THE CRAY OPERATING SYSTEM

UNIT 1

CONCEPTS & FACILITIES

TABLE OF CONTENTS

Introduction ii

COS Purpose and Features. 1.1

Hardware Configurations and Characteristics 2.1

Software Components 3.1

Memory Layout 4.1

Mass Storage Organization 5.1

Job Processing Overview 6.1

Memory Management 7.1

Tasks and Multitasking Concepts 8.1

Exchange Mechanism. 9.1

EXEC Purpose and Function 10.1

STP Purpose and Function. 11.1

System Tasks: Purpose and Function. 12.1

CSP Purpose and Function. 13.1

Job Processing Execution Sequence 14.1

INTRODUCTION

AUDIENCE

The Cray Operating System course is intended to provide technical knowledge and skills to Cray Site Analysts and other software personnel who support or maintain COS.

PREREQUISITES

To ensure maximum benefit from this course, the participant should have previously attended the Cray JCL course and the CAL course, or have comparable work experience.

It should be understood that a lack of this requisite knowledge or experience will seriously impair the participant's progress.

COURSE STRUCTURE

The Cray Operating System course is organized into two major parts:

COS I

Concepts
&
Facilities.

Internals.

COS II

Installation
&
Operations.

SKILLS ADDRESSED IN THE COS I COURSE

This two week course is intended for Cray and Customer analysts who are responsible for maintaining, debugging and modifying COS. It presents an overview of the CRI software environment and takes the student through the internal interactions of COS. Dumps are presented to reinforce fundamental COS interactions and to build trouble shooting skills. Some basic operational skills are also covered. With the skills developed in this course, the student is ready to move into the COS II course to develop operational and site management skills.

THESE SKILLS ARE PREREQUISITES FOR THE COS II COURSE.

COS I								
Skills At the end of the course the learner is able to:								
List the general purpose of the Cray Operating System and its primary functions.		8/27						
Identify CRI software functions, characteristics, and components.		8/27						
Describe CRI software external interactions.	8/27							
Describe the CRI software life cycle.	8/27							
Identify the CRI software manuals and the function of each.		8/27						
Describe COS internal interaction.	8/27							
Describe COS external to internal interaction.	8/27							
Read and interpret COS code.		8/27						
Evaluate system performance and reliability.	8/27							
Print formatted and raw dumps.		8/27						
Analyze COS dumps to isolate system malfunctions.	8/27							
Given a problem, state the recommended procedure for reporting and recovering from the problem.	8/27							
Competency Levels	0	1	2	3	4	5	6	7

EVALUATION METHOD

Evaluation of your progress in gaining expertise in these skills is accomplished by assigning a competency level to each skill.

Level	
0	No knowledge and no experience.
1	Has some knowledge and limited experience with this skill, but not sufficient to contribute in a work environment.
2	Can perform some parts of this skill satisfactorily but requires instruction and supervision to perform the entire skill.
3	Can perform some parts of this skill satisfactorily but requires periodic supervision and/or assistance.
4	Can perform this skill satisfactorily without assistance and/or supervision.
5	Can perform this skill with proficiency in speed and quality without supervision or assistance.
6	Can perform this skill with initiative and adaptability to special situations without supervision or assistance.
7	Can perform this skill and can lead others in performing it.

Successfully completing this course should give you a competency level of at least 3 for most skills. Experience on the job will continue to increase your competency level.

LEARNING LOG DESCRIPTION

Progress in your level of competency can be graphed on a learning log.

This is an example of one course participant's learning log.

On the following page is an empty graph for you to use to indicate your current competency level and your continuing progress.

At the completion of the course this learning log on which you've evaluated yourself will be annotated by the course instructor and a copy sent to your supervisor.

LEARNING LOG

COS I									
Skills At the end of the course the learner is able to:									
List the general purpose of the Cray Operating System and its primary functions.		6/11	→	6/15					
Identify CRI software functions, characteristics, and components.	6/11	→	→	→	6/15				
Describe CRI software external interactions.	6/11	→	→	6/15					
F.S. → Describe the CRI software life cycle.		6/11 →	6/15						
Identify the CRI software manuals and the function of each.		6/11	→	6/18					
Describe COS internal interaction.	6/11	→	→	6/19					
Describe COS external to internal interaction.	6/11	→	→	6/20					
Read and interpret COS code.	6/11	→	→	6/20					
Evaluate system performance and reliability.	6/11	→	→	6/21					
Print formatted and raw dumps.	6/11	→	→	6/21					
Analyze COS dumps to isolate system malfunctions.	6/11	→	6/22						
F.S. → Given a problem, state the recommended procedure for reporting and recovering from the problem.	6/11	→	6/13						
Competency Levels	0	1	2	3	* 4	5	6	7	No Basis For Judgement

REFERENCE MANUALS AVAILABLE IN

TERMINAL ROOMS A AND B

1) SR-0000 CAL Reference Manual
2) SR-0009 CFT Reference Manual
3) SR-0011 COS Reference Manual
4) SR-0013 UPDATE Reference Manual
5) SR-0014 Library Reference Manual
6) SM-0036 APML Reference Manual
7) SR-0038 MVS Station Reference Manual
8) SR-0039 COS Message Manual
9) SM-0044 COS Operational Aids Reference Manual
10) SG-0055 TEDI User's Guide
11) SG-0056 SID User's Guide
12) SR-0060 PASCAL Reference Manual
13) SR-0068 VM Station Reference Manual
14) SR-0073 CSIM Reference Manual

SR-0066 SEGLDR

If you have any suggestions for additions to this collection, please mention them to your instructor.

LISTINGS BIN

A bin for printer listings is located in Terminal Room B.

If you remove listings from our local printers (TNGA or TNGB) which are not yours, please be sure to put them into the appropriate place in the bin.

Listings printed in the Mendota Heights building will be brought over by van on a daily basis and distributed to the bin.

Every Friday, listings more than one week old will be discarded from the bin.

COS I DAILY SCHEDULE

Monday	Tuesday	Wednesday	Thursday	Friday
	COS OVERVIEW		Task to Task Communication	DQM, DEC, FVD
	FIELD SUPPORT	EXEC		
COS OVERVIEW				
	COS Internals		SCP, STG	COS - IOS Interaction
	EXEC	STP Common Routines		

Monday	Tuesday	Wednesday	Thursday	Friday
Labor Day				Dump Analysis
PDM	EXP, CSP, USER	TQM		
JSH		Startup	Dump Analysis	Review
				Evaluation
		Dump Analysis		
JCM	MEP, MSG, SPM			

SKILLS ADDRESSED IN THE COS II COURSE

This one week course is intended for Cray and Customer analysts who are responsible for generating, operating and generally managing a COS site. Topics include: System generation; installation and operation; debugging; permanent file maintenance; and defining the system operating environment.

THESE SKILLS ARE PREREQUISITES FOR THE IOS COURSE.

COS II								
Skills At the end of the course the learner is able to:								
Startup/shutdown/dump a CRI system.								
Create and edit parameter files.								
Communicate with COS using IOS station commands.								
Run an interactive COS job using the IOS.								
Select installation parameters.								
Build a COS system.								
Use CSIM to test a COS system.								
Install a COS system.								
Establish system security.								
Establish system accounting.								
Establish a job class structure.								
Establish permanent dataset privacy.								
Build the system directory.								
Use permanent file procedures to maintain the permanent file base.								
Use the SCP debug facility to aid in isolating and validating a system malfunction.								
Install and run on-line diagnostics.								
Competency Levels	0	1	2	3	4	5	6	7

COS II Pre-Course Assignment:

The primary focus of COS II is hands-on experience with the Cray and IOS. Relatively little total time in the course is allocated to lecture. Because the class takes turns in small teams to use the lab, lab time is also at a premium. You will need to do advance preparation in order to get the maximum value out of lab. Keep the COS II objectives in mind during COS I so that you will be aware when the COS I material ties in with them.

Also, please read the following materials in advance, with a focus on preparing for the COS II objectives:

SM-0043

SM-0044

SG-0051

OUTLINE OF COS OVERVIEW

1. COS: General Purpose and Primary Features

2. CRI Software: Functions, Characteristics and Components

   A. CSP

   B. EXEC

   C. STP

   D. System Tasks

   E. Stations

   F. IOS

   G. Libraries, Utilities, and Language Processors

   H. $SYSTXT, $COSTXT, Common Decks

3. A Day in the Life of a Typical Job

4. CRI Software Life Cycle
   (Guest presentation on Field Support)

5. Overview of CRI User Publications

CRAY-1 OPERATING SYSTEM (COS)

- MULTIPROGRAMMING OF USER APPLICATIONS

- SCHEDULING OF APPLICATIONS BY PRIORITY (JOB CLASS)

- MANAGES DISK AND TAPE RESOURCES

- MANAGES FRONT-END COMMUNICATIONS

  SCP

- MANAGES FILE MAINTENANCE

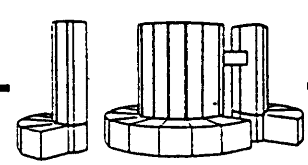
  PDM

- MANAGES PROGRAM MAINTENANCE

- CAPABLE OF MODIFICATION AT STARTUP

X-MP SERIES SOFTWARE

- CRAY-1 SOFTWARE INVESTMENT IS PRESERVED
- WIDE RANGE OF APPLICATION CODES AVAILABLE ON X-MP SERIES COMPUTERS
- COMMON SOFTWARE THROUGHOUT X-MP SERIES
- MULTITASKING FEATURES AVAILABLE THROUGH FORTRAN LIBRARY ROUTINES
- IMPROVED VECTORIZATION OF CONDITIONAL STATEMENTS ON X-MP/48
- DISK STRIPING TECHNIQUES FOR IMPROVED I/O PERFORMANCE
- SSD RESOURCE MANAGEMENT
- WIDE VARIETY OF STATION SOFTWARE, INCLUDING NEW APOLLO STATION, FOR EASY INTEGRATION WITH USER ENVIRONMENT



NEW FEATURES IN COS 1.13 RELEASE

* Multitasking within job steps

* Disk striping

* On-line tape positioning

Multiprogramming

COS provides for the sharing of processor resources among up to 255 independent jobs. With a single-CPU Cray, several processes are ready to run, and if one process is delayed by I/O, another job is immediately scheduled to run on the CPU. Jobs are assigned priorities, and each is allocated a CPU time-slice commeasurate with its priority. In a multi-CPU Cray, each CPU can be shared by several jobs.

High speed communications channels provide for remote users in large volume environments.

Multitasking

Multitasking allows a single user job to create multiple tasks which can execute simultaneously in more than one CPU on a multi-CPU CRAY. Multitasking within job steps provides a higher degree of parallelism within the program, and execution-time performance improvement for those applications that are appropriate for multitasking.

A COS job that is multitasked can run on the same system with jobs that are not multitasked.

On-Line Tape Positioning

Users can position a tape dataset at any block on any volume, obtain the current position information for a tape dataset, and enable recovery of tape jobs after a system interruption.

Disk Striping

(2-7 disk drives)

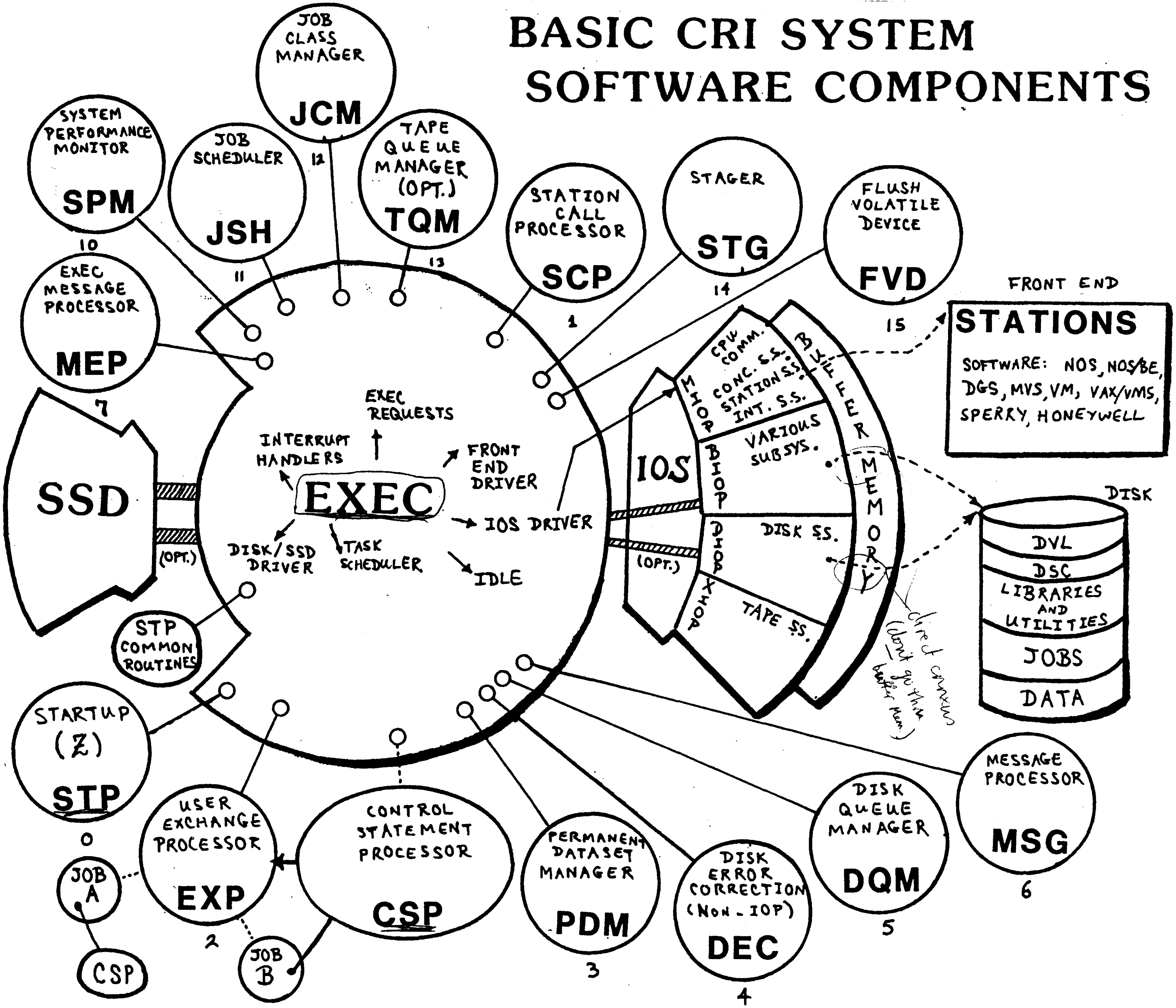
Disk striping allows users to distribute datasets across several disks in the I/O Subsystem, allowing parallel data movement from each disk. This feature provides vastly improved disk performance when larger buffer sizes are used.

Disks in the system that will be part of a stripe group will typically be used as request-by-name devices.

Re-Configure At Start-Up

Configuration changes can be made interactively during start-up. Devices can be added or deleted, or attributes or status can be changed without the necessity of a full-scale system re-generation.

BASIC CRI SYSTEM SOFTWARE COMPONENTS



1.5

STP TASK NAMES, IDs AND PRIORITIES

Task Name	ID (Octal)	PRI (Octal)
STP - Startup task	00	[illegible]
SCP - Station Call Processor	01	10
EXP - Exchange Processor	02	12
PDM - Permanent Dataset Mgr.	03	14
DEC - Disk Error Correction	04	20
DQM - Disk Queue Manager	05	02
MSG - Message Processor	06	04
MEP - Exec Message Processor	07	05
SPM - System Perform. Monitor	10	24
JSH - Job Scheduler	11	13
JCM - Job Class Manager	12	11
TQM - Tape Queue Manager	13	03
STG - Stager	14	06
FVD - Flush Volatile Device	15	15

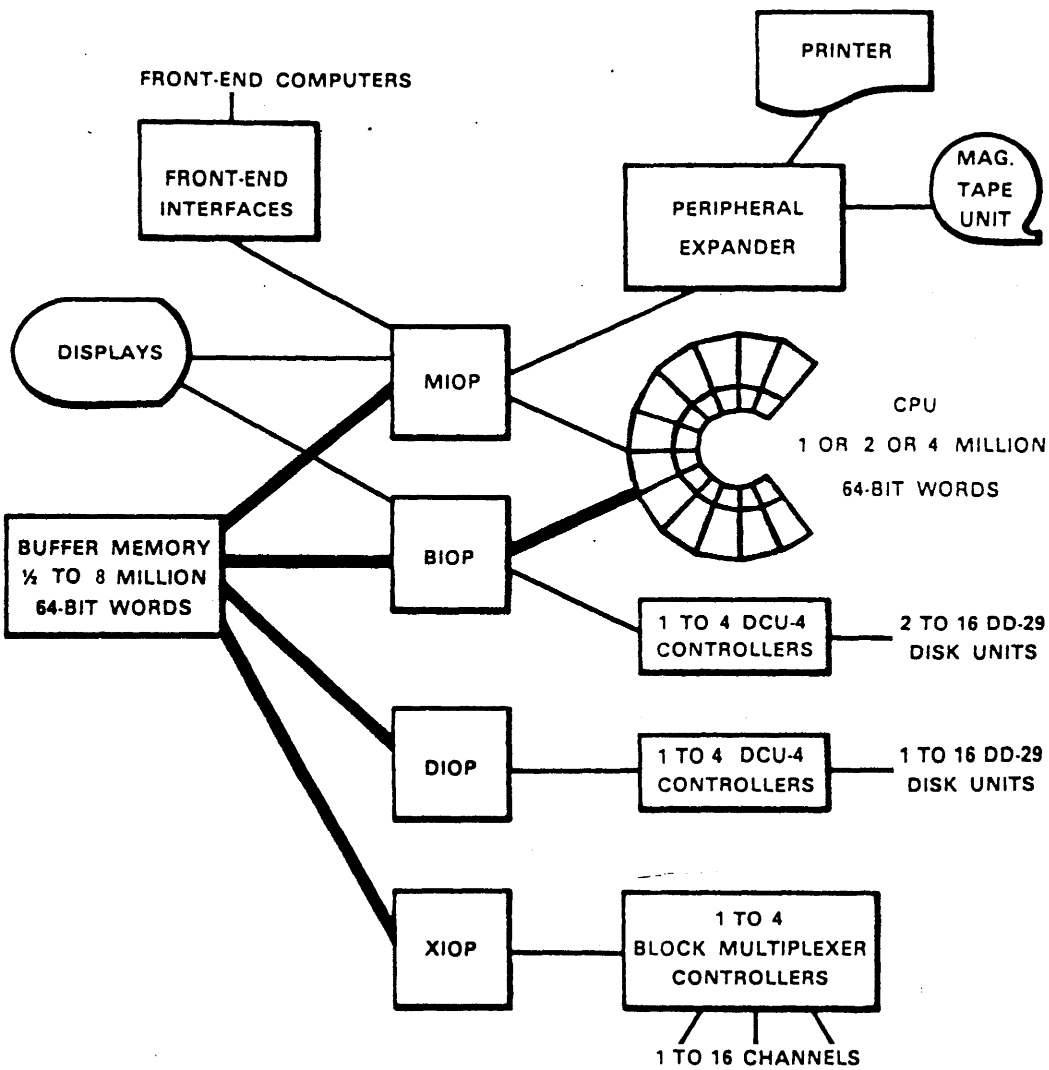

FRONT-END COMPUTERS
FRONT-END INTERFACES
PRINTER
PERIPHERAL EXPANDER
MAG. TAPE UNIT
DISPLAYS
MIOP
CPU
1 OR 2 OR 4 MILLION
64-BIT WORDS
BUFFER MEMORY
½ TO 8 MILLION
64-BIT WORDS
BIOP
1 TO 4 DCU-4 CONTROLLERS
2 TO 16 DD-29 DISK UNITS
DIOP
1 TO 4 DCU-4 CONTROLLERS
1 TO 16 DD-29 DISK UNITS
XIOP
1 TO 4
BLOCK MULTIPLEXER
CONTROLLERS
1 TO 16 CHANNELS

Table 2-1. DD-19, DD-29, and DD-49 DSU capacities

Capacity	DD-19	DD-29	DD-49
Words per sector	512	512	512
Sectors per track	18	18	42
Tracks per cylinder	10	10	8
Cylinders per device	411	822	886
Total data sectors	73,980	147,960	297,696
Total data words	38,877,760	75,755,520	152,420,352

2.1.2 CE CYLINDERS

Each DD-49 DSU contains two hardware-protected cylinders, known as *CE cylinders*. Data cannot be written to either CE cylinder until the CE cylinder is write enabled by a diagnostic command. The two hardware-protected cylinders are called CE1 and CE2. DD-49 DSU cylinders are numbered as follows.

- The data cylinders are numbered 0-885.
- CE1, cylinder 887, contains the Factory Flaw Table described later in this section.
- CE2, cylinder 889, is the diagnostic scratch cylinder. (Cylinders 886 and 888 are inaccessible.)

By default, the data cylinders are write enabled while CE1 and CE2 are write protected. A diagnostic command is required to write protect the data cylinders and write enable the CE cylinders. CE cylinders can be individually write protected or enabled.

COS also reserves cylinders for CE use by entering them in the operating system flaw table.

CRI SOFTWARE "UNIVERSE"

COS:
- EXEC (# varies)
- STP - 13 TASKS
- CSP

IOS:
- KERNEL
- 5 SUBSYSTEMS

STATIONS:
- 7 MFGRS
- 9 OPER. SYST.

UTILITIES:
- BUILD
- UPDATE
- TEDI

14 OPERATIONAL AIDS

DEBUGGERS:
- SID
- CSIM

LOADERS:
- LDR
- SEGLDR

LANGUAGES:
- CAL
- AMPL
- CFT (SKOL → no more!)
- PASCAL ("C")

LIBRARIES:
- $ARLIB
- $FTLIB
- $IOLIB
- $SCILIB
- $SYSLIB
- $UTLIB
- $PSCLIB

SYSTEM TEXTS

- DATASETS NAMED BY S= PARAMETER ON CAL CONTROL STATEMENT
- CONTAINS DEFINITION OF GLOBAL
  - MACROS
  - OPDEFS
  - MICROS
  - SYMBOLS
- $SYSTXT IS DEFAULT
- COSTXT IS USED IN ASSEMBLING COS.
- COMMON DECKS
  - DEFINED BY UPDATE DIRECTIVE COMDECK
  - CONTENTS CAN BE COPIED TO ANY NUMBER OF LOCATIONS IN THE COMPILE DATASET.
  - CAN BE CALLED FROM ANYWHERE IN A REGULAR OR COMMON DECK

CONTENTS (Macros and Opdefs Manual)

S ⇒ in $SYSTXT

C ⇒ in COSTXT

PREFACE iii

1. INTRODUCTION 1-1

2. SYSTEM ACTION REQUEST MACROS 2-1

S JOB CONTROL MACROS 2-1
ABORT - Abort program 2-2
CONTRPV - Continue from reprieve condition 2-2
CSECHO - Send statement image to the logfile 2-2
DELAY - Delay job processing 2-3
DUMPJOB - Dump job image 2-3
ENDP - End program 2-3
ENDRPV - End reprieve processing 2-4
IOAREA - Control user access to I/O area 2-4
JTIME - Request accumulated CPU time for job 2-5
MEMORY - Request memory 2-5
MESSAGE - Enter message in logfile 2-7
MODE - Set operating mode 2-8
NORERUN - Control detection of nonrerunnable functions . 2-9
RECALL - Recall job upon I/O request completion 2-9
RERUN - Unconditionally set job rerunnability 2-10
ROLL - Roll a job 2-10
SETRPV - Set job step reprieve 2-11
SWITCH - Set or clear sense switch 2-13
DATASET MANAGEMENT MACROS 2-14
CLOSE - Close dataset 2-14
DISPOSE - Dispose dataset 2-14
DSP - Create dataset parameter table 2-15
OPEN - Open dataset 2-16
RELEASE - Release dataset to system 2-18
SUBMIT - Submit job dataset 2-19
TIME AND DATE REQUEST MACROS 2-19
DATE - Get current date 2-19
DTTS - Date and time to timestamp conversion 2-20
JDATE - Return Julian date 2-20
MTTS - Machine time to timestamp conversion 2-21
TIME - Get current time 2-21
TSDT - Timestamp to date and time conversion 2-21
TSMT - Timestamp to machine time conversion 2-22

DEBUGGING AID MACROS 2-22
DUMP - Dump selected areas of memory 2-23
FREAD - Read data 2-25
FWRITE - Write data 2-26
INPUT - Read data 2-27
LOADREGS - Restore all registers 2-31
OUTPUT - Write data 2-31
SAVEREGS - Save all registers 2-35
SNAP - Take snapshot of selected registers 2-36
UFREAD - Unformatted read 2-38
UFWRITE- Unformatted write 2-39
MISCELLANEOUS MACROS 2-40
GETMODE - Get mode setting 2-40
GETSWS - Get switch setting 2-40
INSFUN - Call installation-defined subfunction 2-41
SYSID - Request system identification 2-41

3. LOGICAL I/O MACROS 3-1

SYNCHRONOUS READ/WRITE MACROS 3-1
READ/READP - Read words 3-1
READC/READCP - Read characters 3-3
WRITE/WRITEP - Write words 3-5
WRITEC/WRITECP - Write characters 3-6
WRITED - Write end of data 3-7
WRITEF - Write end of file 3-7
ASYNCHRONOUS READ/WRITE MACROS 3-8
BUFCHECK - Check buffered I/O completion 3-9
BUFEOD - Write end of data on dataset 3-9
BUFEOF - Write end of file on dataset 3-10
BUFIN/BUFINP - Transfer data from dataset to user record area 3-10
BUFOUT/BUFOUTP - Transfer data from user record area to dataset 3-12
UNBLOCKED READ/WRITE MACROS 3-13
READU - Transfer data from dataset to user's area 3-13
WRITEU - Transfer data from user's area to dataset . . . 3-14
POSITIONING MACROS 3-15
ASETPOS - Asynchronously position dataset 3-15
BKSP - Backspace record 3-16
BKSPF - Backspace file 3-17
GETPOS - Get current dataset position 3-18
POSITION - Position tape 3-19
REWIND - Rewind dataset 3-21
SETPOS - Synchronously position dataset 3-21
SYNCH - Synchronize 3-23
TAPEPOS - Tape position information 3-23

4. PERMANENT DATASET MACROS 4-1

PERMANENT DATASET DEFINITION MACROS 4-1
LDT - Create label definition table 4-1
PDD - Create permanent dataset definition table 4-3
ACCESS - Access permanent dataset 4-9
ADJUST - Adjust permanent dataset 4-9
DELETE - Delete permanent dataset 4-10
PERMIT - Explicitly permit dataset 4-10
SAVE - Save permanent dataset 4-11

5. CFT LINKAGE MACROS 5-1

DESIGN OF THE ENTRY BLOCK MACROS 5-1
DEFARG - Define calling parameters 5-2
DEFB - Assign names to B registers 5-2
DEFT - Assign names to T registers 5-3
ALLOC - Allocate space for local temporary variables . . 5-5
MXCALLEN - Declare maximum calling list length 5-5
PROGRAM - Generate mainline CAL routine start point . . . 5-6
ENTER - Generate CFT-callable entry point 5-7
RETRIEVE PASSED-IN ARGUMENT LIST INFORMATION MACROS 5-12
ARGADD - Fetch argument address 5-13
NUMARG - Get the number of arguments passed in 5-14
REFERENCE LOCAL TEMPORARY VARIABLE STORAGE MACROS 5-16
LOAD - Get value from memory into a register 5-16
STORE - Store the value from a register into memory . . . 5-19
VARADD - Return the address of a memory location 5-21
CALL EXTERNAL ROUTINES MACROS 5-23
CALL - Call a routine using call-by-address sequence . . 5-23
CALLV - Call a routine using call-by-value sequence . . . 5-25
EXIT SUBROUTINE MACRO 5-26
EXIT - Terminate subroutine and return to caller 5-26

6. TABLE AND SEMAPHORE MANIPULATION 6-1

TABLE DEFINITION AND CONSTRUCTION MACROS 6-1
Normal Macros 6-1
BUILD - Construct a table structure 6-2
ENDTABLE - Designate the end of a table definition . 6-6
FIELD - Define a field with current table structure 6-6
NEXTWORD - Advance a specified number of words . . . 6-7
REDEFINE - Redefine a specified number of words . . 6-8
SUBFIELD - Identify fields within a larger field . . 6-9
TABLE - Define the overall table attributes 6-10
Complex macros 6-11
CENDTAB - End a complex table structure 6-12
CFIELD - Define a field in the current complex table 6-12
CNXTWORD - Advance a specific number of 64-bit words 6-14

Complex macros (continued)
CREDEF - Redefine specific number of 64-bit words . 6-15
CSBFIELD - Define field entirely within another field 6-16
CTABLE - Define overall table attributes 6-17
PARTIAL-WORD MANIPULATION OPDEFS 6-18
Normal Opdefs 6-19
GET - Fetch contents of a field 6-19
GETF - Fetch contents of a field 6-20
PUT - Store data from a register into a field . . . 6-21
SET - Pack field value into a register 6-22
SGET - Fetch contents of a field 6-23
SPUT - Store data from a register into a field . . . 6-24
Complex Opdefs 6-25
CGET - Fetch contents of a field into a register . . 6-25
CPUT - Store contents of a register into a field . . 6-26
SEMAPHORE MANIPULATION MACROS 6-27
DEFSM - Define semaphore name 6-27
CLRSM - Unconditionally clear a semaphore, do not wait . 6-28
GETSM - Get current status of semaphore bit 6-29
SETSM - Unconditionally set a semaphore, do not wait . . 6-30
TEST$SET - Test semaphore and wait if set, set if clear . 6-30
CAL EXTENTION MACROS AND OPDEFS 6-31
DIVIDE OPDEF - Provide a precoded divide routine 6-31
PVEC MACRO - Pass elements of vector register to scalar routine 6-32
$CYCLES MACRO- Generate timing-related symbols and constants 6-33
$DECMIC MACRO - Convert a positive integer to a micro string 6-34
RECIPCON MACRO - Generate floating-point reciprocals . . 6-35

7. COS DEPENDENT MACROS 7-1

SYSTEM TASK OPDEFS 7-1
ERDEF - Generate error processing entries in the Exchange Processor 7-1
GETDA - Obtain first DAT page address 7-2
GETNDA - Obtain next DAT page address 7-4
OVERLAY MANAGER TASK MACROS 7-6
CALLOVL - Request Overlay Manager Task to load 7-7
DEFINOVL - Generate a list of modules 7-7
DISABLE - Prevent use of current memory-resident copy . . 7-8
GOTOOVL - Request Overlay Manager Task to load 7-8
LOADOVL - Request an initial overlay load 7-9
OVERLAY - Define a module as a system overlay 7-9
OVLDEF - Define overlay name 7-10
RTNOVL - Signal completion of an overlay execution . . . 7-11
MESSAGE PROCESSOR MACRO 7-11
LOGMSGM - Construct the LGR control word 7-11

1.14

COS INTERNAL SUBROUTINE LINKAGE MACRO 7-13
$SUB - Define a subroutine entry point 7-13
DATASET LOCKING MACRO 7-14
DSPLOCK - Set or clear lock in dataset parameter area . . 7-14

8. STRUCTURED PROGRAMMING MACROS 8-1

CONDITIONS 8-1
MACRO DESCRIPTIONS 8-4
$GOTO MACRO - Compute a GOTO statement 8-4
$GOSUB MACRO - Call local subroutine 8-4
$IF, $ELSEIF, $ELSE, and $ENDIF MACROS - Conditional macros 8-5
$JUMP MACRO - Accept any $IF or $ELSEIF parameters . . . 8-7
$LOOP, $EXITLP, and $ENDLOOP MACROS - Define program loops 8-8
$RETURN MACRO - Return from local subroutine 8-9
$SUBR MACRO - Declare local subroutine entry point . . . 8-10

APPENDIX SECTION

OUTMODED FEATURES A-1
BREG - Assign names to B registers (obsolete) A-1
TREG - Assign names to T registers (obsolete) A-2

TABLES

8-1 Conditions 8-1

INDEX

Contents of $SYSTEXT not listed in SR-0012

PDUMP
SETSTKAT
FORCE
NEWENTRY
INSMAC
FTIO
BASERR
CPUTYPE
SVREGS
LDREGS
OPENA
@VALREG
@GENREW
@GENTP
$RCW
$TSTRG
$BIO
PRM1SYM
PRM2SYM
PRM2MSC
RECUR
SNAPSHOT

