CRAY
RESEARCH, INC.

CRAY-1® AND CRAY X-MP COMPUTER SYSTEMS

COS
EXEC/STP/CSP
INTERNAL REFERENCE
MANUAL

SM-0040

CRAY RESEARCH, INC.

CRAY-1® AND CRAY X-MP COMPUTER SYSTEMS

COS EXEC/STP/CSP INTERNAL REFERENCE MANUAL

SM-0040

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.
01	July, 1981 - This change packet reflects the feature changes made to COS for the 1.10 release, including changes to JCL, disk flaw processing, partial deallocation, and the Network Systems Corporation HYPERchannel feature. Other minor technical and editorial changes are also included.
A	August, 1981 - This printing incorporates change packet 01. No other changes have been made.
A-01	June, 1982 - This change packet describes the new Tape Queue Manager (TQM) task; substantial changes to the Disk Queue Manager (DQM) task, the Overlay Manager (OVM) task, and EXEC; and other minor technical and editorial changes to bring this publication into agreement with the 1.11 version of COS.
B	July, 1983 - This rewrite describes the new Stager (STG) task; substantial changes to the System Executive (EXEC), Job Scheduler (JSH), and Tape Queue Manager (TQM); other changes required to bring this publication into agreement with the 1.12 version of COS. Numerous editorial changes have been made to enhance readability. This printing obsoletes all previous printings.
C	February, 1984 - This reprint with revision reflects the feature changes made to COS for the 1.13 release, including multitasking support, volatile device support, and tape positioning. This printing obsoletes all previous printings.

PREFACE

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

This publication is part of a set of manuals that describes the internal design of the Cray Operating System (COS) and its product set.

Manuals in this set that describe the internal design of COS and other software products from Cray Research, Inc. (CRI), are as follows:

SM-0017	FORTRAN (CFT) Internal Reference Manual
SM-0040	COS EXEC/STP/CSP Internal Reference Manual
SM-0041	COS Product Set Internal Reference Manual
SM-0045	COS Table Descriptions Internal Reference Manual†
SM-0046	IOS Software Internal Reference Manual
SM-0049	Data General Station (DGS) Internal Reference Manual
SM-0072	COS Simulator (CSIM) Internal Reference Manual

Manuals that define procedures and external features of tools needed for installing and maintaining CRI software are as follows:

SM-0043	COS Operational Procedures Reference Manual
SM-0044	COS Operational Aids Reference Manual
SR-0073	COS Simulator (CSIM) Reference Manual

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

† This manual is distributed on magnetic tape and can be obtained through your Cray Research analyst.

CONTENTS

APPENDIX SECTION

FIGURES

FIGURES (continued)

TABLES

GLOSSARY

INDEX

INTRODUCTION 1

1.1 GENERAL DESCRIPTION

CRAY-OS (COS) is a multiprogramming operating system for the Cray Computer System. The operating system provides 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 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 or CRAY X-MP Central Memory, on system mass storage, and in the I/O Subsystem (on some models). (Startup is the process of bringing the computer and operating system to an operational state.)

Jobs are presented to the Cray mainframe 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.

Cray Research, Inc., products, the FORTRAN compiler, the CAL assembler, 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 executable form and (2) source language programs in the form of UPDATE program libraries. UPDATE is a system program used to maintain programs and other data on permanent datasets. See the UPDATE Reference Manual, CRI publication SR-0013.

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

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

1.2 SOFTWARE CONFIGURATION

The Cray computer 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, CRI publication SM-0046.

1.2.1 CRAY OPERATING SYSTEM (COS)

The Cray Operating System (COS) consists of memory resident and mass storage resident programs that

- Manage resources,
- Supervise job processing, and
- 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 a Maintenance Control Unit (MCU), which can be an I/O Subsystem. A job can 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.

COS consists of the following modules that execute on the mainframe central processing unit(s) (CPUs) (figure 1-1):

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.

The Control Statement Processor (CSP), described in section 20, is responsible for interpreting all job control statements and for either performing the requested function or making the appropriate system request. An installation option specifies whether an image of CSP resides after the STP area in memory or whether it resides on disk. In either case, it is copied into a user field for execution.

Utility programs (described in the COS Product Set Internal Reference Manual, publication SM-0041) include the loader (LDR), a library generation program (BUILD), a source language maintenance program (UPDATE), permanent dataset utility programs, copy and positioning routines, and so on.

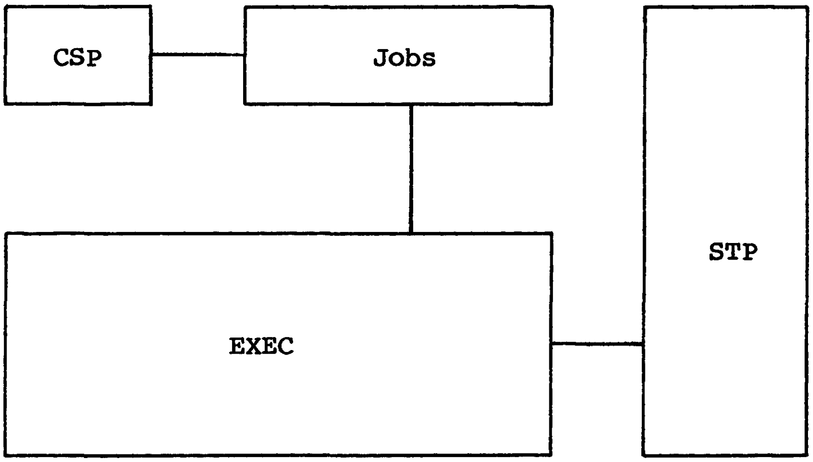


Figure 1-1. Elements of CRAY-OS

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

1.2.2 LANGUAGE SYSTEMS

Currently, five language systems developed by Cray Research, Inc., are provided for the Cray Computer System. They are the FORTRAN compiler (CFT), the Cray Assembly Language program (CAL), the Pascal compiler, the SKOL macro translator, and A Programming Macro Language (APML) for the I/O Subsystem.

FORTRAN compiler

Developed in parallel with the Cray Computer System, the Cray Research, Inc., FORTRAN compiler is designed to take advantage of the vector capability of the various computers.

The compiler itself determines the need for vectorizing and generates code accordingly, removing such considerations from the programmer. Optimizing routines examine FORTRAN source code to see if it can be vectorized. The compiler conforms with ANSI FORTRAN 77 standards.

A description of the design of the compiler is outside the scope of this publication, but is included in the Cray FORTRAN (CFT) Internal Reference Manual, publication SM-0017.

CAL assembler

The CAL assembler provides users with a means of expressing all hardware functions of the CPU symbolically. Augmenting the instruction repertoire is a set of versatile pseudo instructions that provides 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 computers. Much of the operating system as well as other software provided by Cray Research, Inc., is coded in CAL.

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

Pascal compiler

The Cray Research, Inc., Pascal compiler supports the International Standards Organization (ISO) Version 1 Pascal standard. Cray Pascal also includes extensions to the ISO standard. The compiler optionally issues messages identifying these extensions to help transport a program to a machine running a different implementation of the language.

The Pascal Reference Manual, CRI publication SR-0060, describes the language and notes all Cray Research, Inc., extensions. The Pascal Internal Reference Manual, CRI publication SM-0061, describes the design of the compiler.

APML assembler

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

APML has a 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. In addition, 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, CRI publication SM-0036.

SKOL macro translator

SKOL, a high-level programming language that stresses readability and extensibility, offers the user a well structured language while retaining the power and efficiency of the CFT compiler. 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 individual needs. By inserting macros directly into the SKOL source program, the programmer can define changes in the language for a specific run.

SKOL is described in the SKOL Reference Manual, CRI publication SR-0033.

1.2.3 LIBRARY ROUTINES

Cray software includes a group of subprograms that are callable from user programs. These subprograms reside in the $FTLIB, $PSCLIB, $SYSLIB, $ARLIB, $IOLIB, $UTLIB, and $SCILIB libraries. They are grouped by UPDATE deck name within each library. The subprograms are divided among the libraries on a functional basis.

1.2.4 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 are not described in this publication.

1.3 SYSTEM RESIDENCE

The system components reside in areas of memory defined during startup (section 5). 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.

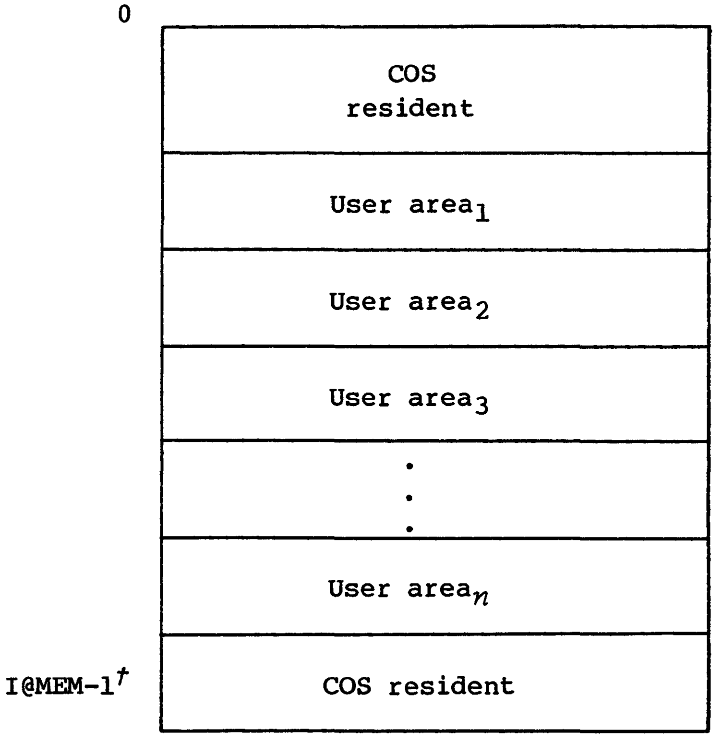


Figure 1-2. Memory assignment

Figure 1-2 illustrates the general contents of memory following startup. Figure 1-3 illustrates the general layout of a user area at job initiation. Figure 1-4 itemizes the memory resident portions of the operating system.

[†] Installation parameter that defines maximum memory in words

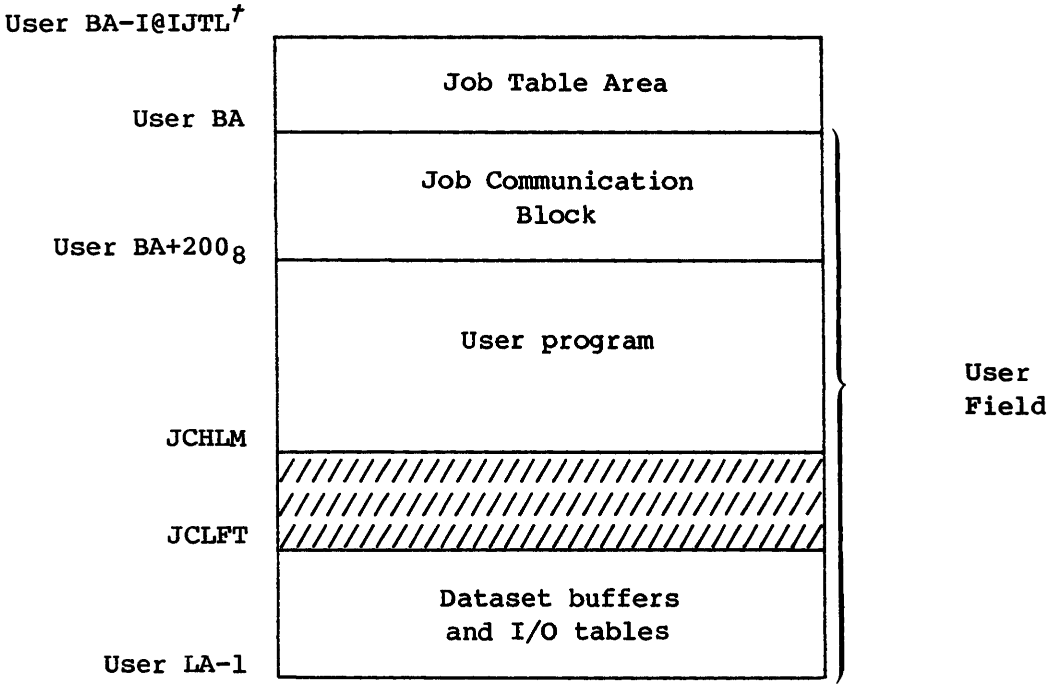


Figure 1-3. Expansion of a user area

1.3.1 EXEC CONSTANT, DATA, AND TABLE AREAS

The EXEC constant area contains all EXEC constants. The constants are functionally grouped, and include:

Constant memory locations
Front-end Driver constants
Packet I/O Driver constants

The EXEC data area contains all EXEC data not in the form of tables. The data in this area is functionally grouped, and includes:

Initial and warm-boot exchange packages (at location 0)
Space reserved for DDC (SYSDUMP utility)
Identification (at location 1400 octal)
Pointers to EXEC tables
Stop message buffer
X-MP cluster register dump area

† This value is correct at job initiation and until JTA expansion occurs.

Disk/SSD Driver data
Packet I/O Driver data
Front-end Driver data
Miscellaneous data
EXEC messages

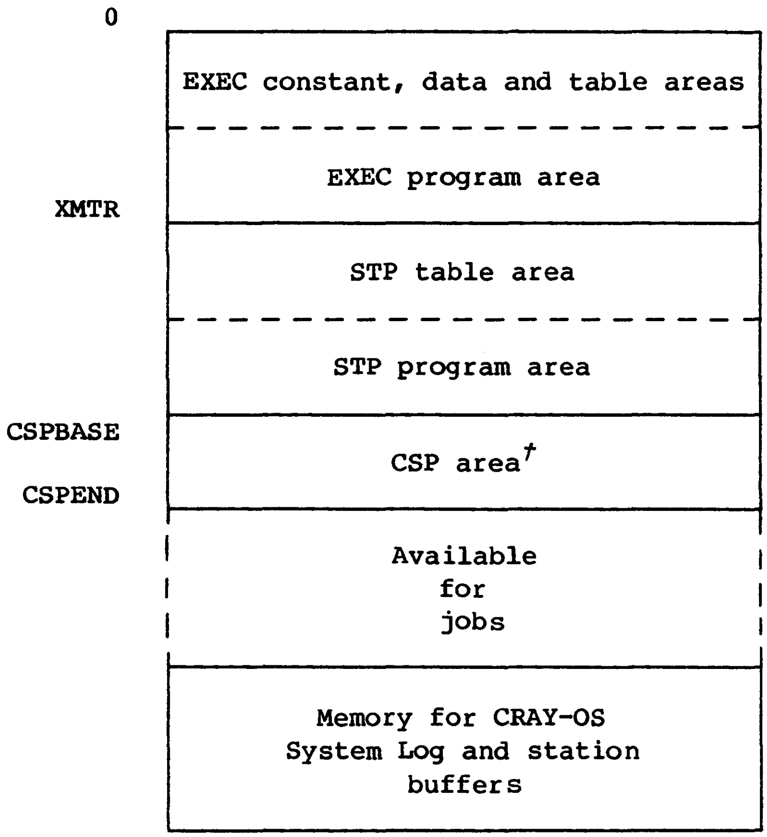


Figure 1-4. Expansion of COS resident

The EXEC table area contains all EXEC tables, alphabetically ordered. Most table layouts are described in the COS Table Descriptions Internal Reference Manual, CRI publication SM-0045. The other tables are internally documented. The tables are:

† This area is available for jobs if CSP resides on disk.

CAT	Channel Address Table
CBT	Channel Buffer Table containing one entry of working storage for each disk driver channel.
CHT	Channel Table containing a 1-word entry for each side (input and output) of a physical channel. An entry contains a pointer to the Channel Processor Table 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.
CLT	Channel Limit Table
CXT	Channel Extension Table
FIQ	Free input packet queue
FOQ	Free output packet queue
ICT	Interrupt Count Table
IHT	Interrupt Handler Table
MCT	Monitor Count Table
MEL	Memory Error Log Table
MRT	Monitor Request Table
PWS	Processor working storage
RMS	Read Margin Select Table
SCT	Subsystem Control Table
STT	System Task Table consisting of three parts: a header, a task parameter word area, and an exchange package area
STX	System Task Exchange Package Table
TBT	Task Breakpoint Table
TET	Time Event Table
XFT	History Function Table
XTT	History Trace Table

1.3.2 EXEC PROGRAM AREA

Included in the area occupied by the System Executive (EXEC) 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 base address (BA) of 0 and a limit address (LA) equal to the installation parameter I@MEM. EXEC is described in section 2 of this manual.

1.3.3 SYSTEM TASK PROCESSOR (STP) TABLE AREA

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

AUT Active User Table containing an entry for each logged on interactive user

CMCC Communication Module Chain Control for controlling 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 in 6-word groups 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.

CNT Configuration Table containing information on the availability and type of each device known to the system†

CPT Class Parameter Table used by JCM. It contains all job statement parameters used to determine the job class.

CSD Class Structure Definition Table containing 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. A DAT exists for each dataset known to the system and defines where the dataset logically resides on mass storage, that is, on which logical devices and what portion of a device.

† Currently used only for tape devices

DCT Device Channel Table serving as a link between a physical or logical disk channel and the EQT. It is an interface to the EXEC disk driver. The DCT holds channel system performance data.

DRT Device Reservation Table. A DRT exists for each logical disk device known to the system. A DRT contains a bit map showing available and reserved tracks on the device.

DXI Permanent Dataset Catalog Extension Information Table containing information used by the Permanent Dataset Manager (PDM) such as the size of the Dataset Catalog Extension Table (DXT)

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

EQT Equipment Table containing an entry for each disk device known to the system

GRT Generic Resource Table containing an entry for each generic resource in the system.

IBT Interactive Buffer Table for managing the Interactive Buffer Pool

JXT Job Execution Table. The JXT controls all active jobs in the system and can contain as many as 256 entries. Entry 0 (the first entry) is used to represent the system itself.

LCT Link Configuration Table containing an entry for each CPU channel used for front-end communications

LIT Link Interface Table. SCP assigns an LIT entry at startup to each CPU channel used for front-end communications. This table is used primarily for channel control.

LXT Link Interface Extension Table. EXEC assigns an LXT entry for a front-end station at log-on time and releases the entry at log off. This table is used primarily for EXEC-STP communication of information on a front-end station.

MST Memory Segment Table containing an entry for each segment of memory allocated by the Job Scheduler (JSH) as well as an entry for each free segment. The number of entries in the MST is set to twice the number of JXT entries plus four words. Each MST entry is one word in length.

ODT Overlay Directory Table. Each overlay defined by a DEFINOVL macro contains an entry in the ODT. Each entry contains addressing information and data on the overlay's use.

OLL Overlay Load Request List holding a backlog of requests for overlays. When an overlay load is requested and the memory pool is full, an entry is added to the OLL to be processed when space becomes available.

PDI Permanent Dataset Information Table containing information used by the Permanent Dataset Manager (PDM), such as the number of overflow and hash pages.

PDS Permanent Dataset Table consisting 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.

PXT Processor Execution Table contains status information for each physical processor, including which user task is currently connected.

QDT Queued Dataset Table describing the multitype attributes for a disposed dataset. The table is managed by the Permanent Dataset Manager (PDM) and Exchange Processor (EXP) tasks. The number of entries in the QDT must equal the SDT entry count.

RJI Rolled Job Index Table containing for each defined JXT, an entry describing the job assigned to the JXT entry, allowing the recovery of jobs from mass storage.

RQT Request Table used to queue transfer requests for disk management. DQM uses the RQT to manage both logical and physical disk requests. RQT entries are queued to an EQT entry.

SBU System Billing Unit Table containing the values obtained when system billing units are calculated for system resources.

SDR System Directory containing a Dataset Name Table for each of the datasets comprising the system library. The SDR is initialized after a system startup.

SDT System Dataset Table containing an entry for each dataset spooled to or from a front-end system. An SDT entry can have appendages allocated out of an STP memory pool to contain TEXT field and station slot information.

SST Stager Stream Table. Eight input stream and eight output stream SSTs are contained within each LXT.

STPD STP Dump Directory containing pointers to task origins, buffers, and so on. An entry gives a mnemonic in ASCII plus the relative STP address for the area.

TDT Tape Device Table. The Tape Queue Manager task uses the Tape Device Table to control online tape devices. The TDT contains an entry for each tape device in the system.

TXT Task Execution Table contains all information to control all user tasks within the system.

UCT User Call Table containing a count of the number of times each type of user call is made. This table is used by the System Performance Monitor (SPM).

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

1.3.4 STP PROGRAM AREA

The System Task Processor (STP) consists of system tasks and reentrant code common to all of the system tasks. System tasks cannot access the memory area occupied by EXEC but can access the rest of memory.

Although system tasks are loaded into memory during startup, they are recognized only through an Executive create system 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 of this publication.

1.3.5 CONTROL STATEMENT PROCESSOR (CSP) AREA

An image of CSP is maintained either in memory following STP or on mass storage, depending upon the setting of an installation option. This program is copied into each user field where it executes each time the job requires interpretation of a control statement.

CSP is further described in section 20 of this publication.

1.3.6 USER AREA

The user area of memory is assigned to one or more jobs. Each job has an area called 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 and logfile DSPs; a logfile buffer; and a DNT area, which contains an entry for each dataset used by the job. In addition, task control blocks (TCBs) defining attributes of each executable user task are maintained in the JTA.

Each user field begins with a 128-word block called 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 part 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 where utility programs such as the loader, copy and positioning routines, and permanent dataset utility programs execute. 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 job user number.

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

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

JAC Job Accounting Table. This table defines an area for data to be returned to the user by an accounting request.

JCB Job Communication Block, residing at the very beginning of the user area and containing information used by both COS and library routines. Copies of the more important pointers are kept in the job's JTA to assist in JCB validation and re-creation.

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 in CSP is used for many permanent dataset requests.

See the COS Table Descriptions Internal Reference Manual, publication SM-0045, for detailed descriptions of these tables. This table is available as a listable tape.

1.4 MASS STORAGE SUBSYSTEM ORGANIZATION

Depending on the Cray computer 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 Cray model-dependent. These controllers are physically nonremovable.

Each disk storage unit contains a device label, datasets, and unused space to be allocated to datasets (figure 1-5). Additionally, one disk storage unit is designated as the master device and contains a table area called the Dataset Catalog (DSC), which contains information about permanent datasets.

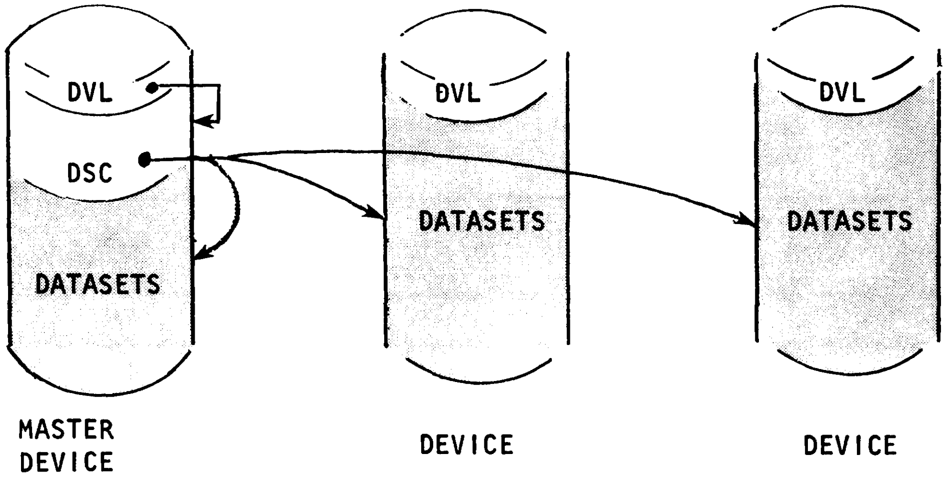


Figure 1-5. Mass storage organization

1.4.1 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.

1.4.2 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 before the Install program. This initial flaw information is stored on the device in a special table called the Engineering Flaw Table (EFT). The EFT is written to sector 17_{10} of the first track that can be succesfully reread on the device (no more than 10 tracks are tried). No EFT is written if no track in the first 10 tracks can be written and reread successfully. 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 Device Label Table (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.

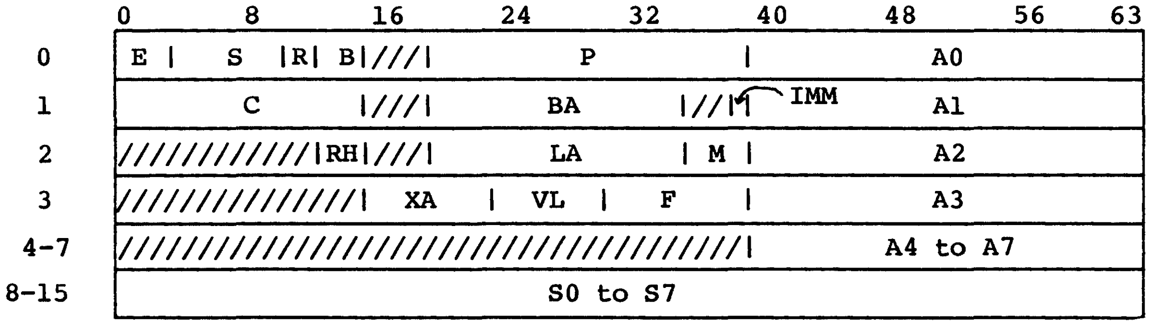
1.4.3 DATASET CATALOG (DSC)

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 Install which assumes that all of mass storage is vacant. Special consideration is given to job input and output datasets. Deadstart deletes all 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.

1.5 EXCHANGE MECHANISM

The technique employed in Cray computers to switch execution from one program to another is called 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 and to be exchanged with the information in the operating registers. Operating register contents are saved for the terminating program and the registers entered with data for the new program.

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



```
        0       8      16      24      32      40      48      56     63
       ----------------------------------------------------------------
  0    |E |   S  |R| B|////|         P         |          A0          |
       ----------------------------------------------------------------
  1    |       C       |////|      BA      |//|M|  IMM     A1          |
       ----------------------------------------------------------------
  2    |/////////////|RH|////|      LA     | M |          A2          |
       ----------------------------------------------------------------
  3    |////////////////|  XA  |  VL  |   F   |          A3          |
       ----------------------------------------------------------------
  4-7  |//////////////////////////////////////|       A4 to A7       |
       ----------------------------------------------------------------
  8-15 |                         S0 to S7                              |
       ----------------------------------------------------------------
```

Figure 1-6. CRAY-1 Exchange Package

Field	Word	Bits
Error type (E)	0	0-1
Syndrome bits (S)	0	2-9
Read mode (R)	0	10-11
Bank error address (B)	0	12-15
Program register (P)	0	18-39
Chip error address (C)	1	0-15
Base address (BA)	1	18-35
Interrupt Monitor Mode bit (IMM)	1	39
High-order bits of memory error read address (RH)	2	14-15
Limit address (LA)	2	18-35
Mode bits (M)	2	36-39
Exchange address (XA)	3	16-23
Vector length (VL)	3	24-30
Flag register (F)	3	31-39
Current contents of the eight A registers	0-7	40-63
Current contents of the eight S registers	8-15	0-63

As shown in section 2, the System Executive (EXEC) is always a partner in the exchange; that is, it is either the program relinguishing 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.

1.5.1 EXCHANGE PACKAGE

An Exchange Package is a 16-word block of data in memory that is associated with a particular computer program. An Exchange Package contains the basic hardware parameters necessary to provide continuity from one execution interval for the program to the next. The CRAY-1 Exchange Package is illustrated in figure 1-6; the CRAY X-MP Exchange Package is illustrated in figure 1-7.

1.5.2 EXCHANGE PACKAGE AREAS

System hardware requires all Exchange Packages to be located in the first 4096 words of memory. In addition, the deadstart function expects an Exchange Package to be at address 0. This Exchange Package 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 1-8).

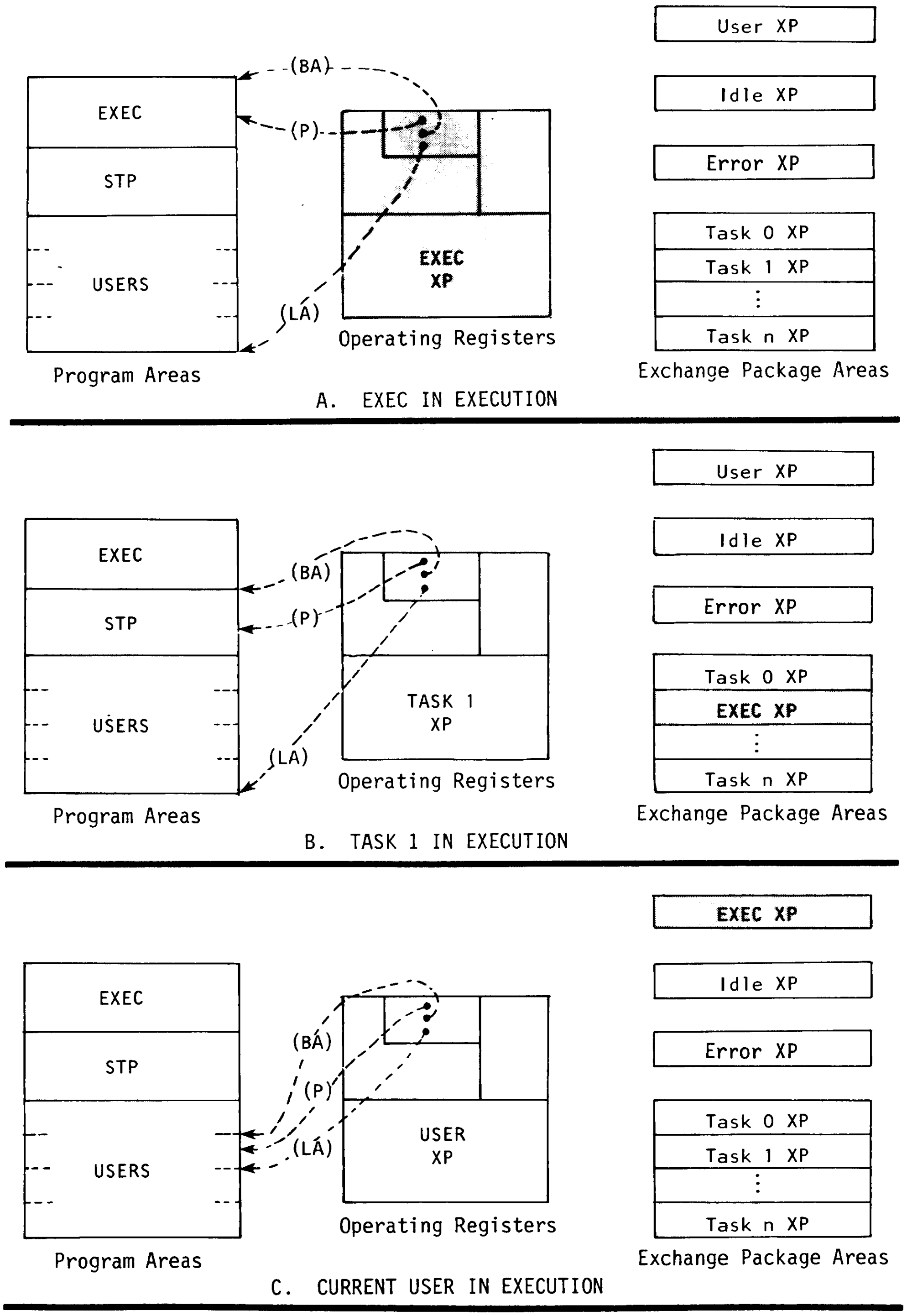
```
       0      8      16      24      32      40      48      56    63
PN    -------------------------------------------------------------
 0    |E|  S   |////|          P            |         A0          |
 1    R| CS | B |//////|     IBA      |ML1 |         A1          |
 2    |/VNU////////////|     ILA      |ML2 |         A2          |
 3    //////////////|F|  XA   | VL  |   F   |         A3          |
 4    //////////////////|    DBA    PS |/|  CLN      A4          |
 5    //////////////////|    DLA      |/////|         A5          |
6-7   ///////////////////////////////////////|     A6 to A7       |
8-15  |                        S0 to S7                           |
      -------------------------------------------------------------
```

Figure 1-7. CRAY X-MP Exchange Package

Field	Word	Bits
Processor number (PN)	0	1
Error type (E)	0	2-3
Syndrome bits (S)	0	4-11
Program Address register (P)	0	16-39
Read mode (R)	1	0-1
Read address (CSB)	1	2-6 (CS); 7-11 (B)
Instruction Base Address (IBA)	1	18-34
Instruction Limit Address (ILA)	2	18-34
Mode register (M)	1-2	35-39
Vector not used (VNU)	2	0
Flag register (F)	3	14-15; 31-39
Exchange Address register (XA)	3	16-23
Vector Length register (VL)	3	24-30
Data Base Address (DBA)	4	18-34
Program State (PS)	4	35
Cluster Number (CLN)	4	38-39
Data Limit Address (DLA)	5	18-34
Current contents of the eight A registers	0-7	40-63
Current contents of the eight S registers	8-15	0-63

The 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). This second portion of the STT is called the System Task Exchange Package Table (STX), and contains one Exchange Package for each STP task.



A. EXEC IN EXECUTION

B. TASK 1 IN EXECUTION

C. CURRENT USER IN EXECUTION

Figure 1-8. Exchange Package management

- The active user Exchange Packages. One user Exchange Package per CPU resides in the Processor working storage (PWS) entry at W@PWUSXP and is copied from the user's Job Table Area (JTA) when the job is connected to a CPU. The Exchange Package is then copied into the user's JTA when the job is disconnected from a CPU.

- The idle task Exchange Packages. One idle Exchange Package per CPU resides in the PWS entry at W@PWIDXP and is selected when no STP tasks or user jobs are scheduled for execution for a particular CPU.

- The Memory Error Correction task Exchange Packages. One correction Exchange Package per CPU resides in the PWS at W@PWCOXP and is selected when a memory parity error causes an exchange.

1.5.3 B, T, AND V REGISTERS

On any exchange to EXEC, the task or user program's B00 register (EXEC uses register B00) is saved. A task's B00 register value is stored in the System Task Table (STT). The active user's B00 value is stored during interrupt processing. When EXEC exchanges out, it restores the proper B00 register value.

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 the job is reconnected to the CPU. These registers are maintained in the job's Job Table Area (JTA).

1.6 COS STARTUP

During system startup, the operating system is loaded into Central Memory, begins execution, and generates or recovers 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 of this manual.

Install COS is started as if for the very first time. All Cray mass storage is assumed to be vacant. The startup program labels devices and establishes the Dataset Catalog (DSC) on mass storage.

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 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.

1.7 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

1.7.1 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 executing the following steps:

1. Making an entry in the System Dataset Table (SDT) and creating a memory pool entry containing station slot data
2. Requesting that an entry be created in the Dataset Catalog (DSC), thereby making the dataset permanent
3. Readying the Job Scheduler Task (JSH)

1.7.2 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),
- No other job in the same class of higher priority is waiting to begin processing, and
- The requested generic resources (for example, tape devices) are available.

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.

1.7.3 JOB ADVANCEMENT

The Job Scheduler (JSH) gives each job a CPU priority reflecting 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 can change a job's priority while a job is running.

Not all jobs having entries in the JXT are in memory; some are rolled out to mass storage when an event occurs causing 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 loaded whenever a job step terminates. Normal job step termination occurs when an F$ADV call is made to the system by the user program. Abnormal termination occurs upon detection of an error by either COS or hardware error interrupts during the job step or an F$ABT call by the user program.

1.7.4 JOB TERMINATION

When a job terminates, the following actions occur:

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

1.8 TASKS AND MULTITASKING

While this manual frequently refers to some particular *task*, several types of tasks occur in COS:

- The idle and memory error correction tasks resident in EXEC
- System tasks resident in STP
- User tasks resident in user jobs
- User library tasks resident in user jobs but under library control

This section defines several terms related to the above types of tasks. See the Multitasking User Guide, CRI publication SN-0222, for a full description of multitasking concepts.

1.8.1 MULTIPROGRAMMING

Multiprogramming is a mode of operation that provides for the sharing of processor resources among multiple, independent, software processes. This mode, used by many computing systems, makes most efficient use of a single CPU. In the multiprogramming mode, when several processes are ready to run, should one process be delayed by I/O, for example, another process can immediately be switched in to run on the CPU. In contrast, a system running in monoprogramming mode has only one process ready to run and any delays will leave the CPU idle. Processor resources could include more than one CPU, and in a multiprogramming environment, these multiple CPUs would be shared between multiple, independent software processes.

1.8.2 MULTIPROCESSING

Multiprocessing is a mode of operation that provides for parallel processing by two or more processors. That is, all processors work at the same time without adversely affecting each other.

1.8.3 TASKS

A *task* is a software process. It is a unit of computation that can be scheduled and whose instructions must be processed in sequential order.

Idle and memory error correction tasks

The EXEC idle task is described in section 2.11. Memory error correction is described in section 2.10. The memory correction task is unique in that it is not executed through the EXEC task scheduler.

System task

The tasks comprising the System Task Processor (STP) are referred to as system tasks. STP is described in section 3.

NOTE

The term *task*, as used in this manual, refers to a system task, unless otherwise noted.

User task

A *user task* is the entity referred to in the F$TASK system action request, as described in section 8.1. User jobs are generally unaware of user tasks; user task management requests are usually made by the multitasking library routines.

User library task

A *user library task* is the entity created by calling TSKSTART (initiate a task) in the multitasking library. Multitasking in a FORTRAN program is done as user library tasks. That is, when a FORTRAN program creates multiple tasks, the tasks created are user library tasks. User library tasks are created and synchronized by user-program calls to the multitasking library.

The multitasking library scheduler manages (schedules) user library tasks. The library scheduler creates, deletes, activates, and deactivates user tasks as needed; the library scheduler is responsible for assigning user library tasks to user tasks. Within a user job, the user program only knows about user library tasks; EXEC and STP only know about user tasks; the multitasking library scheduler forms the interface between user tasks and user library tasks.

1.8.4 MULTITASKING

Multitasking is a special case of multiprocessing, where more than one task can be executing in a user job. When multitasking, there is no guarantee that more than one processor will be allowed to work on the tasks of a given job, no guarantee that the tasks will execute in any particular order, and no guarantee of which task will finish first.

In this manual, *multitasking* refers only to user-level tasks (user tasks and user library tasks).

1.8.5 JOBS AND USER TASKS

Each user job consists of one or more user tasks. Most COS-managed resources, except a CPU, are allocated to the entire job, whereas each user task includes an exchange package, and an environment save/restore area. A user task can have a physical CPU allocated to it, and on the CRAY X-MP, can have a physical cluster allocated.

When the Job Scheduler (JSH) initially places a job into memory, the first task control block is created automatically by the system. An initial task is entered on the CPU queue. Other task creations within the job are the responsibility of the first task or tasks that are spawned by the first task. Other than this relationship, no hierarchy of tasks exists within a job.

A job can be rolled into and out of memory. An individual task cannot be rolled. Whenever JSH rolls a job out of memory, all tasks are marked as not schedulable and any tasks currently connected are disconnected from the CPU. A task has the same execution priority as the parent job.

When the cumulative execution times of all the tasks within a job exceed the job's time limit (from the JOB control statement), the job is marked as time limited and aborted.

1.9 MASS STORAGE DATASET MANAGEMENT

All information maintained on mass storage by the Cray Operating System (COS) is organized into collections of information called datasets. Mass storage 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 mass storage dataset is permanent if it has an entry in the Dataset Catalog (DSC) on disk. Permanent datasets are of two types: those created with directives (user permanent datasets), and those representing 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 through permission control words. The user can create a user permanent dataset by prestaging 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 can be removed from the system with the DELETE control statement or macro. More than one authorized user can 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. Users cannot delete or access these datasets, because the system has unique access to them. One such dataset 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 Computer System. 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 mass storage. The output datasets that were made permanent are sent to a front-end computer system for processing. They are deleted from mass storage when their receipt has been acknowledged by the front-end computer system.

1.10 I/O INTERFACES

Figure 1-9 presents an overview of the interfaces and system components involved in performing input and output in the system. This figure 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 (CFT) Reference Manual, publication SR-0009, and in the CRAY-OS Version 1 Reference Manual, publication SR-0011. Details for EXEC (driver level) I/O are given in section 2. Details for STP interfaces are given in section 3.

I/O can be on any dataset structure 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 (that is, 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 can be formatted by the user or made through CAL macros.

The Logical Record I/O routines issue Exchange Processor requests (that is, F$ calls) consisting of read circular and write circular requests to the Circular Input/Output (CIO) routines resident in STP (see section 4).

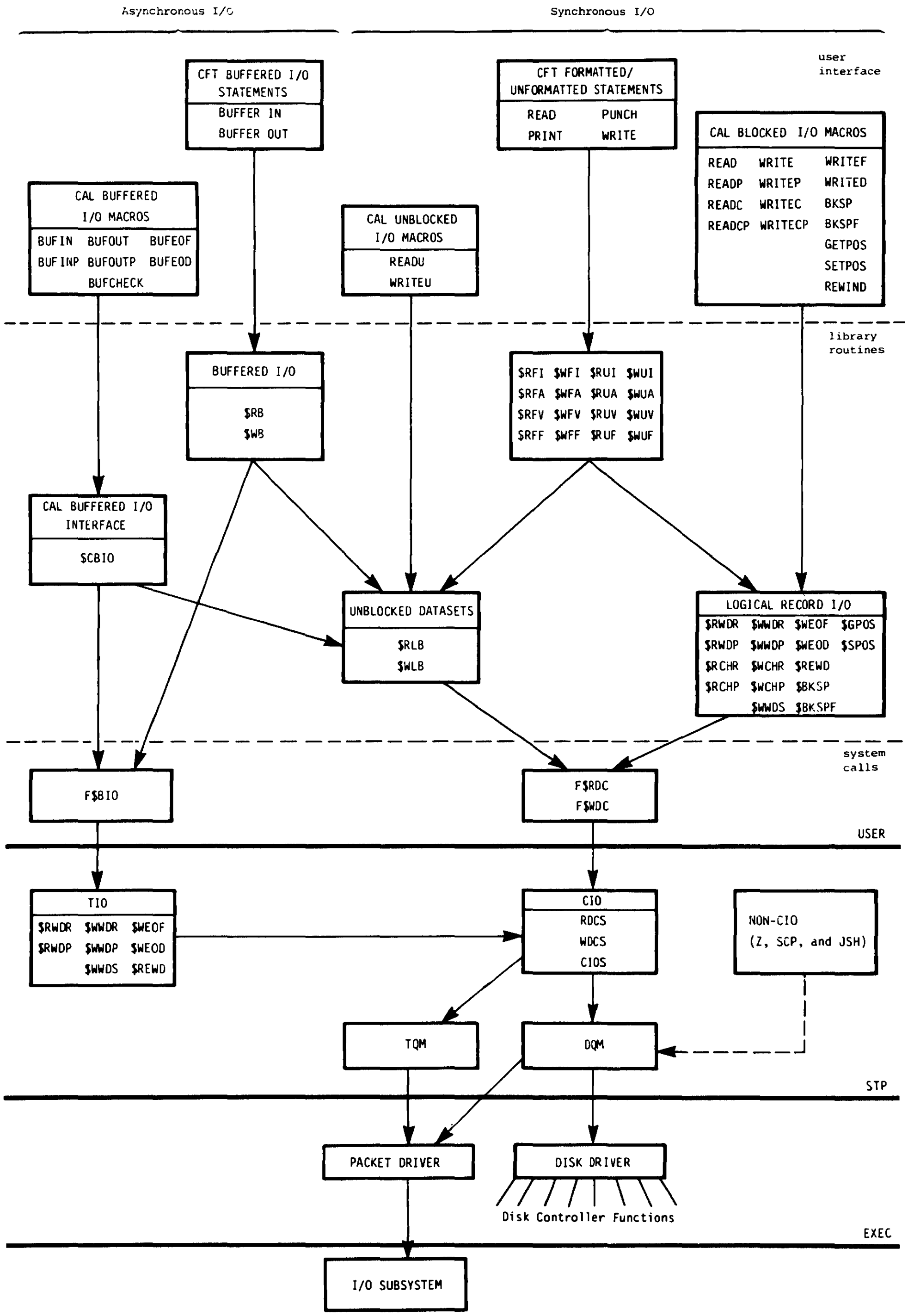


Figure 1-9. Overview of COS I/O

System logical I/O required by COS tasks (for example, management of the DSC) is generally performed through Task I/O (TIO) routines resident in STP (see section 4). 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 either to the Tape Queue Manager (TQM) or to the Disk Queue Manager (DQM) through DNT and DSP tables. The DNT for a dataset points to its DSP, which specifies the request.

CIO is the normal mode of communication with DQM. However, DQM also communicates with the station and startup interfaces. In these interfaces, SCP and Startup 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 (see section 2.8) or, if an I/O Subsystem is present, to disk I/O software in the I/O Subsystem (see the IOS Software Internal Reference Manual, CRI publication SM-0046). The disk driver consists of a number of channel processors that issue functions to the disk controllers.

TQM manages tape I/O between user jobs and the I/O Subsystem (IOS). Software in the IOS responds to requests for tape I/O received from TQM and physically controls block multiplexer channels, control units, and tape devices. (See the IOS Software Internal Reference Manual, CRI publication SM-0046, for a description of this software.)

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 next program to execute. Components of EXEC include the following.

- An interchange routine
- Interrupt handlers
- Channel processors
- A monitor request processor
- A Front-end Driver
- A Disk and SSD Driver
- A Packet I/O Driver
- A task scheduler

These routines are integral to EXEC. Control transfers from routine to routine through simple jumps.

After CPU startup, EXEC begins execution (at EX) whenever a system, user, or idle task is interrupted. The interrupt can result from the execution of an exit instruction (EX or ERR), or from a variety of hardware-related interrupts (operand range, program range, programmable clock, I/O channel, deadlock, or interprocessor interrupts). On reentry EXEC saves B00, performs various accounting and validation functions, ensures that the operating system is single-threaded (that is, it executes in only one CPU in multiprocessor systems), and enters the interchange analysis routine (ENA).

The interchange analysis routine examines the interprocessor communications area, the channel interrupt register, the real-time clock, and the interrupted exchange package to determine the cause of the interrupt and passes control to the appropriate handler. Each interrupt handler clears the appropriate flag in the interrupted exchange package and, after processing the interrupt condition, returns to interchange analysis (which checks for additional conditions). When all outstanding interrupt conditions have been processed, the system task scheduler (TS0) is entered.

The task scheduler selects the highest priority system task which is ready to run and causes it to be executed. If no system tasks are ready, the user task scheduler (SCHUSER) is invoked.

If no user task is currently connected, the user task scheduler selects either the currently-connected user task, or the idle task for execution.

After the selection of a task (system, user, or idle), an exchange out of EXEC occurs. The cycle begins again when the task is interrupted. Figure 2-1 illustrates the execution flow into and out of EXEC.

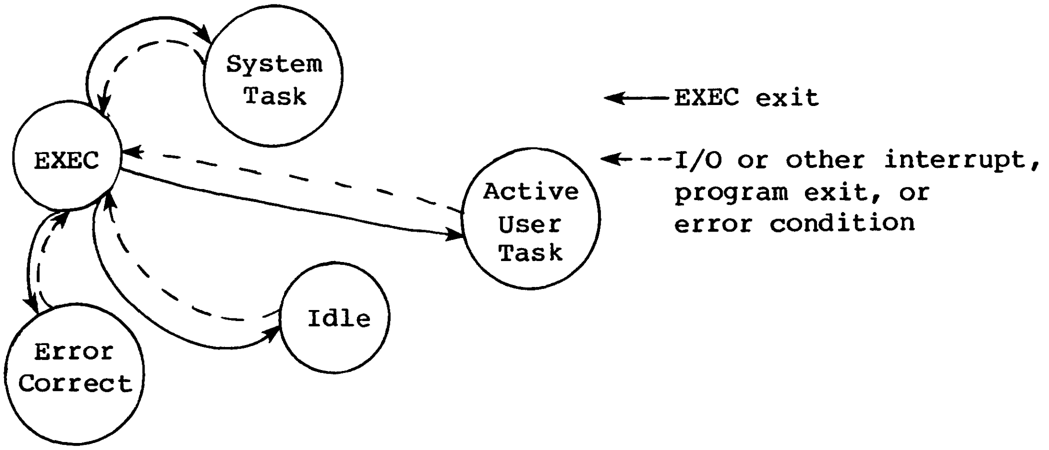


Figure 2-1. EXEC-controlled exchange sequences

2.1 INTERCHANGE ANALYSIS

Each time ENA is entered, the interprocessor request queue is checked for an interprocessor message. If a message is found, IPREQST is called to process it. The message is then cleared, and control returns to ENA.

ENA next looks for pending I/O channel interrupts. When an I/O channel is found to have an interrupt pending, control transfers to IOI which clears the I/O interrupt bit in the active exchange package, selects a processing routine based on the channel number, and enters that routine. The channel processor returns control to ENA.

Next, the real-time clock and the time event table are examined. If a timer event is pending, control is passed to TEI (the expired time event interrupt handler). After processing the timer event, control is returned to ENA.

Finally, after ENA has processed all of the above conditions, the flags in the interrupted exchange package are examined to determine the cause of the exchange. Note that the I/O Interrupt flag is ignored since ENA has already processed pending I/O interrupts. The Interrupt Handler Table (IHT) maps each flag into a handling routine. The flags are processed in order from left (high-order) to right (low-order). When a flag is set, the corresponding interrupt handler is entered. Again, when processing is complete, control returns to ENA.

After a pass through ENA with none of the above conditions encountered, the task scheduler (TSO) is invoked.

Figure 2-2 illustrates the relationship of the elements of EXEC to other system components.

2.2 INTERRUPT HANDLERS

Each interrupt handler routine can invoke further routines for processing. When an interrupt is processed, control returns to Interchange.

2.2.1 I/O INTERRUPT HANDLER (IOI)

IOI clears the I/O Interrupt flag in the interrupted exchange package, increments the interrupt count for the channel, sets the next channel processor to RJ (reject), makes a history trace entry, and exits to the current channel processor.

2.2.2 EXPIRED TIME EVENT INTERRUPT HANDLER (TEI)

TEI clears the Programmable Clock Interrupt flag in the interrupted exchange package, makes a history trace entry, sets up the next scheduled time event for the CPU, and exits to the time event processor.

2.2.3 PROGRAMMABLE CLOCK INTERRUPT HANDLER (PCI)

PCI clears the Programmable Clock Interrupt flag in the interrupted exchange package, makes a history trace entry, and sets up the next

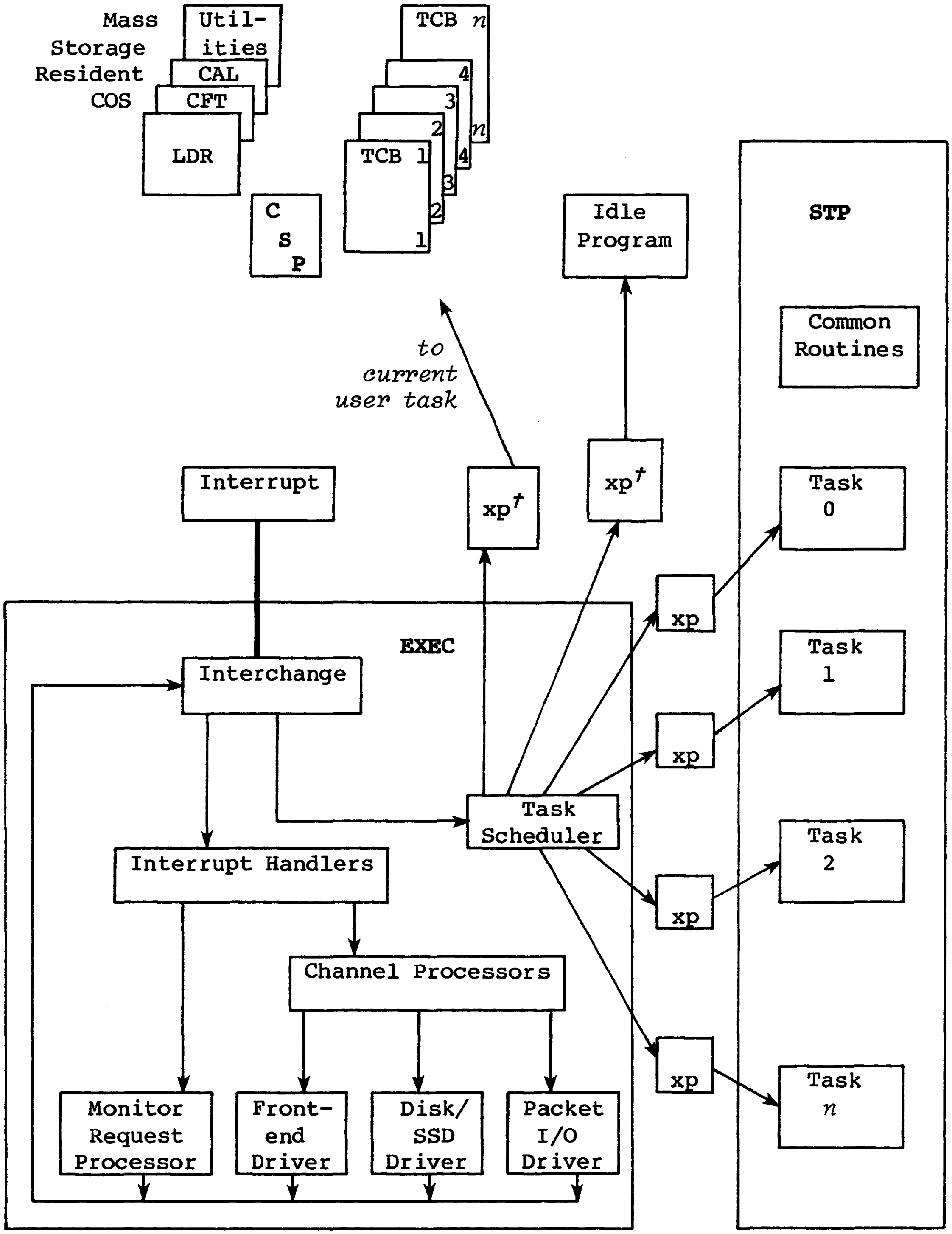


Figure 2-2. System control

† One Exchange Package per CPU

default time event. This interrupt is unexpected, since the calculated time event expiration should process all PCI interrupts. This routine is present but unused on mainframes without a programmable clock.

2.2.4 MCU INTERRUPT HANDLER (CII)

CII clears the MCU Interrupt flag in the interrupted Exchange Package.

2.2.5 ERROR INTERRUPT HANDLER (EE)

EE clears the appropriate flag in the interrupted Exchange Package and makes a history trace entry. Interrupts handled by this routine are:

- Floating-point error interrupt
- Operand range error interrupt
- Program range error interrupt
- Error exit

Processing depends on the type and cause of the error.

2.2.6 MEMORY ERROR INTERRUPT HANDLER (ME)

ME clears the Memory Error flag in the interrupted Exchange Package, corrects the error if it is a single-bit error, and logs the error by sending a packet to the Message Processor task (MEP). A multibit error causes the system to halt if the error occurred in the operating system or by a channel read from an I/O buffer.

2.2.7 NORMAL EXIT INTERRUPT HANDLER (NE)

NE clears the Normal Exit flag in the interrupted exchange package and determines whether a system task or user job made the exit. A system task exit causes the Monitor Request Processor to be invoked; a user job exit causes the Exchange Processor (EXP) task to be scheduled.

2.2.8 INTERPROCESSOR INTERRUPT HANDLER (IPI)

IPI clears the Interprocessor Interrupt flag in the interrupted exchange package on a CRAY X-MP mainframe.

2.2.9 DEADLOCK INTERRUPT HANDLER (DLI)

On the CRAY X-MP mainframe, deadlock interrupts can occur that do not indicate that a programming error occurred. For instance, a deadlock interrupt occurs whenever a Test & Set Semaphore (0034) instruction is executed while the semaphore in question is already set and no other CPUs are in the executing CPU's cluster.

The only level where detection of true deadlocks can take place is user task scheduling (in JSH) and the only path to JSH from EXEC is through EXP. Thus, deadlock interrupts generally require the scheduling of both EXP and JSH, a large burden on the system when multitasking is used.

In order to reduce this system overhead, EXEC does some screening of deadlock interrupts. The goal of this screening is to avoid the trip through EXP/JSH when there is a high probability that the user task can successfully resume processing, while at the same time avoiding needless deadlock interrupts. To these ends, EXEC maintains a counter in the user's TCB, and uses this counter plus other global information to (selectively) return to the user task, rather than scheduling EXP when a deadlock is encountered. This direct return to the user takes place when all of the following conditions hold:

- The user is multitasking (that is, more than one user task is active).
- More than one CPU is in the user's cluster.
- An excessive number of deadlock interrupts have not occurred since the user task's last trip to EXP. This number is defined in STP low-memory location DLIGNORE.

If any of the above conditions does not hold, EXP deals with the deadlock.

Deadlock interrupts should never occur from within system tasks, such interrupts cause a system halt.

2.3 CHANNEL MANAGEMENT

EXEC manages channels in pairs, with the even-numbered side an input channel and the odd-numbered side an output channel. A channel pair consisting of channels 2 and 3 is referred to as channel pair 1, and so on.

EXEC manages the mainframe's physical I/O channels based on parameter settings in the configuration deck CONFIG@P. The channel parameters are:

C@CPLCHN Lowest physical I/O channel number. This parameter is an even number.

C@CPHCHN Highest physical I/O channel number. This parameter is an odd number.

C@CPMCHN Maintenance Control Unit (MCU) input channel number. May be equal to C@CPLCHN. This parameter is an even number.

C@CPSCHN CRAY X-MP Solid-state Storage Device (SSD) control channel number. This value is used during system initialization to master clear the SSD channel.

For more information on the configuration deck CONFIG@P, consult the COS Operational Procedures Reference Manual, publication SM-0043.

Typical channel layouts are shown below.

CRAY-1 mainframes:

Channel	Pair	Description
2,3	1	6 Mbyte channel to MCU (MIOP or Data General Eclipse)
4,5	2	Depends on configuration
6,7	3	Depends on configuration
8,9	4	Depends on configuration
10,11	5	Depends on configuration
12,13	6	Depends on configuration
14,15	7	Depends on configuration
16,17	8	Depends on configuration
18,19	9	Depends on configuration
20,21	10	Depends on configuration
22,23	11	Depends on configuration
24,25	12	Depends on configuration

CRAY X-MP mainframes:

Channel	Pair	Description
6,7	3	SSD 1250 Mbyte channel[†]
8,9	4	6 Mbyte channel (MIOP)
10,11	5	6 Mbyte channel
12,13	6	6 Mbyte channel
14,15	7	6 Mbyte channel

2.3.1 CHANNEL MANAGEMENT TABLES

The following tables aid in channel management:

CBT	Channel Buffer Table
CHT	Channel Table
LIT	Link Interface Table
SCT	Subsystem Control Table
STT	System Task Table
LIT or CBT	I/O Service Processor Tables

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.

Channel Buffer Table (CBT)

EXEC assigns one CBT entry to each pair of Channel Table (CHT) entries during EXEC initialization. The CBT is the default processor table for channel activity and is used by the Disk/SSD Driver.

Channel Table (CHT)

Each site configures one CHT entry per mainframe I/O channel, plus enough dummy entries at the beginning, so the physical I/O channel number is an index into the CHT. (Site configuration information is provided in the COS Operational Procedures Reference Manual, publication SM-0043.) Each entry contains: a task parameter block address linking the channel to an STP task (not for the MIOP channel), a table address, and an interrupt handler address.

[†] Optional. Channel 6 is unused but is allocated to make a pair if this option is present.

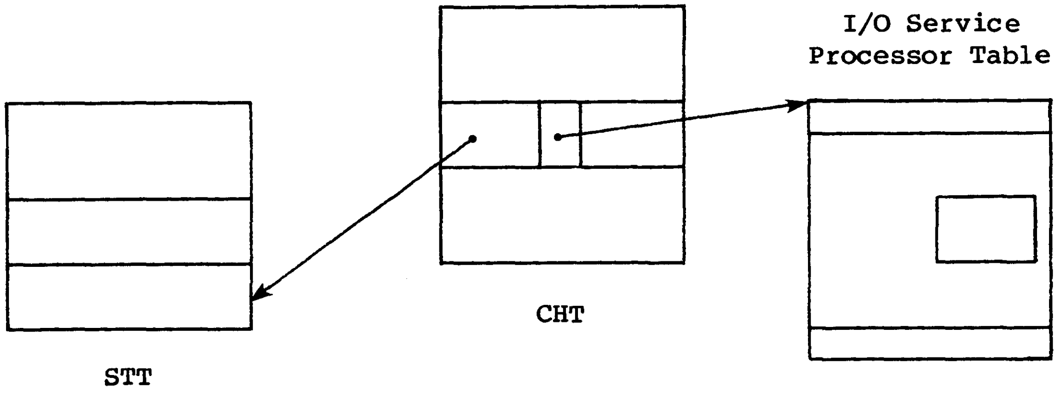


Figure 2-3. Channel Table linkage with assigned task

Link Interface Table (LIT)

The Front-end Driver assigns one LIT entry to a pair of Channel Table (CHT) entries if the channel pair is to be used for front-end I/O.

Subsystem Control Table (SCT)

EXEC uses the SCT to select a processor for a packet received from the MIOP (I/O Subsystem). See the discussion on the Packet I/O Driver for details, in section 2.9.

System Task Table (STT)

The STT contains information about each STP task for scheduling a task to run if channel activity warrants it.

I/O Service Processor tables

The I/O Service Processor tables contain information for control of the channel processor and can contain pointers to other tables. Front-end and mass storage channels have different I/O Service Processor tables. The service table is the LIT for Front-end Driver requests and the CBT for Disk/SSD Driver requests.

2.3.2 CHANNEL ASSIGNMENTS

When an STP task makes an I/O request for a specified channel pair, EXEC assigns the STP task that channel pair. The monitor requests involved are R005 (Front-end Driver) and R011 (Disk/SSD Driver). The R022 request (packet I/O) can also involve I/O, but the channel to the I/O Subsystem is not assigned to a specific task.

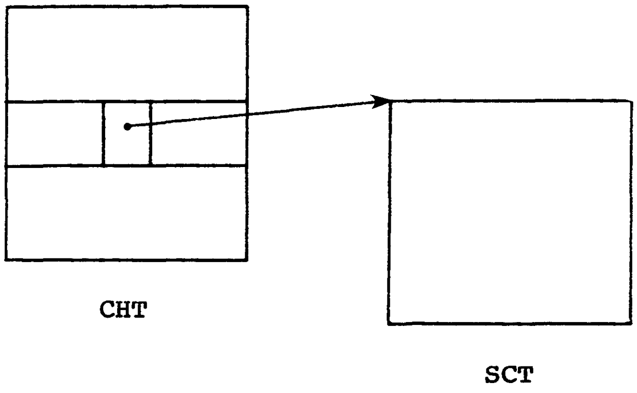


Figure 2-4. Channel Table linkage for packet I/O

2.3.3 CHANNEL PROCESSORS

The Channel Table (CHT) has a processor address for each physical mainframe channel configured. By default, this channel processor is the reject (RJ) processor, which ignores all interrupts on the channel. If an I/O operation is in progress, each processor address indicates the interrupt handler that receives control when an interrupt is received on a particular channel.

EXEC has the following categories of interrupts (and corresponding interrupt processors):

- Front-end Driver interrupts (R005)
- Disk/SSD Driver interrupts (R011)
- MIOP (I/O Subsystem) Driver interrupts (R022)

Front-end driver interrupt handlers

Any of the following processors are assigned to a CPU I/O channel as a result of monitor request 5 (Front-end Driver) for front-end computers and network adapters attached directly to a CPU channel.

Processor	Function
R005C/RLCP	Reads (or waits for) a link control package (LCP)
R005C/RSSEG	Reads (or waits for) a data subsegment
R005C/RLTP	Reads (or waits for) a link trailer package (LTP)
R005C/WLCP	Writes a link control package
R005C/WSSEG	Writes a data subsegment
R005C/WLTP	Writes a link trailer package
R005C/WXLCP	Writes an error link control package
R005C/WXLTP	Writes an error link trailer package
R005C/CCLRB	Reads input channel before master clear
R005C/CCLRD	Writes a Select function code (VAX interface only)
R005N/NRLCF	Reads zero data (Wait for Message function acknowledgement)
R005N/NRLCP	Reads a link control package and link control package extension (LCPE)
R005N/NRSEG	Reads a data segment
R005N/NWLCF	Reads zero data (Transmit Message function acknowledgement)
R005N/NWLCP	Writes a link control package and link control package extension
R005N/NWSEF	Reads zero data (Transmit Data function acknowledgement)
R005N/NWSEG	Writes a data segment
R005N/NWXLF	Reads zero data (Transmit Message function acknowledgement)

Processor	Function
R005N/NWXLC	Writes an error link control package or link control package extension
R005N/NCLRB	Reads zero data (Clear Adapter function acknowledgement)
R005N/STATA	Reads an adapter status word
XPROC/ENA	No operation

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 (SCP). See section 2.7 for details on the Front-end Driver.

Disk/SSD Driver interrupt handlers

EXEC assigns any of the following processors to a mainframe I/O channel as a result of monitor request 11_8 (Disk/SSD Driver). This request performs I/O on disk controllers and disk storage units connected directly to mainframe channels or to an optional Solid-state Storage Device (SSD).

Processor	Function
DDC280	Indicates disk block transfer is complete
DDFCT10	Output interrupt handler, no response expected
DDRSP	Output interrupt handler, response expected
DDTO	Disk software timeout interrupt handler
DDE140	Input interrupt handler, correction vector received
SSINT	Input interrupt handler, SSD status received

When a processor completes its function, it assigns the next processor to the channel without involving a task. See section 2.8 for details on the Disk/SSD Driver.

I/O Subsystem MIOP command and status processors

Either of the following processors is assigned to a mainframe I/O channel as a result of monitor request 22 (packet I/O).

Processor	Function
APIIP	Processes MIOP status input interrupt
APOIP	Processes MIOP command output interrupt

These interrupt processors are part of the Packet I/O Driver, which is detailed in section 2.9.

2.4 TASK SCHEDULER

Task scheduling is entered when all interrupt conditions are processed and the CPU is looking for something to do. If one or more system tasks are ready to run, the task with the highest priority is selected for execution. If no system task is eligible, the user task connected to the CPU in question is selected. If no user is connected, the idle package is selected for execution. The variables used in system task scheduling are:

- STAPB, a field in the System Task Table (STT) header that contains the STT address of the previously-active system task.
- STPLK, the STP lock indicator. When nonzero, the previously-executing STP task has disabled preemptive task scheduling, indicating that the task scheduler should return to that task.
- TBIDLE, a field in the Task Breakpoint Table. When nonzero, a system task is stopped at a breakpoint, indicating that only the breakpoint-processing task (SCP) is a candidate for scheduling.
- TPT, the Task Priority Table. This table is indexed by priority, and each table entry contains the address of the system task with the corresponding priority.
- STPRL, the System Task Priority Ready List, contains a bit for each possible task priority. When a bit in STPRL is set, the system task with the corresponding priority is ready to run, that is, it is not suspended.

The basic decisions of task scheduling, in order, are:

- If STPLK is nonzero, return to the previously active system task. The STT address of this task is contained in STT field STAPB. If any system tasks with a higher priority than the selected task are found, set the STP Lock Recall flag (LKRCL) so that the UNLOCK macro will exchange to EXEC to allow the higher-priority task to be executed when the lock is released.

- If a system task is at a breakpoint (TBIDLE is nonzero), select SCP if it has been initialized and is not suspended. If SCP has not yet been initialized, or if it is suspended, select the idle package instead.

- If any system task is ready to run, select the task with the highest priority and cause it to be executed. (The tests for ready-to-run and highest-priority are combined since STPRL implicitly contains a priority-ordered list of ready tasks.)

- If no exchange package was selected as a result of the above steps, user task scheduling (SCHUSER) is entered.

Figure 2-5 illustrates the table linkages for task scheduling.

2.5 EXEC RESOURCE ACCOUNTING

EXEC maintains the following performance information in EXEC tables:

- Accumulated CPU time for itself (in Processor Working Storage)

- Accumulated CPU time for each task (in STT)

- Total time given to users (in Processor Working Storage)

- Count of all channel interrupts for both real and pseudo channels (IC)

- Each user's execution time (in TCB)

- 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

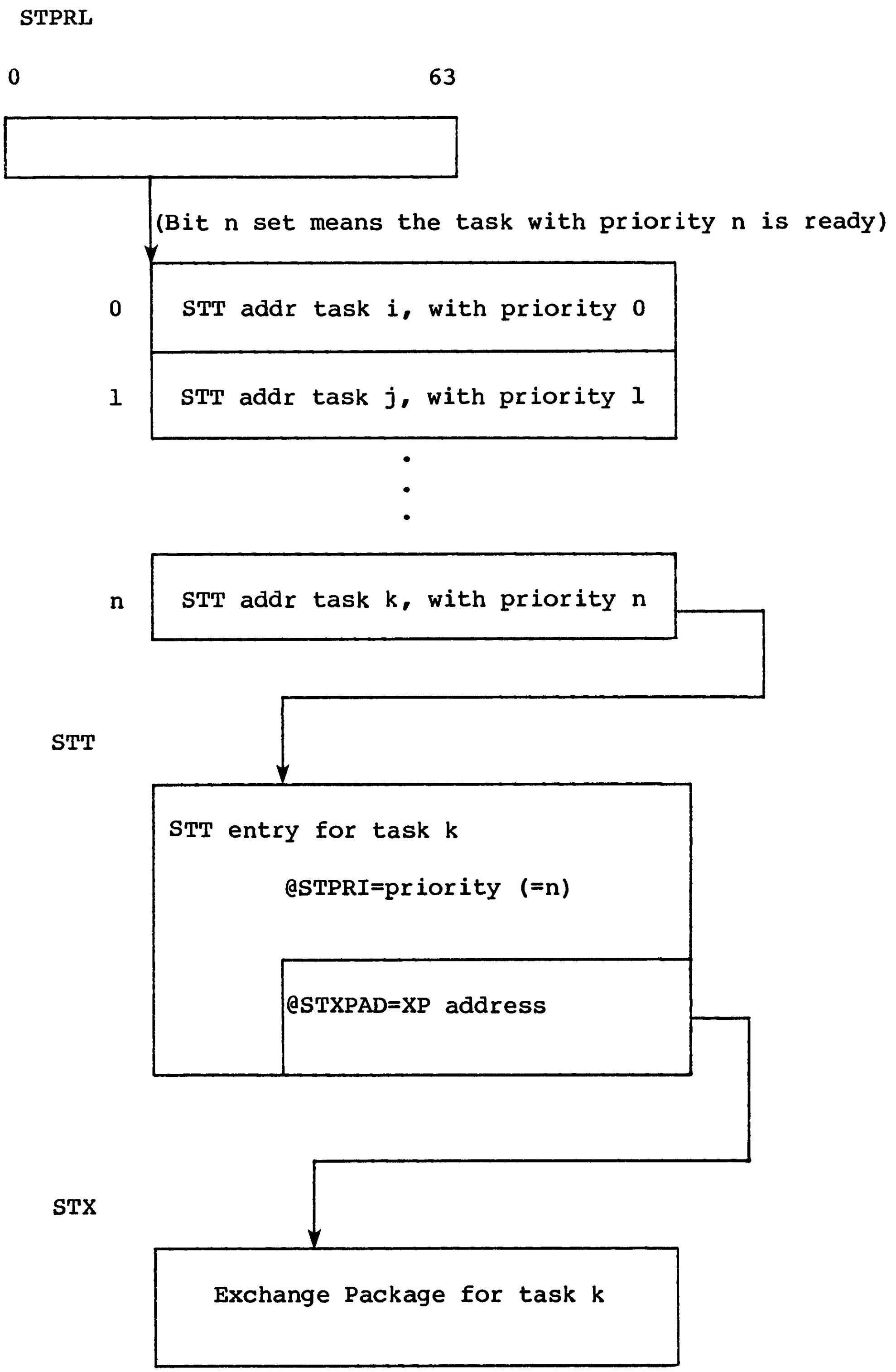

STPRL
0
63
(Bit n set means the task with priority n is ready)
0
STT addr task i, with priority 0
1
STT addr task j, with priority 1
n
STT addr task k, with priority n
STT
STT entry for task k
@STPRI=priority (=n)
@STXPAD=XP address
STX
Exchange Package for task k

Figure 2-5. Task scheduling table linkages

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 Executive Request Processor handles the requests defined by the Monitor Call Table (MCT).

When EXEC returns to a task following processing of an Executive request, control returns at (P)+2 for a normal return and at (P) if an error occurred. (P) is the address of the instruction following the exit to EXEC. When an error return is made by EXEC to (P), S6 contains an error code.

2.6.1 EXECUTIVE REQUESTS

This section provides the request format and functional flow of executive requests issued by tasks. Executive replies are described in section 3.2.1 of this publication, EXEC/Task Communication; error return codes are described in section 2.6.2 of this publication, EXEC Error Codes.

Create a system task request (CTSK=01)

This request initializes table space within EXEC defining a new system task and invokes the newly created task.

Format:

```
         0       8       16      24      32      40      48      56    63
        ------------------------------------------------------------------
   S5  |                             @CTTN                                |
       |------------------------------------------------------------------|
   S6  |////////////////////////////////////////////////|@CTPRI | @CTID   |
       |------------------------------------------------------------------|
   S7  |////////////////|       @CTPR                          |  @CTFC    |
        ------------------------------------------------------------------
```

Field	Word	Bits	Description
@CTTN	S5	0-63	Task name, left-justified, binary zero-filled; must be unique.
@CTPRI	S6	48-55	Task priority; must be unique.
@CTID	S6	56-63	Task ID number; must be unique.

Field	Word	Bits	Description
@CTPR	S7	16-39	Initial P register for task
@CTFC	S7	55-63	Request code (CTSK=01_8)

System tasks (normally Startup) use the CTSK request to create and invoke a new system task. Control returns to the requesting task as priorities permit, but EXEC allocates space in the System Task Table (STT) for the newly created task. EXEC sets the task status to not suspended, initializes an entry in the Task Priority Table (TPT) pointing to the new STT entry, and marks the task as ready in the System Task Priority Ready List (STPRL).

EXEC sets up a standard system task Exchange Package for the task, with a base address set to B@STP, the initial P register as specified in the request, X-MP cluster CLSYS selected, @XPSEI (selected for external interrupt) set, @XPORE (interrupt on operand range error) set, and interrupt on memory errors (single- and double-bit) set according to system defaults.

ERROR CONDITIONS:

(P) exits:

(S6)=ERTALC (026) if task is already created

(S6)=ERNTS (001) if all tasks have already been created

$STOP023: if a duplicate task priority is encountered

Ready system task request (RTSK=02)

The ready system task request causes another system task to be readied, and (optionally) causes an entry to be entered into the system trace buffer.

Format:

```
     0       8      16      24      32      40      48      56    63
S1  |                          Trace-1                              |
S2  |                          Trace-2                              |
S6  |                           Task                                |
S7  |T|/////////////////////////////////////////////////////|  FC   |
```

Field	Word	Bits	Description
Trace-1	S1	0-63	First word of intertask message
Trace-2	S2	0-63	Second word of intertask message
TASK	S6	0-63	Task number to be readied
T	S7	0	Intertask Message flag: If set, S1 and S2 hold an intertask message which is to be entered into the History Trace Table. S1 and S2 are ignored if T=0.
FC	S7	55-63	Request code (RTSK=02_8)

System tasks use the RTSK request to ready another system task. Since this function is often associated with intertask messages (replies), the request allows the caller to place an entry in the History Trace Table as part of the call (rather than requiring an additional EXEC request to perform the trace).

If the sign bit of S7 is clear in the request, the target task is readied with no additional processing. If the sign of S7 is set in the request, then S1 and S2 in the requesting task's Exchange Package are assumed to contain a two-word entry for the History Trace Table. The message set produced by the latter looks like:

Trace code	Words 1 and 2 in the history trace
012	S1 and S2 from requesting task's Exchange Package
042	"READY xxx->yyy" in ASCII

xxx are the first three characters of the requesting task name, and *yyy* are the first three characters of the target (readied) task name. 042 is the history trace code for EXEC ASCII message, and 012 is the history trace code for intertask message.

ERROR CONDITIONS:

(P) exits:

(S6)=ERTNX (003) if target task does not exist

System task self-suspend request (SUSP=03)

A system task uses the system task self-suspend request when it wishes to suspend itself.

Format:

```
      0       8       16      24      32      40      48      56     63
 S6  |/////////////////////////////////////////////////////////////////|
 S7  |///////////////////////////////////////////////////////|   FC    |
```

Field	Word	Bits	Description
FC	S7	55-63	Request code (SUSP=03_8)

The request has no parameters. When the task is readied (by some other task, or by EXEC), the task resumes at (P)+2.

Front-end Driver request (FET=05)

This request invokes the Front-end Driver (FED). FED either processes the request and/or formats a message for the Master I/O Processor (MIOP) and the IOP Driver. See section 2.7 for a more detailed description of the Front-end Driver.

Format:

```
      0       8       16      24      32      40      48      56     63
 S6  | REQ|////////////|        LXT            |          LIT          |
 S7  |//////////////////////////////////////////|CHT| CHO  | CHN |  05  |
```

Field	Word	Bits	Description
REQ	S6	0-3	Operation request code: FETCON (0) Channel on FETCOF (1) Channel off FETOUT (2) Output to front end
LXT	S6	16-39	Absolute address of LXT entry (only if REQ=FETOUT)
LIT	S6	40-63	If REQ is FETCON or FETCOF, absolute address of LIT entry

Field	Word	Bits	Description
CHT	S7	37-40	Channel type
CHO	S7	41-48	Channel ordinal (nonzero only for IOP channel)
CHN	S7	49-54	Channel pair number
FC	S7	55-63	Monitor request number (05_8)

The 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 channel registers CA and CL for input and/or output.
   d. Start processing by station channel driver.
   e. Release task from channel.

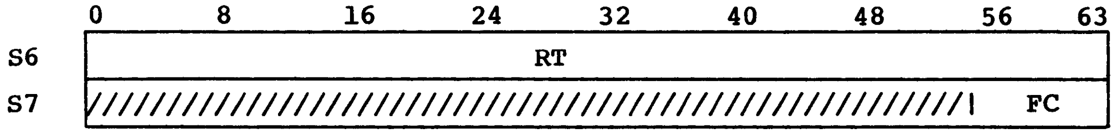
2. Otherwise:

   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.

Delay system task for time request (TDELAY=06)

A system task uses the delay system task request when it wishes to delay (give up the CPU) until a specified time.

Format:



Field	Word	Bits	Description
RT	S6	0-63	RT clock after which task wishes to be readied.
FC	S7	55-63	Request code (06_8)

Several warnings are associated with this request:

- The requesting task may be readied before the time specified in the event of requests by other system tasks.
- The requesting task may be readied any time after the specified RT clock value has arrived, depending on system load and task priorities.

Because of the above conditions, callers of the TDELAY request must conduct their own timing when they are concerned with specific delays.

Reserved for site use request (RESERVED=07)

This request is reserved for site use.

