Activity (TMA$) call, and not with a Terminate Task (TMT$) call. If a task terminates with a TMT$ call and there is no other task requested for execution within the activity, the activity will be aborted.
- A restricted activity cannot schedule a task within another activity. If the task entry point specified in the STS$ call is not within the activity area (and within the B-space in a 64K system), the activity will be aborted.
- A restricted activity cannot call the Return Block (RBL$) executive function to return a TCB to free memory. Every task of a restricted activity must, therefore, return its TCB by giving the TCB address as a parameter of the TMA$ or TMT$ call.
- The resources used by a restricted activity - reserved I/O device, open files and libraries, connected volumes, and the TCB - are returned automatically to the system when a restricted activity terminates or is aborted (with the exception of work areas explicitly allocated on the disk. However, an activity that is being checked out as a restricted activity must return these resources if it is later to be run as a nonrestricted activity.
- Tasks within a restricted activity may request only a limited subset of the available executive functions. See Table F-2 in Appendix F for the list of permissible functions.

Define Task Control Block (TCB$)

The TCB$ macro allows the user to access specific words in the TCB with symbolic names instead of by numeric displacements.

MACRO EXPANSION

The TCB$ macro defines by EQU statements the relative position of words in the TCB which must be accessible by the user.

MACRO CALL FORMAT

Location	Operation	Operand
	TCB$	

MACRO EXPANSION DETAILS

The user can access the following relative locations within a TCB by their symbolic names:

Symbolic Name	Contents of Relative Location
ZTCBST	status
ZTCBEN	task entry address
ZTCBP1	parameter 1
ZTCBP2	parameter 2
ZTCBCW	TCB control word
ZTCBPT	pointer to primary TCB
ZTCBAC	ACB address

OUTLINE PARMETER LIST

The outline pamameter list is not applicable to the TCB$ macro.

Example:

The following example shows the definition of specific words within a task control block by a TCB$ macro call.

```
         TCB$
         .
         .
         .
*                            UPON ENTRY TO THE TASK, X-REGISTER
*                            POINTS TO TCB
         .
         .
         .
         LDA   ZTCBP1,1      GET PARAMETER 1 FROM TCB
         .
         .
         .
         LDA   ZTCBP2,1      GET PARAMETER 2 FROM TCB
```

Schedule Task (STS$)

The STS$ function is used to schedule a task for execution. The task must be currently resident in main memory.

FUNCTION ACTION

After obtaining a block from free memory the STS$ action routine creates a TCB for the task. The task is then scheduled by attaching the TCB to the end of the priority level schedule queue, and control is returned to the calling program.

MACRO CALL FORMAT

Location	Operation	Operand
[symbol]	STS$	task entry address, [level], [ACB address pointer], [parameter 1], [parameter 2], error return address

symbol - Optional. The symbolic location of the STS$ macro instruction.

task entry address - The address of the entry point of the task.

level - Optional. The user priority level number on which the task is to be scheduled. If a level number lower than the lowest user priority level is specified, the lowest user priority level is used to schedule the task. If the task is within a restricted activity and the level is higher than the highest level permitted for restricted activities (user level 3), the task will be scheduled at the highest level. If this parameter is omitted, the task is scheduled on the default level specified in the activity control block of the activity in which the task resides.

ACB address pointer - Optional. The address of a word containing the address of the activity control block associated with the activity in which the task resides and under whose control the task is to be executed. If this parameter is omitted, the task is scheduled as part of the current activity. If this parameter is the address of a word containing zero (i.e., ACB address = zero), the task is scheduled as part of the current activity. If the function call is made by a restricted activity, the action routine assumes that the task being scheduled is part of the same activity, and this parameter is ignored.

parameter 1 - Optional. A parameter word that can be used by the caller to send data to the task. The action routine puts this parameter in word ZTCBP1 of the TCB. If this parameter is omitted, a zero word is generated in the TCB for word ZTCBP1. If parameter 2 is not zero, parameter 1 must not be zero unless the free memory block parameter passing technique is being used (see Appendix G, "Free Memory Block Parameter Passing Technique").

parameter 2 - Optional. A parameter word that can be used by the caller to send data to the task. The action routine puts this parameter in word ZTCBP2 of the TCB. If this parameter is omitted, a zero word is generated in the TCB for word ZTCBP2. If parameter 1 is zero, parameter 2 must be zero unless the free memory block parameter passing technique is being used (see Appendix G, "Free Memory Block Parameter Passing Technique").

error return address - The address to which control is returned if an error is found during the processing of the STS$ function call.

NORMAL RETURN

Control is returned to the calling program, at the instruction following the STS$ function call after the task has been scheduled.

ERROR RETURN

Control is returned to the error return address specified in the STS$ parameter list with the error code in the A-register when the following error is detected:

A-Register Contents (Octal)	Error Condition
34	Requested executive function is not configured.

ACTION ROUTINE DETAILS

A block is omitted from free memory and used to contain the TCB generated using the specified STS$ parameters. If the level parameter is omitted, OS/700 schedules the task on the default level of the activity in which it resides by using the default level in generating the TCB. The default level is found in the ACB associated with the activity. If the ACB address pointer parameter is omitted, the address of the ACB of the current activity is used in generating the TCB.

After the TCB is generated, the task is scheduled by placing the TCB at the end of the priority level schedule queue. If the priority level in the TCB is lower than the lowest user priority level, the task is scheduled on the lowest user priority level. The STS$ action routine exits by returning control to the calling program at the instruction following the STS$ function call. When the calling program regains control, the scheduled task may not have been dispatched.

If the action routine is being called by a restricted activity, the task is not scheduled immediately; instead it is placed on a separate queue of tasks waiting to be executed within that activity, and scheduled later when the current task has terminated. The waiting tasks are queued in priority order.

OUTLINE PARAMETER LIST

Word Number	Operation	Operand
0	DAC	task entry address
1	DEC	[level]
2	DAC	[ACB address pointer]
3	DAC	[parameter 1]
4	DAC	[parameter 2]
5	DAC	error return address

Parameters in the outline parameter list are the same as those described above for the inline parameter list. If any of the optional parameters is omitted, a BSZ 1 statement must replace the respective statement.

Examples:

A task, starting at ATASK in the current activity, is to be scheduled on level 2. The task requires two parameters: the location T1, and the location T2.

```
        STS$    ATASK,2,,T1,T2,TERR     DEFAULT ACB
         ⋮
ATASK   ----                        START OF THE TASK
         ⋮
T1      BSZ     1                   PASS ADDRESS TO TASK AS PARAMETER 1
         ⋮
T2      BSZ     1                   PASS ADDRESS TO TASK AS PARAMETER 2
         ⋮
TERR    ----                        ERROR RETURN
```

Using the outline parameter list, the above example would be written as follows:

```
        LDX     PLIST               SET UP POINTER TO OUTLINE LIST
        STS     (X)
         ⋮
PLIST   DAC     *+1                 POINTER TO OUTLINE LIST
        DAC     ATASK               POINTER TO START OF TASK
        DEC     2                   LEVEL PARAMETER
        BSZ     1                   DEFAULT ACB ADDRESS POINTER
        DAC     T1                  PARAMETER 1
        DAC     T2                  PARAMETER 2
        DAC     TERR                ERROR RETURN ADDRESS
         .
T1      BSZ     1                   PASS ADDRESS TO TASK AS PARAMETER 1
T2      BSZ     1                   PASS ADDRESS TO TASK AS PARAMETER 2
TERR    HLT                         ERROR RETURN
         ⋮
ATASK   ----                        START OF THE TASK
```

Suspend Task (SUS$)

The SUS$ function call is used to suspend the task in which it appears and allows the possible dispatching of a task currently scheduled at another priority level.

FUNCTION ACTION

The SUS$ action routine saves all hardware registers (except the A-register) and pseudoregisters, and exits to the executive which dispatches the highest priority scheduled task other than the task being suspended, and returns control to the calling program when the suspended task is resumed.

MACRO CALL FORMAT

Location	Operation	Operand
symbol	SUS$	

symbol - Optional. The symbolic location of the SUS$ macro instruction.

This macro instruction has no parameters.

NORMAL RETURN

Control is returned to the calling program, at the instruction following the SUS$ function call when the task is resumed. The suspended task is resumed with all hardware registers (except the A-register), indexing mode, keys, pseudoregisters, and banks restored.

As is the case after any executive function call, when control is returned, the addressing mode is set to extended mode, interrupts are enabled, and the J-base is set to the activity J-base value. This setting of the return state may result in a state being changed from what it was when the SUS$ function call was issued.

ERROR RETURN

There is no error return for this function.

ACTION ROUTINE DETAILS

The state of the suspending task is saved in the priority level suspend area. All hardware registers (except the A-register) and pseudoregisters are saved there. The system then determines if any task is currently scheduled at a priority level higher than the suspending level. If so, the highest priority task is dispatched. If no tasks are scheduled at a level higher than the suspending level, the highest priority scheduled task under the suspended level is dispatched. This may result in tasks of lower priority than the suspended task being dispatched and terminated before the suspended task is resumed.

The suspended task regains control with all hardware registers (except the A-register), pseudoregisters, indexing mode, keys, and banks restored. The contents of the A-register are destroyed by the subroutine ZALINK through which the SUS$ function call passed; it cannot be restored. Control is returned with extended addressing mode set, interrupts enabled, and the J-base set to the J-base value of the activity in which this task resides. This may result in a mode change from that of when the task was suspended.

While a task is suspended, no other scheduled task on the same level can be dispatched.

OUTLINE PARAMETER LIST

There is no outline parameter list for this function, which has no parameters.

Examples:

A task to be suspended to allow any currently scheduled tasks on different levels to be dispatched.

```
        SUS$                 SUSPEND THE TASK
        ----                 RESUME TASK PROCESSING
```

Terminate Task (TMT$)

The TMT$ function call is used to terminate the task in which it appears. If the task being terminated is the last task in an activity, the TMA$ function must be used instead of TMT$, because TMA$ terminates the activity as well as the task.

FUNCTION ACTION

The TMT$ action routine returns the TCB (if present) to free memory, terminates the current task and exits to the executive, which dispatches the highest priority scheduled task.

MACRO CALL FORMAT

Location	Operation	Operand
[symbol]	TMT$	[TCB address pointer]

symbol - Optional. The symbolic location of the TMT$ macro instruction.

TCB address pointer - Optional. The address of a word containing the address of the task control block associated with the terminating task. The TCB is returned to free memory. If this parameter is omitted, no block is returned to free memory.

NORMAL RETURN

The TMT$ action routine exits to the executive, and control does not return to the calling program.

ERROR RETURN

There is no error return for this function. Control is always returned to the executive.

ACTION ROUTINE DETAILS

If the TCB address pointer parameter is present, and the TCB address is not zero, the block is returned to free memory. If the TCB address pointer parameter is omitted or if the parameter is present but the TCB address is zero, no block is returned to free memory.

TMT$ exits to the executive, which dispatches the highest priority scheduled task. No return is made to the calling program. Because a task does not regain control after it terminates, it must replace all allocated system resources that have not been passed to other tasks in the system. This requires that before terminating a task, the user must:

1. Make sure all I/O operations initiated by the task are completed.
2. Release all reserved I/O devices.
3. Close all opened files.
4. Close all opened libraries.
5. Return all blocks obtained from free memory.
6. Disconnect all explicitly connected volumes.

If the activity containing the task is a restricted activity, the action routine checks that the TCB address (which must be present) is correct, and that another task within the activity has been requested and is waiting to be scheduled. If there is no other task, the activity is aborted. If another task exists, the routine schedules the first task from the queue of waiting tasks, and exits to the executive dispatcher.

OUTLINE PARAMETER LIST

Word Number	Operation	Operand
0	DAC	[TCB address pointer]

The parameter in the outline parameter list is the same as that described above for the inline parameter list. If the TCB address pointer is omitted, a BSZ 1 statement must replace the respective DAC statement.

Examples:

A task is to be terminated. Its TCB has been returned to free memory prior to termination.

```
        TMT$                          TERMINATE THE TASK
```

In the following example, a task is to be terminated; its TCB address is stored in word TCBA, and this TCB is returned to free memory.

```
        TMT$     TCBA                 TERMINATE THE TASK
          :
TCBA    BSZ      1                    USED TO STORE THE TCB ADDRESS
```

Using an outline parameter list, the above example would be written as follows:

```
        LDX      PLIST                SET UP POINTER TO OUTLINE LIST
        TMT$     (X)                  TERMINATE THE TASK
          :
PLIST   DAC      *+1                  POINTER TO OUTLINE LIST
        DAC      TCBA                 POINTER TO TCB ADDRESS
          :
TCBA    BSZ      1                    TCB ADDRESS STORAGE
```

Create a Task Control Block (CTC$)

The CTC$ function is used to create a TCB without scheduling the task. The created TCB can be scheduled later using the STC$ function.

FUNCTION ACTION

The CTC$ action routine creates a TCB for the task after obtaining a block from free memory. It then places the address of the created TCB in the X-register, and returns to the calling program without scheduling the task.

A restricted activity cannot call the CTC$ function. If it attempts to do so, the activity will be aborted.

MACRO CALL FORMAT

Location	Operation	Operand
[symbol]	CTC$	task entry address, [level], [ACB address pointer], [parameter 1], [parameter 2], error return address

symbol - Optional. The symbolic location of the CTC$ macro instruction.

task entry address - The address of the entry point of the task.

level - Optional. The user priority level number (1 through n) on which the task is to be scheduled later. If a level number lower than the lowest user priority level is specified, the lowest user priority level is used when the TCB is scheduled. If this parameter is omitted, the TCB is generated using the default level specified in the ACB of the activity in which the task resides.

ACB address pointer - Optional. The address of a word containing the address of the activity control block associated with the activity in which this task resides and under whose control the task is to be executed. If this parameter is omitted, the action routine puts into the TCB the ACB address of the activity making the function call or is the address of a word containing zero (i.e., ACB address = zero) the TCB is generated with the task as part of the current activity.

parameter 1 - Optional. A parameter word that can be used by the calling program to pass data to the task. The action routine puts this parameter in word ZTCBP1 of the TCB. If this parameter is omitted, a zero word is generated in the TCB for word ZTCBP1. If parameter 2 is not zero, parameter 1 must not be zero unless the free memory block parameter passing technique is being used (see Appendix G, "Free Memory Block Parameter Passing Technique").

parameter 2 - Optional. A parameter word that can be used by the calling program to pass data to the task. The action routine puts this parameter in word ZTCBP2 of the TCB. If this parameter is omitted, a zero word is generated in the TCB for word ZTCBP2. If parameter 1 is zero, parameter 2 must be zero unless the free memory block parameter passing technique is being used (see Appendix G, "Free Memory Block Parameter Passing Technique").

error return address - The address to which control is returned if an error is found during the processing of the CTC$ function call.

NORMAL RETURN

Control is returned to the calling program, at the instruction following the CTC$ function call after the TCB has been created. On return, the address of the created TCB is in the X-register.

ERROR RETURN

Control is returned to the error return address specified in the CTC$ parameter list with the error code in the A-register when any of the following errors is detected:

A-Register Contents (Octal)	Error Condition
34	Requested executive function is not configured.
35	Requested executive function is disk-resident and there was a disk error during an attempt to load the function into main memory.

ACTION ROUTINE DETAILS

A block is obtained from free memory and is used to contain the TCB generated using the specified CTC$ parameters. If the level parameter is omitted, the task is scheduled on the default level of the activity in which it resides by using the default level in generating the TCB. The default level is found in the ACB associated with the activity. If the level is specified as lower than the lowest user priority level, the lowest user priority level is used when the TCB is scheduled. If the ACB address pointer parameter is omitted, the address of the ACB of the current activity is used in generating the TCB.

CTC$ exits by returning control to the calling program at the instruction following the CTC$ function call with the address of the created TCB in the X-register.

OUTLINE PARAMETER LIST

Word Number	Operation	Operand
0	DAC	task entry address
1	DEC	[level]
2	DAC	[ACB address pointer]
3	DAC	[parameter 1]
4	DAC	[parameter 2]
5	DAC	error return address

Parameters in the outline parameter list are the same as those described above for the inline parameter list. If any of the optional parameters are omitted, a BSZ 1 statement must replace the respective statement.

Examples:

A TCB for a task, ATASK, is to be created. This task resides in the current activity and is to be scheduled later on priority level 2; using the STC$ function call. The two parameters required by the task are to be provided prior to its scheduling. The address of the TCB is stored in word T1.

```
        CTC$    ATASK,2,,0,0,TERR  CREATE TCB
        STX     T1                 SAVE TCB ADDRESS
         :
        LDX     T1
        STA     ZTCBP1,1           STORE PARAMETER 1 IN TCB
        LDA     PARAM
        STA     ZTCBP2,1           STORE PARAMETER 2 IN TCB
         :
*
ATASK   ----                       START OF TASK
         :
T1      BSZ     1                  STORAGE FOR TCB ADDRESS
TERR    HLT                        ERROR RETURN
*
```

Using an outline parameter list, the above example would be written as follows:

```
        LDX     PLIST              SET UP POINTER TO OUTLINE LIST
        CTC$    (X)
         :
        LDX     T1
        STA     ZTCBP1,1           STORE FIRST PARAMETER
        LDA     PARAM
        STA     ZTCBP2,1        STORE SECOND PARAMETER
         .
         .
         .
PLIST   DAC     *+1             POINTER TO OUTLINE LIST
        DAC     ATASK
        DEC     2
        BSZ     1
        BSZ     1
        BSZ     1
        DAC     TERR            ERROR RETURN ADDRESS
*
ATASK   ----                    START OF TASK
         .
         .
         .
T1      BSZ     1               TCB ADDRESS
         .
         .
         .
TERR    HLT                     ERROR RETURN
```

Schedule Task Control Block (STC$)

The STC$ function is used to schedule a task that is currently resident in main memory by using a previously defined TCB. The TCB may have been created by a previous CTC$ function call.

FUNCTION ACTION

The STC$ action routine schedules the task by attaching the TCB to the end of the priority level schedule queue and returns to the calling program.

A restricted activity cannot call the STC$ function. If it attempts to do so, the activity will be aborted.

MACRO CALL FORMAT

Location	Operation	Operand
[symbol]	STC$	TCB address pointer, error return address

symbol - Optional. The symbolic location of the STC$ macro instruction.

TCB address pointer - The address of a word containing the address of a previously generated task control block associated with the task to be scheduled.

error return address - The address to which control is returned if an error is found during the processing of the STC$ function call.

NORMAL RETURN

After the task has been scheduled, control is returned to the calling program, at the instruction following the STC$ function call.

ERROR RETURN

Currently, the error return is not applicable.

ACTION ROUTINE DETAILS

The task is scheduled by placing the TCB at the bottom of the priority level schedule queue. If the priority level in the TCB is lower than the lowest user-priority level, the task is scheduled on the lowest user-priority level. STC$ exits by returning control to the calling program at the instruction following the STC$ function call. When the calling program regains control, the scheduled task may not have been dispatched.

OUTLINE PARAMETER LIST

Word Number	Operation	Operand
0	DAC	TCB error pointer
1	DAC	error return address

Parameters in the outline parameter list are the same as those described above for the inline parameter list.

Examples:

A task is to be scheduled. Its TCB has been created previously by a CTC$ function call. The address of the TCB has been stored in TCBA.

```
        STC$     TCBA,TERR       SCHEDULE TASK
         ⋮
TCBA    BSZ      1               ADDRESS OF TCB STORED
*
TERR    HLT                      ERROR RETURN, CURRENTLY NOT USED
```

Using an outline parameter list, the above example would be written as follows:

```
        LDX      PLIST           SET UP POINTER TO OUTLINE LIST
        STC$     (X)
         ⋮
PLIST   DAC      *+1             POINTER TO OUTLINE LIST
        DAC      TCBA
        DAC      TERR            ERROR RETURN ADDRESS
         ⋮
TCBA    BSZ      1               ADDRESS OF TCB STORED
TERR    HLT                      ERROR RETURN, CURRENTLY NOT USED
```

Connect Clock Task (CCL$)

The CCL$ function is used to connect a clock task to a system timer.

FUNCTION ACTION

The CCL$ action routine generates a clock TCB and places it in the proper timer queue. Then the CCL$ action routine returns control to the calling program through the normal return.

A restricted activity cannot call the CCL$ function. If it attempts to do so, the activity will be aborted.

MACRO CALL FORMAT

Location	Operation	Operand
[symbol]	CCL$	task entry address, [level], [number of time units], time unit type, [connect type], [parameter 1], [parameter 2], error return address

symbol - Optional. The symbolic location of the CCL$ macro instruction.

task entry address - The address of the entry point of the task.

level - Optional. The user priority level number on which the task is to be scheduled. If a level number lower than the lowest user priority level is specified, the lowest user priority level is used to schedule the task. If this parameter is omitted, the task is scheduled on the default level specified in the ACB of the activity in which the task resides.

number of time units - Optional. An integer expressing the number of time units from now until the task is to be scheduled.

If this parameter is omitted, zero is used. The range of this parameter depends on the time unit type parameter.

time unit type - An integer specifying the time unit for the number of time units parameter:

0 = Absolute time of day in minutes from midnight.

1 = millisecond. The range of number of time units is 0 through 1023.

2 = half-second. The range number of time units is 0 through 4095.

4 = second. The range of number of time units is 0 through 4095.

8 = minute. The range of number of time units is 0 through 4095.

If this parameter is omitted, the user return is taken with the value 1 in the A-register.

connect type - Optional. An integer specifying whether the task is to be scheduled on a cyclic or noncyclic basis.

0 = cyclic

1 = noncyclic

If this parameter is omitted, zero (cyclic) is used.

parameter 1 - Optional. A parameter word that can be used by the caller to pass data to the task. The action routine puts this parameter in word ZTCBP1 of the TCB. If this parameter is omitted, a zero word is generated in the TCB for word ZTCBP1. If parameter 2 is nonzero, parameter 1 must not be zero unless the free memory block parameter passing technique is being used (see Appendix G, "Free Memory Block Parameter Passing Technique").

parameter 2 - Optional. A parameter word that can be used by the caller to pass data to the task. The action routine puts this parameter in word ZTCBP2 of the TCB. If this parameter is omitted, a zero word is generated in the TCB for word ZTCBP2. If parameter 1 is zero parameter 2 must be zero unless the free memory block parameter passing technique is being used (see Appendix G, "Free Memory Block Parameter Passing Technique").

error return address - The address to which control is returned if an error is found during the processingof the CCL$ function call.

NORMAL RETURN

Upon normal return, the clock task is connected to the proper clock timer queue. Also Upon normal return, the X-register contains the address of the clock TCB used to connect the task to the clock. This address must be saved if a subsequent DCL$ function call is to be issued.

ERROR RETURN

Control is returned to the error return address specified in the CCL$ parameter list with the error code in the A-register when any of the following errors is detected:

A-Register Contents (Octal)	Error Condition
1	Timer not configured.
34	Requested executive function not configured.
35	Requested executive function is disk-resident and there was a disk error during an attempt to load the function into main memory.

ACTION ROUTINE DETAILS

A clock TCB, which defines the clock task, is generated. This TCB is placed in the proper timer queue. Return is then made to the normal return.

When the specified time interval has elapsed, the clock task is scheduled and dispatched. If the task is to be cyclic, its TCB remains on the timer queue until removed by a DCL$ function call. If the task is not cyclic, its TCB is removed from the timer queue when the task is scheduled.

OUTLINE PARAMETER LIST

Word Number	Operation	Operand
0	DAC	task entry address
1	DEC	[level]
2	DEC	[number of time units]
3	DEC	[time unit type]
4	DEC	[connect type]
5	DAC	[parameter 1]
6	DAC	[parameter 2]
7	DAC	error return address

Parameters in the outline parameter list are the same as those described above for the inline parameter list. If any of the optional parameters are omitted, a BSZ 1 statement must replace the respective statement.

Examples:

The example below illustrates the use of the CCL$ function to connect a task to the clock for cyclic execution. The task CGOLT is to be connected to the clock for scheduling at 20-second intervals. The task is scheduled on user priority level 4.

```
*                                 CONNECT TASK CGOLT TO THE CLOCK FOR
*                                 CYCLIC EXECUTIONS
        CCL$     CGOLT,4,20,4,0,,,ERRT
        STX      ATCB             SAVE TCB ADDRESS
*
ATCB    BSZ      1
ERRT    ---
```

Using an outline parameter list, the above example would be written as follows:

```
*                                        CONNECT TASK "CGOLT" TO THE CLOCK FOR
*                                        CYCLIC EXECUT1ON
        LDX      CLST                    GET LIST ADDRESS
        CCL$     (X)
        STX      ATCB                    SAVE TCB ADDRESS
*
ATCB    BSZ      1
CLST    DAC      *+1                     ADDRESS OF OUTLINE LIST
        DAC      CGOLT                   ADDRESS OF TASK ENTRY
        DEC      4                       PRIORITY LEVEL = 4
        DEC      20                      20 UNITS OF TIME
        DEC      4                       UNITS = SECONDS
        DEC      0                       CYCLIC CLOCK TASK
        BSZ      1                       USER PARAMETER 1
        BSZ      1                       USER PARAMERER 2
        DAC      ERRT                    ERROR RETURN ADDRESS
*
*                                        ABSOLUTE TIMER NOT CONFIGURED
*
ERRT    ---
```

Disconnect Clock Task (DCL$)

The DCL$ function is used to disconnect a specified clock task from a system timer.

FUNCTION ACTION

The DCL$ action routine removes the clock task TCB from the timer queue on which it resides, thus disconnecting the task. It then returns all associated blocks to free memory and returns control to the calling program.

A restricted activity cannot call the DCL$ function. If it attempts to do so, the activity will be aborted.

MACRO CALL FORMAT

Location	Operation	Operand
[symbol]	DCL$	TCB address pointer, error return address

symbol - Optional. The symbolic location of the DCL$ macro instruction.

TCB address pointer - The address of a word in which the address of the clock TCB has been stored. This TCB address has been returned to the caller in the X-register by the CCL$ function.

error return address - The address to which control is transferred if an error is discovered during the processing of the DCL$ function call.

NORMAL RETURN

Upon normal return, the clock task is disconnected from the clock.

ERROR RETURN

Control is returned to the error return address specified in the DCL$ parameter list when any of the following errors is detected:

A-Register Contents (Octal)	Error Condition
34	Requested executive function is not configured.
35	Requested executive function is disk-resident and there was a disk error during an attempt to load the function into main memory.
Not Significant	The specified clock TCB was not found in the timer queue. (There is no specific error code associated with this error condition.)

ACTION ROUTINE DETAILS

The clock task TCB is removed from the timer queue. If, in a search of the timer queue, the given TCB cannot be found on the queue, the error return is taken. After the TCB is removed, all associated blocks are returned to free memory. Return is to the normal return.

OUTLINE PARAMETER LIST

Word Number	Operation	Operand
0	DAC	TCB address pointer
1	DAC	error return address

Parameters in the outline parameter list are the same as those described above for the inline parameter list.

Examples:

The example below illustrates the use of the DCL$ function to disconnect a clock TCB from the clock queue. The address of the TCB to be disconnected was stored in word ATCB after the clock task was originally connected by CCL$ function.

```
*                              DISCONNECT A CLOCK TCB FROM CLOCK
*
        DCL$    ATCB,ERRC
*
*                              CLOCK TCB ADDRESS
*
ATCB    BSZ     1              TO HOLD TCB ADDRESS
*
ERRC    ----                   ERROR RETURN
```

Using an outline parameter list, the above example would be written as follows:

```
        LDX     PLIST          SET UP POINTER TO OUTLINE LIST
        DCL$    (X)            DISCONNECT CLOCK TCB
         :
PLIST   DAC     *+1            POINTER TO OUTLINE LIST
        DAC     ATCB           POINTER TO TCB ADDRESS
        DAC     ERRC           ERROR RETURN ADDRESS
         :
ATCB    BSZ     1              TCB ADDRESS, SAVED AFTER CCL$ CALL
         :
ERRC    ----                   ERROR RETURN
```

SECTION IV
SCHEDULING ACTIVITIES

The following system functions are used to initiate and terminate activities.

ACTIVITY FUNCTIONS

The five activity functions — Schedule Activity (SAC$), Terminate Activity (TMA$), Abort Activity (ABT$), Connect Clock Activity (CCA$), and Disconnect Clock Activity (DCA$) — are used:

- To schedule activities at any point in a task,
- To indicate that an activity is to be terminated,
- To abort a restricted activity,
- To schedule a clock activity to be executed after a specified delay or at regular intervals (cyclically),
- To discontinue execution of a cyclic clock activity.

An activity comprises a collection of instructions and data that form one or more tasks. The entire activity resides in one contiguous main memory area, called the activity area, when it is executed. If disk-resident, an activity is brought into main as one complete logical entity memory from disk. The lead task is scheduled when the activity is requested.

Activities can be nonreusable, reusable, or reentrant. An activity is nonreusable if it must be reloaded prior to each execution. A reusable activity is one that must be reinitialized (by internal code) prior to each execution. It does not need to be reloaded unless it has been overwritten by another activity. A reusable activity must run to completion before being executed again.

A reentrant activity can be interrupted and executed (at a higher priority level) with a different set of input parameters without destroying the former state. Later, execution can be continued in the former state as though the activity had not been interrupted. The variable data areas required for a reentrant activity are obtained from free memory.

An activity must remian in main memory until its execution is completed; all I/O operations must be finished; all clock tasks must be terminated (if noncyclic) or disconnected (if cyclic); all blocks obtained from free memory must be returned unless they are being passed to another activity. All activities should terminate through Terminate Activity (TMA$) function call, although restricted activities can be terminated by an ABT$ function call from non-restricted activity, or from the system executive.

An activity is defined when it is loaded. This is accomplished by the utility function Load Activity or the system command $LA (see OS/700 Operators Guide). Permanent and Temporary main memory-resident activities are defined when the system is configured. At this time, the activity is placed on the disk and an entry for it is made in the disk activity directory. This directory contains information concerning the activity, including the name and the default priority level.

The name of an activity can be from one through six alphanumeric characters, the first of which must be alphabetic. This name is used to identify an activity when it is scheduled (see the description of the SAC$ function). The default priority level, specified in the activity control block (ACB), is the level at which the lead task of the activity will be scheduled if the level parameter in the SAC$ function call is omitted. The STS$ action routine uses this default level when scheduling tasks in this activity.

Reentrant Activities

A reentrant activity is one which may be interrupted and executed again on behalf of a different set of input parameters without destroying the interrupted state. The interrupted activity may then be resumed as though no interruption had taken place.

The executive saves the state of the executing task when the activity is interrupted and restores this state when the task is resumed. The state which is preserved and restored includes:

The hardware registers: A, B, X, S, Y, banks and keys register.
Three pseudoregisters: ZCR1, ZCR2, ZCR3.

The three pseudoregisters are provided by the executive and may be referenced from an activity indirectly through pointer words (see the TMA$ function examples for illustrations of this technique).

To make a set of task code reentrant, the user must obtain a block of free memory for a variable data area, and store the address of this block in one of the hardware registers or pseudoregisters.

Activity Interaction

Activity interaction may occur in two ways: scheduling and monitoring. Using the SAC$ executive function, an activity can schedule a second activity and allow it to execute to completion by itself; or an activity can cause a second activity to be brought into main memory and subsequently control (monitor) the executive of the second activity. To allow monitoring to take place, after reading the second activity into memory the executive must pass control to the monitoring activity instead of starting execution of the second activity. To cause this to happen, the monitoring activity uses the secondary TCB parameter of the SAC$ function call. This secondary TCB action is supported only in DOS.

The secondary TCB defines a secondary task within the monitoring activity. When the second activity is brought into main memory, this secondary task is scheduled in place of the lead task of the second activity.

When the secondary task is dispatched, the monitoring activity has control with the second activity being main-memory-resident, and monitoring may take place. When the secondary task is dispatched, the X-register contains the address of the secondary TCB. The address of the second activity's primary (lead task) TCB is in the word whose displacement in ZTCBPT with respect to the beginning of the secondary TCB. It is the responsibility of the monitoring activity to ensure that the second activity is terminated. This may be accomplished in either of two ways:

- Schedule the lead task of the second activity and allow the second activity to execute to completion and terminate itself.
- Terminate the second activity from within the monitoring activity (DOS only).

The latter method of terminating the second activity involves use of special capabilities of the TMA$ function. One of the parameters of the TMA$ function call defines a TCB which is to be returned to free memory upon termination of the activity. The monitoring activity, wishing to terminate the second activity, can issue a TMA$ function call, with the TCB of the lead task of the second activity as a parameter. The TMA$ action routine terminates the second activity, but returns control to the monitoring activity, rather than to the executive.

NOTE: Regardless of the method used to terminate the second activity, the monitoring activity must terminate itself. In addition, the monitoring and the second activities must be main-memory-resident at the same time; therefore, they must occupy different nonoverlapping activity areas.

Scheduling a Restricted Activity With a Secondary TCB

When a nonrestricted activity schedules a restricted activity with a secondary TCB, the following rules must be observed:

- The secondary TCB must not start up a task within a restricted activity.
- Only the primary TCB may be used to schedule execution of the restricted activity.
- The monitoring activity must not start up more than one task within the restricted activity.

Clock Activities

The two-clock activity functions are used to schedule activities by means of the clock, and to stop such scheduling. The functions are: Connect Clock Activity (CCA$) and Disconnect Clock Activity (DCL$).

Clock activities can be scheduled to be executed once after a specified delay (noncyclic), or repeatedly at regular intervals (cyclic). A cyclic activity must be disconnected when no further executions of the activity are required; a noncyclic activity does not have to be disconnected.

Care must be taken to ensure that a clock activity is not scheduled cyclically at too short an interval. Otherwise, if the activity is scheduled a second time before it has finished executing the first time, scheduling requests may mount up until the system runs short of free memory blocks. The user must ensure that the necessary activity area is free for a clock activity to run. If it is not, the activity will still be scheduled at the specified intervals, and eventually the system could run out of free memory.

Schedule Activity (SAC$)

A request for the execution of an activity is made through the SAC$ function.

FUNCTION ACTION

The SAC$ action routine creates a TCB for the activity lead task after obtaining a block from free memory. If immediate scheduling was requested, the activity is dispatched at once or the error return is taken. Otherwise, SAC$ then places the request to schedule the activity in the activity request queue and returns control to the calling program. Later, when the activity's memory area becomes available, the activity supervisor brings the activity into main memory and schedules the lead task of the activity (or in a DOS, the task defined by the secondary TCB).

If a restricted activity attempts to schedule a nonrestricted activity or to schedule another activity with a secondary TCB, an error status will be returned.

MACRO CALL FORMAT

Location	Operation	Operand
[symbol]	SAC$	activity name block address, [level], 6 Characters [parameter 1], [parameter 2], error return address, [secondary TCB address], [schedule immediate]

symbol - Optional. The symbolic location of the SAC$ macro instruction.

activity name block address - The address of the first word of the 6-character name of the activity. The name is left-justified, right-filled with spaces.

level - Optional. The user priority level number at which the lead task of the activity is to be scheduled. If a level number lower than the lowest user-priority level is specified, the lowest user-priroity level will be used to schedule the task. If this parameter is omitted, the task is scheduled on the default level specified in the ACB of the activity at activity load time.

parameter 1 - Optional. A parameter word that can be used by the calling program to pass data to the lead task. The action routine puts this parameter in word ZTCBP1 of the lead task TCB. If this parameter is omitted, word ZTCBP1 of the TCB is set to zero. Note that if parameter 2 is not zero, parameter 1 must not be zero unless the free memory block parameter passing technique is used (see Appendix G, "Free Memory Block Parameter Passing Technique").

parameter 2 - Optional. A parameter word that can be used by the calling program to pass data to the lead task. The action routine puts this parameter in word ZTCBP2 of the lead task TCB. If this parameter is omitted, word ZTCBP2 of the TCB is set to zero. Note that if parameter 1 is zero, parameter 2 must be zero unless the free memory block parameter passing technique is being used (see Appendix G, "Free Memory Block Parameter Passing Technique").

error return address - The address to which control is returned if an error is found during the processing of the SAC$ function call.

secondary TCB address - Optional. Specifies the TCB to be scheduled. If this parameter is present, it specifies the address of a secondary TCB (defining a secondary task) which will be dispatched in place of the lead task of the activity (supported only in DOS). If this parameter is omitted, the lead task of the activity will be dispatched.

schedule immediate - Optional. A number that indicates whether the request to schedule an activity is to be queued if the activity area in which the scheduled activity resides is occupied by another activity.

If the schedule immediate parameter is a 1 in the SAC$ parameter list and the secondary TCB address is also specified in the SAC$ parameter list, the request to schedule the activity is not queued. Therefore, if the activity area in which the scheduled activity resides is occupied by another activity, the error return is taken to the calling program.

If the schedule immediate parameter is zero or omitted in the SAC$ parameter list, or if the secondary TCB address is not specified in the SAC$ parameter list, the request to schedule an activity is queued if the activity area in which the scheduled activity is to reside is occupied by another activity.

NORMAL RETURN

Upon normal return, the request for the activity has been queued. The lead (or secondary) task may already have been scheduled if the activity is reentrant or reusable. If not already scheduled, the lead (or secondary) task will be scheduled at a later time.

ERROR RETURN

Control is returned to the error return address specified in the SAC$ parameter list with the error code in the A-register when any of the following errors is detected:

A-Register Contents (Octal)	Error Condition
16	Activity not on disk.
26	No activity area for the activity (starting address given in the disk directory is not equal to the starting address of any activity area), or the activity is too large for the allocated activity area.
33	The schedule immediate parameter in the SAC$ parameter list is set, the secondary TCB address is specified in the SAC$ parameter list, and the activity area where the scheduled activity is to reside is occupied by another activity.