%REGADDR
%QDOUBLE
%FORTIO
%ARGADDR
FEDLP
(MACROS FOR CSIM:)
SIMMSG
SIMIDLE
SIMABORT
SIMRTC
SIMAVAIL
ENDIR
DISDIR
TAGON
TAGOFF
(SYERP ERROR CODES)

DEFLOCK
LOCKON
LOCKOFF

Contents of COSTXT not listed in SR-0012:

DRT
EQT
$GTTOKEN
$DOICH
CONFIG
%GETNUM
FLAW
STRTFLW
ENDFLW
SETIF
POST
VTRACE
TSKREQ
RJ
LOCK
UNLOCK
ERRU
ERRAZ
ERRAN
ERRAP
ERRAM
ERRSZ
ERRSN
ERRSP
ERRSM

CAPTION
ENDFIELD
%GETSYM
FIELD@
$LINE
GN
GD
(CAL BOOLEAN EXPRESSIONS of form $$BOOLxxx)
$SREG
$LREG
LOADCL
SAVECL
MSGQ
SMDEF
WAIT$SET
(CSIM Macros: same as in $SYSTXT)

COMMON DECKS:

CONFIG@P
- Processor dependant parameters

COMSYMB
- Common system sympols

COSI@P
- Installation dependant parameters

COMHD
- Hardware description parameters

USERI@P
- Install. parameter definitions for users

COMSYSDP
- System defined parameters

COMSYSEQ
- System equates

LOWSTP
- Pointers in low memory -

*** IDENTIFIERS IN PROGRAM LIBRARY COSPL ***   PLDATE 03/14/84 LASTID PC-ERNST

**AUDBUF	**AUDCFT	**AUDCOM	**AUDPAGE	**AUDPARAM	**AUDSUM	**AUDTABX	**AUDTM	**COMAC	**COMAE	**COMAER
**COMAM	**COMAP	**COMAR	**COMAU	**COMAUX	**COMBA	**COMBG	**COMBIO	**COMBP	**COMBRT	**COMCB
**COMCC	**COMCH	**COMCN	**COMCS	**COMCW	**COMCX	**COMDA	**COMDC	**COMDD	**COMDDFC	**COMDFT
**COMDMP	**COMDN	**COMDP	**COMDPT	**COMDR	**COMDT	**COMDV	**COMDXI	**COMDXT	**COMEFT	**COMENT
**COMEP	**COMEQ	**COMET	**COMEVW	**COMEXERR	**COMEXFC	**COMEXPFC	**COMEXRQ	**COMFER	**COMFH	**COMGR
**COMHD	**COMHS	**COMHTFC	**COMIB	**COMIER	**COMIOF	**COMJB	**COMJC	**COMJCM	**COMJS	**COMJSH
**COMJT	**COMJX	**COMLC	**COMLCK	**COMLD	**COMLF	**COMLG	**COMLT	**COMLX	**COMMATH	**COMMCT
**COMMD	**COMMDW	**COMMEM	**COMMENT	**COMMP	**COMMS	**COMMTQ	**COMOD	**COMOP	**COMPA	**COMPC
**COMPD	**COMPDMFC	**COMPDT	**COMPER	**COMPERT	**COMPI	**COMPM	**COMPPL	**COMPQ	**COMPR	**COMPS
**COMPT	**COMPVC	**COMPW	**COMPX	**COMQC	**COMQD	**COMRJ	**COMRPV	**COMRQ	**COMSB	**COMSC
**COMSCP	**COMSD	**COMSDP	**COMSDR	**COMSEG	**COMSEQ	**COMSM	**COMSMC	**COMSML	**COMSPM	**COMSR
**COMSS	**COMSSL	**COMST	**COMSTK	**COMHEAP	**COMSTP	**COMSW	**COMSYMB	**COMSYSDP	**COMSYSEQ	**COMTA
**COMTAPE	**COMTB	**COMTC	**COMTCA	**COMTD	**COMTE	**COMTEXT	**COMTK	**COMTIB	**COMTLB	**COMTN
**COMTP	**COMTQM	**COMTR	**COMTV	**COMTX	**COMTXT	**COMUER	**COMUP	**COMUTSB	**COMXAT	**COMXF
**COMXP	**COMXRT	**COMXT	**COMZ	**CONFIG@P	**COPYRIT	**COSI@P	**CSMAC	**CYCLES	**ENDCOM	**EXFBITTB
**EXFC+DEF	**EXFCMPTM	**EXFDBUG	**EXFFILE	**EXFHCR	**EXFHCRD	**EXFMFTYP	**EXFPARM	**EXFREC	**EXFSPEC	**EXFSTF
**EXFSUBT1	**EXFSXRAT	**EXFTEXT	**GETSRO	**GETVRB	**GLOBALC	**IFTHEN	**LOGMAC	**LOWSTP	**MSGCLSS	**POSTMIC
**STARTCOM	**STATSIO	**SYNONYM	**TABMAC	**TABX	**USERI@P	*ST	*CT	*UT	*E	*GLOBAL
*STPTAB	*BGNSTPCM	*BFMAN	*BTAD	*CCOPY	*CHAINS	*CLEAR	*CONFIG	*CONVTS	*COPY	*CRACKER
*FIXJXPR	*GETOWN	*GETPARM	*GTMEM	*FNDJB	*LGMSG	*GTTCB	*GTTXT	*JMEM	*JTADNT	*MSGQUE
*PWENC	*QMSG	*QUEUES	*REQRPLY	*RLMEM	*RLTXT	*SD2PD	*STPDATS	*STPERR	*STPMEM	*STPTIME
*STPUTIL	*ENDSTPCM	*STP	*SCP	*EXP	*CIO	*TQM	*MSG	*JSH	*JCM	*DQM
*DEC	*PDM	*MEP	*SPM	*STG	*FVD	*STARTUP	*J	*ACCOUNT	*ACCTDEF	*ADS
*AUDIT	*CD	*CF	*CHARGES	*CM	*CR	*CU	*DBF	*DD	*DDC	*DM
*EXC	*EXF	*FLOWD	*JCC	*JCF	*PD	*PL	*PRVDEF	*SD	*SETOWN	*SF
*SR	*STATS	*UNB	*WD	KC4351A	KC2281A	KC3860A	KC3947A	KC3947B	KC3948A	KC4308A
KC4312A	KC4320A	KC4352A	KC4365A	KC4366A	KC4379A	KC4379B	KC4457A	KC4463A	KC4477A	KC4478A
KC4369A	KC4520A	KC4536A	KC4541A	DD10	E015	EXF12	KC3836A	KC3836B	KC3836C	KC3749A
KC3749B	KC3749C	KC3300A	KC3300B	KC3536A	KC3767A	KC3770A	KC3842A	KC3856A	KC3864A	KC3864B
KC3864C	KC3938A	KC4068A	KC4074A	KC4082A	KC4091A	KC4136A	KC4136B	KC4176A	KC4197A	KC4264A
KC4264B	KC4264C	KC4264D	KC4264E	KC4284A	KC4284B	KC4288A	KC4319A	KC4354A	KC4355A	KC4363A
KC4363B	KC4363C	KC4363D	KC4363E	KC4374A	KC4383A	KC4383B	KC4383C	KC4383D	KC4383E	KC4383F
KC4383G	KC4383I	KC4383J	KC4383K	KC4383L	KC4383M	KC4383N	KC4383O	KC4383P	KC4383Q	KC4383R
KC4383AA	KC4383AB	KC4383AC	KC4383AD	KC4383AE	KC4383AF	KC4383AG	KC4383AH	KC4383AI	KC4383AJ	KC4383AK
KC4383AL	KC4383AM	KC4383AN	KC4383AO	KC4383AP	KC4383AQ	KC4383AR	KC4383AS	KC4383AT	KC4383AU	KC4383AV
KC4383AW	KC4383AX	KC4383AY	KC4383AZ	KC4383BA	KC4383BB	KC4383BC	KC4383BD	KC4383BE	KC4383BF	KC4383BG
KC4383BH	KC4383BI	KC4383BJ	KC4383BK	KC4383CA	KC4383CB	KC4383CC	KC4383CD	KC4383CE	KC4383DA	KC4383DB
KC4383DC	KC4383DD	KC4383DE	KC4383DF	KC4383DG	KC4383DH	KC4383DI	KC4383DJ	KC4383DK	KC4383DL	KC4383DM
KC4383DN	KC4383DO	KC4383DP	KC4383DQ	KC4383DR	KC4383DS	KC4383DT	KC4383DU	KC4383DV	KC4383DW	KC4383DX
KC4383DY	KC4383DZ	KC4383EA	KC4383EB	KC4383EC	KC4383ED	KC4383EE	KC4383FA	KC4401A	KC4532A	KC4532B
KC4532C	KC4532D	KC4532E	KC4532F	KC4532G	KC4532H	KC4532I	KC4382A	KC3123A	KC3499A	KC3528A
KC3528B	KC3528C	KC3540A	KC3695A	KC3696A	KC3797A	KC3797B	KC3829A	KC4001A	KC4243A	KC4357A
KC4391A	KC4393A	KC4406A	KC4460A	KC4472A	KC4472B	KC4530A	KC4533A	KC4556A	KC4557A	KC4384A
KC4560A	KC2787A	KC2787B	KC3423A	KC3423B	KC3423C	KC3423D	KC3499AA	KC3500A	KC3500B	KC3500C
KC3500D	KC3500E	KC3500F	KC3500G	KC3500H	KC3500I	KC3500J	KC3500AA	KC3500AB	KC3507A	KC3780A
KC3780B	KC3780C	KC3780D	KC3780E	KC3780F	KC3843A	KC4205A	KC4281A	KC4315A	KC4532AA	KC4532AB
KC4532AC	KC4532AD	KC4532AE	KC4532AF	KC4532AG	KC4532AH	KC4532AI	KC4532AJ	KC4532AK	KC4532AL	KC4532BA
KC4585A	KC4585B	KC4585C	KC4585D	KC4585E	KC4585F	KC4585G	KC4585H	A008	KC1761A	KC2153A
KC3133A	KC3133B	KC3133C	KC3410A	KC3500BA	KC3500BB	KC3780AA	KC3852A	KC3852B	KC3854A	KC4245A
KC4279A	KC4279B	KC4280A	KC4375A	KC4529A	KC4532CA	KC4532CB	KC4532DA	KC4532DB	KC4532DC	KC4532DD
KC4532DE	KC4532DF	KC4532DG	KC4532DH	KC4532DI	KC4532DJ	KC4532DK	KC4532DL	KC4532EA	KC4560AA	KC4569A
KC4600A	KC4600B	KC4625A	KC4626A	KC4626B	KC4636A	KC4636B	KC4636C	KC4636D	KC4636E	KC4636F
KC4636G	KC4636H	KC4636I	KC4636J	KC4636K	KC4636L	KC4636M	KC4636N	KC4636AA	KC4637A	KC4642A
KC4683A	KC4686A	KC4532FA	KC4532FB	KC4532FC	KC4532FD	KC4532FE	KC4532GA	KC4532GB	KC4532HA	KC4532IA
KC4532IB	KC3780BA	CM11	KC3908A	KC4532KA	KC4532LA	KC4532LB	KC4532LC	KC4532MA	KC4532NA	KC4532NB
KC4532NC	KC4532OA	KC4532OB	KC4532OC	KC4532PA	KC4532PB	KC4532QA	KC4532QB	KC4532SA	KC4532SB	KC4532TA
KC4532TB	KC4532TC	KC4532TD	KC4534A	KC4545A	KC4624A	KC4636BA	KC4636BB	KC4636BC	KC4682A	KC4736A

FUNCTION OF $UTILTXT:

IT PROVIDES DEFINITIONS FROM COSPL TO UTILITIES IN UTILPL. THIS ELIMINATES THE NEED FOR HAVING THE DEFINITIONS from COSTXT + COMMON DECKS IN TWO PLACES.

HISTORICALLY, UTILITIES IN UTILPL WERE IN COSPL SO THERE WASN'T A PROBLEM.

LH A

I-TR

**DEBUGGER	**DEBUGIOU	**DUMPREGS	*GLOBEF	*MEMTST	*MOSCLK	*SYMACS	[illegible]	[illegible]	[illegible]	*ABOVE
*AMAP	*ACOM	*AMSG	*BADDAT	*BEGIN	*BLOCK	*BTD	*BTO	*BXSET	*CALL	*CDEM
*CHNOFF	*CHNTST	*CLOCK	*CONFIG	*CPSPIN	*CPTEST	*CRAY	*CRTDEM	*DEVICE	*DISK	*DISKIO
*DIVIDE	*DKDMP	*DKIOEX	*DKLOOK	*DKSET	*DKSETO	*DKSTAT	*DM3	*DOM	*DOMP	*DTB
*ERR	*ERRDMP	*ERRECK	*FDMPDR	*FIRECODE	*F80M	*F80ME	*HDRPAG	*HPDATA	*HPLOAD	*HSPTES
*LISTO	*LISTP	*MASTER	*MOSTES	*MSGHND	*MSTNEW	*MULTIPLY	*NOBEAT	*OBIT	*OTB	*OUTCALL
*PATCH	*PLOTIT	*PRTAPE	*REPORT	*SCRUB	*START	*STARTO	*START1	*START2	*START3	*START4
*START5	*STATS	*STOP	*SUMDAY	*SUMHOW	*SUMTIME	*SYSS	*SYSTEXT	*TCOM	*TDUMP	*TIME
*TRACE	*TRACK	*TSTASH	*UBTAPE	*UNBLK	*USURP	*WATCH	*XDISK	*XDISKA	*XDK	*XMT
*XPR	*XPRINT	*XPRNTA	*XTAPE	*XTAPEA	*XTAPEB	*XTAPEC	*XTAPED	*ZKOVL	*ABMX	*BMXAIO
*BMXCON	*BMXCPU	*BMXDEM	*BMXOPE	*BMXSIO	*BMXTPO	*ZBMX	*ATAPE	*BCOM	*BUFMAN	*BYPASS
*CONMAN	*DSCGET	*TAPEIO	*TAPMOV	*TDEM	*TDEMO	*TDEM1	*TERROR	*TEX	*TRBOC	*TRCER
*TRCLN	*TRCMR	*TRDCK	*TRDCKO	*TRDSE	*TREQC	*TRFUN	*TRIDB	*TRINR	*TRLPT	*TRORN
*TRRDB	*TRRDF	*TRREDO	*TRSET	*TRTELL	*TRWRT	*ZTAPE	*ASDMP	*RSTRTO	*RSTRT1	*RSTRT2
*SDMPO	*SDMP1	*SDMP2	*SDMP3	*SDMP4	*SDMP5	*SDMP6	*SDMP7	*SDMP8	*SDMP9	*SDMP10
*ZSDMP	*AFILE	*CLEAR	*CLEARO	*CLEAR1	*COPY	*COPYO	*COPY1	*COPY2	*COPY3	*COPY4
*COPY5	*COPY6	*COPY7	*COPY8	*COPY9	*COPY10	*COPY11	*DELETE	*DELETO	*DELET1	*DSKIO
*EDDELE	*EDINST	*EDIT	*EDITO	*EDIT1	*EDPRNT	*EDREPL	*EDTYPE	*FILACC	*FILCLS	*FILCRE
*FILDEL	*FILGET	*FILNIT	*FILPOP	*FILPUT	*FILSTT	*FLAW	*FSTAT	*FSTATO	*FSTAT1	*INIT
*INITO	*LINGET	*LINPUT	*XFMACC	*XFMCLS	*XFMCRE	*XFMDEL	*XFMDIR	*XFMFLW	*XFMFND	*XFMGET
*XFMIO	*XFMNIT	*XFMPOP	*XFMPUT	*XFMSTR	*XFMSTT	*ZFILE	*ACONC	*CHKSMI	*CHKSMO	*CONC
*CONCERR	*CONCI	*CONCO	*CRAYMSG	*ENDCONC	*ENTRID	*FEREAD	*FEWRIT	*FREEBUFS	*LOGOFF	*LOGONA
*LOGONB	*LOGONC	*MSGIN	*MSGIO	*MSGOUT	*REMVID	*SRCHID	*ZCONC	*ASTAT	*ACQTRM	*ACQUIRE
*AMPEX	*BABEL	*BARDAT	*BMGET	*BMAGET	*BXDIS	*CLI	*CLINIT	*CONSL	*COMBO	*COMMO1
*COMMO2	*COMMO3	*COMMO4	*COMMO5	*COMMO6	*COMMO7	*COMMO8	*COMMO9	*COMM10	*COMM11	*COMM12
*COMM13	*CPUGET	*CRAYIO	*DBGET	*DECODE	*DECOD2	*DELMSG	*DESCRIBE	*DEVDAT	*DISPLAY	*DISPO1
*DISPO2	*DKDIS	*ERRDIS	*ERROR	*FMGET	*GRAPH	*HSPGET	*ICONSL	*IDEBUG	*IDLGET	*IDRCT
*IFRMT	*KEYBD	*LCP	*LOGON	*LINK	*MESSAGE	*MSTAT	*MSTDIS	*NEWDIS	*OFRMT	*ONLINE
*POST	*PROTINIT	*PROTOCOL	*QUEUE	*READ	*REPLY	*SNAP	*SOROC	*STADIS	*STAGEIN	*STAGEOUT
*STATCL	*STATINIT	*STATION	*STIO	*STMSG	*STPLOT	*STREAMS	*STSGET	*STTAPI	*STTAPO	*STUBPR
*SYNTAX	*SYSTAT	*TAPEC	*TEC455	*TEXT	*TJOB	*TKSTAT	*UPDATE	*XFRMT	*XMPXP	*ZSTAT
*AINTER	*IACMD	*IACON	*IACON1	*IAFUNC	*IAIOP	*IAIOP1	*IAMSG	*IAOUT	*ZINTER	*ANSC
*NIDEND	*NSC	*NSCEND	*NSCID	*NSCIO	*NSCMSG	*NSCONC	*NSCOR	*ZNSC	*TAPELOAD	*DISKLOAD
*DMP	*EOF2	*ACOVL	*SDMPA	*SDMPB	*SDMPC	*SDMPD	*SDMPE	*ZCOVL	REL112	

*** 365 DECKS 7 COMMON DECKS 1 CORRECTION SET IDENTIFIERS

1.19

STATIONS (Example: MVS Station)

- FUNCTION
  - JOB SUBMISSION
    - TSO user entry
    - local batch entry
    - remote batch entry
  - OPERATOR CONTROL of JOB PROCESSING
    - TSO user
    - MVS station console
    - Master operator station
- CHARACTERISTICS
  - RUNS UNDER MVS
  - USES JES2 or JES 3
  - HARDWARE CONNECTION TO CRAY IS
    - FEI built by CRAY
      or
    - HYPERchannel, built by NSC
- COMPONENTS

Table 3-1. Command availability

Command	Availability codes					
	M	T	O	R	L	N
CANCEL			X†	X	X	
CHANNEL	X			X	X	
CLASS	X			X	X	
CONFIGURE	X			X	X	
DATASET		X	X		X	
DEVICE	X			X	X	
DROP		X	X	X	X	
END			X†	X		X
ENTER			X	X	X	
JOB		X	X		X	
KILL		X	X	X	X	
LIMIT	X			X	X	
LINK			X		X	
LOGOFF			X	X	X	
LOGON			X	X		X
MESSAGE		X	X	X	X	
OPERATOR			X	X		
POSTPONE			X†	X	X	
PRINT			X	X		
RECOVER	X			X	X	
RERUN		X	X	X	X	
RESUME	X			X	X	
ROUTE	X			X	X	
SET			X	X		X
SHUTDOWN	X			X	X	
STAGE			X	X		
STATCLASS			X		X	
STATION			X			
STATUS		X	X		X	
STORAGE			X		X	
STREAM			X	X	X	
SUSPEND	X			X	X	
SWITCH		X	X	X	X	
TAPE			X		X	
TJOB		X	X		X	
TRACE			X	X		
TRANSFER		X	X	X		

† Command available only to the MVS operator.

IBM MVS STATION OPERATOR'S GUIDE

1.21

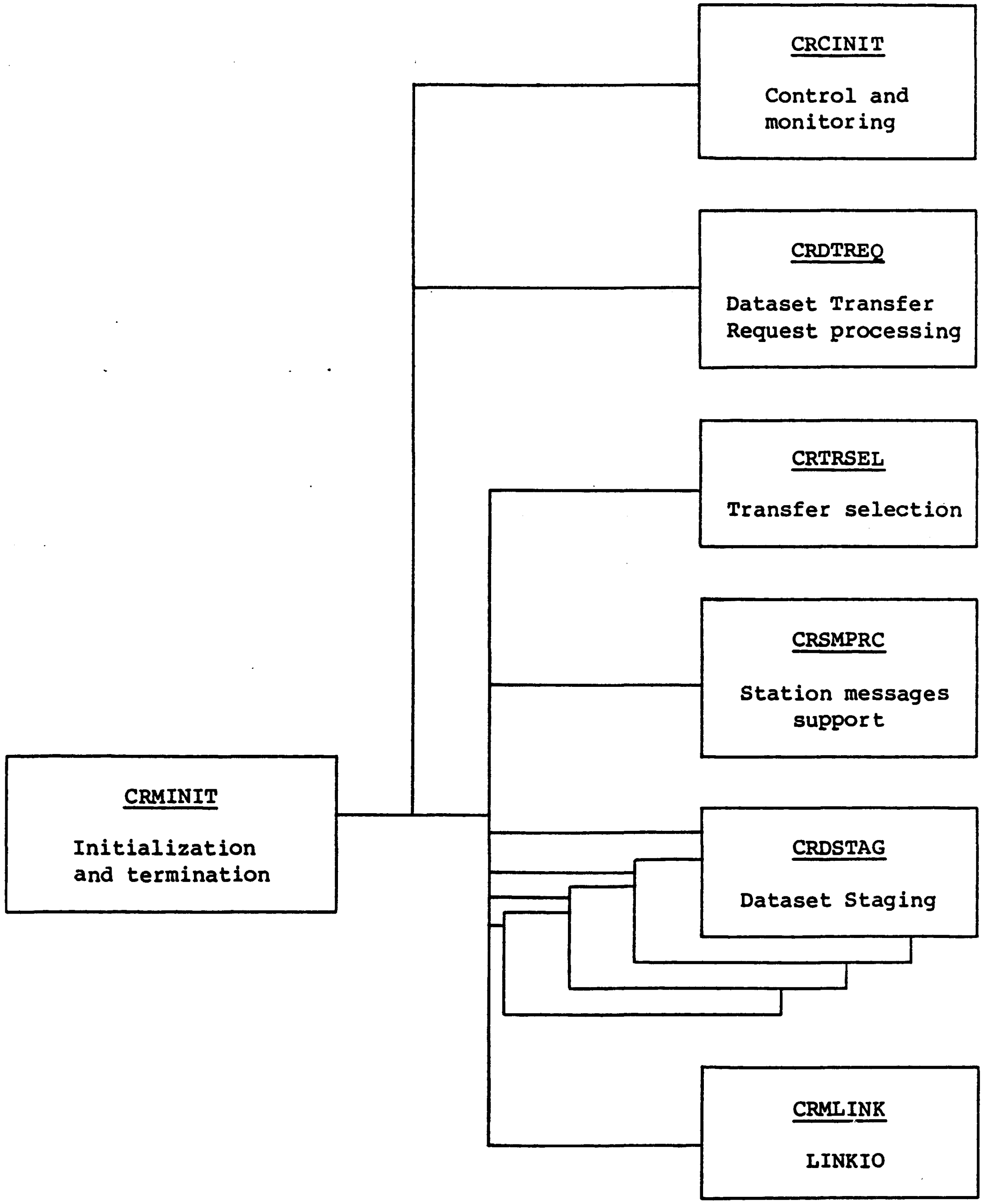


Figure 2-1. Station task structure

IBM MVS STATION INTERNAL REF. MAN.

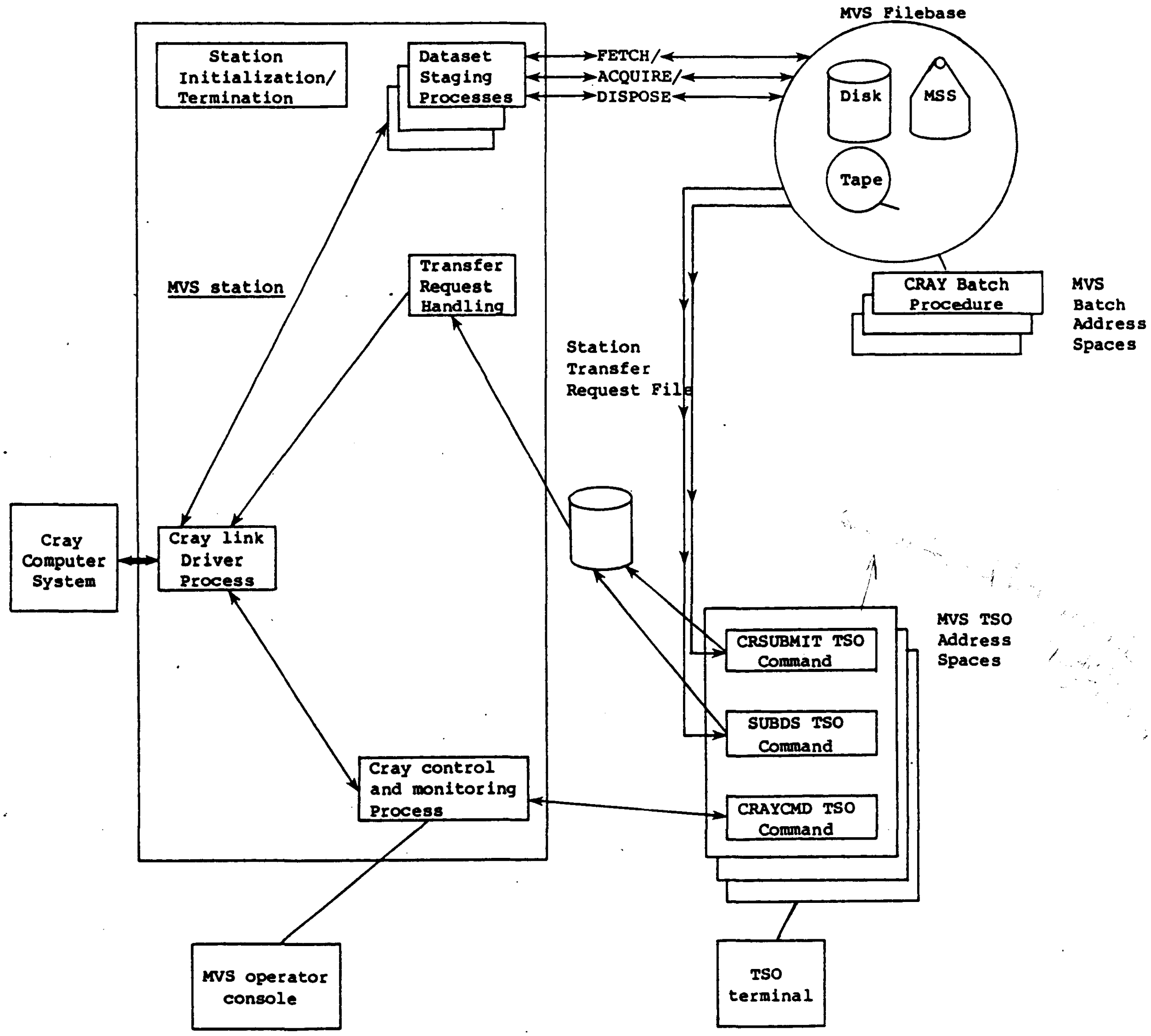


Figure 1-1. MVS user's view of the Cray Computer System through MVS station

IBM MVS STATION REF. MAN.

1.23

LESSON 2: Hardware Configurations & Characteristics

Objective: Describe the various hardware configurations and characteristics of the computer systems on which COS executes.

HARDWARE REQUIREMENTS

The Cray Operating System (COS) executes on the basic configurations of any CRAY-1 or CRAY X-MP Computer System. Each computer system contains the following components:

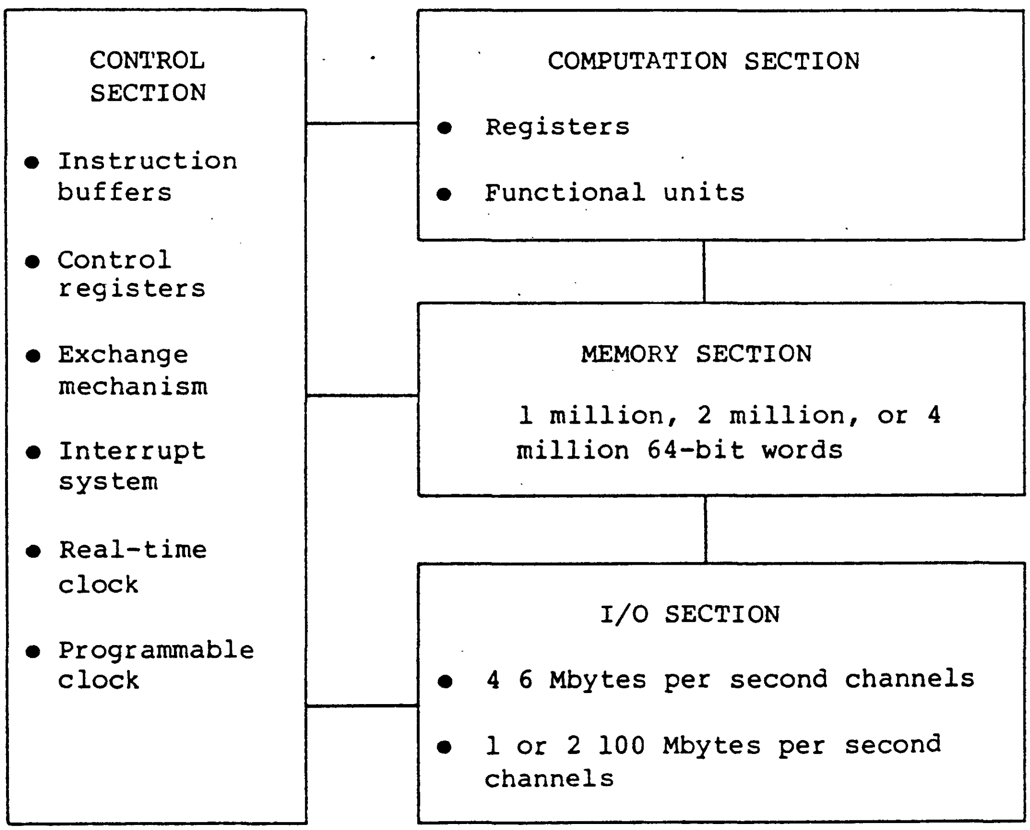
* One or two CPUs; a CRAY-1 contains one CPU and the CRAY X-MP contains two CPUs.

* Central Memory. COS operates with any of four Central Memory size options: one-half million words, one million, two million, and four million.

* A minicomputer-based Maintenance Control Unit (MCU) or I/O Subsystem (IOS). The I/O Subsystem, if present, performs all required Maintenance Control Unit functions.

* A Mass Storage Subsystem. The Mass Storage Subsystem may consist of DD-19 or DD-29 disk drives, a Solid-state Storage Device (SSD), or Buffer Memory (BMR). BMR storage can be accessed only through an I/O Subsystem; disk drives may be connected either to an I/O Subsystem or Cray mainframe. SSDs are connected directly to the CRAY-1 or X-MP mainframe.

* An optional IBM-compatable tape subsystem. The tape subsystem requires that an I/O Subsystem be present.


