CRAY RESEARCH, INC.

CRAY-1® COMPUTER SYSTEMS

COS
EXEC/STP/CSP
INTERNAL REFERENCE
MANUAL

SM-0040

PUBLICATION CHANGE NOTICE

October, 1980

TITLE: COS EXEC/STP/CSP Internal Reference Manual

PUBLICATION NO. SM-0040 **REV.**

This manual supports COS Version 1.09 and obsoletes portions of the CRAY-OS Version 1 System Programmer's Manual, publication 2240012.

CRAY RESEARCH, INC.

CRAY-1® COMPUTER SYSTEMS

COS
EXEC/STP/CSP
INTERNAL REFERENCE
MANUAL

SM-0040

Copyright© 1980 by CRAY RESEARCH, INC. This manual or parts thereof may not be reproduced in any form without permission of CRAY RESEARCH, INC.

RECORD OF REVISION

CRAY RESEARCH, INC.

PUBLICATION NUMBER SM-0040

Each time this manual is revised and reprinted, all changes issued against the previous version in the form of change packets are incorporated into the new version and the new version is assigned an alphabetic level. Between reprints, changes may be issued against the current version in the form of change packets. Each change packet is assigned a numeric designator, starting with 01 for the first change packet of each revision level.

Every page changed by a reprint or by a change packet has the revision level and change packet number in the lower righthand corner. Changes to part of a page are noted by a change bar along the margin of the page. A change bar in the margin opposite the page number indicates that the entire page is new; a dot in the same place indicates that information has been moved from one page to another, but has not otherwise changed.

Requests for copies of Cray Research, Inc. publications and comments about these publications should be directed to:

CRAY RESEARCH, INC.,
1440 Northland Drive,
Mendota Heights, Minnesota 55120

Revision	Description
	October, 1980 - Original printing; supports COS Version 1.09. This manual obsoletes portions of the CRAY-OS Version 1 System Programmer's Manual, publication 2240012.

PREFACE

This manual describes the internal features of the EXEC, STP, and CSP portions of the CRAY-1 Operating System.

This publication is part of a set of manuals that describes the internal design of the CRAY-1 Operating System and its product set.

Other publications in this set are:

SM-0042	COS Front-End Protocol Internal Reference Manual
SM-0043	COS Operational Procedures Reference Manual
SM-0044	COS Operational Aids Reference Manual
SM-0045	COS Table Descriptions Internal Reference Manual
SM-0046	IOS Software Internal Reference Manual
SM-0049	DGS Internal Reference Manual
SM-0050	COS Simulator (CSIM) Reference Manual

The following, which are available for use only by Cray Research personnel, complete the set of software maintenance documentation.

SM-0017	FORTRAN (CFT) Internal Reference Manual
SM-0041	COS Product Set Internal Reference Manual

Manuals designated as internal describe the internal design of the software, whereas the other manuals in the set define procedures and external features of tools needed for installing and maintaining CRI software.

The reader is assumed to be familiar with the contents of the CRAY-OS Version 1 Reference Manual (SR-0011) and to be experienced in coding the CRAY-1 Assembly Language (CAL) as described in the CAL Version 1 Reference Manual (SR-0000). In addition, the I/O Subsystem assembler language (APML) is described in the APML Reference Manual (SR-0036).

Operating information is available in the following publications:

SG-0006	Data General Station (DGS) Operator's Guide
SG-0051	I/O Subsystem (IOS) Operator's Guide

CONTENTS

FIGURES

TABLES

TABLES continued

INTRODUCTION 1

1.1 GENERAL DESCRIPTION

CRAY-OS (COS) is a multiprogramming operating system for the CRAY-1 Computer System. The operating system provides for efficient use of system resources by monitoring and controlling the flow of work presented to the system in the form of jobs. The operating system centralizes many of the job functions such as input/output and memory allocation and resolves conflicts when more than one job is in need of resources.

CRAY-OS is a collection of programs that, following startup of the system, resides in CRAY-1 Central Memory, on system mass storage, and in the I/O Subsystem on some models of the CRAY-1 S Series. (Startup is the process of bringing the CRAY-1 and the operating system to an operational state.)

Jobs are presented to the CRAY-1 by one or more computers referred to as front-end systems, which may be any of a variety of computer systems. Since a front-end system operates asynchronously under control of its own operating system, software executing on the front-end system is beyond the scope of this publication.

The FORTRAN compiler, the CAL assembler, the SKOL macro translator, the UPDATE program, and utility programs execute as parts of user jobs and are described in separate publications.

The operating system is available in two forms: (1) preassembled into absolute binary programs in an unblocked format and (2) source language programs in the form of UPDATE decks.

The binary form of the program is provided for the installation of the basic system. The UPDATE decks provide a means of modifying and updating the source code and generating a new system in binary form by reassembling the modified programs.

Details for generating, installing, and starting up the operating system are given in COS Operational Procedures Reference Manual, CRI publication SM-0043.

1.2 HARDWARE CHARACTERISTICS

This section briefly summarizes the hardware characteristics of the CRAY-1 Computer System. The basic components of the system are summarized in table 1-1 and illustrated in figures 1-1 and 1-2. Figure 1-1 illustrates basic components of a CRAY-1A/B or CRAY-1 Model S/250, S/500, or S/1000 Computers. These systems consist of a central processing unit (CPU), power and cooling equipment, a minicomputer maintenance control unit (MCU), a mass storage disk subsystem, and a front-end system.

Table 1-1. Characteristics of Models of the CRAY-1 Computer Systems

Model	S/250	S/500 or 1/B	S/1000 or 1/A	S/1200	S/1300	S/1400	S/2200	S/2300	S/2400	S/4200	S/4300	S/4400
CPU												
Memory size in 64-bit words	1/4M	1/2M	1 M	1 M	1 M	1 M	2 M	2 M	2 M	4 M	4 M	4 M
FRONT-END INTERFACES	1-3	1-3	1-3	1-3	1-3	1-3	1-3	1-3	1-3	1-3	1-3	1-3
I/O SUBSYSTEM												
I/O Processors				2	3	4	2	3	4	2	3	4
Buffer Memory				.5 or 1M	.5 or 1M	.5 or 1M	.5 or 1M	.5 or 1M	.5 or 1M	.5 or 1M	.5 or 1M	.5 or 1M
DCU-4 Controllers				1-4	1-8	1-12	1-4	1-8	1-12	1-4	1-8	1-12
DD-29 Disk Storage Units				2-16	2-32	2-48	2-16	2-32	2-48	2-16	2-32	2-48
MASS STORAGE SUBSYSTEM§												
DCU-3 Disk Controllers	2-8	2-8	2-8	2-8§§	2-8§§	2-8§§	2-8§§	2-8§§	2-8§§	2-8§§	2-8§§	2-8§§
DD-29 Disk Storage Units	2-32	2-32	2-32	1-32§§	1-32§§	1-32§§	1-32§§	1-32§§	1-32§§	1-32§§	1-32§§	1-32§§
DCU-2 Disk Controllers		2-8	2-8									
DD-19 Disk Storage Units		2-32	2-32									

NOTES:

§ Mass storage limits assume a configuration with a maximum of 8 channels available.

§§ While connection of mass storage devices through the I/O Subsystem is preferred, where possible, available CPU channels can be used for additional mass storage.

Figure 1-2 illustrates the CRAY-1 S Series Models S/1200 through S/4400 Computer Systems. These systems are characterized by the incorporation of an I/O Subsystem comprised of two to four I/O Processors.

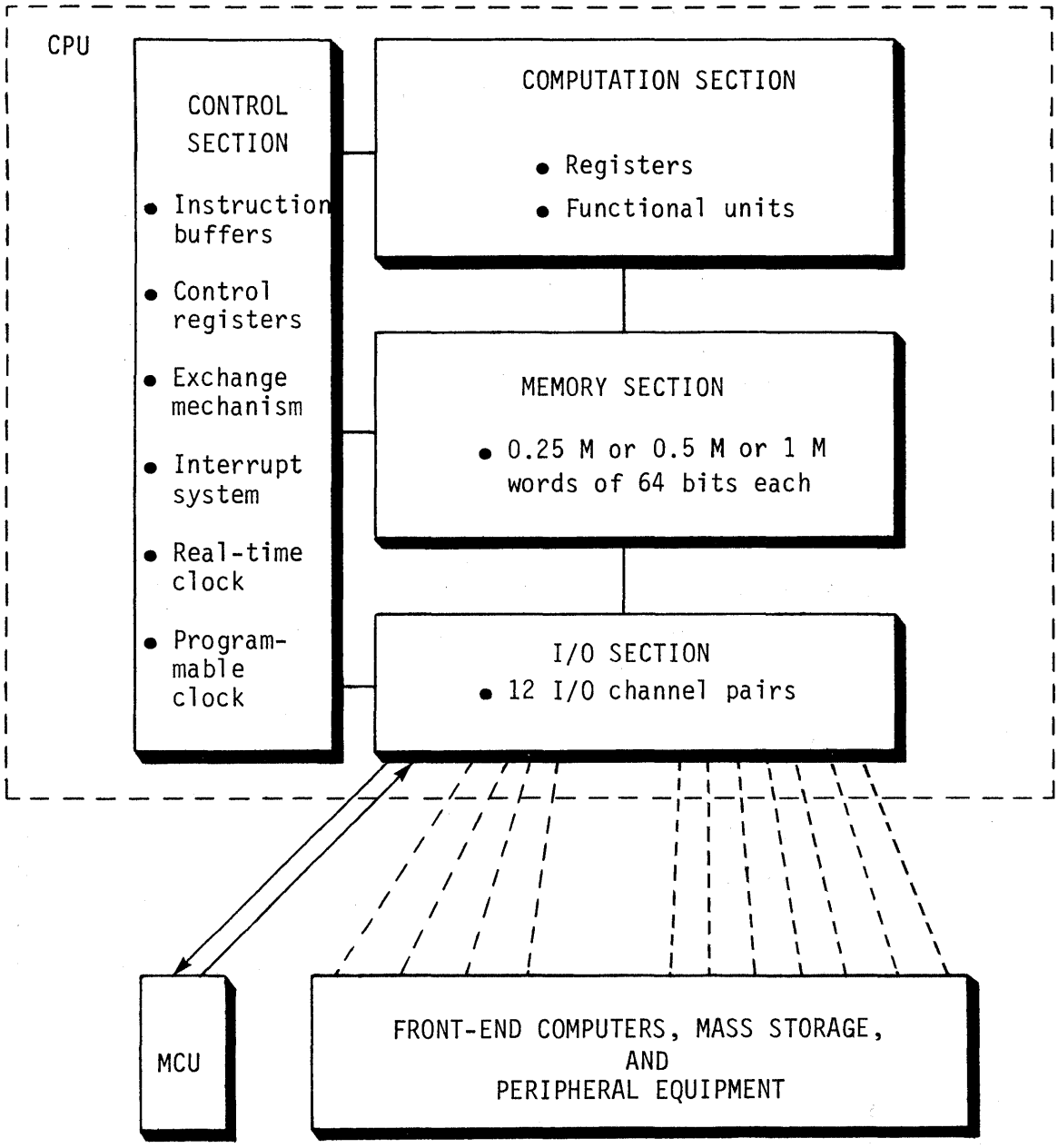


Figure 1-1. CRAY-1A/B or CRAY-1 S Series, Model S/250, S/500 or S/1000 Computer Systems

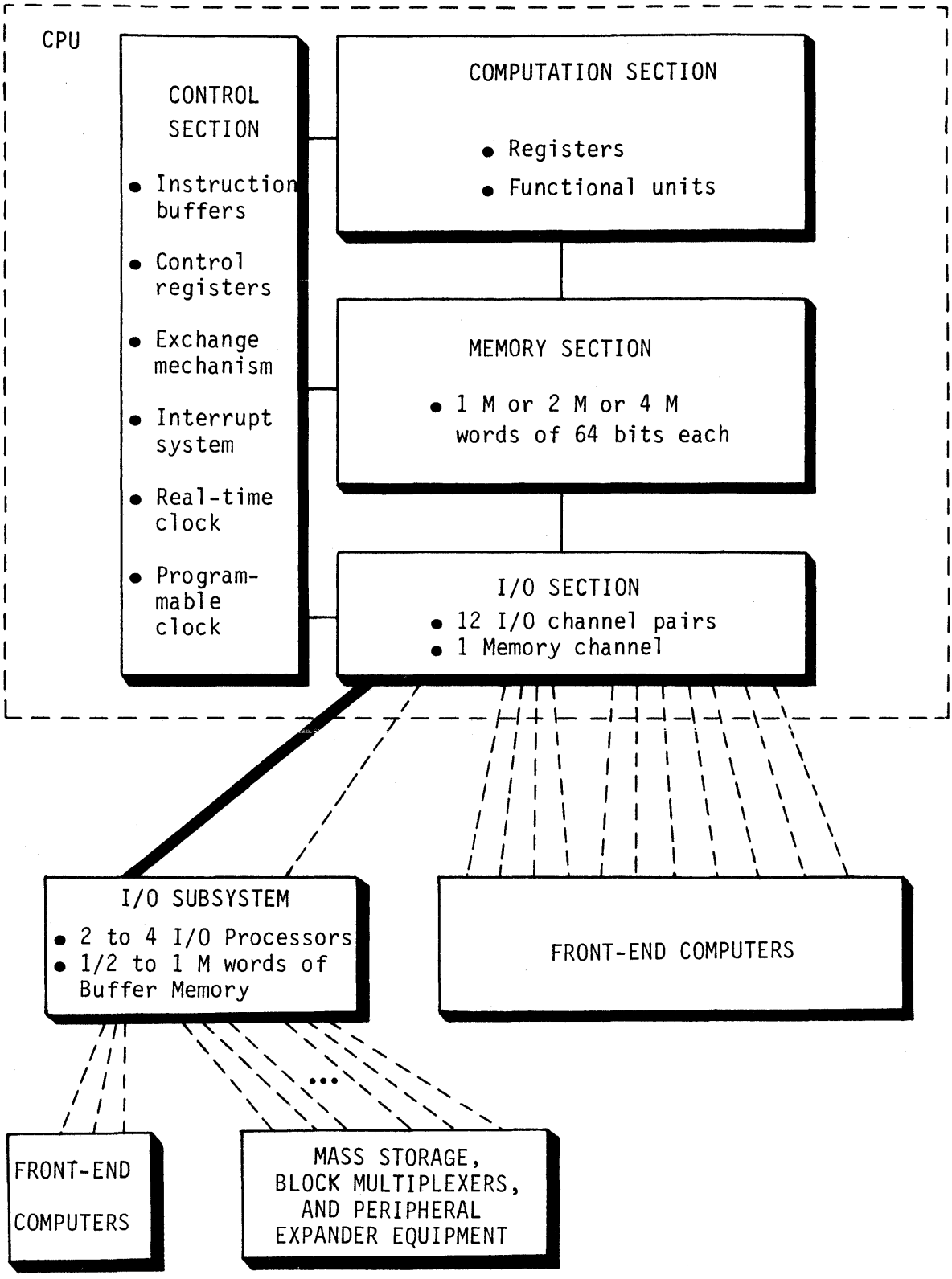


Figure 1-2. CRAY-1 S Series Model S/1200 through S/4400 Computer Systems

1.2.1 COMPUTATION SECTION

The computation section is composed of instruction buffers, registers, and functional units that operate together to execute sequences of instructions. At any one time, only one program can be in execution although several programs may be candidates for execution. This means that multiprogramming, the sharing of the computation section among multiple programs, is possible; but multiprocessing, the concurrent execution of multiple programs, is not possible.

1.2.2 CENTRAL MEMORY SECTION

The CRAY-1 Central Memory is constructed of LSI chips arranged in 8 or 16 banks. Memory sizes depend on the model. Available sizes are: 262,144 words, 524,288 words, 1,048,576 words, 2,097,152 words, or 4,194,304 words. A word is 64 bits.

The lower memory addresses contain exchange packages, operating system tables and pointers, and operating system programs. The extreme upper memory addresses contain operating system I/O buffers. The remainder of memory is available for user jobs.

An algorithm that calculates the maximum memory size allocation for a job is described in Appendix B of the COS Operational Procedures Reference Manual, publication SM-0043.

1.2.3 MEMORY PROTECTION

Two registers (BA and LA) define the field of memory addresses that can be referenced by the executing program (see figure 1-3). The base address (BA) register contents define the beginning address; the limit address (LA) register contents define the upper address. The last usable address is at $(LA)x2^4-1$.

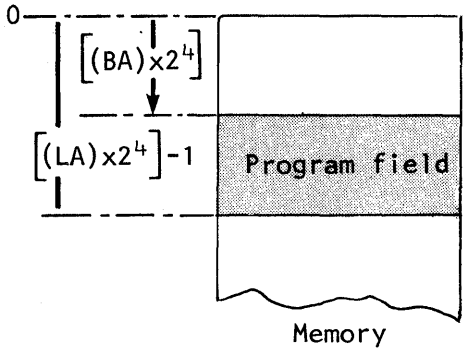


Figure 1-3. Program field

The hardware senses an attempt by a program to reference an address not in this range of addresses and sets an error interrupt flag. Note that both BA and LA addresses are with reference to absolute address 0.

Some of the operating system programs are privileged by having access to all of memory; others are limited to certain portions of the operating system and to user program areas. Each user program has access to its own defined field only.

1.2.4 MASS STORAGE

CRAY-1 mass storage consists of one or more Cray Research DCU-2 or DCU-3 Disk Controllers for CRAY-1A/B Systems or CRAY-1 S/250, S/500 or S/1000 Systems and multiple DD-19 or DD-29 Disk Storage Units (DSUs). The disk controller is a Cray Research product and is implemented in ECL logic similar to that used in the mainframe. Each controller may have up to four DD-19 or DD-29 disk storage units attached to it. Operational characteristics of the DSUs are summarized in table 1-2. (The DD-29 resembles the DD-19, except that it has approximately twice the storage capacity of the DD-19.) Additional information about the CRAY-1 mass storage subsystem is given in the CRAY-1 DCU-2, DCU-3 Disk Controller Reference Manual, publication 2240630.

For the CRAY-1 S/1200 and above, the mass storage is attached to the I/O Subsystem. The I/O Subsystem consists of 2 or more I/O processors. One of these serves as the Master I/O Processor. A second processor (the Buffer I/O Processor or BIOP) is dedicated to mass storage. The other two processors may be dedicated to mass storage. If they are, they are referred to as Data I/O Processors (DIOP). Each BIOP or DIOP can drive up to four DCU-4 Disk Control Units. Each DCU-4 Disk Control Unit supports up to four disk storage units. All units connected to a DCU-4 may be simultaneously active. However, the number of concurrent data streams is limited by the Buffer Memory size, the Buffer I/O Processor (BIOP) transfer capacity, and software overhead. For example, a Model S/x200 might be limited to 6 streams while a larger system could have as many as 12 streams.

1.2.5 I/O SUBSYSTEM

Starting with the S/1200 (1 million words), I/O throughput to front-end computers and to mass storage devices is significantly enhanced with the incorporation of an I/O Subsystem. The I/O Subsystem is a Cray Research product specifically designed to complement the CRAY-1 CPU requirements. A primary feature is the incorporation of a Memory Channel linking the I/O Subsystem to Central Memory. Maximum transfer rates of approximately 850 Mbits per second are achievable on this channel. The power of the I/O Subsystem relates directly to the number of I/O Processors it contains. Two, three, or four I/O Processors may comprise the I/O Subsystem. With each addition of another I/O Processor, significant increases in mass storage capacity or the ability to drive peripheral devices is achieved.

Table 1-2. Operational characterstics of disk storage units

	DD-19	DD-29
Word capacity per drive	3.723×10^7 7.585×10^7	
Word capacity per cylinder	92,160	92,160
Bit capacity per drive	2.424×10^9	4.854×10^9
Tracks per surface or cylinders per drive	404 (411 less 7 cylinders reserved for diagnostics)	814 (823 less 9 cylinders reserved for diagnostics)
Sectors per track	18	18
Bits per sector	32,768	32,768
Number of head groups	10	10
Latency	16.7 ms	16.7 ms
Access time	15 - 80 ms	15 - 80 ms
Data transfer rate (average bits per second)	35.4×10^6	35.4×10^6
Longest continuous transfer per command	92,160 words (1 cylinder)	92,160 words (1 cylinder)
Total bits that can be streamed to a unit (disk cylinder capacity)	5.9×10^6	5.9×10^6

1.2.6 FRONT-END COMPUTER SYSTEMS

The CRAY-1 Computer System may be equipped with one or more front-end computer systems that provide input data to the CRAY-1 and receive output from the CRAY-1 for distribution to a variety of slow-speed peripheral equipment. Peripherals attached to the front-end system vary with application requirements (i.e., local or remote job entry stations, data concentrator for multiplexing remote stations, etc.). On CRAY-1 Models S/1200 and above, the front-end computers are usually connected through the I/O Subsystem. Front-end systems connect directly to the CPU I/O channels on systems that do not have I/O Subsystems.

The CRAY-1 is interfaced to front-end systems through special interface controllers that compensate for differences in channel widths, machine word size, electrical logic levels, and control protocols. The interface controller is a Cray Research product implemented in logic compatible with the host system.

CRAY-1 front-end systems connect directly to the CPU I/O channels on systems that do not have I/O Subsystems. On Models S/1200 and above, the front-end computers normally connect through the I/O Subsystem but may also be connected to the CPU I/O channels.

1.2.7 MAINTENANCE CONTROL UNIT (MCU)

On CRAY-1A/B Systems and Models S/250, S/500, and S/1000 Systems, a Data General minicomputer serves as a maintenance control unit. The MCU performs initial system startup and recovery for the operating system. Included in the MCU system is a software package that enables the minicomputer to monitor CRAY-1 performance during production hours. When not used for maintenance purposes, the MCU can serve as a front-end system for the CRAY-1 by employing CRI-supplied software.

A description of the software for the MCU is beyond the scope of this publication.

1.2.8 PERIPHERAL EXPANDER

On CRAY-1 models S/1200 through S/4400, peripheral devices connected to the I/O Subsystem through a Peripheral Expander interface allow for maintenance operations such as initial system startup and recovery.

1.3 SOFTWARE CONFIGURATION

The CRAY-1, as with any other computer system, requires three types of software: an operating system, language systems, and applications programs. The I/O Subsystem, when present, also requires its own software. The internal features of the I/O Subsystem Software are described in the IOS Software Internal Reference Manual, publication SM-0046.

1.3.1 CRAY-1 OPERATING SYSTEM (COS)

The CRAY-1 Operating System (COS) consists of memory-resident and disk resident programs that (1) manage resources, (2) supervise job processing, and (3) perform input/output operations. COS also contains a set of disk resident utility programs. The operating system is activated through a system startup operation performed from the MCU or the I/O Subsystem. A job may consist of a compilation or assembly of a program written in some source language such as FORTRAN, followed by execution of the program resulting from the compilation or assembly.

The CRAY-1 Operating System consists of the following modules that execute on the CPU (figure 1-4):

Executive (EXEC)
System Task Processor (STP)
Control Statement Processor (CSP)
Utility programs (not shown)

EXEC (described in section 2) runs in monitor mode and is responsible for control of the system. It schedules STP tasks, manages exchange packages, performs I/O, and handles all interrupts. EXEC has access to all of memory.

STP (described in section 3) runs in object program (user) mode. It accesses all memory other than that occupied by EXEC and is responsible for processing all user requests. STP is composed of a number of programs known as tasks, each of which has its own exchange package.

CSP (described in section 5) is responsible for interpreting all job control statements and either performing the requested function or making the appropriate system request. An image of CSP is resident after the STP area of memory but is copied into a user field for execution.

Utility programs (described in the COS Product Set Internal Reference Manual) include the loader, a library generation program (BUILD), a source language maintenance program (UPDATE), permanent dataset utility programs, copy and positioning routines, and so on.

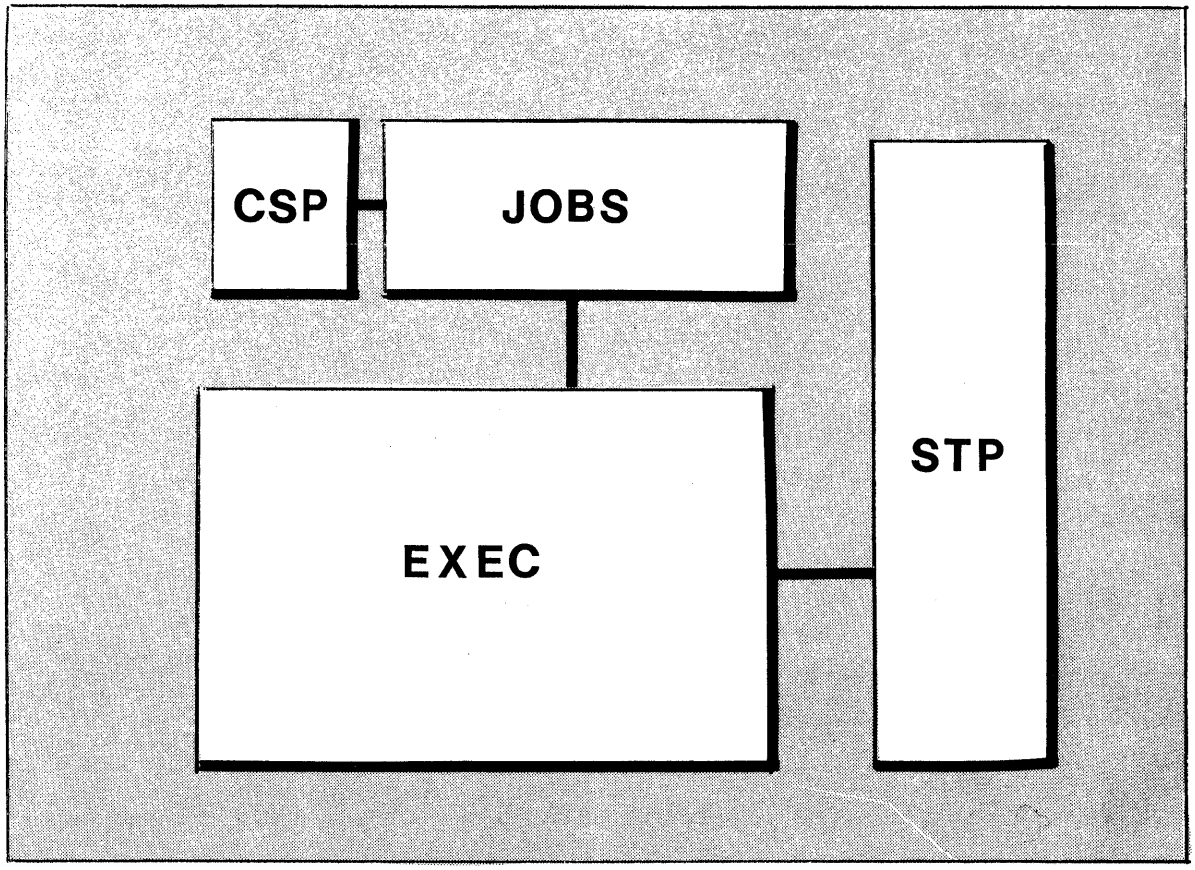


Figure 1-4. Elements of CRAY-OS

Images of utility programs are resident on disk storage and are summoned for loading and execution in the user field through control statements.

1.3.2 LANGUAGE SYSTEMS

Currently, four language systems developed by Cray Research are available for use on the CRAY-1. They are the FORTRAN compiler (CFT), the CRAY-1 Assembler language program (CAL), the SKOL macro translator, and A Programming Macro Language (APML) for the I/O Subsystem.

FORTRAN compiler

Developed in parallel with the CRAY-1 Computer System, the Cray Research FORTRAN compiler is designed to take advantage of the vector capability of the computer.

The compiler itself determines the need for vectorizing and generates code accordingly, removing the burdens of such considerations from the programmer. Optimizing routines examine FORTRAN source code to see if it can be vectorized. The compiler adheres closely to the ANSI 1966 standards and includes many ANSI 1978 extensions.

A description of the design of the compiler is outside the scope of this publication. It is included in the CRAY-1 FORTRAN (CFT) Internal Reference Manual, CRI publication SM-0017 which is distributed only to CRI personnel.

CAL assembler

The CAL assembler provides users with a means of expressing symbolically all hardware functions of the CPU. Augmenting the instruction repertoire is a set of versatile pseudo instructions that provide users with options for generating macro instructions, organizing programs, and so on. Programs written in CAL may take advantage of Cray Research-provided system macros that facilitate communication with the operating system. CAL enables the user to tailor programs to the architecture of the CRAY-1. Much of the operating system as well as other software provided by Cray Research is coded in CAL assembly language.

A description of the design of the CAL assembler is beyond the scope of this publication. See The CRAY-1 CAL Assembler Language Reference Manual, publication SR-0000 for assembler information.

APML assembler

The APML assembler executes on the CRAY-1 CPU and generates absolute code that is executable in the CRAY-1 I/O Processors. APML allows the system programmer to express symbolically all hardware functions of a CRAY-1 I/O Processor. It is used to generate the I/O Subsystem software.

In addition to the full range of symbolic instructions, which allow the APML user to fully use the I/O Processors arithmetic and I/O instructions, registers, and memory, APML provides a number of macro, conditional assembly, and pseudo instructions that simplify the task of creating assembly language programs.

APML is described in the APML Reference Manual, publication SR-0036.

SKOL macro translator

SKOL is a high-level programming language that stresses readability and extensibility. It offers the user a well structured language while retaining the power and efficiency of the CFT compiler. This is possible because SKOL is translated into FORTRAN code by a set of string-processing macro instructions. By adding to these instructions, the user can extend the language to suit his own purposes. By inserting macros directly into the SKOL source program, changes in the languages can be defined for a specific run.

Many of the control statements are familiar to users of other high-level languages. For example, SKOL's IF-ELSEIF-ELSE-ENDIF structure is derived from LISP and ALGOL, and its LOOP-WHILE-ENDLOOP subsumes all single-exit loop structures. The scalar case structure is derived from Pascal. The important situation case structure, which eliminates the need for labels and GOTOs, is unique to SKOL.

The use of the record and pointer data structures in SKOL also largely parallels Pascal. Character string processing is performed in SKOL with the STRING data structure, and partial-word variables can be defined by the WORD structure. The user can also define his own enumerated data types.

Since any valid FORTRAN code is also valid SKOL code, SKOL makes use of the subroutine and the function. Additionally, SKOL offers routines without parameters, recursive routines, and the concept of a process. A process consists of several cooperating coroutines that can activate one another or suspend the process.

SKOL provides a number of tools for testing and debugging programs. Among the tools are:

- Conditional compilation, which specifies a statement, part of a statement, or a series of statements to be either compiled or not compiled, as determined by the user for a specific run.
- The TRACE statement, which prints the value of a variable whenever an assignment is made to it.
- The VALIDATE statement, which enables or disables the output of built-in run-time debugging messages.

1.3.3 LIBRARY ROUTINES

The CRAY-1 software includes a group of subprograms that are callable from CAL and CFT programs. These subprograms reside in the $FTLIB, $SYSLIB, and $SCILIB libraries. They are grouped by UPDATE deck name within each library. The subprograms have been divided among the three libraries generally on a functional basis.

$FTLIB contains routines that are an intrinsic part of CFT, such as the mathematical functions. All of the basic external functions as specified by ANSI X3.9-1966 are incorporated in the library. Additionally, a large number of vector FORTRAN library routines are also provided. $FTLIB also contains nonmathematical routines such as the DATE routine.

$SYSLIB routines, which link directly to the operating system, are not usually accessible from a CFT program but are callable from $FTLIB routines for specific tasks. In general, $SYSLIB serves as a link between the general-purpose $FTLIB routines and the details of COS.

The routines in $SCILIB usually perform mathematics in the scientific process such as matrix multiply or Fourier transformation.

1.3.4 APPLICATIONS PROGRAMS

Applications programs are specialized programs usually written in a source language such as FORTRAN to solve particular user problems. These programs are generally written by customers and as such are not described in this publication.

1.4 SYSTEM RESIDENCE

This section describes the locations of the various components of the operating system without attempting to explain what they are. The components are described in later sections. The system components reside in areas of memory as defined during startup (section 4.1).

Figure 1-5 illustrates the general contents of memory following startup. Figure 1-6 illustrates the general layout of a user area. Figure 1-7 itemizes the memory resident portions of the operating system.

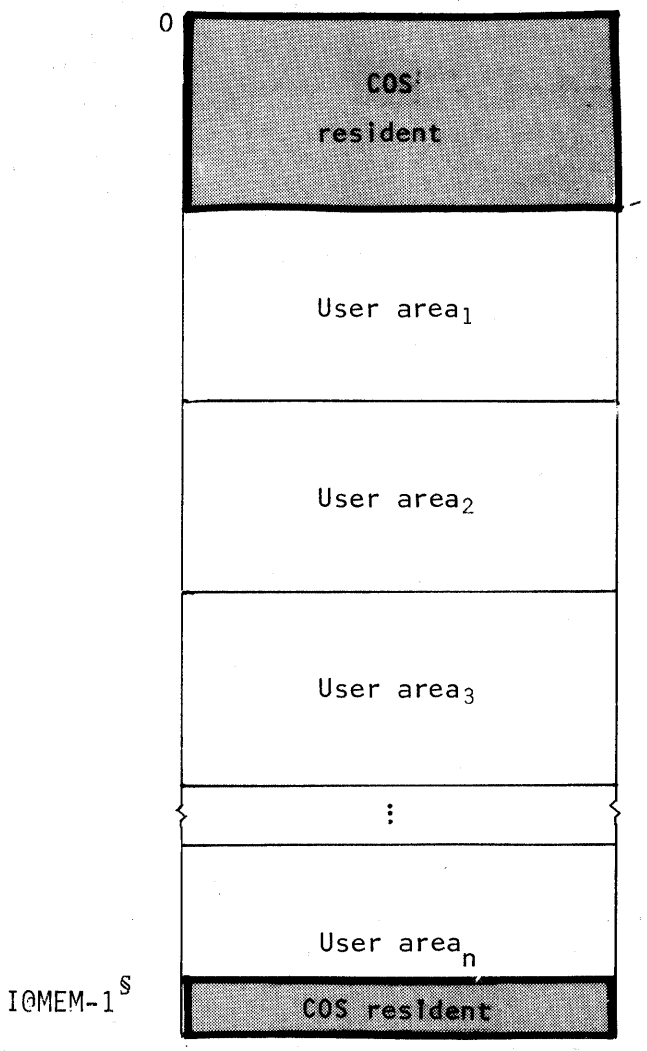


Figure 1-5. Memory Assignment

§ Installation parameter that defines maximum memory in words

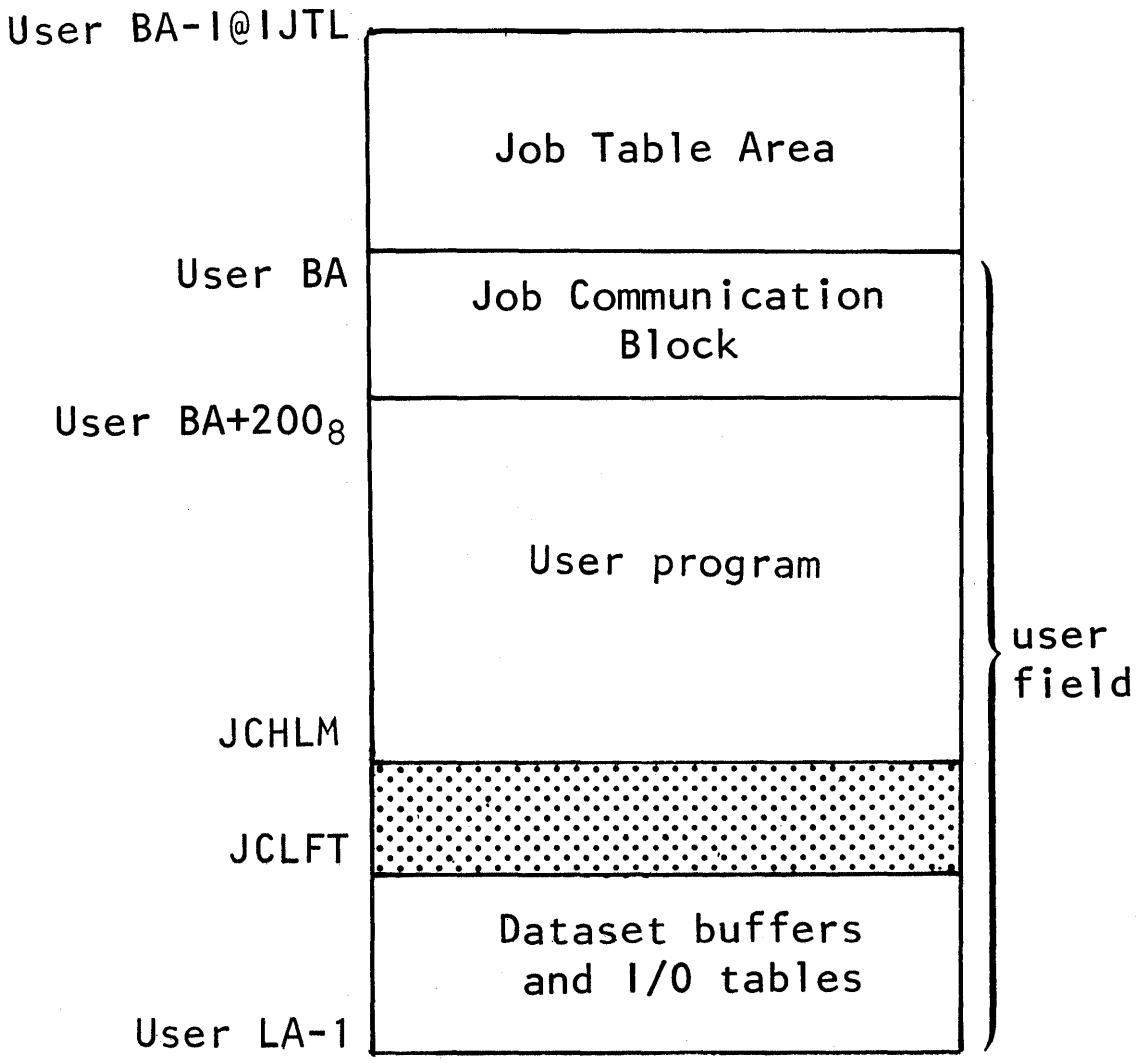


Figure 1-6. Expansion of a user area

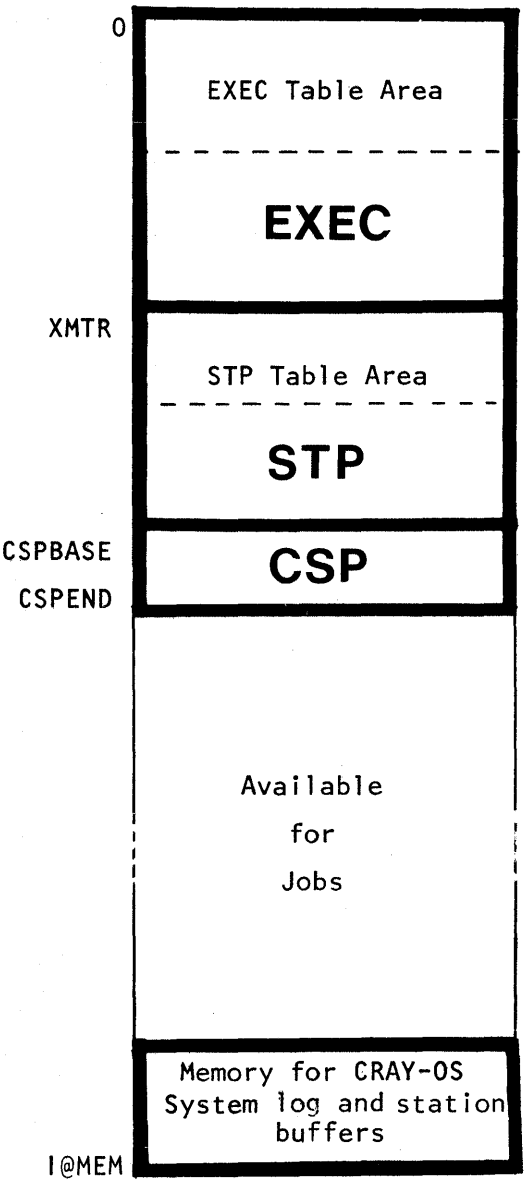


Figure 1-7. Expansion of COS resident

1.4.1 EXEC TABLE AREA

The EXEC table area contains the following tables and parameters used by EXEC. Detailed descriptions of the tables are given in COS Table Descriptions Internal Reference Manual, publication SM-0045.

IC	Channel interrupt counters
--	Miscellaneous pointers and constants
XMELIM	Logged single-bit error limit
XMECNT	Single-bit error count
XMEDIS	Single-bit interrupt disabled flag
SAXP	Pointer to currently connected user job
SAEF	Error flags from current exchange package
SUXC	User exchange package in JTA flag
NCAERR	Count of channel address errors on disk channels
SXBF	Current user exchange package
IDXP	Idle exchange package
CORXP	Correction exchange package
MCCCNT	Disk master clear count
MCLCNT	Disk master clear loop count
SIDLE	Alternate scheduling flag
SERRLIM	Station input error retry limit
DSLWA	1+LWA of COS binary and parameter file
--	Miscellaneous pointers and constants
CRT	Disk channel reservation table
SSB0	Contents of B0 register of interrupted processor
SXTC	Clock at beginning of interrupt
ETIM	Accrued executive time
ITIM	Accrued idle time
UTIM	Accrued user time
BTIM	Accrued system I/O blocked time
RUNTIM	Accrued time since deadstart
MSLIM	Idle memory scan limit
STT	System Task Table. This table consists of three parts: a 4-word header, a task parameter word area, and an exchange package area. The sign bit of the second word of the STT header is set if the highest priority STP task is to execute. The address in the low-order bits of the word points to the parameter word for the task to be executed. The third header word contains a bit for each STP task. The bit is set if the task is created. This word also contains a pointer to the exchange package for the currently scheduled STP task. The fourth word contains a breakpoint flag.
CHT	Channel Processor Table. This table contains a 1-word entry for each side (input and output) of a physical channel and a pseudo channel. An entry contains a pointer to the channel message buffer for the channel-assigned task ID and the address of the channel processor assigned to the side of the channel. Input sides are assigned even numbers; output sides odd numbers.

CBT	Channel Buffer Table. This table contains one entry of working storage for each disk driver channel.
AET	Assigned Equipment Table. This table points to entries in the PUT based on channels.
---	T11 tables: T11CSW, T11LID, T11LIDC, T11CHUN
PUT	Physical Unit Table. This table contains one entry of working storage for each disk drive on the system.
MTCT	Executive Request Table. This jump table contains a 1-word entry for each executive request that can be made by a task. The entry consists of the address of the routine that processes the request.
DOFS	A set of constants used by the system
SMSC	1 ms in 12.5 ns counts
SSEC	1 second in 12.5 ns counts
SHMS	ASCII time in hours, minutes, and seconds
SMDY	ASCII date in month, day, and year
TBPT	Task Breakpoint Table
PERT	Parity Error Table
DBF	History trace buffer
SCT	Subsystem control table
CXT	Channel extension table

1.4.2 EXEC PROGRAM AREA

Included in the System Executive (EXEC) occupied area are interrupt handlers, channel processors, task scheduler, the drivers (disk, I/O Subsystem, and front-end), system interchange, request processors, and debug aids. EXEC has a BA of 0 and an LA equal to the installation parameter I@MEM.

1.4.3 STP TABLE AREA

This area contains tables accessible to all STP tasks (not necessarily in the order noted).

AUT	Active User Table. It contains an entry for each interactive user that is logged on.
CMCC	Communication Module Chain Control. This table controls task-to-task communication. It is a contiguous area containing an entry for each combination of tasks possible within the system. The CMCC is arranged in task number sequence. The IDs of the requesting task and requested task determine the appropriate CMCC entry.

CMOD Communications Modules. These are groups of six words each that form a pool from which they are allocated as needed. Two words are used as control; two are used as input registers; and two are used as output registers. A task receives all of its requests and makes all of its replies through a CMOD.

CSD Class Structure Definition. CSD contains the job class structure. For each class defined in the structure, there is a class map; these appear in CSD in descending order. A header precedes the class maps. Variable length characteristic expressions for each class follow the maps.

DAT Dataset Allocation Table. There is a DAT for each dataset known to the system that defines where the dataset logically resides on mass storage, that is, on which logical device(s) and what portion of a device.

DCT Device Channel Table. The DCT serves as a link between the channel and the EQT. It is used by the disk driver to report completion of I/O and to report disk status.

DET Device Error Table. The DET is used to build messages for the system log.

DRT Device Reservation Table. There is a DRT for each device known to the system. The DRT contains a bit map showing available and reserved tracks on the device.

ECT Error Code Table. This table controls abort and reprieve processing done by UEP. It contains a 1-word entry for each system error code and is defined using the ERDEF macro.

EQT Equipment Table. The EQT contains an entry for each device known to the system.

IBT Interactive Buffer Table. It manages the Interactive Buffer Pool Table.

JXT Job Execution Table. The JXT contains an entry for each job that has begun processing. The table is used to control all active jobs in the system and may contain from 0 to 63 entries. A 64th entry is reserved to represent the operating system, itself.

LCT Link Configuration Table. It contains an entry for each front end connected to physical channels.

LIT Link Interface Table. SCP assigns an LIT entry at deadstart to each channel used for interface communications.

LST — Link Interface Stream Table. Eight input stream and eight output stream LSTs are contained within each LXT as used by SCP.

LXT — Link Interface Extension Table. An LXT entry is assigned by SCP to an active LIT entry for each front-end ID at LOGON and deassigned at LOGOFF. The LXT contains SCP working storage and input and output LSTs.

MST — Memory Segment Allocation Table. The MST contains an entry for each segment of memory that has been allocated by JSH as well as an entry for each free segment. It may contain from 1 to 127 1-word entries.

PDI — Permanent Dataset Information Table. This table contains information used by the Permanent Dataset Manager, such as the number of overflow and hash pages.

PDS — Permanent Dataset Table. The PDS table consists of a 1-word header followed by a 1-word entry for each active permanent dataset. The entry indicates how a dataset is accessed and if multiple access exists. If so, the entry tells how many users are accessing the dataset.

RJI — Rolled Job Index Table. For each defined JXT entry, the RJI Table contains an entry that describes the job assigned to the JXT entry and controls the recovery of jobs from mass storage.

RQT — Request Table. This table is used to queue transfer requests for disk management.

QAT — Queued Dataset Table. This table describes the multitype attributes for a dataset that has been disposed. It is managed by PDM and EXP. The number of entries in the QDT must equal the SDT entry count.

SDR — System Directory. This area contains a Dataset Name Table (DNT), section 1.4.6, for each of the datasets comprising the system library. The SDR is initialized after a system Startup.

SDT — System Dataset Table. This table contains an entry for each dataset spooled to or from a front-end system.

STPDD — STP Dump Directory. This area contains pointers to task origins, buffers, etc. An entry gives a mnemonic in ASCII plus the relative STP address for the area.

Details of the STP tables are given in the COS Table Descriptions Internal Reference Manual, publication SM-0045.

1.4.4 STP PROGRAM AREA

The System Task Processor (STP) consists of tasks and re-entrant code common to all of the tasks. Tasks cannot access the memory area occupied by EXEC but may access the rest of memory.

Although tasks are loaded into memory during Startup, they are recognized only through an Executive create-task request (usually issued by the Startup task). The Startup task is a special case since it executes only when the system is started up and is created by EXEC itself. Recovery of rolled-out jobs executes as a portion of the Startup task rather than as a separate task. STP is described further in section 3.

1.4.5 CSP AREA

A prototype of the Control Statement Processor (CSP) is maintained in memory following STP. This program is copied into each user program field where it executes each time the job requires interpretation of a control statement.

CSP is further described in section 5.

1.4.6 USER AREA

The user area of memory is assigned to one or more jobs. Each job has an area referred to as the Job Table Area (JTA) preceding the field defined for the user. A JTA is accessible to the operating system but not to the user.

The JTA contains job-related information such as accounting data; JXT pointer; sense switches; areas for saving B, T, and V register contents; control statement, logfile, and EXU DSPs (user calls that load the binaries); a logfile buffer; and a DNT area.

DNT — Dataset Name Table. This area in each user's JTA contains an entry for each dataset used by the job.

Each user field begins with a 128-word block referred to as the Job Communication Block (JCB), which contains a copy of the current control statement for the job as well as other job-related information. The highest of the user field contains dataset buffers and I/O tables.

The user field, in addition to being used for user-requested programs such as the compiler, assembler, and object programs, is also the area in which the operating system utility programs such as the loader, copy and positioning routines, and permanent dataset utility programs execute. The Control Statement Processor (CSP) also executes in the user field.

Tables that may reside in the user field include the following:

BAT — Binary Audit Table. This table contains an entry for each permanent dataset that meets requirements specified on the AUDIT control statement and for which the user number matches the user number for the job.

DDL — Dataset Definition List. A DDL in the user field accompanies each request to create a DNT.

DSP — Dataset Parameter Area. A DSP area in the user field contains information concerning the status of a particular dataset and the location of the I/O buffer for the dataset.

JAC — Job Accounting Table. This table defines the format of data returned to the user by an accounting request.

LFT — Logical File Table. This table in the user field contains an entry for each dataset name and alias referenced by FORTRAN users. Each entry points to the DSP for a dataset.

ODN — Open Dataset Name Table. A request to open a dataset for a job contains a pointer to the ODN table in the user field.

PDD — Permanent Dataset Definition Table. A PDD table in the Control Statement Processor (CSP) is used for saving, accessing, and deleting permanent datasets.

Refer to COS Table Descriptions Internal Reference Manual, publication SM-0045 for detailed descriptions of these tables.

1.5 MASS STORAGE SUBSYSTEM ORGANIZATION

Depending on the CRAY-1 model, mass storage consists of either DD-19 or DD-29 Disk Storage Units and DCU-2, DCU-3, and DCU-4 Disk Control Units. The controllers are model dependent. These devices are physically non-removable. For models that do not have an I/O Subsystem, assignment of units and DCU-2 and DCU-3 DCUs to channels is assembled into the Equipment Table (EQT). The DCU-4 controllers and their corresponding units are on the I/O Subsystem.

Each disk storage unit contains a device label, datasets, and unused space to be allocated to datasets (figure 1-8). Additionally, one disk storage unit is designated as the master device and contains a table area called the Dataset Catalog (DSC), which is used for maintaining information about permanent datasets.

1.5.1 FORMATTING

Before a unit can be introduced into the system, it must be formatted. Formatting is the process of writing cylinder, head, and sector identification on the disk storage unit. This process is performed off-line by field engineers. Unless addressing information has been inadvertently destroyed, formatting is performed only once.

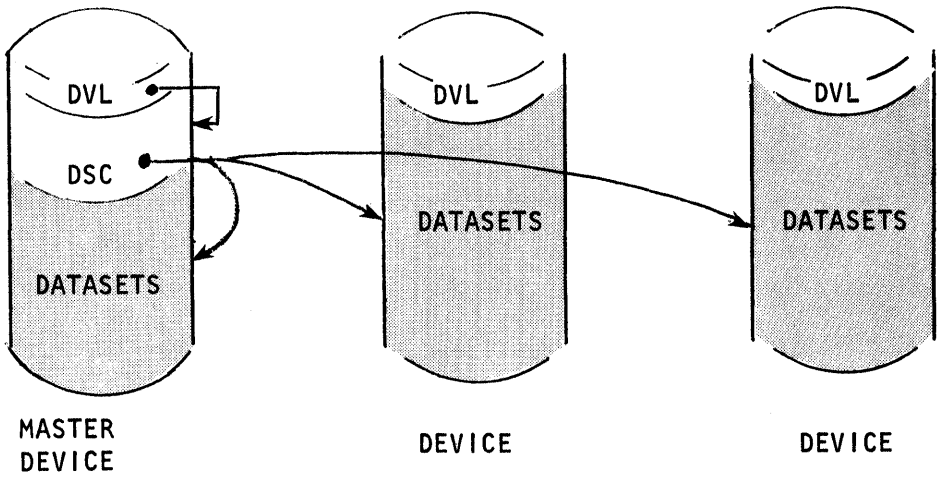


Figure 1-8. Mass storage organization

1.5.2 DEVICE LABEL (DVL)

A disk storage unit (DSU) must be labeled before it can be used by the system. The Install program writes a Device Label Table (DVL) on one track of each DSU. The DVLs act as the starting point for determining the status of mass storage when the system is deadstarted or restarted. The location of the DVL is usually, but is not required to be, the first track on the device.

Flaw information

A DVL contains a list of flaws (bad tracks) for its DSU. Initial flaw information is obtained from an engineering diagnostic run prior to the Install program. Install reads back each DVL after writing it to verify the integrity of the DVL. If a DVL cannot be read back perfectly, then the track is overwritten with a test pattern and a different track is tried.

The DVL is the last track written by Install so that all flaws, even any discovered while trying to write the DVL itself, are recorded in the DVL.

Dataset Allocation Table (DAT) for DSC

The DVL for the master device maps the Dataset Catalog (DSC) since it contains the complete Dataset Allocation Table (DAT) for the DSC except for DAT page headers.

1.5.3 DATASET CATALOG (DSC)

The Device Label Table (DVL) for the master device states which tracks comprise the Dataset Catalog (DSC). Similarly, the DSC states which tracks comprise each of the currently cataloged datasets. Deadstart and Restart update the Disk Reservation Table (DRT) in STP-resident memory to reserve these dataset tracks so that the existence of permanent datasets is known to the system when it is deadstarted or restarted, as opposed to an install which assumes that all of mass storage is vacant. Special consideration is given to job input and output datasets, however. Deadstart deletes all of the input and output datasets, defined by flags in the DSC. Entries for these datasets in the DSC are zeroed. Restart, on the other hand, recovers the job input and output datasets.

1.6 EXCHANGE MECHANISM

The technique employed in the CRAY-1 to switch execution from one program to another is termed the exchange mechanism. A 16-word block of program parameters is maintained for each program. When another program is to begin execution, an operation known as an exchange sequence is initiated. This sequence causes the program parameters for the next program to be executed to be exchanged with the information in the operating registers. The operating register contents are thus saved for the terminating program and the registers entered with data for the new program.

Exchange sequences may be initiated automatically upon occurrence of an interrupt condition or may be voluntarily initiated by the user or by the operating system through normal (EX) or error (ERR) exit instructions.

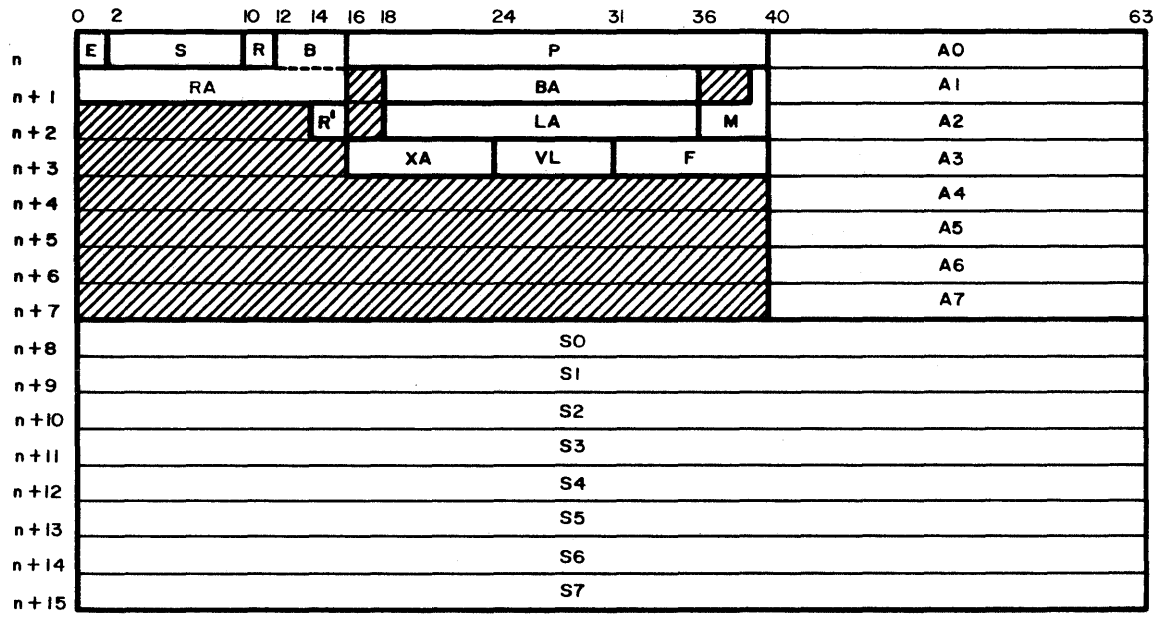
As will be shown in section 2, the System Executive (EXEC) is always a partner in the exchange; that is, it is either the program relinquishing control or receiving control. All other programs must return control to EXEC. The contents of the interrupt flag register (F) are instrumental in the selection of the next program to be executed.

1.6.1 EXCHANGE PACKAGE

An exchange package is a 16-word block of data in memory that is associated with a particular computer program. It contains the basic hardware parameters necessary to provide continuity from one execution interval for the program to the next. The exchange package is illustrated in figure 1-9.

1.6.2 EXCHANGE PACKAGE AREAS

System hardware requires that all exchange packages be located in the first 4096 words of memory. In addition, the deadstart function expects an exchange package to be at address 0. This is the exchange package that initiates execution of EXEC and, consequently, the operating system. The EXEC exchange package is either active or is in one of the other exchange package areas (figure 1-10).



Registers

S	Syndrome bits
R'RAB	Read address for error (where B is bank)
P	Program address
BA	Base address
LA	Limit address
XA	Exchange address
VL	Vector length

E - Error type (bits 0,1 of n)

10	Uncorrectable memory
01	Correctable memory

R - Read mode (bits 10,11 of n)

00	Scalar
01	I/O
10	Vector
11	Fetch

M - Modes

n+1	39	Interrupt monitor mode†
n+2	36	Interrupt on correctable memory error
n+2	37	Interrupt on floating point error
n+2	38	Interrupt on uncorrectable memory error
n+2	39	Monitor mode

F - Flags

n+3	31	Programmable clock interrupt††
n+3	32	MCU interrupt
n+3	33	Floating point error
n+3	34	Operand range error
n+3	35	Program range error
n+3	36	Memory error
n+3	37	I/O interrupt
n+3	38	Error exit
n+3	39	Normal exit

† Supports Monitor Mode Interrupt option on CRAY-1A and CRAY-1B.

†† Supports Programmable Clock (optional on CRAY-1A and CRAY-1B; standard on CRAY-1 S Series computers)

Figure 1-9. Exchange package

These other exchange packages, summarized below, are selected by EXEC depending on interrupt flags and other conditions as defined later:

- Any of a set of exchange packages in the System Task Table (STT). (There is one exchange package for each STP task.)

- The active user exchange package. This exchange package, located at location SXBF, points to the currently active user program as selected by the Job Scheduler. When a user program becomes inactive (is disconnected from the CPU) or causes a normal or error exit, its exchange package is copied from the active user exchange package area into the Job Table Area (JTA) for that job. When a user program is disconnected, some other job may be connected to the CPU and its exchange package is copied from its JTA to the active user exchange package area.

- The idle program exchange package. This exchange package, located at location IDXP, is selected when there are no tasks or user programs are scheduled for execution.

- The memory error correction exchange package. This exchange package, located at CORXP, is selected when an exchange is caused by a memory parity error.

1.6.3 B, T, AND V REGISTERS

On any exchange to EXEC, EXEC saves the task or user program's B0 register because EXEC uses B0. A task's B0 register values are stored in the STT. The active user's B0 value is stored in SSB0 during interrupt processing. When EXEC exchanges out, it restores the proper B0 register value.

All B, T, and V register values are saved by EXEC only when the current user job is being disconnected from the CPU in favor of some other job. A job's B, T, and V register values are restored when it is reconnected to the CPU. These registers are maintained in the job's JTA.