Start second CPU request (STRTCP2=10)

This request is valid only on CRAY X-MP mainframes. It sets up an initial Exchange Package at location 0 and deadstarts the second CPU.

Format:

```
      0       8       16      24      32      40      48      56    63
     +----------------------------------------------------------------+
 S6  |////////////////////////////////////////////////////////////////|
     +----------------------------------------------------------------+
 S7  |///////////////////////////////////////////////////////|   10₈  |
     +----------------------------------------------------------------+
```

Field	Word	Bits	Description
FC	S7	55-63	Monitor request number (10_8)

The flow is as follows:

1. Error if mainframe is not CRAY X-MP.
2. Build an initial Exchange Package at location 0.
3. Deadstart the second CPU with an interprocessor interrupt.
4. Exit to exchange processor.

When the second CPU gains control, the flow is:

1. Clear interprocessor interrupt.
2. Wait for access to operating system, if necessary.
3. Set up initial default time event.
4. Indicate second processor is started.
5. Set up Processor Execution Table (PXT) entry to reflect the status of the newly-started processor.
6. Set up HIGHCPUN cell in low-STP memory to reflect the number of the newly-started processor.
7. Exit by simulating an exchange into EXEC at EN.

Disk block I/O request (IO=11)

The disk block I/O request results in execution of the disk driver. See section 2.8 for detailed information on the Disk/SSD Driver.

Format:

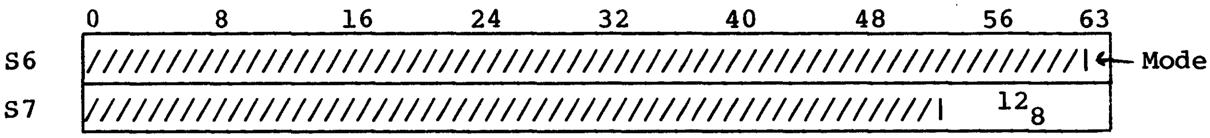
```
       0      8       16      24      32      40      48      56    63
      ----------------------------------------------------------------
  S6  |////////////////////////////////////////|     EQT address     |
      ----------------------------------------------------------------
  S7  |///////////////|     DCT address        | Channel no.  | 11_8 |
      ----------------------------------------------------------------
```

Field	Word	Bits	Description
EQT	S6	40-63	Equipment Table address
DCT	S7	16-39	Device Channel Table address
CHN	S7	40-54	Software channel (channel pair) number
FC	S7	55-63	Monitor request number (11_8)

Select single-bit error detection mode request (SEDSEL=12)

This request enables or disables single-bit memory error detection for the idle and all STP tasks. Memory error detection mode bits are set for user jobs only when the user's Exchange Package is copied to the active user Exchange Package area. The memory error detection mode applies to all CPUs in the mainframe.

Format:



Field	Word	Bits	Description
Mode	S6	63	Memory error detection mode: 0 Disable single-bit error interrupts 1 Enable single-bit error interrupts
FC	S7	55-63	Monitor request number (12_8)

The flow is as follows:

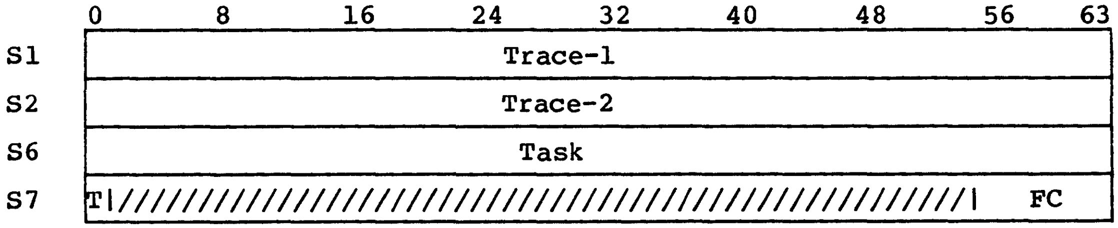
1. Set selected mode for idle tasks.

2. Set selected mode for all STP tasks.

Ready system task and suspend self request (RTSS=14)

The ready system task and suspend self request permits one system task to ready another and then suspend itself. This request is typically used in intertask communications.

Format:



Field	Word	Bits	Description
Trace-1	S1	0-63	First word of intertask message
Trace-2	S2	0-63	Second word of intertask message
Task	S6	0-63	Task number to be readied

Field	Word	Bits	Description
T	S7	0	Intertask Message flag: if set, S1 and S2 hold an intertask message which is to be entered into the History Trace Table. S1 and S2 are ignored if T=0.
FC	S7	55-63	Request code (RTSS=14_8)

The RTSS request readies a system task and suspends the caller. Since this request is generally associated with intertask messages, the caller is allowed to place an entry in the History Trace Table as part of the call (rather than requiring an additional EXEC request to perform the trace).

If the sign bit of S7 is clear in the request, EXEC readies the target task with no additional processing. If the sign bit of S7 is set in the request, then S1 and S2 in the requesting task's Exchange Package are assumed to contain a two-word entry for the History Trace Table. (The calling task is suspended in any case.) The message set produced when the sign of S7 is set has the following format:

Trace code	Words 1 and 2 in the history trace
012	S1 and S2 from requesting task's Exchange Package
042	"RDY-SUS *xxx->yyy*" in ASCII

xxx are the first three characters of the requesting task name, and *yyy* are the first three characters of the target (readied) task name. 042 is the history trace code for EXEC ASCII message, and 012 is the history trace code for intertask message.

ERROR CONDITIONS:

(P) exits:

(S6)=ERTNX (003) if target task does not exist

$STOP064: when task ID in request indicates the calling task.

Connect user task to CPU request (RCP=16)

The connect user task to CPU request logically connects a user task to a physical CPU.

Format:

```
          0     8      16      24      32      40      48      56   63
        ---------------------------------------------------------------
   S6   |>|@RCCLN|//////////////////////////|        @RCTXT            |
        ---------------------------------------------------------------
   S7  /|                        @RCTS                      | @RCFC    |
      / ---------------------------------------------------------------
 @RCLDCL
```

Field	Word	Bits	Description
@RCLDCL	S6	0	=1 to load CRAY X-MP cluster from JTA
@RCCLN	S6	1-5	CRAY X-MP cluster number for user task
@RCTXT	S6	32-63	STP-relative TXT address of user task to connect
@RCTS	S7	0-54	Number of CPU cycles in time slice
@RCFC	S7	55-63	Request code (016_8)

The job scheduler task (JSH) uses this request to associate a user task with a particular CPU. When a user task is connected and no system tasks are ready to execute, EXEC's task scheduler will exchange to the user task connected to its CPU.

Processing for this request consists of the following:

- Ensure CPU not already connected to another user task.
- Ensure cluster number selection valid for machine type.
- Calculate EXEC-relative addresses for TXT, TCB, JTA, task status block (TSB), Exchange Package, and store in PWS.
- Load CRAY X-MP cluster if requested to do so, and execute on CRAY X-MP.
- Set connected task information into TXT and TSB (if present).
- Set up timer event for time slice expiration.
- If SPY is enabled for user, set up timer event for SPY event.

ERROR CONDITIONS:

(P) exits:

(S6)=ERNCP (014) if a user task is already connected in the requesting CPU

(S6)=ERCLN (035) if an invalid cluster number is specified

(S6)=ERMT (034) if cluster loading is selected on a non- CRAY X-MP mainframe

$STOP034: occurs if a zero time slice is specified on the connect

$STOP044: occurs if a zero SPY time slice is encountered

$STOP045: occurs if the SETCL macro does not find the requested cluster number in its tests; indicates a hardware problem.

Disconnect user task from CPU request (DCP=17)

The job scheduler task (JSH) uses this request to disassociate a user task from a physical CPU to which it had been previously connected.

Format:

```
       0       8       16      24      32      40      48      56     63
      +-----------------------------------------------------------------+
  S6  |  |/// @DCSTCL ////////////////|       @DCTXT                    |
      +-----------------------------------------------------------------+
  S7  |/////////////////////////////////////////////////////////| @DCFC |
      +-----------------------------------------------------------------+
```

Field	Word	Bits	Description
@DCSTCL	S6	0	=1 to store X-MP cluster in the JTA
@DCTXT	S6	32-63	STP-relative TXT address of user task to disconnect
@DCFC	S7	55-63	Request code (017_8)

JSH uses this request to remove the association between a user task and a particular CPU and CRAY X-MP cluster.

Processing for this request consists of the following:

- Ensure that the specified user is connected to the requested CPU.

- Ensure that the TXT address specified in the request matches that of the connected user.
- Cancel time slice event.
- Cancel SPY time slice event.
- If requested to save cluster, and a cluster is assigned, save cluster in the JTA.
- Copy Exchange Package to the TCB, if it isn't already there.
- Clear all connected user fields from the PWS.

ERROR CONDITIONS:

(P) exits:

(S6)=ERNTC (033) if no user task is connected to the CPU

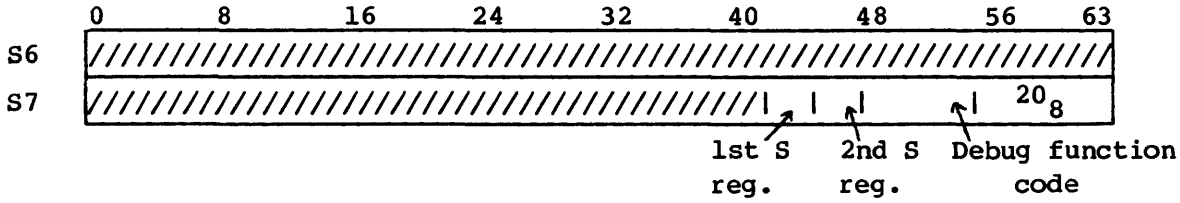
$STOP016: occurs if the TXT address passed in the request does not match the address of the TXT entry connected

$STOP047: occurs if the SETCL macro does not find the cluster number from the PWS in its tests

Post message in history buffer request (POST=20)

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

Format:



Field	Word	Bits	Description
First S register	S7	42-44	Ordinal of S register containing first word of information to post
Second S Register	S7	45-47	Ordinal of S register containing second word of information to post

Field	Word	Bits	Description
Debug function code	S7	48-54	History trace function number (see section 2.12.1 of this manual)
FC	S7	55-63	Monitor request number (20_8)

The flow is as follows:

1. Set up call to EXEC subroutine DEBUG by moving 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 issue location stamp.

Set memory size request (SMSZ=21)

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

Format:

	0	8	16	24	32	40	48	56	63
S6	//					New limit address			
S7	//							21_8	

Field	Word	Bits	Description
LA	S6	40-63	New system limit address
FC	S7	55-63	Monitor request number (21_8)

The processing consists of setting the new system limit address in all system exchange packages.

Packet I/O request (PIO=22)

This request invokes the I/O Subsystem driver called the IOP driver.

Format:

	0	8	16	24	32	40	48	56	63
S6	//						SCT		
S7	//						PFC	22_8	

Field	Word	Bits	Description
SCT	S6	40-63	Subsystem Control Table address (absolute)
PFC	S7	52-54	Function code: 0 Clear 1 Send packet 2 Receive packet
FC	S7	55-63	Monitor request number (22_8)

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 can only thereafter be changed if a task issues a new PIO clear function. A task monitors the status of the subsystem by inspecting the status field (SCSTAT) of the SCT table. The following flags are maintained by the IOP driver in the SCSTAT field:

• 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 is restarted or initialized. A task can then acknowledge reset by issuing a PIO clear request to clear the SCRST flag. The driver cannot accept a clear until all input is processed, which means 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 is received when the SCIR flag is set, and a packet is sent when all status flags are clear.

Boot a new system request (BOOT=23)

This request moves an image of an operating system down to the executable area.

Format:

```
      0       8       16      24      32      40      48      56    63
     ------------------------------------------------------------------
S6  |////////////////|        BOOTNW         |         BOOTAD          |
     ------------------------------------------------------------------
S7  |////////////////////////////////////////////////////////|  23₈    |
     ------------------------------------------------------------------
```

Field	Word	Bit	Description
BOOTNW	S6	16-39	Number of words in new system, including parameter file
BOOTAD	S6	40-63	Base address of new system, EXEC relative
BOOTFC	S7	55-63	Function code (23_8)

The flow is as follows:

1. Copy the new system down to location 0.
2. Set the system length in the new system's exchange package register S7.
3. Exchange to the new system's boot exchange package.

When the new system gets control, the flow is:

1. Save the system length.
2. Continue with system initialization.

Start system request (START=24)

This request starts the system after a system breakpoint is encountered or after a stop function is issued.

Format:

```
      0       8       16      24      32      40      48      56    63
     ------------------------------------------------------------------
S6  |//////////////////////////////////////////////////////////////////|
     ------------------------------------------------------------------
S7  |////////////////////////////////////////////////////////|  24₈    |
     ------------------------------------------------------------------
```

Field	Word	Bit	Description
FC	S7	55-63	Monitor request number (24_8)

The flow is as follows:

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)

This request stops the system except for entry of station debugging commands.

Format:

```
      0       8      16      24      32      40      48      56    63
     +---------------------------------------------------------------+
 S6  |///////////////////////////////////////////////////////////////|
     +---------------------------------------------------------------+
 S7  |//////////////////////////////////////////////////////|   25₈   |
     +---------------------------------------------------------------+
```

Field	Word	Bit	Description
FC	S7	55-63	Monitor request number (25_8)

The flow consists of setting the Alternate Task Scheduling flag. The alternate scheduling allows only SCP to execute so station debugging commands can be entered.

Display memory request (DMEM=26)

This request copies memory to a specified area. It is used to display memory during debugging.

Format:

```
      0       8      16      24      32      40      48      56    63
     +---------------------------------------------------------------+
 S6  |///////////////| Display area FWA       |  Buffer area FWA     |
     +---------------------------------------------------------------+
 S7  |///////////////|       Length           |///////////////|  26₈ |
     +---------------------------------------------------------------+
```

Field	Word	Bit	Description
Source	S6	16-39	Absolute address of first word of memory to copy for display

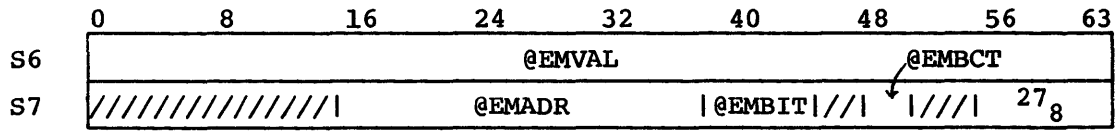
Field	Word	Bit	Description
Destination	S6	40-63	Absolute address of display buffer
Length	S7	16-39	Number of words to copy to display buffer
FC	S7	55-63	Monitor request number (26_8)

The flow consists of copying the memory block from the requested area to the display buffer.

Enter memory request (EMEM=27)

This request enters the bit string into memory at the specified bit position.

Format:



Field	Word	Bit	Description
@EMVAL	S6	0-63	Value to be entered; right-justified.
@EMADR	S7	16-39	Absolute address of memory word to modify
@EMBIT	S7	40-45	Starting bit position of field to modify within memory word
@EMBCT	S7	48-53	Number of bits to modify in memory word
FC	S7	55-63	Monitor request number (27_8)

The flow is as follows:

1. Get value, bit position, bit count, and absolute address from the request.

2. Get the word to be modified.

3. Form and position a mask of bits to save in the requested word.

4. Shift the new value to the specified bit position.

5. Clear the old contents of the field and merge in the new value.

6. Update the memory copy of the word.

Display Exchange Package request (DXPR=30)

This request moves the contents of the Exchange Package and B0 to a buffer.

Format:

```
      0       8       16      24      32      40      48      56     63
     ---------------------------------------------------------------------
S6  |///////////////////////////////////////|      Buffer area FWA        |
     ---------------------------------------------------------------------
S7  |//////////////////////////////////////////////////|Task ID|   30₈    |
     ---------------------------------------------------------------------
```

Field	Word	Bit	Description
Buffer area FWA	S6	40-63	Absolute address of a 17-word buffer to receive the information
Task ID	S7	47-54	ID of the task being displayed. Nontask Exchange Packages can be displayed using the display memory request.
FC	S7	55-63	Monitor request number (30_8)

The flow is as follows:

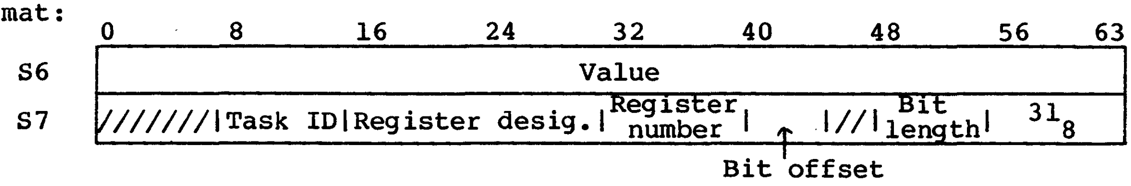
1. Copy Exchange Package to words 0 through 15 of buffer.

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

Enter Exchange Package register request (EXPR=31)

This request inserts the bit string into the specified Exchange Package register.

Format:



Field	Word	Bit	Description
Value	S6	0-63	Value to be entered
Task ID	S7	8-15	ID of the task being modified. Nontask Exchange Packages can be modified using the enter memory request.
Register designator	S7	16-31	Register designator (see below)
Register number	S7	32-39	Ordinal of designated register
Offset	S7	40-45	Starting bit position of value being entered
Bit length	S7	48-53	Number of bits in value being entered
FC	S7	55-63	Monitor request number (31_8)

The flow is as follows:

1. Determine word length and position of specified register in memory.
2. Shift value to desired position.
3. Merge into addressed memory word.

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

Set system breakpoint request (SBKPT=32)

This request sets a single or double breakpoint in the system by changing an instruction parcel to an error exit instruction with the breakpoint number in the rightmost bits. If a breakpoint exists 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.

Format:

```
        0       8      16      24      32      40      48      56     63
       ------------------------------------------------------------------
  S6  |////////////////|     Breakpoint        |      Breakpoint         |
      |                |  parcel address 2     |   parcel address 1      |
       ------------------------------------------------------------------
  S7  |////////////////////////////////////////| ^ |///////////|   32_8  |
       ------------------------------------------------------------------
                                             Breakpoint number
```

Field	Word	Bit	Description
Parcel 2	S6	16-39	Absolute parcel address of first breakpoint
Parcel 1	S6	40-63	Absolute parcel address of second breakpoint
Number	S7	40-42	Breakpoint number (0-7)
FC	S7	55-63	Monitor request number (32_8)

The flow is as follows:

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)

This request clears a system task breakpoint entry.

Format:

```
         0     8     16    24    32    40    48    56   63
        ----------------------------------------------------
   S6  |////////////////////////////////////////////////////|
       |----------------------------------------------------|
   S7  |/////////////////////////////////| ↑ |///////////| 33 |
        ---------------------------------------------------8-
                                      Breakpoint number
```

Field	Word	Bit	Description
Breakpoint number	S7	40-42	Breakpoint number (0-7)
FC	S7	55-63	Monitor request number (33_8)

The flow is as follows:

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 address.
5. Clear breakpoint table entry.

Report CPU use request (CPUTIL=34)

This request puts data on CPU use into the assigned buffer.

Format:

```
         0     8     16    24    32    40    48    56   63
        ----------------------------------------------------
   S6  |///////////////////| Buffer length |  Buffer address  |
       |----------------------------------------------------|
   S7  |///////////////////////////////////////////////| 34 |
        ---------------------------------------------------8-
```

Field	Word	Bit	Description
Buffer length	S6	24-39	Buffer size in words
Buffer address	S6	40-63	Absolute address of receiving buffer
FC	S7	55-63	Monitor request number (34_8)

The flow is as follows:

1. Validate buffer size.

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

Report task use request (TASKUTIL=35)

This request puts task usage data into the assigned buffer.

Format:

```
      0       8       16      24      32      40      48      56    63
     ┌───────────────────────────────────────────────────────────────┐
S6   │////////////////////////| Buffer length |     Buffer address   │
     ├───────────────────────────────────────────────────────────────┤
S7   │/////////////////////////////////////////////////////////| 35₈ │
     └───────────────────────────────────────────────────────────────┘
```

Field	Word	Bit	Description
Buffer length	S6	24-39	Buffer size in words
Buffer address	S6	40-63	Absolute address of receiving buffer
FC	S7	55-63	Monitor request number (35_8)

The flow is as follows:

1. Validate buffer size.

2. Put number of tasks into buffer.

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

Report EXEC request (EREQNT=36)

This request puts the EXEC request count of each task into the assigned buffer.

Format:

```
        0       8      16      24      32      40      48      56     63
       ---------------------------------------------------------------------
  S6   |/////////////////////////| Buffer length |     Buffer address      |
       |-------------------------------------------------------------------|
  S7   |/////////////////////////////////////////////////////////| $36_8$  |
       ---------------------------------------------------------------------
```

Field	Word	Bit	Description
Buffer length	S6	24-39	Buffer size in words
Buffer address	S6	40-63	Absolute address of receiving buffer
FC	S7	55-63	Monitor request number (36_8)

The flow is as follows:

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 that collect such data in the STT.

Report EXEC call counts request (ECALLCNT=37)

This request puts the number of EXEC requests of each type into the assigned buffer.

Format:

```
        0       8      16      24      32      40      48      56     63
       ---------------------------------------------------------------------
  S6   |/////////////////////////| Buffer length |     Buffer address      |
       |-------------------------------------------------------------------|
  S7   |/////////////////////////////////////////////////////////| $37_8$  |
       ---------------------------------------------------------------------
```

Field	Word	Bit	Description
Buffer length	S6	24-39	Buffer size in words
Buffer address	S6	40-63	Absolute address of receiving buffer
FC	S7	55-63	Monitor request number (37_8)

The flow is as follows:

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 that collect such data in the STT.

Report interrupt counts request (CHINTCNT=40)

This request puts interrupt counts of each channel and pseudo channel into the assigned buffer.

Format:

```
      0      8       16      24      32      40      48      56    63
     ┌──────────────────────────────────────────────────────────────┐
S6   │/////////////////////////| Buffer length |    Buffer address  │
     ├──────────────────────────────────────────────────────────────┤
S7   │///////////////////////////////////////////////////////| 40₈  │
     └──────────────────────────────────────────────────────────────┘
```

Field	Word	Bit	Description
Buffer length	S6	24-39	Buffer size in words
Buffer address	S6	40-63	Absolute address of receiving buffer
FC	S7	55-63	Memory request number (40_8)

The flow is as follows:

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.

Switch processors request (PSWITCH=41)

On CRAY X-MP mainframes, the switch processors request causes STP to be switched to the processor number specified in PN field.

Format:

```
      0      8      16      24      32      40      48      56   63
     ┌────────────────────────────────────────────────────────────┐
  S6 │//////////////////////////////////////////////////////////|PN│
     ├────────────────────────────────────────────────────────────┤
  S7 │/////////////////////////////////////////////////////|  41₈ │
     └────────────────────────────────────────────────────────────┘
```

Field	Word	Bit	Description
PN	S6	60-63	Ordinal of CPU to switch to
FC	S7	55-63	Monitor request number (41_8)

Processing for this request consists of the following:

- Increment the P register of the calling task by 2.
- Ensure that the calling task is the job scheduler, JSH.
- Suspend the calling task, and clear all system task scheduling information pertaining to it.
- Issue an interprocessor message to the other CPU, requesting that it complete processing of the PSWITCH request.
- Clear all operating system locks, and wait for the other CPU to enter the operating system.
- Jump to LOCKOS to reenter single-threaded operation.

ERROR CONDITIONS:

$STOP012: occurs if the calling task is not JSH.

Dump CRAY X-MP cluster registers request (DUMPCL=42)

System tasks use the dump CRAY X-MP cluster registers request to obtain a copy of all registers in a CRAY X-MP cluster.

Format:

```
      0      8      16      24      32      40      48      56   63
     ┌────────────────────────────────────────────────────────────┐
  S6 │@DUCLN |/////////////////////////|          @DUADDR          │
     ├────────────────────────────────────────────────────────────┤
  S7 │/////////////////////////////////////////////////////| @DUFC│
     └────────────────────────────────────────────────────────────┘
```

Field	Word	Bits	Description
@DUCLN	S6	0-7	CRAY X-MP cluster to be dumped
@DUADDR	S6	32-63	STP-relative address of buffer to receive cluster image; must be at least C@CLSIZE words long.
@DCFC	S7	55-63	Request code (042_8)

Processing for this request consists of the following:

- Ensure that a buffer was specified, that the request is being made on a CRAY X-MP mainframe, and that the cluster number is valid.
- Enter the selected cluster, dump the cluster to the specified buffer, and reenter the system cluster.

ERROR CONDITIONS:

(P) exits:

(S6)=ERIA (036) if a zero buffer address is specified

(S6)=ERMT (034) if the request is made on a machine other than a CRAY X-MP mainframe

(S6)=ERCLN (035) if an invalid cluster number is specified

$STOP048: occurs if the SETCL macro does not find the requested cluster number in its tests; indicates a hardware problem.

2.6.2 EXEC ERROR CODES

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

Symbol	Octal code	Processing routine and significance	
ERNTS	1	<R001>	No task space left
ERIDA	2	<R011>	No task assigned
ERTNX	3	<R000, R002, R004, R014, R031> Task does not exist	

Symbol	Octal code	Processing routine and significance	
ERRAT	4	<R004>	Resource already assigned to a task
ERCHA	5	<R011>	Channel already active
ERITN	6	<R014>	Illegal task call (also returned if unknown task makes a request)
ERBPN	11	<R032, R033>	Illegal breakpoint number
ERBPB	12	<R032>	Address already has a breakpoint
ERBFD	13	<R027, R031>	Bad field definition
ERNCP	14	<R016>	Job already connected
ERGSY	15	<R021>	Disk malfunction
ERIPS	16	<R033>	Breakpoint invalid
ERIRN	20	<R031>	Illegal register name
ERQFULL	24	<R006>	Time queue is full
ERINB	25	<R034, R035, R036, R037, R040>	Insufficient buffer length
ERTALC	26	<R001>	Task already created
ERSID	27	<R022>	Source ID mismatch
ERBFC	30	<R022>	Bad function
ERTPB	31	<R022>	Task parameter block changed
ERICH	32	<R022>	Invalid channel number

2.7 FRONT-END DRIVER

The Front-end Driver (FED) physically controls I/O between the Cray mainframe and front-end computers attached directly to the Cray mainframe. In addition, it passes requests to the MIOP for I/O between the Cray mainframe and front-end computers attached to an I/O Subsystem.

The Front-end Driver is invoked by EXEC monitor request 5. The Station Call Processor (SCP) is the only task to use FED. FED requires the use of COS front-end protocol. See the Front-end Protocol Internal Reference Manual, CRI publication SM-0042, for detailed information on COS protocol.

FED processes task requests for channel control and front-end I/O. FED performs hardware-level error recovery and some logical error recovery. Most logical error recovery is provided by the requesting task.

2.7.1 THEORY OF OPERATION

FED processes the following operations:

- Channel on

- Channel off
- Output to front end

Channel on operation

The following processing flow performs the channel on operation:

```
Assign channel pair to requesting task
Master clear interface
IF direct coupled interface THEN
  Send restart message
ENDIF
WHILE channel remains on
  Wait for input message
  IF direct coupled interface THEN
    Terminate any output active on channel pair
  ENDIF
  IF input error THEN
    Increment error count
    IF error retry count not exceeded THEN
      Send MESSAGE ERROR message
    ELSE
      Exit and wait for operator intervention (channel hung)
    ENDIF
  ELSE
    Notify requesting task of an input message
  ENDIF
  Check for deferred output operation on this channel
ENDWHILE
```

Channel off operation

The following processing flow performs the channel off operation:

```
Deassign channel
Terminate any I/O on channel
Reject further interrupts from channel
```

Output to front-end operation

The following processing flow performs the output to front-end operation:

```
Send output message
IF network error THEN
```

```
      IF local adapter busy THEN
        Defer output operation
      ELSE
        Increment error count
        IF error retry limit not exceeded THEN
          Resend output message
        ELSE
          Exit and wait for operator intervention (channel hung)
        ENDIF
      ENDIF
    ENDIF
```

2.7.2 SYSTEM TABLES USED BY THE FRONT-END DRIVER

FED uses the following system tables:

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

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

Channel Table (CHT)

A channel pair is assigned to the requesting task when the channel is turned on, by placing the task parameter block address and LIT entry address in the CHT entries for the channel pair. The interrupt handler address is updated by the FED after each interrupt.

Channel Extension Table (CXT)

The Channel Extension Table (CXT) has an entry for each I/O Subsystem channel ordinal. A request is passed to the MIOP by building a message in the entry for the channel ordinal specified in the monitor request. Further communication with the MIOP is handled by the IOP driver.

Link Interface Table (LIT)

The Link Interface Table (LIT) has an entry for each channel configured for front-end communications. The entry address is passed in the monitor request. FED uses the LIT to control the interfaces and to multiplex multiple logical front ends on one channel.

Link Extension Table (LXT)

The Link Extension Table (LXT) has an entry for each logical front-end ID configured. It is used for communication between EXEC and STP. An entry is allocated by FED upon receipt of a logon message, and released after an output operation if the OFF bit is set. Receipt of a front-end message is signaled by FED with the INT (interrupt) bit. FED does not modify an entry after setting INT until the next output request is received for that entry.

2.7.3 FRONT-END DRIVER PROCESSORS

The Front-end Driver consists of a request dispatcher (R005) and the following request processors:

Processor	Function
R005C	IFC (channel coupler) request processor
R005I	IOP request processor
R005N	NSC (HYPERchannel) request processor

R005 request dispatcher

The R005 request dispatcher determines the pertinent request processor and transfers control to it. R005 processing is as follows:

```
IF nonzero channel ordinal THEN
  Exit to R005I
ELSE
  IF channel off THEN
    Perform processing and exit
  ELSE
    Set up tables
    Exit to R005C or R005N, depending on type
  ENDIF
ENDIF
```

The following R005 subroutines are available to all request processors:

Processor	Function
FNDLX	Looks up an LXT entry

Processor	Function
GETLX	Allocates an LXT entry
ITERM	Performs input termination processing
MVLCP	Moves an LCP from an LIT entry to an LXT entry
OTERM	Performs output termination processing
TACT	Activates requesting task

FNDLX - Processing is as follows:

```
Look up:
  An LXT entry allocated to a given ID
Return to calling routine
```

GETLX - Processing is as follows:

```
  LABEL 1
  Find an LXT with the same ID as the input LCP.
  If no matching LXT entry:
       Get an inactive LXT.
  If a free entry exits:
       Set up the free entry.
  ELSE
       Set an error and return.
  ENDIF
ELSE
  If the LXT is not on the same channel:
       Set an error and return.
ENDIF
  If the LXT is logging off:
  If the LXT is ready for input:
       Clear the ID (deactivate the LXT).
       Jump to LABEL 1.
  ELSE
  Reject the logon (allowing SCP to finish current processing).
  ENDIF
ENDIF
  Set the relog flat in the LXT.
  ENDIF
```

ITERM - Processing is as follows:

```
Set LXT Entry Interrupt flag
Mark LXT entry ineligible for input (RDY=0)
```

```
Save channel information in LXT entry for next output operation
Increment message counters in LIT entry
Return to calling routine
```

<u>LPEND</u>

```
Find first LXT that has deferred output pending.
```

<u>OTERM</u> - Processing is as follows:

```
Increment message counters in LIT entry
IF the OFF bit is set in the LXT entry THEN
  Deallocate the entry
ENDIF
Return
```

<u>TACT</u> - Processing is as follows:

```
Activate the requesting task
Return to the calling routine
```

<u>R005C request processor</u>

R005C processing is as follows:

```
IF channel on THEN
  Call CCLR to master clear the interface
  Set up to write a restart message LCP; WLCP is interrupt handler
  Set up to read an LCP into the LIT entry; RLCP is interrupt handler
ELSE IF output THEN
  Mark the LXT entry eligible for input (RDY=1)
  Set up to write an LCP from the LXT entry; WLCP is interrupt handler
ENDIF
```

Each routine designates another routine as an interrupt handler if I/O is pending on the channel pair. The R005C routines are:

Processor	Function
CCLR	Master clears interface
CCLRB	Processes interrupt from reading input channel
CCLRC	Processes timeout interrupt and master clear channel
CCLRD	Processes interrupt from writing function code

Processor	Function
CHKSM	Checksums a given area
FOLD	Folds a 64-bit checksum
LIRCV	Processes input error
LORCV	Processes output error
RLCP	Processes interrupt from reading LCP
RLTP	Processes interrupt from reading LTP
RSSEG	Processes interrupt from reading subsegment
WLCP	Processes interrupt from writing LCP
WLTP	Processes interrupt from writing LTP
WSSEG	Processes interrupt from writing subsegment
WXLCP	Processes interrupt from writing error LCP
WXLTP	Processes interrupt from writing error LTP

CCLR/CCLRA - Processing is as follows:

```
Terminate any input or output on channel
Set up a timer for I@MCLDLY clock ticks; CCLRC is interrupt handler
Set up to read any input in LCP-size pieces; CCLRB is interrupt
handler
```

CCLRB - Processing is as follows:

```
Cancel the timer
Exit to CCLRA
```

CCLRC - Processing is as follows:

```
Issue the master clear sequence for low-speed asynchronous channel
IF a VAX channel type THEN
  Set up to write a SELECT function code; CCLRD is interrupt handler
ELSE
  Return to calling routine
ENDIF
```

CCLRD - Processing is as follows:

```
    Return to calling routine
```

CHKSM - Processing is as follows:

```
    Calculate checksum and return to calling routine
```

FOLD - Processing is as follows:

```
    Fold checksum and return to calling routine
```

LIRCV - Processing is as follows:

```
    Increment error counters
    IF error retry count exceeded THEN
      Exit
    ELSE IF output channel is not active THEN
      Format a message error message LCP
      Set up to write the error LCP; WXLCP is interrupt handler
    ENDIF
    Set up to read the next LCP; RLCP is interrupt handler
```

LORCV - Processing is as follows:

```
    Increment error counters
    IF no LXT entry is available for the ID THEN
      Set up to send a restart LCP; WLCP is interrupt handler
    ELSE
      Set up to resend last output LCP; WLCP is interrupt handler
    ENDIF
```

RLCP - Processing is as follows:

```
    Call DEBUG for history trace entries (HTFEI=7 and HTSCI=16)
    IF channel error or short input THEN
      Exit to LIRCV
    ELSE
      Stop any output in progress on channel
      IF a logon message THEN
        Set up to read short segment; RSSEG is interrupt handler
      ELSE IF a hardware (3xx) message error message THEN
        Exit to LORCV
      ELSE IF the ID has no LXT entry or no LXT entries are available THEN
        Exit to LIRCV
      ELSE
        Call MVLCP to move the LCP to the LXT entry
        IF a segment is present THEN
```

```
          Set up to read the first subsegment; RSSEG is interrupt handler
        ELSE IF checksumming enabled THEN
          Set up to read the LTP; RLTP is interrupt handler
          Call ITERM to perform input termination processing
          Call TACT to activate the requesting task
        ENDIF
      ENDIF
    ENDIF
```

RLTP - Processing is as follows:

```
    IF a channel error or short input THEN
      Exit to LIRCV
    ELSE
      Set up to read the next LCP; RLCP is interrupt handler
      Call ITERM to perform input termination processing
      Call TACT to activate the requesting task
    ENDIF
```

RSSEG - Processing is as follows:

```
    IF a channel error or short input THEN
      Exit to LIRCV
    ELSE IF a logon message segment THEN
      Call GETLX to allocate an LXT entry and move the logon segment
      IF no LXT entries are available THEN
        Exit to LIRVC
      ELSE
        Call MVLCP to move the LCP into the LXT entry
        Call DEBUG for history trace entry (HTSEG=15)
        Set up to read the next LCP; RLCP is interrupt handler
        Call ITERM to perform input termination processing
        Call TACT to activate the requesting task
      ENDIF
    ELSE IF no more subsegments are available
      Set up to read the next subsegment; RSSEG is interrupt handler
    ELSE IF checksumming enabled THEN
      Set up to read the LTP; RLTP is interrupt handler
    ELSE
      Call DEBUG for history trace entry (HTSEG=15)
      Set up to read the next LCP; RLCP is interrupt handler
      Call ITERM to perform input termination processing
      Call TACT to activate the requesting task
    ENDIF
```

WLCP - Processing is as follows:

```
    Call DEBUG for history trace entries (HTFEO=14 and THSCO=20)
    IF a segment is present THEN
```

```
      Set up to write the first subsegment; WSSEG is interrupt handler
    ELSE IF checksumming enabled THEN
      Set up to write the LTP; WLTP is interrupt handler
    ELSE
      Call OTERM to perform output termination processing
    ENDIF
```

WLTP - Processing is as follows:

```
    Call OTERM to perform output termination processing
```

WSSEG - Processing is as follows:

```
    IF no more subsegments are present THEN
      Set up and write the next subsegment; WSSEG is interrupt handler
    ELSE
      Make history trace entry (HTSEG=15)
      IF checksumming enabled THEN
        Set up to write the LTP; WLTP is interrupt handler
      ELSE
        Call OTERM to perform output termination processing
      ENDIF
    ENDIF
```

WXLCP - Processing is as follows:

```
    Call DEBUG for history trace entry (HTFEE=17)
    IF checksumming enabled THEN
      Call CHKSM and FOLD to calculate checksum; format error LTP
      Set up to write the error LTP; WXLTP is interrupt handler
    ENDIF
```

WXLTP - Processing is as follows:

```
    Clean up
```

R005I request processor

The processing for R005I is as follows:

```
    IF the operation is a channel on or channel off operation THEN
      Format an X packet in the CXT entry for ordinal specified
    ELSE IF the operation is an output operation THEN
      Format a B packet in the CXT entry for ordinal specified
    ENDIF
    Call IOPRDV/APENQ to queue the packet to the MIOP
```

Further communication with the MIOP is handled by the IOP driver (R022).

The APRCV subroutine recovers from an I/O Subsystem shutdown or restart. APRCV issues a K packet to the MIOP for each active CXT entry. The IOP driver uses the K packet upon receipt of an initialization sequence from the MIOP.

APRCV processing is as follows:

```
LOOP for all CXT entries
  IF entry is active THEN
    Call DEQ to obtain a packet from the free queue
    Build a K packet
    Call IOPDRV/APENQ2 to queue the packet to the MIOP
  ENDIF
ENDLOOP
```

R005N request processor

R005N processing is as follows:

```
IF a CHANNEL ON request
  Master clear the adapter.
  Set up to write a WAIT FOR MESSAGE function.
  Exit - input pending.
ELSEIF an output request
  Build the output message LCPE based on
  information received for that ID.
  If the adapter is busy (a non-WAIT FOR MESSAGE
  function or if output recovery in progress)
    Defer the current output operation.
    Exit.
  ELSE
    Clear outstanding WAIT FOR MESSAGE function
    with two END OPERATIONS.
    Get the adapter status.
    IF error
      Issue END OP to clear adapter.
    ELSEIF message received
      Defer current write request.
      Exit - input pending.
    ENDIF
  ENDIF
  Set up to write a transmit message function.
  Exit - output pending.
ELSE (unknown request)
  STOP
ENDIF
```

Each routine designates another routine as an interrupt handler if I/O is pending on the channel pair. Since the NSC adapter gives both an output interrupt and an input interrupt for each function, output interrupts require no special processing and are assigned to ENA in XPROC. The RO05N routines are:

Processor	Function
NCLR	Master clears adapter
NCLRA	Processes interrupt acknowledging clear-adapter function
NEND	Issues adapter end operation function
NENDA	Processes interrupt acknowledging end operation function
NETO	Processes interrupt from a time event expiration
NIRCV	Processes input error
NORCV	Processes output error
NPEND	Processes any pending output
NRLCF	Processes interrupt acknowledging wait-for-message function
NRLCP	Processes interrupt from reading LCP
NRSEG	Processes interrupt from reading segment
NWLCF	Processes interrupt acknowledging transmit-data function
NWLCP	Processes interrupt from writing LCP
NWSEF	Processes interrupt acknowledging transmit-data function
NWSEG	Processes interrupt from writing segment
NWXLC	Processes interrupt from writing error LCP
NWXLF	Processes interrupt acknowledging transmit-message function

Processor	Function
OUTFC	Writes a function code
STAT	Obtains adapter status
STATA	Processes interrupt from reading adapter status word

R005N also uses the alternate entry point MVLCE of routine MVLCP to perform the move of both LCPE and LCP from the LIT to the LXT.

NCLR/NCLRA - Processing is as follows:

```
Save return address in LIT entry
Terminate any input or output active on channel
Set up to write a clear-adapter function; ENA is interrupt handler
Set up to read the acknowledgment; NCLRA is interrupt handler
```

NCLRB - Processing is as follows:

```
IF channel error or adapter error THEN
  Increment error counters
  IF error retry limit exceeded THEN
    Exit
  ELSE
    Exit to NCLRA
  ENDIF
ELSE
  Return to calling routine
ENDIF
```

NEND - Processing is as follows:

```
Ensure no interrupts.
Set up return address.
Issue END OP function.
```

NENDA - Processing is as follows:

```
Ensure no output interrupts.
Cancel time event.
Set up return address.
```

NETO - Processing is as follows:

```
IF time out recovery already in progress
  Set channel hung.
  Exit.
```

```
ENDIF
Increment the timeout counter.
Issue an NSC END operation to clear errors.
IF output in progress
  Increment output error count for this LXT.
  IF error count exceeded
    EXIT - check for transfer pending.
  ENDIF
  Set up to restart output.
ENDIF
Exit - check for transfer pending.
```

NIRCV - Processing is as follows:

```
Save message error subcode.
Ensure input and output channels are inactive.
Issue an ENDOP to clear the adapter.
Increment error counters in the LIT
IF too many errors
  Clear error counter, and increment retry
  counter exceeded.
  Exit - check for transfer pending.
ENDIF
If message has no source or destination
  Increment the unknown interrupt counter.
EXIT - check for transfer pending.
ENDIF
  Set up to write a transmit message function.
EXIT - input pending.
```

NORCV - Processing is as follows:

```
Deactivate both sides of the channel pair.
Issue an END OP to clear the adapter.
Increment the error counters in the LIT.
Set up to write a transmit message function.
IF within error limit
  Wait before retrying. This means exit
  and reenter here.  Issue a Wait For Message
  during the delay time.
  IF output has been completed when the delay
  expires
    Clear error counts.
IF current function is WAIT FOR MESSAGE
  Clear WAIT FOR MESSAGE with END OF.
  EXIT - check for transfer pending.
ELSE
  EXIT - to ENA.
```

```
    ENDIF
  ELSE
      IF function is not WAIT FOR MESSAGE
        Requeue output operation.
        EXIT - to ENA.
    ELSE
      Clear WAIT FOR MESSAGE with END OF
      Set up to retransmit the message.
      EXIT - output pending.
    ENDIF
  ENDIF
      Exit - output pending.
    ELSE
      Clear error count, and increment retry
      count exceeded.
      Exit - check for transfer pending.
    ENDIF
```

NPEND - Processing is as follows:

```
Get adapter status.
IF status error
  Issue END OP to clear adapter.
ELSEIF message received
  Setup to input message.
  Exit - input pending.
ENDIF
IF error message pending
  Set up to write error message.
  Exit - output pending.
ENDIF
IF output pending and no output recovery in progress
  Look for pending LXT.
  IF LXT found
    Set up to write TRANSMIT MESSAGE function.
    Exit - output pending.
  ELSE
    Clear output pending count.
  ENDIF
ENDIF
Set up to write WAIT FOR MESSAGE function.
Exit - input pending.
```

NRLCF - Processing is as follows:

```
Ensure the output channel is inactive.
Set up to write an INPUT MESSAGE function code and read the message
proper (LCPE/LCP) into the LIT entry.
Exit.
```

NRLCP - Processing is as follows:

```
Ensure the output channel is inactive.
Cancel the time event.
Record the input LCP in the history trace buffer.
IF a channel error, short LCP, or adapter error
  Exit - input error.
ENDIF
IF a LOGON message
  Set up to write an INPUT DATA function, and
  read the short LOGON segment.
  Exit - input pending.
ENDIF
IF a MESSAGE ERROR message with a 3xx subcode
  Look up the LXT entry for input source ID.
  IF no matching LXT
    Exit - input error.
  ENDIF
  Exit - output error.
ENDIF
IF LXT entry not ready for input
  Exit - input error.
ENDIF
Move the LCPE and LCP into the LXT entry.
IF a data segment expected
  Set up to write the INPUT DATA function code
  and read the segment.
  Exit - input pending.
ELSE
  IF associated data present
    Exit - input error.
  ENDIF
  Issue END OP to complete transfer.
  Perform input termination.
  Exit - check for transfer pending.
ENDIF
```

NRSEG - Processing is as follows:

```
Ensure the output channel is inactive.
Cancel the time event.
IF a channel error, short segment, or adapter error
  Exit - input error.
ENDIF
Issue an END OP to complete the transfer.
IF a LOGON message
  Allocate an LXT entry.
  IF no available entries
```

```
      Exit - input error.
    ENDIF
    Move the LCPE and LCP into the LXT.
  ELSE
    Get the LXT address for this message.
  ENDIF
  Record the input segment in the history trace.
  Perform input termination.
  Exit - check for transfer pending.
```

<u>NWLCF</u> - Processing is as follows:

```
  Ensure no output interrupts.
  Cancel the time event.
  Set up to write the message proper (LCPE/LCP) from the LXT.
  Exit.
```

<u>NWLCP</u> - Processing is as follows:

```
  Ensure output channel is inactive.
  Cancel the time event.
  Record the output LCP in the history trace buffer.
  IF adapter status good
    IF there is a segment
      Set up to write a TRANSMIT LAST DATA function.
      Exit - output pending.
    ELSE
      Perform output termination.
      Exit - check for transfer pending.
    ENDIF
  ELSE
    IF status indicates no message received
      Exit - output error.
    ENDIF
    Issue an END UP to clear the adapter.
    Defer current operation.
    Exit - input pending.
  ENDIF
```

<u>NWSEF</u> - Processing is as follows:

```
  Ensure the output channel is inactive.
  Cancel the time event.
  Set up to write the segment.
  Exit.
```

NWSEG - Processing is as follows:

```
Ensure the output channel is inactive.
Cancel the time event.
Record the output segment in the history trace.
Perform output termination processing.
Exit - check for transfer pending.
```

NWXLC - Processing is as follows:

```
Ensure the output channel is inactive.
Cancel the time event.
Record the LCP in the history trace.
Get the adapter status.
IF the status indicates an error
  IF message received
    Issue an END OP to clear the adapter.
    Defer the current output operation.
    Exit - process forced input.
  ELSE
    Increment retry count exceeded.
    Exit - check for transfer pending.
  ENDIF
ENDIF
Set up to write a wait-for-message function; ENA is interrupt handler
Exit - output pending
```

NWXLF - Processing is as follows:

```
Ensure the output channel is inactive.
Set up to write the message proper(LCPE/LCP).
Exit.
```

OUTFC - Processing is as follows:

```
Ensure input channel inactive.
Set up to input response.
Output function code.
IF not WAIT FOR MESSAGE or CLEAR ADAPTER functions
  Set time event timer.
ENDIF
WHILE time to wait not exceeded
  EXITIF any I/O interrupt received.
  Decrement time to wait.
ENDWHILE
```

STAT - Processing is as follows:

```
Save return address in the LIT entry
Issue STATUS function.
Set up to write a status function; ENA is interrupt handler
Set up to read the acknowledgment; STATA is interrupt handler
```

STATA - Processing is as follows:

```
Determine response code
Exit
```

OUTFC - Processing is as follows:

```
Ensure input channel inactive.
Set up to input response.
Output function code.
IF not WAIT FOR MESSAGE or CLEAR ADAPTER functions
  Set time event timer.
ENDIF
WHILE time to wait not exceeded
  EXITIF any I/O interrupt received.
  Decrement time to wait.
ENDWHILE
```

2.7.4 FRONT-END DRIVER ERROR RECOVERY

The Front-end Driver (FED) attempts recovery from hardware-related errors. In addition, it detects and attempts recovery from some software-related errors (for example, lack of table space).

An error condition results in one of the following:

- Retry of the operation that caused the error

- Generation and transmission of a Message Error message

If error conditions persist and the error retry limit for a channel is exceeded, operator intervention is required.

A typical sequence for recovering from multiple errors is as follows:

1. At the master operator station, turn the affected channel off.

2. Terminate the front-end stations using the affected channel (perhaps turn off the front-end channel).

3. If necessary, manually master clear the interfaces.

4. Turn on the channel.

5. Restart the front-end stations including the front-end channel.

If this sequence fails, contact the Cray Research engineers.

R005C (IFC interface) error processing

An input error causes the issue of a Message Error message to the front end. If the input error retry count is exceeded, operator intervention is required. The error retry limit for R005C is 8192.

The following circumstances indicate an input error:

- The Channel Error flag is set.
- A transfer length error (short transmission) occurs.
- No LXT entries are available for a front-end log on.

An output error causes a retry of the last operation by the front end. If the output error retry count is exceeded, no further retries are attempted, but the channel remains active.

A Message Error message with an octal 3*xx* message subcode indicates an output error.

R005I (I/O Subsystem) error processing

All error processing is handled by the I/O Subsystem. See the IOS Software Internal Reference Manual, CRI publication SM-0046, for details.

R005N (NSC HYPERchannel interface) error processing

An input error results in a Message Error message to the front end. If the input error retry count is exceeded, no further retries are attempted, but the channel remains active. The error retry limit for R005N is 10.

The following circumstances indicate an input error:

- The Channel Error flag is set.

- A transfer length error (short transmission) occurs.
- An invalid HYPERchannel adapter status is returned.
- No LXT entries are available for a front-end log on.

An output error causes a retry of the last operation by the front end. If the output error retry count is exceeded, no further retries are attempted, but the channel remains active.

The following circumstances indicate an output error:

- An invalid HYPERchannel adapter status is returned.
- A Message Error message with an octal 3*xx* message subcode is received.

2.8 DISK/SSD DRIVER

The Disk/SSD Driver controls the following devices connected to a mainframe I/O channel:

- DCU-2 Disk Controller
- DCU-3 Disk Controller
- SSD (Solid-state Storage Device)

Each disk controller can drive from one to four disk storage units of the following types:

- DD-19 Disk Storage Unit
- DD-29 Disk Storage Unit

As an option, an SSD can be part of the configuration.

- On the CRAY-1 M Series and CRAY-1 S Series mainframes, the SSD is controlled by a High-speed Channel Controller (HSC) which connects to a 6-Mbyte channel pair. The HSC moves data to and from the SSD over a 100-Mbyte channel.
- On the CRAY X-MP mainframe, the SSD is connected directly to the mainframe through a 1250-Mbyte channel. Note that only one half of a channel pair is required to control a CRAY X-MP SSD, but the pair must be configured.

2.8.1 DISK/SSD DRIVER TABLES

The parameters in the requesting task's S6 and S7 registers specify table addresses and a channel pair number to use for all devices controlled by the R011 monitor request.

The monitor request includes addresses of the following tables:

DCT	Device Channel Table
EQT	Equipment Table

Device Channel Table (DCT)

The DCT contains the channel characteristics.

Equipment Table (EQT)

The EQT contains information on the type of I/O request and the device characteristics (disk storage unit type, SSD, and so on).

2.8.2 R011 MONITOR REQUEST

R011 checks the validity of the request parameters. If there is an illegal value or if the request would interfere with a request already in progress (except for master clear requests), R011 immediately makes an error return.

A normal return is scheduled if the request is well formed.

R011 selects the initial processor depending on device characteristics. Once the parameter validation is performed, one of the following processors is selected:

Processor	Description
DDI	DD19/29 disk request initialization
SSREQ	CRAY-1 M Series or S Series SSD request
XSREQ	CRAY X-MP Series SSD request

R011 is interrupt driven and executes a request in short bursts. Each processor selects the next processor to execute upon receipt of an interrupt. Time between interrupts is available to the rest of the system for task or user job execution.

R011 informs its calling STP task of the progress of the request after each transfer of a sector (if I/O interrupts occur after each sector is transferred) and at the completion of the request. The Disk Queue Manager (DQM) is the only task that calls R011.

2.8.3 LOST DISK INTERRUPTS

An interrupt could fail to occur due to hardware failure. Therefore, R011 protects itself by scheduling a timeout interrupt for each request to R011. As a result, each execution of Interchange compares the current contents of RTC (real-time clock) to the timeout value. Interchange gives control to R011 if the timeout occurs.

Exchanges or other interrupts might not occur for an extended period. The MCU real-time interrupts should be enabled, if available, to ensure frequent execution of Interchange if the computer is not equipped with a programmable clock. The time delay scheduled for timeout reflects the magnitude of the request to R011 while being liberal enough to avoid needless timeouts.

A single R011 request can involve many interrupts; thus, the single timeout scheduled per R011 request acts as a blanket protection.

Lost interrupts are rare; generally, only expected interrupts occur.

When the request completes, the timeout is released.

2.8.4 STATUS CHECKING AND ERROR RECOVERY

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

2.8.5 HARDWARE SEQUENCES FOR SAMPLE REQUESTS

This subsection assumes the reader is familiar with the Mass Storage Subsystem Hardware Reference Manual, CRI publication HR-0630. The processing sequence for several requests is presented here.

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

Processing is as follows:

1. Master clear channel with recommended I/O master clear sequence.
2. Reserve unit.
3. Read subsystem status.
4. Read fault status.
5. Read interlock status.
6. Read cylinder status.
7. Read head status.
8. Read sector status.
9. Read offset status.
10. Release unit, clear fault, and return to cylinder 0.
11. Reserve unit.
12. Release unit, clear fault, and return to cylinder 0.
13. Reserve unit.

Margin select

The margin select is driven by the Margin Select Table in EXEC. The table is initialized for 40 retries, starting at the smallest offset and working out to two-thirds of the maximum offset. Each word in the Margin Select Table contains eight margin values, one per byte.

2.9 PACKET I/O DRIVER

The Packet I/O Driver consists of two major parts:

1. The MIOP driver, which controls the 6-Mbyte channel to the Master I/O processor in the I/O Subsystem.

2. Packet queueing, which routes packets among three areas of the system:

   - STP tasks

   - EXEC

   - I/O Subsystem

   Packets can originate in and be sent to any of these areas.

2.9.1 PACKET I/O DRIVER TABLES

The following tables are used by the Packet I/O Driver:

APT	Any Packet Table
CXT	Channel Extension Table
FIQ	Free Input Queue Table
FOQ	Free Output Queue Table
QCT	Queue Control Table
SCT	Subsystem Control Table

Any Packet Table (APT)

The APT defines most of the packet formats and all of the packet formats recognized by EXEC.

Channel Extension Table (CXT)

The CXT controls front ends connected through the I/O Subsystem. Each IOS channel ordinal has one entry for handling one or more of the logical front-end IDs.

Free Input Queue Table (FIQ)

The FIQ contains input packets. The packet to be read from the MIOP contains "NEXTPACK" in ASCII replicated throughout.

Free Output Queue Table (FOQ)

The FOQ contains pointers to queued output packets.

Queue Control Table (QCT)

The QCT is a general format for tables manipulated by the EXEC queue management subroutines. Specific tables using this format are the FIQ, FOQ, and SCT.

Subsystem Control Table (SCT)

The SCT contains an entry for each type of packet EXEC can receive from the MIOP or send to STP. Each entry contains the address of a routine that either processes the packet or forwards it to an STP task for processing.

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

2.9.2 PACKET DESCRIPTION

The unit of information passed is known as a packet and is always six 64-bit words long. The Any Packet Table (APT) describes most of the formats the packet can take. The packet always has a 16-bit destination ID (DID) and a 16-bit source ID (SID) used by the Packet I/O Driver to route the packet to its destination.

The following ASCII identifiers are valid in the SID and DID fields. The identifiers are right-justified and null (binary zero) filled.

Identifier	Description
C1	Cray mainframe identifier
EX	EXEC identifier
A	Disk I/O
B	Front-end I/O
C	Error message
D	Tape I/O
E	Echo
G	Tape configuration
I	Initialization part 1
J	Initialization part 2
K	Kernel request
N	Null request
S	Statistics request

2.9.3 R022 MONITOR REQUEST

STP tasks can send packets to and receive packets from the I/O Subsystem with a R022 monitor request (Packet I/O). STP tasks can also receive packets from EXEC using the R022 request. See the PIO request in section 2.6.1 for details of this R022 monitor request.

2.9.4 MIOP DRIVER PROCESSORS

The following processors are interrupt-driven:

Processor	Description
APIIP	Input interrupt handler; packet received from MIOP.
APOIP	Output interrupt handler; MIOP has received packet.

The APIIP routine uses the Subsystem Control Table (SCT) to determine the packet queueing processor for the packet.

2.9.5 PACKET QUEUEING PROCESSORS

The following processors are used by the MIOP driver to process packets from the I/O Subsystem and are also used by EXEC to send packets to STP tasks.

Packet	Processor	Action taken
A	APXP	Forward packet to DQM
B	APBP	Forward packet to SCP, or process request
C	APXP	Forward packet to MEP
D	APXP	Forward packet to TQM
E	APEP	Process request (echo the packet)
I	APIP	Process request (Subsystem is down)
J	APJP	Process request (Subsystem is up)
N	APNP	Process request (ignore packet)
S	APSP	Process request (return statistics)

2.10 MEMORY ERROR CORRECTION

Memory error correction logs memory errors in an Executive table (MEL) at the time of interrupt. In addition, all memory errors are logged with MSG in the System Log except those which are double bit in nature and have forced a STOP at the time of interrupt. Between the MEL table and the System Log, all memory errors should be fairly easy to locate. When a memory error occurs, it is logged in the MEL. The MEL format is illustrated in figure 2-6.

A STOP (see section 2.13.2) occurs immediately after receiving the interrupt if any of the following conditions occur:

- Double-bit count (I@MEUCT) is exceeded ($STOP037).
- Population count of the syndrome bits is all zero ($STOP038) indicating the hardware reported an error with contradictory syndrome bits.
- Decoded syndrome bits do not match the correctable/uncorrectable code contained within the exchange package ($STOP039).
- EXEC idle loop detects a multibit memory error ($STOP040).
- Multibit memory error occurs while STP is executing ($STOP041).
- Multibit error occurs during an I/O reference ($STOP042).
- An IOP packet cannot be obtained to send the error packet to MEP ($STOP031).

If an uncorrectable memory error stops the system, the MEL should contain sufficient information, through raw dump or system dump, to isolate the error and failing module.

	0 8 16 24 32 40 48 56 63
0	Total Error Count
1	Single-bit Count
2	Double-bit Count
3	Current Bank
4	Current Chip Select
5	Current Syndrome
6	Current Error Type (Correctable/Uncorrectable)
7	Current RTC
8	Last Bank
9	Last Chip Select
10	Last Syndrome
11	Last Error Type (Correctable/Uncorrectable)
12	Last RTC

Figure 2-6. Memory Error Log (MEL)

Messages from EXEC to the Message Processor task (MEP) consist of the standardized Any Packet Table (APT) header (1 word) followed by five words of memory error information as follows:

	0	8	16	24	32	40	48	56 63
0	DID		SID		//////////////////////////			FC
1	/SYS//	MF	BANKS	CHIP	CONF	////////////////		JXO
2	JN							
3	ET ////////// RM		CODE		SYN	ERROR ADDRESS		
4	////////////////		BASE ADDRESS			P ADDRESS		
5	RTC							

Field	Word	Bits	Description
DID	0	0-15	Destination ID (ASCII "C")
SID	0	16-31	Source ID (ASCII "EX")
FC	0	56-63	Function code (6)

Field	Word	Bits	Description
SYS	1	0	If set, error occurred in system
MF	1	8-15	Mainframe type as follows: @CRAY1=1 @CRAY1S=2 @CRAYXMP=3
BANKS	1	16-23	Number of banks in mainframe (C@MMBANK)
CHIP	1	24-31	Chip size as follows: @M1KCH=1 1024 bits @M2KCH=2 2048 bits @M4KCH=3 4096 bits
CONF	1	32-39	Memory configuration: @MLEFT=1 @MRIGHT=2 @BOTH=3
JXO	1	56-63	Job Execution Table offset if error occurred in job; otherwise, 0.
JN	2	0-63	Job name of job in which error occurred. If error occurred in STP, this field is filled with ASCII "STP".
ET	3	0-1	Error type: 10 Uncorrectable 01 Correctable
RM	3	14-15	Read Mode: CRAY-1 mainframes: 0 Scalar 1 I/O 2 Vector, B or T 3 Instruction fetch CRAY X-MP mainframes: 0 I/O 1 Scalar 2 Vector, B or T 3 Instruction fetch

Field	Word	Bits	Description
CODE	3	16-31	Code indicating type of error encountered: 0 No error encountered 1 Check bit 2 Double bit 3 Single bit 4 Multiple bit
SYN	3	32-39	Syndrome bits
ERROR ADDRESS	3	40-63	Error address (see below)
BASE ADDRESS	4	16-39	Base address
P ADDRESS	4	40-63	P address of interrupted exchange package
RTC	5	0-63	RT value at time of error interrupt

The error address in word 0 has different values based upon which mainframe type is being used. See table 2-1 for the error address interpretation.

Also note that MEP communicates only the last five words of this message to the MSG task (through PUTREQ) when requesting entry of the error in the System Log.

Table 2-1. Address bits in word 0, depending on mainframe models

Field	Description	CRAY-1 Model A or B Banks		CRAY-1 S or M Series Banks		CRAY X-MP Banks	
		8	16	8	16	16	32
CA	High-order bits	41-42	40-41	40-41	31-40	43-47	42-46
BIT	Bit in chip	43-60	42-59	42-60	41-59	48-59	47-58
BK	Bank address	61-63	60-63	61-63	60-63	60-63	59-63

2.11 IDLE TASK

When the Task Scheduler finds no work for the CPU to perform, the Task Scheduler exchanges to the Idle task. The Idle task is a small loop that periodically reads EXEC's portion of memory. The Idle loop continues until an interrupt occurs.

The Idle task loops making memory references because EXEC normally executes with memory error detection disabled. The Idle task is designed to pick up any memory failures that EXEC does not normally detect.

2.12 EXEC DEBUG AIDS

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

2.12.1 HISTORY TRACE

The history trace is an EXEC-resident circular buffer of 4-word messages. The EXEC routine DEBUG makes entries in the History Trace Table (XTT) based on a function code plus control information in the History Function Table (XFT). DEBUG is accessed from STP with monitor request R020 (Post). The following tables are used by DEBUG:

XFT	History Function Table
XTT	History Trace Table

History Function Table (XFT)

The XFT determines which calls to the DEBUG routine result in entries in the History Trace Table (XTT). The first word is a global-enable word, with the mnemonic "ALL" in ASCII. If the low-order 24 bits of this word are set to nonzero, all calls to the DEBUG routine result in entries in the XTT. This is the default setting.

If the low-order 24 bits of this global-enable word are 0, the history trace function number passed to DEBUG is used as an index into the XFT. For example, a function code of 1 would point to the entry with the mnemonic "IOI" in ASCII, for I/O interrupt trace. If the low-order 24 bits of the indexed word are nonzero, then an entry is made in the XTT. If the low-order 24 bits of the indexed word are 0, no entry is made.

To disable all traces, clear the low-order 24 bits of the global-enable word, and ensure that no individual functions are enabled.

To selectively enable traces, clear the low-order 24 bits of the global-enable word, and set the low-order 24 bits of one or more other words in the XFT.

STP functions are further selected by assembly selection of the POST macro. An STP function not listed in the POST macro in the early part of STP is disabled and can be re-enabled only through reassembly.

History Trace Table (XTT)

The header of this table contains the real time of the last call to DEBUG and the offset from B@XTT where the next trace entry is formatted.

Each entry in this table contains the following information:

```
     0      7   10          24                        48    51      63
    ----------------------------------------------------------------
 0 |  FC   | PN |     SM     |             P           |//////|  XA  |
    ----------------------------------------------------------------
 1 |          B00            |                INT                   |
    ----------------------------------------------------------------
 2 |                            WD1                                 |
    ----------------------------------------------------------------
 3 |                            WD2                                 |
    ----------------------------------------------------------------
```

Field	Word	Bits	Description
FC	0	0-6	Function number
PN	0	7-9	Processor number (CRAY X-MP only)
SM	0	10-23	First 14 semaphores in system cluster (CRAY X-MP only)
P	0	24-47	Current exchange package P register
XA	0	51-63	Current exchange package address
B00	1	0-23	Last B00 value (if task related)
INT	1	24-63	Interval in cycles since previous entry
WD1	2	0-63	Caller supplied word 1
WD2	3	0-63	Caller supplied word 2

Consult the COS Table Descriptions Internal Reference Manual, publication SM-0045, for detailed information on these tables.

Use the following macro to make a trace entry from a task in STP. This example assumes that O'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
	POST	O'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	O'77 DEBUG	function number

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

I/O interrupt (IOI=1) - For I/O interrupt trace entries, the first data word transferred serves as the only trace of the following events: HYPERchannel function output, HYPERchannel status input. The contents of word 2 is the HYPERchannel status for input and output HYPERchannel segment interrupts.

Trace entry format:

	0	8	16	24	32	40	48	56	63
2	CHE	CHN		CHT			IH		
3				Data					

Field	Word	Bits	Description
CHE	2	0-6	Channel Error flag
CHN	2	7-15	Hardware channel number
CHT	2	16-39	Channel Table address

Field	Word	Bits	Description
IH	2	40-63	Interrupt handler address
Data	3	0-63	First data word transferred

User-initiated normal exchange (UNE=2) - Trace entry format:

	0	8	16	24	32	40	48	56	63
2	S0								
3	S1								

Field	Word	Bits	Description
S0	2	0-63	User S0
S1	3	0-63	User S1

STP-initiated normal exchange (SNE=3) - Trace entry format:

	0	8	16	24	32	40	48	56	63
2	S6								
3	S7								

Field	Word	Bits	Description
S6	2	0-63	Task S6
S7	3	0-63	Task S7

Exchange to system task (ENE=4) - Trace entry format:

	0	8	16	24	32	40	48	56	63
2	"TO:" in ASCII								
3	target								

Field	Word	Bits	Description
target	3	0-63	ASCII name of system task

Exchange to idle package (ENE=4) - Trace entry format:

	0	8	16	24	32	40	48	56	63
2	"TO:" in ASCII								
3	"IDLE" in ASCII								

Exchange to user task (ENE=4) - Trace entry format:

<table>
<tr><th></th><th>0</th><th>8</th><th>16</th><th>24</th><th>32</th><th>40</th><th>48</th><th>56 63</th></tr>
<tr><td>2</td><td colspan="8">"TO:" in ASCII</td></tr>
<tr><td>3</td><td colspan="5">"USER" in ASCII</td><td colspan="3">TXT</td></tr>
</table>

Field	Word	Bits	Description
TXT	3	40-63	STP-relative TXT address of user task being entered

Canceled timer event (PCI=5) - Trace entry format:

<table>
<tr><th></th><th>0</th><th>8</th><th>16</th><th>24</th><th>32</th><th>40</th><th>48</th><th>56 63</th></tr>
<tr><td>2</td><td colspan="8">"CANCEL" in ASCII</td></tr>
<tr><td>3</td><td colspan="8">"TIMEVENT" in ASCII</td></tr>
</table>

This trace entry occurs when a global timer event, which is set up for processing by both CPUs, is processed by a CPU. Since the timer was set in each CPU but need be processed by only one, the timer for the other CPU is canceled.

Time event (PCI=5) - Trace entry format:

<table>
<tr><th></th><th>0</th><th>8</th><th>16</th><th>24</th><th>32</th><th>40</th><th>48</th><th>56 63</th></tr>
<tr><td>2</td><td colspan="2">Zero</td><td colspan="3">PT</td><td colspan="3">EH</td></tr>
<tr><td>3</td><td colspan="8">TN</td></tr>
</table>

Field	Word	Bits	Description
PT	2	16-39	Parameter table address
EH	2	40-63	Event handler address
TN	3	0-63	"TIMEVENT" in ASCII

Default time event pulse (PCI=5) - Trace entry format:

<table>
<tr><th></th><th>0</th><th>8</th><th>16</th><th>24</th><th>32</th><th>40</th><th>48</th><th>56 63</th></tr>
<tr><td>2</td><td colspan="8">RTC</td></tr>
<tr><td>3</td><td colspan="8">TN</td></tr>
</table>

Field	Word	Bits	Description
RTC	2	0-63	Real-time clock at interrupt
TN	3	0-63	"DEFPULSE" in ASCII

Unexpected PCI interrupt (PCI=5) - Trace entry format:

<table>
<tr><th></th><th>0</th><th>8</th><th>16</th><th>24</th><th>32</th><th>40</th><th>48</th><th>56</th><th>63</th></tr>
<tr><td>2</td><td colspan="9">RTC</td></tr>
<tr><td>3</td><td colspan="9">TN</td></tr>
</table>

Field	Word	Bits	Description
RTC	2	0-63	Real-time clock at interrupt
TN	3	0-63	"UNEX PCI" in ASCII

Front-end input LCP (FEI=7) - Trace entry format:

<table>
<tr><th></th><th>0</th><th>8</th><th>16</th><th>24</th><th>32</th><th>40</th><th>48</th><th>56</th><th>63</th></tr>
<tr><td>2</td><td colspan="9">LCP0</td></tr>
<tr><td>3</td><td colspan="9">LCP1</td></tr>
</table>

Field	Word	Bits	Description
LCP0	2	0-63	LCP+0
LCP1	3	0-63	LCP+1

Physical disk I/O request (DIO=11) - This entry is not used for disk storage units connected through the I/O Subsystem.

Trace entry format:

<table>
<tr><th></th><th>0</th><th>8</th><th>16</th><th>24</th><th>32</th><th>40</th><th>48</th><th>56</th><th>63</th></tr>
<tr><td>2</td><td colspan="9">LDV</td></tr>
<tr><td>3</td><td colspan="9">CDR</td></tr>
</table>

Field	Word	Bits	Description
LDV	2	0-63	W@EQLDV, logical device name
CDR	3	0-63	W@CBCDR, current disk request word

Disk error retry part 1 (DIO=11) - This entry is not used for disk storage units connected through the I/O Subsystem. Disk error retry part 1 and part 2 always occur in contiguous pairs of trace entries.

Trace entry format:

	0 8 16 24 32 40 48 56 63
2	LDV
3	TD

Field	Word	Bits	Description
LDV	2	0-63	W@EQLDV, logical device name in ASCII
TD	3	0-63	W@EQTD, transfer direction word

Disk error retry part 2 (DIO=11) - This entry is not used for disk storage units connected through the I/O Subsystem. Disk error retry part 1 and part 2 always occur in contiguous pairs of trace entries.

Trace entry format:

	0 8 16 24 32 40 48 56 63
2	ST0
3	ST1

Field	Word	Bits	Description
ST0	2	0-63	W@EQST0, first status word
ST1	3	0-63	W@EQST1, second status word

Intertask message (ITM=12) - Trace entry format:

	0 8 16 24 32 40 48 56 63
2	WD1
3	WD2

Field	Word	Bits	Description
WD1	2	0-63	Input word 0 or output word 0
WD2	3	0-63	Input word 1 or output word 1

Error exchange (EEI=13) - Trace entry format:

```
     0      8      16     24     32     40     48     56    63
    ----------------------------------------------------------
 2 |                  "ERR EXCH" in ASCII                     |
   |----------------------------------------------------------|
 3 |                        Flags                             |
    ----------------------------------------------------------
```

Field	Word	Bits	Description
Flags	3	0-63	W@XPF (exchange package flags word) from the package which detected the error exchange

Front-end output LCP (FEO=14) - Trace entry format:

```
     0      8      16     24     32     40     48     56    63
    ----------------------------------------------------------
 2 |                        LCP0                              |
   |----------------------------------------------------------|
 3 |                        LCP1                              |
    ----------------------------------------------------------
```

Field	Word	Bits	Description
LCP0	2	0-63	LCP+0
LCP1	3	0-63	LCP+1

Front-end segment (SEG=15) - This entry is not used for front-end mainframes connected to the Cray mainframe through an I/O Subsystem.

Trace entry format:

```
     0      8      16     24     32     40     48     56    63
    ----------------------------------------------------------
 2 |                        Zero                              |
   |----------------------------------------------------------|
 3 |     Zero     |          LA           |        BA         |
    ----------------------------------------------------------
```

Field	Word	Bits	Description
LA	3	16-39	Absolute limit address of the segment buffer
BA	3	40-63	Absolute base address of the segment buffer

Front-end input SCBs (SCI=16) - Trace entry format:

	0	8	16	24	32	40	48	56	63
2	LCP3								
3	LCP4								

Field	Word	Bits	Description
LCP3	2	0-63	LCP+3
LCP4	3	0-63	LCP+4

Front-end error LCP (FEE=17) - This entry is not used for front-end mainframes connected to the Cray mainframe through an I/O Subsystem.

Trace entry format:

	0	8	16	24	32	40	48	56	63
2	LCP0								
3	LCP1								

Field	Word	Bits	Description
LCP0	2	0-63	LCP+0
LCP1	3	0-63	LCP+1

Front-end output SCBs (SCO=20) - Trace entry format:

	0	8	16	24	32	40	48	56	63
2	LCP3								
3	LCP4								

Field	Word	Bits	Description
LCP3	2	0-63	LCP+3
LCP4	3	0-63	LCP+4

User task status change (JST=24) - Trace entry format:

	0	8	16	24	32	40	48	56	63
2	OST					"->" in ASCII			
3	NST					TXT			

Field	Word	Bits	Description
ST	3	0-39	Old TXSTCH (ASCII task status) field
NST	4	0-39	New TXSTCH (ASCII task status) field
TXT	4	40-63	TXT ordinal associated with status change

Job status change (JST=24) - Trace entry format:

	0	8	16	24	32	40	48	56	63
2	OST					"=>" in ASCII			
3	NST					JXT			

Field	Word	Bits	Description
ST	3	0-39	Old JXSTCH (ASCII task status) field
NST	4	0-39	New JXSTCH (ASCII task status) field
JXT	4	40-63	JXT ordinal associated with status change

Search for a free memory segment (GET=25) - This entry is disabled in the default system.

Trace entry format:

	0	8	16	24	32	40	48	56	63
2	STCH								
3	SZ								

Field	Word	Bits	Description
STCH	2	0-63	JXSTCH for job that needs memory
SZ	3	0-63	Size of free segment sought

Allocation of a memory segment (GET=25) - This entry is disabled in the default system.

Trace entry format:

	0	8	16	24	32	40	48	56	63
2	MST								
3	N								

Field	Word	Bits	Description
MST	2	0-63	MST entry for the free segment from which the allocation is to be taken
JXORD	2	0-15	JXT ordinal
SGZ	2	16-39	Segment size
SGA	2	40-63	Segment address
N	3	0-63	Number of words to allocate

Liberation of a memory segment (LIB=26) - This entry is disabled in the default system.

Trace entry format:

	0	8	16	24	32	40	48	56	63
2	MSTO								
3	MST								

Field	Word	Bits	Description
MSTO	2	0-63	Offset of MST entry for segment to be freed
JXORD	2	0-15	JXT ordinal
SGZ	2	16-39	Segment size
SGA	2	40-63	Segment address
MST	3	0-63	The MST entry itself

Request received by JSH (JSH=30) - This entry is disabled in the default system.

Trace entry format:

	0	8	16	24	32	40	48	56	63
2	FN					JXORD			
3	JN							///////	

Field	Word	Bits	Description
FN	2	0-39	ASCII function name
JXORD	2	40-63	JXT ordinal
JN	3	0-55	ASCII job name

SSD transfer (SSD=31) - Trace entry format:

<table>
<tr><th></th><th>0</th><th>8</th><th>16</th><th>24</th><th>32</th><th>40</th><th>48</th><th>56 63</th></tr>
<tr><td>2</td><td colspan="8">TN</td></tr>
<tr><td>3</td><td colspan="8">FCT</td></tr>
</table>

Field	Word	Bits	Description
TN	2	0-63	"GO SSD" in ASCII
FCT	3	0-63	Function word (CBFCT from CBT)

SSD error (SSD=31) - Trace entry format:

<table>
<tr><th></th><th>0</th><th>8</th><th>16</th><th>24</th><th>32</th><th>40</th><th>48</th><th>56 63</th></tr>
<tr><td>2</td><td colspan="8">TN</td></tr>
<tr><td>3</td><td colspan="8">EC</td></tr>
</table>

Field	Word	Bits	Description
TN	2	0-63	"SSD ERR" in ASCII
EC	3	0-63	Error code

J$ALLOC requests (MEM=32) - Trace entry format:

<table>
<tr><th></th><th>0</th><th>8</th><th>16</th><th>24</th><th>32</th><th>40</th><th>48</th><th>56 63</th></tr>
<tr><td>2</td><td colspan="5">//</td><td colspan="3">MRWA</td></tr>
<tr><td>3</td><td colspan="8">MRW</td></tr>
</table>

Field	Word	Bits	Description
MRWA	2	40-63	Address of memory request word; initial processing done in STP
MRW	3	40-63	Memory request word itself

Entry to MOVEMEM routine (MEM=32) - This trace entry is suppressed if no data is moved.

Trace entry format:

	0	8	16	24	32	40	48	56	63
2	TN		FA			FL			
3	////////////////		TA			TL			

Field	Word	Bits	Description
TN	2	0-15	"MV" in ASCII
FA	2	16-39	From address
FL	2	40-63	From length
TA	3	16-39	To address
TL	3	40-63	To length

Entry to ERASEMEM routine (MEM=32) - This trace entry is suppressed if no data is erased.

Trace entry format:

	0	8	16	24	32	40	48	56	63
2	TN								
3	////////////////		EA			EL			

Field	Word	Bits	Description
TN	2	0-63	"ERASE" in ASCII
EA	3	16-39	Address
EL	3	40-63	Length of area to be erased

Exit from RELOCATE routine (MEM=32) - There are always two trace entries, one for before and one for after, relocating.