A-Register Contents (Octal)	Error Condition
34	Requested executive function is not configured.
35	Requested executive function is disk-resident and there was a disk error during an attempt to load the function into main memory.
67 (See Note)	I/O error on activity directory segment during input.
71 (See Note)	I/O error on activity descriptor directory segment during input.
107	Restricted activity specified a secondary TCB when scheduling another activity.
170	Restricted activity attempted to schedule a non-restricted activity.
171	The requested activity is restricted and cannot be scheduled because free memory is low.
172	The requested activity is being aborted.

NOTE: If the A-register indicates an I/O error, the X-register contains the following additional information:

- If bits 6 and 7 of the A-register contain the value 01_2, the X-register contains the setup error code from the I/O request. (See Appendix A for error codes returned from the INP$ and OTP$ function calls.)
- If bits 6 and 7 of the A-register contain the value 10_2, the X-register contains the software status from the I/O status block.
- If bits 6 and 7 of the A-register contain the value 11_2, the X-register contains the hardware status from the I/O status block. Refer to Appendix B for software and hardware status information.

Because an activity can be loaded after the SAC$ macro call has been processed, a load error cannot be returned to the error return address specified in the parameter list. When a load error occurs under this condition, an error message is sent to the operator:

Operator Message	Error Condition
SE= 100030 000000 actnam	Activity area will be overrun.
SE= 100031 000000 actnam	Disk error while loading the activity.

actnam - The name of the activity being loaded.

ACTION ROUTINE DETAILS

If the specified activity is not already resident in main memory (COS only) or on the disk (defined; DOS only), the error return is taken. The error return is also taken when the activity is defined to OS/700, the schedule immediate parameter and the secondary TCB address are specified in the SAC$ parameter list, and the activity area in which the scheduled activity is to reside is occupied by another activity.

If the activity is defined to OS/700, the request for execution of the activity is queued, if necessary, and the normal return is taken.

Later, when the area for the activity becomes available, the activity is brought into main memory from the disk and the lead task is scheduled. If, in a DOS system, the secondary TCB has been specified, the secondary task is scheduled.

OUTLINE PARAMETER LIST

Word Number	Operation	Operand
0	DAC	activity name address
1	DEC	[level]
2	DAC	[parameter 1]
3	DAC	[parameter 2]
4	DAC	error return address
5	DAC	[secondary TCB address]

Parameters in the outline parameter list are the same as those described above for the inline parameter list. If any of the optional parameters are omitted, a BSZ 1 statement must replace the respective statement.

NOTE: The outline parameter list does not include the schedule immediate parameter. To use this parameter as described above, the sign bit of word 1 must be set as follows:

Word Number	Operation	Operand
1	OCT	1000xx (where xx is the level)

When the lead task or the secondary task is dispatched, the location of the lead or secondary TCB will be in the X-register. The words which are of interest to the user are:

ZTCBST - Status word.

ZTCBP1 - Parameter 1.

ZTCBP2 - Parameter 2.

ZTCBPT - Pointer to lead task TCB if secondary task has been dispatched.

The above symbols, defined by a TCB$ macro call, have addresses which are relative to word 0 of the TCB. Thus, for example, if upon start of the lead task parameter 2 must be examined by the user, the coding would be:

```
LDA ZTCBP2,1 - The index register points to the start of the TCB.
```

The status word appears in either a lead task TCB or the secondary task TCB. It has the following meaning:

Bits 1 and 2 - Task designation (binary)

00 - Reserved.

01 - TCB is for lead task.

10 - TCB is for secondary task.

11 - Reserved.

Bits 3 through 16 - Status (decimal)

0 - No errors.

2 - Activity area overrun error while loading the activity.

3 - Disk read error while loading the activity.

Status number 2 and 3 are errors returned to the secondary task only if the secondary task is not part of the same activity as the lead task.

The user is responsible for generating a secondary TCB. The Create TCB function (CTC$) can be used to do this.

Examples:

In the following example the activity ACTIV3 is scheduled immediately. ACTIV3 is to run at priority level 5 and requires the passing of one parameter. As no secondary TCB is provided, the lead task of ACTIV3 will be executed.

```
        SAC$  ACTNAM,5,'302,,ERR1,,1  SCHEDULE ACTIVITY
         .
         .
         .
ACTNAM  BCI   3,ACTIV3          ACTIVITY NAME BLOCK
*
ERR1    ---                     ERROR RETURN
```

Using an outline parameter list, the above example would be coded as follows:

```
         LDX    PLIST           SET UP POINTER TO OUTLINE LIST

         SAC$   (X)             SCHEDULE THE ACTIVITY
          .
          .
          .
PLIST    DAC    *+1             POINTER TO OUTLINE LIST

         DAC    ACTNAM          ACTIVITY NAME BLOCK ADDRESS

         OCT    100005          PRIORITY LEVEL 5 - SCHEDULE IMMEDIATELY

         OCT    302             PARAMETER 1

         BSZ    1               DEFAULT

         DAC    ERR1            ERROR RETURN ADDRESS

         BSZ    1               NO SECONDARY TCB

*
ACTNAM   BCI    3,ACTIV3        ACTIVITY NAME BLOCK
*
ERR1     ---                    ERROR RETURN
```

NOTE: The activity scheduling parameter is not a self-standing parameter in the outline parameter list form, but is the sign bit of the level parameter.

Terminate Activity (TMA$)

The TMA$ function call is used to terminate either the activity in which it appears or another activity. Termination of an activity by another activity is supported only in DOS. A restricted activity can terminate only itself.

FUNCTION ACTION

The TMA$ action routine returns the TCB to free memory, if the TCB is to be released. TMA$ then terminates the activity and returns control to the calling program or exits to the executive, which dispatches the highest priority scheduled task.

MACRO CALL FORMAT

Location	Operation	Operand
[symbol]	TMA$	[TCB address pointer], error return address

symbol - Optional. The symbolic location of the TMA$ macro instruction.

TCB address pointer - Optional. The address of a word containing the address of the task control block associated with the activity to be terminated. The activity associated with the activity control block pointed to by the task control block is terminated. If this parameter is omitted, the current activity will terminate itself and the TCB is not returned.

error return address - The address to which control is returned if an error is detected during the processing of the TMA$ function call.

NORMAL RETURN

Control is returned to the calling program, at the instruction following the TMA$ function call, if an activity is terminated another activity (DOS only). If an activity is terminating itself, the TMA$ function exits to the executive, and control does not return to the calling program.

ERROR RETURN

When any of the following errors is detected, control is returned to the error return address specified in the TMA$ parameter list with the error code in the A-register.

A-Register Contents (Octal)	Error Condition
34	Requested executive function is not configured.
35	Requested executive function is disk-resident and there was a disk error during an attempt to load the function into main memory.
174	The activity whose termination is requested is not running.

ACTION ROUTINE DETAILS

If the TCB address pointer parameter is present, the TCB is examined to determine if its block is to be returned to free memory. If so, the block is released. If the TCB address pointer parameter is omitted or if the parameter is the address of a word which contains zero, no block is returned to free memory.

The activity associated with the ACB pointed to by the TCB specified in the TMA$ parameter list is the activity to be terminated. If the parameter is omitted, the current activity is terminated. This termination may result in the lead task or secondary task being scheduled if the activity is reusable or non-reusable and requests are queued. If no requests are queued for the terminating activity or if this request is the last of the dispatched requests for a re-entrant activity, the activity area occupied by the activity is freed for use by other activities that are queued for the same activity area. If no requests are queued for the same activity area, that area of main memory is freed for use by any overlapping activity area that has queued requests.

TMA$ then determines if it was called by an activity terminating itself or by an activity terminating another activity. If the calling activity terminates itself, TMA$ exits to the dispatcher and does not return to the calling activity. If the calling activity terminates another activity (DOS only), TMA$ returns to the calling program at the instruction following the TMA$ function call.

NOTE: Before using the TMA$ function to terminate an activity, the user must perform the following:

1. Make sure all I/O operations initiated by the activity are complete.
2. Release all I/O devices reserved by the activity.
3. Close all files opened by the activity.
4. Close all libraries opened by the activity.
5. Return all blocks obtained from free memory by the activity.
6. Disconnect all clock tasks that reside in the activity.
7. Make sure all tasks in the activity have terminated.
8. Disconnect all volumes explicitly connected by the activity.

If the activity was a restricted activity, these items (except for items 5, 6 and 7, which are not applicable) will be performed automatically by the system.

OUTLINE PARAMETER LIST

Word Number	Operation	Operand
0	DAC	[TCB address pointer]
1	DAC	error return address

Parameters in the outline parameter list are the same as those described above for the inline parameter list. If the TCB address pointer is omitted, a BSZ 1 statement must replace the respective statement.

Examples:

In the following example, the activity is terminated using a nonreentrant inline parameter list.

```
START  STX     PTTCB           SAVE THE TCB ADDRESS
        ⋮
       TMA$    PTTCB,ERA       TERMINATE ACTIVITY
        ⋮
ERA    HLT                     ERROR HALT
        ⋮
PTTCB  DAC     **              TCB ADDRESS
```

In the following example, the activity is terminated using an inline parameter list but is reentrant because the parameter is in a pseudo-register.

```
START  STX*    AZCR1           SAVE LEAD TASK TCB ADDRESS
        ⋮
       GBL$    8,,,GBLERR      GET BLOCK FOR PARAMETER LISTS
       STX*    AZCR2           SAVE BLOCK ADDRESS
        ⋮
GBLERR ----                    ERROR ON BLOCK FETCH
        ⋮
       LDX*    AZCR2           SET UP OUTLINE PARAMETER LIST
       LDA     ASTRT2          ESTABLISH 2ND TASK'S START ADDRESS
       STA     0,1
       CRA
       STA     1,1             DEFAULT TO SAME LEVEL AS THIS ACTIVITY
       STA     2,1             DEFAULT TO RUN UNDER THIS ACTIVITY
       LDA*    AZCR2           SAVE BLOCK ADDRESS
```

```
         STA     3,1
         LDA*    AZCR3         SAVE THE VARIABLE BLOCK WHICH WAS FOUND
         STA     4,1           PREVIOUSLY
         LDA     AERRX         ESTABLISH ERROR RETURN ADDRESS
         STA     5,0
         CTC$    (X)           CREATE 2ND TASK'S TCB
         STX*    AZCR3         SAVE 2ND TASK'S TCB ADDRESS
         STC$    ZCR3,ERRY     SCHEDULE THE 2ND TASK
           ⋮
AERRX    DAC     ERRX
ERRX     ----                  ERROR ON TCB CREATION
           ⋮
ERRY     ----                  ERROR ON SCHEDULING 2ND TASK
           ⋮
         TMT$    ZCR1          TERMINATE THE LEAD TASK
           ⋮
*        2ND TASK --- SCHEDULED BY LEAD TASK
*
ASTRT2   DAC     STRT2
STRT2    STX*    AZCR1         SAVE 2ND TASK'S TCB ADDRESS
         LDA     3,1
         STA*    AZCR2         PRESERVE TCB ADDRESS
         LDA     4,1
         STA*    AZCR3         PRESERVE VARIABLE BLOCK ADDRESS
           ⋮
*        RETURN VARIABLE PARAMETER BLOCK
*        (ASSUME THE BLOCK SIZE WAS PLACED IN WORD 0 OF THE BLOCK
*        ITSELF WHEN FETCHED IN THE LEAD TASK)
         LDX*    AZCR2         SET UP OUTLINE PARAMETER LIST
         LDA*    AZCR3         ESTABLISH ADDRESS OF BLOCK
         STA     3,1
         LDAQ    X             CONSTRUCT POINTER TO THAT ADDRESS
         ADD     =3
         STA     0,1           USE IT AS PARAMETER 1
         LDA*    AZCR3         FETCH ADDRESS OF THE BLOCK
         LDAQ*   A             ITS SIZE IS IN WORD 0
         STA     1,1           SAVE AS PARAMETER 2
         LDA     AERRA         SET UP ERROR RETURN ADDRESS
         STA     2,1           USE AS PARAMETER 3
         RBL$    (X)           RETURN THE BLOCK
           ⋮
*        RETURN CANNED BLOCK FETCHED IN LEAD TASK
*
```

```
         RBL$      ZCR2,8,ERRB    RETURN THE BLOCK
           ⋮
AERRA    DAC       ERRA
ERRA     ----                     ERROR ON VARIABLE BLOCK RETURN
           ⋮
ERRB     ----                     ERROR ON CANNED BLOCK RETURN
           ⋮
*        TERMINATE THE SECOND TASK
*
         TMA$      ZCR1,ERRC      TERMINATE THE ACTIVITY
           ⋮
ERRC     HLT                      ERROR ON TERMINATION
           ⋮
         LNK$                     PROVIDE LINKAGE AND PSEUDOREGISTER
*                                 DEFINITIONS
*
AZCR1    DAC       ZCR1           USER ACCESSIBLE PSEUDOREGISTERS ADDRESSES
AZCR2    DAC       ZCR2           (ACCESS INDIRECTLY TO ATTAIN TRUE SECTOR
AZCR3    DAC       ZCR3           ZERO WITHOUT J-BASE OFFSET)
```

In the following example, the activity is terminated using a nonreentrant outline parameter list.

```
         LDX       PLISTA         SET UP POINTER TO OUTLINE LIST
         TMA$      (X)            TERMINATE THE ACTIVITY
           ⋮
ERRA     HLT                      ERROR HALT
           ⋮
PLISTA   DAC       PLIST          OUTLINE LIST ADDRESS
PLIST    DAC       PTTCB          PARAMETER LIST, TCB ADDRESS POINTER
         DAC       ERRA           ERROR RETURN ADDRESS
           ⋮
PTTCB    DAC       TCB            TCB ADDRESS
TCB      BSZ       8              TCB
```

NOTE: There is no need to use a reentrant outline parameter list for terminating an activity because pseudoregisters are available for saving the task's TCB address.

Abort Activity (ABT$)

The ABT$ function enables a user program to abort a restricted activity.

FUNCTION ACTION

The ABT$ action routine aborts the named activity. ABT$ can be called only by a nonrestricted activity, and the activity to be aborted must be a restricted activity which is running or has been requested. If a restricted activity attempts to call this function, it will itself be aborted.

The ABT$ function is available only in DOS with system integrity.

MACRO CALL FORMAT

Location	Operation	Operand
[symbol]	ABT$	activity name block pointer, error return address

symbol - Optional. The symbolic location of the ABT$ macro instruction.

activity name block pointer - The address of a 3-word block containing the name, in ASCCI, of the activity to be aborted.

error return address - The address to which control is to be returned if an error is detected during the processing of the ABT$ function call.

NORMAL RETURN

Control is returned to the calling program at the instruction following the ABT$ function call after the abort procedure has been initiated.

ERROR RETURN

When any of the following errors is detected, control is returned to the error return address specified in the ABT$ parameter list with the error code in the A-register.

A-Register Contents (Octal)	Error Condition
34	Requested executive function is not configured.
35	Requested executive function is disk-resident, and there was a disk error during an attempt to load the function into main memory.
115	The activity to be aborted is not a restricted activity, or was not requested.

ACTION ROUTINE DETAILS

The ABT$ action routine searches the Resident/Requested Activity Directory (RRAD) for an ACB containing the specified activity name. The RRAD is a queue in main memory containing the ACB's of all activities currently requested or resident in memory. This includes activities which have been requested but have not yet been brought into memory, activities which have been requested and brought into memory but not yet dispatched, activities currently executing, and reusable activities which have terminated but are still intact in memory.

If the ACB is not found in the RRAD, the error return to the caller is taken with the appropriate error code in the A-register. If the ACB is found, but indicates that the named activity is nonrestricted, the error return to the caller is taken with the appropriate error code in the A-register.

Otherwise, the action routine sets the abort flag in the ACB, schedules the abort task if this is not already running, and returns to the calling program at the normal return.

If the named activity is not yet dispatched, the Activity Supervisor will examine the ACB and refuse to proceed with the scheduling of the activity when it sees that the abort flag is set. If the activity is already running, the system will refuse to resume it or return to it from an executive function, and the abort task will supervise the return of any system resources (open files, reserved devices etc.) held by the activity. When all cleanup work has been done, the ACB will be removed from the RRAD.

OUTLINE PARAMETER LIST

Word Number	Operation	Operand
0	DAC	activity name block pointer
1	DAC	error return address

Parameters in the outline parameter list are the same as those described above for the inline parameter list.

Examples:

The example below shows the use of the ABT$ function to abort the restricted activity named RESTAC.

```
          ABT$    ACTNAM,ABTERR ABORT THE ACTIVITY "RESTAC"
            :
ABTERR ----                     ERROR RETURN
            :
ACTNAM BCI        3,RESTAC      ACTIVITY NAME BLOCK
```

Using an outline parameter list, the above example would be written:

```
         LDX      PLIST          SET UP POINTER TO OUTLINE LIST
         ABT$     (X)            ABORT THE ACTIVITY "RESTAC"
          .
          .
          .
ABTERR   ----                    ERROR RETURN
          .
          .
          .
PLIST    DAC      *+1            ADDRESS OF OUTLINE LIST
         DAC      ANAME          POINTER TO ACTIVITY NAME BLOCK
         DAC      ABTERR         ERROR RETURN ADDRESS
          .
          .
          .
ACTNAM   BCI      3,RESTAC       ACTIVITY NAME BLOCK
```

Connect Clock Activity (CCA$)

A request for the execution of an activity on a timed basis (either cyclic or noncyclic) is made by the CCA$ function.

FUNCTION ACTION

The CCA$ action routine connects a special system clock task to the timer specified in the call. The routine generates a SAC$ function call parameter list and passes it as a parameter to this special task.

CCA$ then returns control to the calling program at the normal return. When the time expires, the activity is scheduled.

A restricted activity cannot call the CCA$ function. If it attempts to do so, the activity will be aborted.

MACRO CALL FORMAT

Location	Operation	Operand
[symbol]	CCA$	activity name address, [level], [number of time units], time unit type, [connect type], [parameter 1], [parameter 2], error return address

symbol - Optional. The symbolic location of the CCA$ macro instruction.

activity name address - The address of the first word of the 6-character name of the activity. The name is left-justified, right-filled with spaces.

level - Optional. The user-priority level number on which the task is to be scheduled. If a level number lower than the lowest user-priority level is specified, the lowest user-priority level will be used to schedule the task.

If this parameter is omitted, the task is scheduled on the default level specified for the activity.

number of time units - Optional. An integer expressing the number of time units from now until the activity is to be scheduled.

If omitted, zero is used.

The range of this parameter depends on the time unit type parameter.

time unit type - An integer specifying the units of time in which the number of time units parameter is expressed.

1 = millisecond. The range of number of time units is 0 through 1023.

2 = half-second. The range of number of time units is 0 through 4095.

4 = second. The range of number of time units is 0 through 4095.

8 = minute. The range of number of time units is 0 through 4095.

If omitted, the error return is taken with the value 1 in the A-register.

connect type - Optional. An integer specifying whether the activity is to be scheduled on a cyclic or noncyclic basis.

0 = cyclic.

1 = noncyclic.

If omitted, zero (cyclic) is used.

parameter 1 - Optional. A parameter word that can be used by the calling program to pass data to the lead task. The action routine puts this parameter in word ZTCBP1 of the lead task TCB. If this parameter is omitted, a zero word is generated in the TCB for word ZTCBP1. If parameter 2 is nonzero, parameter 1 must not be zero unless the free memory block parameter passing technique is being used (see Appendix G, "Free Memory Block Parameter Passing Technique").

parameter 2 - Optional. A parameter word that can be used by the calling program to pass data to the lead task. The action routine puts this parameter in word ZTCBP2 of the lead task TCB. If this parameter is omitted, a zero word is generated in the TCB for word ZTCBP2. If parameter 1 is zero, parameter 2 must be zero unless the free memory block parameter passing technique is being used (see Appendix G, "Free Memory Block Parameter Passing Technique").

error return address - The address to which control is returned if an error is found during the processing of the CCA$ macro call.

NORMAL RETURN

Upon normal return, the special system clock task has been connected to the proper timer queue. The parameter to this special clock task is the pointer to the SAC$ parameter list needed to schedule the activity. Also upon normal return, the X-register contains the address of the TCB used to connect the special system task to the clock. This address must be saved if a subsequent DCA$ (Disconnect Clock Activity) function call is to be issued.

ERROR RETURN

Control is returned to the error return address specified in the CCA$ parameter list with the error code in the A-register when any of the following errors is detected:

A-Register Contents (Octal)	Error Condition
1	Timer not configured.
34	Requested executive function is not configured.
35	Requested executive function is disk-resident and there was a disk error during an attempt to load the function into main memory.

Because the activity is scheduled at a later time (when the special system task is dispatched), errors discovered by the schedule activity action are reported to the operator.

Operator Message				Error Condition
SE=	100116	000036	actnam	Activity not on disk.
SE=	100122	000036	actnam	Disk error.
SE=	100126	000036	actnam	There is no activity area for the activity, or activity is too large to fit into its area.
SE=	100030	000000	actnam	Activity area will be overrun.
SE=	100031	000000	actnam	Disk error while loading the activity.

actnam - The name of the activity.

ACTION ROUTINE DETAILS

A clock TCB is generated for the special system clock task. A SAC$ parameter list is generated. The address of this list is placed in the clock TCB to be passed as a parameter to the special system clock task.

After the special system clock task is connected to the proper timer queue, return is made at the normal return.

At the time specified, the special system clock task is dispatched. This task issues a schedule activity request, using the SAC$ parameter list passed to it as a parameter. At this time, the regular SAC$ action proceeds (see description of the SAC$ function).

If the activity is to be cyclic, the special system task remains connected to the clock, so that the activity is rescheduled when the time interval has elapsed once more. This action continues until the clock activity is disconnected by the DCA$ (Disconnect Clock Activity) function. If the activity is to be noncyclic, the special system clock task is disconnected from the clock, after the activity is scheduled.

OUTLINE PARAMETER LIST

Word Number	Operation	Operand
0	DAC	activity name address
1	DEC	[level]
2	DEC	[number of time units]
3	DEC	time unit type
4	DEC	[connect type]
5	DAC	[parameter 1]
6	DAC	[parameter 2]
7	DAC	error return address

Parameters in the outline parameter list are the same as those described above for the inline parameter list. If any of the optional parameters are omitted, a BSZ 1 statement must replace the respective statement.

NOTE: The special system clock task is scheduled on the same level as that specified for the level parameter, unless this parameter is omitted or zero. In this case, the special system clock task is scheduled on the highest user level.

Examples:

The example below illustrates the use of the CCA$ function to connect an activity to the clock for noncyclic execution. The activity named ACTNAM is to be connected to the clock for scheduling 4 hours (240 minutes) from now. The level at which the activity is scheduled is the default level. One parameter, the address T1, is passed to the activity.

```
       CCA$    ANAME,,240,8,1,T1,,ERR    DEFAULT LEVEL,SECOND PARAMETER
       STX     ATCB               SAVE ADDRESS OF SPECIAL SYSTEM TASK TCB
        :
ANAME  BCI     3,ACTNAM           ACTIVITY NAME BLOCK
*
T1     BSZ     30                 PASS ADDRESS TO ACTIVITY AS PARAMETER 1
ATCB   BSZ     1
        :
ERR    ----                       ERROR RETURN
```

Using an outline parameter list, the above example would be written as follows:

```
         LDX       PLIST            SET UP POINTER TO OUTLINE LIST
         CCA$      (X)
         STX       ATCB
          .
          .
          .
PLIST    DAC       *+1              POINTER TO OUTLINE LIST
         DAC       ANAME            ACTIVITY NAME ADDRESS
         BSZ 1     0                DEFAULT LEVEL
         DEC       240              240 MINUTES
         DEC       8                MINUTES
         DEC       1                NONCYCLIC
         DAC       T1               PARAMETER 1
         BSZ 1     0                DEFAULT PARAMETER 2
         DAC       ERR              ERROR RETURN ADDRESS
*
ANAME    BCI       3,ACTNAM
T1       BSZ       30
ATCB     BSZ       1
*
ERR      ----                       ERROR RETURN
```

DCA$

Disconnect Clock Activity (DCA$)

The DCA$ function is used to disconnect a clock activity from the timer queue on which it resides.

FUNCTION ACTION

The DCA$ action routine removes the clock TCB from the timer queue on which it resides, thus disconnecting the activity from the clock. It then returns all associated blocks to free memory and returns control the caller.

A restricted activity cannot call the DCA$ function. If it attempts to do so, the activity will be aborted.

MACRO CALL FORMAT

Location	Operation	Operand
[symbol]	DCA$	TCB address pointer, error return address

symbol - Optional. The symbolic location of the DCA$ macro instruction.

TCB address pointer - The address of the word in which the address of the clock TCB has been stored. This TCB address has been returned to the caller in the X-register by a CCA$ function call.

error return address - The address to which control is returned if an error is found during the processing of the DCA$ function call.

NORMAL RETURN

Upon the normal return, the clock activity is disconnected from the clock.

ERROR RETURN

Control is returned to the error return address specified in the DCA$ parameter list when any of the following errors is detected:

A-Register Contents (Octal)	Error Condition
34	Requested executive function is not configured.
35	Requested executive function is disk-resident and there was a disk error during an attempt to load the function into main memory.
Not Significant	The specified clock TCB was not found in the timer queue. (There is no specific error code associated with this error.)

ACTION ROUTINE DETAILS

The special system TCB is removed from the timer queue on which it resides. If, in a search of the timer queue, the given TCB cannot be found on the queue, the error return is taken. After the TCB is removed, all associated blocks are returned to free memory.

OUTLINE PARAMETER LIST

Word Number	Operation	Operand
0	DAC	TCB address pointer
1	DAC	error return address

Parameters in the outline parameter list are the same as those described above for the inline parameter list.

Examples:

In the example below, an activity is to be disconnected from the clock. At some prior time, the activity has been connected to the clock and the address of the special clock task TCB has been stored in word ATCB (see the description of the CCA$ function). If this TCB cannot be found in the clock queue, control returns to NOTCB.

```
        DCA$    ATCB,NOTCB
          .
          .
ATCB    BSZ     1               CONTAINS TCB ADDRESS
          .
          .
NOTCB   ----                    ERROR RETURN
```

Using an outline parameter list, the above example (with expanded error analysis) would be written as follows:

```
        LDX     PLIST           SET UP POINTER TO OUTLINE LIST
        DCA$    (X)
          .
          .
PLIST   DAC     *+1             POINTER TO OUTLINE LIST
        DAC     ATCB            TCB ADDRESS POINTER
        DAC     NOTCB           ERROR RETURN ADDRESS
          .
          .
ATCB    BSZ     1               STORAGE FOR TCB ADDRESS
          .
          .
NOTCB   CAS     ='34            ERROR RETURN
        NOP                     ASSESS THE CAUSE
        CAS     ='36
        NOP
        SKP                     TCB IS NOT FOUND IN QUEUE
        JMP     FNCERR          FUNCTION LOAD OR CONFIGURATION PROBLEM
```

SECTION V

QUEUE AND FREE MEMORY BLOCK MANAGEMENT

QUEUE MANAGEMENT

The queue executive functions allow the user to create a queue, add items to the beginning (top) or end (bottom) of the queue and remove items from the beginning of the queue. In actual operation, the Create Queue function (CRQ$) sets up a 2-word queue header. As items are added to the beginning or end of the queue, the first word of the queue header points to the first item in the queue and the second word of the queue header points to the last item in the queue.

The first word of each item in the queue points to the next item in the queue. (The first word of the last item in the queue therefore is always zero.) Pointer operation is illustrated in Figure 5-1.

The queue functions are:

- Create Queue (CRQ$)
- Attach Entry to Queue (ATQ$)
- Get Beginning Entry From Queue (GTQ$)

A queue must first be created using the CRQ$ function. Thereafter, any reference to this queue, for the purpose of queue management, must be made through the queue header address given to the CRQ$ system function. The ATQ$ function permits an entry to be made either to the beginning or the end of a queue. Thus LIFO (last-in, first-out), and FIFO (first-in, first-out) queues can be generated. A LIFO queue is generated by always attaching new items to the beginning of the queue. A FIFO queue is generated by always attaching new items to the end of the queue.

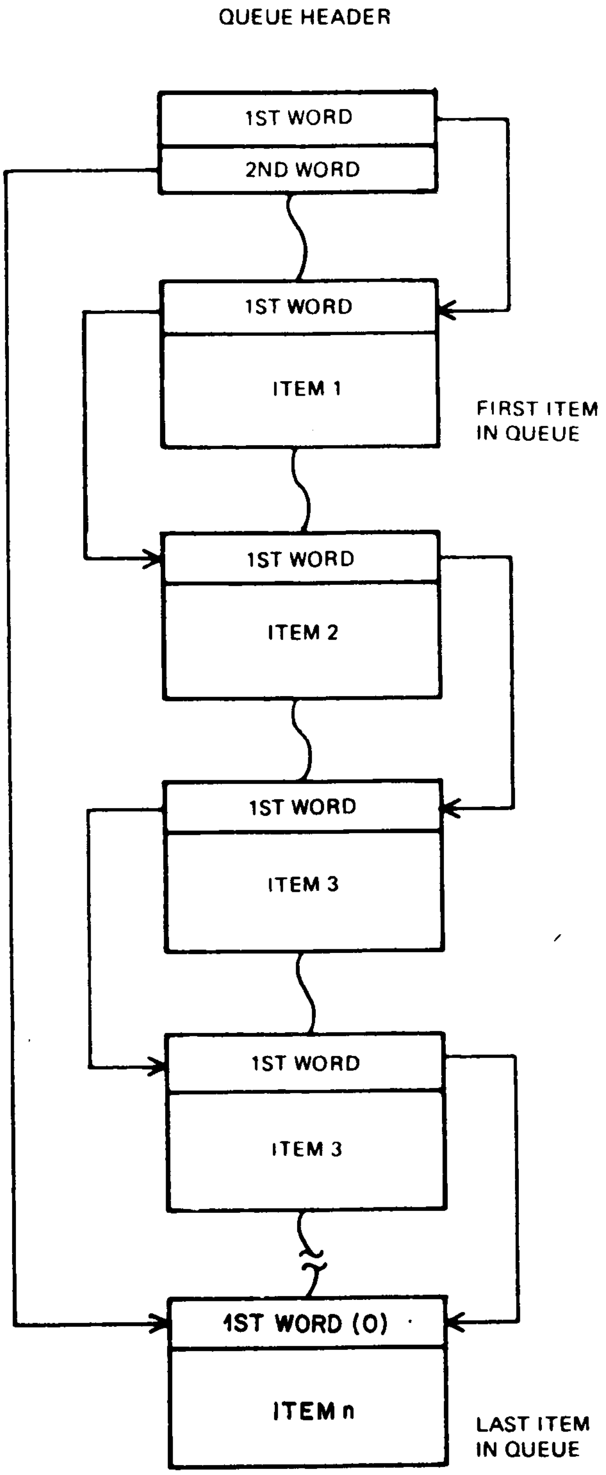


Figure 5-1. Queue Operation

Create Queue (CRQ$)

A queue can be initialized through the CRQ$ function.

FUNCTION ACTION

The CRQ$ system function initializes the queue as being empty and returns control to the calling program at the normal return.

MACRO CALL FORMAT

Location	Operation	Operand
[symbol]	CRQ$	queue header address, error return address

symbol - Optional. The symbolic location of the CRQ$ macro instruction.

queue header address - The address of the first word of the queue header.

error return address - The address to which control is returned if an error is found during the processing of the CRQ$ function call.

NORMAL RETURN

Upon normal return, the named queue header has been initialized.

ERROR RETURN

Control is returned to the error return address specified in the CRQ$ parameter list with the error code in the A-register when any of the following errors is detected:

A-Register Contents (Octal)	Error Condition
34	Requested executive function is not configured in the system.
35	Requested execution function is disk-resident and there was a disk error during an attempt to load the function into main memory.

ACTION ROUTINE DETAILS

The two words of the queue header, identified by the queue header address parameter, are assigned the value of zero, initializing the queue as an empty queue. Control returns to the calling program at the normal return.

OUTLINE PARAMETER LIST

Word Number	Operation	Operand
0	DAC	queue header address
1	DAC	error return address

Parameters in the outline parameter list are the same as those described above for the inline parameter list.

Example:

The following example illustrates the use of the CRQ$ function to initialize the queue header specified by RTRQ.

```
*                                CREATE A QUEUE HEADER
*
        CRQ$    RTRQ,ERR
*
*                                THE FOLLOWING WILL BE A QUEUE HEADER
*
RTRQ    BSZ     1                START OF QUEUE
        BSZ     1                END OF QUEUE
ERR     ---                      ERROR RETURN
```

Attach Entry to Queue (ATQ$)

The ATQ$ function is used to permit an entry to be attached either to the beginning or the end of a queue.

FUNCTION ACTION

The ATQ$ action routine links the entry either to the beginning or the end of the queue and returns to the calling program at the normal return.

MACRO CALL FORMAT

Location	Operation	Operand
[symbol]	ATQ$	queue header address, [queue attach mode], queue entry block address, error return address

symbol - Optional. The symbolic location of the ATQ$ macro instruction.

queue header address - The address of the first word of the queue header. This address is the same as used in the CRQ$ function.

queue attach mode - Optional. An integer specifying the place in the queue to attach the entry.

0 - Attach to end of queue.

1 - Attach to beginning of queue. If omitted, zero is used.

queue entry block address - The address of the first word of the entry block to be linked to the queue. The first word becomes the link word.

error return address - The address to which control is returned if an error is found during the processing of the ATQ$ function call.

NORMAL RETURN

Upon normal return, the entry is linked either to the beginning or the end of the specified queue.

ERROR RETURN

Control is returned to the error return address specified in the ATQ$ parameter list with the error code in the A-register when any of the following errors is detected:

A-Register Contents (Octal)	Error Condition
34	Requested executive function is not configured in the system.
35	Requested executive function is disk-resident and there was a disk error during an attempt to load the function into main memory.

ACTION ROUTINE DETAILS

The specified entry is linked to either the beginning or end of the queue as indicated by the queue attach mode parameter. Control is returned to the calling program at the normal return.

OUTLINE PARAMETER LIST

Word Number	Operation	Operand
0	DAC	queue header address
1	DEC	[queue attach mode]
2	DAC	queue entry block address
3	DAC	error return address

Parameters in the outline parameter list are the same as those described above for the inline parameter list. If the queue attach mode is omitted, a BSZ 1 statement must replace the respective statement.

Examples:

The example below illustrates the use of the ATQ$ function to attach an entry to a queue. The entry specified as ENT will be attached to the end of the queue RTRQ. An inline parameter is used in this example.

```
*                                       ATTACH AN ENTRY TO BOTTOM OF QUEUE
*
         ATQ$      RTRQ,ENT,ERR
*
*                                       THE FOLLOWING IS THE ENTRY TO BE ATTACHED
ENT      BSZ       8
ERR      ---                            ERROR RETURN
```

In the example below, the ATQ$ function is used with an outline parameter list to attach an entry to the beginning of the queue RTRQ. The address of the first word of the entry is placed in word ENTA (the third word of the parameter list) before the ATQ$ function is called.

```
*                                   ATTACH AN ENTRY TO TOP OF QUEUE
*
      LDX     LIST                  GET LIST ADDRESS
      ATQ$    (X)                   ATTACH ENTRY TO QUEUE
*
*                                   PARAMETER LIST FOR ATQ$
*
LIST  DAC     *+1                   ADDRESS OF LIST
      DAC     RTRQ                  ADDRESS OF QUEUE HEADER
      DEC     1                     ATTACH TO BEGINNING OF QUEUE
ENTA  BSZ     1                     FOR ENTRY ADDRESS
      DAC     ERR                   ERROR RETURN ADDRESS
```

Get Beginning Entry From Queue (GTQ$)

The GTQ$ function is used to access the entry currently at the beginning of the queue, and to make the next entry in the queue the beginning entry.

FUNCTION ACTION

The GTQ$ action routine unlinks the entry from the beginning of the queue, returns control to the calling program at the normal return with the address of the unlinked entry, if the queue is not empty, or returns control to the calling program at the error return if the queue is empty.

MACRO CALL FORMAT

Location	Operation	Operand
[symbol]	GTQ$	queue header address, error return address

symbol - Optional. The symbolic location of the GTQ$ macro instruction.

queue header address - The address of the first word of the queue header. This address is the same as used in the CRQ$ function call.

error return address - The address to which control is returned if an error is found during the processing of the GTQ$ function call.

NORMAL RETURN

Upon normal return, the entry currently at the beginning of the queue has been unlinked. The X-register contains the address of the first word of this entry.

ERROR RETURN

Control is returned to the error return address specified in the GTQ$ parameter list when any of the following errors is detected:

A-Register Contents (Octal)	Error Condition
34	Requested executive function is not configured.
35	Requested executive function is disk-resident and there was a disk error during an attempt to load the function into main memory.
Not Significant	No entry in the queue; the queue is empty. (There is no specific error code associated with this error condition.)

ACTION ROUTINE DETAILS

If the queue is empty, return is made to the error return. If the queue is not empty, the entry at the beginning of the queue is unlinked. Its location is placed in the X-register, and return is to the normal return.

OUTLINE PARAMETER LIST

Word Number	Operation	Operand
0	DAC	queue header address
1	DAC	error return address

Parameters in the outline parameter list are the same as those described above for the inline parameter list.