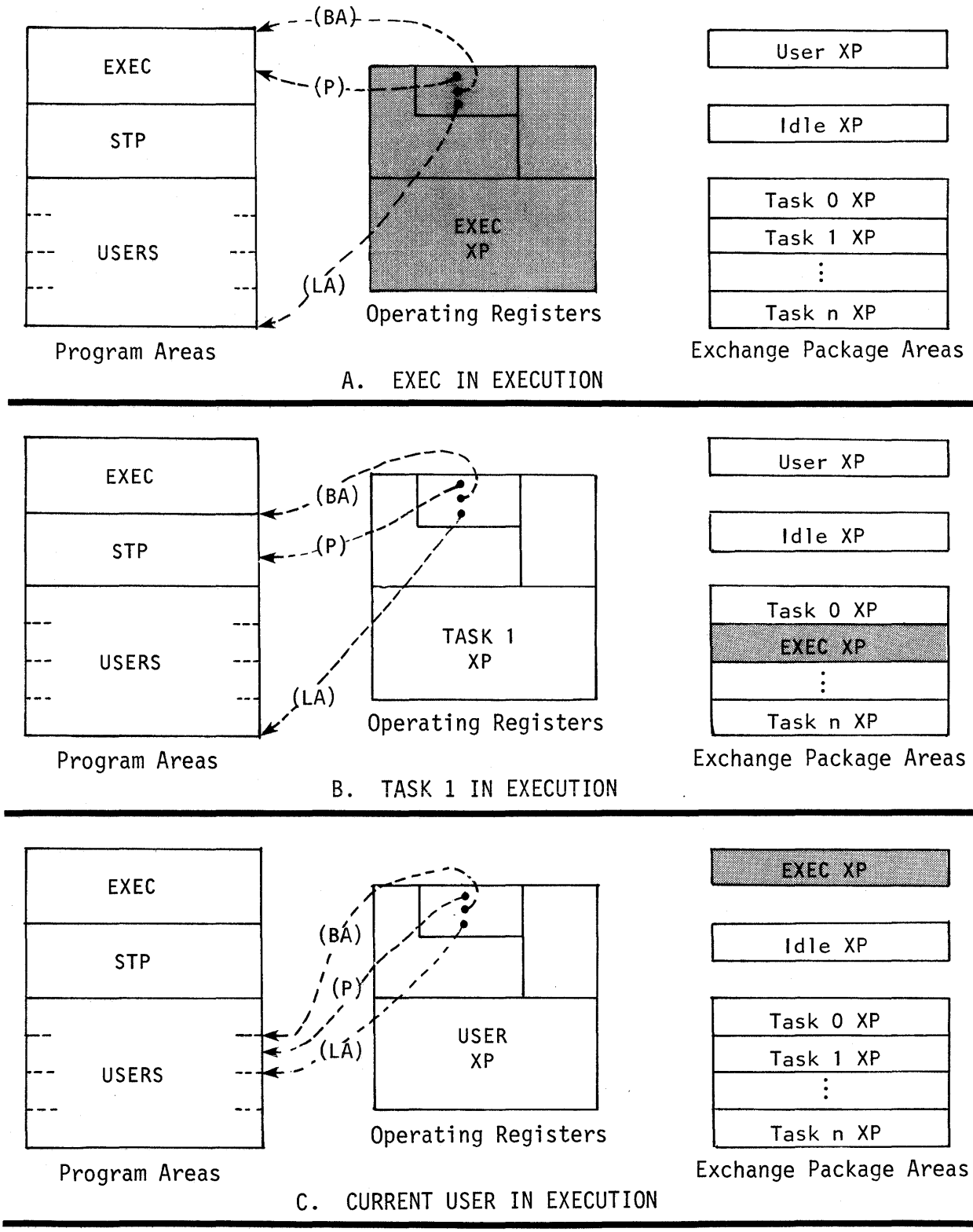


Figure 1-10. Exchange package management

1.7 COS STARTUP

System Startup is the process of loading the operating system into CRAY-1 memory, beginning execution, and generating or recovering tables for the operating system. There are three types of startup: Install, Deadstart, and Restart. A general description follows; details are given in section 5.

Install	For an install, COS is started as if for the very first time. All CRAY-1 mass storage is assumed to be vacant. The startup program labels devices and establishes the Dataset Catalog (DSC) on mass storage.
Deadstart	For a deadstart, COS is started as if after a normal system power-down. Permanent datasets are recovered but input queues and output queues are not reconstructed. Rolled-out jobs cannot be recovered during a deadstart.
Restart	For a restart, COS is started as if after a system failure (crash). Input queues and output queues as well as permanent datasets are recovered. Rolled-out jobs may be recovered according to operator selection.

1.8 GENERAL DESCRIPTION OF JOB FLOW

A job passes through the following stages from the time it is read by the front-end system until it terminates:

- Entry
- Initiation
- Advancement
- Termination

1.8.1 JOB ENTRY

A job enters the system from a front-end system. The Station Call Processor task (SCP) in STP is responsible for making the job's existence known to the system. It does this by:

- Making an entry in the System Dataset Table (SDT),
- Requesting that an entry be created in the Dataset Catalog (DSC), thereby making the dataset permanent, and
- Readying the Job Scheduler Task (JSH).

1.8.2 JOB INITIATION

The Job Scheduler Task (JSH) scans the SDT looking for candidates for processing. A job is scheduled to begin processing (initiated) when:

- An entry for a job of the correct class is available in the Job Execution Table (JXT) (the maximum number of entries in the JXT is 63), and
- No other job of higher priority is waiting to begin processing.

JSH uses an available entry in the JXT to create an entry for the job being initiated. The Job Scheduler continues to use the JXT entry during the life of the job to control CPU use, job roll-in/roll-out, and memory allocation.

JSH also moves the job's SDT entry from the input queue to the executing queue, still in the SDT.

The Rolled Job Index entry corresponding to the assigned JXT entry is also initialized at this point.

1.8.3 JOB ADVANCEMENT

The Job Scheduler gives each job a CPU priority that reflects its history of CPU usage so that I/O-bound jobs can have a greater chance of being assigned to the CPU. A job requiring a large memory area is allowed to stay in memory longer to compensate for its greater roll-in/roll-out time. A job assigned more than average CPU time for its priority is liable to be rolled out sooner as a consequence. The operator may change a job's priority while a job is running.

Not all jobs having entries in the JXT are in memory. Some may be rolled out to mass storage when some event has occurred that causes other jobs to replace them in memory.

The Control Statement Processor (CSP) advances a job through its program steps. CSP is first loaded and executed in the user field following job initiation; thereafter, it is called whenever a job step terminates. Normal job step termination occurs when an F$ADV call is made to the system by the user program. Abnormal termination occurs upon detection of an error during the job step or an F$ABT call by the user program.

1.8.4 JOB TERMINATION

When a job terminates, the following action occurs:

- A DSC entry is created for the job's output datasets.
- A SDT entry is created for the job's output datasets.
- The DSC entry is deleted for the input dataset.
- The user dayfile, $LOG, is copied onto the end of $OUT.
- The SDT entry is deleted from the executing queue.
- The JXT entry and the memory assigned to the job are released
- The Rolled Job Index entry is cleared (zeroed).
- SCP is readied and scans the SDT for output to send to the front-end system.
- SCP deletes its DSC and SDT entries after the output dataset is totally transmitted to the front-end system.

1.9 DATASET MANAGEMENT

All information maintained on mass storage by the CRAY-1 Operating System is organized into collections of information known as datasets. Datasets are of two types: local or permanent. A local dataset exists only for the life of the job that created it and can be accessed only by that job. A permanent dataset is available to the system and can survive system deadstarts.

A dataset is permanent if it has an entry in the Dataset Catalog on disk. Permanent datasets are of two types: those that are created through use of directives (user permanent datasets), and those that represent standard job input and output datasets (system permanent datasets).

User permanent datasets are maintained for as long as the user or installation desires. A user permanent dataset is protected from unauthorized access by use of permission control words. The user may create a user permanent dataset by pre-staging in a dataset from a front-end computer system or by using the SAVE or ACQUIRE control statement or macro. A user accesses a user permanent dataset by using the ACCESS control statement or macro. The dataset may be removed from the system with the DELETE control statement or macro. More than one authorized user may access a permanent dataset. A user wishing to write on or otherwise alter a permanent dataset must have unique access; multiple users wishing to read the dataset may have multiaccess.

Some permanent datasets similar to user permanent datasets are created and maintained by the system. No user can either delete or access these datasets because the system has unique access to them. Among these datasets is the Rolled Job Index dataset, which is created or accessed by the Startup task and remains in use throughout the operation of the system.

System permanent datasets are job related. Each job's input dataset is made permanent when the job is received by the CRAY-1. When job processing ends, certain of the job's local datasets having special names or which were given a disposition other than scratch by the user are made permanent and the job's input dataset is deleted from CRAY-1 mass storage. The output datasets that were made permanent are sent to a front-end computer system for processing. They are deleted from CRAY-1 mass storage when their receipt has been acknowledged by the front-end computer system.

1.10 I/O INTERFACES

Figure 1-11 presents an overview of the interfaces and system components involved in performing input and output in the system. It summarizes the request levels and routine calls without going into details on the movement of data. That is, it does not describe how data is transferred from disk to a circular buffer and then to a user area on a read; nor does it describe how it is transferred in the reverse sequence on a write.

Major interfaces exist between the user and STP and between STP and EXEC. Details of the user levels of I/O are presented in the FORTRAN Reference Manual, publication 2240009, and in the CRAY-OS Version 1 Reference Manual, publication SR-0011. Details for EXEC (driver level) I/O are given in section 1 of this publication. Details for STP interfaces are given in section 3.3 of this publication.

I/O can be blocked or unblocked and can be initiated by the user or by the system.

FORTRAN statements for logical I/O represent the highest level of I/O requests. The FORTRAN statements fall into two categories: formatted/unformatted and buffered. The formatted/unformatted statements (i.e., READ, PUNCH, WRITE, and PRINT) result in calls to library routines $RFI through $WUF. These routines contain calls to the Logical Record I/O routines, also on the library. These calls may be formatted by the user or may be made through CAL language macros.

The Logical Record I/O routines issue Exchange Processor requests (i.e., F$ calls) that consists of read circular and write circular requests to the Circular Input/Output (CIO) routines resident in STP (see section 3.3.3).

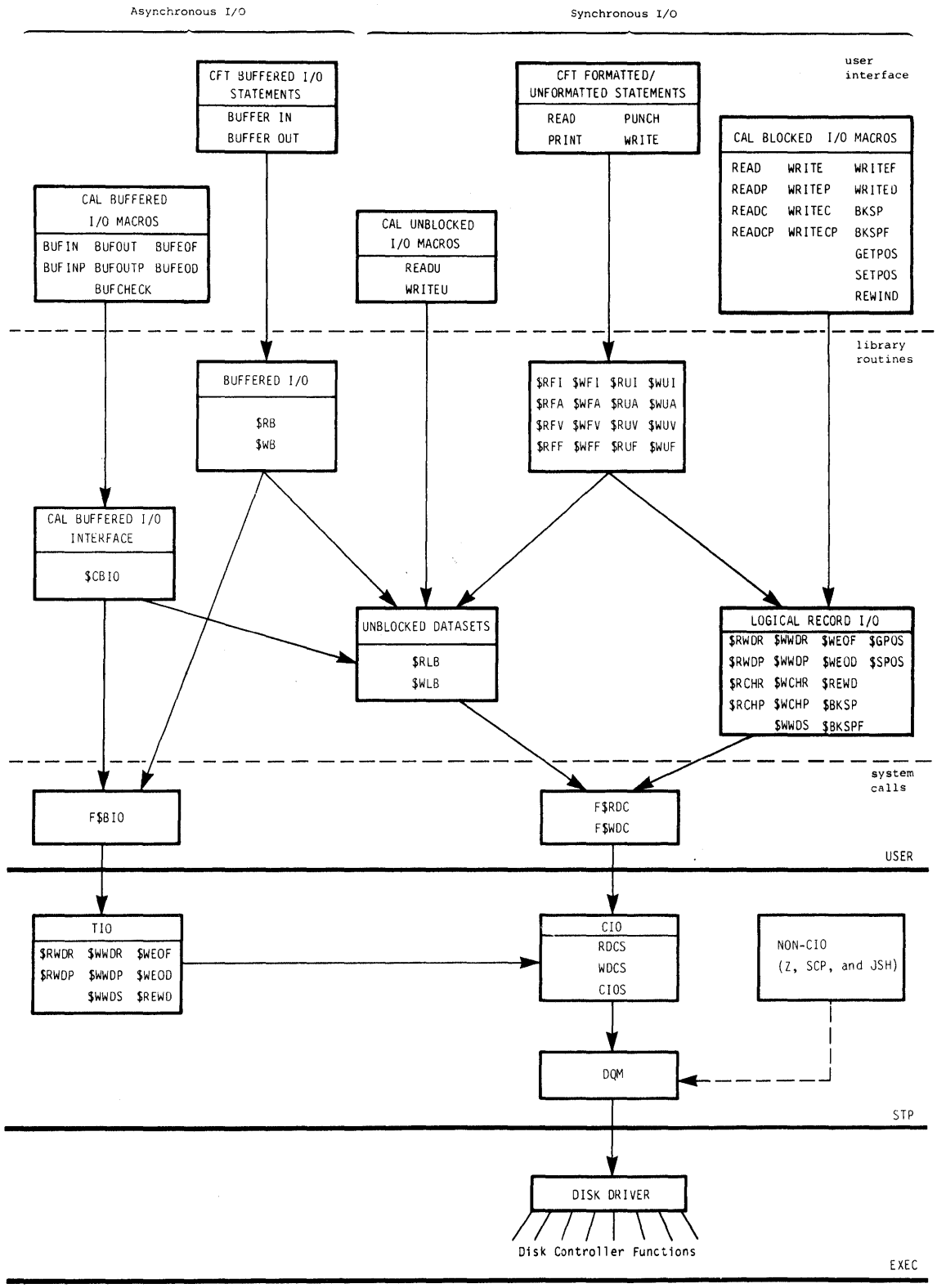


Figure 1-11. Overview of COS I/O

System logical I/O required by COS tasks (e.g., management of the Dataset Catalog, etc.) is generally performed through Task I/O routines resident in STP (see section 3.3.2). TIO routines closely resemble the Logical Record I/O routines. In addition to supporting I/O for system tasks, TIO routines also handle FORTRAN buffered I/O. At the FORTRAN level, the BUFFER IN and BUFFER OUT statements are compiled into calls to two library routines, $RB and $WB. These routines issue F$BIO Exchange Processor requests that interface with a subset of TIO routines in STP.

Since TIO routines reside jointly with CIO in STP, they directly call CIO routines to perform the same functions as requested through F$ calls by the Logical Record I/O routines. Thus, CIO becomes the focal point for all logical I/O in the system.

CIO communicates its needs for physical I/O to the Disk Queue Manager (DQM) through DNT and DSP tables. The DNT for a dataset points to its DSP, which specifies the request. This is the normal mode of communication with DQM. Currently, however, DQM also communicates with the Station and Startup interfaces. In these interfaces, SCP and Z pass a caller-built DNT containing the I/O request for DQM. The Job Scheduler (JSH) also uses a non-CIO interface to process job roll-in/roll-out and to manipulate the Rolled Job Index dataset.

DQM coordinates physical I/O activity on the disks by queueing executive requests for the Disk Driver (section 2.8). This driver consists of a number of channel processors that issue functions to the disk controllers.

EXEC 2

The system Executive module (EXEC) is the control center for the operating system. It alone accesses all of memory, controls the I/O channels, and selects the program to execute next. Components of EXEC include an interchange routine, interrupt handlers, channel processors, an EXEC request processor, and a task scheduler. These programs are integral to EXEC. Control transfers from routine to routine through simple jumps.

EXEC first begins execution when the system is started up. Following this initial system interchange, EXEC performs the following functions.

- Disables and clears the programmable clock,
- Determines the size of the deadstart binary by reading the channel address for the MCU channel,
- Receives a word packet containing time and date from the MCU,
- Sets the real-time clock,
- Master clears each disk control unit,
- Sets the SECDED bits in memory in case a power off has occurred,
- Sets the limit address to the proper value, and
- Starts up the root task (Startup).

After the system is started up, EXEC begins execution whenever a task or the currently active user program is interrupted. This interrupt may result from the program executing an exit instruction, ERR or EX, or may be an I/O or error condition. EXEC saves (B0) and saves the clock upon entry. If one second has elapsed, it sets the real-time (RT) flag in the exchange package. After setting and checking the clock, EXEC initiates execution of the Interchange routine.

Interchange analyzes the cause of the interrupt (details are given below) and passes control to the appropriate handler. The interrupt handler, in turn, clears the interrupt flag and activates a channel processor. After processing the interrupt condition, the channel processor returns to Interchange which checks for additional conditions. When all of the outstanding interrupts have been analyzed and processed, the Task Scheduler selects the highest priority task ready for execution. If no tasks are ready, the currently active user program is scheduled for execution. If no user program is currently active, the idle program is executed. EXEC then initiates an exchange sequence to the selected program and does not regain control until the next interrupt occurs.

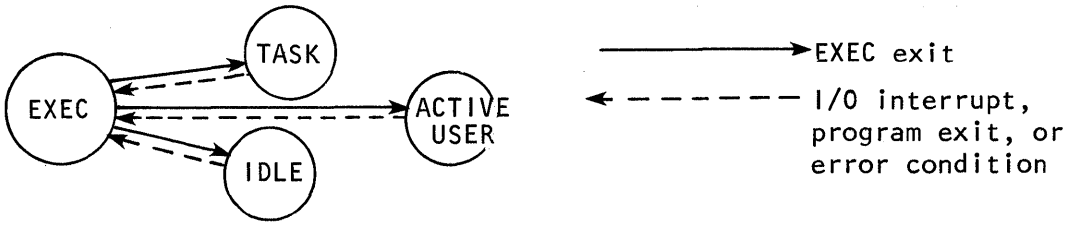


Figure 2-1. EXEC-controlled exchange sequences

2.1 INTERCHANGE INTERRUPT ANALYSIS

Each time Interchange is entered, it checks to see if the cause of the interrupt was an I/O condition. It does not do this by directly checking the I/O interrupt flag in the F register but rather by requesting the channel number of the highest priority channel with an I/O request. If a channel has an I/O request, Interchange calls the I/O Interrupt Handler, IOI, which then uses the channel number to initiate the correct channel processor. The channel processor returns control to Interchange which repeats the inquiry for the highest priority channel having an I/O request. In this way, Interchange continues to process I/O requests until none remains.

When no I/O requests are outstanding, Interchange checks each of the following interrupt flags in turn, summoning the appropriate interrupt handler:

MCU
Programmable clock
Real-time
Error exit, floating-point error, program range error, and operand range error (tested as a group)
Normal exit
Memory error exit

After a flag is processed, control always returns to the beginning of EXEC to ensure that any I/O interrupts occurring in the interim are handled.

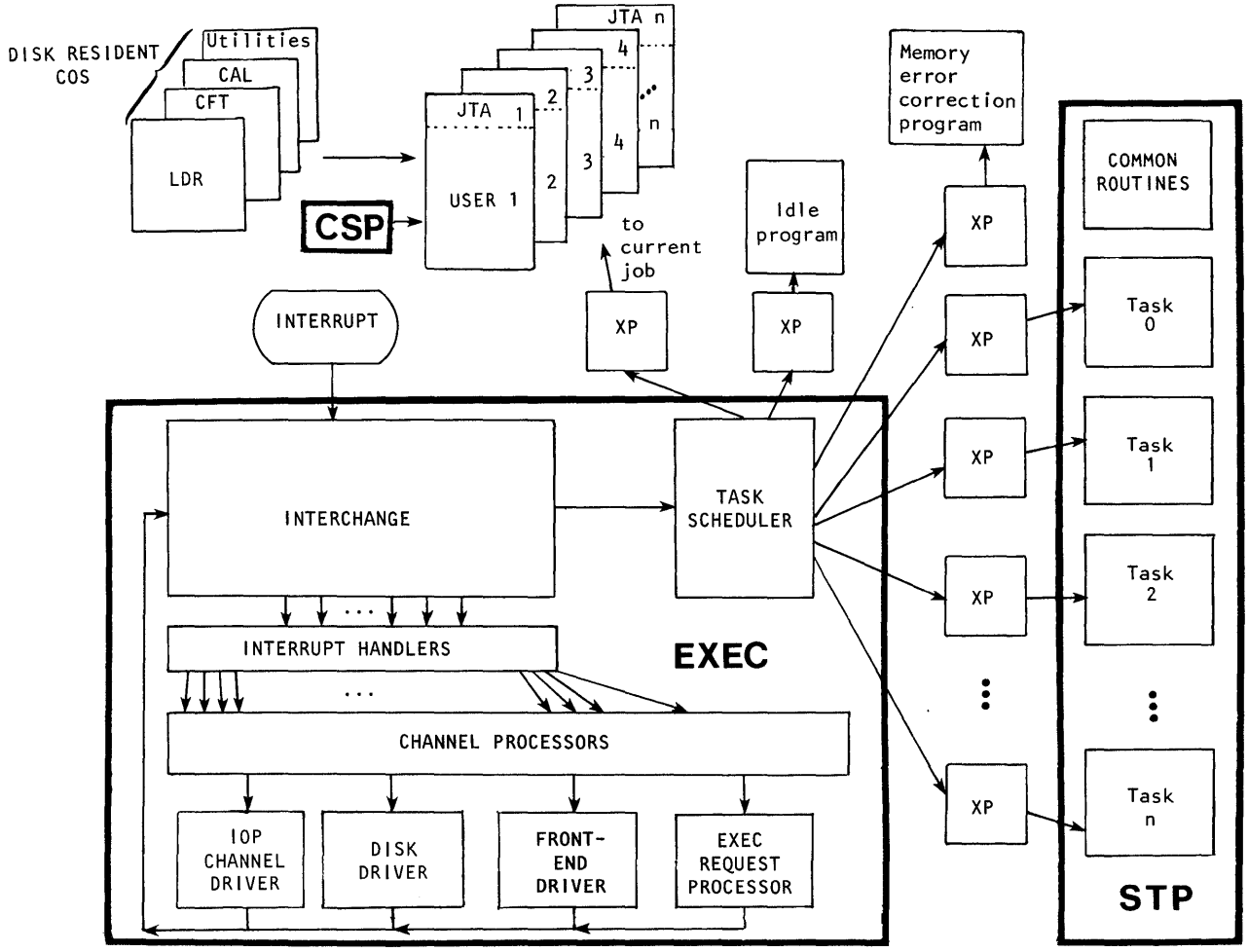


Figure 2-2. System control

2.2 INTERRUPT HANDLERS

An interrupt occurs, triggering an exchange to EXEC, if one or more of the interrupt flags is set in the F register. One or more of the following interrupt handlers will then be executed.

IOI	I/O interrupt handler
CII	MCU interrupt handler
RTI	Real-time interrupt handler
NEI	Normal exit interrupt handler
EEI	Error exit interrupt handler
MEI	Memory error interrupt handler

The interrupt handler clears the interrupt flag in the exchange package of the interrupted program and determines the channel on which the interrupt occurred. Execution of the processor assigned to that channel is then initiated.

2.3 CHANNEL MANAGEMENT

The operating system recognizes 12 physical I/O channels and 4 pseudo channels. The four pseudo channels are for normal exit, error exit, memory error, and real-time interrupts and can be processed in a manner similar to I/O interrupts. Each channel is considered as having two sides, an input side and an output side, each numbered separately.

Input sides are assigned even numbers; output sides are assigned odd numbers. When both sides of a channel are referenced (sometimes referred to as a channel pair), the number is the input channel number divided by two. That is, channel pair 5 is channels 10 and 11. Thus, channel pairs are assigned decimal numbers as follows:

0	Console interrupt
2 4 6 . . . 24	12 I/O channel pairs
26	Normal exit pseudo channel pair
28	Error exit pseudo channel pair
30	Real-time interrupt pseudo channel pair
32	Memory error pseudo channel pair

2.3.1 CHANNEL TABLES

The following tables aid in channel management:

Channel Table (CHT)
I/O Service Task Defined Table
System Task Table (STT)
Channel Extension Table (CXT)
Channel Buffer Table (CBT)
Subsystem Control Table (SCT)

Detailed information on these tables is available in the COS Table Descriptions Internal Reference Manual, publication SM-0045.

Figure 2-3 illustrates how these tables are linked together.

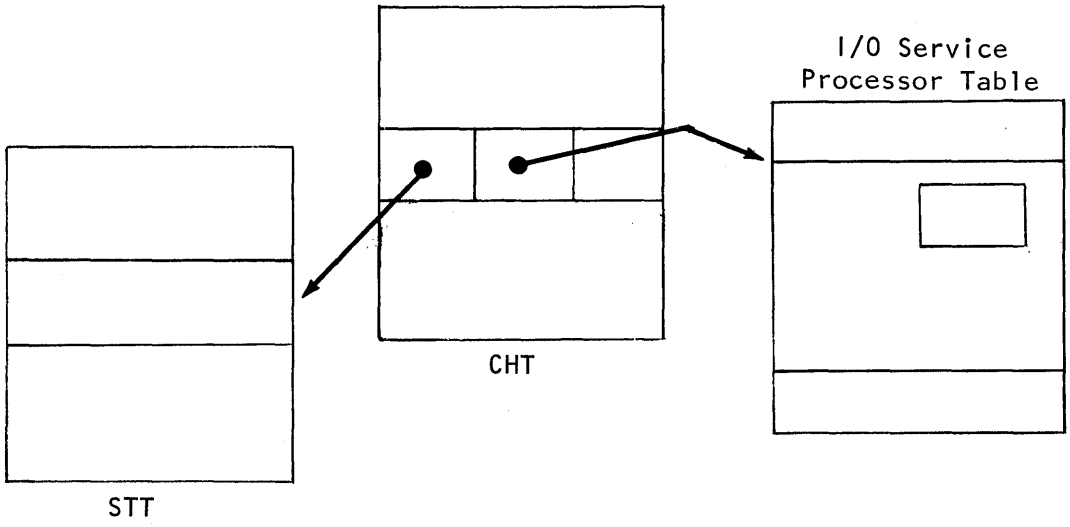


Figure 2-3. Channel table linkage

Channel Table (CHT)

CHT contains an entry for the input side and the output side of each channel, real or pseudo. The entry contains (1) a pointer to the I/O Service Processor table used by the channel processor to control the channel, (2) the address of a processor for that side of the channel, and (3) a pointer to the STT parameter block for the task using the request on this channel.

I/O Service Processor Tables

The I/O Service Processor tables contain information for control of the channel processor and may contain pointers to other tables. Each of the following channel types has a different I/O Service Processor Table: front-end channel, mass storage channel, exchange pseudo channel, memory error, and real-time pseudo channel.

System Task Table (STT)

The STT contains a parameter block and the exchange package area for each task. Tasks are identified by numeric IDs.

2.3.2 CHANNEL ASSIGNMENTS

Channel assignment is a 2-level process:

- Assign a task to a channel
- Assign an EXEC processor to a channel

Startup performs the first level when it enters the task ID and Channel Control Table address in the CHT. The second level assignments occur either at system build time or dynamically during I/O initiation.

The following executive requests relate to channel assignment:

- Assign channel
- Disk block I/O request
- Station I/O request

2.3.3 CHANNEL PROCESSORS

Some channel processors are assigned to a channel permanently; others are assigned on an as-needed basis as a result of task requests to EXEC. Each side of a channel always has one of the following processors assigned to it. The current assignment is maintained in the Channel Table (CHT).

Console Exit Processor (CEP)

The Console Exit Processor is always assigned to channel 0 and to the MCU interrupt flag. It forces the executive to check the top event on its time event stack.

Reject Processor (RJ)

The Reject Processor is assigned to any unused channels. A transient interrupt could result in RJ being executed. The processor simply returns control to the interchange routine.

Normal Exit Processor (NE)

The Normal Exit Processor is assigned to pseudo channel 26. If the normal exchange is from a task, bit STRTS of word 1 of the STT header is nonzero and NE initiates execution of the executive request processor. Otherwise, the job has issued a system task request and the Exchange Processor task is scheduled.

Error Exit Processor (EE)

The Error Exit Processor is assigned to pseudo channel 28. If the error exit was from a task, EE checks for a possible breakpoint condition. If the error exit (includes error exit and operand range and floating-point errors) was from the currently active user, the Exchange Processor task is readied. The interrupt register error flags are passed to the Exchange Processor for interpretation of the cause of the error exit.

Memory Error Processor (ME)

The Memory Error Processor is assigned to pseudo channel 32. When a single-bit memory error occurs, it compares the total single-bit error count against the limit for disabling single-bit error detection. If the count exceeds the limit, the processor disables the interrupt on single-bit errors. For single-bit errors, the processor attempts to restore the memory address to its correct value so that the error does not occur on the next read of that address.

For all memory errors, the processor readies the memory error task, which determines what software corrective action should occur.

Real-time Interrupt Processor (RTP)

The Real-time Interrupt Processor is assigned to pseudo channel 30. If the job's time slice has elapsed, it schedules the Job Scheduler task.

Disk Processors

Any of the following disk processors can be assigned to a CPU I/O channel as a result of an executive request for the disk driver from the Disk Queue Manager.

US	Unit select
DP	Disk position
WD	Write
MS	Margin select
RD	Read
SS	Subsystem status
MC	Master clear controller
EC	Error correction

When a processor completes its function, the disk driver in the Executive Request Processor assigns a processor for the next function in the sequence to be performed without involving a task. Refer to Disk Driver, section 2.8, for details.

Front-end Processors

Any of the following processors can be assigned to a CPU I/O channel as a result of an executive request for the front-end driver from the Station Call Processor task.

WLCP	Write link control package
WSSEG	Write subsegment
WLTP	Write link trailer package
RLCP	Read link control package
RSSEG	Read subsegment
RLTP	Read link trailer package
WXLCP	Write error link control package
WXLTP	Write error link trailer package

When a processor completes its function, it assigns the next front-end processor or reject (RJ) to the channel without involving the Station Call Processor. Refer to Front-end Driver, section 2.7, for details.

I/O Subsystem MIOP Command and Status Processors

The following two processors are assigned respectively to the I/O Subsystem Master I/O Processor (MIOP) command and status channels which are handled in a fully asynchronous manner.

- APIIP Processes MIOP status input interrupt
- APOIP Processes MIOP command output interrupt

APIIP calls the following packet processors to handle the appropriate status packets.

APPACN	Null packet
APPACX	STP task packet
APPACE	Echo packet
APPACI	I-packet processor
APPACJ	J-packet processor
APPACA	Disk packet
APPACB	Station packet

I/O Subsystem BIOP Simulated Memory Channel

The following processors handle the I/O Subsystem Buffer I/O Processor (BIOP) simulated memory channel in a synchronous manner.

SHSRQT	Simulated memory channel request processor
SHSOT	Simulated memory channel output processor
SHSIT	Simulated memory channel input processor

2.4 TASK SCHEDULER

If a channel processor has requested execution of the Task Scheduler, the request task scheduler flag (STRTS) is set. If execution of the Task Scheduler has not been requested, the currently executing task resumes execution.

The Task Scheduler executes, if requested, when Interchange has checked all of the interrupt conditions. If the STP interlock flag is set, the Task Scheduler returns to the previously executing task. This flag allows tasks to run in uninterruptible mode.

Depending on the condition of the debug scheduling flag, one of two parts of the Task Scheduler executes. If the flag is not set, normal scheduling occurs. The STT is examined for the highest priority task ready to execute and its exchange package is selected. If two tasks have equally high priority, the first encountered task is selected.

If debug scheduling is indicated (the debug scheduler flag is set), only the Station Call Processor (SCP) is a candidate for scheduling.

A task is a candidate for execution when all of its status bits[§] in its parameter word in the STT are clear. The status of a task may change when a task issues one of the following executive requests:

- Create a task
- Ready a task
- Suspend self
- Ready called task and suspend self

Figure 2-4 illustrates the table linkage for task scheduling.

If no task is ready for execution, the currently active job as defined by the Job Scheduler task is allowed to run. If no job is currently active, the idle program is selected.

2.5 EXEC RESOURCE ACCOUNTING

EXEC maintains the following performance information in EXEC tables:

- Accumulated CPU time for itself (in EXEC table ETIM)
- Accumulated CPU time for each task (in STT)
- Total time given to users (in table UTIM)
- Count of all channel interrupts for both real and pseudo channels (IC)
- Each user's execution time (in Job Table Area)
- Number of normal exits for each task (in STT)
- Number of ready task requests, both from other tasks and from external and internal interrupts, for each task (in STT)
- Number of each type of EXEC request

2.6 EXECUTIVE REQUEST PROCESSOR

The Executive Request Processor is initiated by the Normal Exit (NE) channel processor when a normal exchange from a task implies the presence of a request for the Executive. The request is passed to EXEC in registers S6 and S7 of the task's exchange package. The low-order bits of word 1 of the STT header point to the parameter word for the interrupted task.

[§] Ready is not considered a status bit.

The Executive Request Processor handles the requests defined by the Task Call Table (TCT).

When EXEC returns to a task following processing of an Executive request, control returns at (P)+3 for a normal return and at (P)+1 if an error occurred. (P) is the address of the exit to EXEC.

2.6.1 EXECUTIVE REQUESTS

This section provides the request format and functional flow of executive requests issued by tasks.

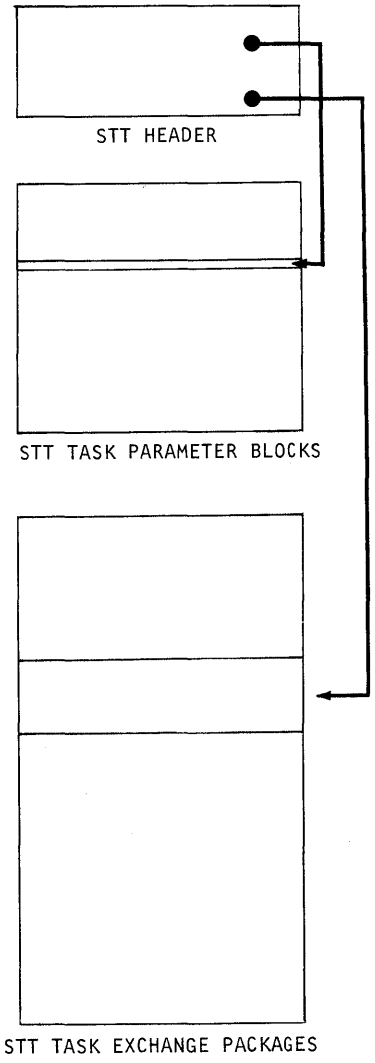
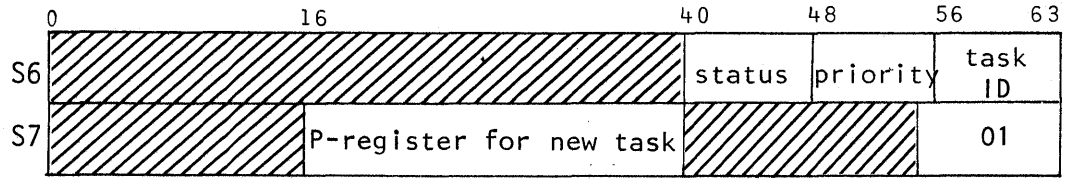


Figure 2-4. Task Scheduler table linkage

Create a task request (CTSK=01)

Request format:

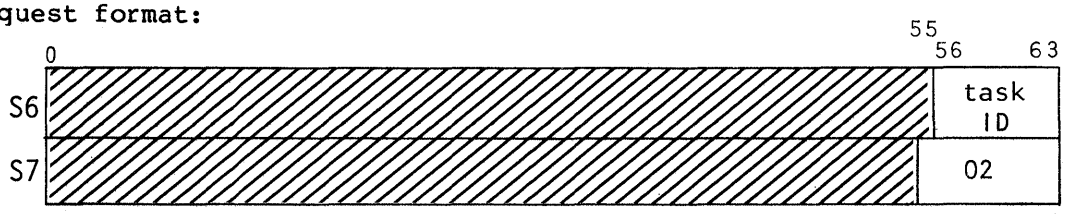


This function adds a task to the EXEC STT. The flow is:

1. If table is full, report table full error to STP.
2. Retrieve task's XP information from caller.
3. Set task status to requested status.
4. Set task ID to requested priority.
5. Set task priority to requested ID.
6. Construct task XP in STT XP area.
7. Set task defined bit in header of STT.
8. Build disk I/O reply queue control word.
9. Force task into execution.

Ready a task request (RTSK=02)

Request format:

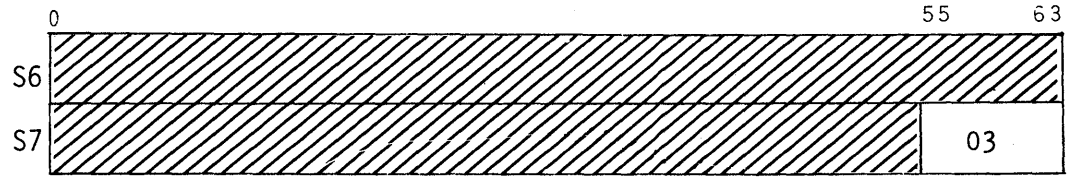


This function readies a task. Its flow is:

1. Find task with desired ID; if none, report error to STP.
2. Clear suspend flags; set reready status if task currently ready.
3. If task is suspended, request task scheduler.

Self-suspend task request (SUSP=03)

Request format:



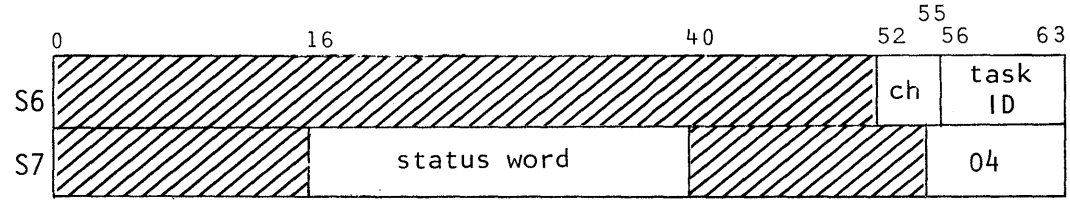
This request suspends a task. Its flow is:

1. If reready bit is clear, set suspend bit; otherwise, clear reready bit.

2. Request task scheduler to run.

Assign channel request (ARES=04)

Request format:



This request assigns a task to a channel pair. The flow is:

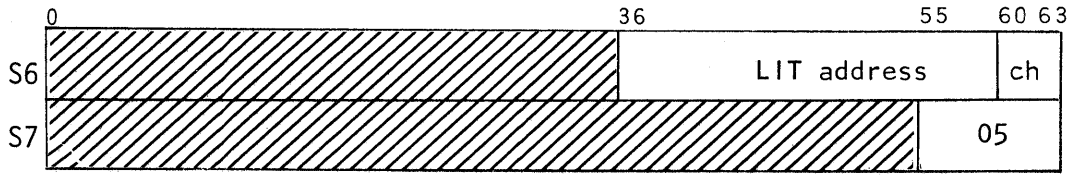
1. Find task with desired ID.

2. Check for task assigned to channel; if one is, report status to STP.

3. Link task to channel by entering the Task Parameter Block (TPB) address of task in CHT entry for the channel pair.

Station I/O request (FET=05)

Request format:



This request activates the input and/or output sides of a channel pair. The processing flow is as follows:

1. If channel ordinal is 0:

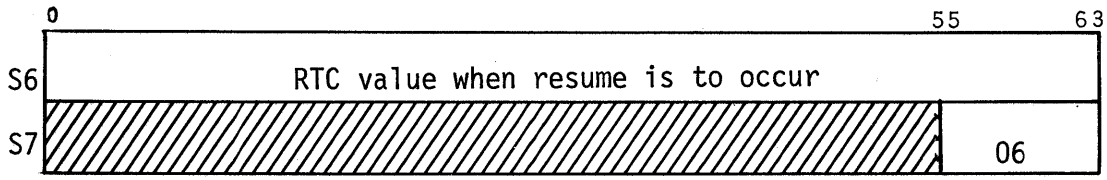
    a. Assign task to channel.
    b. Set input and/or output active flags.
    c. Set CA and CL for input and/or output.§
    d. Start processing by station channel driver.
    e. Release task from channel.

2. Otherwise:

    a. Build MIOP station request in CXT.
    b. When MIOP requests addresses, put message on send queue to MIOP. (The CXT contains a flag indicating that an address request has arrived and addresses should be queued immediately.)
    c. Return to requesting task.

Timed ready request (TDLY=06)

Request format:



§ Privileged to monitor (EXEC)

This request suspends a task until the specified time occurs. The processing flow is:

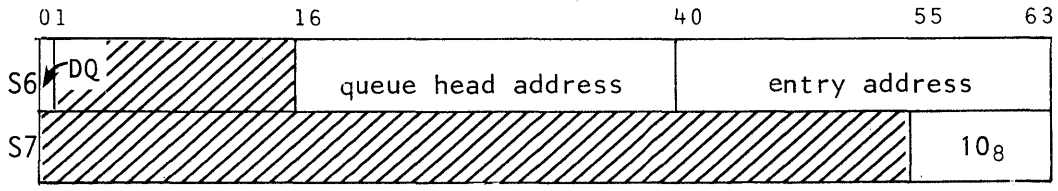
1. Clear current task time delay, if set.

2. Set resume time in time event stack.

3. After time delay, ready task and request scheduling.

Dequeue SDT entry request (DQSD=10)

Request format:



DQ
0 if FIFO dequeuing
1 if entry dequeuing

The processing flow is:

1. Type of dequeue is determined.
   - If FIFO dequeuing, the first entry is used. Its address is placed in S6.
   - If entry dequeuing, the entry specified in S6 is used.

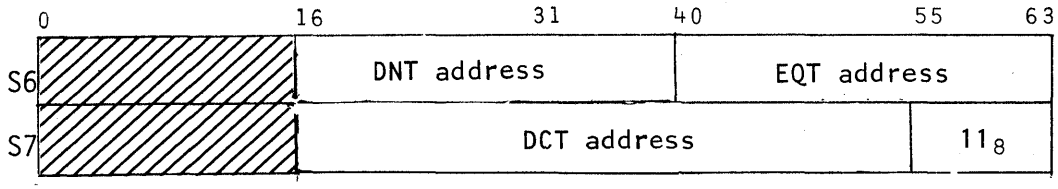
2. Entry dequeued.

3. Count in queue head is decremented by 1.

4. Entry is unlocked.

Disk block I/O request (IO=11)

Request format:



The processing flow is as follows:

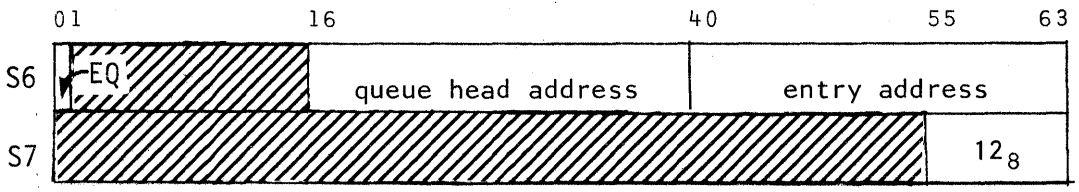
1. If channel number greater than 12:

   a. Build a MIOP disk request.
   b. Set a request timeout.
   c. Return to the calling task.

2. Otherwise, the disk block I/O request results in execution of the disk drive. See section 2.8.

Enqueue SDT entry request (EQSD=12)

Request format:



EQ
0 if FIFO enqueuing
1 if class-priority enqueuing

The processing flow is:

1. Type of enqueue is determined.

   - If FIFO enqueueing, the last entry position is used.

   - If class-priority enqueueing, the queue is searched by class rank, priority, and time submitted to determine position.

2. Entry is enqueued.

3. Count in queue head is incremented by 1.

4. Entry is locked.

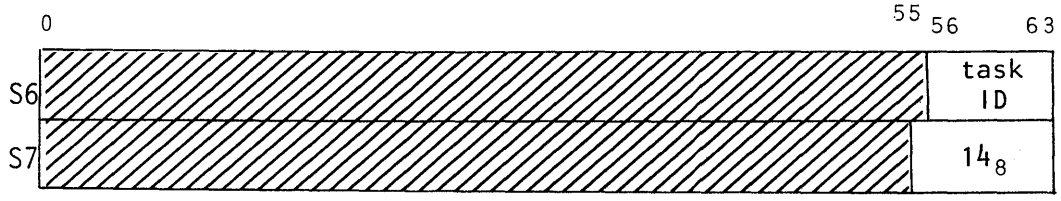
5. Set up SDT with control information.

6. Set return address.

The routine that processes this request is actually the disk driver (section 2.8).

Ready task and suspend self request (RTSS=14)

Request format:

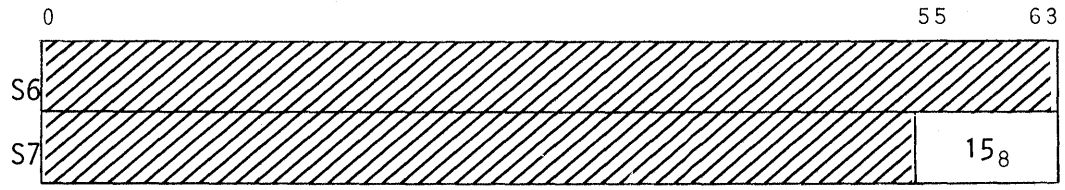


This request permits one task to ready another and then have itself suspended. The processing flow is:

1. Find task ID.
2. Clear suspend bit of called task.
3. Set suspend bit of calling task.
4. Set EXEC scheduler request bit.

Get time and date request (RQST=15)

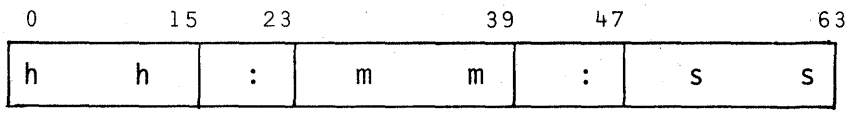
Request format:



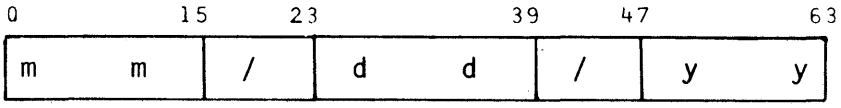
This request returns the time and date to the requesting task. The flow is:

1. Get time and date from real-time pseudo channel.
2. Set time and date into S6 and S7 of requesting task's XP area.

Time format:

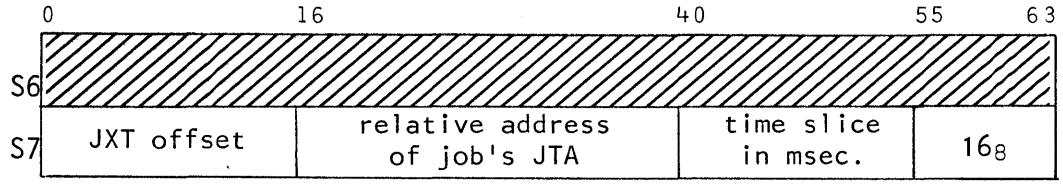


Date format:



Connect user job to CPU request (RCP=16)

Request format:

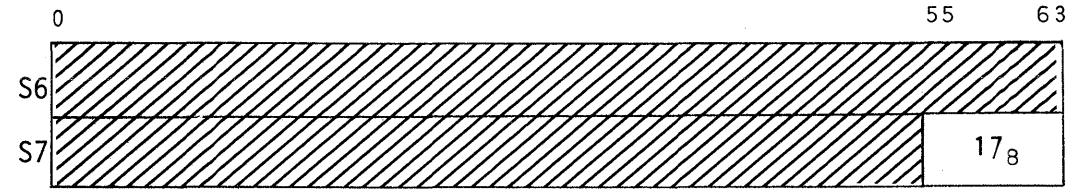


The job scheduler issues this request when the CPU is to be switched from the currently executing job to a newly selected job. The executive request flow is:

1. Verify CPU not in use by another job.
2. Get absolute address of job.
3. Set SAXP to user P register and XP pointer.
4. Copy XA from user XP area.
5. Set time slice value into real-time pseudo channel.
6. Set start time into real-time pseudo channel.
7. Load B, T, and V registers.

Disconnect user job from CPU request (DCP=17)

Request format:

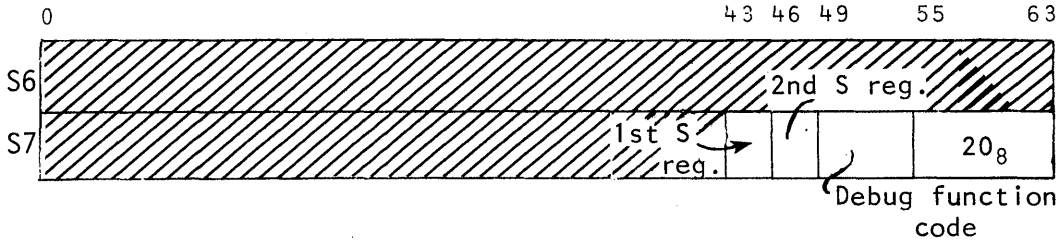


This request must precede an RCP request issued by the job scheduler. The processing flow for this request is:

1. Clear SAXP.
2. Save B, T, and V registers.
3. Relocate BA and LA.
4. Copy XP to Job Table Area for the job being disconnected.

Post message in history buffer request (POST=20)

Request format:

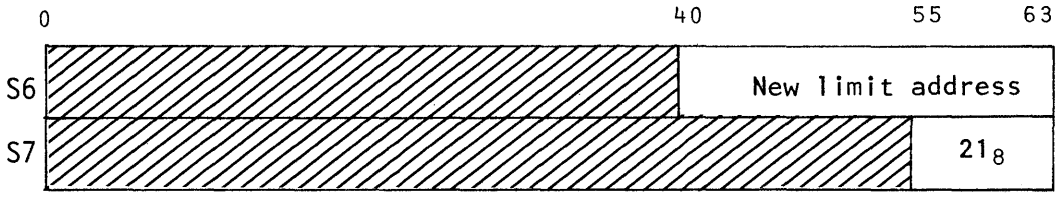


This request permits any STP task to enter two S registers of information into the history buffer, when that debug function is selected. The processing flow is:

1. Set up call to EXEC subroutine DEBUG. In other words, move debug function code to A5, first S register to S6, and second S register to S7.
2. Call subroutine DEBUG to enter message in trace with time and issuing location stamp.

Set memory size request (SMSZ=21)

Request format:

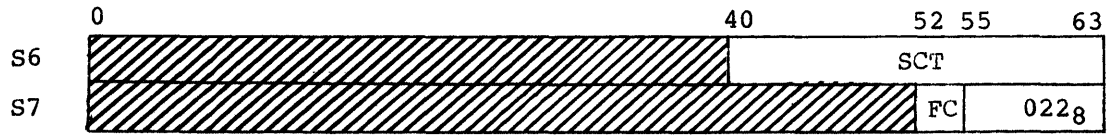


This request is used during system initialization when the size of memory is changed through a Startup *SIZ parameter.

1. Set new system limit address in all system exchange packages.

Packet I/O request (PIO=22)

Request format:



Field	Word	Bits	Description
SCT	S6	40-63	Subsystem Control Table address
FC	S7	52-54	Function code 0 Clear 1 Send packet 2 Receive packet

This request invokes the I/O Subsystem driver called the IOP driver. Before a task uses this request to perform I/O, the IOP driver must be linked to the STP resident table called the Subsystem Control Table (SCT). This linking is accomplished when the task issues the first clear PIO request. The SCT address must never change thereafter. A task monitors the status of the subsystem by inspecting the status field (SCSTAT) of the SCT table. Three flags are maintained by the IOP driver in the SCSTAT field. They are as follows:

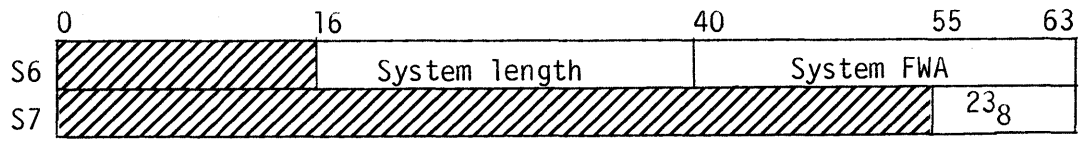
- SCDOWN=1 I/O Subsystem Down flag
- SCRST=1 I/O Subsystem Reset flag
- SCIR=1 Input Ready flag

Flag SCDOWN is cleared and flag SCRST is set by the IOP driver when the I/O Subsystem has been restarted or initialized. A task can then acknowledge reset by issuing a PIO clear request which clears the SCRST flag. The driver cannot accept a clear until all input has been processed which means the flag SCIR must be clear.

Sending or receiving a packet requires that a packet address (SCCIP) and packet size in words (SCPSZ) be passed in the SCT table. In general a packet can be received when the SCIR flag is set, and a packet can be sent when all status flags are clear.

Move system down to execution area request (MVEDWN=23)

Request format:



This request moves an image of an operating system down to the executable area. The processing flow is:

1. Build an exchange package at location 0, with monitor mode set and with P as the first instruction in the move-down loop.

2. Consider that exchange package to be the current one and activate it.

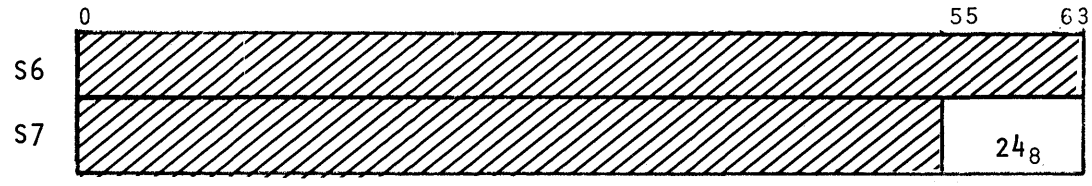
The move-down flow is:

1. Starting with the high address of the system, perform full-length vector transfers.

2. Assign the exchange package at location 0 and exchange to it.

Start system request (START=24)

Request format:



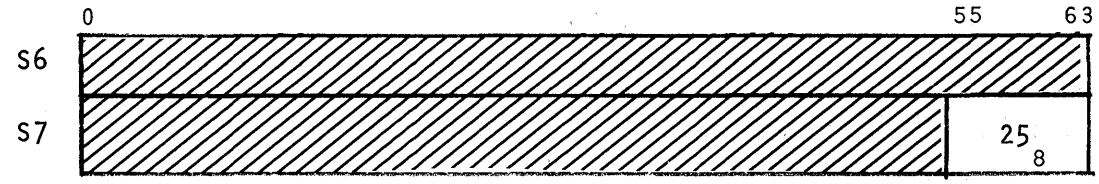
This request starts the system after a system breakpoint is encountered or after a stop function has been issued. The flow is:

1. Clear alternate task scheduling flag that forced system to idle except for external requests to the station (SCP).

2. Request execution of the task scheduler.

Stop system request (STOP=25)

Request format:

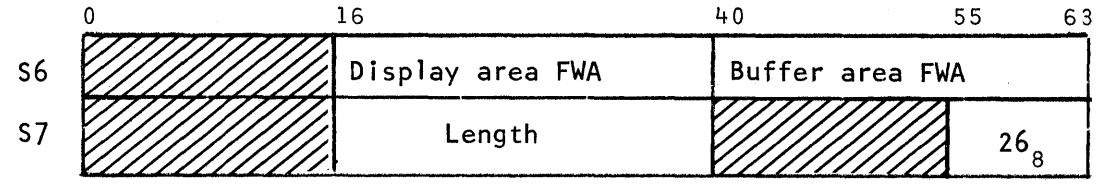


This request stops the system except for entry of interactive debugging commands. The processing flow is:

1. Set alternate task scheduling flag. The alternate scheduling allows only SCP to execute so interactive debugging commands can be entered.

Display memory request (DMEM=26)

Request format:

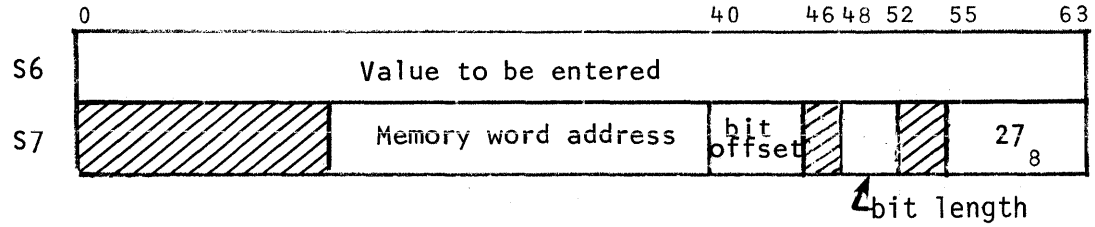


This request copies memory to a specified area. It is used to display memory during interactive debugging. The processing flow is:

1. Move the memory block from the requested area to the display buffer.

Enter memory request (EMEM=27)

Request format:



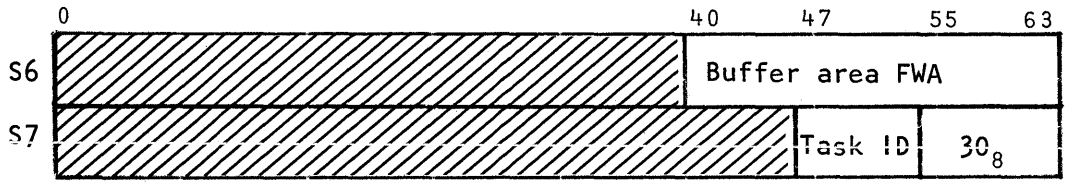
This request enters the bit string into memory at the specified bit position. The flow is:

1. Shift value right (64 - bit offset - bit length).

2. Merge into the memory address.

Display exchange package request (DXPR=30)

Request format:



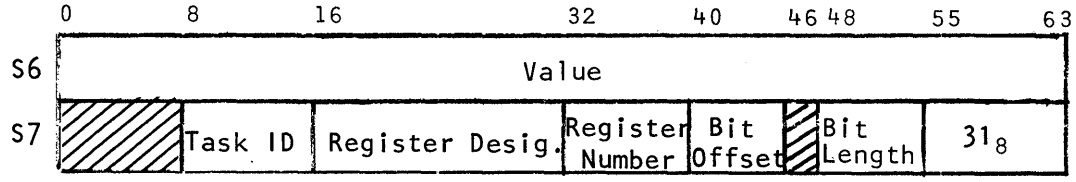
This request moves the contents of the exchange package and B0 to a buffer. The processing flow is:

1. Copy exchange package to words 0 through 15 of buffer.

2. Copy (B0) to word 16 of buffer from the task B0 Save Table.

Enter exchange package register request (EXPR=31)

Request format:



This request inserts the bit string into the specified exchange package register. The flow is:

1. Determine word length and position of specified register in memory.

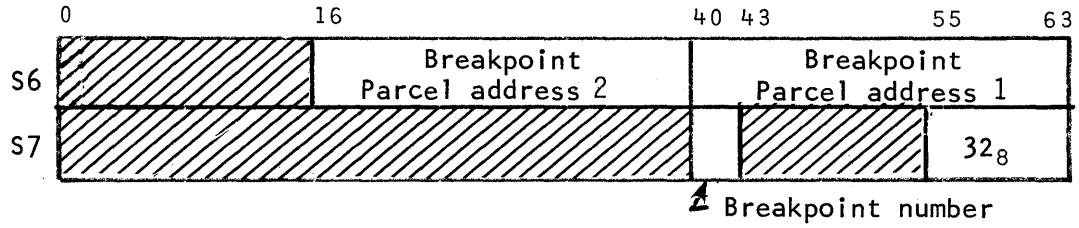
2. Shift value right to desired position.

3. Merge into memory address.

Register designators can be any of those noted in COS Front-end Protocol Internal Reference Manual, publication SM-0042, Debug Function Request (027_8).

Set system breakpoint request (SBKPT=32)

Request format:

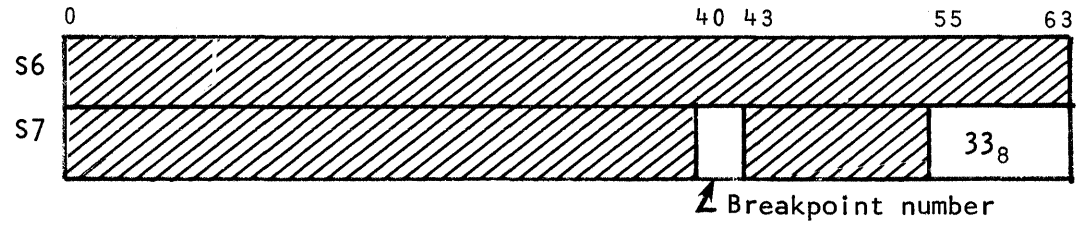


This request sets a single or double breakpoint in the system by changing an instruction parcel to an illegal value. If a breakpoint exits at the address, an error is reported. The double breakpoint allows for automatic resetting of the initial breakpoint when the second breakpoint is encountered. Up to eight system task breakpoints are allowed. The processing flow is:

1. Verify breakpoint number.
2. Verify breakpoint number not in use.
3. Verify memory address not already in Breakpoint Table.
4. Store information in task breakpoint table.
5. Save breakpoint instruction parcel.
6. Set breakpoint.

Clear system breakpoint request (CBKPT=33)

Request format:

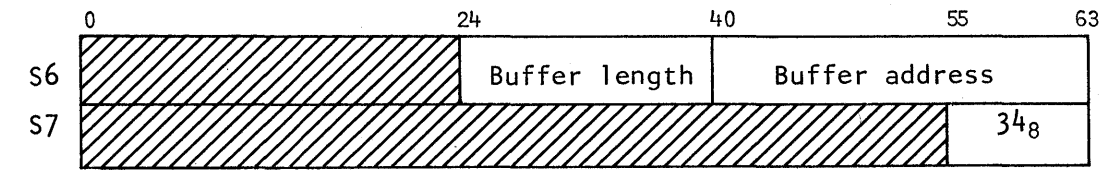


This request clears a system task breakpoint entry. The processing flow is:

1. Verify breakpoint number.
2. Verify breakpoint number in use.
3. Determine which of two possible breakpoint addresses is active.
4. Restore instruction parcel at the active addresses.
5. Clear breakpoint table entry.