Trace entry format:

```
      0       8      16      24      32      40      48      56    63
     ----------------------------------------------------------------
  2 |         HLM           |        LFT          |       DSP        |
     ----------------------------------------------------------------
  3 |         BFB           |        BBFL         |       FL         |
     ----------------------------------------------------------------
```

Field	Word	Bits	Description
HLM	2	0-21	High limit of memory
LFT	2	22-42	LFT address
DSP	2	43-63	DSP address
BFB	3	0-21	BFB address
BBFL	3	22-42	Buffer boundary in first entry; change in FL in second entry.
FL	3	43-63	Field length

MCU interrupt (HTMCU=33) - Trace entry format:

```
      0       8      16      24      32      40      48      56    63
     ----------------------------------------------------------------
  2 |                              RTC                               |
     ----------------------------------------------------------------
  3 |                              TN                                |
     ----------------------------------------------------------------
```

Field	Word	Bits	Description
RTC	2	0-63	Value of real-time clock at event detection
TN	3	0-63	"MCU INT" in ASCII

Interprocessor interrupt (HTIPI=34) - Trace entry format:

```
      0       8      16      24      32      40      48      56    63
     ----------------------------------------------------------------
  2 |                              RTC                               |
     ----------------------------------------------------------------
  3 |                              TN                                |
     ----------------------------------------------------------------
```

Field	Word	Bits	Description
RTC	2	0-63	Value of real-time clock at event detection
TN	3	0-63	" IP INT" in ASCII

Deadlock interrupt (HTDLI=35) - Trace entry format:

	0 8 16 24 32 40 48 56 63
2	RTC
3	TN

Field	Word	Bits	Description
RTC	2	0-63	Value of real-time clock at event detection
TN	3	0-63	"DEADLOCK" in ASCII

System wait for single threading (HTSYS=36) - Trace entry format:

	0 8 16 24 32 40 48 56 63
2	RTC
3	TN

Field	Word	Bits	Description
RTC	2	0-63	Value of real-time clock at entry to code at SYSWAIT
TN	3	0-63	"SYSWAIT" in ASCII

Operating system entry after single-thread wait (HTNWT=37) - Trace entry format:

	0 8 16 24 32 40 48 56 63
2	CYC
3	TN

Field	Word	Bits	Description
CYC	2	0-63	RT clock cycles spent waiting
TN	3	0-63	"ENDWAIT" in ASCII

Logical interprocessor request (HTIPSET=40) - Trace entry format:

	0 8 16 24 32 40 48 56 63
2	REQ
3	TN

Field	Word	Bits	Description
REQ	2	0-63	Interprocessor request code: 0 (IPRQNOOP) No specific request 1 (IPRQPSW) Switch operating system to other CPU
TN	3	0-63	"IP SET" in ASCII

Logical interprocessor request acknowledgement (HTIPACK=41) - Trace entry format:

	0 8 16 24 32 40 48 56 63
2	REQ
3	TN

Field	Word	Bits	Description
REQ	2	0-63	Interprocessor request code: 0 (IPRQNOOP) No specific request 1 (IPRQPSW) Switch operating system to other CPU
TN	3	0-63	"IP ACK" in ASCII

Intertask message - task request (HTASCII=42) - Trace entry format:

	0 8 16 24 32 40 48 56 63
2	"RDY SUS" in ASCII
3	source "->" dest

Field	Word	Bits	Description
Source	3	0-23	Name of system task that initiated an intertask message and issued an RTSS (ready task, suspend self) request
Dest	3	40-63	Name of system task that was readied to receive an intertask message

This trace message is always part of a set of messages:

003- System task normal exchange
012- Intertask message; request in trace words 2 and 3
042- Intertask message; task request "RDY SUS xxx->yyy"

Intertask message - task reply (HTASCII=42) - Trace entry format:

	0 8 16 24 32 40 48 56 63
2	"READY" in ASCII
3	source "->" dest

Field	Word	Bits	Description
Source	3	0-23	Name of system task which initiated an intertask message and issued an RTSK (ready task) request
Dest	3	40-63	Name of system task which was readied to receive an intertask message

This trace message is always part of a set of messages:

003 System task normal exchange
012 Intertask message; request in trace words 2 and 3.
042 Intertask message; task reply "READY xxx->yyy".

Memory error (HTMEC=43) - Trace entry format:

	0 8 16 24 32 40 48 56 63
2	SYNDROME
3	"MEM ERR" in ASCII

Field	Word	Bits	Description
Syndrome	2	0-63	Syndrome bits of memory error

This trace message always occurs for single and double-bit memory errors.

2.12.2 SYSTEM STOP BUFFER

The System Stop Buffer is a feature of EXEC which assists the computer operator or system analyst in finding the general cause of a system crash. When EXEC detects a fatal error condition, it builds a STOP message in a buffer called the stop buffer. This buffer is located in EXEC at B@STOP. The buffer is loaded with the label in EXEC where the error is detected, the word address of P and B0, and an ASCII stop message. The buffer is formatted as follows:

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

The stop label is used in EXEC with the STOP macro. The STOP macro does not convert the values in P and B0 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. These two values have been truncated to words.

The following 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. Table 2-2 shows the EXEC stop messages.

Table 2-2. EXEC stop messages

Label	Code	Significance
$STOP000	EEF	Unknown error
$STOP001	EX	(A1) does not equal XP exchange address
$STP0002	APOIP	IOP channel error
$STOP003	APIIP	IOP channel error
$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 (usually accompanied by SY006 message in STP memory)
$STOP009	EN	CPU halt requested by IOS
$STOP010	TECAN	Invalid event number
$STOP011	TS0	The STP Lock flag STPLK is set, but no task is marked as active.
$STOP012	R041	A PSWITCH request was received from a task other than the job scheduler (JSH).

Table 2-2. EXEC stop messages (continued)

Label	Code	Significance
$STOP013	TS2	No system task was selected on entry to TS2.
$STOP014	TEREQ	Invalid event number
$STOP015	ENQ	Queue entry contains bad ID
$STOP016	R017	The TXT address specified in the request does not match the address of the TXT entry which is connected in the requesting CPU.
$STOP017	R005I/APRCV	B@FIQ empty
$STOP018	APENQ	B@FOQ empty
$STOP019	API	B@FIQ empty
$STOP020	ENQ	Maximum queue length exceeded
$STOP021	DEQ	Queue empty
$STOP022	DEQ	Header queue ID does not equal entry queue ID
$STOP023	R001	This error occurs when a duplicate task priority is encountered on a task create.
$STOP024	R005	Undefined operation
$STOP025	R005	Undefined channel type
$STOP026	R005C	Undefined operation
$STOP027	EXA	A zero exchange package address was presented to the exchange processor.
$STOP028	R005I	Undefined operation
$STOP029	R005N	Undefined operation
$STOP030	R011	Maximum cylinder exceeded
$STOP031	MCOR/XBMSG	An IOP packet cannot be obtained to send the error packet to MEP.
$STOP032	XPROC/EN	System cluster not set up
$STOP034	R022/APIIP	Short packet received from IOS
$STOP035	INIT	Error reading date and time from MCU
$STOP036	INIT/GETRT	Bad time or date parameter from MCU
$STOP037	MCOR/XMCNT	This error occurs when the double-bit count (I@MEUCT) is exceeded.
$STOP038	MCOR/XSYND	This error occurs if the population count of the syndrome bits is 0, indicating the hardware reported an error but the syndrome bits show no error.

Table 2-2. EXEC stop messages (continued)

Label	Code	Significance
$STOP039	MCOR/XSYND	This error occurs if the decoded syndrome bits do not match the correctable/uncorrectable code contained within the exchange package.
$STOP040	MCOR/XMHLT	This error occurs if the EXEC idle loop detects a multibit error.
$STOP041	MCOR/XMHLT	This error occurs if a multibit error occurs while STP is executing.
$STOP042	MCOR/XMHLT	This error occurs if any multibit error occurs during an I/O reference.
$STOP043	R016	A zero time slice was selected in the RCP=016 EXEC request.
$STOP044	R016	A zero time slice for SPY was found in the connected user task's TCB.
$STOP045	R016	The SETCL macro was unable to find the requested cluster number in the list of valid clusters. Since the cluster number has been previously validated, this indicates a hardware problem.
$STOP046	R017	A zero time slice for SPY was found in the connected user task's TCB.
$STOP048	R042	The SETCL macro was unable to find the requested cluster number in the list of valid clusters. Since the cluster number has been previously validated, this indicates a hardware problem.
$STOP050	EX	SM@PLOCK was cleared while EXEC was executing.
$STOP051	EX	SM@EXEC was cleared while EXEC was executing.
$STOP052	EX	The PWS addresses calculated before and after an exchange do not match.
$STOP053	EX	SM@PLOCK was cleared during the execution of an STP task.

Table 2-2. EXEC stop messages (continued)

Label	Code	Significance
$STOP054	EX	The SETCL macro was unable to find the requested cluster number in the list of valid clusters.
$STOP055	IPREQST	An invalid interprocessor request code was found in the IPRQ table.
$STOP056	EX	The current task ID, maintained in low-STP memory by EXEC, was found to exceed the maximum task number.
$STOP057	NE	A system task made an EXEC request while holding STPLK.
$STOP058	NE	The active task STT addresses in the PWS entry for the executing CPU and in the STT header differ.
$STOP059	SCHUSER	W@TCEPAL in the connected user task's TCB was found to have all flags clear, but to be nonzero.
$STOP060	IPCPU	An interprocessor request is already pending for the other CPU.
$STOP061	EX	No system task was selected for execution at label EX, in violation of the specified entry conditions.
$STOP062	TS0	The active system task STT addresses in the PWS (for the executing CPU) and in the STT header differ. This indicates a problem with CPU scheduling in EXEC.
$STOP063	DLI	A deadlock condition was detected that did not occur in the user area.
$STOP064	R014	A system task is attempting to both ready and suspend itself.
$STOP065	IDLE	An exit was made from the main idle loop, indicates a hardware problem.
$STOP069	APSP	An unknown S-packet type was received from the I/O Subsystem.
$STOP072	BOOT	Undefined CPU type.
$STOP073	SETLA	Undefined CPU type.
$STOP074	SETXP	Undefined CPU type.

2.13 INTERACTIVE SYSTEM DEBUGGING

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.

SCP, common routines used by SCP, and EXEC cannot be breakpointed; the debugging commands use SCP and EXEC to communicate with the operator.

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

2.14 MULTIPROCESSOR CONSIDERATIONS

Several aspects of EXEC reflect its need to support multiprocessor as well as uniprocessor configurations.

2.14.1 SINGLE-THREADING

COS was originally developed on a uniprocessor system. Many code sequences reflect the assumption that only one processor is active by how they access and update tables.

Rather than locating and changing all explicit and implicit assumptions regarding process synchronization within COS, the much less timeconsuming decision was made, that COS should run in only one CPU at any one time.

On the CRAY X-MP mainframe, the hardware semaphore registers are used by EXEC to ensure that only one CPU is active in either EXEC or in the STP area. When one CPU is found to be in COS, the other CPU waits for the other CPU to leave the operating system; the code that accomplishes this begins at label LOCKOS in EXEC and extends through the end of the SYSWAIT subroutine.

Because SYSWAIT executes in monitor mode with external interrupts disabled, and because when one CPU is in monitor mode no I/O interrupts are posted to another CPU of a CRAY X-MP system, SYSWAIT polls for I/O interrupts. When an I/O interrupt is found and when the other CPU is not in EXEC, SYSWAIT sends an interprocessor message and interrupts the other CPU.

2.14.2 SEMAPHORE USAGE

Management of CRAY X-MP cluster registers is the responsibility of three entities in the COS environment: EXEC, which manages the system cluster, COS locks, and saves and restores user cluster registers; JSH, which assigns nonsystem clusters to user jobs (and the contained user tasks) and directs EXEC to load or save user clusters; and to user-mode code, both user programs and library subroutines.

EXEC saves and restores user cluster registers at the direction of JSH. As an aid to the library scheduler, EXEC also clears semaphore SM@BWAIT (SM00) on every exchange to a user task. EXEC does not modify any other registers in user clusters.

Several semaphore registers are used within EXEC for interprocessor communication and coordination:

- SM@ALOCK - referred to as the active lock - is the master lock within EXEC. It is a short-term lock, used when attempting to gain access to other longer-term locks. Code sequences involving SM@ALOCK always follow the general pattern:

```
WAIT$SET  ALOCK       Test and set master lock
GETSM     xxxx        Test secondary lock
$IF S0,MI             If secondary lock is busy
  CLRSM     ALOCK     Release master lock
  J         delay     Exit to do something else, or try again
$ENDIF
SETSM     xxxx        Set secondary lock
CLRSM     ALOCK       Release master lock
```

- SM@PLOCK - called the passive lock - is used within EXEC to ensure single-threading of COS. The only portions of COS that are not single-threaded are small portions of EXEC: between labels EN and LOCKOS, and subroutines called by SYSWAIT (DEBUG, and IPCPU). When SM@PLOCK is set, one X-MP CPU is active in COS.

- SM@EXEC flags for the SYSWAIT subroutine. When SM@EXEC is set, a CPU is in the single-threaded section of EXEC; when clear, then no CPU is in EXEC. SM@PLOCK is always set when SM@EXEC is set, though the reverse is not true.

- SM@IPRQ controls access to the Interprocessor Request Table (B@IPRQ). Writes into the table can only take place after issuing a WAIT$SET IPRQ instruction. This lock is needed because writes into the IPRQ table can take place outside of code protected by SM@PLOCK.

- SM@DEBLK ensures single-threading in the DEBUG subroutine. This semaphore will always be set in the history trace entries for X-MP systems. This lock is needed because DEBUG calls can be made from outside of code protected by SM@PLOCK.
- SM@BWAIT flags for the memory error correction code. When set, SM@BWAIT indicates that a CPU is in the SYSWAIT subroutine. Memory error correction uses this flag to determine that the other CPU is parked in EXEC so that memory correction can be safely done. SM@BWAIT (in the system cluster) is set and cleared only in the SYSWAIT subroutine.

2.14.3 INTERPROCESSOR COMMUNICATIONS

The RCP and DCP Executive requests (connect and disconnect user task) are issued by JSH to associate and disassociate a user task from the specified CPU. Because the requests are CPU-specific, EXEC provides JSH with a mechanism (PSWITCH) for switching between physical CPUs.

The PSWITCH Executive request, available only to JSH, performs the necessary interprocessor communication. When JSH calls PSWITCH, EXEC suspends the caller and verifies that the caller was indeed JSH.

EXEC, in addition to accomodating JSH, also needs the ability to switch physical CPUs. EXEC causes the other CPU to enter EXEC either because an I/O interrupt is found while in SYSWAIT, or because memory error correction needs to have all user-mode exchange packages in memory (having all CPUs in EXEC guarantees no user-mode exchange package is active).

Processor switching is accomplished through a mechanism called interprocessor communication. Interprocessor communication takes place through messages. Interprocessor messages are currently of two types:

- Processor switch messages to allow JSH to use the other CPU
- No-op messages to allow EXEC to use the other CPU

EXEC routine IPCPU places the interprocessor message in the Interprocessor Request Table (IPRQ) and issues an interprocessor interrupt (IP 1) instruction. Interprocessor interrupts, while used in sending messages, are basically ignored by EXEC.

When the other CPU enters EXEC, EXEC receives the message. IPREQST processes processor switch messages and schedules JSH in the receiving CPU.

2.14.4 PROCESSOR WORKING STORAGE AREA (PWS)

The processor working storage area (PWS) contains data specific to each CPU, including:

- Addresses and fields associated with a connected user task, including the user task exchange package, if any.
- Idle task exchange package and B00 register save areas.
- Memory error correction exchange package.
- Statistics and timing counters, both cumulative and in last statistics interval.
- A word indicating which software process is executing in the CPU in question (USER, EXEC, MEM-COR, or system task).

The GETPW macro can be used to determine the PWS address of the current CPU (see section 2.15.4).

2.15 EXEC-SPECIFIC MACROS

A number of macros are defined locally within EXEC. These macros are generally appropriate only in the EXEC environment, or are used to ease the writing of machine-independent code without conditional assembly at the source statement level.

2.15.1 CLEARIP

The CLEARIP macro clears the Interprocessor Interrupt flag in the CRAY X-MP CPU which encounters the macro; on other systems the macro does not generate any code.

2.15.2 COPYXP

The COPYXP macro copies exchange packages from one area of memory to another. The macro was written with the goal of removing the run-time loops and vector-register operations which had previously been used to perform the copy.

2.15.3 X$SIO

The X$SIO macro initiates an I/O operation on a low-speed channel, to record the Interrupt Handler Table address for the channel in question, and to record the starting and ending addresses sent to the channel (for debugging).

2.15.4 GETPW

The GETPW macro obtains the address of the processor working storage area for the CPU which is executing the GETPW macro. On uniprocessor systems, this is always the first PWS entry; on multiprocessor systems the PWS address is based on the CPU number.

2.15.5 GETSR0

The GETSR0 macro (defined in comdeck GETSR0) extracts fields from CRAY X-MP status register zero. When assembled on uniprocessor systems, the macro returns a zero as the value of the requested field.

2.15.6 I$FWB

The I$FWB macro forces alignment of code to a specific word boundary. Its most typical usage is to force tables to 4- or 8- word boundaries to make dumps easier to read.

2.15.7 SETCL

The SETCL macro issues one of the CRAY X-MP set cluster number instructions, and was written because:

- Selection of a cluster by the hardware requires that the cluster number be a constant embedded in the instruction. No instruction is available to set the cluster number from a register.

- CAL allows only the numbers 0, 1, 2, 3 on the CLN instruction; it does not allow constants with the values above.

SETCL allows either a constant or a register as the cluster designator, and either issues the instruction directly (if a constant was specified) or uses a set of $IF/CLN statements to select the desired cluster.

When assembled on uniprocessor systems, SETCL does not generate any code.

2.15.8 SETIP

The SETIP macro sets the Interprocessor Interrupt flag in the other CPU in CRAY X-MP systems. On uniprocessor systems, the macro does not asssemble any code.

2.15.9 STOP

The STOP macro stops EXEC and issues a message when a fatal error occurs. The STOP call can be either unconditional, or alternately the programmer can elect to stop only when a specific condition or set of conditions occurs. In the latter case, any condition accepted by the $IF family of macros may be used on a STOP call.

2.15.10 FALLTHRU

The FALLTHRU macro allows the programmer to insert visual aids for comprehension while ensuring that later modifications (possibly by another programmer) do not insert code into what was originally intended to be contiguous code sequences.

SYSTEM TASK PROCESSOR (STP) 3

3.1 GENERAL DESCRIPTION

The System Task Processor (STP) runs in non-monitor (user) mode and accesses all memory other than that occupied by EXEC. STP is responsible for processing all user requests. STP consists of tables, a set of programs called tasks, and some reentrant routines common to all tasks.

A system task 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 the tasks allowed to communicate. Each task and many of the common subroutines are separate UPDATE decks and CAL IDENTs.

The system tasks (deck and IDENT names are noted in parentheses) are:

Startup (STP, STARTUP)
Disk Queue Manager (DQM)
Station Call Processor (SCP)
Exchange Processor (EXP)
Job Scheduler (JSH)
Permanent Dataset Manager (PDM)
Log Manager (MSG)
Message Processor (MEP)
Disk Error Correction (DEC)
System Performance Monitor (SPM)
Job Class Manager (JCM)
Overlay Manager (OVM)
Tape Queue Manager (TQM)
Stager (STG)
Flush Volatile Device (FVD)

Each system task is fully described in a later section of this manual.

The addresses in 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 through the task's parameter block in the STT.

Tasks execute in program mode and are thus interruptible. An interrupt occurs as a result of the task executing an exit instruction (ERR or EX) or results from one of the interrupt flags being set automatically (for example, 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. A task 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 also occurs as a result of an explicit EXEC request issued by one task for the execution of another task. A task is readied or suspended by a master 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 COMMUNICATION

Tasks communicate with EXEC, with each other, with user jobs, and with the front end.

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 registers S6 and S7.

EXEC requests are described in detail in section 2.6 of this publication.

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.

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

PUTREQ	Put request routine, asynchronous; destroys A6.
GETREQ	Get request routine; destroys A6 and A7.
PUTREPLY	Put task reply routine; destroys A6 and A7.
GETREPLY	Request status routine; destroys A6.

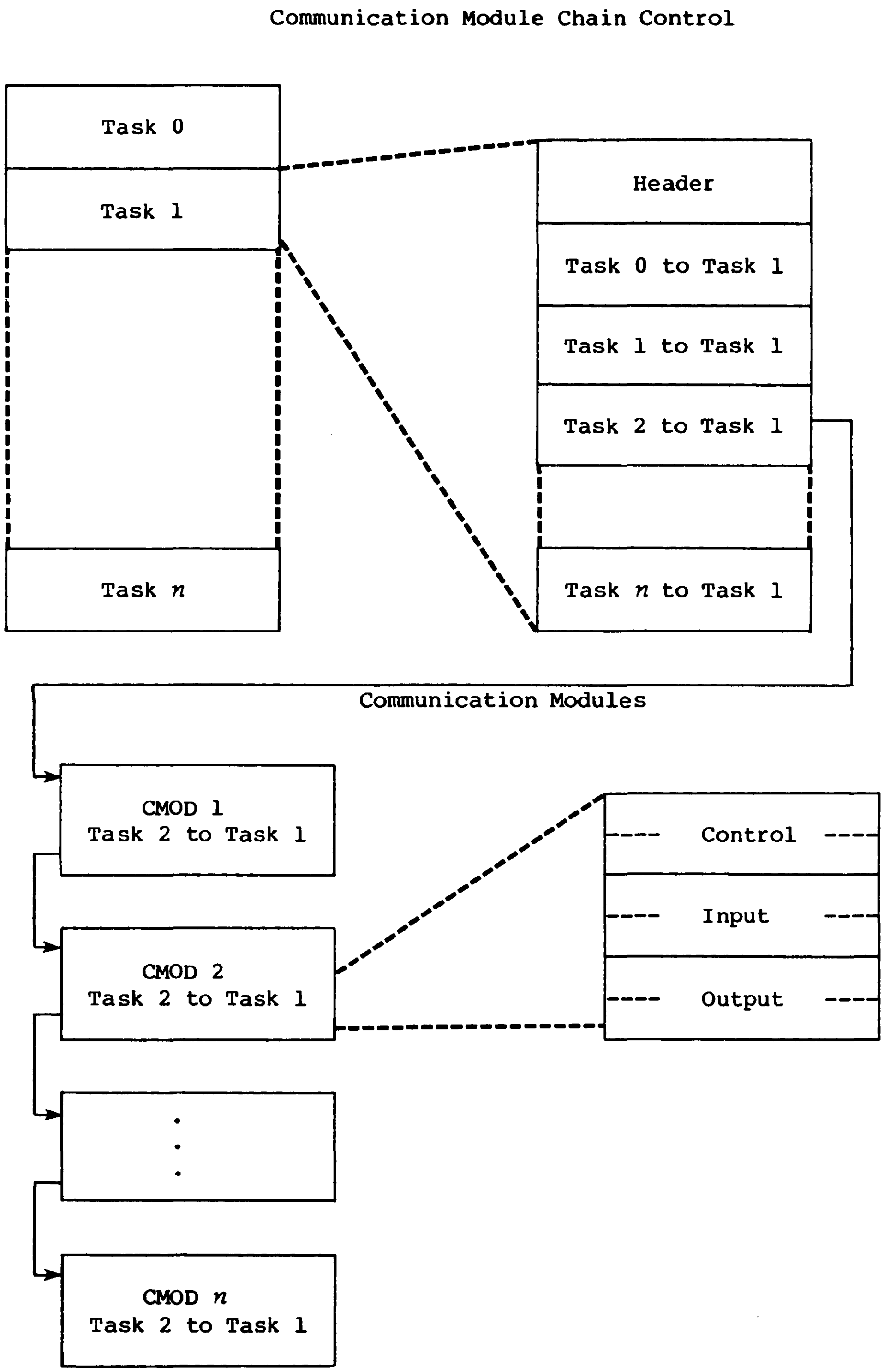


Figure 3-1. Task communication tables

TSKREQ	Task request routine, synchronous; destroys A3.
REPLIES	Queues unrequested reply; destroys A6.

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. If TSKREQ is used, PUTREQ and GETREPLY must not be used.

TSKREQ is incompatible with PUTREQ and GETREPLY; if TSKREQ is used, PUTREQ and GETREPLY must not be.

PUTREQ

This STP common subroutine places the request in the input registers of a CMOD and links the appropriate communications module chain control. If the request cannot be chained because 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 through a return jump with the caller providing the following values.

INPUT REGISTERS: (A1) Discard indicator. If (A1) is positive, control does not return to the caller until the request is queued. If (A1) is negative and the request cannot be queued without suspending the caller, control returns with no action taken.

(A2) Requested task's ID

(S1) INPUT+0

Request

(S2) INPUT+1

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 through a return jump.

INPUT REGISTERS: None

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

(A2) ID of task that generated the request

(S1) INPUT+0

Request

(S2) INPUT+1

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 represents an unrelated request and reply. The subroutine readies the task where the reply is directed and returns to the requester. PUTREPLY is called through a return jump.

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

(S1) OUTPUT+0

Reply

(S2) OUTPUT+1

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 through a return jump.

INPUT REGISTERS: None

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

(A2) ID of replying task

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

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 is queued. Once the request is queued, the caller is suspended until a reply is received. TSKREQ is called through a return jump. If one task makes a request to another using TSKREQ, all requests from the first task to the second must be made using TSKREQ. Mixed use of TSKREQ and PUTREQ/GETREPLY can cause unpredictable results.

INPUT REGISTERS: (A2) ID of requested task

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

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

REPLIES

This subroutine queues a reply for which no corresponding request has been made. The reply is queued at the beginning of the reply queue. A reply sent through this subroutine is seen by GETREPLY before any reply sent through PUTREPLY.

INPUT REGISTERS: (A1) Discard indicator. If (A1) is positive, control does not return to the caller until a reply is queued. If (A1) is negative and the reply cannot be queued without suspending the caller, control returns with no action taken.

(A2) ID of task to receive the reply

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

OUTPUT REGISTERS: None

3.2.3 USER/STP COMMUNICATION

User tasks initiate user/STP communication. A user program request to STP is performed when the user task loads register S0 (and optionally S1 and S2) and executes the normal exit instruction. Most system action requests can be issued through a CAL macro (see the Macros and Opdefs Reference Manual, CRI publication SR-0012). The user macro also results in a normal exit from the user program. EXEC routes all normal exits from a user task to the User Exchange Processor. The handling of these requests by the User Exchange Processor is described in section 8 of this manual.

3.2.4 TASK/FRONT-END COMMUNICATION

Tasks can issue messages to any logged on front-end station with a message processing capability. Messages are either strictly informative or require a response by the operator.

Messages are queued by the common subroutine MSGQUE and processed by the Station Call Processor (SCP) task at the first opportunity for communication to the front end. (See section 7 for detailed information on message handling by SCP.)

The MSGQ system macro queues a message to the front end using the interactive queueing mechanism and assigns a message number. The macro call has the following format:

Location	Result	Operand
	MSGQ	ADR=*x*

x Symbolic name of, or an A or S register containing the address of the message buffer

The calling task builds the message buffer (including the header and the text) and supplies a buffer address pointer, ADR. The macro routine queues the message and supplies any necessary default values.

Register S1 is set up and returned as follows:

```
     0       8       16      24      32      40      48      56     63
S1  |///////////////////////////////////|      MN      |    Status   |
```

MN Assigned message number, not meaningful if status is not normal reply.

Status One of the following status codes:

Code	Meaning
000	Normal reply
100	Station not logged on
101	Station message processing disabled
102	Message format error
103	Outstanding message count exceeded
104	Message word count too large for station
105	Message type not supported

MSGQ enters either the complete message, the message header, or nothing at all into the COS System Log depending on what is specified in the message's LOG field. (The Log Manager task is not active during Startup. Therefore, messages sent during Startup are not entered into the System Log, regardless of the contents of the LOG field.)

STP COMMON ROUTINES 4

Certain reentrant routines resident in STP are called by return jumps rather than by a call to another task. These include:

- Task logical I/O routines (TIO)
- Circular I/O routines (CIO)
- Memory management routines
- Item chaining/unchaining routines
- Interactive communication buffer management routines
- Password encryption
- System buffer management

4.1 TASK I/O ROUTINES (TIO)

Task I/O (TIO) is a set of reentrant common routines in STP logically considered part of any system task that calls it. TIO interprets only COS blocked format and therefore, only operates on blocked datasets. It allows a systems programmer to do logical I/O at the system task level without being concerned about physical I/O. The following COS system tasks currently call TIO:

Exchange Processor (EXP)
Startup (Z)
Log Manager (MSG)

Primary inputs to TIO consist of a Task Execution Table (TXT) address, a Dataset Name Table (DNT) address, a Dataset Parameter Table (DSP) address, and the address of the system buffer area. The logical I/O may be performed on either a dataset related to the system or a user task related dataset. TIO does not allocate or deallocate any of the control structures or buffers for the request, but assumes all control structures and buffers are set up correctly before the request by the system task. Figure 4-1 illustrates the linkages between the DNT, DSP and buffers.

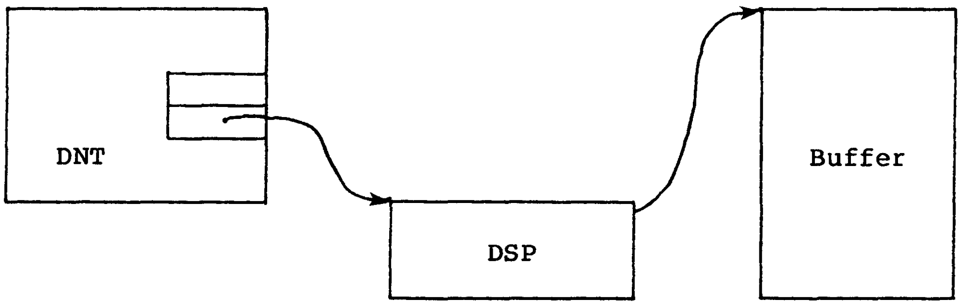


Figure 4-1. Dataset table linkages

A system task calls TIO directly by performing a return jump to one of the TIO externalized labels after setting up the proper input parameters as delineated in section 4.1.3. After calling TIO, the calling system task should perform the I/O complete sensing by directly calling a routine in CIO (CRCIO). The sensing consists of receipt of the DQM or TQM acknowledgement through a GETREPLY call. It should be noted at this point that TQM acknowledgements must first be converted to DQM acknowledge format before calling CRCIO. The format of the input to CRCIO is as follows:

```
        0      8       16      24      32      40      48      56    63
       ┌──────────────────────────────────────────────────────────────┐
  S1   │                        Return status                         │
       ├──────────────────────────────────────────────────────────────┤
  S2   │      Return address      | |   TXT offset   |   DNT address  │
       │        (optional)        | |                |  (JTA relative)│
       └──────────────────────────────────────────────────────────────┘
                           Partial
```

Field	Register	Bits	Description
Return status	S1	0-63	Return status of request; 0 indicates no error. Other errors are described in UPDATE comdeck COMEXERR.
Return address	S2	0-23	Optional return address returned on acknowledge
Partial	S2	24	Partial Recall flag. If set, the request is a partial recall from DQM or TQM.
TXT offset	S2	25-39	Relative TXT offset (from B@TXT)

Field	Register	Bits	Description
DNT address	S2	40-63	JTA-relative DNT address (if an STP DNT the address is relative to B@STP)

TIO exits to the calling system task's main interrupt loop when awaiting completion of physical I/O. This exit is performed through CIO when it is called to perform physical sector reads and writes. TIO returns to the calling task only upon completion of the logical I/O request. The calling task cannot make another TIO request for a given dataset until any previous logical request is complete.

The following stepchart illustrates the TIO flow:

1. System Task calls TIO with proper input parameters

2. TIO blocks or deblocks the user data between the user buffer and the system buffer.

3. If necessary, TIO calls CIO to perform a physical read/write. CIO exits to the calling task's main interrupt loop.

The calling task is responsible for calling CRCIO in CIO upon receipt of the DQM acknowledge for the physical sector read/write complete. CRCIO returns to the main interrupt loop of the calling task until all sectors have transferred.

When all physical sectors of the request have been transferred and the block/deblock is complete, the calling task receives control immediately after the call to TIO.

The following TIO routines are available to system tasks:

Routine	Functions
$RWDP	Read one or more words; partial mode (will not skip to next end of record).
$RWDR	Read one or more words; record mode (will skip to next end of record).
$WWDP	Write one or more words; partial mode (no end of record written).

Routine	Functions
$WWDS	Write one or more words with unused bits in last word; record mode (end of record written).
$WWDR	Write zero or more words; record mode (end of record written).
$WEOF	Write EOF; calls $WWDR if no end of record was written.
$WEOD	Write EOD; calls $WEOF if no end-of-file was written.
$REWD	Rewind dataset; calls $WEOD if the dataset is in write mode and no end-of-data 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 through 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 IOLIBPL program library.

****

4.1.1 SYSTEM TABLES USED BY TIO

TIO uses the following system tables for the dataset where I/O is to be performed:

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 (see description of the Exchange Processor in section 8 of this publication).

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 (EOD). TIO uses reserved words at the end of the DSPs. These are saved in the JTA when a TIO routine goes into recall for a job doing buffered I/O.

4.1.2 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 Dataset Parameter Table (DSP) error status (DPERR) indicate which error occurred.

4.1.3 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 4-2 provides an overview of the logical read operation. The calling routine must examine DPEOR, DPEOF, and DPEOD in the Dataset Parameter Table (DSP) to determine end-of-record (EOR), end-of-file (EOF), or end-of-data (EOD) 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 routine

Words are transmitted from the I/O buffer defined by the Dataset Parameter Table (DSP) to the area beginning at the first destination word address (FWA) until either the word count in A3 is satisfied or an EOR is encountered. $RWDP calls $RBLK, as necessary.

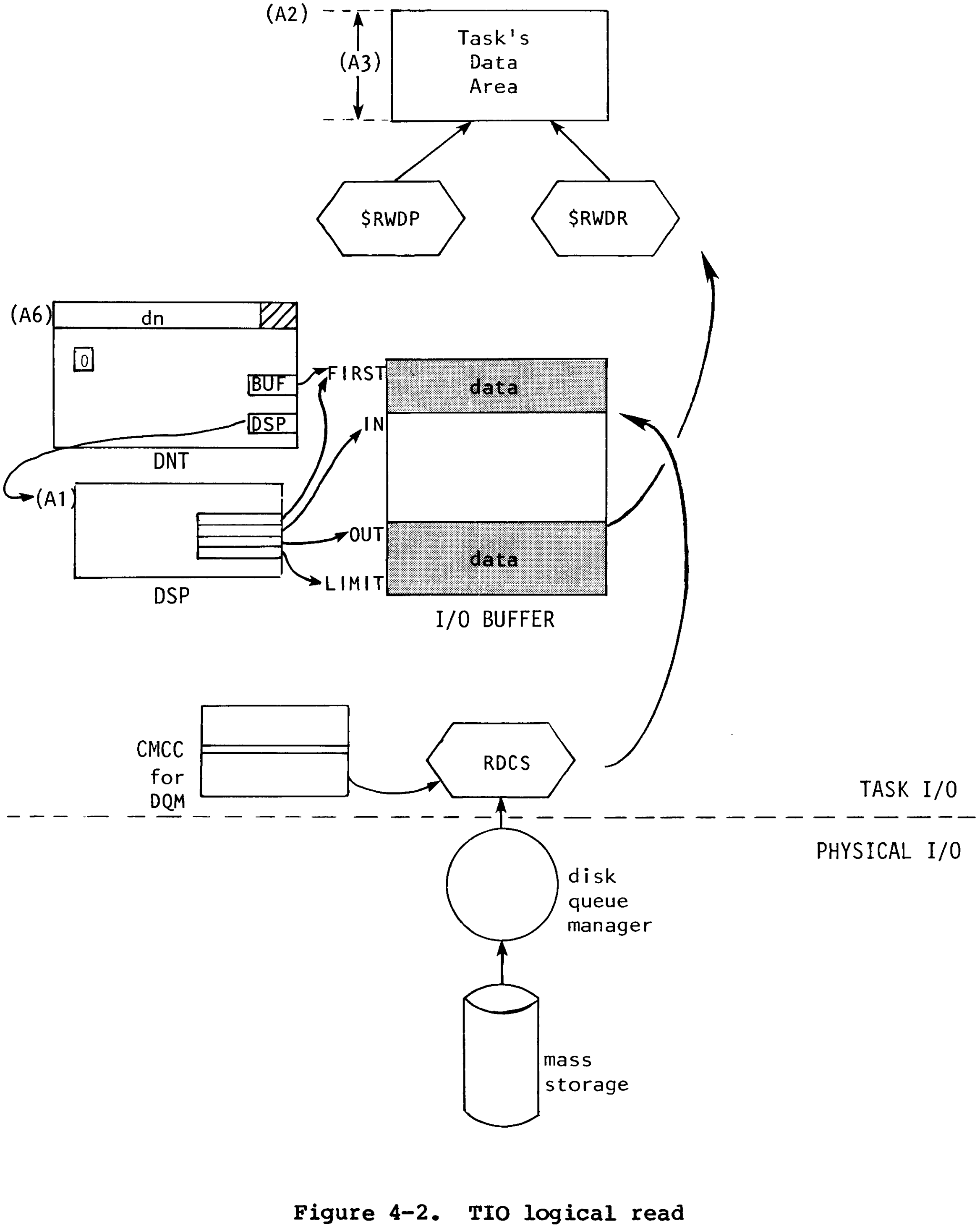


Figure 4-2. TIO logical read

SUBROUTINE NAME: $RWDP - Read words, partial mode

ENTRY CONDITIONS: (A1) DSP address

(A2) FWA of task's data area

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

(A6) DNT address

(A7) TXT address; 0 if not user task related.

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

(A1) DSP address

(A2) FWA of task's data area

(A3) Word count

(A4) LWA+1, end of data area

(A6) DNT address

(A7) Same value as on input

(S0) Status:
<0 End-of-record (EOR)
=0 Null record
>0 End-of-count

STEPFLOW:

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. Skip to next EOR.
7. If not at BCW, go to 9.
8. Call $RBLK.
9. Update DSP.
10. Exit.

$RWDR routine

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

STEPFLOW: Same as $RWDP

4.1.4 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 4-3 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 (RCW) indicates unused bits, and the bits are then cleared when the record is read.

$WWDP routine

The number of words specified by the count are transmitted from the task's data area beginning at the supplied first word address (FWA) and are written in the I/O buffer defined by the Dataset Parameter Table (DSP). $WWDP automatically calls $WBLK, as needed.

SUBROUTINE NAME: $WWDP - Write words, partial mode

ENTRY CONDITIONS: (A1) DSP address

(A2) FWA of task's data area

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

(A6) DNT address

(A7) TXT address; 0 if not user task related.

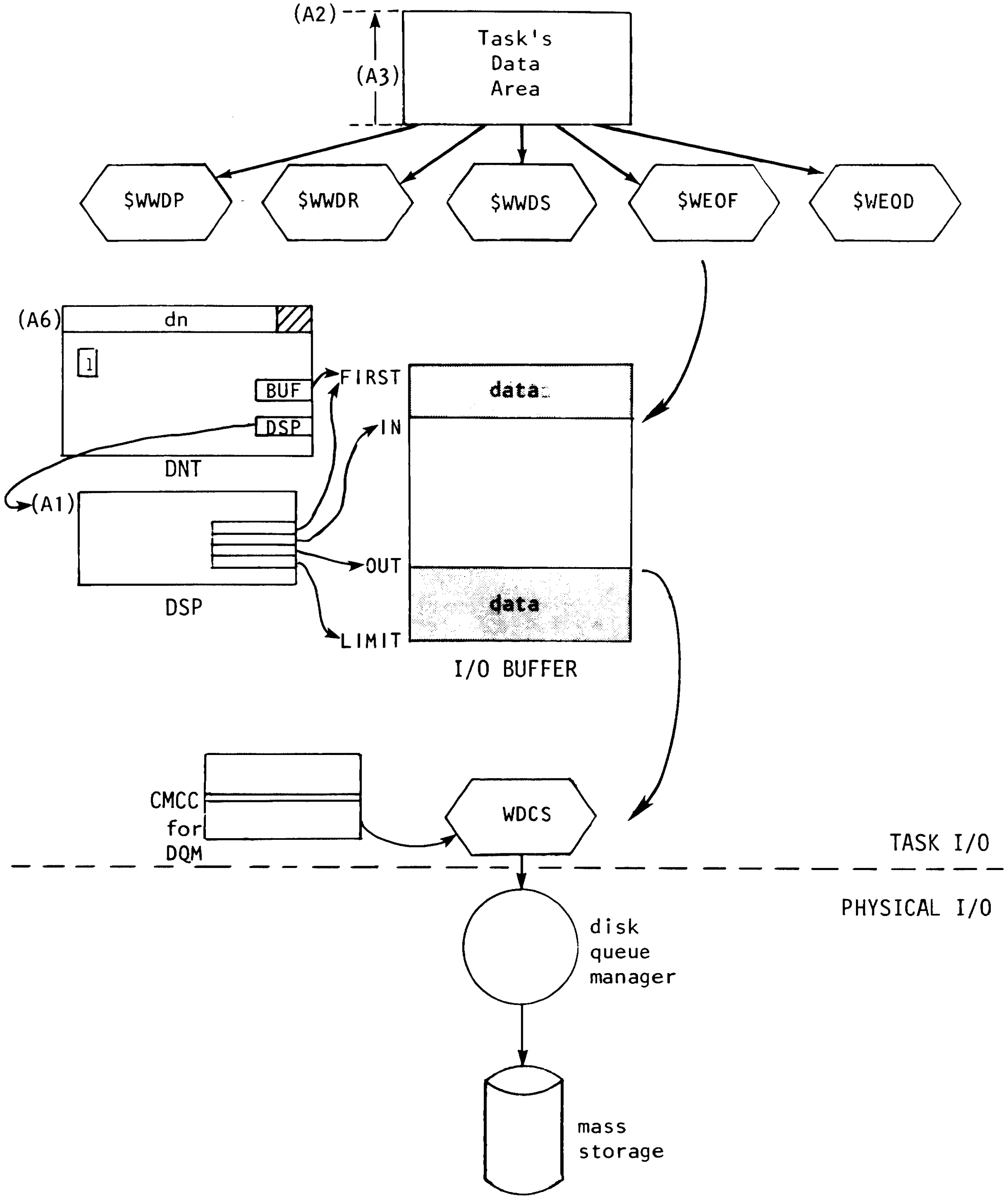


Figure 4-3. TIO logical write

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

(A1) DSP address

(A2) FWA of task's data area

(A3) Word count

(A4) LWA+1 of data area

(A6) DNT address

(A7) Same value as on input

(S0) Status:
<0 End-of-record (EOR)
=0 Null record
>0 End-of-count

STEPFLOW:

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.

$WWDR routine

The $WWDR routine resembles $WWDP. However, an EOR record control word (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

STEPFLOW: Same as $WWDP

$WWDS routine

The $WWDS routine is identical to $WWDR, except that the last word of the record contains unused bits, and the EOR record control word (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

STEPFLOW: Same as $WWDP

$WEOF routine

This routine writes an EOF record control word (RCW) preceded by an EOR RCW, if necessary, as the next words in the I/O buffer.

SUBROUTINE NAME: $WEOF - Write end-of-file RCW

ENTRY CONDITIONS: (A1) DSP address

(A6) DNT address

(A7) TXT address; 0 if not user task related.

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

(A6) DNT address

(A7) Same value as on input

STEPFLOW:

1. If EOR not written, call $WWDR.
2. Call $WWDR to write EOF.
3. Exit.

$WEOD routine

This routine writes an EOD record control word (RCW) preceded by an EOR and an EOF, if necessary, as the next words in the I/O buffer. $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 RCW

ENTRY CONDITIONS: (A1) DSP address

(A6) DNT address

(A7) TXT address; 0 if not user task related.

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

(A6) DNT address

(A7) Same value as on input

STEPFLOW:
1. If EOF not written, call $WEOF.
2. Call $WWDR to write EOD.
3. Exit.

4.1.5 POSITIONING ROUTINE

TIO supports a single positioning routine, $REWD.

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

SUBROUTINE NAME: $REWD - Rewind dataset

ENTRY CONDITIONS: (A1) DSP address

(A6) DNT address

(A7) TXT address; 0 if not user task related.

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

(A6) DNT address

(A7) Same value as on input

STEPFLOW:
1. If EOD not written, call $WEOD.
2. Reset DSP.
3. Exit.

4.1.6 BLOCK TRANSFER ROUTINES

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

<u>$RBLK routine</u>

$RBLK is called only by other task I/O routines and cannot 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 that CIO finds buffer space available. If the buffer is more than half full when $RBLK is called, $RBLK verifies the next block control word (BCW) (its block number must equal the relative sector number of the dataset) and returns to the caller.

SUBROUTINE NAME: $RBLK - Read blocks

ENTRY CONDITIONS: (A1) DSP address

(A5) Current BCW address

(A6) DNT address

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

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

(A1) DSP address

(A4) DPOUT field from DSP

(A6) DNT address

(A7) Same value as input

STEPFLOW:
1. If buffer more than half empty, call CIO at entry point RDCS.
2. Update DSP.
3. Exit.

$WBLK routine

$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 block control word (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 full when $WBLK is called, $WBLK does no more than insert BCWs as needed.

SUBROUTINE NAME: $WBLK - Write blocks

ENTRY CONDITIONS: (A1) DSP address

(A5) Next BCW address

(A6) DNT address

(A7) Base address of DSP buffer pointers

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

(A1) DSP address

(A6) DNT address

(A7) Same value as input

STEPFLOW:
1. If buffer more than half full, call CIO at entry point WDCS.
2. Update DSP.
3. Exit.

4.2 CIRCULAR I/O ROUTINES (CIO)

Physical I/O on a dataset uses a circular buffering technique initiated by a set of STP common routines known as CIO (Circular Input/Output).

CIO routines are directly callable from system tasks. The following system tasks directly call CIO within COS:

Exchange Processor (EXP)
Log Manager (MSG)
Permanent Dataset Manager (PDM)

CIO calls either the Disk Queue Manager (DQM) or the Tape Queue Manager (TQM) to perform physical sector transfers. These calls occur through intertask communication (PUTREQ) from CIO.

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 fixed-length sectors. The default number of sectors is defined as installation parameter I@DNBFZ sectors. For a COS blocked file, the first word of each sector is a block control word. The size and location of the buffer are normally defined when the DSP is generated. The default size is defined by an installation parameter.

Logical I/O on a buffer can be concurrent with physical I/O. That is, on a read operation, the user can be extracting data from the buffer at the same time the system is inserting data, with the user read lagging the system read (sometimes referred to as read-ahead).

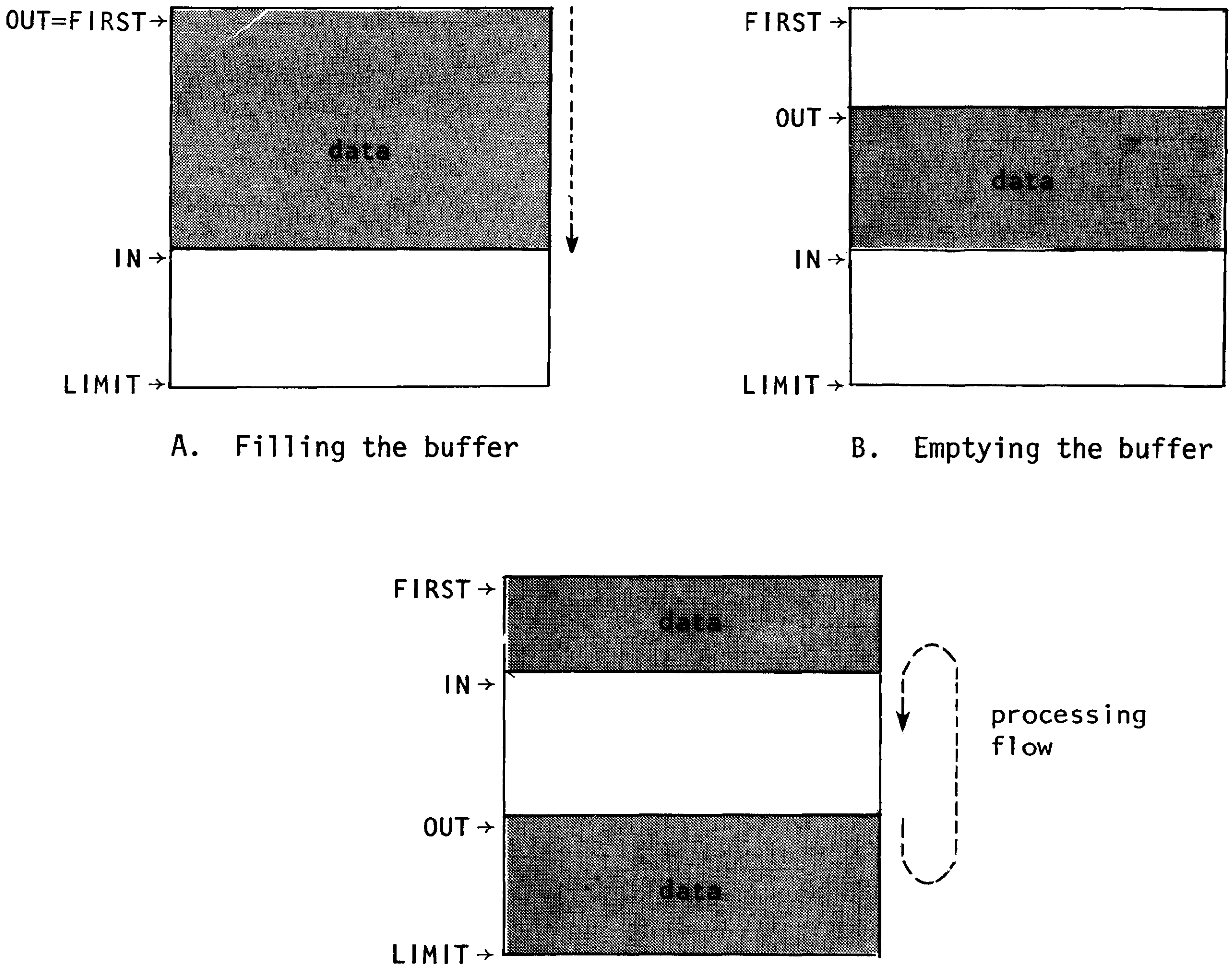
Alternatively, on a write operation, the user can be inserting data into the buffer at the same time the system is emptying it. In this case, the user write leads the system write (sometimes referred to as write-behind).

The buffers are managed through the IN, OUT, FIRST, and LIMIT pointers in the DSP. Figure 4-4 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.



A. Filling the buffer

B. Emptying the buffer

C. Concurrently filling and emptying the buffer

Figure 4-4. Physical I/O

Physical reads and writes always involve C@BLKSZ words. On a read, IN is always at a sector boundary, but OUT, which is being modified by the user, need not be. Conversely, on a write, OUT is always at a sector boundary but IN need not be.

On a read operation, the physical device queue manager (DQM, TQM) 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.

****

CAUTION

When executing on multiple CPU machines such as the CRAY X-MP, it is possible for the operating system to be executing in one CPU at the same time that the user program is executing in another CPU. When both the user and the operating system are operating on the same DSP, a timing condition can exist which might cause the user to believe that the buffer is full (IN=OUT and not busy following a read) when in fact the buffer is empty. This timing condition occurs when the user examines IN after the operating system has set it. During the filling of the buffer, after the user has emptied the buffer, the user can empty the buffer quickly enough so that the operating system has not yet cleared the DPBSY field when the user reads it. If the user program then enters recall believing that an I/O operation is still active, when resumed, the buffer is in fact empty, since no I/O request was actually outstanding -- the previous request was being terminated. It is the user program's responsibility to determine whether the buffer is full, and if not, to initiate an I/O request. All CRI products and library routines concerned with I/O correctly determine the true state of the buffer, and re-issue I/O requests when necessary. It is the responsibility of the developer of any non-CRI program to make the necessary modifications. This caution does not apply to those programs which wait until all I/O has completed on the dataset before attempting to reference the IN and OUT pointers.

****

Dataset I/O streaming occurs when the user is able to remove data from the buffer (on a read) quickly enough, so the buffer always has room for the queue manager to initiate another physical I/O request when the previous request completes.

On a write operation, the physical device queue manager 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 is not equal to 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.

****

CAUTION

When executing on multiple CPU machines such as the CRAY X-MP, a timing condition can exist which might cause the user to believe that the buffer is empty (IN=OUT and not busy following a write) when in fact the buffer is full. See the preceding caution concerning a false buffer full condition following a read.

****

A mass storage dataset can 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*C@BLKSZ≤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 block boundaries within the buffer, and
4. Sets the EOI bit in DSP. Any I/O suspend calls made to the Job Scheduler are canceled before returning.

If mass storage space is allocated and the dataset size from the Dataset Allocation Table (DAT) is greater than the buffer size, CIO clears the memory-resident indicators and the read proceeds normally.

4.2.1 CIO ENTRY POINTS

Three main entry points within CIO are externalized for direct linkage between system tasks and the CIO routines:

CPROC - Main read/write entry point to CIO
CTRCL - Synchronous System Task dataset recall
CRCIO - Asynchronous System Task dataset recall

A system task calls CPROC to initiate I/O on a dataset. The type of recall the system task performs depends on whether the system task has no other processing to do while the I/O is in progress (synchronous), or whether the system task has other processing to do while the I/O is in progress (asynchronous).

Calls to CTRCL can suspend the calling system task until the DQM/TQM acknowledge has been received.

Calls to CRCIO assume the initiating system task has already received the DQM/TQM acknowledge and only wants to perform common acknowledge processing.

4.2.2 CIO MAIN READ/WRITE ENTRY

The following describes the entry and exit parameters for the main entry point into CIO:

```
ENTRY CONDITIONS:  (A1)  TXT address of the task entry (if user task)
                         0 (if system task)

                   (A2)  DNT address

                   (A3)  DSP address

                   (A7)  Calling system task stack address-XXXSTK.  This
                         address must also be stored in field DNSTK in
                         the DNT before the call.

                   (S1)  0 for Read
                         1 for Write

                   R    CPROC
```

RETURN CONDITIONS: (A1) TXT address of the task entry (if user task)
0 (if system task)

(A2) DNT address

(A3) DSP address

(A7) Stack address

Control returns immediately to the caller.

The field DNRCL should be set if the caller wishes to be I/O suspended when the I/O is initiated and I/O resumed when the I/O is done. This field is set automatically by CIO in CTRCL. The equivalent of this field within CIO is DNCRC which gets set whenever a user task is to be I/O suspended and cleared when the task is I/O resumed.

4.2.3 CIO SYNCHRONOUS RECALL

The following describes the entry and exit parameters for the synchronous entry point into CIO:

ENTRY CONDITIONS: (A1) TXT address of the task entry (if user task)
0 (if system task)

A2 = DNT address

A3 = DSP address

A7 = Calling system stack address-XXXSTK. This address must also be stored in field DNSTK in the DNT before the call.

R CTRCL

RETURN CONDITIONS: (A0) 0 if no error, nonzero if error

(A1) TXT address of the task entry (if user task)
0 (if system task)

(A2) DNT address

(A3) DSP address

(A7) Stack address

(S0) 0 if no error, nonzero if error

Control returns to the caller when the entire request has been completed.

The field DNRCL should be set if the caller wishes to be I/O suspended when the I/O is initiated and I/O resumed when the I/O is done. This field is set automatically by CIO in CTRCL. The equivalent of this field within CIO is DNCRC which is set whenever a user task is to be I/O suspended and cleared when the task is I/O resumed.

4.2.4 CIO ASYNCHRONOUS RECALL

The following describes the entry and exit parameters for the asynchronous entry point into CIO:

ENTRY CONDITIONS: (A1) TXT address of the task entry (if user task), 0 (if system task)

(A2) DNT address

(A3) DSP address

(A7) Calling system stack address-XXXSTK. This address must also be stored in field DNSTK in the DNT before the call.

(S1) DQM/TQM reply word 0. See section 4.1 for the reply word format.

(S2) DQM/TQM reply word 1

R CRCIO

RETURN CONDITIONS: (A0) 0 if no error, nonzero if error

(A1) TXT address of the task entry (if user task), 0 (if system task)

(A2) DNT address

(A3) DSP address

(A7) Stack address

(S0) Zero if no error, nonzero if error

Control returns to the caller at its main interrupt loop.

4.3 MEMORY ALLOCATION/DEALLOCATION ROUTINES

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

Allocation and deallocation are from memory pools. The number and size of memory pools are determined when the operating system is generated.

As illustrated in figure 4-5, 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.

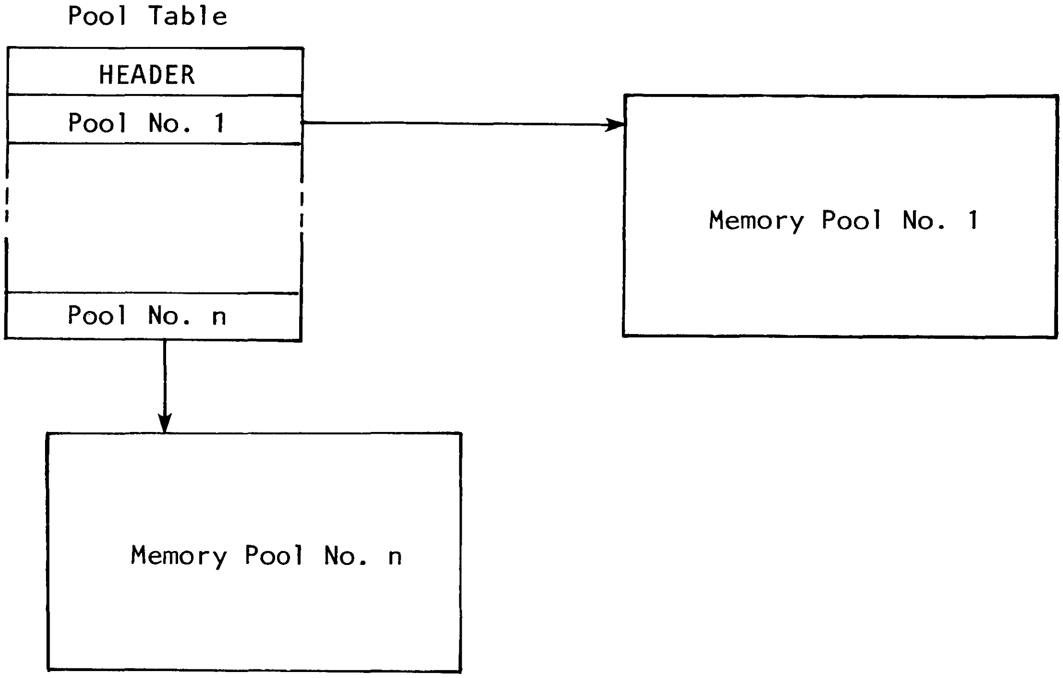


Figure 4-5. Memory allocation tables

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 size of the areas change dynamically as tasks obtain words from or return words to a pool.

4.3.1 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 pool number 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 is allocated, the address of the first usable word. The allocated area is zeroed.

ENTRY CONDITIONS:	(A6)	Number of memory pool from which to allocate
	(A7)	Number of words desired
RETURN CONDITIONS:	(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.

4.3.2 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.

ENTRY CONDITIONS: (A6) Memory pool number

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

RETURN CONDITIONS: (A6) Status:

0 Good return
1 Invalid address
2 Area not currently allocated
3 Invalid memory pool number

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

4.3.3 PARTIAL MEMORY DEALLOCATION - PMEMDE

PMEMDE is an STP common subroutine that returns a portion of memory to its memory pool for reallocation. The portion being returned can be at either end of the allocated space. Memory cannot be returned to the pool from the middle of the allocated space. The freed area is combined with any adjacent available space.

ENTRY CONDITIONS: (A6) Memory pool number

(A7) Address of first usable word of allocated area

(A5) Count of words to free. If (A5) is negative, ABS (A5) words are released from the beginning of the allocated space. If (A5) is positive, (A5) words are released from the end of the allocated space.

RETURN CONDITIONS: None The requested number of words have been made available for other use.

The minimum request to PMEMDE is to release three words. A request to deallocate fewer than three words is ignored without comment unless the request would result in the deallocation of the entire allocated area. If the request is greater than or equal to the size of the allocated area, the entire area is released.

4.4 CHAINING/UNCHAINING SUBROUTINES

The CHAIN and UNCHAIN 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 can 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 4-6 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 exist on the chain and a count of the number of items on the chain. However, because the chain control word reflects 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 where this item is linked.

4.4.1 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.

ENTRY CONDITIONS:	(A6)	Address of chain control word
	(A7)	Address of the item to be chained
RETURN CONDITIONS:	(A6)	Unchanged from input
	(A7)	Unchanged from input

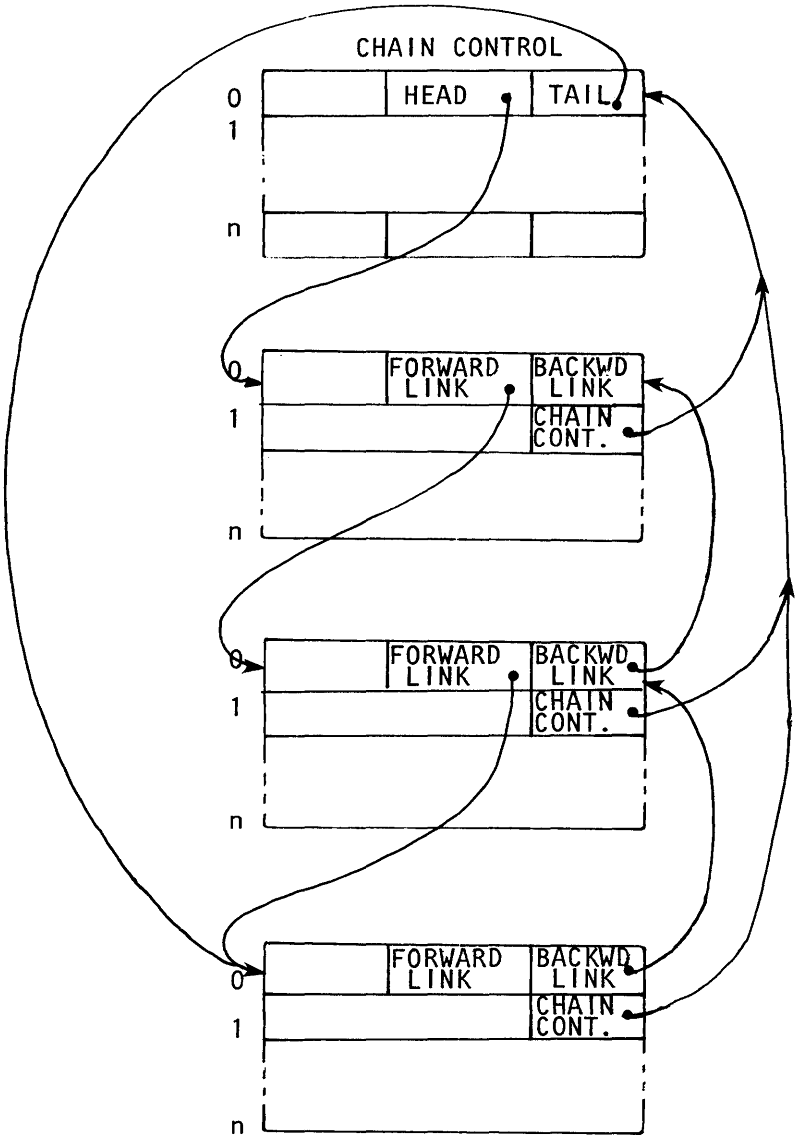


Figure 4-6. Chain tables

4.4.2 UNCHAIN ITEM - UNCHAIN

UNCHAIN is an STP common subroutine that removes an item from a queue. The item to be removed can 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.

ENTRY CONDITIONS: (A7) Address of item to be unchained

RETURN CONDITIONS: (A7) Unchanged from input

4.5 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 in nonpreemptable mode.

The interactive communication buffer area is in the upper 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.

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

```
0                 16                32                48              63
|     QCOUNT      |////////////////|     QTAIL        |     QHEAD       |
```

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

4.5.1 ENQMSG ROUTINE

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

(A4) Queue control word address

(A5) Message length

(A6) Message address

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

Registers A1 through A4, A7, and S1 through S5 are saved and restored.

4.5.2 NXTMSG ROUTINE

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

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

(A4) Queue control word address

(A5) Maximum move size

(A6) Move address

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

	(A4)	Queue control word address
	(A5)	Buffer ID
	(A6)	Address within record area where next buffer can be moved

Registers A1 through A4, A7, and S1 through S5 are saved and restored.

4.5.3 FREEMSG ROUTINE

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

ENTRY CONDITIONS:	(A4)	Queue control word address
	(A5)	Buffer ID
RETURN CONDITIONS:	(A0)	Operation status: =0 Successful ≠0 Buffer not on queue

Registers A1 through A5, A7, and S1 through S5 are saved and restored.

4.6 PASSWORD ENCRYPTION

PWENC is a common routine used to encrypt passwords. The parameters are passed through the Encryption Parameter Table (ETT), which is described in publication SM-0045, the COS Table Descriptions Internal Reference Manual. The table parameters currently used are:

- The password to encrypt
- The ordinal of the keyword to use in the encryption

The supplied encryption algorithm contains three keywords; any one may be specified for use in the encryption. PWENC replaces the password to be encrypted by its encrypted version in the ETT.

SUBROUTINE NAME: PWENC - Password encryption

ENTRY CONDITIONS: (S1) User ETT address

EXIT CONDITIONS: Encrypted passwords replace unencrypted passwords in ETT

(S0) Status:
=0 Normal return
<0 Invalid keyword index specified

4.7 SYSTEM BUFFER MANAGEMENT

The System Buffer or SYSBUF is an area of memory between PDM tables and user memory. This places the buffer area very high in Central Memory. This buffer zone is used by SCP and STG for COS/front-end communication buffers. This is the same relative location of memory that the communication buffers were allocated from in pre-1.12 releases of COS.

The original buffer is allocated by JSH and is I@SYSBUF words. As more space is needed, the buffer manager, a common subroutine called BFMAN, requests JSH for I@BFINCR words to be added to the System Buffer. The maximum size of the buffer area cannot exceed all of user memory because STP cannot be rolled. If there is space that is unused (2*I@BFDECR words or more), BFMAN requests JSH to deallocate I@BFDECR words of memory from the available buffer pool. Memory is added or removed from the end of the buffer adjacent to user space, which means that availability of user memory space is affected by fluctuations in communication load. Figure 4-7 shows these memory usage states.

The allocation and deallocation of buffers within the System Buffer is handled by the BFMAN common subroutine. There are two types of allocation requests:

- A request for space

- A reallocation

In the second form of allocation request, an existing buffer is traded for another buffer, allowing BFMAN to attempt to pack buffers together.

Allocation is always done on a first-fit basis, starting at the highest addresses to force buffer space to be at the user end of the System Buffer. If no buffer space is available, the request is rejected and a request for more memory is posted to JSH. The requester should wait and then renew the request. SCP and STG reissue such requests after an exchange of messages with the front end. This message exchange allows enough time for JSH to allocate the memory, and ensures that there are no front-end timeouts while SCP or STG are waiting for memory. However,

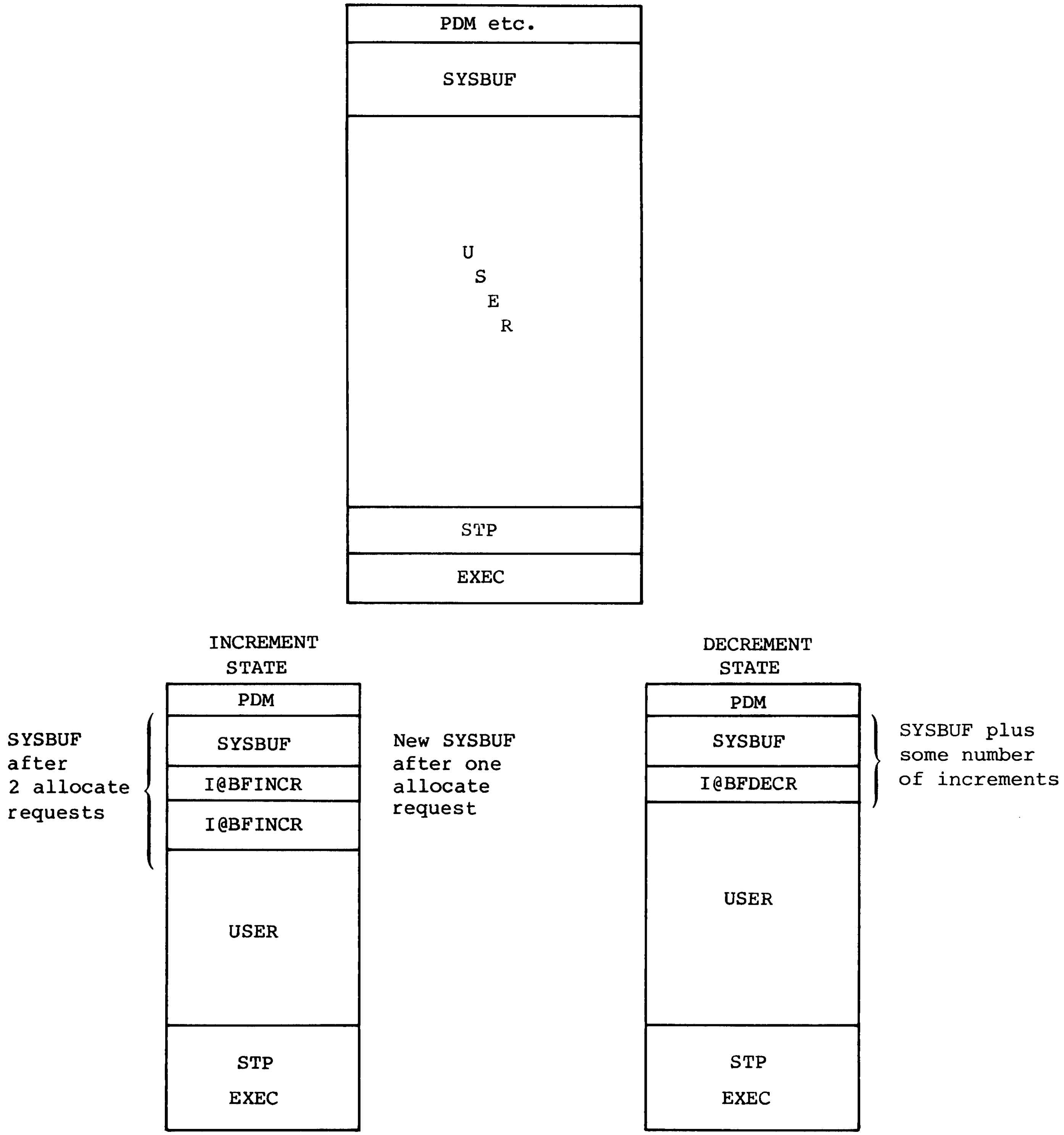


Figure 4-7. System Buffer memory management

suspensions of a transfer or initially slow transfers may result. If SCP or STG is reallocating a buffer, then BFMAN should be able to at least reallocate the same buffer.

The processing of a deallocation request is a simpler process. The space is freed and the BFMAN routine attempts to combine the free space with adjacent free buffers.

4.7.1 SYSTEM BUFFER INITIALIZATION

The BFMAN common subroutine uses control words within the System Buffer to manage the individual buffers. A control word precedes and terminates each buffer within the System Buffer, as shown in figure 4-8.

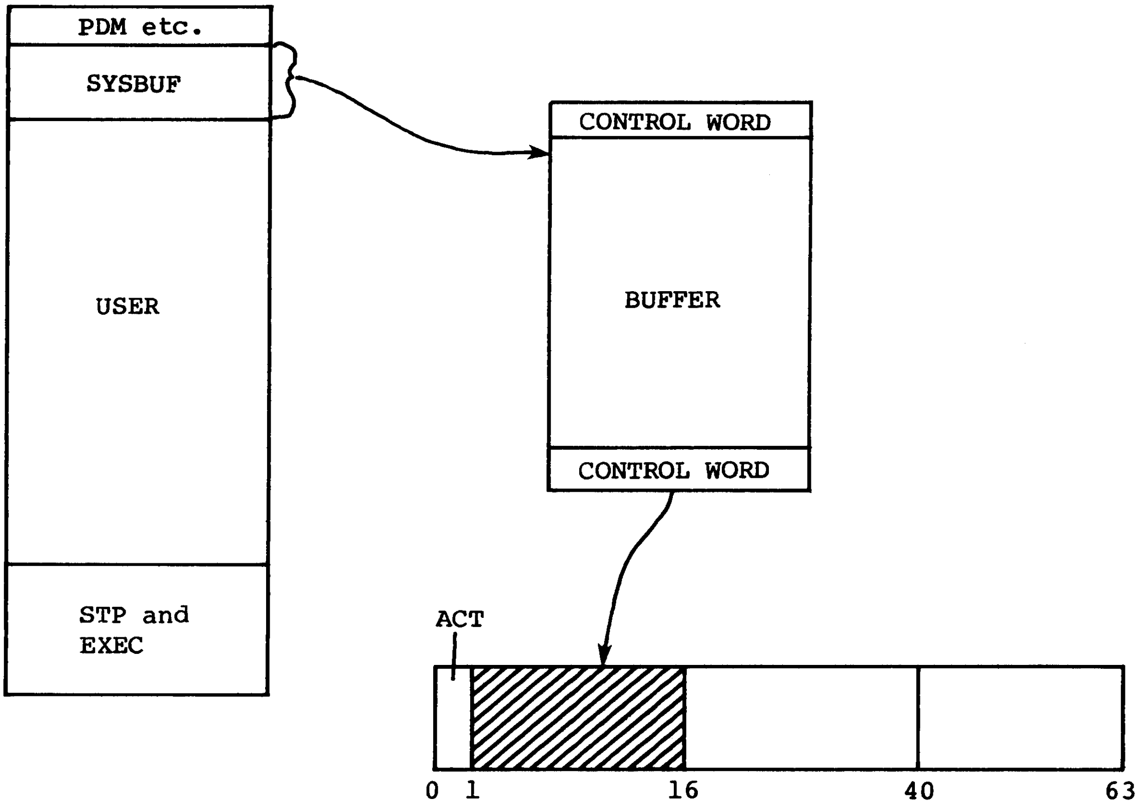


Figure 4-8. System Buffer control words

Each control word has three fields.

- An Activity flag (ACT) indicating that the buffer above the control word, with a higher address, is allocated and active.
- The higher pointer (HP), which points to the next control word having an address greater than the current control word.
- The lower pointer (LP), which points to the next control word having an address lower than the current control word.

The high and low pointers provide links through the buffer chain that the buffer manager traverses when looking for allocatable space.

4.7.2 SYSTEM BUFFER INTERNAL MANAGEMENT

The BFMAN routine initializes the System Buffer by putting control words in the first and last words of the buffer, and setting the highest address in the variable BUFMAX and lowest address in BUFMIN. As shown in figure 4-9, the variables BUFMAX and BUFMIN provide pointers to the buffer limits.

In the just initialized System Buffer, the control word at BUFMAX has the Active flag set. This flag is a secondary indicator of a lack of buffer space above that point. The higher pointer is 0, indicating no control word above BUFMAX. The lower pointer points to the control word at BUFMIN where the only other control word has been placed. The control word at BUFMIN does not have the Active flag set indicating that the space between that control word and the control word pointed to by the higher pointer, pointing to BUFMAX, is available for allocation. The lower pointer is 0, indicating no space exists beyond this control word.

4.7.3 BUFFER ALLOCATION

In the allocation of a buffer, the BFMAN routine searches the buffer space for the first available space at least large enough for the requested buffer. The search starts at the highest memory address, forcing buffers to be allocated as high in the buffer zone as possible. If a perfect fit is found, then only the Active flag need be set and the request is complete. If a perfect fit is not available, a new control word is created.

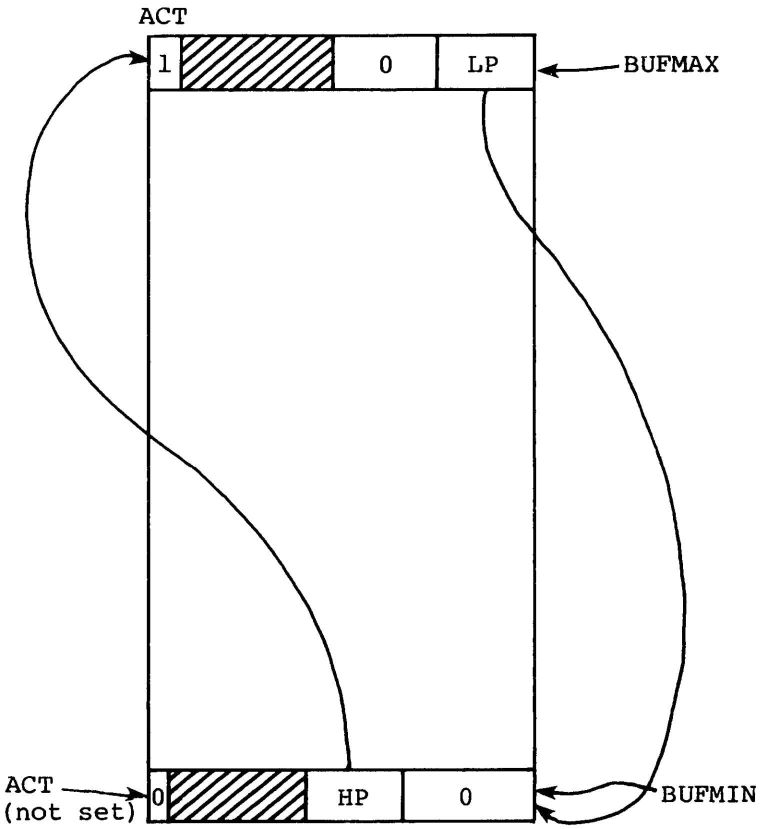


Figure 4-9. Initialized System Buffer

The following method is used for allocations requiring creation of a control word.

1. Subtract the requested buffer size plus one from the address of the control word above the available space, giving the address of the new control word, NCW.

2. Subtract the request buffer size from the higher pointer of the control word below the available space (HPB), creating a new HPB, or HPB1.

3. Place the old HPB in the new control word higher pointer. Place HPB1 in the control word below the space, that is, in the lower control word (LCW).

4. Place the old LPA in the NCW.

5. Place the address of the new control word (NCW), in the lower pointer of the control word above the space, that is, in the higher control word (HCW).

6. Set the Active flag in the NCW.

Figure 4-10 illustrates System Buffer space allocation.

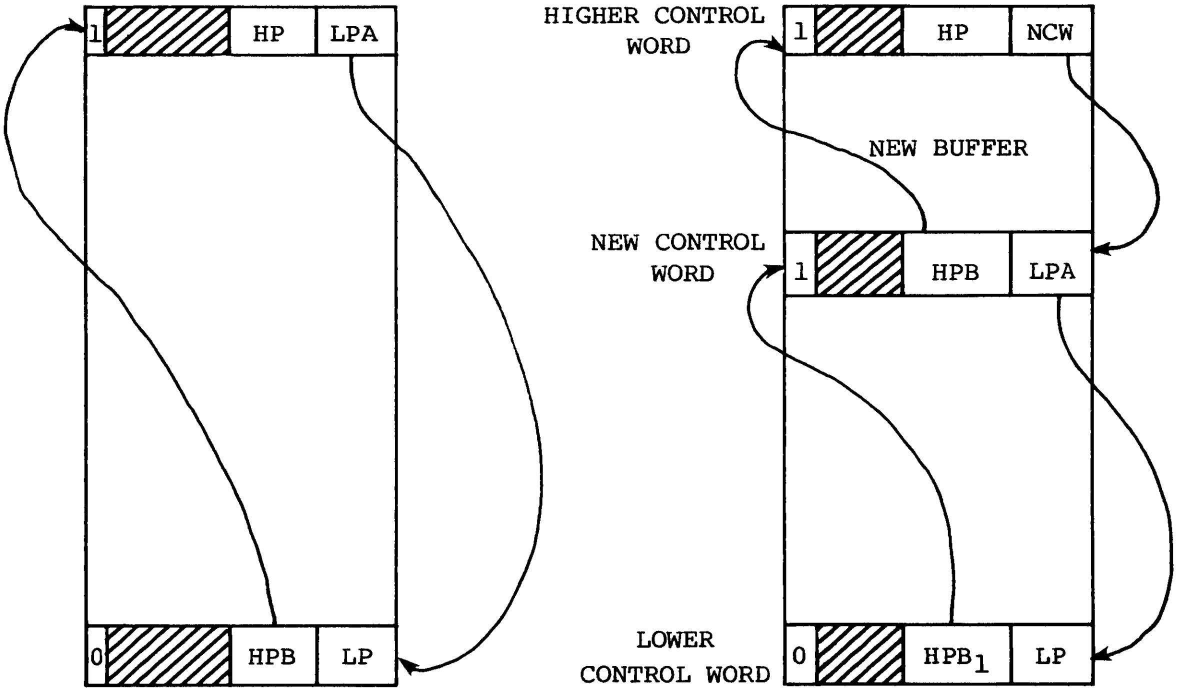


Figure 4-10. System Buffer space allocation

4.7.4 SYSTEM BUFFER DEALLOCATION

The deallocation of buffers within the System Buffer can be separated into two cases:

- Deallocation that does not allow merging of either of the adjacent buffers or spaces with the area to be deallocated
- Deallocation that does allow at least one of the adjacent areas to be joined

Deallocation that does not allow the merger consists of clearing the Active flag in the control word below the buffer. The deallocation with merge is more complicated. The following procedure shows the processing necessary to merge both adjacent buffers. Deallocation where only one adjacent buffer is inactive is only slightly different.