Examples:

In the following example, the GTQ$ function, is used with an inline parameter list to obtain the first entry from the RTRQ queue. If the queue is empty when the GTQ$ function call is executed, then control is transferred to location NENT so that the error can be processed.

```
*                                GET BEGINNING ENTRY FROM QUEUE
*
         GTQ$      RTRQ,NENT
*
*                                ERROR RETURN IF QUEUE IS EMPTY
*
NENT     ---
```

Using an outline parameter list, the preceding example would be written as follows:

```
*                                GET BEGINNING ENTRY FROM QUEUE
*
         LDX       PLST          GET LIST ADDRESS
         GTQ$      (X)           GET ENTRY
*
*                                PARAMETER LIST FOR GTQ$
*
PLST     DAC       *+1           ADDRESS OF LIST
         DAC       RTRQ          ADDRESS OF QUEUE HEADER
         DAC       NENT          ERROR RETURN ADDRESS
*
*                                ERROR RETURN IF QUEUE IS EMPTY.
*
NENT     ---
```

FREE MEMORY BLOCK MANAGEMENT

Block executive functions are made to obtain or release a block of free memory. The block functions are:

- Get Storage Block (GBL$)
- Return Storage Block (RBL$)

Free memory is divided into pools of different size blocks. The block sizes in the system are configurable and may range from a minimum of four words to the maximum configured length. The system provides only blocks that are 2^n words long; where n is an integer. However, the user can request blocks of any size; if the requested block size is not configured, then he receives the next larger configured block; if no larger block size is configured, an error status is returned to the user.

When a block is requested and the pool for that block size is empty, the error status is placed in the A-register and control returns to the user program through the specified error address. In returning a block to free memory through the RBL$ function, the block must have been obtained through a GBL$ function call.

The block function GBL$ and RBL$ may not be used by a restricted activity. Attempt to do so will result in the activity being aborted.

In a 64K system, free memory blocks are located in bank 0. Therefore, an activity wishing to access a free memory block must either have bank 0 as its A-bank, or use 64K indexing.

Get Storage Block (GBL$)

The GBL$ function is used to allocate a block of free memory to the task in which the macro routine is used.

FUNCTION ACTION

The GBL$ action routine obtains a block of the size requested from free memory, and returns to the normal return with the address of this block in the X-register, if the block is available. The GBL$ function returns to the error return if there are no blocks or if the size requested is illegal.

A restricted activity cannot call the GBL$ system function. If it attempts to do so, the activity will be aborted.

Location	Operation	Operand
[symbol]	GBL$	block-size, [unused], [unused], error return address

symbol - Optional. The symbolic location of the GBL$ macro instruction.

block-size - A positive integer specifying the number of words required. If it is not a power of two, the next larger integer which is a power of two is used.

unused - Optional. This parameter position is not used in the current release of OS/700. However, the parameter position is maintained for compatibility with Release 0100 of OS/700.

unused - Optional. This parameter position is not used in the current release of OS/700. However, the parameter position is maintained for compatibility with Release 0100 of OS/700.

error return address - The address to which control is returned if an error is found during the processing of the GBL$ function call.

NORMAL RETURN

Upon normal return, the address of the first word of the block obtained from free memory is in the X-register.

ERROR RETURN

Control is returned to the error return address specified in the GBL$ parameter list with the error code in the A-register when any of the following errors is detected:

A-Register Contents (Octal)	Error Conditio.
0	No blocks available.
1	Block size was zero, less than zero, or larger than the largest configured block.
34	Requested executive function is not configured.

ACTION ROUTINE DETAILS

The block size parameter is examined to determine if it is greater than zero. If not, return is made through the error return address. Otherwise, if block size is not a power of 2, the size is increased to the next higher power of 2.

An attempt is made to obtain the block from free memory. If a block is available, return is made to the normal return. If a block of the size requested is not available, return is made to the error return. If a block of the size requested is not configured, and a block of a larger size is configured and available, then the larger block is allocated to the user program and the normal return is taken.

OUTLINE PARAMETER LIST

Word Number	Operation	Operand
0	DEC	block size
1	BSZ1	[unused]
2	BSZ1	[unused]
3	DAC	error return address

Parameters in the outline parameter list are the same as those described above for the inline parameter list.

Examples:

In the following example, a block of 32 words is to be obtained. If no 32-word block is available, then control is transferred to location BER, with the "no-block" status in the A-register. If the 32-word size is larger than the largest configured block size, then control is transferred to location BER, with the "illegal size" status in the A-register. If there are no 32-word blocks configured in the system, but there are larger size blocks, an attempt is made to obtain the next largest size block. The user is unaware of the increased size of the block; he uses it as a 32-word block and returns it to free memory as a 32-word block. The Return Block action routine will return the block to the correct block pool.

```
*                                   GET A BLOCK OF FREE MEMORY
        GBL$     32,,,BER
        STX      ADDB               SAVE ADDRESS OF FREE MEMORY BLOCK
          .
          .
          .
*                                   ERROR RETURN, DETERMINE CAUSE OF ERROR
BER     SZE                         WAS SIZE TOO LARGE?
        JMP      BER1               YES, OR FUNCTION NOT CONFIGURED
        ---                         NO, NO BLOCK AVAILABLE
```

Using an outline parameter list, the above example would be written as follows:

```
*
                                    GET A BLOCK OF FREE MEMORY
        LDX      BLST               GET ADDRESS OF PARAMETER LIST
        GBL$     (X)                GET STORAGE BLOCK
        STX      ADDB               SAVE ADDRESS OF FREE MEMORY BLOCK

*                                   ERROR RETURN, DETERMINE CAUSE OF ERROR
BER     SZE                         WAS SIZE TOO LARGE?
        JMP      BER1               YES, OR FUNCTION NOT CONFIGURED
        ----                        NO, NO BLOCK AVAILABLE

*                                   PARAMETER LIST FOR GBL$ CALL
BLST    DAC      *+1                ADDRESS OF LIST
        DEC      32                 BLOCK SIZE = 32 WORDS
        BSZ 1    0                  UNUSED
        BSZ 1    0                  UNUSED
        DAC      BER                ERROR RETURN ADDRESS
```

Return Storage Block (RBL$)

The RBL$ function is used to return a block to free memory.

FUNCTION ACTION

The RBL$ action routine returns the specified block to free memory, and returns to the calling program at the normal return, or returns to the calling program through the error return if the block size is larger than that of any block size configured.

A restricted activity cannot call the RBL$ system function. If it attempts to do so, the activity will be aborted.

MACRO CALL FORMAT

Location	Operation	Operand
[symbol]	RBL$	block address pointer, block size, error return address

symbol - Optional. The symbolic location of the RBL$ macro instruction.

block address pointer - The address of a word containing the address of the first word of the block to be returned. This block address must have been obtained through a GBL$ function call.

block size - An integer specifying the size of the block (in words) to be returned. This integer is the same as the block size in the GBL$ parameter list used to obtain the block.

error return address - The address to which control is returned if an error is found during the processing of the RBL$ function call.

NORMAL RETURN

The normal return is made to the calling program when the block is returned to free memory.

ERROR RETURN

Control is returned to the error return address specified in the RBL$ parameter list with the error code in the A-register when any of the following errors is detected:

A-Register Contents (Octal)	Error Condition
0	Block size was zero or less, or block size was too large.
34	Requested executive function not configured.

ACTION ROUTINE DETAILS

The block size parameter is examined. If it is not greater than zero, return is made to the error return. If a block size is not a power of 2, the size is increased to the next higher power of 2. If the result is larger than any size configured, return is made to the calling program through the error return. If not, the block is returned to free memory, and return is made to the calling program at the normal return.

OUTLINE PARAMETER LIST

Word Number	Operation	Operand
0	DAC	block address pointer
1	DEC	block size
2	DAC	error return address

Parameters in the outline parameter list are the same as those described above for the inline parameter list.

Examples:

In the example below, the address of the 32-word block that is to be returned to free memory is stored in word ADDB before the RBL$ function call is executed. If the 32-word size is larger than the largest configured block size, then control is transferred to instruction SERR so that the error can be processed.

```
*                                 RETURN STORAGE BLOCK
*
        RBL$    ADDB,32,SERR
*
*                                 BLOCK SIZE IS IN ERROR
*
SERR    ---
*
*                                 THE FOLLOWING LOCATION WILL CONTAIN THE
*                                 ADDRESS OF THE BLOCK TO BE RETURNED
ADDB    BSZ     1
```

Using an outline parameter list, the above example would be written as follows:

```
*                               RETURN STORAGE BLOCK
*
        LDX     PLST            GET ADDRESS OF LIST
        RBL$    (X)             RETURN BLOCK
*
*                               BLOCK SIZE ERROR
SERR    ---
*
*                               PARAMETER LIST
*
PLST    DAC     *+1             ADDRESS OF LIST
        DAC     ADDB            BLOCK ADDRESS POINTER
        DEC     32              SIZE = 32 WORDS
        DAC     SERR            ERROR RETURN ADDRESS
*
ADDB    BSZ     1               ADDRESS OF FREE MEMORY BLOCK
```

SECTION VI
CONTROLLING PHYSICAL I/O OPERATIONS

This section describes the executive functions used for physical I/O operations for all peripheral devices except communications equipment (communication functions are described in the OS/700 Communications manual.) Certain functions peculiar to physical I/O operations on the disk are described in the OS/700 File Management manual.

PHYSICAL I/O EXECUTIVE FUNCTIONS

Physical I/O macros are used to initiate physical I/O operations on all peripheral devices that can be used under System 700. They include:

- Assign Device Control Block (DCB$)
- Reserve a Device (RSV$)
- Release a Device (REL$)
- Input (INP$)
- Output (OTP$)
- Write End of file (EOF$)
- Space File (SPF$)
- Space Record (SPR$)
- Rewind (RWD$)
- Unload (ULD$)
- Wait for I/O (WIO$)

For physical I/O operations on disks supervised by the Volume Manager, the following functions are used, in addition to the INP$, OTP$, and W10$ mentioned above:

• Assign Volume Control Block	(VCB$)
• Connect Volume	(CVL$)
• Disconnect Volume	(DVL$)
• Allocate Work Area	(ALC$)
• Deallocate Work Area	(DLC$)

These functions are described in the OS/700 File Management manual. Disks not supervised by the Volume Manager (e.g., disk units configured in a COS) are utilized as normal peripheral devices as described in this section. Refer to the OS/700 System Generation manual for details of disk unit configuration.

Physical I/O operations may be requested on input and output devices which have been configured into the system. Before any physical I/O request is issued, a device control block must be generated, and the desired device reserved using a RSV$ call. (When performing physical I/O operations on a disk supervised by the Volume Manager, a volume control block must be generated, and a CVL$ call issued. (See the OS/700 File Management manual.) Peripheral devices, with the exception of disk units supervised by the Volume Manager fall into two classes. Disk units are sharable; they may be reserved even if currently reserved. They are also public; the user ID is ignored. All other peripherals are nonsharable and private; they may not be reserved if currently reserved, and require a user ID when being reserved so that subsequent physical I/O requests can be verified. A disk unit supervised by the Volume Manager is sharable, but may be public or private depending on the type of volume mounted on the unit. The physical I/O requests include:

- Input (INP$)
- Output (OTP$)
- Space File (SPF$)
- Space Record (SPR$)
- End of file (EOF$)
- Rewind (RWD$)

A Wait for I/O (WIO$) function call must be issued following each of these physical I/O function calls. When the physical I/O operation is completed, execution of the user's activity resumes at the I/O completion return address. At this point, the user must determine whether the operation was successful by examining the status block supplied with the I/O request. See Appendix B for the specific information returned by each device driver. When all of the physical I/O requests for the device which was reserved have been completed, the device must be released by issuing a Release (REL$) function call. If the device is magnetic tape or cassette, an Unload (ULD$) call may be used as an alternative to the Release call; the Unload call rewinds the device before releasing it.

Input and output operations under OS/700 are handled by the system in the following manner. The user (or the system) makes a request for a specific I/O function by calling one of the six I/O executive functions listed above. The system checks the request for legality, initiates the operation on the specified device, and returns immediately to the caller, who continues to run at the same priority level while the I/O is in progress. Ultimately the device signals the completion of the operation by means of an interrupt, which causes the system to schedule the caller's I/O completion code as a new task, at the same priority level at which the caller was running when he made the I/O request. This new task, however, cannot be dispatched while the user's previous task is still running (or suspended), since no two tasks can run at the same priority level concurrently. The user must therefore call the WIO$ executive function when he is ready to have the I/O completion code executed; this function terminates the current task and allows the I/O completion code to be dispatched.

In systems with more than 32K of memory or with the System Integrity option, the I/O completion code is scheduled using an I/O TCB from the free memory pool. The new task starts execution in the WIO$ action routine, which returns the I/O TCB to free memory, to free the caller of this responsibility, before exiting to the caller's I/O completion code. In other systems, where the new task starts at the caller's I/O completion code itself and not in the executive, the caller's I/O status block is used as a TCB to avoid having a free memory block to return.

The user need not normally concern himself with the fact that the WIO$ function terminates his current task and dispatches the remainder of his code as a new task. The program continues to run on the same priority level as before, and the WIO$ function also has an option that allows the program's registers to be saved through the call. The user does not even have to return the TCB used to schedule the I/O completion task, for the reasons described above. To all intents and purposes it is as if the same user task had continued to run throughout. However, the user must be aware that if he calls any other executive function which implicitly terminates his current task between the I/O request and the WIO$ call, this may allow the I/O completion code to be dispatched before the user is ready for it. Any executive function call which causes I/O to be performed on behalf of the user, e.g., to the operator's console (TYP$ or TPR$ request) or to the disk (file operations) will implicitly terminate the current task, and may cause the user's I/O completion code to be dispatched prematurely. This includes any executive function which may be disk-resident, such as the queue handling routine, since disk input must be done in order to read the executive function action routine into the system overlay area.

The safe course, therefore, is for the user to ensure that no executive functions are called between an I/O request and the WIO$ call following it, except for the following functions which are always memory-resident:

- A concurrent I/O request (INP$, OTP$, SPF$, SPR$, EOF$, RWD$)
- GBL$ and RBL$ (get and return free memory block) - if configured
- CTC$ and STC$ (create and schedule TCB), STS$ (schedule task), and SUS$ (suspend task)

As implied above, the user can do parallel I/O on two or more devices by making two or more consecutive I/O requests. There must be one WIO$ call for each I/O request. I/O completion routines will be dispatched in the order in which the operations are completed, not the order in which the requests were made. Each I/O completion routine must therefore contain code to test whether or not the other I/O completion routines have yet been executed, to determine whether or not a further WIO$ call is necessary.

In several cases the caller's I/O completion code is not scheduled to start execution directly at the I/O completion return. Instead the system schedules a task within the WIO$ action routine, which will then return to the caller's I/O completion address through the function manager. This is an additional reason why the user must always call the WIO$ function to allow the I/O completion routine to be executed, and never use TMT$, for example. This method enables the WIO$ action routine to preserve the caller's registers (if this option is specified), and also to ensure that in a 64K system the I/O completion routine is executed with the correct banks set up. Note that in a 64K system if the register preservation option is specified on the WIO$ function call, the I/O completion code will be executed with the banks which were set up when the WIO$ call was made, whereas if the register preservation option is not specified, the I/O completion code will be executed with the banks which were set up when the activity was dispatched (the banks specified in the ACB). This need not concern the user unless he has performed explicit bank switching within the activity.

Because the executive uses the I/O status block for the storage of data during the processing of the I/O request, and can use it as a TCB to schedule the user's I/O completion code, the following rules must be observed:

- The user must not alter the contents of the status block between the time he makes the I/O request and the time he calls the WIO$ function. To do so could interfere with the dispatching of the I/O completion code, since this task could already have been scheduled by the system before the WIO$ call was made.
- If two or more I/O operations are in progress concurrently; i.e., if the user has made another I/O request before executing a WIO$ call for a previous request, he must supply a separate status block for each request.

The RSV$ request with a queued reserve return address specified works in a manner analogous to that of an I/O request. If the error return to the user is taken and the error code indicates that the device is currently not available, this means that when the device is available the system will schedule the user's special return as a new task. In this case, the user must make a WIO$ call when he wishes to wait for the device, and the considerations described above also apply.

The Unload (ULD$) function works differently from the other six I/O functions, and does not need to be followed by a Wait for I/O call. When the rewind operation is completed, the system does not schedule a user routine, but instead schedules a system task which takes care of releasing the device and terminates automatically. The user therefore need not concern himself any further after making the ULD$ call, and can treat ULD$ like any other non-I/O system function.

Restricted activities may issue requests for all of the physical I/O functions listed previously; however, there are certain limitations enforced by the operating system.

1. All parameter specified directly or indirectly by the I/O parameter lists must like within the activity area, or the activity will be aborted.
2. Each device reserved and volume connected by the activity is recorded. For each I/O request issued by the activity, a check is made to determine if the requested device or volume is recorded as being allocated to the activity. If it is not recorded, the I/O request is rejected. This record of devices and volumes allocated is also used to release them if the activity is aborted or terminates prematurely.
3. The number of outstanding I/O requests is also recorded. More than one I/O request may be issued by the activity prior to a WIO$ request; however, while there is one or more I/O requests outstanding, only I/O and WIO$ requests may be issued by the activity. The activity will be aborted if any other request, including RSV$ and REL$ is issued. A ULD$ request will be rejected if the activity has an I/O request outstanding on the same device for which the ULD$ request was issued.
4. If a WIO$ request is issued and there are no I/O requests outstanding or a queued reserve request waiting to be processed, the activity will be aborted.
5. If one or more I/O requests are being processed when a restricted activity is aborted, the devices being used by the activity will not be released and the activity will not be terminated by the abort task until all the requests have been completed. However, there is no wait for the queued I/O requests that have not been initiated; they are removed from the queue.

DCB$

Assign Device Control Block (DCB$)

The DCB$ call generates a list of parameters required for Reserve (RSV$), Release (REL$), Input (INP$), Output (OTP$), Space File (SPF$), Space Record (SPR$), End of file (EOF$), Rewind (RWD$), and Unload (ULD$) functions.

MACRO EXPANSION

The DCB$ macro call is expanded when the program is assembled to generate a list of parameters which contain no executable code.

MACRO CALL FORMAT

Location	Operation	Operand
DCB name	DCB$	[generic device type], [logical unit number], data mode, [user-ID], I/O status block address, I/O completion return address

DCB name - The symbolic location of the device control block created by the DCB$ macro instruction.

generic device type – Optional. A number indicating the generic type of device which is to perform the I/O operation. Refer to the Appendix D for the assignments of the generic device types. The default value is minus one (=1), and implies the system disk. If the generic device type is -1, the generic device type and the logical unit number for the system disk will be stored in the user's DCB$ parameter list.

logical unit number – Optional. A number that specifies the logical unit of a generic device type which is to perform the I/O operation, (logical unit numbers are assigned to units when the system is configured.). For example, assume that the system is configured for two magnetic tape controllers and that each controls four units. The physical unit numbers for the units on each of the controllers range from 0 to 3. The logical unit numbers for the units on both of the controllers range from 0 to 7.

The default value is minus one (-1), and may be used only when the generic device type is the default value (-1).

data mode - A number which specifies the kind of data transfer that is to be performed. Refer to Appendix C for the assignment of the types of data modes.

user-ID – Optional. One word (16 bits) of identification. This must be nonzero when reserving a private device. It is ignored when reserving a public device.

I/O status block address - The address of an 8-word block. For I/O requests, the status block is used by the device driver to schedule the I/O completion return to the user and to store the status information as described in Appendix B. The I/O status block address is in the X-register when control is returned to the user at the I/O completion return address or at the queued RSV$ request return address.

I/O completion return address - The address where control is to be returned after I/O completion.

MACRO CALL ACTION DETAILS

The code produced by the DCB$ macro call has no action since it is a list of parameters. The DCB$ macro call must therefore be coded in a program as a data block or buffer.

OUTLINE PARAMETER LIST

Word Number	Operation	Operand
0	BSZ	1 (Reserved for system use)
1	DEC	[generic device type]
2	DEC	[logical unit number]
3	DEC	data mode
4	OCT	[user-ID]
5	DAC	I/O status block address
6	DAC	I/O completion return address

The parameters in the outline parameter list are the same as those described above for the inline parameter list. However, note that the first word of the list must be a BSZ 1 statement. If the generic device type and the logical unit number are not specified, the respective DEC statements must be replaced with DEC -1 statements. If the user-ID is not specified, a BSZ 1 statement must replace the appropriate OCT statement.

Examples:

The example shown below illustrates the generation of a device control block by issuing a DCB$ request.

```
*                                  GENERATE A DCB TO BE REFERENCED WHEN MAKING
*                                  I/O REQUESTS
*                                  FOR MAGNETIC TAPE, LOGICAL UNIT NO. 1
DCNM    DCB$    10,1,0,'146724,TBAD,RTAD
*
TBAD    BSZ     8                  I/O STATUS BLOCK, TBAD+1 WILL CONTAIN 0 IF
*                                  NO ERROR OCCURRED ON I/O COMPLETION
*

        LNK$                       LINK TO SYSTEM
```

The example below shows how the device control b'.ɔck would be written without the use of a DCB$ request.

```
*                              GENERATE A DCB TO BE REFERENCED WHEN MAKING
*                              I/O REQUESTS FOR MAGNETIC TAPE, LOGICAL
*                              UNIT NO. 1
DCNM     BSZ     1             RESERVED FOR SYSTEM USE.
         DEC     10            GENERIC DEVICE TYPE FOR MAGNETIC TAPE
         DEC     1             LOGICAL UNIT NUMBER 1
         DEC     0             DATA MODE (ASCII).
         BCI     1,MT          USER ID.
         DAC     TBAD          I/O STATUS BLOCK ADDRESS
         DAC     RTAD          I/O COMPLETION RETURN ADDRESS
*
TBAD     BSZ     8             I/O STATUS BLOCK, TBAD+1 WILL CONTAIN 0 IF
*                              NO ERROR OCCURRED ON I/O COMPLETION
         LNK$                  LINK TO SYSTEM.
```

In the above example, the device control block is formatted by the user and no DCB$ macro call is made.

Reserve Device (RSV$)

The RSV$ system function associates a peripheral device with a user-ID. The device may be nonsharable, in which case only one RSV$ request for the device will be honored until the device is released, or sharable, in which case simultaneous RSV$ requests will be honored.

If a disk unit supervised by the Volume Manager is to be used, a CVL$ request should be made instead of RSV$.

The I/O devices reserved and the volumes connected by a restricted activity are recorded by the operating system. This allows the operating system to release the devices reserved and the volumes connect if the restricted activity is aborted or terminates prematurely. In addition it allows the operating system to ensure that the I/O requests made by a restricted activity are permissible.

ROUTINE ACTION

The RSV$ system function checks the parameters in the RSV$ parameter list and the device control block, and takes the error return if any of them is illegal. If the requested device is nonsharable and already reserved, the error return will be taken. If, however, the user has specified a queued reserve return address in the parameter list, the request will be queued before taking the error return. The user is then notified when the device becomes available. If the device is not reserved, or is sharable, the device will be reserved and the normal return taken.

MACRO CALL FORMAT

Location	Operation	Operand
[symbol]	RSV$	DCB address, error return address, [queued RSV$ request return address , [control-P TCB address pointer]

symbol - Optional - The symbolic location of the RSV$ macro instruction.

DCB address - The name of the device control block associated with the device to be reserved.

error return address - The address to which control is returned if an error is found in the RSV$ parameter list, the specified device control block parameters, or if the specified device is currently reserved.

queued RSV$ request return address - Optional. The address to which control is transferred after a queued reserve request is fulfilled. If this address is present in the RSV$ parameter list, and if the requested nonsharable device was already reserved when the reserve request was issued, the reserve request is queued according to the priority level of the task that issued the RSV$ request. Control is then transferred to the error return address with zero in the A-register, which indicates that the requested device is already reserved. The user who issued the reserve request must at this time issue a WIO$ macro instruction to wait for the requested device to be reserved. When the user's reserve request is removed from the reserve request queue and processed, control will be transferred to the user at the queued RSV$ request return address with the I/O status block address in the X-register.

If the queued RSV$ request return address is not specified (equal to zero) in the RSV$ parameter list, and the requested device was already reserved when the reserve request was issued, control is transferred to the error return address with zero in the A-register. The reserve request is not queued in this case.

control-P TCB address pointer - Optional. A pointer to a word that contains the address of a TCB. This TCB must have been created by the user before the RSV$ request was issued. The TCB may be created by the use of the Create TCB macro (CTC$). The TCB must contain the address of a special action routine which will be scheduled by the KSR or ASR driver when a control-P character is typed on the KSR or ASR keyboard. The special action routine must be within the same activity which contains the RSV$ request. This parameter is optional when the requested device is either the KSR (generic device type 0) or the ASR (generic device type 12), and is ignored for other devices. The control-P TCB may not be used by a restricted activity. (If the control-P TCB address is specified, it is ignored).

NORMAL RETURN

Control is returned to the calling program at the instruction following the RSV$ macro call, after the requested device has been reserved for the caller.

QUEUED RSV$ REQUEST RETURN

After a WIO$ request has been completed, control is returned to the queued RSV$ request return address when the reserve request which was queued is fulfilled.

ERROR RETURN

Control is returned to the error return address specified inthe RSV$ parameter list with the error code in the A-register when any of the following errors is detected.

A-register Contents (Octal)	Error Condition
0	The requested device, which is nonsharable, was already reserved when another RSV$ request was issued.
1	The generic device type specified in the device control block is not configured for this system.
3	The logical unit number specified in the device control block is not configured for this system.
14	An RSV$ request was issued for a disabled device.
30	The device driver for the requested device is disk resident, and there was an activity area overrun error while loading the device driver.
31	The device driver for the requested device is disk resident, and there was a disk read error while loading the device driver.
33	The device driver for the requested device is disk resident, the queued RSV$ request return address is null (equal to zero) in the RSV$ parameter list or the request was made by a restricted activity, and the activity area for the device driver is in use by another device driver or activity.
34	Requested executive function is not configured.
35	Requested executive function is disk resident and there was a disk error during an attempt to bring the function into main memory.
155	Illegal reserve request.

ACTION ROUTINE DETAILS

The following checks are made to determine if the RSV$ request is legal:

1. A test is made to determine if the generic device type and the logical unit number specified in the device control block have been configured into the system.
2. A test is made to determine if the requested device is enabled.
3. A test is made to determine if the I/O status block address is specified (≠0) in the device control block.
4. A test is made to determine if the user-ID specified in the device control block is nonzero when a nonsharable device is requested.

If the RSV$ request is not legal, control is returned to the error return address specified in the RSV$ parameter list with the error code in the A-register. If the RSV$ request is legal, a test is made to determine if the device driver for the requested device is disk resident, and if it is, the device driver is brought into main memory.

Now, if the device is sharable, or is nonsharable but not currently reserved, it is reserved for the caller, and control is returned to the user at the normal return. If a nonsharable device is already reserved, a test is made to determine if the queued RSV$ request return address specified in the RSV$ parameter list is omitted (equal to zero). If omitted, control is returned to the error return address specified in the RSV$ parameter list with zero in the A-register. If the queued RSV$ request return address is specified, the Reserve request is queued according to the priority level of the task which issued the RSV$ request, and control is returned to the error return address specified in the RSV$ parameter list with zero in the A-register.

When control is returned to the user at the error return address with a zero in the A-register, and the queued RSV$ request return address was specified in the RSV$ parameter list, the user must issue a WIO$ request to wait for the device to be reserved. When reserve request is processed, control is returned to the user at the queued RSV$ request return address.

If a nonrestricted activity specifies a queued reserve return address, regardless of whether the device is sharable or nonsharable, a reserve request for a device with a disk-resident driver will be honored even if the driver's activity area is occupied by another activity. When the activity area becomes available, the driver will be loaded into main memory, the device reserved, and the normal return taken to the caller. If the queued reserve return address is not specified, or if the activity is restricted, and the driver may not be loaded into main memory immediately, the error return is taken.

OUTLINE PARAMETER LIST

Word Number	Operation	Operand
0	DAC	DCB address
1	DAC	error return address
2	DAC	[queued RSV$ request return address]
3	DAC	[control-P TCB address pointer]

The parameters in the outline parameter list are the same as those described above for the inline parameter list. Note that if it is not desirable to have the reserve request queued when the requested device is already reserved, word 2 must contain a BSZ 1 statement. If the RSV$ call is for an ASR or KSR device and the control-P TCB address pointer is not specified, a BSZ 1 statement must replace the respective DAC statement. If the RSV$ call is not for an ASR or KSR device, this parameter should be omitted.

Examples:

The examples below illustrate the use of the RSV$ function to reserve a device so that I/O requests for magnetic tape logical unit 1 can be processed. The device control block is designated by the name DCNM (see the examples under "Assign Device Control Block (DCB$)" above). The queued reserve request return, QUED, specifies that the reserve request is to be added to the reserve request queue if the magnetic tape, logical unit 1, has already been reserved by another user.

```
*                              RESERVE DEVICE
*
         RSV$   DCNM,ERAD,QUED RESERVE THE MAGNETIC TAPE, LOGICAL UNIT
*                                  NO. 1
         ---                   NORMAL RETURN, RESERVE PROCESSED, I/O MAY
                               BE ISSUED
*
*                              ERROR RETURN - A-REGISTER CONTAINS ERROR
*                              CODE.
*
ERAD     SZE                   WAS CALL QUEUED?
         JMP    ERR            NO, PARAMETER IN DCB ILLEGAL.
         JST    SETP           YES, SET UP FOR I/O CALL.
         WIO$                  WAIT FOR RESERVE TO BE PROCESSED.
*
*                              QUEUED RESERVE REQUEST RETURN AFTER RESERVE
*                              IS PROCESSED.
QUED     ---                   RESERVE PROCESSED, I/O MAY BE ISSUED.
```

Using an outline parameter list, the above example would be written as follows:

```
*                              RESERVE DEVICE
*
         LDX    PLT1           PARAMETER LIST POINTER TO X.
         RSV$   (X)            RESERVE THE MAGNETIC TAPE, LOGICAL UNIT NO. 1
*        ---                   NORMAL RETURN, RESERVE PROCESSED, I/O MAY BE
*                              ISSUED.
*
*                              ERROR RETURN - A-REGISTER CONTAINS ERROR CODE.
*
ERAD     SZE                   WAS CALL QUEUED?
         JMP    ERR            NO, PARAMETER IN DCB ILLEGAL.
         JST    SETP           YES, SET UP FOR I/O CALL.
         WIO$                  WAIT FOR RESERVE TO BE PROCESSED.
*
*                              QUEUED RESERVE REQUEST RETURN AFTER RESERVE
*                              IS PROCESSED.
QUED     ---                   RESERVE PROCESSED, I/O MAY BE ISSUED
*
*                              PARAMETER LIST
*
PLT1     DAC    *+1            POINTER TO PARAMETER LIST
         DAC    DCNM           DCB ADDRESS
         DAC    ERAD           ERROR RETURN ADDRESS
         DAC    QUED           QUEUED RESERVE REQUEST RETURN ADDRESS.
```

Release Device (REL$)

The REL$ function is used to release a device previously reserved by a RSV$ function call.

If a restricted activity is aborted or terminates, all devices reserved by the activity but not released are released for the activity by the operating system.

FUNCTION ACTION

The REL$ action routine checks to determine the legality of the parameters specified in the device control block and takes the error return if any parameter is illegal. Otherwise, it releases the specified device. The REL$ action routine also checks the reserve request queue for users waiting to reserve the released device, and reserves the device for the user of the highest priority. Then, it returns to the calling program at the normal return.

MACRO CALL FORMAT

Location	Operation	Operand
[symbol]	REL$	DCB address, error return address

symbol - Optional. The symbolic location of the REL$ macro instruction.

DCB address - The name of the device control block associated with the device to be released.

error return address - The address to which control is returned if an error is found in the specified device control block parameters or if the device was not previously reserved.

NORMAL RETURN

After the specified device has been released, control is returned to the calling program, at the instruction following the REL$ request.

ERROR RETURN

Control is returned to the error return address specified in the REL$ parameter list with the error code in the A-register when any of the following errors is detected:

A-register Contents (Octal)	Error Condition
1	The generic device type parameter specified in the device control block is not configured for this system.

A-register Contents (Octal)	Error Condition
3	The logical unit number specified in the device control block is not configured for this system.
10	The device was not previously reserved. The user-ID specified in the device control block is not correct, or the disk resident driver for the device is not currently resident in main memory.
15	The device was not reserved by the restricted activity that issued the REL$ request.
34	Requested executive function is not configured.
35	Requested executive function is disk resident and there was a disk error during an attempt to load the function into main memory.
154	An I/O request is being processed on the device requested to be released, and fulfillment of the REL$ request would mean that no one had the device reserved.

ACTION ROUTINE DETAILS

The following checks are made to determine if the release request is legal:

1. A test is made to determine if the generic device type and the logical unit number specified in the device control block have been configured into the system.
2. A test is made to determine if the driver for the specified device is currently resident in main memory.
3. A test is made to determine if the user-ID specified in the device control block matches the ID for which the device was reserved.
4. A test is made to determine if there is an I/O request being processed on the specified device. If there is no I/O request being processed, the REL$ request is legal. If there is an I/O request being processed, and the device is nonsharable, the REL$ request is illegal. If there is an I/O request being processed, and the device is sharable, another test is made to determine if the device will still be reserved for another user if the REL$ request is fulfilled.
5. If the activity is restricted, a test is made to determine if the specified device was reserved by the activity.

If the REL$ request is not legal, control is returned to the error return address specified in the REL$ parameter list with the error code in the A-register. If the REL$ request is legal, the specified device is released. A check is then made to determine if there are entries in the reserve request queue. If there are no users waiting to reserve the device, control is returned to the user at the normal return. If there are one or more entries in the reserve request queue, the device is reserved for the user of highest priority, the queued RSV$ request return address is scheduled and control is returned to the user who issued the REL$ request at the normal return.

OUTLINE PARAMETER LIST

Word Number	Operation	Operand
0	DAC	DCB address
1	DAC	error return address

The parameters in the outline parameter list are the same as those described above for the inline parameter list.

Examples:

The examples shown below illustrate the use of the REL$ function to release the magnetic tape, logical unit 1, specified in the device control block, DCNM (see the examples under Assign Device Control Block (DCB$) above).

```
*                               RELEASE DEVICE
*
*      REL$    DCNM,ERAD        RELEASE LOGICAL UNIT NO. 1 OF THE MAGNETIC
*                               TAPE
       ---                      NORMAL RETURN, DEVICE RELEASED
*
*                               ERROR RETURN - A-REGISTER CONTAINS THE ERROR
*                               CODE
*
ERAD   ---                      ERROR, ILLEGAL PARAMETER IN THE DCB, DCNM
```

Using an outline parameter list, the above example would be written as follows:

```
*                               RELEASE DEVICE
*
       LDX     PLS3             PARAMETER LIST POINTER TO X
       REL$    (X)              RELEASE LOGICAL UNIT NO. 1 OF THE MAGNETIC
*                               TAPE
       ---                      NORMAL RETURN, DEVICE RELEASED.
*
*                               ERROR RETURN - A-REGISTER CONTAINS THE ERROR
*                               CODE
*
ERAD   ---                      ERROR, ILLEGAL PARAMETER IN THE DCB, DCNM
*
*                               PARAMETER LIST
*
PLS3   DAC     *+1              POINTER TO PARAMETER LIST
       DAC     DCNM             DCB ADDRESS
       DAC     ERAD             ERROR RETURN ADDRESS
```

Input (INP$)

The INP$ function is used to initiate the transfer of a record from an I/O device into main memory.

FUNCTION ACTION

The INP$ action routine checks to determine if the specified device has been previously reserved by a RSV$ function call; if not, the error return is taken. Then, the parameters specified in the INP$ parameter list and the device control block are checked. The error return is taken if any one of the parameters is illegal. The INP$ action routine then queues the input request according to the priority level of the task which issued the INP$ request, initiates the input request if the specified device is not currently busy processing another request, and returns to the calling program at the normal return. Normally, the WIO$ function should be called immediately. However, further user code may be executed, and additional physical I/O requests made before calling WIO$.

MACRO CALL FORMAT

Location	Operation	Operand
[symbol]	INP$	DCB address, error return address, buffer address, range, [mass memory segment number address] , [I/O status block address], [I/O completion return address]

symbol - Optional. The symbolic location of the INP$ macro instruction.

DCB address - The name of the device control block associated with the input device.

error return address - The address to which control is returned if an error is found in the INP$ parameter list, the specified device control block parameters, or if the input device was not previously reserved.

NOTE: Control does not return to the error return address if an error occurs during the retrieval from the input device. A user tests for retrieval error by checking the I/O status block after control is returned to the I/O completion return address.

buffer address - The address of the first word of the buffer into which the record is to be placed. In a 64K system, this buffer may reside in the callers' A-bank or B-bank, or partially in both as long as it does not cross from bank 1 to bank 2. In addition, the buffer address may be a logical or a physical address. If a physical address is specified, the sign bit of the range word must be set to indicate that this is so. Only nonrestricted activities can specify a physical address. If a restricted activity attempts to do so, it will be aborted.

range - The number of words to be read, from a minimum of 1 to a maximum of 4095. The number of words transferred into the buffer will be determined by the expiration of the range count or the end of record, whichever occurs first.

In a 64K system, the sign bit of the range word must be set if the buffer address is specified as a physical rather than a logical address.

mass memory segment number address - Optional. The address of a word containing the number of the disk segment from which record is to be read. This parameter is omitted for all but disk devices.

I/O status block address - Optional. The address of an 8-word block used by the device driver to schedule the I/O completion return to the user and to store the status information as described in Appendix B. The I/O status block address is in the X-register when control is returned to the user program at the I/O completion return address. If this parameter is omitted, the I/O status block address specified in the device control block is used. If this parameter is present, it is used to replace the I/O status block address in the device control block and any subsequent I/O call using the same device control block uses the new I/O status block address.