Report CPU usage request (CPUTIL=34)

Request format:



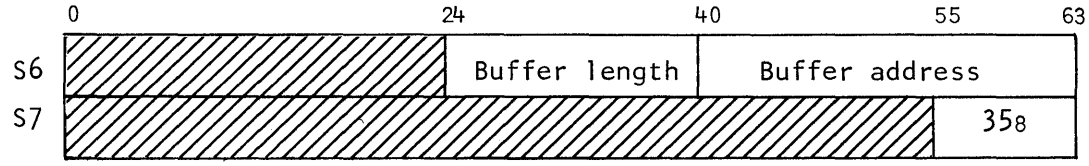
This request puts CPU usage data into the assigned buffer The processing flow is:

1. Validate buffer size.

2. Fill the buffer with CPU usage data, zeroing the fields in EXEC that collect such data.

Report task usage request (TASKUTIL=35)

Request format:



This request puts task usage data into the assigned buffer. The processing flow is:

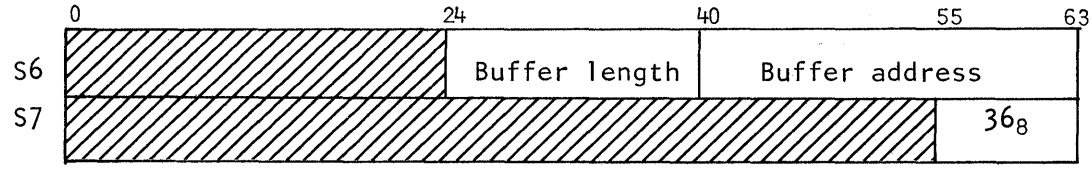
1. Validate buffer size.

2. Put number of tasks into buffer.

3. Put number of readies of each task into buffer, zeroing the fields in the STT that collect such data.

Report EXEC request (EREQNT=36)

Request format:



This request puts the EXEC request count of each task into the assigned buffer. The processing flow is:

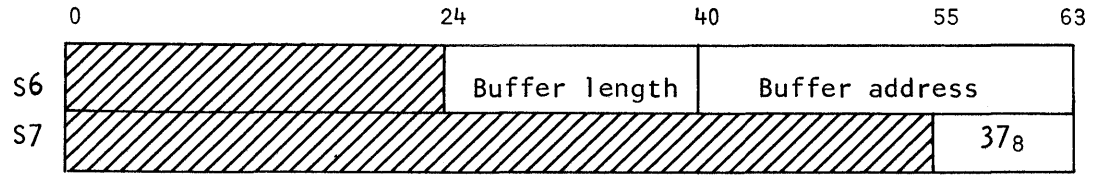
1. Validate buffer size.

2. Put number of tasks into buffer.

3. Put number of requests made by each task into buffer, zeroing the fields in the STT that collect such data.

Report EXEC call counts request (ECALLCNT=37)

Request format:

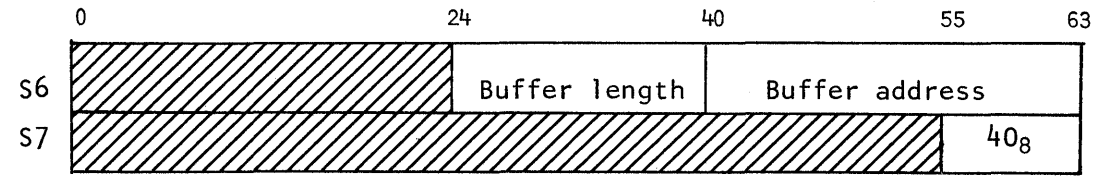


This request puts the number of EXEC requests of each type into the assigned buffer. The processing flow is:

1. Validate buffer size.
2. Put number of task EXEC request types into buffer.
3. Put number of requests of each type into buffer, zeroing the fields in the STT that collect such data.

Report interrupt counts request (CINTCNT=40)

Request format:



This request puts interrupt counts of each channel and pseudo channel into the assigned buffer. The processing flow is:

1. Validate buffer size.
2. Put number of interrupt channels into buffer.
3. Put interrupt count of each channel into buffer, zeroing the table entries that collect such data.

2.6.2 EXEC ERROR CODES

EXEC returns one of the following error codes in register S6 if a request cannot be processed. The P register is not incremented in this case.

Code (octal)	Significance
1	No task space left
2	No task assigned
3	Task does not exist
4	Resource already assigned to a task
5	Channel already active
6	Illegal task call
7	Input side of channel active
10	Output side of channel active
11	Illegal breakpoint number
12	Address already has a breakpoint
13	Bad field definition
14	Job already connected
15	Disk malfunction
16	Breakpoint invalid
17	Device does not exist
20	Illegal register name
21	Equipment not in system
24	Time queue is full
25	Insufficient buffer length

2.7 FRONT-END DRIVER

The front-end driver (FED) physically controls I/O between front-end computer systems and STP. It performs all hardware error recovery, while STP provides all logical error recovery.

FED supports configurations allowing up to four front-end channels with multiple front-end computer systems attached to each channel. These multiple front-end systems may be attached either directly to the channel or may be remote batch entry stations attached to the front-end system.

The front-end driver also intercepts and processes maintenance control unit (MCU) request messages. The format and function of these messages is described in CRAY-OS Message Manual, publication SR-0039.

2.7.1 THEORY OF OPERATION

The front-end driver is a slave to the Station Call Processor (SCP) in that all external interrupts are rejected on front-end channels until SCP requests an output/input (O/I) pair. The driver then becomes the passive partner of the front-end computer system.

Error recovery is initiated by the front-end computer system. FED only reports input hardware errors to the front-end system and retransmits output messages on request from the front-end system.

2.7.2 SYSTEM TABLES USED BY FED

FED uses the following system tables:

CHT Channel Table
LIT Link Interface Table
LXT Link Extension Table
CXT Channel Extension Table

Detailed information on these tables is available in the COS Tables Descriptions Internal Reference Manual, publication SM-0045.

Channel Table (CHT)

The front-end driver sets the link interface table address and requesting task parameter block address into the channel table (CHT) when the O/I is initiated, and clears them when the O/I completes. FED sets the interrupt processor address for the next interrupt in the processing sequence.

Link Interface Table (LIT)

FED uses the Link Interface Table to determine the type of O/I request, to communicate O/I completions to SCP, to maintain usage statistics, and to maintain auxiliary control information.

Link Extension Table (LXT)

FED uses the LXT to validate source ID and obtain parameters for control of checksumming.

Channel Extension Table (CXT)

FED uses this table to build output message for I/O Subsystem.

2.7.3 PROCESSORS

The front-end driver is composed of a request processor and the following set of re-entrant interrupt processors.

RLCP — Processes interrupt from input of Link Control Package (LCP). If channel error flag, RLCP sets up WXLCP to process output interrupt from error LCP. If the LCP is a hardware error LCP, RLCP restarts driver; otherwise, it checks input LCP for following subsegments. If no subsegments exists, RLCP terminates the O/I operation or sets up for receipt of LTP. When there are subsegments, RLCP sets the channel to receive the first subsegment and requests RSSEG to process the next interrupt on this input channel.

RSSEG — Processes interrupt from input of subsegment. If a channel error occurs, RSSEG requests WXLCP to process next interrupt on output side and resets RLCP to process next interrupt on input side. When there are no errors and no LTP is expected, RSSEG readies the requesting task if there are no more subsegments; otherwise, it sets up to receive the next subsegment or LTP.

RLTP — Processes interrupt from input of Link Trailer Package (LTP).

WLCP — Processes interrupt from output of LCP. If a subsegment is to follow, WLCP sets the channel to send the first subsegment and requests WSSEG to process the next interrupt on this output channel. If an LTP is to follow, WLCP sets the channel for a data transfer and requests WLTP to process the interrupt.

WSSEG — Processes interrupt from output of LCP. If another subsegment remains to be sent, WSSEG sets up to send the next subsegment and requests itself to process the interrupt; otherwise, WSSEG sets up for an LTP and requests WLTP to process the interrupt. If there is to be no LTP, it clears output active in the LIT and sets reject on the output channel.

WLTP — Processes interrupt from LTP and clears output active and sets reject on the output channel.

WXLCP — Processes interrupt from output of hardware error LCP. WXLCP clears output active and sets reject on output channel. If an LTP is to follow, WXLCP sets the channel for a data transfer and requests WXLTP to process the interrupt.

WXLTP — Processes interrupt from output of error LTP.

APIIP — Processes MIOP status input interrupt

APOIP Processes MIOP command output interrupt

APPACN Null packet processor

APPACX STP task packet processor

APPACE Echo packet processor

APPACI I-packet processor

APPACJ J-packet processor

APPACA Disk packet processor

APPACB Station packet processor

SHSRQT BIOP simulated memory channel request processor

SHSOT BIOP simulated memory channel output processor

SHSIT BIOP simulated memory channel input processor

2.8 DD-19/29 DISK DRIVER

The disk driver drives either a DD-19 or a DD-29 disk storage unit connected to a CRAY-1 I/O channel. The disk driver executes in monitor mode as that part of the Executive Request Processor devoted to processing the I/O Executive request. This request is described in section 2.6.1.

2.8.1 R01l

The disk driver is labeled R01l in EXEC. R01l receives control when an STP task executes an EX instruction with an I/O function code in S7. The contents of S6 and the remainder of S7 specify parameter addresses relative to STP. The parameters at these addresses completely specify the current request to R01l.

R01l does no I/O until it has checked the legality of the request parameters. If the request parameters are legal, R01l activates the CRAY-1 channels and disk hardware specified by the request. Channel activation consists of the sending of data and disk hardware functions and the receiving back of data and disk hardware status.

An illegal value causes ROll to schedule its caller to resume execution at the parcel immediately following the EX instruction. A legal request causes ROll to schedule its caller at the third parcel following the EX instruction.

ROll is interrupt driven and executes a request in short bursts. Upon activating the I/O hardware, ROll gives control to Interchange. It regains control when the I/O hardware generates an interrupt on the CRAY-1 channel involved.

ROll keeps its caller STP task aware of the progress of the request after each transfer of a sector and at the completion of the request.

Lost interrupts

An interrupt could fail to occur due to a hardware failure. Therefore, ROll protects itself by scheduling a timeout interrupt for each request to ROll. As a result, each execution of Interchange compares the current contents of RTC (Real-Time Clock) to the timeout value. Interchange gives control to ROll if the timeout occurs.

Since it is possible that exchanges or other interrupts might not occur for an extended period, the MCU real-time interrupts to the CRAY-1 should be enabled to ensure frequent execution of Interchange. The time delay scheduled for timeout reflects the magnitude of the request to ROll while being liberal enough to avoid needless timeouts.

A single ROll request may involve many interrupts; thus, the single timeout scheduled per ROll request acts as a blanket protection.

Lost interrupts are rare, generally only expected interrupts occur.

When the request completes, the timeout is released.

Multiprogramming

Between the short bursts of ROll activity, Interchange finds other work for the CPU. Therefore, user programs, STP, and even other parts of EXEC are multiprogrammed with ROll. A trace of COS activity generally shows considerable non-ROll activity between disk interrupts.

ROll also helps facilitate CPU-I/O overlap for user programs by helping them to take advantage of each sector of transfer when that sector completes as well as when the entire request completes.

The user program is taken out of recall as soon as possible so that the user program may immediately begin to process the next block of data.

I/O requests by user programs come to R011 via the disk queue manager task (DQM) of STP. DQM queues these user calls, issuing corresponding requests to R011 in a sequence that optimizes system throughput; that is, seeks are optimized. DQM passes to R011 the address of the DNT for the dataset involved.

Status checking and error recovery

R011 checks hardware status at the start of each request, during the request, and at the completion of the request.

R011 notifies the calling STP task when a request completes whether successfully or in error. To effect error recovery, the calling task must make the appropriate calls to R011.

R011 contains features explicity for error recovery. R011 performs the servo offset and data strobe functions if requested by the calling task.

Callers

The Disk Queue Manager is the only STP task that calls R011. R011 rejects any call that would interfere with an ongoing request, except for the call to master clear the disk controller and the CRAY-1 input and output channels to which it is connected.

2.8.2 HARDWARE SEQUENCES FOR SAMPLE REQUESTS

This subsection assumes that the reader is familiar with the CRAY-1 DCU-2, DCU-3 Reference Manual, publication 2240630. The processing sequence for several requests are presented here.

Multiple sector read

If another DSU is selected, release connected DSU.

If the DSU is not selected, then,

Set unit selected flag,
Wait 5 usec after last head select,
Activate output channel to send unit reserve function to disk control unit, and
Await interrupt causes by output channel completion.

1.
- If in error recovery mode, then,
  - Activate output channel to send status readout function (requesting subsystem status) to disk control unit,
  - Await interrupt caused by output channel completion,
  - Activate input channel to receive subsystem status from disk control unit,
  - Await interrupt caused by input channel completion,
  - If subsystem status is bad, then go to error recovery,
  - If not desired cylinder, then,
    (1) Update current cylinder to desired cylinder,
    (2) Activate output channel to send cylinder select function to disk control unit, and
    (3) Await interrupt caused by output channel completion.
- If sectors to transfer equal 0, then go to 2.
- If retry is enabled, then,
  - Read continuity is broken.
  - Activate output channel to send margin select function to disk control unit, and
  - Await interrupt caused by output channel completion.
- If read continuity is broken, then,
  - Await 5 usec after last head select.
  - Activate output channel to send read function to DCU,
  - Await interrupt caused by output channel completion,
  - Activate input channel to receive read-write response from DCU,
  - Await interrupt caused by input channel completion, and
  - If read-write response is bad, then go to error recovery.
- Activate input channel to receive data block from DCU-2.
- Await interrupt caused by input channel completion.
- If channel error, get subsystem status, then go to error recovery.
- Update tables.
- If sectors to transfer equal 0, then go to 2.
- If recall flag in DNT is set, then ready caller task.
- Go to 1.

2.
- Activate output channel to send status readout function (requesting subsystem status) to DCU.
- Await interrupt caused by output channel completion.
- Activate input channel to receive subsystem status from DCU.
- Await interrupt caused by input channel completion.
- If subsystem status is bad, then go to error recovery.
- Update tables.
- Ready caller task.

Multiple sector write

A multiple sector write resembles the multiple sector read; however, retry is disabled implicitly and write continuity is checked. Either a margin select function or a read function destroys write continuity. A write function destroys read continuity.

Cylinder select

A cylinder select resembles a multiple sector read or write except that *sectors* to transfer are 0 on entry to the driver.

Controller master clear

1. Clear reserved bits in all PUTs for the channel.
2. Clear PUTs to force unit select and seek.
3. Master clear channel with recommended I/O master clear sequence.
4. Reserve unit.
5. Send subsystem status.
6. Open input channel for one parcel immediately.
7. If input interrupt preceeds output interrupt for subsystem status, reject further input interrupts until output interrupt.
8. If subsystem status is not ready
   - Send fault status function,
   - Release unit with fault clear bit set.
   - If fault status shows a seek error, perform clear fault and return to zero seek.
   - Reserve unit.
9. If subsystem status is ready, return to caller; otherwise, retry clearing procedure until it is successful.

Unit release

1. Activate output channel to send unit release function to DCU-2.
2. Await interrupt caused by output channel completion.
3. Update tables.
4. Ready caller task.

Margin select

The margin select algorithm selects margins in the following sequence:

1. Late strobe, maximum offset toward center (position 37_8)
2. Late strobe, maximum offset toward edge
3. Early strobe, maximum offset toward center
4. Early strobe, maximum offset toward edge
5. Maximum offset toward center
6. Maximum offset toward edge

7-12. Same strobe and direction but with offset position 24_8

13-18. Same strobe and direction but with offset position 11_8

2.9 I/O SUBSYSTEM DRIVER

The I/O Subsystem driver, called the IOP driver, is responsible for controlling all normal I/O channels between the CRAY-1 CPU and its I/O Subsystem.

2.9.1 FUNCTIONAL DESCRIPTION

All information passed through the driver is in the form of a six 64-bit word packet. The general format of a packet as well as specific formats can be found in the COS Table Descriptions Internal Reference Manual, publication SM-0045. The tables to be concerned with are:

ADC I/O Processor Disk Command
ASC IOP Station Command
APT Any Packet Table

Processing of a packet by the driver is determined by the packet's source ID (SID). Possible SIDs are listed below.

SID	Definition and processing
C1	Any packet to be queued for output to the I/O Subsystem
A	Disk request reply packet is passed to disk driver for processing.
B	Station request packet passed to station driver for processing
C	IOP message packet (Not implemented)
D	Reserved for expansion
E	Echo packet; the source and destination IDs are swapped and the packet queues for output to the I/O Subsystem.
I	Initialize channel; subsystem downed; output queue cleared and packet echoed to sender
J	Acknowledgement of channel initialization; up the subsystem and echo the packet.
N	Null Packet; packet discarded and no further processing.

STP tasks interface to the IOP driver via the PIO monitor request (PIO=22) as described in section 2.6.1 of this manual.

Tables used internally to the IOP driver are described as follows.

Channel Extension Table (CXT)

The station driver interfaces to the IOP driver through the CXT table. This table holds some control information for the station driver as well as the station command packet (ASC).

Queue Control Table (QCT)

This table is used internally by the IOP driver to maintain an input queue to each STP task and an output queue to each I/O Subsystem.

Subsystem Control Table (SCT)

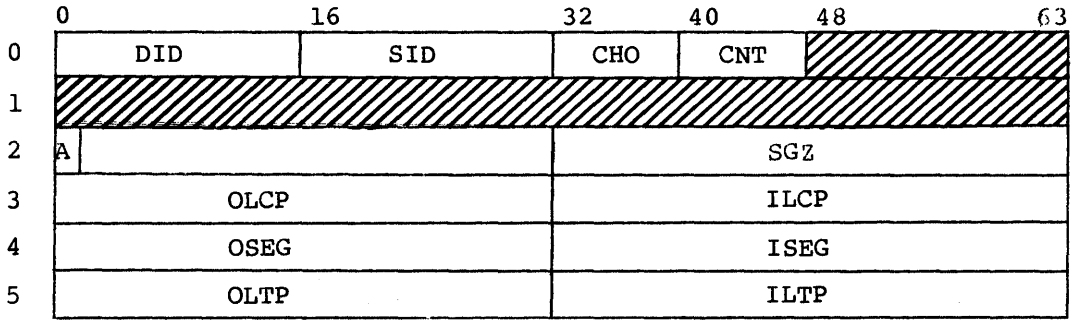
This table resides in EXEC and is used by the driver to manage I/O with the I/O Subsystem. A duplicate of this table is required in STP for each task which is using the IOP driver. The tables are linked when the first IOP driver request is made. The task can then monitor the subsystem status in its SCT table.

2.9.2 RECOVERY

The IOP driver does not handle the recovery of any requests. It will only notifies each requester when a subsystem has gone down and comes up again.

2.9.3 MIOP COMMAND AND STATUS PACKET FORMATS

Station packet format



Field	Word	Bits	Description
DID	0	0-15	Destination ID
SID	0	16-31	Source ID
CHO	0	32-39	Channel ordinal
CNT	0	40-47	Message number
A	2	0	Address request flag. Set by MIOP when requesting next set of addresses only. When this field is clear, SCP will be initiated.
SGZ	2	32-63	Segment size. Set by EXEC; MIOP does not need to return these values.

Field	Word	Bits	Description
OLCP	3	0-31	Output LCP address. Set by EXEC; MIOP does not need to return these values.
ILCP	3	32-63	Input LCP address. Set by EXEC; MIOP does not need to return these values.
OSEG	4	0-31	Output segment address. Set by EXEC; MIOP does not need to return these values. If 0, there is no segment or LTP.
ISEG	4	32-63	Input segment address. Set by EXEC; MIOP does not need to return these values. If 0, there is no segment or LTP.
OLTP	5	0-31	Input LTP address. Set by EXEC; MIOP does not need to return these values. If 0, there is no segment or LTP.
ILTP	5	32-63	Input LTP address. Set by EXEC; MIOP does not need to return these values. If 0, there is no segment or LTP.

The station command protocol interleaves with the standard station protocol in the following cycle:

1. At polling interval, MIOP sends station packet with address request flag set.

2. CPU returns addresses to MIOP.

3. CPU input LCP and segment are transfered over the memory channel.

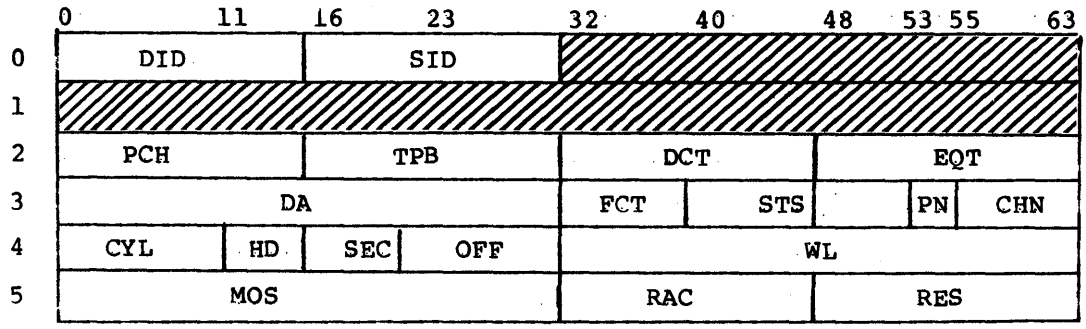
4. MIOP sends station packet with address request flag clear, causing CPU to execute station task.

5. CPU returns addresses to MIOP.

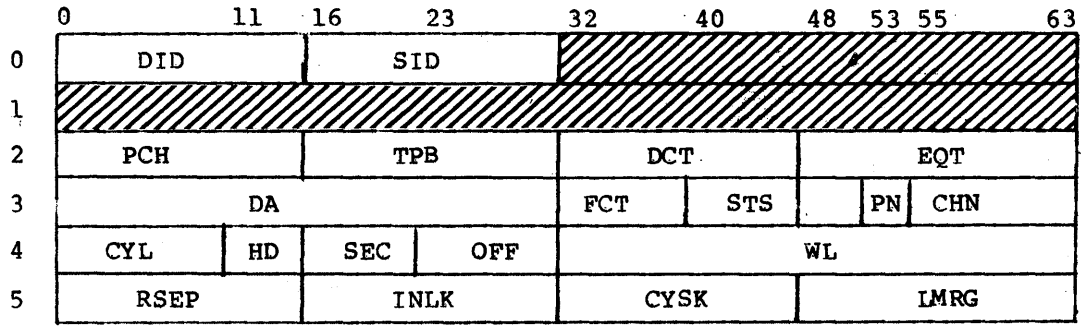
6. CPU output LCP and segment are transfered over the memory channel.

Disk packet format

Request:



Reply:



Field	Word	Bits	Description
DID	0	0-15	Destination ID
SID	0	16-31	Source ID
PCH	2	0-15	Pseudo channel number
TPB	2	16-31	Task parameter block address
DCT	2	32-47	DCT address
EQT	2	48-63	EQT address
DA	3	0-31	Bipolar data address
FCT	3	32-39	Function
STS	3	40-47	Logical status
---	3	48-52	Unused

Field	Word	Bits	Description
PN	3	53-54	IOP Processor number
CHN	3	55-63	IOP channel number
CYL	4	0-10	Cylinder
HD	4	11-15	Head
SEC	4	16-22	Sector
OFF	4	23-31	Word offset
WL	4	32-63	Word length
MOS	5	0-31	MOS data address
RAC	5	32-47	Read ahead count
RES	5	48-63	Reserved for IOP
RESP	5	0-15	Error flags
INLK	5	16-31	Interlock status
CYSK	5	32-47	Cylinder status
LMRG	5	48-63	Last margin

The disk packet command protocol interleaves with data flow over memory channel with the following cycle.

1. CPU sends MIOP the disk request.
2. Data transfers over memory channel.
3. MIOP sends the disk status reply.

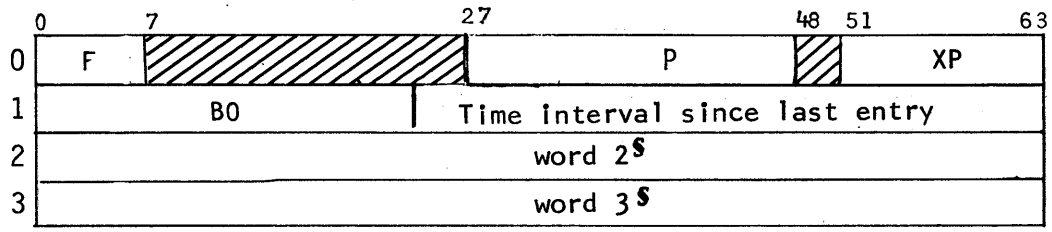
Note that only one request is outstanding to the IOP on each pseudo channel.

2.10 EXEC DEBUG AIDS

EXEC has two debugging aids: history trace and the stop buffer.

2.10.1 HISTORY TRACE

History trace is an EXEC-resident routine composed of 4-word messages entered in a circular buffer by the subroutine DEBUG. The buffer begins at location DBF and has room for 1024 messages before previous messages are overwritten. DEBUG maintains the table offset to the next message address at location DBFP. The general format of a trace message address is as follows:



Field	Word	Bits	Description
F	0	0-6	Function number
P	0	27-48	P register of interrupted exchange package
XP	0	51-63	Exchange package address
B0	1	0-23	Contents of B0 of interrupted task
	1	24-63	Time interval since last entry

The purpose of each trace type and the contents of the two registers are defined in table 2-1. The function trace tool changes frequently; check listings for current formats.

The function trace may be selectively or collectively enabled. To enable all functions, set location DBUGM nonzero. To selectively enable a function clear (DBUGM) and set (DBUGM plus the function number). Any combination of functions may be enabled at a given time. Disabling is accomplished by clearing the location that enabled the debug function. STP functions (12 and 21 through 32) are further selected through the POST micro acting together with the POST macros occurring throughout STP. An STP function not listed in the POST micro in the early part of STP is disabled and can be re-enabled only through reassembly or by patching the code generated by a particular POST macro.

In table 2-1, disabled functions are indicated with a § flag. Functions 1-11 and 13-20 are used by EXEC and are not affected by the POST micro.

---

§ Refer to table 2-1 for the contents of words 2 and 3.

Use the following macro to make a trace entry from a task in STP. This example assumes that 0'77 is the function number and S2 and S3 contain the information to be captured. Note that any register values other than S0 and S7 can be used instead.

Location	Result	Operand	Comment
1	10	20	35
	POST	0'77,S2,S3	

In EXEC, to perform a history trace, place the information of interest in S6 and S7 and execute the following:

Location	Result	Operand	Comment
1	10	20	35
	A5 R	0'77 DEBUG	function number

The history trace is easily expandable so new function types may be added. New DEBUG function numbers may be assigned up to a maximum value of 77 octal.

Table 2-1. History trace functions

Function Number	Event causing trace entry	Contents of words 2 and 3
1	I/O trace interrupt	7/channel error flag, 9/channel number, 24/fwa of CBxx, 24/fpa 64/1st word of relevent PUT table if disk
2	User-initiated normal exchange	64/user S0 64/user S1
3	STP-initiated normal exchange	64/task S6 64/task S7
4	EXEC-initiated normal exchange	64/MCPT[$] 64/SAXP[$$]

[$] MCPT = 1/task scheduler request flag, 51/not used, 12/task parameter block address (zero if user program was interrupted)

[$$] SAXP = 13/JXT offset, 24/JTA address, 15/time slice in milliseconds, 12/XP address (constant)

Table 2-1. History trace functions (continued)

Function Number	Event causing trace entry	Contents of words 2 and 3
5	Real-time interrupt	64/MCPT 64/SAXP
6	Copy user XP to JTA	64/MCPT 64/SAXP
7	Station input interrupt for LCP	64/LCP+0 64/LCP+1
10	User is being given control of CPU	7/not used, 9/XP flags, 24/not used, 24/nonzero if user XP is in JTA 64/SAXP
10	Input SCBs received	64/input stream control bytes 64/output stream control bytes
11	Physical disk I/O request	16/transfer length, 24/disk address, 24/buffer address 1/transfer direction, 54/unused, 3/1, 4/software channel number,2/unit number
11	Physical disk I/O retry first message	16/last function, 24/last status, 24/first parcel address 16/edited status
11	Physical disk I/O retry second message	16/retry length, 24/disk address, 24/memory address 16/, 24/CA IN, 24/CA OUT
11	Physical disk I/O channel error	16/channel no, 24/desired CA, 24/actual CA 64/'CA ERROR'
12	Intertask message	64/input word 0 or output word 0 64 input word 1 or output word 1
13	Error exchange	64/MCPT 7/unused, 9/XP flags, 40/unused
14	Station output interrupt for LCP	64/LCP+0 64/LCP+1

Table 2-1. History trace functions (continued)

Function Number	Event causing trace entry	Contents of words 2 and 3
15	Input/output segment termination	63/, 1/input active 63/, 1/output active
16	Input SCBs received	64/input stream control bytes 64/output stream control bytes
17	Station output interrupt for error LCP	63/, 1/input active 63/, 1/output active
20	Output SCBs sent	64/input stream control bytes 64/output stream control bytes
21§	User's time slice expired	24/'RTC', 40/number of RTC interrupts 64/RTC interrupt word (JRTCWORD)
22§	Job being initiated	64/ASCII job name from SDT 16/priority, 24/job size (including JTA), 24/time limit in seconds
23§	Job being reactivated after a time elapsed or an event occurred	64/ASCII job name 64/RTC value, or the event comparison value
24	Job status change	8/0, 56/Old JXSTCH (with "-->" if room) 56/New JXSTCH, 8/JXT ordinal (1-0'77)
25§	Search for a free memory segment	64/JXSTCH for job that needs memory; 64/size of free segment sought
25§	Allocation of a memory segment	64/MST entry§§ for the free segment from which the allocation is to be taken; 64/number of words to be allocated

§ Entries disabled in default system

§§ MST entry = 16/JXT ordinal, 24/segment size, 24/segment address.

Table 2-1. History trace functions (continued)

Function Number	Event causing trace entry	Contents of words 2 and 3
26§	Liberation of a memory segment	64/offset of the MST entry§§ for the segment to be freed; 64/the MST entry itself
27§	User job about to be connected	64/job name 64/'GOT CPU'
27§	Disconnected user job losing X status	64/Job name 64/'HAD CPU'
27§	Disconnected user job about to be reconnected	64/Job name 64/'KEPT CPU'
27§	CPU going to idle state	64/'CPU IDLE', 64/JXTPOP in the first trace entry; for each job in the JXT, another function-27 entry follows, containing: 64/Job name, 56/JXSTCH, 8/JXORD
30§	Request received by JSH	40/ASCII function name, 24/JXT ordinal 64/ASCII job name
32	J$ALLOC request's initial processing done	64/address of memory request word in STP 64/memory request word itself
32	Entry to MOVEMEM routine (trace entry suppressed if no data will be moved)	16/'MV', 24/from-address, 24/from-length, 40/to-address, 24/to-length
32	Entry to ERASEMEM routine (trace entry suppressed if no data will be erased)	64/'ERASE', 40/address, 24/length of area to be erased
32	Exit from RELOCATE routine (Always two trace entries: before and after relocating)	Values before relocating: 22/HLM 21/LFT, 21/DSP, 22/BFB, 21/buffer boundary, 21/FL Values after relocating: 22/HLM, 21/LFT, 21/DSP, 22/BFB, 21/change in FL, 21/FL

§ Entries disabled in default system

2.10.2 SYSTEM CRASH MESSAGE BUFFER

This error reporting feature of EXEC assists the computer operator or system analyst in finding the general cause of a system crash. When EXEC has detected a fatal error condition, a STOP message is built in a buffer called the stop buffer. This buffer is located in EXEC at B@STOP, which is located just before the history trace buffer, is loaded with the label in EXEC where the error was detected, the word address of P and B0, and a stop message in ASCII. The buffer is dumped with the following format.

```
*==============================*
!    S T O P   B U F F E R     !
*==============================*
EXEC STOPPED AT LABEL:  $STOP006
  W.P =           W.B0 =
--------------------------------
EEF - OPERAND RANGE ERROR
------------END BUFFER ---------
```

The stop label is the label used in EXEC to call the STOP macro. The values in P and B0 are not converted to ASCII characters, so their values appear in the dump. The value of P is in the word after the word containing W.P and the value of B0 is in the word after the word containing W.B0. Remember these two values have been truncated to words.

A convention is used for STOP labels and messages. The label has the form $STOP*ec*, where *ec* is a unique decimal number for each error condition. The stop message contains the routine name where the stop occurred and a short descriptive error message. EXEC stop messages are shown in table 2-2.

Table 2-2. EXEC stop messages

Label	Code	Significance
$STOP000	EEF	Unknown Error
$STOP001	DEQRT	Invalid Time Index
$STOP002	DEQRT	Parameter Word Mismatch
$STOP003	DEQRT	Time Queue Empty
$STOP004	EE	Program Address Range Error
$STOP005	EEF	Floating-point Error
$STOP006	EEF	Operand Range Error
$STOP007	EEF	Program Range Error
$STOP008	EEF	STP Error Exit (See STP Hang Message)
$STOP009	ENSEC1	Time Queue Full
$STOP010	T11CLCA	Illegal Disk Channel Selected
$STOP011	NER	Uncorrectable I/O Read Memory Error
$STOP012	EE	Double Bit Memory Error
$STOP013	EEF	Memory Error
$STOP014	MC	Disk Controller Not Responding

STOP-macro to hang EXEC

The STOP macro calls routine $STOP when the selected condition is true. Otherwise execution continues after the macro call. Routine $STOP builds the message in the STOP buffer, restores all the registers to their values before the STOP macro was entered, and then hangs in a tight loop.

Format:

Location	Result	Operand
p_0	STOP	p_1,(p_2)

p_0 Label field is required.

p_1 Stop condition argument. EXEC hangs when this condition is true.

Condition	Significance
UC	Unconditional stop
SZ	Stop if S0 zero
SN	Stop if S0 not zero
SP	Stop if S0 positive
SM	Stop if S0 negative
AZ	Stop if A0 zero
AN	Stop if A0 not zero
AP	Stop if A0 positive
AM	Stop if A0 negative

p_2 Message argument; string of 1 to 64 characters enclosed by parentheses.

2.11 INTERACTIVE SYSTEM DEBUGGING

The executive requests described in section 2.6.1 provide the mechanism through which interactive system debugging control passes from the user to SCP to EXEC. The debugging capability provides for memory entry and display, operating register entry and display, setting and clearing breakpoints, and starting and stopping the system.

The operator debug commands that use this capability are described in the COS Operational Procedures Reference Manual, publication SM-0043.

SYSTEM TASK PROCESSOR (STP) 3

3.1 GENERAL DESCRIPTION

The System Task Processor (STP) consists of tables, a set of routines called tasks, and some re-entrant routines common to all tasks.

A task is a routine that serves a specific purpose and usually recognizes a set of subfunctions that can be requested by other tasks. Characteristics of a task are that it has its own ID (a number in the range $0\text{-}35_8$), an assigned priority ($000\text{-}377_8$), its own exchange package area in the System Task Table (STT), and its own intertask communication control table which defines which tasks are allowed to communicate.

The addresses of the Base Address (BA) register and Limit Address (LA) register are the same for all tasks; BA is set to the beginning of STP and LA is set to I@MEM (an installation defined maximum memory value).

Although a task is loaded into memory during system startup, it does not normally become known to the system until an existing task issues an executive request for the creation of some other task. COS Startup is the necessary exception. A "create task" request assigns an ID and a priority to a task via the task's parameter block in the STT.

Tasks execute in program mode and are thus interruptible. An interrupt may occur as a result of the task executing an exit instruction (ERR or EX) or may result from one of the interrupt flags being set automatically (e.g., an I/O interrupt occurred).

When a task is created, it is forced into execution. During this initial execution, it usually performs some initialization and setup operations and then suspends itself. Thereafter, a task is executed only if it is readied. Readying of a task consists of altering its suspend bit. It is not a candidate for execution, however, unless all of the bits in its status field are 0, including the breakpoint and stop bits.

Task readying occurs automatically or explicitly. Readying occurs automatically for tasks assigned to a channel when an interrupt occurs on the assigned channel. Readying of a task may also occur as a result of an explicit EXEC request issued by one task for the execution of another task. A task may be readied or suspended by a System Operator Station request (Station Debug Command). A task remains ready (unless breakpointed or stopped) until EXEC receives a request to suspend it.

A task requests self suspension when it has completed an assigned function or posts a request for another task. Note that if the task being requested is of lower priority than the task making the request, the requesting task must suspend itself to allow the lower priority task to execute.

Subsequent requests to ready a task already readied cause the ready request bit in the task's parameter word to be set. When this bit is set, the next suspend request for the task causes the task to be rereadied rather than suspended. The task ready request bit is then cleared.

3.2 TASK COMMUNICATIONS

Tasks may communicate with EXEC, with each other, and with user jobs.

3.2.1 EXEC/TASK COMMUNICATION

A task communicates with EXEC by placing a request and parameters in registers S6 and S7 and by executing an EX instruction. When a task executes an EX, the error return is to the instruction following the EX; the normal return is to the instruction following the error return. The error return instruction must be a 2-parcel instruction. A reply to the request is returned in S6 and S7.

EXEC requests are described in detail in section 2.6.

3.2.2 TASK-TO-TASK COMMUNICATION

STP contains two areas used for intertask communication. The first area is the Communication Module Chain Control (CMCC); the second area is the Communication Module (CMOD).

The CMCC is a contiguous area containing an entry for each combination of tasks possible within the system. The CMCC is arranged in task number sequence, that is, all possible task 0 combinations of requests to task 0 are followed by all possible combinations of requests to task 1. The task ID of the requesting task and the task ID of the requested task are the values that determine the appropriate CMCC entry.

CMODS are allocated from a pool as needed and, therefore, have no fixed location. Memory Pool 2 is reserved exclusively for use by intertask communications. A CMOD consists of six words: two are used for control; two are used as input registers; and two are used as output registers. A task receives all of its requests and makes all of its replies through a CMOD.

Figure 3-1 illustrates the tables used for task communication.

One task communicates with another by placing a request in the input word of a CMOD. The requested task replies by placing the request status in the output words of the CMOD. The format of a request is subject to the requirements defined by the called task. Requests recognized by a task are described with the task later in this section. However, some conventions do exist. Conventionally, the requested function is placed in INPUT+0. Output usage is conventionally defined such that OUTPUT+0 is 0 if no error has occurred; otherwise, it contains a nonzero error code.

Task communication routines

Six re-entrant routines in STP that are common to all tasks facilitate intertask communication. They are:

PUTREQ	Put request routine, asynchronous, destroys A6 and A7
GETREQ	Get request routine; destroys A6 and A7
PUTREPLY	Put task reply routine; destroys A6 and A7
GETREPLY	Request status routine; destroys A6 and A7
TSKREQ	Task request routine, synchronous; destroys A3
REPLIES	Queues unrequested reply; destroys A6 and A7

The task placing a request calls PUTREQ to place the request and calls GETREPLY to check for a status from the requested task. Conversely, the requested task uses GETREQ to locate outstanding requests and uses PUTREPLY to return the status.

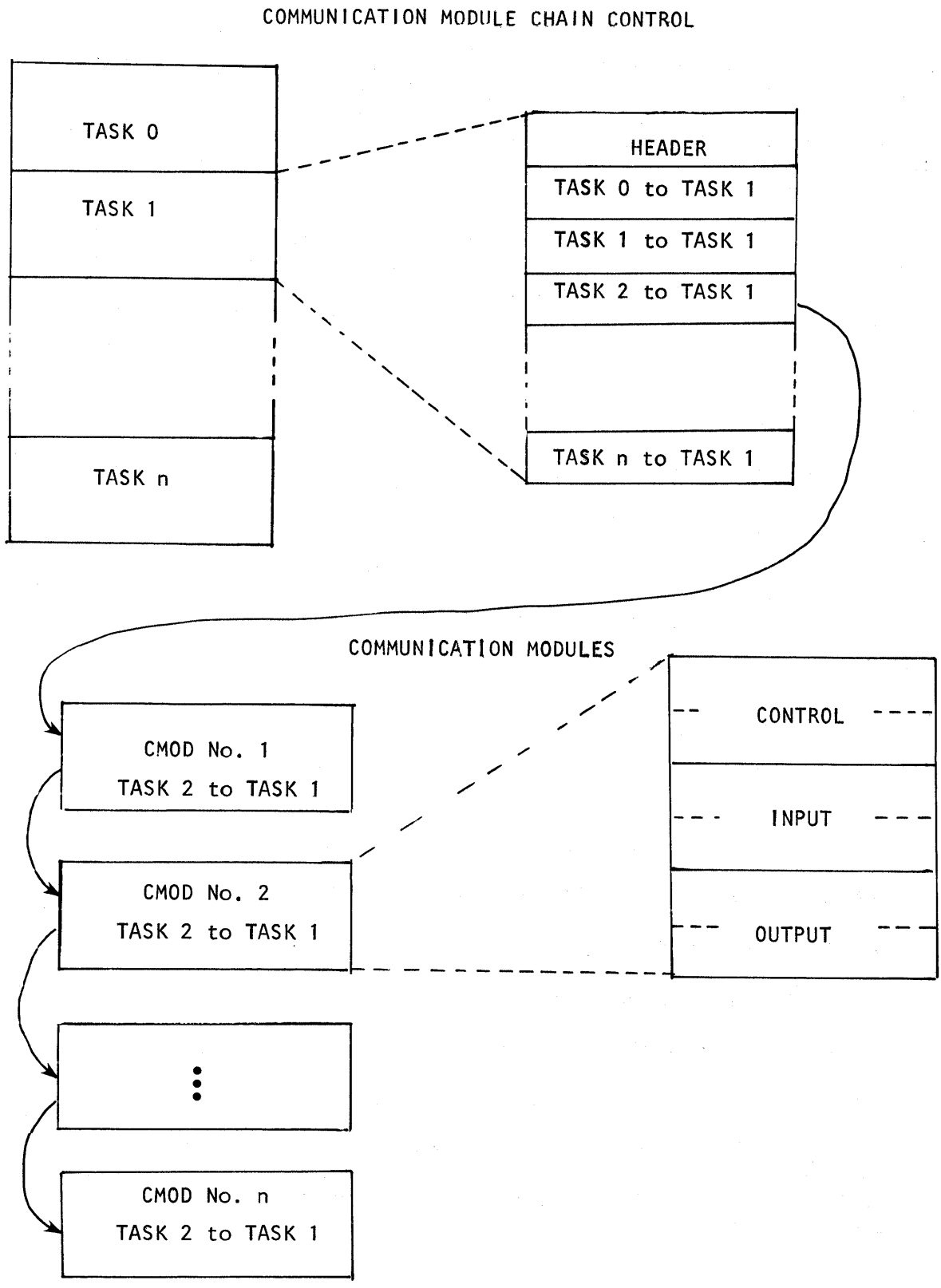


Figure 3-1. Task communication tables

PUTREQ - This STP common subroutine places the request in the input registers of a CMOD and links it to the appropriate communications module chain control. If the request cannot be chained because either no CMODs are available or the chain is at its maximum, PUTREQ suspends the calling task or, at the caller's discretion, returns control to the requester with no action taken. Once PUTREQ has successfully generated the CMOD and linked it to the CMCC, the requested task is readied and control returns to the requester. PUTREQ is called via a return jump with the caller providing the following values:

INPUT REGISTERS: (A1) = "Throw-away" indicator. If (A1) is positive, control is not returned to caller until request is queued. If (A1) is negative, control returns with no action taken if the request cannot be queued without suspending the caller.

(A2) = Requested task's ID

(S1) = INPUT+0
(S2) = INPUT+1
} Request

OUTPUT REGISTERS: None

GETREQ - This STP common subroutine locates any outstanding request for the caller. Using the CMCC, GETREQ searches for a CMOD representing a request not yet given to the requester. GETREQ begins the CMCC search with the lowest numbered task and returns the first request encountered to the caller. A task calls GETREQ via a return jump.

INPUT REGISTERS: None

OUTPUT REGISTERS: (A0) = "Found" indicator. If (A0)=0, no outstanding requests exist. If (A0)≠0, a request is being returned.

(A2) = ID of task that generated the request

(S1) = INPUT+0
(S2) = INPUT+1
} Request

PUTREPLY - This STP common subroutine places the reply to a request in the first available CMOD. Requests and replies are stored in the CMOD in the sequence in which they are generated. Therefore, a single CMOD may represent an unrelated request and reply. The subroutine readies the task to which the reply is directed and returns to the requester. PUTREPLY is called via a return jump.

INPUT REGISTERS: (A2) = ID of task to receive the reply

(S1) = OUTPUT+0
(S2) = OUTPUT+1
} Reply

OUTPUT REGISTERS: None

GETREPLY - This STP common subroutine searches for a reply to the calling task. The search begins with the lowest numbered task and ends with the highest numbered task, returning the first reply encountered. GETREPLY removes the CMOD from the CMCC and releases it for reallocation. The subroutine is called via a return jump.

INPUT REGISTERS: None

OUTPUT REGISTERS: (A0) = "Found" indicator. If (A0)=0, no reply was located; if (A0)≠0, a reply is being returned to the caller.

(A2) = ID of replying task

(S1) = OUTPUT+0
(S2) = OUTPUT+1
} Reply

TSKREQ - This STP common subroutine makes a request to a task for processing and suspends the caller until a reply is received. If the request cannot be queued immediately, because either the queue is at its maximum or because no communication modules are available, the caller is suspended until the request can be queued. Once the request has been queued, the caller is suspended until a reply is received. TSKREQ is called via a return jump.

INPUT REGISTERS: (A2) = ID of requested task

(S1) = INPUT+0
(S2) = INPUT+1
} Request

OUTPUT REGISTERS: (S1) = OUTPUT+0
(S2) = OUTPUT+1
} Reply

REPLIES - This subroutine queues a reply for which no request was made. The reply is queued at the beginning of the reply queue. A reply sent via this subroutine will be seen by GETREPLY before any reply sent via PUTREPLY.

INPUT REGISTERS: (A1) = "Throw-away" indicator. If (A1) is positive, control is not returned to caller until reply is queued. If (A1) is negative, control returns with no action taken if the reply cannot be queued without suspending the caller.

(A2) = Replied task's ID

(S1) = INPUT+0
(S2) = INPUT+1
} Reply

OUTPUT REGISTERS: None

3.2.3 USER/STP COMMUNICATION

All user/STP communication is initiated by user jobs. A user-program request to STP may be issued as a CAL macro (see CRAY-OS Version 1 Reference Manual, publication SR-0011). The user macro results in a normal exit from the user program. EXEC routes all normal exits from a job to the Exchange Package Processor. Exchange Package Processor handling of these requests is described in section 4.4

3.3 STP COMMON ROUTINES

Certain re-entrant routines resident in STP may be called via return jumps rather than via a call to another task. These include task logical I/O routines (TIO), circular I/O routines (CIO), memory management routines, and item chaining/unchaining routines.

3.3.1 TASK I/O ROUTINES (TIO)

Task I/O is a re-entrant routine in STP that logically can be considered a part of any task that uses it. It operates only on blocked datasets. TIO allows a system programmer to do logical I/O at the task level without being concerned about physical I/O.

A task that uses TIO must have a DSP, DNT, and buffer assigned for the dataset; TIO does no allocation or deallocation of DSPs, DNTs, and buffers. A task may use a dataset's DNT, DSP, and buffer in a user field or may generate its own DNT, DSP, and buffer for the dataset local to STP. For example, SCP may use an existing DNT portion of an SDT entry while generating a DSP and buffer to accommodate the logical I/O. Similarly, EXP may reference the existing DNT for a user's $OUT dataset or JSH may reference its own DNT for a roll-out dataset. Figure 3-2 illustrates the linkages between the DNT, DSP, and buffer.

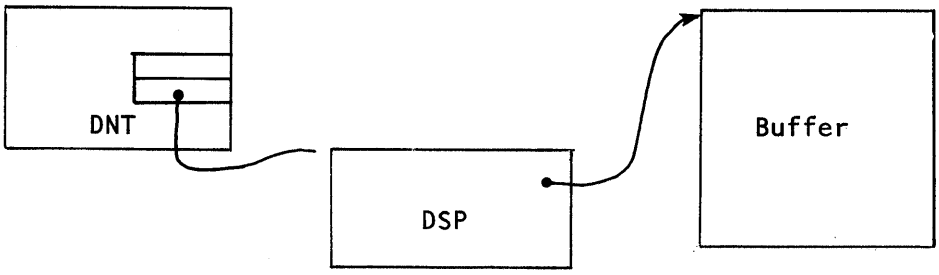


Figure 3-2. Dataset table linkages

Since task I/O cannot sense completion of physical I/O, each task must provide the sensing. To do this, each task, when it is readied from a suspension, should call GETREPLY. If a reply is found that belongs to task I/O (A2=DQMID), the task should jump to REPCIO with S1 and S2 intact from GETREPLY.

When TIO must wait for completion of physical I/O before completing a task's logical request, it returns to the task's main interrupt loop. TIO returns to the task's calling address only on completion of the logical request. The calling task may not make another TIO request for a particular DNT until any previous logical request has completed.

The following TIO routines are available to system programmers:

$RWDP	Read word(s); partial mode (will not point to next *eor*)
$RWDR	Read word(s); record mode (will point to next *eor*)
$WWDP	Write word(s); partial mode (no *eor* written)
$WWDS	Write word(s) with unused bits in last word; record mode (*eor* written)
$WWDR	Write word(s); record mode (*eor* written)
$WEOF	Write *eof*; calls $WWDR if no *eor* was written
$WEOD	Write *eod*; calls $WEOF if no *eof* was written
$REWD	Rewind dataset; calls $WEOD if no *eod* was written

To call a TIO routine, a task places parameters required by the routine in A registers and executes a return jump to the routine. The routine returns results to the caller via A registers.

CAUTION

These TIO routines have the same names as logically equivalent routines in the system library, $SYSLIB. However, the TIO routines reside in STP and the source for library routines resides in the SYSLBPL program library.

3.3.2 SYSTEM TABLES USED BY TIO

TIO uses the following system tables for the dataset on which I/O is to be preformed:

DNT Dataset Name Table
DSP Dataset Parameter Area

Detailed information on these tables is available in the COS Table Descriptions Internal Reference Manual, publication SM-0045.

Dataset Name Table (DNT)

TIO uses the DNT as indicated by the F$RDC and F$WDC routines available to users (refer to description of the Exchange Package Processor in section 4.4).

Dataset Parameter Area (DSP)

TIO uses certain DSPs located in the user field, such as those for $IN, $OUT, datasets read or written by BUFFER IN/OUT, and sequential COS blocked datasets that are being closed when in write mode and not positioned to end of data. TIO uses reserved words at the end of the DSPs. These are saved in the JTA when a TIO routine goes into recall.

Error processing

When TIO detects an error, a negative value is returned in A0. The caller is responsible for processing these errors. Appropriate error bits in the DSP error status (DPERR) indicate which error occurred.

TIO logical read routines

The TIO read routines transfer partial or full records of data from the I/O buffer to the task's data area. The data is placed in the data area in full words depending on the read request issued. Figure 3-3 provides an overview of the logical read operation. The calling routine must examine DPEOR, DPEOF, and DPEOD in the DSP to determine end of record, end of file, or end of data status. If the record control word indicates unused bits in the last word of the record, these bits are zeroed in the data area and field DPUBC is set to the number of unused bits.

$RWDP - Words are transmitted from the I/O buffer defined by DSP to the area beginning at FWA until either the word count in A3 is satisfied or an *eor* is encountered. $RWDP calls $RBLK, automatically.

SUBROUTINE NAME: $RWDP - Read words, partial mode

ENTRY CONDITIONS: (A1) Address of DSP

(A2) FWA of task's data area

(A3) Word count. If word count is 0, no data is transferred.

(A6) Address of DNT

(A7) Address of JXT
(=0 if not job related)

RETURN CONDITIONS: (A0) Status
<0 TIO error (block number error, null dataset, etc.)
=0 Logical I/O complete

(A1) Address of DSP

(A2) FWA of task's data area

(A3) Word count

(A4) LWA+1

(A6) Address of DNT

(A7) Same value as input

$RWDR - This routine resembles $RWDP; however, following the read, the dataset is positioned after the *eor* that terminates the current record.

SUBROUTINE NAME: $RWDR - Read words, record mode

ENTRY CONDITIONS: Same as $RWDP

RETURN CONDITIONS: Same as $RWDP

TIO logical write routines

The TIO write routines transfer partial or full records of data from the task's data area to the I/O buffer. The data is transferred in full words depending on the write operation requested. Two additional write routines provide for writing an *eof* or an *eod* on the dataset. Figure 3-4 provides an overview of the logical write operations. When writing in record mode, it is possible to provide a count of unused bits in the last word of the record. These bits are not zeroed in the buffer, but the record control word indicates unused bits, and the bits are then cleared when the record is read.

$WWDP - The number of words specified by the count are transmitted from the task's data area beginning at FWA and are written in the I/O buffer defined by DSP. $WWDP automatically calls $WBLK, as needed.

SUBROUTINE NAME: $WWDP - Write words, partial mode

ENTRY CONDITIONS:
(A1) Address of DSP

(A2) FWA of task's data area

(A3) Word count. If count is 0, no data is transferred.

(A6) Address of DNT

(A7) Address of JXT
(=0 if not job related)

RETURN CONDITIONS:
(A0) Status
<0 TIO error
=0 Logical I/O complete

(A1) Address of DSP

(A2) FWA of task's data area

(A3) Word count

(A4) LWA+1

(A6) Address of DNT

(A7) Same value as on input

$WWDR - The $WWDR routine resembles $WWDP. However, an *eor* RCW terminating the record is inserted in the I/O buffer in the next word following the data. To simply write an *eor*, the task issues a $WWDR with (A3)=0.

SUBROUTINE NAME: $WWDR - Write words, record mode

ENTRY CONDITIONS: Same as $WWDP

RETURN CONDITIONS: Same as $WWDP

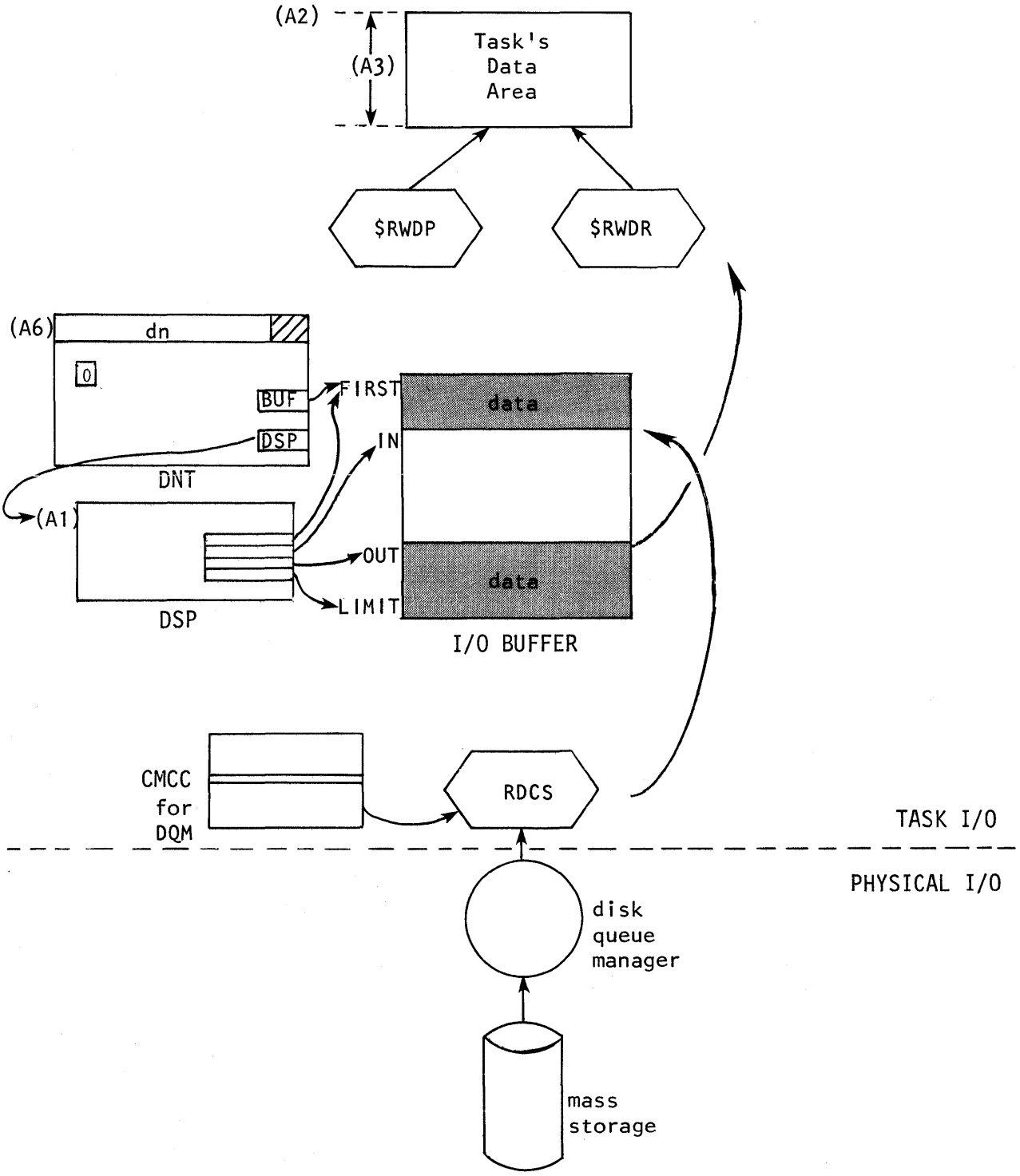


Figure 3-3. TIO logical read

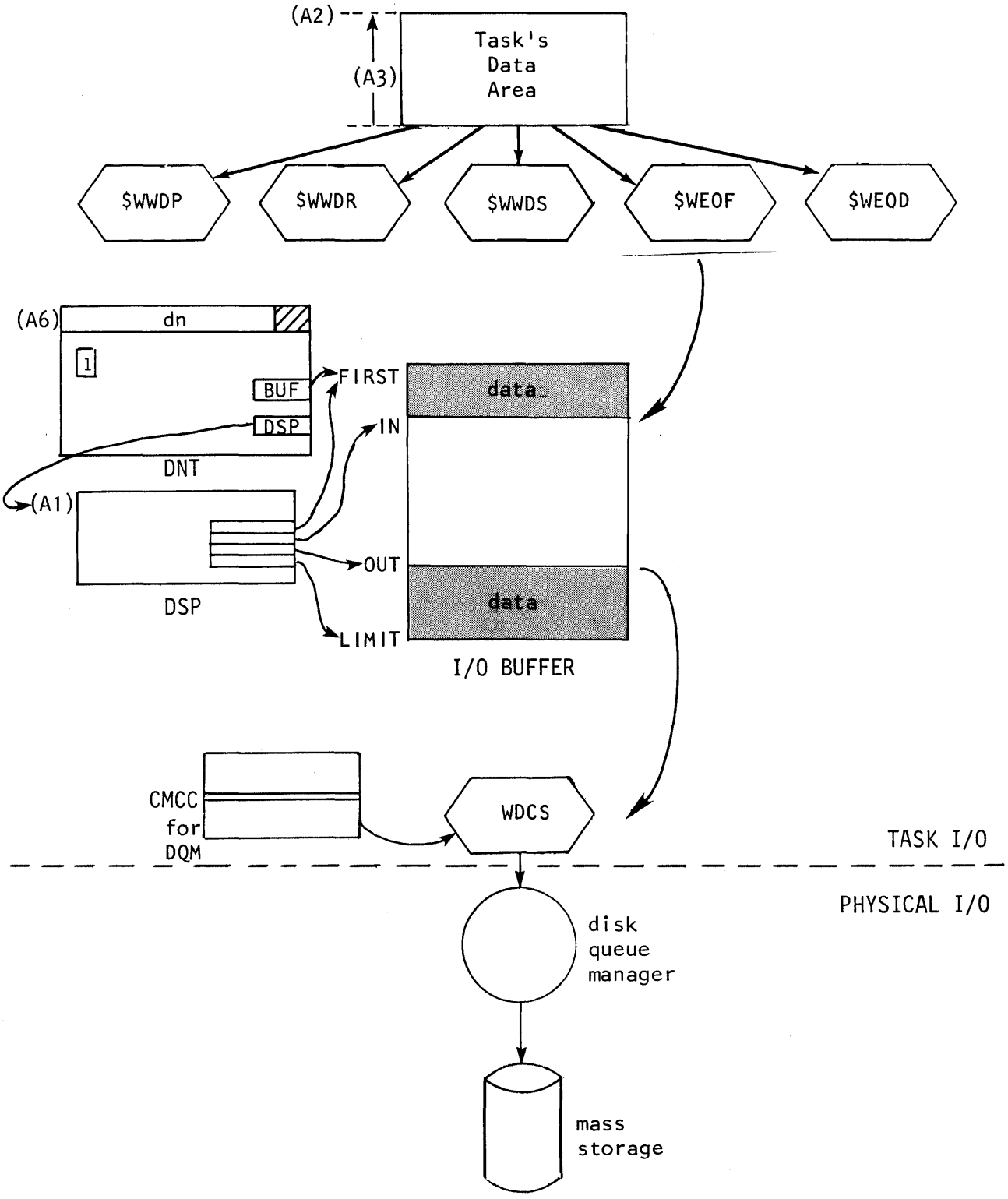


Figure 3-4. TIO logical write

$WWDS - The $WWDS routine is identical to $WWDR, except that the last word of the record contains unused bits, and the *eor* RCW constructed contains the unused bit count.