CONTROL SECTION
● Instruction buffers
● Control registers
● Exchange mechanism
● Interrupt system
● Real-time clock
● Programmable clock
COMPUTATION SECTION
● Registers
● Functional units
MEMORY SECTION
1 million, 2 million, or 4 million 64-bit words
I/O SECTION
● 4 6 Mbytes per second channels
● 1 or 2 100 Mbytes per second channels

System Components

CPU

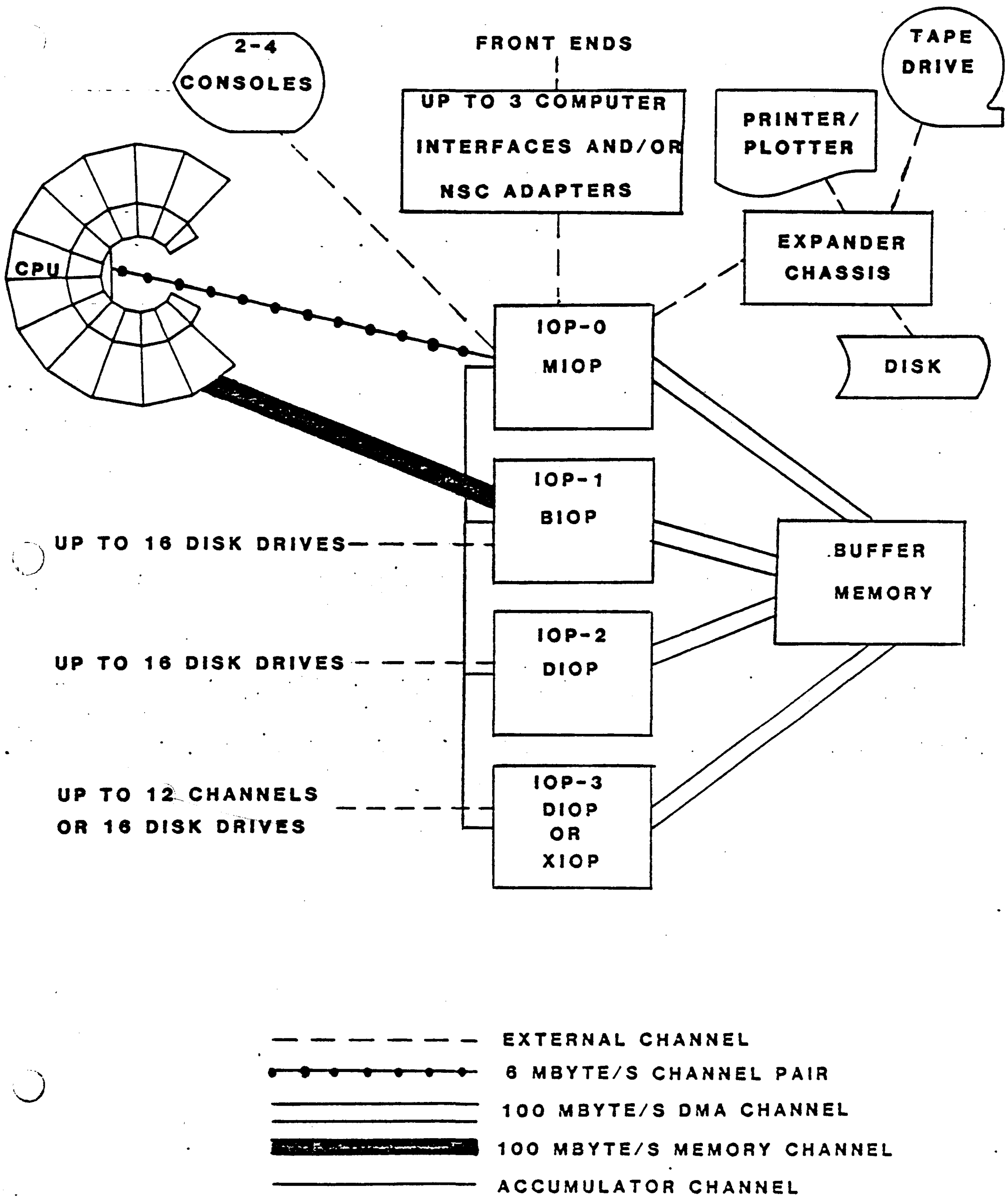
The Cray CPUs are designed for speed and large volume processing. This is accomplished with large, high-speed channels, fast memory, large instruction buffers, and a large number of registers and functional units. The segmented functional units can receive a different operand every clock period.

With parallel processing, operands can be supplied to different functional units every clock period, resulting in a processing speed of up to 105 million instructions per second.

The CRAY 1-S, 1-M and X-MP CPUs have the following characteristics:

	1-S	**1-M**	**X-MP**
CPUs	1	1	2
# System	Octal	Octal	Octal
Clock Period	12.5 ns	12.0 ns	9.5 ns
M.I.P.S.	80	80.33	105
Max. Memory	4 Mil.	4 Mil.	4 Mil.
Word size	64 bits	64 bits	64 bits
Columns	8 or 12	6	8 or 12

FOUR PROCESSOR SYSTEM



I/O Channels

The CRAY computers are equiped with several types of I/O Channels designed for communicating with different devices within the system.

Synchronous - 6 Mbytes/sec

Synchronous channel pairs are used for data transfer between the Cray 1-S or X-MP and the MIOP of the I/O Subsystem.

Transfers are synchronized blocks of 512-bit words.

Asynchronous - 7.5 Mbytes/sec

Due to the unpredictable nature of transfer between the Front Ends and the Cray, asynchronous channels are provided for this purpose.

The channels transfer at a rate of up to 7.5 Mbytes per second, and use a protocol which is sychronized on every 16 bits of message. An asychronous channel is also used for the communication between the IOS and MCU (Manintenance Control Unit).

High-Speed Memory - 100 Mbytes/sec

To handle the large volumes of data transfered between the Cray and the IOS (BIOP/DIOP), 100 MByte channels are used. Several of these channels could be employed, depending on the Cray computer in use.

Direct-Wired HYPERchannel - 1250 Mbytes/sec

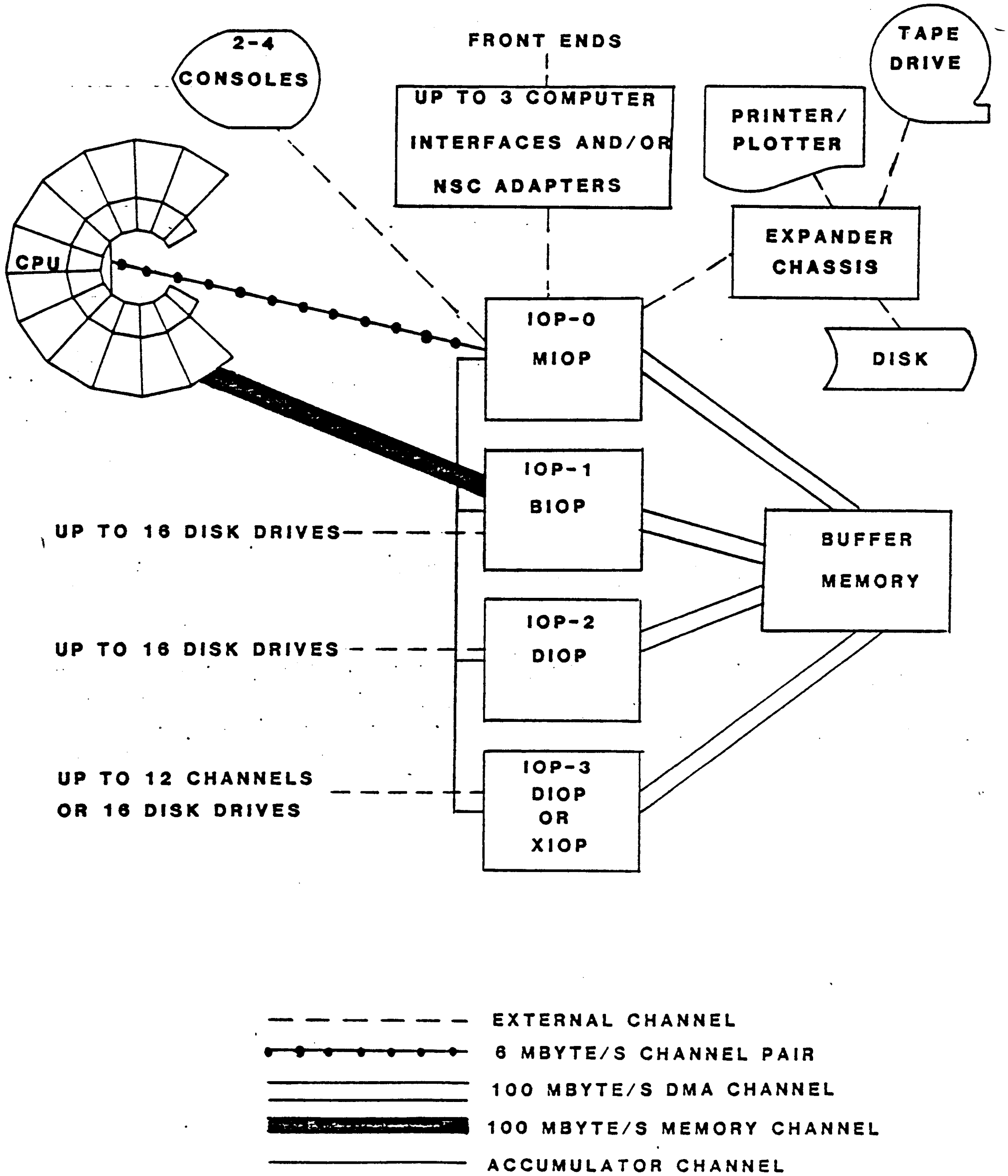
For Cray X-MPs which utilize the SSD (Solid state Storage Device) a 1250 Mbyte per second channel is used for very high speed data transfer capabilities.

The channel is sychronized on 128-bit blocks.

Cabled HYPERchannel - 10-12 Mbytes/sec

For systems equiped with NSC networking configurations, data is tranferred between devices within the network via a hyperchannel cable capable of 10-12 MBytes/sec, depending upon the length of the cable. The protocol is sychronized on every 16 bits of message.

FOUR PROCESSOR SYSTEM



I/O Subsystem

The purpose of the I/O Subsystem is to increase Cray throughput by providing large volume I/O capabilities for the system. It can also act as the system maintenance control unit (MCU) through which the Cray would be deadstarted and operated.

The I/O Subsystem consists of two, three, or four I/O Processors, Buffer Memory, and required interfaces.

Each IOP is an independent minicomputer responsible for some portion of the I/O requirements of the system. Each has its own memory (65K), computation, control, and I/O sections. They are designed for fast data transfer between front-end computers, mass storage devices, peripheral devices, Buffer Memory, and the central memory of the CRAY mainframe.

Buffer Memory sizes can be one-half million, one-million, four million, or eight million words.

MIOP

The Master I/O Processor (MIOP) is the first I/O Processor in the subsystem to be deadstarted. The MIOP initializes the contents of Buffer Memory and accumulator channels to the other processors.

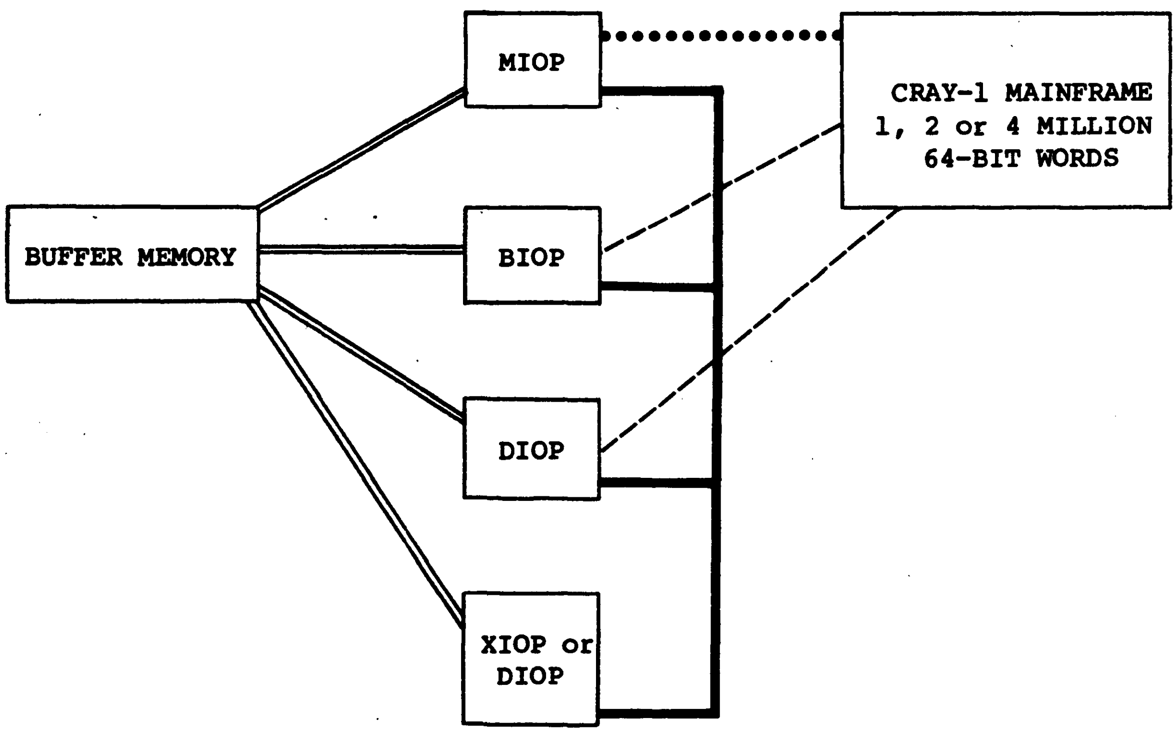
The MIOP deadstarts the Cray mainframe and directly handles all communications with the mainframe over the 6Mbyte channel. This traffic includes disk and tape requests and station communications.

BIOP

The Buffer I/O Processor (BIOP) transfers data between Cray central memory and IOS Buffer Memory and vice versa across a 100 Mbyte channel. The BIOP performs disk I/O to and from disk units attached to its channels.

DIOP

The optional DIOP moves data from Buffer Memory to disk and vice versa at the request of packets from the mainframe via the MIOP. If the optional second 100 Mbyte channel is present, the DIOP transfers data between Cray central memory and DIOP local memory and vice versa.



•••••••• 50 Mbit/s CRAY-1 I/O channel pair

------ Approximately 850 Mbit/s Memory Channel

═══ Approximately 850 Mbit/s DMA channel

▬▬▬ Accumulator channel

I/O Subsystem communication

XIOP

The optional XIOP handles data from IBM-compatable tape drives and buffers the data to Buffer Memory at the request of packets from the mainframe.

I/O Subsystem Communication

The Cray computer system provides communication paths between central memory and the MIOP and BIOP (and DIOP if a second memory channel is present); between each IOP and Buffer Memory; and among all the IOPS.

Data is transferred between Cray memory and the IOPs (BIOP/DIOP) over one or more 100 Mbyte/sec Memory Channels. The Cray I/O Channel pairs exchange system control information with the MIOP at 6 Mbytes/sec.

One 100 Mbyte/sec DMA port for each IOP is connected to Buffer Memory. Buffer Memory receives data from one IOP and temporarily stores it until the Buffer IOP or Disk IOP can remove that data and pass it to Cray central memory. In this way, each IOP communicates with every other IOP in high-speed data block transfers.

Each IOP is also connected with the other IOPs by slower channels called accumulator channels. These channels pass one 16-bit parcel at a time from the accumulator of one IOP to the accumulator of another IOP and are used primarily for control and status reporting.

Figure 1-6. DD-29 Disk Storage Unit

Mass Storage Units

The basic mass storage unit for the Cray is the DD-29 Disk Storage Unit (DSU). This unit is a 606 Mbyte disk drive with data transfer rate of 35.3 Mbits per second.

Up to four DD-29 drives can be connected to one DCU-4 Disk Controller. The disk controller interfaces the four disk drives with an I/O Processor through one direct memory access (DMA) port. With up to 12 controllers per system, up to 48 disk drives can be connected to the IOS.

DD-29 operational characteristics include:

Bytes per sector: 4096

Words per sector: 512 (64-bit words)

Sectors per track: 18

Words per track: 9216

Tracks per cylinder: 10

Words per cylinder: 92,160

Access time: 15-80 msec.

Transfer rate: approx. 34 Mbits per second

Latency : 16.7 ms
(revolution time)

Magnetic Tape

An I/O Subsystem can include an Auxiliary I/O Processor (XIOP) with the capability of addressing up to 16 block multiplexer channels of tape units.

Each block multiplexer channel can be attached to IBM-compatable control units and tape drives in a variety of configurations.

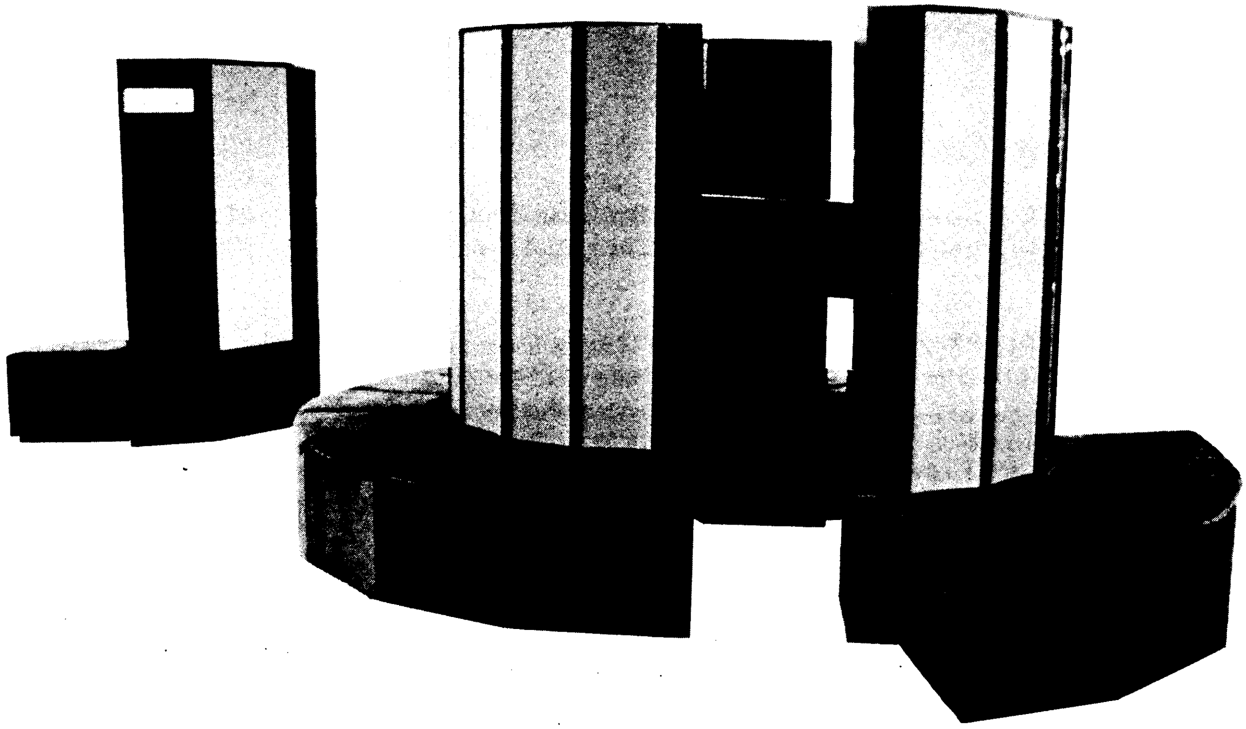
The block multiplexer channels communicate with the control units and tape units to allow reading and writing data that can also be read and written on IBM-compatable CPUs.

The physical characteristics of tape devices are summarized below.

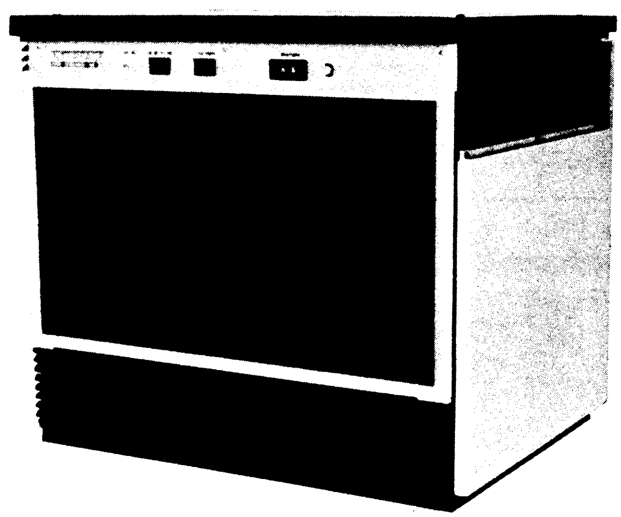
The block sizes listed are for transparent-format tape datasets (described in Lesson 5).

Physical characteristics of 200 ips, 9-track tape devices.

Density (bits/inch)	Transfer rate (kilobytes/sec)	Data/2400 ft. reel (megabytes)	% of reel containing data	Block size (bytes)
6250	1170	168	94	32768
1600	300	43	94	16384



CRAY X-MP mainframe with a Cray I/O Subsystem and an SSD



Typical interface cabinet

Solid-state Storage Device

The SSD is a volatile mass storage device which uses MOS memory chips to hold large volumes of data.

Storage capacities available include 64, 128 or 256 megabytes, arranged in banks similar to those used in the Cray central memory layout, although datasets are logically identical to those stored on a disk.

The SSD avoids the mechanical constraints of conventional disk drives (rotation, seek times, etc.) which result in significant performance improvements when accessing datasets.

Data is transferred over four 100 Mbyte/sec channels on the CRAY-1 machines and over the 1250 Mbyte/sec channel on the X-MP.

Interface Cabinet

The CRAY computer system is designed for use with a network of front-end computers. Front-ends connect to the MIOP of the IOS via the asynchronous channels discussed earlier.

The front-end interfaces, consisting of electronics and cable connections, are housed in a stand-alone cabinet.

The hardware performs command translation and protocol conversion needed to transfer data.

SUMMARY OF CRAY 1/S AND CRAY 1/M DIFFERENCES

CRAY 1/S	CRAY 1/M
ECL MEMORY (EMITTER-COLLECTOR (EMITTER-COLLECTOR LOGIC)	MOS MEMORY (METAL-OXIDE SEMI- (METAL-OXIDE SEMI-CONDUCTOR)
8 OR 12 COLUMNS	6 COLUMNS
IOS OPTIONAL PART	IOS INTEGRAL PART
½ MILLION WORDS IN 8 BANKS 1 OR 2 MILLION WORDS IN 8 OR 16 BANKS	1 MILLION WORDS IN 8 BANKS
4 MILLION WORDS IN 16 BANKS	2 OR 4 MILLION WORDS IN 16 BANK
MODULE IN MEMORY CONTAINS 1 BIT OF A 64-BIT CRAYWORD	MODULE IN MEMORY CONTAINS 8 BITS. OF A 64-BIT CRAYWORD
1 BANK/CHASSIS - 2 BANKS/COLUMN	4 BANKS/CHASSIS - 8 BANKS/COLUMN
12/5 NSEC CLOCK PERIOD	12.0 NSEC CLOCK PERIOD
4 C.P. BANK CYCLE TIME	8 C.P. BANK CYCLE TIME
12 SYNCHRONOUS/ASYNCHRONOUS CHANNEL PAIRS	4 ASYNCHRONOUS CHANNEL PAIRS
11 C.P. FOR SCALAR MEMORY REF.	13 C.P. FOR SCALAR MEMORY REF.
14 C.Ps FOR FETCH ON 16 BANK	18 C.Ps FOR FETCH ON 16 BANK
18 C.Ps FOR FETCH ON 8 BANK	22 C.Ps FOR FETCH ON 8 BANK
50 C.Ps FOR EXCHANGE SEQUENCE	54 C.Ps FOR EXCHANGE SEQUENCE
1/4 & 1/2 SPEED CONTROL FOR VECTOR REGISTER LOADS/STORES	1/8, 1/4 & 1/2 SPEED CONTROL FOR VECTOR REGISTER LOADS/STORES

System Configurations

Several combinations of the basic system components are supported in the Cray computer series. Central memory is available in several different sizes, the I/O Subsystem is available in several different configurations, and peripheral equipment like the SSD are optional.

The following is a summary of the various configurations available with the three Cray computer systems.

CRAY X-MP system characteristics

Configuration	- Mainframe with 2 Central Processing Units (CPUs) - I/O Subsystem with 2, 3, or 4 I/O Processors - Optional Solid-state Storage Device (SSD)
CPU speed	- 9.5 ns CPU clock period - 105 million floating-point additions per second per CPU - 105 million floating-point multiplications per second per CPU - 105 million half-precision floating-point divisions per second per CPU - 33 million full-precision floating-point divisions per second per CPU - Simultaneous floating-point addition, multiplication, and reciprocal approximation within each CPU
Memories	- Up to 4 million 64-bit words in mainframe Central Memory - 65,536 16-bit parcels in Local Memory of each I/O Processor of the I/O Subsystem - 6 direct memory access (DMA) ports to Local Memory (each I/O Processor) - 1, 4, or 8 million 64-bit words of I/O Subsystem Buffer Memory - 8, 16, or 32 million words of SSD memory
Mass storage	- 600 million byte disk drive - 48 disk drives maximum for I/O Subsystem - 35.4 Mbits per second disk drive transfer rate
Input/Output	- One 1250 Mbytes per second Solid-state Storage Device (SSD) channel on mainframe - Two 100 Mbytes per second channels between mainframe and I/O Subsystem for a system with an SSD - Four 100 Mbytes per second channels between mainframe and I/O Subsystem for a system without an SSD - Four 6 Mbytes per second channels - 40 channels; input or output, 24 of which share the six DMA ports per I/O Processor - Mainframe interfaces to I/O Subsystem
Physical	- 45 sq ft floor space for mainframe - 15 sq ft floor space for I/O Subsystem - 15 sq ft floor space for SSD - 5.25 tons, mainframe weight - 1.5 tons, I/O Subsystem weight - 1.5 tons, SSD weight - Liquid refrigeration of each chassis - 400 Hz power from motor-generators

MCU

For systems with an MCU, after the Cray Operating System has been initialized and is operational, communication with the MCU is by software protocol.

The MCU has a software package that enables it to serve as a local batch station during production hours. As a local station, the MCU can submit diagnostic routines for execution or can submit other batch jobs. These diagnostics are typically stored on a local disk and are submitted to the Cray mainframe by operator command.

System Startup

The Cray mainframe is deadstarted by loading the operating system from the 80 Mbyte MCU disk into central memory.

For systems configured with an I/O Subsystem, the IOS is first started from the peripheral expander magtape unit, or the 80 MB disk drive. Once the IOS is started, the Cray can be deadstarted from the IOS.

Exchange Mechanism

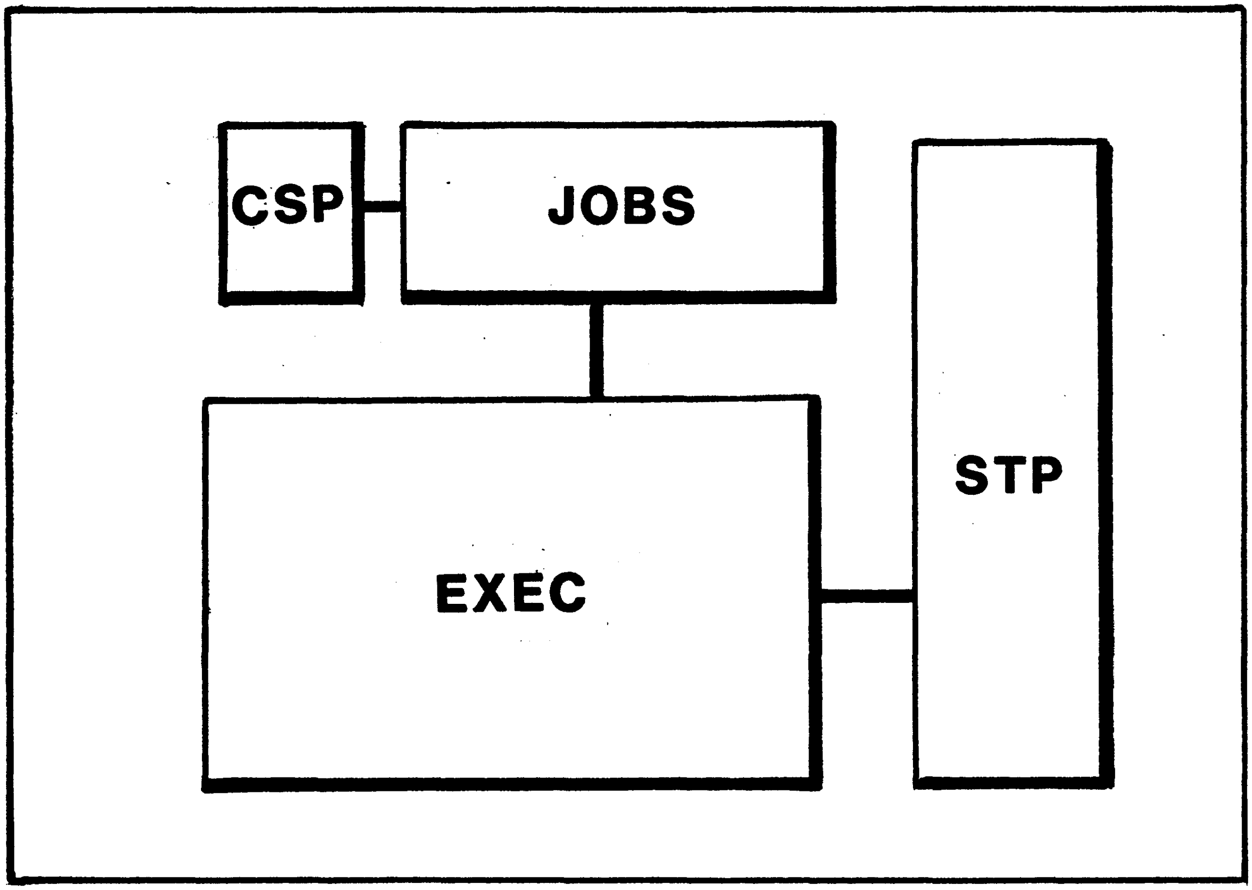
Since the Cray is a multiprogramming machine, the hardware must be capable of switching execution from one program to another. This is called the Exchange Mechanism. A 16-word block of program parameters is maintained for each program. When another program is to begin execution, an operation known as 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 interupt condition, or voluntarily by the user or by the operating system through normal exit instructions.

The Exchange Mechanism and Exchange Packages for the Cray-1 and X-MP are discussed in detail later in this course unit.


CSP
JOBS
EXEC
STP

LESSON 3: Software Components

Objective: List the software components of the Cray system, and their function.

CRAY SOFTWARE

The Cray computer requires three types of software:

* an Operating System
* Language Systems
* Applications Programs

The I/O Subsystem also requires its own set of software, including:

* An Operating System
* I/O and Communications software

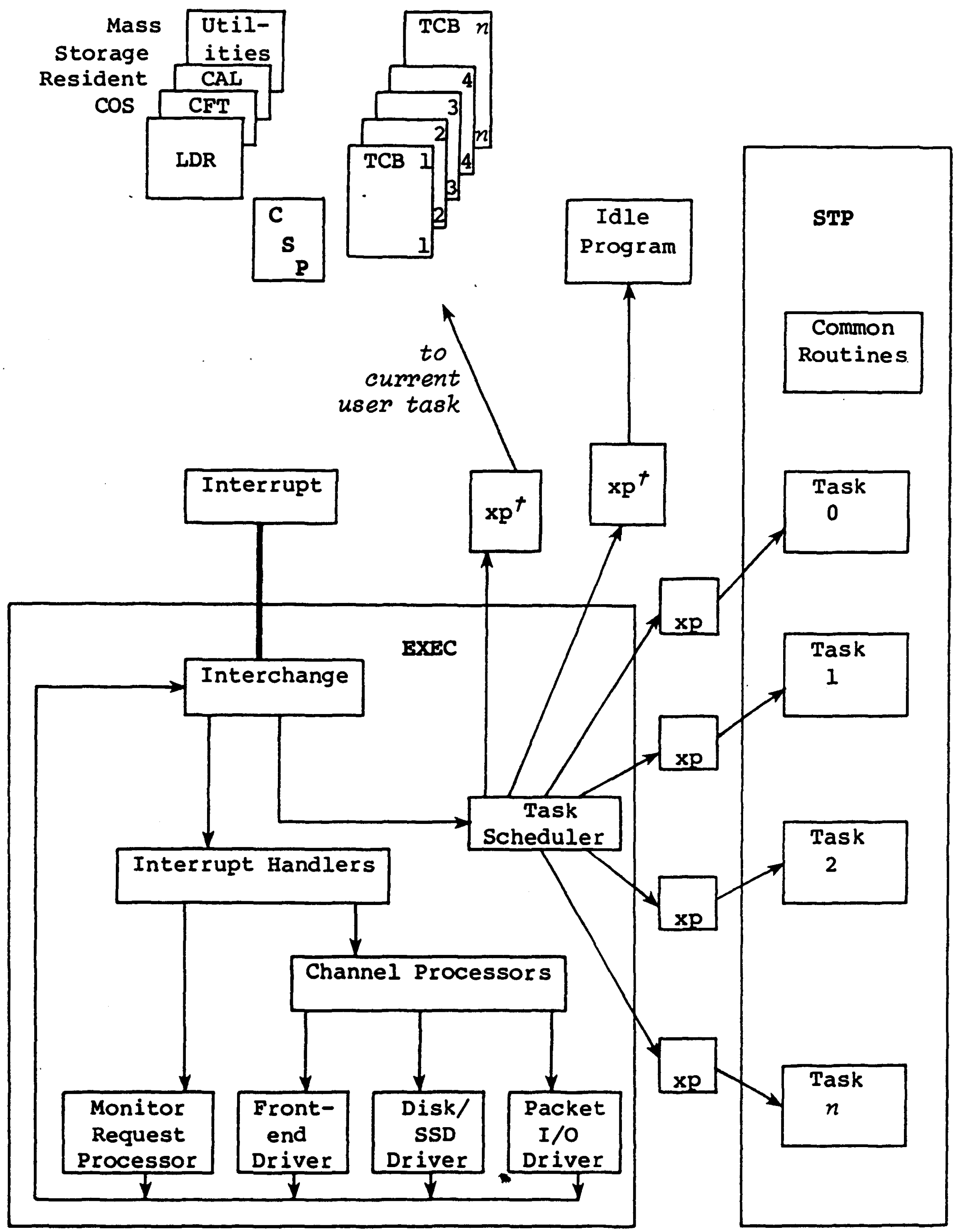
The CRAY OPERATING SYSTEM

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 consists of the following modules that execute on the CPU(s):

* The Executive (EXEC)
* System Task Processor (STP)
* Control Statement Processor (CSP)
* Utility Programs

INTERRUPT HANDLERS



System control

† One Exchange Package per CPU

EXEC

The system Executive (EXEC) is the control center for the Cray operating system. It alone accesses all of memory, controls the I/O channels, and selects the next program to execute. Components of EXEC include:

* 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

STP

The System Task Processor (STP) runs in user mode and accesses all memory other than that occupied by EXEC and is responsible for processing all user requests.