NOTE: Updating of the I/O status block address in the DCB is performed by the execution of the inline macro expansion and not by the action routine.

I/O completion return address - Optional. The address where control is to be returned after I/O completion. If this parameter is omitted, the I/O completion return address specified in the device control block is used. If this parameter is present, it is used to replace the I/O completion return address in the device control block and any subsequent I/O call using the same device control block uses the new I/O completion return address.

NOTE: Updating of the I/O completion return address in the DCB is performed by the execution of the inline macro expansion and not by the action routine.

NORMAL RETURN

Control is returned to the calling program at the instruction, following the INP$ request, after the input operation has been initiated (and probably before it has been completed). The calling program must wait for input completion by issuing a WIO$ function call.

I/O COMPLETION RETURN

After a WIO$ request has been completed, control is returned to the I/O completion return address whether the record has been retrieved successfully or unsuccessfully. The user must then test for input errors by checking the I/O status block (see I/O status block error code in Appendix B).

ERROR RETURN

Control is returned to the error return address specified in the INP$ parameter list with the error code in the A-register when any of the following errors is detected:

A-register Contents (Octal)	Error Condition
1	The generic device type specified in the device control block is not configured for this system.
3	The logical unit number specified in the device control block is not configured for this system.
4	The data mode specified in the device control block is not in the range 0 to 4, or is not a valid mode for the specified input device.
5	The number of words specified in the range parameter is smaller than 1 or more than 4095, or the I/O buffer crosses the boundary between banks 1 and 2 (64K systems only).
10	The device was not previously reserved under the user-ID specified in the device control block.
11	An input request was issued for an output only device.
14	An input request was issued for a disabled device.
15	The device was not reserved by the restricted activity that issued the INP$ request.
32	The disk-resident driver for the specified device is not currently resident in main memory.
34	Requested executive function is not configured.
156	The input request for the disk cannot be accepted because a mount is in progress on the requested unit.

ACTION ROUTINE DETAILS

The following checks are made to determine if the input request is legal:

1. A test is made to determine if the generic device type and the logical unit number specified in the device control block have been configured into the system.
2. A test is made to determine if the requested device is enabled.

3. A test is made to determine if the requested device has been previously reserved by a RSV$ function call.

4. A test is made to determine if the user ID specified in the device control block matches the ID for which the device was reserved (private devices only).

5. A test is made to determine if the data mode specified in the device control block is within the range from 0 to 4, and is a valid mode for the specified device.

6. A test is made to determine if the range value specified in the INP$ function call is within the limits from 1 to 4095.

7. A test is made to determine if the specified device is an input device.

8. If the specified device is a disk, a test is made to determine if a removable disk pack is not in the process of being mounted on the requested unit.

9. If the activity is restricted, a test is made to determine if the specified device was reserved by the activity.

If the input request is not legal, control is returned to the error return address specified in the INP$ parameter list with the error code in the A-register. If the input request is legal, the input request is queued according to the priority level of the task which issued the INP$ request. If the requested device is currently busy processing another request, control is returned to the calling program at the normal return address. If the requested device is not currently busy, the input request is initiated, and control is returned to the caller at the normal return. When the calling program receives control at the normal return, a WIO$ request must be issued immediately if the calling program requires sequential input. If nonsequential input is desired, the calling program continues processing to the point that the input completion is required. At this point, a WIO$ request must be issued. Upon completion of the requested input, the code specified by the I/O completion return address is scheduled. When the calling program receives control at the I/O completion return address, the calling program must assume the responsibility of interrogating the I/O status block to determine if the input request was successful. Word 1 of the status block is zero if no error occurred. (See Appendix B for the status information.)

OUTLINE PARAMETER LIST

Word Number	Operation	Operand
0	DAC	DCB address
1	DAC	error return address
2	DAC	buffer address
3	DEC	range
4	DAC	[mass memory segment number address]

The parameters in the outline parameter list are the same as those described above for the inline parameter list. If the INP$ call is for a mass memory device, the segment number address must be specified. If the call is not for a mass memory device, this parameter should be omitted.

NOTE: If an outline parameter list is used and it is desired to update the I/O status block address or the I/O completion return address in the DCB, the user must precede the INP$ function call with the instructions to store the new I/O status block address in the sixth word (word 5) of the DCB or the new I/O completion return address in the seventh word (word 6) of the DCB.

Examples:

The examples below illustrate the use of the INP$ function to input sequentially from magnetic tape. The call is sequential because the WIO$ function call is issued immediately following the input request. The device control block, DCNM, is shown in the example given under "Assign Device Control Block (DCB$)," and the RSV$ request is shown in the example given under the description of the RSV$ function. The buffer address is BFAD. The I/O completion return address, RTAD is overlaid and becomes NMAD, but the I/O status block address, TBAD, remains the same as in the DCB, since a new I/O status block address is not specified in the input request.

```
*                                INPUT A 60-WORD RECORD FROM MAGNETIC TAPE,
*                                LOGICAL UNIT NO. 1
        INP$    DCNM,ERAD,BFAD,60,,,NMAD  INPUT A RECORD
        WIO$    PTBAD            SEQUENTIAL, WAIT FOR COMPLETION OF INPUT
*                                I/O COMPLETION RETURN
NMAD    LDA     TBAD+1           FETCH THE STATUS
        SZE                      DID AN ERROR OCCUR?
        JMP     ERR              YES, PROCESS ERROR
        ---                      NO ERROR, PROCESS DATA
*
*                                ERROR RETURN - A-REGISTER CONTAINS THE ERROR
*                                CODE
*
ERAD    ---                      ERROR, ILLEGAL PARAMETER IN THE INP$ PARAM-
*                                ETER LIST OR IN THE DCB, DCNM.
PTBAD   DAC     TBAD             I/O STATUS BLOCK ADDRESS
BFAD    BSZ     60               INPUT BUFFER
```

Using an outline parameter list, the above example would be written as follows:

```
*                          INPUT A 60-WORD RECORD FROM MAGNETIC TAPE,
*                          LOGICAL UNIT NO. 1
        LDA     PNAD       A-REGISTER SET TO DAC NMAD
        STA     DCNM+6     STORE I/O COMPLETION RETURN ADDRESS IN
*                          THE DCB
        LDX     PLS7       POINTER TO PARAMETER LIST TO X
        INP$    (X)        INPUT
        WIO$    PTBAD      SEQUENTIAL, WAIT FOR COMPLETION OF INPUT
*                          I/O COMPLETION RETURN
NMAD    LDA     TBAD+1     FETCH THE STATUS
        SZE                DID AN ERROR OCCUR?
        JMP     ERR        YES, PROCESS ERROR
        ---                NO ERROR, PROCESS DATA
*
*                          ERROR RETURN - A-REGISTER CONTAINS THE ERROR
*                          CODE
*
ERAD    ---                ERROR, ILLEGAL PARAMETER IN THE INP$
*                          PARAMETER LIST OR IN THE DCB, DCNM
*                          PARAMETER LIST
*
PLS7    DAC     *+1        POINTER TO PARAMETER LIST
        DAC     DCNM       DCB ADDRESS
        DAC     ERAD       ERROR RETURN ADDRESS
        DAC     BFAD       BUFFER ADDRESS
        DEC     60         RANGE
PTBAD   DAC     TBAD       I/O STATUS BLOCK ADDRESS
PNAD    DAC     NMAD       I/O COMPLETION RETURN ADDRESS
BFAD    BSZ     60         INPUT BUFFER
```

Output (OTP$)

The OTP$ function is used to initiate the transfer of a record from main memory to an I/O device.

FUNCTION ACTION

The OTP$ action routine checks to determine if the specified device has been previously reserved by a RSV$ request, and, if not, takes the error return. Next, the parameters specified in the OTP$ parameter list and the device control block are checked. OTP$ takes the error return if any of the parameters is illegal. The output request is then queued according to the priority level of the task which issued the OTP$ function call, and is initiated if the specified device is not currently busy processing another request, and control returns to the calling program at the normal return. Normally, the WIO$ function should be called immediately. However, further user code may be executed, and additional physical I/O requests made before calling WIO$.

MACRO CALL FORMAT

Location	Operation	Operand
[symbol]	OTP$	DCB address, error return address, buffer address, range, [mass memory segment number address], [I/O status block address], [I/O completion return address]

symbol - Optional. The symbolic location of the OTP$ macro instruction.

DCB address - The name of the device control block associated with the output device.

error return address - The address to which control is returned if an error is found in the OTP$ parameter list, the specified device control block parameters, or if the output device was not previously reserved.

NOTE: Control does not return to the error return address if an error occurs during the transfer to the output device. A user must test for a transfer error by checking the I/O status block after control is returned to the I/O completion return address.

buffer address - The address of the first word of the buffer from which the record is to be transferred. In a 64K system, this buffer may reside in a caller's A- or B-bank or partially in both as long as it does not cross from bank 1 to bank 2.

range - The number of words to be transferred, from a minimum of 1 to a maximum of 4095. The number of words transferred from the buffer will be determined by the expiration of the range count or the end of record (see Appendix B).

In a 64K system, the sign bit of the range word must be set if the buffer address is specified as a physical rather than a logical address.

In addition, the buffer address may be a logical or a physical address. If a physical address is specified, the sign bit of the range word must be set to indicate that this is so. Only nonrestricted activities may specify a physical address. If a restricted activity attempts to do so, it will be aborted.

mass memory segment number address - Optional. The address of a word containing the number of the disk segment into which record is to be placed.

This parameter is omitted for all but disk devices.

I/O status block address - Optional. The address of an 8-word block used by the device driver to schedule the I/O completion return to the user and to store the status information as described in Appendix B. The I/O status block address will be in the X-register when control is returned to the user at the I/O completion return address. If this parameter is omitted, the I/O status block address specified in the device control block will be used. If this parameter is present, it replaces the I/O status block address in the device control block and any subsequent I/O call using the same device control block uses the new I/O status block address.

NOTE: Updating of the I/O status block address in the DCB is performed by the execution of the inline macro expansion and not by the action routine.

I/O completion return address - Optional. The address where control is to be returned after I/O completion.

If this parameter is omitted, the I/O completion return address specified in the device control block will be used. If this parameter is present, it replaces the I/O completion return address in the device control block and any subsequent I/O call using the same device control block uses the new I/O completion return address.

NOTE: Updating of the I/O completion return address in the DCB is performed by the execution of the inline macro expansion and not by the action routine.

NORMAL RETURN

Control is returned to the calling program, at the instruction following the OTP$ function call, after the output request has been initiated, and probably before it has been completed. The calling program must wait for output completion by issuing a WIO$ function call.

I/O COMPLETION RETURN

After a WIO$ request has been completed, control will be returned to the I/O completion return address whether the record has been transferred successfully or unsuccessfully. The user must then test for output errors by checking the I/O status block (see I/O status block error code in Appendix B).

ERROR RETURN

Control is returned to the error return address specified in the OTP$ parameter list with the error code in the A-register when any of the following errors is detected:

A-register Contents (Octal)	Error Message
1	The generic device type specified in the device control block is not configured for this system.
3	The logical unit number specified in the device control block is not configured for this system.
4	The data mode specified in the device control block is not in the range 0 to 4, or is not a valid mode for the specified output device.
5	The number of words specified in the range parameter is less than 1 or more than 4095, or the buffer crosses the boundary between banks 1 and 2 (64K systems only).
10	The device was not previously reserved under the user ID specified in the device control block.
12	An output request was issued for an input only device.
14	An output request was issued for a disabled device.
15	The device was not reserved by the restricted activity which issued the OTP$ request.
32	The disk-resident driver for the specified device is not currently resident in main memory.
34	Requested executive function is not configured.
153	The mass memory segment number specified in the parameter list is illegal.
156	The output request for the disk cannot be accepted because a mount is in progress on the requested unit.

ACTION ROUTINE DETAILS

The following checks are made to determine if the output request is legal:

1. A test is made to determine if the generic device type and the logical unit number specified in the device control block have been configured into the system.
2. A test is made to determine if the requested device is enabled.
3. A test is made to determine if the requested device has been previously reserved by a RSV$ request.

4. A test is made to determine if the user ID specified in the device control block matches the ID for which the device was reserved (private devices only).

5. A test is made to determine if the data mode specified in the device control block is within the range from 0 to 4, and is a valid mode for the specified device.

6. A test is made to determine if the range value specified in the OTP$ parameter list is within the limits from 1 to 4095 and is no greater than the segment length if the specified device is a disk.

7. A test is made to determine if the specified device is an output device.

8. If the specified device is a disk, a test is made to determine if a removable disk pack is not in the process of being mounted on the requested unit, and a test is made to determine if the mass memory segment number specified is within the user area on a labeled disk volume.

9. If the activity is restricted, a test is made to determine if the specified device was reserved by the activity.

If the output request is not legal, control is returned to the error return address specified in the OTP$ parameter list with the error code in the A-register. If the output request is legal, the output request is queued according to the priority level of the task which issued the OTP$ request. If the requested device is currently busy processing another request, control is returned to the calling program at the normal return. If the requested device is not currently busy, the output request is initiated, and control is returned to the calling program at the normal return. When the calling program receives control at the normal return, a WIO$ request must be issued immediately if the calling program desires sequential output. If nonsequential output is desired, the calling program continues processing to the point that the output completion is required. At this point, a WIO$ request must be issued.

Upon completion of the requested output, the code specified by the I/O completion return address will be scheduled. When the calling program receives control at the I/O completion return address, the calling program must assume the responsibility of interrogating the I/O status block to determine if the output request was successful. Word 1 of the status block will be zero if no error occurred. (See Appendix B for the status information.)

OUTLINE PARAMETER LIST

Word Number	Operation	Operand
0	DAC	DCB address
1	DAC	error return address
2	DAC	buffer address
3	DEC	range
4	DAC	[mass memory segment number address]

Parameters in the outline parameter list are the same as those described above for the inline parameter list. If the OTP$ call is for a mass storage device, the segment number address must be specified. If the call is not for a mass storage device, this parameter should be omitted.

NOTE: If an outline parameter list is used and it is desired to update the I/O status block address or the I/O completion return address in the DCB, the user must precede the OTP$ function call with the instructions to store the new I/O status block address in the sixth word (word 5) of the DCB or the new I/O completion return address in the seventh word (word 6) of the DCB.

Examples:

The examples below illustrate the use of the OTP$ function to send multiple files as output to the magnetic tape nonsequentially. The calls are nonsequential because processing of data is done after the output request is issued. The device control block, DCNM, and the RSV$ request are shown in the examples given under "Assign Device Control Block (DCB$)" and RSV$ function, respectively. The I/O status block address and the I/O completion return address are changed in the DCB for each output request.

```
*                               OUTPUT MULTIPLE 60-WORD RECORDS TO
*                               MAGNETIC TAPE, LOGICAL UNIT NO. 1,
*                               NONSEQUENTIALLY
*
      JST    STB1               SET UP BUF1 FOR OUTPUT
RETT  OTP$   DCNM,ERA,BUF1,60,,STA1,RET1  OUTPUT BUF1
      JST    STB2               SET UP BUF2 FOR OUTPUT
      WIO$   PST1               WAIT FOR OUTPUT COMPLETION OF BUF1
RET1  LDA    1,1                OUTPUT OF BUF1 COMPLETE, FETCH STATUS,
*                               X CONTAINS POINTER TO STA1
      SZE                       DID AN ERROR OCCUR?
      JMP    ERR                YES, PROCESS ERROR
      IRS    CNTR               HAS ALL DATA BEEN OUTPUT?
      SKP                       NO
      JMP    FNSH               YES, ALL DATA OUTPUT
      OTP$   DCNM,ERA,BUF2,60,,STA2,RET2  OUTPUT BUF2
      JST    STB1               SETUP BUF1 FOR OUTPUT
      WIO$   PST2               WAIT FOR OUTPUT COMPLETION OF BUF2
RET2  LDA    1,1                OUTPUT OF BUF2 COMPLETE, FETCH STATUS,
*                               X CONTAINS POINTER TO STA2
      SZE                       DID AN ERROR OCCUR?
      JMP    ERR                YES, PROCESS ERROR
      IRS    CNTR               HAS ALL DATA BEEN OUTPUT?
      JMP    RETT               NO, GO TO OUTPUT BUF1
FNSH  ----                      DATA OUTPUT COMPLETE
*
*                               ERROR RETURN - A-REGISTER CONTAINS ERROR
*                               CODE
*
```

```
ERA     ---                     ERROR, ILLEGAL PARAMETER IN THE OTP$
*                               PARAMETER LIST OR IN THE DCB, DCNM
PST1    DAC     STA1            I/O STATUS BLOCK ADDRESS
PST2    DAC     STA2            I/O STATUS BLOCK ADDRESS
STA1    BSZ     8               I/O STATUS BLOCK
STA2    BSZ     8               I/O STATUS BLOCK
BUF1    BSZ     60              OUTPUT BUFFER
BUF2    BSZ     60              OUTPUT BUFFER
```

Using an outline parameter list, the above example would be written as follows:

```
*                               OUTPUT MULTIPLE 60-WORD RECORDS TO
*                               MAGNETIC TAPE, LOGICAL UNIT NO. 1,
*                               NONSEQUENTIALLY
*
        JST     STB1            SET UP BUF1 FOR OUTPUT
RETT    LDA     PST1            A-REGISTER SET TO DAC STA1
        STA     DCNM+5          STORE I/O STATUS BLOCK ADDRESS IN THE
                                DCB
        LDA     PRT1            A-REGISTER SET TO DAC RET1
*       STA     DCNM+6          STORE I/O COMPLETION RETURN ADDRESS IN
*                               THE DCB
        LDX     PLS8            POINTER TO PARAMETER LIST TO X
        OTP$    (X)             OUTPUT BUF1
        JST     STB2            SET UP BUF2 FOR OUTPUT
        WIO$    PST1            WAIT FOR OUTPUT COMPLETION OF BUF1
RET1    LDA     1,1             OUTPUT OF BUF1 COMPLETE, FETCH STATUS, X
*                               CONTAINS A POINTER TO STA1
        SZE                     DID AN ERROR OCCUR?
        JMP     ERR             YES, PROCESS ERROR
        IRS     CNTR            HAS ALL DATA BEEN OUTPUT?
        SKP                     NO
        JMP     FNSH            YES, ALL DATA OUTPUT
        LDA     PST2            A-REGISTER SET TO DAC STA2
        STA     DCNM+5          STORE I/O STATUS BLOCK ADDRESS IN THE DCB
        LDA     PRT2            A-REGISTER SET TO DAC RET2
*       STA     DCNM+6          STORE I/O COMPLETION RETURN ADDRESS IN
*                               THE DCB
        LDX     PLS9            POINTER TO PARAMETER LIST TDX
        OTP$    (X)             OUTPUT BUF2
        JST     STB1            SET UP BUF1 FOR OUTPUT
        WIO$    PST2            WAIT FOR COMPLETION OF BUF2
```

```
RET2  LDA   1,1          OUTPUT OF BUF2 COMPLETE, FETCH STATUS,
*                        X CONTAINS A POINTER TO STA2
      SZE                DID AN ERROR OCCUR?
      JMP   ERR          YES, PROCESS ERROR
      IRS   CNTR         HAS ALL DATA BEEN OUTPUT?
      JMP   RETT         NO, GO TO OUTPUT BUF1
FNSH  ----               DATA OUTPUT COMPLETE
*                        ERROR RETURN - A-REGISTER CONTAINS ERROR
*                        CODE
*
ERA   ---                ERROR, ILLEGAL PARAMETER IN THE OTP$ MACRO
*                        PARAMETER LIST OR IN THE DCB, DCNM
*
*                        PARAMETER LIST FOR BUF1 OUTPUT
*
PLS8  DAC   *+1          POINTER TO PARAMETER LIST
      DAC   DCNM         DCB ADDRESS
      DAC   ERA          ERROR RETURN ADDRESS
      DAC   BUF1         BUFFER ADDRESS
      DEC   60           RANGE
*
*                        PARAMETER LIST FOR BUF2 OUTPUT
*
PLS9  DAC   *+1          POINTER TO PARAMETER LIST
      DAC   DCNM         DCB ADDRESS
      DAC   ERA          ERROR RETURN ADDRESS
      DAC   BUF2         BUFFER ADDRESS
      DEC   60           RANGE
PST1  DAC   STA1         I/O STATUS BLOCK ADDRESS
PRT1  DAC   RET1         I/O COMPLETION RETURN ADDRESS
STA1  BSZ   8            I/O STATUS BLOCK
STA2  BSZ   8            I/O STATUS BLOCK
PST2  DAC   STA2         I/O STATUS BLOCK ADDRESS
PRT2  DAC   RET2         I/O COMPLETION RETURN ADDRESS
BUF1  BSZ   60           OUTPUT BUFFER
BUF2  BSZ   60           OUTPUT BUFFER
```

The example below illustrates the use of the INP$, OTP$ and EOF$ functions to copy an ASCII file from one magnetic tape to another, using nonsequential, parallel I/O. This technique is faster than sequential I/O because the I/O operations are performed on both devices simultaneously. The time saved by the technique is at a maximum when the data processing speeds of the two peripheral devices are equal.

The program uses two I/O buffers. After initially inputting the first record of the file into the first buffer, it outputs this record and simultaneously inputs the second record into the other buffer. It then alternates the buffer addresses and repeats the process until end of file is detected. The same set of I/O complete code services all I/O operations, including the writing of an end-of-file mark to the output tape. The program keeps a count (REQCNT) of the number of I/O requests for which a WIO$ call is still pending; even if processing is interrupted by an error, the program must still issue a WIO$ call for all such outstanding I/O requests before terminating.

Outline parameter lists are used for all function calls except WIO$. This is necessary for the INP$ and OTP$ calls because the program has to alter the contents of the parameter lists when swapping buffer addresses. Note also that the EOF$ (write end-of-file) call can use the OTP$ parameter list. (The EOF$ function is described next in this manual.)

NOTE: To derive maximum processing speed from this parallel I/O technique, the two magnetic tape drives used should be attached to two different hardware controllers, since I/O operations on two different magnetic tape drives attached to the same controller are (with the exception of a Rewind operation) performed sequentially rather than in parallel.

```
*
* COPY 60-WORD ASCII RECORDS FROM MAGNETIC TAPE LOGICAL UNIT 0 TO
* MAGNETIC TAPE LOGICAL UNIT 4, USING NONSEQUENTIAL, INTERLEAVED I/O
*
*
* INITIALIZE
*
        CRA
        STA     REQCNT    I/O REQUEST PENDING COUNT
        STA     ENFLG     END OF PROCESSING FLAG
        STA     EOFFLG    END-OF-FILE READ FLAG
*
* INPUT RECORD FROM MAGNETIC TAPE LOGICAL UNIT 0
*
```

```
*
INPUT    LDX      PINPPL     POINT TO INP$ PARAMETER LIST
         INP$*    (X)        INPUT
*
* WAIT FOR I/O
*
WAIT     IRS      REQCNT     TALLY I/O REQUEST COUNT
*
WAIT2    WIO$*               WAIT FOR I/O
*
* I/O COMPLETION RETURN FOR ALL REQUESTS
*
IOCOMP   LDA      REQCNT
         SUB      =1
         STA      REQCNT     OPERATION COMPLETE; DECREMENT COUNT
*
         LDA      1,1        WORD 1 OF STATUS BLOCK
         SNZ                 WAS THERE AN ERROR?
         JMP      TRC        NO, ALL OK; GO ON
*
         ANA      ='177767   DISCARD END-OF-FILE BIT
         SZE                 WAS IT END OF FILE AND NO OTHER ERROR?
         JMP      IOERR      NO, IT WAS AN ERROR - HANDLE IT
         IRS      EOFFLG     MARK END OF FILE READ
*
* DECIDE WHAT TO DO NEXT
*
TRC      LDA      REQCNT     I/O REQUEST PENDING COUNT
         SZE                 IS THERE STILL A REQUEST OUTSTANDING?
         JMP      WAIT2      YES - WAIT FOR IT
*
         LDA      ENDFLG     END OF PROCESSING FLAG
         SZE                 HAVE WE FINISHED PROCESSING?
         JMP      WRAPUP     YES - TERMINATE PROGRAM
*
         LDA      EOFFLG     END-OF-FILE FLAG
         SZE                 HAVE WE READ END OF FILE?
         JMP      EOF        YES - WRITE END OF FILE ON OUTPUT TAPE
*
* SWAP BUFFERS, RE-OUTPUT THE BUFFER JUST INPUT, AND INPUT TO
* THE OTHER BUFFER
*
         LDA      INBUF      CURRENT INPUT BUFFER
         IMA      OUTBUF     SWAP WITH OUTPUT BUFFER
         STA      INBUF
*
         LDX      POTPPL     POINT TO OTP$ PARAMETER LIST
         OTP$*    (X)        OUTPUT
*
         IRS      REQCNT     TALLY REQUEST COUNT
         JMP      INPUT      INPUT TO OTHER BUFFER
*
* WRITE END OF FILE TO OUTPUT TAPE
*
EOF      IRS      ENDFLG     TERMINATE PROCESSING NEXT TIME AROUND
*
         LDX      PEOFPL     POINT TO EOF$ PARAMETER LIST
         EOF$*    (X)        WRITE END OF FILE
*
         JMP      WAIT       WAIT FOR IT
*
```

```
* HANDLE ERROR RETURNS FROM SYSTEM FUNCTIONS
*
SYSERR  IRS      ENDFLG    TERMINATE PROCESSING NEXT TIME AROUND
         .
         .
         .
        JMP      TRC       WAIT FOR OUTSTANDING I/O (IF ANY)
*                          PROCESS ERROR
* HANDLE I/O DEVICE ERRORS
*
IOERR   IRS      ENDFLG    TERMINATE PROCESSING NEXT TIME AROUND
                           PROCESS ERROR
         .
         .
         .
        JMP      TRC       WAIT FOR OUTSTANDING I/O (IF ANY)
*
* TERMINATE PROCESSING
*
WRAPUP  ----
         .
         .
         .
*
* CONSTANTS AND VARIABLES
*
REQCNT  BSZ      1         I/O REQUEST PENDING COUNT
EOFFLG  BSZ      1         END-OF-FILE READ FLAG
ENDFLG  BSZ      1         END OF PROCESSING FLAG
*
*
        FIN                LITERALS
*
* OUTLINE PARAMETER LIST FOR INPUT CALL
*
PINPPL  DAC      *+1       POINTER TO INP$ PARAMETER LIST
        DAC      INPDCB    POINTER TO DCB
        DAC      SYSERR    ERROR RETURN
INBUF   DAC      BUF1      CURRENT INPUT BUFFER ADDRESS
        DEC      60        RANGE
*
* OUTLINE PARAMETER LIST FOR OUTPUT AND WRITE END-OF-FILE CALLS
*
POTPPL  DAC      *+1       POINTER TO OTP$ PARAMETER LIST
        DAC      OTPDCB    POINTER TO DCB
        DAC      SYSERR    ERROR RETURN
OUTBUF  DAC      BUF2      CURRENT OUTPUT BUFFER ADDRESS
        DEC      60        RANGE
*
PEOFPL  EQU      POTPPL    EOF$ SHARES PARAMETER LIST WITH OTP$
*
* DCB'S, STATUS BLOCKS AND BUFFERS
*
INPDCB  DCB$*    10,0,0,'125252,ISTAT,IOCOMP
*
OTPDCB  DCB$*    10,4,0,'125252,OSTAT,IOCOMP
*
ISTAT   BSZ      8         STATUS BLOCK FOR INPUT
OSTAT   BSZ      *         STATUS BLOCK FOR OUTPUT
*
BUF     BSZ      60        FIRST BUFFER
BUF2    BSZ      60        SECOND BUFFER
```

End of File (EOF$)

The EOF$ function is used to initiate the writing of an end-of-file record on the specified device.

FUNCTION ACTION

The EOF$ action routine checks to determine if the specified device has been previously reserved by a RSV$ request; if the device has not been reserved, the error return is taken. Next, the parameters specified in the device control block are checked and the error return taken if any of the parameters is illegal. Then, the EOF$ request is queued according to the priority level of the task which issued the EOF$ request, and is initiated if the device is not currently busy processing another request. Control returns to the calling program at the normal return address. Normally, the WIO$ function should be called immediately. However, further user code may be executed, and additional physical I/O requests made before calling WIO$.

MACRO CALL FORMAT

Location	Operation	Operand
[symbol]	EOF$	DCB address, error return address, [I/O status block address], [I/O completion return address]

symbol - Optional. The symbolic location of the EOF$ macro instruction.

DCB address - The name of the device control block associated with the device.

error return address - The address to which control is returned if an error is found in the specified device control block parameters or if the specified device was not previously reserved.

NOTE: Control does not return to the error return address if an error occurs during the writing of the end of file. The user must test for such an error by checking the I/O status block after control is returned to the I/O completion return address.

I/O status block address - Optional. The address of an 8-word block used by the device driver to schedule the I/O completion return to the user and to store the status information as described in Appendix B. The I/O status block address will be in the X-register when control is returned to the user at the I/O completion return address. If this parameter is omitted, the I/O status block address specified in the device control block is used. If this parameter is present, it replaces the I/O status block address in the device control block and any subsequent I/O call using the same device control block uses the new I/O status block address.

NOTE: Updating of the I/O status block address in the DCB is performed by the execution of the inline macro expansion and not by the action routine.

I/O completion return address - Optional. The address where control is to be returned after I/O completion. If this parameter is omitted, the I/O completion return address specified in the device control block is used. If this parameter is present, it replaces the I/O completion return address in the device control block and any subsequent I/O call using the same device control block uses the new I/O completion return address.

NOTE: Updating of the I/O completion return address in the DCB is performed by the execution of the inline macro expansion and not by the action routine.

NORMAL RETURN

Control is returned to the calling program, at the instruction following the EOF$ request, after the writing of the end of file has been initiated, and probably before it has been completed. The calling program must wait for the completion of the writing of the end of file by issuing a WIO$ function call.

I/O COMPLETION RETURN

After a WIO$ request has been completed, control is returned to the I/O completion return address whether the end of file has been written successfully or unsuccessfully. The user must then test for an error which occurred during the writing of the end of file by checking the I/O status block (see I/O Status Block Error Code in Appendix B).

ERROR RETURN

Control is returned to the error return address specified in the EOF$ parameter list with the error code in the A-register when any of the following errors is detected:

A-register Contents (Octal)	Error Condition
1	The generic device type specified in the device control block is not configured for this system.

A-register Contents (Octal)	Error Condition
3	The logical unit number specified in the device control block is not configured for this system.
7	The EOF$ request was issued for a device other than magnetic tape, the high-speed paper tape punch, cassette tape, card punch, or the ASR paper tape punch.
10	The device has not been previously reserved under the user ID specified in the device control block.
14	An EOF$ request was issued for a disabled device.
15	The device was not reserved by the restricted activity which issued the EOF$ request.
32	The disk-resident driver for the specified device is not currently resident in main memory.
34	Requested executive function is not configured.

MACRO ROUTINE ACTION DETAILS

The following checks are made to determine if the EOF$ request is legal:

1. A test is made to determine if the generic device type and the logical unit number specified in the device control block have been configured into the system.
2. A test is made to determine if the specified device is enabled.
3. A test is made to determine if the specified device has been previously reserved by a RSV$ function call.
4. A test is made to determine if the user ID specified in the device control block matches the ID for which the device was reserved.
5. A test is made to determine if the requested device is magnetic tape, the high-speed paper tape punch, cassette tape, card punch, or the ASR paper tape punch.
6. If the activity is restricted, a test is made to determined if the specified device was reserved by the activity.

If the EOF$ request is not legal, control is returned to the error return address specified in the EOF$ parameter list with the error code in the A-register. If the EOF$ request is legal, the EOF$ request is queued according to the priority level of the task which issued the EOF$ request. If the specified device is currently busy processing another request, control is returned to the calling program at the normal return. If the specified device is not currently busy, the EOF$ request is initiated, and control is returned to the calling program at the normal return. When the calling program receives control at the normal return, a WIO$ request must be issued immediately if the calling program desires sequential I/O. If nonsequential I/O is desired, the calling program continues processing to the point that the end-of-file completion is required.

At this point, a WIO$ request must be issued. Upon completion of the end of file, the code specified by the I/O completion return address is scheduled. When the calling program receives control at the I/O completion return address, the calling program must assume the responsibility of interrogating the I/O status block to determine if the EOF$ request was successful.

Word 1 of the status block is zero if no error occurred. (See Appendix B for the status information.)

OUTLINE PARAMETER LIST

Word Number	Operation	Operand
0	DAC	DCB address
1	DAC	error return address

Parameters in the outline parameter list are the same as those described above for the inline parameter list.

NOTE: If an outline parameter list is used and it is desired to update the I/O status block address or the I/O completion return address in the DCB, the user must precede the function call with the instructions to store the new I/O status block address in the sixth word (word 5) of the DCB or the new I/O completion return address in the seventh word (word 6) of the DCB.

Examples:

The following examples illustrate the use of the EOF$ function to write an end-of-file record on a magnetic tape. The device control block, DCNM, and the RSV$ request, which must be executed prior to the request to write an end of file, have been illustrated previously in the DCB$ example. The EOF$ request is sequential, and the I/O status block address and the I/O completion return address remain as shown in DCNM, since they are not specified in the EOF$ request.

```
*                               WRITE END OF FILE ON MAGNETIC TAPE
*
      EOF$    DCNM,EFER         WRITE EOF
*     WIO$                      SEQUENTIAL, WAIT FOR WRITING OF EOF TO
*                               BE COMPLETED
RTAD  LDA     TBAD+1            EOF WRITTEN, FETCH STATUS
      SZE                       DID AN ERROR OCCUR?
      JMP     ERR               YES, PROCESS ERROR
      ---                       NO ERROR, CONTINUE
*
*                               ERROR RETURN - A-REGISTER CONTAINS THE
*                               ERROR CODE
*
EFER  ---                       ERRORS, ILLEGAL PARAMETER IN THE DCB,
                                DCNM
```

Using an outline parameter list, the above examples would be written as follows:

```
*                                     WRITE END OF FILE ON MAGNETIC TAPE
*
        LDX      PL13                 POINTER TO PARAMETER LIST TO X
        EOF$     (X)                  WRITE EOF
*       WIO$                          SEQUENTIAL, WAIT FOR WRITING OF EOF TO
*                                     BE COMPLETED
RTAD    LDA      TBAD+1               I/O COMPLETION RETURN, FETCH STATUS
        SZE                           DID AN ERROR OCCUR?
        JMP      ERR                  YES, PROCESS ERROR
        ---                           NO ERROR, CONTINUE
*
*                                     ERROR RETURN - A-REGISTER CONTAINS THE
*                                     ERROR CODE
*
EFER    ---                           ERROR, ILLEGAL PARAMETER IN THE DCB,
*                                     DCNM
*
*                                     PARAMETER LIST
*
PL13    DAC      *+1                  POINTER TO PARAMETER LIST
        DAC      DCNM                 DCB ADDRESS
        DAC      EFER                 ERROR RETURN ADDRESS
```

SPF$

Space File (SPF$)

The SPF$ function is used to initiate the spacing past one or more file marks on a magnetic tape or cassette tape.

FUNCTION ACTION

The SPF$ action routine checks to determine if the specified device has been previously reserved by a RSV$ function call. If the device has not been reserved, the error return is taken. Next, the parameters specified in the SPF$ parameter list and the device control block are checked, and the error return is taken if any one of the parameters is illegal. If legal, the space file request is queued according to the priority level of the task which issued the SPF$ request, the space file request is initiated if the specified device is not currently busy processing another request, and return is made to the calling program at the normal return. Normally, the WIO$ function should be called immediately. However, further user code may be executed and additional physical I/O requests made before calling WIO$.

The SPF$ function causes the specified number of file marks to be passed, in either the forward or reverse direction. Thus, after spacing backwards, the tape is positioned such that input operation would encounter a file mark immediately. To position a tape at the beginning of a file which has been passed on the tape, two SPF$ requests must be issued: one to move the tape backwards, and one forwards to position the tape on the proper side of the file mark.

MACRO CALL FORMAT

Location	Operation	Operand
[symbol]	SPF$	DCB address, error return address, number of files address, [I/O status block address], [I/O completion return address]

symbol - Optional. The symbolic location of the SPF$ macro instruction.

DCB address - The name of the device control block associated with the space file device.

error return address - The address to which control is returned if an error is found in the SPF$ parameter list, the specified device control block parameters, or if the specified device was not previously reserved.

NOTE: Control does not return to the error return address if an error occurs during the spacing of a file. The user must test for such an error by checking the I/O status block after control is returned to the I/O completion return address.

number of files address - The address of a word containing the number of files to be spaced. A positive number indicates forward spacing, and a negative number indicates backward spacing.

NOTE: Backward spacing is not allowed for tape cassette files.

I/O status block address - Optional. The address of an 8-word I/O status block used by the device driver to schedule the I/O completion return to the user and to store the status information as described in Appendix B. The I/O status block address will be in the X-register when control is returned to the user at the I/O completion return address. If this parameter is omitted, the I/O status block address specified in the device control block is used. If this parameter is present, it is used to replace the I/O status block address in the device control block and any subsequent I/O call using the same device control block uses the new I/O status block address.

NOTE: Updating of the I/O status block address in the DCB is performed by the execution of the inline macro expansion and not by the action routine.

I/O completion return address - Optional. The address to which control is returned after I/O completion.