SUBROUTINE NAME: $WWDS - Write words, record mode, with unused bit count

ENTRY CONDITIONS: Same as $WWDR, plus:

(A4) Unused bit count in the last word of the record; a value from 0-63

RETURN CONDITIONS: Same as $WWDR

$WEOF - This routine writes an *eof* RCW preceded by an *eor* RCW, if necessary, as the next words in the I/O buffer.

SUBROUTINE NAME: $WEOF - Write end of file

ENTRY CONDITIONS: (A1) Address of DSP

(A6) Address of DNT

(A7) Address of JXT
(=0 if not job related)

RETURN CONDITIONS: (A0) Status
<0 TIO error
=0 Logical I/O complete

(A6) Address of DNT

(A7) Same value as on input

$WEOD - This routine writes an *eod* RCW preceded by an *eor* and an *eof*, if necessary, as the next words in the I/O BUFFER. The $WEOD forces the final block of data to be written on the disk; that is, it flushes the I/O buffer. A $WEOD cannot be followed by a write.

SUBROUTINE NAME: $WEOD - Write end of data

ENTRY CONDITIONS: (A1) Address of DSP

(A6) Address of DNT

(A7) Address of JXT
(=0 if not job related)

RETURN CONDITIONS: (A0) Status
<0 TIO error
=0 Logical I/O complete

(A6) Address of DNT

(A7) Same value as on input

Positioning routine

TIO supports a single positioning routine, $REWD.

$REWD - The $REWD routine positions the dataset at the beginning of data (*bod*). If the deadstart is in write mode and no *eod* has been written, $REWD calls $WEOD.

SUBROUTINE NAME: $REWD - Rewind dataset

ENTRY CONDITIONS: (A1) Address of DSP

(A6) Address of DNT

(A7) Address of JXT
(=0 if not job related)

RETURN CONDITIONS: (A0) Status
<0 TIO error
=0 Logical I/O complete

(A6) Address of DNT

(A7) Same value as on input

Block transfer routines

TIO supports two block transfer routines, $RBLK and $WBLK.

$RBLK - $RBLK is called only by other task I/O routines and may not be called directly by a task. $RBLK looks to see if the buffer is less than half full. If it is, it calls CIO to initiate a disk read. CIO continues to read as long as the user continues to empty the buffer fast enough to keep it half empty. If the buffer is more than half full when $RBLK is called, $RBLK verifies the next BCW (its block number must equal the relative sector number of the dataset) and returns to the caller.

SUBROUTINE NAME: $RBLK - Read block(s)

ENTRY CONDITIONS: (A1) Address of DSP

(A5) Address of current block control word

(A6) Address of DNT

(A7) Base address of DSP buffer pointers (either uses BA or JM address)

(W.DPTM) PW

RETURN CONDITIONS: (A0) Status
<0 TIO error
=0 Logical I/O complete

(A1) Address of DSP

(A4) OUT

(A6) Address of DNT

(A7) Same value as input

$WBLK - $WBLK is called only by other task I/O routines. $WBLK checks to see if the buffer is more than half full. If it is, it calls CIO to initiate a disk write and writes a BCW. CIO continues to write as long as the user continues to fill the buffer fast enough to keep it more than half full. If the buffer is less than half empty when $WBLK is called, $WBLK does no more than insert BCWs as needed.

SUBROUTINE NAME: $WBLK - Write block(s)

ENTRY CONDITIONS: (A1) Address of DSP

(A5) Address of next block control word

(A6) Address of DNT

(A7) Base address of DSP buffer pointers

RETURN CONDITIONS: (A0) Status
<0 TIO error
=0 Logical I/O complete

(A1) Address of DSP

(A6) Address of DNT

(A7) Same value as input

Stepflows of TIO subroutines

$REWD

1. If *eod* not written, call $WEOD.
2. Reset DSP.
3. Exit.

$WEOD

1. If *eof* not written, call $WEOF.
2. Call $WWDR to write *eod*.
3. Exit.

$WEOF

1. If *eor* not written, call $WWDR.
2. Call $WWDR to write *eof*.
3. Exit.

$WWDP/$WWDR/$WWDS

1. If preceding function was a write, go to 3.
2. Process write after read.
3. Move words into buffer; if end of move, go to 7.
4. If not at BCW, go to 3.
5. Call $WBLK.
6. Go to 3.
7. If not record mode ($WWDR), go to 11.
8. Insert *eor*.
9. If not at BCW, go to 11.

10. Call $WBLK.

11. Update DSP.

12. Exit.

$WBLK

1. If buffer more than half used, call WDCS.

2. Update DSP.

3. Exit.

$RWDP/$RWDR

1. Move words out of buffer; if end of move, go to 5.

2. If not at BCW, go to 5.

3. Call $RBLK.

4. Go to 1.

5. If not record mode ($RWDR), go to 9.

6. Point at next *eor*.

7. If not at BCW, go to 9.

8. Call $RBLK.

9. Update DSP.

10. Exit.

$RBLK

1. If buffer more than half empty, call RDCS.

2. Update DSP.

3. Exit.

3.3.3 CIRCULAR I/O ROUTINES (CIO)

Physical I/O on a dataset uses a circular buffering technique and is initiated by a set of STP common routines known as CIO (Circular I/O). The Exchange Processor uses CIO to handle I/O calls from user programs. CIO is also accessible to all other tasks through TIO and through direct calls.

The Disk Queue Manager initiates circular I/O operations on mass storage in response to CIO calls. These calls are issued by user programs or tasks when data is to be transferred between the I/O buffer defined by the DSP and mass storage. However, these requests need not be explicitly issued. FORTRAN I/O routines in user programs and TIO routines in STP manage the I/O buffers and make calls to CIO.

The I/O buffer consists of an integral number of 512-word blocks. For a COS blocked file, the first word of each block is a block control word. The size and location of the buffer are defined when the DSP is generated. The default size is defined by an installation parameter.

Logical I/O on a buffer may be concurrent with physical I/O. That is, on a read operation, the user may be extracting data from the buffer at the same time the system is inserting data, with the user read lagging the system read.

Alternatively, on a write operation, the user may be inserting data into the buffer at the same time the system is emptying it. In this case, the user write leads the system write.

The buffers are managed through the IN, OUT, FIRST, and LIMIT pointers in the DSP. Figure 3-5 illustrates the format of physical I/O. Referring to step A, the IN pointer advances from FIRST to LIMIT as data is inserted into the buffer.

Step B illustrates how emptying the buffer lags filling the buffer. The OUT pointer, which is initially the same as IN, advances toward LIMIT but always lags IN.

For writing, a buffer can become full when data is inserted faster than it is extracted.

For reading, a buffer can become empty if data is extracted faster than it is inserted.

Physical reads and writes always involve 512-word blocks. On a read, IN is always at a 512-word buffer boundary, but OUT, which is being modified by the user, need not be. Conversely, on a write, OUT is always at a 512-word buffer boundary but IN need not be.

On a read operation, EXEC and CIO modify the IN pointer and the caller modifies the OUT pointer. If IN=OUT, the buffer is empty if errors have occurred (DPERR≠0) or if the DSP is busy (DPBSY=1). The buffer is full when IN=OUT, the DSP is not busy, and no errors have occurred.

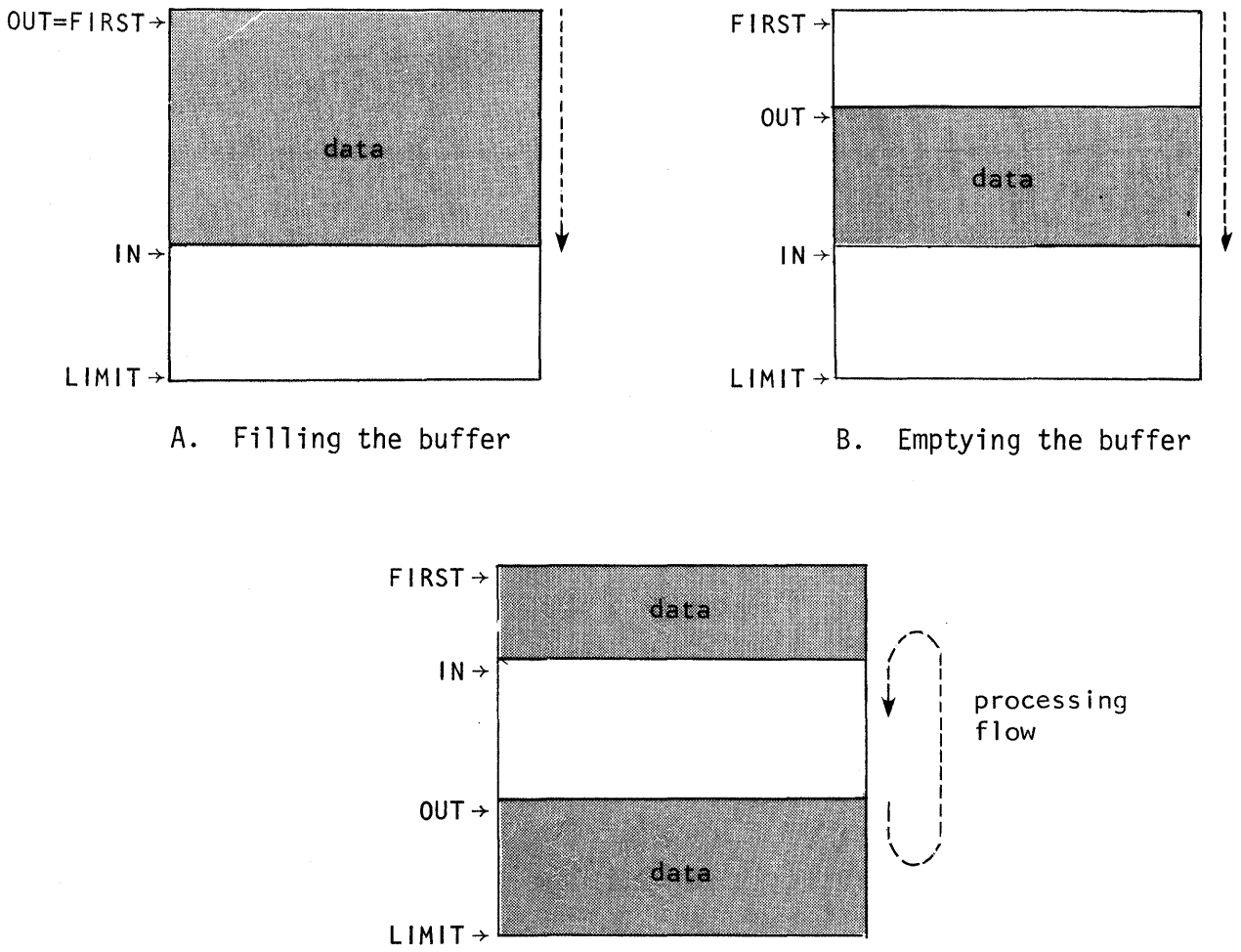


Figure 3-5. Physical I/O

On a write operation, DQM and CIO modify the OUT pointer and the caller modifies the IN pointer. If IN=OUT, the buffer is full if errors have occurred (DPERR≠0) or if the DSP is busy (DPBSY=1). The buffer is empty if IN=OUT, the DSP is not busy, and no errors have occurred.

A dataset may be declared memory resident. If so, CIO determines whether a physical I/O request should be issued for the dataset based on processing direction and whether the buffer is full or empty. If the request is to write the dataset and the buffer is full (IN=OUT), CIO issues a physical I/O request. In this case, CIO also clears the memory resident indicators in the DSP and DNT. If the buffer is not full, CIO merely returns to the caller.

If the request is to read the dataset and the buffer is empty (IN=OUT and DPIBN=0), CIO issues a physical request if the DNT shows that mass storage space exists. If CIO is called to read and the buffer is not empty, CIO returns as if a successful read had occurred. If the buffer is empty, CIO determines whether the requested block (DPIBN) is within the buffer (IBN*512≤LIMIT-FIRST) and whether the block exists (IBN<DNLBN). If either condition is not true, CIO clears the memory resident flags and the read proceeds as for a null dataset. If both conditions are true, CIO:

1. Sets DPIBN=DNLBN,
2. Sets DPOBN=requested block (old DPIBN),
3. Sets IN and OUT to point to the correct 512-word boundaries within the buffer, and
4. Sets the EOI bit in DSP.

If mass storage space is allocated and the dataset size from the DAT is greater than the buffer size, CIO clears the memory resident indicators. Any I/O suspend calls made to the Job Scheduler are canceled before returning.

CIO call format

A task calls CIO by calling CIO subroutines RDCS or WDCS. User CIO calls are processed by a special entry point, CIOS. These routines return control immediately after calling the Disk Queue Manager. A task that wants to resume processing upon completion of part of the physical I/O should call subroutine TRCL.

RDCS - This subroutine is a task's entry point to CIO for read requests.

ENTRY CONDITIONS: (A1) Address of DSP

(A6) Address of DNT

(A7) Address of JXT
(=0 if not job related)

RETURN CONDITIONS: Control returns immediately to the caller.

(A1) Address of DSP

(A6) Address of DNT

(A7) Address of JXT
(=0 if not job related)

WDCS - This routine is a task's entry point to CIO for write requests.

ENTRY CONDITIONS: (A1) Address of DSP

(A6) Address of DNT

(A7) Address of JXT
(=0 if not job related)

RETURN CONDITIONS: Control returns immediately to the caller.

(A1) Address of DSP

(A6) Address of DNT

(A7) Address of JXT
(=0 if not job related)

CIOS - CIOS is an entry point for user CIO calls.

ENTRY CONDITIONS: DNP contains processing direction.

DNDSP contains DSP address.

(A6) Address of DNT

(A7) Address of JXT
(=0 if not job related)

RETURN CONDITIONS: Control returns immediately to the caller.

(A1) Address of DSP

(A6) Address of DNT

(A7) Address of JXT
(=0 if not job related)

CIOR - Return from CIOS - REPCIO returns control here based on the return address passed to the Disk Queue Manager (DQM) from CIOS. Control may return here several times if DQM generates multiple replies while chasing DSP buffer pointers to read ahead or write behind.

ENTRY CONDITIONS: (S1) Word 0 of DQM reply (DQM error status)

(S2) Word 1 of DQM reply

(A1) Address of DSP

(A6) Address of DNT

(A7) Address of JXT
(=0 if not job related)

RETURN CONDITIONS: (A1) Address of DSP

(A6) Address of DNT

(A7) Address of JXT
(=0 if not job related)

REPCIO - This routine processes Disk Queue Manager replies.

ENTRY CONDITIONS: (S1) Word 0 of reply

(S2) Word 1 of reply

RETURN CONDITIONS: (S1)=(S0) Word 0 of reply (error status)

(S2) Word 1 of reply (24/return addr, 16/JXT offset, 24/DNT address)

(A1) Address of DSP

(A6) Address of DNT

(A7) Address of JXT
(=0 if not job related)

TRCL - Task I/O recall - This routine resumes the task when I/O is complete or partially complete. TRCL may be called only by the task that initiated I/O on this DNT.

ENTRY CONDITIONS: (A1) Address of DSP

(A6) Address of DNT

(A7) Address of JXT
(=0 if not job related)

RETURN CONDITIONS: (S0) Contents of DSP word containing DPERR

(A1) Address of DSP

(A6) Address of DNT

(A7) Address of JXT
(=0 if not job related)

Control returns to caller when at least one sector has been transferred without error or when an error is detected. Control is returned via CIOR when this task is resumed by a Disk Queue Manager reply.

Stepflows of CIO Subroutines

RDCS/WDCS

1. Set processing direction in DNP.
2. Save DSP address in DNDSP.
3. Set DNIOU if JXT address is nonzero.

CIOS

1. If dataset is not memory resident, go to step 14.
2. If dataset is being written, go to step 12.
3. If IN≠OUT, go to step 9.
4. If DPIBN≠0, go to step 13.
5. If dataset does not exist on mass storage, go to step 8.
6. If dataset is larger than buffer, go to step 13.
7. Go to step 14.
8. If last block written (DNLBN) is greater than buffer size in blocks, go to step 13.
   a. If user requested block is greater than or equal to last block, go to step 13.
   b. Set DPOBN=DPIBN. Set DPIBN=DNLBN. Set IN and OUT to point to the current block, determined from IBN and OBN. Set DPEOI=1.
9. Clear any DNT I/O wait flags. If not job related, return to caller.

10. If job was suspended, cancel suspend (J$IODONE).

11. Return to caller.
12. If buffer is not full (IN≠OUT), go to 9.

13. Clear DNMEM and DPMEM.

14. System error if the DNT is active, (DNAIO)=1.

15. User error if the DSP is busy, (DPBSY)=1.

16. If not job related, JXT address=0, go to step 14.

17. Increment job I/O request count (JTIOR) and DNT I/O request count (DNIOR).

18. If CIO is called by Task I/O, (DNIOU)=1, or if CIO is called for user-buffered I/O, (DNBIO)=1, or if the I/O buffer is in the user field between 0 and the LFT table, then continue with step 19; else, go to step 14.

19. Increment the user I/O count, JTIOUC. Set DNIOU=1.

20. Set the DSP busy bit.

21. If job related I/O, JXT address≠0, then set DNDSP equal to the DSP address relative to the JTA address.

22. Call the Disk Queue Manager:

```
S1 = 24/return address, 16 JXT offset, 24/DNT address
S2 = 0 (read/write function code)
R  PUTREQ
```

The return address is set to CIOR.
The DNT address is relative to the job's JTA address if job related.

23. Return to the caller.

CIOR

1. Clear the busy bit in the DSP (if this is not a DQM intermediate recall reply).

2. If the job is in recall, (DNJIO=1), call the Job Scheduler with function code J$IODONE. Clear DNJIO.

3. If I/O was in the user field below the LFT table, (DNIOU)=1, then clear DNIOU and decrement the user I/O count, JTIOCU. Ready the Job Scheduler if JTIOCU is decremented to 0.

4. If the Disk Queue Manager return status is 0, go to step 6.

5. Process Disk Queue Manager (DQM) error.

DQM error	DPERR bit set	Exchange Processor error	Description
EREOI	DPEOI		Read beyond EOI
EREXT	DPEOI		Write beyond EOI without extend permission
ERUDE	DPUDE		Data error
ERUHE	DPUHE		Hardware error
ERIDP		ERBDP	Invalid DSP
ERNLD		ERLDV	Unknown logical device name
ERNMT		ERNDT	No more DAT space
ERNMS		ERNDS	No more disk space
ERNCS		ERNDS	Not enough contiguous disk space

For each error from DQM, CIOR either sets a bit in the DSP error status field, DPERR, or processes the error as a fatal error.

For a fatal error, the system is aborted if CIO is not executing in the Exchange Processor. If CIO is executing in the Exchange Processor, the job is aborted with an appropriate error message. (In this case, CIO exits directly to entry ABTJ1 in the Exchange Processor.)

6. Clear DNRCL.

7. If called for buffered I/O, (DNBIO)=1, then clear DNBIO and validate the DPTM words in the DSP by comparing with the copy saved in the JTA at JTDTM.

8. If Task I/O recall, (DNTIO)=1, then clear DNTIO and return to the address in DNCIO.

9. Return to the executing task's main loop as determined by the address in the CIOJ table for the task.

REPCIO

1. Set A7 = JXT offset + JXT base,
   A2 = DNT address, and
   A1 = (DNSP) + JTA address.

2. Exit to return address in word 1 of reply.

TRCL

1. If DNTIO=1 or if DNRCL=1, abort the job if the request is job related; otherwise, abort the system.

2. Set DNTIO=1 and DNRCL=1.

3. Set DNCIO=B0.

4. If not a buffered I/O request, suspend the job; set DNJIO=1 and call the Job Scheduler.

5. Save DSP words in the JTA if a buffered I/O request.

6. Exit to this task's main loop. Use the CIOJ jump table containing the main loop address for each task that calls CIO routines.

3.3.4 MEMORY ALLOCATION/DEALLOCATION ROUTINES

The MEMAL and MEMDE STP common subroutines provide for allocation and deallocation of variable-size memory areas for temporary use by a task.

Allocation and deallocation is from memory pool(s). The number and size of memory pools is determined when the operating system is generated.

As illustrated in figure 3-6, the Pool Table and the header and trailer words are used for controlling memory allocation and deallocation. The Pool Table consists of a header word and one word for each memory pool in the system. The Pool Table header defines the maximum valid pool number. The word associated with the memory pool provides the base address and the size of the memory pool.

Each area of a memory pool is surrounded by a header word and a trailer word. The header and trailer words are identical and indicate the status (available or unavailable) and the size of the area. The number and sizes of the areas changes dynamically as tasks obtain words from or return words to a pool.

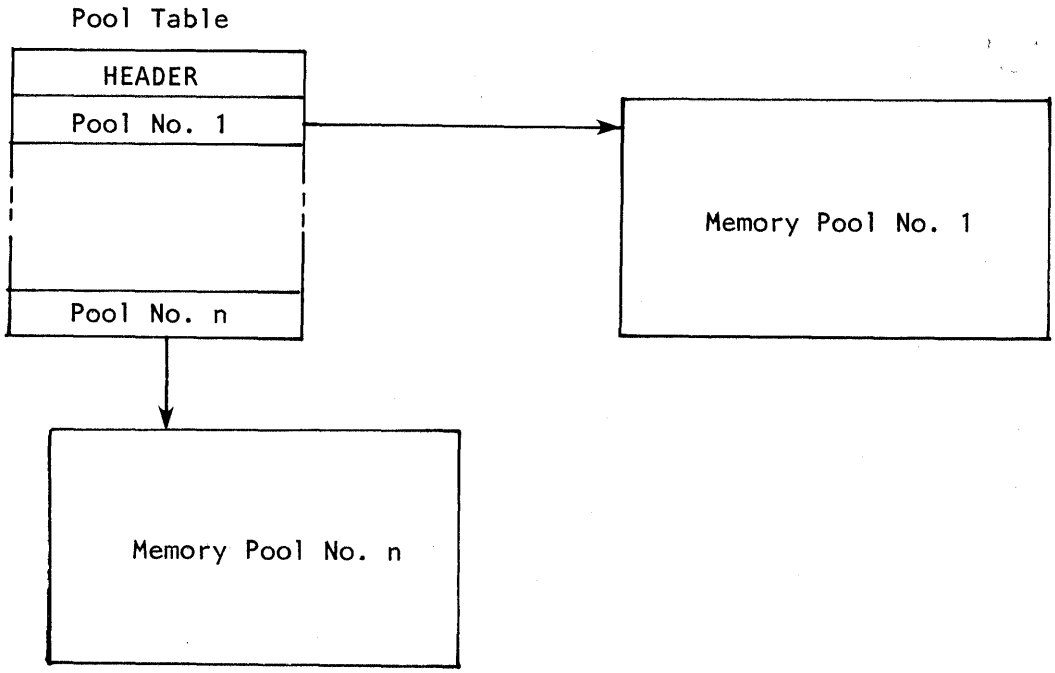


Figure 3-6. Memory allocation tables

Memory allocation - MEMAL

MEMAL is an STP common subroutine that allocates a variable size memory area for temporary use by a task.

Memory is allocated from a memory pool. The caller provides MEMAL with the number of the pool from which allocation is to occur and the number of words desired. The number of words must be at least one and not more than the pool size less 2. MEMAL allocates two words more than requested; these are used by MEMAL as header and trailer words for the area to be allocated. On return to the caller, MEMAL provides a status and, if memory was allocated, the address of the first usable word. The allocated area is zeroed.

INPUT REGISTERS: (A6) = Number of memory pool from which to allocate

(A7) = Number of words desired

OUTPUT REGISTERS: (A6) = Status:

0 Good status
1 Invalid memory pool number
2 Invalid number of words requested
3 Memory not available

(A7) = Address of first usable word of memory to be allocated (meaningless if A6≠0).

Memory deallocation - MEMDE

MEMDE is an STP common subroutine that returns memory to its memory pool for reallocation.

In addition to marking the memory as available for allocation, MEMDE combines the area with any adjacent available areas thereby maintaining the largest possible size for allocation.

The caller must provide MEMDE with the memory pool number to which memory should be returned and the address of the first usable word of the memory to be deallocated.

INPUT REGISTERS: (A6) = Memory pool number

(A7) = Address of first usable word of memory to be deallocated

OUTPUT REGISTERS: (A6) = Status:
- 0 Good return
- 1 Invalid address
- 2 Area not currently allocated
- 3 Invalid pool number

(A7) = Address of memory released; meaningful only if status is 0.

3.3.5 CHAINING/UNCHAINING SUBROUTINES

The CHAIN and UNCHAIN STP common subroutines provide tasks with a means of linking data. Each piece of data is termed an item and consists of two words of header information followed by the information being added to the chain. As an example, an item could be the input and output registers used for intertask communications. By chaining registers, tasks need not be limited to two words of input and two words of output. However, the CHAIN/UNCHAIN subroutines are not restricted to use for intertask communications; the amount of information in an item and its type is defined entirely by the task using the subroutines.

Chaining is established through a chain control word and the first two words of each item in the chain. Figure 3-7 illustrates a chain of items.

Pointers in the chain control word identify the first and last items on the chain. The chain control word also contains space for the maximum number of items that may exist on the chain and a count of the number of items on the chain. However, because the chain control word may reflect only a portion of the entire chain, the maintenance of the count is the responsibility of the calling task.

The two words used in the chain item provide a forward link to the next item on the chain, a backward link to the preceding item on the chain, and the address of the chain control word to which this item is linked.

Chain item - CHAIN

CHAIN is an STP common subroutine that places an item in a queue (chain). CHAIN always adds items at the end of the existing queue. Therefore, if a single destination accepts multiple priorities, creation of a separate queue for each priority is necessary.

The caller must provide CHAIN with the address of the chain control word and the address of the chain item.

INPUT REGISTERS: (A6) = Address of chain control word

(A7) = Address of the item to be chained

OUTPUT REGISTERS: (A6) = Unchanged from input

(A7) = Unchanged from input

Unchain item - UNCHAIN

UNCHAIN is an STP common subroutine that removes an item from a queue. The item to be removed may be anywhere in the queue.

Although the chain control word contains a count of the items in the queue, UNCHAIN does not adjust this count; this is the responsibility of the caller.

The caller must provide UNCHAIN with the address of the item to be unchained. UNCHAIN determines the appropriate chain control word from the item.

INPUT REGISTER: (A7) = Address of item to be unchained

OUTPUT REGISTER: (A7) = Unchanged from input

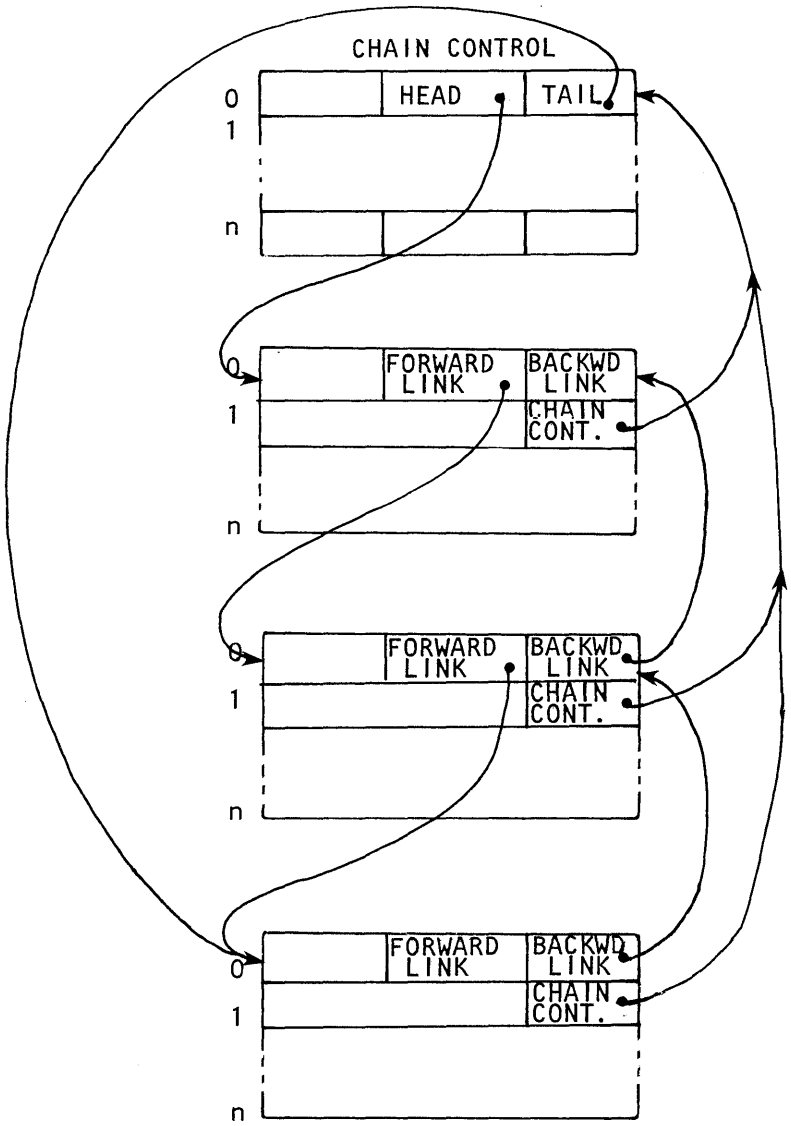


Figure 3-7. Chain tables

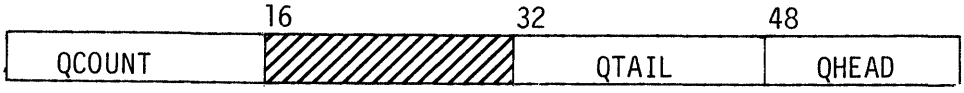
3.3.6 INTERACTIVE COMMUNICATION BUFFER MANAGEMENT ROUTINES

The interactive communication buffer management routines are a set of common routines that operate on the Interactive Buffer Table (IBT) and queue control words in the Active User Table (AUT). They allocate and deallocate buffer space, queue and dequeue messages, and transfer messages to and from the buffer area. To ensure proper management of the buffers, these routines allocate and deallocate buffers in STP non-interruptible mode.

The interactive communication buffer area is in the high range of memory, and the IBT is constructed so that, in the future, the buffer can be expanded, contracted, or relocated as required by dynamic memory management. Features of the IBT and the management routines that minimize overhead in providing dynamic memory management of this area are the interactive buffer base address, the use of buffer identifiers, and inverting entry allocation (the highest address buffer is allocated first). Furthermore, the bit map in the IBT minimizes overhead in allocating and deallocating buffer space. Buffer area fragmentation is prevented by allocating memory in small, fixed-size blocks, which can be linked together.

Queue control word structure

The interactive buffer management routines manipulate a queue control word with the following structure:



Field	Bits	Description
QCOUNT	0-15	Count of entries on the queue
QTAIL	32-47	Buffer identifier of the last buffer on the queue
QHEAD	48-63	Buffer identifier of the first buffer on the queue

ENQMSG - This routine allocates buffer space, moves the message into the buffer, and queues the buffer on the desired queue.

ENTRY CONDITIONS: (A0) Enqueue type:
=0 Queue at tail
≠0 Queue at head

(A5) Message length

(A6) Message address

(A7) Queue control word address

RETURN CONDITIONS: (A0) Operation status:
=0 Successful
≠0 Inadequate buffer space

Registers A1 through A4 are saved and restored.

NXTMSG - NXTMSG moves the next message in the queue to the record area.

ENTRY CONDITIONS: (A0) Type of move (presently only block is supported)

(A5) Maximum move size

(A6) Move address

(A7) Queue control word address

RETURN CONDITIONS: (A0) Operation status:
=0 Successful
≠0 No message or message too long

(A5) Buffer ID

(A6) Address within record area where next buffer may be moved

(A7) Queue control word address

Registers A1 through A4 are saved and restored.

FREEMSG - This routine removes a message from the queue and releases the buffer space.

ENTRY CONDITIONS: (A6) Buffer ID

(A7) Queue control word address

RETURN CONDITIONS: (A0) Operation status:
=0 Successful
≠0 Buffer not on queue

SYSTEM TASKS 4

The system tasks are described in this section. They are:

4.1 COS STARTUP

System startup is the process of loading the CRAY-1 Operating System (COS) into CRAY-1 memory, beginning execution, and generating or recovering tables for the operating system. The operating system to be started may or may not be on a user permanent dataset. If it is on a CRAY-1 disk storage unit, then a 2-pass startup is performed, and the parameter file must specify a 2-pass startup. COS has three types of startups: Install, Deadstart, and Restart.

Install — For an Install, COS is started as if for the first time. All CRAY-1 mass storage is assumed to be vacant. Therefore, a label is written on each disk storage unit, and the Dataset Catalog (DSC) is written on the master unit. An amount of space, determined by the value of the installation parameter I@DMPSIZ, is allocated and zeroed on the master device. This space is used to dump memory following system failures. The DSC and the tables residing in memory are set up to indicate that no datasets (permanent or temporary) are there other than the dataset containing the DSC

itself. The Rolled Job Index dataset is initialized and is entered into the DSC. A 2-pass startup is meaningless during an Install. If a 2-pass startup is requested during an Install, startup halts and issues a message to the operator in the S-registers explaining why it has halted. Install execution is described further in section 4.1.4.

Deadstart — For a Deadstart, COS is started as if after a normal system shutdown. That is, permanent datasets mentioned in the DSC are preserved via proper setup of tables in memory. However, any input or output queues mentioned in the DSC are deleted. The area reserved for the system dump is also preserved. An attempt is made to access the Rolled Job Index dataset. If the dataset cannot be accessed, a new edition is created and initialized. If the dataset can be accessed, all entries other than entry 0 are cleared (since I/O datasets are not recovered by a Deadstart) and the dataset is rewritten. For a 2-pass Deadstart, the specified system dataset is located and read into memory on pass 1. Once the system is read, the system and the parameter file are moved down over the current system, and a normal startup is initiated. Deadstart execution is described further in section 4.1.5.

Restart — For a Restart, COS is started as if after a system interruption. That is, input and output queues as well as permanent datasets are recovered. The area reserved for the system dump is also preserved. The Rolled Job Index dataset is accessed and read into memory. If the access or the read receives an error, a new edition is initialized and written to disk. If the operator chooses not to recover rolled jobs, the index is cleared and rewritten. The first pass of a 2-pass Restart is identical to the first pass of a 2-pass Deadstart. Restart execution is described further in sections 4.1.6 and 4.1.7.

Most of COS Startup resides in the System Task Processor (STP) for convenient access to system tables and facilities. However, some COS Startup logic resides in operator station software and in EXEC. Some tables, such as the Permanent Dataset Information Table are initialized when STP is assembled.

4.1.1 INPUT TO STARTUP

Input to Startup may consist of a parameter file and the $SDR and $ROLL datasets.

Parameter file

Control of the COS startup procedure is through parameters in the form of statements on a special file. These statements, which are described in the COS Operational Procedures Reference Manual, publication SM-0043, are sent from the operator station. The parameter file may be prepared from punched cards or from the operator station with the aid of a text editor. However, parsing of the command language is performed in the CRAY-1. In this way, rewriting of the parsing logic for each front-end system becomes unnecessary.

A closed subroutine parses the parameter file and correspondingly modifies memory. Thereafter, Startup proceeds as if no parameters have been passed; that is, the intent is to drive Startup directly via STP tables and only indirectly via parameters from the operator station. The operator station commands copy the parameter file into CRAY-1 memory along with EXEC, STP, and CSP.

System Directory dataset ($SDR)

A permanent dataset, $SDR, is maintained to contain records specifying System Directory datasets to be recovered during Restart or Deadstart. The Dataset Catalog (DSC) contains an entry for $SDR, which is initialized during an Install. Space is allocated based on the number of SDR entries specified in the system. During Restart or Deadstart, the dataset is read to rebuild the System Directory. If a failure occurs, a message is issued to the system log, and the operating system initialization is abnormally terminated.

The $SDR dataset is constructed of 512-word blocks. Each block contains eight logical records, with the first word of each block holding the block number relative to the beginning of the $SDR file. The first block in the dataset is a header record containing the maximum number of SDR entries as specified in the last system that recovered the System Directory. The value is updated if the number of entries in the system increases or decreases and recovery is not to be performed. Logical records in the file are accessed by using the formula: (Relative resident SDR entry +1)/8. The quotient gives the block number of the entry within the file, and the remainder gives the logical record number within the block.

Each $SDR record except the header contains the Permanent Dataset Definition Table (PDD) of a dataset entered into the System Directory. When an ACCESS request with the ENTER operand is processed by the Exchange Processor (EXP), the Dataset Name Table (DNT) of the dataset is saved in the resident SDR table. The PDD of the dataset is written to the $SDR dataset. The dataset update is complete before EXP completes processing the request.

Whenever a Restart or a Deadstart is performed by the operating system, the resident System Directory (SDR) is recovered unless the operator specifies that recovery is not to be performed by means of the *SDR parameter. When an Install is performed, the System Directory is not recovered, and a user job (JSYSDIR) must be run to create the initial System Directory entries.

Rolled job index dataset ($ROLL)

The operating system maintains a special permanent dataset so Startup can determine which jobs were in execution prior to a system recovery. This dataset, referred to as $ROLL, contains information about each job that has entered execution and has not yet terminated. $ROLL is maintained in the Dataset Catalog (DSC) with a permanent dataset name of SYSROLLINDEX. Read, write, and maintenance passwords are defined for it. $ROLL is initialized and saved during an Install.

During either Restart or Deadstart, the recovery of rolled jobs subroutine, RRJ, attempts to access $ROLL. If the access fails, a new edition of $ROLL is created, initialized, and saved, and recovery of rolled jobs is disabled with a message to the system log if recovery would have been performed otherwise. No message appears if $ROLL cannot be accessed and recovery was not requested.

The information in $ROLL consists of fixed-length entries, one for each defined Job Execution Table (JXT) entry. The entry corresponding to JXT ordinal zero is used for validation of the $ROLL dataset and does not correspond to any job in the system. Information in entry 0 consists of:

- The number of JXT entries defined in the previously deadstarted system. Recovery is not possible if the previous system defined more JXT entries than the current system. An error message is issued in this case.

- The memory size of the previously deadstarted system. This is informational only.

- The logical name of the device containing $ROLL. This is compared with the device name from the Dataset Allocation Table (DAT) that is supplied by the Permanent Dataset Manager when $ROLL is accessed. A mismatch causes an error message to be issued, and recovery is disabled.

- The track number allocated to $ROLL. JXT limitations assume that $ROLL will never exceed one allocation unit. This number is compared with the AI from the DAT for the accessed $ROLL. A mismatch causes an error message to be issued, and recovery is disabled.

- The sizes of key tables contained in the Job Table Area (JTA) on the roll index, in particular, LE@RJ, LE@DNT, and LE@JXT. These must be the same in the recovered system or RRJ halts. RRJ halts rather than continuing with recovery disabled so the operator can restart with a correct system file without having the roll index overwritten.

All other entries in $ROLL correspond to one specific JXT entry. These entries contain enough information to identify which job has been assigned to the JXT entry and to locate the roll image if the job has been rolled out. The index entry also contains a flag that indicates whether a job has performed some function that will invalidate the roll image. (See the description of the RJ table in the COS Table Descriptions Internal Reference Manual, publication SM-0045, for detailed descriptions of the formats of these entries.)

Information contained in these entries includes:

- The first three words of the first partition from the DAT for the roll image dataset. This includes the 2-word partition header and one word containing up to four allocation unit indices. If the job has never been rolled out, these words are zeros.

- The job name, job sequence number, station, and terminal ID of job origin. These determine which SDT entry in the input queue corresponds to this job.

- A "not recoverable" flag. This indicates that the job cannot be recovered from the roll image. This flag is set whenever the job performs one of the following functions:

    1. DELETE, ADJUST, or MODIFY on a permanent dataset. Since these functions change the DSC in a manner that could cause the job to fail if they were repeated, the roll image cannot be relied upon.

    2. Random write to any dataset. The system circular I/O (CIO) routines recognize a random write to a dataset and declare the job not recoverable, since the difference in data may change job results if the job is restarted at an earlier point.

3. Write following read, rewind, or skip forward on any dataset. Since a program that reads or skips to end of data or end of file may have different results if the terminator is moved or removed completely by overwriting, the job is considered not recoverable.

4. Release of a local dataset. Since disk space returned to the system is available for use by other jobs, release of a local dataset causes the job to be not recoverable. Release of a permanent dataset does not affect disk allocation and therefore does not affect recoverability.

   Note that every job that becomes irrecoverable due to any of the above becomes recoverable again as soon as it has subsequently been successfully rolled out.

- Date/time stamp of system that was running when job was rolled out.
  This is generated from the STP assembly date and time; it detects jobs being recovered on a different system.

$ROLL is maintained jointly by EXP and the Job Scheduler (JSH) during system operation. At job initiation JSH sets up the corresponding index entry to reflect the fact that the job was never rolled out and is therefore not recoverable. Subsequently, each time JSH rolls the job, it sets up the index to point to the roll dataset and designates the job to be recoverable. The index is written to disk when the Disk Queue Manager (DQM) informs JSH that the rollout has completed successfully. EXP recognizes the fact that a job is performing one of the functions that causes the job to become irrecoverable and signals the Job Scheduler to set the index entry accordingly and to rewrite the index. The rewrite of the index always occurs before EXP completes processing the function.

4.1.2 TABLES USED BY STARTUP

The Startup task uses the following tables to initialize the system for an Install, Deadstart, or Restart.

DRT	Device Reservation Table
DSC	Dataset Catalog
SDT	System Dataset Table
DNT	Dataset Name Table
DAT	Device Allocation Table
DSP	Dataset Parameter Area
PDI	Permanent Dataset Information Table
DVL	Device Label
EQT	Equipment Table
RJI	Rolled Job Index Table
QDT	Queued Dataset Table
OCT	Overlay Directory Table
OCT	Overlay Control Table
OLL	Overlay Load Request List

Detailed information about these tables is available in the COS Table Descriptions Internal Reference Manual, publication SM-0045.

Device Reservation Table (DRT)

The DRT, as initially assembled and updated through parameter file options, lists disk flaws that Install uses to locate the first good track. Install updates the DRT upon detecting additional flaws. Deadstart and Restart reconstruct the DRT based on flaw table information and DATs for permanent datasets; they also set up the area reserved for system dumps.

Dataset Catalog Table (DSC)

Install creates the DSC dataset on the master device. Deadstart and Restart use the DSC for bringing up the system following a idle down or system interruption,.

When Startup sets the DRT bits for the Dataset Catalog (DSC), the system dump area, the system overlay dataset, or datasets encountered in the DSC, the DRT is checked to see if the bit is already set. If any occurrences of such a conflict are found, a special flag is set and a note is made of each such device and track number. Following completion of the normal dataset recovery pass, a second pass is made through the DSC and all datasets which have conflicts are identified with system log messages.

System Dataset Table (SDT)

Install and Deadstart initialize the SDT as having all entries in the available queue. Restart uses the DSC to recover the queues for system input and output datasets and makes entries in the SDT accordingly. SDT entries are threaded into the input and output queues.

Dataset Name Table (DNT)

The Startup task initializes the DNT for the DSC.

Device Allocation Table (DAT)

Deadstart creates a DAT for the DSC dataset.

Dataset Parameter Area (DSP)

Startup and PDM use their own internal separate DSPs for manipulating the DSC. There is no central DSP in this regard.

Permanent Dataset Information Table (PDI)

Install computes the number of hash pages and the number of overflow pages and stores them in the PDI and in the device label. Deadstart and Restart retrieve these values from the device label.

Device Label (DVL)

Install writes a device label on the first usable track of each mass storage device. The label contains the device name and flaw information for each device. The master device label also contains the DAT for the DSC and a pointer to the first track of the reserved system dump area as well as a pointer to the system overlay dataset. A special flag in the label identifies the master device.

A Device Label (DVL) usually resides on the first track of its disk storage unit (DSU). This is because of I@DVLRES, an installation parameter, that reserves tracks at the front of a DSU for attempts to write the DVL. Usually, the first attempt is successful and the rest of the I@DVLRES tracks are available for user datasets. However, if enough bad tracks are discovered when trying to write a DVL, the DVL could inhabit any track.

Because the DVL track location cannot be known beforehand, Deadstart and Restart search for each DVL. To prevent false DVL finds, each DVL contains the ASCII characters DLB, the logical device name, and a checksum.

When a DVL is located, its track is reserved in the Device Reservation Table (DRT). Also, each flaw mentioned in the DVL is reserved in the DRT. For the master device, the DSC tracks mentioned by the DVL are also reserved in the DRT and the DSC DAT is rebuilt in memory. The master device label also contains the track number of the first track pre-allocated for system dumps and the system overlay dataset. These areas are also reserved in the DRT.

Equipment Table (EQT)

The EQT is used by Startup as a source of information to describe devices and the hardware configuration. It may be modified by the parameter file.

Rolled Job Index Table (RJI)

The Rolled Job Index is either initialized or read into memory, depending on type of Startup and operator options in the Startup parameter file. This index controls the recovery of rolled out jobs.

Queued Dataset Table (QDT)

Install and Deadstart initialize the QDT with no entries in use. Restart uses the DSC, as well as the information in the roll files, to recover the QDT. During a Deadstart, those user-permanent DSC entries with a non-zero QDT index are rewritten with the QDT field cleared.

Overlay Control Table (OCT)

The OCT contains flags and pointers for determinine the current status of the overlay area for the associated task.

Overlay Call Stack (OCS)

The Overlay Call Stack contains ten entries for each task.

Overlay Load Request List (OLL)

The OLL tracks initial load requests occurring while the overlay areas is busy.

4.1.3 STARTUP SUBROUTINES

The COS startup task (Startup) is created by EXEC. Startup executes only once--when the operating system is loaded and started up. Although communication areas exist for Startup, no tasks can ever place requests in the registers or request that this task be readied. (In this section, readying the task means clearing its suspended bit.)

Startup leaves messages in memory to notify the operator of failures during the COS Startup procedure.

Three main subroutines along with many helper subroutines comprise the Startup Task. They are Z, RRJ, and SDRREC.

Z subroutine

The three Startup options (Install, Deadstart, and Restart) run as the first portion of Startup in STP in the form of a closed subroutine called by Startup via a return jump to entry point Z. Z resides at the high-address end of STP. Z is executed just after Startup has created all STP tasks with the exception of the Job Scheduler (JSH), log manager, job class manager, and memory error tasks. When Z completes execution, Startup creates the remaining tasks in STP. Z executes a return jump to subroutine RRJ (Recovery of Rolled Jobs) just prior to exiting. RRJ in turn carries out any manipulation of the Rolled Job Index dataset that may be required due to operator specification or installation-selected defaults. Since the code of Z is not needed again, as one of its final functions, Startup moves the image of CSP to overwrite Z and adjusts pointers accordingly so that the unused memory is made available for user jobs.

JSH and the Station Call Processor (SCP) are two of the tasks created by Startup. JSH activity is stimulated by the System Dataset Table (SDT) entries comprising the input queue. JSH is not readied if the input queue is empty. Similarly, SCP activity is stimulated by entries in the output queue. The queues are assembled as being empty and are left empty by the Install and Deadstart options; therefore, JSH and SCP remain idle when either of these options is selected during Startup. Similarly, the queue of available SDT entries is assembled as containing all of the SDT entries.

When the Restart option is selected, however, it sets up SDT entries from the DSC and, therefore, alters the input, output, and available queues. In this way, Restart notifies JSH and SCP that queues exist for them to process.

If recovery of rolled jobs is selected, JXT entries may also be constructed to reflect jobs that can be successfully recovered. In this case, certain JSH flags are set up so that the Job Scheduler will be aware that jobs are already in the execution queue.

The COS startup procedure requires the time and date for handling the DSC entries. It obtains the current time and date from the operator station which is passed to EXEC. EXEC converts the date and time to CRAY-1 clock periods and sets the real-time clock to this value.

The SCP task may be active during execution of Z, responding to station messages. However, a flag in STP controls which messages STP processes immediately and which are postponed until Z completes.

Also required is the memory size of the CRAY-1 on which COS is executing. The actual memory size is defined by an installation parameter (I@MEM) or via a Startup parameter file statement (*MEMSIZ).

RRJ subroutine

All three Startup routines call subroutine RRJ before calling System Directory Recovery (SDRREC) and before processing system dumps. RRJ executes as a closed subroutine called by Z and performs any processing of rolled out jobs or the index dataset required. RRJ is called before Z executes SDR recovery or copies any existing system dump, since disk space needed to restart a rolled job must be recovered and allocated in the Disk Reservation Table (DRT) before any new space can be used. RRJ does not return any status used by Z; it does set a status word indicating the type of recovery performed, which is used by JSH to determine how much JXT initialization JSH must perform.

RRJ performs one of several activities depending on the type of Startup being performed.

RRJ execution during Install - Recovery of rolled jobs cannot be performed during an Install, since permanent datasets and input/output queues are not recovered. Therefore, RRJ merely initializes $ROLL and issues a SAVE request to the Permanent Dataset Manager (PDM). The initialization of $ROLL consists of setting up entry 0 (See RJ table description in the COS Table Descriptions Internal Reference Manual) and zeroing all other entries. The buffer used to write $ROLL remains intact in memory throughout normal operation of the system, and $ROLL is never read during normal operation.

RRJ execution during Deadstart - Since input/output queues are not recovered during Deadstart, rolled jobs cannot be recovered. RRJ attempts to access $ROLL and read it into memory. The buffer remains intact throughout normal operation, and $ROLL is never read again during normal operation. If RRJ was enabled by operator specification, RRJ detects that it is a Deadstart, issues an error message, and disables recovery.

Once $ROLL has been successfully accessed and read in, the contents of entry 0 contents are checked. If errors occur on the access or read, or if entry 0 does not validate correctly, RRJ issues error messages and re-initializes $ROLL. A new edition of $ROLL is created if the access was unsuccessful or if the existing edition received an error while being read. Otherwise, the new $ROLL is written over the existing one. If no errors are received, the $ROLL buffer is cleared to indicate no executing jobs and the dataset is rewritten.

RRJ execution during Restart - If Restart was selected, RRJ may attempt to recover jobs. This depends on whether RRJ is able to successfully access and read $ROLL. Error conditions here are handled as for Deadstart. If the access and read are successful but RRJ was not enabled by the operator, then RRJ clears $ROLL as for Deadstart. If RRJ is enabled, RRJ begins scanning the index entries following verification of entry 0. If an error occurs during entry 0 validation, RRJ disables recovery with a message to the system log and continues as for Deadstart.

If no errors occur during $ROLL validation, RRJ attempts to recover jobs. Messages are issued to the system log explaining why a job mentioned in the index is not recovered or indicating a successful recovery. A successful recovery means that the job has been entered into the JXT chain at the appropriate spot and the input System Dataset Table (SDT) entry has been moved from the input queue to the execute queue. The job status in the JXT is then set to "rolled out" and "suspended by recovery." The "waiting for memory", "pending abort", and "operator suspended" bits are maintained. All other status bits are set to 0, as are any event wait words. Any caller who requested recall based on an event is responsible for determining whether the event is satisfied or whether the recall should be re-issued. For example, any outstanding ACQUIRE requests may have to be re-issued.

SDRREC subroutine

SDRREC is executed as a closed subroutine that is called by Z after Recovery of Rolled Jobs (RRJ) is complete but before the system dump is copied. RRJ must be executed first to ensure the integrity of datasets belonging to any jobs being recovered. Any failures during SDR recovery cause the operating system to terminate abnormally.

File allocation - SDR recovery begins with a request to access the $SDR dataset. If no dataset exists, the number of blocks (segments) required to contain the current number of generated resident SDR entries is computed. A request is issued to the Disk Queue Manager to allocate disk space for the dataset. Then a request is made to the Permanent Dataset Manager to SAVE the dataset. Once the operating system initialization is complete, entries can be added to the SDR by ACCESS requests specifying the ENTER parameter.

SDR recovery - If the $SDR dataset exists, each block of the dataset is read and processed until a logical record with a binary zero dataset name is found or until the system-specified number of SDR entries is processed. A Dataset Name Table (DNT) is built for each dataset. The DNT and PDD in the logical record are used to ACCESS the dataset. Then the dataset is entered into the Permanent Dataset Table (PDS). If the dataset access fails, a message is issued to the system log, and the entry is ignored.

No recovery specified - If the operator specifies *SDR in the parameter file to indicate that the System Directory is not to be recovered, a new edition of $SDR is allocated. Once the operating system initialization is complete, entries can be added to the SDR by ACCESS requests specifying the ENTER operand.

Changes in the number of SDR entries - If System Directory recovery detects that the system-generated number of SDR entries is greater than the value saved in the $SDR header record, the number of blocks required by the system is calculated. If additional blocks are required, write requests are issued until all blocks are allocated. An ADJUST request is issued to the Permanent Dataset Manager to update the DSC for $SDR, and processing continues for SDR recovery.

If System Directory recovery detects that the number of SDR entries specified by the system has decreased, and if no recovery is specified, then the dataset is cleared, and the altered number of SDR entries is recorded in the header record. Once the operating system initialization is complete, entries can be added to the SDR by ACCESS requests specifying the ENTER parameter.

If the number of SDR entries specified by the system has decreased and recovery is to be performed, a message is issued to the system log, and initialization is abnormally terminated.

Two-pass Startup

A 2-pass startup is detected when Startup encounters the *BOOT and *SYSTEM directives in the parameter file. These directives tell Startup that the system currently executing is for locating and transferring control to another version of the operating system resident on the CRAY-1 disks. The procedure followed by pass 1 is identical to that of a normal Startup to the point where the datasets in the Dataset Catalog (DSC) are to be recovered. At that point, pass 1 makes a task request to the Permanent Dataset Manager (PDM) to locate the system dataset. Once the system dataset has been found, Startup validates the DSC entries for the dataset. Next, the Disk Queue Manager (DQM) reads the dataset into available memory. The final step Startup performs before requesting EXEC to move the system is to build the boot exchange package, location 20, of the new system. This indicates that pass 2 is about to begin.

4.1.4 INSTALL

Install requires a track on each mass storage unit and uses the first track that the DRT says is good for the device label. After writing the device label, Install reads it back to verify it. If the verification fails, the track is noted in the DRT as being bad and the next available track is used.

Flaw information for a device is written on the disk as part of the device label and is updated at the end of Install. Anything Install writes is validated, and any flaws are noted in the DRT and in the device labels.

On the master device, the device label points to the DSC by containing the DAT for the DSC. The master device label also contains a pointer to the first track of an area allocated by Install to contain system dumps.

Install creates the DSC in blocked format upon the master device while logging any disk errors found in the DRT and in the flaw table within the DVL image being built in memory. The DSC (except for block control words) is initialized to all zeros.

The system dump area is zeroed, and any flaws are entered into the DRT and flaw table. The amount of mass storage allocated is determined from the value of I@DMPSIZ. The list of allocation indices allocated to the system dump is written to the first sector of the first track in the reserved area for use by Deadstart and Restart. The system overlay dataset is zeroed and the Allocation Index list written to sector 0 of the first track.

In general, I/O on a dataset requires a buffer, DSP, DNT, and DAT. Install uses separate buffers and DSPs of its own when working with device labels and the DSC. At assembly of STP, space for the DNT and DAT are set aside for post-Install use. However, Install must complete initialization of the DNT and the DAT.

The default job class structure is in effect after an Install.

4.1.5 DEADSTART

Deadstart executes as an operator option if recovery of the input and output queues is not possible or desirable. Deadstart scans the DSC for input and output datasets, deleting all such datasets to help create the effect of an idle system. Permanent datasets are preserved.

In memory, Deadstart updates the DRT to reflect those tracks reserved by datasets mentioned in the DSC and tracks mentioned in the flaw portion of the device labels. Deadstart also creates the DAT (derived from the DSC DAT in the Master Device label) and initializes the DNT for the DSC. Deadstart calls RRJ to set up the Rolled Job Index. This is done before system dump processing.

Deadstart also attempts to locate the disk area pre-allocated for the system dump. If this area can be found, Deadstart reserves the tracks in the DRT. If a dump that has not been copied exists in the pre-allocated area, Deadstart copies the dump to a new dataset and requests the Permanent Dataset Manager to save the copy so that it can be accessed by user jobs following completion of startup. If no new dump exists, the disk space remains reserved, but the pre-allocated area is not copied. If the pre-allocated area cannot be found, no space is reserved in the DRT.

Deaddstart also attempts to locate the system overlay area. It it can be found, Deadstart reserves tracks for it in the DRT.

The buffers and DSPs for the DSC are assembled into STP.

To summarize, Deadstart:

- Finds Device Label (DVL)
- Reserves flaws
- If master device, reserves DSC and the disk space occupied by system dump and system overlay areas and initializes DNT and DAT for the DSC
- Deletes each input and output dataset and reserves all other permanent datasets
- Establishes Rolled Job Index in memory
- Copies system dump from the pre-allocated area to available space and saves the copy as a permanent dataset.

Deadstart places the job class structure that was written to the permanent dataset named in the Deadstart parameter file into effect. If no dataset is named, or if it cannot be accessed or read, the default job class structure goes into effect.

4.1.6 RESTART

Restart executes as an operator option following an interruption of the system where recovery of the input and output queues and jobs in process is desired.

In memory, Restart updates the DRT to reflect those tracks reserved by datasets mentioned in the DSC and tracks mentioned in the flaw portion of the device labels. Restart also creates the DAT (derived from the master device label) and initializes the DNT for the DSC.

The DSPs for the DSC are defined during assembly of STP. Recovery of rolled jobs is performed if necessary. The system dump area is recovered and possibly copied as described under Deadstart.

Restart scans the DSC to find all entries with the input flag or output flag set. From these input and output DSC entries on mass storage, Restart creates SDT entries and DATs in memory. The input and output queues are threaded by forward and backward link pointers. The first item in the queue is the one first saved.

During Dataset Catalog (DSC) recovery, Startup processes the allocation information for multitype datasets. Startup processes the first DSC entry for a multitype dataset as a normal entry, allocating the dataset and initializing it for associated DSC entries. Processing of the subsequent entries differs from normal recovery for error pre-processing, DAT body processing, and post-DAT validating. Since all DSC entries for a dataset are interrelated, any recovery errors are carried through to all of the entries.

DAT body processing involves a comparison of the DAT body chain from a previous DSC entry. Any differences cause the dataset to be flagged as having inconsistent allocation, and all DSC entries are processed accordingly during a second pass over the DSC.

Post-DAT validation processing essentially involves QDT update and is performed only when all preceding entries for the dataset pass recovery validation.

When Restart completes execution, the Startup task creates the JSH task. A non-empty input queue causes JSH to begin job scheduling. Similarly, a non-empty output queue activates SCP. The SCP task was created before the call to the Z routine.

JSH may also be activated if at least one job was recovered and placed into the execution queue by subroutine RRJ.

A Restart with recovery of rolled jobs recovers the job class structure that was in effect before the system interruption from a permanent dataset, PDN=JOBCLASSROLLED. If a disk error makes recovery impossible, the structure that was written to the permanent dataset named in the Restart parameter file goes into effect.

Restart without recovery of rolled jobs places into effect the structure that was written to the permanent dataset named in the Restart parameter file.

In either case (Restart with or without recovery), the default structure goes into effect if no dataset is named or if the named dataset is inaccessible.

The system overlay dataset is reserved in a way similar to the way in which the system dump area is treated. If no system overlay dataset was preallocated or if the validation checks for it fail, recovery of rolled jobs cannot be performed. In this case Startup halts if recovery is specified, or it allocates the area following permanent dataset recovery if recovery was disabled.

To summarize, Restart:

- Finds Device Label (DVL)
- Reserves flaws
- If master device, reserves DSC and initializes DNT and DAT for the DSC
- Reserves all permanent datasets and the system dump area and system overlay dataset, if possible
- In memory, builds DAT and SDT for each input/output dataset
- If specified, recovers rolled out jobs via call to RRJ
- Allocates system overlay dataset if necessary
- Copies system dump if necessary and saves the copy as a permanent dataset.

4.1.7 JOB RECOVERY BY RESTART

Following any system failure, whether due to software, hardware, or environmental problems, the system operator may attempt to recover any job in the execution queue at the time of the failure. This section describes job recovery and related operations.

Currently, Startup can successfully recover and restart all jobs that were rolled out to mass storage at the time of the system failure, or those that rolled out, rolled back in, and performed no additional activity to cause the roll image on mass storage to be unusable. A job can be recovered only if it is certain that the roll image is valid and that repeating any of the activities of the job following roll-in will not cause the results of the job to change. In some cases, a job that has been rolled out but has subsequently been rolled in and reconnected to the CPU may have executed some function that makes the system unable to determine whether the job can be successfully restarted from the roll dataset image. In this case, the job is declared irrecoverable and the Startup task leaves the job in the input queue. Subsequently, EXP attempts to rerun the job from the beginning. If a job is not recoverable and is ineligible for rerun, it is returned to the input

queue by Startup and terminates with an informative message in both the user and system logs as soon as the Job Scheduler attempts to re-initiate the job. A job that has initiated but has never been rolled out cannot be recovered since there is no roll image to recover. However, permanent datasets accessed following roll-in might not be available following a system recovery if one or more mass storage devices become unavailable. In this event, the recovered job receives an error status when attempting to re-access the datasets. Any permanent datasets already accessed by a job prior to roll-out must be re-accessed successfully during Startup before the job is considered successfully recovered.