STP consists of a set of routines called **tasks**, tables, and some reentrant routines common to all tasks. It is these tasks that perform the bulk of the work in job processing.

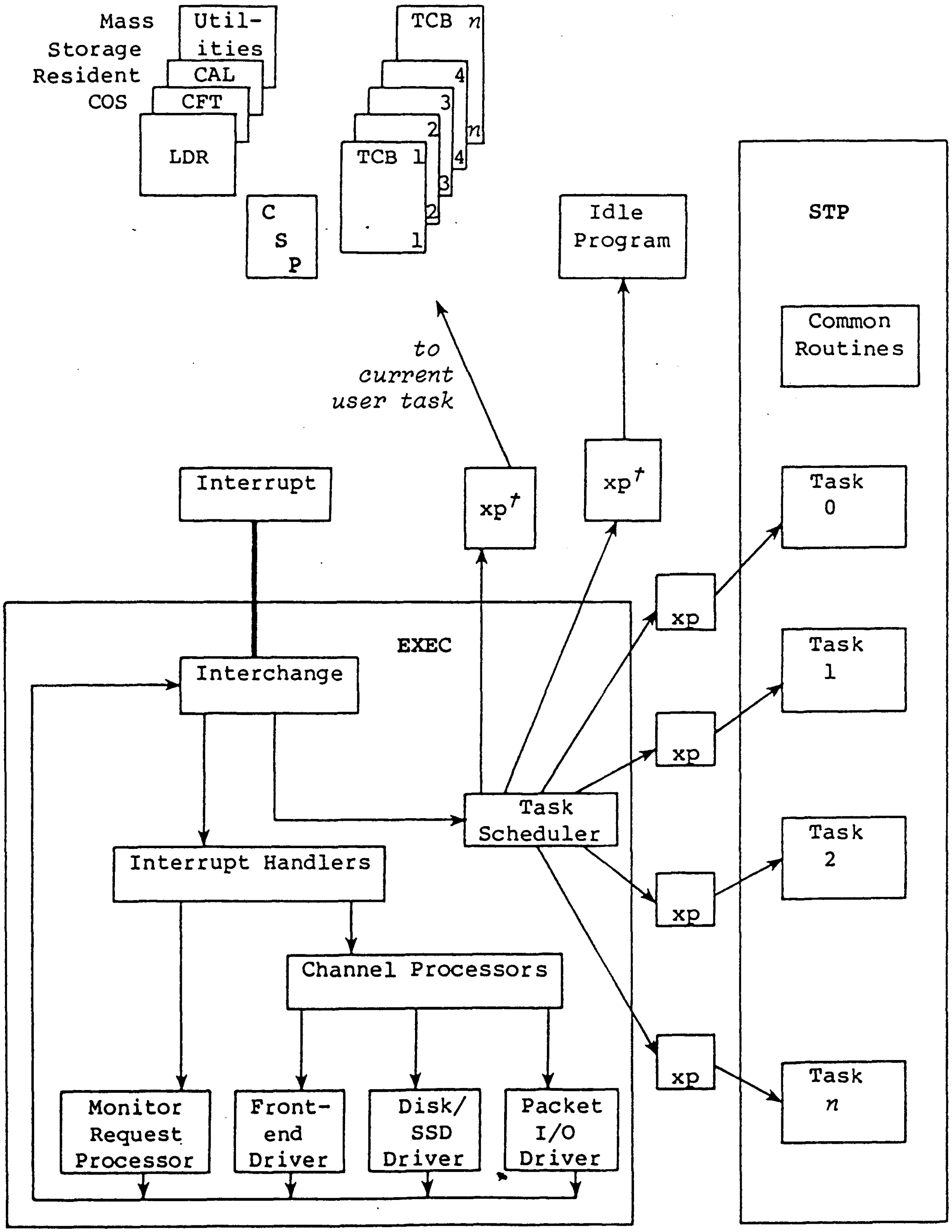
CSP

The Control Statement Processor (CSP) is a system program that executes in the user field. CSP initiates the job, cracks the JCL statements, processes system verbs, advances the job step-by-step, processes errors, and ends the job.

Utilities

Utility programs include the loader, a library generation program (BUILD), a source language maintenance program (UPDATE), permanent dataset utility programs, copy, and positioning routines, etc.

INTERRUPT HANDLERS



System control

† One Exchange Package per CPU

LANGUAGE SYSTEMS

Currently, five language systems developed by Cray Research are provided for the Cray computer system. They are:

* The FORTRAN compiler (CFT)
* The Cray Assembly Language program (CAL)
* The PASCAL compiler
* and A Programming Macro Language (APML) for the IOS.

CFT

The Fortran compiler is designed to take advantage of the vector capability of the Cray computers.

The compiler itself determines the need for vectorizing and generates the code accordingly. Optimizer routines examine Fortran source code to see if it can be vectorized. The compiler conforms to ANSI Fortran 77 standards.

CAL

The CAL assembler provides users with a means of expressing all hardware functions of the CPU symbolically. Augmenting the instruction set are pseudo instructions that provide users with options for generating macros.

Most of the software provided by Cray Research, including the operating system, is coded in CAL.

Pascal

The Pascal compiler supports the ISO Version 1 Pascal standard and also provides extensions to that standard.

APML Assembler

The APML assembler executes on the Cray mainframe and generates absolute code that is executable in the Cray I/O processors of the I/O Subsystem.

It is used to generate the I/O Subsystem software.

LIBRARY ROUTINES

Cray software includes a group of subprograms that are callable from CAL or FORTRAN programs. These subprograms reside in libraries named $ARLIB, $FTLIB, $IOLIB, $UTLIB, $SCILIB, $SYSLIB, and $PSCLIB.

They are grouped by UPDATE deck name within each library. (UPDATE is the source maintenance utility.)

$ARLIB contains routines primarily concerned with returning some numeric result. Mathematical routines intrinsic to FORTRAN such as SIN and XOR reside here.

$FTLIB contains CFT-specific routines such as LEN (length of argument)

$IOLIB contains routines that move data from external devices to main memory or control that movement (WRITEC, COPYR, etc.)

$UTLIB contains special utility programs such as TIMEF which returns elapsed time in millisecs since last call, and CRAYDUMP which prints a memory dump to a specified dataset.

$SCILIB routnies perform operations such as matrix multiply or Fast Fourier transform and must be explicitly called.

$SYSLIB routines usually link directly to the operating system through a normal exit. These routines are not usually accessable to a user, but are called by $IOLIB and $UTLIB routines for specific tasks.

In general, $SYSLIB serves as a link between the general purpose $IOLIB and $UTLIB routines and the details of COS. $SYSLIB routines depend on specific COS features.

An example of a $SYSLIB routine is CCS which cracks job control statements for the Control Statement Processor (CSP).

IOS SOFTWARE

The major parts of I/O Subsystem software are:

* The Kernel
* Disk input/output
* Tape Exec
* Block Multiplexer Channel Interface
* The Station
* The Front-End Concentrator
* The Interactive Station

The Kernel

The Kernel serves as the operating system for the I/O Processors. A copy of the Kernel runs in each IOP and adapts itself to the special functions of that processor.

Kernel functions consist of answering interrupts, managing overlays areas, and handling independent activities running in the I/O Subsystem.

Because of the limited size of local memory (65K), the I/O Processor software uses overlays extensively. An overlay is an executable program or subroutine that normally resides in Buffer Memory. It is called into local memory by the Kernel to perform its specific function.

Disk Input/Output

The I/O Subsystem for the Cray X-MP and Cray 1 models S/1200 through S/4400 provides for operator station-maintenance functions and also is a substitute for disk controllers.

The disk software provides I/O execution times for I/O requests that are comparable to times using the disk controller and allows a greater number of concurrent disk I/O requests or streams to occur through the use of buffering in Buffer Memory.

Tape Exec

The block multiplexer Tape Exec is composed of activities necessary to route messages, process requests, format and move data, and recover from errors.

The Tape Exec receives tape requests from the Tape Queue Manager (an STP system task). Requests are read into the MIOP across the 6 Mbyte channel. Based on the ID, the request is sent to the XIOP for disposition.

The Block Multiplexer Channel Interface

The block multiplexer channel interface provides an interface between any block multiplexer device driver (such as the driver for magnetic tape) and block mulitplexer channel hardware. The interface performs channel selection and load leveling, channel error recovery and reporting, and device-independent command and interrupt sequences.

The Station

The station is a collection of closely associated tasks executing in the MIOP that provide operator command and display facilities and dataset staging capabilities independent of any front-end computers.

The Front-End Concentrator

The I/O Subsystem Concentrator relieves the mainframe from the burden of handling the interrupts for each subsegment of messages transferred between the mainframe and attached front ends.

The concentrator looks like a Cray channel pair from the front-end's point of view, so no changes are necessary in existing front end stations. The concentrator can handle data from multiple IDs through one channel, even though each front-end ID may have a different segment size.

The Interactive Station

The interactive station is a set of tasks running in the Master I/O Processor that permit consoles connected directly to the MIOP to become attached to mainframe jobs. A job is created in the mainframe when an interactive console logs on.

This station is composed of two parts, the interactive concentrator and the interactive console. The interactive concentrator gathers messages from the consoles, sends them to the mainframe, receives responses, and distributes them to the console routines. The interactive console routines handle the input and output from and to the consoles and prepare messages to be sent to the mainframe via the interactive concentrator.

STATION SOFTWARE

Station software provides the interface between the Cray computer system and the Front-end computers supplied by other manufacturers.

Among the front-end systems currently being used with Cray computers are:

* IBM / MVS
* IBM / VM
* CDC NOS
* CDC NOS/BE
* DEC VAX/VMS
* DGC RDOS

IBM /MVS

The Cray/IBM MVS station provides the link between an IBM System/370 or 370 compatable and the Cray computer system.

The station provides for:

* Job submission at TSO terminals
* Local Batch Entry
* Remote Batch Entry
* Transfer of job and data files between MVS magnetic storage and Cray mass storage.

An operator communicates with the MVS station and the Cray via commands entered at an MVS station console or a TSO terminal.

IBM /MVS (continued)

The operator at an MVS station console can query and dynamically alter the status of jobs and the MVS station. The TSO user can only query and dynamically alter the status of jobs with a terminal ID (TID) equal to the user's TSO logon ID.

One station in the Cray system is designated at installation time as the master operator station. The operator of this master station has complete control of COS and can manipulate all jobs in the system, control all mass storage, and set COS system parameters. All other stations in the system can only alter those jobs pertaining to that station.

CDC NOS & NOS/BE

The CDC NOS station controls the link between a Cray computer system and the Control Data Corporation Cyber 70, 170 or 6000 Computer System. This interface enables:

* Remote and local batch access to the Cray for users of the CDC system, and

* Job and data file transfer between CDC mass storage or tape and Cray mass storage.

Job files can be transmitted only from NOS to COS, not vice versa.

The physical connection between the Cray and the CDC computers may be a channel-to-channel front-end coupler device manufactured by Cray Research, or via a Hyperchannel network, manufactured by Network Systems Corp.

LESSON 4: Memory Layout

Objective: Describe the general organization and layout of the Cray and IOS main memories.

INTRODUCTION

COS is loaded into Central memory and activated through a system startup procedure performed at the MCU or I/O Subsystem.

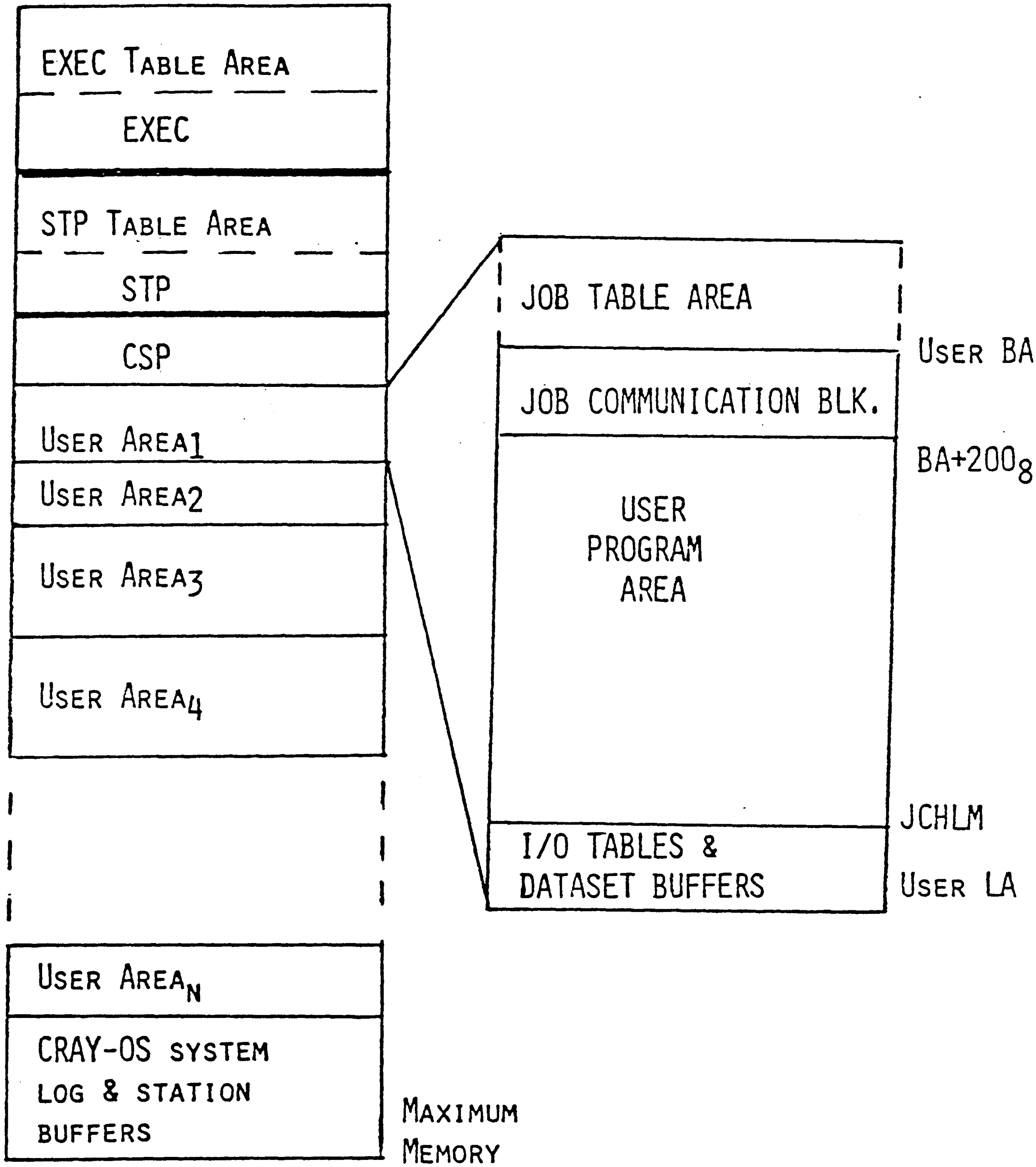
Memory is shared by:

* COS,
* jobs running on the Cray mainframe,
* dataset I/O buffers,
* and system tables associated with those jobs.

COS allocates resources to each job, when needed, as these resources become available.

As a job progresses, information is transferred between central memory and mass storage. These transfers can be initiated by either the job or by COS.

SYSTEM MEMORY ASSIGNMENTS



Memory Resident COS

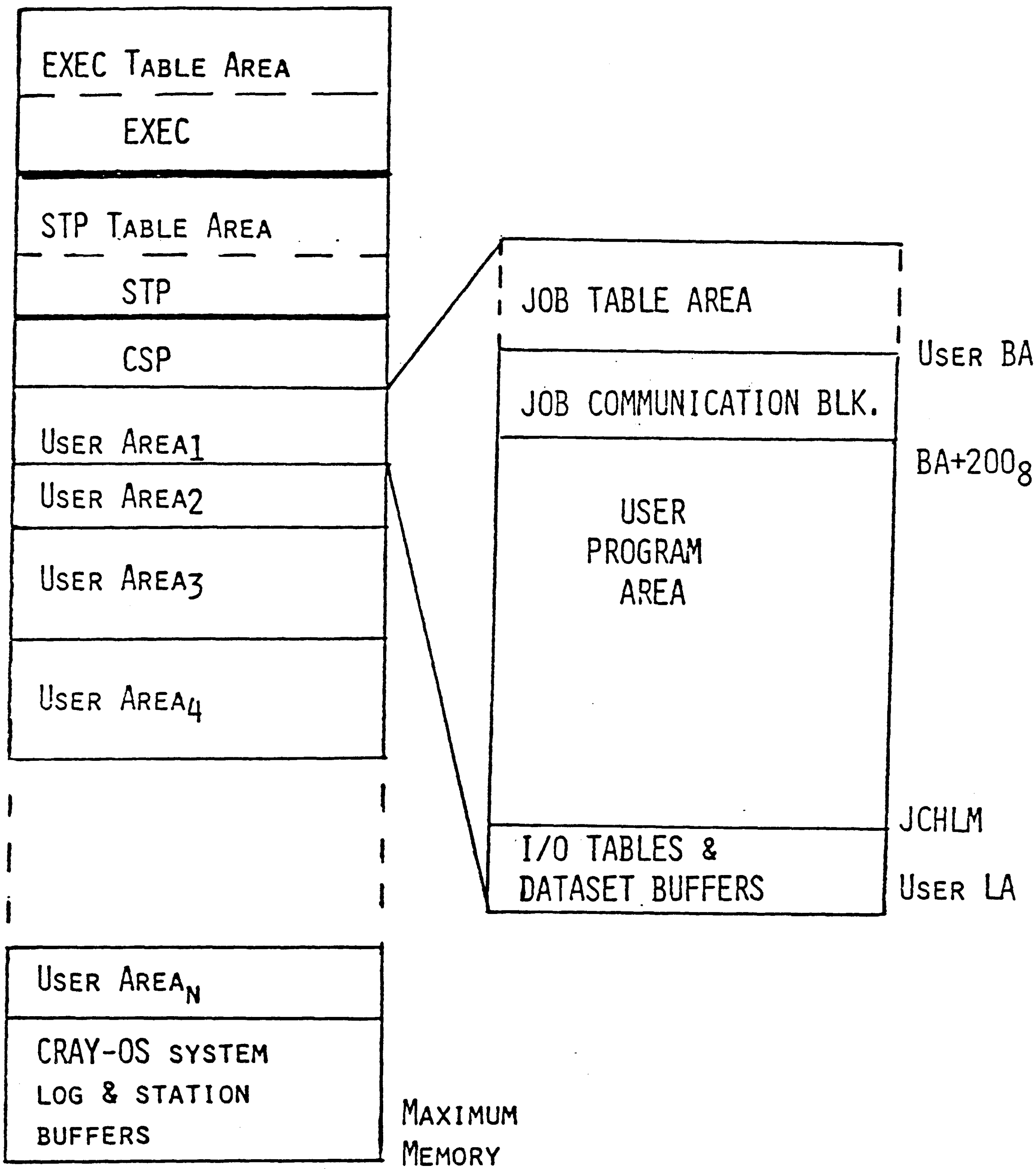
COS occupies two areas of central memory. The memory-resident portion of the operating system occupying the lower memory consists of:

* Exchange Packages
* The System Executive (EXEC)
* The System Task Processor (STP)
* and optionally, The Control Statement Processor (CSP)

The memory-resident portion of the operating system occupying extreme upper memory contains:

* Station I/O buffers
* space for the System Log buffer, and
* Permanent Dataset Catalog (DSC) information & buffers

SYSTEM MEMORY ASSIGNMENTS



USER AREA

COS assigns every job a user area in central memory. The user area consists of a Job Table Area (JTA) and a User Field.

Job Table Area - JTA

For each job, the operating system maintains an area in memory that contains the parameters and information required for monitoring and managing the job. This area is called the **Job Table Area** (JTA). Each active job has a separate Job Table Area adjacent to the job's User Field. The Job Table Area is not accessable to the user, although it can be dumped for analysis.

The JTA contains jobrelated information such as accounting data; Job Execution Table pointer; areas for saving B, T, and V register contents; control statement and logfile Dataset Parameters; a logfile buffer; and a Dataset Name Table 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.

User Field

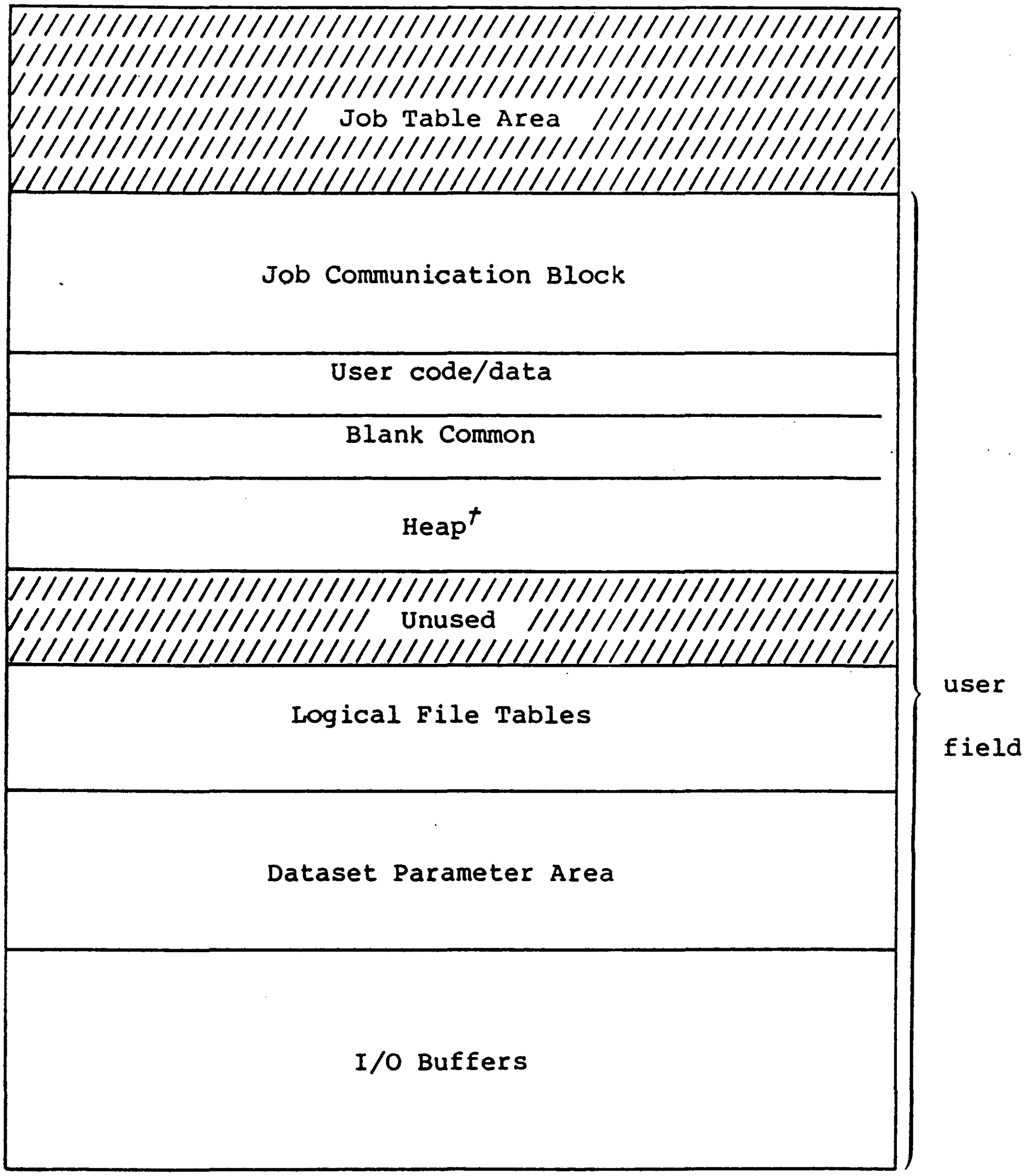
The **user field** for a job is a block of memory immediately following the job's JTA. The user field is always a multiple of 512 words.

The user field, in addition to being used for user-requested programs such as the compiler, assembler, and application 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.

The beginning or **Base Address** (BA) and the end or **Limit Address** (LA) are set by the operating system. The maximum user field size is specified by a parameter on one of the JCL statements that accompany the job, or by an installation-defined default.

A user can request that the user field size increase during the course of a job.

The first 128 words of the user field (200 octal) are reserved for an operating system/job communication area known as the Job Communication Block (**JCB**). The JCB contains a copy of the current control statement for the job as well as other job-related information.


Job Table Area
Job Communication Block
User code/data
Blank Common
Heap†
Unused
Logical File Tables
Dataset Parameter Area
I/O Buffers
user field

Programs are loaded starting at BA + 200 (octal) and reside in the lower portion of the user field. The user field addressing limit is equal to LA1.

The upper portion of the user field contains dataset buffers and I/O tables.

Tables that reside in the user field include:

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 (Dataset Name Table) in the user's JTA.

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 of the dataset.

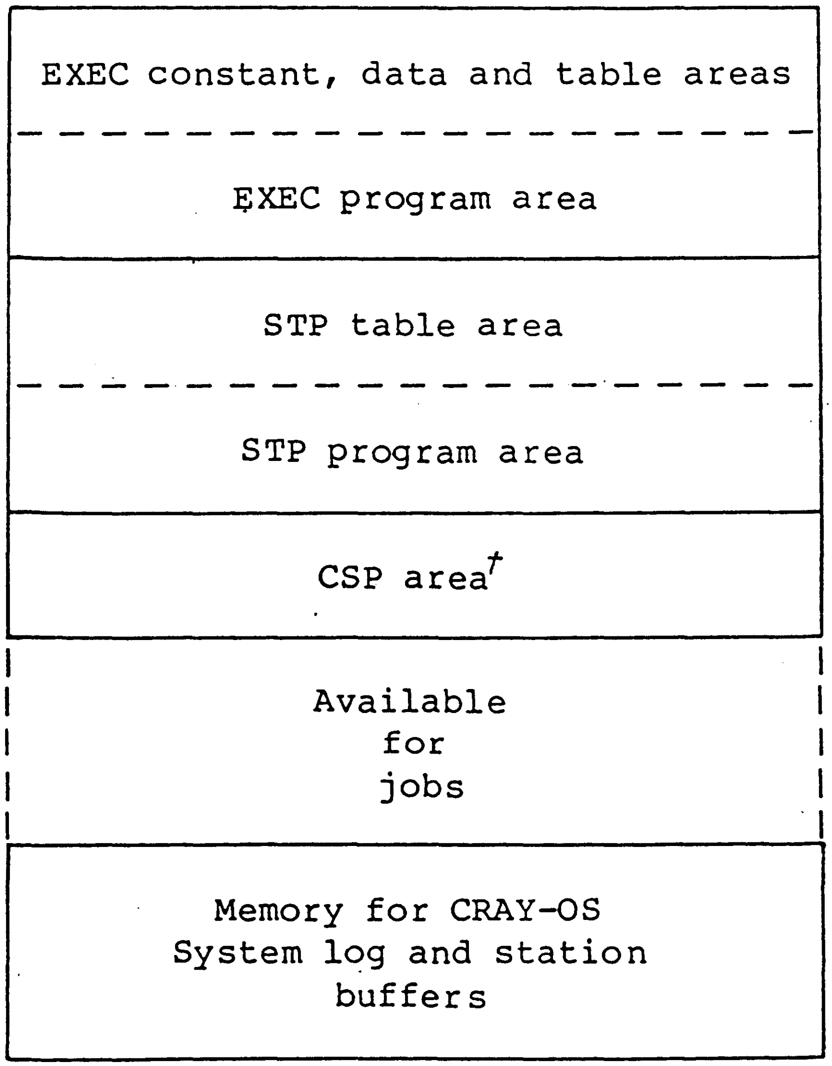
JAC Job Accounting Table. This table defines an area for data to be returned to the user by an accounting request.

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 recreation.

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 is used by CSP for many permanent dataset requests.


EXEC constant, data and table areas
EXEC program area
STP table area
STP program area
CSP area†
Available for jobs
Memory for CRAY-OS System log and station buffers

COS Residence

As mentioned previously, the lower portion of COS residence includes:

- * The System Executive (EXEC)
- * The System Task Processor (STP)
- * and optionally, The Control Statement Processor (CSP)

EXEC

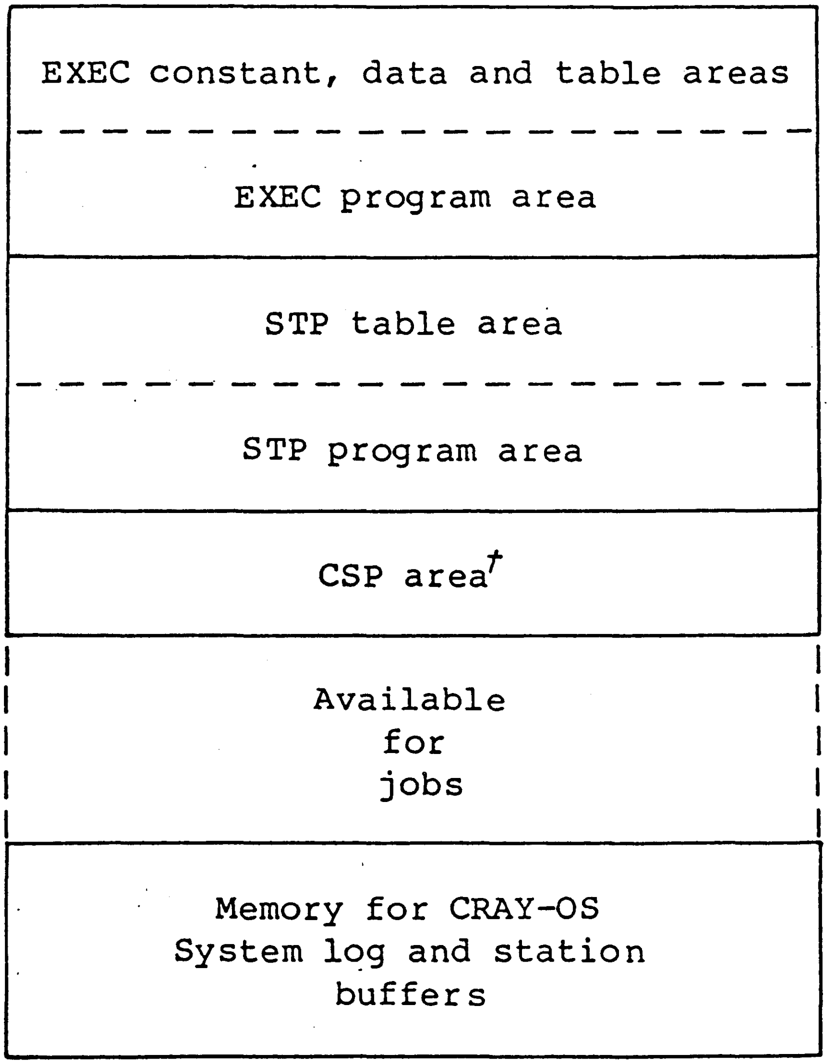
The EXEC portion of COS has a base address (BA) of 0 and a limit address which is set by an installation parameter. The EXEC area of memory consists of the EXEC Constant, Data, and Table Areas, and the EXEC Program area.