If this parameter is omitted, the I/O completion return address specified in the device control block is used. If this parameter is present, it is used to replace the I/O completion return address in the device control block and any subsequent I/O call using the same device control block uses the new I/O completion return address.

NOTE: Updating of the I/O completion return address in the DCB is performed by the execution of the inline macro expansion and not by the action routine.

NORMAL RETURN

Control is returned to the calling program, at the instruction following the SPF$ request, after the spacing of the file has been initiated, and probably before it has been completed. The calling program must wait for space file completion by issuing a WIO$ function call.

I/O COMPLETION RETURN

After a WIO$ request has been completed, control is returned to the I/O completion return address whether the file(s) has been spaced successfully or unsuccessfully. The user must then test for space file errors by checking the I/O status block (see I/O status block error code in Appendix B).

ERROR RETURN

Control is returned to the error return address specified in the SPF$ parameter list with the error code in the A-register when any of the following errors is detected:

A-register Contents (Octal)	Error Condition
1	The generic device type specified in the device control block is not configured for this system.
3	The logical unit number specified in the device control block is not configured for this system.
6	The number of files to be spaced is illegal.
7	The space file request was issued for a device other than magnetic tape or cassette tape.
10	The device was not previously reserved under the user ID specified in the device control block.
14	A space file request was issued for a disabled device.
15	The device was not reserved by the restricted activity which issued the SPF$ request.
32	The disk-resident driver for the specified device is not currently resident in main memory.
34	The requested system function is not configured.

MACRO ROUTINE ACTION DETAILS

The following checks are made to determine if the SPF$ request is legal:

1. A test is made to determine if the generic device type and the logical unit number specified in the device control block have been configured into the system.
2. A test is made to determine if the specified device is enabled.
3. A test is made to determine if the specified device has been previously reserved by a RSV$ function call.
4. A test is made to determine if the user ID specified in the device control block matches the ID for which the device was reserved.
5. A test is made to determine if the specified device is magnetic tape or cassette tape.
6. A test is made to determine if the number of files to be spaced is legal. The number of files is illegal if zero for magnetic tape or if zero or negative for cassette tape.
7. If the activity is restricted, a test is made to determine if the specified device was reserved by the activity.

If the SPF$ request is not legal, control is returned to the error return address specified in the SPF$ parameter list with the error code in the A-register. If the SPF$ request is legal, the SPF$ request is queued according to the priority level of the task which issued the SPF$ request. If the specified device is currently busy processing another request, control is returned to the calling program at the normal return. If the specified device is not currently busy, the SPF$ request is initiated, and control is returned to the calling program at the normal return. When the calling program receives control at the normal return, a WIO$ request must be issued immediately if the calling program desires sequential I/O. If nonsequential I/O is desired, the calling program continues processing to the point that the space file completion is required. At this point, a WIO$ request must be issued. Upon completion of the space file, the code specified by the I/O completion return address is scheduled. When the calling program receives control at the I/O completion return address, the calling program must assume the responsibility of interrogating the I/O status block to determine if the SPF$ request was successful.

Word 1 of the status block contains an 8 if no error occurred. (See Appendix B for the status information.)

OUTLINE PARAMETER LIST

Word Number	Operation	Operand
0	DAC	DCB address
1	DAC	error return address
2	DAC	number of files address

Parameters in the outline parameter list are the same as those described above for the inline parameter list.

NOTE: If an outline parameter list is used and it is desired to update the I/O status block address or the I/O completion return address in the DCB, the user must precede the SPF$ function call with the instructions to store the new I/O status block address in the sixth word (word 5) of the DCB or the new I/O completion return address in the seventh word (word 6) of the DCB.

Examples:

The following examples illustrate the use of the SPF$ function to forward space five files on a magentic tape. The device control block, DCNM, and the RSV$ request, which must be executed prior to the forward space, have been illustrated previously in the DCB$ example. The I/O status block address and the I/O completion return address are not referenced in the SPF$ function call, so they remain as shown in DCNM.

```
*                                    FORWARD SPACE 5 FILES
*
      SPF$    DCNM,ERSP,P5           FORWARD SPACE
      WIO$                           SEQUENTIAL, WAIT FOR SPACE FILE COMPLETION
RTAD  LDA     TBAD+1                 FORWARD SPACE COMPLETED, FETCH STATUS
      CAS     EOF                    DID AN ERROR OCCUR?
      JMP     ERR                    YES, PROCESS ERROR
      SKP                            NO ERROR, CONTINUE
      JMP     ERR                    YES, PROCESS ERROR
NOERR ----
*                                    ERROR RETURN - A-REGISTER CONTAINS THE
*                                    ERROR CODE
*
ERSP  ---                            ERROR, ILLEGAL PARAMETER IN THE SPF$ PARAMETER
*                                    LIST OR IN THE DCB, DCNM
EOF   OCT     10                     I/O STATUS BLOCK INDICATOR FOR END OF FILE
P5    DEC     5                      NUMBER OF FILES TO BE SPACED FORWARD
```

Using an outline parameter list, the above example would be written as follows:

```
*                                    FORWARD SPACE 5 FILES
*
      LDX     PL11                   POINTER TO PARAMETER LIST TO X
      SPF$    (X)                    FORWARD SPACE FILES
      WIO$                           SEQUENTIAL, WAIT FOR SPACE FILE COMPLETION
RTAD  LDA     TBAD+1                 FORWARD SPACE COMPLETED, FETCH STATUS
      CAS     EOF                    DID AN ERROR OCCUR?
      JMP     ERR                    YES, PROCESS ERROR
      SKP                            NO ERROR, CONTINUE
      JMP     ERR                    YES, PROCESS ERROR
NOERR ----
*
*                                    ERROR RETURN - A-REGISTER CONTAINS THE
*                                    ERROR CODE
*
ERSP  ---                            ERROR, ILLEGAL PARAMETER IN THE SPF$ PARAMETER
*                                    LIST OR IN THE DCB, DCNM
*                                    PARAMETER LIST
*
```

```
PL11  DAC    *+1         POINTER TO PARAMETER LIST
      DAC    FCNM        DCB ADDRESS
      DAC    ERSP        ERROR RETURN ADDRESS
      DAC    P5          NUMBER OF FILES ADDRESS
*
*
EOF   OCT    10          I/O STATUS BLOCK INDICATOR FOR END OF FILE
P5    DEC    5           NUMBER OF FILES TO BE SPACED FORWARD
```

SPR$

Space Record (SPR$)

The SPR$ function is used to initiate the spacing of one or more records on a magnetic tape or cassette tape.

FUNCTION ACTION

The SPR$ action routine checks to determine if the specified device has been previously reserved by a RSV$ request; if the device has not been reserved, the error return is taken. Otherwise, the parameters specified in the SPR$ parameter list and the device control block are checked, and the error return is taken if any of the parameters is illegal. If legal, the SPR$ request is queued according to the priority of the task which issued the SPR$ request. The SPR$ request is initiated if the specified device is not currently busy processing another request, and the normal return is taken. Normally, the WIO$ function should be called immediately. However, further user code may be executed, and additional physical I/O requests made before calling WIO$.

MACRO CALL FORMAT

Location	Operation	Operand
[symbol]	SPR$	DCB address, error return address, number of records address, [I/O status block address], [I/O completion return address_

symbol - Optional. The symbolic location of the SPR$ macro instruction.

DCB address - The name of the device control block associated with the space record device.

error return address - The address to which control is returned if an error is found in the SPR$ parameter list, the specified device control block parameters, or if the specified device was not previously reserved.

NOTE: Control does not return to the error return address if an error occurs during the spacing of a record. A user must test for such an error by checking the I/O status block after control is returned to the I/O completion return address.

number of records address - The address of a word containing the number of records to be spaced. A positive number indicates forward spacing, and a negative number indicates backward spacing.

NOTE: Backward spacing is not allowed for files on cassette tape.

I/O status block address - Optional. The address of an 8-word block used by the device driver to schedule the I/O completion return to the user and to store the status information as described in Appendix B. The I/O status block address will be in the X-register when control is returned to the user program at the I/O completion return address. If this parameter is omitted, the I/O status block address specified in the device control block is used. If this parameter is present, it replaces the I/O status block address in the device control block and any subsequent I/O call using the same device control block uses the new I/O status block address.

NOTE: Updating of the I/O status block address in the DCB is performed by the execution of the inline macro expansion and not by the action routine.

I/O completion return address - Optional. The address where control is to be returned after I/O completion.

If this parameter is omitted, the I/O completion return address specified in the device control block is used. If this parameter is present, it replaces the I/O completion return address in the device control block and any subsequent I/O call using the same device control block uses the new I/O completion return address.

NOTE: Updating of the I/O completion return address in the DCB is performed by the execution of the inline macro expansion and not by the action routine.

NORMAL RETURN

Control is returned to the calling program, at the instruction following the SPR$ request, after the spacing of the record has been initiated, and probably before it has been completed. The calling program must wait for space record completion by issuing a WIO$ function call.

I/O COMPLETION RETURN

After a WIO$ request has been completed, control is returned to the I/O completion return address whether the record(s) has been spaced successfully or unsuccessfully. The user must then test for space record errors by checking the I/O status block (see I/O status block error code in Appendix B).

ERROR RETURN

Control is returned to the error return address specified in the SPR$ parameter list with the error code in the A-register when any of the following errors is detected:

A-register Contents (Octal)	Error Condition
1	The generic device type specified in the device control block is not configured for this system.
3	The logical unit number specified in the device control block is not configured for this system.

A-register Contents (Octal)	Error Condition
6	The number of records to be spaced is illegal.
7	The SPR$ request was issued for a device other than magnetic tape or cassette tape.
10	The device was not previously reserved under the user ID specified in the device control block.
14	A space record request was issued for a disabled device.
15	The device was not reserved by the restricted activity which issued the SPR$ request.
32	The disk-resident driver for the specified device is not currently resident in main memory.
34	Requested executive function is not configured.

ACTION ROUTINE DETAILS

The following checks are made to determine if the SPR$ request is legal:

1. A test is made to determine if the generic device type and the logical unit number specified in the device control block have been configured into the system.
2. A test is made to determine if the specified device is enabled.
3. A test is made to determine if the specified device has been previously reserved by a RSV$ macro instruction.
4. A test is made to determine if the user ID specified in the device control block matches the ID for which the device was reserved.
5. A test is made to determine if the specified device is magnetic tape or cassette tape.
6. A test is made to determine if the number of records to be spaced is legal. The number of records is illegal if zero for magnetic tape or if zero or negative for cassette tape.
7. If the activity is restricted, a test is made to determine if the specified device was reserved by the activity.

If the SPR$ request is not legal, control is returned to the error return address specified in the SPR$ parameter list with the error code in the A-register. If the SPR$ request is legal, the SPR$ request is queued according to the priority level of the task which issued the SPR$ request. If the specified device is currently busy processing another request, control is returned to the calling program at the normal return. If the specified device is not currently busy, the SPR$ request is initiated, and control is returned to the calling program at the normal return. When the calling program receives control at the normal return, a WIO$ request must be issued immediately if the calling program desires sequential I/O. If nonsequential I/O is desired, the calling program continues processing to the point that the space record completion is required. At this point, a WIO$ request must be issued. Upon completion of the space record, the code specified by the I/O completion return address will be scheduled. When the calling program receives control at the I/O completion return address, the calling program must assume the responsibility of interrogating the I/O status block to determine if the SPR$ request was successful.

Word 1 of the status block will be 0 if no error occurred. (See Appendix B for the status information).

OUTLINE PARAMETER LIST

Word Number	Operation	Operand
0	DAC	DCB address
1	DAC	error return address
2	DAC	number of records address

The parameters in the outline parameter list are the same as those described above for the inline parameter list.

NOTE: If an outline parameter list is used and it is desired to update the I/O status block address or the I/O completion return address in the DCB, the user must precede the SPR$ function call with the instructions to store the new I/O status block address in the sixth word (word 5) of the DCB or the new I/O completion return address in the seventh word (word 6) of the DCB.

Examples:

The following examples illustrate the use of the SPR$ function to backward space two records on a magnetic tape. The device control block, DCNM, and the RSV$ request, which must be executed prior to the backward space, have been illustrated previously in the DCB$ example. As in the examples in the description of the SPF$ function the I/O status block address and the I/O completion return address remain as shown in DCNM.

```
*                               BACKWARD SPACE 2 RECORDS
*
        SPR$    DCNM,ERSR,M2    BACKWARD SPACE RECORDS
*       WIO$                    SEQUENTIAL, WAIT FOR SPACE RECORD
*                               COMPLETION
RTAD    LDA     TBAD+1          BACKWARD SPACE COMPLETED, FETCH STATUS
        SZE                     DID AN ERROR OCCUR?
        JMP     ERR             YES, PROCESS ERROR
        ---                     NO ERROR, CONTINUE
*
*                               ERROR RETURN - A-REGISTER CONTAINS THE
*                               ERROR CODE
*
ERSR    ---                     ERROR, ILLEGAL PARAMETER IN THE SPR$
*                               PARAMETER LIST OR IN THE DCB, DCNM
M2      DEC     -2              NUMBER OF RECORDS TO BE SPACED BACKWARD
```

Using an outline parameter list, the above example would be written as follows:

```
*                                   BACKWARD SPACE 2 RECORDS
*
       LDX     PL12                 POINTER TO PARAMETER LIST TO X
       SPR$    (X)                  BACKWARD SPACE RECORDS
*      WIO$                         SEQUENTIAL, WAIT FOR SPACE RECORD
*                                   COMPLETION
RTAD   LDA     TBAD+1               BACKWARD SPACE COMPLETED, FETCH STATUS
       SZE                          DID AN ERROR OCCUR?
       JMP     ERR                  YES, PROCESS ERROR
       ---                          NO ERROR, CONTINUE
*
*                                   ERROR RETURN - A-REGISTER CONTAINS THE
*                                   ERROR CODE
*
ERSR   ---                          ERROR, ILLEGAL PARAMETER IN THE SPR$
*                                   PARAMETER OR IN THE DCB, DCNM
*                                   PARAMETER LIST
*
PL12   DAC     *+1                  POINTER TO PARAMETER LIST
       DAC     DCNM                 DCB ADDRESS
       DAC     ERSR                 ERROR RETURN ADDRESS
       DAC     M2                   NUMBER OF RECORDS ADDRESS
M2     DEC     -2                   NUMBER OF RECORDS TO BE SPACED BACKWARD
```

Rewind (RWD$)

The RWD$ function is used to initiate the rewinding of a magnetic tape or cassette tape.

FUNCTION ACTION

The RWD$ action routine checks to determine if the specified device has been previously reserved by a RSV$ request; if it has not been reserved, the error return is taken. If it has been reserved, the parameters specified in the device control block are checked, and the error return is taken if any of the parameters is illegal. If legal, the rewind request is inserted at the beginning of the queue for the requested device. The rewind request is initiated if the specified device is not currently busy processing another request, and the normal return is taken. Normally, the WIO$ function should be called immediately. However, further user code may be executed, and additional physical I/O requests made before calling WIO$.

MACRO CALL FORMAT

Location	Operation	Operand
[symbol]	RWD$	DCB address, error return address, [I/O status block address], [I/O completion return address]

symbol - Optional. The symbolic location of the RWD$ macro instruction.

DCB address - The name of the device control block associated with the rewind device.

error return address - The address to which control is returned if an error is found in the specified device control block parameters, or if the specified device was not previously reserved.

NOTE: Control does not return to the error return address if an error occurs during the rewinding of a magnetic tape or cassette tape. An error during the rewind must be tested for by checking the I/O status block after control is returned to the I/O completion return address.

I/O status block address - Optional. The address of an 8-word block used by the device driver to schedule the I/O completion return to the user and to store the status information as described in Appendix B. The I/O status block address will be in the X-register when control is returned to the user at the I/O completion return address. If this parameter is omitted, the I/O status block address specified in the device control block is used. If this parameter is present, it replaces the I/O status block address in the device control block and any subsequent I/O call using the same device control block uses the new I/O status block address.

NOTE: Updating of the I/O status block address in the DCB is performed by the execution of the inline macro expansion and not by the action routine.

I/O completion return address - Optional. The address where control is to be returned after I/O completion.

If this parameter is omitted, the I/O completion return address specified in the device control block is used. If this parameter is present, it replaces the I/O completion return address in the device control block and any subsequent I/O call using the same device control block uses the new I/O completion return address.

NOTE: Updating of the I/O completion return address in the DCB is performed by the execution of the inline macro expansion and not by the action routine.

NORMAL RETURN

Control is returned to the calling program, at the instruction following the RWD$ request, after the rewind has been initiated, and probably before it has been completed. The calling program must wait for rewind completion by issuing a WIO$ function call.

I/O COMPLETION RETURN

After a WIO$ request has been completed, control will be returned to the I/O completion return address whether the tape has been rewound successfully or unsuccessfully. The user must then test for errors that may have occurred during the rewind, by checking the I/O status block (see I/O Status Block Error Code in Appendix B).

ERROR RETURN

Control is returned to the error return address specified in the RWD$ parameter list with the error code in the A-register when any of the following errors is detected:

A-register Contents (Octal)	Error Condition
1	The generic device type specified in the device control block is not configured for this system.
3	The logical unit number specified in the device control block is not configured for this system.
7	The RWD$ request was issued for a device other than magnetic tape or cassette tape.

The following is a description of the status information returned to the user when control is transferred to the user's I/O completion return. The states described below are indicated if the appropriate bit is set.

Word 1 (second word) of I/O Status Block

Bit	Interpretation
1	Word 4 contains the hardware status word which indicates the error. If no error, this bit = 0.
2-3	Reserved
4	Not operational
5	Disabled
6-9	Reserved
10	Recovery error
11-16	Reserved

Word 3 (fourth word) of I/O Status Block

Word 3 contains the actual number of words transferred.

Status Information for Line Printer Types 552x

Word 4 (fifth word) of I/O Status Block

In addition to the software status returned in word 1 of the I/O status block, the following hardware status is always returned in word 4 of the I/O status block.

Bit	Interpretation
1	Busy indicator
2	Ready indicator
3	Paper advancing indicator
4	Vertical format tape channel 2 (end of form)
5-13	Always zero
14	Cycle indicator
15-16	Always zero

Status Information for Line Printer Types 554x, 555x, and 556x

Word 4 (fifth word) of I/O Status Block

In addition to the software status returned in word 1 of the I/O status block, the following hardware status is always returned in word 4 of the I/O status block.

LINE PRINTERS (TYFES 5520, 5524, 5526, 5527, 5529, 5541-2, 5551-2, 5565-9)

The only legal physical I/O request for the line printer is output (OTP$); the only legal data mode is 0 (ASCII). When output to the line printer is requested, the characters that are to be printed must be packed two characters per word; the first word must contain a right-justified control character and must be included in the range parameter of the OTP$ function call. There are no restrictions on the contents of the left byte of this word. If the number of words to be printed exceeds the limit for one line on the line printer, only one line is printed and the remaining words in the user's output buffer are ignored (Printer Types 554x will overprint). The control character is acted upon before the line is printed.

The first word of the user's output buffer must contain an ASCII forms control character, right-justified, as follows:

Character	ASCII (Octal)	Description
Δ (space)	240	Advance one line
+ (plus)	253	No line advance
0 (zero)	260	Advance two lines
1	261	Advance to top of form
2[a,b,c]	262	Advance according to channel 2
3[b]	263	Advance according to channel 3
4[b]	264	Advance according to channel 4
5[b]	265	Advance according to channel 5
6[b]	266	Advance according to channel 6
7[b]	267	Advance according to channel 7
8[b,c]	270	Advance according to channel 8
9[a,b,c]	271	Advance according to channel 9
A[a,b,c]	301	Advance according to channel 10
B[a,b,c]	302	Advance according to channel 11
C[a,b,c]	303	Advance according to channel 12
H	310	Advance to top of form (channel 1)

[a] For Line Printer Types 5565-9, these characters will result in a single line space.

[b] For Line Printer Types 5551-2, if the vertical format unit (VFU) option is not present, a single line space will occur. For Line Printer Types 5541-2, a single line space will occur.

[c] For Line Printer Types 552x, these characters will result in a single line space.

If none of the above is specified, an advance of one line is the default condition.

Word 4 (fifth word) of I/O Status Block

In addition to the status returned in word 1 of the I/O status block, the following hardware status is returned in word 4 of the I/O status block whenever bit 1 of word 1 is set.

Bit	Interpretation
1	Seek error
2	Data unsafe condition
3	Failure of CPU to maintain transfer rate
4	Format error
5	Head selection error
6	Record address comparison failure
7	Bus parity error
8	Write operation requested while write protect is in force
9	Data check word comparison failure
10	Time-out error
11	Wrong cylinder comparison failure
12-15	Reserved
16	Missed data synchronization pulse

Word 6 (seventh word) of I/O Status Block

The following hardware status is always returned in word 6 of the I/O status block.

Bit.	Interpretation
1	Operational
2	Busy
3	Active
4	Reserved
5	Unit 0 ready
6	Unit 1 ready
7	Unit 2 ready
8	Unit 3 ready
9-12	Reserved
13	Write track format end-of-range interrupt
14	Seek complete interrupt
15	Device going active interrupt
16	Busy being reset interrupt

Bit	Interpretation
1	Controller busy
2	Data ready for transfer
3	Requested address not found
4	Attempt to format over index mark
5	Heads not loaded on selected unit
6	Requested unit not available
7	Seek error (attempt to seek track outside — 0 to 202 limits)
8-10	Reserved
11	Write operation requested while in protect mode
12	Data unsafe (inconsistency in internal logic such as read and write at the same time or erase with detent; refer to the hardware manual for complete list of data unsafe conditions)
13	Checksum error
14	Data transfer rate failure
15	Logical OR of bits 3 through 7 and 11 through 14
16	End-of-record mark found

Status information returned for requests to a DMA disk.

Word 1 (second word) of I/O Status Block

Bit	Interpretation
1	Word 4 contains the hardware status word which indicates the error; word 6 also contains hardware status information.
2	Reserved
3	Missed interrupt
4	Unit not operational
5	Reserved
6	Missed data; transfer rate failure
7	Checksum error
8	Bus parity error
9	Reserved
10	Recovery error (miscellaneous)
11	Write protect error
12-15	Reserved
16	Controller busy

Word 3 (fourth word) of I/O Status Block

Word 3 contains the actual number of words transferred or received.

REMOVABLE DISK SUBSYSTEMS (TYPES 471x, 472x, 473x, 474x, 475x, AND 478x)

The only legal physical I/O requests for the removable disk subsystem are input (INP$) and output (OTP$).

For each input request, the range, which must be specified in the I/O parameter list, is used to determine the number of words to transfer to the user-supplied I/O buffer. For each output request to a DMC disk, the number of words transferred from the user-supplied I/O buffer is determined by the physical disk record length. The range specified for input and output requests to a DMA or DMC disk must be a positive nonzero number less than or equal to the physical disk record length. If the range is zero, negative, or exceeds the physical disk record length, the error return to the calling program is taken; an error code of 5 is returned in the A-register.

The following is a description of the status information returned to the user when control is transferred to the user's I/O completion return. The states described below are indicated if the appropriate bit is set.

Word 1 (second word) of I/O Status Block

Bit	Interpretation
1	Word 4 contains the hardware status word which indicates the error
2	Reserved
3	Missed interrupt
4-15	Reserved
16	Device busy

Word 3 (fourth word) of I/O Status Block

Word 3 contains the actual number of words transferred or received.

Word 4 (fifth word) of I/O Status Block

In addition to the software status returned in word 1 of the I/O status block, the following hardware status is returned in word 4 of the I/O status block whenever bit 1 of word 1 is set and also whenever word 1 is 0.

FIXED-HEAD DISK SUBSYSTEM (TYPE 451x)

The only legal physical I/O requests for the fixed-head disk subsystem are input (INP$) and output (OTP$).

For each input and output request, the range, which must be specified in the I/O parameter list, is used to determine the number of words to transfer to or from the user-supplied I/O buffer. The range must be a positive nonzero number less than or equal to the physical disk record length. If the range is zero, negative, or greater than the physical disk record length, the error return is taken to the calling program with the error code of 5 in the A-register.

The following is a description of the status information returned to the user when control is transferred to the user's I/O completion return. The states described below are true if the appropriate bit is set.

Word 1 (second word) of I/O Status Block

Bit	Interpretation
1	An error has occurred; the hardware status is in word 4
2-16	Reserved

Word 3 (fourth word) of I/O Status Block

Word 3 contains the actual number of words transferred or received.

Word 4 (fifth word) of the I/O Status Block

The hardware status is always returned in word 4 of the I/O status block.

Bit	Condition Tested
1	Operational
2	Busy
3	Active
4	Check byte error
5	Time-out error
6	Device not active error
7	Write protect error
8	Access error
9	Bus parity error
10	Device 0 active
11	Device 1 active
12	Device 2 active
13	Device 3 active
14	Reserved
15	Device going active interrupt
16	Busy reset interrupt

Word 3 (fourth word) of I/O Status Block

For input and output requests, word 3 contains the actual number of words transferred or received.

Word 4 (fifth word) of I/O Status Block

In addition to the status information returned in word 1 of the I/O status block, the following hardware status is always returned in word 4 of the I/O status block.

Bit	Interpretation
1	Operational
2	Busy
3	Active
4	Stop code
5	Read check error or range error
6	Validity error
7	Punch echo error
8	Punch cycle error
9	Mark sense (option)
10	40 column (option)
11	51 column (option)
12	External clock track (option)
13-14	Reserved
15	Device going active interrupt
16	Busy reset interrupt

Bit	Interpretation
1	Option operational
2	Option busy
3	Device active
4	Access error present
5	Registration error present
6	Invalid character present
7	Trap flop
8	Reserved
9	Cycle error bit 1
10	Cycle error bit 2
11	Cycle error bit 3
12	Cycle error bit 4
13	Cycle timing error
14	Busy reset interrupt
15	Device going active interrupt
16	Ready interrupt

The above described states are true if the appropriate bit is set.

Status Information for Card Devices (Types 5151-5153, 5161-5164, 5172, and 5176)

The following is a description of the status information returned to the user when control is returned to the user's I/O completion return. The states described below are indicated if the appropriate bit is set.

Word 1 (second word) of I/O Status Block

Bit	Interpretation
1	An error has occurred; the hardware status is in word 4.
2	Reserved
3	Missed interrupt
4	Device not operational
5	Device disabled
6	Reserved
7	Read or punch error
8	Reserved
9	Mode error (conversion not configured)
10-12	Reserved
13	End of file; not an error condition
14	Reserved
15	No free memory block available for conversion
16	Reserved

Bit	Condition Tested
1	Controller busy
2	Ready
3	End of card
4-11	Reserved
12	Punch check error
13	Read check error
14	Validity error
15	Data access error
16	Read end of file

Status Information for Card Reader Type 5100

The following is a description of the status information returned to the user when control is returned to the user's I/O completion return. The states described below are indicated if the appropriate bit is set.

Word 1 (second word) of I/O Status Block

Bit	Interpretation
1	An error has occurred; the hardware status is in word 4.
2	Reserved
3	Missed interrupt
4	Data not ready (not operational)
5	Device disabled (not operational)
6-9	Reserved
10	Recovery error
11-12	Reserved
13	End of file; not an error condition
14	Reserved
15	No free memory block available
16	Reserved

Word 3 (fourth word) of I/O Status Block

Word 3 contains the actual number of words received.

Word 4 (fifth word) of I/O Status Block

In addition to the status information returned in word 1 of the I/O status block, the following hardware status is always returned in word 4 of the I/O status block.

Word 1 (second word) of I/O Status Block

Bit	Interpretation
1	An error has occurred; the hardware status is in word 4
2-4	Reserved
6-9	Reserved
10	Recovery error; an attempt to retry has resulted in an error
11-12	Reserved
13	End of file detected; not an error condition
14-16	Reserved

The above described states are indicated if the appropriate bit is set.

Word 3 (fourth word) of I/O Status Block

For INP$ and OTP$ requests, word 3 contains the actual number of words transferred or received.

Word 4 (fifth word) of I/O Status Block for Input Requests for the Type 5121 Card Reader and Type 5140 Card Reader/Punch Devices

In addition to the status information returned in word 1 of the I/O status block, the following hardware status is always returned in word 4 of the I/O status block.

Bit	Interpretation
1	Busy indicator
2	Ready indicator
3	End-of-card indicator
4-12	Always zero
13	Cycle indicator
14	Validity indicator
15	Data access error indicator
16	End-of-file indicator

Word 4 (fifth word) of I/O Status Block for Output Requests for the Type 5140 Card Reader/Punch Device

In addition to the status information returned in word 1 of the I/O status block, the following hardware status is always returned in word 4 of the I/O status block.

CARD READER (TYPES 5100, 5121-5123, 5151-5153, 5161-5164)
CARD PUNCH (TYPE 5176)
CARD READER/PUNCH (TYPES 5140 AND 5172)

The legal physical I/O request for the card reader is input (INP$); legal physical I/O requests for the card punch are output (OTP$) and end of file (EOF$). Legal data modes for the card reader and card punch are 0 (ASCII), 1 (binary), and 2 (verbatim).

In the verbatim mode, bits 5 through 16 of the buffer word are replaced by a card-image character.

Card-Image Character (Row Number)					12	11	0	1	2	3	4	5	6	7	8	9
Buffer Word	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

When input from the card reader is requested, the data is read from one card and converted from 026 or 029 Hollerith card code to ASCII code, if the configurable conversion routines have been loaded in the system. The conversion routine 026 or 029 Hollerith is specified at system configuration time. The number of words stored in the user's input buffer will equal the range value, unless the range value exceeds the number of words read on a single card. In this case, only the number of words on the card are stored in the user's input buffer.

When output to the card punch is requested, the data in the user's output buffer is converted, if the configurable conversion routines have been loaded in the system. If the range of the words to be written is greater than the number of words that can be punched on a single card, only one card is punched, and the remaining words in the user's output buffer are ignored.

Status Information for Card Readers (Types 512x) and Card Reader/Punch (Type 5140)

The following is a description of the status information returned to the user when control is transferred to the user's I/O completion return.

IS ONE ASCII STORED/WORD RIGHT-JUSTIFIED

HIGH-SPEED PAPER TAPE PUNCH (TYPE 5210)

Legal physical I/O requests for the paper tape punch are output (OTP$) and end of file (EOF$); the legal data modes are listed in Appendix C. When output to the paper tape punch is requested, data in the user's output buffer is punched on the paper tape until the range count is exhausted. Punching of the end-of-file characters is initiated only by an EOF$ request.

The following is a description of the status information returned to the user when control is transferred to the user's I/O completion return:

Word 1 (second word) of I/O Status Block

Bit	Interpretation
1-2	Reserved
3	Missed interrupt
4	Data not ready
5	Device disabled
6-9	Reserved
10	Recovery error
11-14	Reserved
15	No free memory available
16	Reserved

The above described states are indicated if the appropriate bit is set.

Word 3 (fourth word) of I/O Status Block

For OTP$ requests, word 3 contains the actual number of words transferred.

HIGH-SPEED PAPER TAPE READER (TYPE 5010)

The only legal physical I/O request for the paper tape reader is input (INP$). Legal data modes for the paper tape reader are those listed in Appendix C. When input from the paper tape reader is requested, data on the paper tape is read until an X-OFF character is reached, and then stored in the user's input buffer. The word count in word 3 (fourth word) of the I/O status block indicates the number of words stored in the user's input buffer.

The following is a description of the status information returned to the user when control is transferred to the user's I/O completion return:

Word 1 (second word) of I/O Status Block

Bit	Interpretation
1-2	Reserved
3	Missed interrupt
4	Data not ready
5	Device disabled
6	Reserved
7	Checksum error
8	Parity error
9	Format error
10	Recovery error
11-12	Reserved
13	End of file detected; not an error condition
14	Range error
15	No free memory available
16	Reserved

The above described states are indicated if the appropriate bit is set.

Word 3 (fourth word) of I/O Status Block

Word 3 contains the actual number of words received.

Word 6 (seventh word) of I/O Status Block

The following hardware status is always returned in word 6 of the I/O status block.

Bit	Interpretation
1	Operational
2	Busy
3	Ready
4	Busy seek initiation
5	Unit 0 ready
6	Unit 1 ready
7	Unit 2 ready
8	Unit 3 ready
9	Unit 0 interrupt
10	Unit 1 interrupt
11	Unit 2 interrupt
12	Unit 3 interrupt
13	Reserved
14-15	Physical unit number of selected device
16	Busy being reset interrupt

Bit	Interpretation
1	Word 4 contains the hardware status word which indicates the error. Word 6 also contains hardware status information.
2	Reserved
3	Missed interrupt
4	Not operational
5	Reserved
6	Transfer rate failure
7	Checksum error
8	Parity error
9	Reserved
10	Recovery error (Miscellaneous)
11	Volume protected on OTP$ request
12-14	Reserved
15	No free memory available
16	Controller busy

Word 3 (fourth word) of I/O Status Block

Word 3 contains the number of words transferred or received. This word will always contain the range.

Word 4 (fifth word) of I/O Status Block

In addition to the status returned in word 1 of the I/O status block, the following hardware status is returned in word 4 of the I/O status block whenever bit 1 of word 1 is set.

Bit	Interpretation
1	Reserved
2	Write timing error
3	Failure of CPU to maintain transfer rate
4	Format error
5	Sector pulse time-out
6	Record address comparison failure
7	DMA has parity error
8	Write inhibit error
9	Data check word comparison failure
10	Time-out error
11-12	Reserved
13	Read timing
14	Seek error
15	Reserved
16	Missed data synchronization pulse

CARTRIDGE DISK SUBSYSTEM (TYPE 476x)

The only legal physical I/O requests for the cartridge disk subsystem are input (INP$) and output (OTP$).

For each input and output request, the range, which must be specified in the I/O parameter list, must be a positive nonzero number which is less than or equal to the physical disk record length. If the range is zero, negative, or greater than the physical disk record length, the error return is taken to the calling program with the error code of 5 in the A-register.

The physical disk record length is used for all I/O except for segment 0. Therefore, if an input request is issued with a range which is less than the physical disk record length, a free memory block is used as the input buffer. The range is then used to determine how many words are to be transferred from the free memory block to the user-supplied buffer. The use of the free memory block means that the system free memory requirements are increased by one segment size block for sequential I/O.

The range is always used for I/O to segment 0; however, since the label resides on segment 0 of labeled volumes, the user should not write on segment 0 unless the volume is unlabeled.

The following is a description of the status information returned to the user when control is transferred to the user's I/O completion return. The states described below are indicated if the appropriate bit is set.

Word 1 (second word) of I/O Status Block

The following error status is returned in word 1 of the I/O status block. If the data transfer was successful, this word will be 0.

All other output carriage control is the user's responsibility and must be included in his buffer.

The following is a description of the status information returned to the user when control is transferred to the user's I/O completion return:

Word 1 (second word) of I/O Status Block

Bit	Interpretation
1-3	Reserved
4	Data not ready
5	Missed interrupt
6-8	Reserved
9	Wrong mode
10	Control-K received during output
11-13	Reserved
14	Range error
15-16	Reserved

The above described states are indicated if the appropriate bit is set.

Word 3 (fourth word) of I/O Status Block

Word 3 contains the actual number of words transferred or received.

APPENDIX B
PHYSICAL I/O DEVICE INFORMATION

TELEPRINTER (TYPE 5310 KSR-33)

Legal physical I/O requests for the KSR-33 teleprinter are input (INP$) and output (OTP$); the only legal data mode for it is 0 (ASCII).

When input is requested from the KSR, the characters are read from the keyboard, packed two characters per word, and stored in the user's buffer. Characters are read until the user's buffer is full, or until a carriage return character read from the keyboard is stored in the user's buffer. The following control characters are checked on input and the action described is taken:

@ - Ignore last input line and start reading a new line.

← - Ignore last character typed in.

Control-K - Terminate output. If control-K is struck when an OTP$ (output) request is in progress, output is terminated and a bit is set in the status word. If an INP$ (input) request is in progress, the control-K is treated as an ordinary character. If no input or output is in progress, the control-K is ignored.

Control-P - Schedule attention task. If control-P is struck when no input or output is in progress, or when an OTP$ (output) request is in progress, the control-P TCB supplied when the device was reserved is scheduled. If the control-P TCB is already scheduled, or if no control-P TCB was specified, the character is ignored. If an INP$ (input) request is in progress, the control-P is treated as an ordinary character.

Control-Shift-M - Terminate input request and return end-of-file status word.

When output to the teleprinter is requested, all data specified in the buffer must be packed two characters per word, and is transmitted to the printer until the range count is exhausted. Before physical output is started, the first word of the buffer is examined for line spacing control. If required, the necessary carriage control characters are set up. The second byte of the first word of the caller's buffer is treated as follows:

Blank - Advance one line (insert CR, LF)

0 - Advance two lines (insert CR, LF, LF)

+ - Do not advance (no insert)

Other - Ignore character and advance one line (insert CR, LF)

Table A-1 (cont). Executive Function Call Error Codes

Error Codes (A-register)	Indication(s)
175	No free memory blocks configured of sufficient size to handle segments on a non-system disk (CRL$ or CLL$).
176	There is no suitable volume available to satisfy the request to connect a nonremovable volume (CVL$).