Recovering a job from its latest roll image is performed in the steps described below. An error in any validation step renders the job irrecoverable, and an appropriate message is sent to the system log.

Index entry validation

The first step is to validate the information in the index entry. The job cannot be recovered if the index states that the job is irrecoverable, or if the roll dataset is either non-existent or resides on a non-existent or unavailable device.

The job is also considered irrecoverable if the date/time stamp in the index entry does not match the date/time stamp of the system being restarted, if field JTEPC is nonzero in the job's JTA, and if the operator or installation specifies not to recover such jobs.

Roll dataset validation

The partition header information in the index entry is used to read in as much of the roll dataset as can be located from the one word of allocation indices contained in the index. It must contain enough of the JTA for the job to locate the copy of the full roll dataset DAT, which was copied along with the JXT image to the JTA by the Job Scheduler immediately before rollout. An error on the read makes the job irrecoverable.

Once the first read completes, the JTA size values taken from the JTA and from the saved copy of the JXT, are compared. An error occurs if the two do not match. This size is then used to determine if more JTA exists. If more does exist, the additional information is read in.

Normally, the entire JTA is read in by the first read, but if many large datasets exist, the JTA may be quite large. RRJ must have the whole JTA in memory at once. It is an error if the JTA does not fit into available memory above the message stack.

The image of the roll DAT is moved from the JTA to the STP DAT area. An error results if not enough DAT pages can be allocated in STP to hold the DAT. The roll DAT is then validated. If no errors are found in the DAT, any remaining portion of the JTA and the last block of the user field are read in. They must fit, and the reads must have no errors. The last block of the user field is located using the JXCJS field of the saved JXT copy. The Job Scheduler stores the current value of the real-time clock in the first block of the JTA and in the last block of the user field immediately prior to roll-out. If they do not match, the roll-out was only partially complete at the time the system failure occurred, which is an error condition.

DAT validation

Each dataset, including the roll image dataset, must have a zero DAT address in the DNT or must point to a valid DAT. The roll image dataset and the $CS and $IN datasets point to DATs in the STP tables; all others point to DATs in the JTA. To be considered a valid DAT, the following points must be satisfied:

- A multipage DAT must be entirely within the STP tables or entirely within the JTA. It cannot be in both places.
- The DAJORD field for a DAT in STP must be equal to 0; the DAJORD field for a DAT in the JTA must be equal to the JXT ordinal.
- Successive pages must be numbered correctly.
- A DAT in the JTA must be pointed to by a negative offset that must be within the range indicated by the JTA size; the same is true for each successive page.
- For each partition, the named device must exist and must be available (EQNA must equal 0).
- Each allocation unit index for a partition must be within range for the device.
- For a multitype DAT (DNQDT is nonzero), each allocation unit index must have its corresponding DRT bit set; otherwise an inconsistent allocation has occurred.
- For a DAT that is not multitype (DNQDT is 0), each allocation unit index must not have the corresponding DRT set; otherwise an allocation conflict has occurred.
- When the end of the last page or last partition is reached, the remaining AI count and next partition pointers must be 0.

- When the end of a partition is reached, the next partition pointer (DANPA) must point to either the next word in the current page or the first word following the page header in the next page, or it must be 0.

DAT validation occurs in two passes. The first pass serves as an error scan and does not set the DRT bits. The second pass actually sets the DRT bits and decrements the available space counts for the device. In this way, RRJ can be sure that a dataset is either completely reserved or completely unreserved. This is necessary for successful deallocation of resources if a later dataset is found to have an error.

Dataset reservation

Each dataset named in the DNT chain in the JTA must be processed. Local datasets must have their DATs validated and the DRT bit maps updated. Permanent datasets must be validated against the Dataset Catalog. Startup will already have updated the DRT bit maps for permanent datasets. Permanent Dataset Table (PDS) entries must also be reconstructed for permanent datasets.

The DNT chain is scanned from the beginning to the end. The memory pool control word preceding each DNT is checked to be sure that the pool entry is in use, and the DNT is checked to ensure that there is a name. If there is no DAT, RRJ goes on to the next DNT. If there is a DAT and it is in STP (DNDAT is greater than 0), the DNT must be for either $CS or $IN. The SDT entries are searched for an SDT with the correct sequence number, and the DAT address field of the DNT is corrected. RRJ then goes to the next DNT.

If the DAT is in the JTA (DNDAT is less than 0), the DNT is checked to see if the dataset is permanent. If it is not, the DAT is validated. If it is, a pseudo access is performed. If no errors are found, RRJ goes to the next DNT. When the end of the DNT scan is reached, the job is considered successfully recovered.

Pseudo access of permanent datasets

When a permanent dataset is encountered in the DNT scan, RRJ requests the Permanent Dataset Manager (PDM) to perform a pseudo access on the dataset. This process causes the Permanent Dataset Manager to locate the DSC entry for the dataset from the DADSC field of the DAT and to compare the DAT in the JTA with the DAT in the DSC.

If the DAT appears valid, PDM attempts to construct or update a PDS entry. The DNT permission flags are used to set the PDS permission flags. If the PDS entry already exists, the DNT must indicate read-only permission.

Resource de-allocation

If an error occurs at any point in the recovery of a job, any system resources assigned to that job by RRJ must be released. In particular, any disk space reserved for local datasets prior to finding an error on a later dataset, or any PDS entries corresponding to datasets that have already been pseudo accessed must be de-allocated. For this purpose, the DNT chain is searched until the DNT with the error is reached again. For releasing local datasets, the Disk Queue Manager de-allocate request is used. For releasing PDS entries, the PDM request PMFCRL is used. The disk space for datasets such as $CS or $IN, which has its DAT in STP, is not released. The roll image dataset is released and its STP DAT pages are returned to the system.

Job recovery completion

When the end of the DNT chain is reached without error, the job is successfully recovered. The copy of the JXT from the JTA is placed in the JXT area, and the JXT entries are re-linked by priority. The roll image DNT within the JXT is updated to point correctly to the DAT, the SDT entry is moved to the execute queue, and the JXT ordinal is placed in the SDT. All wait words are cleared. The JXT status bits are set to R, N, and B ("rolled out," "not in memory," and "suspended by recovery") and all other bits except O, A and M (operator suspended, abort pending, and waiting for memory) are zeroed. The SDT address in the JTA is corrected, and the JTA is rewritten to the roll image dataset. If field JTEPC is nonzero, if the date/time stamps in the index entry and the current system do not match, and if the operator or installation has elected to recover and lock out such jobs, the JXT and SDT lockout bits are set. A message is sent to the system log and RRJ advances to the next index entry.

Termination of RRJ

When the end of the roll index is reached, all entries corresponding to jobs that were not recovered have been cleared. The input queue is scanned, and all jobs that were previously initiated are flagged with a status in the SDT so that CSP will issue log messages when the jobs are re-initiated. Such jobs may be ineligible for rerun, in which case the status passed to CSP reflects that condition. CSP then terminates the job immediately after issuing the log file messages. The status word RRJSTAT is set to indicate to the Job Scheduler that the JXT entries are already initialized and linked, and the JSH flag SWAPFLAG is set if any jobs were recovered. RRJ then returns to Z.

4.2 DISK QUEUE MANAGER (DQM)

The Disk Queue Manager task (DQM) controls the simultaneous operation of Disk Storage Units on CPU I/O channels or the I/O Subsystem. DQM provides:

- Allocation/deallocation of mass storage
- Management of mass storage resources (channels, controllers, and disk storage units)
- Management of disk storage unit request queues

Another task readies DQM whenever it needs allocation, deallocation, or access of mass storage. After satisfying the request, DQM readies the calling task and suspends itself. EXEC readies DQM when an I/O request finishes or when a sector transfer completes for a dataset in recall.

4.2.1 SYSTEM TABLES USED BY DQM

DQM uses the following system tables. Detailed information on these tables is available in the COS Table Descriptions Internal Reference Manual, publication SM-0045.

DAT	Dataset Allocation Table
DCT	Device Channel Table
EQT	Equipment Table
RQT	Request Table
DRT	Device Reservation Table
DSP	Dataset Parameter Table
JXT	Job Execution Table
JTA	Job Table Area
DNT	Dataset Name Table

Figure 4.2-1 illustrates the linkages of tables used by DQM.

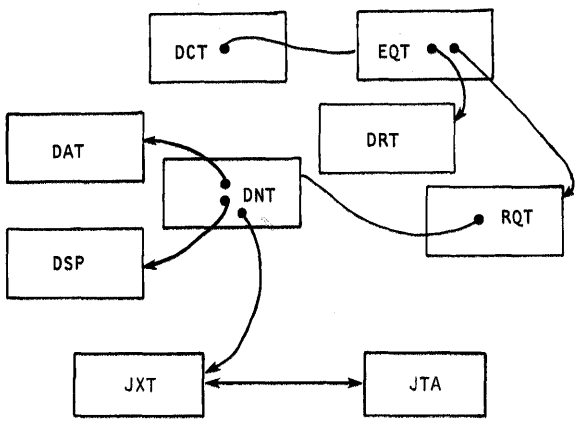


Figure 4.2-1. DQM table linkages

Dataset Allocation Table (DAT)

The DAT resides in the STP common table area or the JTA and associates datasets with physical space in one or more devices. The DAT consists of a header and a body. The header is used primarily for managing the body which is composed of a pool of 16-word pages. Pages are assigned to DAT entries as needed.

The DAT contains an entry for each active dataset. DQM creates a DAT entry for a dataset when the dataset is opened (if preallocation is requested) or when the user makes the first write request on the dataset. For a permanent dataset, a new DAT entry is not created but rather the DAT for the dataset maintained in the DSC is copied into as many DAT pages as are required.

Figure 4.2-2 illustrates the structure of the DAT.

A dataset's DAT entry contains a header and one or more partitions. Each partition represents a separate device on which space is allocated for the dataset. A partition header contains a pointer to the next partition for the dataset. The size of a partition depends on the number of allocation units assigned to the dataset on the device. A partition has four 16-bit allocation indices per word in the partition.

When a partition overflows the current page, DQM adds a page to the DNT's dataset entry. Page headers logically link pages comprising the entry.

Device Channel Table (DCT)

The DCT contains information for channel control among each channel's attached devices. Only one device may be active on a channel at a time.

Equipment Table (EQT)

The EQT contains information for device allocation, physical operation control, device request queue management, channel configuration, performance monitoring, and error counting.

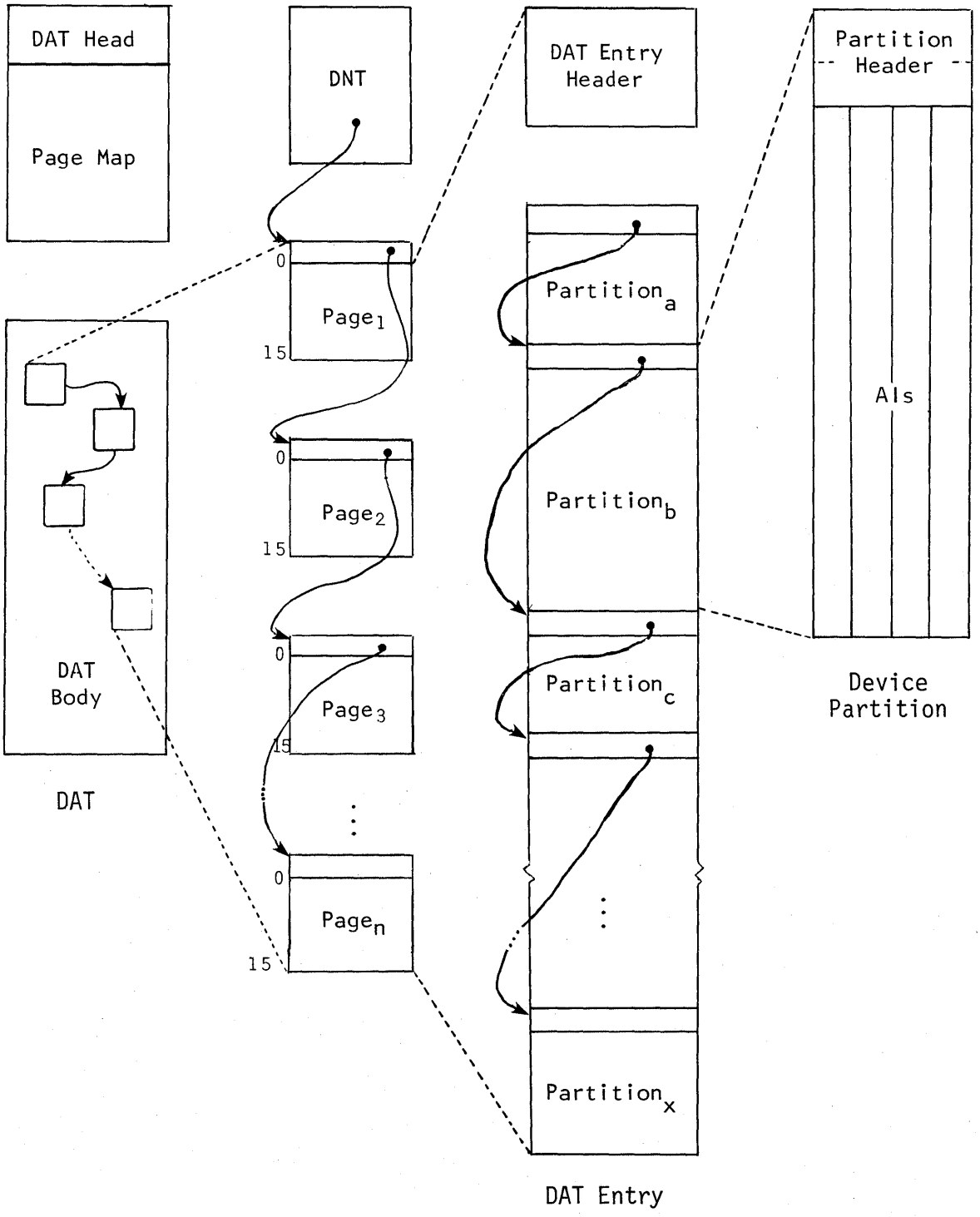


Figure 4.2-2. DAT structure

Request Table (RQT)

The RQT contains information to be communicated between the I/O requester and the Disk Queue Manager.

Device Reservation Table (DRT)

The DRT for a device contains a bit map showing reserved allocation units for the device. DQM manages this table when it allocates or deallocates space on the device.

Dataset Parameter Table (DSP)

The DSP contains information for logical I/O requests. If the DNT has a DSP associated with it, DQM uses this table to build the DNT request word.

Job Execution Table (JXT)

The JXT contains information about all active jobs. DQM uses this table to determine the Job Table Area (JTA) location.

Job Table Area (JTA)

The JTA contains information about a specific job. DQM uses this table for I/O accounting.

4.2.2 DQM INTERFACE WITH OTHER TASKS

A task calls DQM through the PUTREQ routine which places the requested function in INPUT+1 and the dataset's DNT address in its INPUT+0 register and exits with an EXEC request to ready DQM.

Allocation

INPUT REGISTERS:	INPUT+0	40/,24/DNT address
	INPUT+1	56/,8/1
OUTPUT REGISTERS:	OUTPUT+0	64/,status
	OUTPUT+1	40/,24/DNT address

Deallocation

INPUT REGISTERS:	INPUT+0	24/0,16/JXT,24/DNT
	INPUT+1	56/,8/2

OUTPUT REGISTERS:	OUTPUT+0	64/,status
	OUTPUT+1	24/0,16/JXT,24/DNT

Queue I/O

INPUT REGISTERS:	INPUT+0	24/return[S],16/JXT,24/DNT
	INPUT+1	56/,8/0
OUTPUT REGISTERS:	OUTPUT+0	64/,status
	OUTPUT+1	24/return,16/JXT,24/DNT

4.2.3 DATASET ALLOCATION

The Disk Queue Manager supports two allocation modes: preallocation and dynamic allocation. Preallocation is supported both explicitly and implicitly. That is, other parts of the system may separately request preallocation prior to writing a dataset or may simply write to the dataset, in which case, preallocation is performed as the first step in the write process. Dynamic allocation is performed on datasets that are not preallocated or on which overflow occurs for their preallocated sizes. Storage is allocated dynamically in multiples of tracks as specified at the time of dataset definition.

A dataset explicitly assigned a logical device starts on that device if space is available but may overflow to other devices.

To dynamically allocate a dataset, the DNT need only contain a logical write request specifying the starting sector number and the number of sectors or a DSP address. If the currently assigned device becomes full, another device is selected by the device allocation scheme defined later.

For those allocation requests not specifying a logical device name, the available disks are assigned to new dataset requests in a round-robin fashion.

The order of allocation is the order of entries in the Equipment Table (EQT). The Equipment Table should be constructed so rotation occurs among channels first and then among units. In other words, the first entries should be unit 0 entries arranged in order of ascending channel number. Varying the order of entries in the EQT generates a system that allocates units in a different order.

[S] The return field is normally used to save a return address needed in particular by CIO. If the queue I/O interface is used directly without going through CIO, the return field may contain any information that needs to be preserved.

Currently, only one allocation style is supported; that is, one track per allocation index.

Allocation units

Each time DQM assigns an allocation unit to a dataset, it places the allocation index (AI) for the device in the dataset's DAT entry and sets the corresponding bit in the DRT. Deallocation causes the DAT and the DRT to be cleared and the DAT to be released.

The size of an allocation unit is currently defined as one track (eighteen 512-word sectors).

Allocation is always to the device most recently assigned to the dataset, that is, to the most recent device partition in the DAT entry.

4.2.4 RESOURCE MANAGEMENT

DQM manages the channels, controllers, and storage units assigned to it to provide:

- Maximum responsiveness to I/O requests
- Maximum throughput of I/O requests
- Streaming of data where possible

DQM manages the following three types of hardware controllers: DCU-2, DCU-3, and DCU-4. The DCU-2 and DCU-3 controllers are directly connected to the CRAY-1 CPU I/O channels. The DCU-4 controlles are integral to the Buffer I/O Processor and Data I/O Processors in the I/O Subsystem and control DD-29 Disk Storage Unit.

DCU-2 and DCU-3 controller management

For the DCU-2 and DCU-3 controllers, DQM maximizes channel availability by releasing channels while positioning occurs on units. This allows several units to be positioning simultaneously or several units to be positioned simultaneously with transfer from a single unit. The channel and controller configuration is illustrated in figure 4.2-3.

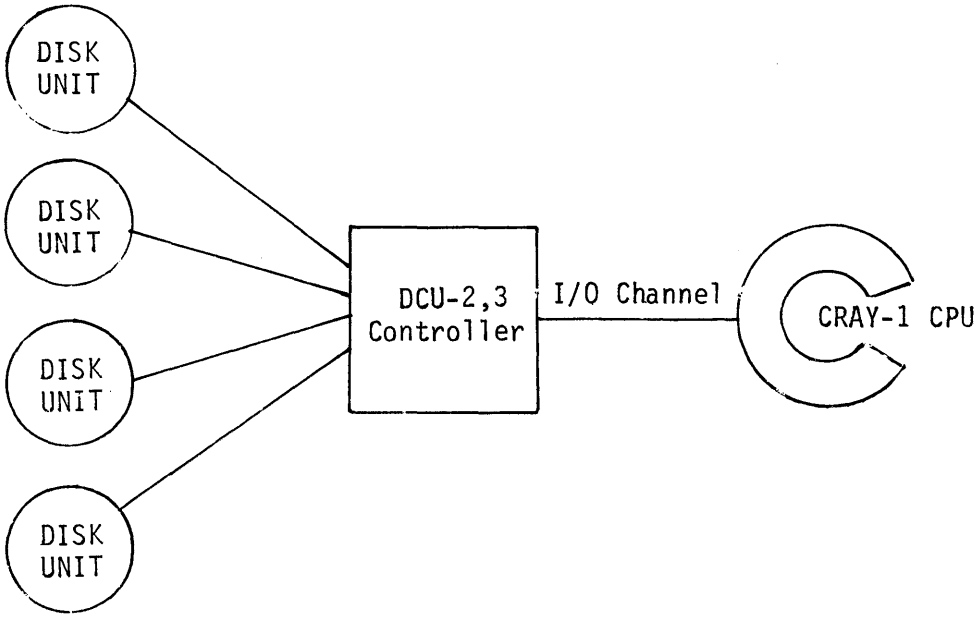


Figure 4.2-3. DCU-2, 3 Controller configuration

DCU-4 controller management

The DCU-4 controllers reside in the BIOP and DIOP in the I/O Subsystem (See figure 4.2-4).

Each DCU-4 controller supports four simultaneous data paths. To DQM, these appear as if there were a one-to-one relationship between controllers and disk storage units. Figure 4.2-4.

Storage unit management

DQM is responsible for maintaining the logical status of each unit, advancing each unit to the next state, and initiating and controlling error recovery. The possible logical states for the storage units are:

- Idle
- Waiting seek issue
- Seek issued
- Waiting transfer issue
- Transfer issued
- Transfer reissued (error recovery in process)

DQM controls error recovery. It initiates retries and selects whether retry shall be with margins. This is in addition to the automatic retry on time-out or error that is present in the disk driver. Each retry failure is logged in the system log. If a correctable error occurs on the last retry, DQM initiates a low priority task (Disk Error Correction task) to correct any correctable disk errors.

4.2.5 QUEUE MANAGEMENT

Each device has a separate queue. The disk queue manager enters requests in the queue and services them in the order they are received. To provide streaming, a request is not considered complete until a DSP indicates that no more I/O can be performed for this request.

When a request overflows a device, the DQM places it at the bottom of the queue for the next device.

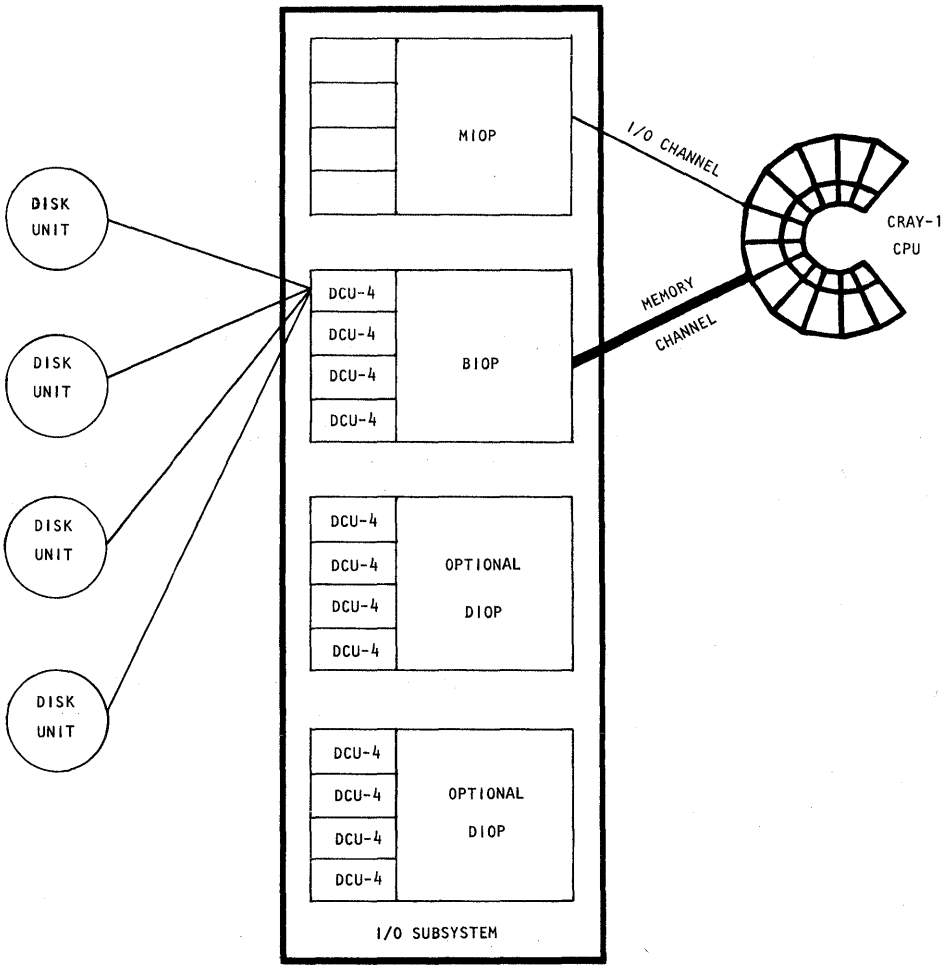


Figure 4.2-4. DCU-4 controller configuration

4.2.6 I/O REQUEST FLOW IN DQM

1. DQM receives I/O request from another task.

2. The requesting task is determined and request entry space is reserved.

3. The DSP request is mapped into a logical request in the DNT.

4. The DQM is started; enqueue transforms the logical request into a physical request and enqueues it.

5. DQM is started. If no requests are outstanding on the requested channel, a zero-length transfer seek is issued and the task is suspended.

6. DQM is restarted after the hardware accepts the seek function and checks for any other seeks. If none, it issues the transfer request and suspends.

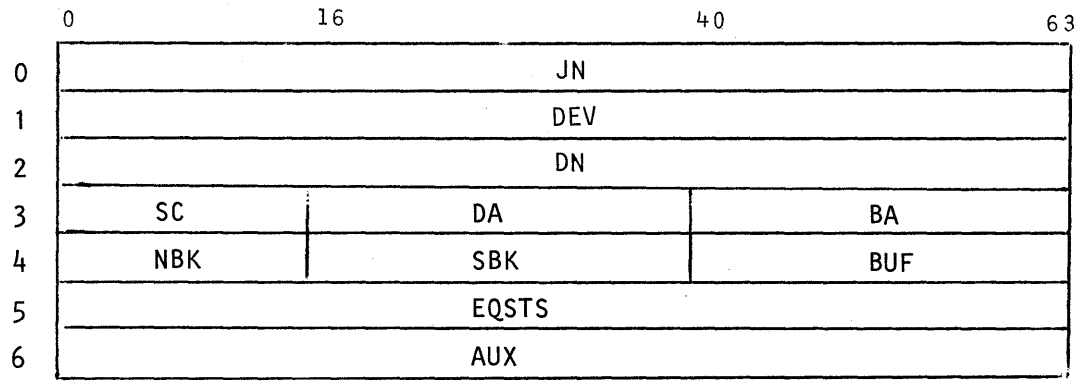
7. DQM is restarted. If the dataset is in recall and a half buffer has been transferred, then the requesting task is restarted. If it is the last sector of the request, the DSP is examined for more I/O. If there is more I/O, go to step 1 and repeat the sequence. If there is no more I/O, go to step 8. If an error occurs, an entry is made in the log and the failing block is retried with margins.

8. DQM is restarted. It dequeues the finished request and starts the requesting task. DQM is requested to initiate the next request.

9. The task is suspended.

4.2.7 HARDWARE ERROR LOGGING

DQM logs disk errors by calling the message task with a binary record to be inserted into the system and user logs. The record format is:



Field	Word	Bits	Description
JN	0	0-63	Job name or system in ASCII
DEV	1	0-63	Device name
DN	2	0-63	Dataset name
SC	3	0-15	Sectors remaining in request
DA		16-39	Disk address (12/cylinder, 6/track, 6/sector)
BA		40-63	Current buffer address
NBK	4	0-15	Number of blocks
SBK		16-39	Starting block number
BUF		40-63	Initial buffer address
EQSTS	5	0-63	Edited status from EQT
		1	Error flag
		2	Time out flag
		3	Read/write response error
		24-39	Subsystem status last retry
		24-29	Undefined
		30	Channel parity $2^{12} - 2^{15}$
		31	Channel parity $2^{8} - 2^{11}$
		32	Channel parity $2^{4} - 2^{7}$
		33	Channel parity $2^{0} - 2^{3}$
		34	Read checkword error
		35	DSU ready error
		36	DSU not on cylinder
		37	DSU index error
		38	IO verification error
		39	DSU reservation error
		48-63	Subsystem status first try
		48-53	Undefined
		54	Channel parity $2^{12} - 2^{15}$
		55	Channel parity $2^{8} - 2^{11}$
		56	Channel parity $2^{4} - 2^{7}$
		57	Channel parity $2^{0} - 2^{3}$
		58	Read checkword error
		59	DSU ready error
		60	DSU not on cylinder
		61	DSU index error
		62	IO verification error
		63	DSU reservation error
AUX	6	0-63	Auxiliary status (to be defined)

4.3 STATION CALL PROCESSOR (SCP)

The Station Call Processor (SCP) handles functions for one or more front-end computer systems and provides for:

- Establishing communications with the front-end system,
- Responding to front-end requests for functions such as stream control, I/O transfer, and status requests,
- Managing I/O transmission buffers,
- Receiving datasets containing jobs or data,
- Transmitting output datasets to the designated front-end system,
- Multiplexing of streams for each logical station,
- Multiplexing of logical stations on the same hardware channel,
- Recovering from link errors and front-end system failures,
- Routing of messages and datasets to networks, and
- Allowing operator intervention to re-establish link communication.

The SCP task is readied by the system executive (EXEC) front-end driver whenever an output/input pair completes on a channel assigned to front-end communications.

This section assumes the reader is familiar with the contents of the Front-end Protocol Internal Reference Manual, publication SM-0042.

4.3.1 SYSTEM TABLES USED BY SCP

SCP uses the following system tables:

LIT	Link Interface Table
LST	Logical Stream Table
LXT	Link Extension Table
SDT	System Dataset Table
PDD	Permanent Dataset Definition Table
LCT	Link Configuration Table
AUT	Active User Table
IBT	Interactive Buffer Table

These tables are described in detail in the COS Table Descriptions Internal Reference Manual, publication SM-0045.

Link Interface Table (LIT)

SCP maintains one LIT entry for each station hardware channel. This table communicates between the front-end driver and SCP and controls multiplexing of logical stations.

Link Extension Table (LXT)

The Link Extension Table is the working storage area for SCP. The LXT contains all the information to control stream multiplexing for a logical station.

Link Stream Table (LST)

LST contains the information for controlling an input or output stream.

System Dataset Table (SDT)

SCP has prime responsibility for the SDT. It takes entries from the available queue for jobs being assigned to the input queue and returns entries to the available queue after staging output datasets to the front-end system.

Permanent Dataset Definition (PDD)

SCP, as a user of permanent dataset management, must generate PDDs to accompany requests for saving and deleting permanent datasets. These PDDs reside in SCP rather than in a user field.

Link Configuration Table (LCT)

LCT contains information for reserving buffer space and other configuration information for each physical channel that is attached to a front end.

Active User Table (AUT)

The Active User Table controls interactive communication. It contains messages queued for input and output as well as associated information.

Interactive Buffer Table (IBT)

The Interactive Buffer Table provides control over buffer space allocated for interactive messages.

4.3.2 PROCESSING FLOW FOR SCP

Upon receipt of each message, SCP checks the input LCP in the channel buffer for illegal message code and illegal parameters. Any error causes an immediate MESSAGE ERROR response. Errors are retried to a limit, then a RESTART code is sent.

If a data segment was sent on the previous transmission from the CRAY-1, SCP assumes that the segment has been accepted and calls the Disk Queue Manager (DQM) to transfer the next segment of this dataset from disk to the output buffer.

SCP then processes the input LCP message code as follows:

1. LOGON causes SCP to save LOGON parameters and to initialize the buffer pool. If LOGON was not expected, the LOGOFF procedure is also followed.

2. LOGOFF causes SCP to deallocate all incoming datasets and makes the associated SDT entries available. All outgoing dataset SDT entries are returned to the output queue.

3. RELOG is a no-op.

4. CONTROL is a no-op.

5. DATASET HEADER causes an SDT entry to be assigned and the header parameters to be saved in the SDT. If edition number is specified, SCP calls PDM to verify that this edition number does not already exist. If edition number exists, a CANCEL is sent for the stream.

6. DATASET SEGMENT causes a segment to be written to disk using dynamic allocation. If DC=IN, SCP interprets the first record (the JOB statement), and saves the jobname, priority, time limit, and field length in the SDT entry.

7. JOB STATUS, SYSTEM STATUS, LINK STATUS, MASS STORAGE STATUS, DATASET STATUS, OPERATOR DEBUG, LOGFILE INFORMATION, LOGFILE ENTRY REQUEST, and INTERACTIVE REQUEST must be replied to before a second request of this class may be sent. Except for the DATASET STATUS, replies are built immediately.

8. DIAGNOSTIC ECHO REQUEST causes the appropriate ECHO REPLY to be built and transmitted immediately.

9. MESSAGE ERROR causes immediate retransmission of the prior message.

10. DATASET TRANSFER REPLY causes the status field to be examined. If the status is NO, the request is deleted and the originating job is dropped. If the status is YES, the request is removed from the request queue. If the status is POSTPONE, the request is reissued after a delay defined by I@DTRDLY.

SCP then processes the input stream control bytes. Input stream (front end is the sender) types are as follows:

1. IDLE - Normal state if front end has no activity for this stream. No SDT entry is associated. CRAY-1 response is IDLE.

2. REQUEST TO SEND - Front end wishes to stage a dataset. SCP immediately responds with PREPARING TO RECEIVE.

3. SENDING - Front end is prepared to send DATASET HEADER. SCP immediately responds with RECEIVING.

4. END DATA - Front end is awaiting DATASET SAVED response from CRAY-1. CRAY-1 responds with SAVING until dataset is saved.

5. CANCEL - Front end requests that this stream's current activity be dropped. SCP deallocates incoming dataset and makes the associated SDT entry available. If this dataset originated from a dataset transfer request, the originating job is dropped. The CRAY-1 response is IDLE.

6. POSTPONE - Front end requests that this stream's current activity be dropped. SCP deallocates the incoming dataset and makes the associated SDT entry available. If this dataset originated from a dataset transfer request, the originating job is not dropped. The CRAY-1 response is IDLE.

7. MASTER CLEAR - Front end has encountered an invalid CRAY-1 response. For an input dataset, SCP deallocates it and makes the associated SDT entry available. CRAY-1 response is IDLE.

SCP then processes the output stream control bytes. Output stream (CRAY-1 is the sender) types are as follows:

1. IDLE - Normal front end response if CRAY-1 state is IDLE, POSTPONE, CANCEL, or MASTER CLEAR. CRAY-1 responds by issuing either IDLE or REQUEST TO SEND if an output dataset was queued. The SDT entry is removed from the output queue and is assigned to this stream.

2. PREPARING TO RECEIVE - Front end response indicating preparing to accept DATASET HEADER. The header is not eligible to be sent.

3. RECEIVE - Front end response indicating readiness to accept DATASET HEADER or DATASET SEGMENT. Next segment is transferred from disk to memory and is then made eligible to be sent.

4. SUSPEND - Front end wishes to temporarily stop receiving segments for this stream. CRAY-1 holds response as SENDING.

5. DATASET SAVED - Front end indicates dataset has been saved and stream may be released. SCP calls PDM to delete the dataset, makes the SDT entry available, and responds by issuing IDLE.

6. POSTPONE - Front end wishes to postpone receipt of this dataset. SCP places the associated SDT entry for this dataset at the end of the output queue and responds by issuing IDLE.

7. CANCEL - Front end wishes to cancel receipt of this dataset. SCP releases the dataset and makes the associated SDT entry available. If this dataset originated from a dataset transfer request and the WAIT flag is set in the SDT, then the job is aborted. The response is IDLE.

8. MASTER CLEAR - Front end has encountered an invalid CRAY-1 response. For a dataset in the process of being staged out, SCP returns the associated SDT entry to the output queue and responds by issuing IDLE.

The output LCP must now be constructed. If a DIAGNOSTIC ECHO REPLY is ready, it must be sent; otherwise, if a DATASET TRANSFER REQUEST is queued, it is sent. If any nonstaging reply is ready, the appropriate message code -- JOB STATUS, SYSTEM STATUS, DATASET STATUS, LINK STATUS, MASS STORAGE STATUS, OPERATOR DEBUG, LOGFILE INFORMATION, INTERACTIVE REPLY, and LOGFILE ENTRY REPLY -- is used.

If there is no non-staging transmission to be sent, SCP examines the output streams for transmission of eligible headers or segments using the appropriate message code -- DATASET HEADER or DATASET SEGMENT. If no information is to be returned, the message code is CONTROL. SCP moves the CRAY-1 stream control bytes (SCBs) into the output Link Control Package (LCP) to complete the output LCP.

Finally, SCP assigns an input buffer and requests the station driver (via EXEC) to complete the output transmission and await input.

4.3.3 INTERACTIVE PROCESSING

Interactive processing requires an interactive terminal attached to a concentrator CPU, since the CRAY-1 does not support its own terminals. The concentrator packs terminal messages into a message segment and distributes them to the users. Output messages destined for an interactive concentrator are also packed into a segment by SCP for distribution.

Station types

COS recognizes three station types: batch only, interactive only, and both. The batch-only station cannot send or receive interactive messages. The interactive-only station can exchange only logon, logoff, start, restart, control, diagnostic echo, and interactive message types. A station that supports both interactive and batch can exchange all message types. The station type is defined in a parameter of the logon segment.

Terminal modes

Two terminal modes, buffered and unbuffered, are supported. An unbuffered terminal must receive a response message before it can send another message. A buffered terminal can send or receive a terminal message in any interactive segment. The exceptions are the terminal logon and logoff messages, which require explicit responses before further messages can be processed.

Interactive request message processing

Upon receipt of an interactive request, SCP verifies that the station is of the proper type and that no interactive reply is outstanding. If the segment bit count (SGBC) is nonzero, the data within the segment is also validated. Individual terminal messages are then processed by message type.

Logon (1) - A logon message type is checked to determine whether it is an initial or a subsequent logon by scanning the Active User Table (AUT) for an entry belonging to the user. If no entry is found, an initial logon sequence begins. If an entry is found and the currently logged off flag is set, the subsequent logon sequence begins. If the entry is active, access is denied, and a restart is queued for output with an error code stating the problem.

For an initial logon, the user name and password are verified against the system dataset containing a list of valid user names.§ If there is no match, access is denied, and a restart message is sent to the terminal. A System Dataset Table (SDT) entry is created with the origin type set to interactive for a valid user. The Control Statement Processor (CSP) is called to process the first input message. A start message containing the process identifier for all future communication is queued to the terminal.

---

§ Deferred implementation

Processing a subsequent logon is identical to processing an initial logon except that an SDT entry is not created.

In the event of a concentrator crash, the station logon processing logs off all users coming through that station. A flag is set in the AUT showing the problem. The user must perform another LOGON, which is treated as a subsequent LOGON.

Data (4) - With a data message type, SCP validates the process number and places the data on the input message queue indicated by the process number. If the job was suspended for input, processing is resumed.

Special Function (5) - SCP responds to a special function message type by performing an abort or a status, as required.

For the abort function, SCP requests the Job Scheduler (JSH) to perform an advance job step, which loads CSP. All messages in the job's input message queue are flushed, and CSP waits for input from the terminal.

When the status function is received, SCP queues a data message to the terminal containing CPU time used, job status, and last log message.

Logoff (6) - When a logoff message type is received, SCP determines which of the following three types of bye processing to select by examining the value in the special function field.

0 Terminal is marked as logged off in the AUT; and SDT and JXT entries are retained. Only the most recent output messages are retained.

1 The job presently running is killed at the completion of its current job step. CSP is then loaded to perform termination, and all output messages are discarded.

2 Terminal is marked as logged off in the AUT. The SDT and JXT entries are retained and all output messages are retained. If necessary, job processing is suspended until the output is transmitted.

Control (10) - SCP performs the state change associated with a control message. A data message may also contain control information.

Error (11) - When an error message is received, SCP posts the error in the AUT and attempts to retransmit the last message.

Interactive reply processing

After a request has been processed a reply is built. A Dataset Transfer Request from a station of both batch and interactive types can interrupt this process.

SCP then places terminal messages in the segment buffer in the following order:

1. Error,

2. Controls for buffered mode state changes,

3. Restarts,

4. Data for buffered mode and for logon and logoff replies for both terminal modes, and

5. Data for nonbuffered mode terminals and control for nonbuffered mode terminals if I@IAPOLL time has expired.

If there are no terminal messages to be sent, an Interactive Reply with SGBC=0 is sent.

Start (2) - SCP generates a start message type when a logon is valid. This message contains a process identifier, which must be used on all future communications and echoes the logon text.

Restart (3) - SCP creates a restart message type when an invalid logon or unexpected message is received. The error code field describes the nature of the problem. The possible problems are:

- Invalid user name
- Invalid password
- Already logged on

If the response is to an invalid logon, the logon data is returned with the restart.

Logoff Reply (7) - A logoff reply message type is created when a logoff is received, and all required termination is finished. The logoff message type always contains a BYE text message.

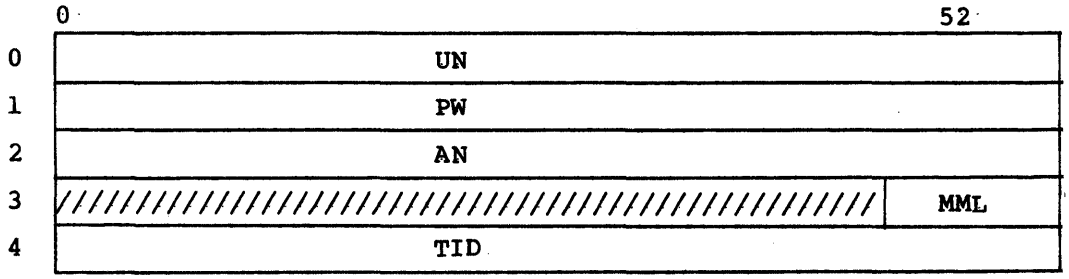
Data (4) - SCP builds the output data type text message from the output message queue for a process. More than one output line may be sent in a message. Record control words for interactive messages must be present in the text. Word 0 of the text is the first interactive record control word. If a job is suspended for output, it is reactivated.

The maximum length of a data message is set by the MML field in the logon message. A message is sent only if it fits entirely within the segment buffer. Messages may cross subsegment boundaries but may not cross segment boundaries.

Error (11) - SCP sends an error message to indicate a problem with the most recent message, identified by the message number.

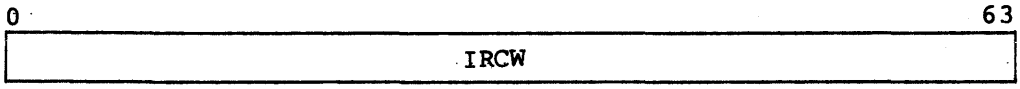
Control (10) - SCP sends a control message to change the state of an input process or to respond to a nonbuffered terminal if there is no data and the poll interval specified by I@IAPOLL has expired.

Text for logon, start, and restart messages - There are no record control words associated with this text.

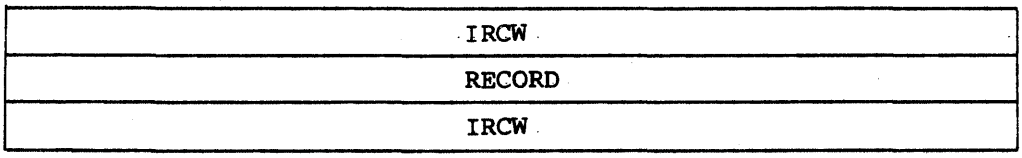


Field	Word	Bits	Description
UN	0	0-63	User name (bits 0-55 are used for the job name)
PW	1	0-63	Password
AN	2	0-63	Account number
MML	3	52-63	Maximum message length in CRAY-1 words
TID	4	0-63	Terminal ID

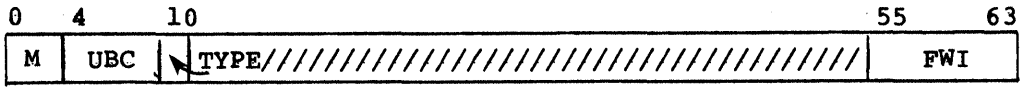
Text structure for input or output data - The text associated with a terminal entry may contain one or more variable length records with a record control word between each record as illustrated below.



RECORD



The interactive record control word (IRCW) has the following form:



Field	Bits	Description
M	0-3	An octal number indicating the mode: 10 End of record (EOR) 16 End of file (EOF) 17 End of data (EOD) 0 Block control word (BCW)
UBC	4-9	Unused bit count of the previous record
TYPE	10	Mode for the following record 0 Line (default) 1 Transparent
FWI	55-63	Pointer to the next record control word

This record structure is similar to CRAY-1 dataset format. Each record is terminated with a record control word. The last record control word contains a 0 FWI. The first record control word must be a BCW.

Transparent record mode

To allow more control over the output to terminals, COS supports a transparent record mode. When this mode is specified, there is no blank compression, no trailing blank compression, and no line feed caused by the *eor* RCW. The mode is defined as a property of the dataset and may be altered on a record bias. The mode is set by a bit in the DDL when the dataset is defined.

Operations control

An operator KILL or DROP command returns the SDT and the JXT resources to the system. Further attempts by the terminal to access the job results in error messages being returned to the front end. The error code indicates either not logged on or logged on but no job exists.

4.4 EXCHANGE PROCESSOR (EXP)

The Exchange Processor (EXP) is a task that processes all user system action requests and user error exits. The Exchange Processor also handles certain requests from the Job Scheduler (JSH) to initiate or abort a job.

EXP recognizes that certain functions prevent the restarting of a job from its most recent roll image without potentially yielding results different from those that would be obtained had the job not been restarted. In these cases, EXP declares the job to be irrecoverable and causes the Job Scheduler to update the Rolled Job Index accordingly.

Similarly, EXP recognizes that certain functions (notably permanent dataset manipulations) make it uncertain whether a job could be rerun from the beginning without changing its results. In these cases, EXP declares the job to be non-rerunnable. In cases where the user knows that changes to permanent datasets would not affect the correct execution of the job if it is rerun, the user may override EXP and declare the job rerunnable or may prevent EXP from declaring the job not rerunnable. See Job Rerun and Job Recovery later in this section for an explanation of recoverable and rerunnable jobs.

When a user program exchanges to the system due to normal exit, error exit, or execution error, the Executive (EXEC) sets flags in the word JTEP of the Job Table Area (JTA) to request execution of Exchange Processor and to indicate whether the exchange was a normal or an error exchange.

JSH requests EXP by setting another flag in the same word (JTEP) in the JTA. The EXEC readies EXP and exchanges to it instead of exchanging to the user whenever the JTEP word is nonzero for the currently connected job.

When EXP finishes processing a request, it clears the JTEP word to allow EXEC to return to the user job.

If EXP cannot finish a request immediately, it suspends itself without clearing JTEP. EXEC then continues to return control to EXP rather than the user, as long as that job is assigned to the CPU.

In general, EXP calls JSH to suspend the job before suspending itself when it must wait for completion of a request to another task, such as for an I/O request. This allows other jobs in the system to be assigned the CPU.

4.4.1 SYSTEM TABLES USED BY EXP

All EXP functions are job related. Consequently, most of the tables used by EXP are either in the user field or in the Job Table Area (JTA) which is immediately below the user field.

The Exchange Processor accesses most system tables, the most important of which are:

JXT	Job Execution Table
CALL	Call table
SDT	System Dataset Table
QDT	Queued Datset Table

Detailed information of the JXT and SDT tables is available in the COS Table Descriptions Internal Reference Manual, publication SM-0045.

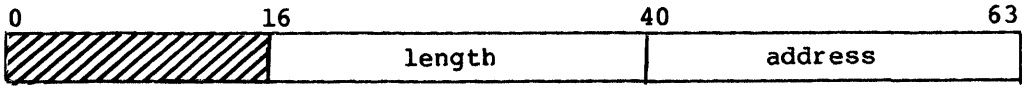
Under certain circumstances, EXP allocates a table in the memory pool area of STP. This table is used as a PDD for rewriting an SDT entry in the DSC.

Job Execution Table (JXT)

The Job Execution Table contains an entry for each job which has been initiated. It contains job parameters and statistics which may be required while the job is rolled out to disk.

CALL Table (CALL)

The CALL table is a static table composed of 1-word entries for each user system action request. The contents of S0 on a user call serves as an index into the table. The format of each entry is the following:



length Length of a table whose address is in S1 of the user exchange package. Length is 0 if S1 does not contain a table address.

address Address of the routine that processes the request

System Dataset Table (SDT)

The System Dataset Table contains an entry for the job dataset for each job in execution. EXP creates an entry in the SDT for each output dataset (job output and disposed datasets). It also allocates an SDT if something is disposed to the input queue.

Queued Dataset Table (QDT)

EXP modifies the QDT (via the common subroutines RELDNT) when a job releases a local scratch dataset that has related disposes.

4.4.2 USER AREA TABLES USED BY EXP

EXP uses the following tables located either in the user field or in the JTA.

Dataset Definition List (DDL)

The Dataset Definition List is used to pass dataset parameters used in creating the DNT on a F$DNT call.

Dataset Name Table (DNT)

The DNT is a table in the JTA containing an entry for each dataset of a job. The DNT is used to pass parameters to the Disk Queue Manager. The DNT contains pointers to the Device Allocation Table and to the active DSP for the dataset. The DNT also contains important dataset characteristics and status.

Dataset Parameter Table (DSP)

The DSP is required for all user I/O. It contains pointers to the dataset buffer. DSPs for system-managed datasets such as the control statement file, $CS, and the user logfile ($LOG) are contained in the JTA. An additional DSP, for the F$DJA and F$EXU functions, is also contained in the JTA. All other DSPs are located in the user field, conventionally in an area reserved for DSPs at the high end of the user field.

Job Communication Block (JCB)

The JCB occupies words 0 through 177_8 of the user field and is used for communicating between EXP and the user.

Logical File Table (LFT)

The LFT is a table near the high end of the user field containing a 2-word entry for each dataset which has a DSP in the DSP area. The LFT points to the DSP. If the dataset has been assigned an alias, it will have more than one LFT entry.

Open Dataset Name Table (ODN)

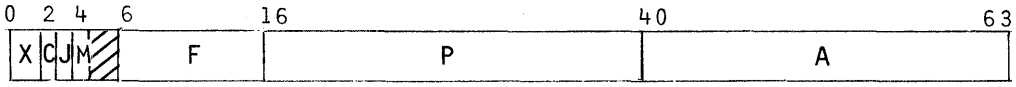
The ODN table in the user field is required when opening (or closing) a dataset (F$OPN call).

Permanent Dataset Definition (PDD)

A PDD table in the user field is required for a user permanent dataset management request (F$PDM request). A PDD table also exists in the JTA for use by EXP when releasing a dataset.

4.4.3 EXCHANGE PROCESSOR REQUEST WORD

All requests to the Exchange Processor are made via the Exchange Processor request word (JTEP) in the JTA for the job assigned to the CPU. The Exchange Processor is readied by EXEC whenever JTEP is nonzero. The format of JTEP is as follows:



Field	Bits	Description
JTEPX	0-1	User exit 2 Normal exit 1 Error exit or execution error 0 Not user exit
JTEPC	2	Continuation flag
JTEPJ	3	Job Scheduler request flag
JTEPM	4	JTA expansion request flag
JTEPF	7-15	Exchange package flags
JTEPP	16-39	P register for errors
JTEPA	40-63	Memory error address; EXP address if JTEPC=1

The flags in fields X, C, and J are mutually exclusive. The user exit flags in field X, normal exit and error exit, are set by EXEC when the user causes a normal or error exchange. The continuation flag, field C, is set by the Exchange Processor when an EXP function must be restarted after being partially processed. In this case, JTEPA contains the P address for the interrupted function. The Job Scheduler request flag is set by the Job Scheduler to request that a job be initiated or aborted.

4.4.4 USER NORMAL EXIT

Exit from a user program occurs when the user program executes a program exit instruction (004). The user issues a system action request on a program exit by setting S0 to the desired function code and setting S1 and S2 to optional arguments before exiting. When the request completes, the user's S0 contains a status code. Conventionally, (S0)=0 indicates no error.

If an error is encountered, the job normally aborts with appropriate messages issued to the logfile. For some errors, however, an error code is placed in the user's S0 and the user is allowed to continue processing. If the Control Statement Processor (CSP) is executing as the user, S0 returns an error code for all but a few fatal errors.

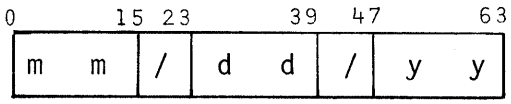
When EXP is readied, it detects the user request because the JTEPX field for the currently executing job contains a 2. The function code in S0 is then used as an index into the CALL table to obtain the address of the routine to process this request. If the length field in the CALL table entry is nonzero, EXP verifies that the address in S1 points to a table within the user field length. This address is converted to an STP-relative address. Next, the vital parameters in the Job Communication Block (JCB) are verified by comparing them with duplicate values in the JTA. The parameters checked are JCFL, JCNPF, JCBFB, and JCDSP. Several other fields are also verified as reasonable values.

4.4.5 SYSTEM ACTION REQUESTS

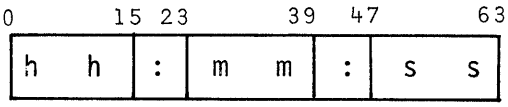
The mnemonic values used to assemble user codes are supplied by the system text ($SYSTXT). These mnemonics should be used to provide function codes for register S0 when making system action requests. Unless otherwise specified, a function has no effect on a job's ability to be recovered or rerun. See Job Rerun and Non-Recoverability of jobs in this section.

Mnemonic Code	Octal Value	Task Description
F$ADV	000	Advance job. The current job step is terminated and the job is advanced to the next control statement.
F$ABT	001	Abort. The job is advanced to the EXIT control statement if one exists. If none exists, the job is terminated.
F$DAT	002	Get current date. The current date in ASCII format is returned at the location specified in S1 in the following format:

Mnemonic Code	Octal Value	Task Description
(continued)		



F$TIM	003	Get current time. The current time in ASCII format is returned at the location specified in S1 in the following format:



F$MSG	004	Enter message in logfile. A message beginning at the location specified by S1 is written to the logfile. S2 is used to determine to which logfile the message is written.

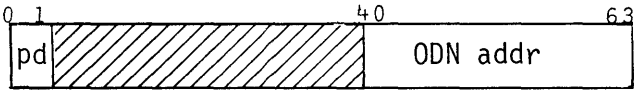
(S2)	Significance
1	User logfile only
2	System logfile only
3	System and user logfiles

The message is 1-80 characters and is terminated by a zero byte.

F$RCL	005	Dataset recall. The job is removed from execution until another block of data has been transferred without error or until I/O is complete on the dataset specified. S1 contains the ODN or DSP address.
F$TRM	006	Terminate job. The job is terminated normally and its resources are returned to the system.
F$SSW	007	Set pseudo sense switch. S1 contains the number of the switch to be set.
F$OPN	010	Open dataset. S1 contains processing direction in bits 0 and 1 and the address of the Open Dataset Name (ODN) table. Bits 40-63 of S1 contain the address.

Mnemonic Code	Octal Value	Task Description

(continued)



An OPEN call creates the following entries (if not already created) for the dataset whose name is in the first word of the ODN table:

- A DNT entry in the user's JTA
- An LFT entry
- A DSP entry
- Allocates a buffer if the dataset is to be in blocked format

Entries 2, 3, and 4 may result in moving existing LFT entries, DSP entries, and buffers. Additional user field is allocated if insufficient room exists for adding the LFT, DSP, or buffer. Parameters in the JCB of the user field reflect any movement of these tables.

o The negative DSP offset is equal to the DSP base address (JCDSP)-DSP entry address.

  This value is returned in bits 40-63 of word 2 of the ODN table.

o The DNT and DSP are modified to reflect the processing direction requested.

Mnemonic Code	Octal Value	Task Description

(continued)

NOTE

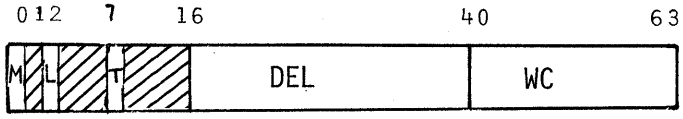
If word 2 of ODN, bits 39-63, already contains a positive, nonzero address, then the user is requesting that a user's own DSP and buffer be used. The address in the ODN points to the DSP, and the DSP and buffer must be contained in the user field below the high limit of user code (JCHLM). In this case, entries 2, 3, and 4 are omitted.

Processing Direction

01 Output
10 Input
11 Input/Output

F$MEM 011

Request memory. The amount of memory assigned to a job may be determined or changed. S1 contains the address of the memory request word. The job is aborted if filling the request would exceed the maximum allowable memory for the job. The memory request word has the following format:



M Maximum memory flag. If M is set by the caller, JSH returns in WC the maximum allowable amount of memory (in words) excluding the JTA. No memory is allocated.

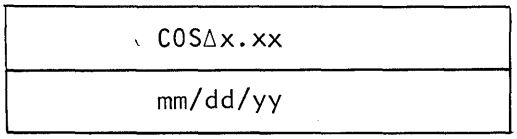
F$MEM

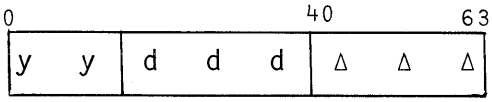
L Limit flag. The system sets this flag when it has assigned the maximum allowable amount of memory to the user.

Mnemonic Code	Octal Value	Task Description
(continued)		T Total flag. If T is set, WC represents the total memory requested (excluding the JTA) rather than an increment or decrement, and DEL is ignored. DEL Deletion pointer. If the user wants an increase in memory, DEL must be 0. If the caller wants a decrease in memory, DEL must contain the beginning address of the area to be deleted. WC Word count. The user must supply the absolute number of words to be added to or deleted from the user's field length. Any words added to the user's field length are added to its high-address end. If WC=0, no action is taken other than to return the user's field length in WC.
F$LBN	012	Return last block number. S1 contains the address of the Open Dataset Name table. On return, S2 contains the block number of the last block of the dataset. S2 contains -1 (all bits set) if the dataset is empty.
F$CLS	013	Close dataset. S1 contains the address of the Open Dataset Name (ODN) table. A close call does the following processing for the dataset whose name is in the first word of the ODN table: • Writes end of data on a sequential blocked dataset if dataset is write mode • If the dataset is in write mode and is a blocked dataset, flushes data in the buffer to disk and writes an *eod* RCW, if necessary. This may result in the job being declared temporarily irrecoverable. An unblocked dataset has no system buffer. • Releases any buffer for the dataset • Releases the DSP for the dataset

Mnemonic Code	Octal Value	Task Description
		• Releases any LFT entries for the dataset • Updates the DNT entry for the dataset to indicate that the dataset is closed
F$DNT	014	Return dataset characteristics - DDNFE=1, DDSTAT=1 S1 contains the address of the Dataset Definition List (DDL). This call performs the sense local dataset function first. If the DNT is located, a copy is made in the user area at the location specified by DDDNT. Additioan DDL parameters are ignored. On return: (S0)=0 if the dataset exists; (S0)≠0 otherwise. Create local dataset - DDNFE=0 S1 contains the address of the DDL. This call creates a DNT if one does not already exist. If the dataset already exists, it must be closed. Parameters from the DDL are inserted into the DNT. Sense local dataset - DDNFE=1, DNSTAT=0 S1 contains the address of the DDL. This call searches for a DNT. On return: (S0=0 if the dataset exists; (S0)≠0 otherwise. The dataset need not be closed. Additional DDL parameters are ignored.
F$MDE	015	Set exchange package mode. S1 contains the address of the word containing new mode setting. See CRAY-OS Version 1 Reference Manual, for mode settings.
F$GNS	016	Get next control statement. Copy one card image from control statement buffer to address specified in S1. Error code EREFR (1) is returned in S0 if end of file is encountered on the control statement file.
F$EXU	017	Load binary dataset at location specified in the PDT in the user field and begin execution. The address of the word containing the name of the dataset is in S1. Additional memory is allocated for the job if required to load the binary dataset.