1. Take the control word's higher pointer and pick up the HCW.
2. Verify that the HCW's lower pointer does point to the original control word (OCW).
3. If the Active flag is not set in the HCW, use that pointer for the OCW higher pointer.
4. Take the OCW's lower pointer and pick up the LCW.
5. Verify that the LCW's higher pointer points to the OCW.
6. If the Active flag is not set, then replace the OCW's lower pointer with the LCW's lower pointer.
7. Place the LCW's address in the HCW's lower pointer.
8. Place the OCW at the LCW's address.

Figure 4-11 illustrates System Buffer space deallocation.

4.7.5 SYSTEM BUFFER PERFORMANCE CONSIDERATIONS

Use of memory within the System Buffer is the primary performance consideration. Optimum use is achieved when all allocated buffers are packed in the upper end (higher addresses) of the System Buffer. The lower end of the System Buffer should contain enough space for frequent bursts of front-end activity but should not have space that is unused for long periods.

Achieving optimum use requires the following:

- Buffers must be packed; that is, holes must be squeezed out.
- Space at the end of the buffer area must be monitored so that unused space can be returned to the user area of memory.

The first-fit method of allocation is one means of achieving buffer packing. The other is assuring that buffers do not stay allocated for long periods. The tasks SCP and STG reallocate buffer space after each use. In the case of output to a front end after a segment buffer is transmitted, the BFMAN routine is called to reallocate the buffer. The worst case of reallocation is that the same buffer is reallocated. The only cost of reallocation is CPU overhead.

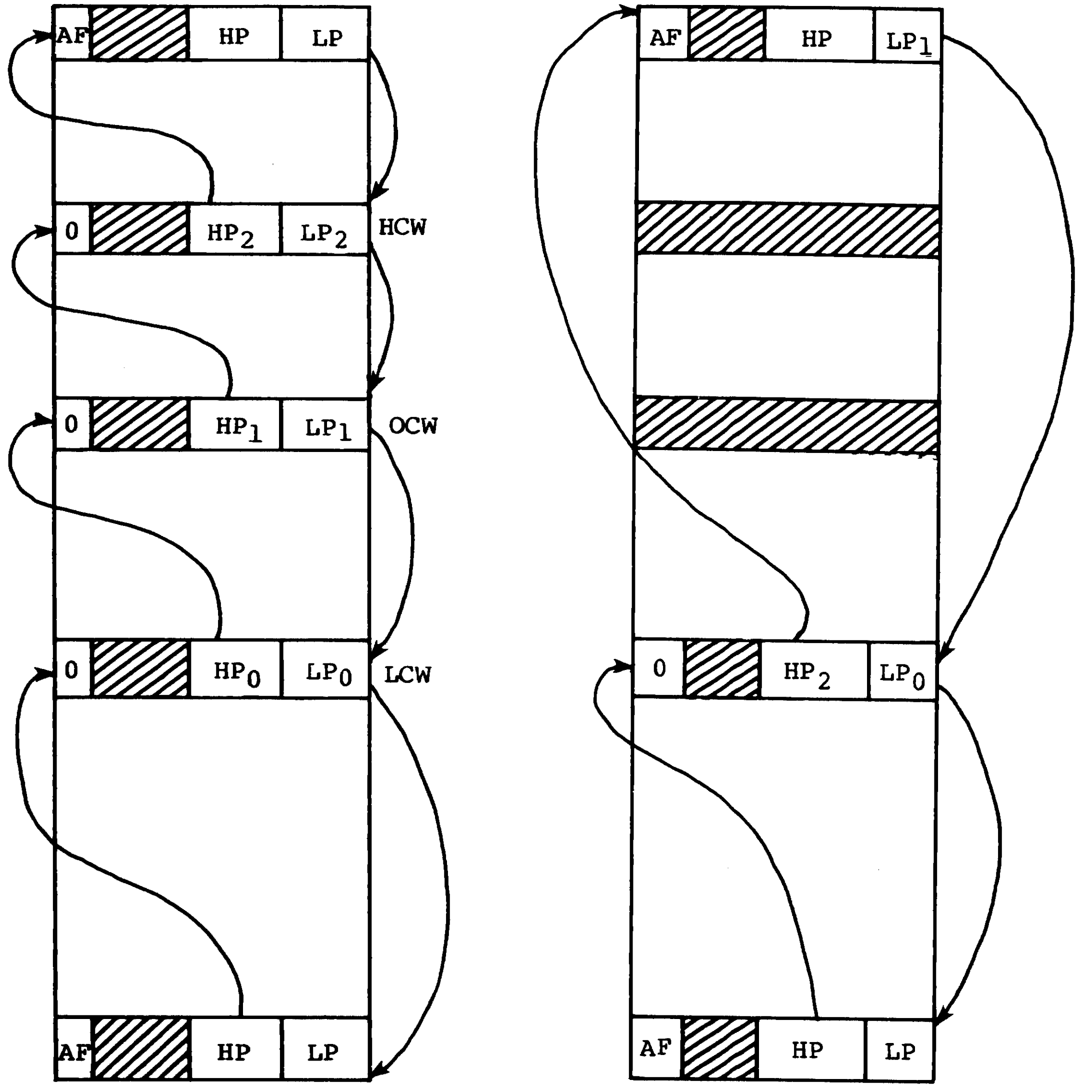


Figure 4-11. System buffer space deallocation

The second consideration is monitoring space at the end of the System Buffer. Having enough extra space to efficiently buffer data can be balanced against a need for keeping unused space at a minimum, thus increasing the amount of user memory. This monitoring is accomplished, in part, by setting a timer if the size of the space passes some threshold. The space is returned if it stays over the threshold size until the timer expires. The other part of this balance ensures that the buffer allocation increment and decrement are set to values that avoid constant requests to JSH.

COS STARTUP 5

System startup is the process of loading the Cray Operating System (COS) into Central Memory, beginning execution, and generating or recovering tables for the operating system. The operating system to be started can be on a user permanent dataset. If it is on a user-permanent dataset, then a 2-pass startup is performed, and the parameter file must specify a 2-pass startup. There are three ways to start the system:

- Install
- Deadstart
- Restart

Most of COS Startup resides in the System Task Processor (STP) so that it can conveniently access system tables and facilities. However, some COS Startup logic resides in the station software of the station from which startup occurs (such as the I/O Subsystem station) and in EXEC. Some tables, such as the Permanent Dataset Information Table, are initialized when STP is assembled.

5.1 INSTALL OPTION

With Install, COS is started as if for the first time. All CRAY-1 or CRAY X-MP mass storage is assumed to be vacant, except for areas reserved for Cray Research customer engineers and for the Engineering Flaw Table (EFT). Briefly, when the Install option is selected, the Startup task:

- Searches for the EFTs, if they exist
- Writes a device label (DVL) on each mass storage unit
- Accumulates flaw information. Flaw information can originate from the parameter file, from flaws assembled into COS, or from the Engineering Flaw Table (EFT) which is constructed by engineering diagnostics and written to sector 17 of the first track on the device with a usable sector 17.
- Processes mass storage groups. Mass storage groups are described in section 5.2.
- Creates the Dataset Catalog (DSC) on the master device

- Sets up the DSC and tables in memory, indicating that the only existing dataset, permanent or temporary, is the one containing the DSC itself
- Reserves space on the master device for system dumps for use if the system fails
- Reserves space for the three datasets maintained by an IOS
- Allocates disk space to contain copies of the system overlay dataset
- Initializes the Rolled Job Index dataset (RJI) and enters it into the DSC
- Optionally creates the Dataset Catalog Extension Table (DXT) on the master device (or on some other device according to installation parameter) and enters it in the DSC
- Initializes job class structure and system directory datasets and enters them into the DSC
- Allocates disk space for volatile device backup dataset

When Install writes a device label on each mass storage unit, it uses the first track on each mass storage device that the Device Reservation Table (DRT) identifies as 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.

The device label on the master device points to the DSC by containing the Device Allocation Table (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.

The Install procedure accumulates flaw information for a device during startup. (Startup flaws are described in detail in section 5.5.) At the end of the Install process, Install packs this information into the device label created in memory and writes the label to the disk. Install acquires flaw information from the following: the previously existing label, if one exists; the Engineering Flaw Table, if one exists; information assembled into the Disk Reservation Table entry for the device; and parameter file directives. Whatever Install writes to disk is validated; any flaws are noted in the DRT and the device label.

Install creates the Dataset Catalog (DSC) in blocked format on the master device while logging any disk errors found in the Device Reservation Table (DRT) and in the flaw table in the DVL image being built in memory. Install initializes the DSC to all zeros, except for block and record control words.

Install optionally creates the Dataset Catalog Extension (DXT) in a manner similar to the DSC. It can be created during Install or Deadstart, and is fully described in section 5.2 (Deadstart option).

When Install reserves space on the master device for system dumps for use if the system fails, it zeroes the reserved area. Any flaws are entered into the DRT and flaw tables. The amount of space allocated for system dumps is determined from the value of the installation parameter I@DMPSIZ. Install writes the list of allocation indexes allocated to the system dump to the first sector of the first track in the reserved area, so that Deadstart and Restart can use it.

If an I/O Subsystem (IOS) with at least one disk is part of the installation, Install reserves and zeroes space for the three directories maintained by an IOS. The amount of space allocated for each directory is determined by the installation parameters I@IOPCOS, I@IOPPRM, and I@IOPIOP. Install also writes the allocation index list to sector 0 of each dataset.

In general, I/O on a dataset requires a buffer, Dataset Parameter Table (DSP), Dataset Name Table (DNT), and Dataset Allocation Table (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 is set aside for post-Install use by Startup. However, Install must complete initialization of the DNT and the DAT.

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

A 2-pass startup is meaningless during an Install. If one is requested, Startup halts and issues a message to the operator in the S registers explaining why it has halted.

Install allocates space for the backup datasets for all volatile devices, then saves these datasets as *$dname,* where *dname* is the corresponding device name.

5.2 DEADSTART OPTION

For a Deadstart, COS is started as if after a normal system shutdown. That is, permanent datasets mentioned in the DSC are preserved through proper setup of tables in memory. However, input or output queues mentioned in the Dataset Catalog are deleted.

Briefly, when the Deadstart option is selected, the Startup task:

- Searches for the Engineering Flaw Table (EFT)
- Finds device label (DVL) on each mass storage unit
- Preserves flaw information
- Processes mass storage groups
- If master device, reserves DSC and the disk space occupied by system dump; initializes DNT and DAT for the DSC.
- Preserves allocated space for the three datasets maintained by I/O Subsystem
- Attempts to locate and reserve all disk space allocated for system overlay dataset copies
- Restores all data on volatile devices from the backup datasets as directed by the parameter file
- Deletes all input and output datasets and reserves all other permanent datasets
- Either creates the DXT or recovers and validates the DXT if one already exists
- Establishes Rolled Job Index in memory
- Copies system dump, if one exists, from the preallocated area to available space and saves the copy as a permanent dataset
- For volatile devices, either allocates and saves backup datasets, or invalidates information contained on the previously existing datasets

5.2.1 DEVICE SPACE RESERVATION

Deadstart updates the Device Reservation Table in memory to reflect the tracks reserved by datasets mentioned in the Dataset Catalog and tracks mentioned in the flaw portion of the device label, the Engineering Flaw Table (EFT) on the disk, if one exists, and parameter file directives.

Deadstart rewrites the label on the disk at the end of Deadstart, if any flaws other than those already mentioned in the device label are added by the EFT or the parameter file, or are specified to be deleted by the parameter file.

Deadstart attempts to locate the disk area preallocated for the system dump. If this area can be found, Deadstart reserves the tracks in the Device Reservation Table. If a dump that has not been copied exists in the preallocated 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 preallocated area is not copied. If the preallocated area cannot be found, no space is reserved in the Device Reservation Table.

5.2.2 MASS STORAGE GROUPS

The installation can choose to group several physical mass storage devices together as one logical device. The logical device is called a *stripe group*. When device striping is used, Startup sets up the Equipment Table (EQT) entries to reflect all such groups.

Groups are identified by a group identifier contained in the device label. Devices can be added to or deleted from the group and new groups can be defined during Startup by an entry in the parameter file or by an operator entry during configuration change processing.

The logical device corresponding to a stripe group must be present in the EQT or added during configuration change processing and must have the device name STRIPE-*n*. *n* is a decimal digit in the range 1-9. Up to 9 stripe groups can be defined; each stripe group can contain up to 7 physical devices. Device Reservation Table (DRT) space must also be available the logical device.

Following processing of all groups, Startup sets all physical members of a group to indicate the device is down and all datasets are released. Startup discards all permanent datasets residing on a physical member of a group. Permanent datasets residing on the logical group device are retained unless the device is configured as released. If permanent datasets reside on a group device, devices can not be added to or deleted from the group.

When a group member is identified, all flaw information for that member is merged into the DRT entry for the group device. Thus a flawed track on one physical device causes that track to be unused on all devices that are members of the same group.

See the COS Operational Procedures Reference Manual, publication SM-0043, for a description of how to define a group, add a device to a group, or delete a device from a group.

The master device cannot be part of a mass storage group.

5.2.3 DATASET CATALOG EXTENSION

Deadstart locates or creates the Dataset Catalog Extension Table (DXT). The DXT is a system dataset similar to the DSC itself and serves as a repository for information that will not fit into the DSC conveniently.

The DXT is a permanent dataset with a permanent dataset name of $DSC-EXTENSION and edition number of 4095. Startup ensures that this dataset belongs to the system and prohibits unauthorized access. An exception is made for utilities such as PDSDUMP and AUDIT.

The DXT is created during an Install or a Deadstart. Once created, subsequent Deadstarts or Restarts recover the DXT dataset during the normal DSC recovery and validate the DXT information. When the DXT is created, it is, by default, placed onto the master device; overflow to another device is not allowed. An installation parameter can be changed allowing both a nonmaster device to be selected and/or overflow to occur. Contiguous disk allocation for the DXT is not guaranteed, even though it can be requested.† A site might wish to free disk space on the selected device through PDSDUMP and PDSLOAD before a Deadstart. Although contiguous disk space for the DXT is not required, it enhances the AUDIT,PERMIT and/or AUDIT,TEXT,NOTES and/or AUDIT,B=*bdn* performance significantly.

An installation parameter controls the size of the DXT. The parameter can be altered during Startup through a Startup parameter file directive. Startup can be requested to create the DXT or to increase the size of the DXT.

To provide high performance in allocating entries in the DXT and to minimize the impact of introducing another element in the structure of the current permanent dataset directory, a memory resident allocation table is maintained in the upper end of memory.

† When contiguous space is requested, Startup ensures that the contiguous space is available. However, even if the requested contiguous space is available, there is no guarantee that DQM will allocate it; DQM currently does not guarantee contiguous allocation. If contiguous space is requested but unavailable, a fatal error occurs.

The public access mode and access tracking attributes are placed in the main DSC entry. This placement is primarily done to reduce the I/O requirements for the default use of AUDIT. All other attributes (permits, tracking, text, and notes) are placed in the DXT. Each permanent dataset has its own chain of DXT entries which significantly reduces the search time required when interrogating permit information at the expense of increased disk space needed for the DXT. The DXT entries are designed to be installation friendly; that is, additional DXT entry types can be defined by a site without concerning themselves with future changes that might be made by CRI.

Any inconsistencies between the DSC and DXT entries for a given permanent dataset encountered during Startup cause the DXT error flag to be set in the main DSC entry for the dataset and cause a message to be posted to both the System Log and the operator.

5.2.4 OTHER STARTUP PROCESSING

If an I/O Subsystem (IOS) with at least one disk is part of the installation and space has been allocated for the three datasets maintained by the IOS, the space is preserved. If the space has not been allocated, it is allocated following permanent dataset and rolled job recovery.

Before system dump processing, Deadstart calls the RRJ routine (Recover Rolled Jobs) to set up the Rolled Job Index. Deadstart attempts to access the Rolled Job Index dataset. 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. If the dataset cannot be accessed, Deadstart creates and initializes a new edition.

Deadstart also attempts to locate the system overlay area. If it can be found, Deadstart reserves tracks for it in the DRT. If not found, it is allocated.

Deadstart scans the Dataset Catalog for input and output datasets, deleting all such datasets to create an idle system. Permanent datasets are preserved.

The buffers and Dataset Parameter Tables (DSPs) for the Dataset Catalog and its extension are assembled into STP.

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

If a 2-pass Deadstart is requested, Deadstart locates the specified system dataset and reads it 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 attempts, for all volatile devices, to access the backup dataset, $*dname*, where *dname* is the corresponding device name. If such a dataset exists, the data contained on it is marked invalid. Otherwise, Deadstart allocates space and saves the dataset.

5.3 RESTART OPTION

Restart is an operator option after a system interruption when recovery of input and output queues and possibly the jobs in process is desirable.

Briefly, when the Restart option is selected, the Startup task:

- Searches for the Engineering Flaw Table (EFT)
- Finds device label (DVL) on each mass storage unit
- Reserves flaw information
- Processes mass storage groups. Mass storage groups are described in section 5.2.
- If master device, reserves Dataset Catalog and initializes Dataset Name Table (DNT) and Dataset Allocation Table (DAT) for the Dataset Catalog
- Attempts to preserve the area reserved for system dumps
- Restores information on volatile devices from their associated backup dataset as directed by the parameter file
- Attempts to preserve all permanent datasets and recovers input and output queues. In memory, builds DAT and System Dataset Table (SDT) for each input/output dataset. (The SDT entry can have one or more attached memory pool areas containing TEXT field or station slot information.)
- If specified, recovers rolled out jobs through call to Recover Rolled Jobs routine (RRJ)
- Preserves or allocates space for the three datasets maintained by I/O Subsystem

- Allocates the system overlay dataset
- Locates the Dataset Catalog Extension if available, validates it, and builds an allocation table in upper memory
- Copies system dump if necessary and saves the copy as a permanent dataset (in the same way as for Deadstart; see section 5.2.)
- For all volatile devices if the backup dataset already exists, the data contained on it is marked invalid; otherwise, the dataset is created.

Restart updates the Device Reservation Table in memory to reflect the tracks reserved by datasets mentioned in the Dataset Catalog and tracks mentioned in the flaw portion of the device label, the Engineering Flaw Table (EFT) on the disk (if one exists), and parameter file directives. The label on the disk is rewritten at the end of Restart if (a) any flaws other than those already mentioned in the device label are added through the EFT or the parameter file, or (b) any flaws are specified to be deleted through the parameter file.

If an I/O Subsystem (IOS) with at least one disk is part of the installation, and if space has been allocated for the three datasets maintained by the IOS, the space is reserved. If the space has not been allocated, it is allocated.

The system overlay dataset is reserved in a way similar to the way in which the system dump area is treated. (The system dump area for Restart is recovered and copied as described under Deadstart, section 5.2.) If a system overlay dataset was not 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.

In attempting to recover rolled out jobs, Restart accesses the Rolled Job Index dataset and loads it into memory. If the access or the read receives an error, Restart initializes a new edition and writes it to disk. If the operator chooses not to recover rolled jobs, Restart clears and rewrites the index.

The DSPs for the Dataset Catalog are defined during assembly of STP. 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 encountered in the DSC.

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 preprocessing, 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 nonempty input queue causes JSH to begin job scheduling. Similarly, a nonempty output queue activates SCP. The SCP task was created before the call to the Z routine within Startup.

JSH can 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 first pass of a 2-pass Restart is identical to the first pass of a 2-pass Deadstart. (See section 5.2.)

5.3.1 JOB RECOVERY BY RESTART

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

Startup successfully recovers and restarts all jobs that are 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 repetition of the activities of the job following roll in will not cause the results of the job to change. A job with on-line tapes assigned cannot be recovered.

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, COS attempts to rerun the job from the beginning. If a job is irrecoverable and is ineligible for rerun, Startup returns it to the input queue, and it terminates with an informative message in both the user and system logs as soon as the Job Scheduler attempts to reinitiate the job.

A job that has been initiated but has not been rolled out cannot be recovered since there is no roll image to recover.

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 reaccess the datasets. Any permanent datasets already accessed by a job before roll out must be reaccessed successfully during Startup for a job to be considered successfully recovered.

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

5.3.2 INDEX ENTRY VALIDATION

The first step of validation of job recovery is validation of 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 nonexistent or resides on a nonexistent 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, <u>and</u> if the operator or installation specifies not to recover such jobs.

5.3.3 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. Enough of the JTA must be available for the job to locate the copy of the full roll dataset Dataset Allocation Table (DAT). This DAT was copied along with the Job Execution Table (JXT) image to the Job Table Area (JTA) by the Job Scheduler immediately before rollout. An error on the read renders 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 can 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, and the job is considered irrecoverable.

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 before 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.

5.3.4 DAT VALIDATION

Each dataset, including the roll image dataset, must have a Dataset Allocation Table (DAT) address of zero 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 Job Table Area (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 is 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, which is necessary for successful deallocation of resources if a later dataset has an error.

5.3.5 DATASET RESERVATION

Each dataset named in the Dataset Name Table (DNT) chain in the Job Table Area (JTA) must be processed. Local datasets must have their Dataset Allocation Tables (DAT) validated and the Device Reservation Table (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 beginning to 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 and the dataset device type is not online tape, RRJ goes on to the next DNT. If the device type is online tape, the job cannot be recovered. 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.

5.3.6 PSEUDO ACCESS OF PERMANENT DATASETS

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

If the DAT appears valid, PDM attempts to construct or update a Permanent Dataset Table (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.

5.3.7 RESOURCE DEALLOCATION

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 before finding an error on a later dataset, or any PDS entries corresponding to datasets that have already been pseudo accessed must be deallocated. For this purpose, the Dataset Name Table (DNT) chain is searched until the DNT with the error is reached again. For releasing local datasets, the Disk Queue Manager (DQM) deallocate request is used. For releasing Permanent Dataset Table (PDS) entries, the Permanent Dataset Manager (PDM) request PMFCRL is used. The disk space for datasets such as $CS or $IN, which have their Dataset Allocation Table (DAT) in STP, is not released. The roll image dataset is released and its STP DAT pages are returned to the system.

5.3.8 JOB RECOVERY COMPLETION

When the end of the Dataset Name Table (DNT) chain is reached without error, the job is successfully recovered. Assigned resource information is moved from the Job Execution Table (JXT) to the Generic Resource Table (GRT). The copy of the JXT from the Job Table Area (JTA) is placed in the JXT area, and the JXT entries are relinked by priority. The roll image DNT within the JXT is updated to point correctly to the Dataset Allocation Table (DAT), the System Dataset Table (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. If there is a dataset in the output queue which was disposed by this job with the WAIT parameter, the E (waiting for event) bit is set and an event-wait table entry is constructed. The SDT address in the JTA is corrected, and the JTA is rewritten to the roll image dataset. If the operator or installation has decided to recover and lock out certain jobs (I@LOCK=1), the following conditions cause the JXT and SDT lockout bits to be set and a message to be issued to the System Log.

- JTEPC is nonzero and the date/time stamps in the index entry and the current system do not match.
- The current job size or requested job size exceeds the maximum available.

Under the above conditions with I@LOCK=0, the jobs are recovered as normal and, with I@LOCK=2, the jobs are rerun if possible.

RRJ then advances to the next index entry.

5.3.9 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 System Dataset Table (SDT) so that Control Statement Processor (CSP) will issue log messages when the jobs are reinitiated. 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 logfile messages. The status word RRJSTAT is set to indicate to the Job Scheduler that the JXT entries are already initialized and linked. RRJ then returns to Z, the main Startup routine.

5.4 2-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 mainframe disks.

The procedure followed by pass 1 is identical to that of a 1-pass 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.

Pass 2 of a 2-pass startup is exactly the same as a regular startup except that the *BOOT parameter file directive is ignored (by changing it to a *NOOP directive before beginning pass 2).

5.5 STARTUP FLAW PROCESSING

During the initial installation of a disk drive, engineering diagnostics are executed to analyze the surface of the disk and note any disk addresses that cannot be reliably written and reread. Such areas are called flaws. A special table referred to as the Engineering Flaw Table (EFT) is written to the disk and contains information identifying the flaws found by the surface analysis. By convention, this table is always written at a specified address. If this address is flawed, the table may be offset by as much as 10 tracks (decimal). The address used by the diagnostic is cylinder 0, head group 0, sector 17 (decimal).

If this sector cannot be written and successfully reread, an attempt is made to write the table to the next head group address, same cylinder and sector. This continues to a limit of 10 head groups.

Regardless of which startup option is selected, Startup searches each device for the EFT during the startup process. If Startup finds an EFT, it prevents the operating system from overwriting the EFT by setting the Device Reservation Table (DRT) for the device to indicate that the allocation unit corresponding to the EFT address is not available. Startup first examines the predefined disk location for the EFT. If it does not find an EFT or if an error is received on the read request for the predefined address, Startup advances to the next head group address and tries again. This process continues until Startup has attempted 10 reads. If no EFT can be found on the device, Startup optionally sends a warning message to the master operator station, giving the operator the option of continuing without EFT information. (A *SKIPEFT directive in the startup parameter file means no warning messages are sent.)

Once the EFT search is complete, Startup searches each device for a device label. When the device label is located, Startup reserves the corresponding track in the DRT. The device label can be on the same track as the EFT. If no device label is found, Startup examines the Equipment Table (EQT) entry for the device. A fatal Startup error occurs if no device label is found and if the parameter file does not contain the WDL parameter on a CONFIG parameter file directive.

If a device label exists, Startup reserves in the DRT any tracks indicated in the label as flaws. These flaws normally overlap with flaws indicated by the EFT, but they also normally contain tracks not mentioned in the EFT. If no device label exists and the UP flag in the EQT is set, Startup omits this step. Additional flaws may have been specified by either assembling them into the DRT during system generation (the method usually used to reserve a specific set of disk addresses for use by engineering diagnostics) or by *FLAW directives in the Startup parameter file. If any *FLAW directives were present for the device, Startup automatically assumes that the specified flaws are new; that is, not already in the list in the device label. This causes Startup to force a rewrite of the device label to update the flaw list.

A *DELFLAW directive in the parameter file can cause a flaw previously noted in the device label to be removed. In this case, Startup forces a rewrite of the device label to update the flaw list. Depending on how the flaw being deleted was initially specified, either the flaw is permanently deleted or each subsequent startup repeats the *DELFLAW directive. If the flaw was initially entered through a parameter file *FLAW directive, the deletion is permanent, and the *DELFLAW directive can be removed. If the flaw is mentioned in the EFT, the *DELFLAW directive must be retained. If the flaw is assembled into the DRT during system generation, the *DELFLAW directive must be retained until the system is reassembled without the flaw.

In either case, if there are differences between the flaws accumulated in the DRT and the flaw list in the device label, the flaw list in the label is recreated and the label is rewritten. If the label and the EFT occupy the same allocation unit, the sector containing the EFT is not rewritten. Not rewriting prevents problems encountered during the label rewrite from overwriting the EFT.

5.6 INPUT TO STARTUP

Input to Startup may consist of a parameter file, the Dataset Catalog Extension (DXT) Table, and the $SDR and $ROLL datasets. Startup may also receive configuration and status changes to devices from the system master operator station.

5.6.1 CONFIGURATION CHANGES

Startup can receive configuration information from any of the following sources.

- Information assembled into tables at system generation time
- Information entered through parameter file commands
- Information entered interactively during Startup at the configuration change time

At these times, devices can be added or deleted, or attributes or status can be changed. These devices include any described in the Equipment Table (EQT) or Tape Device Table (TDT)/Tape Configuration Table (CNT). To be able to enter information during the actual Startup processing, the master operator station must support the station message feature. If station messages are supported and the operator enters a CONTINUE reply once all configuration changes are made, Startup scans the Configuration Table (CNT) in STP memory to ensure that everything is correct. Then, from the information contained in the EQT, Startup constructs the Device Channel Table (DCT) and Device Reservation Table (DRT). If any errors are detected during the configuration processing, the operator is informed and, if possible, is allowed to correct the error.

5.6.2 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 can be prepared from punched cards or from the operator station with the aid of a text editor. Each parameter must be terminated by an ASCII carriage return character and cannot be in COS blocked format. Parsing of the command language is performed in COS, eliminating rewriting of the parsing logic for each front-end system. The operator station commands copy the parameter file into Central Memory along with EXEC, STP, and CSP.

The ZY portion of Startup processes the parameter file and modifies memory as specified by the directives. This processing takes place before any of the other tasks are initialized or Startup begins the processing of any information in the STP tables, thus allowing the ability to change critical system installation information without requiring system reassembly. If the operator station handles operator messages, Startup requests configuration and other necessary information during its processing.

5.6.3 DATASET CATALOG EXTENSION DATASET (DXT)

The Dataset Catalog extension is further described in section 5.2. This section describes Startup processing of:

- DXT recovery and validation
- DXT access and control

Recovery and validation

No DXT recovery or validation occurs during an Install or Deadstart when the DXT is being created. DXT creation is described under Deadstart, section 5.2. When the DXT is recovered, Startup completes the intialization of the STP assembled DNT and DAT for the DXT dataset. The memory-resident DXT Allocation Table (XAT) is also set up at this time.

Presence of the permanent DXT dataset during a Deadstart or Restart is determined immediately after the DSC recovery. If the DXT is permanent and the *DXT directive indicates an adjustment to the DXT is required, the adjustment is made after the DXT recovery is completed if the main DSC entry for the DXT has no error flags set. For DXT expansion, the following conditions must also be satisfied:

- Sufficient disk space is available to satisfy the overflow requirement (the OVF= parameter on the *DXT directive or I@DXTOVF).
- Enough contiguous disk space is available to satisfy the contiguous allocation requirement (the CAI= parameter on the *DXT directive or I@DXTCAI).

If any of these conditions is not satisfied, the operator is informed through the operator interface mechanism.

Any DSC error flag set in the main DSC entry for the DXT dataset causes one or more informative messages describing the error flags encountered, followed by a request that a Startup-Install be performed.

If, during the DSC recovery, it was determined that one or more DSC entries had associated DXT entries, DXT validation occurs before any DXT size adjustment. If DXT validation is not required, then any requested DXT size adjustment is accommodated and followed immediately with the construction of the upper memory DXT entry allocation bit map (XAT). The XAT size is directly proportional to the DXT dataset size. The DXT validation worst case causes the DSC to be scanned twice followed with a final complete read of the DXT dataset.

Validation begins with the DSC being read sequentially. As each main DSC entry is encountered, any DXT entries chained to it are reserved in the DXT allocation bit map (XAT). If a DXT entry is found to be already reserved in the XAT, the equivalent bit in a pseudo bit map is set rather than re-setting the real bit and the DSC main entry flagged as having a DXT chain error. As each DXT entry is read, the In-use flag (DXUSE), the ordinal sequence (DXORD), and DSC main entry pointer (DXTFPE) are verified. Any discrepency causes the DXT error flag (DCDXE) to be set in the main DSC entry. When the end of the DXT chain is reached, the last DXT entry pointer (DCLDX) in the main DSC entry is verified. Here again, an error causes DCDXE to be set. The DCDXE flag in the main DSC entry is similar to other DSC error flags. After the DSC read is completed, a second DSC read is performed if any bits were set in the pseudo XAT. During this second DSC read, the pseudo XAT is used to capture all other crossed allocations. Whenever an error is encountered while validating a DXT chain, the individual DXT chain validation is halted and continues onto the next DSC main entry in order to avoid looping. When the second DSC read completes, any orphaned DXT entries are deactivated. Orphaned entries result from the Startup DXT validation or from a system interruption occurring while DXT entries are being deallocated through a user's delete command. To remove these orphaned entries and to ensure that the whole DXT dataset can be read from disk without I/O errors, Startup reads the DXT dataset sequentially. During this read operation each DXT entry not allocated in the XAT has its In-use flag (DXUSE) cleared on disk. This procedure ensures that all unreserved DXT entries are marked. Thus, whenever a new DXT entry is requested by the running system the DXUSE flag is examined. If the DXUSE flag is already set, the system is halted. This technique should protect the system against XAT disagreement with the actual disk allocation.

DXT access and control

During Startup, the Dataset Catalog Extension Table (DXT) dataset is either created or recovered, and Startup makes an entry indicating the DXT dataset is held in unique access mode into the Active Permanent Dataset Table (PDS). In order to facilitate DXT I/O, Startup also completes the initialization of the STP-resident Dataset Name Table (DNT) and Dataset Allocation Table (DAT) for the DXT dataset. The Dataset

Parameter Table (DSP) and associated buffer space for the DXT are created during the initialization of task PDM.

A job cannot access the DXT dataset because the PDS indicates it already has been accessed uniquely. However, System Directory (SDR) utilities, such as PDSDUMP and AUDIT, must be able to access the DXT. This capability is provided with the PDM function PMFCPX (41 octal). This function is similar to the PDM function PMFCPG (40 octal) except that PMFCPX reads DXT pages rather than DSC pages. The same security offered to the PDM function PMFCPG is also offered to PMFCPX.

5.6.4 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 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 an empty $SDR is created.

The $SDR dataset consists 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 Restart or 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 Install is performed, the System Directory is not recovered, and a user job (JSYSDIR) must be run to create the initial System Directory entries.

5.6.5 ROLLED JOB INDEX DATASET ($ROLL)

The operating system maintains a special permanent dataset so Startup can determine which jobs were in execution before 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 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. If recovery is requested but $ROLL cannot be accessed, recovery of rolled jobs is disabled with a message to the System Log. 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. Job Execution Table (JXT) limitations assume that $ROLL will never exceed one allocation unit. This number is compared with the Allocation Index (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 the job assigned to the JXT entry and to locate the roll image if the job is rolled out. The index entry also contains a flag indicating whether a job has performed some function that invalidates 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.

- An Irrecoverable 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. Deletes, adjusts, or modifies a permanent dataset. Since these functions change the DSC in a manner that could cause the job to fail if repeated, the roll image is unreliable.

  2. Randomly writes to any dataset. The system circular I/O (CIO) routines recognize a random write to a dataset and declare the job irrecoverable, since the difference in data may change job results if the job is restarted at an earlier point.

  3. Writes following a 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 irrecoverable.

  4. Releases 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 irrecoverable. Release of a permanent dataset does not affect disk allocation and therefore does not affect recoverability.

  Every job rendered irrecoverable by any of the above becomes recoverable again as soon as it is 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 the User Exchange Processor (EXP) and the Job Scheduler (JSH) during system operation. At job initiation, JSH sets up a corresponding index entry reflecting that the job was never rolled out and is, therefore, irrecoverable. 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. Rewriting of the index always occurs before EXP completes processing the function.

5.7 TABLES USED BY STARTUP

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

AUT	Active User Table
CNT	Configuration Table
DAT	Device Allocation Table
DNT	Dataset Name Table
DRT	Device Reservation Table
DSC	Dataset Catalog
DSP	Dataset Parameter Area
DVL	Device Label
DXT	Dataset Catalog Extension
EFT	Engineering Flaw Table
EQT	Equipment Table
GRT	Generic Resource Table
JTA	Job Table Area
JXT	Job Execution Table
ODT	Overlay Directory Table
PDI	Permanent Dataset Information Table
QDT	Queued Dataset Table
RJI	Rolled Job Index Table
SDT	System Dataset Table
TDT	Tape Descriptor Table

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

5.7.1 ACTIVE USER TABLE (AUT)

Startup creates and initializes an Active User Table entry for any interactive job it recovers.

5.7.2 CONFIGURATION TABLE (CNT)

The CNT informs the operating system of the status of online tape devices. The table can be changed during startup by the parameter file or by operator commands.

5.7.3 DATASET ALLOCATION TABLE (DAT)

The Startup task creates a DAT for the Dataset Catalog (DSC) dataset.

5.7.4 DATASET NAME TABLE (DNT)

The Startup task initializes the DNT for the Dataset Catalog (DSC). DNTs are also used for I/O on datasets such as $ROLL.

5.7.5 DEVICE RESERVATION TABLE (DRT)

The DRT, as initially assembled and updated through parameter file options, lists disk flaws that Startup uses during an install 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, and set up the area reserved for system dumps.

5.7.6 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 an idle down or system interruption.

Startup sets the DRT bits for the Dataset Catalog (DSC), the system dump area, the system overlay dataset, or datasets encountered in the DSC, and checks the DRT to see if the bit is already set. If a conflict is found, a special flag is set and a note is made of each such device and track number. Following completion of the normal 1-pass dataset recovery pass, a second pass is made through the DSC to identify datasets with conflicts, with System Log messages.

5.7.7 DATASET PARAMETER AREA (DSP)

Startup and Permanent Dataset Manager (PDM) use their own internal separate DSPs for manipulating the Dataset Catalog (DSC). There is no central DSP in this regard.

5.7.8 DEVICE LABEL (DVL)

Install writes a device label on the first usable track of each mass storage or SSD device. The label contains the device name and flaw information for each device. The master device label also contains the Dataset Allocation Table (DAT) for the Dataset Catalog (DSC), a pointer to the first track of the reserved system dump area, and a pointer to the system overlay dataset. A special flag in the label identifies the master device.

The Device Label (DVL) usually resides on the first track of its disk storage unit (DSU) because I@DVLRES, an installation parameter, 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 can 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 preallocated for system dumps and the system overlay dataset. These areas are also reserved in the DRT.

5.7.9 DATASET CATALOG EXTENSION (DXT)

The *DXT directive in the Startup Parameter File controls DXT creation at Install and Deadstart. The size may be changed through the *DXT directive during any startup. DXT recovery and validation is described in section 5.6, Input to Startup. DXT creation is described in section 5.2, Deadstart. Use of the *DXT directive is described in the COS Operational Procedures Reference Manual, publication SM-0043.

5.7.10 ENGINEERING FLAW TABLE (EFT)

Startup uses the Engineering Flaw Table (EFT) as a source of flaws to be entered into the DRT. The EFT exists in sector 17 (decimal) of one of the first 10 tracks on the device.

5.7.11 EQUIPMENT TABLE (EQT)

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

5.7.12 GENERIC RESOURCE TABLE (GRT)

Startup uses the GRT to preset the JOB Statement Parameter Table, and initialize available resource counts in the GRT from information in the EQT and TDT. Allocated resource counts for rolled jobs are moved from the JTA to the GRT during rolled job recovery.

5.7.13 JOB TABLE AREA (JTA)

When recovering rolled jobs, Startup searches the JTA for local datasets. Startup verifies and allocates the Allocation Indexes (AIs) associated with these local datasets.

5.7.14 JOB EXECUTION TABLE (JXT)

Startup gets the Job Execution Table image for a job from the roll file and rebuilds the entry in the JXT.

5.7.15 OVERLAY DIRECTORY TABLE (ODT)

The ODT defines what overlays exist in the system and is used by Startup when it is searching for overlays and moving them to their resident locations. Errors occur if overlays are found which are not in the ODT or the ODT contains overlays which are not located by Startup.

5.7.16 PERMANENT DATASET INFORMATION TABLE (PDI)

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

5.7.17 QUEUED DATASET TABLE (QDT)

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

5.7.18 ROLLED JOB INDEX TABLE (RJI)

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

5.7.19 SYSTEM DATASET TABLE (SDT)

Install and Deadstart initialize the SDT as having all entries in the available queue. Restart uses the Dataset Catalog (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.

5.7.20 TAPE DEVICE TABLE (TDT)

The Tape Queue Manager (TQM) uses the Tape Device Table (TDT) to control online tape devices. The TDT can be changed by Startup as the result of changes made in the Tape Configuration Table (CNT).

5.8 STARTUP SUBROUTINES

The COS initialization 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. The main routines are: Z, RRJ, and SDRREC.

5.8.1 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 through a return jump to entry point Z. Z resides at the upper 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 Message Processor 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 before 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; Startup can also place one or more copies of the image of CSP on mass storage (installation defined). In the latter case, pointers are adjusted to allow the space otherwise occupied by CSP to be used 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 Dataset Catalog (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, Job Execution Table (JXT) entries can 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 machine clock periods and sets the real-time clock to this value.

The SCP task can 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 computer on which COS is executing. The actual memory size is defined by an installation parameter (I@MEM) or through a Startup parameter file statement (*MEMSIZ).

5.8.2 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 Job Scheduler (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, publication SM-0045) 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 is 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 are checked. If errors occur on the access or read, or if entry 0 does not validate correctly, RRJ issues error messages and reinitializes $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 is selected and if RRJ is able to successfully access and read $ROLL, RRJ attempts to recover jobs. 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 certain key tables (for example, DNT or PDD) change in size, RRJ detects the change during validation of entry 0. This causes a fatal Startup error.

If no errors occur during $ROLL validation, RRJ attempts to recover jobs. Messages are issued to the System Log when a job is not recovered and when recovery has been successful. 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 becomes 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 if the event is satisfied or if the recall should be reissued. For example, any outstanding ACQUIRE requests may have to be reissued. Event wait can be reset if the job has an outstanding DISPOSE,WAIT request.

5.8.3 SDRREC SUBROUTINE

System Directory (SDR) Recovery (SDRREC) is executed as a closed subroutine that is called by Startup 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 (DQM) to allocate disk space for the dataset. Then a request is made to the Permanent Dataset Manager (PDM) 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 Permanent Dataset Definition Table (PDD), in the logical record, and the Dataset Name Table (DNT) 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, indicating 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.

DISK QUEUE MANAGER (DQM) 6

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. In a Cray Computer System without an I/O Subsystem, EXEC readies DQM when an I/O request finishes or when a sector transfer completes for a dataset in recall. In an I/O Subsystem, EXEC readies DQM when an I/O request finishes.

6.1 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 Dataset Name Table (DNT) address in its INPUT+0 register and exits with an EXEC request to ready DQM.

In the following, JXO is the Job Execution Table (JXT) offset, calculated by subtracting B@JXT, the base address of the JXT, from the individual JXT entry address. JXO is 0 if the call is not job related. DNT is the DNT address; this address is relative to the JTA if the call is job related. RCL is a flag that is set if DQM sends an intermediate reply to recall a task or job when DNRCL=1 and some I/O has been completed.

6.1.1 ALLOCATION

Figure 6-1 illustrates the input and output values used by allocation.

The allocator references the following DNT fields: DNLDV, DNAS, DNSE.

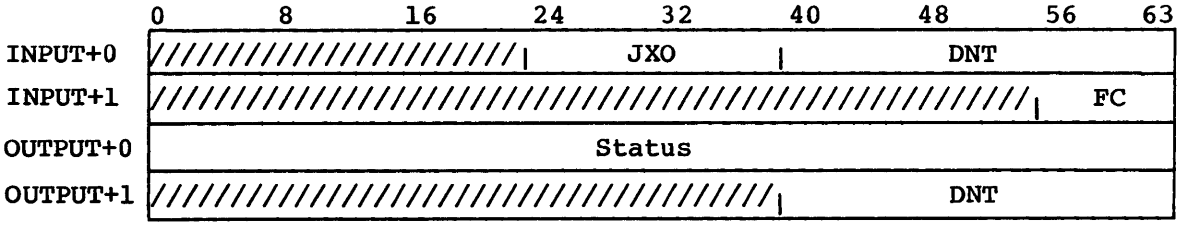


Figure 6-1. DQM allocation interface

Field	Word	Bits	Description
JXO	INPUT+0	24-39	JXT offset
DNT	INPUT+0	40-63	DNT address
FC	INPUT+1	56-63	Function code (1_8)
Status	OUTPUT+0	0-63	Return status (see section 6.1.4)
DNT	OUTPUT+1	40-63	DNT address

6.1.2 DEALLOCATION

Figure 6-2 illustrates the input and output values used by deallocation.

The deallocator references the following DNT fields: DNDAT and DNDCZ (and DNPDS, for permanent datasets only). Partial deallocation is detected by comparing DNDCZ and DADSE.

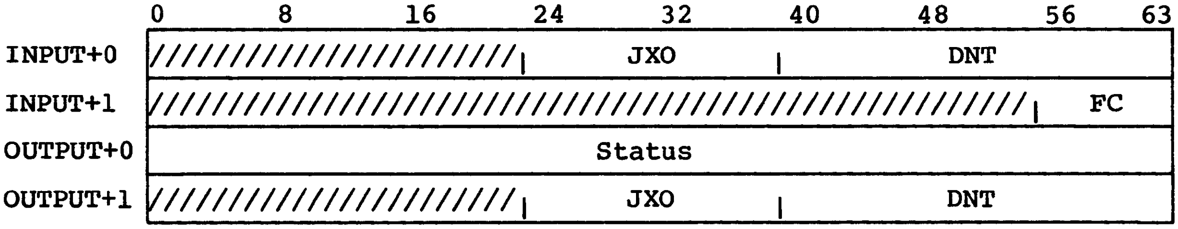


Figure 6-2. DQM deallocation interface

Field	Word	Bits	Description
JXO	INPUT+0	24-39	JXT offset
DNT	INPUT+0	40-63	DNT address
FC	INPUT+1	56-63	Function code (2_8)
Status	OUTPUT+0	0-63	Return status (see section 6.1.4)
JXO	OUTPUT+1	24-39	JXT offset
DNT	OUTPUT+1	40-63	DNT address

6.1.3 QUEUE I/O

Figure 6-3 illustrates the input and output values used by Queue I/O.

The transfer request processor references the following DNT fields: DNDAT, DNLDV, DNAS, DNSZ, DNNBK, DNSBK, DNDSP, DNP, DNPBS, DNIOB, DNLM, DNBUF, DNJTF, and DNEND.

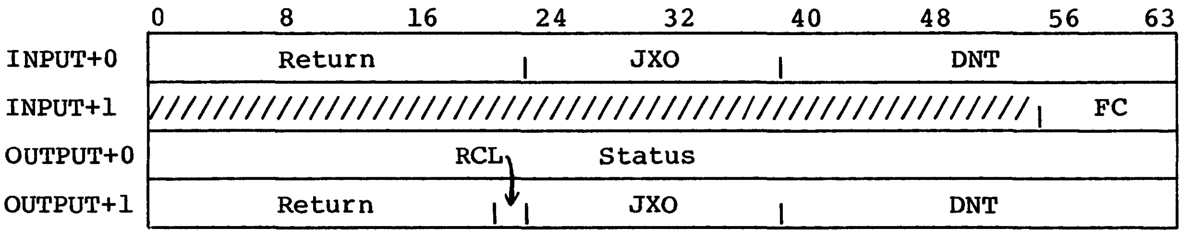


Figure 6-3. DQM Queue I/O interface

Field	Word	Bits	Description
Return	INPUT+0	0-23	The return field is normally used to save a return address for CIO. If the queue I/O interface is used directly without going through CIO, the return field can contain any information that needs to be preserved.
JXO	INPUT+0	24-39	JXT offset

Field	Word	Bits	Description
DNT	INPUT+0	40-63	DNT address
FC	INPUT+1	56-63	Function code (0_8)
Status	OUTPUT+0	0-63	Return status (see section 6.1.4)
Return	OUTPUT+1	0-23	Same as INPUT+0 field
RCL	OUTPUT+1	24	Recall flag; see introduction to this subsection.
JXO	OUTPUT+1	25-39	Same as INPUT+0 field
DNT	OUTPUT+1	40-63	Same as INPUT+0 field

6.1.4 RETURN STATUS

The status words contain one of the following values:

Mnemonic Code	Octal Value	Meaning
ERUNK	1	Unknown request
ERNMT	2	DAT space exhausted
ERNMS	3	Disk space exhausted
ERNLD	4	Logical device not in system
EREOI	6	Attempt to read beyond end of data
ERUHE	7	Unrecovered hardware error
ERUDE	10	Uncorrected data error
ERIDP	11	Invalid DSP
ERXLM	56	Dataset size limit exceeded

6.2 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
DNT	Dataset Name Table
DRT	Device Reservation Table
DSP	Dataset Parameter Table
EQT	Equipment Table
GRT	Generic Resource Table
JTA	Job Table Area
JXT	Job Execution Table
RQT	Request Table
SCT	Subsystem Control Table

Figure 6-4 illustrates the linkages of tables used by DQM.

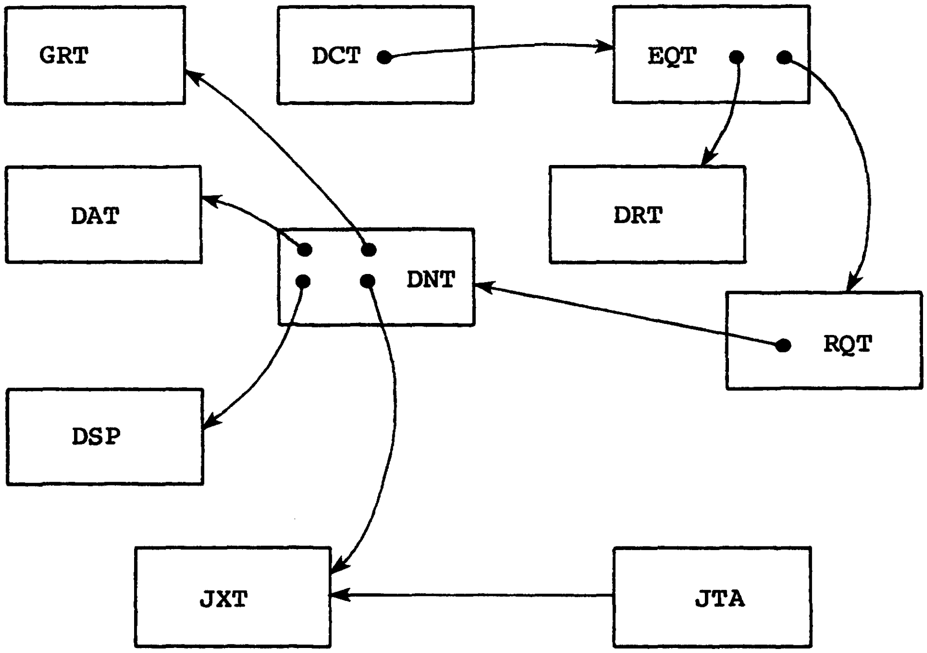


Figure 6-4. DQM table linkages

6.2.1 DATASET ALLOCATION TABLE (DAT)

The DAT resides in the STP common table area (for system datasets) or the Job Table Area (JTA) (for user datasets) and associates datasets with physical space on one or more devices. The DAT consists of a header and a body. The header is used primarily for managing the main body of the table 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 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 6-5 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 dataset's DAT entry. Page headers logically link pages comprising the entry.

6.2.2 DEVICE CHANNEL TABLE (DCT)

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

6.2.3 DATASET NAME TABLE (DNT)

DQM uses the DNT to process a disk request. DQM does the following:

- Gets the DSP address to determine the logical request (if provided)
- Gets the DAT address to determine the physical address
- Stores the logical request in the DNT
- Stores the processing direction in the DNT

6.2.4 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.

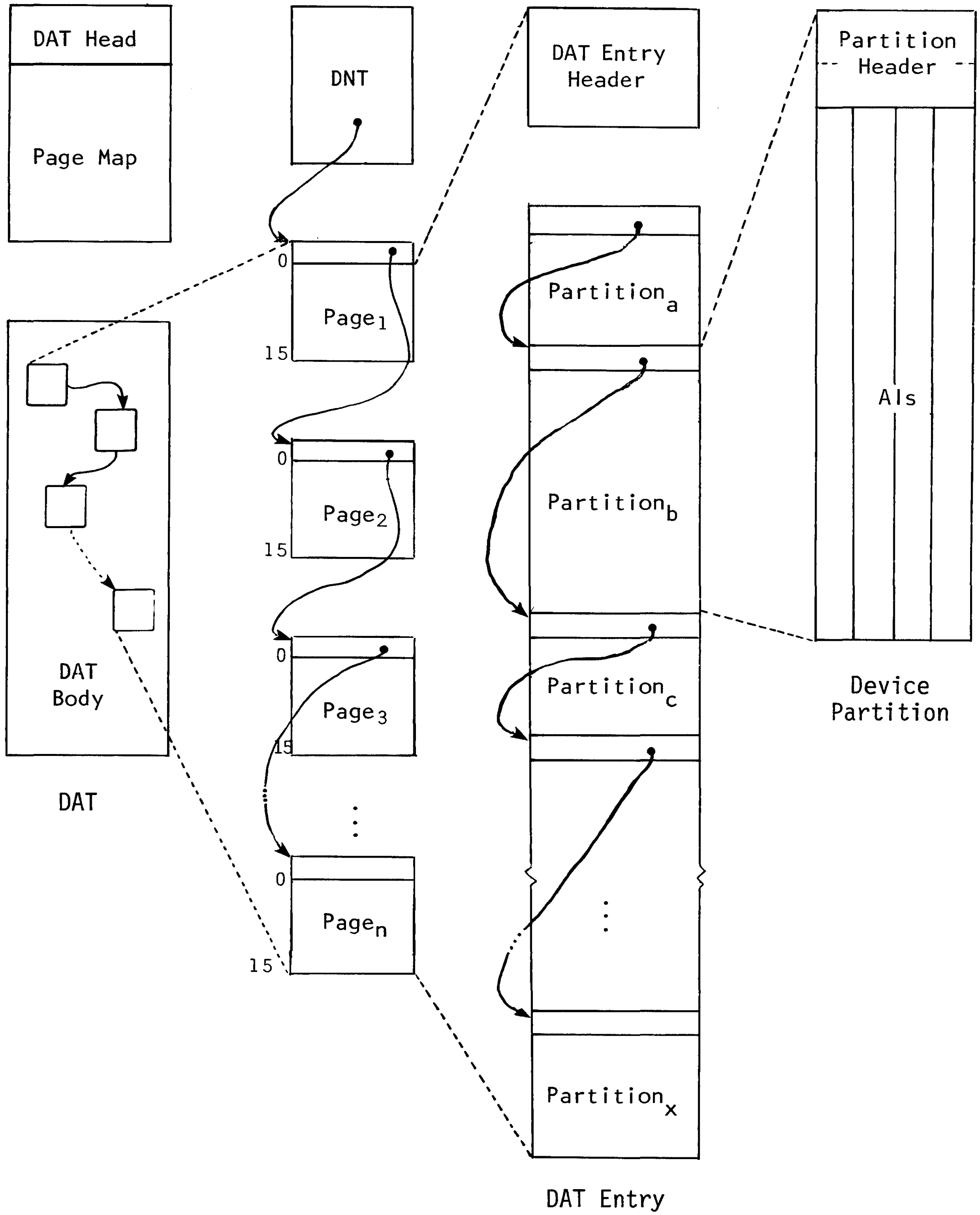


Figure 6-5. DAT structure

6.2.5 DATASET PARAMETER TABLE (DSP)

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

6.2.6 EQUIPMENT TABLE (EQT)

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

6.2.7 GENERIC RESOURCE TABLE (GRT)

The GRT contains information about all defined generic resources. If a mass storage device is declared as controlled, and made a member of a generic resource, DQM ensures that job limit declarations are enforced.

6.2.8 JOB TABLE AREA (JTA)

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

6.2.9 JOB EXECUTION TABLE (JXT)

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

6.2.10 REQUEST TABLE (RQT)

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

6.2.11 SUBSYSTEM CONTROL TABLE (SCT)

The SCT serves as an interface between DQM and the I/O Subsystem driver in EXEC. This table holds status information and request parameters.

6.3 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 before writing a dataset or can 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 that overflow their preallocated sizes.

A dataset explicitly assigned a logical device starts on that device if space is available but can overflow to other devices. Devices can be designated as private, which means that their space is allocated only by explicit assignment.

A device can also be declared as controlled, meaning user jobs must declare a limit on the amount of space used by the job on the device. DQM enforces the declared limit and allows device overflow at the declared limit (if the user job indicates that overflow is allowed).

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

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. 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.

COS supports one allocation style, one track per allocation index.

Each time DQM assigns an allocation unit to a dataset, it places the allocation index (AI) for the device in the dataset's Dataset Allocation Table (DAT) entry and sets the corresponding bit in the Device Reservation Table (DRT). Deallocation causes the DAT and the DRT to be cleared and the DAT to be released. Additionally, when the device is a member of a generic resource, DQM maintains global allocation counts in the GRT during allocation and deallocation.

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 if space is available, that is, to the most recent device partition in the DAT entry.

6.4 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 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 mainframe I/O channels and control either DD-19 or DD-29 Disk Storage Units (DSUs). The DCU-4 controllers are integral to the Buffer I/O Processor and Disk I/O Processors in the I/O Subsystem and control DD-29 Disk Storage Units. The DCU-2 and DCU-3 controllers are described in the Mass Storage Subsystem Hardware Reference Manual, CRI publication HR-0630. The DCU-4 controller is described in the I/O Subsystem Reference Manual, CRI publication HR-0030.

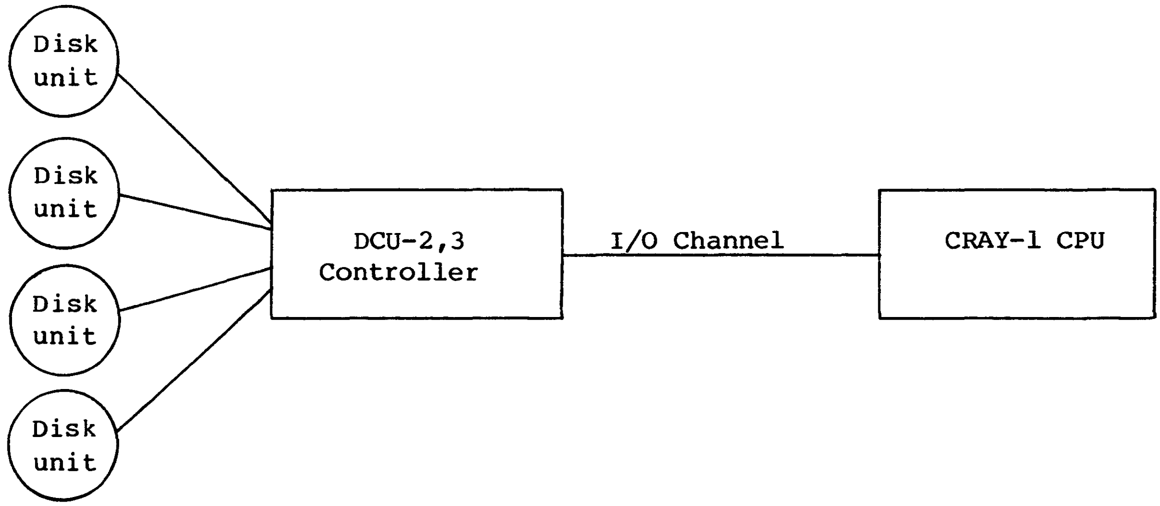


Figure 6-6. DCU-2, DCU-3 controller configuration

6.4.1 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 positioned simultaneously or several units to be positioned simultaneously with transfer from a single unit. The channel and controller configuration is illustrated in figure 6-6.

6.4.2 DCU-4 CONTROLLER MANAGEMENT

The DCU-4 controllers reside in the Buffer I/O Processor (BIOP) and Disk I/O Processor (DIOP) in the I/O Subsystem.

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. (See figure 6-7.)

6.4.3 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

DQM logs disk error conditions and calls the Disk Error Correction (DEC) task whenever a potentially correctable error occurs.

6.5 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 Dataset

Parameter Table (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.

6.6 I/O REQUEST FLOW IN DQM

The processing flow is as follows:

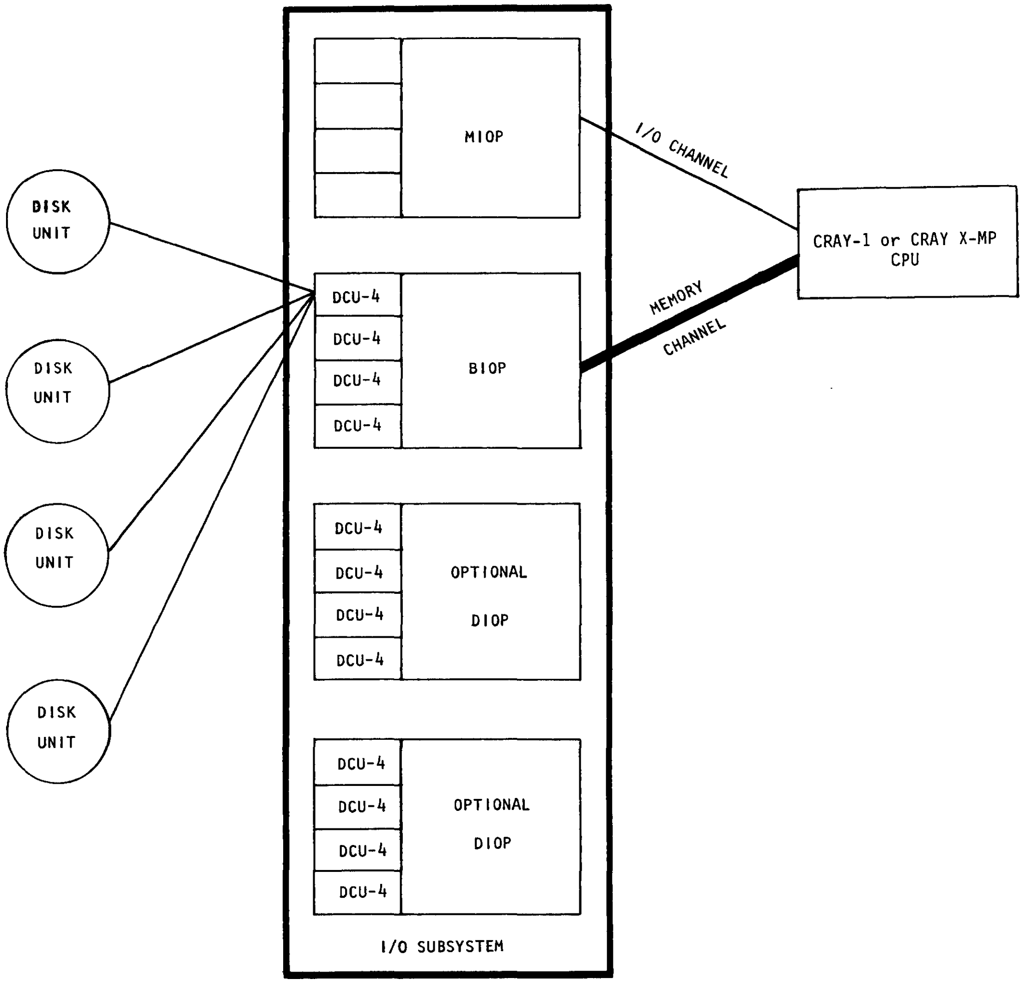


Figure 6-7. DCU-4 controller configuration

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

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

3. The Dataset Parameter Table (DSP) request is mapped into a logical request in the Dataset Name Table (DNT). (If no DSP exists, the caller must have already set the logical request up.)

4. The DQM enqueue subroutine transforms the logical request into a physical request and enqueues it.

5. 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.

6.7 DISK HARDWARE ERROR LOGGING

DQM logs all disk errors by placing a binary copy of the Equipment Table (EQT) in the System Log at the time of the error.

Disk errors are selected by the EXTRACT utility when TYPE=HARDWARE, SUBTYPE=DISK is specified on the EXTRACT control statement. (For more information on EXTRACT, see the COS Operational Aids Reference Manual, publication SM-0044.)

6.8 UNCORRECTED DATA ERROR RECOVERY

When an uncorrected data error is encountered, the bad sector is written to the I/O buffer, but the buffer pointer DPIN is not advanced to reveal the transfer. DQM returns an Uncorrected Data Error status (ERUDE). In order to see the bad sector, the caller must set a flag (DPABS) in the Dataset Parameter Table (DSP) and issue another F$RDC request, after having first cleared DPEDE. DQM will then advance DPIN to reveal the bad sector and clear DPABS. DQM uses two fields of the Dataset Name Table (DNT) to verify that bad data was encountered (DNBDF) and to determine whether the first sector requested is the bad sector (DNBBN). If the Accept Bad Sector flag is set (DPABS) but the first sector requested is not bad, DQM returns an error status of Invalid DSP (ERIDP). If the first sector requested is bad but DPABS is not set, DQM returns an error status of Uncorrected Data Error (ERUDE).

6.9 MAINTENANCE TEST FEATURE

DQM's maintenance test feature validates disk error processing and logging. It is invoked by loading the maintenance test field of the Dataset Parameter Table (DSP) (DPMTF) with a test code and data. The field is loaded on a user job level.

When the feature is enabled, DQM copies the DPMTF field to the EQT to be read by the disk driver or to the IOP disk packet processing code. This data is then used to simulate a selected error condition.

NOTE

This feature is normally disabled. It is intended for use only when performing software maintenance.

The maintenance field is formatted as follows:

```
0       8     15
┌────────────────┐
│ TFC|TD4|  TD8  │
└────────────────┘
```

TFC Test function code (in octal) as follows:

0 No maintenance test
1 Recoverable error test (TD8 contains the extended error status code)
2 Irrecoverable error test (TD8 contains the extended error status code)
3 Timeout error test
4 IOP disk error test (TD8 contains the disk packet status code)
5 Write with zero check word
6 Simulated memory error (TD4 contains the exchange package read mode field, TD8 the exchange package syndrome bits)
7 Undefined test

TD4 Content varies; see the TFC description above.

TD8 Content varies; see the TFC description above.

STATION CALL PROCESSOR (SCP) 7

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
- Multiplexing of streams for each logical station
- Multiplexing of logical stations on the same hardware channel

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, CRI publication SM-0042.

7.1 SYSTEM TABLES USED BY SCP

SCP uses the following system tables:

AUT	Active User Table
IBT	Interactive Buffer Table
LCT	Link Configuration Table
LIT	Link Interface Table
LXT	Link Extension Table
PDD	Permanent Dataset Definition Table
SDT	System Dataset Table
SST	Stager (STG) Stream Table

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

7.1.1 ACTIVE USER TABLE (AUT)

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

7.1.2 INTERACTIVE BUFFER TABLE (IBT)

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

7.1.3 LINK CONFIGURATION TABLE (LCT)

The installation builds an entry in the Link Configuration Table (LCT) for each CPU channel used for front-end communications. The LCT determines the channel characteristics.

7.1.4 LINK INTERFACE TABLE (LIT)

SCP assigns a Link Interface Table (LIT) entry at startup to each CPU channel used for front-end communications. This table contains channel buffer control information and EXEC working storage.

7.1.5 LINK EXTENSION TABLE (LXT)

EXEC assigns a Link Extension Table (LXT) entry for a front-end computer system at logon time and releases the entry at logoff. This table contains SCP working storage and is used for communication between EXEC and SCP concerning a logical front-end ID. Each LXT entry contains 16 Stager Stream Table (SST) entries.

7.1.6 PERMANENT DATASET DEFINITION (PDD)

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

7.1.7 SYSTEM DATASET TABLE (SDT)

SCP has prime responsibility for the System Dataset Table (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.

The queue control word (header) and word 36 (W@SDLK) of each SDT entry create a circular linked list. FLP contains the address of the next SDT entry or points to the queue header. BLP contains the address of the previous SDT entry or points back to the queue header. Starting at any entry, the circular linked list can be searched backward or forward, stopping at any entry.

When a queue is empty, the pointer points to itself minus 36 (W@SDLK) (figure 7-1). When a queue is not empty, the first and last entries in the queue point to the header word minus 36 (W@SDLK) (figure 7-2).

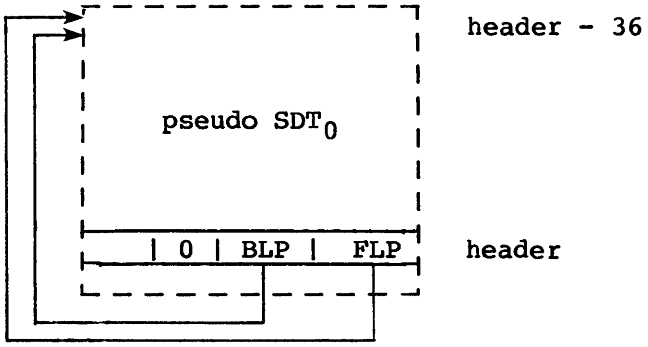


Figure 7-1. Header when queue is empty

7.1.8 STAGER STREAM TABLE (SST)

The Stager Stream Table (SST) contains the information for controlling the input or output of a stream and is the communication link between SCP and the Dataset Stager (STG) task.

7.2 PROCESSING FLOW FOR SCP

Upon receipt of each message, SCP checks the input link control package (LCP) in the Link Configuration Table Extension (LXT) LCP buffer for

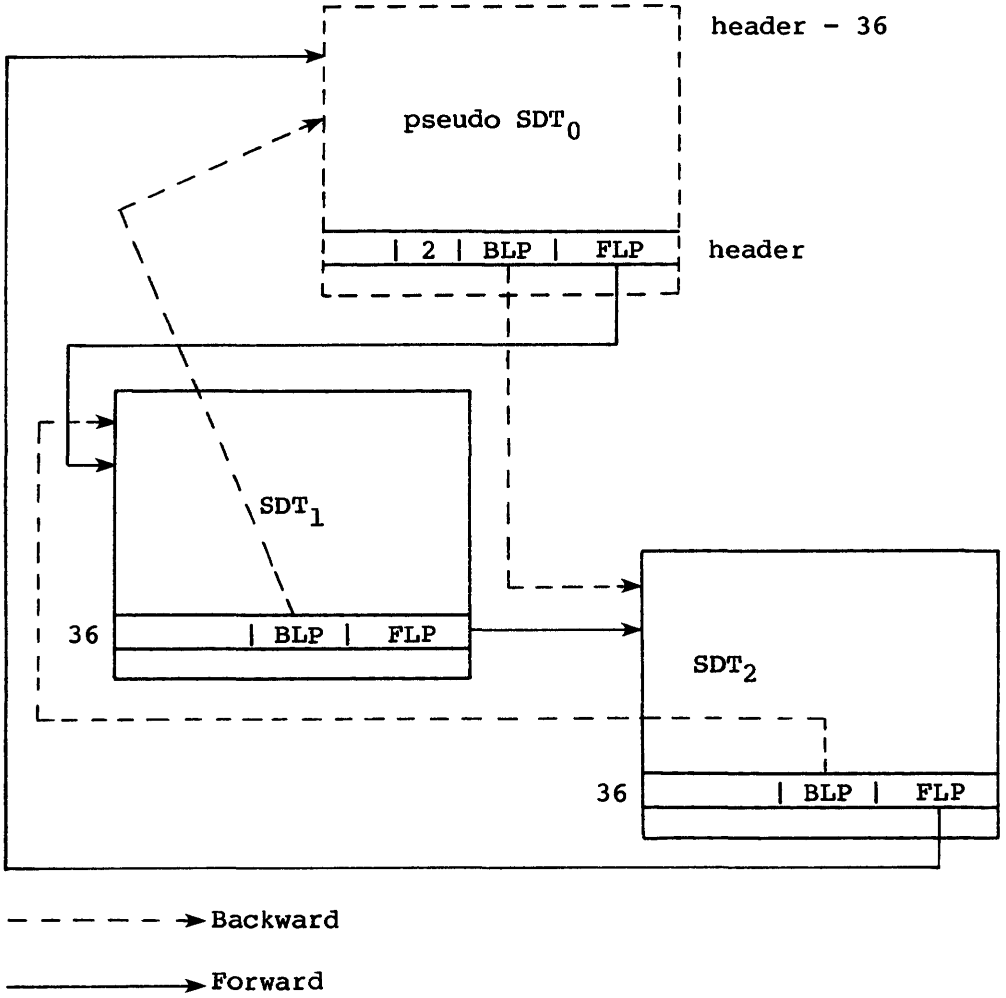


Figure 7-2. Queue with two entries

illegal message code and illegal parameters. Any error causes an immediate Message Error response.

SCP then processes the input LCP message code as follows:

1. Log on causes SCP to save log on parameters and to initialize the buffer pool. If log on is not expected, the log off procedure is executed first, followed by log on processing.

2. Log off causes SCP to deallocate all incoming datasets and makes the associated System Dataset Table (SDT) entries available. All outgoing dataset SDT entries are returned to the output queue.

3. Control is a no-op.

4. Dataset Header causes an SDT entry to be assigned and the header parameters to be saved in the SDT. A start request is then posted in the Stager Stream Table (SST) for the STG task.

5. Dataset Segment causes SCP to verify the stream is ready and in an RCV state. A switch of buffers is made. SCP trades the input buffer for the empty buffer pointed to by the SST for this stream. The STG task is then activated with a process buffer code.

6. Job Status, System Status, Link Status, Mass Storage Status, Dataset Status, Operator Debug, Logfile Information, Logfile Entry Request, Configure, Tape Configure, Tape Job Status, and Interactive Request must be replied to before a second request of this class is sent. SCP returns an error message if there is an outstanding message of this class waiting to be processed. The LCP and segment are verified and the function is processed.

7. Diagnostic Echo Request causes the appropriate Echo Reply to be built and transmitted immediately. The input segment is used for the output segment.

8. Message Error causes immediate retransmission of the prior message.

9. Station Message Reply causes SCP to verify the LCP and segment size. A memory pool buffer is acquired and the message is moved to the buffer. If deadstart is finished the reply is logged in the system logfile. The reply is forwarded to the appropriate task.

10. Dataset Transfer Reply causes the status field to be examined. If the status is NO, the request is deleted and the originating job aborted. If the status is YES, the request is removed from the request queue. The SDT is placed on the available queue. If the status is Postpone, the request is put on the bottom of the request queue and reissued after at least a delay defined by I@DTRDLY.

SCP then processes the input stream control bytes. Invalid streams are master cleared and active transfers on these streams are terminated. Input stream (front end is the sender) types are as follows:

1. Idle - Normal state if front-end station has no activity for this stream. No System Dataset Table (SDT) entry is associated. The COS response is Idle.

2. Request To Send - Front-end station is ready to stage a dataset. SCP immediately responds with Receiving unless there are no SDTs. SCP responds with Postpone if no SDT entry is available.

3. Sending - Front-end station is prepared to send Dataset Header or Dataset Segment. SCP immediately responds with Receiving if STG is finished processing the last segment, or with Suspend if STG is not yet finished. If the Dataset Stager (STG) task reports that the dataset is already saved on Cray mass storage, SCP cancels the stream and stream termination begins.

4. End Data - Front-end station is waiting for Dataset Saved response from COS. SCP checks to see if the STG task has been notified that an End was received for this stream. If not, SCP activates STG with an End function code. COS responds with Saving. STG reports the dataset is saved. If STG reports the dataset is already on Cray mass storage, SCP cancels the stream. If STG detects any error, the stream is canceled and stream termination begins.

5. Cancel - Front-end station requests that the stream's current activity be dropped. SCP requests that STG deallocate the input dataset and make the SDT entry available. The cancel is logged in the System Log. The COS response is Idle.

6. Postpone - Front-end station requests that the stream's current activity be dropped. SCP sets the termination code in the Stager Stream Table (SST) for the stream. This causes STG to terminate the transfer. (If this dataset originated from a dataset transfer request, the originating job is not aborted.) The COS response is Idle.

7. Master Clear - Front-end station encountered an invalid COS response. SCP processes master clear in the same manner as postpone. COS response is Idle.

SCP processes the output stream control bytes. Output stream (COS is the sender) types are as follows:

1. Idle - Normal front-end response if COS state is Idle, Postpone, Cancel, or Master Clear. COS responds by issuing either Idle or Request To Send if an output dataset is queued. In the latter case SCP removes the System Dataset Table (SDT) entry from the output queue and assigns it to this stream. It is also queued onto the sending queue for the Link Configuration Table Extension (LXT).

2. Preparing To Receive - Front-end station response indicating preparing to accept Dataset Header. The header is not eligible to be sent.

3. Receive - Front-end station response indicating readiness to accept Dataset Header or Dataset Segment. The STG task response is checked. If no buffer is ready, SND is the next Stream Control Byte (SCB). If a buffer is ready, the Buffer Ready flag is set in the Stager Stream Table (SST), and a buffer is sent if not preempted by another message type.

4. Suspend - Front-end station is ready to temporarily stop receiving segments for this stream. SCP maintains the response as Sending. If necessary a segment is prepared for sending.

5. Dataset Saved - Front-end station indicates dataset is saved and stream can be released. SCP sets the Stager Stream Table (SST) code requesting stream End termination. The saved response is logged in the System Logfile. The response is Idle.

6. Postpone - Front-end station is ready to postpone receipt of this dataset. If the stream is active SCP places the postpone request in the SST termination field. The response is Idle.

7. Cancel - Front-end station is ready to cancel receipt of this dataset. The abort code is set in the SST termination field. The abort is logged in the System Log. The response is Idle.

8. Master Clear - Front-end station encountered an invalid COS response. For a dataset in the process of being staged out, SCP sets a postpone request in the Stager Stream Table (SST) termination field. The response is Idle. (Note this is the same as Postpone.)

The output link control package (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 nonstaging 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 COS 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 (through EXEC) to complete the output transmission and await input.

EXCHANGE PROCESSOR (EXP) 8

The Exchange Processor (EXP) task processes all user system action requests and user error exits. The Exchange Processor also handles requests from the Job Scheduler (JSH) for initiating or aborting 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 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 nonrerunnable. When the user knows that changes to permanent datasets will not affect the correct execution of the job if it is rerun, the user can override EXP and declare the job rerunnable or can 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 word TCEP of the Task Control Block requesting execution of the Exchange Processor and indicating whether the exchange is normal or in error.

JSH requests EXP by setting another flag in the same word (TCEP) in the TCB. EXEC readies EXP and exchanges to it, instead of exchanging to the user, whenever the TCEP word is nonzero for the currently connected job.

After EXP processes a request, it clears TCEP to allow EXEC to return to the user job.

If EXP cannot finish a request immediately, it suspends itself without clearing TCEP. EXEC then returns control to EXP, rather than the user, whenever the user task is assigned to the CPU.

In general, EXP calls JSH to suspend the user task before suspending itself when it must wait for completion of a request, such as an I/O request to another task. This allows other user tasks to be assigned the CPU.

The entity for which EXP executes is a user task. A typical job step will not be multitasked and for those job steps the terms job and task are often used interchangeably. In reality, however, EXP makes very few requests on a job basis. Physical I/O queue manager requests are made on a task basis, JSH requests are made on a task basis, and so on.