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

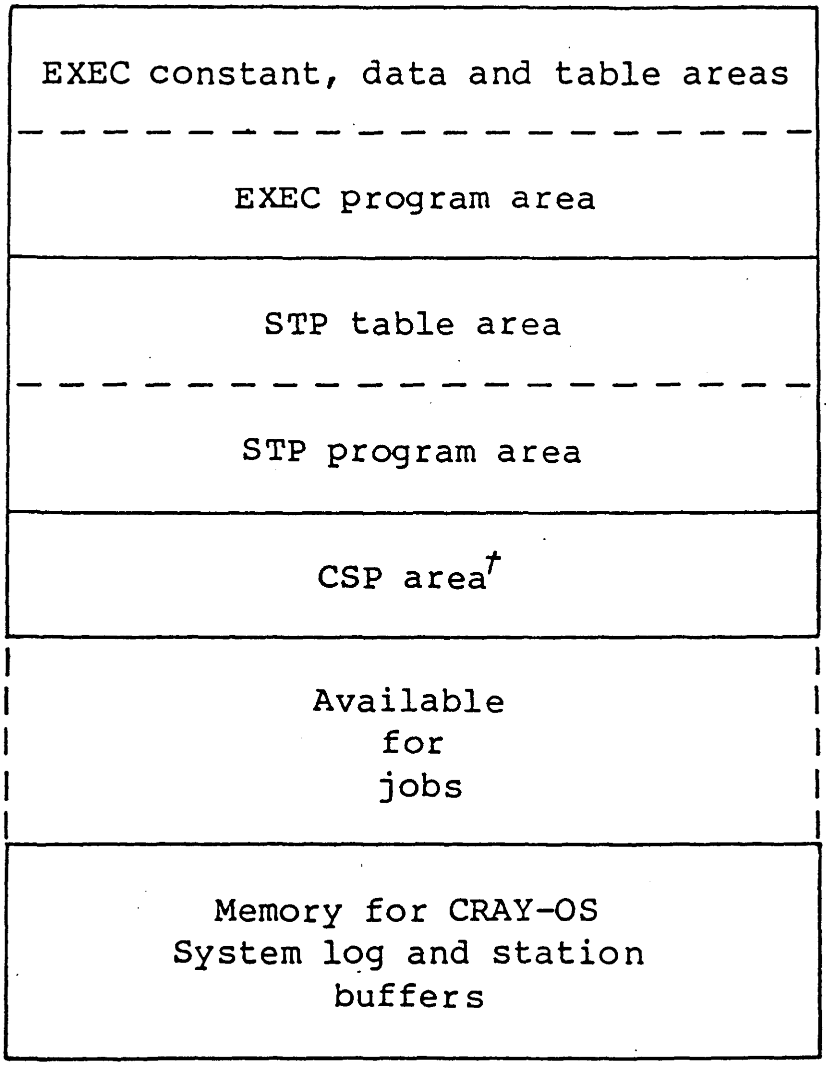

EXEC constant, data and table areas
EXEC program area
STP table area
STP program area
CSP area†
Available for jobs
Memory for CRAY-OS System log and station buffers

The EXEC Table area contains all EXEC tables, alphabetically ordered. A description of these tables and there function is discussed in Lesson 10 of this unit.

The EXEC Program area contains 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.

Explanation of the purpose and function of these EXEC components is discussed in Lesson 10 of this unit.


EXEC constant, data and table areas
EXEC program area
STP table area
STP program area
CSP area†
Available
for
jobs
Memory for CRAY-OS
System log and station
buffers

System Task Processor

The second major component of COS residence is STP. STP is the portion of COS which is responsible for processing all user requests. The STP area consists of tables, a set of programs called tasks, and some reentrant routines common to all tasks.

STP Program area

The STP program area consists of the system tasks and the reentrant routines. A System task serves a specific purpose in the job processing cycle.

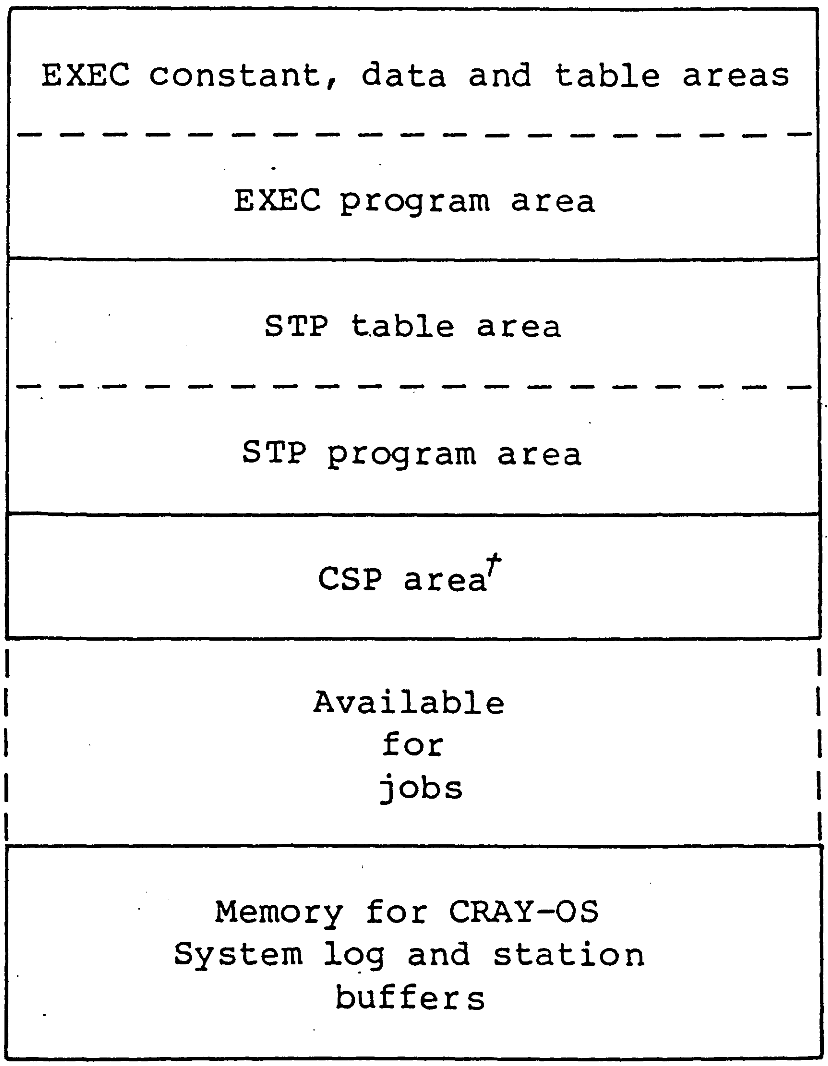
The system tasks and their abbreviations are:

* Startup (STP)
* 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)

The detailed function of these system tasks and STP in general is discussed in Lessons 12 and 13 of this unit.

STP Table area

This area contains 35 different tables accessable to all STP tasks. The purpose and layout of the individual tables will be discussed in Lesson 12 of this unit as well as during Unit 3: COS Internals.

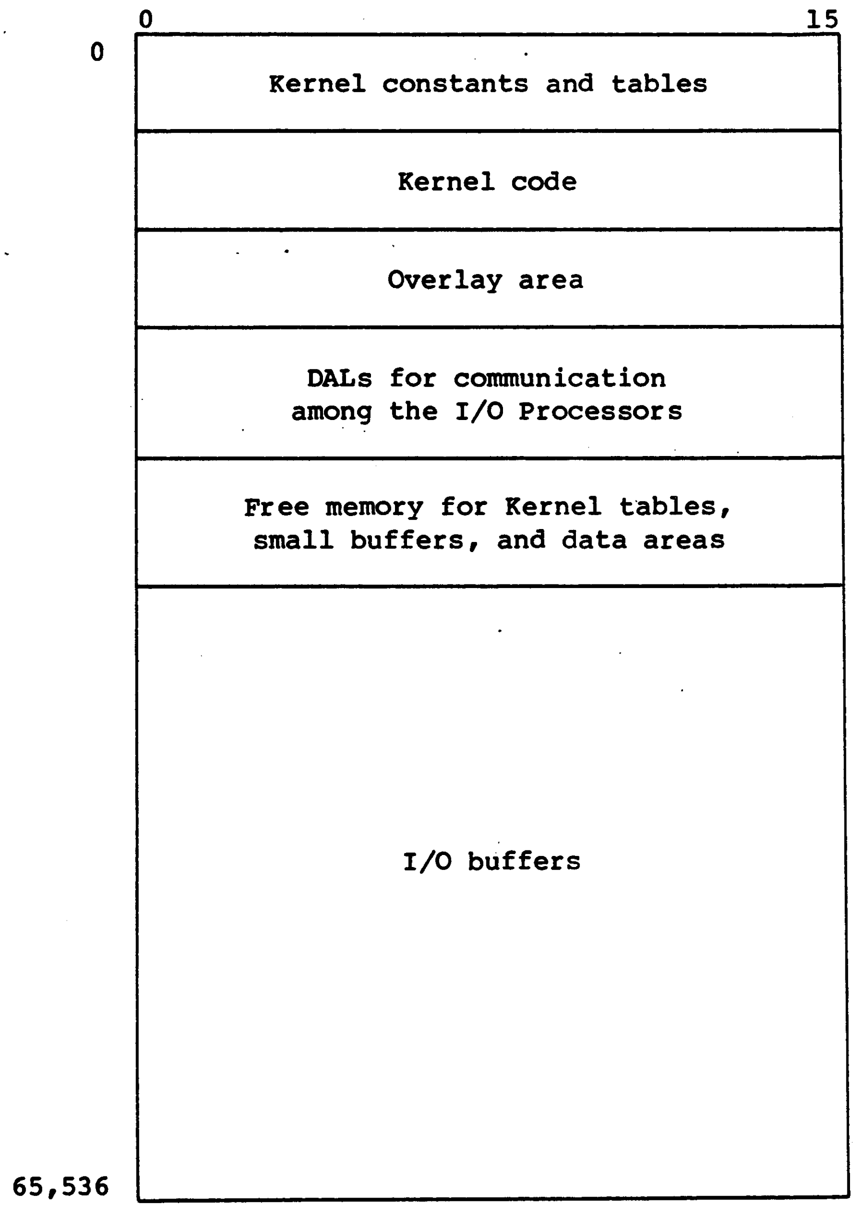

EXEC constant, data and table areas
EXEC program area
STP table area
STP program area
CSP area†
Available for jobs
Memory for CRAY-OS System log and station buffers

Control Statement Processor (CSP)

An image of CSP is maintained either in memory following STP or on mass storage, depending on an installation parameter.

CSP is copied into each user field where it executes each time the job requires interpretation of a control statement.

CSP is discussed in greater detail in Lesson 14 of this unit.



Local Memory structure

I/O Subsystem

The Kernel

The Kernel is the software package that controls activities running in each I/O Processor. Although each I/O Processor has its own copy of the Kernel, all copies are basically the same.

Local Memory Usage

Kernel code is stored in local memory apart from the constants and tables it references. The table area contains configuration maps, memory allocation tables, activity dispatching parameters, and information about overlays in buffer memory and local memory.

Overlays

Because of the limited size pf local memory, the I/O Processor uses overlays extensively. The overlay is read into loacl memory when activated to perform some function. Overlay space is allocated dynamically as new overlays are loaded.

DALs

The DAL area contains a linked list of 32-parcel communications packets.

Free Memory

The free memory area is used for Kernel tables and small buffers and is organized as a chain structure. Free memory is allocated in multiples of four parcels.

I/O Buffer area

The I/O buffer area is allocated in increments of 512 64-bit words.

The relative size of each of these types of areas in local memory is determined by installation parameters, and depends on the functions that each IOP performs. For example, an IOP used exclusively for disk I/O has a greater share of local memory assigned to I/O buffers than an IOP which performs a different function.

0	63

0

Buffer Memory	
Deadstart package	
System Directory	
Message area (For communicating control information) MIOP BIOP IOP-2 (DIOP) IOP-3 (DIOP or XIOP)	(Size of area set in AMAP for each IOP. Each message area is 40_8 parcels.)
AMAP (Units attached to each IOP)	
Overlays (Read only, shared by all IOPs)	
MIOP Kernel storage: - Tables and queues - Software stack area (400_8 each) - I/O buffers - Trace buffer - Other memory requirements	
IOP-2 Kernel storage (same as MIOP Kernel storage description)	
IOP-3 Kernel storage (same as MIOP Kernel storage description)	
BIOP Kernel storage (same as MIOP Kernel storage description)	
Buffer Memory resident datasets	

Buffer Memory organization

Buffer Memory Usage

The I/O Processors share Buffer Memory.

The first locations are reserved for a deadstart package. During deadstart, the MIOP initializes common tables and the System Directory so that all the control information is ready to begin execution when the other I/O Processors are deadstarted.

System Directory

The System Directory begins at the first address after the deadstart package. It contains pointers to other information saved in Buffer Memory, including message area locations for each processor, and pointers to Kernel storage reserved for each processor. All the IOPs can access the System Directory, but information in the directory can be changed only by the MIOP during deadstart.

Message Areas

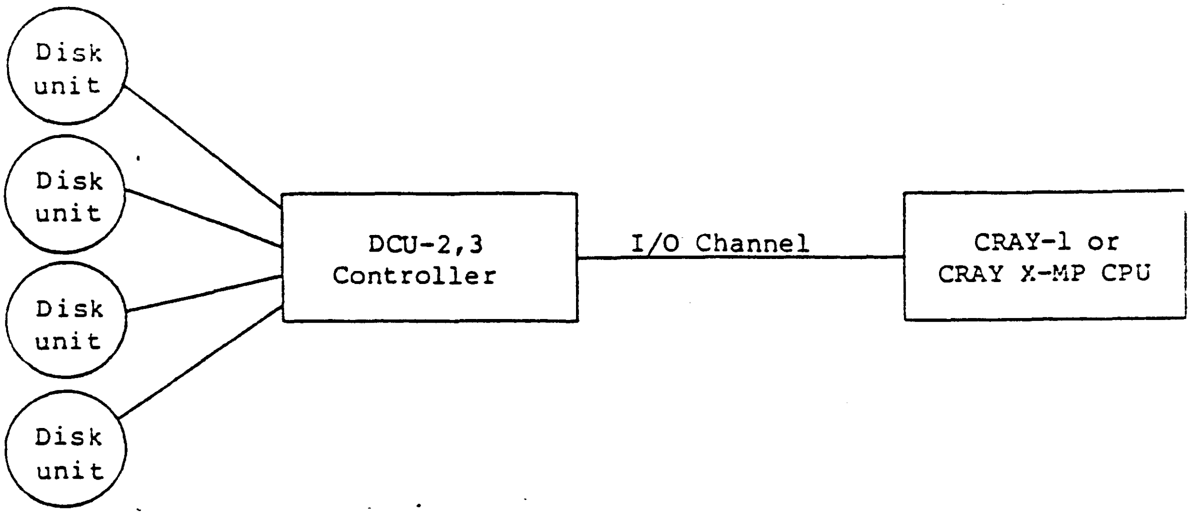
Message areas accessed by senders and receivers of messages follow the System Directory. The sending I/O Processor maintains control of the area and allocates or deallocates memory within it. The receiving processor signals when the message has been received and processed. The memory is then released to the pool of message areas belonging to the sender.

Kernel Area

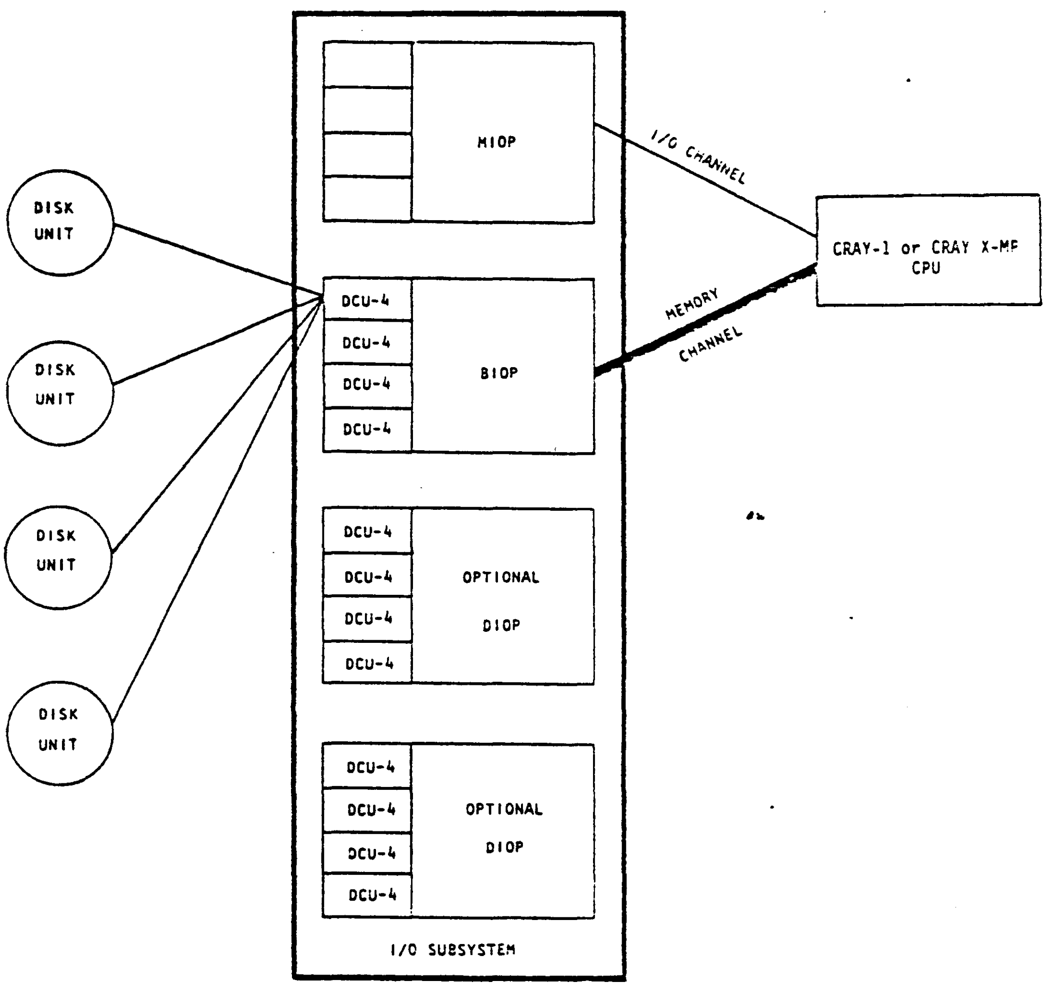
Each IOP has access to its own reserved Kernel storage area, which holds temporary information about activities and swapped activity areas. Reserved areas also provide data buffer storage for disks and other peripherals. A buffer also is reserved for history trace information. Each area is solely under the control of its respective I/O Processor.

Buffer Memory Resident Datasets

Part of Buffer Memory can be allocated for COS dataset storage.



DCU-2, DCU-3 controller configuration



DCU-4 controller configuration

LESSON 5: Mass Storage Organization

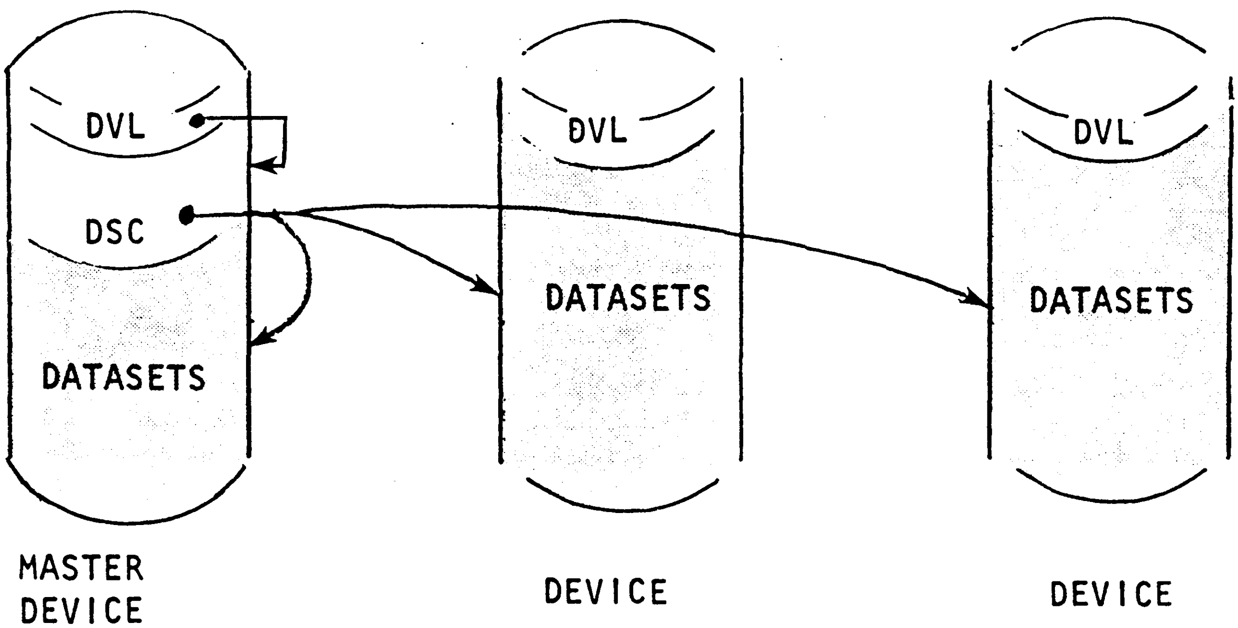
Objective: Describe the organization and layout of the Mass Storage subsystem, and the types and formats of the datasets used within the system.

DISK STORAGE

Depending on the Cray computer model, mass storage consists of IOS Buffer Memory, an SSD, and/or up to 48 DD-19 or DD-29 disk drives.

The DD-29 Disk Storage Unit is a 606 Mbyte drive with a data transfer rate of 35.3 Mbits per second. Each disk storage unit contains a device label, datasets, and unused space to be allocated to datasets.

One of the storage units will be designated the Master unit, and will contain the Dataset Catalog for all the datasets in the system.



Mass storage organization

Formatting

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

Device Label

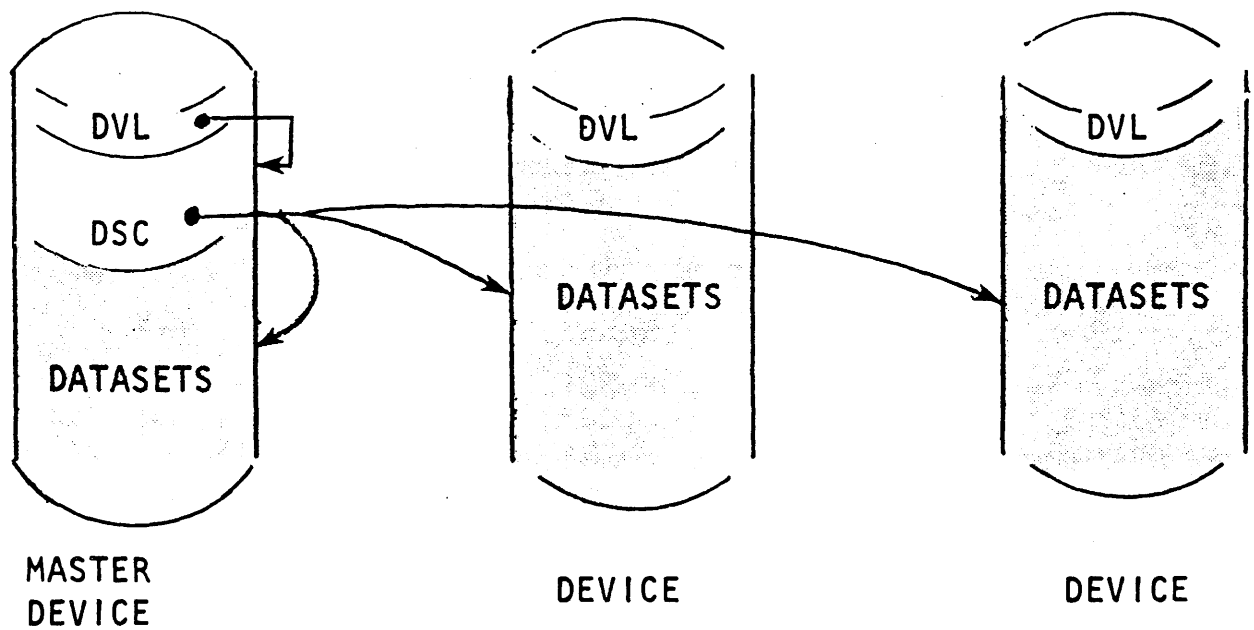
A disk storage unit must be labeled before it can be used by the system. The Install program writes a **Device Label Table** (DVL) on one track of each DSU. The DVLs act as the starting point for determining the status of mass storage when the system is dead-started or restarted. The location of the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

Flaw Information

A Device Label contains a list of flaws (bad tracks) for its disk storage unit. 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 (decimal) of the first track that can be successfully 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 the Device Label cannot be read back perfectly, then the track is overwritten with a test pattern and a different track is tried.

The Device Label 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.



Mass storage organization

Device Label (continued)

Dataset Allocation Table (DAT) for DSC

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

Dataset Catalog (DSC)

The Device Label Table (DVL) for the Master device states which tracks comprise the Dataset Catalog (DSC). Similarly, the Dataset Catalog states which tracks comprise each of the currently cataloged datasets.

Device Reservation Table

Deadstart and Restart update the **Device 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 job output datasets. Deadstart deletes all input and output datasets, defined by flags in the Dataset Catalog. Entries for these datasets in the DSC are zeroed. Restart, on the other hand, recovers the job input and output datasets.

Mass Storage Management

The system task **Disk Queue Manager** (DQM) controls the simultaneous operation of disk storage units on CPU I/O channels or the I/O Subsystem. DQM provides allocation and deallocation of mass storage and other management functions. A detailed discussion of DQM will occur later in this course unit.

5.6

DATASETS

Nearly all information maintained by COS is organized into units of information known as datasets. The following are some of the important factors to remember about datasets:

* The dataset **medium** is the type of physical device on which the dataset resides.

* The dataset **structure** is the logical organization of the dataset.

* The dataset **longevity** is the retention period for the dataset.

* A dataset must be **local** to be usable.

* The dataset **disposition code** tells COS what action to take when the dataset is no longer local.

* Each dataset is known by its **dataset name.**

* Datasets are read and written using operating system requests (user I/O interfaces).

Dataset Medium

Datasets can be classified by medium, such as:

* Mass Storage datasets

* Memory-resident datasets

* Interactive datasets

* Magnetic Tape datasets

Mass Storage Datasets

Mass storage datasets are of two types:

* **LOCAL**
* **PERMANENT**

Local Datasets

A local dataset exists only for the life of the job that created it and can be accessed only by that job.

Permanent Datasets

A mass storage dataset is permanent if it has an entry in the **Dataset Catalog** on disk. A permanent dataset is available to the system and can survive system deadstarts.

Permanent datasets are of two types:

* **User Permanent datasets**
  (those created with directives), and
* **System Permanent datasets**
  standard job input and output datasets

User 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 permanent dataset by prestaging a dataset from a front-end computer system or by using the **SAVE** or **ACQUIRE** control statements or macro.

A user accesses a user permanent dataset by using the **ACCESS** control statement or macro.

User Permanent Datasets (continued)

A 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 simply read the dataset may have multiaccess.

System Permanent Datasets

Some permanent datasets similar to user 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. (more about "rolled job index" later).

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 fron-end computer system for processing. They are deleted from mass storage when their receipt has been acknowledged by the front-end computer system.

Memory Resident Datasets

Some datasets can be specified by the user to be memory-resident datasets. A memory-resident dataset is wholly contained within one buffer and remains in the user's area of memory at all times.

A dataset can be declared memory resident to reduce the number of I/O requests and disk blocks transferred. Memory residence is particularly useful for intermediate datasets not intended to be saved or disposed to another mainframe.

All I/O performed on a memory-resident dataset takes place in the dataset buffers in user memory and the contents of the buffers are not ordinarily written to mass storage. Such a dataset cannot be made permanent, nor may it be disposed to another mainframe, unless first copied to mass storage.

A user attempting to write to a memory-resident dataset must have write permission. However, as long as the buffer is not full, no actual write to mass storage ever occurs. Therefore, changes made to an existing dataset declared memory-resident are not reflected on the mass storage copy of the dataset (if one exists).

If at any time the system I/O routines are called to write to the dataset and the buffer appears full, the dataset ceases to be treated as memory-resident, the buffer is flushed to mass storage, and all memory-resident indicators for the dataset are cleared.

Magnetic tape, execute-only, and interactive datasets cannot be declared memory-resident.

Interactive Datasets

A dataset can be specified as interactive by an interactive job, provided that interactive datasets are supported by the front-end. Batch users cannot create interactive datasets.

An interactive dataset differs from a local dataset in that a disk image of the dataset is not maintained. Instead, records are transmitted to and from a terminal attached to a front-end station. Record positioning (such as Rewind or Backspace) is not possible.

Magnetic Tape Datasets

A magnetic tape dataset is available to any job declaring tape resource requirements on the JOB statement and specifying the appropriate information on its access request.

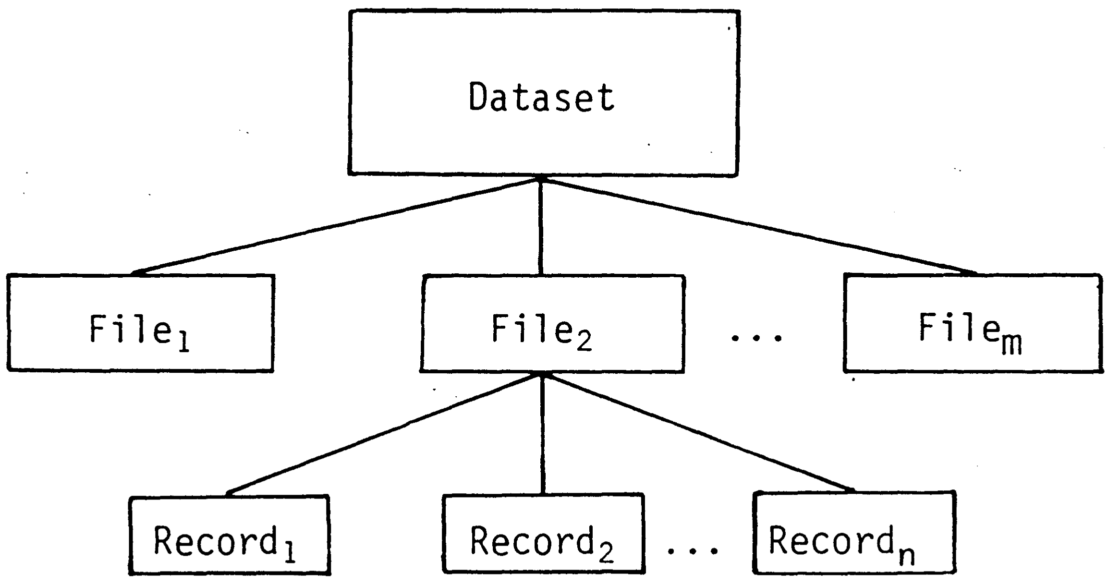
A magnetic tape can be unlabeled (NL), ANSI standard labeled (AL), or IBM standard labeled (SL), and can be recorded or read at either 1600 or 6250 bits per inch (bpi).

COS automatically switches volumes during dataset processing and returns to the first volume of a multivolume dataset in response to a REWIND command. If a permanent write error occurs when trying to write a tape block for the user, COS automatically attempts to close the current volume and continues to the next volume.

The COS tape system uses Buffer Memory as a tape block buffering area so that the job's I/O buffer need not be as large as the tape block. This technique can result in significant memory savings whenever large tape blocks are being processed and in increased transfer rates whenever smaller blocks are being processed. The advantage in having a large COS buffer is a reduction in the overhead in the tape subsystem.