Mnemonic Code	Octal Value	Task Description
F$RLS	020	Return dataset whose Open Dataset Name table address is specified in S1 to the system. The dataset is closed and disposed of according to the disposition code contained in the Dataset Name Table entry for this dataset. The dataset is no longer available to the job. This may result in the job being declared irrecoverable.
F$PDM	021	Permanent dataset management request. S1 contains address of the Permanent Dataset Definition (PDD) table. The format of the PDD depends on the function requested. This may result in the job being declared irrecoverable, not rerunnable, or both.
F$RDC	022	Read disk circular. S1 contains the DSP address. The error bits and the busy bit in the DSP must be monitored by the caller. Automatic recall is requested if bit 0 of S1 is set.
F$WDC	023	Write disk circular. S1 contains the DSP address. The error bits and the busy bit in the DSP must be monitored by the caller. Automatic recall is requested if bit 0 of S1 is set. This may result in the job being declared irrecoverable, not rerunnable, or both.
F$GRN	024	Get system revision numbers. S1 contains address of 2-word table. Information is returned in ASCII format, left-justified and blank-filled as follows:
F$DIS	025	Dispose dataset. S1 contains the PDD address. This may result in the job being declared irrecoverable.
F$JDA	026	Get current Julian data in ASCII format. The date is returned at the location specified in S1. The date is returned as follows:

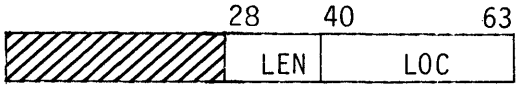


Mnemonic Code	Octal Value	Task Description
(continued)		

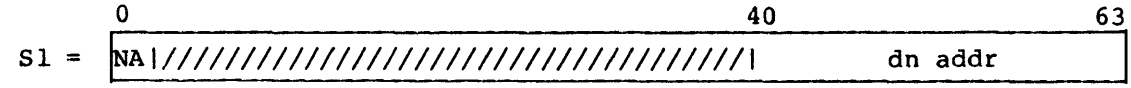
F$JTI	027	Return accumulated CPU time for the job in the location specified by S1. The time is expressed in seconds in floating-point format.
F$ACT	030	Accounting information from the JXT and the JTA is returned at locations starting with the address in S1. The format of the information returned is described by the Job Accounting Table (JAC).
F$SPS	031	Set P register and suspend user. New program address in S1.
F$CSW	032	Clear sense swtich. S1 contains the switch number to be cleared.
F$TSW	033	Test sense switch. S1 contains the switch number to be tested. On return: (S1)≠0 if sense switch is set (S1)=0 if sense switch not set
F$BIO	034	Buffered I/O request. S1 contains the DSP address. Perform record oriented I/O request on a COS blocked dataset. A record or partial record is transferred to or from a user-specified data area. Control returns immediately to the user, allowing the user to do processing in parallel with the I/O. The user must check status in the DSP for completion of the request and for errors. This may result in the job being declared irrecoverable.
F$BIO		The DSP must contain the following fields set by the user when the call is made: DPBIO Buffered I/O busy flag must be 0 indicating that any previous request has completed. This flag is set by the system when the call is made and cleared when the request is completed.

Mnemonic Code	Octal Value	Task Description
(continued)		If a user wants to relinquish the CPU and wait for completion of the buffered I/O request, the user should continue to call recall (F$RCL) until the buffered I/O busy flag is cleared.
	DPBER	Buffered I/O error flag must be 0, indicating that any error on the previous request has been recognized by the user. If an error has occurred when a request is completed, DPBER is set to 1. The user may then check DPERR to determine the nature of the error.
	DPBF	Function code: 000 Read partial record, logically equivalent to $RWDP 010 Read record, logically equivalent to $RWDR 040 Write partial record, logically equivalent to $WWDP 050 Write record, logically equivalent to $WWDR 052 Write end of file, logically equivalent to $WEOF 056 Write end of data, logically equivalent to $WEOD 156 Rewind, logically equivalent to $REWD
F$BIO	DPBWC	Word count is the number of words to transfer to or from the user's record area. On a read request, the system returns the actual number of words read. If a null record is read, a zero word count is returned in DPBWC. The user may then use DPEOR, DPEOF and DPEOD to determine if end of record, end of file, or end of data has been reached.

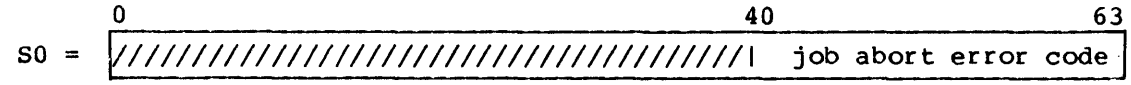
Mnemonic Code	Octal Value	Task Description
		DPBWA Word address of user's record area.
F$DLY	035	Delay job. The job is removed from processing for the number of milliseconds contained in the rightmost 24-bits of the location specified by S1.
F$AQR	036	Acquire dataset from front end. F$AQR first checks to see if the requested dataset exists on the CRAY-1 by issuing an ACCESS request to the Permanent Dataset Manager to obtain the dataset. If it is not present on the CRAY-1, it acquires the dataset from the front end and accesses it. S1 contains the address of the PDD.
F$NRN	037	Enable or disable job-not-rerunnable checks. S1 contains the address of a word containing the enable/disable flag. If the flag is 0, the job can be declared not rerunnable. If the flag is 1, the job cannot be declared not rerunnable. This does not affect the existing rerunnability of the job; if the job has already been declared not rerunnable, it remains so. Other flag values are illegal.
F$RRN	040	Enable or disable job rerun. S1 contains the address of a word containing the enable/disable flag. If the flag is 0, rerun is enabled; that is, an operator RERUN command or a system recovery places the job back into the input queue. If the flag is 1, rerun is disabled; that is, an operator RERUN command is rejected and a system recovery does not allow the job to be rerun from the beginning.
F$IOA	041	Set (lock) or clear (unlock) IOA bits in the JCB and JTA and alter accordingly the limit address in the user's exchange package. When the IOA bits = 1 (lock user's I/O area), the limit address is set to (JCDSP); when the IOA bits = 0 (unlock user's I/O area), the limit address is set to (JCFL). S1 contains the address of the LOCK/UNLOCK indication.

Mnemonic Code	Octal Value	Task Description
F$LFT	042	Delete, change, or create an LFT in the JTA. S1 contains the LFT address in the user field. S2 contains the operation to be performed on an LFT. DELLFT = 0 Delete an LFT CHGLFT = 1 Change an LFT CRELFT = 2 Create an LFT
F$INV	043	Invoke a job class structure. The job aborts if an F$INV request is already pending. S1 must contain the address of the invoke request word, which has the following form:  LEN The length of the array located at LOC. The job aborts if LEN is not a multiple of 1000_8 or is greater than the maximum size allowed. LOC The address of the array containing the job class structure to be invoked The array at LOC is copied to the Class Structure Definition Table (CSD) as soon as all of the job's I/O requests are complete. Then the class assignments of all the jobs in the input queue are redetermined. No Job Execution Tables (JXTs) are allocated while a J$INVOKE request is pending. (See JSH functions in section 4.5).
F$DJA	044	Dump job area. S1 contains the address of a local dataset name. If non-existent, the dataset is created.
F$RPV	045	Enable or disable reprieve processing. S1 contains the address of a 3-word table in the following format: 0 FWA of reprieve code +1 FWA of 30-word save area for exchange package and system use +2 Mask defining error classes to be reprieved

Mnemonic Code	Octal Value	Task Description
		S2 is set to 0 to activate reprieve processing. If S2 is nonzero, abort processing continues as if the reprieve never occurred, providing the capability to continue abort processing at the termination of reprieve.
F$BGN	046	Begin user code execution. S1 contains the address of the BGN table. The code to be executed is assumed to be loaded in the user field. BGN performs necessary housekeeping and calls the Job Scheduler to begin job execution.
F$SKP	047	Skip or end skipping of control statements. Set S1=1 to begin skipping control statements; set S1=0 to end skipping. This function forces the Exchange Processor ADV routine to ignore all control statements (for example, DUMPJOB).
F$PRC	050	Procedure dataset invocation. S1 contains the address of the procedure dataset name. Control statements are read from the indicated dataset until *eof* or RETURN is encountered, at which point reversion to the dataset containing the invocation occurs.
F$RTN	051	Procedure return; resume reading from the previous control statement dataset.
F$TBL	052	Save CSP-managed tables
F$INS	053	Jump to installation-reserved function. S1 should contain the subfunction code, an offset into the subfunction table. This function allows the installation an unlimited number of subfunctions.
F$UROLL	054	Roll a job; user requested rollout to protect against system interruptions.
F$ASD	055	Access system dataset. Search the System Directory for the dataset name which is pointed to by S1. S1 may also have the sign bit set to indicate no abort.



On return, if there was an error and the NA flag was set:



Job abort error codes:

12 JTA overflow
21 Dataset not found
76 Dataset already accessed by job

4.4.6 USER ERROR EXIT

When a user program executes an error exit instruction or encounters a hardware execution error (such as a floating point error, operand range error, or program range error), an exchange to EXEC occurs. EXEC readies the Exchange Processor after setting the following fields in the JTA of the job:

- JTEPX is set to 1.
- JTEPF is set to the exchange package flags in the user exchange package, bits 30-38 of word 3.
- JTEPP is set to the P register in the user exchange package.
- JTEPA is set to the word address of the failing memory word for an uncorrected memory error.

The Exchange Processor issues appropriate error messages and initiates job abort or reprieve processing.

The Exchange Processor skips through the job control statements to the statement following the next EXIT statement or to the end of file. If the statement following the EXIT statement is DUMPJOB, a dataset named $DUMP is created if it does not already exist. This dataset contains the job image, including the Job Table Area (JTA) and the entire user field.

4.4.7 JOB SCHEDULER REQUESTS

The Job Scheduler (JSH) requests the Exchange Processor to initiate or abort a job by setting the JTEP word in the job's JTA. The JTEP word must be 0 before JSH can modify it. JSH sets the JTEPJ field to 1, indicating a JSH request.

The Exchange Processor is readied by EXEC when the job is connected if the JTEP word is nonzero, that is, when it becomes the currently executing job. If JTEP is nonzero at the time JSH needs to use it, JSH sets the flag in JXT. As soon as the job is connected again and JTEP is 0, JSH sets the JTEPJ field to 1.

4.4.8 JOB RERUN

Under certain conditions, termination of job processing and returning to the input queue for reprocessing at some later time may be desirable or necessary. This is known as rerunning a job. When a job is rerun, the results of the second (or subsequent) execution should be the same as those that would have been obtained had the original execution continued to a normal termination. However, after a job has performed certain functions that have a lasting effect on the system (in particular, functions that make changes in the contents of permanent datasets or the Dataset Catalog), the system is unable to guarantee that the results of the rerun job will be the same.

Normally, when EXP recognizes that the user is performing one of these functions, the job is declared ineligible for rerun. A job ineligible for rerun cannot be rerun under any conditions. Normally, once a job is declared ineligible for rerun, that status is permanent. The job may become eligible for rerun again only if the user program specifically requests such a change using the F$RRN system call (RERUN macro or RERUN control statement). The system never declares a job to be again eligible for rerun without a specific user request.

Through the F$NRN system call, the user may prevent EXP from declaring a job not rerunnable regardless of what functions are performed. This prohibition remains in effect until the user specifically re-enables the detection of non-rerunnable functions. This does not affect the current rerunnability of a job; it merely prevents future declaration of non-rerunnability. The NORERUN macro and control statements may be used to enable or disable the detection of conditions that normally cause a job to be not rerunnable.

The functions that cause a job to be declared ineligible for rerun are:

- A SAVE of a permanent dataset,
- A DELETE of a permanent dataset,
- Any write operation involving a permanent dataset, and
- An ADJUST or MODIFY of a permanent dataset.

Conditions that may cause the system to attempt to rerun a job are:

- Operator entry of a RERUN command.

If the job has already been declared ineligible for rerun, the RERUN command is not accepted and the job is not affected.

- Disk error while attempting to read the roll image of a job that has been copied to mass storage.

  If the job is ineligible for rerun and the roll image cannot be read, the job is placed back in the input queue and aborts with an informative message as soon as the Job Scheduler attempts to re-initiate the job.

- System software or hardware failure necessitating a system restart.

  If recovery of rolled jobs is not performed or if the job is not recoverable (see Job Recovery), Startup attempts to rerun the job from the beginning. If the job is ineligible for rerun, it aborts with an informative message when the Job Scheduler attempts to re-initiate the job.

In any case, an informative message appears in the user log and in the system logfile whenever a job is rerun or a rerun is necessary but the job is ineligible for rerun. A rejected operator RERUN command produces no messages in either log.

4.4.9 REPRIEVE PROCESSING

Reprieve processing enables a user program to gain control in a uniquely identified routine when a job step completes either normally or abnormally. This routine is entered with reprieve processing disabled. The user program can recover from the termination; however, an abort due to an I/O error can produce unpredictable results if the dataset is accessed in the reprieve routine.

When a job step termination condition occurs, either normally or abnormally, the F$ADV or F$ABT system action routine determines if a reprieve request has been issued and if the termination condition has been specified by the user as reprievable. If so, the reprieve processing routine, ERPV, gains control and performs these tasks:

1. Clears the current reprieve values,

2. Copies the exchange package, vector mask register, error class code, and actual error code contents to the user-specified area, and

3. Sets up the user-specified reprieve routine to receive control when the job is selected for execution by placing its address in the program address register of the exchange package.

Reprieve processing is initiated either by issuing the SETRPV macro instruction in a CAL program or by calling the SETRPV library routine in CFT. Both requests invoke execution of the $SETRPV library routine. $SETRPV issues an F$RPV system action request, which saves the user specified reprievable error class code and the address of the reprieve code in the JTA.

The ENDRPV macro instruction in CAL or the ENDRPV call in CFT terminates the job step. The job step terminates as if reprieve processing had never been in effect.

4.4.10 NON-RECOVERABILITY OF JOBS

Functions that a job may perform causing the job to be declared not recoverable include:

- A random write on any dataset,
- A sequential write on any dataset immediately following any forward positioning, rewind, or read on that dataset. Thus, the position of the end-of-data is changed, which could cause the job to behave differently if started from a previous roll image.
- A SAVE, DELETE, ADJUST, or MODIFY of a permanent dataset, and
- The release of a local dataset, returning disk space to the system.

In any event, the job becomes recoverable as soon as the Job Scheduler rolls the job out to mass storage again.

A job is declared irrecoverable by a call from EXP to the Job Scheduler (JSH). If the job is already marked irrecoverable, JSH returns without further action. If the job is not already marked irrecoverable, JSH suspends the job, changes the Rolled Job Index Table (RJ), and writes the modified index to disk. When the modified index is successfully written, JSH resumes the job. The write of the index always occurs before EXP performs the request that makes the rolled image invalid.

4.5 JOB SCHEDULER (JSH)

The Job Scheduler is the task responsible for initiating the processing of a job, selecting the currently active job, managing job roll-in and roll-out, and terminating a job.

4.5.1 JOB FLOW

A job enters the system as an input dataset spooled from a front-end computer. As illustrated in figure 4.5-1, the input dataset is transferred to CRAY-1 mass storage by the Station Call Processor task (SCP). SCP also makes the job's existence known to the system by creating an entry in the System Dataset Table (SDT) and an entry in the disk resident Dataset Catalog (DSC). The latter entry makes the job's input dataset permanent until it is deleted at the completion of the job.

Any SDT entry contains information sufficient to identify and access the dataset. In addition, the SDT entry for a job input dataset contains the job's name, priority, field length, CPU time limit, $OUT size, and CL parameter all obtained from the JOB statement by SCP. Class membership is assigned by the Job Class Manager (see section 4.11) as soon as the job enters the input queue.

The Job Scheduler (JSH) selects jobs from the SDT for entry into the Job Execution Table (JXT) in the following manner:

- Classes are scanned from highest to lowest rank to find a class with an available JXT. A class has an available JXT when the class is ON, and it either has an unused reserved JXT, or it is not at its maximum and a pool JXT is available.
- When a class with an available JXT is found, its members are initiated in priority order (see section on initial memory priority). Jobs that have a priority of 0 are not initiated until the operator raises their priority. This process continues until all jobs are initiated or belong to a class having no available JXTs.

The size of the JXT depends on the value of the installation parameter I@JXTSIZ (defined in COSTXT), which imposes an absolute limit on the number of JXT entries (from 1-63). The operator can reduce the apparent size of the JXT further by using the LIMIT command, which affects the variable JXTMAX.

After the system Startup or a SUSPEND ALL, SHUTDOWN, or LIMIT 0 operator command, JXTMAX is set to 0 by the system. The LIMIT operator command should be used to set JXTMAX to a nonzero value.

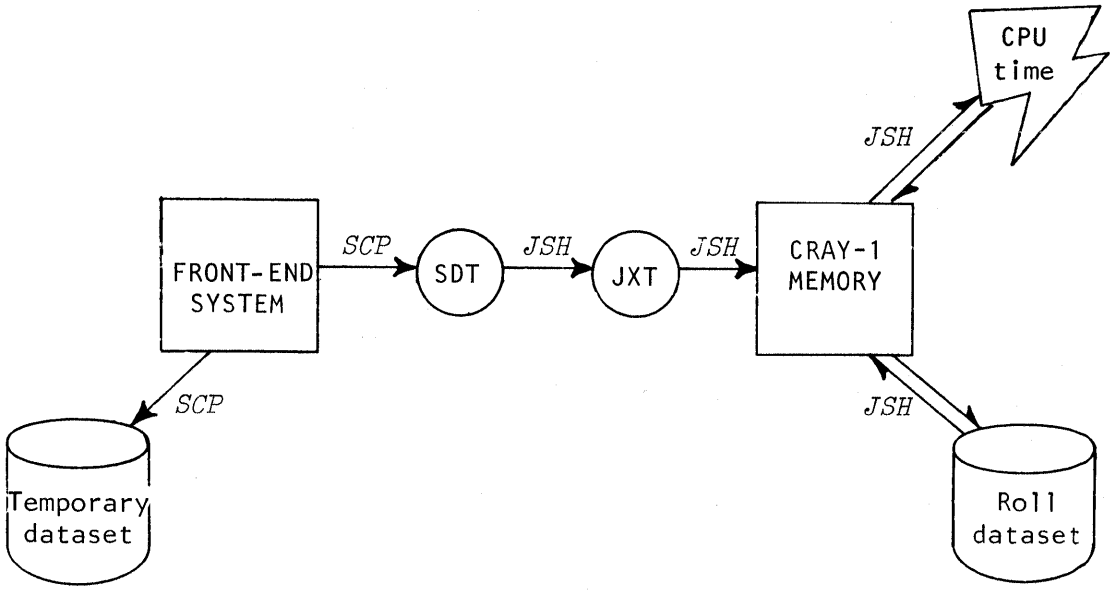


Figure 4.5-1. Job flow

After the Job Scheduler enters a job into the JXT, the job begins contending for memory with the other jobs in the JXT. Jobs with priority 0 are not initiated until the operator raises their priority. Memory allocation is based on priority, memory size, and the time limit specified on the JOB control statement. The Job Scheduler rolls jobs in and out of memory when necessary to ensure reasonably quick turnaround for all jobs.

Jobs that currently reside in memory contend with each other for CPU time. Their CPU priorities may depend on their user-assigned memory priorities, or they may be made independent of their user-assigned priority to maximize overall CPU usage. If I@AGECP, an installation parameter defined in STP, is 1, the Job Scheduler periodically adjusts the CPU priority of each job according to one of two alternate strategies (selected by I@CPPRI). When the currently executing job is disconnected from the CPU, JSH connects in its place whichever job among those ready to run that has the highest CPU priority. According to one strategy, that job is the job that has had the least CPU time in the past; using an alternate strategy, it is the job that has had the most I/O activity in the past.

4.5.2 SCHEDULING PHILOSOPHY

The typical time to service an I/O request is much longer than the average computation period; therefore, multiprogramming is essential to reduce the CPU idle time. A multiprogramming system schedules as many user programs into memory as is possible at any one time. These jobs may then share the CPU.

The two objectives of the scheduling mechanism are: (1) to make efficient use of the resources (memory and CPU time) and (2) to turn jobs around in a time that is consistently faster as the job's priority increases.

These two objectives are somewhat in conflict. However, scheduling occurs on two levels: at a high level where jobs contend for space in memory and at a low level where they contend for CPU time. A low-level scheduling strategy that pays little attention to user priorities is reasonable for the sake of efficiency, as long as it is balanced by a high-level strategy that allows the high priority jobs to initiate (transfer from the input queue, SDT, to the job mix, JXT) earlier and stay in memory longer. Time in memory is measured in terms of time actually used (excluding time spent waiting for resources). Even if the average stay in memory is made so long that most jobs finish before they can be rolled out, advantage is gained by early initiation.

From the viewpoint of system efficiency, each job should be held in memory until it terminates--or at least until a higher priority job displaces it. But then, a large job could monopolize system resources, preventing small or quick jobs from running at all. To avoid this possibility, a small or quick job is given a higher initial priority than requested on the JOB statement, and this initial priority (which governs the allocation of memory to the job) is periodically adjusted after the job joins the job mix. A job's memory priority rises while it is waiting for memory and falls while it is in memory and running. Jobs are swapped in and out of memory when the difference between their memory priorities is great enough. The installation is given several means of control over the rate at which swapping occurs.

Low-level scheduling: CPU priority

If I@CPMULT, an installation parameter defined in STP, is 0, each job begins with the maximum possible CPU priority; otherwise, each job begins with a CPU priority proportional to its initial memory priority. If I@AGECP is 0, CPU priorities do not age (the priorities do not increase or decrease) and memory priorities determine CPU assignments. When I@AGECP is nonzero, the CPU priorities for each job in memory are adjusted at the end of each scheduling interval in an attempt to apportion CPU time fairly between all jobs in memory. (The scheduling interval, I@JSHSI, is a tunable parameter and should have a value large enough so that system performance is not significantly degraded by priority recalculations.)

One of two alternative formulas (selected by the installation parameter I@CPPRI) is used in recomputing the CPU priority (CP) at the end of each scheduling interval (Δt).

If I@CPPRI is 0, the formula is:

$$\text{new } CP = \text{old } CP + \frac{CP_{max}\left(\frac{dbx}{\Delta t} \cdot \frac{dbt}{ndev}\right) - \text{old } CP}{2^{phw}}$$

where

dbx is the number of disk blocks transferred for the job in Δt

dbt is the minimum time needed to transfer one block (74000 cycles)

$ndev$ is the number of mass storage devices (EQNE in EQT)

$cput$ is the time spent by the job actually executing in the CPU in Δt

CP_{max} is the maximum value allowed for CP (2^n, where n= N@JXCP, if I@CPMULT=0; else CP_{max} is proportional to p_0, the initial memory priority as defined in the section on High-Level Scheduling.)

phw is the past-history weight, I@JSHPHW (an integer from 1 to 8 so that 2^{phw} = 2, 4, 8, 16, 32, 64, 128, or 256)(If phw = 1, the behavior during the current scheduling interval has a weight equal to all the past history; if phw = 0, the past history is ignored.)

If I@CPPRI is not 0, the formula is:

$$\text{new } CP = \text{old } CP + \frac{CP_{max}\left(1 - \frac{cput}{\Delta t}\right) - \text{old } CP}{2^{phw}}$$

where $cput$, CP_{max}, and phw have the same meanings as above.

Note that $\left(\frac{dbx}{\Delta t} \cdot \frac{dbt}{ndev}\right)$ and $\left(1 - \frac{cput}{\Delta t}\right)$ are dimensionless numbers.

The first formula expresses the ratio of disk utilization to execution time; the second expresses the fraction of time not used by the job during the scheduling interval. CP is initialized to CP_{max} for each job when it is initialized.

The CPU switching strategy is selected by another installation parameter (I@TSMIN). If I@TSMIN is greater than 0, the following formula is used to determine a maximum time slice for each job that depends on its initial priority:

$$time\ slice = \text{I@TSMIN} + (mct \cdot \text{I@TSCTM}) + (p_0 \cdot \text{I@TSMPM})$$

where

I@TSMIN	is the minimum time slice and the only one of the five terms that is never 0
mct	is the mean connect time for all jobs in the past scheduling interval (in milliseconds, rounded down)
I@TSCTM	is the connect time multiplier (any value from 0.0 to 10.0)
p_0	is the job's initial memory priority, calculated as described in the following section, High-Level Scheduling; it is treated as a scaled integer with values ranging from 0 to 15 and 15/16.
I@TSMPM	is the memory-priority multiplier (any value from 0.0 to 50.0)

If I@TSMIN is 0, the Job Scheduler gives every job the largest possible time slice (32,767 milliseconds), thus effectively disabling the time-slice mechanism for switching jobs. In this case, the Job Scheduler chooses a strategy of instant preemption. Whenever a new job is added to the list of waiting jobs (because of an I/O completion, the lifting of a suspension, or the addition of a new job to the JXT), the job that is currently executing is disconnected if its *CP* value is less than that of the new waiting job.

No matter which of the two strategies is used for disconnecting a job (time-slice versus high-priority preemption), JSH assigns the CPU to the highest priority job that is waiting and ready to run. If a job is suspended at its own request and no other job is ready to run, JSH merely suspends itself and control eventually passes to the idle loop.

Either of the two formulas for *CP* keeps strongly I/O-bound jobs at the top of the priority list and CPU-bound jobs at the bottom. (CPU-bound jobs get their fair share of CPU time on the average simply because they are more likely to be ready when the CPU becomes available.) The first formula gives more weight to jobs with efficient disk utilization. Together with the switching strategy that immediately reconnects a high-priority job that has just become ready, this formula allows a single job that is heavily I/O bound to perform as well as if it were running alone.

Such a job can be prevented from being swapped out in order to take the last step toward letting it run as if it were alone. For details, see section 4.5.3 on Tuning the System.

High-level scheduling: memory priority

Jobs in the JXT are transferred between disk and memory according to their memory priorities. These priorities are made to rise and fall so that jobs share memory fairly and orderly. (Jobs are spoken of here as disk resident whenever they are waiting for memory, although this is not literally true. An inactive job may remain in memory if memory is not in demand, although we treat the job as if it were rolled out; or the job may not reside anywhere at all if it has yet to be initiated.)

The memory priority of a job is approximated by $p = p_0 \pm a(1-e^{-rt})$

where

p_0 is the initial priority

a is the amplitude (the maximum difference between p and p_0)

r is the decay rate (the rate at which p approaches $p_0 \pm a$)

Initial memory priority - The initial priority, p_0, is determined by three values:

P Priority (0-15); the larger the number, the more important the job (and the more expensive it may be to the user). P is either the priority parameter from the JOB statement, the job priority assigned by the class, or the job priority entered by the operator, whichever occurs most recently.

M Memory parameter from the JOB statement (an octal number of 512-word blocks). The job's initial priority is increased by a value that is inversely proportional to its size.

T Time limit parameter from the JOB statement (in seconds). The job's initial priority is increased by a value that is inversely proportional to its time limit.

Calculation of a job's initial priority occurs as soon as COS becomes aware of the job; that is, just before the job is placed on the input queue. (The job's position on the input queue is determined by the calculated priority rather than by the P parameter alone.)

The initial priority, p_0 is determined as follows:

If P = 0, p_0 = 0; if P = 15, p_0 = 16-I@JSHAMP; otherwise,

$$p_0 = P + \left(\frac{flmax - M{\cdot}O'1000}{flmax - flmin} \right) \cdot flinc + \left(\frac{tquick + tlbias}{max(T, tquick) + tlbias} \right) \cdot tlinc$$

where p_0 cannot be less than I@JSHAMP or as large as 16-I@JSHAMP; any value of p_0 outside this range might cause p, the floating memory priority, to become negative or to overflow its field.

Several installation-dependent parameters appear in the above formula for p_0:

flmax is the maximum field length (I@JFLMAX).

flmin is the minimum field length (I@JFLMIN).

flinc is the maximum priority increase (I@FLINC) to be given for small size.

tquick is the time limit of a typical quick job.

tlbias is an arbitrary scaling factor chosen empirically (I@TLBIAS); determines the shape of the curve relating the priority increase in the T parameter.

tlinc is the priority increase (I@TLINC) to be given.

Floating memory priority: - In the formula for p, the sign (+) is taken as plus while the job is disk resident and minus while the job is memory resident. That is, while a job waits for memory, its priority p increases asymptotically toward $p_0 + a$; when the job is initiated or rolled in, p begins to decrease toward $p_0 - a$; and if the job is rolled out, p again begins increasing toward $p_0 + a$.

The increasing function is applied continuously while the job is rolled out, but the decreasing function is applied only to those (discontinuous) intervals of time during which the job is using the CPU or is suspended for I/O. Therefore, r has different values for the two separate functions, with the increasing function having the smaller value. The rise rate is called r_r and r_d is the decay rate.

When memory is available, it is assigned to a job on the basis of its current priority p; when jobs are of equal priority, the largest one that will fit is chosen.

When memory is not available, a job that is waiting for memory may preempt space from jobs in memory only if the waiting job's p value is greater than each of the p values of the jobs that it seeks to replace. Additionally, the waiting job must be greater by a certain deadband amount, db, in order to prevent an immediate counter-preemption once the jobs are swapped.

The deadband height, db, is a tunable parameter (I@JSHDB). The larger it is made, the less the sharing of memory between jobs of low and high priority and, therefore, the less thrashing.

The two r's are also tunable parameters (I@JSHRR and I@JSHDR). The smaller both rates are, the less thrashing that occurs. The ratio rd/rr should be greater than 1 and might be very large.

Finally, the amplitude, a, is a tunable parameter (I@JSHAMP) whose value determines the range of possible p_0's. For instance, with a=3, p_0 must be between 3 and 13; if a is 1 or less, p_0 may range from 1 to 15. JSH forces all values of p_0 into the proper range so that no interim value of p ever goes below 0 or rises as high as 16.

The actual algorithm does not compute exponential values. Instead, it approximates the exponential function by incrementing p every Δt by a Δp, calculated as follows:

$$\Delta p = (p_0 - p + a)\ r\Delta t$$

where $a<0$ and $r = r_d$ for jobs in memory, and

$a>0$ and $r = r_r$ for jobs not in memory.

The approximation improves with decreasing Δt, becoming exact when Δt is infinitesimally small. (Δp is too large by a factor that approaches $1+(r\Delta t)/2$ as $r\,\Delta t$ approaches 0. Any error introduced by too large a scheduling interval causes p to approach its asymptote at $p_0 \pm a$ more quickly.)

The rate of change for a job's memory priority should depend to some extent on the size of the job. A large job takes more time to be rolled in and out than does a small job, so the large job should remain longer in memory and on the disk. Job size causes memory priority to vary when the rate of change (r) is multiplied by

$$1 - jsw \left(\frac{\text{job size}}{flmax} \right)$$

where

jsw is an adjustable weighting factor (I@JSHJSW), and

$flmax$ is the maximum job size (I@JFLMAX).

Note that the value of this expression is always less than or equal to 1, and that it approaches 1 either as *jsw* approaches 0 or as the job size approaches the maximum job size.

Thus, the final version of the algorithm is

$$\Delta p = (p_0 + a - p)\left[1 - jsw\left(\frac{job\ size}{flmax}\right)\right] r\Delta t$$

For the purpose of determining when a job should be rolled out, the time actually used by a job is assumed to be a function of the time spent executing and the time spent waiting for I/O completion; that is, it excludes only the time spent waiting for assignment to the CPU. The underlying assumption is that no time is lost in the I/O request queue and no time lost waiting for the best opportunity to begin the seek operation. To the extent that this assumption is false, jobs are short-changed on memory time. Also penalized are those jobs that access disk cylinders far removed from the centers of disk activity. An offsetting advantage is that, when waiting time is low, memory priorities age nearly as fast in memory as on the disk.

Behavior of the high-level scheduling algorithm

Figures 4.5-2a through 4.5-2f show the variation of the memory priorities for several jobs running at the same time. In all of the examples given below, the following conditions hold:

- Not all the jobs can fit into memory together.
- The scheduling interval is very small, so that exponential curves may be used to describe the variations in each job's running priority.
- The rate of change of memory priority is independent of how much CPU time each job gets. (This is contrary to the facts, but it simplifies the graphs by giving each curve fragment the same decay constant.)
- The deadband interval, *db*, is 2.
- The amplitude, *a*, is 3.
- The solid lines represent memory priority for jobs in memory; the dashed lines represent jobs that are waiting for memory.
- All jobs are submitted at the same time and they each run long enough that their subpriorities are all 0.
- Competing jobs are of the same size (until figure 4.5-2e).
- The rates of rise and decay are independent of job size.

Figure 4.5-2a shows two jobs that are too large to share memory. Their initial priorities (from the job statements) are 3 and 5. The higher priority job runs first and consistently enjoys a longer stay in memory, because its priority has asymptotes at P=2 and 8 (as opposed to 0 and 6 for the other job).

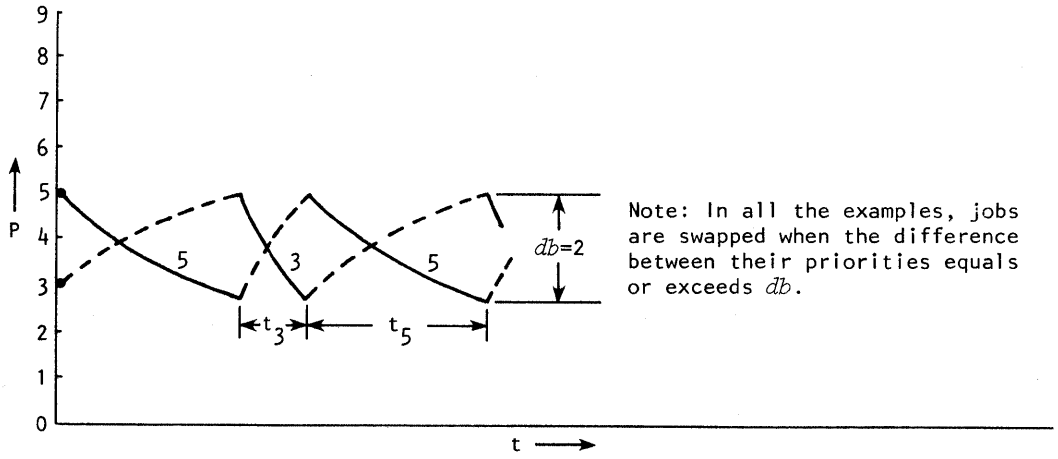


Figure 4.5-2a. Memory priority variation

Figure 4.5-2b shows three jobs, only two of which can share memory at any one time. Except for the presence of the priority-7 job, this graph is identical to figure 4.5-2a. The priority-7 job can never be forced out of memory because its priority never will be even as much as one unit below P=5, which is the maximum attainable by either of the other two jobs. (If its initial priority was 6 instead of 7, it still would never be forced out by the other two jobs.)

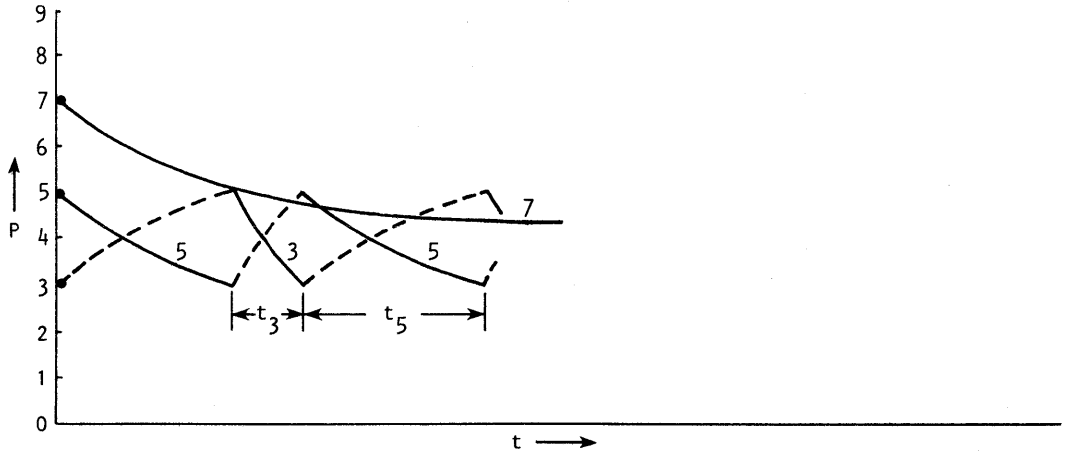


Figure 4.5-2b. Memory priority variation

Figure 4.5-2c shows three jobs which, like those in figure 4.5-2b, must share memory two at a time. In this case, though, the priorities are close enough together that they each get some time in memory. Although the pattern may not be a repeating one, each job's time in memory is consistent with its priority.

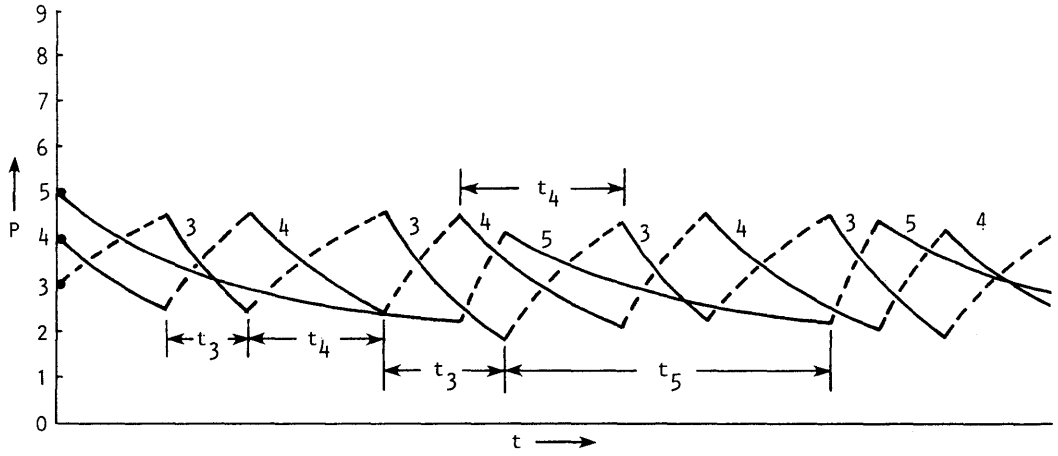


Figure 4.5-2c. Memory priority variation

Figure 4.5-2d shows a very large number of jobs, all of the same initial priority (4), and only one of which can fit in memory at any one time. The uppermost dashed line represents the current priority of all the jobs that have yet to be initiated. The jobs are initiated in the order they were submitted (or in JXT-entry order, if they were submitted simultaneously). Each job that runs gets slightly less time than the previous one, but this time in memory rapidly approaches a limiting value T as t advances. (It may be possible to specify the decay rate r_r indirectly by specifying this value of T, along with db and α.)

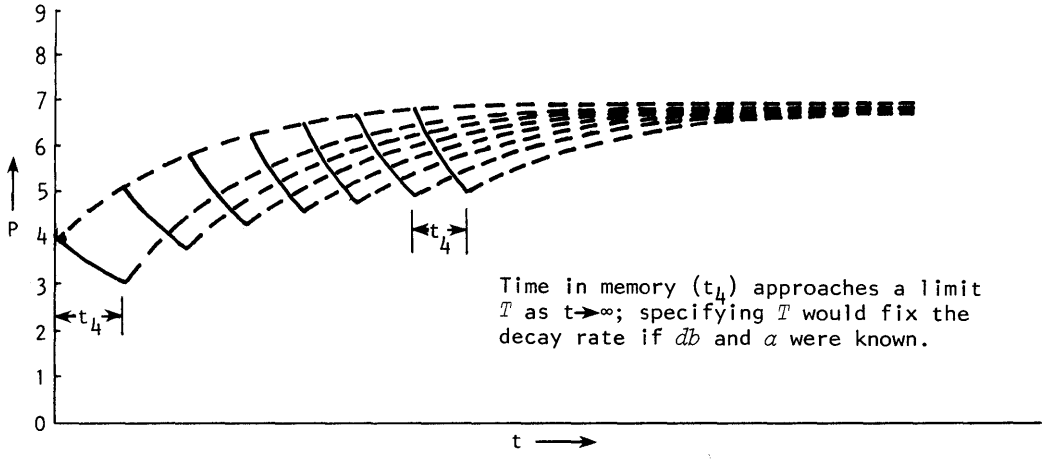


Figure 4.5-2d. Memory priority variation

In figure 4.5-2e is the first example in which the jobs are of different sizes. One high-priority job (6L) fills memory, while two other jobs (6S and 3) are each half as large as 6L. Note that the two possible states of memory (6L versus 6S and 3) each exist half the time, so that the low-priority job gets as much memory time as if it had the same priority as the other two jobs. This unfairness is remedied by making 3's priority decay faster, since the other two jobs completely determine scheduling.

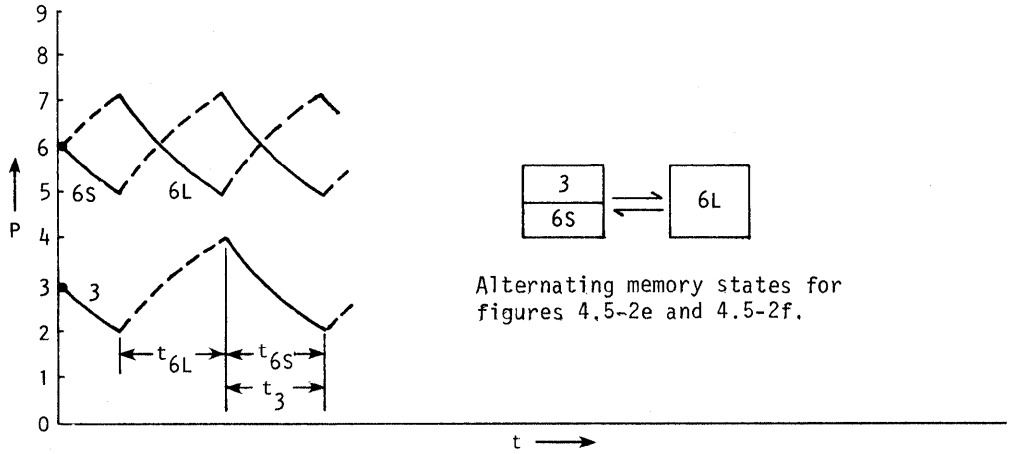


Figure 4.5-2e. Memory priority variation

Figure 4.5-2f is the same job mix as in figure 4.5-2e, but the rates of priority rise and decay are both inversely proportional to job size. The two possible states of memory each exist half the time, just as before. No gain is made in fairness--but making both rates depend on job size in this way improves the efficiency of the operating system because larger jobs take more time to roll in and out. The strength of the relationship between job size and rate of priority change depends on the value of I@JSHJSW.

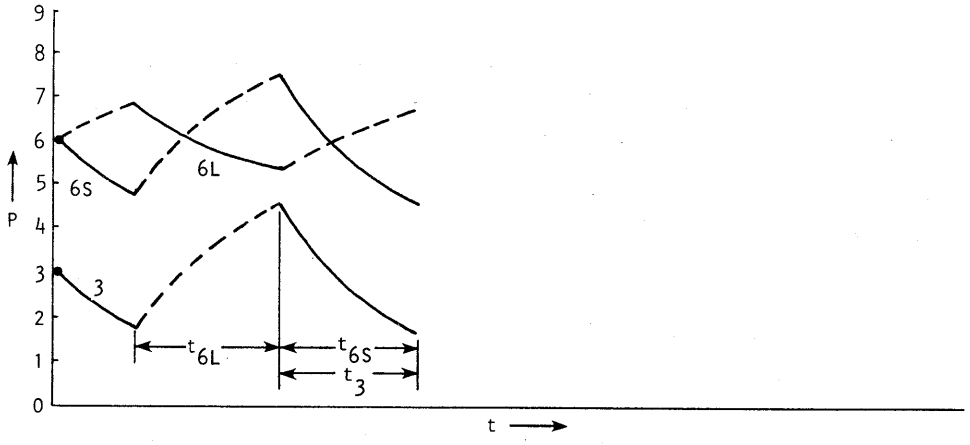


Figure 4.5-2f. Memory priority variation

Interrelationship between a, db, and user priorities

By choosing a value for the amplitude, a, an installation determines the range of permissible user priorities so that the difference between the minimum and maximum p_0 is 15-2a. Furthermore, any two jobs whose p_0s differ by as much as 2a-db can never force each other to roll out. Therefore, the installation should make sure that 15-2a is less than 2a-db by making db < 4a-15 if memory swapping is to affect all jobs.

The installation also selects a default priority (I@MEMPRI) when the P option is missing from the job statement. The priority should be in the middle of the permissible range; ideally, 7 or 8.

Job class scheduling

Job class scheduling consists of identifying sets of jobs having common characteristics so that they can receive special handling. Special handling occurs during job initiation (when a job receives a JXT) and/or job processing (when a job is given the CPU and/or memory).

A class receives a job initiation advantage when a number of JXTs are reserved for its exclusive use. A job belonging to that class is initiated as soon as it enters the input queue unless all of that class's reserved JXTs are allocated. If they are, a JXT may be drawn from the JXT pool. The JXT pool consists of all the non-reserved JXTs available to the system.

Each class may draw from the pool when all of its reserved JXTs are allocated. The total number of a class's allocated JXTs--reserved and in the pool--must not exceed the class's maximum JXT limit. A relative advantage is given to jobs in a class that has a large maximum JXT limit.

A class receives an additional job initiation advantage when it is assigned a high class rank in a CLASS directive, because JXTs are allocated from the pool in class rank order.

Job initiation is disabled when a class is turned OFF in a CLASS directive; that is, no new JXTs are allocated to a class that is OFF. Members of such a class remain in the input queue until the operator turns the class ON.

Job processing is affected when the class provides a class priority that overrides the job priority of all of its members.

Job class structure

Job class scheduling is defined by the job class structure in effect. See JCSDEF for a detailed description of a job class structure.

After a system Install, the following default job class structure is in effect:

```
SNAME,SN=DEFAULT.
CLASS,NAME=JOBSERR,RANK=1,CHAR=JSE,RES=0,MAX=63
CLASS,NAME=NORMAL,RANK=2,CHAR=ORPH,RES=0,MAX=63.
SLIMIT,LI=15.
```

When the default structure is in effect, all jobs are classified as normal.

The operator may use the LIMIT command to set the maximum number of JXTs to the structure's recommended value, 15. When the LIMIT comand sets the maximum number of active JXTs to less than the total reserved JXTs, the JXTs are reserved in class rank order until they are exhausted.

The utility JCSDEF can be used to define a job class structure from a series of card image directives. The operator can run JCSDEF to invoke a new class structure at any time without a system interruption.

A job class structure can be recovered or invoked at system Startup.

Job class structure monitoring

The effectiveness of a job class structure can be monitored by system log entries. When a structure is invoked or recovered, COS enters a message into the system log recording the structure's name. Additionally, the System Performance Monitor task logs scheduling status information at regular intervals.

Scheduling status information includes:

- Number of jobs in the system
- Number of active JXTs
- Number of available pool JXTs
- Number of active JXTs in each class
- Number of classes waiting for JXTs
- Number of jobs waiting for JXTs in each class
- RES, MAX and ON/OFF values of each class

COS enters a message into the system log when a job enters the input queue and again when it is given a JXT. The system utility, EXTRACT, can be used to generate job class reports.

4.5.3 TUNING THE SYSTEM

To reduce thrashing (excessive rolling in and out), several adjustments are possible. For quick results, an installation might increase the deadband, decrease the amplitude, and/or decrease the rise and decay rates, all of which affect high-level scheduling. A much slower change occurs if the operator reduces the maximum number of jobs in the mix. The most drastic approach is to reduce the maximum field length, thus preventing very large jobs from ever being able to run.

To reduce turnaround time for low-priority jobs (at the expense of system throughput), an installation might take the opposite course: increase the number of jobs in the mix, increase the rise and decay rates, increase the amplitude, and/or decrease the deadband.

Installation parameters

Installations can change several JSH parameters to suit their specific needs. (Parameters are made adjustable by using a memory word for each parameter, rather than equated values assembled into multiple locations, and by providing a means for changing them during deadstart and/or while the system is running.)

A list of installation parameters (including JSH parameters) and the values assembled into the released system are given in COS Operational Procedures Reference Manual, publication SM-0043.

Time slice

The time slice is the maximum amount of continuous execution time that a user job is allowed to have before it is disconnected from the CPU. This maximum normally affects only CPU-bound jobs because an I/O-bound job usually relinquishes the CPU before its time slice expires. The formula used by the Job Scheduler allows an installation to determine each job's time slice. If I@TSCTM=0, the time slice is directly proportional to its initial memory priority. If I@TSMPM=0, all jobs have the same time slice as determined by the mean connect time for all jobs in the past scheduling interval. A position anywhere between these two extremes is also possible.

The value of I@TSMIN is important. The smaller it is, the more sensitive the time slice computation is to variations in the other four terms and the more likely it is that the system will be subjected to excessively frequent exchanges when strongly I/O-bound jobs are sharing the CPU. For the latter reason, very small values of I@TSMIN are to be avoided. A value of 0, however, turns off the entire time-slice mechanism in favor of a strategy of instant preemption by the higher priority job, as described under Low-level scheduling.

Maximum number of jobs in mix (LIMIT command)

The maximum number of jobs in the mix (JXTMAX) is set by the operator using the station's LIMIT command. JXTMAX is usually smaller than the number of jobs waiting to be initiated but larger than the number of jobs that can be in memory together. If JXTMAX is greater than the actual number of jobs in the mix, jobs are taken from the input queue (where they are ordered by priority) and added to the mix. This parameter can be reduced to 0 by the operator to prevent any new jobs from being added as the operator cancels or lets complete all jobs in the mix. No job can be added to a full mix, regardless of its priority, until one of the running jobs terminates.

Deadband

Increasing the deadband decreases the importance of priority levels; as an extreme example, if the deadband height were 10 (with a = 3 and p_0 therefore ranging from 3 to 13), all jobs would inevitably run to completion without ever being forced to roll out.

Even with the deadband height as low as 2 (with a = 3), a job with p_0 = 7 would never roll out to make way for one with p_0 = 3; although, reducing the deadband height to 1.9 in this case would allow the job with p_0 = 7 to eventually be forced out by the job with p_0 = 3.

Rise and decay rates

Decreasing the rates at which memory priorities rise and fall reduces the roll-out activity, increases the advantage of jobs with higher initial priorities, and greatly increases the turnaround time of some low-priority jobs. Infinitely slow aging rates prevent any rolling out, just as if a large deadband were used.

Weighting factors

I@JSHPHW: The past-history weighting factor. When increased, this factor reduces the effect (on CPU sharing) of momentary fluctuations in the character of a job from CPU-bound to I/O-bound or vice versa.

I@JSHJSW: The job-size weighting factor. At 1.0, this factor allows close correspondence between job size and rollout frequency so that a job that is twice as large is rolled to disk about half as often. If this factor is set to 0.0, job size is not considered at all in computing the changes in memory priority.

4.5.4 MEMORY MANAGEMENT

The Job Scheduler allocates memory for job initiation and rolling in. It also handles requests made by the job that is currently running for changes in field length.

If a request for more memory cannot be satisfied by annexing part of an area contiguous to the current job, the job must be moved to another free area or rolled out.

The first-fit method is used for allocating memory to jobs that are to be moved or rolled in. That is, JSH allocates from the first free block of memory that is large enough, always beginning its search at the low end of memory to encourage large blocks of free memory to grow at the high end.

If a memory requirement cannot be satisfied immediately, but could have been satisfied if the available blocks were collected into one large block, a storage compaction mechanism is enabled and remains enabled as long as it can help to satisfy such requirements.

If compaction is enabled, it takes place after all allocations are made that can be made without compaction. During compaction, only jobs in wait state (not running and not in I/O suspension) can be moved, so the result is necessarily imperfect. More than one free area will very likely be available. After each cycle of compaction, JSH tries the unsatisfied requests again in the same order in which they originally occurred. When all requests are satisfied, compaction is disabled.

The compaction mechanism moves allocated blocks from the high end of memory to the low end, again (like the "first-fit" allocation strategy) to encourage the growth of large blocks of free memory at the high end.

Jobs to be moved are selected starting at the low end of memory. For each job that can be moved, the memory it occupies is temporarily freed and a search is made for a segment of the same size starting at the low end of memory. (The search is always successful because it can return the same segment or one that overlaps it.) The job is relocated to the found segment, if necessary, and compaction resumes with the next job that is movable.

The compaction mechanism is enabled based on a tally of the total amount of memory available, including blocks that will become available when previously requested rollouts (if any) are completed. If a job is promised memory (to be allocated when made available by compaction), this tally is reduced accordingly. It is only when an unsatisfied memory request is less than this total amount that compaction does any good; larger requests must wait for rollouts to occur (rollouts triggered by changes in memory priorities).

A table of memory segment descriptors (MST) is maintained by the Job Scheduler. Each 1-word entry in this table corresponds to a free or allocated memory segment. A newly freed segment adjacent to another free segment are immediately combined. All of the memory that is allocatable by JSH is initialized as a single free block.

For a description of how the Job Scheduler handles requests for changes in field length, see Allocate Request in section 4.5.7.

4.5.5 JOB STARTUP

When JSH sets up a job for its first shot at the CPU, it allocates the amount of memory specified by the M parameter on the JOB statement (but first adjusting this value if it exceeds I@JFLMAX or is smaller than I@JFLMIN). It then initializes the job's JXT entry and the Job Table Area (JTA). Most of the JTA is initially filled with zeros. The exceptions are the following fields:

Field	Description
JTJN	7-character job name (from JXT)
JTUSR	15-character user number (from SDT)
JTXP+1	Bits 18-35 (BA)
JTXP+2	LA and flags
JTJCB	JCB pointers
JTCMSG	Conditional message flags
JTEPJ	JSH request flag
JTSID	Source ID (from SDT)
JTDID	Destination ID (from SDT)
JTJXT	Pointer to JXT entry
JTTID	Terminal ID (from SDT)
JTDNT	3 DNTs (the first 2 for system use only), in the following order: $CS, $LOG, and $IN The DNTs for $CS and $IN refer to the same dataset.
JTCDP	DSP list for $CS
JTLDP	DSP list for $LOG

JSH makes the DNTs for $CS and $LOG unavailable to the user by storing a nonzero value in the low-order 8 bits of the first word in each DNT. The names $CS and $LOG, therefore, cannot be found by F$DNT and are used only as reference points in a dump. These two DNTs are always placed in the JTA in the order given above. The user DNTs, theoretically, may be in any order following the first two. (The DNTs for $CS and $IN refer to the same dataset.)

JSH sets up DSPs for $CS and $LOG in the JTA, initializing the dataset name (the same as the dataset name in the DNT) and the four I/O pointers--FIRST, IN, OUT, and LIMIT. $CS is opened for input and $LOG is opened for output.

The DNTs are initialized as shown in table 4.5-1. The DNT for the rolled dataset is in the JXT rather than in the JTA.

Table 4.5-1. DNT initialization

	Files initialized			
DNT Field	Roll Dataset	Control Statement File	LOG File	Standard Input Dataset
DNDN ASCII	'ROLLDNT'	'$CS'§	'$LOG'§	'$IN'
DNOC binary	'11'B	'10'B	'01'B	'00'B
DNP binary	1§§	-	-	-
DNDC ASCII	'SC'	'IN'	'PR'	'IN'
DNDAT address	-	copied from SDT	-	copied from SDT
DNPDS binary	0	1	0	1
DNACS octal	0007	0007	0007	0007
DNBFZ decimal	-	1	1	4§§§
DNDSP address	-	yes	yes	no

§ The dataset names $CS and $LOG are unavailable to the user because the low-order byte of the word in which each name is stored is set to a nonzero value. The roll dataset's DNT is unavailable because it is not in the JTA.

§§ The DNT (processing direction) flag for the roll dataset is toggled according to the expected direction of the next I/O transfer.

§§§ The buffer size for $IN is an installation-dependent parameter. The numbers given for DNBFZ are multiples of 512-word blocks.

4.5.6 JOB STATUS AND STATE CHANGES

The 23-bit status field (JXSTAT) in each job's JXT entry is described in table 4.5-2. The bits labeled Q, R, X, I, U, L, S, O, and M are determine the job's state; the other bits modify the job's state.

If all of bits 3 through 22 are 0, the job is said to be waiting to be connected to the CPU (state W).

Table 4.5-2. Status bit assignments

Bit position in JXSTAT	Bit name	Corresponding job state	Interpretation (when bit is set)
0	K	*S*	Keep this job in memory; do not roll it out.
1	A	*any*	Abort pending; reason given in JXEPC
2	H	*O*	Holding operator or shutdown suspension until RN is set
3	O	*O*	Suspended (indefinitely) by operator
4	S	*S*	Suspended (possibly by system)
5	T	*S*	Suspended until a given time elapse
6	E	*S*	Suspended until a given event occurs
7	M	*M*	Memory allocation is pending.
8	Q	*Q*	Queued up; waiting to be initiated
9	R	*R*	Rolled out. The M bit may also be set.
10	X	*X*	Executing
11	I	*I*	Dormant pending recall on I/O completion
12	C	*any*	Rerun request in process
13	D	*any*	Delete request in progress
14	U	*U*	Unloading from memory to roll file

Table 4.5-2. Status bit assignments (continued)

Bit position in JXSTAT	Bit name	Corresponding job state	Interpretation (when bit is set)
15	L	*L*	Loading into a new memory area
16	P	*U* or *L*	Unload or load initiation is pending
17	Y	*M*, *Q* or *R*	Waiting for memory liberation
18	Z	*M*, *Q* or *R*	Waiting for memory compaction
19	B	*S*	Suspended (indefinitely) by recovery
20	V	*I*	Waiting on INDEX write completion
21	F	*I*	Waiting on rollfile write completion
22	N	*M*, *Q* or *R*	Not in memory

Figure 4.5-3 and table 4.5-3 illustrate some of the transitions that normally occur between job states.

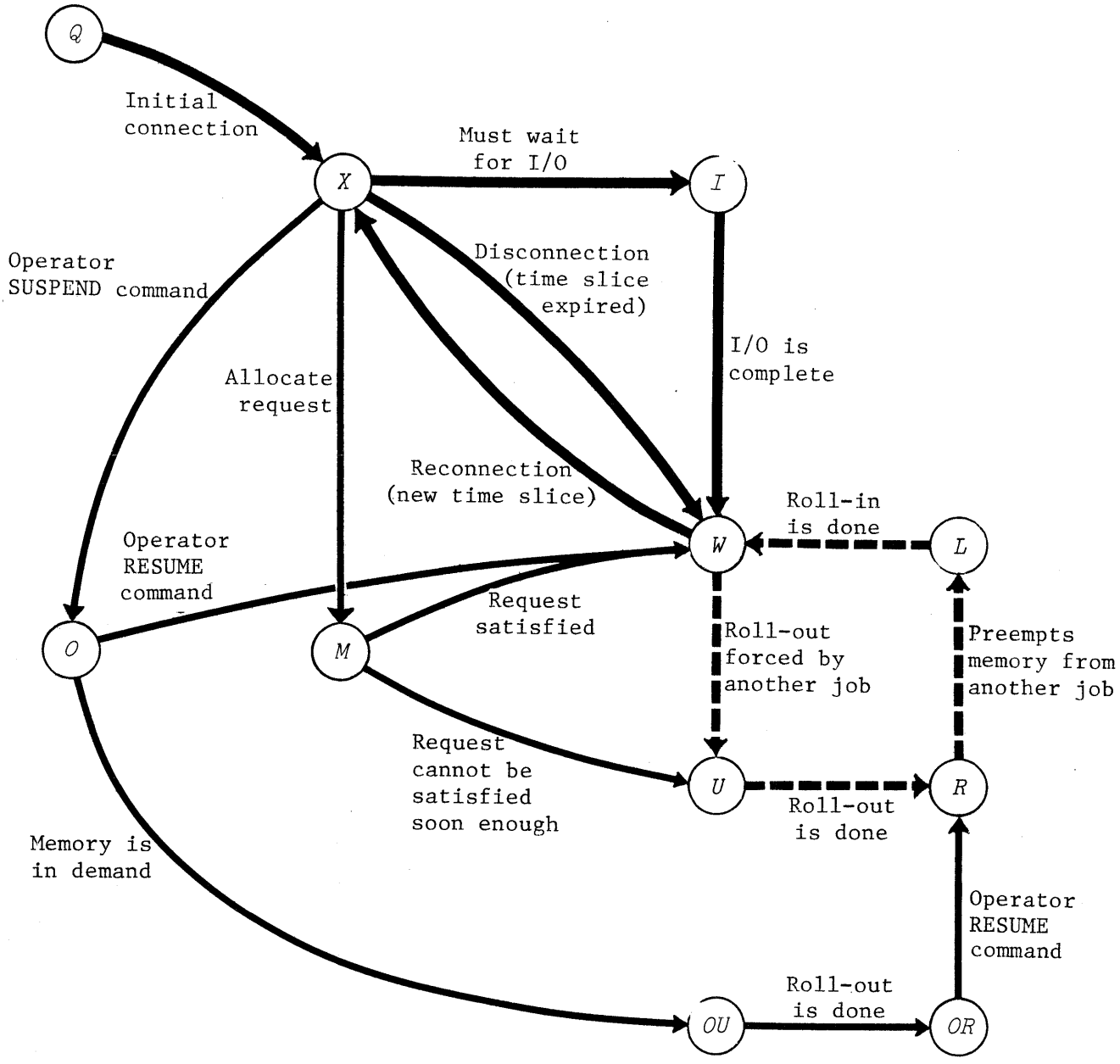


Figure 4.5-3. Normal transitions between job states

Table 4.5-3. State-change sequences

Sequence	Explanation
QN → (W) → X	A job is started up for the first time.
X → I → W	A job requests recall on I/O completion.
X → W	A job's CPU time-slice expires.
W → X	A job is given another time slice.
W → UN → RN	A job is rolled out.
RN → LN → W	A job is rolled in.
X → MN → W	A job's memory allocation is adjusted.
X → MN → UN → RN	Not enough memory is immediately available.
X → UN → RN → NRO	A job is suspended by the operator.
ORN → LN → W	A job is resumed.

State changes involved in CPU swapping

Figure 4.5-3 shows most of the state changes that can occur for any particular job. Those state changes shown with broad solid lines are the basic changes that all jobs must undergo. If all the jobs in the JXT can fit into memory at the same time, these state changes are also the only state changes that the jobs undergo as long as they make no memory requests and open no auxiliary datasets.