The coordination of multiple tasks within a job requires some special handling. In certain situations (deleting a task for instance) EXP has to know that its actions vis-a-vis other tasks in the same job are uninterruptible. For this purpose, it uses the J$SINGLE request to JSH to force all other tasks in the same job to be disconnected until a corresponding end-single-threading call is made. Abort/advance situations use J$SINGLE to enforce single threading while the user's reprieve processing code is executing.

Abort/advance situations require that EXP allow only one task to survive to the next job step. After any reprieve processing, EXP deletes all tasks except the one for which it is executing.

8.1 SYSTEM ACTION REQUESTS

Exit from a user program occurs when the user program executes an exchange instruction (004). The user issues a system action request on a program exit by setting S0 to the desired function code and possibly setting S1 and/or 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.

When EXP is readied, it detects the user request because the TCEPN field for the currently executing task is set. 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, JCDSP, JCMFL, JCLPP, JCILEV, JCPLEV, JCCLEV, and JCREVN. Several other fields are also verified as reasonable values.

The mnemonic values used to assemble user codes are defined in common deck COMEXPFC, which is called into $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 Irrecoverability 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:

```
0        8       16      24      32      40      48      56   63
|    m       m   |  /  |    d       d   |  /  |     y     y     |
```

Mnemonic Code	Octal Value	Task Description
F$TIM	003	Get current time. The current time in ASCII format is returned at the location specified by (S1) in the following format:

```
0        8       16      24      32      40      48      56   63
|    h       h   |  :  |    m       m   |  :  |     s     s     |
```

Mnemonic Code	Octal Value	Task Description
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 if the message is to be written to the user logfile, the System Log, or both. The message is terminated by a zero byte or the 80th character.

```
0                                                     57     61 62
|/////////////////////////////////////////////////////|CLASS|OR| FC |
```

Bits	Significance
57-60	Class assigned to message
61	Message-class suppression to be overridden

Mnemonic Code	Octal Value	Task Description
F$MSG (continued)		Bits: 62-63 — Significance: Destination of message: 1 User logfile only 2 System log only 3 System log and user logfiles
F$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 Open Dataset Name (ODN) Table 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.

```
0  1                                          40                   63
|pd|//////////////////////////////////////////|       ODN addr       |
```

If the OPEN call is for a system area LFT/DSP pair (signified by field ODOST equal to OSTSA), 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

The second, third, and fourth entries may result in moving existing LFT entries, DSP entries, and buffers. Additional user field is allocated if insufficient room exists for

Mnemonic Code	Octal Value	Task Description
F$OPN (continued)		adding the LFT, DSP, or buffer. Parameters in the JCB of the user field reflect any movement of these tables.

- 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.

- The DNT and DSP are modified to reflect the processing direction requested.

Processing Direction

10 Input
11 Input/Output

If the OPEN call is for a user-area, system-managed LFT/DSP (signified by field ODOST equal to OSTUA) the DSP/buffer addresses need to be supplied (in fields ODDSP and DPFRST/ DPLMT, respectively). An existing system-area LFT is searched for. If found, the corresponding DSP is moved to the DSP specified in ODDSP. All system-area LFTs which point to the system-area DSP are moved to the user-area LFTs and changed to point to the new user-area DSP. The system-area buffer pointed to by the system-area DSP is moved to the user-area buffer pointed to by the user-area DSP. The system-area LFTs/DSP/buffer are deleted. The user-area LFT and DSP are marked as such.

If, however, an existing system-area LFT is not found, a user-area LFT is created pointing to the user-area DSP and the user-area DSP is marked as open. The user-area LFT and DSP are marked as such.

If ODDSP is a positive, nonzero address, and ODOST is not equal to OSTUA, the user is requesting direct use of a DSP/buffer pair. No LFT is created, and the DSP and buffer must be

Mnemonic Code	Octal Value	Task Description
F$OPN (continued)		wholly contained within the user area (below JCHLM).

When the user manages the DSP, the system makes no attempt to flush the buffer unless the user makes an explicit F$CLS request pointing to the user-managed DSP.

Redundant opens result in the following:

1. Datasets appear as if I/O has occurred.
2. DNT fields are moved into the DSP.
3. The new open status is merged with the current open status.

Mnemonic Code	Octal Value	Task Description
F$MEM	011	Request memory. The amount of memory assigned to a job can 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:

```
0 1 2      7          16                      40                     63
M|/|L|////|T|////////|         DEL            |          WC           
```

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.

L Limit flag. The system sets this flag when it has assigned the maximum allowable amount of memory to the user.

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.

Mnemonic Code	Octal Value	Task Description
F$MEM (continued)		WC Word count. The user must supply the absolute number of words to be added to or deleted from the user's field length. Any words added to the user's field length are added to the upper end. If WC=0, no action is taken other than to return the user's field length in WC.
F$LBN	012	Return last block number. S1 contains the address of the Open Dataset Name Table, the DSP, or any word containing the local dataset name. 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 uses ODDSP as the DSP address if nonzero. Otherwise, the system-managed user- and system-area LFT areas are searched for an LFT with a dataset name equal to ODDN. If not found, close is a no-op. If it is found (or ODDSP pointed to a DSP), the following processing is performed: ● Writes an EOD on a sequential blocked dataset, if the 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. As a result, the job may be declared temporarily irrecoverable. An unblocked dataset has no system buffer. ● Releases any system-managed buffer for the dataset (the DSP is in the system area and points to a system-area buffer) ● Releases the DSP for the dataset, if it is system managed ● Releases any LFT entries for the dataset ● Updates the DNT entry for the dataset to indicate that the dataset is closed

Mnemonic Code	Octal Value	Task Description
F$DNT	014	Create/modify local dataset: DDNFE=0. S1 contains the address of the DDL. This call creates a DNT if one does not already exist. If the dataset already exists, it must be closed except to change DNLM, DNDC, DNLDV and DNSZ. Parameters from the DDL that do not contain 0 are inserted into the DNT. Sense local dataset. DDNFE=1, DNSTAT=0. S1 contains the address of the DDL. This call searches for a DNT. On return, (S0)=0 if the dataset exists; (S0)≠0 if dataset does not exist. The dataset need not be closed. Additional DDL parameters are ignored. Return dataset characteristics: DDSTAT=1. S1 contains the address of the Dataset Definition List (DDL). If the dataset exists a copy is made in the user area at the location indicated by DDDNT. If the dataset does not exist it is created unless DDNFE is set. If DDNFE is set and the dataset does not exist, S0≠0 on return, otherwise S0=0. Additionally, for the create/alter DNT calls, if DDDNT is nonzero, a copy of the created/altered DNT will be returned to the user area pointed to by DDDNT.
F$MDE	015	Set exchange package mode. S1 contains the address of the word containing new mode setting. See the CRAY-OS Version 1 Reference Manual, publication SR-0011, for mode settings.
F$GNS	016	Get next control statement. Copy one card image from control statement buffer to address specified in S1. Error code EREFR (1) is returned in S0 if EOF is encountered on the control statement file.
F$RLS	020	Return to the system the dataset whose Open Dataset Name Table address, DSP address, or any address containing the local dataset name is specified in S1. The dataset is closed and

Mnemonic Code	Octal Value	Task Description
F$RLS (continued)		disposed of according to the disposition code contained in the Dataset Name Table entry for this dataset. The dataset is no longer available to the job. As a result, the job may be declared irrecoverable.
F$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. As a result, the job may be declared irrecoverable, not rerunnable, or both. A check is made of the caller's privileges to ensure the request is allowed. The action taken for a security violation depends on the security mode (see Appendix A of this manual).
F$RDC	022	Read device circular. S1 contains the DSP address. The error bits and the busy bit in the DSP must be monitored by the caller. Automatic recall is requested if bit 0 of S1 is set.
F$WDC	023	Write device 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. As a result, the job may be declared irrecoverable, not rerunnable, or both.
F$GRN	024	Get system revision numbers. S1 contains address of 2-word table. Information is returned in ASCII format, left-justified and blank-filled as follows: COS *x.xx* *mm/dd/yy*
F$DIS	025	Dispose dataset. S1 contains the PDD address. As a result, the job may be declared irrecoverable.
F$JDA	026	Get current Julian date in ASCII format. The

Mnemonic Code	Octal Value	Task Description
F$JDA (continued)		date is returned at the location specified in S1, as follows:

```
0                                                40                   63
 -------------------------------------------------------------------------
|     y      y     |     d        d        d     |     ASCII blanks       |
 -------------------------------------------------------------------------
```

Mnemonic Code	Octal Value	Task Description
F$JTI	027	Return accumulated CPU time for the job in the location specified by S1. The time is expressed in seconds in floating-point format. If the sign bit of S1 is clear, the job time is returned. Otherwise, the task time is returned.
F$ACT	030	Return accounting information to locations starting at the address specified in S1. The format of the information returned is partially described by the Job Accounting Table. In addition, if requested, information about multiple tasks for multitasked jobs is returned in a buffer pointed to by fields in the JAC. The format of that information is described by the Task Accounting Table.
F$SPS	031	Set P register and suspend user. The operator enters the RUN command to lift the user suspension. New program address in S1.
F$CSW	032	Clear sense switch. S1 contains the switch number to be cleared.
F$TSW	033	Test sense switch. S1 contains the switch number to be tested. On return, (S1)≠0 if sense switch is set; (S1)=0 if sense switch is not set.
F$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.

Mnemonic Code	Octal Value	Task Description
F$BIO (continued)		As a result, the job may be declared irrecoverable.

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.

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 can 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 EOF; logically equivalent to $WEOF.
056	Write EOD; logically equivalent to $WEOD.

Mnemonic Code	Octal Value	Task Description
F$BIO (continued)		156 Rewind; logically equivalent to $REWD. DPBWC Word count is the number of words to transfer to or from the user's record area. On a read request, the system returns the actual number of words read. If a null record is read, a zero word count is returned in DPBWC. The user can then use DPEOR, DPEOF and DPEOD to determine if EOR, EOF, or EOD has been reached. DPBWA Word address of user's record area.
F$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 mass storage by issuing an ACCESS request to the Permanent Dataset Manager to obtain the dataset. If the dataset is not present on mass storage, F$AQR acquires it from the front end and accesses it. S1 contains the address of the PDD.
F$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 might place 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

Mnemonic Code	Octal Value	Task Description
F$RRN (continued)		system recovery does not allow the job to be rerun from the beginning.
F$IOA	041	Set (lock) or clear (unlock) IOA bits in the JCB and TCB and alter accordingly the limit address in the user's Exchange Package. When the IOA bits are 1 (lock user's I/O area), the limit address is set to (JCDSP); when the IOA bits are 0 (unlock user's I/O area), the limit address is set to (JCFL). S1 contains the address of the LOCK/UNLOCK indication.
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:

0	28	40 63
/////////////////////////////	LEN	LOC

LEN 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 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 9.)

Mnemonic Code	Octal Value	Task Description
F$DJA	044	Dump job area. S1 contains the address of a local dataset name. If the dataset is nonexistent, a dataset named $DUMP is created. After rewinding $DUMP, the job's Job Table Area (JTA) and user field are written to the dataset. The number of words written is the value of field JTL plus the job's field length, the limit address (LA) minus the base address (BA). The dataset created is unformatted and has no RCWs or BCWs.
F$RPV	045	Enable or disable reprieve processing. S1 contains the address of a 3-word table in the following format: 0 First word address of reprieve code 1 First word address of 30-word save area for exchange package and system use 2 Mask defining error classes to be reprieved S2 is set to 0 to activate reprieve processing. If S2 does not contain 0 or -1, the interrupted abort processing continues. If S2 contains -1, then S1 contains the address of the exchange package to be substituted for the current user exchange package.
F$BGN	046	Begin user code execution. S1 contains the address of the BGN Table. The code to be executed is assumed to be loaded in the user field.
F$RCS	047	Rewind current control statement file. This function is available only to CSP for the rewinding of $CS after block validation.
F$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.

Mnemonic Code	Octal Value	Task Description

F$LIB 052

Library searchlist maintenance. Either get or set the current library searchlist from or to a specified buffer.

S1 Buffer address
S2 Subfunction symbol

Symbol	Significance
LIBGET (or 0)	Return current searchlist into the specified 64-word buffer
LIBSET (or 1)	Set current list to the contents of the 64-word buffer; length of list is determined either by end-of-buffer or a zeroed word.

F$INS 053

Jump to installation-reserved function. S1 should contain the address of an optional parameter list. (S1=0, if no such list exists.) S2 should contain the subfunction code (greater than 0), 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 if no matching dataset name is found in the System Directory.

```
       0 5                                          40                      63
S1 =  |NA|//////////////////////////////////////////|        dn addr          |
```

On return, if there was an error and the NA flag was set:

Mnemonic Code	Octal Value	Task Description

F$ASD (continued)

```
         0                                          40                    63
   S0 = |/////////////////////////////////////////////|  job abort error code |
```

Job abort error codes:

12	JTA overflow
21	Dataset not found
76	Dataset already accessed by job

F$SYM 056 JCL symbol manipulation. Either set or get a JCL symbol.

S1 JCL Symbol Table (JST) base address
S2 Subfunction symbol

Symbol	Significance
SYMGET (or 0)	Return characteristics and symbolic value of the symbol specified in the JST. JST input values are:
JSSN	Symbol name
JSVAL	Address of value buffer
JSLEN	Length of value buffer; if 0, unlimited length is assumed.
SYMSET (or 1)	Set symbolic value for symbol specified in the JST.

F$CSB 057 Conditional control statement maintenance.

S1 JCL Block Information Table (JBI) address
S2 Subfunction symbol

Symbol	Significance
CSBINF (or 0)	Return information for current conditional level in JBI.
CSBDEC (or 2)	CSP only. Decrement current conditional nesting level; used by ENDIF verb.

Mnemonic Code	Octal Value	Task Description
F$CSB (continued)		

Symbol	Significance
CSBEXC (or 3)	CSP only. Flag current conditional nesting level as not in skip mode; used by IF, ELSE, and ELSEIF verbs.

Mnemonic Code	Octal Value	Task Description
F$ISB	060	Iterative control statement maintenance.

S1 JBI address
S2 Subfunction symbol

Symbol	Significance
ISBINF (or 0)	Return current iterative block to specified JBI.
ISBINC (or 1)	Increment current iterative nesting level; used by LOOP verb.
ISBDEC (or 2)	Decrement current iterative nesting level; used by EXITLOOP verb.
ISBRST (or 3)	Reset iterative block; used by ENDLOOP verb to increment loop count and to reposition control statement

Mnemonic Code	Octal Value	Task Description
F$EKO	061	Alter user's ECHO status. S1 contains address of a 1-word parameter block with the following format:

0	32	48	63
//////////////////////////////////	Change mask	New value mask	

Bits	Significance
32-47	Change mask; has a bit set for each class to be changed.
48-63	New value mask; contains the values for the classes changed. 1 turns the class off; 0 turns class on. Message class 0 cannot be turned off. Message class 0 bit left-justified.

Mnemonic Code	Octal Value	Task Description
F$OPT	062	Change job options that can be specified by the user.

S1 Address of OPT table

Mnemonic Code	Octal Value	Task Description
F$POS	063	Tape dataset position request. Two types of requests can be made, rewind and position by block number. S1 Open Dataset Name Table (ODN) or Position Parameter List (PPL) address S2 Function code (0 to rewind, 2 to position)
F$SPM	064	Ready System Performance Monitor (SPM) task for statistics gathering. This request is valid only if the I@USRSPM installation parameter is nonzero.
F$FCH	065	Fetch dataset from a front end and make it local to the job. The address of the base of the PDD is passed in S1.
F$TDT	066	Convert timestamp to ASCII date and time. Called by the TSDT $SYSTXT macro.
F$DTT	067	Convert ASCII date and time to corresponding timestamp. Called by the DTTS $SYSTXT macro.
F$MTT	070	Convert from machine time (RT register value) to timestamp. Called by the MTTS $SYSTXT macro.
F$TMT	071	Convert timestamp to corresponding machine time (RT register value). Called by the TSMT $SYSTXT macro.
F$SPY	072	Enable or disable user execution profile. S1 Address of Execution Profile Table (EP) Subfunction SPY$ON enables the user execution profile. User specifies a time slice for interrupts (in microseconds), the first and last word addresses of the code area to be monitored, and a bucket size which specifies the number of words between FW and LW to be mapped into each bucket. The bucket size must be an even power of 2. Multiple SPY$ON calls can be made. SPY$OFF disables the user execution profile and returns the accumulated information to a buffer specified by the user. The user must also

Mnemonic Code — Octal Value — Task Description

F$SPY (continued)

specify the length of the buffer. The remaining fields in the EP table are filled with the values specified on the enable call; that is, EPFW, EPTS, and EPBS are returned. If EPCNT=0, the buffer has 3 extra words on the end which hold, respectively, UNDER, BETWEEN, and OVER as counters of times the user P address was less than all SPY areas, between SPY areas, and greater than all SPY areas.

SPY$INFO returns the total amount of space needed for all SPY areas.

The times of interrupts are unreliable on a CRAY-1 system not equipped with a programmable clock.

F$MEMORY 073

Request memory. Change the job's memory allocation.

S1 Address of the memory request word. The memory request word has the following format:

```
0      8      16      24      32      40      48      56     63
|//////|F|//////////////////////////////////|          FC         |
```

F Field Length flag. If F=1, WC specifies the number of words of field length to be allocated to the job.

WC Word count. If F=1, WC specifies the number of words of field length that is to be allocated to the job. If F=0, WC specifies the number of words to be added to (if WC is positive) or subtracted from (if WC is negative) the end of the user code/data area.

Memory can be added to or deleted from the end of the user code/data area by specifying WC and setting F to 0. If the user code/data area is expanded, the new memory is initialized to an installation defined value.

Mnemonic Code	Octal Value	Task Description
F$MEMORY (continued)		The job's field length can be changed by specifying WC and setting F to 1. The field length is set to the larger of the requested amount rounded up to the nearest multiple of 512 words (decimal) or the smallest multiple of 512 words large enough to contain the user code/data, LFT, DSP, and buffer areas. The job is aborted if filling the request would result in a field length greater than the maximum allowed the job. The maximum is the smaller of the total number of words available to user jobs minus the job's JTA or the amount determined by the MFL parameter on the JOB control statement.
F$PRV	074	Process user security requests. Subfunctions are PRVSDR, PRVSPF, PRVRPF, PRVSWP. PRV$SDR defines a module as being loaded from the SDR; used only by CSP. S1 points to a PDD for the SDR dataset. PRV$SPF sets user privilege bits in the JTA. S1 points to a User Privilege Table (UPT). PRV$RPF reads user privilege bits from the JTA. S1 points to a buffer which is to contain the UPT upon completion of the request. PRV$SWP sets system job fields in the JTA. The job fields are changed only for the duration of the job step. This subfunction currently sets the job's dataset ownership value (to the value supplied). S1 points to a Security Swap Table (SWT).
F$DSD	075	Define secure dataset (to be released at end of job step). S1 Pointer to dataset name
F$ENC	076	Call STP common routine PWENC to encrypt password; requires SCPRIV privilege.

Mnemonic Code	Octal Value	Task Description

F$ENC (continued)

S1 Address of Encryption Parameter Table (ETT)
S0 =0 for successful completion
≠0 for invalid keyword index

F$TASK 077

F$TASK allows the user to create and manipulate multiple tasks within a single user job step.

S1 contains the address of a TKT table or, optionally, if the sign bit of S1 is set, the TKT table is assumed to reside in the user's registers starting with S2. The subfunction is passed in field TKFC of the TKT. The possible subfunctions are:

TASK$CRE	Create a task
TASK$DEL	Delete a task
TASK$ACT	Activate a task
TASK$DEA	Deactivate a task

The protocol for each subfunction is as follows: To create a user task, the user fills in TKFC=TASK$CRE, and TKNP with the desired P address where the new task will begin execution and does the exchange. The system creates the new task and fills in TKSTS with the return status (see below) and TKID with a system-assigned task ID. The caller's TKMID is set to the caller's task ID. The new task's register set is identical to the creating task's register set with two exceptions. The created task's P address is as specified in TKNP. The creating task is also assumed to have a TKT in its registers starting with S2. This TKT will have both TKID and TKMID set to the created task's ID.

To delete a user task, the user fills in TKFC=TASK$DEL, and TKID with the task ID of the task to be deleted. The deleted task never comes back into execution. If the task being deleted is the last task for the job, the delete is treated as a job advance.

To activate a user task, the user fills in TKFC=TASK$ACT, and TKID with the task ID of the

Mnemonic Code	Octal Value	Task Description
F$TASK (continued)		task to be activated and exchanges to the system. The system activates the appropriate task, which must have been previously deactivated (with a TASK$DEA). To deactivate a user task, the user fills in TKFC=TASK$DEA, and TKID with the task ID of the task to be deactivated and exchanges to the system. The system deactivates the appropriate task pending a matching TASK$ACT. Any of the subfunctions which use TKID as input default to the calling task, if TKID is zero. The following return statuses are returned in TKSTS, if TKNA is set. If the NA flag is not set, the task is aborted with an appropriate error message. The return statuses are: TK$ERXJT Maximum number of tasks/job exceeded TK$ERBP Bad P address for new task passed TK$ERBTS Bad TSB address passed† TK$ERMEM Job cannot get enough memory for new TCB TK$ERBID No task exists with passed task ID TK$ERCAS Designated task to activate is self TK$ERTAA Designated task already active before activate TK$ERTAI Designated task already inactive before deactivate
F$CRASH	100	Allows a user to halt COS under appropriate conditions. The CRASH flag in low STP memory must be enabled. The F$CRASH request is to be used only in system debugging.
F$SYNCH	101	Synchronize tape dataset. S1 contains the ODN address.

† Deferred implementation

Mnemonic Code	Octal Value	Task Description
F$TPOS	102	Get tape position information. S1 Address of buffer to hold information S2 ODN address
F$TBL	103	Return a copy of the specified system table to the starting location identified on the call. Privilege SCNVOK is required.

S0 Completion status:

Status	Meaning
n	Request completed; *n* words moved
-1	Table name not found
-2	Address range error
-3	Table truncated

S1 Address of the following parameter block:

```
     0       8       16      24      32      40      48      56     63
    ┌───────────────────────────────────────────────────────────────┐
 0  │                          Name                                 │
    ├───────────────────────────────┬───────────────────────────────┤
 1  │            Address            |            Length             │
    └───────────────────────────────┴───────────────────────────────┘
```

Field	Word	Bits	Description
Name	0	0-63	Name of the system table in ASCII, left-justified with zero fill.
Address	1	0-31	Address of the buffer to receive the table.
Length	1	32-63	Length of the receiving buffer, in words.

8.2 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 TCB of the job:

- TCEPX is set to 1.
- TCEPF is set to the exchange package flags in the user exchange package, bits 30-38 of word 3.

The Exchange Processor either initiates reprieve processing or issues appropriate error messages and aborts the job step.

If the job is not reprievable, 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.

8.3 EXCHANGE PROCESSOR REQUEST WORD

All requests to the Exchange Processor are made through the Exchange Processor request word (TCEP) in the JTA for the job assigned to the CPU. The Exchange Processor is readied by EXEC whenever TCEP is nonzero. The format of TCEP is as follows:

```
0 2 4 6     16                              40                       63
|NE|C|J|M|//////////////////////////////////|           A             |
```

Field	Bits	Description
TCEPN	0	Normal exit
TCEPE	1	Error exit or execution error
TCEPC	2	Continuation flag
TCEPJ	3	Job Scheduler Request flag
TCEPM	4	JTA Expansion Request flag
TCEPA	40-63	Continuation address; EXP address if TCEPC=1.

The flags in fields N, E, C, and J are mutually exclusive. The user exit flags (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, TCEPA 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.

8.4 JOB SCHEDULER REQUESTS

The Job Scheduler (JSH) requests the Exchange Processor to initiate or abort a job by setting the TCEP word in the job's JTA. The TCEP word must be 0 before JSH can modify it. JSH sets the TCEPJ field to 1, indicating a JSH request.

The Exchange Processor is readied by EXEC when the job is connected if the TCEP word is nonzero, that is, when it becomes the currently executing job. If TCEP is nonzero at the time JSH needs to use it, JSH sets the A bit in the JXT. As soon as the job is connected again and TCEP is 0, JSH is readied. JSH sets the TCEPJ field to 1, and clears the A bit in the JXT.

8.5 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) immediately below the user field.

System tables usually accessed by the Exchange Processor are:

CALL	Call Table
JXT	Job Execution Table
QDT	Queued Dataset Table
SDT	System Dataset 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 tables in the memory pool area of STP. These tables are used as a PDD for rewriting an SDT entry in the DSC, to hold the contents of the TEXT field of an ACQUIRE, DISPOSE, or FETCH statement, or for reading the first block of a dataset for SUBMITs to the input queue.

8.5.1 CALL TABLE (CALL)

The CALL table is a static table composed of a 1-word entry for each user system action request. The contents of the user's register S0 serve as an index into the table on a user call. The format of an entry is as follows:

```
      0       8       16      24      32      40      48      56    63
 n   |Security|/PDD//|        Length          |        Address          |
```

Field	Bits	Description
Security	0-7	Security flags
PDD	8	Set if a PDD is involved in the request
Length	16-39	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	40-63	Address of the routine that processes the request

8.5.2 JOB EXECUTION TABLE (JXT)

The Job Execution Table contains an entry for each job that has been initiated. The JXT contains job parameters and statistics that may be required while the job is rolled out to disk.

8.5.3 QUEUED DATASET TABLE (QDT)

EXP modifies the QDT (through common subroutine RELDNT) when a job releases a local scratch dataset having related disposes.

8.5.4 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 a dataset is submitted to the input queue. The SDT may have associated memory pool areas containing user TEXT field or station slot data.

8.6 USER AREA TABLES USED BY EXP

EXP uses the following tables located either in the user field or in the JTA:

DDL	Dataset Definition List
DNT	Dataset Name Table
DSP	Dataset Parameter Area
JCB	Job Communication Block
LFT	Logical File Table
ODN	Open Dataset Name Table
PDD	Permanent Dataset Definition
SWT	Security Swap Table
TCB	Task Control Block
UPT	User Security Privilege Table

8.6.1 DATASET DEFINITION LIST (DDL)

The Dataset Definition List is used to pass dataset parameters used in creating or modifying the DNT on a F$DNT call.

8.6.2 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, if one exists. The DNT also contains important dataset characteristics and status.

8.6.3 DATASET PARAMETER TABLE (DSP)

The DSP is required for all user I/O and 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. Three types of user DSPs are known to the system. A system-area DSP is one that resides in the area between JCDSP and JCBFB at the high-address end of the user field length. It is pointed to by a system-area (between JCLFT and JCDSP) LFT (multiple LFTs if the dataset has aliases). It conventionally points to a buffer in the system buffer area (between JCBFB and JCFL). A user-area, system-managed DSP is one that resides in the user area (between L@JCB and JCHLM). It is pointed to by a user-area LFT (see LFT description). It points to a buffer in the user area. A user-area user-managed DSP is one that resides in the user area. It does not not have any LFTs associated with it. It points to a buffer in the user area.

8.6.4 JOB COMMUNICATION BLOCK (JCB)

The JCB occupies words 0 through 177_8 of the user field and supports communication between EXP and the user.

8.6.5 LOGICAL FILE TABLE (LFT)

The LFT is a table having two sections. LFT entries exist for most open datasets. They are created by the system or the user when datasets are opened or aliases added. One section, containing what are known as the system-area LFTs, resides at the high-address end of the user field (between JCLFT and JCDSP). It contains an LFT entry for each system-area DSP. The LFTs in the system area point to system-area DSPs. All active system-area LFTs have field LFOST equal to OSTSA. The system LFT area is increased as necessary by the system.

The second section of the LFTs, containing what are known as the user-area LFTs, resides within the user area. Field JCULFT points to the base of the first user-area LFT entry. Any number of entries can exist contiguous with the first LFT entry. A continuation entry consists of word 0 of the LFT equal to -1, and LFDSP containing a pointer to the next block of LFTs. Field JCNULE reflects the total number of user-area LFTs (including continuation entries). The user-area LFTs contain an entry for each system-managed user-area DSP, and point to those user-area DSPs. All active user-area LFTs have field LFOST equal to OSTUA. When an OPEN is done for a user-area system-managed LFT/DSP/buffer, the user must ensure that sufficient user-area LFT entries are available. This is because the system cannot increase the user LFT area. Additionally, the system maintains in the JTA a copy of all user LFTs, both user-area and system-area, from which it validates the user LFTs at necessary intervals.

8.6.6 OPEN DATASET NAME TABLE (ODN)

The ODN table in the user field is required when opening or closing a dataset (FOPN, FRLS, F$RCL, or F$POS call).

8.6.7 PERMANENT DATASET DEFINITION (PDD)

A PDD table in the user field is required for a user permanent dataset management request (FPDM, FAQR, or F$DIS). A PDD table in the JTA is used by EXP when releasing a permanent dataset.

8.6.8 SECURITY SWAP TABLE (SWT)

The Security Swap Table (SWT) is used to pass system job fields from the caller into the JTA when using a F$PRV call with a subfunction of PRV$SWP.

8.6.9 TASK CONTROL BLOCK (TCB)

The Task Control Block contains all execution-point related information (corresponding to a user task) including the exchange package, B, T, and V registers, EXP save areas, EXP internal use tables (DDL, PDD, etc.), and CPU timing information.

8.6.10 USER SECURITY PRIVILEGE TABLE (UPT)

The User Security Privilege Table (UPT) passes user privileges, violation counts, and the user number of a F$PRV call whose subfunction is PRV$SPF into the JTA. The UPT also passes user privileges, user number, and violation counts to the caller from the JTA on a F$PRV call with subfunction PRV$RPF.

8.7 JOB RERUN

Under certain conditions, termination of job processing and returning to the input queue for reprocessing at some later time is 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 obtained if the original execution continued to a normal termination. However, after a job has performed certain functions having 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 the same results for the rerun job.

Normally, when EXP recognizes that the user is performing one of these functions, the job is declared ineligible for rerun. While a job is declared ineligible for rerun, it cannot be rerun under any conditions. Normally, once a job is declared ineligible for rerun, its status cannot be changed again to make it eligible for rerun. A 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 declares a job eligible for rerun after it has been declared ineligible, only in response to a specific user request.

Through the F$NRN system call, the user can 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 nonrerunnable functions. This does not affect the current rerunnability of a job; it merely prevents future declaration of nonrerunnability. The NORERUN macro and control statements permit enabling or disabling detection of conditions that normally cause a job to be not rerunnable.

The following functions on a permanent dataset cause a job to be declared ineligible for rerun:

- SAVE
- DELETE
- MODIFY
- ADJUST
- Any write operation involving a permanent dataset

Conditions resulting in an 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.

- A disk error while attempting to read the roll image of a job copied to mass storage causing a system interruption and the job to be rerun.

  If the job is ineligible for rerun and the roll image cannot be read, the job is returned to the input queue and aborts with an informative message as soon as the Job Scheduler attempts to reinitiate the job.

- A system software or hardware failure necessitates a system Restart.

  If recovery of rolled jobs is not performed or if the job is irrecoverable (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 reinitiate the job.

In any case, an informative message appears in the user log and 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.

8.8 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 enabled. 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 abort 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

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.

The CONTREPV macro allows the user to pass the system an Exchange Package to copy over the current Exchange Package. Thus interrupted execution can be resumed.

8.9 IRRECOVERABILITY OF JOBS

By performing the following functions, a job will be declared irrecoverable:

- 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, PERMIT, or MODIFY of a permanent dataset, and
- A 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. Writing of the index always occurs before EXP performs the request making the rolled image invalid.

JOB SCHEDULER (JSH) 9

The Job Scheduler (JSH) task is responsible for:

- Initiating processing of a job
- Initiating processing of user tasks
- Selecting a user task to be active
- Managing job roll-in and roll-out
- Terminating user tasks
- Terminating a job

9.1 INTRODUCTION

A batch job enters the system as an input dataset staged to the Cray computer by a front-end processor. An interactive job enters the system as a LOGON request. In either case, the staging task (STG) builds a System Dataset Table (SDT) entry containing the job card parameters and sufficient information to find the input dataset, whether it be a mass storage dataset or an interactive dataset.

When the last user task in a job completes, a J$DELETE (delete task) request is made to the Job Scheduler. The Job Scheduler then disconnects the CPU, deallocates the memory, and frees the JXT. As a final step, the SDT associated with the job is returned to the available queue, and the disk resident dataset containing the input job is deleted. Except for the output datasets, which can still be staging back to the front end, all processing required by the job is complete at this time.

The Job Scheduler performs the following functions for all jobs:

- JXT allocation
- Initial TXT allocation
- Memory allocation
- CPU connection

JSH allocates a Job Execution Table (JXT) entry for each job. The information in the JXT contains the current status of the job, location in memory or on a roll file, and working values of priorities. The TXT contains working values concerning CPU use. The most recent job logfile and most recent control statement message are present in the JXT to allow the operator to determine the current job step.

JSH allocates memory to each job represented by a JXT entry. After the memory is allocated, the job is either relocated in memory, read in from the roll file, or initialized. Based on priority considerations, a memory allocation can be taken away from a job, and the job can be written out to the roll file.

JSH allocates the CPUs among the user tasks present in memory and ready to run. A user task is disconnected from the CPU when it suspends itself to wait for a system service, when it exhausts its allocated time slice, or when it is preempted because another (higher priority) user task is made ready to run.

9.2 JSH DESIGN PHILOSOPHY

The Job Scheduler incorporates the following design criteria:

- Equal jobs should share available resources.
- Resource use should be balanced between CPU-bound and I/O-bound jobs.
- Higher priority jobs should be allowed more resource use than lower-priority jobs.
- Responsiveness (quick completion of a job step by human standards) should be available to those jobs that require it.

System resources must be shared between equal jobs. This criterion leads to a job scheduler acting as a perfect round-robin mechanism. Each user task receives a CPU for a certain time slice and is then disconnected. After a CPU is given to each of the other waiting user tasks for the same time slice, the scheduler returns to the first user task.

The second criterion arises because not all jobs are equal. Some jobs are CPU-bound and others are I/O-bound. An I/O-bound user task does not use its full time slice before beginning the next I/O operation. Such a user task needs to be connected to the CPU more often in order to complete in a reasonable elapsed time and to make efficient use of the external channels. This efficiency is accomplished by breaking the round-robin circle and making a queue. A CPU-bound user task comes to the head of the queue, exhausts its time slice and moves to the bottom of the queue. An I/O-bound user task stays at the head of the queue blocked for I/O and some other user task near the head of the queue is connected. A CPU-bound user task could possibly exhaust its time slice and be moved to the bottom of the queue several times before an I/O-bound user task at the head of the queue exhausts its time slice once.

The third criterion also arises because not all jobs are equal. The jobs more important to an installation are given a higher priority. The scheduler is expected to give preference to these jobs. In terms of time slices, the tasks in a high priority job receive a larger time slice. This is not an absolute preference. Because all time slices expire, there are times when a high priority user task follows a low priority job in the queue. However, if other factors are equal, the high priority job completes before the low priority job.

The fourth criterion is responsiveness. This is first of all an interactive requirement. However, an operator KILL command is an example where responsiveness is needed in a batch environment. While the first solution is to raise the priority in these situations, this can be abused. For example, a long job step should not be allowed to dominate the CPU simply because an operator DROP command terminates the previous job step. An interactive user should not be rewarded for typing the control statements of a job that should be left to run overnight. Responsiveness is worth having only because there is an expectation of quick completion. In terms of the queue of user tasks waiting for CPU connection, responsiveness is provided by placing the job at the head of the queue. Abuse is prevented by giving the user task a short time slice.

These four criteria are design goals applicable to many job schedulers. The JSH task as released contains defaults in an attempt to meet these criteria. Because installations differ, there is an implementation criterion of tunability. The Job Scheduler can be changed to give preference to the mix of jobs which are important to a given installation. Any one of the four design criterion mentioned above can be disabled or emphasized at the expense of the other three. This is done at system generation time or by operator commands without taking down the system.

9.3 JXT ALLOCATION

The scheduling of JXTs is unique in that the JXT allocation cannot be reversed. If a poor decision is made in memory allocation, the job can be rolled out and the memory deallocated. Similarly, the CPU can be disconnected as easily as it can be connected. However, a JXT is locked to the job until job termination. For this reason, the JXT allocation is made cautiously.

The number of JXTs available to the system is set by the installation parameter I@JXTSIZ. The number of JXTs available for allocation is given by the variable JXTMAX, subject to the upper bound I@JXTSIZ. JXTMAX can be changed by the operator LIMIT command. A system startup, SUSPEND ALL operator command, or a SHUTDOWN command will set JXTMAX to zero. No jobs are initiated until an operator command raises JXTMAX.

The jobs awaiting JXTs are on the SDT input queue in class rank and job priority order. The SDT entry contains the job name, maximum field length, $OUT size, generic resource requirements, and CPU time limit from the job card.

The SDT also contains the job class assignment made by the Job Class Manager (JCM) when the job enters the input queue. The class assignment can be based on resource limits such as time limit; in which case it takes the form of a contract between the system and the job. In return for initiation priority, the job promises to finish in a given amount of time. The class assignment can also be used by the installation to reserve an initiation for jobs important to the site.

A final input to the JXT allocation process is the flag JADDFLAG. This flag is set nonzero when a change occurs in the system to make it possible to initiate another job. For example, job termination and job class assignment both set the JADDFLAG flag. When the flag is set, each input SDT entry is examined against the following criteria:

First, the priority must be nonzero.

Second, the job must belong to an active class (that is, a class which is ON).

Third, all JXTs reserved for higher ranked classes must remain available.

Fourth, the job class where this SDT is assigned must have a JXT available. This can be either a JXT specifically reserved for this class or a pool JXT available if the class maximum is not exceeded.

Finally, the system generic resources required by the job must be available. These are resources such as tape drives which are not shared between jobs. As a simple deadlock prevention mechanism, a job is not allowed to initiate unless all the nonsharable resources are allocated to the job.

In short, for any job to be initiated, the first job on the input SDT queue which satisfies these conditions is allocated a JXT. The JXT is placed in the memory request queue where it competes for a memory allocation. The variable JADDFLAG is set nonzero to enable another scan of the SDT queue.

9.4 MEMORY ALLOCATION

Memory allocation is the process of mediating the privilege to reside in memory among jobs which are not in memory and not suspended and jobs which are in memory.

9.4.1 ROLL TIME VERSUS RESPONSIVENESS

Two contradictory goals need to be considered during memory allocation. First, because rolling out one job in favor of a second job requires a comparatively long time in terms of CPU cycles, there should be a lengthy time interval between such roll decisions. Second, in situations where responsiveness is a consideration, the process of rolling in the required job must start immediately. Valid reasons can be given for wanting to emphasize either of these goals or even both at different times. The Job Scheduler algorithm was designed with both of these goals in mind.

9.4.2 MEMORY REQUEST QUEUE

The first tool used to make the memory allocation decision is an ordered queue of all jobs awaiting memory. This memory request queue is kept in order by job card priority. For example, a new priority seven job is added after existing priority seven jobs, but before any priority six job. This gives the queue a first-in, first-out ordering among jobs of equal priority and removes the need to age memory priorities among jobs having no memory allocation.

9.4.3 MEMORY PRIORITY

The second tool used to make the memory allocation decision is simply the memory priority. This is a 64-bit quantity kept in JXFMP which is determined by one of the following five formulas:

The easiest formula gives JXFMP for a suspended job. The formula is: JXFMP=0. If the job is rolled out, it is not in the memory request queue. If the job is in memory, it is in the suspended queue and is rolled first when memory is required.

The next formula gives JXFMP for those jobs in the memory request queue where responsiveness is not a consideration. The formula is: JXFMP=P. That is, the memory priority is equal to the job card priority expressed in floating-point form.

The formula giving JXFMP for those jobs in the memory request queue where responsiveness is required is: JXFMP=SB (sign bit). These jobs are found at the head of the memory request queue.

The fourth formula computes the memory priority of a job which has recently been brought into memory (rolled in or initiated). The formula is:

```
JXFMP = P + I@JSMPA + I@JSMPB*(M/I@JFLMAX) - T/I@JSMPC
```

where:

P is the JOB control statement priority

M is the current job size in 1000_8 word allocation units

T is the elapsed time since roll in or initiation completed

I@JSMPA and I@JSMPB are system tuning parameters.

I@JSMPC is a tuning parameter.

I@JFLMAX is an installation parameter which defines the largest amount of memory granted to any user job.

The two terms I@JSMPA and I@JSMPB raise the priority of a job just rolled in. The term I@JSMPB can be used to give higher priority to a large job requiring more time to roll.

The fifth formula, also used for jobs currently having a memory allocation, sets a floor under the fourth formula. It is given by:

```
JXFMP = P-I@JSMPD
```

Note that if I@JSMPD is less than 0, a job is never rolled out to make room for another job of equal priority. On the other hand, if I@JSMPD is greater than 2, a priority five job can force a priority seven job to roll out.

9.4.4 THRASH LOCKS

The third tool used in making the memory allocation decision is a set of three tuning parameters which put a limit on how quickly a job is rolled. These are the system thrash locks found in I@JSLK1, I@JSLK2, and I@JSLK3. A job receiving a memory allocation cannot be rolled out until I@JSLK1 cycles after the job is rolled in. Conversely, a job rolled out without being suspended does not enter the memory request queue until I@JSLK2 + I@JSLK3*(M/I@JFLMAX) clock periods have elapsed. This minimizes thrashing.

9.4.5 ALLOCATION FLAG

A final input to the memory allocation process is a flag, JALLFLAG. This flag is set nonzero when:

- The head of the memory request queue changes,
- A roll out completes,
- A job in memory is waiting for more memory, or
- A job eligible to be rolled exhausts its time slice.

When the flag is nonzero, the memory allocation process executes.

9.4.6 TABLES USED BY ALLOCATION

Several counters associated with memory allocation are maintained throughout JSH. MEMSUSP, MEMROLNG, MEMTALLY and MEMDEMD (defined below) keep track of the amount of memory currently (or soon to be) available. The flags, JALLFLAG and JSQZREQS (defined below) are the means by which memory scheduling is invoked. JSQZREQS is set whenever compaction is attempted and a job could not be moved because of I/O.

The Memory Request Queue (MEMQ) is used by JSH to keep track of all jobs eligible for memory allocation. The MEMQ is a priority-ordered queue (highest-priority job on top). Within priorities, the queue is first in, first out.

Assuming that free memory is not sufficient for an allocation, a job in the MEMQ gains a memory allocation if:

- Enough memory is occupied by jobs having memory priority less than the job waiting and all such memory has been occupied for at least the in-memory thrash lock (I@JSLK1). This quantity of memory is called aged memory and kept in cell MEMAGED. MEMAGED is calculated in subroutine AGEME.

- The job has a higher priority than everything else in the MEMQ.

Almost every SUSPEND/RESUME transition causes a job to be left in the MEMQ with a special priority called DEMAND priority (JXFMP=SB). The only restriction on a memory allocation for a DEMAND priority job is the number of jobs in memory that exceed their thrash locks.

When memory scheduling is invoked, MEMAGED is calculated, in turn, for each job in the MEMQ. An allocation is performed for jobs with MEMAGED sufficiently large (or the size of the job is sufficiently small), taking free memory into account.

JSH keeps track of the amount of subordinate memory relative to a job waiting for an allocation (MEMSUBRD, defined below) in order to keep small, low-priority jobs (which are easy to fit into memory) from always gaining a memory allocation before large high-priority jobs. Subordinate memory is the amount of memory occupied by jobs of lower JXP. When MEMSUBRD is sufficient for an allocation but MEMAGED is not, a limit (JSALLLIM) is set on the MEMQ search and JSH waits until JALLFLAG is set. Assuming a higher-priority job is not introduced into the MEMQ, the job at JSALLLIM eventually gains an allocation unless the amount of free memory changes (JSALLSMD and JSALLMTD).

Whenever a job gains a memory allocation and is removed from the MEMQ, the MEMQ search limit, JSALLLIM, is set to the bottom of the MEMQ allowing small, low-priority jobs into memory until a normal or subordinate allocation is found.

The following cells are maintained in STPTAB.

- MEMORY is the total amount of memory in the system that can be allocated to user jobs when STAGER is not active.

- MEMJOBS is the total amount of memory that is currently allocatable to jobs, that is, MEMORY less the dynamic portion of STAGER buffers.

- MEMTALLY is the total amount of memory which is unallocated (from the MST).

- MEMROLNG is the amount of memory being rolled out.

- MEMSUSP is the amount of memory occupied by jobs having all tasks suspended.

- MEMAGED is the amount of memory occupied by jobs whose in-memory thrash locks have expired and who have memory priority less than the job for which MEMAGED is calculated.

- MEMSUBRD is the amount of memory occupied by jobs having JXP less than the job for which MEMSUBRD is calculated.

- MEMDEMD is the amount of memory required by all jobs in the MEMQ having DEMAND priority.

- JROLLCTL is a flag indicating whether rolling is allowed. There are no operator commands to control the state of this flag. Rolling is allowed when this flag is set.

- JOBCOUNT is the number of jobs in memory.

- I@EXPANS is the amount of space that must be left free after a memory allocation. The space need not be contiguous.

- I@JOBMIN is the minimum number of jobs that must be in memory before expansion space (I@EXPANS) is enforced.

The following cells are used by JSH internally.

- JSALLCPR is the priority of the current allocation, assuming that MEMQ is not empty. JSALLCPR is almost always checked before signaling memory allocation with JALLFLAG.

- JSALLCSZ is the size of the current allocation.

- JSAMGOAL is used by subroutine MEMRY to keep track of the amount of memory that must be freed or rolled to make room for the current allocation.

- JALLFLAG indicates that memory has been freed or that a new job has been added to or removed from the MEMQ.

- JSQZREQS indicates that a previous attempt to compress memory failed (I/O active, job not movable).

- JSALLPEN indicates that an allocation is pending and the scheduler is waiting for job(s) to finish rolling out.

- JSALLORD is the JXT ordinal of the current allocation.

- JSALLLIM is the JXT ordinal of the lowest priority job that will be considered during the current pass through memory allocation.
- JSALLSUB indicates that the job at JSALLORD could gain a memory allocation, if all jobs with lower JXP become available to roll.
- JSALLMTD indicates that a job in memory successfully expanded, since the last time memory allocation exited with a pending subordinate memory allocation causing MEMTALLY to decrease.
- JSALLSMD indicates that a job with all tasks suspended had at least one of its tasks resumed, since the last time memory allocation exited with a pending subordinate memory allocation causing MEMSUSP to decrease.

All time intervals used in memory priority calculations are elapsed times. It is the responsibility of the CPU connection process described in the next section to schedule CPU time for the job's tasks.

In the examples which follow, jobs A, B, C, and D are identical. Each has equal priority and requires one third of available memory to execute. Job L was obtained by doubling the size of job D. Job d is simply D with a smaller priority.

Figure 9-1a shows four jobs that must share memory three at a time. At time t_0, jobs A, B, and C arrive in memory. At time t_1, the memory priority of jobs A, B, and C ages to job statement priority P. Since job D is waiting for memory, A rolls out and D rolls in with an initial memory priority identical to that of jobs A, B, and C at t_0. At time

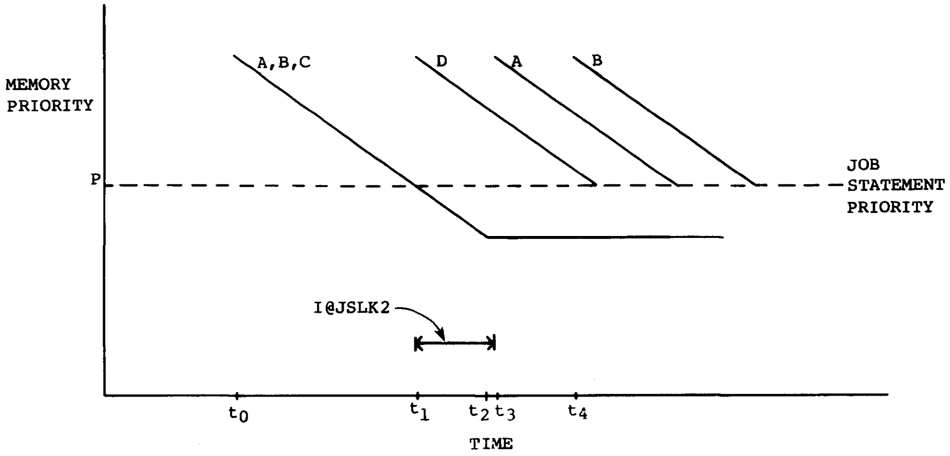


Figure 9-1a. Memory priority variation

t_2, the memory priority of jobs B and C reach their minimum values. At time t_3, the out of memory thrash lock (I@JSLK2) expires for job A. Since B and C have lower memory priority, job B rolls out and job A rolls in. At time t_4, the out of memory thrash lock (I@JSLK2) expires for job B. Since C has lowest priority, C rolls out and B rolls in.

Figure 9-1b shows four equal-sized jobs and one twice as large. Parameter I@JSMPB is 0 giving no extra priority to the extra size. Jobs A, B, C, and L share memory. Job L is twice as large as job A. Jobs A, B, and C are identical. At time t_0, jobs A, B, and C arrive in memory. At time t_1, the memory priority of A, B, and C ages back down to job statement priority P. Job L is now eligible for memory. Jobs A and B roll out. Job L rolls in. At time t_2, the out of memory thrash lock (I@JSLK2) expires for jobs A and B. Thus, C rolls out and A rolls in. At time t_3, the memory priority of job L ages to P and jobs B and C are eligible for memory. Job L rolls out. Jobs B and C roll in.

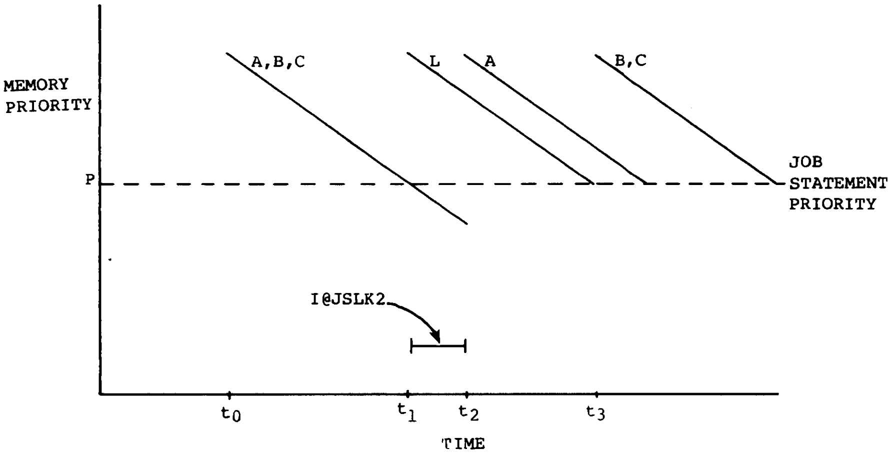


Figure 9-1b. Memory priority variation

Figure 9-1c shows the situation in figure 9-1b again. The difference is that I@JSMPB is positive giving the larger job a longer time in memory. At time t_0, jobs A, B, and C arrive in memory. At time t_1, the memory priority of jobs A, B, and C ages to P. Job L is eligible for memory allocation. Jobs A and B roll out. Job L rolls in with larger initial memory priority. At time t_2, the out of memory thrash lock (I@JSLK2) expires. Job C rolls out. Job A rolls in. At time t_3, the memory priority of job A ages to P. Job A rolls out. Job B rolls in. At time t_4, the memory priority of job L ages to P. Job L rolls out. Job C rolls in. At time t_5, the out of memory thrash lock for job A expires. Job A rolls in. At time t_6, the memory priorities of job B and C have aged to P. Jobs B and C roll out. Job L rolls in.

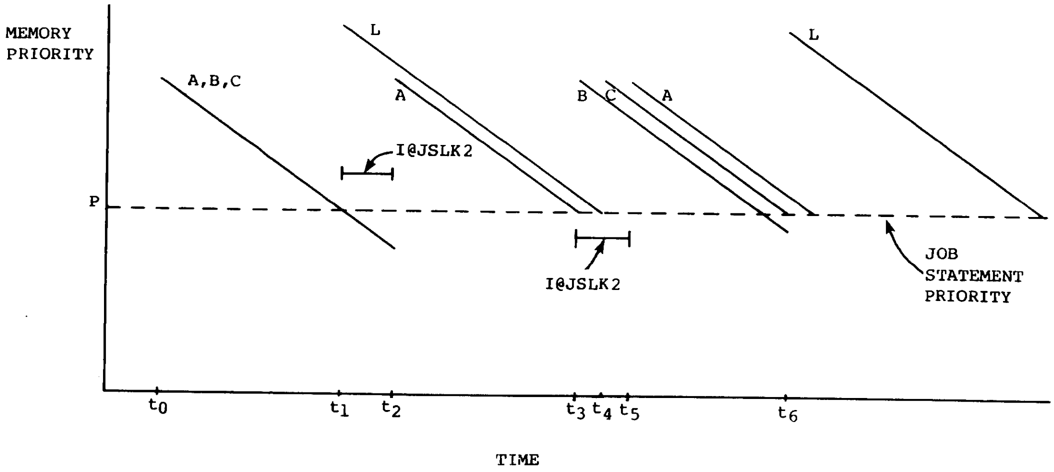


Figure 9-1c. Memory priority variation

Figure 9-1d shows four jobs of equal size, one having smaller priority. Because I@JSMPD is set large enough, the lower priority job will receive a memory allocation at certain times. At time t_0, jobs A, B, and C arrive in memory. At time t_1, job A ages to p. Job A rolls out; job d rolls in. At time t_2, the out of memory thrash lock expires for A. A rolls in and B rolls out. Note that the memory priority of d is still higher than that of C. At time t_3, the out of memory thrash lock expires for B. B rolls in; C rolls out. At time t_4, the out of memory thrash lock expires for C. C rolls in; d rolls out. At time t_5, the memory priority of job A ages to p. A rolls out; d rolls in. At time t_6, the out of memory thrash lock expires for A. A rolls in; B rolls

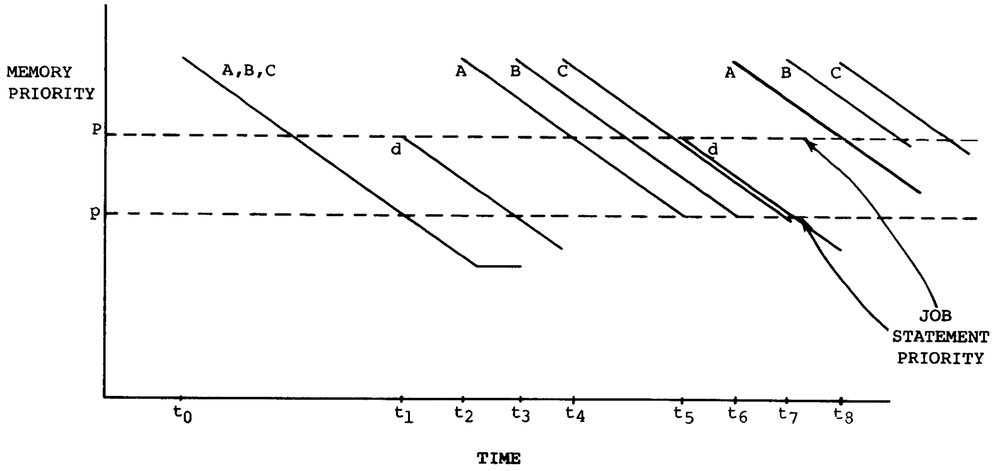


Figure 9-1d. Memory priority variation

out. At time t_7, the out of memory thrash lock expires for B. B rolls in; C rolls out. At time t_8, the out of memory thrash lock expires for C. C rolls in; d rolls out.

Figure 9-1e shows the same situation as figure 9-1d with responsiveness an added consideration. At the beginning of the graph, job d is suspended by the operator. At time t_0, the operator types a DROP command against the job. The job is rolled in quickly. However, at time t_1, the job is no longer protected by I@JSLK1 and is rolled back to the disk.

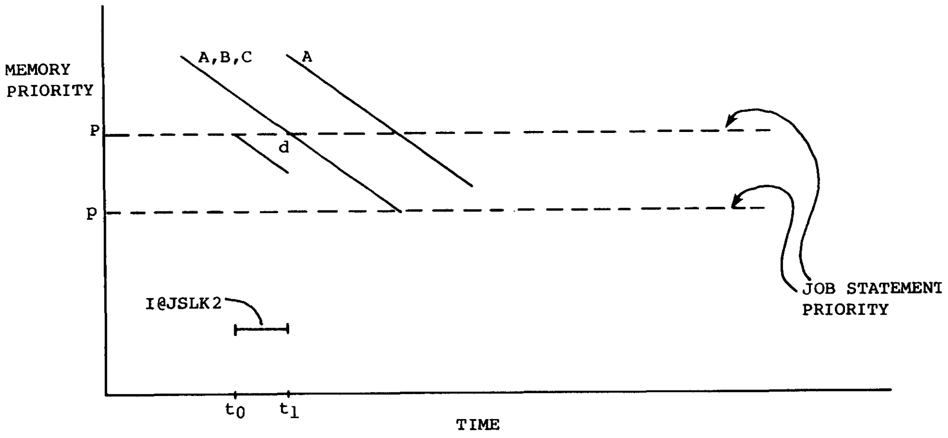


Figure 9-1e. Memory priority variation

9.5 CPU CONNECTION

The goal of the CPU connection algorithm is to quickly find a user task eligible to be connected to a CPU and connect it. In addition, system environment is taken into account to assure that I/O-bound user tasks are connected more often, high-priority user tasks receive more connect time, and user tasks requiring responsiveness receive connect time quickly.

The principal tool making the connect decision is an ordered queue of all user tasks eligible to be connected. To speed the decision, user tasks are removed from this queue when they are suspended for any reason other than an I/O suspend. The user task to be connected is the first user task in the queue which is not suspended for I/O.

Each user task in the queue has a positive time slice kept in JXTS and expressed in units of CPU cycles. This time slice is allocated by one of three methods when the job enters the queue:

- If a user task entering the queue has a zero time slice as happens if the job just rolled in, the time slice is set equal to the system tuning parameter I@JSITS. The user task is entered into the queue at the head.

- If the user task entering the queue has a negative time slice, as happens if the previous time slice is exhausted, a new time slice is computed using the formula:

  $$JXTS = I@JSTS3 + I@JSTS2*(P) + I@JSTS1*(P^2) + I@JSTS0*(P^3)$$

  where:

  P is the job card priority and I@JSTS3, ... , I@JSTS0 are system tuning constants. The user task is entered into the bottom of the queue. A time slice computed by this formula is subject to an upper bound to prevent job time limit overrun.

- The third method for assigning a time slice is used when a user task reenters the queue with an existing positive time slice. This happens when a job completes a job step and is removed from the queue to wait for I/O to finish. Because the queue is constantly changing, the user task cannot be reinserted at its former location. The user task is entered at the head of the queue with a time slice equal to:

  JXTS = MIN(I@JSITS, JXTS).

If the user task which is ready is at the head of the queue, it preempts the currently connected user task.

Each time a user task is disconnected, whether as a result of a suspend call by the user task or as a result of being preempted, the time spent executing is subtracted from the current time slice. Because heavily I/O-bound programs such as disk exercisers can run for a long elapsed time with little CPU time spent in the user space, an additional amount reflecting part of the system overhead of doing a disconnect is also subtracted from the time slice. This additional amount is a system parameter called the disconnect cost I@JSCOS.

When a time slice becomes negative, the user task is removed from the CPU queue. In addition, some bookkeeping is performed for the memory allocation process. The value of the memory priority is updated. If the memory priority is at the minimum specified by P-I@JSMPD, a bit is set and further memory priority updates can be skipped. The elapsed time since last residence change is compared to the system thrash lock parameter. If it is permissible to roll this job, the available-to-roll bit is set. Finally, the memory priority of the job whose task is being

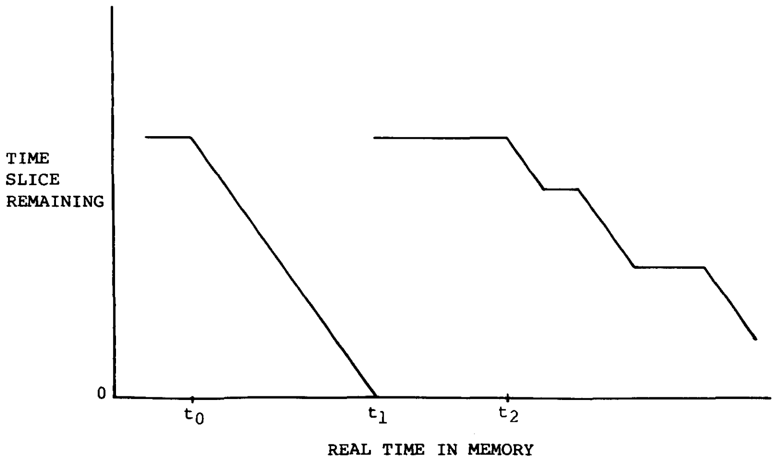


Figure 9-2. Time slice for CPU-bound user task

disconnected is compared against the memory priority of the job awaiting the next memory allocation. If the job whose task is being disconnected is rolled to make way for the waiting job, the size of the first job is added to MEMAGED and the allocation flag JALLFLAG is set nonzero. Figure 9-2 shows the graph of the time slice for a CPU-bound user task. Between time t_0 and t_1, it is at the head of the queue. From time t_1 to t_2, it is too close to the bottom to be connected. After time t_2, it can be connected if the user tasks ahead of it are blocked for I/O.

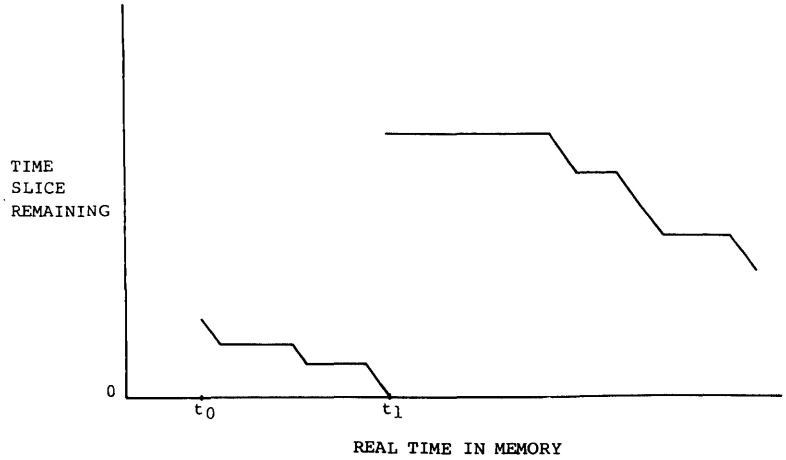


Figure 9-3. Time slice for I/O-bound user task

Figure 9-3 shows the graph of time slice for an I/O-bound user task. Between t_0 and t_1, it is at the head of the queue.

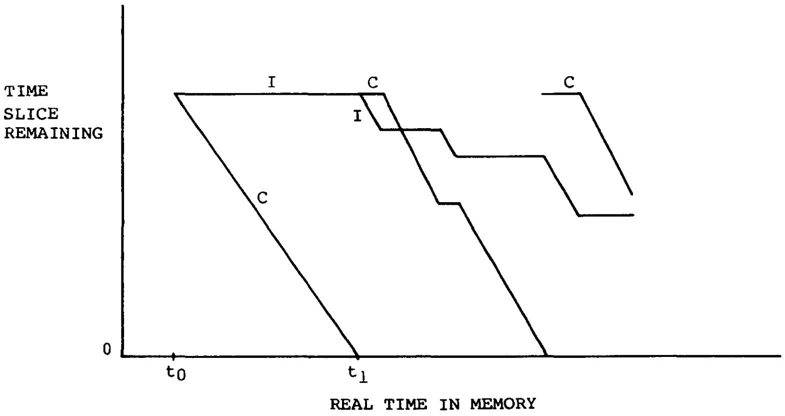


Figure 9-4. CPU competition

Figure 9-4 shows the time slices of a CPU-bound user task C and an I/O-bound user task I as they compete for CPU time. At time t_0, C is at the head of the queue. Note that C exhausts its second time slice without coming to the head of the queue.

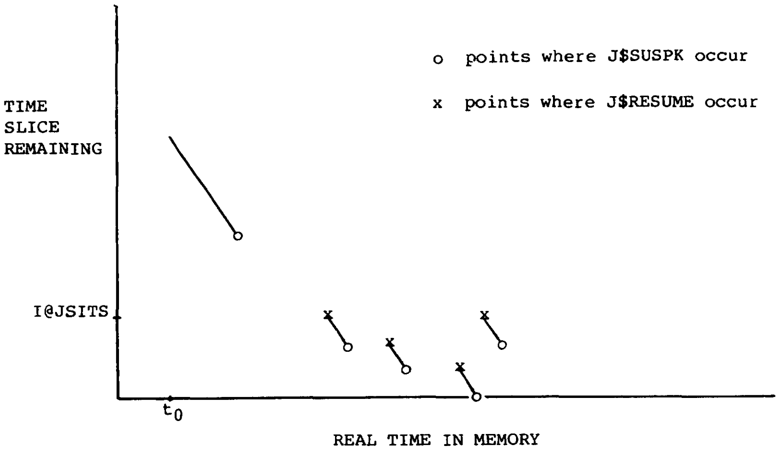


Figure 9-5. Suspended user task

Figure 9-5 shows a user task which is performing an AUDIT. Because the F$PDM call to EXP results in a J$SUSPK (suspend but keep in memory) call to JSH, the user task is removed from the queue until a J$RESUME call signals completion. The reentry time slice is bounded above by I@JSITS.

Figure 9-6 shows an interactive user task beginning a CPU-bound job step. Because the step cannot complete, the user task loses its head-of-the-queue advantage.

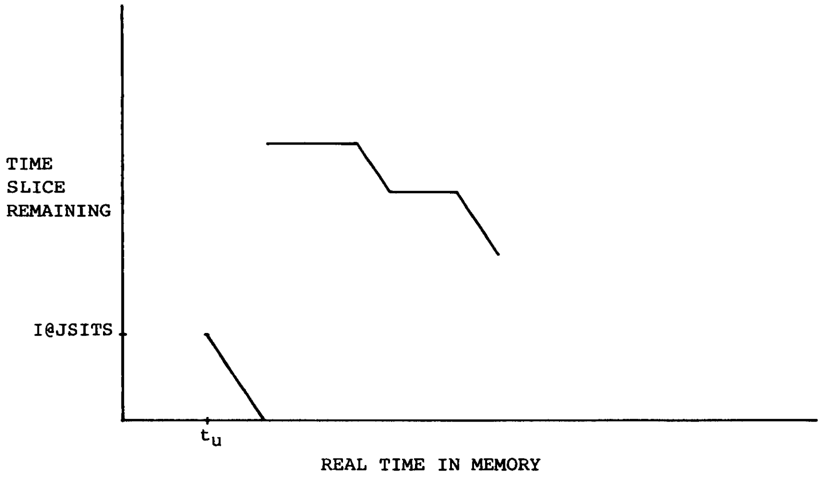


Figure 9-6. Interactive job

9.6 MEMORY MANAGEMENT

Memory management consists of managing the memory available from the system for use by jobs. The Job Scheduler performs this task for the system.

The Job Scheduler (JSH) management of the memory reserved for users will be described first, then management of an individual job's memory will be described followed by a description of the memory requests involved in memory management.

9.6.1 JSH MANAGEMENT OF USER MEMORY

Memory is allocated to the system by STARTUP at both the low-address and high-address ends of memory. After all system tasks have been initialized, JSH claims the remaining 512-word decimal blocks of memory for future allocation to jobs or to the system for system buffers.

Segments of memory are allocated to jobs by the Job Scheduler using a first fit method; that is, the job is allocated memory from the first (lowest addressed) segment large enough to contain it. The last (highest addressed) segment is always allocated to the system. Segments are allocated in multiples of 512-word decimal blocks.

Deciding who gets memory

JSH allocates the initial system segment during JSH initialization. This segment is at the high-address end of the block of memory allocated to jobs (from SBUFBASE to MEMMAX) and is I@BFSIZ words long.

Jobs that are waiting for memory are jobs that are either in memory and need to expand or they are not in memory (initiating or rolled out to disk) and need to be brought into memory. When allocation is possible, JSH looks to see if a job that is waiting for memory can be given memory. Jobs that are waiting for memory are scanned in descending priority order.

The system gets priority over jobs for memory. When a system request is made for memory, JSH immediately looks to see if the system can be given memory.

A tally is kept of the total amount of memory that will be available when all currently scheduled rolls complete. If this tally indicates that there is enough free memory to satisfy the waiting job (system), the job (system) will be given the memory. If there is not enough memory available, any jobs that are either suspended or of a lower priority will be rolled out if rolling them out would enable the request to be satisfied. If a job that is in memory cannot expand (that is, not enough jobs in memory are either suspended or of a lower priority), it will be considered suspended and will be rolled out if any other job or the system needs its space.

Expansion space

A job is brought into memory (initiated or rolled in from disk) only if there is enough memory to contain the job and leave I@EXPANS amount of expansion space. Expansion space is required to allow the jobs that are already in memory to expand. Expansion space is ignored whenever there are fewer than I@JOBMIN jobs in memory.

Figure 9-7 shows memory after JOB1 and JOB2 initiate and JOB3 rolls in. JOB4 will not be brought in because not enough memory is available to contain the job and the required expansion space.

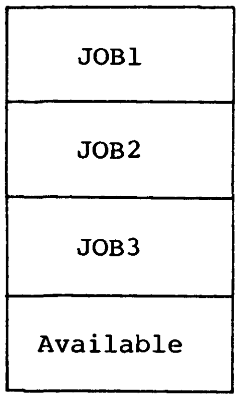


Figure 9-7. Memory allocation

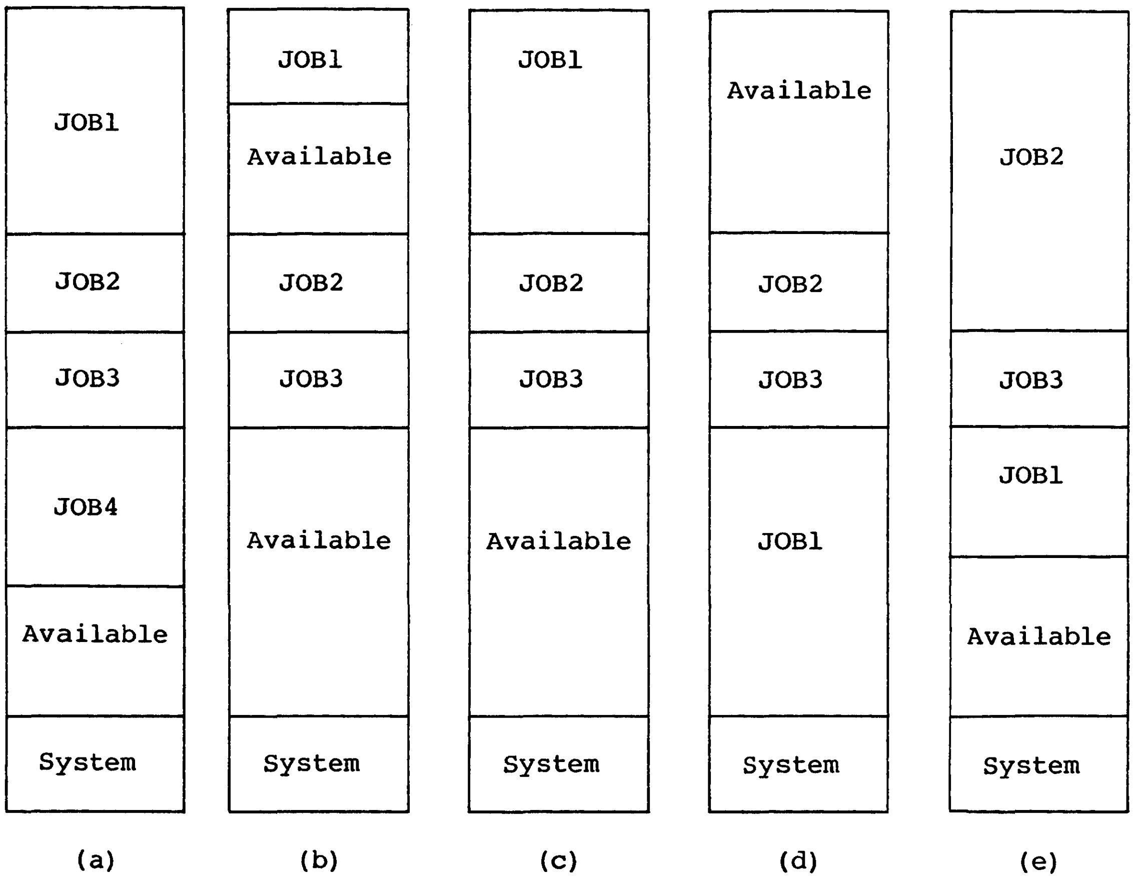


Figure 9-8a - 9-8e. Memory management

Allocating, deallocating, and compacting memory

Figure 9-8a shows memory before any change. Figure 9-8b shows memory after JOB4 terminates and JOB1 decreases its field length. The freed memory is marked available. Consecutive available segments are merged into one larger available segment but no other memory compaction is done.

Figure 9-8c shows memory after JOB1 increases its field length. JSH expands a job in place whenever possible.

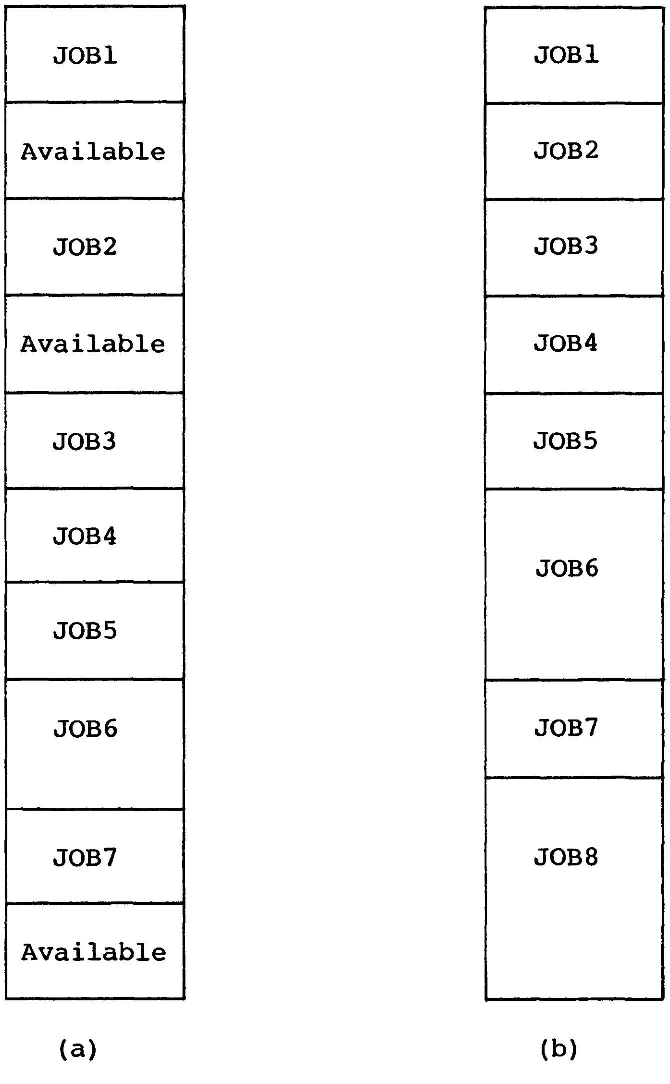


Figure 9-9. Memory compaction

Figure 9-8d shows memory after JOB1 increases its field length again. If expansion in place is not possible, the job is moved to the first (lowest addressed) available segment large enough to contain the job. If there is enough available space to contain the job but it is not contiguous, the job will be rolled out and memory will be compacted.

Figure 9-8e shows memory after the system requests more space. Memory is compacted upward and the system slot is increased by the requested amount.

When a job is being brought into memory and there is enough available space but it is not contiguous, memory will be compacted. Memory is compacted toward the low address end of memory until enough contiguous space is available.

Figure 9-9a shows memory before any change. Figure 9-9b shows memory after memory is compacted and JOB8 is rolled in.

9.6.2 MANAGEMENT OF A JOB'S MEMORY

A job's memory is composed of several areas. Some of these are managed exclusively by the system for the user; others are managed by both the system and the user.

Figure 9-10 illustrates the distinct areas within a job's memory. The total job size equals the length of the job's Job Table Area (JTA) plus field length. The lined area between JCHLM and JCLFT is pad (unused space) within the job. Enough pad is always in the job to guarantee that the job size is a multiple of 512 decimal words.

Requests to change a job's memory are made by the user and the system. JSH is the only task that actually changes any area of memory within the job, except the user code/data area. JCHLM is reset by other tasks and programs (that is, EXP, CFT, etc.).

User requests

User memory requests are made using the MEMORY control statement, $SYSLIB MEMORY routine, and CAL MEMORY macro. CSP processes the MEMORY verb by using the MEMORY macro. The $SYSLIB MEMORY routine also uses the MEMORY macro. The MEMORY macro makes a system call using the F$MEMORY system function code only when necessary. Some requests require only a JCB field access; in those cases the MEMORY macro gets the required JCB information without calling the system.

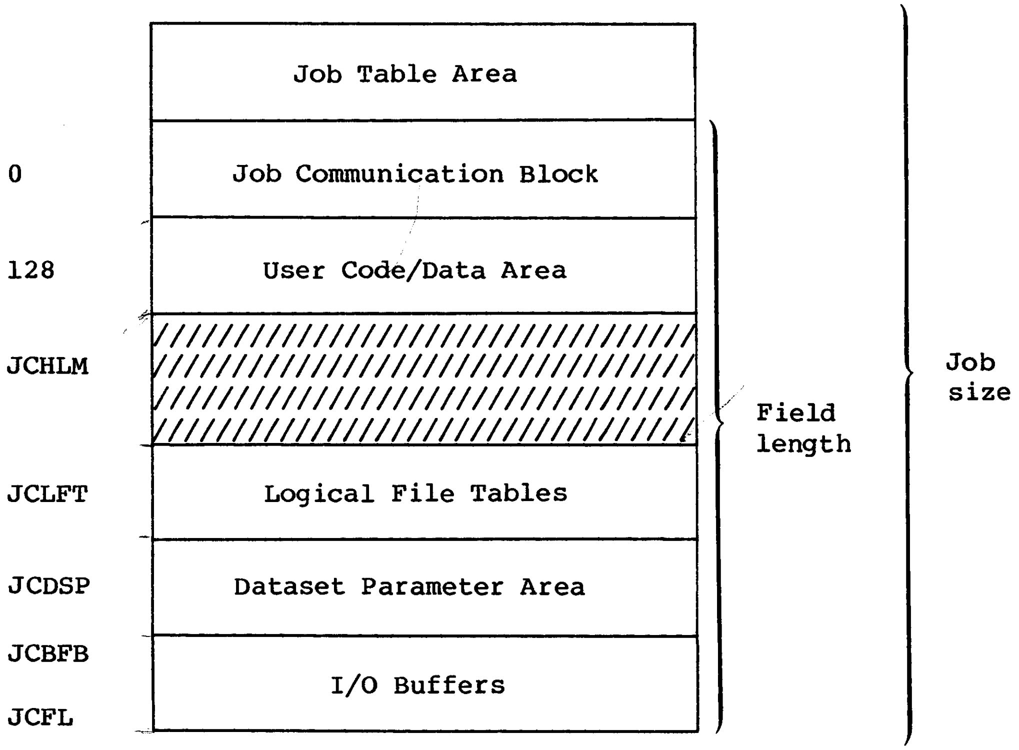


Figure 9-10. The areas of a job's memory

EXP processes the F$MEMORY system request by making a J$ALLOC task request to JSH.