Table A-1 (cont). Executive Function Call Error Codes

Error Codes (A-register)	Indication(s)
151	Disconnect volume failed because the volume specified is not mounted on the unit specified (DVL$).
152	Could not obtain control segment block for first direct access GET$ request.
153	Segment number specified by the OTP$ request is illegal.
154	Device being released is still processing I/O request (REL$).
155	Illegal reserve request (RSV$).
156	Mount is in process on requested unit (ALC$, DLC$, INP$, or OTP$).
157	TPR$ request on behalf of a restricted activity was aborted because the activity is being aborted. This error code is passed to an action routine, such as CVL$, which is making the TPR$ request on behalf of the activity.
160	Cannot change library passwords on library not opened in master mode (CLP$).
161	Configuration not correct for direct access (GET$).
162	Number of buffers to be used for multiple buffering is not 2 or 3 (OPN$).
163	No work area available in user area (ALC$).
164	GSP$ parameter set number invalid.
165	GSP$ called with invalid password in an attempt to get disk volume parameters.
166	The library name or file name to be added to the directory is in an illegal format. The name must be an alphanumeric string, starting with an alphabetic (CRL$ or OPN$).
167	This error code is passed from the abort cleanup routine to the terminate activity action routine when there are no 16-word blocks available to terminate a restricted activity.
170	A restricted activity attempted to schedule a non-restricted activity (SAC$).
171	Attempt to schedule a restricted activity failed because free memory is low (SAC$).
172	Attempt to schedule a restricted activity which is being aborted (SAC$).
173	Attempt to delete an activity which is running or requested (Delete Activity utility (DA)).
174	Attempt to terminate an activity which is not running (TMA$).

Table A-1 (cont). Executive Function Call Error Codes

Error Codes (A-register)	Indication(s)
123	Starting segment number for deallocate is not accessible to the user (DLC$).
124	Starting segment number for deallocate is beyond the range of the bit map (DLC$ or CLS$).
125	Work area to be deallocated had not been allocated (DLC$ or CLS$).
x126	Physical I/O error during input of allocation bit map segment (ALC$, DLC$, CLS$, or PUT$).
x127	Physical I/O error during output of allocation bit map segment (ALC$, DLC$, CLS$, or PUT$).
130	Special action error return on close; data in the block buffer was not written successfully (CLS$).
131	Special action error return on close; the control segment was not updated successfully (CLS$).
132	Special action error return on close; the file descriptor was not successfully updated (CLS$).
133	Request to open library that is already open (OPL$).
134	Request to close library with open files in it (CLL$).
135	Generic device type is not configured or has no associated volume descriptor (CVL$, DVL$, ALC$, or DLC$).
136	Logical unit number is not configured or has no associated volume descriptor (CVL$, DVL$, ALC$, or DLC$).
137	Incorrect specification of labeled/unlabeled volume in the volume control block (CVL$).
140	Incorrect specification of public/private volume in the volume control block (CVL$).
141	Segment size specified in parameter 1 of the volume control block is incorrect (CVL$).
142	Number of segments per track specified in parameter 2 of the volume control block is incorrect (CVL$).
143	Incorrect surface code in the volume control block (CVL$).
144	Segment size specified in the volume control block is not a power of two between 64 and 512 (CVL$).
145	Operator did not supply volume name or mount volume (CVL$).
146	Requested volume is in use on another unit (CVL$).
147	No unit is available for mounting requested volume (CVL$).
x150	Physical I/O error on input by volume manager (CVL$).

Table A-1 (cont). Executive Function Ca'l Error Codes

Error Codes (A-register)	Indication(s)
77	Library not found in the system library directory (CLL$ or CLP$).
100	Add library failed because information in library control block buffer and volume descriptor are inconsistent (CRL$).
101	Delete library failed because library name does not agree with directories (CLL$).
102	Delete library failed because another user has library open or is opening it (CLL$).
103	Delete library failed because library is not empty (first chunk is not empty or it is not the only chunk) (CLL$).
104	Library already deleted from system library directory. It was removed from the volume library directory at this time (CLL$).
105	Improper library master password (CLP$).
106	Change library password function out of range (CLP$).
107	Restricted activity specified a secondary TCB when scheduling another activity.
110	The delete or update file directory entry failed because the file name specified does not agree with the file name in the disk directory entry when the relative position of the entry is specified (CLS$).
111	The delete or update file directory entry failed because the file name specified is not in the disk directory (CLS$).
112	Attempt to open library with invalid password (OPL$).
113	Illegal mode on open or close library request (OPL$ and CLL$).
114	Library to be opened not in identified libraries' queue (OPL$).
115	The abort request failed because the activity to be aborted is not a restricted activity or was not requested (ABT$).
116	Cannot close and delete a library that is not opened in master mode. Normal close is attempted (CLL$).
117	Allocation management is prohibited on unlabeled volumes (ALC$ or DLC$).
120	User I/O is prohibited on this volume (ALC$ or DLC$).
121	Bit map segment contains invalid ID (ALC$, DLC$, OPN$, CLS$, or PUT$).
122	Starting segment number for deallocate is not a work area boundary (DLC$ or CLS$).

Table A-1 (cont). Executive Function Call Error Codes

Error Codes (A-register)	Indication(s)
54	Either file cannot be opened in update mode, because it is being read or updated by another user (OPN$), or requested file cannot be opened in input mode, because it is being updated by another user (OPN$).
55	Delete file request is illegal (CLS$).
x056	The last data segment could not be written correctly; the file was saved, but the record(s) in the last data segment contain(s) extraneous information (CLS$).
x057	The control segment could not be updated properly; the file was saved, but records were lost. Only those records that were in the file when the control segment was last updated properly were saved (CLS$).
x060	The request to save the file could not be accomplished; the file was deleted (CLS$).
x061	The file could not be updated; it remains as it was prior to the open request (CLS$).
x062	The data segment containing the record for the last PUT$ request could not be written properly (CLS$).
x063	Physical I/O error on system library directory segment input (CRL$, OPL$, CLL$, CLP$).
x064	Physical I/O error on system library directory segment output (CRL$, OPL$, CLL$, CLP$).
x065	Physical I/O error on volume library directory segment input (CRL$, CLL$, or CLS$).
x066	Physical I/O error on volume library directory segment output (CRL$, CLL$, or CLS$).
x067	Physical I/O error on file directory segment input (CRL$, CLL$, OPN$, CLS$, or SAC$).
x070	Physical I/O error on file directory segment output (CRL$, CLL$, OPN$, or CLS$).
x071	Physical I/O error on file descriptor directory segment input (OPN$, CLS$, or SAC$).
x072	Physical I/O error on file descriptor directory segment output (OPN$, or CLS$).
73	No more library space in the system library directory (CRL$).
74	No available chunk in the volume library directory or in the file directory (CRL$ or OPN$).
75	Library already exists in the system library directory (CRL$).
76	Library already exists in the volume library directory (CRL$).

Table A-1 (cont). Executive Function Call Error Codes

Error Codes (A-register)	Indication(s)
35	The requested executive function cannot be loaded from disk.
36	● A GET$ or PUT$ request was made for a file that has variable length records, but direct access was specified in the FCB. ● A GET$ or PUT$ request was issued for a file that has variable length records, but the record length address in the FCB is zero or the FCB is short (4 words) and does not contain the record length address. ● A PUT$ request was issued for a file that has variable length records, but the record length specified by the record length address in the FCB is zero or minus one (the legal record length is 1 to 65,534).
37	The file type parameter in the user's file control block (FCB) indicates that the direct access method is desired; however, the record number address in the FCB is zero, or the record number itself is zero or negative. The legal range for record numbers is from 1 to 32,767 (GET$ and PUT$).
x040	Physical I/O error occurred when PUT$ request was being processed.
x041	Physical I/O error occurred when OPN$ request in update mode, CLS$ request in delete mode, GET$ request or PUT$ request was being processed.
42	The record is invalid (GET$).
43	The maximum number of records that may exist in a file (32,767) was exceeded (PUT$).
44	Maximum number of work areas that may be allocated for a file was exceeded (PUT$).
45	The CFP$ request is illegal, because the I/O mode parameter specified when the file was opened was not update.
46	The nondefault library is not open (CLP$, CLL$, OPN$, CLS$, CFP$, GET$, or PUT$).
47	A restricted activity issued a ULD$ request with one or more I/O requests queued but not processed.
50	The OPN$ request is illegal, because the password check failed. A file password exists in the file descriptor entry, but not in the user's FCB; or the file password in the user's FCB does not match the password in the file descriptor entry.
51	The number of work areas in the volume descriptor is less than two (OPN$).
52	Maximum number of work areas that can be allocated for a file is specified as 0 in the volume descriptor (OPN$).
53	Segment size specified in the volume descriptor is illegal (OPN$).

Table A-1 (cont). Executive Function Call Error Codes

Error Codes (A-register)	Indication(s)
17	The file or activity name already exists in the disk directory (OPN$ or LA utility program).
20	No additional space is available, nor is there room for expansion to add a file or activity name to the disk directory (OPN$ or LA utility program).
21	No work areas are available (ALC$, OPN$ to create a file, or PUT$).
22	The request is illegal because the pointers to blocks (LCBB and LNB) used by the library and file managers are invalid for the specified library (CLL$, CLP$, CLS$, CFP$, GET$, or PUT$).
23	GET$, PUT$, CLS$, or CFP$ request issued for an unopened file, or the request is illegal for the mode specified when the file was opened.
24	● OPN$ request issued for a null file. ● The maximum record length address was not specified in the OPN$ parameter list or in the file control block (FCB). ● The maximum record length is 0 or negative for an OPN$ request for a file that is to be created.
25	● The record buffer address was not specified in the FCB for a PUT$ request. ● The record buffer address was not specified in the FCB for a GET$ request when the record length is greater than or equal to the segment length for fixed-length records, or the maximum record length +1 equals or exceeds the segment length for variable-length records.
26	There is no activity area configured for the requested activity, or the activity area(s) configured is too small for the requested activity (SAC$).
27	GET$ or PUT$ request issued for a file opened in update mode, but the maximum record length address was not specified when the file was opened.
30	An RSV$ request was issued for a device, the driver for which is disk resident; however, the driver could not be brought into memory, because its size exceeded the size of the activity area.
31	An RSV$ request was issued for a device, the driver for which is disk resident; however, the driver could not be brought into memory, because there was a disk read error.
32	An INP$, OTP$, SPF$, SPR$, EOF$, RWD$, or ULD$ request was issued for a device, the driver for which is disk resident, but the device was not reserved.
33	The requested activity could not be scheduled immediately (SAC$).
34	The requested executive function is not configured.

Table A-1 (cont). Executive Function Call Error Codes

Error Codes (A-register)	Indication(s)
1	● The generic device type (GDT) specified in the device control block (DCB) is not configured. ● Block size parameter specified in GBL$ parameter list is illegal. ● CCA$ or CCL$ request to connect an unconfigured absolute timer has been made.
2	Attempt to enable a device already enabled.
3	● The logical unit number specified in the device control block is not configured. ● An issued TPR$ request cannot be acknowledged because the message table is full.
4	● The mode indicator specified in the device control block is incorrect, i.e., <0 or >4, or is an invalid mode for the specified device when an INP$ or OTP$ request is issued. (Binary mode is specified for an ASCII only device, ASR-35 or KSR teleprinters, or line printer; verbatim mode only was configured for the card reader, and the mode requested is ASCII or binary.) ● TYP$ or TPR$ request issued, but error was encountered before or during message output.
5	The range value specified in the INP$ or OTP$ parameter list is incorrect; i.e., <1 or >4095, or the I/O buffer crosses the 32K boundary between banks 1 and 2 (64K system only).
6	The humber of records or files specified in the SPR$ or SPF$ parameter list for magnetic tape is zero. For cassette tape, the record or file number is illegal when negative or zero.
7	The function requested (SPF$, SPR$, EOF$, RWD$, or ULD$) for the generic device type specified in the device control block is illegal.
10	The user ID specified in the device control block is incorrect for the REL$, INP$, OTP$, SPF$, SPR$, EOF$, RWD$, or ULD$ request.
11	An INP$ request was issued for an output only device; i.e., paper tape punch, card punch, or line printer.
12	An OTP$ request was issued for an input only device; i.e., paper tape reader or card reader.
13	No free memory block available for an entry in usage request queue (OPN$).
14	RSV$, INP$, OTP$, SPF$, SPR$, EOF$, RWD$, or ULD$ request issued for a disabled device.
15	A restricted activity issued a REL$, INP$, OTP$, SPF$, SPR$, EOF$, RWD$, or ULD$ request for a device which was not reserved by the activity.
16	The file or activity name was not found in the disk directory (OPN$ or SAC$).

APPENDIX A

EXECUTIVE FUNCTION CALL ERROR CODES

When error returns are taken by executive function action routines, the A-register and possibly the X-register contain information describing the error. If the A-register contains a value less than '777, the X-register value should be ignored. However, if the A-register value exceeds '777, the X-register contains physical I/O status information; it is decoded in the following manner:

If A-register contents are:	Then the X-register contains:
'1nnn	Setup error from the I/O input or output request; see the error codes below for INP$ and OTP$.
'2nnn	Software status from word 1 of the I/O status block; refer to Appendix B for status information.
'3nnn	Hardware status from word 4 of the status block; refer to Appendix B for status information.
NOTE: nnn - Octal digits representing the real error code.	

Table A-1 indicates executive function call error codes, excluding those for communications.

Table A-1. Executive Function Call Error Codes

Error Codes (A-register)	Indication(s)
0	• A device is reserved under another user ID when an RSV$ request is made. • A nonsharable device has already been reserved under the same user ID when another RSV$ request is made. • No free memory blocks were available when a GBL$ request was made. • Block size parameter specified in the RBL$ parameter list is illegal.

LNK$

LINK (LNK$)

The LNK$ macro call provides the proper user/system interface between function calls that request executive functions and the System Function Manager. A LNK$ macro call must be present in each assembled segment of code that contains references to the pseudoregisters and/or requests for executive functions.

Macro Expansion

The LNK$ macro call generates the subroutine ZALINK, which saves the address of the executable function call, saves the Bank State Register (in a 64K system), enables extended addressing, and transfers control to the Function Manager for processing of the executable function call. The macro expansion also defines the addresses of pseudoregisters ZCR1 through ZCR6. LNK$ must reside in the user's date.

Macro Call Format

Location	Operation	Operand
	LNK$	

Normal Return

No normal return is associated with the LNK$ macro call, since ZALINK is not exeucted until a system function macro is called.

Error Return

No error return is associated with the LNK$ macro call, since ZALINK is not executed until a system function macro is called.

Outline Parameter List

An outline parameter list is not applicable to the LNK$ macro call.

A-Register Contents (Octal)	Error Condition
34	Requested execution function is not configured.
35	Requested execution function is disk-resident and there was a disk error during an attempt to load the function into main memory.

Action Routine Details

The ASCII date-time is obtained from the system, the date is rearranged into the user format and stored in the user buffer. Return is made to the normal return.

Outline Parameter List

Word Number	Operation	Operand
0	DAC	date-time block address
1	DAC	error return address

Parameters in the outline parameter list are the same as those described above for the inline parameter list.

Example:

In the example below, the buffer TBUF is used to receive the date-time characters upon execution of the GDT$ action routine.

```
*                                GET THE DATE AND TIME
*
        GDT$    TBUF,ERR
*
*                                THIS BUFFER WILL RECEIVE THE DATE-TIME CHARACTERS
*
TBUF    BSZ     6                6 WORDS FOR DATE-TIME
ERR     ---                      ERROR RETURN
```

GET DATE-TIME (GDT$)

The GDT$ function is used to transfer the system ASCII date and time to a user-specified buffer.

Function Action

The GDT$ action routine rearranges the ASCII date-time into user format and stores it in the user's buffer. It then returns control to the calling program at the normal return.

Macro Call Format

Location	Operation	Operand
[symbol]	GDT$	date-time block address, error return address

symbol - Optional. The symbolic location of the GDT$ macro instruction.

date-time block address - The address of the first word of a six-word block into which the action routine will place the date-time.

error return address - The address to which control is returned if an error is found during the processing of the GDT$ function call.

Normal Return

The ASCII date-time has been placed into the user-supplied buffer in the following format (two characters per word):

Word 0 - Hours (00 through 23)

Word 1 - Minutes (00 through 59)

Word 2 - Two space characters

Word 3 - Day (01 through 31)

Word 4 - Month (01 through 12)

Word 5 - Year (xx through 99) (xx is the year in which system configuration was performed.)

Error Return

Control is returned to the error return address specified in the GDT$ parameter list with the error code in the A-register when any of the following errors is detected:

Using an outline parameter list, the above example would be written as follows:

```
        LDX     PLIST          SET UP POINTER TO OUTLINE LIST
        SDT$    (X)            SET SYSTEM DATE-TIME
         :
PLIST   DAC     *+1            POINTER TO OUTLINE LIST
        DAC     DATE           POINTER TO USER DATE-TIME BLOCK
        DAC     ERR1           ERROR RETURN ADDRESS
         :
DATE    BCI     1,12           HOUR
        BCI     1,30           MINUTES
        BCI     1,             REQUIRED SPACES
        BCI     1,19           DAY
        BCI     1,12           MONTH
        BCI     1,73           YEAR
         :
ERR1    ----                   ERROR RETURN
```

A-Register Contents (Octal)	Error Condition
34	Requested executive function is not configured.
35	Requested executive function is disk-resident and there was a disk error during an attempt to load the function into main memory.
Not Significant	One of the items in the date-time block is out of its specified range. (There is no specific error code associated with this error.)

Action Routine Details

The system date-time is set to the value specified by the user in the date-time block, and system date-time updating continues. Return is made to the normal return.

If any of the user-specified items in the date-time block is out of its stated range, the error return is taken.

Outline Parameter List

Word Number	Operation	Operand
0	DAC	date-time block address
1	DAC	error return address

Parameters in the outline parameter list are the same as those described above for the inline parameter list.

Examples:

In the following example, the system date-time is set to the value specified in the user block, DATE.

```
        SDT$     DATE,ERR1     SET SYSTEM DATE-TIME
          .
          .
          .
DATE    BCI      1,12          HOUR
        BCI      1,30          MINUTES
        BCI      1,            REQUIRED SPACES
        BCI      1,19          DAY
        BCI      1,12          MONTH
        BCI      1,73          YEAR
          .
          .
          .
ERR1    ----                   ERROR RETURN
```

SET-DATE-TIME (SDT$)

The SDT$ function is used to set the system date-time to a user-specified value.

Function Action

The SDT$ action routine takes the user-specified date-time value from the user's buffer and rearranges that value into internal format for system use. The macro routine then returns control at the normal return.

A restricted activity cannot call the SDT$ system function. If if attempts to do so, the activity will be aborted.

Macro Call Format

Location	Operation	Operand
[symbol]	SDT$	date-time block address, error return address

symbol - Optional. The symbolic location of the SDT$ macro instruction.

date-time block address - The address of the first word of a 6-word block from which the action routine will obtain the user-specified date-time value.

error return address - The address to which control will be passed if an error is found during the processing of the SDT$ function call.

Normal Return

The system date-time has been set to the user-specified date-time value; system date-time updating continues. The user must specify the date-time value in the following format (two ASCII characters per word):

Word 0 - Hours (00 through 23)

Word 1 - Minutes (00 through 99)

Word 2 - Two space characters

Word 3 - Day (01 through 31)

Word 4 - Month (01 through 12)

Word 5 - Year (xx through 99) (xx is the year in which system configuration was performed.)

Error Return

Control is returned to the error return address specified in the SDT$ parameter list, with the error code in the A-register when any of the following errors is detected:

Using an outline parameter list, the following example obtains from parameter set 5 details on the user area.

```
       LDX    PLIST        SET UP POINTER TO OUTLINE LIST
       GSP$   (X)          FETCH PARAMETER SET 5
        .
        .
        .
PLIST  DAC    *+1          POINTER TO OUTLINE LIST
       DAC    ABLOK2       PARAMETER BLOCK ADDRESS POINTER
       DEC    5            PARAMETER SET 5:  OPERATOR'S CONSOLE SPECIFICS
       DAC    ERR2         ERROR RETURN ADDRESS
*
ABLOK2 DAC    *+1          PARAMETER BLOCK ADDRESS
       BSZ    1            WILL BE OPERATOR'S CONSOLE GENERIC DEVICE TYPE
*
ERR2   ---                 ERROR RETURN
```

Error Return

Control is returned to the error return address specified in the GSP$ parameter list, with an error code in the A-register when any of the following errors is detected:

A-Register Contents (Octal)	Error Condition
34	The requested function is not configured.
35	Requested system function is disk-resident and there was a disk error during an attempt to load the function into main memory.
164	Parameter set type number is out of the range 1 through 5.

Action Routine Details

The parameter associated with the parameter set type number are placed in the block pointed by the parameter block address pointer.

Outline Parameter List

Word Number	Operation	Operand
0	DAC	parameter block address pointer
1	DEC	parameter set type number
2	DAC	error return address

Parameters in the outline parameter list are the same as those described for the inline parameter list.

Examples:

The following example obtains from parameter set 1 details on the system disk.

```
        GSP$    ABLOK1,1,ERR1   FETCH SYSTEM PARAMETERS
         .
         .
         .
ABLOK1  DAC     *+1            PARAMETER BLOCK ADDRESS
        BSZ     1              WILL BE SYSTEM DISK GDT
        BSZ     1              WILL BE SYSTEM DISK UNIT NUMBER
        BSZ     1              WILL BE SYSTEM DISK SEGMENT SIZE (IN WORDS)
        BSZ     1              WILL BE NUMBER OF SEGMENTS PER WORK AREA
*
ERR1    ---                    ERROR RETURN
```

GET SYSTEM PARAMETERS (GSP$)

The GSP$ function is used to obtain certain system parameters associated with the system disk and the operator's console.

Function Action

The GSP$ action routine places a predefined set of system parameters in the user-supplied block.

Macro Call Format

Location	Operation	Operand
[symbol]	GSP$	parameter block address pointer, parameter set type, error return address

symbol - Optional. The symbolic location of the GSP$ macro instruction.

parameter block address pointer - The address of a word containing the address of a word or of a 4-word block where the system parameters defined by the parameter set type are to be stored.

parameter set type - An integer (1 or 5) specifying the particular set of system parameters defined below.

Parameter Set Type	Block Size	Word Number	Contents of Word
1	4	0	system disk generic device type
		1	highest system disk unit number configured
		2	segment size on disk
		3	number of segments per work area
5	1	0	operator console generic device type

NOTE: Parameter set types 2, 3, and 4 appeared in an earlier release of OS/700 but are no longer necessary to the functioning of the system and are not supported in the current release.

error return address - The address to which control is passed if an error is found during the processing of the GSP$ macro call.

Normal Return

Control is returned to the calling program at the instruction following the GSP$ function call.

SECTION VIII
SPECIAL SYSTEM MACRO FUNCTIONS

In this section, four special functions are described in detail. The Get System Parameters function (GSP$), is used to obtain parameters that were specified at system configuration time. The two date/time functions, Set Date/Time (SDT$) and Get Date/Time (GDT$), enable the system, and programs operating under the system, to keep track of the date and the time of day. The Link function (LNK$), as already mentioned, is required when system functions are performed or the pseudoregisters are referenced by a user program.

```
IBUF  BSZ     32
*
ERTYP ---                           ERROR RETURN
```

OUTLINE PARAMETER LIST

Word Number	Operation	Operand
0	DAC	output buffer address
1	DEC	output range
2	DAC	error return address
3	DAC	input buffer address
4	DEC	input range

Parameters in the outline parameter list are the same as those described above for the inline parameter list.

Examples:

A 16-word message is to be printed on the operator console, and the operator must type in a response of not more than 63 characters plus a carriage return. The output message is contained in OBUF, and the operator response will be collected in IBUF.

```
      TPR$   OBUF,16,ERTYP,1BUF,32
       :

       :
OBUF  BSZ    7                RESERVED FOR TPR$ ACTION
      BCI    9,               MESSAGE
*
IBUF  BSZ    32
*
ERTYP ---                     ERROR RETURN
```

Using an outline parameter list, the above example would be written as follows:

```
      LDX    PLIST
      TPR$   (X)
       :

       :
PLIST DAC    *+1
      DAC    OBUF
      DEC    16
      DAC    ERTYP
      DAC    IBUF
      DEC    32
*
OBUF  BSZ    7
      BCI    9,               MESSAGE
*
```

ERROR RETURN

Control is returned to the error return address specified in the TPR$ parameter list with the error code in the A-register when any of the following errors are detected:

A-Register Contents (Octal)	Error Condition
3	The message table (containing the identity of users waiting for response to their message) is full and no more TPR$ requests can be acknowledged at this time.
4	An error was encountered before or during the message output, or the operator typed control-K during message output.
34	Requested executive function is not configured.
35	Requested executive function is disk-resident and there was a disk error during an attempt to load the function into main memory.

ACTION ROUTINE DETAILS

A test is made to see if the message table is full; if so, control is transferred to the error return address with 3 in the A-register. If there are no errors, a message number and the activity name are inserted in front of the message and a request made on the I/O scheduler to output the requested message. An entry is made in the message table to indicate an activity waiting for a response.

Upon the I/O completion return, a check is made to determine if the message output was interrupted by the operator typing control-K or if there were any errors and if so, control is transferred to the error return address with 4 in the A-register. If there are no errors, after a response is keyed in by the operator, a check is made for a valid message number and control is transferred to the normal return of the associated activity. If an invalid message number or an illegal number of characters is keyed by the operator, a message is typed on the operator console indicating the error and allowing the operator to key in the proper response.

Unlike a normal Input (INP$) request, a TPR$ request from a restricted activity being aborted does not have to wait for a response before the abort operation can proceed. If abort of a restricted activity is initiated between the time the activity makes a TPR$ request and the time the operator response is received, the output to the operator's console is allowed to continue, but when the output has terminated, or if it has already terminated, OIP removes the outstanding message request from the message table and the abort proceeds.

TPR$

Type a Message and Input a Response (TPR$)

The TPR$ function is used to type a message on the operator console and request a response from the operator.

FUNCTION ACTION

The TPR$ action routine types on the operator console the message specified in the parameter list, and waits for the operator to input the response. The response is passed to the user program in a buffer. The carriage return terminating the response is also placed in the user's buffer.

MACRO CALL FORMAT

Location	Operation	Operand
[symbol]	TPR$	output buffer address, output range, error return address, input buffer address, input range

symbol - Optional. The symbolic location of the TPR$ macro instruction.

output buffer address - The address of the buffer containing the message to be typed. The first seven words of the buffer must not be used for storing the message. These words must be left for system use and the message to be printed must start on the eighth word. The message can contain ASCII characters only.

output range - The number of words contained in the message. The output range must include the seven words reserved in the output buffer for system use.

error return address - The address to which control is returned if an error is found during the processing of the TPR$ function call.

input buffer address - The address of the first word of the buffer in which the response is to be stored.

input range - The number of words expected in the response message including the terminating carriage return character. The number must not be greater than 36.

NORMAL RETURN

Control is returned to the calling program, at the instruction following the TPR$ request, after the message has been output and response has been input, provided no errors were detected during the processing of the TPR$ function call.

If an outline parameter list were used, the following would be written:

```
      LDX    PLIST
      TYP$   (X)
       :
       :
PLIST DAC    *+1
      DAC    OBUF                BUFFER ADDRESS
      DEC    12                  RANGE
      DAC    ERTYP               ERROR RETURN ADDRESS
*
OBUF  BSZ    7                   RESERVED FOR SYSTEM USE
      BCI    5,XXXXXXXXXX        MESSAGE
*
ERTYP ----                       ERROR RETURN
```

ACTION ROUTINE DETAILS

The name of the activity making the request is inserted in front of the message and a request is made on the I/O scheduler to output the specified number of words on the operator console.

After I/O completion, the I/O status block is checked to see if the operator interrupted message output by typing control-K or if there were any errors and if there are any, control is transferred to the error return address with a 4 in the A-register. If no errors are indicated in the I/O status block, control is transferred to the normal return (the instruction following the TYP$ function call).

OUTLINE PARAMETER LIST

Word Number	Operation	Operand
0	DAC	buffer address
1	DEC	range
2	DAC	error return address

Parameters in the outline parameter list are the same as those described above for the inline parameter list.

Examples:

It is desired to print on the operator console a message consisting of five words. The seven words reserved for the TYP$ macro action make a total range of 12 words. The 12 words begin at word OBUF.

```
      TYP$   OBUF,12,ERTYP
        :
        :
OBUF  BSZ    7            7 WORDS FOR TYP$ ACTION
      BCI    5,XXXXXXXXXX MESSAGE
*
ERTYP ---                 ERROR RETURN
```

Type a Message (TYP$)

The TYP$ function is used to type a message on the operator console.

FUNCTION ACTION

The TYP$ action routine types the message on the operator console, and returns to the calling program after the message is completely typed.

MACRO CALL FORMAT

Location	Operation	Operand
[symbol]	TYP$	buffer address, range, error retrun address

symbol - Optional. The symbolic location of the TYP$ macro instruction.

buffer address - The address of the first word of the buffer containing the message to be typed. The first seven words of the buffer must not be used for storing the message. These words must be left for system use and the message to be printed should start on the eighth word. The message must consist of ASCII characters only.

range - The number of words in the buffer. This range must include the seven words reserved in the buffer for system use.

error return address - The address to which control is returned if an error is found during the processing of the TYP$ function call.

NORMAL RETURN

Control is returned to the calling program, at the instruction following the TYP$ request, after the message is output, provided no errors were detected during the processing of the TYP$ action routine.

ERROR RETURN

Control is returned to the error return address specified in the TYP$ parameter list with the error code in the A-register when any of the following errors are detected:

A-Register Contents (Octal)	Error Condition
4	An error was encountered before or during the message output, the operator typed control-K during message output.
34	Requested system function is not configured.
35	Requested system function is disk-resident and there was a disk error during an attempt to load the function into main memory.

When responding to a TPR$ request, the operator precedes his typed input with a control-P character and the 2-digit message number. The operator can terminate tye typing of a message output from a TYP$ or TPR$ call by typing a control-K character. The calling activity then resumes at the TYP$ or TPR$ call error return with the appropriate error code and can take any desired action in response to the operator's interruption. Typing control-P during output on the console does not effect the output, but allows the operator to type in a message when the output is complete.

SECTION VII
OPERATOR INTERFACE FUNCTIONS

This section describes the executive function used to interface activities and tasks with the operator.

OPERATOR INTERFACE FUNCTIONS

An activity can interface with the operator console through the operator Interface Processor (OIP). An activity can type a message on the operator console (KSR or ASR) with a type message (TYP$) action routine, or it can type a message and request a response from the operator with a type message and input a response (TPR$) action routine.

The format of the message output on the operator console is as follows:

(TYP$ action routine)	ACTVTY MESSAGE
(TPR$ action routine)	MN ACTVTY MESSAGE

ACTVTY - The 6-character name of the associated activity
MN - A 2-character message number

The spaces, the activity name, and the message number are inserted by the OS/700 Executive components. The message number is used to associate a response with a particular message. The Operator Interface Processor (OIP) keeps a table of messages waiting for a reply.

Where the TPR$ request is utilized and the message cannot be typed because the message table is full, the user can try again later. While using the TYP$ or TPR$ requests the user must be aware that control is not transferred to him until the whole message is output (in the case of a TYP$ request) or the response received (in the case of a TPR$ request). The input range for the response cannot be larger than 36 words (72 characters), including a terminating carriage return. The output range can be of any length, however, and the output buffer can include carriage control characters to cause the message to be printed on several lines.

Examples:

For examples of the WIO$ function call with an inline parameter list, see the previous I/O examples, in particular, see the RSV$ and OTP$ description.

The examples below illustrate the use of the WIO$ function call with an outline parameter list.

In the following example, the I/O status block address pointer is not specified.

```
*
*         ---                    I/O REQUEST ISSUED
           .
           .
           .
          LDX     PLST10         PAR. LIST PTR. TO X
          WIO$    (X)            WAIT FOR I/O
           .
           .
           .
*
*         PARAMETER LIST FOR WIO$
*
PLST10    DAC     *+1            PTR. TO PAR. LIST
          BSZ     1              NO I/O STATUS BLK. ADDR. PTR.
```

The example below illustrates the WIO$ function call with the I/O status block address pointer specified in the outline parameter list.

```
*
*
          ---                    I/O REQUEST ISSUED
           .
           .
           .
          LDX     PLST11         PAR. LIST PTR. TO X
          WIO$    (X)            WAIT FOR I/O
           .
           .
           .
*
*         PARAMETER LIST FOR WIO$
*
PLST11    DAC     *+1            PTR. TO PAR. LIST
          DAC     PSTSBK         PTR. TO I/O STATUS BLK. ADDR.
*
*
PSTSBK    DAC     STSBLK         I/O STATUS BLK. ADDR.
*
*
STSBLK    BSZ     8              I/O STATUS BLOCK
```

NOTE: If the user's program is reentrant and it is not desirable to specify the I/O status block address pointer in the WIO$ request, the I/O status block may be a free memory block which is larger than eight words in length, and the area following or prior to the eight words required for the I/O status block may be used to contain user-saved parameters or to contain pointers to other blocks obtained from free memory.

I/O COMPLETION RETURN

There is no normal or error return from the WIO$ call. The activity is restarted at the I/O completion address specified in the preceding I/O request. If more than one I/O request was made, I/O completion for either request may occur first. The activity must continue to call WIO$ until all I/O completions returns occur. That is, as many WIO$ calls must be made as there were physical I/O requests.

ACTION ROUTINE DETAILS

A check is made to determine if the I/O status block address pointer has been specified (is not zero) in the WIO$ parameter list; if it has not, the current task is terminated. If the I/O status block address pointer is specified in the WIO$ parameter list, the pseudoregisters ZCR1, ZCR2, and ZCR3, the user keys, the banks, and the B and S hardware registers are saved, and the current task is terminated.

Before returning to the user at the I/O completion return address or the queued RSV$ request return address, the pseudoregisters ZCR1, ZCR2, and ZCR3, the user keys, the banks, and the B and S hardware registers are restored.

If a restricted activity issues a WIO$ request, a check is made to determine if there is an I/O request or queued reserve request outstanding. If there is no request outstanding, the restricted activity is aborted. If the WIO$ request is legal, the WIO$ function completes the request as described above.

OUTLINE PARAMETER LIST

Word Number	Operation	Operand
0	DAC	[I/O status block address pointer]

Parameters in the outline list are the same as those described above for the inline parameter list. If the I/O status block address pointer is not specified, a BSZ 1 statement must replace the respective DAC statement.

WIO$

Wait For I/O (WIO$)

The WIO$ function is used to terminate the current task following an I/O request or queued RSV$ request. The user activity is restarted when the I/O request is completed.

FUNCTION ACTION

The WIO$ action routine checks to determine if the I/O status block address pointer has been specified in the WIO$ parameter list, and if it has not, terminates the current task. If it has, the user pseudoregisters ZCR1, ZCR2, and ZCR3, the user's state (the keys, the banks, the B and S hardware registers) are saved, and the current task is terminated.

Before returning to the user at the I/O completion return address or the queued RSV$ request return address, the user's pseudoregisters ZCR1, ZCR2, and ZCR3, and the user's state (the keys, the banks, the B and S hardware registers) are restored.

If a restricted activity issues a WIO$ request with no I/O request waiting for I/O completion or with no queued reserve request waiting to be processed, the activity is aborted.

MACRO CALL FORMAT

Location	Operation	Operand
[symbol]	WIO$	[I/O status block address pointer]

symbol - Optional. The symbolic location of the WIO$ macro instruction.

I/O status block address pointer - Optional. The address of a word which contains the address of an 8-word block. The I/O status block must be the I/O status block which was specified in the device control block for an Input (INP$), Output (OTP$), Space File (SPF$), Space Record (SPR$), End of File (EOF$), Rewind (RWD$), or Reserve (RSV$) function call for which the Wait I/O request is being issued. If the I/O status block address pointer is specified in the WIO$ parameter list, the keys, the B and S hardware registers, and the pseudoregisters ZCR1, ZCR2, and ZCR3 will be saved and then restored before returning to the user at the I/O completion return address or the queued RSV$ request return address. In this case, only one I/O request may be issued prior to the WIO$ request. If the I/O status block address pointer is not specified in the WIO$ parameter list, the registers specified above are not saved and restored.

```
*                                 UNLOAD-REWIND MAGNETIC TAPE WITH AUTOMATIC
*                                 RELEASE
     ULD$    DCNM,ERUL            REWIND AND RELEASE THE DEVICE
*    ---                          RETURN BLOCKS TO FREE CORE STORAGE AND
*                                 TERMINATE
*
*                                 ERROR RETURN - A-REGISTER CONTAINS THE
*                                 ERROR CODE
*
ERUL                              ERROR,ILLEGAL PARAMETER IN THE DCB, DCNM
```

Using an outline parameter list, the above example would be written as follows:

```
*                                 UNLOAD-REWIND MAGNETIC TAPE WITH AUTOMATIC
*                                 RELEASE
*
     LDX     PL20                 PARAMETER LIST POINTER TO X
     ULD$    (X)                  REWIND AND RELEASE THE DEVICE
     ---                          TERMINATE
*
*                                 ERROR RETURN - A-REGISTER CONTAINS THE
*                                 ERROR CODE
*
ERUL ---                          ERRORS, ILLEGAL PARAMETER IN THE DCB, DCNM
*
*                                 PARAMETER LIST
*
PL20 DAC     *+1                  POINTER TO PARAMETER LIST
     DAC     DCNM                 DCB ADDRESS
     DAC     ERUL                 ERROR RETURN ADDRESS
```

5. A test is made to determine if the specified device is magnetic tape or cassette tape.

6. If the activity is restricted, a test is made to determine if the specified device was reserved by the activity, and a test is made to determine if all I/O requests issued for the specified device have been completed.

If the ULD$ request is not legal, control is returned to the error return address specified in the ULD$ parameter list with the error code in the A-register. If the ULD$ request is legal, the ULD$ request is placed at the beginning of the queue for the requested device. If the requested device is currently busy processing another request, control is returned to the calling program at the normal return. If the requested device is not currently busy, the ULD$ request is initiated, and control is returned to the calling program at the normal return. When the calling program receives control at the normal return, the task or activity may then be terminated.