QN → X A job that has been queued in the JXT (Job Execution Table) waiting for sufficient memory to become available is given its first CPU time slice. (The job actually exists momentarily in the *W* state before it begins executing; but because the new job's CPU priority is initialized to the highest possible value, the transition to state *X* is immediate.)

X → I The currently executing job becomes dormant, either by requesting suspension pending the completion of a particular I/O transfer. The CPU becomes available for use by another job.

I → W The I/O transfer for which suspension was requested is now complete. The job joins others that may be waiting for CPU time.

X → W The executing job's time slice expires. Unless it is the only job that is running, it is disconnected from the CPU and joins any other jobs that may be waiting.

W → X The CPU has just become available. JSH selects the waiting job that has the highest CPU priority (excluding the job that was just disconnected) and connects it to the CPU.

State changes involved in memory swapping

In figure 4.5-3, the paths involved in rolling jobs in and out are shown as dashed lines.

Any job that is in state *W*, waiting for more CPU time, is liable to be rolled out if it occupies space that can be used by another job with sufficiently high memory priority. In essence, jobs having states *W* and *R* exchange places, but they must each pass first through an intermediate state (*U* or *L*).

W → UN A roll-out I/O request is initiated for a waiting job in order to make memory available.

UN → RN The roll-out I/O request is complete; the job's memory is released.

RN → LN Memory is allocated for the job and a roll-in I/O request is initiated.

LN → W The roll-in I/O request is complete; the job begins contending for CPU time.

State changes involved in job suspension and reactivation

In figure 4.5-3, the suspended states are connected to the rest with narrow solid lines.

A job may be momentarily suspended when it makes an allocation request (J$ALLOC). Shortly after it suspends the job, JSH checks for active I/O requests; if there are no I/O requests and the allocation request can be satisfied, the suspension is lifted. The suspension is kept in force if the job must be moved but there is no space for it right away; that is, the M bit remains set and the job is then liable to be rolled out if memory is in demand.

Suspension can also occur as a result of an explicit user request to suspend processing until a given event has occurred (J$AWAIT) or until a given time has expired (J$DELAY).

Finally, a job can be suspended and resumed by the operator to prevent the job from using any system resources.

X→ M — The currently executing job makes an allocation request and is considered dormant until the request can be satisfied. If the request involves the movement of I/O buffers or tables, it cannot be satisfied until all the job's I/O is done. If, after all I/O is done, the request still cannot be satisfied, the job may be rolled out.

MN→ W — The allocation request was satisfied before the job could be rolled out. The job joins any other jobs that may be waiting for CPU time.

MN→ UN — The job has been rolled out because memory is in demand. More space was required to satisfy the allocation request than could be obtained merely by reallocating memory.

X→ S — The currently executing job makes a suspension request (J$DELAY or J$AWAIT). The job is disconnected and may be rolled out if memory is in demand.

W→ O — A job in memory is suspended by operator intervention. A job that is operator-suspended (O) is always rolled out when active I/O finishes.

S→ SUN — A roll-out I/O request to make memory available is inititated for a suspended job.

SUN → SRN — The roll-out I/O request is complete; the job's memory is released.

ORN→ RN — A job that was suspended by the operator (and subsequently rolled-out is reactivated by the operator. The job is rolled back in when memory is available.

SRN→ RN — A job that was suspended by the system (and subsequently rolled-out) is reactivated because a given time elapsed or a particular event occurred. The job is rolled back in when memory is available.

S→ W — A job that was suspended by the system is reactivated while still in memory.

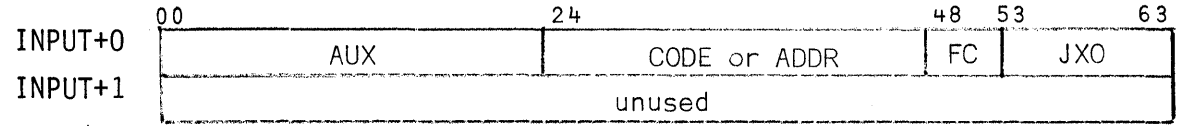
4.5.7 JSH INTERFACE WITH OTHER TASKS

The job scheduler task is created with all other system tasks by the startup procedure. It can then be called by any other task through the sequence of instructions shown later in this subsection.

JSH always replies to each request by setting the appropriate output registers and readying the requesting task. However, the reply is not always immediate. For some requests, JSH must wait until memory is available or until an I/O transfer is complete before it replies, so that the requesting task may proceed correctly.

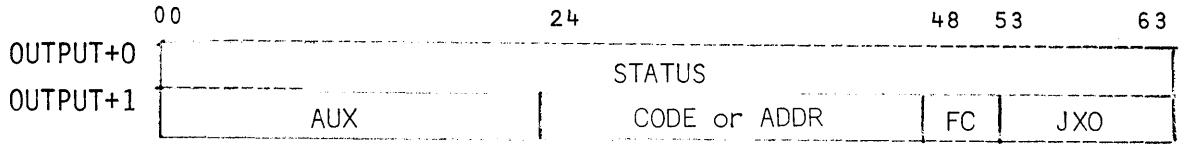
To enable the requesting task to determine what a delayed reply means, JSH echoes the entire contents of the first input register as the second output register. This word contains the JSH function code, the JXT ordinal identifying the job, and additional information as supplied by the requesting task. A status indicator is returned in the first output register.

Input register format:



Field	Word	Bits	Description
AUX	INPUT+0	0-23	Auxiliary information; unused by JSH. (Any value the caller places in INPUT+0 is returned verbatim in OUTPUT+1.)
CODE	INPUT+0	24-47	(J$ABORT request only) Abort code; use equated labels of the form A$*xxxxxx*, where *xxxxxx* = DROP, KILL, RERUN, or other predefined abort code. These abort codes are also used in the J$UROLL request.
ADDR	INPUT+0	24-47	Word address relative to the beginning of STP of an additional word or list of words if needed to fully specify the call. Refer to the individual function descriptions for more detail.
FC	INPUT+0	48-52	Function code; use equated labels of the form J$*xxxxx*, selected from table 4.5-4.
JXO	INPUT+0	53-63	JXT ordinal for the job in question. It can assume a value from 1 to I@JXTSIZ. It cannot be 0.

Output register format:



Field	Word	Bits	Description
STATUS	OUTPUT+0	0-63	Status of requested function =0 Requested function completely accomplished ≠0 Error or system is unable to fulfill request completely
AUX	OUTPUT+1	0-23	Auxiliary information; unused by JSH. (Any value the caller places in INPUT+0 is returned verbatim in OUTPUT+1.)
CODE	OUTPUT+1	24-47	(J$ABORT request only) Abort code; use equated labels of the form A$*xxxxx*, where *xxxxx* = DROP, KILL, RERUN, or other predefined abort code. These abort codes are also used in the J$UROLL request.
ADDR	OUTPUT+1	24-47	Word address relative to the beginning of STP of an additional word or list of words if needed to fully specify the call. Refer to the individual function descriptions for more detail.
FC	OUTPUT+1	48-52	Function code; use equated labels of the form J$*xxxxx*, selected from table 4.5-4.
JXO	OUTPUT+1	53-63	JXT ordinal for the job in question. It can assume a value from 1 to I@JXTSIZ. It cannot be 0.

Calling sequence

JSH can be invoked from any other task by calling either TSKREQ or PUTREQ with the following instruction sequence:

Location	Result	Operand
	A2	JSHID,0
	S1	function code (already shifted)
	S2	job's ordinal in JXT
	S1	S1!S2
[	S2	address if any]
|	S2	S2<D'16 |
|	S1	S1!S2 |
[	S2	auxiliary information if any]
|	S2	S2<D'40 |
|	S1	S1!S2 |
[	R	TSKREQ or PUTREQ

The TSKREQ subroutine enforces synchronous task behavior; that is, it does not return to its caller until JSH has replied. PUTREQ returns as soon as it has stored the input registers, thus permitting asynchronous task execution.

The requests that tasks can make to JSH are described on the following pages in the order listed in table 4.5-4.

Table 4.5-4. JSH functions

Function Code	Input Parameters	Function
-- no input required --		Fills up JXT with jobs from SDT
J$ABORT	JXO,CODE	Aborts a job
J$DELETE	JXO	Releases all space allocated to a job
J$RERUN	JXO	Same as J$DELETE but places a job back in the input queue
J$ALLOC	JXO,ADDR	Allocates or releases memory for a job
J$IOSUSP	JXO	Suspends a job until an I/O request is done

Table 4.5-4. JSH functions (continued)

Function Code	Input Parameters	Function
J$IODONE	JXO	Resumes an I/O suspended job
J$DELAY	JXO,ADDR	Suspends a job for a given time
J$AWAIT	JXO,ADDR	Suspends a job until a given event occurs
J$SUSP	JXO	Suspends a job momentarily; system initiated
J$SUSPK	JXO	Same as J$SUSP but keeps the job in memory
J$REMK	JXO	Lifts the keep-in-memory restriction
J$RESUME	JXO	Ends momentary suspension
J$STOP	JXO	Suspends a job indefinitely; operator action
J$STPALL		Suspends all jobs indefinitely; operator action
J$START	JXO	Ends an indefinite suspension for a job; operator action
J$STRALL		Ends an indefinite suspension for all jobs; operator action
J$CLEAR	JXO	Forces the end of a job's suspension
J$INDEX	JXO	Marks the job irrecoverable
J$SHTDWN		Idles down job activity in preparation for a system interruption
J$RCVR		Lifts the suspension from jobs suspended by a J$SHTDWN or system interruption
J$INVOKE	JXO,ADDR	Invokes a job class structure
J$UROLL	JXO,CODE	Rolls a job; user requested to protect against system interruption

Initialize request

FUNCTION: Transfers as many jobs as possible from the input queue to the executing queue

FUNCTION CODE: None

ENTRY: None

EXIT: None

DESCRIPTION: If there are any available JXT entries, jobs are selected from the input queue (part of the SDT) and entered into the JXT until the JXT is full or nor more jobs can be initiated in the appropriate class. Any job entered into the JXT is also transferred from the input queue to the executing queue (still in the SDT); its SDT entry is not deleted until the job terminates.

The number of JXT entries currently in use is stored in JXTPOP, while the current maximum is in JXTMAX. Because JXTMAX can be reduced by the operator while the system is running, live JXT entries may be scattered throughout the table. They are chained together in order of both memory priority and CPU priority. Empty JXT entries are similarily chained together.

The startup program readies JSH for this request after setting up the input queue in the SDT; subsequently, the Station Call Processor (SCP) readies JSH again whenever it adds a new job to the input queue. The jobs are expected to be queued in order of decreasing priority and, within priority, in order of decreasing age.

Whenever a job terminates, JSH checks the input queue for any waiting jobs previously bypassed because of lack of room in the JXT.

Note that a job does not begin execution until its memory priority (which begins to rise from the value given on the JOB statement as soon as the job gets into the JXT) is higher than the memory priority of whatever jobs it must displace in order to run. (For its initial memory allocation, a job needs only enough room for a copy of the Control Statement Processor, CSP.)

Abort request

FUNCTION: Aborts a job

FUNCTION CODE: J$ABORT (In actual coding, use J$ABORT+1S20*A$*xxxxx*, where *xxxxx* = DROP, KILL, RERUN, or other predefined abort code.)

ENTRY: FC, JXO, and CODE are required. CODE occupies the same position as ADDR does in other requests.

EXIT: The first output register is a status word normally set to 0. The second output register is simply a copy of the first input register.

The output registers are set and the caller is readied immediately after the request is received.

DESCRIPTION: JSH disconnects the job from the CPU and sets a flag (JTEPJ) in the Exchange Processor's request word in the JTA so that, when the job is reconnected, the Exchange Processor task aborts the user. The abort code (JXECP) is stored in the JXT to be picked up by the Exchange Processor.

Delete request

FUNCTION: Releases all memory belonging to a given job

FUNCTION CODE: J$DELETE

ENTRY: FC and JXO are required.

EXIT: The first output register is a status word normally set to 0. The second output register is simply a copy of the first input register.

The output registers are set and the caller is readied immediately after the request is received.

DESCRIPTION: This call is the last action in the terminating of a job. JSH disconnects the job from the CPU if necessary and, as soon as any pending I/O is complete, frees all memory assigned to the job (the user area, the JXT, and the empty SDT entry).

Rerun request

FUNCTION: Releases all memory belonging to a given job except its SDT entry and moves the SDT from the executing queue back to the input queue so that it can be reinitiated.

FUNCTION CODE: J$RERUN

ENTRY: FC and JXO are required.

EXIT: The first output register is a status word normally set to 0. The second output register is simply a copy of the first input register.

The output registers are set and the caller is readied immediately after the request is received.

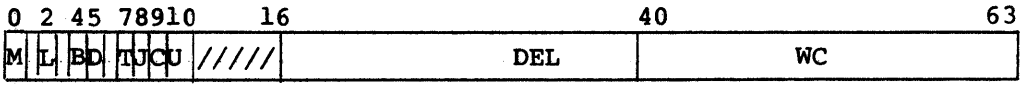
DESCRIPTION: This call is the last action taken in the terminating of a job that is to be rerun. JSH disconnects the job from the CPU if necessary and as soon as any pending I/O is complete, frees the job's user area and its JXT entry. The SDT entry for the job is moved from the execute queue to the input queue.

Allocate request

FUNCTION: Determines or changes a job's memory allocation

FUNCTION CODE: J$ALLOC

ENTRY: FC, JXO, and ADDR are required. ADDR is the STP-relative address of a memory request word having the following format:



Field	Bits	Description
M	0	Maximum memory flag. If M is set by the caller, JSH returns in WC the maximum allowable amount of memory (in words) excluding the JTA. No memory is allocated.
L	2	Limit flag. JSH sets this flag when the job has received the maximum allowable amount of memory.

B	4	Buffer flag. If set by the caller, JSH is to add WC words to the beginning of the I/O buffer area. The B flag should be set if the caller wants the D flag to be tested. WC should be 0 if no buffer space is needed.
D	5	DSP flag. If set by the caller, JSH is to allocate a fixed number of words (I@DSPINC) at the end of the DSP area. D is ignored if B is 0.
T	7	Total flag. WC represents the total memory requested (excluding the JTA) rather than an increment or decrement, and DEL is ignored. (HLM is not changed if the T flag is set; HLM is adjusted only if the B and T flags are both 0.) If the T flag is set, the B and D flags are ignored.
J	8	JTA expansion flag. WC represents the number of words to be added to the JTA.
C	9	Lock flag. If C is set by the caller, the user's I/O area is locked (the user does not have access to his buffers and tables). JSH changes the user's LA to (JCDSP).
U	10	Unlock flag. If U is set by the caller, the user has access to his buffers and tables. JSH changes the user's LA to (JCFL). The user's I/O area status remains unlocked until the lock flag is set by the user.
DEL	16-39	Deletion pointer. If the caller wants an increase in memory, DEL must be 0. If the caller wants a decrease in memory, DEL must contain the address relative to the user's BA of the beginning of the area to be deleted and that address must agree with the B flag. That is, if the B flag is not set, the area to be deleted must lie wholly within the program area, from L@JCB to (HLM); and if the B flag is set, the area to be deleted must lie entirely within the buffer area.
WC	40-63	Word count. Here, unless the total flag is set, the caller must supply an absolute number of words to be added to or deleted from the user area. If the total flag is set, WC must contain the total field length desired by the caller. If the entire memory request word equals 0, no action is taken other than to return the user's

field length as described under EXIT, below. If M is set, no action is taken other than to return the maximum allowable amount of memory as described under EXIT.

NOTE

The caller has no control over where the WC words are to be added, except to say whether they are to be added at the beginning of the I/O buffer area (B=1) or at the end of the program area (B=0). However, the caller has more control over deletion; DEL may point to any location within the program or buffer areas. On a deletion call, if the area to be deleted does not lie entirely within the appropriate area (see DEL, above), an error status is returned and no deletion is performed.

EXIT:

In the memory request word, L may be set by JSH as described above. When the entire memory request word equals 0, JSH sets WC to the current total number of words in the user's field length (which does not include the JTA, but does include the I/O buffers and tables). When M is set, JSH sets WC to the maximum allowable amount of memory, including the I/O buffers and tables excluding the JTA.

The first output register is a status word which is set to 0 unless the job is to be aborted as a result of the request. The second output register is simply a copy of the first input register.

The output registers are set and the caller is readied immediately after the request is received; if the job must be aborted, that is accomplished by returning an appropriate error status to the calling task.

DESCRIPTION:

JSH begins processing the request by placing the job in state *M* and disconnecting the job from the CPU.

A J$ALLOC request may be made for any job, whether currently executing or not. If the B bit in the memory request word is 0, the request applies to the program area rather than to the I/O areas following the program area. The HLM pointer is adjusted but the other pointers are not necessarily changed.

If the size of the user area must be expanded or reduced, nothing can be done until all the job's outstanding I/O requests are complete. In the meantime, the job is dormant.

When the job has no I/O activity, the user area can be contracted, expanded, moved, or rolled out. A request for less space is always honored immediately. A request for more space is honored immediately if there is a large enough area for it. If other jobs would have to be moved but they have I/O in progress, the job remains in state *M* until either a sufficiently large continuous area becomes available or the job is rolled out.

If a job requests more memory than the installation allows for each job, the job is aborted.

Besides the total length of the user area, the JCB has five pointers (HLM, LFT, DSP, BUF, FL) that must be maintained by J$ALLOC when it either expands or compresses the user area. As was mentioned at the end of the INPUT section, compression (deletion) must always be contained completely within the areas that are bounded by these pointers. Furthermore, any expansion or compression always implies the adjustment of at least one of these pointers.

If any expansion of the user area is performed, the additional memory is cleared before JSH reconnects the job to the CPU.

I/O-suspend request

FUNCTION: Suspends execution of a job until an I/O-recall request is made for the same job

FUNCTION CODE: J$IOSUSP

ENTRY: FC and JXO are required.

EXIT: The first output register is a status word normally set to 0. The second output register is simply a copy of the first input register.

The output registers are set and the caller is readied immediately after the request is received.

DESCRIPTION: JSH disconnects the job from the CPU if necessary, and sets the I bit in its status field, changing the job's state to *D* (dormant).

I/O-recall request

FUNCTION: Reactivates an I/O-suspended job

FUNCTION CODE: J$IODONE

ENTRY: FC and JXO are required.

EXIT: The first output register is a status word normally set to 0. The second output register is simply a copy of the first input register.

The output registers are set and the caller is readied immediately after the request is received.

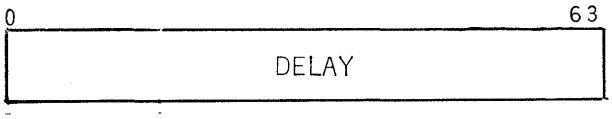
DESCRIPTION: JSH clears the I bit in the job's status field, changing the job's status from *D* to *W* unless the job is suspended for other reasons as well.

Delay request

FUNCTION: Suspends execution of a job for a given number of milliseconds

FUNCTION CODE: J$DELAY

ENTRY: FC, JXO, and ADDR are required. ADDR is the STP-relative address of a delay request word having the following format:



where DELAY is the number of milliseconds for which the job is to be delayed. The maximum delay is 879,609 seconds, which is more than 10 days.

EXIT: The first output register is a status word normally set to 0. The second output register is simply a copy of the first input register.

The output registers are set and the caller is readied immediately after the request is received.

DESCRIPTION: JSH disconnects the job from the CPU if necessary, sets the T bit in the status field, and stores the wake-up time in the JXDLY field in the job's JXT entry.

If the job's T bit is set, the JXDLY field is compared to the real-time clock every time JSH is readied. The suspension is lifted when the wake-up time has been reached.

A job may be rolled out while it is suspended. It enters into the normal memory swapping activity after it is reactivated.

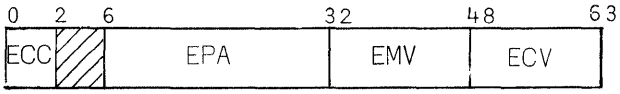
Lifting the suspension involves clearing both the T and E bits (the E bit is set by J$AWAIT). However, the job remains suspended if either the S (J$SUSP), B, or O bit is set. Only a J$RESUME request can clear the S bit. Only a J$START or a J$STRALL request can clear the O bit. Only A J$RCVR, a J$START, or a J$STRALL can clear the B bit.

Await request

FUNCTION: Suspends execution of a job until a given event occurs

FUNCTION CODE: J$AWAIT

ENTRY: FC, JXO, and ADDR are required. ADDR is the STP-relative address of an event word having the following format:



where

ECC is the condition code (0-3)

EPA is the STP-relative address of a parcel to be tested periodically

EMV is the mask value to be applied in the test

ECV is the comparison value to be used in the test

The first output register is a status word normally set to 0. The second output register is simply a copy of the first input register.

The output registers are set and the caller is readied immediately after the request is received.

DESCRIPTION: JSH disconnects the job from the CPU if necessary, sets the E bit in the status field, and stores the event word in the job's JXT entry.

If the job's E bit is set, the parcel at the given address is periodically ANDed with the mask and then subtracted from the comparison value. The event is said to have occurred when the result matches the condition code.

Condition code	Matching result
0	zero
1	nonzero
2	positive
3	negative

As stated in the description of J$DELAY, a suspended job is liable to be rolled out. When the event occurs, both the E and T bits are reset and the suspension is lifted unless the S, B, or O bit is set. (Refer to the description of J$DELAY).

Suspend request

FUNCTION: Suspends execution of a job momentarily

FUNCTION CODE: J$SUSP or J$SUSPK

ENTRY: FC and JXO are required.

EXIT: The first output register is a status word normally set to 0. The second output register is simply a copy of the first input register.

The output registers are set and the caller is readied immediately after the request is received.

DESCRIPTION: JSH disconnects the job from the CPU if necessary and sets the S bit in the status field. The K bit is set for a J$SUSPK request but left unchanged for a J$SUSP request. The T and E bits are left unchanged; if they are set, they can be reset only by the occurrence of an event or the elapsing of a delay time.

The K (keep) bit, if set, prevents the job from being rolled out while it is suspended. A J$RESUME request clears the S and K bits allowing the job to participate in normal rollout activity.

Remove K request

FUNCTION: Lifts the keep-in-memory (K) restriction

FUNCTION CODE: J$REMK

ENTRY: FC and JXO are required.

EXIT: The first output register is a status word normally set to 0. The second output register is a copy of the first input register. The output registers are set and the caller is readied immediately after the request is received.

DESCRIPTION: The K status bit is cleared. The job may begin to participate in normal memory swapping activity, unless it is suspended.

NOTE

The K bit must be set when the J$REMK request is made.

Resume request

FUNCTION: Lifts a suspension imposed on a job by J$SUSP or J$SUSPK

FUNCTION CODE: J$RESUME

ENTRY: FC and JXO are required.

EXIT: The first output register is a status word normally set to 0. The second output register is simply a copy of the first input register.

The output registers are set and the caller is readied immediately after the request is received.

DESCRIPTION: This call must be made to restart a job that has previously been suspended by a call to J$SUSP or J$SUSPK, but it has no immediate effect if the job is still under suspension for another reason.

JSH clears the S and K bits in the job's status field. If or when the H, O, T, and E bits are also clear, the job is no longer suspended and may begin to participate in the normal memory swapping activity.

Stop request

FUNCTION: Suspends execution of a job indefinitely; operator action

FUNCTION CODE: J$STOP

ENTRY: FC and JXO are required.

EXIT: The first output register is a status word normally set to 0. The second output register is simply a copy of the first input register. The output registers are set and the caller is readied immediately after the request is received.

DESCRIPTION: The job's O (operator suspended) and H (holding) bits are set. The job will be rolled out as soon as possible. When the job is rolled out and its memory is released, the H bit is cleared.

Stop all request

FUNCTION: Suspends processing of all jobs in the JXT; rolls them out, and releases their memory

FUNCTION CODE: J$STPALL

ENTRY: FC is required.

EXIT: The first output register is a status word normally set to 0. The second output register is simply a copy of the first input register.

The output registers are set and the caller is readied immediately after the request is received.

DESCRIPTION: The JXT limit (JXTMAX) is set to 0. All jobs are suspended. (Refer to the section on J$STOP.)

Start request

FUNCTION: Lifts the indefinite suspension from a job suspended by J$STOP, J$STPALL, J$SHTDWN or a system interruption

FUNCTION CODE: J$START

ENTRY: FC and JXO are required.

EXIT: The first output register is a status word normally set to 0. The second output register is simply a copy of the first input register.

The output registers are set and the caller is readied immediately after the request is received.

DESCRIPTION: This call must be made to restart a job that has been suspended by J$STOP or J$STPALL. It may also be used to lift the suspension imposed on a job by J$SHTDWN or a system recovery. Both the O (operator suspended) and the B (suspended by recovery) bits are cleared. This has no immediate effect if the jobs are still under suspension for another reason.

Start all request

FUNCTION: Lifts the indefinite suspension from all jobs that were suspended by J$STOP, J$STPALL, J$SHTDWN or a system interruption

FUNCTION CODE: J$STRALL

ENTRY: FC is required.

AEXIT: The first output register is a status word normally set to 0. The second output register is simply a copy of the first input register. The output registers are set and the caller is readied immediately after the request is received.

DESCRIPTION: J$START is applied to all jobs in the JXT.

Clear request

FUNCTION: Forces the end of a job's suspension, no matter for what reason the suspension was imposed

FUNCTION CODE: J$CLEAR

ENTRY: FC and JXO are required.

EXIT: The first output register is a status word normally set to 0. The second output register is simply a copy of the first input register.

The output registers are set and the caller is readied immediately after the request is received.

DESCRIPTION: This call is customarily made only during job termination and job abort, after a J$SUSPK call and prior to a J$DELETE call.

JSH clears the job's H, O, S, T, B, and E bits. The job is no longer suspended and begins to participate in the normal memory swapping activity unless its K bit is set.

Index request

FUNCTION: Marks a job irrecoverable

FUNCTION CODE: J$INDEX

ENTRY: FC and JXO are required.

EXIT: The output registers are set and the caller is readied immediately after the request is received.

DESCRIPTION: This call is made by the Exchange Processor when a job has become non-recoverable from the roll image.

JSH checks the job's roll index entry to see if it is already marked non-recoverable. If it is already marked, JSH does nothing. If it is not marked, JSH sets the non-recoverable bit in the job's roll index entry. The job's V bit is set to indicate an index write is pending. The job is disconnected and the index write is initiated.

Shutdown request

FUNCTION: Shuts down the system; normally used to prepare for an expected system interruption. Job activity is idled down, all jobs are rolled out and their memroy released. Station activity is not affected.

FUNCTION CODE: J$SHTDWN

ENTRY: FC is required.

EXIT: The first output register is a status word normally set to 0. The second output register is simply a copy of the first input register.

The output registers are set and the caller is readied immediately after the request is received.

DESCRIPTION: The B (suspended by recovery) and H (holding) bits are set for all jobs. The jobs are rolled out as soon as possible. When the jobs are rolled out and their memory is released, the H bits are cleared. The JXT limit (JXTMAX) is set to 0.

Recover request

FUNCTION: Recovers all jobs in the system

FUNCTION CODE: J$RCVR

ENTRY: FC is required.

EXIT: The first output register is a status word normally set to 0. The second output register is simply a copy of the first input register.

The output registers are set and the caller is readied immediately after the request is received.

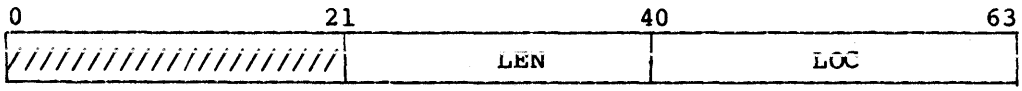
DESCRIPTION: J$RCVR or J$START must be used after a system interruption of a J$SHTDWN request. Either request causes JSH to clear the B bit in the job's status field. This has no immediate effect if the jobs are still under suspension of another reason.

Invoke Request

FUNCTION: Invokes a job class structure

FUNCTION CODE: J$INVOKE

ENTRY: FC, JXO, and ADDR are required. ADDR is the STP-relative address of an invoke request word, which has the following format:



Field	Bits	Description
LEN	21-39	The length of the array located at LOC. LEN must be a positive, nonzero multiple of 1000_8 that does not exceed I@ICSMAX.
LOC	40-63	The address, relative to the user's BA, of the array that contains the job class structure to be invoked.

EXIT:

The first output register is a status word which is normally 0. It is set to an appropriate error status when LEN is either less than 1000_8, not a multiple of 1000_8, or greater than I@JCSMAX. The second output register is a copy of the first input register.

The output registers are set, and the caller is readied immediately after the request is received.

DESCRIPTION:

JSH begins processing the request by placing the job in SK state and disconnecting it from the CPU. The array at LOC is copied to the CSD table as soon as all of the job's I/O requests are complete. Then the class assignments for all jobs in the input queue are redetermined.

No JXTs are allocated while a J$INVOKE request is pending. All jobs that issue a J$INVOKE request while another J$INVOKE request is pending are aborted.

FUNCTION:	Rolls a job out; user requested to protect against system interruptions.
FUNCTION CODE:	J$UROLL
ENTRY:	FC, JXO, and CODE are required. Code indicates whether the job rolled message should be sent to the user log (1) or not sent (0).
EXIT:	The first output register is a status word normally set to 0. The second output register is simply a copy of the first input register. The output registers are set and the caller is readied immediately after the request is received.
DESCRIPTION:	JSH begins processing the request by placing the job in S (suspended) state, disconnecting it from the CPU and marking it to be rolled out. When rollout is complete, the job's memory remains intact and its suspension is lifted. The job may again participate in the normal memory swapping activity.

4.6 PERMANENT DATASET MANAGER (PDM)

The Permanent Dataset Manager task (PDM) provides a means of creating, accessing, deleting, maintaining, and auditing permanent datasets.

Permanent datasets are of two types: user permanent datasets, which are created via a user request, and system permanent datasets, which are created by the system for spooled input and output datasets.

Each type of dataset may take on multitype attributes. A multitype dataset is described by one or more Dataset Catalog (DSC) entries, at least one of which is a spooled (system permanent) entry.

A dataset changes from a single-DSC-entry dataset to a multiple-DSC-entry dataset when it is staged by one or more DISPOSE statements. It returns to single-DSC-entry status when all related disposes have completed.

The PDM coordinates these activities via the Queued Dataset Table (QDT). A temporary dataset may also have multiple DSC entries.

Permanent dataset capabilities that may be requested by the user are divided into two categories: permanent dataset functions and permanent dataset utilities. The permanent dataset functions are:

- SAVE — Creates user permanent dataset
- ACCESS — Associates a user permanent dataset with a job
- DELETE — Removes a user permanent dataset from the system
- ADJUST — Changes the size of an existing permanent dataset
- MODIFY — Changes information for an existing permanent dataset

The system may request that input or output datasets be saved or deleted.

The permanent dataset utilities are:

- PDSDUMP — Dumps permanent datasets to a dataset
- PDSLOAD — Loads permanent datasets that have been dumped by PDSDUMP
- AUDIT — Produces a report containing status information for each permanent dataset

Functions not available to users that the Permanent Dataset Manager may be requested to perform are:

- PSEUDO-ACCESS — Accesses a permanent dataset during recovery of rolled jobs. Available only to the Startup task.
- REWRITE SDT — Updates the DSC copy of the job input SDT. Used by the Exchange Processor to declare a job ineligible for rerun.

4.6.1 TABLES USED BY PDM

The following tables are used in permanent dataset management:

DSC	Dataset Catalog
DNT	Dataset Name Table
DAT	Dataset Allocation Table
PDS	Permanent Dataset Table
PDD	Permanent Dataset Definition Table
JTA	Job Table Area
PDI	Permanent Dataset Information Table
DSP	Dataset Parameter Area
QDT	Queued Dataset Table

Detailed information on these tables is available in the COS Table Descriptions Internal Reference Manual, publication SM-0045.

Dataset Catalog (DSC)

The DSC is the table that makes a dataset permanent. It is a disk resident table partitioned into 512-word pages, each containing a block control word, a 7-word header, and eight 63-word DSC entries. When the DSC is initialized, it is cleared. Each DSC entry contains history information about the dataset and the DAT for the dataset. As the dataset size increases, the DAT size increases; therefore, for large datasets, more than one DSC entry may be required.

The page to which the permanent dataset is assigned is determined by hashing the permanent dataset name for a dataset created via a SAVE control statement or macro and hashing the dataset name for spooled datasets.

Dataset Name Table (DNT)

When the Permanent Dataset Manager is active, two DNTs are of primary concern: the DNT for the dataset currently being processed and the DNT for the DSC (Dataset Catalog). The DNT for the DSC is created by Startup.

Dataset Allocation Table (DAT)

When Permanent Dataset Manager is active, two DATs are of interest: the DAT for the dataset being processed and the DAT for the DSC (Dataset Catalog). The DAT for the DSC is created when the Dataset Catalog DNT is created and is pointed to by the DSC DNT.

Permanent Dataset Table (PDS)

An entry in the PDS exists for each active user permanent dataset. The PDS monitors which user permanent datasets are currently being used and controls dataset access by maintaining the number of users accessing the permanent dataset and controlling the waiting for access to a permanent dataset.

Permanent Dataset Definition Table (PDD)

The PDD is the input to the Permanent Dataset Manager. It contains the operation request for the Permanent Dataset Manager in the function code and all the parameters necessary to perform the operation.

Job Table Area (JTA)

The Permanent Dataset Manager uses the dataset DNT and the user number from the JTA.

Permanent Dataset Information Table (PDI)

The PDI is set up by Startup and contains device label information that is pertinent to permanent dataset management (number of hash and overflow pages) and contains a pointer to the DNT for the DSC.

Dataset Parameter Area (DSP)

A DSP for the DSC is assembled into the Permanent Dataset Manager task. It is used as input to Task I/O for reading and writing DSP pages.

Queued Dataset Table (QDT)

The PDM performs most of the maintenance required by the QDT. Due to the nature of multitype datasets, the common subroutine RELDNT also modifies entries. PDM performs functions such as assigning entries (spooled SAVE), updating assigned entries (SAVE, MODIFY, ACCESS, and DELETE), and releasing entries (DELETE and PDSREL).

4.6.2 SUBFUNCTIONS

A task calls the Permanent Dataset Manager by placing a PDD pointer and possibly a return address in its INPUT+0 and a JTA and/or DAT or DNT pointer in INPUT+1 of CMCC, the Permanent Dataset Manager communication block. The FC field of the PDD indicates the function to be performed.

The function codes processed by the task are as follows:

PMFCSU=10_8	Save user dataset
PMFCSI=12_8	Save input dataset
PMFCSO=14_8	Save output dataset
PMFCAU=20_8	Access user dataset
PMFCAI=26_8	Access spooled dataset
PMFCAO=26_8	Access spooled dataset
PMFCDU=30_8	Delete user dataset
PMFCDI=36_8	Delete spooled dataset
PMFCDO=36_8	Delete spooled dataset
PMFCPG=40_8	Page request
PMFCLU=50_8	Load user dataset
PMFCLI=52_8	Load input dataset
PMFCLO=54_8	Load output dataset
PMFCRL=60_8	PDS/Release request
PMFCPN=70_8	PDN request
PMFCDT=100_8	Dump time request
PMFCDQ=110_8	Dequeue SDT
PMFCEA=120_8	Queue SDT to available queue
PMFCEI=122_8	Queue SDT to input queue
PMFCEO=124_8	Queue SDT to output queue
PMFCAD=130_8	Adjust user dataset
PMFCMD=140_8	Modify user dataset
PMFCRSDT=150_8	Rewrite job's input SDT
PMFCPSAC=160_8	Pseudo-access for RRJ
PMFCPU=170_8	Access user-saved dataset for PDSDUMP
PMFCPO=176_8	Access output dataset for PDSDUMP
PMFCPI=176_8	Access input dataset for PDSDUMP

ACCESS processing (function codes 20, 26, 70)

Access processing handles the ACCESS control statement and macro and PDN requests (PDN requests are used by the system to see if a dataset exists). Input to access processing is a PDD table.

Calling sequence:

INPUT+0: PDD address, bits 40-63
Return address, bits 16-39 (if Exchange Package Processor call)

INPUT+1: JTA address, bits 40-63 (if job call)
DNT address, bits 16-39 (if job call)
SYS call flag, bit 0

DELETE processing (function codes 30, 36)

Delete processing handles the DELETE control statement and macro, PDSDUMP, and the delete spooled dataset request. Input consists of a PDD table.

Calling sequence:

```
INPUT+0:  PDD address, bits 40-63
          Return address, bits 16-39 (if Exchange Package Processor
          call)

INPUT+1:  JTA address, bits 40-63 (if job call)
          DAT address, bits 16-39 (if system call)
          SYS call flag, bit 0
```

Page Request processing (function code 40)

Page request processing is requested by both AUDIT and PDSDUMP to determine the permanent datasets in a given page. Input consists of a PDD table in the page request format.

Calling sequence:

```
INPUT+0:  PDD address, bits 40-63
          Return address, bits 16-39

INPUT+1:  JTA address, bits 40-63
```

LOAD processing (function codes 50, 52, 54)

The load processing request is used by PDSLOAD to reconstruct a DSC entry. Input consists of a PDD table in the load request format.

Calling sequence:

```
INPUT+0:  PDD address, bits 40-63
          Return address, bits 16-39

INPUT+1:  JTA address, bits 40-63
```

PDS/Release processing (function code 60)

PDS/Release processing handles the updating of the PDS table when a user permanent dataset is released.

Calling sequence:

INPUT+0: PDD address, bits 40-63
Return address, bits 16-39 (if Exchange Package Processor call)

INPUT+1: JTA address, bits 40-63

Dump Time processing (function code 100)

Dump time processing sets the dump time in the specified DSC to the current time and returns that time to the requester. Input is a PDD table.

Calling sequence:

INPUT+0: PDD address, bits 40-63
Return address, bits 16-39

INPUT+1: JTA address, bits 40-63

Dequeue SDT processing (function code 110)

Dequeue SDT processing removes the SDT from the input or output queue. Input is a PDD table.

Calling sequence:

INPUT+0: PDD address, bits 40-63
Return address, bits 16-39

INPUT+1: JTA address, bits 40-63

Queue SDT processing (function codes 120, 122, 124)

Queue SDT processing returns an existing SDT to the available, input, or output queue. Input is a PDD table.

Calling sequence:

INPUT+0: PDD address, bits 40-63
Return address, bits 16-39

INPUT+1: JTA address, bits 40-63

ADJUST processing (function code 130)

The ADJUST processing modifies the size of an existing user permanent dataset in the DSC. Input consists of a PDD table.

Calling sequence:

INPUT+0: PDD address, bits 40-63
Return address, bits 16-39 (if Exchange Package Processor call)

INPUT+1: JTA address, bits 40-63 (if job call)
DNT address, bits 16-39
SYS call flag, bit 0

MODIFY processing (function code 140)

MODIFY processing changes established information of an existing user permanent dataset in the DSC. Input consists of a PDD table.

Calling sequence:

INPUT+0: PDD address, bits 40-63
Return address, bits 16-39 (if Exchange Package Processor call)

INPUT+1: JTA address, bits 40-63 (if job call)
DNT address, bits 16-39
SYS call flag, bit 0

SDT rewrite processing (function code 150)

The fixed portion of the input SDT entry for the specified job is rewritten in the DSC. This is used only by EXP to declare a job ineligible for rerun and to signal that the job has been previously initiated so that Startup can recognize if a job is about to be rerun. Input consists of a PDD table and the DAT address for the SDT to be rewritten.

Calling sequence:

INPUT+0: PDD address, bits 40-63
Return address within EXP, bits 16-39

INPUT+1: Address of DAT from SDT, bits 16-39
JTA address, bits 40-63 (ignored)

Pseudo-access processing (function code 160)

When recovery of rolled jobs occurs during Startup, any permanent datasets that were accessed by a job being recovered must be re-linked. This means that PDS entries must be built or updated and the DAT from the rolled image must be verified against the DAT in the DSC. Input consists of a PDD table and DNT.

Calling sequence:

INPUT+0: PDD address, bits 40-63

INPUT+1: SYS call flag, bit 0
DNT address (within JTA), bits 16-39
JTA address, bits 40-63

PDSDUMP Access processing (function codes 170, 176)

Access of a dataset by PDSDUMP prevents the update of the last access time and the number of accesses by the dump itself. Input consists of a PDD table.

Calling sequence:

INPUT+0: PDD address, bits 40-63

INPUT+1: JTA address, bits 40-63
DNT address, bits 16-39

4.6.3 PDD STATUS

The return status is placed in the PMST field of the PDD (table 4.6-1). The logfile contains a corresponding code and message for most of the status conditions.

Table 4.6-1. PDD status

Logfile	PMST	Status
	1	Complete; no error
1	11	A DNT cannot be found for the specified dataset.
2	21	Maintenance permission not granted
3	31	Edition already exists
4	41	DSC full
5	51	Function code out of range
6	61	The job has a dataset of the local name (DN) specified.
7	71	No permission granted
	101	Delay and try again
9	111	DSC does not contain the requested dataset.
10	121	Edition does not exist
11	131	PDS full
12	141	Dataset not permanent
13	151	PDS entry not found
14	161	Continuation error

Table 4.6-1. PDD status (continued)

Logfile	PMST	Status
15	171	DAT full
16	201	DNT full
	211	End of DSC
18	221	PDN already accessed by this job
19	231	Request to read zero pages
20	241	Invalid page number requested
21	251	No data has been written to disk
	261	SDT does not exist
	271	SDT not on input or output queue
	301	Unable to queue SDT
25	311	Dataset name in PDD is 0
26	321	Maximum allowable edition of the dataset will be exceeded if this function is performed.
27	331	Multiple editions of the dataset exist, prohibiting changes to the permission control words.
28	341	Unique access is not acceptable because the dataset is part of the System Directory.
29	351	The PDD contains a text length without a text address, or a text address without a length specified.
30	361	The text length specified exceeds the allowable maximum.
31	371	The device on which all or part of the dataset resides is down.
	401	Error occurred while rewriting the SDT, or the SDT name and dataset type in the DSC do not match those in the PDD.
	411	Permanent dataset to be pseudo-accessed is not available or the DSC DAT does not match the JTA DAT.
34	421	Access is denied because crossed allocation unit exists.
36	441	The DSC entry was flagged by Startup as containing a fatal error. Access is denied.
	461	No available QDT entries exist to coordinate the dispose.
	471	The dataset has outstanding disposes; do not deallocate disk space.
39	501	Allocation of multitype dataset inconsistent with related datasets.
40	511	Multitype dataset has non-existent QDT entry.
41	521	Maximum edition reached

4.6.4 THEORY OF OPERATION

The Permanent Dataset Manager is called by the User Exchange Package Processor for SAVE, ACCESS, DISPOSE, RELEASE, DELETE, ADJUST, and MODIFY verbs and to perform functions for PDSDUMP, PDSLOAD, and AUDIT. The Permanent Dataset Manager is also called (1) by the Station Call Processor task to create DSC entries for spooled input datasets, to delete DSC entries for spooled output datasets, and to perform PDN requests, (2) by the Exchange Package Processor to create DSC entries for spooled output datasets, to delete DSC entries for spooled input datasets, and to rewrite spooled input dataset entries, and (3) by Startup, to rebuild PDS entries for permanent datasets associated with jobs being recovered or to access/save system datasets such as $R011 and $SDR.

Job termination must check to see if a dataset is permanent prior to releasing the dataset from the system.

General flow for SAVE processing

1. Validate parameters.
2. If SYS call, go to 5; otherwise,
3. Check if dataset exists.
4. If not, error; otherwise,
5. Get information from PDI.
6. Hash name (DN for spooled, PDN for non-spooled).
7. Read DSC hash page.
8. If spooled, go to 25; otherwise,
9. Search for matching PDN and ID (with highest or matching edition number).
10. If match not found, go to 15; otherwise,
11. Compare MN permission control words.
12. If MNs do not match, error; otherwise,
13. If edition numbers match, error; otherwise,
14. Save permission control words.
15. If page has never overflowed, go to 25; otherwise,
16. Increment overflow page count.
17. If end of overflow, go to 25; otherwise,
18. Read DSC overflow page.
19. Search for matching PDN and ID (with highest or matching edition number).
20. If no match found, go to 24; otherwise,
21. Compare MN permission control words.
22. If MNs do not match, error; otherwise,
23. If edition numbers match, error; otherwise,
24. If page edition has overflowed, go to 16; otherwise,
25. If hash page not full, go to 30; otherwise,
26. Increment overflow page number.
27. If end of overflow, error; otherwise,

28. Read overflow page.
29. Search for empty slot in page.
30. If empty slot found, go to 34; otherwise,
31. Set page full flag.
32. Write page to DSC.
33. Go to 26.
34. If continuation processing, go to 57; otherwise,
35. Create DSC entry.
36. Put first (and second) DAT page in initial DSC entry or text.
37. If all DAT moved, go to 43; otherwise,
38. Search for empty slot.
39. If empty slot not found, go to 54; otherwise,
40. Put continuation pointer in previous DSC.
41. Create continuation DSC entry.
42. Go to 37.
43. Write page to disk.
44. If spooled, go to 46; otherwise,
45. Create PDS entry.
46. If SYS call, go to 48; otherwise,
47. Update DNT.
48. Put DSC pointer in DAT.
49. Set status.
50. Clear constants.
51. Clear input registers.
52. Set output registers and ready calling task.
53. Exit.
54. Set page full flag.
55. If page not hash page, move to hash buffer.
56. Go to 26.
57. Put continuation pointer in previous DSC entry.
58. Write DSC page.
59. Go to 41.

General flow for ACCESS processing

1. Validate parameters.
2. Check if dataset exists.
3. If it does, error; otherwise,
4. Get information from PDI.
5. Hash name (PDI for non-spooled).
6. Read DSC hash page.
7. Search for matching PDN and ID (with highest, lowest, or matching edition number).
8. If match not found, go to 17; otherwise,
9. If edition numbers match not found, go to 17; otherwise,
10. Compare permission control words.
11. If no permissions granted, error; otherwise,
12. Search PDN for matching DSC pointer and edition number.
13. If match not found, go to 30; otherwise,
14. Check access required.

15. If multiaccess allowable, go to 40; otherwise,
16. Return to caller with delay error code.
17. If page has overflowed, go to 20; otherwise,
18. If PDN and ID match not found, error; otherwise,
19. Go to 10.
20. Increment overflow page number.
21. If end of overflow, go to 26; otherwise,
22. Read overflow page.
23. Search for matching PDN and ID (with highest, lowest, or matching edition number).
24. If match found, go to 29; otherwise,
25. If page has ever overflowed, go to 22; otherwise,
26. If PDN and ID match not found, error; otherwise,
27. Calculate entry number.
28. Go to 10.
29. Move overflow buffer to hash buffer.
30. Move first DAT pages from DSC to memory.
31. Update DSC entry.
32. Write DSC page.
33. If all DAT moved, go to 39; otherwise,
34. If next DSC page in this page, go to 36; otherwise,
35. Read DSC page.
36. Move continuation DAT into memory.
37. If all DAT moved, go to 39; otherwise,
38. Go to 34.
39. Create PDS entry.
40. Create DNT entry.
41. Set status.
42. Clear constants.
43. Clear input registers.
44. Set output registers and ready calling task.
45. Exit.

General flow for DELETE processing

1. Validate parameters.
2. If SYS call, go to 7; otherwise,
3. Check if dataset exists.
4. If not, error; otherwise,
5. If dataset is not permanent, error; otherwise,
6. If MN permission not granted, error; otherwise,
7. Get information from PDI.
8. Get hash page and entry number from DAT.
9. Read DSC page.
10. Clear entry.
11. If DSC not continued, go to 13; otherwise,
12. If next entry in same page, go to 10; otherwise,
13. Write DSC page.
14. If DSC continued, go to 9; otherwise,
15. If spooled dataset, go to 22; otherwise,

16. Search PDS for match (DAT match).
17. If not found, error; otherwise,
18. Decrement PDS user count.
19. Clear PDS entry.
20. Clear permanent flag in DNT.
21. Set status.
22. Clear constants.
23. Clear input registers.
24. Set output register and ready calling task.
25. Exit.

General flow for ADJUST processing

1. Validate parameters.
2. If not a permanent dataset, error; otherwise,
3. If no size change, go to 39; otherwise,
4. If contraction, deallocate excess space; otherwise,
5. Read DSC page.
6. Clear the DAT area.
7. Insert new DAT page in DSC entry.
8. If no more DATs, go to 30; otherwise,
9. Insert new DAT page in DSC entry.
10. If no more DATs, go to 30; otherwise,
11. If DSC continuation entry does not exist, go to 18; otherwise,
12. If DSC continuation on same page, go to 15; otherwise,
13. Write current DSC page.
14. Read next DSC page.
15. Clear the DSC continuation entry.
16. Create DSC continuation entry.
17. Go to 10.
18. Search for empty slot on this page.
19. If no slot found, go to 22; otherwise,
20. Create DSC continuation entry.
21. Go to 10.
22. Set page full and overflow.
23. If page not hash page, move to hash buffer.
24. Read next overflow page.
25. Search for an empty slot in overflow page.
26. If slot found, go to 20; otherwise,
27. Set page full and overflow
28. Write DSC overflow page.
29. Go to 24.
30. If there are no continuations, go to 37; otherwise,
31. If continuation on current page, go to 34; otherwise,
32. Write current page.
33. Read continuation page.
34. Clear continuation entry.
35. Set page not full
36. Go to 30.
37. Write final page.

38. Set DNT Dataset Catalog size equal to dataset size in DAT.
39. Set PDD status.
40. Exit.

General flow for MODIFY processing

1. If SYS call, go to 4; otherwise,
2. Check if dataset exists.
3. If not, error; otherwise,
4. Get PDI information.
5. Verify requester has all permissions.
6. If not, error; otherwise,
7. Read original entry into memory.
8. Determine if PDN, ID and/or ED being changed.
9. Search for duplicate dataset.
10. If found, error; otherwise,
11. If new entry goes on same hash page, use existing entry; otherwise, locate an available slot on new page.
12. Create new entry.
13. Delete the old entry.
14. If continuation entries exist, change the first page/entry indicator; otherwise,
15. Write the new page.
16. Write the old page.
17. Change the DSC pointer in the DAT.
18. Change the DSC pointer in the PDS.
19. Exit.

General flow for Page Request processing

1. Validate PDD input.
2. Get PDI information.
3. Check if pages to read.
4. If not, go to 8; otherwise,
5. Read page.
6. Move page to user buffer.
7. Go to 3.
8. If not end of DSC, go to 10; otherwise,
9. Set end of DSC status.
10. Set complete status.
11. Clear constants.
12. Clear input registers.
13. Set output registers and ready calling task.
14. Exit.

General flow for LOAD processing

1. Check if function code indicates spooled input.
2. If not, go to 4; otherwise,
3. Set spooled indicator.
4. Execute save processing.
5. Check if this was a spooled input.
6. If not, go to 8; otherwise,
7. Create an SDT.
8. Exit.

General flow for PDS/Release processing

1. Validate parameters.
2. Check if PDS entry exists.
3. If not, error; otherwise,
4. Decrement PDS user count.
5. If user count not 0, go to 8; otherwise,
6. Clear PDS entry.
7. Decrement PDS entry count.
8. Set status.
9. Clear constants.
10. Clear input registers.
11. Set output registers and ready calling task.
12. Exit.

General flow for Dump Time processing

1. Check if DNT for the dataset exists.
2. If not, error; otherwise,
3. Check if DNT indicates a permanent dataset.
4. If not, error; otherwise,
5. Get DAT address from DNT.
6. Get DSC address from DAT.
7. Break DSC address into page and entry.
8. Read DSC page.
9. Get current time.
10. Insert time into DSC entry.
11. Save time for the requester.
12. Write DSC page.
13. Exit.

General flow for dequeue SDT processing

1. Search SDT for matching dataset name and job sequence number.
2. If not found, error; otherwise,
3. Verify that SDT is on either the input queue or the output queue.
4. If not, error; otherwise,

5. Remove the SDT from the queue.
6. Exit.

General flow for queue SDT processing

1. Determine appropriate queue from function code.
2. Make queue request.
3. If request incomplete, error; otherwise,
4. Exit.

General flow for SDT rewrite processing

1. Locate DSC entry for desired SDT from the DADSC field of the DAT.
2. Read the DSC page and locate the entry.
3. Verify that the name in the DSC matches the name in the PDD.
4. Move information from the PDD to the DSC entry.
5. Write the DSC page.
6. Exit.

General flow for Pseudo-access processing

1. Using the supplied DNT address, locate the DAT address.
2. Error if no DNT or if DAT pointer is 0 or points to STP DAT
3. Get DSC pointer from DAT; error if none
4. Read DSC page for first entry. Locate entry in page.
5. Check that entry is for saved dataset, not continuation; no down device in DAT; no AI conflicts in DAT. Error if any checks fail.
6. Locate first DAT body in DSC entry. If necessary, read a continuation entry to locate first DAT body.
7. Verify that the DSC DAT size is less than or equal to the JTA DAT size.
8. Verify that the DSC copy of the DAT matches the JTA copy of the DAT, or as much of the JTA copy as there is DSC DAT to compare. The JTA DAT may represent a larger dataset than is reflected in the DSC, but only as much as is in the DSC needs to be verified. Read continuation pages as necessary. Error if JTA DAT ends before DSC DAT, or if any AI in the DSC does not match the corresponding AI in the JTA.
9. Search for existing PDS entry. If none exists, go to 11.
10. See if able to add this job as a user of this dataset. Error if job needs unique access, or if unique access is already granted to another job. If access can be granted, return with successful status.
11. Create a PDS entry. Set permission bits from the DNT and return with successful status.

4.7 LOG MANAGER (MSG)

The log manager for the CRAY-OS is called the Message Processor (MSG).

4.7.1 MESSAGE PROCESSOR (MSG)

The Message Processor (MSG) writes messages in the system and user log files in response to requests from other tasks. Users request entries to be made in these files through requests to the Exchange Processor, which in turn calls the Message Processor. The ID for the Message Processor is MSG; the task priority is set just below that for the Disk Queue Manager (DQM) and the Station Call Processor (SCP).

Two separate queues are created in the STP memory pool: one with messages going to the system log and one with messages going to user logs. For each message request, a system log entry and/or a user log entry is constructed. Then the messages are written from the queues to the appropriate files via Task I/O (see section 3.3).

A system log created by MSG can be used and analyzed by the EXTRACT program and by the STATS program. Any edition, including the running edition, can be accessed by a user having the correct read password.

System log processing

The system log is a permanent dataset named $SYSTEMLOG. If no system log exists when the system is deadstarted, MSG calls the Permanent Dataset Manager (PDM) to create edition number 1 of $SYSTEMLOG. An installation parameter, I@LGDSZ, determines the size of $SYSTEMLOG. Log manager initializes $SYSTEMLOG with end-of-file RCWs and terminates it with an end-of-data RCW. This initialization permits a user program such as EXTRACT to read $SYSTEMLOG without accidentally running off the end. It also allows MSG to recover its position on the system log during a restart.

Disk initialization delays execution only of those system tasks waiting to write messages on the log. After initialization, MSG rewinds the dataset to the beginning of information. PDM enters $SYSTEMLOG in the Dataset Catalog (DSC) after it has been initialized.

If the system log fills up, a new edition is created and initialized. No messages are lost when this happens.

The system log memory buffer and DSP are allocated in high memory to facilitate system log recovery.

At system startup, the log manager attempts to recover the system log using the following procedure.

1. Access $SYSTEMLOG.
   If none exists, go to step 4.

2. Validate table area and buffer pointers; set *x* in recovery message to OK.
   If valid, go to step 5.

3. Read $SYSTEMLOG to *eof*. Then backspace once. Initialize high memory table area. Set *x* in recovery message to 2. Go to step 5.

4. Initialize $SYSTEMLOG with *eof* marks. Call PDM to save $SYSTEMLOG in the Dataset Catalog. Rewind $SYSTEMLOG. Initialize high memory table area. Set *x* in recovery message to 1.

5. Enter SY014 - SYSTEMLOG RECOVERY SUCCESS CODE *x* message in the log. The recovery success codes are as follows:

Code	Definition
1	$SYSTEMLOG did not exist.
2	Recovery validation word was bad.
3	$SYSTEMLOG edition numbers did not match.
4	$SYSLOG DSP failed validation.
5	The *eor* flag was not set in $SYSLOG DSP.
6	Record control was not RCW or BCW.
7	Bad forward word index in record control word
8	Bad block number in block control word
OK	Good recovery

User log processing

A user log dataset named $LOG is created for each job by JSH when a job is initiated. The buffer for this dataset is in the JTA for the job. Tasks (and the user via the Exchange Processor) may request MSG to write messages in the user log.

User log messages are placed on one general queue in the memory pool, and each job may have ten messages backlogged on the queue. If more messages are to be entered in a user log while at its maximum queue count (before its backlog can be written from the queue to $LOG), the MSG task begins discarding messages. The last two messages on the queue for that job are replaced with an overflow message and the new message request, preventing a looping job from filling the entire queue and hanging the system while waiting for I/O.

The maximum size of each $LOG is limited by the installation parameter I@LGUSZ.

4.7.2 SYSTEM TABLES USED BY MSG

The following tables are used for message processing.

DSP	Dataset Parameter Area
JXT	Job Execution Table
JTA	Job Table Area
SDT	System Dataset Table
PDD	Permanent Dataset Definition Table
AUT	Active User Table, for interactive mode
LGJ	Log JXT Table

Detailed information for these tables is available in the COS Table Descriptions Internal Reference Manual, publication SM-0045.

Dataset Parameter Area (DSP)

Task I/O uses the Dataset Parameter Area tables for the $SYSLOG and the $LOG datasets. The DSP and I/O buffer for $SYSLOG are allocated in high memory during system startup. The DSP and I/O buffer for $LOG are in the user JTA.

Job Execution Table (JXT)

MSG uses the following fields in the JXT:

JXSDT	SDT offset
JXJN	Jobname
JXTSX	Time spent executing
JXJTA	JTA address
JXSTAT	Status bit N (not in memory)
JXLFM	Last logfile message
JXSTCH	Job status in displayable form
JXORD	Ordinal number
JXIA	Interactive flag
JXLVL	Procedure level
JXAUT	AUT table address

Job Table Area (JTA)

MSG uses the following fields in the Job Table Area:

JTLDP	User DSP address for $LOG
JTTSX	Time spent executing
JTDLM	Disable log message flag; set during job termination
JTMSG	User log record area
JTLGF	User log buffer
JTJN	Job name
JTJXT	JXT address
JTLOG	Logfile ($LOG) DNT

Permanent Dataset Definition Table (PDD)

Permanent Dataset Manager requests issued by MSG are accompanied by Permanent Dataset Definition tables.

4.7.3 TASK CALLS TO MSG

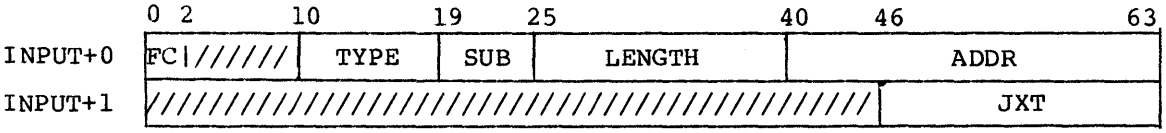
A System Task Processor task calls MSG only through the synchronous TSKREQ routine and receives a reply as soon as the message has been copied to a queue entry in an STP memory pool. This allows any part of STP to enter a message in the system and/or user log very easily.

Before executing a return jump to TSKREQ, place the following information in input registers:

(A2) MSGID,0

(S1) INPUT+0 }
(S2) INPUT+1 } Request

Request format:



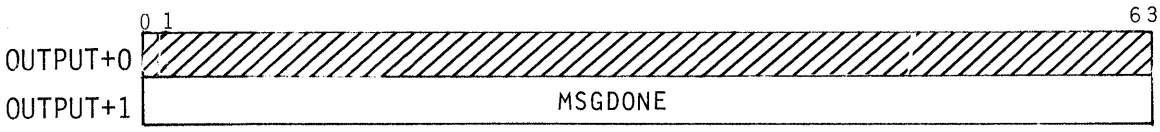
Field	Word	Bits	Description
FC	INPUT+0	0-1	Function code: 1 Write message in user's logfile only 2 Write message in SYSTEMLOG only 3 Write message in both user's logfile and in SYSTEMLOG
TYPE	INPUT+0	10-18	Major class of log record (section 4.7.4)
SUB	INPUT+0	19-24	Subtype of log record (section 4.7.4)
LENGTH	INPUT+0	25-39	Length in words of message. If length is 0, the message is a character string terminated by a zero byte in any position.
ADDR	INPUT+0	40-63	Starting address relative to STP of message to be written
JXT	INPUT+1	46-63	JXT address if message associated with job; otherwise 0

The calling task receives the following information after the message has been built and queued for output:

(S1) OUTPUT + 0

(S2) OUTPUT + 1

Reply format:

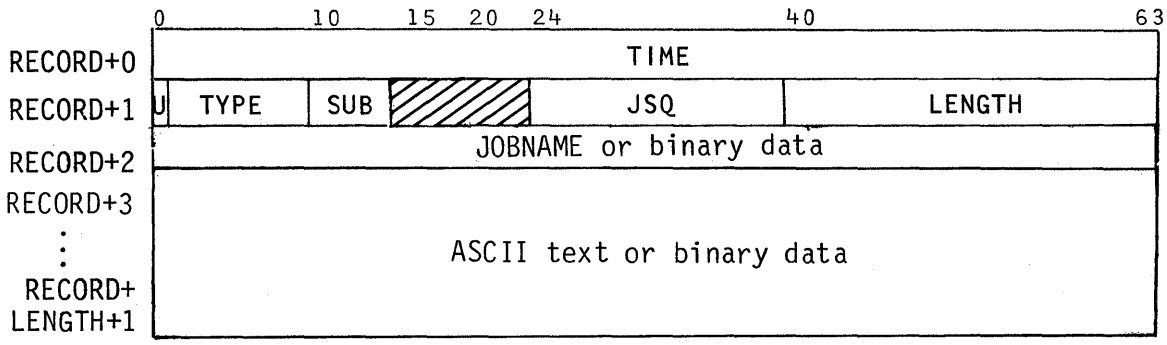


Field	Word	Bits	Description
MSGWORD	OUTPUT+0	0-63	First word of the message located at ADDR specified in the input request. This may be ASCII characters or binary data.
MSGDONE	OUTPUT+1	0-63	Characters 'MSG DONE' in ASCII

4.7.4 $SYSTEMLOG FORMAT

The format of $SYSTEMLOG is easily read by a user program with the correct read password. Each message occupies a single variable-length record on the system log. All records of the same type and subtype have the same format. MSG builds two or three header words, but it does no other formatting of the message. It merely transfers the ASCII or binary data from the location specified by the starting address in the request to the memory pool queue. Then the entire record is written to the system log via Task I/O. The EXTRACT utility program processes each type according to its format for each different message to produce its report, and the STATS utility program uses several of the types to procure its daily and monthly statistical report.