System requests

System memory requests are made for jobs by EXP using a J$ALLOC request to JSH. System memory requests are made for the system by STG and SCP using J$GETM and J$RETM requests to JSH.

Special system handling is required when a job advances a step, loads a binary for execution or terminates.

At job termination, EXP clears JCU (user-managed field length reduction mode bits) and uses a J$ALLOC task request to JSH to decrease the user code/data area to the size of the JCB. Since the job is in automatic field length reduction mode, the field length will automatically be decreased.

When the job advances a step, CSP is loaded with no field length reduction. JCUL (the local memory mode bit) is cleared if the last verb processed was not MEMORY. If the job is in automatic field length reduction mode, the job's field length will be reduced to a size just large enough to contain CSP before a system verb is processed (unless the verb is defined in CSP so as to disallow automatic field length reduction). MEMORY disallows such automatic field length reduction.

CSP and the loader load binaries for execution. The binary's PDT contains the program length, the length of blank common, the amount of pad that is to be left in the field length and whether the job is to be placed in user-managed field length reduction mode before the binary is loaded. CSP and the loader use the PDT information to set the correct field length for the job.

J$ALLOC request processing

JSH processes the J$ALLOC task request. A J$ALLOC request can cause:

- Memory to be added to the end of the JTA,
- Memory to be added to or deleted from the end of the user code/data area,
- Memory to be added to the beginning or end of the buffer area or deleted from anywhere in the buffer area,
- Memory to be added to the beginning of the LFT area,
- Memory to be added to the end of the DSP area, and
- The field length to be increased or decreased.

Changing any area of the job other than the JTA can also cause a change in the job's field length and/or the amount of pad within the job.

Pad can be used to satisfy requests to expand any area in the job (except the JTA) without causing the field length to increase. When the JTA expands, pad is never used. The field length remains the same and the job size increases by the amount of JTA expansion. When the job needs memory to increase any area within the job (except the JTA) and there is not enough pad to satisfy the request, the job's field length will be increased. The field length is increased by an amount large enough to satisfy the request and leave I@MINPAD pad in the job. When the job releases memory from the user code/data and/or buffer areas, the released memory remains within the job (in the JCHLM-JCLFT area) if the job is in user-managed field length reduction mode or the resulting total pad would not exceed I@MAXPAD. If the job is in automatic mode and the resulting pad would exceed I@MAXPAD, the job's pad is reduced to I@MINPAD.

Figure 9-11 shows a decrease in memory at the end of the user code/data area. / indicates pad. x indicates area to be deleted. The field length does not change because the job is in user-managed field length reduction mode or the resulting total pad does not exceed I@MAXPAD. JCHLM is decremented by the requested amount of decrease.

```
                JTA                        JTA
                JCB                        JCB
         user code/data             user code/data
         xxxxxxxxxxxxxx             ////////////// JCHLM
         xxxxxxxxxxxxxx             //////////////
   JCHLM //////////////             //////////////
   JCLFT       LFT                        LFT      JCLFT
               DSP                        DSP
             buffers                    buffers
```

Figure 9-11. Decreasing the user code/data area

```
                JTA                        JTA
                JCB                        JCB
         user code/data             user code/data
   JCHLM //////////////             ////////////// JCHLM
   JCLFT       LFT                  //////////////
               DSP                        LFT      JCLFT
             buffers                      DSP
         xxxxxxxxxxxxxx                 buffers
```

Figure 9-12. Decreasing the buffer area

Figure 9-12 shows a decrease in memory in the buffer area. / indicates pad. x indicates area to be deleted. The field length does not change when the job is in user managed field length reduction mode or the resulting total pad does not exceed I@MAXPAD. The LFTs, DSPs and the buffers preceding the deleted buffer are moved to a location equal to their current location plus the size of the deleted buffer.

Figure 9-13 shows a decrease in memory at the end of the user code/data area. / indicates pad. x indicates area to be deleted. The field length does change when the job is in automatic field length reduction mode and the resulting total pad would exceed I@MAXPAD. JCHLM is decremented by the requested amount of decrease. The pad between HLM and LFT is reduced to I@MINPAD.

```
                       JTA                    JTA
                       JCB                    JCB
                  user code/data         user code/data
                  xxxxxxxxxxxxxx         ////////////// JCHLM
                  xxxxxxxxxxxxxx              LFT       JCLFT
            JCHLM //////////////              DSP
                  //////////////            buffers
            JCLFT      LFT
                       DSP
                     buffers
```

Figure 9-13. Decreasing the user code/data area

Figure 9-14 shows a decrease in memory in the buffer area. / indicates pad. x indicates area to be deleted. The field length does change when the job is in automatic field length reduction mode and the resulting total pad would exceed I@MAXPAD. The field length is reduced to the sum of the user code/data, LFT, DSP and resulting buffer areas, plus I@MINPAD.

```
                       JTA                    JTA
                       JCB                    JCB
                  user code/data         user code/data
            JCHLM ///////////////        ////////////// JCHLM
                  ///////////////             LFT       JCLFT
            JCLFT      LFT                    DSP
                       DSP                  buffers
                  xxxxxxxxxxxxxxx
                     buffers
```

Figure 9-14. Decreasing the buffer area

```
                       JTA                    JTA
                       JCB               00000000000000
                  user code/data              JCB
            JCHLM //////////////         user data/code
                  //////////////         ////////////// JCHLM
                  //////////////         //////////////
            JCLFT      LFT               //////////////
                       DSP                    LFT       JCLFT
                     buffers                  DSP
                                            buffers
```

Figure 9-15. Increasing the JTA area

When the job requests an increase of memory in the JTA, pad is never used and the job size always increases by the requested amount. Figure 9-15 shows an increase in memory at the end of the JTA area. / indicates pad. 0 indicates new area. The field length does not change. The job size is increased by the requested amount. The entire field length is moved to a location equal to its current location plus the requested amount of increase.

When the job requests an increase of memory in any area of the job other than the JTA, the field length does not increase if there is enough total pad in the job to satisfy the request.

Figure 9-16 shows an increase in memory at the end of the user code/data area. / indicates pad. 0 indicates new area. The field length does not change because there is enough total pad to satisfy the request. HLM is increased by the requested amount.

```
             JTA                  JTA
             JCB                  JCB
       user code/data       user code/data
JCHLM //////////////       00000000000000
      //////////////       00000000000000
      //////////////       ////////////// JCHLM
JCLFT       LFT                  LFT       JCLFT
            DSP                  DSP
          buffers              buffers
```

Figure 9-16. Increasing the user code/data area

Figure 9-17 shows an increase in memory in the buffer area. / indicates pad. 0 indicates new area. The field length does not change when there is enough total pad to satisfy the request.

```
             JTA                  JTA
             JCB                  JCB
       user code/data       user code/data
JCHLM //////////////       ////////////// JCHLM
      //////////////       //////////////
      //////////////             LFT       JCLFT
JCLFT       LFT                  DSP
            DSP            00000000000000
          buffers              buffers
```

Figure 9-17. Increasing the buffer area

Figure 9-18 shows an increase in memory at the end of the user code/data area. / indicates pad. 0 indicates new area. The field length does change because there is not enough total pad to satisfy the request. The field length is increased by an amount large enough to satisfy the request and leave I@MINPAD amount of pad.

```
           JTA                    JTA
           JCB                    JCB
      user code/data         user code/data
JCHLM //////////////         00000000000000
JCLFT      LFT               00000000000000
           DSP               ////////////// JCHLM
         buffers             //////////////
                                  LFT       JCLFT
                                  DSP
                                buffers
```

Figure 9-18. Increasing the user code/data area

Figure 9-19 shows an increase in memory in the buffer area. / indicates pad. 0 indicates new area. The field length does change because there is not enough total pad to satisfy the request. The field length is increased by an amount large enough to satisfy the request and leave I@MINPAD amount of pad.

When the user requests a specific field length, the total pad within the job is increased or decreased as needed.

```
           JTA                    JTA
           JCB                    JCB
      user code/data         user code/data
JCHLM //////////////         ////////////// JCHLM
JCLFT      LFT               //////////////
           DSP                    LFT       JCLFT
         buffers                  DSP
                             00000000000000
                                buffers
                             00000000000000
```

Figure 9-19. Increasing the buffer area

Figure 9-20 shows an increase in the job's field length. / indicates pad. 0 indicates new area. The field length is increased to the requested amount rounded up to the nearest multiple of 512-decimal words. The area from JCLFT through the buffers is moved to a location equal to their current location, plus the amount of field length increase.

```
           JTA                     JTA
           JCB                     JCB
      user code/data          user code/data
JCHLM //////////////          ////////////// JCHLM
JCLFT      LFT                00000000000000
           DSP                00000000000000
         buffers                   LFT       JCLFT
                                   DSP
                                 buffers
```

Figure 9-20. Increasing the field length

Figure 9-21 shows a decrease in the job's field length. / indicates pad. As much pad as required is returned to the system until the field length equals the requested amount rounded up to the nearest multiple of 512-decimal words.

```
           JTA                     JTA
           JCB                     JCB
      user code/data          user code/data
JCHLM //////////////          ////////////// JCHLM
      //////////////               LFT       JCLFT
      //////////////               DSP
JCLFT      LFT                   buffers
           DSP
         buffers
```

Figure 9-21. Decreasing the field length

9.7 JOB INITIATION

When JSH sets up a job for its first chance to execute in the CPU, it initializes the job field length based on the size of CSP. It then initializes the job's JXT entry, Job Table Area (JTA), and the first user task. 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)
JTJCB	JCB pointers
JTCMSG	Conditional message flags
JTSID	Source ID (from SDT)
JTDID	Destination ID (from SDT)
JTJXT	Pointer to JXT entry
JTTCB	Offset to first TCB in JTA
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, can 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 9-1. The DNT for the roll-image dataset is in the JXT rather than in the JTA.

9.8 JOB STATUS

The 22-bit status field (JXSTAT) in each job's JXT entry is described in table 9-2. The bits labeled Q, R, L, O, and M determine the job's status; the other bits modify the job's status.

The status of each task in a job is described by the 22-bit task status field TXSTAT in the task's Task Execution Table (TXT). If all task status bits in TXSTAT are 0, the user task is said to be waiting to be connected to the CPU (status W).

Table 9-1. DNT initialization

DNT field	Files initialized			
	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 where 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 DNP (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.

Table 9-2. Status bit assignments

Bit position in JXSTAT	Bit name	Interpretation (when bit is set)
1	K	Keep this job in memory; do not roll it out.
2†	A	Abort pending; reason given in TXEPC.
3†	F	Suspended to single-thread tasks
4†	H	Suspended by user deactivate
5	C	Forced memory allocation pending
6	G	Job class invoke pending
7	B	Suspended (indefinitely) by recovery
8†	E	Suspended until a given event occurs
9†	I	Dormant, pending recall on I/O completion
10	M	Memory allocation is pending
11	O	Suspended (indefinitely) by operator
12†	S	Suspended by system
13†	T	Suspended for a given time interval
14	U	User roll request pending
15	V	Waiting on rolled job index write completion
16	Y	Waiting for I/O quiet
17†	D	Delete request in progress
18	L	Roll image load/unload in progress
19	N	Not in memory

† This bit is a task status bit, and only set in TXSTAT.

Table 9-2. Status bit assignments (continued)

Bit position in JXSTAT	Bit name	Interpretation (when bit is set)
20	Q	Queued up; waiting to be initiated.
21	R	Rolled out. The M bit can also be set.
22†	X	Executing

† This bit is a task status bit, and only set in TXSTAT.

Figure 9-22 and table 9-3 illustrate some transitions that normally occur between job and task statuses. Job status changes are shown with =>. Task status changes are shown with ->.

Table 9-3. Status-change sequences

Sequence	Explanation
X->I	A task requests recall on I/O completion.
X->W	A task's time slice expires.
W->X	A task is given another time slice.
X->W W=>LN=>RN	A job's tasks are disconnected, and the job is rolled out.
RN=>LN=>W W->X	A job is rolled in, and its task is connected.
X->W W=>MN	A task is disconnected, and the job is queued up for a memory request.
X->W W=>LN=>RN=>NRO	The job's tasks are disconnected as the job is suspended by operator.

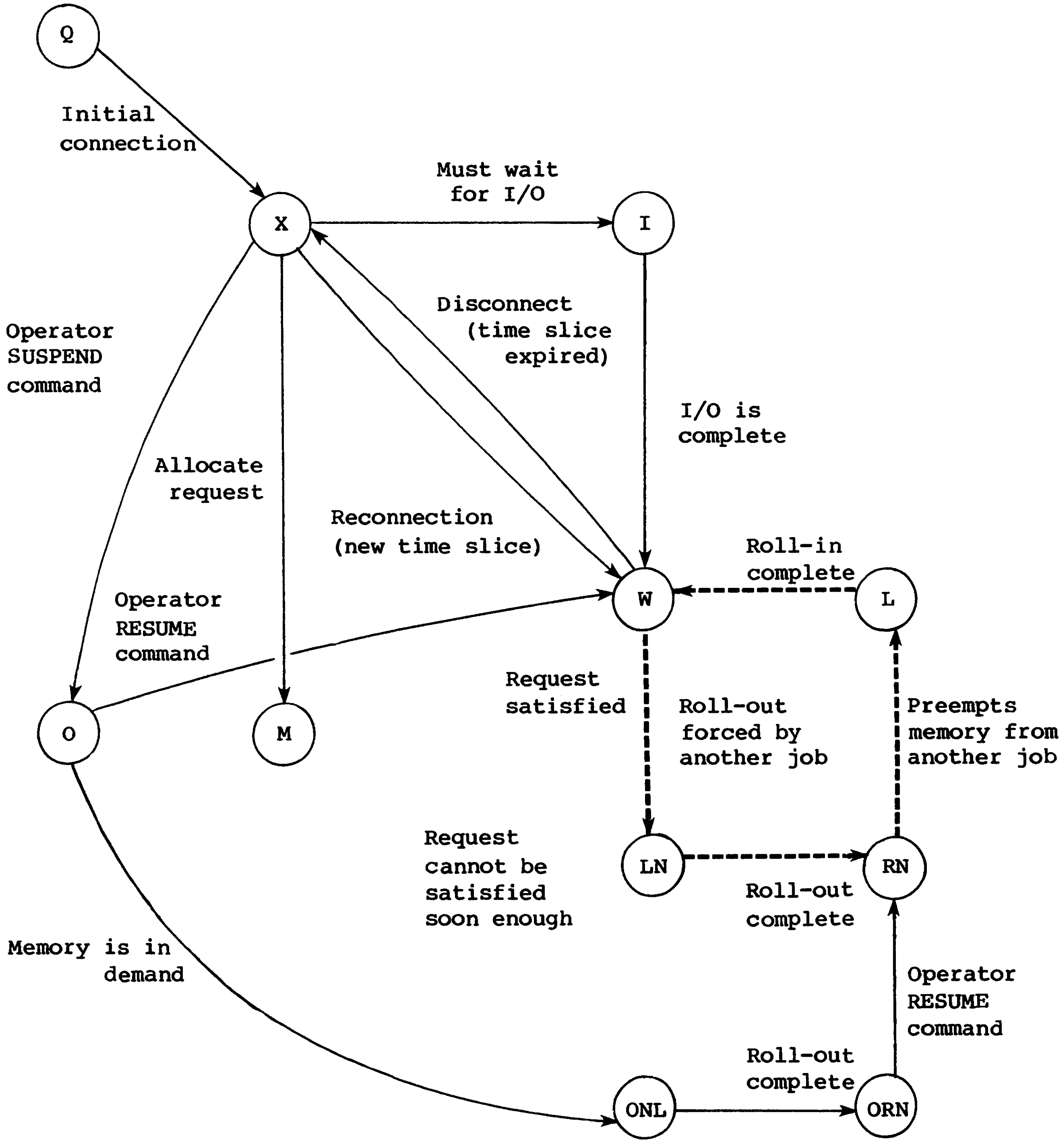


Figure 9-22. Normal transitions between job states

9.8.1 STATUS CHANGES INVOLVED IN CPU SWAPPING

Figure 9-22 shows most of the status changes that can occur for any particular user task. Those status changes shown with broad solid lines are the basic changes that all user tasks must undergo. If all the jobs in the JXT can fit into memory at the same time, these status changes are the only status changes jobs undergo as long as they make no memory requests and open no auxiliary datasets.

QN->X A job queued in the JXT (Job Execution Table) waiting for sufficient memory to become available is given its first CPU time slice to its first task. (The user task actually exists momentarily in the W status before it begins executing; but because the new user task's CPU priority is initialized to the highest possible value, the transition to status X is immediate.)

X->I The currently executing user task becomes dormant by requesting suspension pending the completion of a particular I/O transfer. The CPU becomes available for use by another user task.

I->W The I/O transfer for which suspension is requested is now complete. The user task joins others waiting for CPU time.

X->W The executing user task's time slice expires. Unless it is the only user task eligible for connecting, it is disconnected from the CPU and joins any other user tasks that are waiting.

W->X The CPU has just become available. JSH selects the first user task from the top of the CPU queue which is not suspended for I/O.

9.8.2 STATUS CHANGES INVOLVED IN MEMORY SWAPPING

In figure 9-22, the paths involved in rolling jobs in and out are shown as dashed lines.

Any job with tasks in W status is liable to be rolled out if a job of higher priority is waiting and the job's in-memory thrash lock has expired.

W=>LN A roll-out I/O request is initiated for a waiting job in order to make memory available.

LN=>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's tasks begin contending for CPU time.

9.8.3 STATUS CHANGES INVOLVED IN JOB SUSPENSION AND RESUMPTION

In figure 9-22, the suspended statuses are connected to the rest with narrow solid lines.

A job's tasks are momentarily suspended when one makes an allocation request (J$ALLOC). Shortly after it suspends the tasks, 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 also occurs as a result of an explicit user request to suspend processing 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->W
W=>M The currently executing user task makes an allocation request and is considered dormant until the request is 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 is rolled out.

MN=>W The allocation request is satisfied before the job is rolled out. The job's tasks join any other user tasks waiting for CPU time.

MN=>LN The job is rolled out because memory is in demand. More space is required to satisfy the allocation request than is obtained merely by reallocating memory.

X->T or X->E The currently executing user task or EXP makes a suspension request (J$DELAY or J$AWAIT). The user task is disconnected and maybe rolled out.

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.

W=>LN A roll-out I/O request to make memory available is inititated for a suspended job.

LN=>RN The roll-out I/O request is complete; the job's memory is released.

ORN=>RN A job suspended by the operator (and subsequently rolled-out) is reactivated by the operator. The job is rolled back in when memory is available.

RN=>RN A job suspended by the system (and subsequently rolled-out) is reactivated because the system suspension was lifted. The job is rolled back in when memory is available.

S->W A user task suspended by the system is reactivated while still in memory.

9.9 JSH INTERFACE WITH OTHER TASKS

The Job Scheduler task is created with all other system tasks by the startup procedure. It is then 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 waits until memory is available or until an I/O transfer is complete before it replies, so that the requesting task proceeds 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 TXT ordinal identifying the task, and additional information supplied by the requesting task. A status indicator is returned in the first output register.

Input register format:

```
           0                       24                      48   53        63
INPUT+0   |          AUX            |     CODE or ADDR      | FC |   TXO    |
INPUT+1   |                     unused                           |   JXO    |
```

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$ABORT request only) Abort code; use equated labels of the form A$*xxxxx*, where *xxxxx* = DROP, KILL, RERUN, or other predefined abort code. These abort codes are also used in the J$UROLL request.
ADDR	INPUT+0	24-47	Word address relative to the beginning of STP of an additional word or list of words if needed to fully specify the call. See the following individual function descriptions for more detail.
FC	INPUT+0	48-52	Function code; use equated labels of the form J$*xxxx*, selected from table 9-4.
TXO	INPUT+0	53-63	TXT ordinal for the job in question
JXO	INPUT+1	53-63	JXT ordinal for the job in question. It can assume a value from 1 to I@JXTSIZ.

Output register format:

```
           0                       24                      48   53        63
OUTPUT+0  |                          STATUS                                 |
OUTPUT+1  |          AUX            |     CODE or ADDR      | FC |   TXO    |
```

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$UROLL request.
ADDR	OUTPUT+1	24-47	Word address relative to the beginning of STP of an additional word or list of words if needed to fully specify the call. See the following individual function descriptions for more detail.
FC	OUTPUT+1	48-52	Function code; use equated labels of the form J$*xxxx*, selected from table 9-4.
TXO	OUTPUT+1	53-63	TXT ordinal for the user task in question

9.9.1 CALLING SEQUENCE

JSH is 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	user task's TXT ordinal
	S1	S1!S2
	S2	address if any

Location	Result	Operand
	S2	S2<D'16
	S1	S1!S2
	S2	auxiliary information if any
	S2	S2<D'40
	S1	S1!S2
	S2	job's JXT ordinal
	R	TSKREQ or PUTREQ

The TSKREQ subroutine enforces synchronous task behavior; that is, it does not return to its caller until JSH replies. PUTREQ returns as soon as it stores 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 9-4.

Table 9-4. JSH functions

Function Code	Function Value	Input Parameters	Function
-- no input required --			Fills up JXT with jobs from SDT
J$ALLOC	4000	JXO,ADDR	Changes a job's memory allocation
J$AWAIT	10000	JXO,ADDR, TXO	Suspends a user task until a given event occurs
J$DELAY	14000	JXO,ADDR, TXO	Suspends a user task for a given time
J$SUSP	20000	JXO,TXO	Suspends a user task momentarily; system initiated.
J$SUSPK	24000	JXO,TXO	Same as J$SUSP but keeps the job in memory
J$STOP	30000	JXO	Suspends a job indefinitely; operator action (SUSPEND).

Table 9-4. JSH functions (continued)

Function Code	Function Value	Input Parameters	Function
J$CLEAR	34000	JXO,TXO	Forces the end of a user task's suspension (except I/O suspend)
J$ABORT	40000	JXO,CODE, TXO	Aborts a user task
J$RERUN	44000	JXO	Same as J$DELETE but places a job back in the input queue
J$DELETE	50000	JXO,TXO	Deletes a user task, and if this is the last task for the job, releases all space allocated to a job
J$IOSUSP	54000	JXO,TXO	Suspends a user task until an I/O request receives a response
J$IODONE	60000	JXO,TXO	Resumes an I/O suspended user task
J$RESUME	64000	JXO,TXO	Ends suspension set by J$SUSP or J$SUSPK
J$START	70000	JXO	Ends an indefinite suspension for a job; operator action (RESUME).
J$INDEX	74000	JXO	Marks the job irrecoverable
J$STRALL	100000		Ends an indefinite suspension for all jobs; operator action (RESUME ALL).
J$STPALL	104000		Suspends all jobs indefinitely; operator action (SUSPEND ALL).
J$RCVR	110000		Lifts the suspension from jobs suspended by a J$SHTDWN or system interruption
J$SHTDWN	114000		Idles down all job activity in preparation for a system interruption
J$REMK	120000	JXO,TXO	Lifts the keep-in-memory restriction

Table 9-4. JSH functions (continued)

Function Code	Function Value	Input Parameters	Function
J$INVOKE	124000	JXO,ADDR	Invokes a job class structure
J$UROLL	130000	JXO,CODE	Rolls a job; user requested to protect against system interruption.
J$CHANGP	134000	JXO,ADDR	Change priority - operator request
J$READY	140000	JXO	Force memory allocation for job
J$GETM	144000	CODE	Get memory from jobs
J$RETM	150000	CODE	Return memory to jobs
J$TINIT	154000	TXO,JXO	Initialize a new user task.
J$ACT	160000	TXO,JXO	Activate a user task which has been deactivated through a J$DEACT request. Requested by user.
J$DEACT	164000	TXO,JXO	Deactivate a user task. Requested by user.
J$SINGLE	170000	TXO,JXO	Single thread request. Used by EXP to force only one task in a job to be connected. Allows EXP to handle reprieve processing cleanly.
J$DEADLK	174000	TXO,JXO	Signals a hardware semaphore deadlock situation.

9.9.2 INITIALIZE REQUEST

The initialize request transfers as many jobs as possible from the input queue to the executing queue.

FUNCTION CODE: None

FUNCTION VALUE: 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 no more jobs can be initiated in available classes. 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 can be scattered throughout the table. Empty JXT entries are similarly 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.

9.9.3 ALLOCATE REQUEST

The allocate request changes a job's memory allocation.

FUNCTION CODE: J$ALLOC

FUNCTION VALUE: 4000

ENTRY: FC, JXO, and ADDR are required. ADDR is the STP-relative address of a memory request word having the following format:

```
0       8        16                      40                     63
| Flags |///////|          DEL           |           WC          |
```

Expansion of flag area:

```
 0    1    2    3    4    5    6    7    8
|////////////////////| B  | D  |////| F  | J  |
```

Field	Bits	Description
B	4	Buffer flag. B is ignored if F or J=1. If B=1, WC specifies the number of words to be added to the buffer area. If B=1, DEL, and D are ignored.
D	5	DSP flag. D is ignored if J, F, or B=1 or DEL is nonzero. If D=1, an installation-defined DSP increment is made at the end of the DSP area. If D=1, WC is ignored.
F	7	Field Length flag. F is ignored if J=1. If F=1, WC specifies the number of words of field length that is to be allocated to the job. If F=1, B, D, and DEL are ignored.
J	8	JTA flag. If J=1, WC specifies the number of words to be added to the end of the JTA. If J=1, F, B, DEL, and D are ignored.
DEL	16-39	Deletion pointer/flag. DEL is ignored if J, B, or F=1. If DEL≠0, WC specifies the number of words that are to be deleted from the buffer area and DEL specifies the address relative to the user's base address of the beginning of the area to be deleted. If DEL≠0, D is ignored.
WC	40-63	Word count. WC is ignored if D=1. If F=1, WC specifies the number of words of field length that is to be allocated to the job. If B=1, WC specifies the number of words to be added to the buffer area. If DEL≠0, WC specifies the number of words to be subtracted from the buffer area. If J=1, WC specifies the number of words to be allocated to the JTA. If F, B, D, J, and DEL are 0, WC specifies the number of words to be added to (if WC is positive) or subtracted from (if WC is negative) the user code/data area.

EXIT: 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 a copy of the first input register.

EXIT:
(continued)

The output registers are set and the caller is readied immediately after the request is received; the job can be aborted by returning an appropriate error status to the calling task.

The JCB also contains values (FL, MFL) that must be maintained by J$ALLOC. The current field length must be changed when the field length changes. The maximum field length allowed can be decreased when the JTA is expanded.

The job is aborted if filling the request would result in a field length greater than the maximum allowed the job. The maximum is the smaller of the system maximum or the amount determined by the MFL parameter on the Job statement. The system maximum is the smaller of the total number of words available to user jobs minus the job's JTA or the amount determined by I@JFLMAX.

DESCRIPTION:

JSH begins processing the memory request by placing the job in state M and disconnecting all of its tasks from the CPU. A J$ALLOC request can be made to expand the JTA for a job whose tasks are disconnected from the CPU but still doing I/O.

The memory request is honored only after all the job's outstanding I/O has completed. In the meantime, the job's tasks are suspended. When the I/O activity is quiet, the job can be expanded, contracted, moved, or rolled out.

A request for less space is honored immediately. A request for more space is honored when there is enough memory available. Jobs with status M and are waiting for more memory can be rolled out. When they are rolled in again, they are rolled into an area large enough to satisfy the expansion request. If the job's memory is expanded, the new memory is initialized to an installation defined value.

The JCB has pointers (HLM, LFT, DSP, BFB, FL, MFL) that must be maintained by J$ALLOC. Expansion or compression of any area other than the JTA, always requires the adjustment of at least one of these pointers. The LFT and DSP areas contain pointers that must also be adjusted. The pointers in the LFT area are updated whenever the DSP pointer is changed. The pointers in the DSP are updated when the buffers

DESCRIPTION: (continued) move. Expansion of the JTA can result in a decrease in MFL if MFL plus the new JTA size exceeds all available memory.

9.9.4 AWAIT REQUEST

The await request suspends execution of a user task until a given event occurs.

FUNCTION CODE: J$AWAIT

FUNCTION VALUE: 10000

ENTRY: FC, TXO, JXO, and ADDR are required. ADDR is the STP-relative address of an event word with the following format:

```
0 2   6                         32                48                 63
┌─────────────────────────────────────────────────────────────────────┐
│ ↑|///|           EPA            |      EMV       |       ECV        │
└─────────────────────────────────────────────────────────────────────┘
 ECC
```

Field	Bits	Description
ECC	0-2	Condition code (0-3)
EPA	6-31	STP-relative address of a parcel tested periodically
EMV	32-47	Mask value applied in the test
ECV	48-63	Comparison value used in the test

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: JSH disconnects the task from the CPU if necessary, sets the E bit in the status field, and stores the event word in the job's TXT entry.

If the task's E bit is set, the parcel at the given address is periodically ANDed with the mask and then

DESCRIPTION: (continued) subtracted from the comparison value. The event is said to occur when the result matches the condition code.

Condition code	Matching result
0	Zero
1	Nonzero
2	Positive
3	Negative

A job with all tasks suspended is liable to be rolled out. When the event occurs, both the E bit is cleared and the suspension is lifted unless the S, B, or O bit is set.

9.9.5 DELAY REQUEST

The delay request suspends execution of a user task for a given number of milliseconds.

FUNCTION CODE: J$DELAY

FUNCTION VALUE: 14000

ENTRY: FC, TXO, JXO, and ADDR are required. ADDR is the STP-relative address of a delay request word having the following format:

```
0                                                   63
|                      DELAY                         |
```

where DELAY is the number of milliseconds 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 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 task from the CPU if necessary, sets the T bit in the status field, and stores the wake-up time in the TXDLY field in the task's TXT entry.

If the task's T bit is set, the TXDLY 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. The condition is also unconditionally lifted during system startup.

A job can be rolled out while its tasks are suspended. It enters into the normal memory swapping activity after any of its tasks are reactivated.

Lifting the suspension involves clearing the T bit. However, the task remains suspended if either the S (J$SUSP), B, or O bit is set. Only a J$RESUME request clears the S bit. Only a J$START or a J$STRALL request clears the O bit. Only a J$RCVR, a J$START, or a J$STRALL clears the B bit.

9.9.6 SUSPEND REQUEST

The suspend request suspends execution of a job momentarily.

FUNCTION CODE: J$SUSP or J$SUSPK

FUNCTION VALUE: 20000 or 24000

ENTRY: FC, TXO, 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: JSH disconnects the task 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 are reset only by an event occurring or the elapsing of a delay time.

DESCRIPTION: (continued)
The K (keep) bit, if set in any task, prevents the job from being rolled out while the task is suspended. A J$RESUME request clears the S and K bits allowing the job to participate in normal rollout activity.

9.9.7 STOP REQUEST

The stop request suspends execution of a job indefinitely as a result of operator action.

FUNCTION CODE: J$STOP

FUNCTION VALUE: 30000

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 job's O (operator suspended) bit is set. The job is rolled out as soon as possible.

9.9.8 CLEAR REQUEST

The clear request forces the end of a task's suspension, no matter why the suspension is imposed other than I/O suspension.

FUNCTION CODE: J$CLEAR

FUNCTION VALUE: 34000

ENTRY: FC, TXO, 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: This call is customarily made only during task termination and task abort.

JSH clears the job's O and B bits and the task's S, T, and E bits. The task is no longer suspended and begins to participate in the normal memory swapping activity unless its K bit is set or unless it is in an I/O suspend state.

9.9.9 ABORT REQUEST

The abort request aborts a user task.

FUNCTION CODE: J$ABORT (in actual coding, use J$ABORT+1S20*A$*xxxxx*, where *xxxx* = DROP, KILL, RERUN, or other predefined abort code.)

FUNCTION VALUE: 40000

ENTRY: FC, TXO, 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 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 disconnecting the task from the CPU if necessary, and placing it in A status. The abort code is stored in the TXT (TXEPC) to be picked up later by the Exchange Processor.

When TCEPJ (in the task's TCB) is 0, JSH sets TCEPJ so the Exchange Processor will abort the task when it is reconnected. JSH then removes the task's A status.

9.9.10 RERUN REQUEST

The rerun request releases all memory and datasets belonging to a given job except its System Dataset Table (SDT) entry and moves the SDT from the executing queue back to the input queue so that it is reinitiated.

FUNCTION CODE: J$RERUN

FUNCTION VALUE: 44000

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: This call is the last action taken in the terminating of a job that is to be rerun. JSH disconnects all of the job's tasks 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.

9.9.11 DELETE REQUEST

The delete request deletes a user task and releases all memory belonging to a given job if this is the last task in the job.

FUNCTION CODE: J$DELETE

FUNCTION VALUE: 50000

ENTRY: FC, TXO, 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: This call is the last action in the terminating of a task. JSH disconnects the task from the CPU if necessary. If this is the last task in the job and all pending I/O is complete, JSH frees all memory assigned to the job (the user area, the JXT, and the empty SDT entry).

9.9.12 I/O-SUSPEND REQUEST

The I/O-suspend request suspends execution of a task until an I/O-resume request is made for the same task.

FUNCTION CODE:	J$IOSUSP
FUNCTION VALUE:	54000
ENTRY:	FC, TXO, 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:	JSH disconnects the task from the CPU if necessary, and sets the I bit in its status field. The task does not execute until the I/O-suspend is rescinded.

9.9.13 I/O-RESUME REQUEST

The I/O-resume request reactivates an I/O-suspended task.

FUNCTION CODE:	J$IODONE
FUNCTION VALUE:	60000
ENTRY:	FC, TXO, 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:	JSH clears the I bit in the task's status field, changing the task's status to W unless the task is suspended for other reasons.

9.9.14 RESUME REQUEST

The resume request lifts a suspension imposed on a task by J$SUSP or J$SUSPK.

FUNCTION CODE: J$RESUME

FUNCTION VALUE: 64000

ENTRY: FC, TXO, 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: This call is made to restart a task previously suspended by a call to J$SUSP or J$SUSPK, but it has no immediate effect if the task is still under suspension for another reason.

JSH clears the S and K bits in the tasks status field and the K bit in the job's status field if no other tasks have the K bit set. I the B, O, and K bits are now clear, the job can now particpate in the normal memory swapping activity.

9.9.15 START REQUEST

The start request lifts the indefinite suspension from a job suspended by J$STOP, J$STPALL, J$SHTDWN or a system interruption.

FUNCTION CODE: J$START

FUNCTION VALUE: 70000

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: This call is made to restart a job suspended by J$STOP or J$STPALL. It is also 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.

9.9.16 INDEX REQUEST

The index request marks a job irrecoverable.

FUNCTION CODE: J$INDEX

FUNCTION VALUE: 74000

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 is nonrecoverable from the roll image.

JSH checks the job's roll index entry to see if it is already marked irrecoverable. If it is already marked, JSH does nothing. If it is not marked, JSH sets the irrecoverable 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.

9.9.17 START ALL REQUEST

The start all request lifts the indefinite suspension from all jobs suspended by J$STOP, J$STPALL, J$SHTDWN or a system interruption.

FUNCTION CODE: J$STRALL

FUNCTION VALUE: 100000

ENTRY: FC is 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: J$START is applied to all jobs in the JXT.

9.9.18 STOP ALL REQUEST

The stop all request suspends processing of all jobs in the JXT, rolls them out, and releases their memory.

FUNCTION CODE: J$STPALL

FUNCTION VALUE: 104000

ENTRY: FC is 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 JXT limit (JXTMAX) is set to 0. All jobs are suspended. (Refer to J$STOP.)

9.9.19 RECOVER REQUEST

The recover request recovers all jobs in the system.

FUNCTION CODE: J$RCVR

FUNCTION VALUE: 110000

ENTRY: FC is 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: J$RCVR or J$START must be used after a system interruption or 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 for another reason.

9.9.20 SHUTDOWN REQUEST

The shutdown request shuts down the system; it is normally used to prepare for an expected system interruption. Job activity is idled down, all jobs are rolled out and their memory released. Station activity is not affected.

FUNCTION CODE: J$SHTDWN

FUNCTION VALUE: 114000

ENTRY: FC is 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 B (suspended by recovery) bit is set for all jobs. The jobs are rolled out as soon as possible. The JXT limit (JXTMAX) is set to 0.

9.9.21 REMOVE K REQUEST

The remove K request lifts the keep-in-memory (K) restriction.

FUNCTION CODE: J$REMK

FUNCTION VALUE: 120000

ENTRY: FC, TXO, 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 for the indicated user task. If the K bit is now clear forall tasks in the job, the K bit is cleared in the job's status and the job may begin to participate in normal memory swapping.

NOTE

The K bit must be set when the J$REMK request is made.

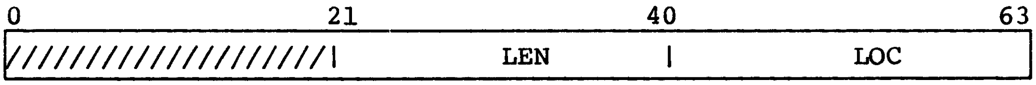
9.9.22 INVOKE REQUEST

The invoke request invokes a job class structure.

FUNCTION CODE: J$INVOKE

FUNCTION VALUE: 124000

ENTRY: FC, TXO, JXO, and ADDR are required. ADDR is the STP-relative address of an invoke request word with the following format:



Field	Bits	Description
LEN	21-39	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	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.

EXIT: (continued) The output registers are set, and the caller is readied immediately after the request is received.

DESCRIPTION: 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. This request is only valid from jobs with a single task.

9.9.23 USER ROLL REQUEST

The user roll request rolls a job out; the function is user requested to protect against system interruptions.

FUNCTION CODE: J$UROLL

FUNCTION VALUE: 130000

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: JSH begins processing the request by placing the job's tasks in S (suspended) state, disconnecting them 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 again participates in the normal memory swapping activity.

9.9.24 CHANGE PRIORITY REQUEST

The change priority request is used by operator command to change the priority of a job.

FUNCTION CODE: J$CHANGP

FUNCTION VALUE: 134000

ENTRY: JXO and CODE are required.

EXIT: The first output register is a status word which is set to 0 if the priority is changed. The status word is set to JR$REJC if the job is in a state where the priority is not changed immediately. 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: If the request is processed, the low-order 4 bits of JFADP, the JFADR field are placed into JXP as an integer and into JXFMP as a floating-point value.

9.9.25 FORCE JOB INTO MEMORY REQUEST

The force job into memory request readies the calling task when the specified job is in memory.

FUNCTION CODE: J$READY

FUNCTION VALUE: 140000

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 on are set and the caller is readied immediately after the request is received.

DESCRIPTION: JSH begins by determining if the job is already in memory. If so, an error response is returned. If not, a flag corresponding to the calling task in the JXRDY field of the JXT is set along with the C status bit. The job is then scheduled for memory allocation regardless of its state. When the job is placed in memory, each task with a flag set in JXRDY is made ready. The C status bit and all JXRDY flags are cleared after processing is complete.

9.9.26 GET MEMORY REQUEST

The get memory request gets memory from jobs.

FUNCTION CODE: J$GETM

FUNCTION VALUE: 144000

ENTRY: FC and CODE are required. CODE occupies the same position as ADDR does in other requests and is the number of words of memory that is needed.

EXIT: The first output register is a status word normally set to 0; any nonzero value is an error response. 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: This call is made by STG or SCP to get memory that is normally used by jobs. The memory allocated to the system is increased by the amount specified in CODE as soon as the space can be made available by rolling jobs out and compacting memory. The new memory allocated to the system is at the low address end of the system's memory segment.

9.9.27 RETURN MEMORY REQUEST

The return memory request returns memory back to jobs.

FUNCTION CODE: J$RETM

FUNCTION VALUE: 150000

ENTRY: FC and CODE are required. CODE occupies the same position as ADDR does in other requests and is the number of words of memory that is being returned.

EXIT: The first output register is a status word normally set to 0; any nonzero value is an error response. 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: This call is made by STG or SCP to return memory to user jobs. The memory allocated to the system is reduced by the amount specified in CODE (starting at the low-address end of the system's memory segment) immediately.

9.9.28 INITIALIZE USER TASK REQUEST

The initialize user task request tells JSH to begin scheduling a new user task.

FUNCTION CODE: J$TINIT

FUNCTION VALUE: 154000

ENTRY: FC, TXO, and JXO are required.

EXIT: The first output reqister is a status word normally set to 0; any nonzero value is an error response. 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.

DESCRIPTION: This request is made by EXP when a user requests a new task. EXP has already allocated a TXT and TCB.

9.9.29 ACTIVATE USER TASK REQUEST

The activate user task request tells JSH to resume scheduling a user task which has been idled by a deactivate request.

FUNCTION CODE: J$ACT

FUNCTION VALUE: 160000

ENTRY: FC, TXO, and JXO are required.

EXIT: The first output register is a status word normally set to 0; any nonzero value is an error response. The second output register is a copy of the first input register.

EXIT: (continued) The output registers are set and the caller is readied immediately after the request.

DESCRIPTION: This request is made by EXP when a user requests a task activate. The J%H suspend status is cleared if set.

9.9.30 DEACTIVATE USER TASK REQUEST

The deactivate user task request tells JSH to make a user task ineligible for CPU scheduling.

FUNCTION CODE: J$DEACT

FUNCTION VALUE: 164000

ENTRY: FC, TXO, and JXO are required.

EXIT: The first output register is a status word normally set to 0; any nonzero value is an error response. 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.

DESCRIPTION: This request is made by EXP when a user requests a task deactivate. JSH sets the J%H suspend condition for the task. If all tasks in the job have been deactivated, this task is aborted.

9.9.31 SINGLE-THREAD USER TASKS REQUEST

EXP uses the single-thread user tasks request to remove all but one user task from CPU scheduling. This request is used primarily for reprieve processing.

FUNCTION CODE: J$SINGLE

FUNCTION VALUE: 170000

ENTRY: FC, JXO are required. If TXO is supplied, all other tasks in the job are suspended. If TXO is 0, this is a resume from single-thread request.

EXIT: The first output register is a status word normally set to 0; any nonzero value is an error response. The second output register is a copy fo the first input register.

The output registers are set and the caller is readied immediately after the request.

DESCRIPTION: This request is made by EXP when reprieve processing is necessary. It allows EXP to manipulate the job without interference from tasks for the same job in other CPUs.

9.9.32 PROCESS USER TASK DEADLOCK REQUEST

The process user task deadlock request tells JSH to analyze for a possible deadlock situation in the job.

FUNCTION CODE: J$DEADLK

FUNCTION VALUE: 174000

ENTRY: FC, TXO, and JXO are required. This request cannot be processed on CRAY-1 machines because it requires hardware semaphores.

EXIT: The first output register is a status word normally set to 0; any nonzero value is an error response. The second output register is a copy of th first input register.

The output registers are set and the caller is readied immediately after the request.

DESCRIPTION: JSH receives a deadlock request for a user task from EXP when EXEC has detected a hardware deadlock. The deadlock is detected when a user task is waiting with a test and set instruction, but no other CPU is in the same cluster. On receiving the request, JSH moves the user task from the CPU queue to a new queue, the deadlock queue (DLKQ). JSH aborts the user task if all user tasks are now waiting for semaphores that are still set or for deactivated user tasks. (EXP has recorded the number of the semaphore each task is waiting for.) When connecting user tasks, JSH checks the DLKQ and removes any tasks for the same job that are waiting for semaphores that subsequently cleared.

PERMANENT DATASET MANAGER (PDM) 10

The Permanent Dataset Manager task (PDM) provides a means of creating, accessing, deleting, maintaining, and auditing disk-resident permanent datasets.

Permanent datasets are of two types: user permanent datasets, created through a user request, and system permanent datasets, created by the system for spooled input and output datasets.

Each type of dataset can have 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 as a result of one or more DISPOSE statements. It returns to single-DSC-entry status when all related disposes have completed.

PDM coordinates these activities through the Queued Dataset Table (QDT). A temporary dataset also can 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
- PERMIT   Grants explicit permission to access a dataset

The system can 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 can 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 Dataset Catalog (DSC) copy of the job input System Dataset Table (SDT). Used by the Exchange Processor to declare a job ineligible for rerun and to indicate the job was initiated at least once.

10.1 FUNCTIONS

A task calls the Permanent Dataset Manager by placing a Permanent Dataset Definition Table (PDD) pointer and possibly a return address in its INPUT+0 and a Job Table Area (JTA) and/or Dataset Allocation Table (DAT) or Dataset Name Table (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. A return status is always returned in field ST of the PDD. The return status is described in section 10.2 of this manual.

Calling sequence:

	0 8	16 24 32	40 48 56 63
INPUT+0	////////////////	Return	PDD
INPUT+1	|/ SYS //////	DNT or DAT	JTA

Field	Word	Bits	Description
Return	INPUT+0	16-39	A 24-bit value that remains unchanged and is normally used as a return address
PDD	INPUT+0	40-63	Base address of the PDD relative to STP
SYS	INPUT+1	0	If set, this flag identifies the call as having been initiated by the system.
DNT	INPUT+1	16-39	Dataset Name Table address, if user (job) call

Field	Word	Bits	Description
DAT	INPUT+1	16-39	Dataset Allocation Table address, if system call
JTA	INPUT+1	40-63	Base address of the associated job's JTA. If the SYS flag is not set, the JTA must be specified.

The function codes processed by the task follow:

Code	Description
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	Dataset Catalog (DSC) page request
PMFCPX=41_8	Dataset Catalog Extension Table (DXT) page request
PMFCLU=50_8	Load user dataset
PMFCLI=52_8	Load input dataset
PMFCLO=54_8	Load output dataset
PMFCRL=60_8	Update Active Permanent Dataset Table (PDS)/Release request
PMFCPN=70_8	Permanent dataset name (PDN) request
PMFCDT=100_8	Dump time request
PMFCDQ=110_8	Dequeue System Dataset Table (SDT) entry
PMFCEA=120_8	Queue System Dataset Table (SDT) entry to available queue
PMFCEI=122_8	Queue System Dataset Table (SDT) entry to input queue
PMFCEO=124_8	Queue System Dataset Table (SDT) entry to output queue
PMFCAD=130_8	Adjust user dataset
PMFCMD=140_8	Modify user dataset
PMFCRSDT=150_8	Rewrite job's input System Dataset Table (SDT) entry
PMFCPSAC=160_8	Pseudo access for Rolled Job Recovery (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
PMFCPE=200_8	Permit alternate user dataset access

10.1.1 SAVE USER DATASET PROCESSING (FUNCTION CODE 10)

A local dataset can be registered in the Dataset Catalog (DSC) by a user issuing a SAVE command. By default, the user registering the dataset becomes its owner. All access permissions (execute-only, read, write and maintenance) are available to the owner as long as correct control words are specified. The dataset is uniquely identified in the DSC by recording the PDN, ID, ED and ownership value. Besides these attributes, other attributes can be registered such as public access permissions, and specific user access permissions (permits). Note that PERMIT-specified access permissions take precedence over permissions granted to the public-at-large. Spooled datasets always belong to COS (the system).

Whenever the system requests PDM to determine the existence of a permanent dataset it must supply the ownership value in the PDD; otherwise, PDM assumes the dataset belongs to the system and consequently uses the system's ownership value (I@SYSOWN).

The following PDD fields are used as input: PDN, ID, ED, TXT, ADN, NOTE, PAM, TRA, RD, WT, MN, USR (input datasets only), RT, EXO, IA, FM, OJB, JSQ, DC, DID, SID, TID, SF, PR, TL, MFL, JCN, CL, SYS, JSP, IJSP, JCR, OLM JST, MML, RGX, OJSQ, WAIT, TXL, SSC, NOTL. In general, PDM saves anything in the PDD for all types of saves (except the station slot for user-requested saves).

10.1.2 SAVE INPUT OR OUTPUT DATASET PROCESSING (FUNCTION CODES 12, 14)

Save input or output dataset processing uses the same PDD fields as save user dataset processing.

The following PDD fields are returned: ED, FPE.

10.1.3 ACCESS PROCESSING (FUNCTION CODES 20, 26)

The ACCESS function attempts to access a dataset whose characteristics are defined in the supplied PDD. The ACCESS function is also used by the permanent dataset name (PDN) request and the PDSDUMP ACCESS request.

A permanent dataset belonging to a user is uniquely identified by matching on PDN, ID, ED and ownership value. If no match is found, the ACCESS request is aborted. If a match is found and one or more control words (CWs) are specified, the specified CWs must also match; otherwise, the ACCESS request is aborted. For each nonzero CW recorded in the Dataset Catalog (DSC) entry that is not specified in the ACCESS command, the corresponding permission is denied to the user making the request.

The following Permanent Dataset Definition Table (PDD) fields are used as input:

Field	Contents
DN	Local (temporary) dataset name applicable only to user calls
PDN	Permanent dataset name
ID	User identifier for the dataset
ED	Edition number; if this is 0, the highest edition is accessed.
JSQ	Job sequence number, used instead of ED when accessing spooled datasets.
RD	Read control word for read permission
WT	Write control word for write permission
MN	Maintenance control word for maintenance permission
UQ	Unique access request. This is required if write and/or maintenance permission is desired.
IR	Immediate return on delayed accesses
TXT	Base address of the area to receive text. If this is 0, no text is transferred. If this number is negative, the address has been adjusted to be STP-relative. If this number is positive, the address is to be relocated if an associated job exists.

Return sequence:

All input values remain unchanged unless the call is a system call, in which case the following fields are changed:

- The SYS flag is zeroed.
- The DNT field is zeroed.
- The JTA field contains the base address of the first DAT body for the accessed dataset.

The PDD fields returned are as follows:

Field	Contents
ED	Edition number accessed
JSQ	Job sequence number of dataset accessed (spooled datasets)
FPE	DSC page/entry of dataset

10.1.4 DELETE PROCESSING (FUNCTION CODES 30, 36)

Delete processing handles the DELETE control statement, DELETE macro, the PDSDUMP delete option, and the delete spooled dataset request. When a dataset is registered in the Dataset Catalog (DSC), a user authorized for the maintenance of the dataset can delete the dataset from the DSC. Before issuing the DELETE command, the user must have previously accessed the dataset uniquely with maintenance permission granted. The DELETE command can be used to deallocate the disk space occupied by the dataset while retaining the DSC and Dataset Catalog Extension Table (DXT) entries (a partial delete). Input consists of a Permanent Dataset Definition Table (PDD).

The following PDD fields are used on input: DN, PDE.

10.1.5 PAGE REQUEST PROCESSING (FUNCTION CODES 40 and 41)

AUDIT and PDSDUMP use the page request function to determine the permanent datasets in the Dataset Catalog. Page request uses format 5 of the Permanent Dataset Definition Table (PDD). Function code 40 reads pages from the Dataset Catalog (DSC) and function code 41 reads pages from the Dataset Catalog Extension Table (DXT).

The input from PDD is as follows:

Field	Contents
BPG	Beginning page number
NPG	Number of pages to read
BUF	JCB-relative buffer address

The output to PDD is:

Field	Contents
NPG	Number of pages actually read
NHP	Number of hash pages in the DSC or DXT
NOP	Number of overflow pages in the DSC or DXT

10.1.6 LOAD PROCESSING (FUNCTION CODES 50, 52, 54)

The load processing request is used by PDSLOAD to reconstruct a Dataset Catalog (DSC) entry. Input consists of a Permanent Dataset Definition Table (PDD) in the load request format.

The following PDD fields are used on input: USR, ACN, OWN, DNS, ACS, CAT, ACT, TDM, MOD, and all fields used by subfunction 10, save user dataset processing (see section 10.1.1).

10.1.7 PDS/RELEASE PROCESSING (FUNCTION CODE 60)

Active Permanent Dataset Table (PDS)/Release processing handles the updating of the PDS table when a user permanent dataset is released.

The following PDD field is used on input: DN.

10.1.8 PDN REQUEST PROCESSING (FUNCTION CODE 70)

The permanent dataset name (PDN) request checks for the existence of a dataset with the characteristics defined by the PDD. The processing of this request is handled in large part by the ACCESS process. The input and returned values are identical to those of the ACCESS function.

10.1.9 DUMP TIME PROCESSING (FUNCTION CODE 100)

Dump time processing sets the dump time in the specified Dataset Catalog (DSC) entry to the current time and returns that time to the requester in a user specified buffer. Input is a Permanent Dataset Definition Table (PDD).

The following PDD fields are used on input: DN, BUF.

10.1.10 DEQUEUE SDT PROCESSING (FUNCTION CODE 110)

Dequeue System Dataset Table (SDT) processing removes the SDT entry from the input or output queue. Input is a Permanent Dataset Definition Table (PDD).

The following PDD fields are used on input: DN, PDN, JSQ.

The following PDD field is used on output: SDT.

10.1.11 QUEUE SDT PROCESSING (FUNCTION CODES 120, 122, 124)

Queue System Dataset Table (SDT) processing returns an existing SDT entry to the available, input, or output queue. Input is a Permanent Dataset Definition Table (PDD).

The following PDD field is used on input: SDT.

10.1.12 ADJUST PROCESSING (FUNCTION CODE 130)

The ADJUST processing modifies the size of an existing user permanent dataset in the Dataset Catalog (DSC). Input consists of a Permanent Dataset Definition Table (PDD).

The following PDD field is used on input: DN.

10.1.13 MODIFY PROCESSING (FUNCTION CODE 140)

Once a dataset has been registered in the Dataset Catalog (DSC), a user authorized for maintenance permission can modify its attributes. The MODIFY command allows the following attributes to be changed: PDN, ID, ED, R, W, M, RT, EXO, PAM, TA, TEXT, NOTES.

Before the MODIFY command is issued, the user must have accessed the dataset uniquely with maintenance permission granted. Input consists of a Permanent Dataset Definition Table (PDD).

The following PDD fields are used on input: DN, PDN, ED, ID, RD, WT, MN, PAM, TRA, RT, EXO, TXT, TXL, NOTE, NOTL.

10.1.14 SDT REWRITE PROCESSING (FUNCTION CODE 150)

The fixed portion of the input System Dataset Table (SDT) entry for the specified job is rewritten in the Dataset Catalog (DSC). This is used only by User Exchange Processor (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 Permanent Dataset Definition Table (PDD) and the Dataset Allocation Table (DAT) address for the SDT to be rewritten.

The following PDD fields are used on input: PDN, ID, USR, FM, RT, ED, OJB, SID, DID, DC, JSQ, TID SF, MFL, TL, PR, IJSP, RD, WT, MN, NRR, INIT.

10.1.15 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 relinked. Thus Active Permanent Dataset Table (PDS) entries must be built or updated and the Dataset Allocation Table (DAT) from the rolled image must be verified against the DAT in the Dataset Catalog (DSC). Input consists of a Permanent Dataset Definition Table (PDD) and Dataset Name Table (DNT).

The only PDD field used on input is the function code. On output, the status and dataset size (PMDSZ) fields are returned.

10.1.16 PDSDUMP ACCESS PROCESSING (FUNCTION CODES 170, 176)

The PDSDUMP ACCESS request is used solely by the PDSDUMP utility. This request disables updating of the last access time and the count of the number of accesses for the dataset it accesses. The processing is essentially that of the ACCESS function. The input values are identical to those of the ACCESS function, but the following additional PDD fields are optional:

Field	Contents
DTR	Flag indicating that the last-time-dumped field is to be updated during the access. Setting this flag makes issuing a dump time request unnecessary.
FPE	If this field is nonzero, it specifies the entry and the DSC page number of the dataset to access. If the dataset at the specified address does not match the dataset specified in the PDD, a normal scan of the DSC is performed.

10.1.17 PERMIT PROCESSING (FUNCTION CODE 200)

When a dataset is registered in the DSC, the owner can use the PERMIT function to create or change alternate user access permissions. A permit pertaining to the owner is ignored without comment. Permits apply to every edition of a dataset.

The following PDD fields are used on input: PDN, ED, OWN, PAM, ADN.

10.2 PDD STATUS

The return status is placed in the PMST field of the Permanent Dataset Definition Table (PDD) (table 10-1). The logfile contains a corresponding code and message for most of the status conditions.

Table 10-1. PDD status

Logfile Code	PMST	Status
	1	Complete; no error.
1	11	No DNT 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 local dataset name (DN) specified is already in use by the job
7	71	No permission granted
	101	Delay and try again
9	111	Requested dataset not in DSC

Table 10-1. PDD status (continued)

Logfile Code	PMST	Status
10	121	Edition does not exist
11	131	Active PDS full
12	141	Dataset not permanent
	151	Unused
14	161	Continuation error
15	171	DAT full
16	201	DNT full
	211	End of DSC
18	221	Specified permanent dataset already accessed by this job
	231	Request to read zero pages
	241	Invalid page number requested
21	251	No data has been written to disk
	261	SDT does not exist
	271	SDT entry not on input or output queue
	301	Unable to queue SDT entry
25	311	Dataset name in PDD is 0
26	321	Access control word validation error
27	331	Notes length exceeds maximum allowable value
28	341	Unique access is not acceptable because the dataset is part of the System Directory.

Table 10-1. PDD status (continued)

Logfile Code	PMST	Status
29	351	The text length is zero.
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 DAT in the DSC does not match the JTA DAT.
34	421	Access is denied because crossed allocation unit exists.
35	431	The dataset is already permanent.
36	441	The DSC entry was flagged by Startup as containing a fatal error; access is denied.
	451	The DSC or DXT page buffer supplied is outside the user field length.
	461	No available QDT entries exist.
	471	The dataset has outstanding disposes; do not deallocate disk space.
40	501	Allocation of multitype dataset inconsistent with related datasets.
41	511	Multitype dataset has nonexistent QDT entry.
42	521	Maximum edition reached

Table 10-1. PDD status (continued)

Logfile Code	PMST	Status
43	531	Dataset is on an active SDT queue.
44	541	Bad SDT address on Enqueue SDT request
45	551	Dataset is on a scratch device.
46	561	Access denied because of DXT error
47	571	Notes length is zero
48	601	Unused
49	611	Maximum number of DXT entries per dataset reached
50	621	Attributes dataset not local
51	631	Attributes dataset not permanent
52	641	Invalid notes buffer specified
53	651	Invalid text buffer specified
54	661	Specified permit entry not found
	671	Invalid DXT buffer address (get/link DXT)
	701	Bad DXT linkage pointer (get/link DXT)
	711	PMPDN and DCPDN do not match (get/link DXT)
	721	Unused
	731	PMSIZE greater than maximum PDD size
	2001	Parameter error (internal to $SYSLIB)
	2002- 2777	This range of status codes is reserved for magnetic tape support.

10.3 TABLES USED BY PDM

The following tables are used in permanent dataset management:

CSD	Class Structure Definition Table
DAT	Dataset Allocation Table
DNT	Dataset Name Table
DRT	Device Reservation Table
DSC	Dataset Catalog
DSP	Dataset Parameter Area
DXT	Dataset Catalog Extension
EQT	Equipment Table
JCB	Job Communication Block
JTA	Job Table Area
JXT	Job Execution Table
PDD	Permanent Dataset Definition Table
PDI	Permanent Dataset Information Table
PDS	Permanent Dataset Table
QDT	Queued Dataset Table
SDT	System Dataset Table
XAT	DXT Allocation Table

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

10.3.1 CLASS STRUCTURE DEFINITION TABLE (CSD)

The CSD is used by load input dataset processing.

10.3.2 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 Dataset Catalog. The DAT for the DSC is created when the Dataset Catalog DNT is created and is pointed to by the DSC DNT.

10.3.3 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.

10.3.4 DEVICE RESERVATION TABLE (DRT)

PDM updates the DRT to reflect the amount of space in use by permanent datasets.

10.3.5 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 Dataset Allocation Table (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.

For an input or output dataset, in addition to control information, the DSC can also contain user TEXT field information, station slot information, or both.

The page to which the permanent dataset is assigned is determined by hashing the permanent dataset name for a dataset created through a SAVE control statement or macro and hashing the dataset name for spooled datasets.

10.3.6 DATASET PARAMETER AREA (DSP)

A DSP for the Dataset Catalog (DSC) is assembled into the Permanent Dataset Manager task. It is used as input to Task I/O for reading and writing DSC pages.

10.3.7 DATASET CATALOG EXTENSION (DXT)

Many permanent dataset attributes will not fit in the Dataset Catalog itself, such as multiple PERMIT entries for a dataset. This additional information is kept in the DXT. Each main DSC entry has a chain of zero or more DXT entries associated with it.

The DXT is a permanent disk resident dataset and as such has an entry in the DSC. It is partitioned into 512-word pages, each containing a block control word, a 7-word header, and eight 63-word entries.

Each DXT entry belongs to a specific DSC main entry and contains a 3-word header containing the entry type (CRI or site), an In-use flag, a DXT entry ordinal, link information for the next DXT entry, and the main DSC page/entry number of the owning dataset.

The DSC main entry contains head and tail pointers for its associated DXT entries. The absence of such pointers indicates that no DXT entries exist.

10.3.8 EQUIPMENT TABLE (EQT)

PDM uses the EQT to locate the DRT.

10.3.9 JOB COMMUNICATION BLOCK (JCB)

Pointers in the JCB are used to validate buffer addresses passed by the user.

10.3.10 JOB TABLE AREA (JTA)

The Permanent Dataset Manager uses the dataset Dataset Name Table (DNT) and the user number from the JTA.

10.3.11 JOB EXECUTION TABLE (JXT)

The JXT is used when delaying a user requesting unique access to a dataset, and in validating buffer addresses passed by the user.

10.3.12 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 parameters necessary to perform the operation.

10.3.13 PERMANENT DATASET INFORMATION TABLE (PDI)

The PDI is set up by Startup and contains device label information pertinent to permanent dataset management (number of hash and overflow pages) and contains a pointer to the Dataset Name Table (DNT) for the Dataset Catalog (DSC).

10.3.14 PERMANENT DATASET TABLE (PDS)

An entry in the PDS exists for each active user permanent dataset. The PDS monitors user permanent datasets currently in use and controls dataset access by maintaining the number of users accessing the permanent dataset and controlling the waiting for access to a permanent dataset.

10.3.15 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).

10.3.16 SYSTEM DATASET TABLE (SDT)

PDM creates and queues an SDT entry when it loads a spooled dataset. The SDT entry information is gathered from the Permanent Dataset Definition Table (PDD).

10.3.17 DXT ALLOCATION TABLE (XAT)

The XAT is created by Startup, and contains a bit map showing which DXT entries are in use, and which are available.

10.4 THEORY OF OPERATION

The Permanent Dataset Manager is called by the Exchange Processor for SAVE, ACCESS, DISPOSE, RELEASE, DELETE, ADJUST, MODIFY, and PERMIT verbs and to perform functions for PDSDUMP, PDSLOAD, and AUDIT. The Permanent Dataset Manager is also called by the following:

- The Station Call Processor (SCP) to create Dataset Catalog (DSC) entries for spooled input datasets, to delete DSC entries for spooled output datasets, to perform permanent dataset name (PDN) requests, and to SAVE datasets staged from front-end stations

- The Exchange Processor (EXP) to create DSC entries for spooled output datasets, to delete DSC entries for spooled input datasets, and to rewrite spooled input dataset entries, and

- Startup, to rebuild Active Permanent Dataset Table (PDS) entries for permanent datasets associated with jobs being recovered or to access/save system datasets such as $ROLL and $SDR.

Job termination must check to see if a dataset is permanent before releasing the dataset from the system.

LOG MANAGER (MSG) 11

The Message Processor task (MSG) is the COS Log Manager. 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 (EXP), 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).

11.1 LOG PROCESSING

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 through Task I/O (see section 4).

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.

11.1.1 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 (EOF) record control word (RCW)s and terminates it with an end-of-data (EOD) 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. A new edition is also created whenever the current edition cannot be accessed or when validation of the edition number in upper memory fails. Additionally, creation of a new edition of $SYSTEMLOG may be forced by a parameter file option at startup time (see the COS Operational Procedures Reference Manual, publication SM-0043).

The System Log memory buffer and Dataset Parameter Table (DSP) are allocated in upper memory to facilitate System Log recovery. A startup parameter file option provides for no recovery of the System Log memory buffer once the System Log has been successfully accessed. (See the COS Operational Procedures Reference Manual, publication SM-0043.)

At system startup, the Log Manager attempts to recover the System Log using the following procedure.

1. Check the Force-new-$SYSLOG-edition flag. If set, go to step 4. Access $SYSTEMLOG. If none exists, set *x* in recovery message to 1 and go to step 5. Check suppress recovery of $SYSLOG buffer flag. If set, set *x* in recovery message to 9 and go to step 3.

2. Validate upper memory table area and buffer pointers. If valid, set *x* in recovery message to OK and go to step 6. Set *x* in recovery message to appropriate number (2 through 8).

3. Initialize upper memory table area; read $SYSTEMLOG to EOF, and backspace once. Go to step 6.

4. Call Permanent Dataset Manager to make a permanent dataset name request (determine current highest edition of $SYSLOG). If edition requested is less than or equal to the current highest edition, the system will hang.

5. Initialize upper memory table area. Initialize $SYSTEMLOG with EOFs. Rewind $SYSTEMLOG. Call PDM to save $SYSTEMLOG in the Dataset Catalog.

6. Enter SY014 - SYSTEMLOG RECOVERY SUCCESS CODE *x* message in the System Log. The recovery success codes are as follows:

Code	Definition
1	$SYSTEMLOG does not exist.
2	Recovery validation word is bad.
3	$SYSTEMLOG edition numbers do not match.

Code	Definition
4	$SYSLOG DSP failed validation.
5	The EOR flag is not set in $SYSLOG DSP.
6	Record control is not RCW or BCW.
7	A bad forward word index is in RCW.
8	A bad block number is in BCW.
9	The $SYSTEMLOG buffer in memory is not recovered.
10*nnnn*	The System Log is read to EOF. The block with the EOF RCW is not the same block pointed to in the $SYSTEMLOG buffer. The recovered buffer is discarded, and writing is resumed in the block with the EOF. *nnnn* indicates the number of blocks lost.
11	Error on System Log read to EOF. A new edition is created.
OK	A good recovery occurred.

7. If $SYSLOG was saved, enter SY015 - $SYSTEMLOG SAVED AS EDITION *x* message in the System Log and go to step 8. If an error is returned from PDM and $SYSLOG is not saved, the system hangs.

8. Initialization complete.

11.1.2 USER LOG PROCESSING

A user log dataset named $LOG is created for each job by Job Scheduler (JSH) when a job is initiated. The buffer for this dataset is in the Job Table Area (JTA) for the job. Tasks (and the user through the Exchange Processor) can 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 can 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 sends a delay status to the calling task; processing the delay status is up to the calling task. The queue limit prevents 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.

11.2 TASK CALLS TO MSG

A System Task Processor (STP) calls MSG either through the synchronous TSKREQ routine or the asynchronous PUTREQ routine. MSG sends a reply to the calling task containing a return status informing the calling task if the message was copied to a queue entry in an STP memory pool, or if the message could not be copied to a queue entry in an STP memory pool because of $LOG queue overflow or no available consecutive memory pool space. This allows any part of STP to enter a message in the System Log and/or user log.

Before executing a return jump to TSKREQ or PUTREQ, STP places the following information in input registers:

```
(A2)    MSGID,0

(S1)    INPUT+0
                        Request
(S2)    INPUT+1
```

Request format (LOGMSGM macro):

```
             OR CLASS
           0        7 10       19    25             40    46               63
          --------------------------------------------------------------------
INPUT+0   |FC M   |///| TYPE  | SUB |     LENGTH     |           ADDR         |
          --------------------------------------------------------------------
INPUT+1   |///////////////////////////////////////////////|        JXT        |
          --------------------------------------------------------------------
```

Field	Word	Bits	Description
FC	INPUT+0	0-1	Function code: 1 Write message in user's logfile only. 2 Write message in System Log only. 3 Write message in both user's logfile and in System Log.

Field	Word	Bits	Description
OR	INPUT+0	2	Override bit; if set, disregard suppresssion of this message's message class.
CLASS	INPUT+0	3-6	Class assigned to message
TYPE	INPUT+0	10-18	Major type of log record (see section 11.4)
SUB	INPUT+0	19-24	Subtype of log record (see section 11.4)
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:

(S1) OUTPUT+0
(S2) OUTPUT+1

Reply format:

	0	63
OUTPUT+0	MSGWORD	STATUS
OUTPUT+1	REQWORD	

Field	Word	Bits	Description
MSGWORD	OUTPUT+0	0-39	First 5 characters of the message located at ADDR specified in the input request; either ASCII characters or binary data.

Field	Word	Bits	Description
STATUS	OUTPUT+0	40-63	Status of requested function: =0 Message was successfully queued in memory pool. ≠0 Message was not successfully queued in memory pool.
REQWORD	OUTPUT+1	0-63	Copy of INPUT+0

11.3 SYSTEM TABLES USED BY MSG

The following tables are used for message processing.

AUT	Active User Table, for interactive mode
DSP	Dataset Parameter Area
JTA	Job Table Area
JXT	Job Execution Table
LGJ	Log JXT Table
PDD	Permanent Dataset Definition Table
SDT	System Dataset Table

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

11.3.1 ACTIVE USER TABLE (AUT)

MSG uses the following fields in the AUT:

AULSF	Logoff special function field
AUFF	Logged Off Terminal flag
AUOQC	Output queue control word

11.3.2 DATASET PARAMETER AREA (DSP)

Task I/O uses the Dataset Parameter Area (DSP) tables for the $SYSLOG and the $LOG datasets. The DSP and I/O buffer for $SYSLOG are allocated in upper memory during system startup. The DSP and I/O buffer for $LOG are in the user Job Table Area (JTA).

11.3.3 JOB TABLE AREA (JTA)

MSG uses the following fields in the Job Table Area:

JTLDP	User Dataset Parameter Table (DSP) address for $LOG
JTTSX	Time spent executing
JTDLM	Disable Log Message flag; set during job termination.
JTMSG	User log record area
JTLGF	User log buffer
JTJN	Job name
JTJXT	Job Execution Table (JXT) address
JTLOG	Logfile ($LOG) Dataset Name Table (DNT)
JTCSTK	PRECHO field in JTCSTK procedure stack field
JTTRM2	Terminate Job Immediately flag; set when $LOG is ready to overflow.

11.3.4 JOB EXECUTION TABLE (JXT)

MSG uses the following fields in the JXT:

JXSDT	System Dataset Table (SDT) offset
JXJN	Jobname
JXTSX	Time spent executing
JXJTA	Job Table Area (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	Active User Table (AUT) address

11.3.5 LOG JXT TABLE (LGJ)

The LGJ contains a 1-word entry for each job having records placed in its user logfile ($LOG).

11.3.6 PERMANENT DATASET DEFINITION TABLE (PDD)

Permanent Dataset Manager (PDM) requests issued by MSG are accompanied by Permanent Dataset Definition tables (PDDs).

11.3.7 SYSTEM DATASET TABLE (SDT)

MSG obtains the job sequence number, if the message is associated with a job, from the SDT.

11.4 $SYSTEMLOG FORMAT

Each message in $SYSTEMLOG occupies a single variable-length record on the System Log. All records of the same type and subtype have the same format. MSG builds four 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 through 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. (For more information on EXTRACT and STATS, see the COS Operational Aids Reference Manual, publication SM-0044.)

Format 1 System Log records are created by 1.12 and later versions of COS; records created by earlier COS versions are considered format 0. Records are formatted as follows:

Format 0:

```
           0     10     15     24              40                 63
RECORD+0  |                       TIME                              |
RECORD+1  |U| TYPE|  SUB |FMT|////|     JSQ     |       LENGTH       |
RECORD+2  |              JOBNAME or binary data                     |
RECORD+3  |                                                         |
    .     |                                                         |
    .     |              ASCII text or binary data                  |
    .     |                                                         |
 RECORD+  |                                                         |
LENGTH+1  |                                                         |
```

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 types.
FMT	RECORD+1	16-18	System Log record format (0)
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.

Format 1:

	0		10	15	24	40 63
RECORD+0	TIME					
RECORD+1	U	TYPE	SUB	FMT	JSQ	LENGTH
RECORD+2	TASK					
RECORD+3	JOBNAME					
RECORD+4 • • • RECORD+*n* LENGTH+1	ASCII text or binary data					

Field	Word	Bits	Description
TIME	RECORD+0	0-63	Timestamp; encoded date/time.
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 subtypes.
SUB	RECORD+1	10-15	Subtype of message. See System Log message subtypes.
FMT	RECORD+1	16-18	System Log record format (1)
JSQ	RECORD+1	24-39	Job sequence number (SDJSG). Used only if entry is associated with a job
LENGTH	RECORD+1	40-63	Number of words of actual ASCII or binary data
TASK	RECORD+2	0-63	Calling task name in ASCII
JOBNAME	RECORD+3	0-63	Job name, (JXJN), used for type 1 and type 4 records. For other record types, this word contains 0.
	RECORD+4 through RECORD+LENGTH+1	0-63	ASCII text or binary information as supplied by caller

The following paragraphs describe formats for different message types recorded in the System Log.

11.4.1 TYPE 0 - NULL MESSAGES

A type 0 message is a null record that can 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.

11.4.2 TYPE 1 - ASCII STRING MESSAGES

Type 1 messages include all user-requested messages and system-generated informative messages.

The subtype field indicates the originator of the message, as shown in table 11-1.

Only type 1 messages can be written in a user logfile.

Table 11-1. ASCII message subtypes

Originator	Format 0 Subtype	Format 1 Subtype
STP	0	0
SCP	1	1
EXP	2	2
PDM	3	3
DEC	4	4
DQM	5	5
MSG	6	6
MEP	7	7
SPM	8	8
JSH	9	9
JCM	10	10
TQM	11	11
OVM	12	12
STG		13
USER	15	61
CSP	16	62
ABORT	17	63

11.4.3 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	STAT-CMD; messages relating to station commands.
4	STAT-MSG; messages relating to station messages.

11.4.4 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†
5	ASCII error messages
6	I/O Subsystem channel error messages
7	I/O Subsystem disk error messages
8	I/O Subsystem tape error messages

11.4.5 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
3	TQM accounting messages

11.4.6 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
3	Hardware characteristics record

† Deferred implementation

11.4.7 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 14 describes the System Performance Monitor task in detail.

Subtype	Message
2	Task usage
3	EXEC requests
4	User memory usage
5	Disk usage
6	Disk channel usage
7	Link usage
8	EXEC call usage
9	User call usage
11	Job Scheduler management statistics
12	Job class information
13	CPU usage
14	Interrupt count

11.4.8 TYPE 7 - TASK DEBUG MESSAGES

Tasks write type 7 messages in binary. Type 7 messages are used for debugging.

Subtype	Message
1	TQM trace buffer

11.5 $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:

	0	6	24	32	40	63
RECORD-3	Chain items[†]					
RECORD-2	Chain items[†]					
RECORD-1	L[†]	//////////////////	JQ[†]		JX[†]	
RECORD+0	R					
RECORD+1	TI					
RECORD+2	CI					
RECORD+3	CF			S	LV	SP
RECORD+4	ID					
RECORD+5	T					
.	.					
.	.					
.	.					
RECORD+*n*						

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
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+3	0-31	CPU time, four fractional digits
S	RECORD+3	32-47	ASCII space codes

[†] 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. Item chaining is described in section 4.4 of this manual.

Field	Word	Bits	Description
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 through RECORD+*n*	0-63	Message text in ASCII code; last word is left-justified with zero fill.

MESSAGE PROCESSOR (MEP) 12

The Message Processor task (MEP) exists so that EXEC and the I/O Subsystem (IOS) can communicate with the System Log. MEP can also pass configuration information to the IOS (when requested by the IOS). MEP relays information about errors from EXEC or the I/O Subsystem to MSG, and detects and reports any problems in I/O Subsystem status. This information describes double- or single-bit memory errors, Buffer Memory errors, IOS recovered disk errors, IOS error channel status, and IOS availability. The installation parameter I@MAXME limits the number of memory errors that can be entered into the System Log so that it does not overflow.

12.1 EXEC MEMORY ERROR MESSAGE FORMAT

Messages from EXEC to MEP consist of the standardized Any Packet Table (APT) header (1 word) followed by five words of memory error information. The memory error message format is described in section 2.10.

MEP communicates only the last four words of this message to the MSG task (through PUTREQ) when requesting entry of the error in the System Log.

12.2 I/O SUBSYSTEM INTERFACE

MEP initializes itself with an I/O Subsystem I/O request to all I/O Subsystem message sources. If a request is rejected, a system down message is placed in the System Log. If initialization is successful, a system up message is sent to the System Log.

When the MEP task is in execution, it searches for I/O Subsystem status changes and for requests to input messages. Whenever an I/O Subsystem changes to the system up status, a system up message is placed in the System Log. Whenever an IOS status changes to system down, a system down message is placed in the System Log. When an input is requested, MEP supplies a buffer and requests transfer of the message.

12.3 I/O SUBSYSTEM HARDWARE ERROR MESSAGE FORMATS

IOS hardware generates two types of error messages: error channel status and disk error status.

The IOS error channel status is a 2-word binary message.

	0	16	32	48	63
0	ECSSID	//			ECTYPE
1	ECSTAT	ECP1	ECP2	ECP3	

Field	Word	Bits	Description
ECSSID	0	0-15	ASCII C for error logging purposes
ECTYPE	0	61-63	Type of error: 4 First error of this type 3 Last error of this type 1 Between first and last error of this type
ECSTAT	1	0-15	15-bit augmented error channel status
ECP1	1	15-35	First parameter associated with error channel
ECP2	1	32-47	Second parameter associated with error channel
ECP3	1	48-53	Third parameter associated with error channel

The I/O Subsystem disk error detailed status message[†] is six words.

	0	16	32	48	63
0	DSSID	//////////////////	DSLBK		
1	DSLDV				
2	DSEF	DSLM	DSIR	DSSKS	
3	Flags	//////////////////////////////	DSPN	DSCH	
4	DSFCO		DSFC1		
5	DSFC2		DSFC3		

[†] Deferred implementation

Field	Word	Bits	Description
DSSID	0	0-15	I/O Subsystem station ID
DSLBK	0	32-63	Logical block number
DSLDV	1	0-63	Logical device name
DSEF	2	0-15	Initial status flags
DSLM	2	16-31	Margin position at recovery
DSIR	2	31-47	Initial interlock status register
DSSKS	2	48-63	Initial seek status
Flags:			
DSTO	3	0	Timed-out flag
DSRE	3	1	Reservation Error flag
DSCE	3	2	Error Correction Used flag
DSIFC	3	3	Inconsistent Firecode flag
DSPN	3	32-47	IOS processor number
DSCH	3	48-63	IOP channel number
DSFCO	4	0-31	Firecode parameter word 0
DSFC1	4	31-63	Firecode parameter word 1
DSFC2	5	0-31	Firecode parameter word 2
DSFC3	5	32-63	Firecode parameter word 3

12.4 ASCII MESSAGES

MEP requests that the following ASCII messages be entered into the System Log:

IOP SUBSYSTEM IS UP - IOS is initialized or has been restarted.

IOP SUBSYSTEM IS DOWN - IOS does not respond.

IOP SUBSYSTEM LOGGING IS ENABLED - IOS error channel interrupt is enabled.