NOTE: Before the ULD$ request is issued, all I/O requests for the device specified in the device control block of the ULD$ request must be completed. Also, after the ULD$ request is issued, no other I/O requests for the device specified in the device control block of the ULD$ request can be issued until the device is again reserved by issuing a RSV$ request.

OUTLINE PARAMETER LIST

Word Number	Operation	Operand
0	DAC	DCB address
1	DAC	error return address

Parameters in the outline parameter list are the same as those described above for the inline parameter list.

NOTE: If an outline parameter list is used and it is desired to update the I/O status block address in the DCB, the user must precede the ULD$ function call with the instructions to store the new I/O status block address in the sixth word (word 5) of the DCB.

Examples:

The following examples illustrate the use of the ULD$ function to rewind a magnetic tape and automatically release the device specified in the DCB. This allows the user to terminate without waiting for the magnetic tape to finish rewinding. The device control block is DCNM, which is shown in the DCB$ example, and the RSV$ request, which must be executed prior to the unload, is also shown in the DCB$ example.

I/O COMPLETION RETURN

There is no I/O completion return for ULD$ requests. The WIO$ function is not used following the ULD$ function call.

ERROR RETURN

Control is returned to the error return address specified in the ULD$ parameter list with the error code in the A-register when any of the following errors is detected:

A-register Contents (Octal)	Error Condition
1	The generic device type specified in the device control block is not configured for this system.
3	The logical unit number specified in the device control block is not configured for this system.
7	The ULD$ request was issued for a device other then magnetic tape or cassette tape.
10	The device has not been previously reserved under the user ID specified in the device control block.
14	An ULD$ request was issued for a disabled device.
15	The device was not reserved by the restricted activity that issued the ULD$ request.
32	The disk-resident driver for the specified device is not currently resident in main memory.
34	Requested executive function is not configured.
47	I/O request remains to be processed on the device (restricted activities only).

ACTION ROUTINE DETAILS

The following checks are made to determine if the ULD$ function call is legal:

1. A test is made to determine if the generic device type and the logical unit number specified in the device control block have been configured into the system.
2. A test is made to determine if the specified device is enabled.
3. A test is made to determine if the specified device has been previously reserved by a RSV$ function call.
4. A test is made to determine if the user ID specified in the device control block matches the ID for which the device was reserved.

ULD$

Unload (ULD$)

The ULD$ function is used to initiate the rewinding of a magnetic tape or cassette tape, and, upon the completion of the rewind, releases the requested device without the calling program issuing a REL$ request.

FUNCTION ACTION

The ULD$ action routine checks to determine if the specified device has been previously reserved by a RSV$ request; if it has not been reserved, the error return is taken. If it has, parameters specified in the device control block are checked, and the error return is taken if any of the parameters is illegal. If the parameters are legal, the ULD$ request is inserted at the beginning of the queue for the specified device. The unload is initiated if the specified device is not currently busy processing another request, and the normal return is taken. All I/O requests for the device must have been completed when ULD$ is called. Unlike other physical I/O functions, ULD$ must not be followed by a WIO$ call.

MACRO CALL FORMAT

Location	Operation	Operand
[symbol]	ULD$	DCB address, error return address, [I/O status block address]

symbol - Optional. The symbolic location of the ULD$ macro instruction.

DCB address - The name of the device control block associated with the unload device.

error return address - The address to which control is returned if an error is found in the specified device control block parameters, or if the specified device was not previously reserved.

I/O status block address - Optional. The address of an 8-word block used to insert the ULD$ request in the queue for the specified device.

If this parameter is omitted, the I/O status block address specified in the device control block is used. If this parameter is present, it replaces the I/O status block address in the device control block and any subsequent I/O call using the same device control block uses the new I/O status block address.

If additional I/O requests are to be made to the specified device following a ULD$ request, the device must first be reserved.

NOTE: Updating of the I/O status block address in the DCB is performed by the execution of the inline macro expansion and not by the action routine.

```
        LDX     PLIO            PARAMETER LIST POINTER TO X
        RWD$    (X)             REWIND
        WIO$                    SEQUENTIAL, WAIT FOR REWIND COMPLETION
RTAD    LDA     TBAD+1          REWIND COMPLETION, FETCH STATUS
        SUB     BOT             CHECK FOR BEGINNING OF TAPE
        SZE                     DID AN ERROR OCCUR?
        JMP     ERR             YES, PROCESS ERROR
        ---                     NO ERROR, CONTINUE
*
*                               ERROR RETURN - A-REGISTER CONTAINS THE
*                               ERROR CODE
*       .
ERRR    ---                     ERROR, ILLEGAL PARAMETER IN THE DCB, DCNM
*
*                               PARAMETER LIST
*
PLIO    DAC     *+1             POINTER TO PARAMETER LIST
        DAC     DCNM            DCB ADDRESS
        DAC     ERRR            ERROR RETURN ADDRESS
*
BOT     OCT     2               I/O STATUS BLOCK INDICATOR FOR BEGINNING
*                               OF TAPE
```

OUTLINE PARAMETER LIST

Word Number	Operation	Operand
1	DAC	DCB address
2	DAC	error return address

The parameters in the outline parameter list are the same as those described above for the inline parameter list.

NOTE: If an outline parameter list is used and it is desired to update the I/O status block address or the I/O completion return address in the DCB, the user must precede the RWD$ function call with the instructions to store the new I/O status block address in the sixth word (word 5) of the DCB or the new I/O completion return address in the seventh word (word 6) of the DCB.

Examples:

The following examples illustrate the use of the RWD$ function to rewind a magnetic tape. The device control block, DCNM, and the RSV$ request, which must be executed prior to the rewind, have been shown previously in the DCB$ examples.

```
*                               REWIND MAGNETIC TAPE
*
     RWD$   DCNM,ERRR           REWIND
     WIO$                       SEQUENTIAL, WAIT FOR REWIND COMPLETION
RTAD LDA    TBAD+1              REWIND COMPLETION, FETCH STATUS
     SUB    BOT                 CHECK FOR BEGINNING OF TAPE
     SZE                        DID AN ERROR OCCUR?
     JMP    ERR                 YES, PROCESS ERROR
     ---                        NO ERROR, CONTINUE
*
*
*                               ERROR RETURN - A-REGISTER CONTAINS THE
*                               ERROR CODE
ERRR ---                        ERROR, ILLEGAL PARAMETER IN THE DCB, DCNM
BOT  OCT    2                   I/O STATUS BLOCK INDICATOR FOR BEGINNING
                                OF TAPE
```

Using an outline parameter list, the above example would be written as follows:

```
*                               REWIND MAGNETIC TAPE
*
```

A-register Contents (Octal)	Error Condition
10	The device was not previously reserved under the user ID specified in the device control block.
14	A RWD$ request was issued for a disabled device.
15	The device was not reserved by the restricted activity which issued the RWD$ request.
32	The disk-resident driver for the specified device is not currently resident in main memory.
34	Requested executive function is not configured.

ACTION ROUTINE DETAILS

The following checks are made to determine if the rewind request is legal:

1. A test is made to determine if the generic device type and the logical unit number specified in the device control block have been configured into the system.
2. A test is made to determine if the specified device is enabled.
3. A test is made to determine if the specified device has been previously reserved by a RSV$ function call.
4. A test is made to determine if the user ID specified in the device control block matches the ID for which the device was reserved.
5. A test is made to determine if the specified device is magnetic tape or cassette tape.
6. If the activity is restricted, a test is made to determine if the specified device was reserved by the activity.

If the RWD$ request is not legal, control is returned to the error return address specified in the RWD$ parameter list with the error code in the A-register. If the RWD$ request is legal, the RWD$ request is placed at the beginning of the queue for the specified device. If the requested device is currently busy processing another request, control is returned to the calling program at the normal return. If the specified device is not currently busy, the RWD$ request is initiated, and control is returned to the calling program at the normal return. When the calling program receives control at the normal return, a WIO$ function call must be issued immediately if the calling program desires sequential I/O. If nonsequential I/O is desired, the calling program continues processing to the point that the rewind completion is required. At this point, a WIO$ function call must be issued. Upon completion of the rewind, the code specified by I/O completion return address will be scheduled. When the calling program receives control at the I/O completion return address, the calling program must assume the responsibility of checking the I/O status block to determine if the RWD$ request was successful. Word 1 of the status block will contain a 2 if no error occurred (see Appendix B for the status information). Note that before the RWD$ request is issued, all I/O requests for the device specified in the device control block of the RWD$ request must be completed.

Bit	Interpretation
1	Operational indicator
2	Busy indicator
3	Active indicator
4	End of form detected (See Note.)
5	Cycle check error (Line Printer Types 556x only) or 0 (See Note.)
6-8	Always zero
9	DMA parity error
10-14	Always zero
15	Active interrupt
16	Not busy interrupt

NOTE: Bits 4 and 5 are interchanged from the status word received from the hardware for compatibility with Printer Types 552x. For Printer Types 554x, 555x, and 556x, the end-of-form bit will be set in the I/O status block only if it is detected by the hardware and the forms control specified to the hardware was not a channel advance, including top of form.

Status Information for Line Printer Types 556x

Word 6 (seventh word) of I/O Status Block

In addition to the status information returned in words 1 and 4 of the I/O status block, the following hardware status is always returned in word 6 of the I/O status block.

Bit	Interpretation
1	Printer pattern parity error
2	Line buffer parity error
3	Sentinel bit error
4	Index check error
5	Load cycle in progress
6	Print cycle in progress
7	Format cycle in progress
8	Test mode
9	Interrupt request
10	Trap circuit
11	True comparison
12-16	Always zero

7-TRACK MAGNETIC TAPE SUBSYSTEMS (TYPES 402x AND 404x)
9-TRACK MAGNETIC TAPE SUBSYSTEMS (TYPES 405x, 415x, 418x, AND 419x)

The following applies to 7-track magnetic tape (Types 4021 and 4041) and to 9-track magnetic tape (Types 4051, 4150, 4180, and 4190).

Physical I/O requests for magnetic tape are input (INP$), output (OTP$), end of file (EOF$), space file (SPF$), space record (SPR$), rewind (RWD$), and unload (ULD$). Legal data modes for magnetic tape are 0 (ASCII), 1 (binary), and 2 (verbatim).

For 9-track tapes, ASCII data is not translated to BCD, but remains in ASCII format. For all devices, the entire 16 bits of each data word are transferred to/from the magnetic tape.

For 7-track tapes (Type 4021), binary mode specifies that all 16 bits of each word (three characters per word) are to be transferred to/from the magnetic tape:

1 6	7 12	13 16
first frame	second frame	third frame

Verbatim mode specifies that the high-order 12 bits of each word (two characters per word) are to be transferred to/from the magnetic tape:

1 6	7 12	13 16
first frame	second frame	not used

For 7-track tapes (Type 4041), binary mode specifies that the 16 bits of each data word are to be transferred to/from the magnetic tape in binary/word mode:

1 6	7 12	13 16
first frame	second frame	third frame

Verbatim mode specifies that 12 bits of data are transferred to/from the magnetic tape in binary/byte mode:

1 2	3 8	9 10	11 16
	first frame		second frame

(This BCD tape format is also used for 9-track magnetic tape Type 4051.)

When input from magnetic tape is requested, one record is read, and the data stored in the input buffer. The range value must be greater than or equal to the length of the record that is to be read from the tape. If the range value is less than the physical tape record, the parity error indicator is set in the I/O status block. If the data mode specified was ASCII, the data read from the magnetic tape is converted from BCD to ASCII for 7-track tape and stored in the user's buffer.

When output to the magnetic tape is requested, all data in the user's output buffer is written to the record on the tape. If the data mode specified is ASCII, the data in the user's output buffer is converted to BCD for 7-track tape before being written to the magnetic tape. Note that for 7-track tapes when the ASCII data mode is specified, the conversion from ASCII to BCD on output to tape and the reverse conversion from BCD to ASCII on input will result in the @ (at sign) character (ASCII '300) being converted to a ' (single quote) character (ASCII '247). This occurs as follows:

Character in Memory		→	Character on 7-track Tape	→	Character in Memory	
Graphic	ASCII		BCD		Graphic	ASCII
'	'247	→	'14	→	'	'247
@	'300	→	'14	→	'	'247

If the end of tape or the beginning of tape is detected before the specified number of files to be spaced has been reached when an SPF$ request is being processed, spacing of the files does not continue. The number of files actually spaced is returned in word 3 of the I/O status block.

If the end of file is detected before the specified number of records to be spaced has been reached when an SPR$ request is being processed, spacing of the records is not continued. The number of records actually spaced is returned in word 3 of the I/O status block.

The following is a description of the status information returned to the user when control is transferred to the user's I/O completion return.

Word 1 (second word) of I/O Status Block

Bit	Interpretation
1	Types 4041 and 4051 only: word 4 contains one hardware status word indicating the error; word 6 contains the second hardware status word.
2	Reserved
3	Missed interrupt
4	Reserved
5	Device disabled
6-7	Reserved
8	Parity error
9-10	Reserved
11	Protect error
12	Reserved
13	End of file
14	End of tape
15	Beginning of tape
16	Device busy

The above described states are indicated if the appropriate bit is set.

Word 3 (fourth word) of I/O Status Block

For input and output requests, this word contains the actual number of words transferred or received. For Space File and Space Record requests, this word contains the number of files or records spaced.

Status Information for Magnetic Tapes (Types 4041 and 4051)

Word 1 (second word) of I/O Status Block

These tapes return the same status information to the user when control is returned to the user's I/O completion return as defined for Types 4021 and 4150 Magnetic Tapes. (See word 4.)

Word 3 (fourth word) of I/O Status Block

For input and output requests, this word contains the actual number of words transferred or received. For Space File and Space Record requests, this word contains the number of files or records spaced.

Word 4 (fifth word) of I/O Status Block

In addition to the status information returned in word 1 of the I/O status block, these tapes return the following hardware status word in word 4 of the I/O status block:

Bit	Interpretation
1	Operational
2	Busy
3	Active
4	Device 0 selected
5	Device 1 selected
6	Device 2 selected
7	Device 3 selected
8	Rewind in process
9	Write protect
10-14	Reserved
15	Device going active interrupt
16	TCU busy reset interrupt

Word 6 (seventh word) of I/O Status Block

In addition to the status information returned in words 1 and 4 of the I/O status block, these tapes also return the following status word in word 6 of the I/O status block.

Bit	Interpretation
1	Beginning of tape status
2	End of tape status
3	File mark detected
4	Premature termination
5	Range too short
6	Range equals zero
7	LRC error
8	Low not high error
9	Skew error
10	CRC error
11	False gap/error detectable
12	Invalid setup
13	Data rate error
14	Write current failure
15	Write runaway
16	CRC parity/VRC error

CASSETTE TAPE SUBSYSTEM (TYPE 5400)

The physical I/O requests for the cassette tape are input (INP$), output (OTP$), end of file (EOF$), space file (SPF$), space record (SPR$), rewind (RWD$), and unload (ULD$). Spacing records and/or files, however, can only be forward. The legal data modes for cassette tape are 0 (ASCII), 1 (binary), and 2 (verbatim). Each of these data modes results in all 16 bits of each word being transferred.

When input from cassette tape is requested, one record is read from the tape and stored in the user-specified buffer. The range value must be greater than or equal to the length of the record that is to be read. If the range value is less than the length of the record, the parity error indicator is set in word 1 of the I/O status block. If a file mark is encountered when a record is read, the EOF status is set in word 1 of the I/O status block, and tape motion ceases immediately after the file mark.

When output to the cassette tape is requested, all data in the user's output buffer is written as a single record on the cassette tape. If a tape trailer label is encountered while a record is being written, no word count is returned to the user's status array. The end-of-tape indicator in word 1 of the I/O status block is set when the end-of-tape marker (18" before the physical EOT) is encountered. Writing after this is permitted but not recommended, as it is impossible to tell when the tape trailer will be encountered. Writing on this section of tape is mostly for a file mark, to denote EOT.

If the physical end of tape is encountered before the specified number of records or files have been spaced, tape motion ceases and word 3 of the I/O status block reflects the number of records or files successfully spaced. Further, if a file mark is encountered when records are spaced, tape motion ceases, and the number of records successfully spaced is returned in word 3 of the I/O status block.

The following describes status information returned to the user when control is transferred to the user's I/O completion return. The states described below are indicated if the appropriate bit is set.

Word 1 (second word) of I/O Status Block

Bit	Interpretation
1	An error has occurred; the hardware status is in word 4
2	Reserved
3	Missed interrupt
4	Reserved
5	Device disabled
6-7	Reserved
8	Parity error
9-10	Reserved
11	Write protect error
12	Reserved
13	End of file
14	End of tape
15	Beginning of tape
16	Device busy

Word 3 (fourth word) of I/O Status Block

For input and output requests, this word contains the actual number of words transferred or received. For Space File and Space Record requests, this word contains the actual number of files/records spaced.

Word 4 (fifth word) of I/O.Status Block

In addition to the status information returned in word 1 of the I/O status block, the following hardware status word is returned in word 4 of the I/O status block, if an irrecoverable error has occurred.

Bit	Interpretation
1	Operational
2	Busy
3	Active, first handler
4	Active, second handler
5	EOT marker
6	BOT marker
7	Rewind
8	Handler select
9	Access error
10	Not used
11	Write phase-encoded data
12	Read phase-encoded data
13	Write protect
14	Not busy interrupt
15	Active interrupt
16	Data ready interrupt

TELEPRINTERS (KEYBOARD/PRINTER) (TYPE 5507 ASR-35 AND TYPE 5307 ASR-33)

The physical I/O requests, the mode, and the control characters for the ASR-35 (keyboard/printer) are the same as described earlier for the KSR-33 teleprinter. The only difference is that bit 15 in word 1 (second word) of the I/O status block is set if there are no free memory blocks available for the ASR device driver to use.

TELEPRINTERS (READER/PUNCH) (TYPE 5507 ASR-35 AND TYPE 5307 ASR-33)

The legal physical I/O requests for the ASR-33 (reader/punch) are input (INP$), output (OTP$), and end of file (EOF$). Legal data modes for the ASR device are listed in Appendix G.

When input is requested, data on paper tape is read until an X-OFF character is reached, and then stored in the user's input buffer. The word count in word 3 (fourth word) of the I/O status block indicates the number of words stored in the user's input buffer.

When output is requested, data in the user's output buffer is punched on paper tape until the range count is exhausted. Punching the end-of-file character is initiated only by an EOF$ request.

The following is a description of the status information returned to the user when control is transferred to the user's I/O completion return.

Word 1 (second word) of I/O Status Block

Bit	Interpretation
1-2	Reserved
3	Missed interrupt
4	Device not ready
5-6	Reserved
7	Checksum error
8	Parity error
9	Mode error or format error
10	Control-K received during printer output
11-12	Reserved
13	End of file
14	Range error
15	Free memory not available
16	Reserved

Word 3 (fourth word) of I/O Status Block

For INP$ and OTP$ requests, word 3 contains the actual number of words transferred or received.

NOTE: If power failure occurs, the state of the I/O devices is not saved. When the user restarts the system, he must again reserve the I/O devices before using them.

APPENDIX C

PHYSICAL I/O DATA MODE ASSIGNMENTS

Mode Number	Data Mode
0	ASCII without checksum
1	Binary without checksum
2	Verbatim
3	ASCII with checksum for Type 5307 ASR-33, Type 5507 ASR-35, Type 5010 Paper Tape Reader, and Type 5210 Paper Tape Punch only.
4	Binary with checksum for Type 5307 ASR-33, 5507 ASR-35, Type 5010 Paper Tape Reader, and Type 5210 Paper Tape Punch only.

APPENDIX D
PHYSICAL I/O GENERIC DEVICE TYPE ASSIGNMENTS

Type Number	Generic Device Type
0	KSR
1	Cartridge disk subsystem
2	High-speed paper tape reader
3	High-speed paper tape punch
4	Reserved
5	Card punch
6	Card reader
7	Fixed-head disk subsystem
8	Removable disk subsystem
9	Line printer
10	Magnetic tape subsystem
11	Cassette tape subsystem
12	ASR - Keyboard and printer
13	ASR - Reader and punch

APPENDIX E

SYSTEM ERROR MESSAGES

Three types of system error messages are possible in OS/700: I/O device errors, executive errors, and communications supervisor errors. These messages are inhibited if there is insufficient free memory.

I/O DEVICE ERRORS

An I/O device error generates a message with the following format on the system operator device:

```
SE=0eeeee dddddd
```

SE - Indicates an error message

0eeeee - 6-digit octal number that specifies the type of error

dddddd - 6-digit octal number that specifies the device that caused the error

An I/O error is identified by the high-order (leftmost) digit of 0eeeee equal to zero. To determine the source and cause of an error, proceed as follows:

1. Convert the octal number dddddd to a 16-bit binary number and divide it into two bytes. The left byte contains the generic device type, and is represented in Table E-1 as the number in parentheses in the "Device Type and Device in Error" column. The right byte contains the logical unit number, of the device in error, and is represented in the same column in Table E-1 as the "uu" value in the parentheses.
2. Interpret the value 0eeeee, which specifies the type of error that occurred, by referring to the "Error Type" column in Table E-1.

Table E-1. I/O Error Codes and Meanings

Generic Device Number (Decimal)	Device Type and Device in Error (dddddd) (octal)	Error Type (0eeeee) (octal)	Error Condition
0	KSR-33 teleprinter		None
1	Cartridge disk (4uu)	1	Missed interrupt
		2[a]	Device not operational
		3	Missed data (transfer rate failure)
		4	Recovery error (miscellaneous)
		5	Protect error on OTP$
		6	Controller busy
		7	DMA bus parity error
		10	Checksum error
		11	Segment not found
		12	Fixed volume missing
		13	No free memory block available
2	Paper tape reader (10uu)	1	Unit disabled due to hardware error
		2[a]	Unit not operational – power off
3	Paper tape punch (14uu)	1	Unit disabled due to hardware error
		2[a]	Not operational – power off
		3	Tape low
5	Card punch (24uu)	1[a]	Missed interrupt
		2[a]	Device not operational
		3	Punch check error
		4	Data access error
		7	No free memory block available for data conversion
		10	Controller failed to respond
		11	Retry failed
		12	Operator action timer timed out
6	Card reader (30uu)	1[a]	Missed interrupt
		2[a]	Device not operational
		4[a]	Data access error
		5[a]	Read cycle error
		6[a]	Invalid Hollerith code
		7	No free memory block available for data conversion
		10	Unit disabled error (Type 5100 only) Controller failed to respond
		11	Stacker full or hopper empty (Type 5100 only) Retry failed
		12	Registration error (Type 5100 only) Operator action timer timed out
		13	Correct column option settings

Table E-1 (cont). I/O Error Codes and Meanings

Generic Device Number (Decimal)	Device Type and Device in Error (dddddd) (octal)	Error Type (0eeeee) (octal)	Error Condition
7	Fixed-head disk (34uu)	1[a]	Missed interrupt
		2[a]	Device not operational
		3	Access error
		4	Recovery error
		5	Protect error
		7	Parity error
8	Removable disk (40uu)	1[a] (DMC only)	Missed interrupt
		2[a]	Device not operational
		3	Missed data
		4	Recovery error
		5	Protect error
		6[a] (DMC only)	Controller busy
		7	Bus parity error
		10	Checksum error
		11	Segment not found
9	Line printer (44uu)	1	Unit disabled due to hardware error
		2[a]	Device not operational
10	Magnetic tape unit (50uu)	1	Missed interrupt
		2[a]	Device not operational
		3	Operator failed to make device operational
		4	Operator failed to permit writing
		5[a]	Writing not permitted
		6	Controller busy
		7	Write parity error
		10	Read parity error
		11	Hardware error (Types 4041 and 4051 only)
11	Cassette tape	1	Missed interrupt
		2[a]	Device not operational
		3	Operator failed to make device operational
		4	Operator failed to permit writing
		5[a]	Writing not permitted
		6	Controller busy or not operational
		7	Write parity or access error
		10	Read parity or access error
12	ASR-33 teleprinter		None
13	ASR-35 teleprinter		None

[a] The system allows the operator several minutes to recover from this error.

EXECUTIVE ERRORS

Executive errors have two formats:

```
SE=1fffff ssssss
SE=1fffff ssssss actnam
```

SE - Indicates system error

1fffff and ssssss - 6-digit octal numbers

actnam - 6-character ASCII string specifying the name of the activity that was executing or requested when the error occurred.

Executive errors are indicated by the high-order (leftmost) digit of 1fffff equal to 1. The activity name is printed only for certain errors, which are specified in Table E-2.

In all cases, ssssss, which is called the second error code indicator, is meaningful only for certain errors, as shown in Table E-2. When the second indicator is not meaningful, siz zeros are printed.

Table E-2. Executive Error Codes and Meanings

Error Code	Meaning
100001	In a COS, indicates that $SA or $LA command was being executed when another such command was entered. Wait for present function to be completed before typing another command. In a DOS, system failed to schedule an activity as requested by $SA command, either because activity did not exist or because error occurred in scheduling process <u>or</u> while under CI mode, a command other than $TR was typed on the console. Second error code indicator contains the ASCII characters "OI" ('147711).
100002	Operator typed a line which, subsequent to the initial (<u>P</u>), had neither a dollar sign (indicating system command) nor a valid message number (one associated with an unanswered message). The line is ignored. Second error code indicator contains the ASCII characters "OI" ('147711).
100003	Console I/O error occurred during operator typein; usually means operator waited too long to complete typing in a line once the (<u>P</u>) was typed. The line is ignored. Second error code indicator contains the ASCII characters "OI" ('147711).
100004	Operator's response to system or activity message contained too many characters. The line is ignored. Second error code indicator contains the ASCII characters "OI" ('147711).
100005	The text following a typein of (<u>P</u>)$ was invalid system command. (The set of valid system commands varies. In a nondedicated COS, $SA and $LA are valid. In a DOS, $SA is valid. If command input mode is configured, $CI and $TR are valid. If system integrity is configured, $AB is valid. Dedicated COS recognize no system commands.) Second error code indicator contains the ASCII characters "OI" ('147711).
100006	The abort activity request which was made by the operator ($AB) is invalid because either the activity is nonrestricted or the activity was not requested. Second error code indicator contains the ASCII characters "OI" ('147711).

Table E-2 (cont). Executive Error Codes and Meanings

Error Code	Meaning
100010	The file or activity could not be deleted because of deallocation error. The volume name is also specified.
100013	No free memory available to disk initialization.
100021	No work area available for allocation in either system or user area on volume. Operator should write "volume full" or "volume user area full" on appropriate disk volume.
100022	Disk error during disk initialization. Second indicator specifies disk unit number.
100024	Overlay cannot be read into main memory. Second indicator contains starting segment number of desired overlay.
100030	Activity area overrun. Activity name is specified with error indicators.
100031	Activity supervisor disk error while reading activity. Name of activity is specified.
100116	Specified activity name not found in the activity directory. Second indicator contains function number of Connect Clock Activity function. Activity name also specified.
100122	Disk error while referencing the disk activity directory. Second indicator contains function number of Connect Clock Activity function. Activity name also specified.
100126	No activity area for the named activity (the main-memory starting address given in the disk directory is not equal to the beginning of any activity area) or the activity is too large to fit into the allocated activity area (i.e., the activity ending address exceeds the activity area ending address). Second indicator contains function number of Connect Clock Activity function. Activity name also specified.
100163	No work area available for allocation in user area on volume. Operator should write "volume full" or "volume user area full" on appropriate disk volume.
100210	CI failed to get the next line in the CI command file and CI mode terminated. The GET$ function took the error return and 'ssssss' contains the A-register setting. See Appendix A, Executive Function Call Error Codes.
100211	CI was unable to open the file specified in the $CI command and CI mode terminated. The OPN$ function took the error return and 'ssssss' contains the A-register setting. See Appendix A, Executive Function Call Error Codes.
100212	If 'ssssss' contains OI in ASCII ('147711), a $SA command attempted to start a second activity under CI control and CI mode terminated. If 'ssssss' does not contain OI, CI was unable to schedule the activity in a $SA command in the command file. The SAC$ function took the error return and 'ssssss' contains the A-register setting. See Appendix A, Executive Function Call Error Codes.
100213	CI was unable to close the CI command file and CI mode terminated. The CLS$ function took the error return and 'ssssss' contains the A-register setting. See Appendix A, Executive Function Call Error Codes.

Table E-2 (cont). Executive Error Codes and Meanings

Error Code	Meaning
100214	A syntax error in a system command: ● Invalid command in a CI command file ● Filename following $CI is too long ● Activity name following $SA on console is too long ● No initial $ in a command file line when no activity is running under CI control and a response line is not expected and CI mode terminated. 'ssssss' is OI in ASCII ('147711).
100215	The activity name following $SA in a CI command file is too long and CI mode terminated. 'ssssss' is OI in ASCII ('147711).
100216	The response line in a CI command file is longer than the activity, running under CI control, expects and CI mode terminated. 'ssssss' is OI in ASCII ('147711).
100217	CI was unable to release the old output device while processing a $OD command and CI mode terminated. The REL$ function took the error return and 'ssssss' contains the A-register setting. See Appendix A, Executive Function Call Error Codes.
100220	CI was unable to reserve the new output device while processing a $OD command and CI mode terminated. The RSV$ function took the error return and 'ssssss' contains the A-register setting. See Appendix A, Executive Function Call Error Codes.
100221	The device specified in a $OD command is not configured and CI mode terminated. 'ssssss' contains OI in ASCII ('147711).
100222	CI encountered a bad I/O status on return from the OTP$ function when writing a record to the current output device and CI mode terminated. 'ssssss' contains word 1 of the physical I/O status block. See Appendix B, Physical I/O Device Information.
100223	CI was unable to output a record to the current output device and CI mode terminated. The OTP$ function took the error return and 'ssssss' contains the A-register setting. See Appendix A, Executive Function Call Error Codes.

COMMUNICATIONS SUPERVISOR MESSAGES

The OS/700 Communications Supervisor issues system error messages on the console in two formats. They are: communications supervisor event reports (CSEVRT), and communications supervisor configuration errors (CSCNFE). Each is identified by the 6-letter code word in the activity name field of the system error message.

Event Report (CSEVRT)

A communications supervisor event report message reports changes in the status of the communications subsystem in the following format:

```
SE= xxxxxx yyyyyy CSEVRT
```

1. The octal value xxxxxx is treated as a 16-bit binary number.

   Bit 1 indicates:

   0 - User program command response

   1 - Alarm condition response

   Bits 2 and 3 contain internal information.

   Bits 4 through 8 contain an octal message code specified in Table E-3.

   Bits 9 through 16 may contain additional information. See Table E-3.

2. Interpretation of yyyyyy depends on the message code. See Table E-3.

Table E-3. Communications Supervisor Message Codes

Code Bits 4-8 of xxxxxx	Message Type	Parameter 2 CSEVRT (yyyyyy)	Parameter 2 CSCNFE (yyyyyy)	Parameter 1 Bits 9-16 of xxxxxx
01	Device State Change		DLT	
04	System Status	SSW	SSW	
06	Device Failure	DLT	LTA	
07	Free Core Alarm	SSW	SSW	
10	Line Status	LSW	DLT	
11	Terminal Status	TSW	TSW	
12	Format Error	Parameter 2[a]		
13	Line State Change		DLT	
14	Terminal State Change		DLT	
15	Device Enabled	DLT	DLT	
16	Line Alarm	DLT	LTA	See Figure E-6.
21	Discipline Failure	DLT	LTA	See Figure E-7.
22	Device Looped	DLT	DTA	
23	Device Unlooped	DLT	DTA	
24	Line Poll Failure	DLT	LTA	See Figure E-8.
25	Line Select Fail	DLT	LTA	See Figure E-8.
26	Output Select Fail	DLT	LTA	See Figure E-8.
27	Device Status		DTA	
30	VIP Status	DLT	LTA	See Figure E-9.

DTA - Device Table Address

LTA - Line Table Address

DLT - Device/Line/Terminal Number (See Figure E-5.)

SSW - System Status Word (See Figure E-4.)

DSW - Device Status Word (See Figure E-1.)

LSW - Line Status Word (See Figure E-2.)

TSW - Terminal Status Word (See Figure E-3.)

[a] The original user parameter is given.

Configuration Errors (CSCNFE)

A communications supervisor configuration error message has the format:

SE= xxxxxx yyyyyy CSCNFE

This message indicates that a processing request to the communications supervisor cannot be completed because a required communications supervisor function is not configured. The two octal numbers xxxxxx and yyyyyy contain the original communications supervisor parameters for calls that were not completed due to the unconfigured function. Values are interpreted as follows:

1. The xxxxxx is treated as a 16-bit binary number.

   Bit 1 indicates that the requested function was for:

   0 - A communications supervisor command

   1 - An alarm condition processor

   Bits 2 and 3 contain internal information.

   Bits 4 through 8 contain an octal message code specified in Table E-3.

   Bits 9 through 16 may contain additional information. See Table E-3.