Records are formatted as follows:



Field	Word	Bits	Description
TIME	RECORD+0	0-63	Real-time clock expressed in cycles
U	RECORD+1	0	User flag; message also written in user's logfile

Field	Word	Bits	Description
TYPE	RECORD 1	1-9	Major message type. See system log message types.
SUB	RECORD+1	10-15	Subtype of message. See system log message type.
JSQ	RECORD+1	24-39	Job sequence number (SDJSQ). Used only if entry is associated with a job.
LENGTH	RECORD+1	40-63	Number of words excluding the first two words in this record
JOBNAME	RECORD+2	0-63	Job name (JXJN), used for type 1 and type 4 records. For other record types, this word contains the beginning of binary information.
	RECORD+3 through RECORD+LENGTH+1	0-63	For all but type 1 records, binary information as supplied by caller; for type 1 records, ASCII text.

The following paragraphs describe formats for different message types recorded in the system log.

Type 0 - null messages

A type 0 message is a null record that may have a length of 0. Null messages are used to pad blocks so that no message in the system log crosses a disk sector boundary. Type 0 messages do not have subtypes.

Type 1 - ASCII string messages

Type 1 messages includes all user-requested messages and system-generated informative messages.

The subtype field contains the task number of the task that requested the message or is a 16_8 for a user-requested entry. (Only type 1 messages can be written in a user logfile.)

Type 2 - Station Call Processor messages

Type 2 messages are issued by the Station Call Processor. Message subtypes relating to the station are:

Subtype	Message
1	Staging in of datasets
2	Staging out of datasets
3	Retransmission§

Type 3 - Hardware messages

Type 3 messages record hardware errors detected during normal operations. These errors are of possible interest to field engineers.

Subtype	Message
1	1-bit corrected memory errors
2	Uncorrectable memory errors
3	Disk errors
4	Channel errors§

Type 4 - Accounting messages

Type 4 messages include all job-related accounting information. Subtypes provide for the following types of accounting messages:

Subtype	Message
1	Job termination
2	PDM accounting messages

Type 5 - Startup messages

Type 5 messages are issued during system startup processing. Message subtypes are as follows:

Subtype	Message
1	Permanent dataset recovery
2	Rolled job recovery

§ Deferred implementation

Type 6 - System performance messages

Type 6 messages are issued by the System Performance Monitor on a periodic basis. These records report on the performance and usage of COS, the processors, and I/O. Section 4.10 describes the System Performance Monitor Task in detail.

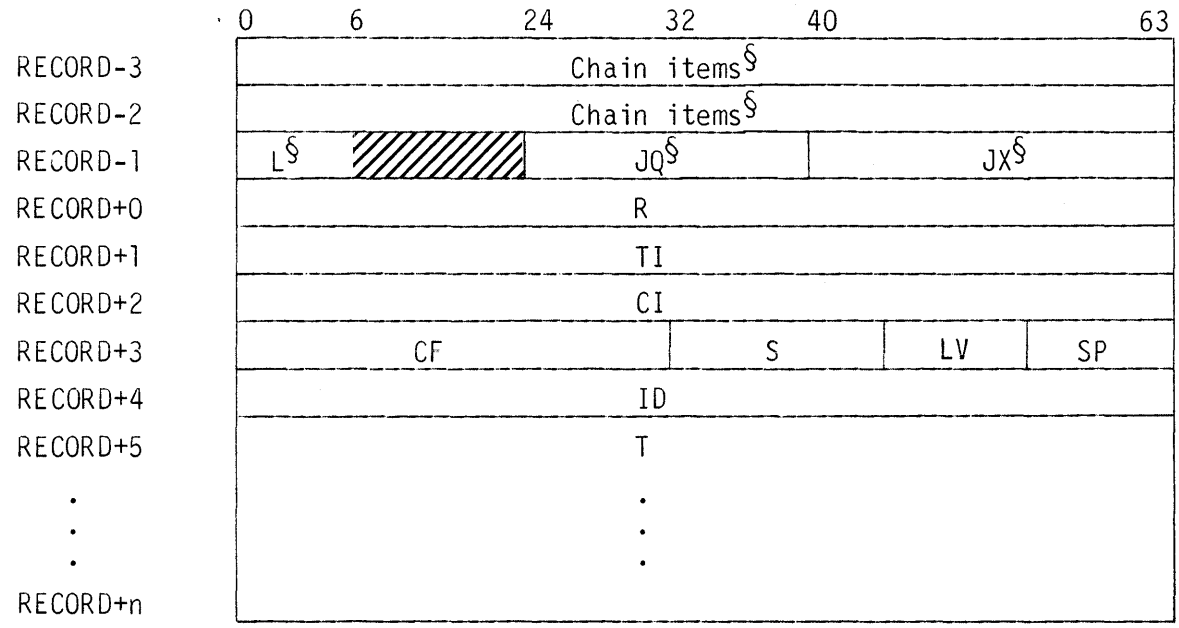
Subtype	Message
1	CPU utilization
2	Task utilization
3	EXEC requests from each task
4	User memory utilization§
5	Disk utilization§
6	Disk channel utilization
7	Link utilization§
8	EXEC call usage
9	User call usage§
10	Channel interrupt counts
11	Job Scheduler statistics
12	Job class statistics

§ Deferred implementation

4.7.5 $LOG FORMAT

Each job has a user log named $LOG containing a history of the job. Log messages are generated by the operating system and by the user job itself. Each message occupies a single variable-length record on $LOG. At the completion of the job, $LOG is copied to $OUT, which is then staged for output.

Records are formatted as follows:



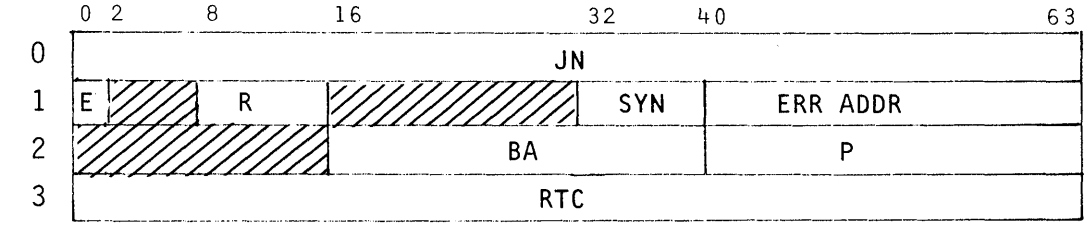
Field	Word	Bits	Description
L	RECORD-1	0-5	Number of words, excluding the first three header words
JQ	RECORD-1	24-39	Job sequence number
JX	RECORD-1	40-63	JXT address

§ These fields exist only while the record is being built in the memory pool area, but are discarded when the record is written into the actual $LOG dataset.

Field	Word	Bits	Description
R	RECORD+0	0-63	ASCII space codes
TI	RECORD+1	0-63	Wall clock time expressed in ASCII code in the format hh:mm:ss
CI	RECORD+2	0-63	CPU time expressed in ASCII code, seven digits and a decimal point
CF	RECORD+2	0-31	CPU time, four fractional digits
S	RECORD+3	32-47	ASCII space codes
LV	RECORD+3	48-55	Procedure level
SP	RECORD+3	56-63	ASCII space code
ID	RECORD+4	0-63	ID of calling task or user in ASCII code (left-justified)
T	RECORD+5-n	0-63	Message text in ASCII code; last word is left-justified with zero fill

4.8 MEMORY ERROR PROCESSOR (MEP)

The Memory Error Processor (MEP) exists so that EXEC can communicate with the system log. Its purpose is to relay information about memory errors from EXEC to the Message Processor (MSG). Messages from EXEC to MSG are in the following format:



Field	Word	Bits	Description
JN	0	0-63	Jobname or 'STP'
E	1	0,1	Error type (binary): 10 Uncorrectable memory error 01 Correctable memory error
R	1	8-15	Read mode: 0 Scalar 1 I/O 2 Vector 3 Fetch
SYN	1	32-39	Syndrome bits
ERR ADDR	1	40-63	Error address
BA	2	26-39	Base address
P	2	40-63	Program address
RTC	3	0-63	Real-time clock

4.9 DISK ERROR CORRECTION (DEC)

The Disk Error Correction Task (DEC) is called by the CRAY-0S Disk Queue Manager Task (DQM) to attempt correction of a disk error by applying the cyclic redundancy checkword (CRC) algorithm described in the DCU-2 Disk Controller Reference Manual, CRI publication 2240630.

4.9.1 SYSTEM TABLE USED BY DEC

DEC uses the Equipment Table (EQT).

Equipment Table (EQT)

The EQT contains information for device allocation, physical operation control, device request queue management, channel configuration, performance monitoring, error counting, and error correction. Detailed information on this table is available in the COS Table Descriptions Internal Reference Manual, publication SM-0045.

4.9.2 DEC INTERFACE WITH OTHER TASKS

DEC is called by a task through PUTREQ, which places the EQT address in INPUT+0. DEC returns the request word in the reply.

INPUT REGISTERS:	INPUT+0	24/return,16/0,24/EQT
	INPUT+1	64/0
OUTPUT REGISTERS:	OUTPUT+0	64/error status
		Error status: 0 corrected error -1 uncorrected error
	OUTPUT+1	24/return,16/0,24/EQT

4.10 SYSTEM PERFORMANCE MONITOR (SPM)

The System Performance Monitor (SPM) is a low priority task that collects system performance data and periodically sends it to the system log. Once SPM is created, it goes into an infinite loop where it is readied by EXEC, collects information sending it to the Log Manager, and performs a time delay. SPM's only communication with other tasks is one-way to the Log Manager.

4.10.1 CONTROL PARAMETERS

The following System Task Processor (STP) parameters control SPM's data collection:

I@SPMDLY	Delay interval between collection periods in seconds
I@SPMMIN	Delay interval when waiting for buffer space in which to collect information
I@SPMON	SPM task enable flag, checked every I@SPMDLY seconds
I@SPMTYP	SPM subtype enable vector. Each bit set turns on the data collection for the respective subtype (or group of data, as enumerated in section 4.10.3). The vector is right-adjusted so that the rightmost bit corresponds to subtype 12.

The value of I@SPMDLY is an installation option. The suggested value is 30 minutes (1800 seconds). In general, the advantages of setting a large value for the interval are:

- Smaller system overhead and
- Smaller volume of EXTRACT output.

The advantages of a small interval are:

- More detailed statistics of specific period of time and
- Smaller probability of losing SPM data through system crashes.

The operator may change parameters while COS is running by entering new values into the corresponding STP locations using system debug commands.

4.10.2 METHOD OF DATA COLLECTION

Performance data is stored in STP as well as in EXEC. In general, such data accumulates with time until it is read by SPM. At that time, the data areas are zeroed and accumulation resumes. Therefore, each system log record contains data accumulated since the last collection period. EXEC tables are read and zeroed via EXEC requests. During collection periods, STP is locked, ensuring that other tasks do not read and update a data area between the time the data is read by SPM and the time the data area is zeroed.

4.10.3 DATA COLLECTION AND RECORD DEFINITION

Twelve groups (or subtypes) of data are collected by SPM. Each group is sent to the Log Manager as a record. All SPM records belong to Log Manager record type 6. The data subtypes and record definitions are given in tables 4.10-1 through 4.10-12.

A listing of subtypes follows:

Table	Subtype	Description
4.10-1	1	CPU usage
4.10-2	2	Task usage
4.10-3	3	EXEC requests
4.10-4	4	User memory usage§
4.10-5	5	Disk usage§
4.10-6	6	Disk channel usage
4.10-7	7	Link usage§
4.10-8	8	EXEC call usage
4.10-9	9	User call usage§
4.10-10	10	Interrupt count
4.10-11	11	Job scheduler management statistics
4.10-12	12	Job class information

§ Deferred implementation

Table 4.10-1. CPU usage record - subtype 1

Word	Data	Origin of Data
0	Time interval	Calculated in SPM
1	User time	UTIM (in EXEC)
2	Idle time	ITIM (in EXEC)
3	Blocked time	BTIM (in EXEC)
4	EXEC time	ETIM (in EXEC)
5	Number of tasks	NE@STT (in COSTXT)
6	Task 0 time	STTIME field of STT (in EXEC)
7	Task 1 time	STTIME field of STT (in EXEC)
.	.	.
.	.	.
.	.	.
N+6	Task N time	STTIME field of STT (in EXEC)

Table 4.10-2. Task usage record - subtype 2

Word	Data	Origin of Data
0	Time interval	Calculated in SPM
1	Number of tasks	NE@STT (in COSTXT)
2	Number of task 0 readies	STCNT field of STT (in EXEC)
3	Number of task 1 readies	STCNT field of STT (in EXEC)
.	.	.
.	.	.
.	.	.
N+2	Number of task N readies	STCNT field of STT (in EXEC)

Table 4.10-3. EXEC requests record - subtype 3

Word	Data	Origin of Data
0	Time interval	Calculated in SPM
1	Number of tasks	NE@STT (in COSTXT)
2	Number of task 0 requests	STNEC field in STT (in EXEC)
3	Number of task 1 requests	STNEC field in STT (in EXEC)
.	.	.
.	.	.
.	.	.
N+2	Number of task N	STNEC field in STT (in EXEC)

Table 4.10-4. User memory usage record - subtype 4§

Word	Data	Origin of Data
0	Time interval	Calculated in SPM
1	Available memory integral	PMO4AMI in STP
2	I/O wait integral	PMO4IMI in STP
3	CPU wait integral	PMO4CMI in STP
4	CPU execute integral	PMO4WMI in STP

§ Deferred implementation

Table 4.10-5. Disk usage record - subtype 5§

Word	Data	Origin of Data
0	Time interval	Calculated in SPM
1	Number of devices	
2	Logical device 0 name	
3	Blocks transferred (device 0)	
4	Seek time (device 0)	
5	Transfer time (device 0)	
6	Number of physical requests (device 0)	
7	Number of non-seek requests (device 0)	
.	.	
.	.	
.	.	
6*N+2	Logical device N name	
6*N+7	Number of non-seek requests (device N)	

Table 4.10-6. Disk channel usage record - subtype 6

Word	Data	Origin of Data
0	Time interval	Calculated in SPM
1	Channel 0 time	DTCTA field in DCT (in STP)
2	Channel 1 time	DTCTA field in DCT (in STP)
.	.	.
.	.	.
.	.	.
12	Channel 11 time	DTCTA field in DCT (in STP)

§ Deferred implementation

Table 4.10-7. Link usage record - subtype 7§

Word	Data	Origin of Data
0	Time interval	Calculated in SPM
1	Number of links	
2	Logical ID (link 0)	
3	Number of messages (link 0)	
4	Number of words sent (link 0)	
5	Number of words received (link 0)	
.	.	
.	.	
.	.	
4*N+2	Logical ID (link N)	
4*N+3	Number of messages (link N)	
4*N+4	Number of words sent (link N)	
4*N+5	Number of words received (link N)	

Table 4.10-8. EXEC call usage record - subtype 8

Word	Data	Origin of Data
0	Time interval	Calculated in SPM
1	Number of types	MTCTL (in COSTXT)
2	Number of type 0 requests	MCT (in EXEC)
3	Number of type 1 requests	MCT (in EXEC)
.	.	.
.	.	.
.	.	.
N+2	Number of type requests	MCT (in EXEC)

§ Deferred implementation

Table 4.10-9. User call usage record - subtype 9§

Word	Data	Origin of Data
0	Time interval	Calculated in SPM
1	Number of user request types	
2	Number of type 0 requests	
3	Number of type 1 requests	
.	.	
.	.	
.	.	
N+2	Number of type N requests	

Table 4.10-10. Interrupt count record - subtype 10

Word	Data	Origin of Data
0	Time interval	Calculated in SPM
1	Number of channels	NE@IC (COSTXT)
2	Number of channel 0 interrupts	IC table (EXEC)
3	Number of channel 1 interrupts	IC table (EXEC)
.	.	.
.	.	.
.	.	.
N+2	Number of channel N interrupts	IC table (EXEC)

§ Deferred implementation

Table 4.10-11. Job Scheduler management statistics record - subtype 11

Word	Data	Origin of Data
0	Time interval	Calculated in SPM
1	Number of memory compacts	PM11MC in STP
2	Number of rolls	PM11NR in STP
3	Number of expands	PM11FLE in STP
4	Number of reduces	PM11FLR in STP
5	Number of initiates	PM11NI in STP
6	Number of terminates	PM11NT in STP
7	Number of scheduling intervals	PM11SI in STP
8	Number of index writes	PM11IN in STP
9	Job class structure name	CSSNM in STP
10	Number of jobs in the system	Input and execute queue counts in STP
11	Number of active IXTs	JXTPOP in STP
12	Maximum number of JXTs	JXTMAY in STP
13	Number of available pool JXTs	JXTMAX-CSSCUM-CSAPL in STP
14	Number of defined JXTs	CSNCL in STP
15	Number of classes waiting for JXTs	CSNCW in STP

Table 4.10-12. Job class information record - subtype 12

Word	Data	Origin of data
0	Time interval	Calculated in SPM
1	Job class name	CSCNM in STP
2	Number of active JXTs	CSACT in STP
3	Number of jobs waiting for JXTs	CSWTG in STP
4	Number of reserved JXTs	CSRES in STP
5	Maximum number of JXTs	CSMAX in STP
6	Status (ON/OFF)	CSOFF in STP

4.10.4 TASK FLOW FOR SPM

1. If SPM not enabled, go to 15; otherwise,
2. Get a buffer for each subtype.
3. If not enough memory, return buffers and go to 15; otherwise,
4. Lock STP.
5. If each enabled subtype buffer filled, go to 9; otherwise,
6. Put time interval into word 0 of buffer.
7. Call corresponding collection routine to fill the buffer.
8. Go to 5.
9. Unlock STP.
10. If there are no buffers left, go to 17; otherwise,
11. Call MSG to write the subtype to system log.
12. Check MSG reply.
13. Return the buffer.
14. Go to 10.
15. Time delay for I@SPMMIN seconds.
16. If now have buffer space, go to 4; otherwise,
17. Time delay for I@SPMDLY seconds.
18. Go to 1.

4.10.5 SYSTEM TABLES USED BY SPM

The following system tables are used by SPM:

IC	Interrupt Count Table
STT	System Task Table
MCT	Monitor Call Table
DCT	Device Channel Table
CSD	Class Structure Definition Table

Detailed information on these tables is available in the COS Table Descriptions Internal Reference Manual, publication SM-0045.

Interrupt Count Table (IC)

The IC counts interrupts for each channel or pseudo channel. This table is read and zeroed indirectly through EXEC calls.

System Task Table (STT)

Four fields of the STT are used by SPM. STCNT maintains the ready count, STNEC the normal exit count, STTIME the task execution time, and STLPMC the time of the last SPM call.

Monitor Call Table (MCT)

Monitor call table serves only SPM. It counts each type of call to EXEC from various tasks.

Table format:

0	Number of type 0 requests
1	Number of type 1 requests
.	.
.	.
.	.
N	Number of type N requests

Device Channel Table (DCT)

The DCT is an STP-resident table. The only field used by SPM is DTCTA, the cumulative channel reserve time field.

Class Structure Definition Table (CSD)

The CSD is an STP-resident table containing all job class information.

4.11 JOB CLASS MANAGER (JCM)

Before a job enters the input queue, it must be given a job class assignment. The Job Class Manager Task (JCM) assigns a job to a class. JCM uses the job class structure currently in effect to determine the class assignment. See JCSDEF in the COS Operational Aids Reference Manual, publication SM-0044, for a detailed description of a job class structure.

After a system Install, the following default job class structure is in effect:

```
SNAME,SN=DEFAULT.
CLASS,NAME=JOBSERR,RANK=1,CHAR=JSE,RES=0,MAX=63.
CLASS,NAME=NORMAL,RANK=2,CHAR=ORPH,RES=0,MAX=63.
SLIMIT,LI=15.
```

4.11.1 JOB CLASS ASSIGNMENT

A job can belong to only one class. A job that qualifies for more than one class is assigned to the highest ranked class for which it qualifies. The user may override this assignment (to lower the class only) by using the CL parameter on the JOB control statement to specify the class in which the job is to be run. The job must still meet the qualifications of the specified class. If a job does not qualify for any class, it is assigned to the class defined using CHAR=ORPH (ORPH suggests orphan).

A JOB statement error occurs in the following cases:

- The job does not qualify for any class, and no class is defined using CHAR=ORPH.
- The user has overridden class assignment via the CL parameter on the JOB statement but the job does not meet the class qualifications of the specified class, or the specified class does not exist.
- The job is neither recoverable nor rerunnable during a system restart with recovery of rolled job selected.

Job class assignments are redetermined in the following cases:

- After a system startup, job classes are reassigned for all jobs that are in the input queue at the end of the startup. Jobs that are recovered are not affected.
- After a new structure is invoked, job classes are reassigned for all jobs in the input queue.

- After an operator uses the ENTER command to change a job's class, priority, time limit, TID, or DID, the job class of the specified job is determined if it is in the input queue.

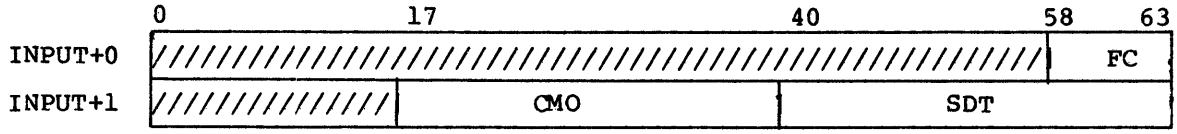
- After an operator uses the ROUTE command to change a DID, job classes are reassigned for all jobs in the input queue that had the original DID.

Once a job receives a JXT, its class assignment does not change unless the job is rerun. After a restart, jobs are either recovered, rerun, or marked by the system as having a JOB statement error. Recovered jobs maintain the class assignment they had before the system interruption.

4.11.2 INTERFACE BETWEEN JCM AND OTHER TASKS

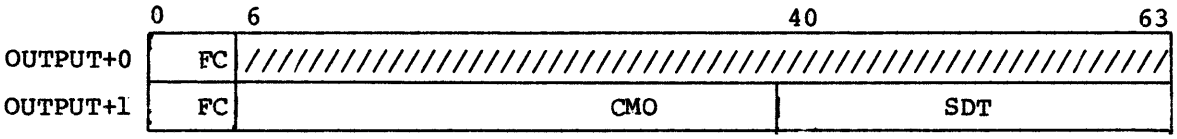
The Job Class Manager task is created with all other system tasks by the startup procedure. A task can call JCM by setting the appropriate input registers and calling PUTREQ and TSKREQ. JCM replies to each request by setting the appropriate output registers. See section 3.2 for a complete description of task communications.

INPUT register format:



Field	Word	Bits	Description
FC	INPUT+0	58-63	Function code
CMO	INPUT+1	17-39	Class Map Offset of the associated class
SDT	1	40-63	SDT address of the associated job

OUTPUT register format:



Field	Word	Bits	Description
FC	OUTPUT+0	0-5	Function code
FC	OUTPUT+1	0-5	Function code
CMO	1	17-39	Class Map Offset of the associated class
SDT	1	40-63	SDT address of the associated job

The requests that tasks can make to JCM are described on the following pages in the order listed in the following table:

Table 4.11-1. JCM functions

Function Code	Input Parameters	Function
CASS	SDT	Assigns specified job to a class
RCASS	SDT	Reassigns specified job to a class
ASSIT	SDT, CMO	Assigns specified job to a specified class
FXC		Fixes invoked class structure
FXCI		Fixes recovered class structure

Classify request

FUNCTION: Assigns a specific job to a class; called just before entering a job into the input queue for the first time.

FUNCTION CODE: CASS

ENTRY: SDT of specific job required

EXIT: Specified job is assigned to a class and the appropriate class-waiting-counts in the class structure are updated

Reclassify Request

FUNCTION:	Reassigns a specific job to a class; called when a job in the input queue has one of its job statement parameters (that may affect job classification) altered. The job must be removed from the input queue before JCM is called and returned to the input queue after JCM is called.
FUNCTION CODE:	RCASS
ENTRY:	SDT of specific job required
EXIT:	Specified job is removed from its current class. The appropriate class-waiting-counts in the class structure are updated; the job then reassigned to a class and the appropriate class-waiting-counts are updated.

Assign Request

FUNCTION:	Assigns a specific job to a specific class; called when the operator issues an ENTER,CL command to assign a job to a specific class.
FUNCTION CODE:	ASSIT
ENTRY	SDT of specific job and class map offset of specific class
EXIT:	Specific job is assigned to the specified class and the class-waiting-counts are updated in the class structure

Fixclass Requests

FUNCTION:	Fixes the class structure; called after a new class structure has been invoked and all waiting and allocated counts in the structure are 0.
FUNCTION CODE:	FXC
ENTRY:	Nothing required
EXIT:	All jobs in the input queue are reclassified, and all waiting and allocated counts in the class structure are determined.

FUNCTION:	Fix the class structure; called after a system recovery when all the waiting and allocated counts in the structure may not be 0.
FUNCTION CODE:	FXCI
ENTRY:	Nothing required
EXIT:	All waiting and allocated counts in the class structure are zeroed. Then all jobs in the input queue are reclassified, and all waiting and allocated counts in the class structure are determined.

4.12 OVERLAY MANAGER (OVM)

The Overlay Manager Task (OVM) controls the loading and unloading of task overlays. Loading an overlay means locating the required disk resident portion of COS and reading it into a predefined area of memory for execution. Unloading an overlay simply means to mark the overlay area occupied by the overlay as being available for use by another overlay.

4.12.1 TASK COMMUNICATION WITH OVM

Each task which uses overlays must define an overlay area to be used for reading and executing overlays related to that task. OVM keeps track of which overlay areas are currently in use and which overlays currently are memory resident. An overlay is not reloaded from disk if it is currently in memory unless it has requested that reuse of the overlay be disabled.

While a user job is waiting for EXP to process a request and EXP must load an overlay, a job is marked "I/O Suspend" to force it to be disconnected, but remain in memory until OVM signals to EXP that the overlay load is complete. When the load is complete, EXP issues an I/O Done request. Similar actions must be taken by any other task that requires an overlay load as a result of a job-related request.

A task communicates with OVM by a PUTREQ/GETREPLY sequence. The request contains a function code, telling OVM which of five functions is to be performed. The five functions recognized by OVM include:

- Initial overlay load
- Overlay Call request
- Overlay Return request
- Overlay Goto request
- Overlay Re-use Disable request

Each function is accompanied by parameters which are part of the request.

Initial load overlay request

When a task determines that an overlay load is required to process some event and the determination is made by a non-overlay portion of the task, an "Initial Overlay Load" request must be made. For this request, OVM does not begin loading until the task overlay area is available. If an Initial Overlay Load request is received while the overlay area is marked as busy, it is entered into a list of outstanding requests. OVM does not send a reply to the calling task until the request has been processed and the overlay is available.

OVM maintains a stack of initial overlay load requests for each task. This allows EXP to request an overlay load in response to a user job function request and then to request another overlay load in response to a request from another job without having to wait until the initial overlay load has completed. In general, requests are processed in the order in which they are received, with the exception that, if two or more requests are for the same overlay, the second and subsequent such requests are honored as soon as the first completes, even though there may have been different intervening requests.

The overlay IDs are defined at installation.

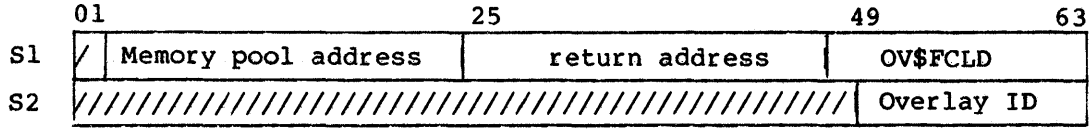
If an error occurs which prevents honoring a request, an error is returned to the caller. The only currently defined one is:

Code	Significance
OV$ECNS	No room in the overlay load list

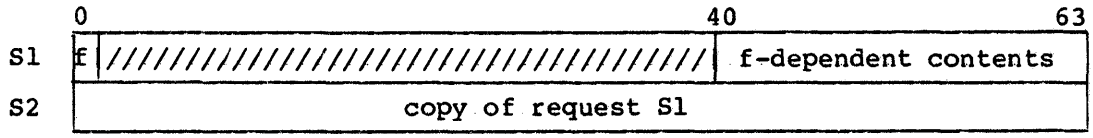
Input required by OVM for an Initial Overlay Load request consists of the overlay ID. This ID consists of an overlay identifier.

LOADOVL macro

Request:



Reply:



f	Significance
1	Bits 40-63 contain error code
0	Bits 40-63 contain address to which control is transferred

Call an overlay and goto an overlay requests

When an overlay is executing and determines that a different overlay is needed to perform some function and the second overlay is associated with the same task, either an Overlay Call request or an Overlay Goto request must be made. An Overlay Call request is used if the calling overlay wants control returned to it when the called overlay completes; an Overlay Goto request is used if control is not to return to the caller.

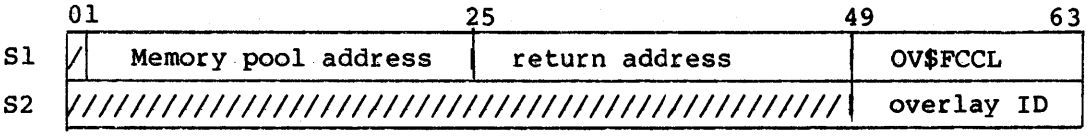
When either request is made, the new overlay is read into the overlay area on top of the calling overlay. If the calling overlay wants to save data, or wants to pass data to the called overlay, it must place the data in a memory pool area. OVM passes that address to the called overlay in S2, the reply word. The content or format of the data is determined by the calling task and OVM does not use or examine any such information.

Where an Overlay Call request is made, OVM saves the caller ID, caller-supplied return address, and caller-supplied memory pool address. The memory pool address is made available to the called overlay. When the called overlay completes execution, the calling overlay is reloaded, reply word S2 contains the supplied memory pool address, and OVM restarts the caller at the supplied return address. For an Overlay Goto request, the information about the caller is not saved and when the called overlay terminates, OVM restarts the last overlay that made an Overlay Call request, or returns to the task main loop if no such request is outstanding. The caller information is saved in a first-in-last-out (FILO) stack.

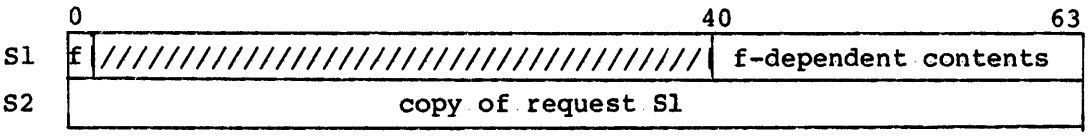
Input to OVM for an Overlay Call request is the desired overlay ID, the return address, and a memory pool address. For an Overlay Goto request, the return address is not used.

CALLOVL

Request:



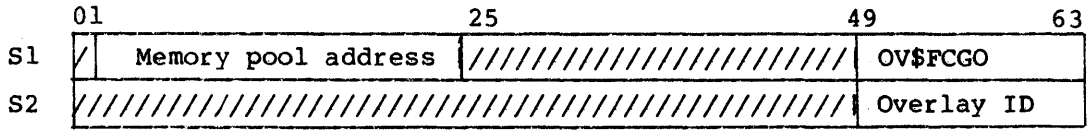
Reply:



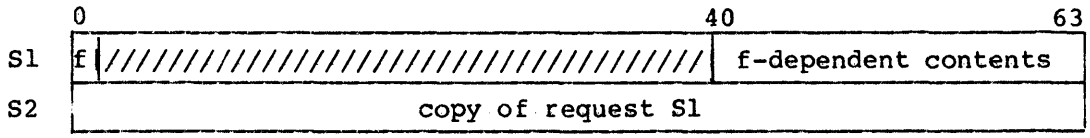
f	Significance
1	Bits 40-63 contain error code
0	Bits 40-63 contain address to which control is transferred

GOTOOVL macro

Request:



Reply:



f	Significance
1	Bits 40-63 contain error code
0	Bits 40-63 contain address to which control is transferred

Inhibit overlay reuse

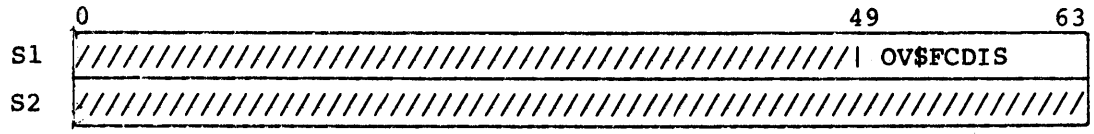
The Overlay Reuse disable request signals OVM that the current overlay must be reloaded before it can be used again. This is used only if some change has been made in code or some internal table requiring reinitialization that cannot be restored by the overlay.

OVM sets a flag in its internal tables signaling that any Initial Load Request that references the current overlay must be satisfied by going back to the disk copy.

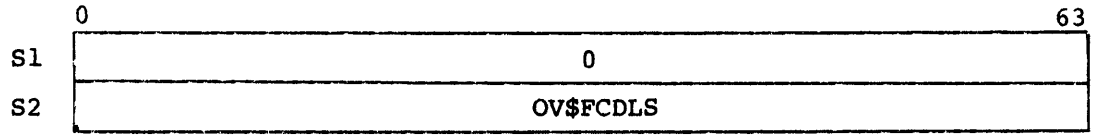
No input is required. No status is returned in the reply.

Disable macro

Request:



Reply:



Return to caller overlay

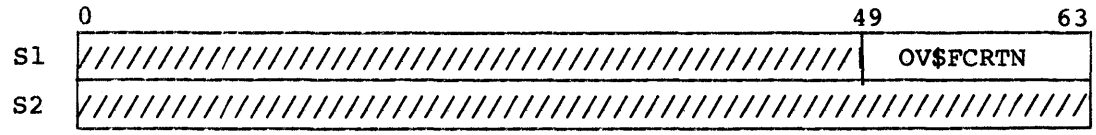
An overlay uses an Overlay Return Request to signal OVM that it has completed execution and may be unloaded. When this request is received, OVM determines which overlay, if any, should be loaded next. This is done by the following process:

- Take the last entry made in the Overlay Call stack, and restart the specified overlay.
- If the Call Stack is empty, OVM examines the Load Request List for any load request which needs the currently loaded overlay. If one is found and the current overlay is reuseable, OVM restarts the current overlay.
- If no load request needs the current overlay or the overlay is not reuseable, OVM takes the oldest unprocessed entry from the Load Request List and loads the specified overlay.
- If no unprocessed entry exists in the Load Request List, OVM returns to the main loop of the associated task.

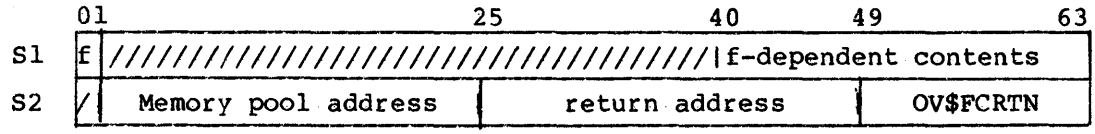
Other than the return function code, no input is required. When no Call Stack entries exist, the reply to the calling task contains a copy of the S1 from the Initial Load request which initiated the sequence of overlays which is now terminating. This copy of S1 is in S2.

RTNOVL macro

Request:



Reply:



f	Significance
1	Bits 40-63 contain error code
0	Bits 40-63 contain address to which control is transferred

The addresses are from the previous CALLOVL, LOADOVL, or GOTOOVL

4.12.2 SYSTEM GENERATION/OVERLAY DEFINITION

Portions of tasks which are intended to be overlays must be identified as such during system assembly so that the address references can be properly defined by the assembler. Macros are provided to define the limits of overlay code. These macros are used to construct the initial Overlay Directory, and to ensure code is physically located at the end of STP. The CAL LOC and BLOCK pseudo-ops are used to control addressing and physical placement of code. The macros provided are described in the COS Operational Aids Reference Manual, publication SM-0044.

4.12.3 OVERLAY CALLING MACROS

Macros are provided to generate jumps to each task's resident overlay loader. The task resident overlay loader is merely a subroutine within each task which actually makes the call to OVM. The macros set up the OVM parameters. The purpose of the task resident overlay loader is to provide one place within a task to perform the PUTREQ and process replies from OVM. Entry to the task resident overlay loader is via return jump (R) instructions. Macros are provided to generate each of the OVM function requests. When a reply is received from OVM, S1 contains the address to which the resident loader should transfer control or an error code. The macros are described in the COS Operational Aids Reference Manual, publication SM-0044.

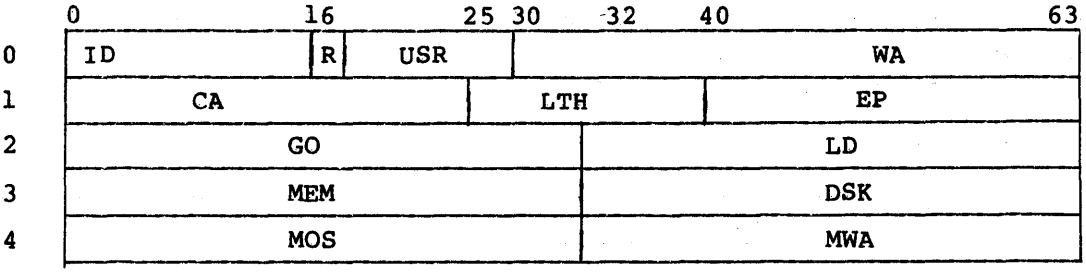
4.12.4 OVM TABLES

The Overlay Manager task uses certain tables which it uses to control the loading and unloading of overlays. These tables include all necessary stacks, the DNTs needed for the I/O and the Overlay Directory Table (ODT). Each task has an Overlay Control Table (OCT). Information in each table is as described below.

Overlay Directory Table (ODT)

The ODT is constructed partially during system assembly and partially during Startup. During assembly, the overlay id, memory address, entry point address, and length in 64-bit words are inserted. Startup converts the memory address to a word address within the overlay dataset. A residence flag is provided to allow for future use of MOS memory on the I/O Subsystem as an overlay storage medium. There is one entry in the ODT for each overlay.

Format of an entry:



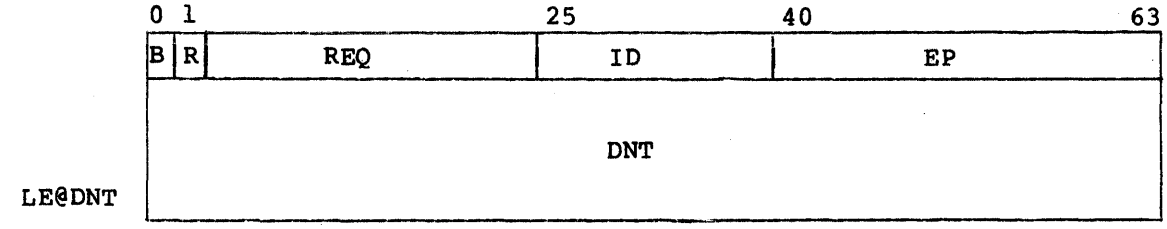
Field	Word	Bits	Description
ID	0	0-15	Installation - defined overlay ID
R	0	16	Residence: 0=disk; 1=MOS
USR	0	17-29	Count of current users if re-entrant overlay$
WA	0	30-63	Address - word address in dataset
CA	1	0-24	Count of CALLOVL requests for this overlay
LTH	1	25-39	Length of overlay in words
EP	1	40-63	Entry point within overlay for transfer of control
GO	2	0-31	Count of GOTOOVL requests for this overlay

Field	Word	Bits	Description
LD	2	32-63	Count of LOADOVL requests for this overlay
MEM	3	0-31	Count of times overlay was already available
DSK	3	32-63	Count of times overlay was loaded for disk
MOS	4	0-31	Count of times overlay was loaded for MOS
MWA	4	32-63	MOS word address of overlay

Overlay Control Table (OCT)

The OCT contains flags and pointers to determine current status of the overlay area for the associated task. The OCT is constructed and used only by OVM. If the task does not use overlays its OCT contains zeroes. There is one Overlay Control Table for each task.

Format of an entry:



Field	Bits	Description
B	0	Busy; if 1, overlay area in use
R	1	Reusable; if 1, overlay issued DISABLE call
REQ	2-24	Request; count of OVM requests from this task
ID	25-39	ID; current overlay in area if B is nonzero
EP	40-63	Entry; entry point of current overlay

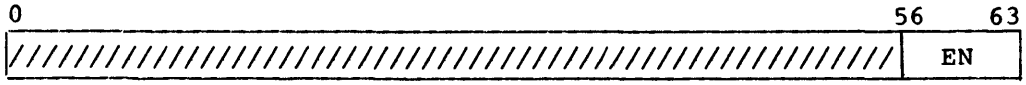
§ Deferred implementation

The DNT following the 1-word entry reads all disk resident overlays for the associated task. Separate DNTs allow more than one task to be loading overlays concurrently. All DNT entries point to the single copy of the DAT constructed by Startup.

Overlay Call Stack (OCS)

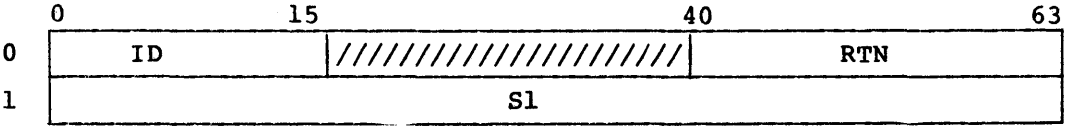
One Overlay Call Stack containing ten entries is maintained for each task. A 1-word header contains control information for the stack.

Format of the header:



Field	Bits	Description
EN	56-63	Number of entries in use. Stack is empty when EN=0

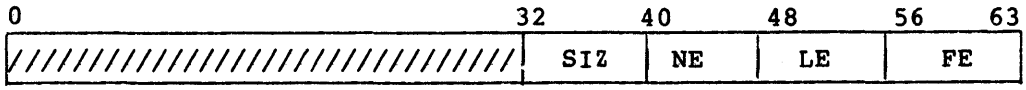
Format of each entry:



Field	Word	Bits	Description
ID	0	0-15	Calling overlay ID
RTN	0	40-63	Return address
S1	1	0-63	Copy of S1 from CALLOVL request

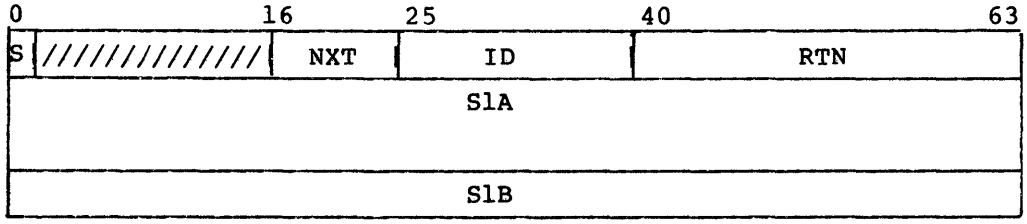
Overlay Load Request List (OLL)

The Overlay Load Request List tracks initial load requests occuring while the overlay area is busy. The first initial load request is also kept. A 1-word header controls the list. Each entry is linked for ease in adding or deleting requests, not necessarily in order of receipt. Format of the header is:



Field	Bits	Description
SIZ	32-39	Number total entries in list (space reserved)
NE	40-47	Number entries in use
LE	48-55	Last entry in list in use
FE	56-63	First entry in list in use

Format of an entry:



Field	Bits	Description
S	0	Submitted (loading initiated or overlay executing if nonzero)
NXT	16-24	Next entry in use in order load will occur
ID	25-39	Installation-defined overlay ID
RTN	40-63	Return address within calling task
S1A	0-63	Copy of S1 from PUTREQ/TSKREQ call
S1B	0-63	Copy of S1 from most recent CALLOVL/GOTOOVL request during the sequence initiated by this LOADOVL request

CONTROL STATEMENT PROCESSOR (CSP) 5

The Control Statement Processor (CSP) is a system program that executes in the user field. CSP initiates the job, analyzes and stores the various elements of the control statements (that is, cracks them), processes system verbs, advances the job step by step, processes errors, and ends the job.

5.1 SYSTEM TABLES USED BY CSP

CSP uses system tables to communicate with STP tasks and system or user-supplied programs. These tables are located in the user field or in JTA and are preserved or updated by CSP, STP, or other programs during the duration of the job. The tables CSP uses are:

JCB Job Communication Block
LFT Logical File Table
DSP Dataset Parameter Area
DNT Dataset Name Table

Detailed information for these tables is available in the COS Table Descriptions Internal Reference Manual, publication SM-0045.

5.1.1 JOB COMMUNICATION BLOCK (JCB)

The Job Communication Block (JCB) is the first 200_8 words of the user field. CSP places the current control statement in cracked and uncracked format and information for accessing or building I/O buffers, DSPs, and LFT in the JCB. Other information can also be maintained there.

5.1.2 LOGICAL FILE TABLE (LFT)

The Logical File Table (LFT) is located at the address specified in the JCLFT field of the JCB. CSP makes entries for $OUT, $IN, and any other datasets or aliases known to the job. The LFT entries point to the DSPs.

5.1.3 DATASET PARAMETER AREA (DSP)

The Dataset Parameter Area (DSP) contains dataset status and information needed for I/O. It begins at the address specified in the JCDSP field of the JCB. Individual DSP addresses are specified in the LFT. CSP makes a DSP for $OUT, $IN, and any other datasets known to the job.

5.1.4 DATASET NAME TABLE (DNT)

The Dataset Name Table (DNT) is located in the JTDNT field of JTA. STP creates a DNT entry as a result of an F$DNT system call by CSP. All datasets known to the jobs have a DNT. Each DNT entry contains dataset status, information for identifying the dataset, and pointers to other tables. The information required for the system call is supplied by CSP from arguments in the JOB statement, ASSIGN statment, or from default lists. This information is put in the format of a Dataset Definition List (DDL) described with the F$DNT system call.

5.2 THEORY OF OPERATION

The CSP binary is loaded during system generation and is copied to the user field when the job is initiated. The job's control statements are passed from the first file of the dataset *jobname* which is on the disk, to CSP. Exits are by normal exchange sequence. CSP initiates the job, cracks control statements, processes some system verbs, advances the job step by step, processes errors, and ends the job.

5.2.1 CSP LOAD PROCESS

A copy of the CSP binary is appended at the end of STP during system generation. Following deadstart, this copy resides in the system resident memory with EXEC and STP. When the job is submitted, JSH allocates memory for the job, sets up the JTA, and causes the Exchange Processor to copy CSP into the user field at location (BA)+200_8. Following loading, CSP is ready to process a control statement. Any user program called as a result of processing the control statement is loaded over CSP. When the user program ends, CSP is again loaded into the user field. Since CSP executes in the user field, it is subject to roll-in/roll-out procedures the same as a user program.

CSP executes as a user program and shares the user exchange package, JTA, JCB, LFT, DSPs, and I/O buffers with user programs. CSP may, however, make some requests of STP not allowed by a user program.

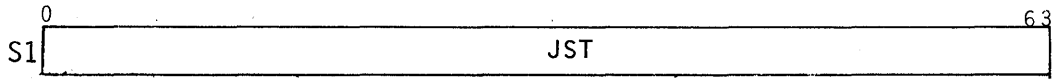
5.2.2 ENTRY AND EXIT CONDITIONS

CSP assumes that a control statement file for the job has been staged from the front-end processor to the CRAY-1 disk as the first file of the dataset *jobname*. Control statements are passed by STP from the disk to CSP using the control statement buffer (CSB) of the JTA as the input buffer.

Also, if it is the first time into CSP, register S7 will contain a status from the job's input SDT entry. This status determines which log message, if any, is to be issued by CSP immediately following the processing of the JOB control statement. It may also cause CSP to terminate the job immediately. This status is used by recovery of rolled jobs to inform users of jobs that were rerun by system recovery or that could not be recovered and were also not rerunnable.

Entry condition

Every time CSP is loaded into the user field, STP passes exchange status to CSP in S1 as follows:



Field	Description
JST	Job status: 1 Job initiation 0 Subsequent normal exchange

(S1) is tested to see if this is the first time into CSP. If it is, the JOB control statement must be the first control statement encountered.

Exit conditions

Exits from CSP are by the normal exchange sequence described in section 4.4. The exit and re-entry conditions for the system calls are documented with the system task calls. Re-entry from most calls is to the instruction following the EX instruction. One exception, which is important for the functioning of CSP, is the F$TRM calls.

When CSP makes an F$TRM call, it has prepared all output datasets for return to the front-end processor and passes no other values to STP. CSP is never again re-entered for that job.

5.2.3 BEGIN JOB

CSP begins a job by first opening the user logfile and entering a headline message. Next, it processes the JOB control statement, which must be the first statement in the control statement file. The job parameter values are set according to the arguments in the JOB statement. LFT entries, DSPs, and I/O buffers are made for $OUT and $IN. Aliases FT05 and FT06 in the LFT are created for $IN and $OUT, respectively.

Depending on the status that EXP passed to CSP in S7, a message may be written to the user and system logs, immediately following the JOB card. Sometimes this message represents a fatal error from Startup, in which case CSP terminates the job immediately.

5.2.4 CRACK STATEMENTS

CSP makes a request to STP to place one control statement in the JCCCI field of the JCB and in the logfile. CSP then cracks the statement into verb, separators, keywords, and values. It places the cracked statement in the JCCPR field of the JCB. The cracked format is described in the CRAY-OS Version 1 Reference Manual, publication SR-0011. In the cracked format, the parameter keywords and values are available for processing by CSP, by system-supplied programs, or by user-supplied programs.

5.2.5 PROCESS STATEMENTS

Every statement is a user's request for some action and has either a system verb or a dataset verb. System verbs are processed by the system (CSP and/or STP). The verbs processed in part by CSP are:

*	ASSIGN	EXIT	RELEASE	SWITCH
ACCESS	CALL	MODE	REWIND	
ACQUIRE	DELETE	MODIFY	RFL	
ADJUST	DISPOSE	PAUSE	SAVE	

Dataset verbs are processed by loading a program into the user field and then executing it.

System calls

In processing control statements, CSP makes frequent system calls to STP. These calls, which are described in section 4.4, are:

F$DAT	F$OPN	F$GNS	F$ACT	F$RLS
F$MSG	F$MEM	F$PDM	F$CSW	F$SPS
F$TRM	F$DNT	F$GRN	F$AQR	F$ASD
F$SSW	F$MDE	F$DIS	F$ABT	F$PRC

Parameters

Job processing requires assigning values to many parameters. Most have default values; some have a second default value called a keyed value. Other parameters have no default values and require that the user specify a value. The default values for a given control statement are contained in a default list. Also included in the list are keywords and the destination address for the final value.

CSP sets the parameter values by using the statement keyword to locate the default list entry, taking the user-specified value if the keyword is equated to a value, taking the keyed value if the keyword stands alone, or taking the default value if the keyword does not appear in the control statement. The value is then entered at the address specified in the default list. If there is no default value or keyed value, the sign bit is set in these words. If the user does not specify a value, the sign bit remains and an error results.

5.2.6 ADVANCE JOB

A job step is the action taken as a result of a control statement. CSP advances the job, step by step, in the sequence specified by the control statement file. These steps are summarized under CSP Step Flow.

5.2.7 ERROR EXIT PROCESSING

As CSP advances the job, it alters the normal sequence from that of the control statement file if an error occurs. If an error is detected in the JOB statement, all other statements in the control file are ignored and the job is not processed. If the error occurs for any other control statement, CSP makes an ABORT system call. The Exchange Processor Task then performs the control statement processing.

Once an error is encountered, all statements are skipped until an EXIT control statement is found in the control statement file. If an EXIT is found, job processing resumes with the control statement after the EXIT statement; however, the statements after EXIT are processed only after an error; they are ignored if no error occurred.

The job is ended if there is no EXIT control statement in the remainder of the control statement file.

5.2.8 END JOB

Every job goes through the end job procedure whether or not an error occurs. End of job and job accounting messages are placed in the logfile. $LOG and CSP make a job termination system call, F$TRM. The Exchange Processor Task completes job termination. $LOG is written to $OUT; $OUT is closed; all buffers are flushed; the name of the $OUT dataset is changed to the jobname; and the dataset is routed to the front-end processor. Finally, the user area is released to the system.

5.3 CSP STEP FLOW

The system processes jobs and errors as outlined below:

General Step Flow

1. Load CSP into the user field.

2. Begin job with JOB statement; open logfile; enter headline and JOB statement into logfile; set job parameters; and set up $IN and $OUT DSPs and buffers.

3. Issue any logfile message required by EXP status code. Terminate job immediately if status indicates fatal error.

4. Get next statement and enter statement into logfile; if a CSP verb is encountered, process it; if not, attempt to locate a local dataset with the matching name. If one is found, load it into memory and execute it. If a local dataset exists with a matching name, attempt to locate a dataset in the System Directory with a matching name. If one is found, load and execute it. If no such dataset is in the System Directory, abort the job. When the system or user program ends, CSP is reloaded for the next statement.

5. End job when next statement is EXIT or there are no more statements.

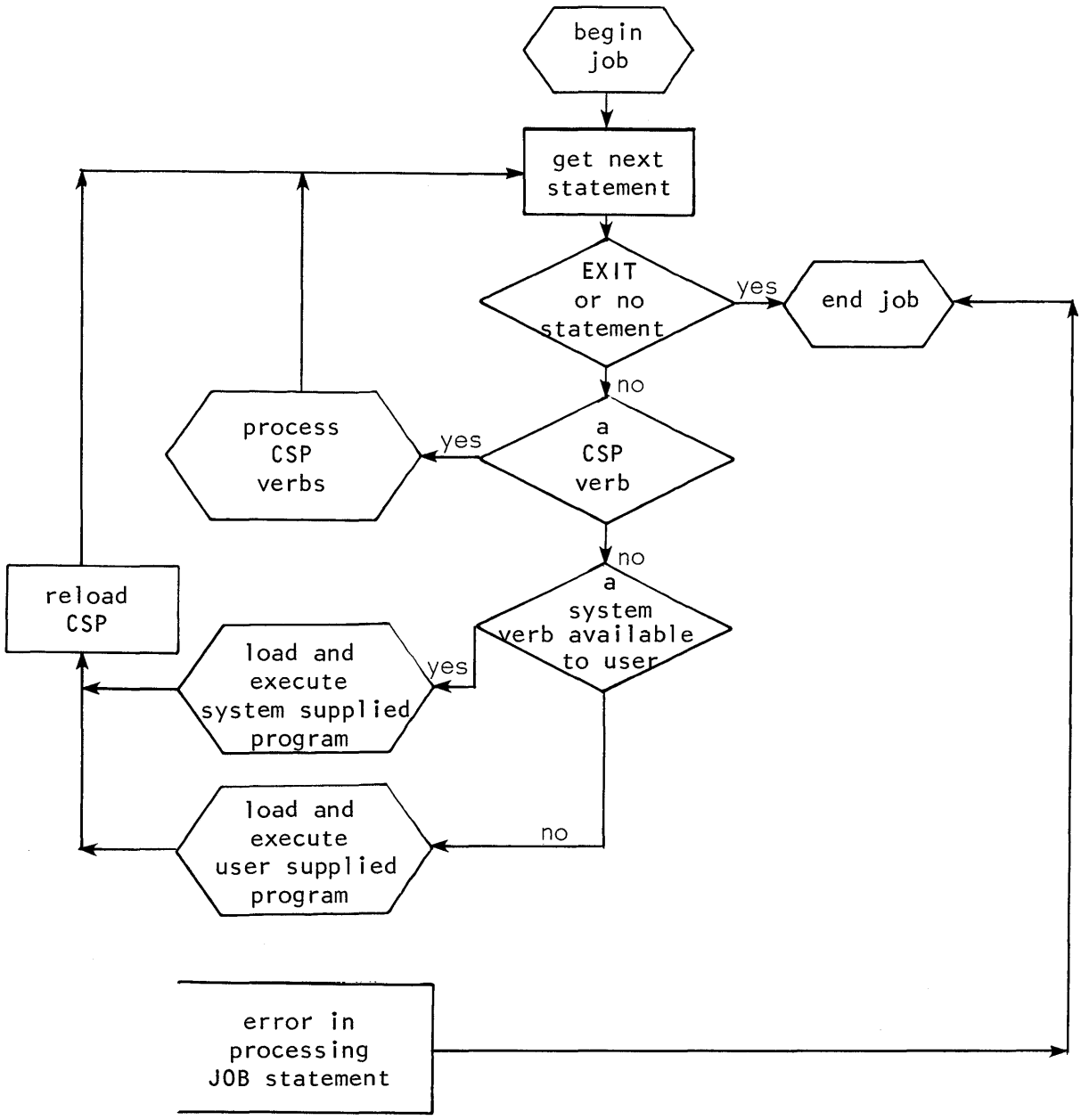


Figure 5-1. CSP general flow diagram

Error Processing Step Flow

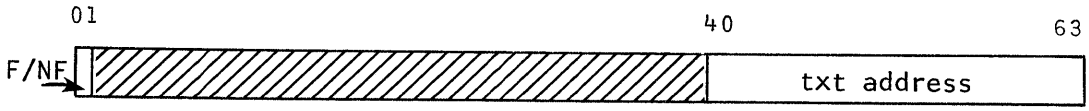
1. If first control statement is not JOB or if there is any error in the JOB statement, enter error message into logfile and then end the job.

2. If an error is detected while processing any other verb (CSP, system, or user), enter an error message into the logfile and abort the job.

5.4 RECOVERY STATUS MESSAGES

CSP may issue messages immediately following the JOB control statement. These messages are described in the CRAY-OS Message Manual, publication SR-0039. They are issued in response to a status code sent by EXP on the first entry into CSP. This status code acts as an index into a table of message control words. If the message control word indicates that the status is fatal, CSP ends the job immediately; if it indicates that the status is non-fatal, CSP continues normally.

Message control word format



Field	Description
F/NF	Flag 1 Fatal 0 Nonfatal
txt address	Address of message text. The message must be terminated by a zero byte.

READERS COMMENT FORM

CRAY-1 COS EXEC/STP/CSP Internal Reference Manual SM-0040

Your comments help us to improve the quality and usefulness of our publications. Please use the space provided below to share with us your comments. When possible, please give specific page and paragraph references.

NAME _____

JOB TITLE _____

FIRM _____

ADDRESS _____

CITY _____ STATE _____ ZIP _____

CRAY RESEARCH, INC.

CUT ALONG THIS LINE

FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY CARD

FIRST CLASS PERMIT NO 6184 ST PAUL, MN

POSTAGE WILL BE PAID BY ADDRESSEE

CRAY RESEARCH, INC.

Attention:
PUBLICATIONS

1440 Northland Drive
Mendota Heights, MN 55120
U.S.A.

FOLD

STAPLE

READERS COMMENT FORM

CRAY-1 COS EXEC/STP/CSP Internal Reference Manual SM-0040

Your comments help us to improve the quality and usefulness of our publications. Please use the space provided below to share with us your comments. When possible, please give specific page and paragraph references.

NAME _____

JOB TITLE _____

FIRM _____

ADDRESS _____

CITY _____ STATE _____ ZIP _____

CRAY RESEARCH, INC.

CUT ALONG THIS LINE

FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY CARD

FIRST CLASS PERMIT NO 6184 ST PAUL, MN

POSTAGE WILL BE PAID BY ADDRESSEE

CRAY RESEARCH, INC.

Attention:
PUBLICATIONS

1440 Northland Drive
Mendota Heights, MN 55120
U.S.A.

FOLD

STAPLE

Cray Research, Inc.
Publications Department
1440 Northland Drive
Mendota Heights, MN 55120
612-452-6650
TLX 298444