IOP SUBSYSTEM LOGGING IS DISABLED - IOS error channel interrupt is disabled.

DISK ERROR CORRECTION (DEC)

The Disk Error Correction task (DEC) is called by the CRAY-OS Disk Queue Manager task (DQM). DEC attempts correction of a disk error by applying the cyclic redundancy checkword (CRC) algorithm described in the Mass Storage Subsystem Hardware Reference Manual, CRI publication HR-0630.

13.1 DEC INTERFACE WITH OTHER TASKS

DEC is called by a task through PUTREQ, which places the Equipment Table (EQT) address in INPUT+0. DEC returns the request word in the reply.

Input registers:

	0	8	16	24	32	40	48	56	63
INPUT+0	Return			Zero		EQT			
INPUT+1	Zero								

Field	Word	Bits	Description
Return	INPUT+0	0-15	Return address; saved across the call.
EQT	INPUT+0	40-63	Equipment Table (EQT) address

Output registers:

	0	8	16	24	32	40	48	56	63
OUTPUT+0	Status								
OUTPUT+1	Return			Zero		EQT			

Field	Word	Bits	Description
Status	OUTPUT+0	0-63	Error status: 0 Corrected error 1 Uncorrected error

Field	Word	Bits	Description
Return	OUTPUT+1	0-15	Copy of INPUT+0
EQT	OUTPUT+1	40-63	Copy of INPUT+0

13.2 SYSTEM TABLE USED BY DEC

DEC uses the 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.

SYSTEM PERFORMANCE MONITOR (SPM) 14

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 and sends it to the Log Manager, and performs a time delay. SPM's only communication with other tasks is one way to the Log Manager. SPM may be readied by EXP if an installation parameter is defined to allow a user job to request SPM initiation.

14.1 SYSTEM TABLES USED BY SPM

The following system tables are used by SPM:

CSD	Class Structure Definition Table
DCT	Device Channel Table
IC	Interrupt Count Table
MCT	Monitor Call Table
STT	System Task Table

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

14.1.1 CLASS STRUCTURE DEFINITION TABLE (CSD)

The CSD is an STP-resident table containing all job class information.

14.1.2 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.

14.1.3 INTERRUPT COUNT TABLE (IC)

The IC counts interrupts for each channel or pseudo channel. This table is read and zeroed indirectly through EXEC calls.

14.1.4 MONITOR CALL TABLE (MCT)

Monitor Call Table (MCT) serves only SPM. It counts each type of call to EXEC from various tasks.

Table format:

	0 8 16 24 32 40 48 56 63
0	Number of type 0 requests
1	Number of type 1 requests
.	.
.	.
.	.
n	Number of type *n* requests

14.1.5 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.

14.2 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

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 14.4). The vector is right-adjusted so that the rightmost bit corresponds to subtype 12.

I@USRSPM Allow user jobs to ready SPM through the F$SPM EXP function if I@USRSPM is set nonzero. If I@USRSPM is 0, a job issuing F$SPM is aborted with the message:

"AB026-INVALID (UNDEFINED) USER CALL".

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. However, the changed parameters do not take effect until after the current delay interval.

14.3 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 through 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.

14.4 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 14-1 through 14-12.

A listing of subtypes follows:

Table	Subtype	Description
14-1	2	Task usage
14-2	3	EXEC requests
14-3	4	User memory usage
14-4	5	Disk usage
14-5	6	Disk channel usage
14-6	7	Link usage
14-7	8	EXEC call usage
14-8	9	User call usage
14-9	11	Job Scheduler management statistics
14-10	12	Job class information
14-11	13	CPU usage
14-12	14	Interrupt count

Table 14-1. 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 14-2. 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 14-3. 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

Table 14-4. Disk usage record - subtype 5

Word	Data	Origin of Data
0	Time interval	Calculated in SPM
1	Number of devices	NE@EQT
2	Logical device 0 name	LDV field in EQT
3	Blocks transferred (device 0)	NBK field in EQT
4	Seek time (device 0)	CST field in EQT
5	Transfer time (device 0)	TRT field in EQT
6	Number of physical requests (device 0)	NPR field in EQT
7	Number of nonseek requests (device 0)	RSC field in EQT
8	Maximum allocation units, number of permanent AIs, number of free AIs	MAU, PDA, AIA fields in DRT
.	.	
.	.	
.	.	
7*N+2	Logical device N name	
7*N+8	Number of nonseek requests (device N)	

Table 14-5. Disk channel usage record - subtype 6

Word	Data	Origin of Data
0	Time interval	Calculated in SPM
1	Number of channels	NE@DCT
2	Channel 0 time	DTCTA field in DCT (in STP)
3	Channel 1 time	DTCTA field in DCT
.	.	.
.	.	.
.	.	.
N+2	Channel N time	DTCTA field in DCT

Table 14-6. Link usage record - subtype 7

Word	Data	Origin of Data
0	Time interval	Calculated in SPM
1	Number of links	Number of active LXT entries
2	Logical ID (link 0)	LXLID in LXT entry
3	Number of messages (link 0)	LXNM in LXT entry
4	Number of words sent (link 0)	LXNWS in LXT entry
5	Number of words received (link 0)	LXNWR in LXT entry
.	.	
.	.	
.	.	
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 14-7. 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)

Table 14-8. 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 14-9. 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
16	Number of memory moves	PM11NM in STP
17	Number of memory words moved	PM11NWM in STP

Table 14-10. 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

Table 14-11. CPU usage record - subtype 13

Word	Data	Origin of Data
0	Time interval	Calculated in SPM
1	Number of CPUs	C@CPQUAN (installation parameter)
2	Idle time	ITIM (in EXEC)
3	User time, CPU 0	PWUCUM in PWS (EXEC)
4	Idle time, CPU 0	PWICUM in PWS (EXEC)
5	Blocked time, CPU 0	PWBCUM in PWS (EXEC)
6	EXEC time, CPU 0	PWECUM in PWS (EXEC)
7	STP time, CPU 0	PWSCUM in PWS (EXEC)
.	.	.
.	.	.
.	.	.
5N+3	System task 0 time	STTIME field of STT (EXEC)
5N+4	System task 1 time	STTIME field of STT (EXEC)
.	.	.
.	.	.
.	.	.
5N+M+3	System task M time	STTIME field of STT (EXEC)

Table 14-12. Interrupt count record - subtype 14

Word	Data	Origin of Data
0	Time interval	Calculated in SPM
1	Number of channels	C@CPHCHN+1 (EXEC)
2	Number of flags	NE@IHT (EXEC)
3	Channel 0 interrupts	CIT (EXEC)
4	Channel 1 interrupts	CIT (EXEC)
.	.	.
.	.	.
.	.	.
N+3	Channel N interrupts	CIT (EXEC)
N+4	Flag 0 interrupts	ICT (EXEC)
N+5	Flag 1 interrupts	ICT (EXEC)
.	.	.
.	.	.
.	.	.
N+M+4	Flag M interrupts	ICT (EXEC)
N+M+5	Flag 0 interrupts	IHT (EXEC)
N+M+6	Flag 1 descriptor	IHT (EXEC)
.	.	.
.	.	.
.	.	.
N+M+5	Flag M descriptor	IHT (EXEC)

14.5 TASK FLOW FOR SPM

The following general stepflow describes SPM processing.

1. If SPM not enabled, go to 17; 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 to write, 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.

JOB CLASS MANAGER (JCM) 15

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.
```

15.1 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 can override this assignment to lower the class through use of the CL parameter on the JOB control statement. 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 through 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 jobs selected or is not rerunnable during a restart with recovery of rolled jobs disabled.

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.

15.2 JCM INTERFACE WITH OTHER TASKS

The Job Class Manager (JCM) 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 of this manual for a complete description of task communications.

Input register format:

```
          0                 17                        40           58     63
         ---------------------------------------------------------------------
INPUT+0  |//////////////////////////////////////////////////////////|   FC   |
         ---------------------------------------------------------------------
INPUT+1  |////////////////|          CMO           |          SDT            |
         ---------------------------------------------------------------------
```

Field	Word	Bits	Description
FC	INPUT+0	58-63	Function code
CMO	INPUT+1	17-39	Class Map Offset of the associated class
SDT	INPUT+1	40-63	SDT address of the associated job

Output register format:

```
           0     6          17                          40                    63
          ----------------------------------------------------------------------
OUTPUT+0 | FC  |//////////////////////////////////////////////////////////////|
          ----------------------------------------------------------------------
OUTPUT+1 | FC  |/////////////|          CMO             |          SDT         |
          ----------------------------------------------------------------------
```

Field	Word	Bits	Description
FC	OUTPUT+0	0-5	Function code
FC	OUTPUT+1	0-5	Function code
CMO	OUTPUT+1	17-39	Class Map Offset of the associated class
SDT	OUTPUT+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 table 15-1.

Table 15-1. 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

15.2.1 CLASSIFY REQUEST

The classify request assigns a specific job to a class; it is 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

15.2.2 RECLASSIFY REQUEST

The reclassify request reassigns a specific job to a class; it is 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.

15.2.3 ASSIGN REQUEST

The assign request assigns a specific job to a specific class; it is 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

15.2.4 FIXCLASS REQUEST

The fixclass request fixes the class structure; it is 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 might 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.

OVERLAY MANAGER (OVM) 16

The Cray Operating System (COS), by necessity, is not all concurrently memory resident. Instead, much of it can be on disk in the form of system overlays. Overlay management, in which OVM has the major role, processes function requests that call for the loading and execution of these system overlays. The corresponding macros which provide tasks with a means of communicating with OVM, are described in the Macros and Opdefs Reference Manual, publication SR-0012.

OVM performs the following overlay management functions:

Function	Function code	Macro
Overlay load request	OV$FCLD	LOADOVL
Overlay call request	OV$FCCL	CALLOVL
Overlay goto request	OV$FCGO	GOTOOVL
Overlay reuse disable request	OV$FCDIS	DISABLE
Overlay return request	OV$FCRTN	RTNOVL

The modules of code intended as overlays are identified as such during system assembly through system macros. These macros ensure that the assembler and loader construct the loader tables and that during COS execution, the system is informed of what overlays exist. The overlay definition macros are also described in the COS Operational Aids Reference Manual, publication SM-0044.

The areas of the system involved in overlay management are:

DQM Provides the mechanism for reading overlays from disk

STPMEM Allocates/deallocates memory pool space to hold overlays and their associated tables

STARTUP Allocates or recovers the overlay space on disk, builds the overlay descriptor tables and moves the overlays to disk

OVM Handles the task requests by loading the appropriate overlays, updating the tables which describe overlay residence, and maintaining the overlay calling sequence

The I@ODTSZ and I@SCALLS installation parameters allow an analyst to set characteristics for overlay management according to an installation's needs. Installation parameters are defined in the COS Operational Procedures Reference Manual, publication SM-0043.

16.1 SYSTEM TABLES USED BY OVM

The Overlay Manager task (OVM) uses the following tables to control overlay loading and unloading:

OCS	Overlay Call Stack
OCT	Overlay Control Table
ODT	Overlay Directory Table
OLL	Overlay Load Request List

The formats of these tables are described in detail in the COS Table Descriptions Internal Reference Manual, CRI publication SM-0045.

16.1.1 OVERLAY CALL STACK (OCS)

The OCS contains an entry giving the caller's ID and the return address for each call to an overlay. Each CALL adds an entry to the stack, and each RETURN removes an entry from the stack. OVM allocates space for the OCS when the overlay is first called.

16.1.2 OVERLAY CONTROL TABLE (OCT)

Each overlay residing in memory has an associated OCT. OVM constructs the OCT in the memory pool reserved for overlays. Among other things, the OCT flags whether its associated overlay is currently busy or idle.

16.1.3 OVERLAY DIRECTORY TABLE (ODT)

The ODT is defined in the STP table area. Each overlay defined by a DEFINOVL macro contains an entry in the ODT. Each entry contains addressing information and data on the overlay's use. The I@ODTSZ installation parameter defines the length of the table. The DEFINOVL macros follow the Startup code and precede the Control Statement Processor (CSP) code.

16.1.4 OVERLAY LOAD REQUEST LIST (OLL)

The OLL contains a backlog of requests for overlays. When an overlay load is requested and the memory pool is full, an entry is added to the OLL to be processed when space becomes available. The OLL resides in the

STP table area, and its length is determined by the number of entries specified by the I@OLL installation parameter.

16.2 USING OVM FUNCTIONS

The following sections describe how and when tasks request functions of OVM.

16.2.1 INITIAL LOAD OVERLAY REQUEST

When a task needs to load an overlay to continue executing, it must issue a LOADOVL call. The LOADOVL macro sends an OV$FCLD function code to OVM. For this request, OVM builds an overlay call stack and loads the requested overlay. If the memory pool contains insufficient space for the request, the request is placed in the Overlay Load Request List. If an error occurs that prevents honoring a request, an error is returned to the caller.

Error code	Significance
OV$ECNS	No room in the Overlay Load Request List

Input required by OVM for an initial overlay load request consists of the overlay ID. This ID is an overlay identifier and is defined at installation.

The format of an OV$FCLD request (LOADOVL macro) follows:

```
     0                         24                        48          63
    +--------------------------------------------------------------------+
S1  |///////////////////////////|    Return address     |    OV$FCLD    |
    +--------------------------------------------------------------------+
S2  |/////////////////////////////////////////////////////|  id  | entry |
    +--------------------------------------------------------------------+
```

id	Overlay identifier
entry	Entry point

The OV$FCLD reply follows:

```
      0                        24              40      48             63
     ---------------------------------------------------------------------
  S1 |f|///////////////////////////////////////| f-dependent contents    |
     ---------------------------------------------------------------------
  S2 |//////////////////////////|  Return address    |     OV$FCLD       |
     ---------------------------------------------------------------------
```

f Flag specifying contents of bits 40-63, as follows:
- 1 Error code
- 0 Address to which control is transferred

16.2.2 TRANSFER OF CONTROL REQUESTS

When an overlay in execution determines that it needs another overlay to perform some function, it issues either a CALLOVL or a GOTOOVL. A CALLOVL is used if the calling overlay wants control returned to it when the called overlay completes; a GOTOOVL is used if control is not to return to the caller.

For either request, OVM determines if the overlay is already loaded. If not, OVM attempts to load it by allocating space in the memory pool and having the Disk Queue Manager (DQM) read the overlay from disk. The overlay issuing the request is marked not busy. If there is no space for loading the overlay, an entry is added to the Overlay Load Request List (OLL).

For a CALLOVL, OVM saves the caller ID and return address in the Overlay Call Stack. When the called overlay completes execution, the calling overlay is reloaded and OVM restarts the caller at the return address supplied by the OCS entry. For a GOTOOVL, the information about the caller is not saved. When the called overlay terminates, OVM restarts the last overlay that made a CALLOVL or returns to the task that issued the original LOADOVL.

Input to OVM for a CALLOVL is the desired overlay ID and the return address. For a GOTOOVL, only the overlay ID is used.

OV$FCCL request (CALLOVL macro):

```
      0                        24                      48             63
     ---------------------------------------------------------------------
  S1 |//////////////////////////|  Return address      |    OV$FCCL     |
     ---------------------------------------------------------------------
  S2 |/////////////////////////////////////////////////|  id  |  entry   |
     ---------------------------------------------------------------------
```

id Overlay identifier

entry Entry point

Reply:

```
        0                       24              40                      63
       ---------------------------------------------------------------------
  S1   |f|/////////////////////////////////////| f-dependent contents       |
       ---------------------------------------------------------------------
  S2   |//////////////////////////| Return address     |     OV$FCCL        |
       ---------------------------------------------------------------------
```

f Flag specifying contents of bits 40-63, as follows:
  1 Error code
  0 Address to which control is transferred

OV$FCGO request (GOTOOVL macro):

```
        0                                              48               63
       ---------------------------------------------------------------------
  S1   |///////////////////////////////////////////////|   OV$FCGO         |
       ---------------------------------------------------------------------
  S2   |///////////////////////////////////////////////|  id   |   entry   |
       ---------------------------------------------------------------------
```

id Overlay identifier

entry Entry point

Reply:

```
        0                                      40      48               63
       ---------------------------------------------------------------------
  S1   |f|/////////////////////////////////////////| f-dependent contents  |
       ---------------------------------------------------------------------
  S2   |///////////////////////////////////////////////|   OV$FCGO         |
       ---------------------------------------------------------------------
```

f Flag specifying contents of bits 40-63, as follows:
  1 Error code
  0 Address to which control is transferred

16.2.3 INHIBITING OVERLAY REUSE

The Overlay Reuse Disable request (or DISABLE macro) signals OVM that the current overlay must be reloaded before it can be reused. This is used only if some change in code or an internal table requires over reinitialization.

OVM sets a flag in the OCT signaling that this copy of the overlay cannot be reused.

OV$FCDIS request (DISABLE macro):

```
        0                                               48            63
   S1  |////////////////////////////////////////////////|  OV$FCDIS    |
   S2  |///////////////////////////////////////////////////////////////|
```

Reply:

```
        0                                               48            63
   S1  |                              0                                |
   S2  |////////////////////////////////////////////////|  OV$FCDIS    |
```

16.2.4 RETURNING TO CALLED OVERLAY

An overlay uses an Overlay Return request to notify OVM that it has completed execution. When this request is received, OVM uses the following process to determine which overlay, if any, should be loaded next.

1. Take the last entry made in the Overlay Call Stack and restart the specified overlay.

2. If the OCS is empty, return to the calling task.

After each Overlay Return request, OVM attempts to satisfy any outstanding requests in the OLL.

OV$FCRTN request (RTNOVL macro):

```
        0                                               48            63
   S1  |////////////////////////////////////////////////|  OV$FCRTN    |
   S2  |///////////////////////////////////////////////////////////////|
```

Reply:

```
        0                                       40      48            63
   S1  |f|//////////////////////////////////////| f-dependent contents |
   S2  |////////////////////////////////////////////////|  OV$FCRTN    |
```

f Flag specifying contents of bits 40-63, as follows:
1 Error code
0 Address to which control is transferred

16.3 OVM REQUEST PROCESSING

The following sections briefly describe how OVM processes requests.

16.3.1 OV$FCLD REQUEST (LOADOVL) PROCESSING

The processing flow follows:

LOAD1 Execution initiates in OVM and a GETREQ is performed.

LOAD2 OVM determines that a OV$FCLD request is being processed.

LOAD3 Try to allocate an Overlay Call Stack (OCS) through MEMAL; if no space exists, go to LOAD13.

LOAD4 Build the OCS entry; this includes the overlay/task ID and return address of the calling program, and (S1) from the request.

LOAD5 Search the Overlay Control Table (OCT) to determine if the overlay being called is already in memory and is not busy; if so, go to LOAD11.

LOAD6 Search Overlay Directory Table (ODT) to find matching overlay ID and entry point; error if not found.

LOAD7 Attempt to allocate overlay memory pool space for overlay area and OCT. If space not available, go to LOAD13.

LOAD8 Set up Overlay Control Table (OCT) and put request to Disk Queue Manager (DQM) to load the overlay.

LOAD9 After Disk Queue Manager (DQM) loads the overlay, relocate address references and release space occupied by the Block Relocation Table (BRT).

LOAD10 Send reply to caller that overlay has been loaded.

LOAD11 The overlay is already in memory; set OCT busy and other pertinent information.

LOAD12 Get corresponding ODT entry and determine address of entry point to be used. Go to LOAD10.

LOAD13 Space does not currently exist for an Overlay Call Stack (OCS) or an Overlay Control Table (OCT) and the associated overlay. If no space remains in the Overlay Load Request List (OLL), go to LOAD14. Otherwise, build an OLL entry and try to process other requests.

LOAD14 Send error function back to the caller.

16.3.2 OV$FCCL REQUEST (CALLOVL) PROCESSING

The processing flow follows:

CALL1 OVM determines that an OV$FCCL request is being processed.

CALL2 The Overlay Control Table (OCT) chain is searched for an entry corresponding to the overlay containing the CALLOVL macro.

CALL3 Save information from OCT and set it not busy.

CALL4 Go to LOAD4.

16.3.3 OV$FCGO REQUEST (GOTOOVL) PROCESSING

The processing flow follows:

GOTO1 OVM determines an OV$FCGO request is being processed.

GOTO2 Search for the Overlay Control Table (OCT) corresponding to the overlay containing the GOTOOVL macro.

GOTO3 Save pertinent information from OCT and set it not busy.

GOTO4 Go to LOAD5.

16.3.4 OV$FCDIS REQUEST (DISABLE) PROCESSING

The DISABLE macro can only occur within an overlay. It sets a bit in the OCT indicating that the overlay must be reloaded before being used.

16.3.5 OV$FCRTN REQUEST (RTNOVL) PROCESSING

The processing flow follows:

RTN1 OVM determines that an OV$FCRTN request is being processed.

RTN2 Search for the Overlay Control Table (OCT) entry corresponding to the overlay that issued the RTNOVL macro.

RTN3 Get the Overlay Call Stack (OCS) address from the OCT and set OCT entry not busy.

RTN4 Decrement number of entries in the OCS; if this is the last entry, go to RTN6.

RTN5 Get overlay ID and return address from OCS and go to LOAD5.

RTN6 Put reply to caller containing return address and search for new request to process.

TAPE QUEUE MANAGER (TQM) 17

The Tape Queue Manager (TQM) manages tape I/O between one or more user jobs and the I/O Subsystem (IOS). TQM is responsible for operator communications in the form of mount messages and device, controller, and channel-reconfiguration processing through the CONFIG common subroutine. TQM also maintains global availability and allocation counts for devices so the Job Scheduler (JSH) can appropriately schedule jobs.

The minimum I/O Subsystem configuration needed to support magnetic tape operations consists of a Master I/O Processor (MIOP), a Buffer I/O Processor (BIOP), an Auxiliary I/O Processor (XIOP), and 500K words of Buffer Memory. I/O Subsystem software residing in the XIOP physically controls all block multiplexer channels, control units connected to these channels, and tape devices connected to these control units.

TQM performs two kinds of tape I/O: user and system. The major activity is tape reading or writing directly to or from a user's circular buffer. But at times (for example, for label processing or volume switching), TQM must defer or queue a user request to perform system functions in support of a user job. When this happens, TQM sets the Sequencer-active flag and loads a string of functions (also referred to as sequences) to be executed. The functions can be I/O Subsystem requests or requests to execute a subroutine. When the last function in the string is complete, the Sequencer-active flag is cleared and any queued user request is resumed. The sequencer thus performs system I/O rather than normal user I/O.

The logic flow in TQM is as follows.

1. Perform delayed functions. (See section 17.4.)

2. Process all I/O Subsystem replies. (See section 17.5.)

3. Process all COS or operator requests. (See section 17.6.)

4. Return to the idle loop. (See section 17.7.)

17.1 SYSTEM TABLES USED BY TQM

TQM uses the following tape-related tables to control the configuration and the activity associated with each device:

CNT	Configuration Table
TDT	Tape Device Table

In addition, TQM uses the following system tables in the course of processing:

DEX	Auxiliary Dataset Enquiry Table
DNT	Dataset Name Table
DSP	Dataset Parameter Table
DUX	Auxiliary Dataset Update Table
FSH	Front-end Service Header
GRT	Generic Resource Table
JTA	Job Table Area
JXT	Job Execution Table
LDT	Label Definition Table
SM	Station Message Header
VAX	Auxiliary Volume Access Table
VUX	Auxiliary Volume Update Table

The formats of these tables are illustrated in the COS Table Descriptions Internal Reference Manual, publication SM-0045.

17.2 TQM INTERFACE WITH THE I/O SUBSYSTEM

Communication between TQM and the I/O Subsystem is accomplished through the EXEC I/O Subsystem driver, which manages the transfer of 6-word command and reply packets between TQM and the I/O Subsystem. Packets going to the I/O Subsystem are contained in the first six words of the TDT entry being processed, while packets coming to TQM from the I/O Subsystem are delivered to a single 6-word area near the end of TQM. In both cases, the Subsystem Control Table (SCT) near the end of TQM (B@SCT symbolically), controls communication with the I/O Subsystem driver in EXEC.

As with I/O Subsystem disk I/O, the SCT links TQM to EXEC. For tape packets, data is transferred between Central Memory in the CPU and I/O Subsystem Buffer Memory in 512-word increments to minimize the impact on existing I/O and library routines. The I/O Subsystem notifies TQM of all physical tape block boundaries, enabling TQM to keep an accurate record of tape position.

The 6-word bidirectional packet is the first six words of the Tape Device Table (TDT). The packet format permits asynchronous processing of data transfers between the CPU's Central Memory and Buffer Memory and of the physical reading or writing of tape. Refer to the COS Table Descriptions Internal Reference Manual, publication SM-0045, for a detailed description of the TDT.

17.3 TQM INITIALIZATION

When Startup creates the Tape Queue Manager Task (TQM), TQM receives control at the symbol TQM. Initialization consists of the following functions:

1. TQM checks the ASCII-to-EBCDIC and EBCDIC-to-ASCII conversion tables for compatibility. If the tables are set up improperly, the system halts.

2. TQM establishes a link with the I/O Subsystem driver in EXEC by sending to the driver a packet-I/O (PIO) clear request along with the address of TQM's Subsystem Control Table (SCT) (B@SCT). If EXEC rejects the request (the I/O Subsystem might not be active), TQM delays for 16 seconds and resends the PIO clear request.

3. When EXEC accepts the PIO request, TQM scans the Tape Device Table (TDT) and increments device resource counts for each available device.

17.4 DELAYED FUNCTION PROCESSING

The TQDLY subroutine performs all delayed functions. TQDLY tries to send any pending mount messages (for example, messages waiting for a station to log on), to process a pending RESELECT, update the Label Definition Table (LDT) (for a job that is not rolled in at the moment), complete some other function (such as update the count of volumes mounted) when a job is found to be back in memory, or process any pending requests when there is no associated TDT entry (such as job waiting for a device on an ACCESS request).

Following the scan for all pending conditions, TQDLY returns to process any I/O Subsystem replies, beginning at location TQPXR.

17.5 I/O SUBSYSTEM REPLY PROCESSING

The second major function of TQM is the processing of I/O Subsystem replies to TQM. If there are no replies from the I/O Subsystem to be processed, control passes to TQPCR to perform the third major TQM function, described in section 17.6.

17.5.1 REPLY PACKET FORMAT

The I/O Subsystem sends its replies to TQM in a packet with the same format as the TQM request packet. The replies usually correspond to a physical event, such as a block being written to tape. Relevant fields in the reply packet follow:

TDFCN — The function that TQM requested the I/O Subsystem to perform. The functions are the X$*ff* symbols defined in COMTQM. The value for TDFCN in the reply packet might differ from the value of TDFCN in the TDT.

TDRS — Reply status bits from the I/O Subsystem (and the function to which they belong) indicating that processing of the request is complete and/or that an error occurred for the specified function.

TDVSB — For a read reply, TDVSB contains the number of valid sectors of data in Buffer Memory. For a write reply, TDVSB contains the number of tape blocks held in Buffer Memory.

TDMOS — For read and write replies, TDMOS contains the number of Buffer Memory unallocated sectors that belong to the XIOP.

17.5.2 TYPES OF I/O SUBSYSTEM REPLIES

The I/O Subsystem returns the following kinds of replies to TQM:

Intermediate — An event occurred for the request; other processing replies will follow.

Ending — A terminal state was reached in the processing of the request.

Normal — An event occurred that is a normal part of the request.

Error — An abnormal event occurred during the processing of the request. The TQM error classifications are:

> Soft — Require a volume remount or reject
>
> Firm — Require the associated job to be aborted
>
> Hard — Require that the device be declared down and that the associated job be aborted
>
> Void — Involve the processing or initializing of any status bits that are irrelevant to the request

A return status from the I/O Subsystem can consist of a combination of the replies. For example, if a parity error occurs on the last requested block during a read, the I/O Subsystem replies TDBFN (block finished) and TDPDE (permanent data error). The return status, in this example, includes the following combination of replies:

- Normal. The block finished indicates the completion of a normal event.
- Ending. The TDBFN reply and TDRBC (requested block count) going zero indicate the I/O Subsystem completed the request.
- Error. The TDPDE reply indicates that a block of bad data was read into Buffer Memory.

17.5.3 I/O SUBSYSTEM REPLY PROCESSOR STRUCTURE

Because of the reply mechanism described above, each reply processor in TQM has the following structure:

- Processing of initial reply status
- Processing or initializing of any normal reply status bits
- Processing of any soft errors
- Processing of any firm errors
- Processing of any hard errors

- Processing or initializing of any status bits that are irrelevant to the request

- Performing of any auxiliary processes and proceeding with processing of the most important initialized event if any exist

17.5.4 REPLY-EXIT ADDRESS

Another integral part of each reply processor is the reply-exit address which is the address to which transfer passes when all steps of the reply processor are performed. Initially, each reply processor assumes that more I/O Subsystem replies will occur and sets the reply-exit address to TQPXR. Use of the reply-exit address allows each reply processor to evaluate all possible states conveyed in the reply and to process the most important state. Generally the reply-exit address is set to one of the following:

TQPXR — Check for more replies. If set to this address, no more processing will be done for this device at exit time.

TQPSN — Advance sequencer. A normal-end state was reached for this request, and processing continues with the next step in the current sequence.

TQ$ERROR — Generalized error handler. Causes transfer of control to the appropriate error handler. The appropriate error handler can be TQ$SOFT, TQ$FIRM, TQ$HARD, TQ$VOID, or sequencer-error address (TDSS1).

(TDSS3) — Sequencer intermediate reply address. This address is obtained from the TDT when the active sequence wants to process the reply in a different way than the default reply processor.

TQCIOP2 — Resume I/O request streaming (also known as DSP pointer chasing). This address is used by X$RB and X$WB to continue the streaming of user I/O requests when appropriate.

Other addresses may be used for continued processing of the state, such as EOT on writes. Addresses of this nature generally are contained in the reply processor.

17.5.5 INITIALIZATION SUBFUNCTION (TQPXR)

TQPXR is the most common way in which a reply processor is invoked, though delay operations may reinitiate a reply processor. TQPXR examines the SCCIP flag in the I/O Subsystem Control Table (SCT) for any replies that EXEC has queued from the I/O Subsystem. If replies are queued, TQM requests the packets through the PIORCV monitor request. If no replies are queued, control passes to the next major function of TQM, which is TQPCR (see section 17.7).

When a packet is delivered, the device number (TDDVN) is used as an ordinal into the Tape Device Table (TDT) to determine which device is associated with the reply. The converted device number is kept in the A5 register as the base address of the TDT entry for the device. The contents of A5 generally remain set to this address throughout the processing of the reply.

Under certain conditions, TQM discards all replies for a device, which is reflected by setting TDSDP to a nonzero value. A condition under which this usually occurs is when TQM reaches an ending state with respect to a sequence but the I/O Subsystem has not reached that ending state. For example, while the sequencer is performing a remount of a volume, the user releases the dataset. TQM stops the mount process without negatively affecting the user's RELEASE request by discarding any I/O Subsystem replies.

Before the invocation of the appropriate reply processor, reflected by TDFCN in the reply packet, the TDT entry for the device is updated with values that the I/O Subsystem has returned. These values follow:

TDRS Reply status bits

TDVSB Used in XRB, XCR, and X$WB to reflect the number of blocks (write) or sectors (read) the device has in Buffer Memory.

TDMOS The number of sectors in Buffer Memory that are not allocated to any device.

Once the TDT entry is updated, the appropriate reply processor is determined by using the value of TDFCN in the reply packet (not the TDT entry since it may differ from the value of TDFCN). The current function is used as an offset into a jump table.

17.5.6 WRITE TAPEMARKS AND REWIND FUNCTION

TQ$1TR and TQ$2TR are the reply processors for the write tapemarks and rewind function. TQM uses the write tapemarks and rewind function to complete the writing of trailer labels when at end of volume or end of file; the function is used exclusively under sequencer control. Control immediately passes to an intermediate-reply address if one exists. The normal status replies processed are described below.

- Initial status is an an acknowledgement by the I/O Subsystem that processing of the request is to begin. All status bits are 0. TQM usually waits for other replies before advancing to the next sequence.

- Tapemark status (TDTMS) indicates a tapemark was written to tape. The block-finished status (TDBFN) must accompany TDTMS; if it does not, the I/O Subsystem is in error. Various counts are manipulated, such as volume-block and label-block counts before reply-exit address processing is performed. The reply-exit address can be modified to TQPSN when the current sequence indicates advance on tapemark status or advance on all tapemarks being written.

- Beginning-of-tape status (TDBOT) reflects the normal ending status for this function. TDBOT indicates that the rewind portion of the request is completed. The reply-exit address is modified when the current sequence indicates an advance on ending status or advance on BOT status.

Soft errors require a volume reject or remount. TDNOR (no write ring in volume) is the only soft error handled in conjunction with the write tapemarks and rewrite function. This status will be handled by the sequencer error address (TDSS1).

Firm errors require that the job be aborted. They are permanent dataset error (TDPDE) and operator hit reset (TDRES). Since all sequences that use this function have an error address (TDSS1) defined, error processing occurs at the address defined in TDSS1.

Hard errors reflect a severe hardware failure or a state in which the device will not be available for some time. Possible hard errors follow:

TDLSD The tape is off the end of reel.

TDNTR The device dropped ready status.

TDNOP The path to the device or the device itself is no longer communicating with the I/O Subsystem.

TDPEC A protocol error concerning the write tapemarks and rewind function occurred between TQM and the I/O Subsystem.

Void errors indicate status bits that have no meaning for the tapemarks and rewrite function and should not appear in a reply. Possible void errors follow:

TDDTR Data was transferred to or from Buffer Memory.

TDLBK A block was read from tape that is larger than MBS.

TDNCD The device/controller indicates that it is not capable of performing this function at the requested density.

TDWFE The I/O Subsystem detected an error in the transfer of data from Cray mainframe memory to Buffer Memory.

17.5.7 CONTINUE READ FUNCTION

TQ$CR is the reply processor for the continue-read function. TQM uses the continue-read function at the end of data while reading a tape dataset. The purpose of the function is, first, to append EOR, EOF, and EOD control words to the data read from tape and, second, to treat the last sector in Buffer Memory as a full sector even when the last sector is not full.

The continue-read function reply processor requires that the sequencer be active and that all of the function's status processing be handled by an intermediate address (TDSS3). Initial status, where all status bits are 0, is the only valid status.

17.5.8 FREE-DEVICE FUNCTION

TQ$FD is the reply processor for the free-device function. TQM uses the free-device function at various times to release a device from a job. Device freeing may occur during RELEASE processing or RESELECT processing. The free-device reply processor differs from other reply processors in that the cleanup of the TDT entry occurs regardless of states, etc. The cleanup of the TDT entry includes the following.

- Clearing the assignment and linkage to a job in both the TDT and CNT
- Clearing any pending action states that are canceled by the I/O Subsystem when it processes a free device function
- Clearing any operator or front-end communication messages
- Releasing any pool space associated with the device
- Declaring the drive down if the operator configured it down while it was active.

17.5.9 READ-BLOCK FUNCTION

TQ$RB is the reply processor for the read-block function. The read-block function requests data from the tape and transfers read-in data to Cray Central Memory. The read-block function is used to read user data from the tape and to read any label information that might exist.

The requested block-count (TDRBC) and requested sector-count (TDRSC) fields control the work to be done for this request along with the function code and dataset definition flags in the packet. TDRBC requests that the I/O Subsystem read the requested block count from tape into Buffer Memory. TDRSC requests that the I/O Subsystem transfer the requested sector count from Buffer Memory to Cray Central Memory.

During normal operation, the I/O Subsystem can respond to the request in a number of ways. It replies each time a tape block is read into Buffer Memory. It also replies each time it transfers data from Buffer Memory (which may be less than the requested amount). Unlike write processing, block-finished replies and data-transfer replies can occur in the same reply packet.

Aside from the evaluation of the I/O Subsystem reply, read-block reply processing is an integral part in user I/O request processing. The basic functions for interfacing to the user are to supply proper CIO replies and to evaluate and update the DSP for continued I/O.

The read-block function reply processor informs the user each time the DSP is updated as a result of a data-transfer reply. The read-block processor also tries to keep the data stream active by chasing the DSP pointers. To do so, the following conditions must be met.

- End of data has not yet been read.
- All requested data has been transferred (TDRBC=0).
- There are more full sectors of data in Buffer Memory to transfer (TDMSC≠0).
- The user has room in the circular buffer for more data.

If the conditions listed above are met, an intermediate (recall) reply is given and the reply-exit address is set to TQCIOP2. The intermediate reply to CIO allows the user to be reconnected to the CPU. The setting of the reply-exit address to TQCIOP2 reinitiates a scan of the DSP to determine the amount of new data that can be transferred to the user's circular buffer.

A normal I/O complete reply to CIO requires that the user re-issue an I/O request. Such a reply occurs if one of the following is true.

- End of volume is reached, and all full sectors of Buffer Memory data are transferred to the user.
- End of volume is reached, and all requested sectors of Buffer Memory data are transferred to the user.
- All requested Buffer Memory data is transferred to the user, and the user's buffer has less than a sector of available space.

Other replies to CIO which stop the data stream are as follows.

EREOI	End of information is transferred to the circular buffer.
ERUDE	Bad data is transferred to the circular buffer because of a bad tape block being read.

Initialization for the read-block function reply processor, as with all standard reply processors, involves the transfer of control to an intermediate-sequencer address (TDSS3) when one exists. Also the sequencer must be active. This mechanism is used by the read-label-group subroutine, TQCLT.

Normal replies from the I/O Subsystem are as follows.

- Data-transfer status (TDDTR) reflects that data is transferred from Buffer Memory to Cray Central Memory. When TQM receives the data-transfer status, TQM updates various counts that reflect and control the transfer, such as the number of sectors remaining to be transferred, accounting information, and circular buffer pointers. A reply is also made to CIO concerning this transfer. The read-block function might also modify the reply-exit address in order to continue the data stream. An error status that might accompany this status is TDDBF. TDDBF indicates that part or all of the last sector of the data transferred is bad. At most, one full sector of bad data is transferred per reply.

- Block-finished status (TDBFN) indicates that a single tape block was read into Buffer Memory from the tape. When accompanied by TDTMS, a tapemark is read and read-ahead processing is halted. As with TDDTR accounting, general and control information is updated. The read-block function reply processor also sets the reply-exit address to TQCIOP2 when a deficiency in Buffer Memory data was resolved.

- Tapemark read status (TDTMS) must be accompanied by the TDBFN status. TDTMS indicates that read-ahead processing is halted (that is, TDOBC should be cleared) and that end-of-volume processing is to begin as soon as possible. This is flagged by setting TDEOV. Before end-of-volume processing, the following must occur:

  - Outstanding sector count (TDOSC) is set to the minimum of TDOSC or TDMSC (number of full sectors in Buffer Memory at the time of TMS). This change synchronizes TQM and the I/O Subsystem with respect to any active data-transfer request.

  - Flag the device as not ready for user I/O, which causes the job to be placed in an event-wait condition while end-of-volume processing is active.

  - Reply to CIO that device is not ready when there is no data in Buffer Memory for an active user-I/O request. Normally this reply occurs the next time the user makes an I/O request. The not-ready mechanism allows the job to roll during end-of-volume processing

The read-block function reply processor examines the following firm errors.

TDLBK	A block larger than the value contained in TDMBS was read from tape. TDLBK is associated with TDBFN. This error ends the read-ahead portion of the request.
TDBOT	Beginning-of-tape reflective strip detected, indicating that either the tape has two BOT reflective strips or that a shiny spot on the tape is detected. TDBOT status halts read ahead processing.

TDPEC Protocol error. I/O Subsystem rejected the request because it seems to be out of sequence. This status halts read-ahead processing.

The read-block function processes the following hard errors.

TDLSD The tape is off the end of the reel. TDLSD halts read-ahead processing.

TDRES The operator reset the device. This status stops the read-ahead processing of tape blocks.

TDNOP The path to the device or the device itself is not operational which forces read-ahead processing to stop.

The read-block function processes the following unexpected (void) errors. Void errors halt read-ahead processing.

TDNCD The device is not capable of performing the request at the requested density.

TDWFE An error occurred in the transfer of data from Cray Central Memory to Buffer Memory.

Before the transfer of control to the reply-exit address, wrap-up processing checks if read-ahead processing should be refreshed. This refresh requests the I/O Subsystem to continue the command chaining of tape-block reads. It has been determined that a threshold of three outstanding blocks is the appropriate time to re-issue an X$RB request (with a nonzero TDRBC) to keep a command chain alive and avoid a physical stop of the device. This threshold is decreased for the larger blocks (such as block sizes of one megabyte).

17.5.10 REMOUNT OR MOUNT PROCESSING FUNCTION

TQ$RM or TQ$MN are the reply processors for the remount and mount processing functions. The remount and mount-processing functions inform the I/O Subsystem that the operator might mount a tape volume on the device. The X$MN request also informs the I/O Subsystem that this is the first operation for this device/dataset combination and requests that the I/O Subsystem allocate buffers, etc. for subsequent I/O. This reply processor is under sequencer control. Any of the following can initiate the request to mount or remount: ACCESS processing, volume-reject processing, volume-switch processing, or device-reselect processing.

The operator can direct processing to another device (RESELECT) or can cancel the processing (ABORT). The job itself can no longer request the mount. It terminates because of a job abort or operator action (DROP, KILL, RERUN).

One of the mechanisms used to handle various conditions of this kind is the setting of the TDDMR flag. This flag indicates that mount and/or remount replies for the device are ignored. Another mechanism is the use of an intermediate address for the current sequence. The initialization that takes place in the remount reply processor is to examine these conditions. If neither exists, processing will continue with a scan of the status bits.

The remount or mount processing function examines the following normal replies.

- Initial reply is an informative reply that indicates that the I/O Subsystem began processing the mount or remount request but that the subsequent replies are the ones that require processing.

- Not-ready status (TDNTR) indicates that processing is not ready because the volume is not at beginning of tape. TDNTR does not invoke processing because it is an optional status. Not-ready status is not returned when the volume is mounted at request time.

- Beginning-of-tape status (TDBOT) is the normal ending status for the request. It indicates the volume is mounted and is ready for a new operation. In response to a remount/reselect, it also indicates that the I/O Subsystem linked any residual data from the previous device to the new one. The basic functions are to clear any road-block flags (TDWXL and TDWNW), set reply-exit address to advance sequencer (TQPSN), and clear outstanding mount messages. If the request was a remount due to a RESELECT command, the old device is freed.

The remount or mount function reply processor treats any other status as soft errors. Processing of soft errors includes clearing of the outstanding mount messages, configuring down the old device (necessary so that the device is not reselected to itself) and initiating an automatic reselect. This process keeps the current sequence at the (re)mount request.

17.5.11 REWIND FUNCTION

TQ$RW is the reply processor for the rewind function. TQM uses the rewind function (X$RW) to rewind a single volume dataset to beginning of tape or to rewind nonlabeled volumes after label scans. Normally this request occurs under sequencer control.

Control immediately passes to an intermediate-sequencer address if one exists and if the sequencer is active.

The rewind reply-processing function treats the following as normal replies

- Initial reply status acknowledges that processing of the X$RW request is to begin. TQM normally waits for other replies before advancing to the next sequence.

- Beginning-of-tape status (TDBOT) is a normal ending status indicating the volume is ready for the next operation. The wait on loadpoint (TDWXL) road-block flag is cleared to reflect this state change. If the sequencer advance code is advance on BOT or advance on ending status, the reply-exit address is modified to TQPSN.

The rewind reply-processing function handles the following soft errors.

- Device-reset status (TDRES) indicates that the operator pressed the reset button on the device. Normally this implies simply that rereadying the device has to be done before the next operation.

The rewind reply-processing function handles the following hard errors.

- Tape-off-reel status (TDLSD) indicates the tape is off the end of the reel.

- Device-not-ready status (TDNTR) indicates the device status has changed to not ready.

- Not-operational status (TDNOP) indicates the drive, controller, or channel is no longer operational.

The rewind reply-processing function handles the following void (unexpected) status.

- Block-finished status (TDBFN) indicates a tape block was read or written.

- Data-transferred status (TDDTR) indicates that data was transferred from or to Cray mainframe memory.

- Tapemark status (TDTMS) indicates a tapemark was read or written.

- Large-block status (TDLBK) indicates a tape block larger than MBS was read or written.

- Permanent data error status (TDPDE) indicates a parity error occurred while trying to read or write a tape block

- End-of-tape status (TDEOT) indicates an end-of-tape reflective strip was detected.

- Not-capable status (TDNCD) indicates the device cannot perform the request at the requested density.

- Write-format error status (TDWFE) indicates an error occurred in the transfer of data to be written to tape.

- Protocol-error status (TDPEC) indicates the request violates the protocol between TQM and the I/O Subsystem.

17.5.12 WRITE-TAPEMARK FUNCTION

TQ$WT is the reply processor for the write-tapemark function. TQ$WT is part of the write-label group sequence. It writes the tapemark that separates the user data from the label data. As with all label processing, this request is only made while under sequencer control. Control is transferred to an intermediate sequencer address if one exists before processing of normal reply status.

The write-tapemark function processor processes the following normal replies.

- Initial reply status indicates that the I/O Subsystem began the processing of the request.

- Tapemark status (TDTMS) indicates that a tapemark is written to tape. The block-finished status must accompany TDTMS. After updating the various block counts, reply-exit address processing is performed. The reply-exit address is changed to TQPSN if the sequencer advance code is: advance on tapemark, advance on tapemark count equal 0, or ending status.

The write-tapemark function processor processes the following soft errors.

- No ring in reel status (TDNOR) indicates that the tape is write disabled. Because of the environment in which the request is issued, the receipt of TDNOR usually indicates a floating write-ring detector on the device.

Firm errors require that the job be aborted. They are permanent dataset error (TDPDE) and operator hit reset (TDRES). Since all sequences that use this function have an error address defined, processing occurs at TDSS1.

Hard errors reflect a severe hardware failure or a state in which the device will not be available for some time. Possible hard errors are as follows.

TDLSD The tape is off the end of reel.

TDNTR The device dropped ready status.

TDNOP The path to the device or the device itself is no longer communicating with the I/O Subsystem.

TDPEC A protocol error concerning the write tapemarks function occurred between TQM and the I/O Subsystem.

Void errors indicate status bits that have no meaning for the tapemarks function and should not appear in a reply. Possible void errors are as follows.

TDDTR Data was transferred to or from Buffer Memory.

TDLBK A block was read from tape that is larger than MBS.

TDNCD The device/controller indicates that it is not capable of performing this function at the requested density.

TDWFE The I/O Subsystem detected an error in the transfer of data from Cray Central Memory to Buffer Memory.

TDBOT The beginning of a tape reflective spot was detected. Usually a second BOT reflective strip exists on the tape, or the tape has a shiny spot on it.

17.5.13 UNLOAD-VOLUME FUNCTION

TQ$UL is the reply processor for the unload-volume function. The unload-volume function (X$UL) is used in volume reject situations, volume switches, and release processing. It is followed by either a remount (X$RM) or a free (X$FD) request. The X$UL function is used under sequencer control.

If an intermediate reply address is defined and the sequencer is active, control is transferred to the intermediate reply address.

This reply processor treats all replies except the initial reply as normal ending replies because of two reasons: the subsequent request will have any errors propagated to it and the job and device states are usually defined better during the subsequent request. The normal sequence of replies is initial reply followed by not ready reply (TDNTR).

Before resetting the reply-exit address to TQPSN for ending replies, the TDT entry must be updated to reflect that unload processing is complete and to initialize various fields for the next volume to be mounted.

17.5.14 WRITE-BLOCK FUNCTION

TQ$WB is the reply processor for the write-block function. The function of the X$WB request is similar to the functions of the X$RB request except that all transfers of data are reversed. X$WB controls the transfer of data from Cray Central Memory to Buffer Memory and from Buffer Memory to tape (referred to as write-behind processing).

With write-block processing, unlike with read processing, the number of blocks contained in a sector of data is known. This knowledge eliminates the need for independent requests for write-behind processing. The I/O Subsystem is responsible for writing all tape blocks from Buffer Memory to tape, which in return changes the nature of the various counts (that is, TDOBC, TDMBC, TDMSC, and TDOSC). In write processing the management of Buffer Memory is essentially performed in TQCIW (process user-write request).

Another difference between read and write reply processing is that the basic replies (block finished and data transferred) cannot occur in the same reply packet.

Read processing and write processing are handled the same with respect to intermediate replies to the user for partially completed I/O request, DSP-pointer chasing, volume-switch detection, processing end-of-information special cases, and manangement of Buffer Memory.

The user interface for the write reply processor (as with read processing) includes intermediate (recall) replies, I/O complete replies, and DSP-pointer evaluation. The intent behind the user interface is to keep the original user-write request active as long as possible before the I/O complete reply is given. This cuts down the overhead that both the user and operating system have to perform for each new I/O request. It also allows the device to be driven at either device speed or user-program speed, whichever is the slower.

Intermediate replies are sent to CIO as a result of the transfer of data from the user's circular buffer to Buffer Memory. During the processing of a data-transfer reply, TQM gives an intermediate reply if the user has at least one sector of data in its circular buffer available for transfer and end of data does not exist in that data. I/O complete replies are sent for all other conditions.

As with read processing (though simpler), pointer chasing is involved with intermediate replies. The only condition processed that will halt the chase is a pending end-of-volume state.

Initialization of the X$WB reply processor consists of setting up some basic addresses and transferring control to an active intermediate-sequencer reply address if one exists. An intermediate-reply address is used when writing the label groups.

Normal replies from the I/O Subsystem for the write-block processing function are described below.

- The data-transfer status (TDDTR) reflects the movement of data from Cray Central Memory to Buffer Memory. The amount of data transferred is given in both sectors (TDTSC) and blocks (TDTBC), and the total number of blocks in Buffer Memory for this device is given in TDVSB. If the I/O Subsystem does not transfer the requested values (TDRSC and TDRBC), a fatal error occurs. After updating various control, accounting, and informational counts, the reply-exit address is determined and CIO reply processing is performed.

- The block-finished status (TDBFN) reflects the movement of data from Buffer Memory to tape. Unlike data-transfer status, a tape block is the basic unit for block-finished status. Block-finished status updates various control, accounting, and informational counts. This section also modifies the reply-exit address in order to resume streaming, provided the last DSP evaluation showed that there was at least one full sector of data in the user's buffer.

- End-of-tape status (TDEOT) is always accompanied by TDBFN. End-of-tape status indicates that end-of-tape reflective strip was detected and that a switch to the next volume is required. When detected, the reply-exit address is set to the volume-switch processor within TQ$WB.

- Write-last-block status (TDLBP) is not a status from the I/O Subsystem but instead is an internal flag that controls the writing of the last block for transparent mode datasets. Its intent is to let all outstanding processing finish until the last block of data is in Buffer Memory by itself. This wait is needed because the last block of a transparent tape is generally smaller than the rest of the blocks that are written (transparent tapes have a fixed-block size). This isolation of the last block allows the block size in the packet to change, thereby telling the I/O Subsystem to write a single smaller block.

Write-block processing function error status that imply a volume-reject condition (soft error) are permanent data error (TDPDE), no ring in reel (TDNOR), and beginning of tape (TDBOT). The error is reported to the user through the job logfile, and a switch to the next volume is queued. Since beginning-of-volume processing checks ring status, the receipt of TDNOR indicates a physical device problem and not actual trouble with the volume.

Firm error conditions handled by write-block processing follow:

- Large-block error status (TDLBK) indicates an attempt to transfer a block larger than the value contained in TDMBS.

- Write-format error status (TDWFE) indicates there is a control word error in the data transferred to Buffer Memory.

- Reset-of-device error status (TDRES) indicates that the operator reset the device.

- Protocol-error status (TDPEC) indicates the I/O Subsystem rejected the request because the request is out of sequence.

Hard error conditions that exist for the write-block processor are as follows.

- Lost-data error status (TDLSD) indicates that the tape is off the end of the reel.

- Not-ready error status (TDNTR) indicates the device dropped ready status.

- Not-operational error status (TDNOP) indicates the path or the device is no longer in an operational state.

Unexpected (void) error replies for the write-block processor are as follows.

- Tapemark-detected error status (TDTMS) indicates that a tapemark is physically read or written.

- Not-capable error status (TDNCD) indicates the device is not capable of performing the request at the specified density.

17.6 COS AND OPERATOR REQUEST PROCESSING

The third major activity of TQM (TQPCR) is interfacing with the other tasks in the operating system, which involves station message replies, operator requests, and user task requests. User task requests consist of I/O requests and OPEN, CLOSE, POSITION, ACCESS, SAVE, DELETE, and RELEASE requests.

The only tasks that request action are EXP and SCP. EXP originates all user-related requests, and SCP originates all operator requests.

Format:

```
           0       8      16      24      32      40      48      56     63
          +-------------------------------------------------------------+
Input+0   |////////|  SF  |             RTN           |     TXO    |  FC  |
Input+1   |S|//////////////|            DNT           |          AUX      |
          +-------------------------------------------------------------+
```

Field	Word	Bits	Description
SF	Input+0	8-15	Subfunction code
RTN	Input+0	16-39	Return address
TXO	Input+0	40-55	Task Execution Table (TXT) ordinal
FC	Input+0	56-63	TQM function code, as follows: T$CIO (1) CIO request T$OPN (2) Open dataset T$POS (3) Rewind dataset T$CLO (4) Close dataset T$RLS (5) Release dataset T$PDM (6) Tape Permanent Dataset Manager call T$ILL (7) Codes ≥ T$ILL are invalid
S	Input+1	0	System flag (not currently used)
DNT	Input+1	16-39	STP-relative Dataset Name Table (DNT) address
AUX	Input+1	40-63	Auxiliary address (usually the address of a table)

17.6.1 SCP REPLY

TQPCRR processes replies from SCP that occur as a result of a station-directed message issued by TQM. These replies can be responses from the station operator to a MOUNT or REMOUNT request, responses from the operator to a volume serial number (VSN) validation request, response to front-end servicing request, or indications from SCP that the station to which the message is sent has logged off or relogged on.

When the station operator replies to a MOUNT or REMOUNT, TQM examines the reply. If the reply is invalid, TQM re-issues the MOUNT or REMOUNT request.

When the reply from SCP is an operator response to a VSN validation request (issued only for unlabeled tapes), TQM validates the supplied VSN. If the VSN is invalid (more than six characters), TQM re-issues the VSN validation request. If the response is to a SCRATCH volume (VSN=??????), TQM ensures the validity of the VSN and assigns it to the SCRATCH volume.

If the station logged off or relogged on, TQM sets a flag in the TDT entry indicating to subroutine TQDLY that a message must be re-issued.

17.6.2 OPERATOR COMMAND

TQPOC handles operator-command processing. When SCP signals TQM through PUTREQ that the operator has entered a command to be processed by TQM, common subroutine GETREQ is called to obtain the command. The response from GETREQ contains the following:

Register	Contents
S1	Address of memory pool containing the operator command as typed by the operator
S2	Address of Link Interface Extension Table (LXT) entry for the sending station (must be returned to SCP by TQM)

The format of the data in the memory pool is as follows:

Words 0-4	Station Message (SMT) entry, provided by SCP. (For details of the format of SMT, see the COS Table Descriptions Internal Reference Manual, publication SM-0045.)
Word 5	Beginning of the actual ASCII text entered at the operator console

On return from GETREQ, TQM examines the operator command for validity. The only valid command is the CONFIG command, which modifies the status of a tape device. If the command is a CONFIG command, TQM calls the common subroutine CONFIG to process it. CONFIG determines whether the parameters of the command are valid and consistent; it alters the CNT and TDT entries according to the specified parameters. (For the format of the CONFIG command, see the I/O Subsystem (IOS) Station Operator's Guide, CRI publication SG-0051.)

After processing the command, TQM returns a status to SCP to indicate whether the command was successfully processed. CONFIG returns an error to TQM if errors are found in the parameters. The reply from TQM to SCP has the following format:

Register	Contents
S1	Status, as follows: =0 No error ≠0 Error
S2	LXT address for sending station, echoed from input

17.6.3 CIO REQUESTS

On entry to TQCIO, the following information is available in A registers:

Register	Contents
A1	JTA address
A2	JXT address
A4	DNT address
A5	TDT address

The calling task ID, the COS function, INPUT+0, and INPUT+1 are saved in the TDT, and the I/O active count in STP is incremented. Entry to TQCIO can also occur (at TQCIOP2) because of a DSP-pointer update.

If a queued error condition exists for this dataset (TDQHE is nonzero), the reply to this request is sent immediately with the error status. If no outstanding error condition exists, processing continues by putting the DSP address in register A3 and setting registers S1 through S4 to the circular buffer pointers FIRST, IN, OUT, and LIMIT. Control passes to TQCIR for a read request (F$RDC) or to TQCIW for a write request (F$WDC).

F$RDC request

The F$RDC request (TQCIR) has two primary functions: examination of pending states and requesting transfer of data to circular buffer. The pending states that are examined cover a variety of conditions; the major ones are pending BOV validation, device not ready, and Buffer Memory empty.

The examination of pending actions occurs in TQCIR because of the funneling effect this process has on the various states (job, tape, and operator). The detection of BOV validation is done by TQCIR primarily to link the type of label validation with the type of I/O being performed. The detection of Buffer Memory being empty allows the block-finished reply to determine when to resume a read request. Device-not-ready detection is a result of the internal EOV/BOV processing being invoked. Using TQCIR processing allows the isolation of the various conditions that acccompany a device-not-ready situation. One important condition is the issuing of a DROP, KILL, or RERUN command of a job already in device-not-ready event wait.

F$WDC request

The functions of the F$WDC request (TQCIW) are almost identical to its counter part TQCIR. F$WDC serves as a funnel for the various states that exist and manages the transfer of user data.

The major difference between TQCIW and TQCIR processing is in the evaluation and management of Buffer Memory. The difference in Buffer Memory evaluation is that instead of stopping when Buffer Memory is empty, with TQCIW a stop occurs when Buffer Memory is full.

The differences between TQCIR and TQCIW in producing the I/O Subsystem request are that the TQCIW request is a request to write blocks (as well as a request to transfer data), and the amount of data to transfer is based upon two factors instead of one. The amount of data to transfer is based upon the number of blocks currently in Buffer Memory and the number of blocks in the circular buffer. This aspect of the TQCIW request is limited, based on the write-behind count (TDTRA).

17.6.4 F$CLS CLOSE REQUEST

TQCLO receives control from TQPCR when TQM receives a call from EXP with a T$CLO function code.

TQCLO first checks to see if the sequencer is active. If it is, TQM replies to EXP with a delay status, and EXP tries the call again in a few seconds.

If the sequencer is not active, TQM checks for a redundant CLOSE. An immediate reply is sent to EXP if redundancy is discovered. If the volume is not opened and if the previous TQM request is a T$OPN, the close accounting message is issued and a reply is sent to EXP. If the volume is not opened and the previous TQM request is other than a T$OPN, a reply with delay is sent to EXP. If the volume is opened but a rewind is in progress, a reply with delay is sent to EXP.

The close is completed by merging with the appropriate path (TQPOS10 for writes and TQPOS50 for reads) in rewind processing. After close processing is complete, control returns to TQPCR to check for more requests to process.

17.6.5 F$OPN OPEN REQUEST

TQOPN gains control from TQPCR when TQM receives a call from EXP with a T$OPN function code.

TQOPN checks to see if the sequencer is active. If it is active, a reply with delay is sent to EXP. If the sequencer is not active, TQOPN checks for a redundant open call. If redundancy is discovered, the open state stored in the TDT is updated, and a reply is sent to the calling task.

If the open is not redundant (that is, it is an open following an access or a close), TQM delays open processing until the volume is mounted.

Once the volume is mounted, TQM resumes open processing by reading the header label group. The scan for the header labels is performed, as during a volume switch, by the subroutine TQCLT. The evaluation of the label group for the open is based on the disposition of the dataset.

NEW disposition datasets defer all remaining label processing until the first I/O operation is requested.

Processing OLD disposition datasets involves the following basic operations:

- Ensuring the proper volume is mounted
- Obtaining any values the user omitted from the label group

In checking for the proper volume, TQM verifies that the label types, VSNs, DSNs, and file sections agree. If a discrepancy is found, TQM either rejects the current volume in favor of another or replies with a job abort status. Once TQM is sure the proper volume is mounted, it extracts from the label group (provided the label group exists) any values that the user omits and places them in the Label Definition Area (LDT).

TQM generates default values for fields not in the label group and not specified in the LDT. These values guarantee matches when the labels are reevaluated during the first read or write.

17.6.6 F$PDM DELETE REQUEST

The F$PDM delete request function (TQPDELET) changes the catalog function to that of delete (remove from front-end servicing catalog).

17.6.7 F$PDM SAVE REQUEST

The F$PDM save function (TQPSAVE) changes the catalog function for the dataset to save (update or create an entry in front-end servicing catalog).

17.6.8 T$POS POSITION REQUEST

TQPOS gains control from TQPCR when TQM receives a call from the User Exchange Processor (EXP) with a T$POS function code.

If the specified device is down, an unrecovered hardware error status is returned to EXP. If the sequencer is active, a delay reply is sent to EXP. If a rewind is already in progress, an immediate reply is sent to EXP with control passing to TQPCR to check for more requests.

At this point, a rewind can be executed so the TDT VSN fields are refreshed from the LDT while the job is still in memory. For output mode datasets, all data is flushed to tape before writing trailer labels. For unlabeled (NL) tapes, a trailer label consists of three tapemarks. If the current volume is not the first volume of the dataset, an unload remount sequence is executed in order to return to the first reel. A reply is sent to EXP, and control returns to TQPCR to check for more requests.

17.6.9 F$RLS RELEASE REQUEST

TQRLS receives control from TQPCR when TQM receives a call from the User Exchange Processor (EXP) with a T$RLS function code.

TQRLS checks to determine whether a tape device is assigned to the dataset. If no device is assigned to the dataset, TQRLS sends an immediate reply to EXP, thus completing the release request.

For an assigned device, any outstanding operator messages are cleared. If the device is down, TQM clears the necessary fields in the TDT and only a free request is issued to the I/O Subsystem.

If the sequencer is active, TQM sends a delay status to EXP so the internal sequence completes.

If a rewind must be issued and the I/O Subsystem has no outstanding activity for the dataset, a delay reply is made to EXP. If the dataset is in write mode and the I/O Subsystem has no current activity, any remaining Buffer Memory data is flushed to tape before continuing with the release call. When all Buffer Memory data is flushed, trailer labels are written (three tapemarks on NL tapes).

Once all the data is flushed and the volume properly terminated, TQM issues an accounting message reflecting the release of the device.

If a volume is mounted, TQM initiates a sequence to unload the device.

TQM returns the device to the system pool unless the request specifies a release with a device hold, in which case the user retains the reservation for a generic device. A request is sent to the I/O Subsystem to free the device assignment. A reply is sent to EXP indicating the request is complete.

17.6.10 SEQUENCER REQUESTS (TQPSI OR TQPSN)

This routine has the following entry points:

- TQPSI, to process the initial sequencer request
- TQPSN, to process the next sequencer request
- TQPSEND, to terminate the current sequence

The TQM sequencer provides a facility for tasks, rather than user jobs, to perform I/O and to execute I/O Subsystem functions and related subroutines. It is the vehicle for label reading and writing and volume switching operations.

Sequencer activity is determined by function strings loaded into the sequencer by a call to TQPSI. A function string can contain as many as eight functions. A function can result in either an I/O Subsystem request being issued or a subroutine being executed. These strings are defined in word pairs with one function per parcel. A string is terminated by a parcel of zeros or by the eighth parcel, whichever comes first. These function strings are defined at the end of TQM at TQSSL.

The sequencer function contains the following information:

- Bit 0 of a function parcel (defined by the symbol TQSNXF) indicates function type. If TQSNXF is set, the parcel contains a non-I/O Subsystem function. If TQSNXF is not set, the parcel contains an I/O Subsystem function.

- Bits 4 through 9 of the function parcel hold the sequencer advance code. This code applies only to I/O Subsystem functions and it tells TQM when to advance the sequencer to the next function. These codes are defined at the end of TQM, starting at TQSAC0.

- Bits 10 through 15 of the function parcel contain the sequencer subroutine advance index for non-I/O Subsystem functions or the actual I/O Subsystem function code for I/O Subsystem requests. The sequencer subroutine advance index definitions are located at the end of TQM at TQSAA.

Sequences can be chained as long as the initiation of a chained sequence occurs after the last function or step of the current sequence has been initiated.

The SEQINIT macro is used to set up the following values used during a sequence:

- Sequence-end return address

- Sequence-error return address

- Intermediate processing address

- General 24-bit save value for scratch space

Format:

Location	Result	Operand
	SEQINIT	END=*end*,ERR=*err*,INT=*int*,SAV=*save*

where

end, *err*, *int*, and *save* may be a defined location, an A register, or S register.

17.7 IDLE-LOOP PROCESSING

Idle-loop processing is performed by the TQIDL routine. When TQM completes its work, it returns to the idle loop at TQIDL and suspends itself with a 4-second delay. When called again, it performs the task's major functions in the following order: any delayed functions, any I/O Subsystem replies, any COS or operator requests.

17.8 TQM STEPFLOWS

This section contains some basic stepflows for the more commonly used portions of TQM. The organization of these stepflows essentially corresponds to a job which performs the following.

ACCESS
OPEN
WRITE
REWIND
READ
CLOSE
RELEASE

17.8.1 GENERAL FLOW FOR DATASET ACCESS PROCESSING

1. Select a device for the dataset. If the job requests more devices than are reserved, abort the job. If there is a shortage of devices, enter job-awaiting resource code.

2. Initialize various tables including Tape Device Table (TDT), Dataset Name Table (DNT), Front-end Service Header (FSH), Label Definition Table memory pool area, (LDT), and Job Table Area (JTA).

3. Inquire of the front-end station about the accessibility of this dataset and about any auxiliary information for the dataset. If denied access, abort the job. If access if permitted, copy the LDT (updated by the front-end station) over the user's copy of the LDT.

4. Perform the various functions to initiate the volume mount:

    a. Send a mount message to the master operator.

    b. Transmit a mount request to the I/O Subsystem.

5. Indicate to the user what drive is assigned with a logfile informative message. If the drive assignment changes through a reselect, the user is notified again.

6. Reply to requesting task (EXP) indicating the request is complete.

17.8.2 GENERAL FLOW FOR OPEN PROCESSING

The following is the general flow for open processing.

1. Detect and reply to a redundant open.

2. Determine if volume is mounted. If not, make sure there is an outstanding mount message and request before replying to requesting task with a delay status.

3. Try to determine the label structure of the mounted volume by reading in a possible label group. The label group is defined as 1 to 30 blocks (none larger than 80 bytes) followed by a tapemark with the first four bytes of the first block equal to VOL1 (in ASCII or EBCDIC).

4. If an OPEN of an existing dataset (not NEW) then:

    a. Ensure the actual label type matches requested label type; abort the job if they differ.

b. If the mounted volume is labeled:

1) Perform VOL1 label validation checks, rejecting the mounted volume or aborting the job if any validation checks fail.

2) Copy from the label group any corresponding LDT fields that are not specified by the user.

5. Generate default values for any fields not specified in the label pool copy of the LDT.

6. Update the user's DSP based upon the information in the LDT.

7. Set the Tape Device Table flags CLG, CLT, GVS to indicate BOV validation must occur later.

8. Reply to requesting task (EXP) that open processing is complete.

17.8.3 GENERAL FLOW FOR WRITE DATASET PROCESSING

The following is the general flow for write dataset processing.

1. Detect and process any pending internal processing.

   a. Device not-ready (TDCNR) - Place job in event-wait state for device to become ready.

   b. Pending BOV label processing (TDCLG or TDCLT) - Place job in device-not-ready event-wait and initiate BOV validation (under sequencer control).

   c. Buffer memory full (TDMBC = TDEBC) - Flag the write request as pending and wait for block-finished reply to resume the request.

2. Evaluate the DSP and amount of data already in Buffer Memory to produce an I/O Subsystem write-block request (X$WB). This evaluation is performed differently for the different dataset modes.

   a. Transparent (DF=TR) - Transform the number of sectors in the circular buffer into tape blocks to determine if that number exceeds the excess block count. If so, transfer the maximum number of sectors to reach the excess block count. Otherwise, transfer the number of tape blocks in the circular buffer.

   b. Interchange (DF=IC) - Using the DSP pointers and the control words in the circular buffer, produce a data-transfer request that does not exceed the total number of sectors available to transfer or the excess block count.

3. Send the write-block request to the I/O Subsystem using the requested counts that were generated in the step above. Also save various pointer information that is used to determine the validity of the user's DSP during the data-transfer reply.

17.8.4 GENERAL FLOW FOR BEGINNING OF VOLUME VALIDATION (TQ$WB300)

The following is the general flow for beginning-of-volume validation.

1. Initialize for the new volume.
   a. Clear various state flags from the TDT.
   b. Detect and process a cancelation of the mount from the operator.
   c. Set up nonspecific volume request flags.
   d. Read the label group if it has not already been read.
2. Ensure that the proper volume is mounted.
   a. Request the VSN from the operator for nonlabeled volumes.
   b. Reject the volume when the label type of the mounted volume cannot migrate to the label type requested by the user.
3. Perform initial front-end station communication processing, using an error-bit mask to show discrepancies between the LDT (user specifications) and the label group (actual specifications). The error-bit mask can be modified based upon the reply from the front-end station to the volume-access request.
4. Send a volume-access request to the servicing front-end station. Perform reject/abort processing if the front-end station denies the access.
5. Based on the disposition of the volume, perform an evaluation of the label-group error-bit mask. Volumes which the user specifically requested result in job-abort conditions if label validation fails. Nonspecific volume requests result in a volume reject for a different scratch volume if validation fails.

6. If a MOD dataset, perform MOD tape processing:

   a. Forward space to end of file.

   b. Validate trailer label.

   c. Abort job if not EOF label.

7. If not a MOD dataset, write the label group for this tape:

   a. Generate a label group from the information in the LDT based on label type.

   b. Write the label group by invoking the sequencer under chained control.

8. Send a volume-update message to the servicing front-end station to communicate the use of the volume.

9. Clean up from the internal processing:

   a. Exit the current sequence.

   b. Initiate write-behind processing.

   c. Clear the device-not-ready event.

17.8.5 GENERAL FLOW FOR I/O SUBSYSTEM WRITE REPLY PROCESSING

The following is the general flow for I/O Subsystem write reply processing.

1. Initialize processing by setting up the DNT, DSP, JTA, and JXT addresses that are related to this device. Transfer control to an intermediate reply address if one exists.

2. Process a data transfer reply (TDDTR).

   a. Validate that the I/O Subsystem and TQM are still synchronized by comparing the transferred counts with the requested counts.

   b. Update counts that show what is in Buffer Memory (TDMBC) and what remains to be transferred (TDOBC).

   c. Update the various informational and accounting fields in recognition of the transfer above.

d. Update the DSP to reflect the transfer of data out of the circular buffer.

e. If not a SYNCH request, reply to the user with either an I/O complete or an intermediate reply, based upon how full Buffer Memory is, the amount of data left in the circular buffer, and a possible end-of-data state. Adjust the reply-exit address accordingly.

3. Process a block-finished reply (TDBFN).

   a. Decrement the number of blocks in Buffer Memory that remain to be written (TDMBC).

   b. Update the various informational and accounting fields that correspond to the movement of tape blocks.

   c. If SYNCH request and all blocks written to tape, reply I/O complete.

   d. Detect the possible condition of a write request that is queued because of Buffer Memory being full. If this condition exists, then the reply-exit address is set to TQCIOP2 so as to re-issue the request.

   e. Queue a write last block condition, if necessary, by changing the reply-exit address.

   f. Queue a volume-switch condition if end of tape is detected by setting the reply-exit address (TQ$WB200).

   g. Evaluate the return status for any error conditions that exist.

   h. Transfer control to the address in the reply-exit address.

17.8.6 GENERAL FLOW FOR VOLUME SWITCH DURING WRITE

The following is the general flow for volume switch during write.

1. Wait for any outstanding data-transfer request to complete (TDOSC going to 0).

2. Place the associated job in device-not-ready event wait.

3. Activate the sequencer to write the trailer label group, based on the requested label type.
4. Send a volume update request to the servicing front-end station.
5. Obtain the next volume serial number from the LDT. Load the job into memory, through the J$READY JSH request, if necessary. If a next VSN does not exist, set it to a scratch volume and change the current disposition to NEW.
6. Unload the current volume for another volume by initiating the sequencer.
7. Invoke beginning-of-volume validation.

17.8.7 GENERAL FLOW FOR REWIND/CLOSE PROCESSING

The following is the general flow for rewind/close processing.

1. Initialize processing by detecting the special end cases.
   a. Device is down:
      1) Clear appropriate fields in TDT.
      2) Reply with unrecovered hardware error.
   b. Dataset not open, abort with file not open.
   c. Sequencer is active, reply with delay status.
   d. Rewind is already in progress:
      1) Issue rewind accounting state message.
      2) Reply rewind complete to user.
2. Process rewind/close for a write mode dataset.
   a. Process outstanding write requests if they exist.
      1) Delay processing until all data is transferred to Buffer Memory.
      2) Flush the remaining Buffer Memory sectors to tape.

b. Write the trailer labels by activating the sequencer.

c. Send a volume-update request to the front-end station.

d. Report the end-of-data condition to the system and user logfiles.

e. Report the rewind/close condition to the system and user logfiles.

f. Clear appropriate fields in TDT for the next user request.

g. If current volume is the first volume, then reply rewind/close is complete, else go to step 4.

3. Process rewind/close for a read mode dataset:

   a. Stop any read-ahead activity by activating the sequencer until all outstanding blocks are read.

   b. Report the rewind/close to the system and user logfiles.

   c. If current volume is the first volume then:

      1) Reply rewind/close is complete.

      2) Perform a volume rewind by activating the sequencer, else go to step 4.

4. Process of rewind/close for multivolume dataset:

   a. Set up the TDT for the volume switch by setting CLG, CLT, and GVS.

   b. Process a rewind request:

      1) Determine whether job is in memory. If not, use the J$READY request. Wait for job to be brought into memory before the VSN list is examined.

      2) Issue a mount message to the master operator station.

      3) Issue a remount request to the I/O Subsystem.

      4) Switch volumes by activating the sequencer.

      5) Reply rewind is complete.

c. Process a close request:

   1) Unload the current volume by activating the sequencer.

   2) After the unload is complete, reply that the close is complete.

17.8.8 GENERAL FLOW FOR READ DATASET PROCESSING

The following is the general flow for read dataset processing.

1. Detect and process any pending internal processing.

   a. Device not ready (TDDNR) - Place the job in event-wait state, which waits for the device to become ready.

   b. Pending BOV label processing (TDCLG or TDCLT) - Place job in device-not-ready event-wait and initiate BOV validation under sequencer control.

   c. Buffer memory empty (TDMSC=0) - Flag the read request as pending and wait for block-finished reply to resume the request.

2. Evaluate the DSP so as to produce a data-transfer request. This does not need to relate to the amount of data in Buffer Memory, since the I/O Subsystem transfers any data as soon as data is available.

3. Send the read-block data-transfer request to the I/O Subsystem.

17.8.9 GENERAL FLOW FOR BEGINNING OF VOLUME READ VALIDATION (TQ$RB300)

The following is the general flow for beginning of volume read validation.

1. Initialize for the new volume.

   a. Clear various state flags from the TDT entry.

   b. Detect and process a cancelation of the mount from the operator.

   c. Read the label group if it exists and has not already been read.

2. Ensure the proper volume is mounted.

   a. Reject if label types differ.

   b. Request the VSN from the operator for nonlabeled volumes.

3. Perform initial front-end station commumication processing, using an error-bit mask to determine conflicts between the user and the volume mounted.

4. Send a volume-access request to the servicing front-end station. Reject or abort if the front-end station denies the access.

5. Evaluate the label error-bit mask. Reject the volume, abort the job, or continue depending upon the results of the evaluation.

6. Send a volume-update message to the servicing front-end station to reflect the use of the volume.

7. Reset from the internal processing.

   a. Exit the current sequence.

   b. Initiate read-ahead processing.

   c. Remove the user from the device-not-ready event-wait state.

17.8.10 GENERAL FLOW FOR I/O SUBSYSTEM READ REPLY PROCESSING

The following is the general flow for I/O Subsystem read reply processing.

1. Initialize processing by setting up the DNT, DSP, JTA, and JXT addresses that correspond to this device. Transfer control to an intermediate sequencer reply address if one exists.

2. Process a data-transfer reply (TDDTR).

   a. Decrement the number of sectors waiting to be transferred to the circular buffer.

   b. Update the various informational and accounting fields that correspond to the transfer of sectors.

   c. Update the DSP to reflect the new data that has been placed in the circular buffer.

   d. Reply to the user with an I/O complete, an intermediate reply, or a data-error reply; depending upon the data transferred, the amount of room available in the circular buffer, and if the I/O Subsystem has completed the original request. Modify the reply-exit address so as to comply with the type of reply given to the user.

3. Process a block-finished reply (TDBFN).

   a. Decrement the number of blocks remaining in the read-ahead count.

   b. Update various informational and accounting fields corresponding to the movement of tape blocks.

   c. Detect the possible condition of a read request was halted because no data existed in Buffer Memory. Modify the reply-exit address (if one exists) to TQCIOP2 to re-issue the request if necessary.

4. Process tape mask status (TDTMS).

   a. Set appropriate fields in TDT entry so as to synchronize TQM with the I/O Subsystem on read-ahead processing and data-transfer processing.

   b. Update various counts that reflect label group block counts.

   c. Flag the device, and possibly the user job, as in a device-not-ready state.

   d. Flag the device as having a pending EOV condition, which will be lifted when the data-transfer request is complete.

5. Detect a pending EOV condition. Modify the reply-exit address to process trailer labels (TQ$RB190) when the data transfer request is complete (TDOSC=0).

6. Evaluate the return status for any error conditions that exist.

7. If there are no blocking states, try to issue another read-ahead request.

8. Transfer control to the address given in the reply-exit address.

17.8.11 PROCESS TRAILER LABELS (TQ$RB190)

The following is the general flow to process trailer labels.

1. Set the device as not ready for user I/O.

2. Based on label type, determine if this is an end-of-volume or end-of-data condition

    a. Nonlabeled volumes. Go to end-of-data processing if no more VSNs exist in volume serial list; otherwise, go to volume switch (TQ$RB200).

    b. Labeled volumes

        1) Read trailer label group.

        2) Abort job if trailer labels are of a different type from the header labels. (Apparently, the volume was not terminated properly when written).

        3) Determine if EOV or EOF trailer group. If EOV, go to volume-switch code (TQ$RB200).

    c. End-of-data initialization

        1) Flag the condition in the TDT entry.

        2) Based upon dataset mode:

            a) Transparent - Reply end of information to the user.

            b) Interchange - Issue continue-read request to place EOR, EOF, and EOD control words in user buffer.

        3) Send volume-update message to front-end station concerning the end-of-data condition.

        4) Clear device-not-ready state thus allowing the user to request the last data that exists in Buffer Memory.

17.8.12 PROCESS VOLUME SWITCH FOR READ (TQ$RB200)

The following is the general flow to process volume switch for read.

1. Send volume-update message to servicing front-end station that corresponds to the end-of-volume state.
2. Obtain the next volume serial number from the LDT.
3. Unload the current volume for the next by initiating the sequencer.
4. Perform beginning-of-volume validation.

17.8.13 GENERAL FLOW FOR CLOSE PROCESSING

The following is the general flow for close processing.

1. Initialize processing by detecting and processing the special end cases.

   a. Delay the request if the sequencer is active.

   b. Redundant close request:

      1) Issue close accounting message.

      2) Reply close complete.

   c. Delay the request if there is an outstanding rewind.

   d. Clean up any queued conditions.

   e. If device is closed (due to error during label group write):

      1) Issue close accounting message.

      2) Reply close complete.

   f. Device has been downed:

      1) Clear appropriate fields in TDT.

      2) Reply with unrecovered hardware error.

2. Based upon last I/O operation, continue processing with step 2 of rewind or step 3 of rewind.

17.8.14 GENERAL FLOW FOR RELEASE PROCESSING

The following is the general flow for release processing.

1. Process the special states for a successful release operation.

    a. Clear outstanding operator message.

    b. Cancel any outstanding front-end catalog requests.

    c. Clear appropriate fields in TDT for system-downed device.

    d. Delay release, with delay reply, until sequencer is not active.

2. Process any write-behind/read-ahead condition:

    a. For read-mode datasets, delay until all outstanding read-ahead activity is done.

    b. For write-mode datasets:

        1) Flush any Buffer Memory write-behind data to tape.

        2) Write the end-of-data label trailer group by activating the sequencer.

        3) Send a volume update request to the front-end station.

        4) Report the end-of-data state to the user and system logfiles.

3. Report the release condition:

    a. Send a dataset update request to the front-end station.

    b. Report the release state to the user and system logfiles.

4. Unload the volume by activating the sequencer if a volume is mounted.

5. Return the device to the available pool:

    a. Wait for the job to be in memory.

    b. Place device in available pool and disconnect it from the user.

c. Clean up the TDT.

d. Free any devices linked to the device disconnected from the user as a result of a RESELECT.

e. Issue free device request to the I/O Subsystem.

6. Reply release complete.

17.8.15 PROCESS TAPE POSITIONING REQUEST

The following is the general flow for tape positioning.

1. Verify that the requested operation can be performed.
2. If output tape, write all data to tape and write a trailer label.
3. If input tape, discard all the read ahead data.
4. Switch volume if needed.
5. Position to requested block.
6. Update the DSP pointers so that the circular buffer appears to be empty.

17.9 TQM TRACE BUFFER

The TQM trace buffer and the TDT provide information about the current state of TQM and its recent past. Snaps taken at strategic points within TQM by using the TQSNAP macro consist of entries with the following information:

- Real-time clock at the time of the snap
- Snap number (32 bits)
- TQM-relative address from where the snap was called
- 1-8 characters of comment or the label of the routine from where the snap was called
- Last input packet received by TQM (six words)

- Entire TDT entry
- Registers A0-A7 and S0-S7

Since most of this information is on parcel boundaries, the trace is normally dumped either in parcel format or with the parameter FORMAT=PARCEL selected on an FDUMP of the TDT and trace buffer. The formatting area at TQSNAH near the end of the TQM listing details the precise trace entry format.

Two macros are defined in TQM to make the trace buffer more readable. They are:

- The @ macro
- The TQSNAP macro

The @ macro is used to define labels in TQM.

Format:

Location	Result	Operand	Comment
1	10	20	35
	label		

This macro generates the following:

```
label        =          *
%%TQMTAG     MICRO      'label'
```

The TQSNAP macro has the following format:

Location	Result	Operand	Comment
1	10	20	35
	TQSNAP		

comment From 1 to 8 characters. If more than 8 characters are specified, the first 8 characters are used.

If *comment* is not coded, the micro set up by the @ macro is used.

If @ and TQSNAP are used together, trace entries will contain the TQSNAP comments, or the routine label if no comments are provided.

STAGER (STG) 18

Stager (STG) is a subtask of SCP. The purpose of STG is to separate the disk I/O processing from the protocol processing in SCP. STG fills data segment buffers destined for front-end systems with data from mass storage, and writes data segment buffer contents received from front-end systems to mass storage. STG also initiates input jobs by processing the job card, assigning a job sequence number, and calling the Job Class Manager (JCM) to assign a job class.

SCP makes requests of STG using an unsolicited reply. SCP passes the address of a Stager Stream Table (SST) as a parameter. The contents of the table include an SCP message code indicating the processing being requested and information necessary to process the request. STG processes the request and responds to SCP in fields provided in the SST. STG uses its message code field within the SST to indicate the return status of the request.

18.1 TABLES USED BY STAGER

STG uses the following tables. These tables are fully described in the COS Table Descriptions Internal Reference Manual, publication SM-0045.

PDD	Permanent Dataset Definition
SDT	System Dataset Table
SST	Stager Stream table

18.1.1 PERMANENT DATASET DEFINITION (PDD)

STG uses the PDD in PDM requests to create and release permanent datasets.

18.1.2 SYSTEM DATASET TABLE (SDT)

STG places information in the SDT for datasets being transferred to or from a front-end system concerning block size, processing direction, open/close status, etc.

18.1.3 STAGER STREAM TABLE (SST)

The SST resides in the Link Interface Table Extension (LXT). There is one SST for each stream. This table can be divided into the following sections.

- The first section is accessed by STG only.
- The second section is accessed by both STG and SCP.
- The last section is accessed by SCP only.

The information within this table is interlocked by the task message codes. Both STG and SCP have a message code field in the table. The table is free for use by a task if the field for that tasks message code is null or 0.

18.2 OVERVIEW OF STG PROCESSING

STG is activated for dataset transfers taking place between Cray mainframes and front-end systems. The STG task is dormant when no datasets are being transferred. The protocol message exchange is the responsibility of SCP and the front-end station.

The actual link-level communication between the mainframe and the front end is handled by the EXEC Front-end Driver (FED). The FED and SCP handle the protocol and message exchange. The STG task is responsible for disk transfers of data being staged to and from the front-end systems. STG exchanges data segment buffers with SCP. SCP coordinates the actual reading and writing of the buffers with messages to and from the front-end system.

SCP requests STG processing for active data streams. These requests are made during processing of input and output stream control bytes (SCBs), and the receipt of a dataset header or segment. The number of outstanding requests that STG can have is limited only by the number of SSTs available, not by the 16 stream per LXT limitation of the protocol.

While STG responses to SCP requests normally determine output SCB states and possibly message codes, a lack of response does not keep SCP from sending a message. Each LCP received from a front end causes the active SSTs associated with that front end to be checked. As SCP requests are processed by STG the responses are incorporated into the outgoing messages.

It is not always possible to complete the SCP request before the next LCP is sent because disk reads and writes, PDM requests and space for buffers are not always available. Even if these functions could keep up, a single request to STG from SCP could generate several DQM and PDM requests from STG. The processing of the SCP request will be dropped each time a request to another task is made. The reply from DQM or PDM will reinitiate processing of the initial request. STG does not reenter and process the request from where it left off. Instead it follows the processing path for the request skipping previously completed portions of the request by using flags that indicate the processing is complete. A flag may be an address of a table or buffer. The presence of the address indicates that processing of that part of the request has completed.

STG processing consists of the following sides:

- The input side, datasets being transferred from front ends to Cray mass storage.
- The output side, datasets being transferred from the Cray mass storage to the front end.

The processing side that is being requested is determined by the SST type. SSTs are preset by SCP for either input or output. For each side there are three identifiable phases. The first phase is a startup phase, the second phase is the transfer of the data, and the final phase involves the termination of the transfer. A particular phase of processing is requested by SCP through the SCP message request codes.

The startup phase is mainly concerned with allocating buffers for the transfer and table initialization. The transfer phase involves the reading and writing of disk, and the exchange of segment buffers with SCP. The termination phase requires the closing of files and releasing of buffers.

The errors that occur during the processing include error replies from DQM and PDM, and Postpone and Cancel requests from SCP. In all cases the required processing is to stop the transfer. The processing of errors for input is done by a special error handler. The output side termination process takes care of output errors. The input side requires separate processing because the saving of the dataset is not done. The dataset instead must be deleted. The output side must close the transfer just as it is closed for successful transfers. The error handling is done in much the same manner as the termination phase except that the error handler always sends a successful reply to SCP.

18.2.1 INPUT PROCESSING

The input startup phase is entered when a Start message request code is received by STG.

1. If the dataset already exists, set an End message reply code to terminate the transfer and exit.

2. Allocate an initial segment buffer. If the segment buffer cannot be allocated, set a Buffer Wait message reply code and exit. SCP will re-issue the Start request at a later time.

3. Allocate the initial disk buffer. If no space for the buffer can be found, then release the segment buffer also to prevent buffer deadlock.

The input transfer phase is unlike the others because it is repeated for each segment received. The other phases are processed once. The process buffer, PRBF, message request code initiates this phase. When this code is issued, SCP removes the address of the empty segment buffer from the SST and replaces it with the address of the full segment buffer just received. In this manner, SCP is able to keep one empty segment buffer for each front end so that there will be a buffer for the next segment or request.

1. If the current disk buffer is empty, allocate a new one. This reallocation allows the buffer management routine to pack buffers in upper memory. Note that the reallocation of a buffer should always return at least the same buffer so that no Buffer Wait message reply is necessary.

2. Move data from the segment buffer to the disk buffer. When the disk buffer is full, a write to disk is initiated.

3. If there is data left in the segment buffer, the status is set to busy while the disk write completes. If no data is left in the buffer, then the segment buffer is released and reallocated. This reallocation also forces buffer packing.

Special processing considerations during this phase involve determining if the file is an input job. Input jobs require job card validation. Validation is done by calling the job card processor IND.

The input termination phase is initiated by an End message code from SCP. The End message is requested in response to an End from STG, indicating the end-of-data.

1. Any data in the segment buffer is copied to the disk buffer and a write issued to flush the buffer.

2. The disk buffer and segment buffer are released.

3. If the dataset transfer is from an ACQUIRE or FETCH, exit.

4. A Permanent Dataset Definition (PDD) is allocated. If no buffer space is available a Buffer Wait message reply is set. The SCP message request must be re-issued at a later time.

5. If the dataset is a job, then a job sequence number must be assigned, and the Job Class Manager called to assign a class.

6. If the dataset is a job, then a PDM function request is set to save the input dataset.

7. If the dataset is not a job, then a PDM function request is set to save the user dataset.

8. The PDM function is issued and the status is set Busy. Control is given up until the PDM request completes.

9. If the PDM request is successful, and the file is not a job, and there are DATs assigned in STP, then these are released.

10. If the PDM status indicates the dataset already exists, the message reply to SCP, is set to END

11. If the PDM status indicates any other error, then the message reply is set to CAN. CAN forces a Cancel request from SCP.

Input error handling is initiated by a Postpone or Cancel message request from SCP.

1. The segment buffer is released.

2. If I/O is not busy, then release the disk buffers. If I/O is active, then the request will be processed when I/O terminates.

3. If DATs are allocated, then call DQM to release them. The status is set to busy and control given up until a reply is received from DQM.

4. The Acknowledge message reply code to SCP indicates completion of the error processing.

18.2.2 OUTPUT PROCESSING

The output startup phase is initiated by a Start message request code from SCP.

1. Allocate a segment buffer. If no space is available, then a Buffer Wait reply code is returned to SCP and the Start must be re-issued.

2. Set parameters in the SDT for reading the dataset.

3. Allocate the disk buffer. If no space is available, then a Buffer Wait reply code is returned to SCP and the Start must be re-issued. The segment buffer is also released to prevent deadlock.

4. A disk read is initiated. This read will be active while SCP processes the reply and possibly when the next LCP arrives and SCP issues the first Process Buffer request.

The output transfer phase is repeated each time a Process Buffer is issued by SCP. The phase is repeated until the file is transferred or an error occurs.

1. Reallocate a segment buffer if the current buffer is empty. This reallocation is done to pack down the segment buffers in upper memory.

2. Compute the number of words in the disk buffer and then move all the data that will fit into the segment buffer.

3. If the segment buffer is filled a complete status is set, otherwise, a busy status is set.

4. If EOI is reached on the file, any remainder in the segment is set to 0, and the SST status is set to indicate EOD. The return status is set to indicate a Buffer Ready. The SST status will be the response to the next Process Buffer request.

5. If the disk buffer is empty, it is reallocated to allow for packing of the buffers in high memory.

6. If EOI will be reached by the next read to fill the disk buffers, EOI is set in the SST.

7. A disk read request is issued and the status set to busy until the request can complete.

Output termination phase

The termination phase is initiated by an END, CAN, or PPN message request code from SCP.

1. Release any disk or segment buffers that are allocated.
2. Allocate a PDD. If no memory is currently available, then return a Buffer Wait reply to SCP. SCP re-issues the request later.
3. Issue a PDM request to delete the output dataset. Set the status to busy until the PDM request is complete.
4. If there are other active disposes for this file, then the DATs are released.
5. If there are no other outstanding disposes for the file, then the file is released by making a DQM request. The status is set to busy until the DQM request completes.
6. The PDD used to delete the output dataset is released.

18.3 SCP/STG COMMUNICATION

Like the front-end protocol, the communication between SCP and STG is two way alternate. The SCP task makes requests through the SCP message code field in the SST and STG replies to the request in the STG message code field. This communication is not a protocol, however, but a request/reply mechanism. Each request requires a certain type of processing and one of a limited set of replies. The reply indicates the state of the request if it could not be completed and success or failure if complete.

Under certain circumstances, an STG response forces a specific next request from SCP. This is caused usually by error conditions but is also caused in output of datasets when the end of the dataset is reached. The STG response indicating success and end of data causes the next SCP request to be a termination request.

The following sections describe the SCP message request codes, STG message response codes, valid responses and next requests, and finally termination considerations.

18.3.1 SCP MESSAGE REQUEST CODES

The SCP message code field in the SST contains a request for processing. The five valid message codes and STG processing for each are:

SCP message code   STG processing

STRT   Start this stream. On input, requires checking to see if the dataset already exists. If no dataset exists, a segment buffer is allocated for the stream. On output, a segment buffer and disk buffers are allocated. The filling of the disk buffers is initiated.

PRBF   Process a buffer. This message code requests STG to process the segment buffer pointed to in the SST. This message code is used both on input and output. If a dataset is being input, this code means a full buffer is pointed to by the segment buffer pointer in the SST. If a dataset is being transferred out, STG will return the empty segment buffer pointed to by the segment buffer pointer, allocate a new segment buffer, and begin filling it.

END   END this stream. For input datasets this means the file has completed transferring from the front end. For output datasets this means that SCP acknowledges STG's message END.

CAN   Cancel this stream. This message code to STG normally occurs when the station requests that the stream be canceled or the operator kills the job. The other way that this message code can appear is in response to a cancel by STG. STG sends a cancel when there is an I/O error on a file or an error saving the dataset.

PPN   Postpone this stream. This message code occurs when the station either postpones the dataset or master clears the stream.

18.3.2 STG MESSAGE REPLY CODES

STG issues the following message codes:

STG message code STG processing

ACK Acknowledge the last request. This response acknowledges an END, on either input or output, a CAN and a PPN. This is both the initial and terminal state of STG message code for any file transfers.

BFRD A buffer is ready. This message code informs SCP that STG has completed processing the current buffer pointed to by the segment buffer pointer in the SST. For input datasets the pointer would point to an empty buffer, for output datasets to a full buffer.

BFW Buffer wait code. No buffer space is currently available. The request processing could not be completed. The request must be re-issued. The buffer management routine will eventually make space for the buffer, either through buffers released by other transfers completing or by requesting that JSH make more buffer space available.

END End this stream. STG issues this message code under two conditions. The first is when an input dataset already exists, causing SCP to cancel the stream. The second is to indicate the end of an output dataset. This second case is a normal termination.

CAN Cancel this stream. STG issues this message code for I/O errors on output and errors saving a dataset. CAN is one of two ways that STG can initiate abnormal termination. It is SCP's responsibility to ensure that the stream is terminated by the front end and by STG.

PPN Postpone this stream. STG issues this message code only when an I/O error occurs on input. The reason is that it is possible when reallocating disk space and restarting the transfer that the error will not occur. However, it is the responsibility of the front end and SCP to reinitiate the transfer.

18.4 STG BUFFER MANAGEMENT

STG uses the common subroutine BFMAN, buffer manager, to allocate and deallocate segment buffers, disk buffers, and temporary PDDs. The subroutine uses memory allocated in upper memory for these buffers. The buffers are allocated from the buffer zone on a first fit basis. STG reallocates buffers after each use, that is, after a buffer has been emptied. This allows BFMAN to pack buffers together thereby allowing unused memory to bubble up to the top of the buffer zone.

The initial buffer size is controlled by an installation parameter. The BFMAN subroutine allocates buffers from this area until no more buffers are available. BFMAN can then request that JSH decrease MEMMAX, thereby making the buffer area larger. This additional buffer space can be used for allocating more buffers. If BFMAN detects a large unused segment of the buffer space, then JSH can be called to increase MEMMAX, thereby making the buffer area smaller.

The JSH request is made as a task request, since SCP also uses BFMAN to allocate an initial buffer for each active LXT. SCP always uses task requests to communicate with JSH. The response from JSH is either an error reply or an affirmative reply. The error reply would occur due to some error in the request. The affirmative reply can actually represent two conditions. The first condition is where JSH has actually moved MEMMAX, and BFMAN can allocate buffers from the enlarged buffer zone. The second instance of affirmative reply is when JSH must roll jobs to complete the request. In this circumstance JSH will actually queue the request for processing at a later time. It is the responsibility of BFMAN to compare the new requested MEMMAX to the current MEMMAX to determine when the request is complete. This comparison will be made each time BFMAN is called with a request to allocate a buffer, and has an outstanding JSH request.

It is possible to continue processing without having JSH allocate more memory than initially exists. It is even certain that in many cases JSH will be unable to respond to the request by immediately moving MEMMAX. For input datasets this would mean suspending the stream until another transfer terminated and a buffer was freed. For output datasets the SCB state would remain SND but no data segments would be sent until another stream terminated and buffer space was freed. In both instances SCP would request STG action when preparing an LCP to output to the front end.

18.5 MESSAGE REQUEST CODES AND VALID RESPONSES

The following is a description of the codes used in fields SCPC and STGC in the SST.

Message codes and mnemonics are as follows:

Mnemonic	Significance
ACK	Acknowledge
BFRD	Buffer ready
BFW	Buffer wait
CAN	Cancel
END	End stream
PPN	Postpone stream
PRBF	Process buffer

Valid SCP and STG request and response codes are as follows:

Request codes for SCP	Responses by STG
STRT	BFRD, BFW, END, CAN
PRBF	BFRD, BFW, CAN, END (Output only)
END	ACK, BFW, CAN (Input only), END (output only)
CAN	ACK
PPN	ACK

Response codes for STG	Succeeding responses by SCP
ACK	STRT
BFRD	PRBF, CAN, END, PPN
BFW	PRBF, CAN, END, PPN
END	END, CAN, PPN
CAN	CAN
PPN	PPN

The results of any file transfer will be that the STG message code becomes an ACK, and the SCP message code is clear or zero. The STG code is preset to ACK by SCP at logon.

18.6 DATASET STAGING EXAMPLES

The following examples show the sequence of events during staging to and from COS. These examples are of a single file being transferred. No other activity, (commands, displays, or error recovery) is presented.

The following abbreviations are used:

Abbreviation	Significance
EOI	End of information
SCB	Stream control byte
SCPC	SCP code; field SCPC in the SST.
STGC	STG code; field STGC in the SST.

Staging in a dataset:

Front End	SCP	STG
SCB=RTS		
	SCB=RCV	
	Move info to SDT	
Send dataset header		
SCB=SND		
Send a segment		
	SCPC=STRT, STGC=0	
	Activate STG	
		Allocate segment buffer
		Allocate disk buffer
		STGC=BFRD, SCPC=0
SCB=SND		
	SCB=RCV	
SCB=SND		
Send a segment		
	Swap SST & LXT segments	
	Store SGBC in SST	
	SCPC=PRBF, STGC=0	
	Activate STG	
		Move data to disk buffer
		STGC=BFRD, SCPC=0
	SCB=RCV	
SCB=SND		
Send a segment		
	Swap SST & LXT segments	
	Store SGBC in SST	
	SCPC=PRBF, STGC=0	
	Activate STG	
		Start move to disk buffer
		Write disk buffer when full
	SCB=SUS	
SCB=SND		
		Reallocate disk buffer
		Finish move to disk buffer
		Reallocate segment buffer
		STGC=BFRD, SCPC=0
	SCB=RCV	
SCB=END		
Send a segment		
	Swap SST & LXT segments	
	Store SGBC in SST	
	SCPC=END, STGC=0	
	Activate STG	

Front End	SCP	STG
		Move data to disk buffer Write disk buffer
	SCB=SVG	
		Release disk buffer Save dataset Release segment buffer STGC=ACK, SCPC=0
SCB=END		
	SCB=SVD	
SCB=IDL		

Staging out a dataset:

Front End	SCP	STG
	SCB=RTS	
SCB=RCV		
	Move SDT info to header SCB=SND dataset header	
SCB=SUS		
	SCB=SND	
SCB=RCV		
	SCPC=STRT, STGC=0 Activate STG	
		Allocate segment buffer Allocate disk buffer Begin reading into disk buffer
	SCB=SND	
		Move data to segment buffer Store SGBC in SST STGC=BFRD, SCPC=0
SCB=RCV		
	SCB=SND Send a segment	
SCB=RCV		
	SCPC=PRBF, STGC=0 Activate STG	
		Reallocate segment buffer Start filling segment buffer Reallocate disk buffer Set flag when reading to
EOI		
		Begin reading into disk buffer
	SCB=SND	

Front End	SCP	STG
		Store SGBC in SST
		STGC=END, SCPC=0
SCB=RCV		
	SCB=END	
	Send a segment	
SCB=SVG		
	SCB=END	
SCB=SVD		
	SCPC=END, STGC=0	
	Activate STG	
		Release segment buffer
		Release disk buffer
		Delete output dataset
	SCB=IDL	
		Release output dataset
		STGC=ACK, SCPC=0

18.7 DATASET TRANSFER TERMINATION PROCESSING

The following tables are an overview of stream termination processing. SCP always controls the start of stream initiation and termination, which is a reflection of the STG status as subtask. In any kind of termination, STG responds with an ACK, acknowledge. The following tables show all abnormal termination cases and processing. The termination cases with an STG message code are those in which STG found the error, or for an output dataset transfer the end of information. The STG message code occurs first. In these cases the next SCP message code request is shown under SCPC. The SCP message code is the master request that causes the stream termination. The STG response to this request is ACK, acknowledge.

Input dataset:

Reason for termination	STGC	SCPC	Action on stream	Action on job	Messages
MC or PPN by station		PPN	IDL		
CAN by station		CAN	IDL	Abort	CAN by FE
END from station		END	SVD	Resume	Save/Acquire OK
Operator kill		CAN	CAN	Abort	CAN by COS
Dataset already exists	END	CAN	CAN	Resume	
I/O error	PPN	PPN	PPN		
Error saving dataset	CAN	CAN	CAN	Abort	CAN by COS

Output dataset:

Reason for termination	STGC	SCPC	Action on stream	Action on job	Messages
MC or PPN by station		PPN	IDL		
CAN by station		CAN	IDL	Abort	
Operator kill		CAN	CAN	Abort	CAN by COS
End of data	END	END	END	Resume	Dispose OK
I/O error	CAN	CAN	CAN	Abort	CAN by COS

Termination conditions	Stager processing	SCP processing
Job in OK	Assign a JSQ Call PDM with PMFCSI	Move SDT to Q@INPUT
Dataset in OK	Call PDM with PMFCSU Release DATs in STP	Move SDT to Q@AVAIL
CAN or PPN on input	Call DQM with DELOCATE	Move SDT to Q@AVAIL
PPN on output	No processing	Move SDT to Q@OUTPUT
CAN or END on output	Call PDM with PMFCDO	Move SDT to Q@AVAIL
If not multitype	Call DQM with DELOCATE	
If multitype	Release DATs in STP	

FLUSH VOLATILE DEVICE (FVD) 19

The Flush Volatile Device task (FVD) performs one function which is the backing up of information contained on volatile devices. With the advent of enhanced Buffer Memory and the Solid-state Storage Device (SSD), it became possible to lose information between the time the system is shut down and the subsequent startup due to the volatile nature of these devices. FVD provides the mechanism to back up this information and in conjunction with Startup, which restores the information to the device, prevents the loss of information.

19.1 FVD INTERFACE WITH OTHER TASKS

FVD is called only by SCP through TSKREQ, with the device name of the device to be flushed in INPUT+0. This request is made in conjunction with a received operator function request of FLUSH.

Format:

	0 8 16 24 32 40 48 56 63
INPUT+0	DVN
INPUT+1	Zero

Field	Word	Bits	Description
DVN	INPUT+0	0-63	Device name of the device to be flushed

After determining that the flush request is correct and that all conditions are met so the flush can proceed, a response is sent to SCP through PUTREPLY.

Format:

	0 8 16 24 32 40 48 56 63
OUTPUT+0	STATUS
OUTPUT+1	DVN

Field	Word	Bits	Description	
Status	OUTPUT+0	0-63	Error	Status
			0	No error; request accepted.
			1	Illegal request
			2	Device name not in EQT
			3	Attempt to flush nonvolatile device
			4	Wrong device type to be flushed
			5	No memory pool space available for flush buffers
			6	Error status from PDM on access

FVD also communicates with the PDM task to access the backup dataset and with the DQM task to read the information from the device and write it to the backup dataset.

19.2 SYSTEM TABLES USED BY FVD

FVD uses the following system tables:

EQT	Equipment Table
DRT	Device Reservation Table

FVD uses the EQT to determine physical characteristics about the device being flushed.

FVD also uses Memory Pool One to provide space for its input/output buffers.

19.3 FVD GENERAL FLOW

The FVD general flow is:

1. GETREQ is called to obtain the request.

2. The device name is checked against the EQT to insure that it is a valid request.

3. PDM is called to access the corresponding backup dataset.

4. Memory Pool One space is allocated for the I/O buffers.

5. DQM is called to read information from the device and write to the backup dataset.

6. A pseudo DRT is maintained to indicate any allocation units that could not be backed up.

7. A header is written to the backup dataset; it contains information showing the pseudo DRT and that a flush has been performed.

19.4 INTERACTION BETWEEN FVD AND STARTUP

Even though FVD and Startup do not directly communicate, they still must interact to accomplish the flush/restore procedure. Startup must create the backup dataset $*dname*, where *dname* is the device name, and FVD must write a header to the backup dataset so Startup can recognize that a flush has occurred and can use the pseudo DRT to determine what flushed information is valid.

CONTROL STATEMENT PROCESSOR (CSP) 20

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.

20.1 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:

DSP	Dataset Parameter Area
JCB	Job Communication Block
LFT	Logical File Table

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

20.1.1 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, through EXP, makes a DSP for $OUT, $IN, and any other datasets known to the job.

20.1.2 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 LFTs in the JCB. Other information is also be maintained there.

20.1.3 LOGICAL FILE TABLE (LFT)

The Logical File Table (LFT) is located at the address specified in the JCLFT field of the JCB. CSP, through EXP, makes entries for $OUT, $IN, and any other datasets or aliases known to the job. The LFT entries point to the DSPs.

20.2 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 to CSP from the first file of the dataset named *jobname*. 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.

20.2.1 CSP LOAD PROCESS

During system generation, a copy of CSP is appended to the end of the STP image. During system startup, this copy of CSP can remain in memory immediately following the end of non-Startup code, or it can be written to disk by Startup. The placement of CSP is determined by an installation parameter. The installation can cause Startup to make multiple copies of CSP to reduce conflicts when loading from disk if it chooses to make CSP disk resident. When the job is submitted, Job Scheduler (JSH) allocates memory for the job, sets up the Job Table Area (JTA), and causes the Exchange Processor (EXP) 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.

20.2.2 ENTRY AND EXIT CONDITIONS

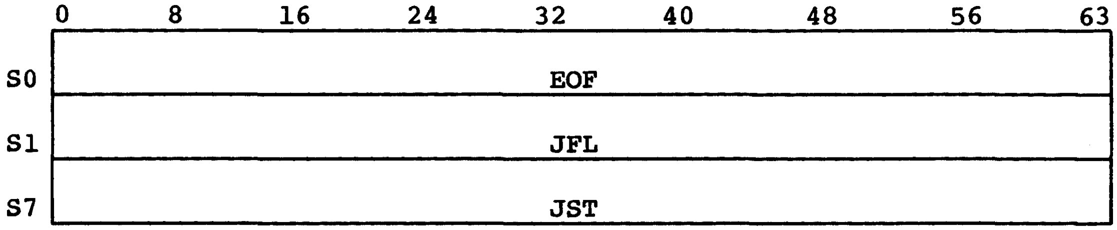
CSP assumes that a control statement file for the job has been staged from the front-end processor to the COS mass storage as the first file of the dataset *jobname* (*jobname* is specified with the JN parameter of the JOB control statement). Control statements are passed by STP from the disk to CSP using the control statement buffer (CSB) of the JTA as the input buffer.

Also, if it is the first time into CSP, register S7 contains a status from the job's input System Dataset Table (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

When control returns to CSP, status is passed in registers S0 and S1. The EOF status for $CS is passed in S0. The status is equal to 1 (EREFR) only at absolute job end, when CSP writes the trailer messages to $LOG. The EOF status is checked only if a control statement is not in the JCB buffer.

The job initiation status is passed in S1. It is 1 for job initiation and 0 thereafter. If (S1)=1, (S7) is significant.



Field	Bits	Description
EOF	0-63	$CS end of file status: 1 EREFR; absolute job end. 0 Not end of file
JFL	0-63	Job Initiation flag: 1 Job initiation 0 Subsequent entry
JST	0-63	Job status at initiation (recovery, etc.); significant only if JFL is 1.

Exit conditions

Exits from CSP are generally through the normal exchange sequence described in section 8. The exit and reentry conditions for the system calls are documented with the system task calls. Reentry 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 call.

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 reentered for that job.

20.2.3 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. Next, the control statement file ($CS) undergoes block validation. If not successful, the job terminates. LFT entries, DSPs, and I/O buffers are made for $OUT and $IN. Unit names 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 statement. Sometimes this message represents a fatal error from Startup, in which case CSP terminates the job immediately.

20.2.4 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 Library Reference Manual, CRI publication SR-0014. In the cracked format, the parameter keywords and values are available for processing by CSP, by system-supplied programs, or by user-supplied programs.

20.2.5 PROCESS STATEMENTS

Every statement is a user's request for some action and is associated with either a system verb, a dataset verb, or a library-defined verb.

System verbs are processed by the system (CSP and/or STP).

Dataset verbs are processed by loading a program into the user field and then executing it. The dataset is either local to the job or resident in the System Directory (SDR). A library-defined verb corresponds to an entry in a named library which is either loaded into the user field and executed or treated as the current control statement file.

System calls

In processing control statements, CSP makes frequent system calls to STP. These calls are described in section 8.

Parameters

Job processing requires assigning values to many parameters. Most parameters 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.

20.2.6 ADVANCE JOB

A job step, the result of a control statement, consists of the execution of a program in the user field. The program may be CSP, or it may be a user-called program which is initiated by CSP and/or LDR. CSP advances the job, step by step, in the sequence specified by the control statement file. These steps are summarized under CSP Step Flow.

20.2.7 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 or if the control statement file contains block structure errors, 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 (EXP) 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 never seen unless an error occurs.

The job is ended if there is no EXIT control statement in the remainder of the control statement file.

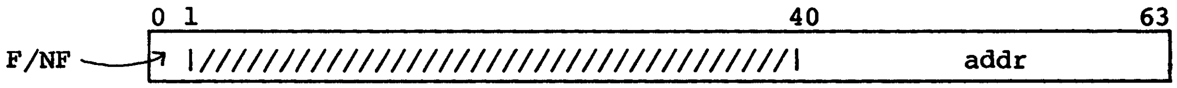
20.2.8 END JOB

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 ($LOG), and CSP makes a job termination system call, F$TRM. The Exchange Processor (EXP) completes job termination. $LOG is written to $OUT; $OUT is closed; buffers for datasets which are open in write mode and are permanent or have a disposition code other than scratch are flushed; the name of the $OUT dataset is changed to the job name; and the dataset is routed to the front-end processor. Finally, the user area is released to the system.

20.3 RECOVERY STATUS MESSAGES

CSP can 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 nonfatal, CSP continues normally.

Message control word format:



Field	Bits	Description
F/NF	0	Flag: 1 Fatal 0 Nonfatal
addr	40-63	Address of message text. The message must be terminated by a zero byte.

20.4 CSP STEP FLOW

The system processes jobs and errors as outlined below and diagrammed in figure 20-1.

Initiate the job:

1. Copy system bulletin to logfile if it exists and is required by the installation.

2. Enter system header into logfile.

3. Process JOB statement, ending job if any errors encountered. Skip this step if job is interactive.

4. If CSP received recovery status:

   a. Issue corresponding message to logfile.
   b. Terminate job if status indicates to do so.

5. Validate the structure of $CS if the job is not interactive; terminate the job if any errors found.

6. Assign the datasets $IN and $OUT.

7. Jump to job advancement.

Process control statements:

1. Examine JCB for a control statement. If there is none, request one through F$GNS; if there are no other statements or the system has requested termination, end the job.

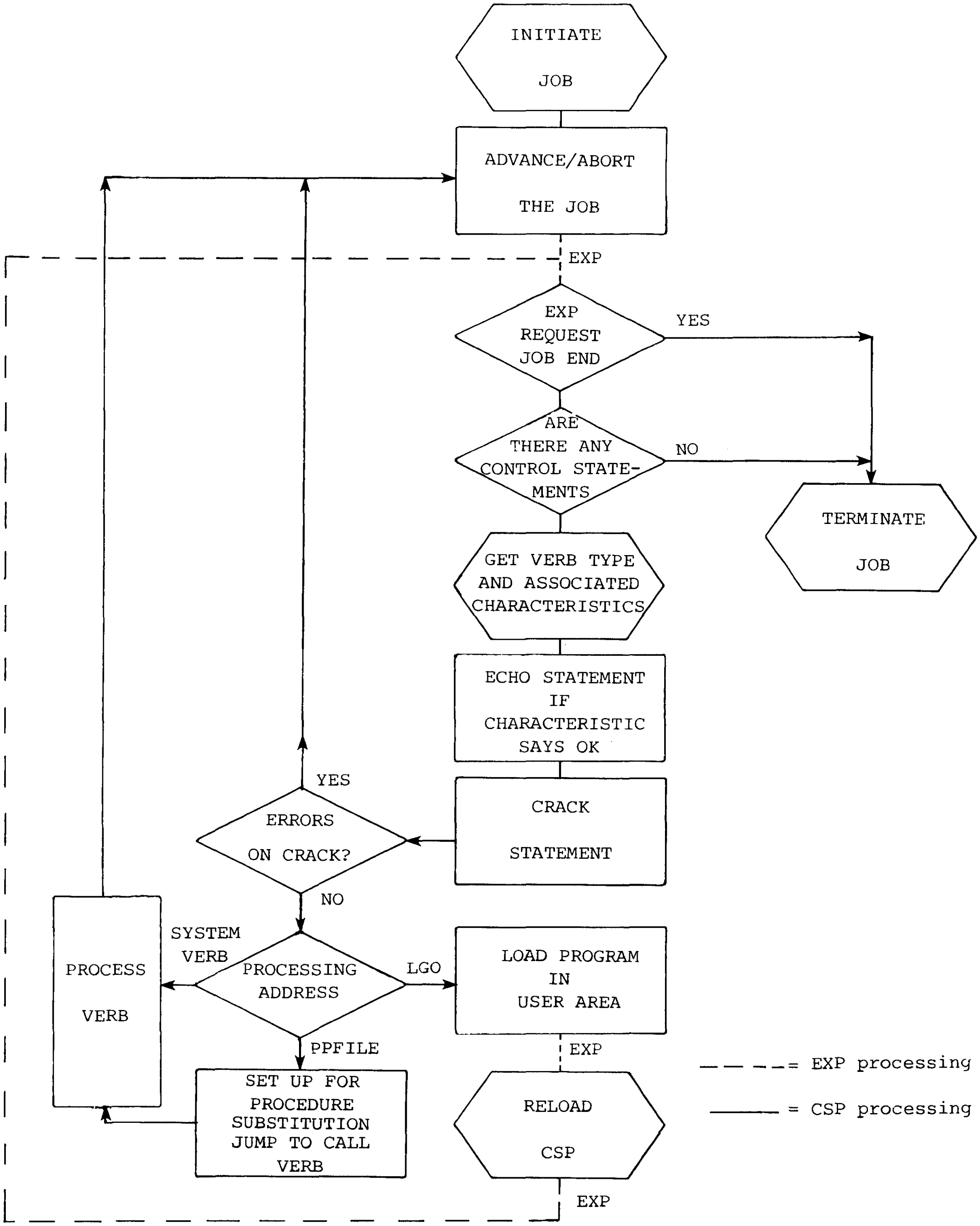


Figure 20-1. CSP control statement flow diagram

2. Initialize verb characteristics; such as aborting when syntax errors are encountered, echoing statement to logfile, and processing of apostrophes and parentheses during statement cracking.

3. Get verb from statement; determine if it is a system verb. If so, get the defined characteristics and the processing address from the system verb table and continue with step 7.

4. Determine if verb is a local dataset. If so:

   a. Set processing address to LGO.
   b. Set up for entry to LGO; open the dataset.
   c. Set verb characteristics to defaults: echo statement before cracking; abort, if errors encountered during cracking; delete apostrophes from strings they delimit.
   d. Continue with step 7.

5. Determine if verb is an entry in one of the accessible libraries in the library searchlist. If so, process according to the entry type:

   Absolute entry:

   a. Set processing address to LGO.
   b. Position library to selected entry.
   c. Set up ODN and DSP to that of current library.
   d. Use default verb characteristics.
   e. Continue with step 7.

   Procedure entry:

   a. Position library to selected entry.
   b. Set processing address to PPFILE.
   c. For verb characteristics: apostrophes are retained in the strings they delimit (for JCL expressions), statements are not echoed.
   d. Continue with step 7.

6. Assume that verb is a dataset resident in the SDR and use default verb characteristics.

7. Echo statement to logfile if characteristics indicate to do so.

8. Ensure that if accounting is mandatory, the ACCOUNT statement appears after the JOB statement.

9. Crack statement, suppressing abort if an error occurs. If the verb was continued illegally or a cracking error occurred with the verb characteristics indicating abort, advance job through an abort.

10. Jump to processing address determined in steps 3 through 6.

End job by CSP or EXP request:

1. Close punch and plot datasets.

2. If CHARGES program is to execute (installation defined):

   a. Indicate special CSP reentry through a flag in the JCB.
   b. Load CHARGES program from SDR via LGO routine.
   c. Reenter CSP from system to finish termination through F$TRM.

Error Processing:

1. Enter error message into logfile and then end the job, if:

   - First control statement is not JOB,
   - The JOB statement contains an error, and
   - Block structure errors are in $CS.

2. Enter an error message into the logfile and abort the job if an error is detected while processing any other verb (CSP, system, or user).

APPENDIX SECTION

THE COS SECURITY SYSTEM A

Security on the Cray Computer System is the mechanism within COS to prevent unauthorized access and use of user information while allowing the system manager a means of controlling access to the system and its data.

The concept of security is divided into three topics: Direct control of the user, defining and tailoring the specific security system, and actual COS implementation of security.

A.1 THE USER

Every user of COS has the following attributes relating to the COS security system:

- User number. The user number is distinct from the account number. The account number identifies the user for recording resource use; the user number identifies the user for system access purposes.
- Password. The password is used to ensure that only authorized users have any access whatsoever to COS.
- Privilege set (or mask). The privilege set defines exactly what types of access a specific user may have to the various functions and capabilities of COS.

User profiles (user number, password, allowed privileges) are maintained in the $ACCT and $VALIDATION system files. Every user must be identified to the system with an ACCOUNT statement.

A.2 COS SECURITY MANAGEMENT

The COS system manager defines two aspects of the security system: profiles for all valid users of the system, and the privileges of COS itself.

A.2.1 DEFINING USER PROFILES

COS user profiles are set up using the ACCTDEF and PRVDEF utilities described in CRI publication SM-0044, the COS Operational Aids Reference Manual.

ACCTDEF creates and maintains $ACCT, the database of valid *account* numbers and passwords.

PRVDEF creates and maintains $VALIDATION, the database of valid *user* numbers and passwords. PRVDEF also allows the site to define for each user which privileges that user is allowed. Each possible privilege is assigned a privilege flag. These privilege flags are written to $VALIDATION for later use during user validation (ACCOUNT processing).

The passwords stored in the user validation dataset are stored in an encrypted form (when I@CRYPT=1, as described below), thereby requiring both the validation dataset generator (PRVDEF) and the Control Statement Processor (CSP) to have some means of encrypting the supplied passwords before writing them or reading them. Password encryption is actually performed by STP common routine PWENC.

A.2.2 DEFINING SYSTEM PRIVILEGES

The security level of COS as a whole must be defined by the system manager. The four defined levels of security are controlled by the installation parameters I@SLVL and I@CRYPT. Note that in all modes, mere system access is enforced by ACCOUNT processing. The modes available are:

QUIET. All privilege checks are in place and processed, but any illegal requests are processed without notification as if there were no checks being made (I@SLVL is -1).

WARN. All privilege checks are in place and processed, but any illegal requests are honored. However, the user is warned that an illegal request has been made, and all security tracking messages are entered into the system log dataset (I@SLVL is 0).

ABORT. All privilege checks are in place and processed and the user is aborted if an illegal request is received by COS. All security tracking messages are entered into the system log dataset (I@SLVL is +1).

CRYPT. CRYPT is the same as ABORT mode but with the added feature of password encryption (I@CRYPT is 1 and I@SLVL is 1).

Since many of the modules residing in the System Directory (SDR) require privileges which the general user should not be allowed to have, each program residing in the SDR must have its own set of privileges and restrictions defined. This is done at utility generation time using Loader control statement parameters GRANT, SECURE, NOECHO. In fact, some of the system utilities contain information which should be secured, such as dataset passwords. In order to prevent the unauthorized user from seeing this data, these modules should be saved as execute-only datasets. Some of the modules residing in the SDR should be removed and saved using passwords; these include JCSDEF, ACCTDEF, and EXTRACT.

A.3 SECURITY IMPLEMENTATION

In order to properly control the operation of the system and its resources, it will sometimes be necessary to override the security features described here. This is accomplished by identifying the privileged requests and functions in COS and restricting use of these to authorized users. This identification of the user (and the user's privileges) is accomplished through the ACCOUNT statement. This, then, means that an ACCOUNT statement is required for the security mechanism described here to work, since the ACCOUNT program will read the $VALIDATION dataset to obtain the privilege flags defined for each user number. These flags can be set or cleared at any time by the site manager with the PRVDEF utility. When the user is validated at job initiation, the user's privilege flags are moved into the JTA and maintained for the duration of the job.

In order to provide further security for passwords, all passwords are immediately encrypted upon definition, and are forever after stored in their encrypted form. This prevents problems when listings or dumps containing passwords are left in a place available to public or prying eyes. Using 1-way encryption methods, it is possible to secure the passwords even if the encrypted password and the encryption method are known.

The encryption of passwords presents special problems regarding front ends and stations. Passing Cray-encrypted read, write, and maintenance passwords to a station would be meaningless to the front end. For this reason, passwords (the R, W, and M parameters on the ACCESS, SAVE, MODIFY, PERMIT, DISPOSE, and ACQUIRE control statements) are defined to apply *only* to Cray resident datasets. Any passwords needed on the front end can be sent in either the text field or station slot area.

System security entails more than just limiting users to privileged functions to which they are entitled. A means of limiting which users are allowed use of the system must also exist. This limitation stems from factors such as including excessive privileged request violations, user number/password pairs entered incorrectly an excessive number of times, or not changing the password often enough to guarantee the required amount of security. The $VALIDATION dataset contains information necessary to track user system accesses and violations. $VALIDATION is updated each time the user enters the system to record such information as number of system accesses, date and time of last access, and any other items that the site needs to track. $VALIDATION also contains information as to whether the user is currently disabled from accessing the system and, if so, why, when and by whom.

A.3.1 SECURITY MANAGEMENT UTILITIES

The utilities ACCTDEF, PRVDEF, and LDR are used as described in the previous section to define and manage the security system.

A.3.2 ACCOUNT STATEMENT

The user is introduced to COS through the ACCOUNT statement which validates the user number/password pair and account number/password pair. A job is processed only if the user number/password pair and the account number/password pair (if specified) are valid. ACCOUNT processing is required to read the privilege words for the user from $VALIDATION and request that they be moved to the Job Table Area (JTA) for use throughout the life of the job. At the same time, certain fields in the dataset are updated. Job/user validation includes the following:

1. Default values are provided for user number and privileges.

2. If either accounting or security is enabled, the ACCOUNT statement is required and the following steps take place; otherwise, the user is validated to access the system.

3. Determine the job's user number and password:

   a. If the ACCOUNT statement contains a US parameter, it is used as the user number. If not, the US parameter from the job statement is checked. If present, it is used as the user number; if not, the account number is used.
   b. If the ACCOUNT statement contains a UPW parameter, it is used as the user password. If not, the account password is used.
   c. The results from the above steps must be two nonzero values (that is, there *must* be a user number/password pair) if security is enabled (I@SLVL positive) or the "no password

is necessary" privilege has been defined for the user by PRVDEF. If either the user number or password is not available, the job is terminated.

d. The result is stored for the ACCOUNT program.

4. ACCOUNT processing is performed:

   a. The ACCOUNT module is loaded from the SDR with SCPRIV privileges.
   b. An F$PRV request is made to set the system dataset ownership value.
   c. The system $VALIDATION dataset is accessed uniquely (for write permission).
   d. The user number entry is located and the user number/password pair is checked; if invalid, the job is terminated.
   e. The entry is validated to be enabled; if not, the job is terminated.
   f. The entry is checked for violation count exceeded; if so, the job is terminated.
   g. The entry is checked for password expired; if so, the job is terminated.
   h. The user entry is updated to indicate latest logon date/time and count, and the dataset is released.
   i. The user is validated for system access. An F$PRV request is made to move the privilege words into the JTA and set the dataset ownership value field to the user number.

5. The account number and password are validated:

   a. An F$PRV request is made to set the dataset ownership value to the system value.
   b. The $ACCT dataset is accessed.
   c. The account number/password pair is checked; if they do not agree, the job is terminated.
   d. The user is permitted to access the system. The $ACCT dataset is released.

6. The job is advanced. This resets the dataset ownership value to the user number.

A.3.3 SYSTEM ACTION REQUESTS

The User Exchange Processor (EXP) processes most of the security checks since EXP is the task that provides the interface between what the user wants to do and the rest of the system. The security checking EXP performs is described in section 8.

A.3.4 DATA SECURITY

The COS security mechanism provides data security in several ways.

Password blanking

The Control Statement Processor (CSP) prevents the writing of control statements to the logfile immediately. CSP waits until the statement is cracked by GPARM before echoing it. GPARM is responsible for editing the secure parameters from the control statement. The secure parameters are identified by the parameter definition list of the parameter to be secured (the initial output array address word). The secure parameter is edited out of the control statement buffer before it is written to the logfile (for example, SAVE, DN=*dn*, R=*rpass*, ID=*id*. is echoed as SAVE, DN=*dn*, R=, ID=*id*.).

Control statement suppression

One goal of COS security is to allow the user to execute programs in as secure an environment as possible. One mechanism for proprietary program security is control statement suppression; other mechanisms (secure memory and secure datasets) are described later in this section. The Loader NOECHO directive is used when building a secure absolute program module to suppress writing the current control statement to the user logfile. That is, the control statement that invoked the actual loading into memory for execution is not written to the user logfile by the loader or by CSP.

Password encryption

Password encryption is described at the beginning of this section. Two situations exist where password encryption is bypassed:

- A dataset's password is lost. A user with the SCACES privilege allowed may access a user-saved dataset without passwords which can then be modified to set new passwords.

- PDSDUMP and PDSLOAD need only use encrypted passwords. PDSDUMP and PDSLOAD read encrypted passwords directly from the DSC, so password encryption is redundant for dump and load requests.

Secure datasets

Secure datasets are released at job step advance with their buffers cleared. The automatic release can be overridden with an F$DSD request as described in section 8. Secure modules are created by the Relocatable Loader when the SECURE parameter is specified. Each dataset created while a secure module is executing is released at the end of that job step unless specifically exempted by a F$DSD request.

ADDING A TASK B

Adding a task to the System Task Processor (STP) affects many areas of the system. This section provides a brief guideline for implementing a task consistent with system conventions.

B.1 TASK ID

Each task has a symbolic ID and a numeric ID. The symbolic task ID is the label of a word containing the numeric ID. These IDs may be in noncontiguous words.

When defining a new task, the programmer assigns a 5-character symbolic ID to a word (in the form *xxx*ID, where *xxx* is unique to the task) and defines an integer (0 to 31) to be stored in the word. Usually, the integer represents the next available task number.

Example:

Location	Result	Operand	Comment
1	10	20	35
NEWID	CON	0	New task ID word in STP tables (about STPTAB.407)

Each task added to the system should also have a unique UPDATE deck name. Each task has a separate CAL IDENT, and the name specified on the IDENT statement should usually match the UPDATE deck name. Any references to tables or data areas in the STP tables module require an EXT statement in the task and possibly an ENTRY statement in module STPTAB. Any data areas used only by the new task that need not be printed in the AUTODUMP, should be included in the task code; such areas should not be added to STPTAB.

All modules containing calls to the system error-halt macros (ERR*xxx*) require a statement defining ERROR1 as external (EXT). All modules require EXT statements for the intertask communication modules (CMODs) they use (for example, TSKREQ, PUTREQ, and so on) and for utility routines (such as CHAIN, MEMAL, and so on).

Module STP requires an EXT statement for the address of the first instruction to be executed during initialization of the task. This EXT statement is necessary for the proper creation of the task. An ENTRY statement is required in the task for the symbol specified in the EXT statement.

An ENTRY in the module and a matching EXT in STPTAB should also be provided so that the table in the common data area pointing to the beginning of each task can be expanded to include the new task. The ID word described above must also be defined as an ENTRY, and a corresponding EXT is required in STP.

Other ENTRY and EXT statements can be required by the addition of a new task, depending on the needs of the task.

B.2 INTERTASK COMMUNICATION

The task communication subroutines (PUTREQ, GETREQ, and so on) use the system constant MAXTN (defined in COSTXT) to determine how many tasks exist. When adding a new task, MAXTN must be incremented to reflect the number of tasks in the system. Since the tasks number from 0, MAXTN has the value of the first illegal task number.

Requests and replies are contained in communications modules (CMODs), obtained from a memory pool. When a new task is added, the size of this memory pool should be reexamined.

The CMODs are controlled by the communications module chain control (CMCC). When a new task is added to the system, a CMCC must be created to accommodate communication from all tasks to the new task. Additionally, each existing CMCC must be expanded to facilitate communication between the new task and all previously existing tasks.

The CMCCs are ordered and accessed according to numeric task ID. Therefore, changes must be properly handled. The CMCCs must be structured as illustrated in section 3.

The new task and other STP tasks must use the common communication subroutines (TSKREQ, PUTREQ, GETREQ, PUTREPLY, and GETREPLY) for all intertask communication. Refer to section 3 for a detailed description of intertask communication.

B.3 TASK I/O

Tasks should use Task I/O (TIO) for system I/O on blocked datasets.

B.4 TASK SUSPENSION

Code the task so that it has a single suspend point.

B.5 TASK CREATION

Insert a Create Task request (CTSK) to EXEC into task 0 (deck STP) so the new task can be scheduled for execution. Be sure to use the same task ID for this call that was used in intertask communication.

The CTSK call also generates the task's entry into the System Task Table (STT), sets the task's priority and execution address, and identifies its exchange package. The STT is large enough to accommodate 32 tasks.

The interaction of the new task with the rest of STP is governed by the task priority set on the Create Task call. The higher the task's priority, the more responsive it is to service requests. The COS task priorities ensure minimum response time to external interrupts. Task priorities are generally arranged so those tasks having the least to do with external interrupts have the lowest priorities. The priority of a new task should be chosen to be consistent with this convention. High priority is reflected by a small value in the CTSK priority field.

B.6 TASK EXECUTION

The new task can execute as a result of implicit or explicit calls. If called implicitly, it is triggered by the expiration of a time event or by an interrupt on a pseudo channel. For interrupts to be recognized and correlated with a task, the task must be assigned to the channel through an ARES, FET, or I/O request of EXEC. ARES assignments are usually made during task initialization. The FET and I/O assignments currently are made dynamically by the Station Call Processor (SCP) and Disk Queue Manager (DQM) tasks. It is unlikely any new task would make FET and I/O calls, since by doing so they could cause serious conflicts in the system. Similarly, no new task can make an ARES call unless a new pseudo channel is added.

Task execution is triggered by an explicit call whenever another task makes an RTSS or RTSK call to EXEC for the task. Such calls should use the symbolic name of the task.

B.7 MODIFICATION TO FDUMP

The data file used by FDUMP to direct AUTO dumping should be modified to include the new task so a properly formatted dump can be taken.

GLOSSARY

GLOSSARY

A

Abort - To terminate a program or job when a condition (hardware or software) exists from which the program or computer cannot recover.

Absolute address - (1) An address that is permanently assigned by the machine designator to a storage location. (2) A pattern of characters that identifies a unique storage location without further modification. Synonymous with machine address.

Absolute block - Loader tables consisting of the image of a program in memory, which can be saved on a dataset for subsequent reloading and execution.

Address - (1) An identification, as represented by a name, label, or number, for a register, location in storage, or any other data source or destination such as the location of a station in a communication network. (2) Any part of an instruction that specifies the location of an operand for the instruction.

Allocate - To reserve an amount of some resource in a computing system for a specific purpose (usually refers to a data storage medium).

Alphabetic - A character set including, $, %, @, as well as the 26 uppercase letters A through Z.

Alphanumeric - A character set including all alphabetic characters and the digits 0 through 9.

Arithmetic operator - Part of an expression that indicates action to be performed during evaluation of expression; can be symbolic character representing addition, unary plus, subtraction, unary minus, multiplication, or division.

Assemble - To prepare an object language program from a symbolic language program by substituting machine operation codes for symbolic operation codes and absolute or relocatable addresses for symbolic instructions.

B

Base address - The starting absolute address of the memory field length assigned to the user's job. This address is maintained in the base address (BA) register. The base address must be a multiple of 20_8.

$BLD - A dataset on which load modules are placed by a compiler or assembler unless the user designates some other dataset.

Blank common block - A common block into which data cannot be stored at load time. The first declaration need not be the largest. The blank common block is allocated after all other blocks have been processed.

Block - (1) A tape block is a collection of characters written or read as a unit. Blocks are separated by an interblock gap and may be from 1 through 1,048,576 bytes. A tape block and a physical record are synonymous on magnetic tape. (2) In COS blocked format, a block is a fixed number of contiguous characters preceded by a block control word as the first word of the block. The internal block size for COS is 512 words (one sector on disk). In Cray manuals, the terms tape block and 512-word block are consistently used to distinguish between the two uses.

Block control word - A word occurring at the beginning of each block in the COS blocked format that identifies the sequential position of the block in the dataset and points forward to the next block control word.

BOT - Beginning of tape; the position of the beginning-of-tape reflective marker.

BOV - Beginning of volume. See BOT.

BPI - Bits per inch. COS supports the 1600 and 6250 bpi recording densities.

Buffer - A storage device used to compensate for the difference in rate of flow of data, or time of occurrence of events, when transmitting data from one device to another. It is normally a block of memory used by the system to transmit data from one place to another. Buffers are usually associated with the I/O system.

Buffer Memory - A memory (composed of 64-bit words) in the I/O Subsystem common to all I/O Processors.

C

Call - The transfer of control to a specified closed routine.

Card image - A one-to-one representation of the contents of a punched card, for example, a matrix in which a 1 represents a punch and a 0 represents the absence of a punch. In COS blocked format, each card image is a record.

Catalog (noun) - A list or table of items with descriptive data, usually arranged so that a specific kind of information can be readily located.

Channel - A path along which signals can be sent.

Character - A logical unit composed of bits representing alphabetic, numeric, and special symbols. COS software processes 8-bit characters in the ASCII character set.

Code - (1) A system of character and rules representing information in a form understandable by a computer. (2) Translation of a problem into a computer language.

Common block - A block that can be declared by more than one program module during a load operation. More than one program module can specify data for a common block but if a conflict occurs, information from later programs is loaded over previously loaded information. A program can declare no common blocks or as many as 125 common blocks. The two types of common blocks are labeled and blank.

Conditional control statement block - Defines the conditions under which a group of control statements are to be processed. The statements which define the block and conditions are: IF, ELSE, ELSEIF, and ENDIF.

Control statement - The format, consisting of a verb and its parameters, used to control the operating system and access its products. Directives are used to control products.

Control statement input file - A dataset containing valid control statements as its first file.

$CS - A primary control statement input file.

D

Data - (1) Information manipulated by or produced by a computer program. (2) Empirical numerical values and numerical constants used in arithmetic calculation. Data is considered to be that which is transformed by a process to produce the evidence of work. Parameters, device input, and working storage are considered data.

Dataset - A quantity of information maintained on mass storage by COS. Each dataset is identified by a symbolic name called a dataset name. Datasets are of two types: temporary and permanent. A temporary dataset is available only to the job that created it. A permanent dataset is available to the system and to other jobs and is maintained across system deadstarts.

Dataset name verb - A verb that is the name of a dataset. See local or system dataset name verb.

Deadstart - The process by which an inactive machine is brought up to an operational condition ready to process jobs.

Debug - To detect, locate, and remove mistakes from a routine or malfunction of a computer. Synonymous with troubleshoot.

Delimiter - A character that separates items in a control statement or a directive; synonymous with separator.

Density - See tape density.

Device - A piece of equipment that mechanically contains and drives a recording medium.

Directive - A command used to control a product, such as UPDATE.

Diagnostic - (1) Pertaining to the detection and isolation of a malfunction or a mistake. (2) A message printed when an assembler or compiler detects a program error.

Disposition code - A code used in I/O processing to indicate the disposition to be made of a dataset when its corresponding job is terminated or the dataset is released.

Dump - (1) To copy the contents of all or part of a storage device, usually from internal storage, at a given instant of time. (2) The process of performing (1). (3) The document resulting from (1).

E

End-of-data delimiter - Indicates the end of a dataset. In COS blocked format, this is a record control word with a 17_8 in the mode field.

End-of-file delimiter - Indicates the end of a file. (1) In COS blocked format, this is a record control word with a 16_8 in the mode field. (2) On magnetic tape, this is a tapemark.

End-of-record delimiter - Indicates the end of a record. (1) In COS blocked format, this is a record control word with a 10_8 in the mode field. (2) In an ASCII punched deck, this is indicated by the end of each card.

Entry point - A location within a block that can be referenced from program blocks that do not declare the block. Each entry point has a unique name associated with it. The loader is given a list of entry points in a loader table. A block can contain any number of entry points.

An entry point name must be 1 to 8 characters and cannot contain the characters blank, asterisk, or slash. Some language processors (for example, FORTRAN) may produce entry point names under more restricted formats due to their own requirements.

EOD - End-of-data on tape. The definition of EOD is a function of whether the tape is labeled or nonlabeled and of the type of operation being performed (input or output). When reading a labeled tape, EOD is returned to the user when an EOF1 trailer label is encountered. When reading a nonlabeled tape, EOD is returned when a tapemark is read on the last volume in the volume list for a particular dataset. When writing a labeled or nonlabeled tape, EOD processing is initiated by a write EOD, rewind, close, or release request.

EOF - End-of-file on tape.

EOI - End-of-information; see EOD.

EOR - End-of-record on tape.

EOT - End-of-tape; a status, set only on a write operation indicating sensing of the end of the tape reflective marker.

EOV - End-of-volume. On output, EOV occurs when end-of-tape (EOT) status is returned on a write operation. This status occurs when the EOT reflective marker is sensed by the tape device. For input of a labeled tape dataset, EOV occurs when an EOV1 trailer label is read; for input of a nonlabeled dataset, EOV is returned when a tapemark is encountered and the volume list is not exhausted.

Exchange package - A 16-word block of data in memory which is associated with a particular computer program or memory field. It contains the basic parameters necessary to provide continuity from one execution interval for the program to the next.

Expression (JCL parameter expression) - A series of characters grouped into operands and operators which are computed as one value during parameter evaluation; should be delimited by parentheses.

External reference - A reference in one program block to an entry point in a block not declared by that program. Throughout the loading process, externals are matched to entry points (this is also referred to as satisfying externals); that is, addresses referencing externals are supplied with the correct address.

F

File - A collection of records in a dataset. In COS blocked format, a file is terminated by a record control word with 16_8 in the mode field.

Filemark - See tapemark.

Foreign label - A special condition that can occur during the label scan at the beginning of a tape. If a NOT CAPABLE status is returned on a BOV label scan, TQM declares the tape to be foreign labeled (FRN) which protects a 7-track tape or a 9-track, 800 bpi tape from being accidently destroyed.

Formal parameter specifications - Parameters in a procedure definition which identify the character strings within the procedure body that can be substituted during the procedure's evaluation.

Front-end processor - A computer connected to a Cray mainframe channel. The front-end processor supplies data and jobs to the Cray computer and processes or distributes the output from the jobs. Front-end systems are also referred to as stations in Cray publications.

G

Generic name - Tape resource requirements are expressed using installation-defined generic names. COS supports up to 16 generic names. A generic device name is used to group tape devices according to some common characteristics. Generic names are defined at system assembly time. Installations can define additional generic names according to their needs. Standard COS provides one generic device name:

*TAPE	Device capable of 6250 and 1600 bpi

H

HLM - High limit of memory, the highest memory address available to the user for program and data area.

I

$IN - A dataset containing the job control language statements as well as the source input and data for compilers and assemblers, unless the user designates some other dataset (FT05 for example).

In-line procedure - A procedure defined in a control statement file.

Input/Output - (1) Commonly called I/O. To communicate from external equipment to the computer and vice versa. (2) The data involved in such a communication. (3) Equipment used to communicate with a computer. (4) The media carrying the data for input/output.

Integer constant - Specifies an octal value or a decimal value that can be signed as positive or negative.

Interchange format - One of the two ways in which tape datasets can be read or written. Each tape block of data corresponds to a single logical record in COS blocked format. Interchange format is selected by setting DF=IC when a tape dataset is accessed. As far as I/O routines in the Cray mainframe are concerned, interchange datasets must be in COS blocked format because the COS blocked structure (BCWs and RCWs) is used to describe each tape block read or written. This blocked structure allows the user to write or read variable-length tape blocks at high speed with data resolution to the 88-bit byte level of the tape device. The record control word (RCW) is used to define the tape block length on output and to describe the block length on input. No BCW or RCW ever appears in the data written on the tape.

Interblock gaps - The physical separation between successive tape blocks on magnetic tape.

I/O Subsystem - Part of a CRAY-1 S Series Model S/1200 through S/4400, all models of the CRAY-1 M Series and CRAY X-MP Series of Computer Systems consisting of two to four I/O processors and one-half, one, four, or eight million words of shared Buffer Memory. The optional tape subsystem is composed of at least one block multiplexer channel, one tape controller, and two tape units. The tape units supported are IBM-compatible 9-track, 200 ips, 1600/6250 bpi devices.

Iterative control statement block - Defines the repeated execution of a series of statements if a condition is satisfied.

J

JCL block control statement - A statement in the control statement file that is part of a group of control statements called a block which specifies an action to be taken by COS; the three types of blocks are: procedure definition, conditional, and iterative.

Job - (1) An arbitrarily defined parcel of work submitted to a computing system. (2) A collection of tasks submitted to the system and treated by the system as an entity. A job is presented to the system as a formatted dataset. With respect to a job, the system is parametrically controlled by the content of the job dataset.

Job Communication Block - The first 200_8 words of the job memory field. This area is used to hold the current control statement and certain job-related parameters. The area is accessible to the user, the operating system, and the loader for inter-phase job communication.

Job control statement - Any of the statements used to direct the operating system in its functioning, as compared to data, programs, or other information needed to process a job but not intended directly for the operating system, itself. A control statement may be expressed in card, card image, or user terminal keyboard entry medium.

Job deck - The physical representation of a job before processing either as a deck of cards or as a group of records. The first file of the job dataset contains the job statements and the job parameters which will be used to control the job. Following files contain the program and data which the job will require for the various job control statements. The job deck is terminated by an end-of-data delimiter.

Job input dataset - A dataset named $IN on which the card images of the job deck are maintained. This consists of programs and data referenced by various job steps. The user can manipulate the dataset like any other dataset (excluding write operations).

Job output dataset - Any of a set of datasets recognized by the system by a special dataset name (for example, $OUT, $PLOT, and $PUNCH), which becomes a system permanent dataset at job end and is automatically staged to a front-end computer for processing.

Job step - A unit of work within a job, such as source language compilation or object program execution.

K

Keyword parameter - A string of 1 to 8 alphanumeric characters that consists of a keyword followed by one or more values; identified by its form rather than by its position in the control statement.

L

$LOG - See logfile.

Labeled common - A common block into which data can be stored at load time.

Library - A dataset composed of sequentially organized records and files. The last file of the library contains a library directory. The rest of the files and records, known as entries, can consist of processed procedure definitions and/or relocatable modules. The directory gives a listing of entry names with their associated characteristics.

Library-defined verb - A 1- to 8-character name of a program or procedure definition residing in a library that is a part of the current library searchlist.

Limit address - The upper address of a memory field. This address is maintained in the limit address (LA) register.

Literal - A symbol which names, describes, or defines itself and not something else that it might represent.

Literal constant - A string of 1 through 8 characters delimited with apostrophes whose ordinal numbers are in the range 040_8 through 176_8; value of a character constant corresponds to the ASCII character codes positioned within a 64-bit word; alignment indicated can be left- or right-adjusted and zero-filled or left-adjusted and space-filled; apostrophes remain as part of value.

Literal string - A string delimited with apostrophes which are normally not treated as part of the value, except with JCL block control statements which treat the apostrophes as part of the string value.

Loader tables - The form in which code is presented to the loader. Loader tables are generated by compilers and assemblers according to loader requirements. The tables contain information required for loading such as type of code, names, types and lengths of storage blocks, data to be stored, etc.

Loading - The placement of instructions and data into memory so that it is ready for execution. Loader input is obtained from one or more datasets and/or libraries. Upon completion of loading, execution of the program in the job's memory field is optionally initiated. Loading may also involve the performance of load-related services such as generation of a loader map, presetting of unused memory to a user-specified value, and generation of overlays.

Load point - See BOT.

Local dataset - A temporary or permanent dataset accessible by the user.

Local dataset name verb - A verb that is the name of a local dataset consisting of an alphabetic character followed by 1 through 6 alphanumeric characters. Requests that COS load and execute an absolute binary program from the first record of the named dataset.

Logfile - During the processing of the job, a special dataset named $LOG is maintained. At job termination, this dataset is appended to the $OUT file for the job. The job logfile serves as a time-ordered record of the activities of the job, that is, all control statements processed by the job, significant information such as dataset usage, all operator interactions with a job, and errors detected during processing of the job.

Logical operator - Represents logical function performed on operands on a bit-by-bit basis, returning a 64-bit result; functions are: inclusive OR, intersection, exclusive OR, unary complement.

M

Macro instruction - An instruction in a source language that is equivalent to a specified sequence of machine instructions.

Magnetic tape - A tape with a magnetic surface on which data can be stored by selective polarization of portions of that surface.

Mainframe - The central processor of the computer system. It contains the arithmetic unit and special register groups. It does not include input, output, or peripheral units and usually does not include internal storage. Synonymous with central processing unit (CPU).

Mass storage - The storage of a large amount of data that is also readily accessible to the central processing unit of a computer.

Memory field - A portion of memory containing instructions and data usually defined for a specific job. Field limits are defined by the base address and the limit address. A program in the memory field cannot execute outside of the field nor refer to operands outside of the field.

Multiprocessing - Use of several computers to logically or functionally divide jobs or processors, and to execute various programs or segments asynchronously and simultaneously.

Multiprogramming - A mode of operation that provides for the sharing of processor resources among multiple, independent, software processes. This mode, used by many computing systems, makes most efficient use of a single CPU. In the multiprogramming mode, when several processes are ready to run, should one process be delayed by I/O, for example, another process can immediately be switched in to run on the CPU. In contrast, a system running in monoprogramming mode has only one process ready to run and any delays will leave the CPU idle. Processor resources could include more than one CPU, and in a multiprogramming environment, these multiple CPUs would be shared between multiple, independent software processes.

Multitasking - A special case of multiprocessing, where more than one task can be executing in a user job. When multitasking, there is no guarantee that more than one processor will be allowed to work on the tasks of a given job, no guarantee that the tasks will execute in any particular order, and no guarantee of which task will finish first. In this manual, *multitasking* refers only to user-level tasks (user tasks and user library tasks).

N

Nesting - Including a block of statements of one kind into a larger block of statements of the same kind, such as an iterative block within a larger iterative block.

Not Capable - A tape status indicating the reel currently mounted cannot be read by the control unit and drive. The Not Capable status would be returned if an 800 bpi tape were mounted on a device that supported only 1600 and 6250 bpi, for example. Since it is not possible to read a Not Capable tape to verify label type and contents, COS rejects (unloads) all tapes that return a Not Capable status.

O

$OUT - A dataset that contains the list output from compilers and assemblers unless the user designates some other dataset. At job end, the job logfile is added to the $OUT dataset and the dataset is sent to a front-end computer.

Operand - A character string in an expression that is operated on during evaluation; types are integer constant, literal constant, symbolic variable, and subexpresion.

Operating system - (1) The executive, monitor, utility, and any other routines necessary for the performance of a computer system. (2) A resident executive program that automates certain aspects of machine operation, particularly as they relate to initiating and controlling the processing of jobs.

Operator - A symbolic representation indicating the action to be performed in an expression; types are arithmetic, relational, and logical operators.

Overlaying - A technique for bringing routines into memory from some other form of storage during processing so that several routines will occupy the same storage locations at different times. Overlaying is used when the total memory requirements for instructions exceeds the available memory.

P

$PROC - A dataset to which in-line procedure definitions are written.

Parameter - A quantity in a control statement which may be given different values when the control statement is used for a specific purpose or process.

Parcel - A 16-bit portion of a word which is addressable for instruction execution but not for operand references. An instruction occupies one or two parcels; if it occupies two parcels, they may be in separate words.

Parenthetic string - A string delimited with parentheses instead of apostrophes; parentheses are treated as part of the string when evaluated except when preceded by an initial, parameter, equivalence, or concatenation separator character.

Permanent dataset - A dataset known to the operating system as being permanent; the dataset survives deadstart.

Positional parameter - A parameter that must appear in a precise position relative to the separators in the control statement.

Procedure - A named sequence of control statements and/or data that is saved in a library for processing at a later time when activated by a call to its name by a calling statement; provides the capability of replacing values within the procedure with other values.

Procedure definition - The definition of a procedure that is saved in a library to be called for processing at a later time; if defined in a job control statement is called an in-line procedure definition.

Program - (1) A sequence of coded instructions that solves a problem. (2) To plan the procedures for solving a problem. This may involve analyzing the problem, preparing a flow diagram, providing details, developing and testing subroutines, allocating storage, specifying I/O formats, and incorporating a computer run into a complete data processing system.

Program block - The block within a load module usually containing executable code. It is automatically declared for each program (though it may be zero-length). It is local to the module; that is, it can be accessed from other load modules only through use of external symbols. Data placed in a program block always comes from its own load module.

Program name - Also referred to as IDENT name or deck name, the name contained in the loader PDT table at the beginning of each load module.

Program library - (PL) The base dataset used by the UPDATE utility. This dataset consists of one or more specially formatted card image *decks*, each separated by an end-of-file.

R

Record - A group of contiguous words or characters related to each other by virtue of convention. A record may be fixed or variable length. (1) In COS blocked format, a record ends with a record control word with 10_8 in the mode field. (2) In an ASCII-coded punched deck, each card is a record. (3) For a listable dataset, each line is a record. (4) For a binary load dataset, each module is a record.

Relational operator - An operator that indicates the comparison to be performed between the operands in an expression (-1 for a TRUE result and 0 for a FALSE result); types are equal, not equal, less than, greater than, less than or equal, and greater than or equal.

Relative address - An address defined by its relationship to a base address (BA) such that the base address has a relative address of 0.

Relocatable address - An address presented to the loader in such a form that it can be loaded anywhere in the memory field. A relocatable address is defined as being relative to the beginning address of a load module program block or common block.

Relocatable module - This is the basic program unit produced by a compiler or assembler. CAL produces a relocatable module from source statements delineated by IDENT and END. In FORTRAN, the corresponding beginning statements are PROGRAM, SUBROUTINE, BLOCK DATA, or FUNCTION. The corresponding end statement is END.

A relocatable module consists of several loader tables that define blocks, their contents, and address relocation information.

Relocate - In programming, to move a routine from one portion of internal storage to another and to adjust the necessary address references so that the routine can be executed in its new location. Instruction addresses are modified relative to a fixed point or origin. If the instruction is modified using an address below the reference point, relocation is negative. If addresses are above the reference point, relocation is positive. Generally, a program is loaded using positive relocation.

S

Sector - A physical area on disk equivalent to 512 64-bit words. In COS blocked format, a block is also 512 contiguous words with a block control word as the first word of the block. Therefore, the internal block size for Cray datasets is equivalent to one disk sector. This is the unit of data transfer between the Cray mainframe and the I/O Subsystem.

Separator - Synonym for delimiter.

String - A sequence of characters delimited by apostrophes or parentheses which is to be taken literally as a parameter value; see literal string and parenthetic string.

Subexpression - An expression that is evaluated so that its result becomes an operand.

Substitution parameters - Parameters on procedure definition prototype statement or procedure calling statement which provide replacement values to be substituted during evaluation for strings flagged within the procedure body.

Symbolic variable - A string of 1 to 8 alphanumeric characters, beginning with an alpha character that represents values maintained by COS and/or the user.

System dataset name verb - A verb that is the name of a system-defined dataset in the System Directory Table (SDR); consists of an alphabetic character which can be followed by 1 through 6 alphanumeric characters.

System logfile - A permanent dataset named $SYSTEMLOG.

System task - The tasks comprising the System Task Processor (STP) are referred to as system tasks. STP is described in section 3. The term *task*, as used in this manual, refers to a system task, unless otherwise noted. Likewise, terms such as *intertask* also refer to system tasks.

System verb - Requests that COS perform a function; consists of an alphabetic character which can be followed by 1 through 6 alphanumeric characters

T

Table - A collection of data, each item being uniquely identified either by some label or by its relative position.

Tape block - A group of contiguous characters recorded on and read from magnetic tape as a unit.

Tape control unit - A piece of equipment connected to a block multiplexer channel that provides the capability for controlling the operation of one or more tape devices. Up to four control units may be combined to drive a maximum of 16 tape devices. The control units are cross-connected to all devices. Such a configuration is called a 4x16 (four by sixteen). If one control unit were to be connected to three devices, it would be referred to as a 1x3 configuration.

Tape density (bpi) - The number of bits per inch on magnetic tape. COS supports 6250 bpi and 1600 bpi.

Tape format - The way tape datasets are read or written. In *interchange format,* each tape block of data corresponds to a single logical record in COS blocked format. In *transparent format,* each tape block is a fixed multiple of 512 words based on the density of the tape.

Tape volume - A reel of magnetic tape.

Tapemark - A special hardware bit configuration recorded on magnetic tape. It indicates the boundary between combinations of datasets and labels. It is sometimes called a filemark.

Task - A software process. It is a unit of computation that can be scheduled and whose instructions must be processed in sequential order. See also system task, user task, and user library task.

TDT - Tape Device Table entry. Contains one entry for each device in the configuration. A TDT entry is used to control the activity associated with a tape device and contains the 6-word packet through which requests to the I/O Subsystem are made.

Temporary dataset - A dataset which is not permanent and is available only to the job that created it.

Time slice - The maximum amount of time during which the CPU can be assigned to a job without re-evaluation as to which job should have the CPU next.

Timestamp - A 1-word encoding of the date and time. A timestamp is expressed in units of (nanoseconds/1.024). When used to express a date and time, a timestamp is the number of timestamp units past the system base date (1 January 1973) and the date/time to be encoded. (See STPUTIL subroutines MTTS, TSMT, TSDT, DTTS.)

Transparent format - One of two ways tape datasets are read or written. Each tape block is a fixed multiple of 512 words. Transparent format is the default tape dataset format and is designated by setting DF=TR when accessing a tape dataset. This format produces a fixed-length block dataset (16384 bytes at 1600 bpi or 32768 bytes at 6250 bpi) that may be a COS blocked or unblocked dataset as far as any I/O routines are concerned. The tape subsytem merely takes four (1600 bpi) or eight (6250 bpi) sectors and processes them as one physical tape block. When a short block is read, it is considered to be EOD.

U

Unit record device - A device such as a card reader, printer, or card punch for which each unit of data to be processed is considered a record.

Unload - To remove a tape from ready status by rewinding beyond the load point. The tape is then no longer under control of the computer.

Unsatisfied external - An external reference for which the loader has not yet loaded a module containing the matching entry point.

User library task - The entity created by calling TSKSTART (initiate a task) in the multitasking library. Multitasking in a FORTRAN program is done as user library tasks. That is, when a FORTRAN program creates multiple tasks, the tasks created are user library tasks. User library tasks are created and synchronized by user-program calls to the multitasking library.

The multitasking library scheduler manages (schedules) user library tasks. The library scheduler creates, deletes, activates, and deactivates user tasks as needed; the library scheduler is responsible for assigning user library tasks to user tasks. Within a user job, the user program only knows about user library tasks; EXEC and STP only know about user tasks; the multitasking library scheduler forms the interface between user tasks and user library tasks.

User logfile - A dataset named $LOG created for a job when it is initiated by the Job Scheduler.

User task - The entity referred to in the F$TASK system action request, as described in section 8.1. User jobs are generally unaware of user tasks; user task management requests are usually made by the multitasking library routines.

V

Verb - The first nonblank field of a control statement; specifies the action to be taken by COS during control statement evaluation.

Volatile device - A physical storage device that loses the information stored when power is lost to the device. COS reserves sufficient space on some nonvolatile device or devices to back up all information on a volatile device. The Solid-state Storage Device (SSD) is an example of a volatile device.

Volume - A physical unit of storage media that can be dismounted from a storage device, for example, a reel of magnetic tape.

Volume identifier - Up to 6 alphanumeric characters used to identify a physical reel of tape. On labeled tapes, the volume identifier is actually recorded on tape in the volume header label. Volume identifier is synonomous with volume serial number.

VSN - Volume serial number. See volume identifier.

W

Word - A group of bits between boundaries imposed by the computer. Word size must be considered in the implementation of logical divisions such as character. The word size of a CRAY-1 or CRAY X-MP computer is 64 bits.

INDEX

INDEX

READERS COMMENT FORM

COS EXEC/STP/CSP Internal Reference Manual SM-0040 C

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.

1440 Northland Drive
Mendota Heights, MN 55120
U.S.A.

Attention:
PUBLICATIONS

FOLD

STAPLE

Cray Research, Inc.
Publications Department
1440 Northland Drive
Mendota Heights, MN 55120
612-452-6650
TLX 298444