2. Interpretation of yyyyyy depends on the message code. See Table E-3.

Status Word Formats

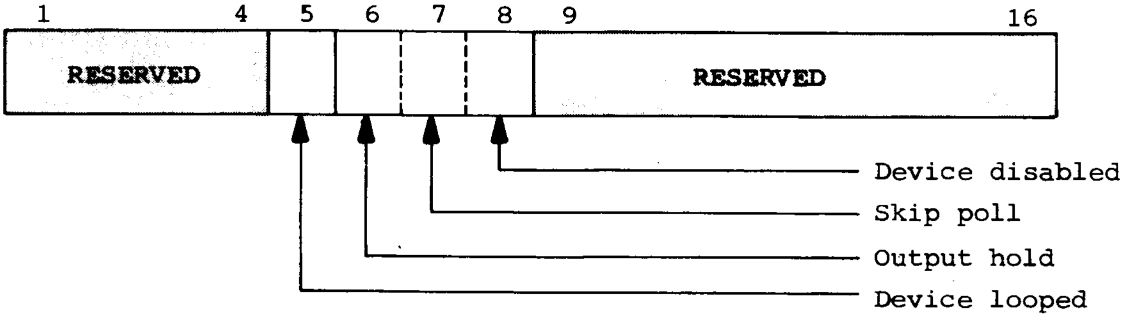


Figure E-1. Device Status Word

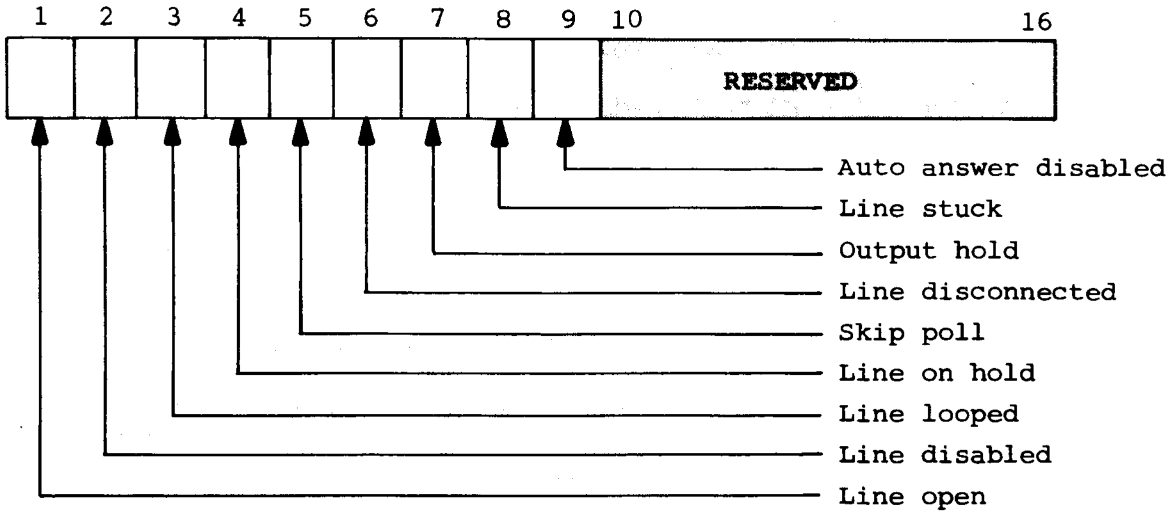


Figure E-2. Line Status Word

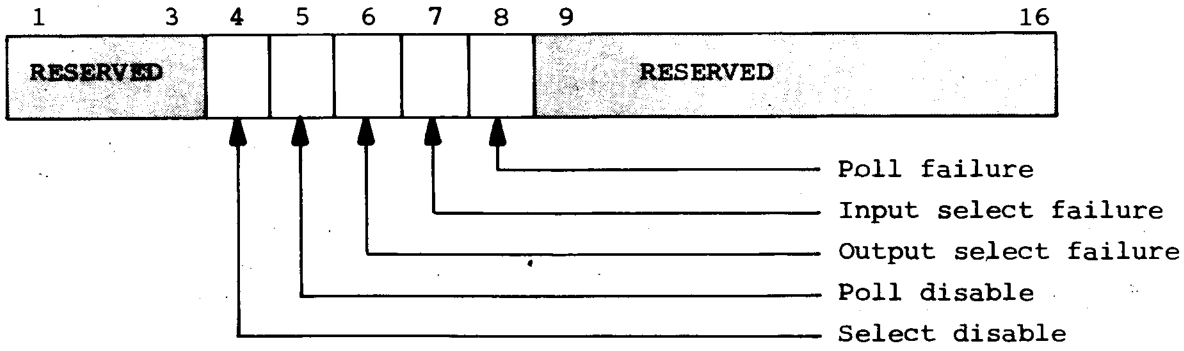


Figure E-3. Terminal Status Word

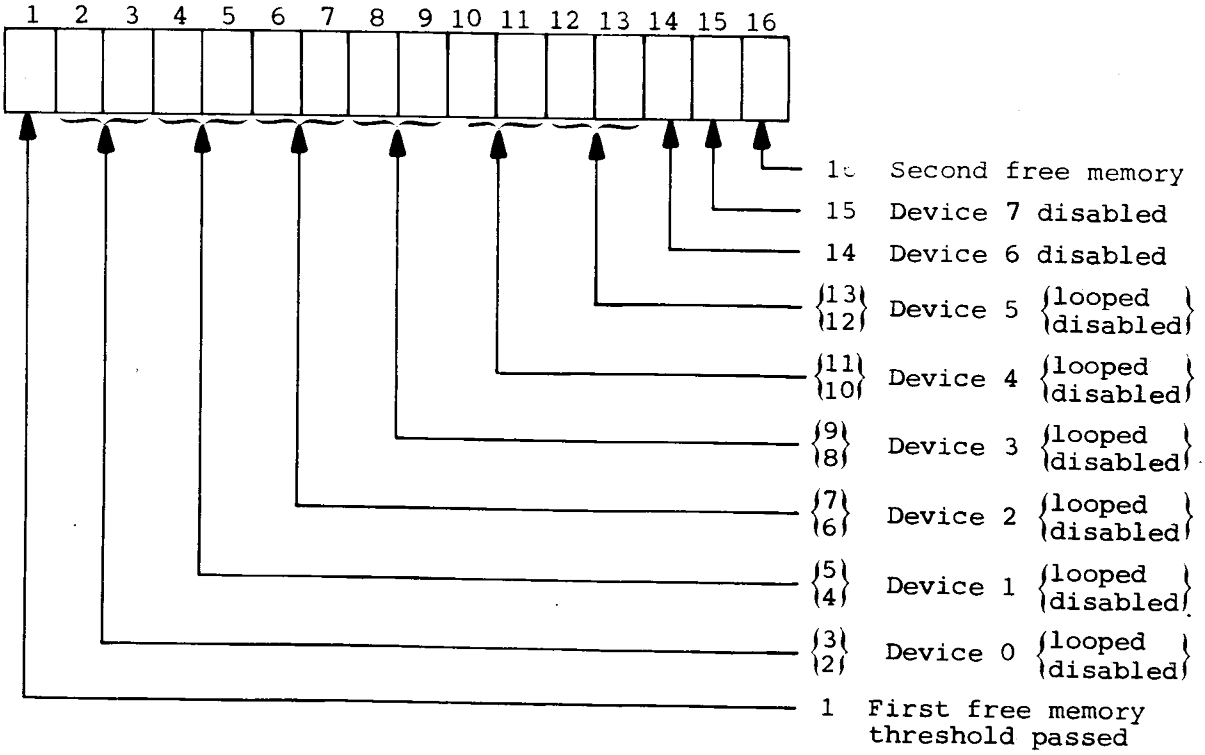

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
16 Second free memory
15 Device 7 disabled
14 Device 6 disabled
{13 12} Device 5 {looped disabled}
{11 10} Device 4 {looped disabled}
{9 8} Device 3 {looped disabled}
{7 6} Device 2 {looped disabled}
{5 4} Device 1 {looped disabled}
{3 2} Device 0 {looped disabled}
1 First free memory threshold passed

Figure E-4. System Status Word

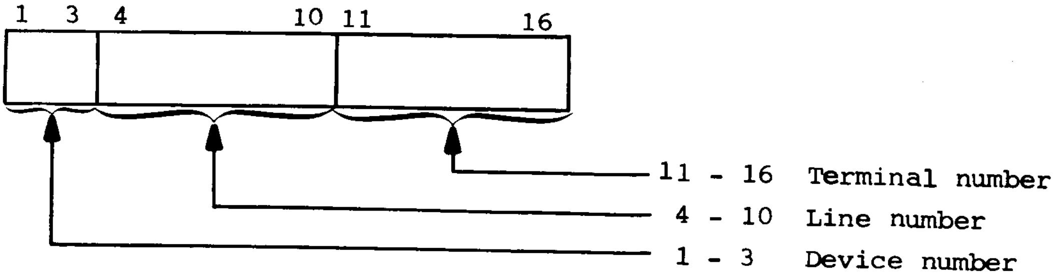


Figure E-5. Device/Line/Terminal Number Word

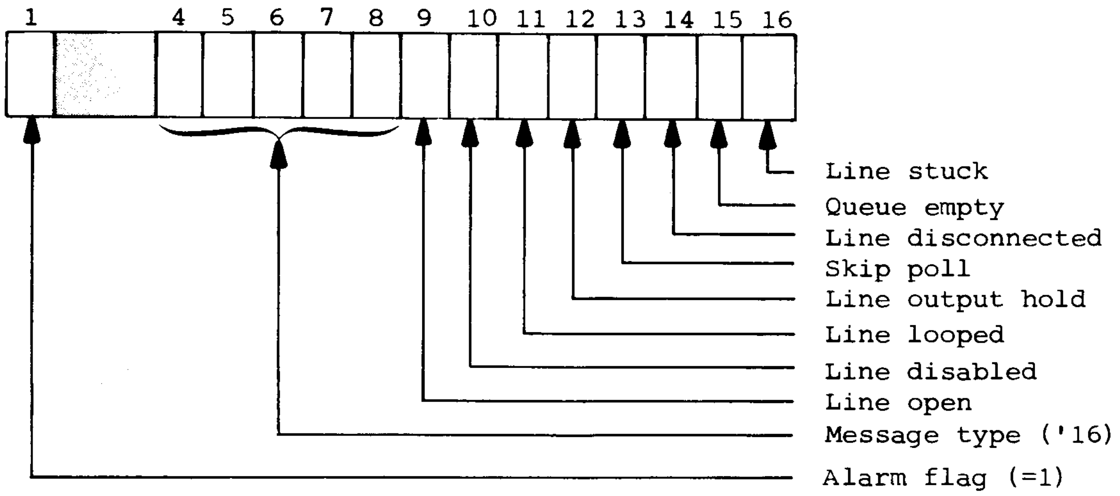


Figure E-6. Parameter 1 of Line Alarm Message

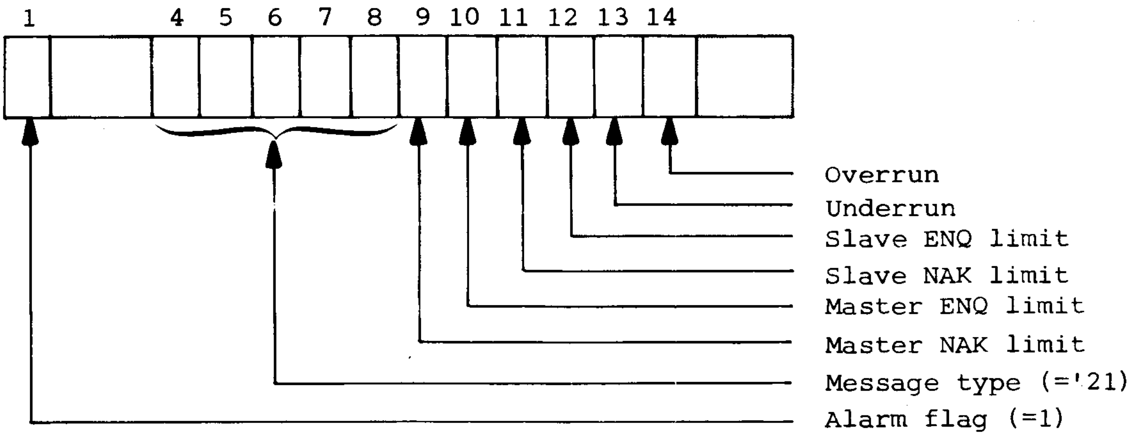


Figure E-7. Parameter 1 of Discipline Failure Message

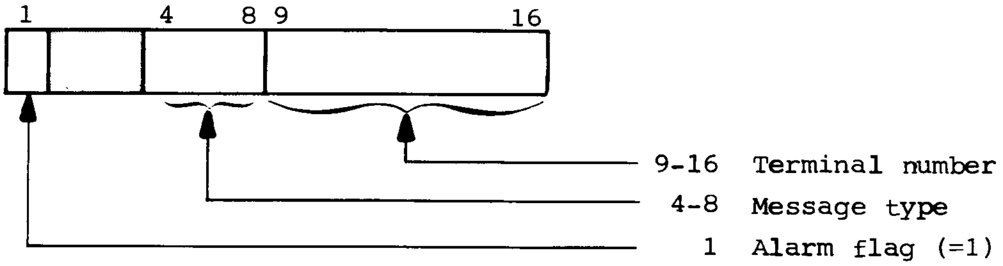
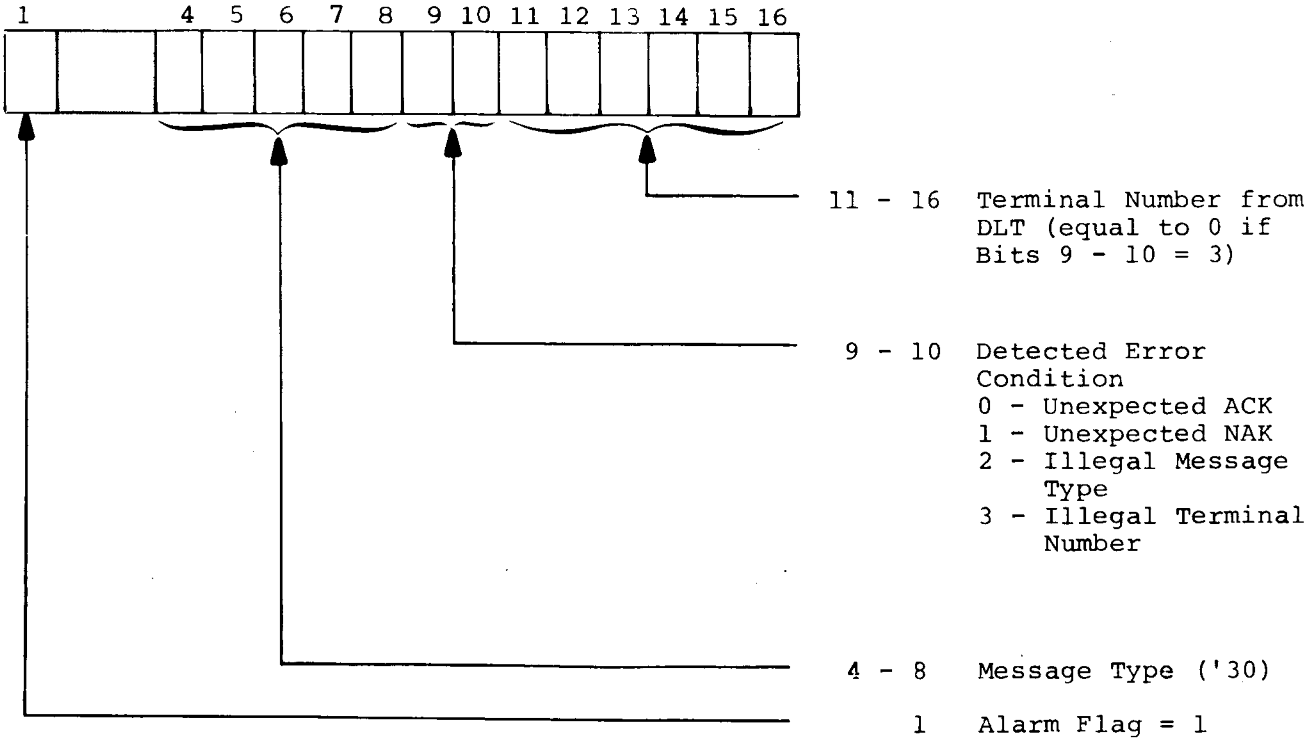


Figure E-8. Parameter 1 of Line Poll Failure, Line Select Failure-Output, Line Select Failure-Input Messages



11 - 16 Terminal Number from DLT (equal to 0 if Bits 9 - 10 = 3)

9 - 10 Detected Error Condition
0 - Unexpected ACK
1 - Unexpected NAK
2 - Illegal Message Type
3 - Illegal Terminal Number

4 - 8 Message Type ('30)

1 Alarm Flag = 1

Figure E-9. Parameter 1 of VIP Status Message

APPENDIX F

ACTIVITY ABORT MESSAGES

An activity abort message occurs whenever the system or the operator aborts a restricted activity. The format is:

```
****** <actnam> <rr> [<'aaaaaa>]
```

<actnam> - 1- to 6-character ASCII name of the restricted activity which was aborted.

<rr> - 2-character abbreviation of the reason for the abort. See Table F-1.

<'aaaaaa> - Address printed if <rr> is MV, BP or IF.

Output of the message is suppressed if free memory is very low.

Table F-1 contains the reasons for the activity abort.

Table F-1. Reason for Abort

<rr>	Meaning
FC	Free memory is low.
OP	Abort was requested by the operator ($AB command) or by a non-restricted activity (ABT$ executive function call).
MV	A memory lockout violation occurred: ● An attempt to write in a protected area of memory (STA, DST, STX, LDX, IMA, IRS, and JST). ● An illegal instruction (HLT, INH, INA, IMK, OTA, OTK, OCP, SKS, SMK, and CAI). ● More than eight levels of indirect addressing. <'aaaaaa> contains the address where the memory lockout violation occurred.
BP	A bad parameter was passed to an action routine. ● A word or block specified directly or indirectly by the parameter list does not reside entirely in the activity. ● The FCBB or LCBB pointer specified by the FCB or LCB is not the one given to the activity by the system when the activity opened the file or library. <'aaaaaa> contains the address of the function number of the executive function called.

Table F-1 (cont). Reason for Abort

<rr>	Meaning
IF	An illegal function was requested ● A TMT$ request with no other task of the activity scheduled. ● A TMA$ request with a bad TCB. ● A WIO$ request with no I/O request pending or no queued reserve request waiting. ● One or more I/O requests pending and the function requested is not: EOF$, INP$, OTP$ RWD$, SPF$, SPR$ ULD$, WIO$. ● Not a permissible function. See Tables F-2, F-3.

The subset of executive functions which may be requested by Restricted Activities is listed in Table F-2.

Table F-2. Permissible Functions

Function	Meaning
ALC$	Allocate a work area
ATQ$	Attach entry to queue
CFP$	Change file password
CLL$	Close library
CLP$	Change library password
CLS$	Close file
CRL$	Create library
CRQ$	Create queue
CVL$	Connect volume
DLC$	Deallocate a work area
DVL$	Disconnect volume
EOF$	End of file
GDT$	Get date and time
GET$	Get a record
GSP$	Get system parameters
GTQ$	Get top entry from queue
INP$	Input
OPL$	Open library
OPN$	Open file
OTP$	Output
PUT$	Put a record
REL$	Release a device
RSV$	Reserve a device
RWD$	Rewind
SAC$	Schedule an activity

Table F-2 (cont). Permissible Functions

Function	Meaning
SPF$	Space file
SPR$	Space record
STS$	Schedule task
SUS$	Suspend task
TMA$	Terminate an activity
TMT$	Terminate task
TPR$	Type a message and input a response
TYP$	Type a message
ULD$	Unload (Rewind with automatic release)
WIO$	Wait for I/O completion

Executive functions that may not be requested by Restricted Activities are listed in Table F-3.

Table F-3. Nonpermissible Functions

Function	Meaning
ABT$	Abort activity
CCA$	Connect clock to activity
CCL$	Connect clock to task
CCSS$	Change station status
CCST$	Connect station
CDST$	Disconnect station
CGCB$	Get communications block
CGSS$	Get station status
CRAR$	Receive and reformat
CRAS$	Reformat and send
CRCB$	Return communications block
CREC$	Receive
CSDC$	Send control
CSND$	Send
CTC$	Create a task control block
CTMC$	Terminate communication task
DCA$	Disconnect activity from clock
DCL$	Disconnect task from clock
GBL$	Get storage block
RBL$	Return storage block
SDT$	Set date and time
STC$	Schedule a task control block

APPENDIX G

FREE MEMORY BLOCK PARAMETER PASSING TECHNIQUE

The format of the parameter passing technique used by FORTRAN under OS/700 when using free memory blocks involves the TCB format, parameter block format, parameter format, and data structure.

TCB FORMAT

Use of the free memory block parameter passing technique is designated whenever the TCB has the following format:

Word Relative Displacement Symbol	Contents
ZTCBP1	Must be zero. This is parameter 1 in the SAC$, STS$, CTC$, CCA$, or CCL$ executive function call.
ZTCBP2	Points to the first word of the block containing the actual parameters. This is parameter 2 in the SAC$, STS$, CTC$, CCA$, or CCL$ executive function call.

PARAMETER BLOCK FORMAT

The block containing the actual parameters has the following format (see Figure G-1):

Word Number	Contents
0	Reserved
1	Reserved
2	Parameter block size descriptor.

Bit	Contents
1	Block release control: 0 - Release block to free memory when done. 1 - Don't release block to free memory when done.
2-8	Must be zero.
9-16	Parameter block size in number of words. (NOTE: it is not expressed as a power of 2).

3 Parameter block descriptor.

Bit	Contents
1-8	Reserved for parameter passing technique type. Must be zero (this is technique 0).
9-16	Number of data words in the parameter block. This is a count of the number of following words which actually contain parameter data.

4-n Parameter data.

Example:

The Start Activity online utility program (SA) allows a user to input an ASCII line in a 64-word parameter block in standard format. The option to accept a parameter line must be specified to the Load Activity (LA) command when the activity is loaded. When the Start Activity command starts such a program, the Start Activity command will output the following message to the operator's console:

PARAMETERS!

The user response line (UVWXYZA<CR>, for example) will be used as the first and only parameter in a 64-word parameter block as follows:

Word Number	Octal Contents	Meaning
0	0	Reserved
1	0	Reserved
2	100	64-word block to be returned
3	5	Five words of data
4	4	Four words for parameter 1
5	152726	UV
6	153730	WX
7	154732	YZ
8	140615	A<CR>
9 through 63		Unused

PARAMETER FORMAT

Each parameter in the parameter block has the following format:

Word Number	Contents
0	Parameter descriptor.

Bits	Contents
1-8	Reserved for parameter type. Must be zero (this is type 0).
9-16	Number of data words in the parameter. This count does not include this word itself.

Word Number	Contents
1	First word of actual parameter.
2	Second word of actual parameter (if more than one word).
3	Third word of actual parameter (if more than two words).

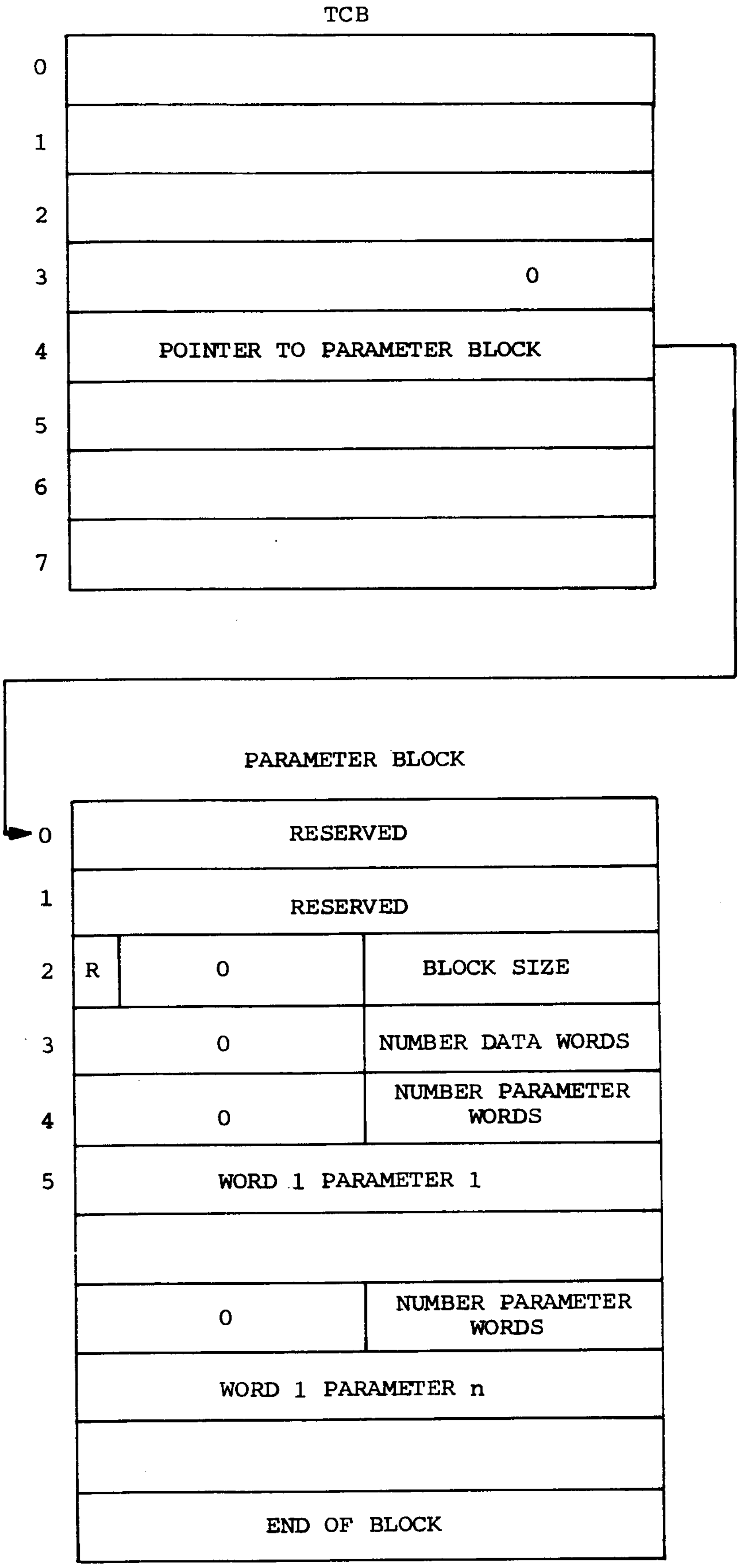


Figure G-1. Parameter Block Data Structure

APPENDIX H
OS/700 DATA STRUCTURES

This appendix describes the organization and contents of those date structures referenced by user programs.

ACTIVITY CONTROL BLOCK (ACB)

Each activity executed under the control of OS/700 is defined by an activity control block (ACB). The ACB defines the activity attributes common to all tasks within an activity. Figure H-1 shows the organization and contents of the ACB.

Activity Control Block		Word Number	Contents
RRAD LINK WORD		0	Pointer to next ACB in the resident or requested activity directory, or 0 if end of directory.
ACTIVITY STARTING ADDRESS		1	Lead task entry point; used in generating the primary TCB.
ACTIVITY J-BASE AND FLAGS		2	See Table H-1.
ACTIVITY NAME		3	Characters 1 and 2 of the activity name.
CHAR 1	CHAR 2		
ACTIVITY NAME		4	Characters 3 and 4 of the activity name.
CHAR 3	CHAR 4		
ACTIVITY NAME		5	Characters 5 and 6 of the activity name.
CHAR 5	CHAR 6		
ACTIVITY DEFAULT PRIORITY LEVEL		6	Bit 1 is a system flag; bits 2 through 4 equal zero, bits 5 through 8 contain the default priority level, bits 9 through 16 equal zero.
ACTIVITY STATUS WORD		7	See Table H-2.
START OF ACTIVITY REQUEST QUEUE		8	Starting address of the queue which contains the TCB's for all requests made on the activity.

Figure H-1. Activity Control Block (ACB) Structure

Activity Control Block	Word Number	Contents
ACTIVITY AREA TABLE	9	Pointer to the Activity Area Table entry for the activity area used by the activity. For a COS or a permanently memory-resident activity, this word contains a pointer to word 11 of the ACB.
ACTIVITY AREA REQUEST QUEUE	10	If bit 1 is 1, the ACB is not linked in the Activity Area Request Queue (the header word for this queue is in the Activity Area Table entry). The ACB is added to this queue, and bit 1 is set to zero when the activity is to be loaded. If bit 1 is zero, bits 2 through 16 contain either zero to indicate the end of the queue, or the pointer to an ACB of another activity which is waiting to be loaded in the same activity area.
FIRST SEGMENT NUMBER OF THE ACTIVITY	11	The first segment number specifies where the activity begins on the disk. For a COS or a permanently memory-resident activity, this word contains the starting address of the activity area.
LAST SEGMENT NUMBER OF THE ACTIVITY	12	The last segment number specifies where the activity ends on the disk. For a COS or a permanently memory-resident activity, this word contains the ending address of the activity area.
NUMBER OF WORDS IN THE LAST SEGMENT OF THE ACTIVITY	13	This word specifies the number of words used to contain the activity in the last segment on the disk. For a COS or a permanently memory-resident activity, this word equals zero.
ACTIVITY KEY REGISTER	14	The default keys of the activity.
SPECIAL ACTION TCB POINTER	15	Pointer to the TCB associated with a user-supplied trace routine.

Figure H-1 (cont). Activity Control Block (ACB) Structure

(Table H-1 lists the contents of the activity J-base and flags word/word 2 of the ACB structure).

Table H-2 lists the contents of the activity status word (word 7 of the ACB structure).

Table H-1. Activity J-Base and Flags (Word 2 of the ACB)

Bit	Interpretation
1-7	Activity J-base sector.
8-10	Reserved for use by abort task.
11	Abort flag (restricted activities only) Bit 11 = 1; activity is being aborted.
12	Activity being loaded indicator Bit 12 = 1; activity is being loaded.
13	Open files indicator (restricted activities only) Bit 13 = 0; all open files will be preserved if activity is aborted or terminates prematurely. Bit 13 = 1; files being created will be deleted if activity is aborted or terminates prematurely.
14-15	Initial A-bank.
16	Restricted mode flag Bit 16 = 1; restricted activity.

Table H-2. Activity Status Word (Word 7 of the ACB)

Bit	Interpretation
1-2	Activity type: Bits 1 and 2 = 00_2; activity is not reusable. Bits 1 and 2 = 01_2; activity is not reusable. Bits 1 and 2 = 10_2; activity is reentrant. Bits 1 and 2 = 11_2; activity is reusable.
3	ACB status: Bit 3 = 0; ACB is not complete. Bit 3 = 1; ACB is complete.
4	Activity residency status: Bit 4 = 0; activity is disk-resident. Bit 4 = 1; activity is permanently resident in main memory.
5	Activity current memory residency status: Bit 5 = 0; activity is not currently resident in main memory. Bit 5 = 1; activity is currently resident in main memory.
6	Activity scheduling status: Bit 6 = 0; activity is not scheduled. Bit 6 = 1; activity is already scheduled. Additional scheduling requests are queued.
7	ACB residency status: Bit 7 = 0; not permanently memory-resident.
8	Reserved.
9-16	Activity request counter: This count is incremented every time the activity is requested, and decremented every time execution of the activity terminates. For a COS this field is zero.

CLOCK TASK CONTROL BLOCK (CLOCK TCB)

When a task is connected to a clock, the user must provide a clock task control block. The clock task control block is used to define and control the task connected to the clock. Figure H-2 illustrates the organization and contents of the clock TCB.

Clock TCB	Word Number	Contents
RESERVED	0	
TIMER WORD	1	Bits 1 through 4: time units (see Table H-3); bits 5 through 16: an integer number of time units from now until the activity is to be scheduled. A time interval of zero indicates that the TCB is currently inactive.
POINTER TO THE TIME-OUT ROUTINE	2	Address of the user-supplied time-out routine, which is executed after the time interval has elapsed.
USER PARAMETER 1	3	
USER PARAMETER 2	4	
CONTROL FLAGS	5	Task control flags (see Table H-4).
RESERVED	6	
ACB ADDRESS	7	Pointer to the ACB of the activity that contains the clock task; zero if this is a system task.

Figure H-2. Clock Task Control Block (Clock TCB)

Table H-3. Time Unit Values

Bit	Interpretation
1	Bit 1 = 1; use the millisecond timer.
2	Bit 2 = 1; use the half-second timer.
3	Bit 3 = 1; use the second timer.
4	Bit 4 = 1; use the minute timer.

NOTES:

1. When more than one timer is indicated, the leftmost bit will be used to determine the timer to be used. Bits 5 through 16 represent the integer number of time units from now until the task is to be executed.
2. The millisecond timer is incremented at each software clock interval (approximately 16.7 milliseconds). If the millisecond timer is specified, and the number of time units specified is not a multiple of the software clock, the number of time units is rounded down to the next lower multiple of 16.7 milliseconds.

Table H-4 lists the control flags of the clock TCB and their meaning.

Table H-4. Clock TCB Control Flags

Control Flag	Meaning
Bit 1	Bit 1 = 1; task is to be cyclic.
Bit 2	Bit 2 = 0; time-out routine is to be scheduled by OS/700. Bit 2 = 1; immediate execution of the task is required.
Bits 3-4	Unused.
Bits 5-8	Priority level of the time-out routine if scheduling is specified.
Bits 9-12	TCB type: zero implies a system request and no ACB pointer in word 7 of the clock TCB; 1 implies an activity request with an ACB pointer in word 7 of the clock TCB; 2 implies a system request on behalf of a user, and word 7 contains an ACB pointer.
Bits 13-16	Unused.

Clock TCB (After Connection to a Clock Queue)

After a clock TCB has been connected to one of the clock queues, three words of the clock TCB are modified and one word is initialized. The words modified or initialized are as follows:

Word 0 - This word used to link the clock TCB to the proper clock queue.

Word 1 - This word is converted to the internal clock manager format. The number of time units is made negative and is used as the running timer.

Word 5 - Bits 1 through 4 of the word 1 are copied into bits 13 through 16 of word 5; thus, bits 13 through 16 act as queue pointers for the clock disconnect functions.

Word 6 - If cyclic scheduling is specified (bit of word 1 is non-zero), word 6 is set to the timer reset value; otherwise, word 6 is set to zero. The timer reset value is a copy of the original negative value contained in word 1 after internal conversion.

DATE/TIME BLOCK

The date/time block contains today's date and the current time coded in ASCII. Today's date is expressed as ddmmyy; dd is the day of the month, mm is the month, and yy is the year. Current time is expressed in hours and minutes, based on a 24-hour clock; e.g., 7:00 PM is 1900. Figure H-3 shows the organization and content of the date/time block.

Date/Time Block	Word Number	Contents
HOURS	0	Current hour (00 through 23).
MINUTES	1	Current minute (00 through 59).
BLANK CHAR. BLANK CHAR.	2	
DAY	3	Current day of the month (01 through 31).
MONTH	4	Current month (01 through 12).
YEAR	5	Current year (xx through 99); xx is the year in which system configuration was performed. The current year cannot be set to earlier than the year of system configuration.

Figure H-3. Date/Time Block Structure

DEVICE CONTROL BLOCK (DCB)

The device control block (DCB) contains information required for the control of physical I/O operations. The user must create a DCB for each device to be used in physical I/O operations. Figure H-4 illustrates the organization and contents of the DCB.

Device Control Block	Word Number	Contents
RESERVED	0	
GENERIC DEVICE TYPE	1	See Appendix D, "Physical I/O Generic Device Type Assignments."
UNIT NUMBER	2	The logical unit number assigned to this I/O device.
DATA MODE	3	See Appendix C, "Physical I/O Data Mode Assignments."
USER IDENTIFIER	4	A 16-bit word identifying the user.
ADDRESS OF THE I/O STATUS BLOCK	5	Address of the I/O status block.
I/O COMPLETION RETURN ADDRESS	6	Address to which control is returned at the completion of the I/O operation.

Figure H-4. Device Control Block (DCB) Structure

QUEUE HEADER

A queue header consists of two words; the first word contains a pointer to the first entry in the queue and the second word contains a pointer to the last entry in the queue. An empty queue is indicated by a zero value in both words. Figure H-5 illustrates the organization and contents of the queue header.

User Queue Header	Word Number	Contents
FIRST ENTRY POINTER	0	Pointer to the first entry in the queue; zero if the queue is empty.
LAST ENTRY POINTER	1	Pointer to the last entry in the queue; zero if the queue is empty.

Figure H-5. User Queue Header Structure

SUSPENDED SAVE AREA

Suspended save areas are reserved for system use; the user programs should not alter or modify any information contained in a suspended save area. It is presented here to familiarize the user executive with its contents, which are used to restart interrupted tasks and to save registers through a function call. The organization and contents of a suspended save area are illustrated in Figure H-6.

Suspended Save Area	Word Number	Contents
X-REGISTER	0	X-register value at time of interrupt.
A-REGISTER	1	A-register value at time of interrupt.
KEYS	2	Value of keys at time of interrupt.
B-REGISTER	3	B-register value at time of interrupt.
P-REGISTER	4	P-register value at time of interrupt. If bit 1 = 1, the contents of the pseudoregisters have been saved.
ZCR1	5	Value in pseudoregister 1 at time of interrupt.
ZCR2	6	Value in pseudoregister 2 at time of interrupt.
ZCR3	7	Value in pseudoregister 3 at time of interrupt.
ZCR4	8	Value in pseudoregister 4 at time of interrupt.
ZCR5	9	Value in pseudoregister 5 at time of interrupt.
ZCR6	10	Value in pseudoregister 6 at time of interrupt.

Figure H-6. Suspended Save Area Structure

Suspended Save Area	Word Number	Contents
START OF QUEUE	11	Pointer to first TCB scheduled and waiting to be dispatched on this level.
END OF QUEUE	12	Pointer to last TCB scheduled and waiting to be dispatched on this level.
S-REGISTER	13	S-register value at time of interrupt.
ZAAREG	14	Value in ZAAREG at time of interrupt. (This register in a fixed memory location containing the ACB address of the activity currently executing (zero if a system task is executing). If bit 16 = 1, an executive function is being executed on behalf of the activity; if bit 1 = 1, it is a disk-resident executive function.
ZARREG	15	Value in ZARREG at time of interrupt. (This register is a fixed memory location containing the address at which return is to be made to the caller from a currently executing executive function.)
ZALREG	16	Value in ZALREG at time of interrupt. (This register is a fixed memory location containing the address of the caller's parameter list for a currently executing executive function.)
BANKS	17	Value in Bank State Register and indexing mode (32K or 64K) at time of interrupt (64K systems only, otherwise zero).
USER'S KEYS	18	User's keys saved through executive function call.
USER'S B-REGISTER	19	Value in user's B-register saved through executive function call.
USER'S S-REGISTER	20	Value in user's S-register saved through executive function call.
USER'S BANKS	21	Value in user's Bank State Register and user's indexing mode (32K or 64K) saved through executive function call (64K systems only, otherwise zero).
ZUSTK	22	Pointer to chain of register save blocks for multilevel executive function call.
ZTM1	23	Temporary to save caller's keys on multilevel executive function call.
ZTM2	24	Temporary to save caller's B-register on multilevel executive function call.
ZTM3	25	Temporary to save caller's S-register on multilevel executive function call.
ZTM4	26	Temporary to save caller's banks and indexing mode on multilevel executive function call.

Figure H-6 (cont). Suspended Save Area Structure

Suspended Save Area	Word Number	Contents
ZTM5	27	Temporary to save caller's parameter list address on multilevel executive function call.
	28	Reserved.
	29	Reserved.

Figure H-6 (cont). Suspended Save Area Structure

TASK CONTROL BLOCK (TCB)

Each task executed under the control of OS/700 is defined by a task control block (TCB). The TCB contains information required for the control of the task. The organization and contents of the TCB are shown in Figure H-7.

Task Control Block	Word Number	Contents
LINK WORD	0	Used for queuing scheduled TCB's.
RESERVED	1	See SAC$ function description.
TASK ENTRY ADDRESS	2	If bit 1 = 0; enter task by a JMP. If bit 1 = 1; enter task by a JST.
USER PARAMETER 1	3	
USER PARAMETER 2	4	
CONTROL WORD	5	See Table H-5.
RESERVED	6	See SAC$ function description.
ACB ADDRESS	7	Pointer to the ACB of the activity which contains the task. If this word is zero, the task is a system task.

task [illegible]
[illegible]

Figure H-7. Task Control Block (TCB) Structure

Table H-5 describes the fields within the task control block control word (word 5 of the TCB).

Table H-5. Task Control Block Control Word

Bit	Meaning
1-2	Reserved for OS/700 use.
3	Canned TCB flag; indicates that TCB is not returnable to free memory.
4	Schedule interlock flag.
5-8	Task priority level.
9-12	TCB type: 0 implies a system request and no ACB pointer in word 7. 1 implies a user request, and word 7 contains an ACB pointer. 2 implies a system request on behalf of a user, and word 7 contains an ACB pointer.
13-16	Reserved for OS/700 use.

Honeywell Bull

Technical Publications Remarks Form* *(please print)*

Title:

SYSTEM 700 EXECUTIVE OS/700

Order: 61A.2-AR22,Rev.0

Dated: JULY 1975

Errors in publication:

Suggestions for improvement to publication:

from : Name

Company

Title

Address

* Your comments will be promptly investigated by appropriate technical personnel, action will be taken as required, and you will receive a written reply. If you do not require a written reply, please check here : □

CUT ALONG LINE

Please hand this technical publication remark form
to your Honeywell Bull representative,

or mail to :

Honeywell Bull

Marketing Communications
Documentation/Publications

94, avenue Gambetta
75960 PARIS CEDEX 20 - FRANCE

Honeywell Bull

HONEYWELL INFORMATION SYSTEMS

Afotek 01-76 (615)