With release 1.13, positioning support for tape datasets is possible. Users can position a tape dataset at any block on any volume, obtain the current position information for a tape dataset, and enable recovery of tape jobs after a system interruption.

Also, a MOD parameter has been added to the ACCESS control statement for use with on-line tapes. When MOD is specified on an access of a tape dataset, any data written to the dataset is appended to the data already contained in the dataset rather than being written from the beginning of the dataset.



Data hierarchy within a dataset

Dataset Structures

COS supports several dataset structures:

* Blocked Format
* Interactive Format
* Unblocked Format
* Tape formats (interchange or transparent)

Blocked Format

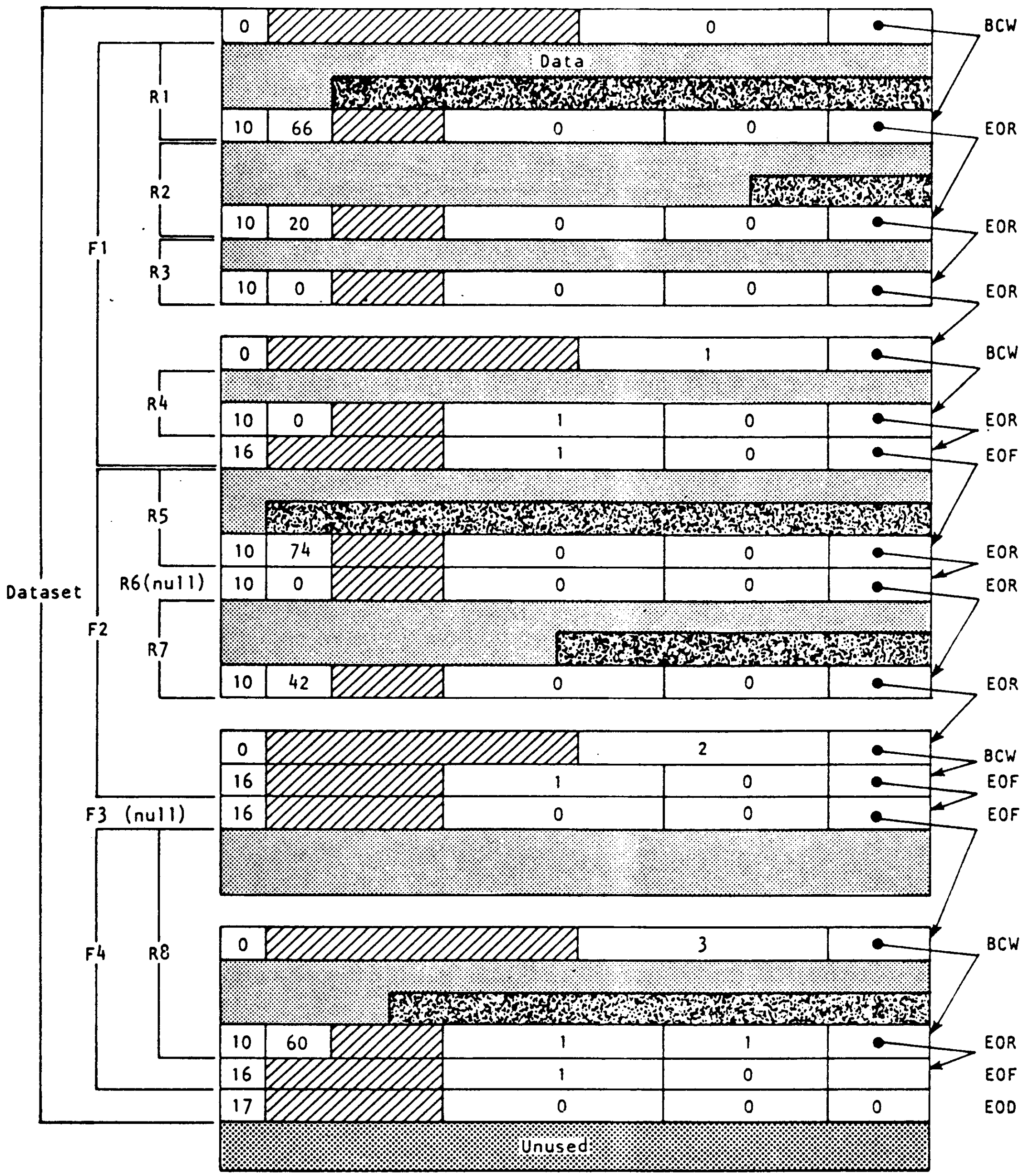
Blocked format is used by default for external types of datasets, such as user input and output datasets. Record positioning requires a blocked format. The blocked format adds control words to the data to allow for processing of variable-length records and to allow for delimiting of levels of data within a dataset.

A blocked dataset can be composed of one or more files, which are in turn, composed of one or more records.

The data in a blocked dataset can be coded and/or binary. Blanks are normally comoressed in block coded datasets. Each block consists of 512 words.

Blocked datasets use two types of control words:

* **block**
* **record.**



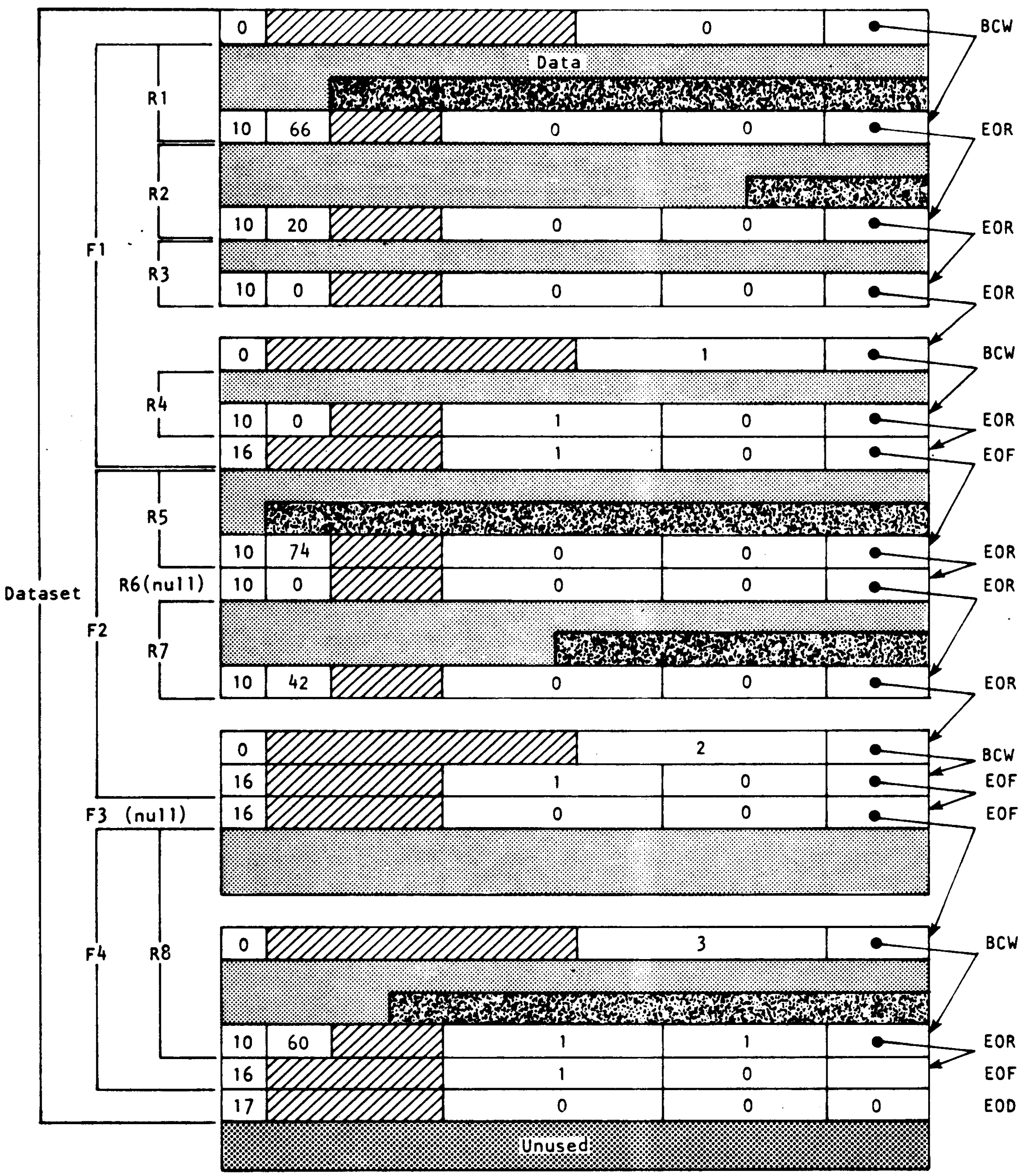
Example of dataset control words

Block Control Word

The block control word (BCW) is the first word of every 512 word block.

```
0        8        16       24       32       40       48       56     63
| M|/////////|  |///////////////////|            BN            |  FWI  |
              ^
               \BDF
```

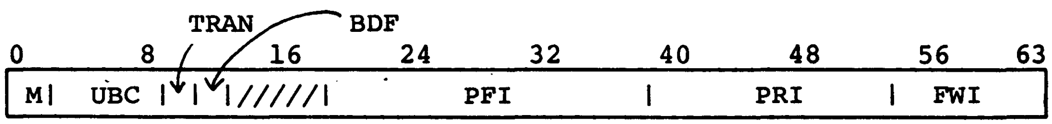
Field	Description
M	Type of control word (BCW=0)
BDF	Bad Data Flag. The following data, up to the next control word, is bad. (magtape datasets only)
BN	Block Number (first is 0)
FWI	Forward Index. The number of words (starting with 0) to the next record or block control word.



Dataset
F1
F2
F3 (null)
F4
R1
R2
R3
R4
R5
R6(null)
R7
R8
Data
Unused
BCW
EOR
EOF
EOD

Record Control Word

A record control word (RCW) occurs at the end of each record, file or dataset.



Field	Description
M	Type of control word: 10 = EOR 16 = EOF 17 = EOD
UBC	Unused Bit Count
TRAN	Transparent record field; used for interactive output dataset only
BDF	Bad Data Flag; the following data, up to the next control word is bad. Used for magtape datasets in interchange format.
PFI	Previous File Index. Index (in blocks) to the beginning of the file.
PRI	Previous Record Index. Index to the block where the current record starts.
FWI	Forward Word Index. Points to the next control word (number of data words up to the control word)

Interactive Format

Interactive format closely resembles blocked format; however, each buffer begins with a block 0 Block Control Word (BCW).

Each record transmitted in an interactive mode to or from COS must contain a single record consisting of a Block Control Word, data, and an end-of-record Record Control Word.

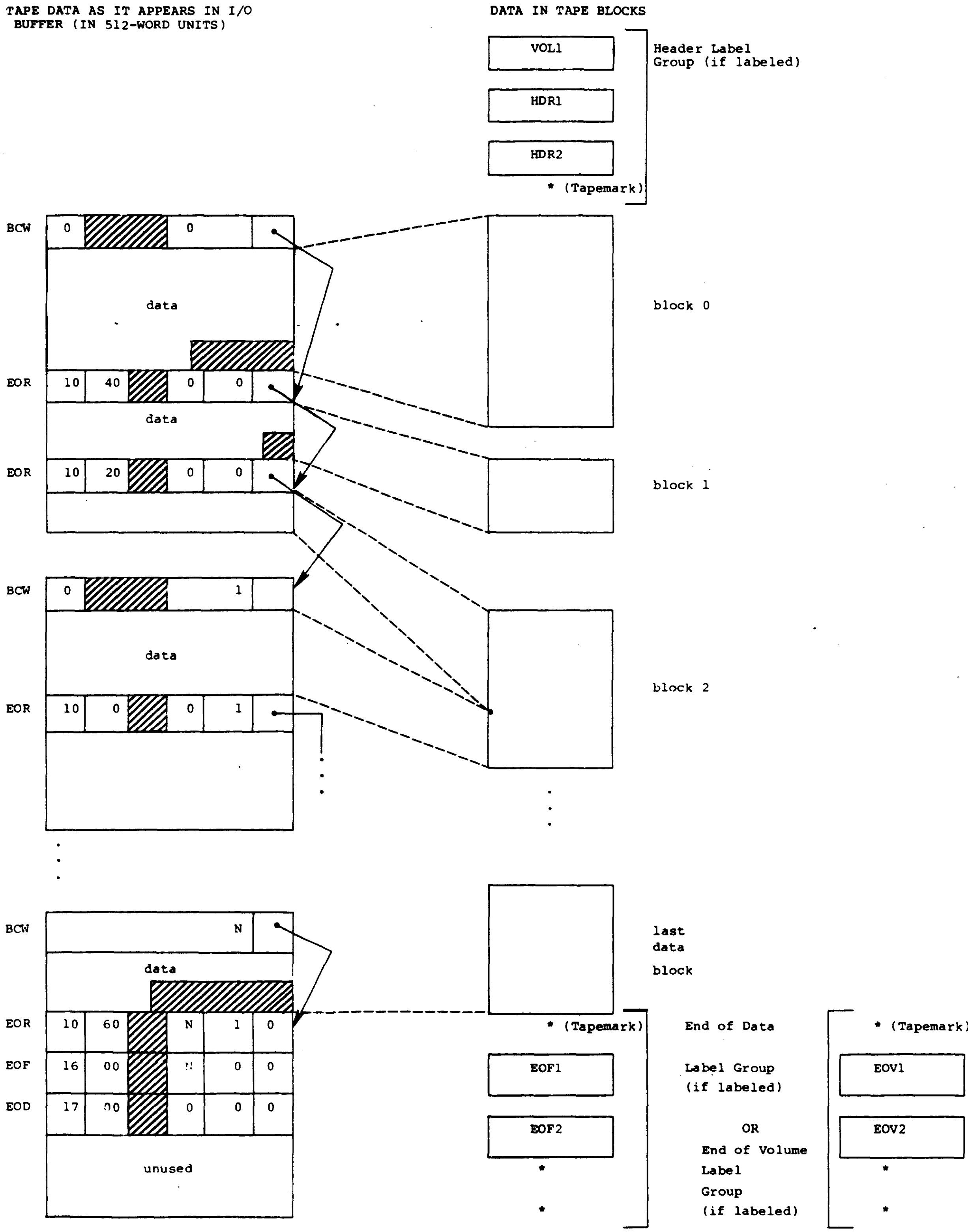
Two formats for interactive output can be assigned when the dataset is created: character blocked and transparent. Character blocked mode is the default. In this mode, an end-of-record RCW is interpreted as a line feed or carriage return. In transparent mode, the end-of-record RCW is ignored and the user must provide carriage control characters.

Unblocked Format

Dataset I/O can alos be performed using unblocked datasets. The data stream for unblocked datasets does not contain RCWs or BCWs.

The stream does not allocate buffers in the job's I/O buffer area for unblocked datasets; the user must specify an area for data transfer.

When a read or write is performed on an unblocked dataset, the data goes directly to or from the user data area without passing through an I/O buffer. The word count of data to be transferred must be in multiples of 512.



Interchange-format tape dataset
(octal values shown)

Tape Formats

Tape datasets can be read or written using two different formats:

* Interchange
* Transparent

Interchange Format

Interchange format enables reading and writing of tapes that are also to be read and written on other vendor's systems.

In interchange format, each tape block of data corresponds to a single logical record in COS blocked format (that is, the data between record control words).

In interchange format, tape blocks lengths can vary up to an installation-defined maximum which cannot exceed 1,048,576 bytes (131,072 64-bit words). It is recommended that the maximum block size not exceed 100 to 200 Kilobytes. Blocks exceeding these sizes may require special operational procedures (such as the use of specially prepared tape volumes having an extended length of tape following the end-of-tape (EOT) reflective marker) and yield little increase in transfer rates or storage capacity.

When a dataset is read in interchange mode, physical tape blocks are represented in the user's I/O buffer with block control words (BCWs) and record control words (RCWs) added by COS. The data in each tape block is terminated by an RCW. The unused bit count field in the RCW indicates the amount of data in the last word of the tape block that is not valid data. A BCW is inserted before every 511 words of data, including the RCWs. The format of RCWs and BCWs are described previously in this lesson.

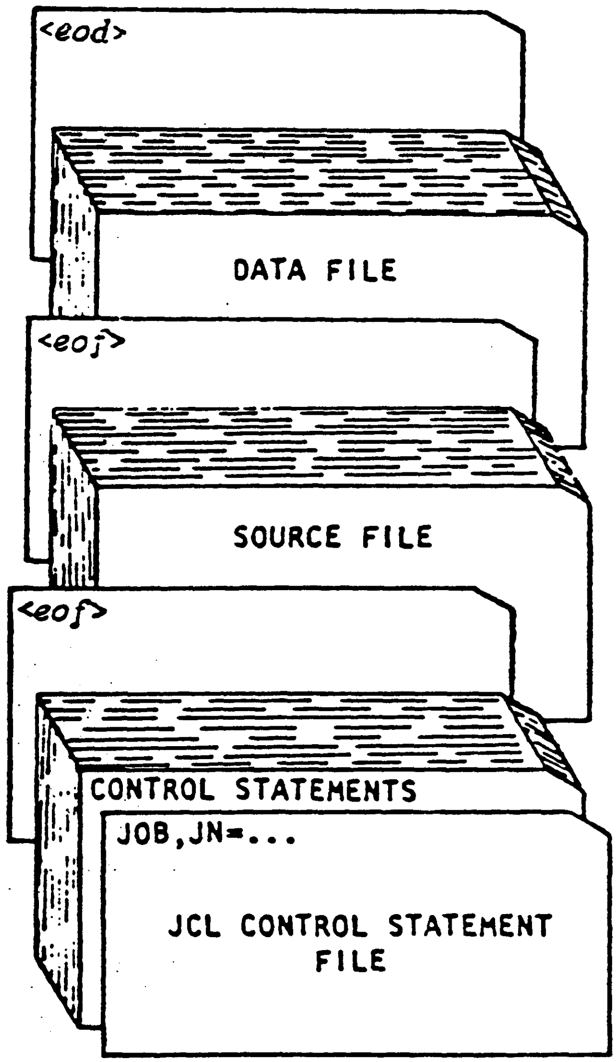
Interchange Format (continued)

When a tape dataset is written in interchange format, the data must be in the I/O buffer in the user field in COS blocked format. The data in each logical record is written as a single tape block. BCWs and RCWs are not recorded on tape. BCWs within a record are discarded and the unused bits and terminating RCW are also discarded. The unused bit count must be a multiple of 8. Tape datasets written in interchange mode must consist of a single file (single EOF RCW). Multiple-file tape datasets are not supported in interchange mode.

Transparent Format

In **transparent format** (disk image), each tape block is a fixed multiple of 4096 bytes (512 words), generally based on the dataset density (that is, 16,384 bytes at 1600 bpi and 32,768 bytes at 6250 bpi). The data in the tape block is transferred unaltered between the tape and the I/O buffer in the user field; no control words are added on reading or discarded on writing.

In transparent mode, the data can be in COS blocked format or COS unblocked format. Transparent format tapes are not generally read or written by other vendor's equipment.


<eod>
DATA FILE
<eof>
SOURCE FILE
<eof>
CONTROL STATEMENTS
JOB,JN=...
JCL CONTROL STATEMENT FILE

LESSON 6: Job Processing Overview

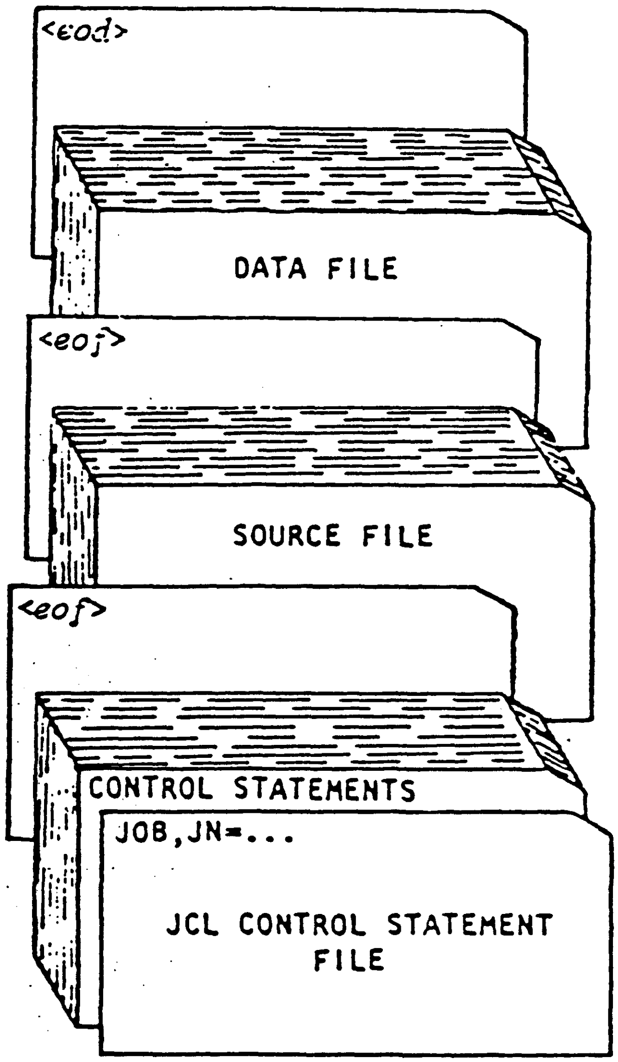
Objective: Trace a user job through the system, beginning with job preparation at the front-end processor, and terminating at its origin after being processed by the Cray.

JOB STRUCTURE

A **Job** is a unit of work submitted to the Cray computer system. It consists of one or more files of card images contained in a job deck dataset. Each job passes through several stages from job entry through job termination.

The job consists of:

* **a Control Statement File**
* **a Source File**
* **and a Data File**


<eod>
DATA FILE
<eof>
SOURCE FILE
<eof>
CONTROL STATEMENTS
JOB,JN=...
JCL CONTROL STATEMENT FILE

JOB STRUCTURE (continued)

The first (or only) file of the job deck must contain the job control language (JCL) control statements that specify the job processing requirements.

Each job begins with a **JOB** statement, identifying the job to the system.

If accounting is mandatory in the user's system, the **ACCOUNT** statement must immediately follow the JOB statement. All other control statements follow the JOB statement.

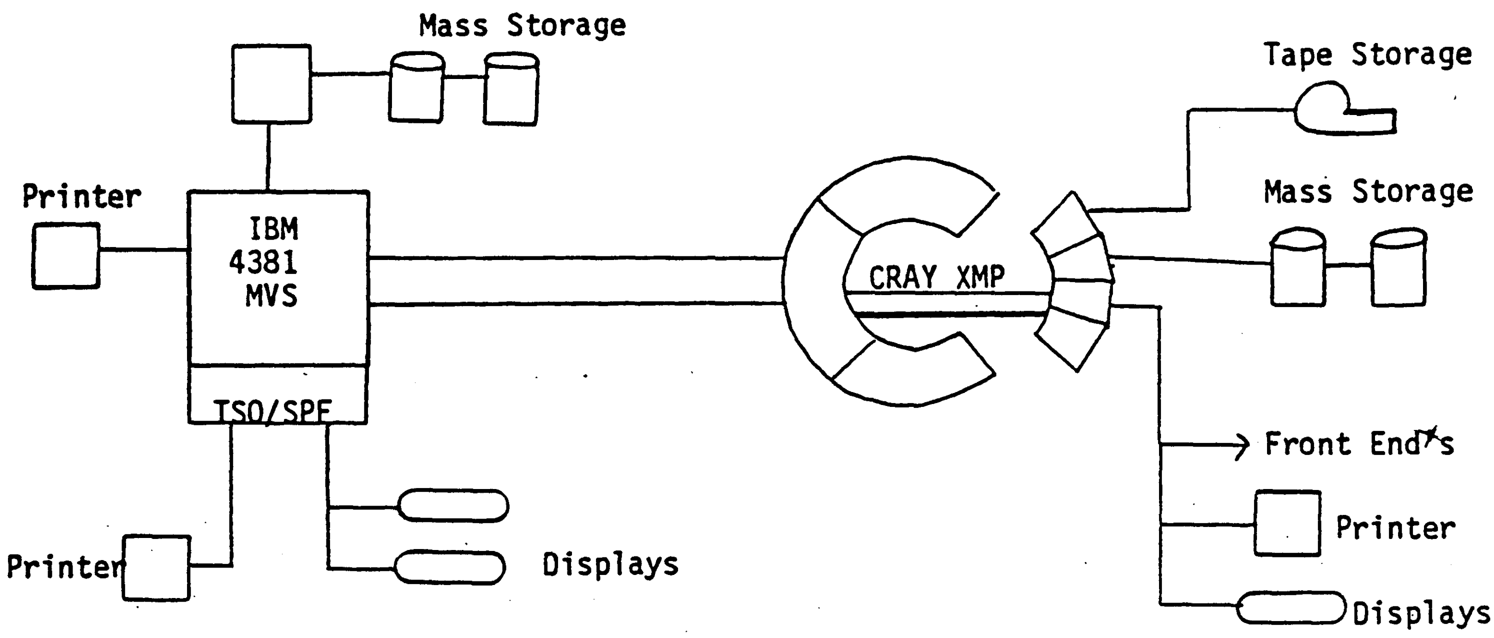
The end of the control statement file is designated by an end-of-file record (or an end-of-data record if the job consists of a control statement file only).

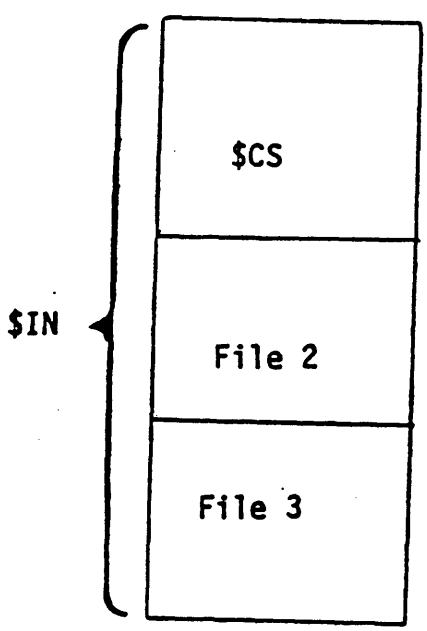
Files following the control statement file can contain source code or data. These files are handled according to instructions given in the control statement file.

The final card in a job deck must be an **end-of-data**.

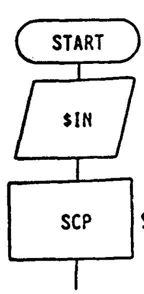
```
JOB,JN=jn,MFL=fl,T=tl,P=p,US=us, OLM=lm,CL=jcn,*gn=nr
```

```
ACCOUNT,AC=ac,PW=pw,NPW=npw,US=us,UPW+upw,NUPW=nupw
```





JOB FLOW



FILE CREATED AT TERMINAL CONNECTED TO F.E.

SUB/B FILENAME - JOB SUBMITTED TO CRAY FOR ASSEMBLY AND EXECUTION

STATION CALL PROCESSOR, STP TASK, MAKES JOB KNOWN TO SYSTEM BY MAKING AN ENTRY IN THE INPUT QUEUE OF THE SYSTEM DATASET TABLE (SDT). CALLS JOB SCHEDULER.

JOB FLOW

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

* **Entry**
* **Initiation**
* **Advancement**
* **Termination**

JOB ENTRY

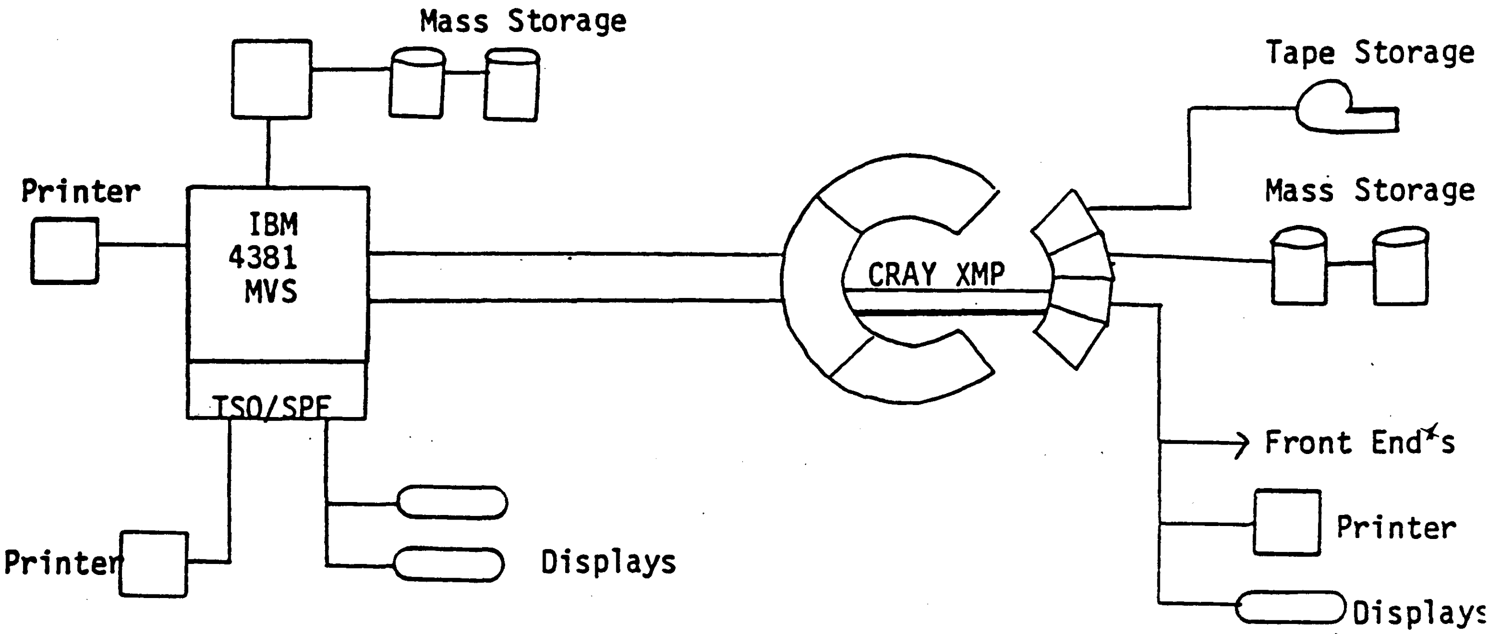
A job can enter the system in the form of a dataset submitted from a front-end computer system or a local or remote job entry station.

The Station Call Processor task (SCP) in STP is responsible for making the job's existance known to the system.

It does this by creating an entry in the System Dataset Table (SDT) (in the STP Table area of memory), creating a memory pool entry, and requesting that an entry be made in the Dataset Catalog (DSC) on the master disk, thereby making the dataset permanent.

The job resides on the disk until it is scheduled to begin processing.

The Station Call Processor (SCP) now readies the Job Scheduler Task (JSH), in effect, calling attention to the new job in the system.



JOB FLOW

START — FILE CREATED AT TERMINAL CONNECTED TO F.E.

$IN — SUB/B FILENAME - JOB SUBMITTED TO CRAY FOR ASSEMBLY AND EXECUTION

SCP — STATION CALL PROCESSOR, STP TASK, MAKES JOB KNOWN TO SYSTEM BY MAKING AN ENTRY IN THE INPUT QUEUE OF THE SYSTEM DATASET TABLE (SDT). CALLS JOB SCHEDULER.

JSH — SELECTS JOB FROM INPUT QUEUE TO PLACE ON JOB EXECUTION TABLE (JXT)

JSH — SELECTS JOB FOR EXECUTION; NOTIFIES EXEC
ALLOCATES MEMORY FOR JOB
SETS THE JTA
INITIATES JOB

JOB INITIATION

The Job Scheduler Task (JSH) scans the System Dataset Table 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) (in the STP Table area of memory),

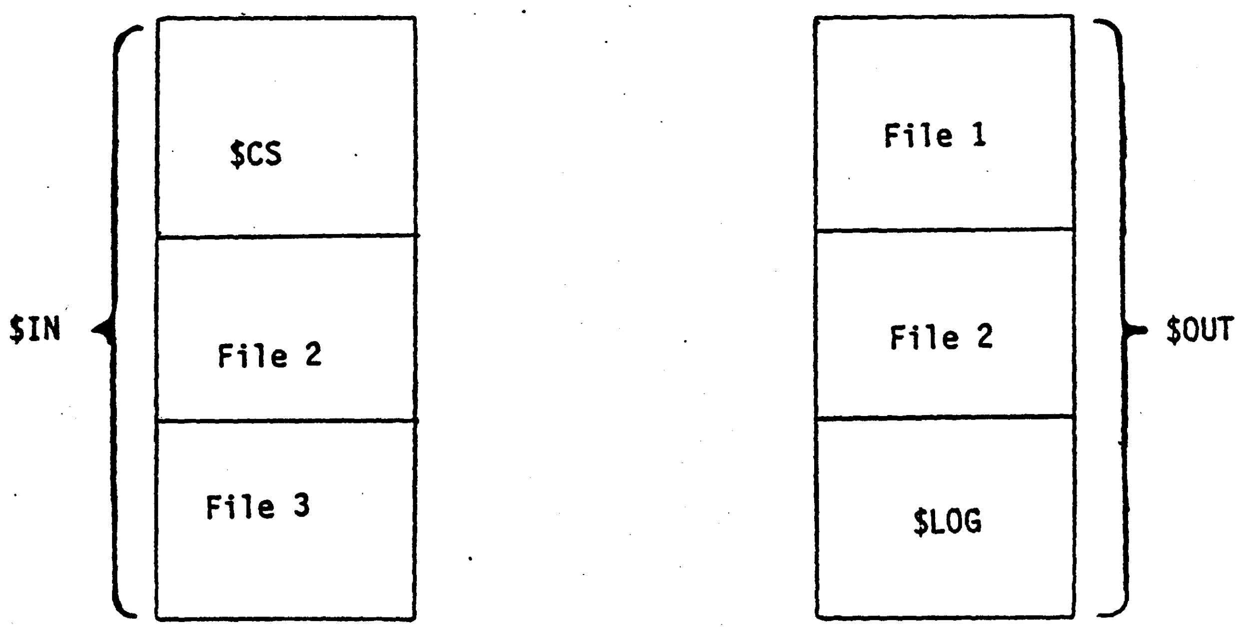
* No other job in the same class of higher priority is waiting to begin processing, and

* The requested generic resources (i.e. tape devices) are available.

The Job Scheduler Task uses an available entry in the Job Execution Table (JXT) to create an entry for the job being initiated, and prepares a Job Table Area (JTA) and user field. 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.

The Job Scheduler (JSH) also moves the job's System Dataset Table entry from the input queue to the executing queue, still in the System Dataset Table.

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


$IN
$CS
File 2
File 3
File 1
File 2
$LOG
$OUT

When COS schedules the job for processing, it creates four datasets:

* $CS
* $IN
* $OUT
* $LOG

$CS

$CS is a copy of the job's control statement file from the input dataset and is used only by the system; the user cannot access $CS by name. This dataset is used to read the job control statements.

$IN

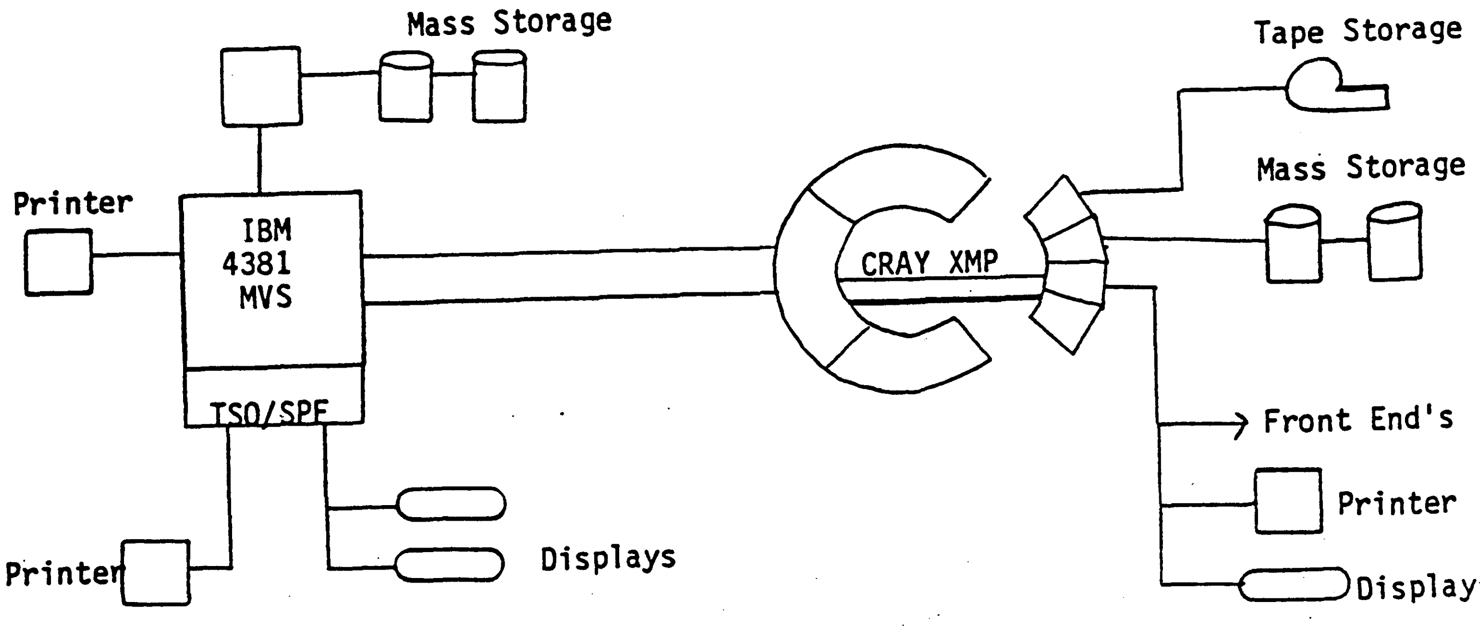
This is the job input dataset. The job itself can access the input dataset, with read-only permission, by its local name, $IN, or as FORTRAN unit 5. The disposition code for $CS is SC (Scratch).

$OUT

This is the job output dataset. The job can access this dataset by name, $OUT, or as FORTRAN unit 6. The disposition code for $OUT is PR (print).

$LOG

The job's logfile contains a history of the job. This dataset is known only to COS and is not accessable to the user. (User messages can be added to the logfile however, using the Message system action request macro or other user Remark subroutines.)



JOB FLOW

START	FILE CREATED AT TERMINAL CONNECTED TO F.E.
$IN	SUB/B FILENAME - JOB SUBMITTED TO CRAY FOR ASSEMBLY AND EXECUTION
SCP	STATION CALL PROCESSOR, STP TASK, MAKES JOB KNOWN TO SYSTEM BY MAKING AN ENTRY IN THE INPUT QUEUE OF THE SYSTEM DATASET TABLE (SDT). CALLS JOB SCHEDULER.
JSH	SELECTS JOB FROM INPUT QUEUE TO PLACE ON JOB EXECUTION TABLE (JXT)
JSH	SELECTS JOB FOR EXECUTION; NOTIFIES EXEC ALLOCATES MEMORY FOR JOB SETS THE JTA INITIATES JOB
EXP	EXCHANGE PROCESSOR COPIES CSP INTO USER FIELD
CSP	PROCESSES CONTROL STATEMENTS ADVANCES JOB CREATES FOUR DATASETS $CS - CONTROL STATEMENT FILE $IN - SOURCE FILE $OUT - JOB LISTING $LOG - PROGRAM INFORMATION

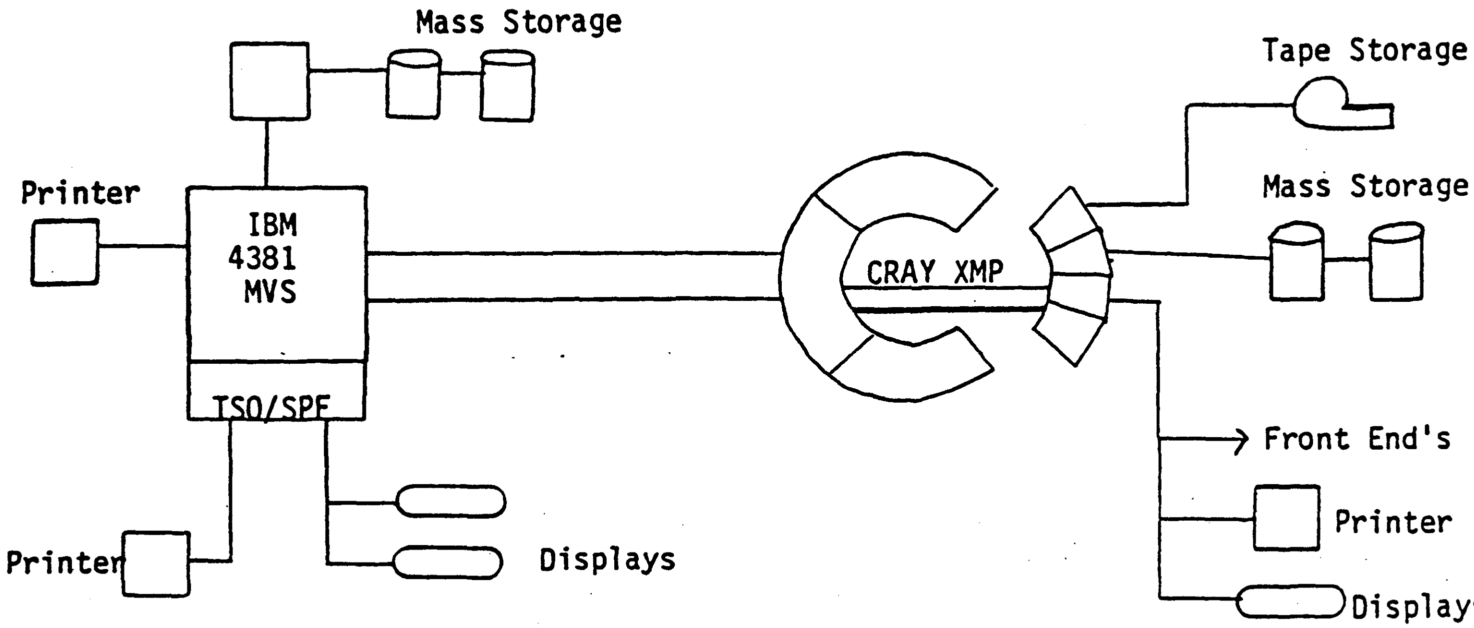
JOB ADVANCEMENT

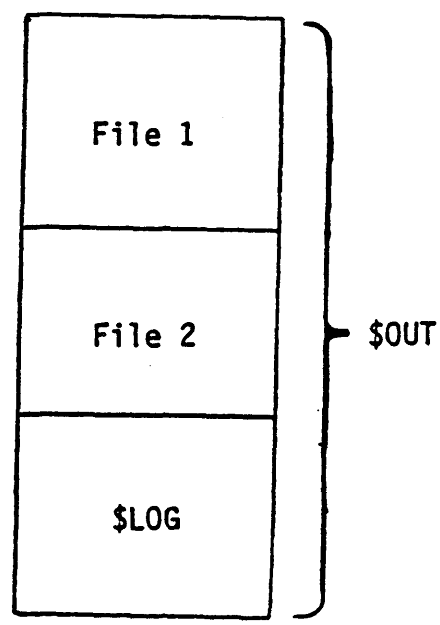
Job advancement is the processing of a job according to the instructions in the control statement file. The Control Statement Processor (CSP) advances a job through its program steps. CSP is first loaded and executed in the user field following job initiation.

A normal advance causes CSP to interpret the next control statement in the job's control statement file. An abort advance occurs if COS detects an error or if the user requests that the job abort.

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 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 Job Execution Table (JXT) are in memory. Some are rolled out to mass storage when an event occurs causing other jobs to replace them in memory.


Mass Storage
Tape Storage
Printer
IBM
4381
MVS
Mass Storage
CRAY XMP
TSO/SPF
Front End's
Printer
Printer
Displays
Displays


File 1
File 2
$LOG
$OUT

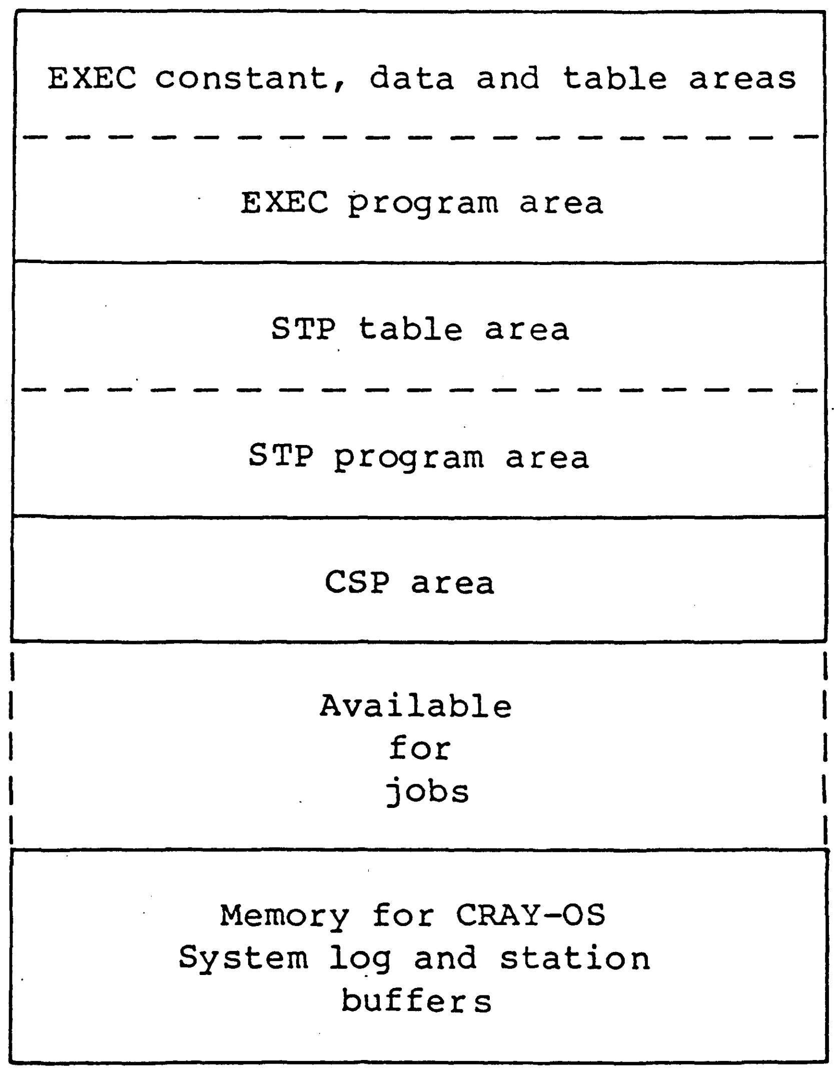
JOB TERMINATION

Output from the job is placed on system mass storage. At completion of the job, COS appends $LOG to $OUT and returns $OUT to its originating station. $IN, $CS, and $LOG are released. $OUT is renamed jn (from the JN parameter value of the JOB control statement and is directed to the output queue for staging to the specified front-end computer system. When the front-end has received the entire contents of $OUT, the output dataset is deleted from COS mass storage.

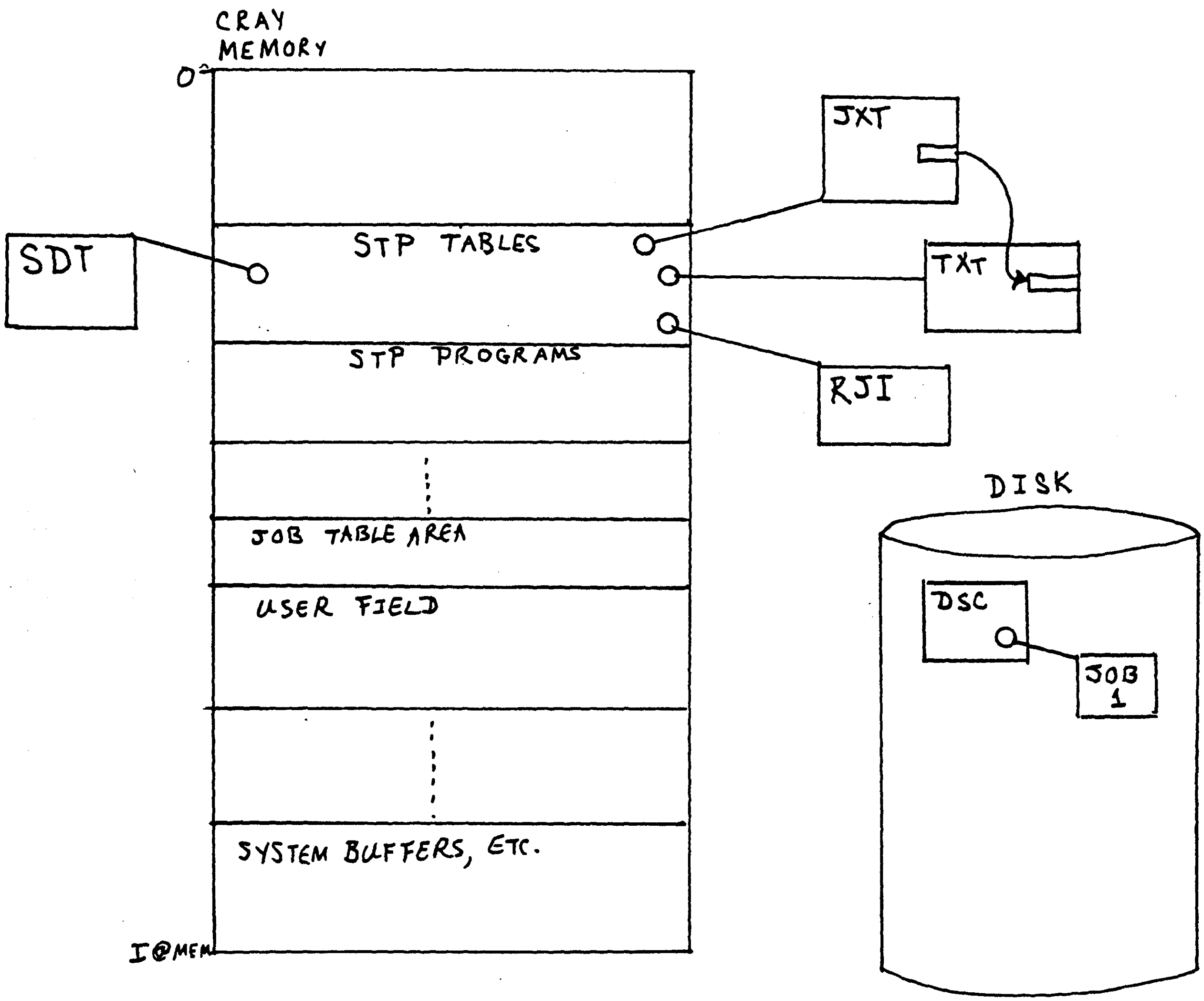
The front-end computer processes $OUT as specified by the dataset disposition code. If, for any reason, $OUT does not exist, $LOG is the only output returned at job termination.

In summary, when a job terminates, the following actions occur:

* A Dataset Catalog (DSC) entry (on the master device) is created for each of the job's output datasets.
* A System Dataset Table (SDT) entry is created (in memory) for each of the job's output datasets.
* The user logfile, $LOG, is copied onto the end of $OUT.
* The Dataset Catalog entry for the input dataset is deleted.
* The job's System Dataset Table (SDT) entry is deleted from the executing queue.
* The Job Execution Table (JXT) entry and Task Execution Table (TXT) entry, and the memory assigned to the job are released.
* The Rolled Job Index entry is cleared.
* The Station Call Processor (SCP) task is readied at the next interrupt from a front-end and scans the System Dataset Table (SDT) for output to send to the front-end system.
* SCP deletes the corresponding Dataset Catalog and System Dataset Table entries after each output is successfully transmitted to the front-end system.


EXEC constant, data and table areas
EXEC program area
STP table area
STP program area
CSP area
Available for jobs
Memory for CRAY-OS System log and station buffers

SOME TABLES RELATED TO JOB PROCESSING



LESSON 7: Memory Management

Objective: Describe the ways in which memory is managed for user and system requirements.

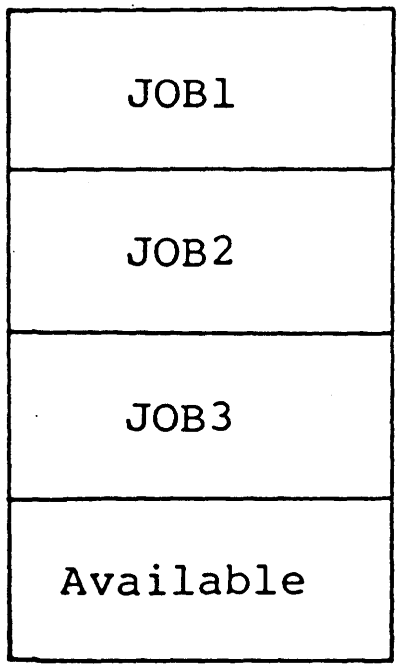
* NOTE: The job of managing memory is accomplished by the Job Scheduler (JSH) system task. Details of this process are discussed in Lesson 14: "System Tasks" Our purpose here is simply to provide an overview of the process.

INTRODUCTION

Central memory is a resource that is allocated to jobs by the operating system. A job's memory is composed of several distinct areas. Some of these are managed exclusively by the system for the user; others are managed by both the system and the user.

Memory is allocated to the system at Startup at both the low-address and high-address ends of memory. After all system components (tasks) have been initialized, the remaining 512-word blocks of memory are allocated for future jobs or for system buffers.

The total job size equals the length of the job's Job Table Area (JTA) plus user field length. The lined area between JCHLM and JCLFT is unused space within the job. This area contains enough memory to guarantee that the job size is always a multiple of 512 words.



7.2

INITIAL MEMORY ALLOCATION

Segments of memory are allocated to jobs using a "first fit" method, that is, the job is allocated memory from the first (lowest addressed) segment large enough to contain it. (Segments are allocated in multiples of 512 words). The last segment is always allocated to the system.

Jobs that are waiting for memory are jobs that are either already in memory and need to expand, or they are not in memory and need to be brought in. Whe allocation is possible, COS looks to see if a job that is waiting memory can be given memory. Jobs that are waiting are scanned in descending priority order.

The system gets priority over jobs for memory. When a system request is made for memory, its requirements are considered first.

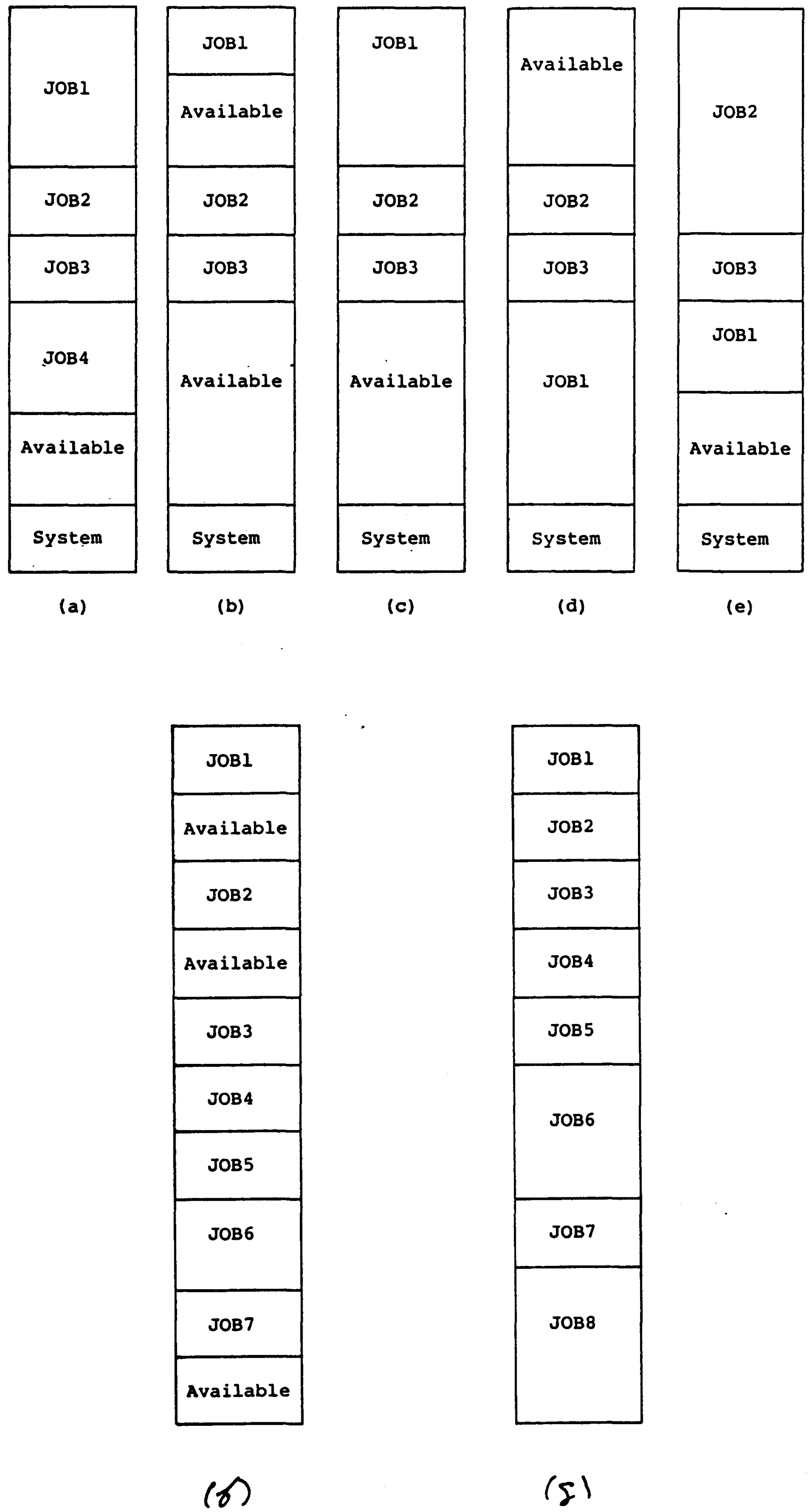
A tally is kept of the total amount of memory that will be available when all currently scheduled rolls complete. If this tally indicates that there is enough free memory to satisfy the system request, the system will be given the memory. If there is not enough memory available, any jobs that are either suspended or of lower priority will be rolled out if rolling them out frees up enough memory for the system request to be satified.

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 the 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 some room for expansion.

When the job initiates it is given sufficient memory for the Control Statement Processor (CSP) to execute. Once the JOB statement is processed, the job is allowed a field length no larger than the amount specified by the MFL parameter on the JOB control statement.


JOB1
JOB2
JOB3
JOB4
Available
System
(a)
JOB1
Available
JOB2
JOB3
Available
System
(b)
JOB1
JOB2
JOB3
Available
System
(c)
Available
JOB2
JOB3
JOB1
System
(d)
JOB2
JOB3
JOB1
Available
System
(e)
JOB1
Available
JOB2
Available
JOB3
JOB4
JOB5
JOB6
JOB7
Available
(f)
JOB1
JOB2
JOB3
JOB4
JOB5
JOB6
JOB7
JOB8
(g)

7.4

Expansion Space (continued)

The first illustration on the facing page 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. Expansion spaced is required to allow the jobs that are already in memory to expand.

Allocating, Deallocating, and Compacting Memory

Figure (a) shows memory before any change.

Figure (b) shows memory after JOB4 terminates and JOB1 decreases its field length. The freed memory is marked available. Contiguous memory segments are merged into one larger available segment but no memory compaction is done.

Figure (c) shows memory after JOB1 increases its field length. A job is expanded in place whenever possible.

Figure (d) 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 (e) 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 avilable.

Figure (f) shows memory before any change,

Figure (g) shows memory after memory is compacted and JOB8 is rolled in.

Modes of Field Length Reduction

There are two modes of field length reduction: automatic and user managed. A user can manage the field length of the job by requesting a specific field length by using a MEMORY control statement in the JCL.

* Automatic

  When the job is in automatic field length reduction mode, the system automatically increases and decreases the job's field length as the areas within the job increase and decrease. A job initiates in automatic field length reduction mode.

* User-Managed

  When the job is in user-managed field length reduction mode, the system continues to increase the job's field length as before, but never automatically decreases it. The job's field length can be decreased only by the user until the job is returned to automatic field reduction mode.

The field length can be reduced at the beginning of each job step and during each job step if the job is in automatic field length reduction mode and any area of the job decreases.

Since increases in field length can result in the job's requiring more memory than can be immediately provided, which causes the job to be delayed until sufficient memory can be given to it, the user may want to manage the job's field length when it is known that the job will undergo frequent short-lived fluctuations in size.

User Management of Memory

A user can dynamically manage the user area of the job by requesting an increase or decrease of memory at the end of the user code/data area, or by requesting a specific field length.

Management by Control Statement from the Run Stream

A user can use the MEMORY control statement to manage the job's field length. When the user manages the job's field length, the job will be placed in user-managed field length reduction mode for the duration of the next job step. The MEMORY control statement may also place the job in user-managed field length reduction mode across job steps or return the job to automatic mode.

Management from within a Program

From within a program, use of the MEMORY macro or MEMORY routine, respectively, requests user management of the job's user code/data area and field length. When the user manages the job's field length, the job is placed in user-managed field length reduction mode for the duration of the job step. The MEMORY macro or MEMORY routine may also place the job in user-managed field length reduction mode across job steps or return the job to automatic mode.

Management Associated with a Program

Use of certain parameters on the LDR control statement causes memory management to be associated with the binary being loaded.

This association is stored with the binary if the binary is saved on a dataset. The management can be user code/data area management or field length management and occurs when the binary is loaded for execution. If the field length is being managed, the job is placed in user-managed field length reduction mode for the duration of the program execution.

System Management of Memory

The system changes appropriate areas of the job's memory when a job initiates certain system actions such as advancing to the next job step, performing I/O etc.

The Job Table Area (JTA), Logical File Tables, and Dataset Parameter Area can increase, but will never decrease.

The user code/data and buffer areas may both increase and decrease in size. If the job is in automatic field length reduction mode, the system automatically increases and decreases the job's field length when any area in the job increases or decreases.

If the job is in user-managed field length reduction mode, the system continues to increase the field length when it needs to, but never automatically decreases the field length.

LESSON 8: Tasks and Multitasking

Objective: Define the various modes of operation used by the Cray computer system, and the units of computation and processes.

INTRODUCTION

Various segments of the computer industry utilize terminology which may differ in meaning and context from segment to segment and company to company. Since the Cray computer is capable of multiprogramming, multiprocessing, and multitasking, a clarification of these terms as they are used in the Cray environment is in order.

PARALLELISM

"Parallel" refers to the manner in which software processes are executed. Jobs, job steps, programs, and parts of programs are parallel if they are processed simultaneously (or nearly so) rather than sequentially.

Levels of parallelism are defined in terms of the types of software processes that are executed in parallel.

- Level 1: Independent jobs, each job having a CPU
- Level 2: Job steps: related parts of the same job
- Level 3: Routines and subroutines
- Level 4: Loops
- Level 5: Statements

The higher the number of the level, the smaller the size or granularity of tasks.

MULTIPROGRAMMING

Multiprogramming is a mode of operation that provides for 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 iummediately 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.

For example, COS 1.11 is a multiprogramming operating system. The processor resource is one CPU, and the software processes are jobs. Sharing is managed by assigning priorities to jobs and allocating CPU time a slice at a time to different jobs.

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.

Under COS 1.12, two independent jobs can be run in parallel on a Cray X-MP computer system. This is sometimes referred to as the processors running separate job streams. The job is the scheduling unit of the system, and two processors are scheduled in a multi-programming mode.

Truly independent jobs won't affect each other, but two jobs using the same dataset can interfere with each other and thus are not independent.

This example of independent "uniprocessing" exploits parallelism at level 1 (independent jobs, each with a CPU). System throughput is enhanced over single processor configurations, but individual jobs receive no real processing benefits.

Applications of more than one processor to a single job implies that the job has software processes (parts) that can be executed in parallel. Such a job can be logically or functionally divided in such a way that two or more parts of the work can be executed simultaneously (that is, in parallel).

An example of this could be a weather modeling job where the northern hemisphere calculation is one part of the job and the southern hemisphere another part. Distinct code segments need not be involved. The same code could run on multiple processors at the same time, with each processor acting on different data.

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.

In a single processor multiprogramming operating system such as COS 1.11, a job is a task. In a multiprocessing environment supported by COS 1.13, a job is still a task, but it may spin off other tasks to run in parallel with it.

To take advantage of a multiprocessing operating system, a job must be divided into two or more tasks. That is, for parts of the job to run in parallel on more than one processor, the parts must be scheduled separately.

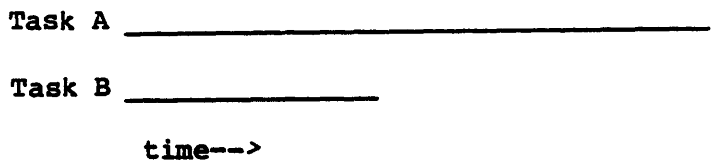
User Library Task

A user library task is a uniquely named process that can have code and data areas in common (or even identical to) other tasks of the same job. The code executed by a user library task is a subroutine. The same work can be performed by calling the subroutine or by starting up a task to execute the subroutine. The difference is that the call causes the work to be performed immediately; in the task, the work is cheduled and performed independently and in parallel with other tasks in the program.

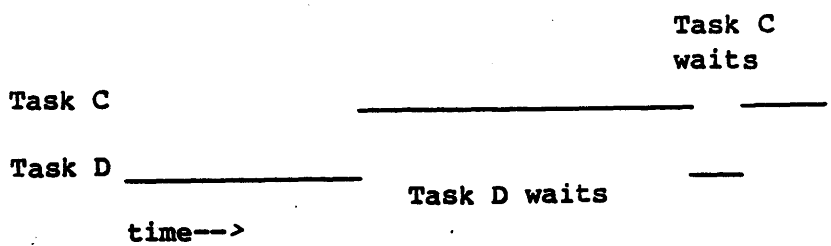
The multitasking library scheduler schedules user library tasks. It creates, deletes, activates, and deactivates user tasks as required.

System Task

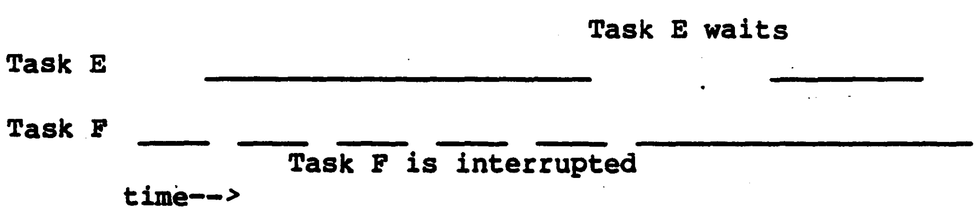
The system tasks are those tasks which make up the System Task Processor (STP). The function of each system task is described in Lesson 12 of this unit.


Task A
Task B
time-->

A


Task C
waits
Task C
Task D
Task D waits
time-->

B


Task E waits
Task E
Task F
Task F is interrupted
time-->

C

MULTITASKING

Multitasking is a special case of multiprocessing defining a "task" to be a job step or subprogram.

Version 1.13 of COS provides for multitasking within job steps. As always, job steps are executed sequentially. Using the library subroutines, a program executing in a job step can create additional tasks, thus bringing about multitasking. A multitasked job is not complete until all tasks within the job step complete.

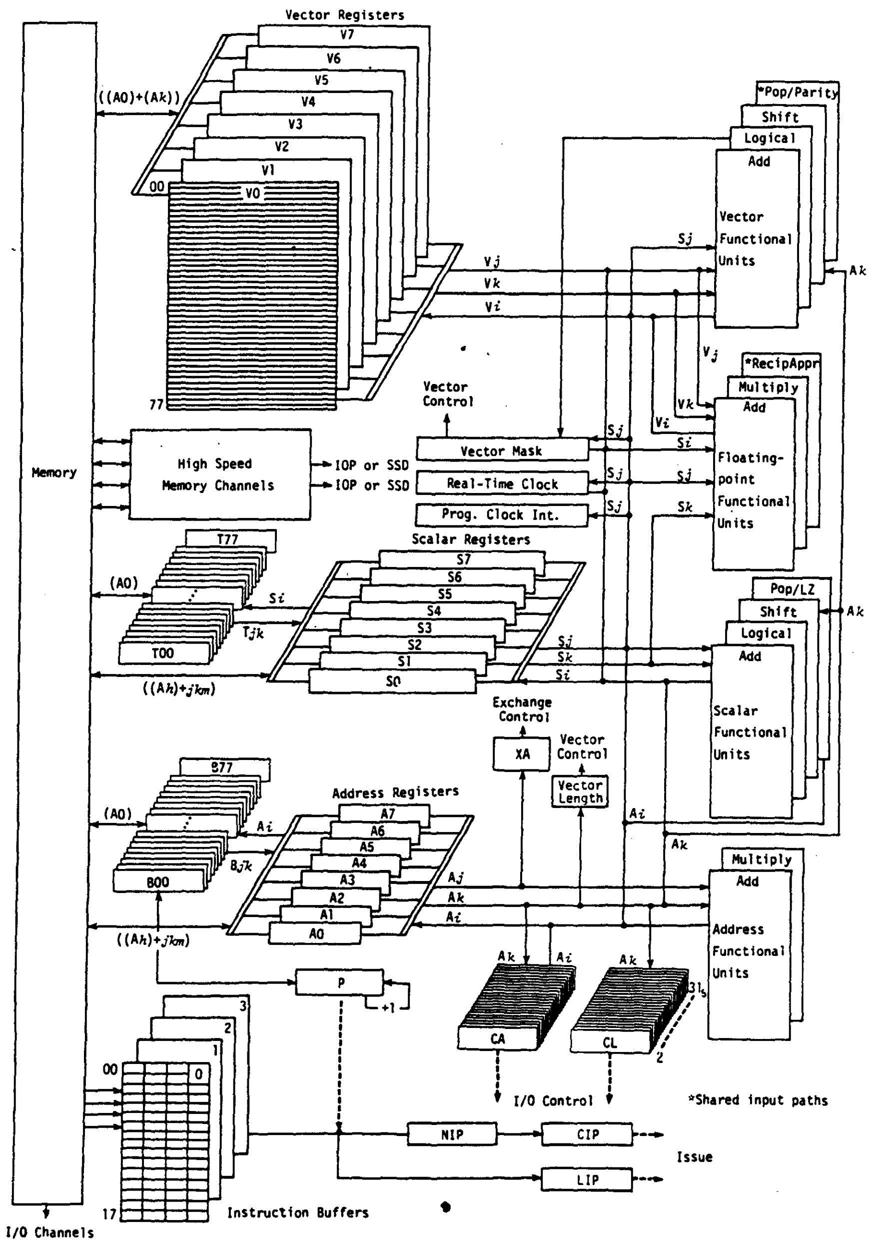
In a multitasking environment, the tasks and data structure of a job must be such that the tasks can run in parallel. However, the availability of processors, and the order of execution and completion of tasks are functions of the scheduling policies of the library scheduler and COS. Consequently, multitasking is nondeterministic with respect to time.

Tasks, however, must be made deterministic with respect to results. The key to a successful multitasked program is to precisely define and add the necessary communication and synchronization mechanisms between parallel tasks and to provide for the protection of shared data.

Figure A is an example of a two tasks executing without interruption on two processors.

Figure B illustrates a case in which only one processor is available, and tasks C and D must share it. Multitasking can be performed on machines with one processor.

In **Figure C**, two tasks share two processors. Note that at several points, only one processor is actually in use by the job, and at one point, neither is assigned to the job. Note also that there is no indication of which physical processor is assigned to which task; this assignment is transparent to the user. However, in a multiprocessor environment, users can specify which CPU is to be assigned.


Vector Registers
V7
V6
V5
V4
V3
V2
V1
V0
00
77
((A0)+(Ak))
Memory
High Speed
Memory Channels
IOP or SSD
IOP or SSD
Vector Control
Vector Mask
Real-Time Clock
Prog. Clock Int.
T77
T00
(A0)
Si
Tjk
Scalar Registers
S7
S6
S5
S4
S3
S2
S1
S0
((Ah)+jkm)
Exchange Control
XA
Vector Control
Vector Length
B77
B00
Ai
Bjk
Address Registers
A7
A6
A5
A4
A3
A2
A1
A0
Aj
Ak
P
+1
CA
CL
I/O Control
3
2
1
0
17
Instruction Buffers
NIP
CIP
LIP
Issue
I/O Channels
*Pop/Parity
Shift
Logical
Add
Vector
Functional
Units
Vj
Vk
Vi
Sj
Sk
*RecipAppr
Multiply
Floating-
point
Functional
Units
Pop/LZ
Scalar
Functional
Units
Address
Functional
Units
*Shared input paths

LESSON 9: Exchange Mechanism

Objective: Describe the method used by the Exchange Mechanism in managing the execution of programs.

EXCHANGE MECHANISM

The technique employed by 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 are entered with the 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 or error exit conditions.

EXEC is always a partner in the exchange; that is, it is either the program relinquishing control or receiving control. All other programs must return control to EXEC.

```
        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                              |
       +---------------------------------------------------------------+
```

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

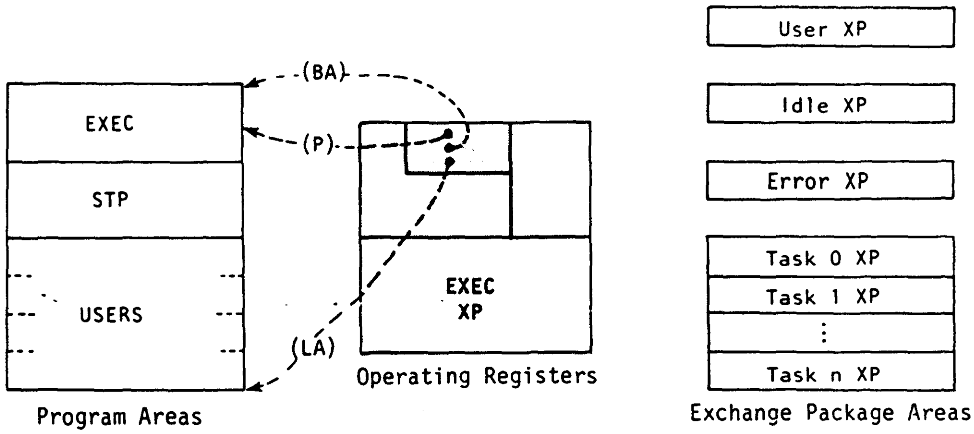
Exchange Package

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.

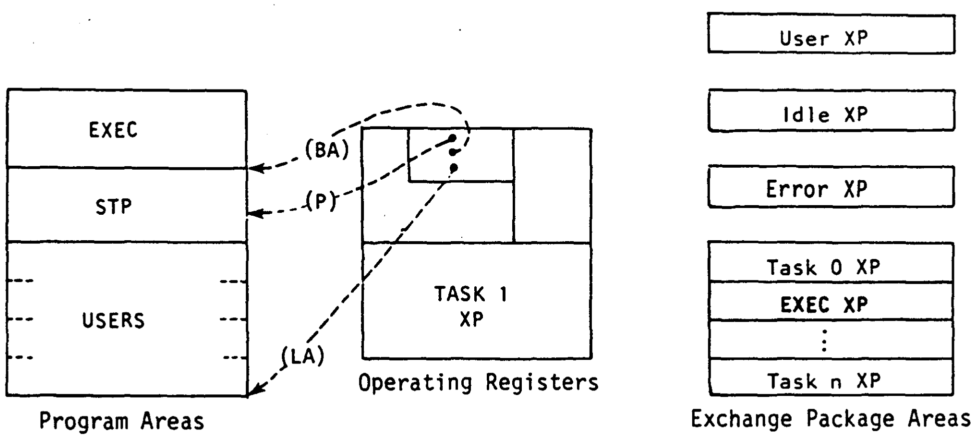
The Cray-1 Exchange Package is shown below. The Cray X-MP Exchange Package is shown on the facing page (9.2).

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

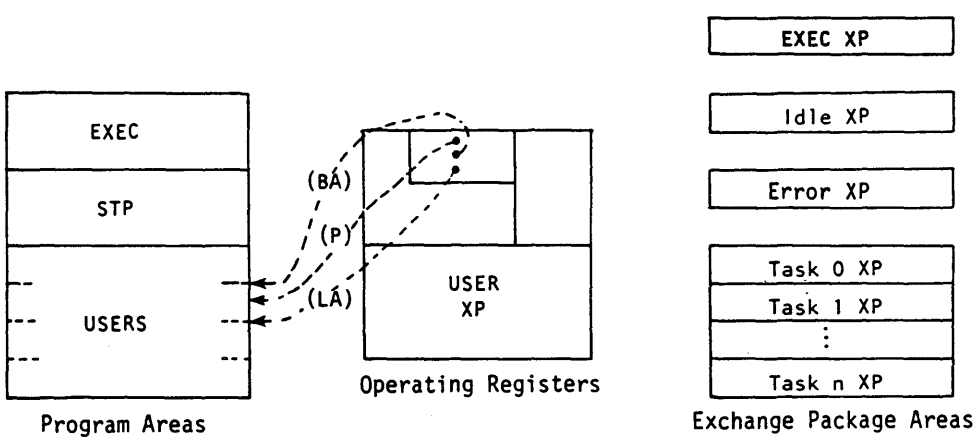
Field	Word	Bits
Error type (E)	0	0-1
Syndrome bits (S)	0	2-9
Read mode (R)	0	10-11
Bank error address (B)	0	12-15
Program register (P)	0	18-39
Chip error address (C)	1	0-15
Base address (BA)	1	18-35
Interrupt Monitor Mode bit (IMM)	1	39
High-order bits of memory error read address (RH)	2	14-15
Limit 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



A. EXEC IN EXECUTION



B. TASK 1 IN EXECUTION



C. CURRENT USER IN EXECUTION

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.

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.

* The active **user** exchange packages. One user exchange package per CPU resides in the Processor Working Storage (PWS) entry and is copied from the user's Job Table Area (JTA) when the job is connected to the CPU. The exchange package is then copied into the user's JTA when the job is disconnected from the CPU.

* The idle task exchange packages. One idle exchange package per CPU resides in the Processor Working Storage (PWS) 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 PWS and is selected when a memory parity error causes an exchange.

B, T, and V Registers

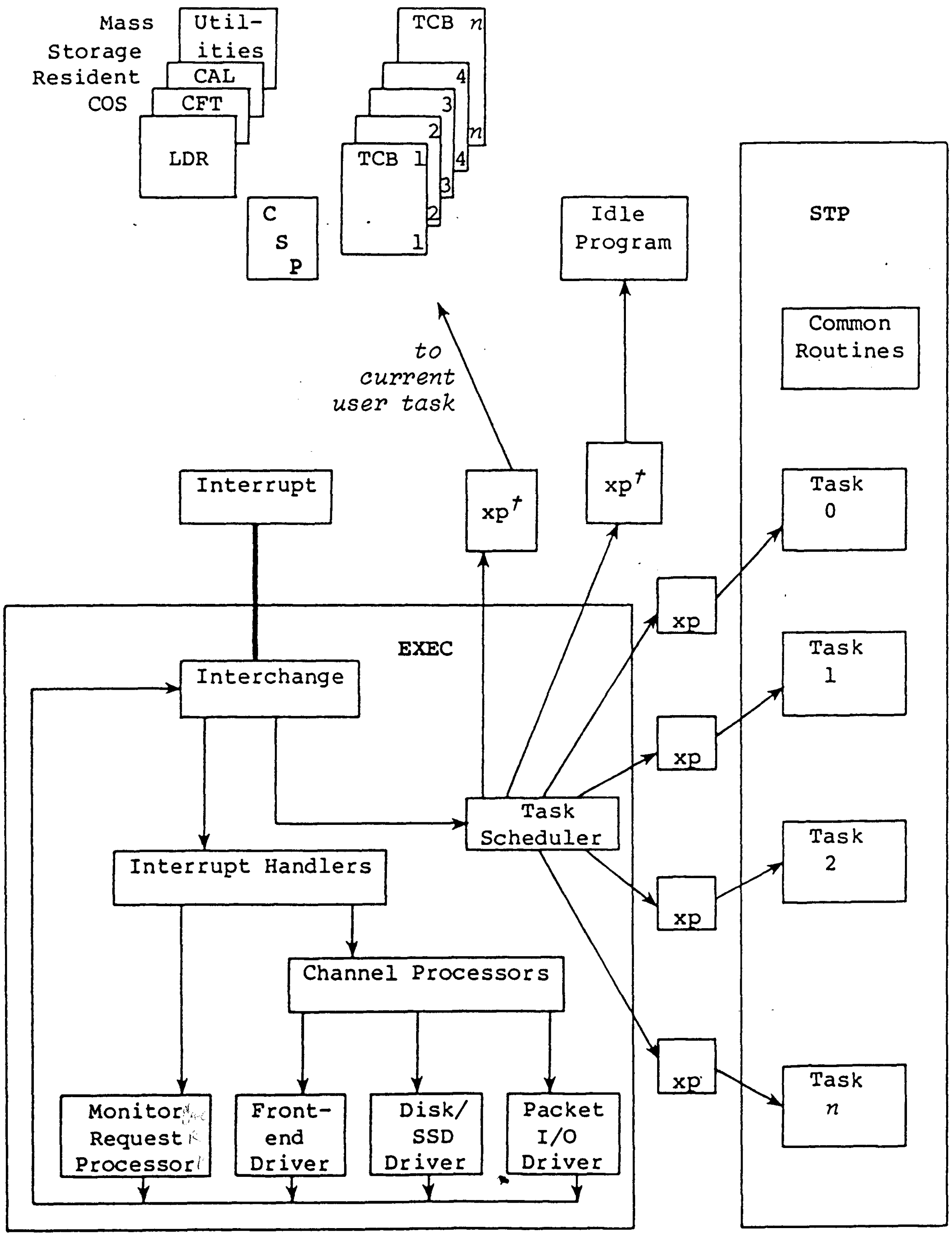
The A and S registers are stored as part of the exchange packages, in the Processor Working Storage, but the B, T and V registers are handled differently.

On any exchange to EXEC, the system task or user program's B00 register is saved because EXEC uses B00.

The active user's B00 is stored during interrupt processing. A system task's B00 register value is stored in the System Task Table (STT). 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 stored in the job's Job Table Area (JTA) and are restored when the job is reconnected to the CPU.

INTERRUPT HANDLERS



System control

† One Exchange Package per CPU

p. 2-4

LESSON 10: EXEC

Objective: State the purpose and function of EXEC.

INTRODUCTION

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.

EXEC has a base address (BA) of 0, and a limit address (LA) equal to the installation parameter I@MEM.

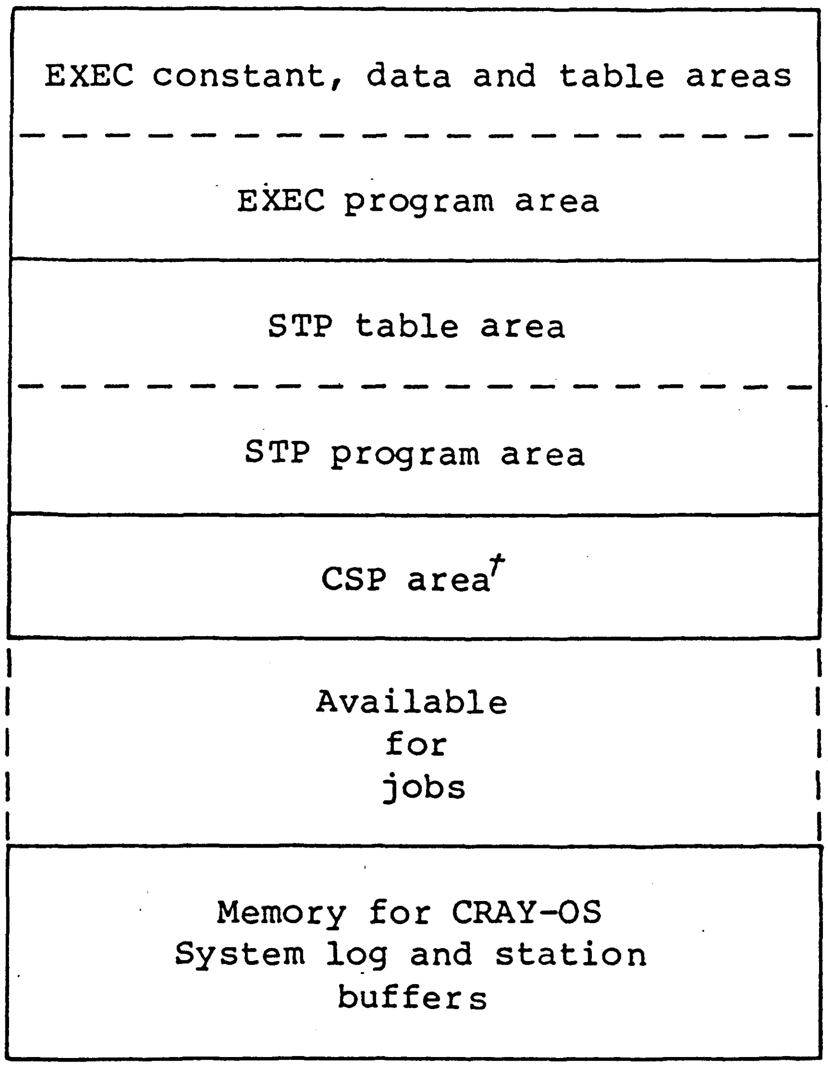
Components of EXEC include:

* 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.

In addition to these routines, the EXEC area of memory also contains:

* EXEC Table Area
* Exchange Packages
* History Trace Table


EXEC constant, data and table areas
EXEC program area
STP table area
STP program area
CSP area†
Available for jobs
Memory for CRAY-OS System log and station buffers

EXEC CONSTANT, DATA, and TABLE AREAS

CONSTANTS

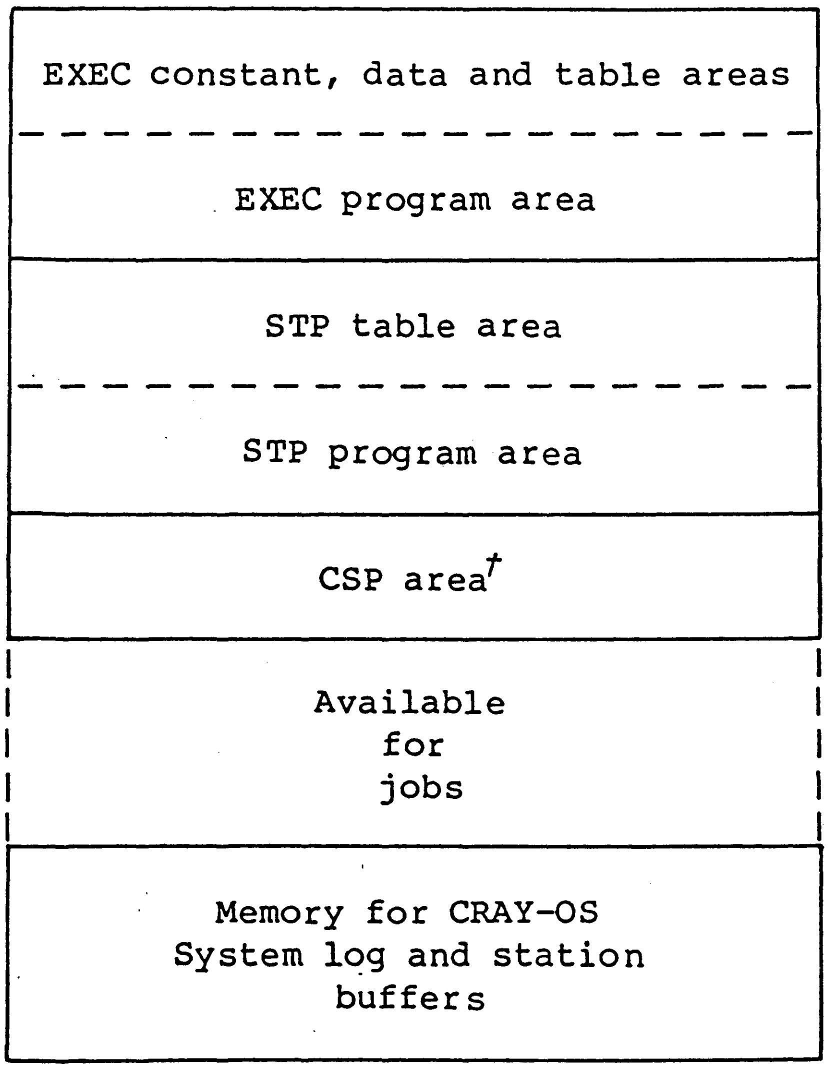
The EXEC constant area contains all EXEC constants. The constants are functionally grouped, and include:

* Constant Memory Locations
* Front-end Driver Constants
* Packet I/O Constants

DATA

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)
* System ID (at location 1400 octal)
* Pointers to EXEC Tables
* Stop Message Buffer
* X-MP cluster register dump area
* Disk/SSD Driver Data
* Packet I/O Driver Data
* Front-end Driver Data
* Miscellaneous data
* EXEC Messages

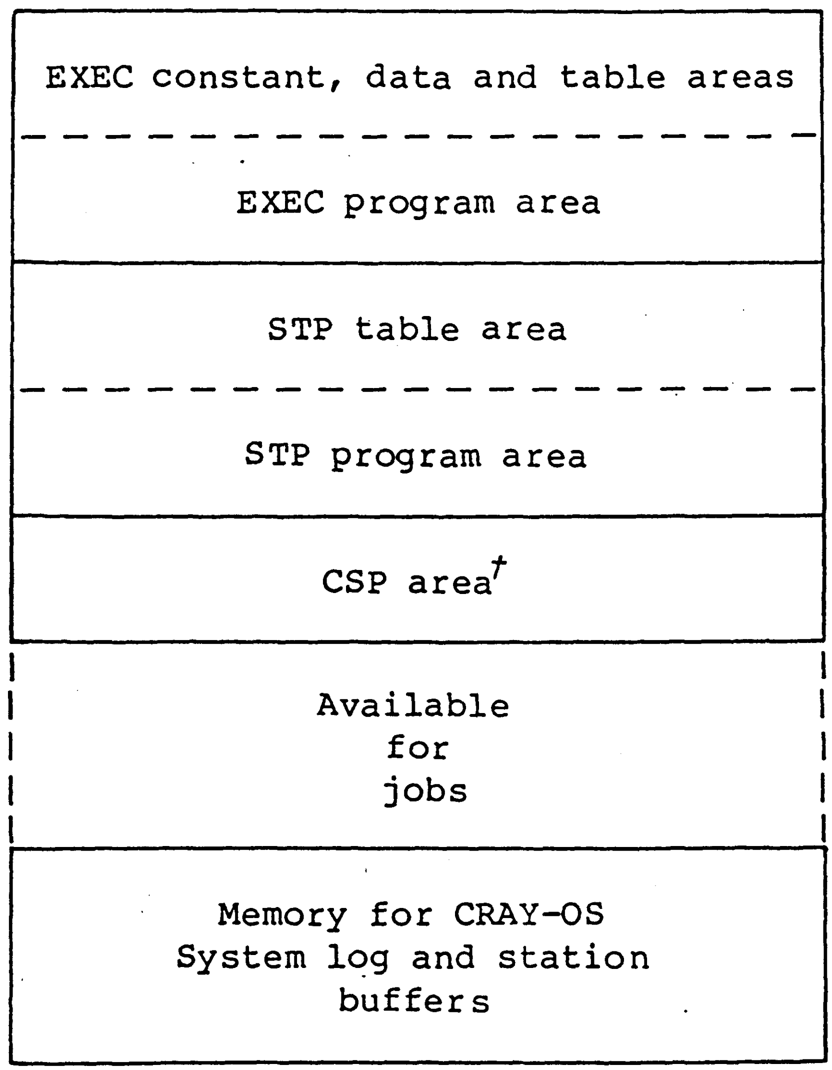

EXEC constant, data and table areas
EXEC program area
STP table area
STP program area
CSP area†
Available
for
jobs
Memory for CRAY-OS
System log and station
buffers

TABLES

The EXEC Table Area contains all EXEC tables, alphabetically ordered. The table descriptions and layouts are addressed in detail in publication SM-0045 "COS Table Descriptions", and will be referenced in Unit 3 of this course - "COS Internals".

The tables used by EXEC include:

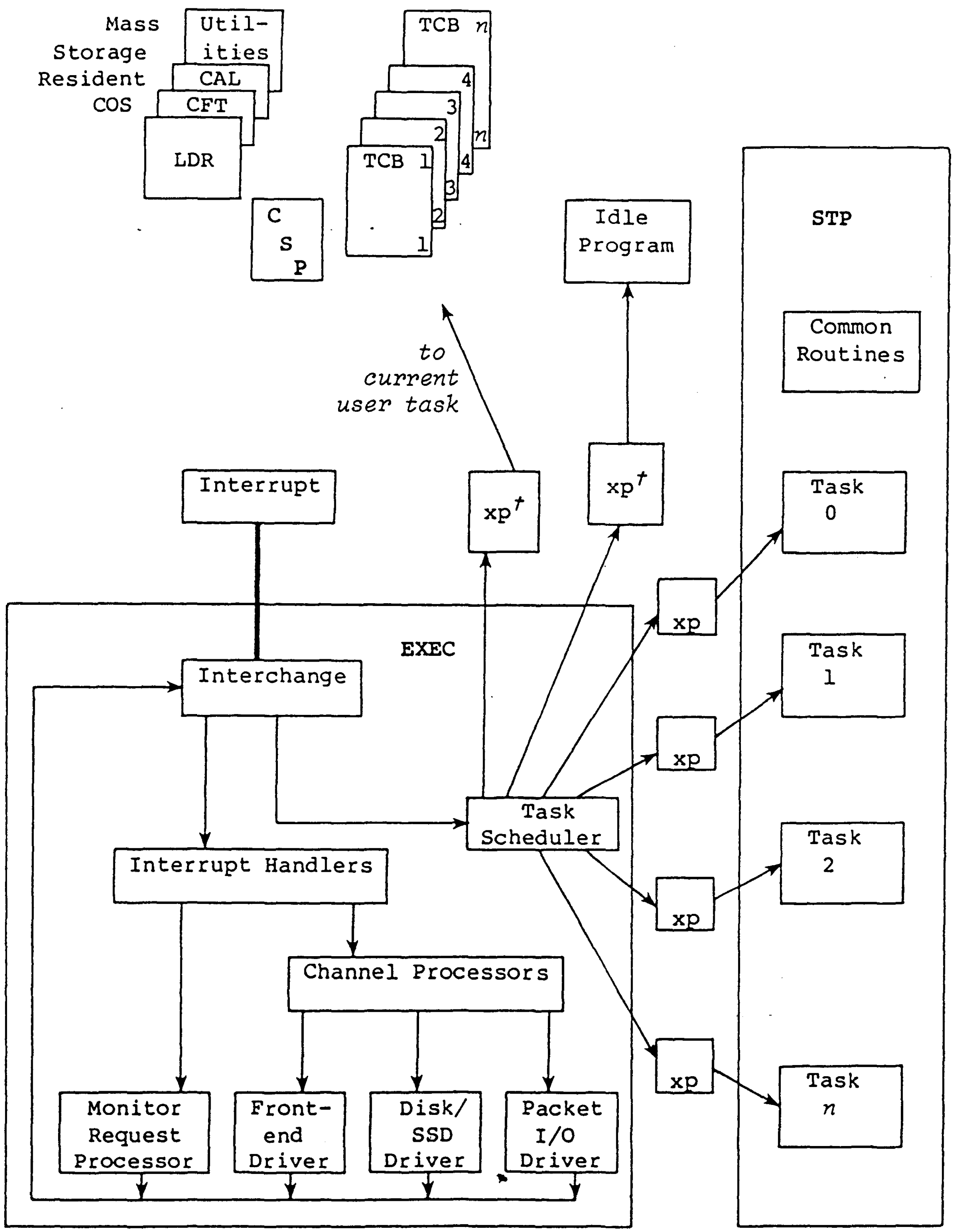
CAT	Channel Address Table
CBT	Channel Buffer Table containing one entry of working storage for each disk driver channel.
CHT	Channel table containing a 1word entry for each side (input and output) of a physical channel. An entry contains a pointer to the Channel Processor Table for the channelassigned 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.
CIT	Channel Interrupt Table
CLT	Channel Limit Table
CXT	Channel Extension Table is used to communicate with the MIOP for front-end I/O.
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


EXEC constant, data and table areas
EXEC program area
STP table area
STP program area
CSP area†
Available for jobs
Memory for CRAY-OS System log and station buffers

EXEC Tables (continued)

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

INTERRUPT HANDLERS



System control

† One Exchange Package per CPU

EXEC Overview

Interrupts

After CPU startup, EXEC begins execution 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.

Interchange Routine

Upon receipt of an interrupt, the interachange 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.

Interrupt Handlers

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 (TSO) is entered.

System Task Scheduler

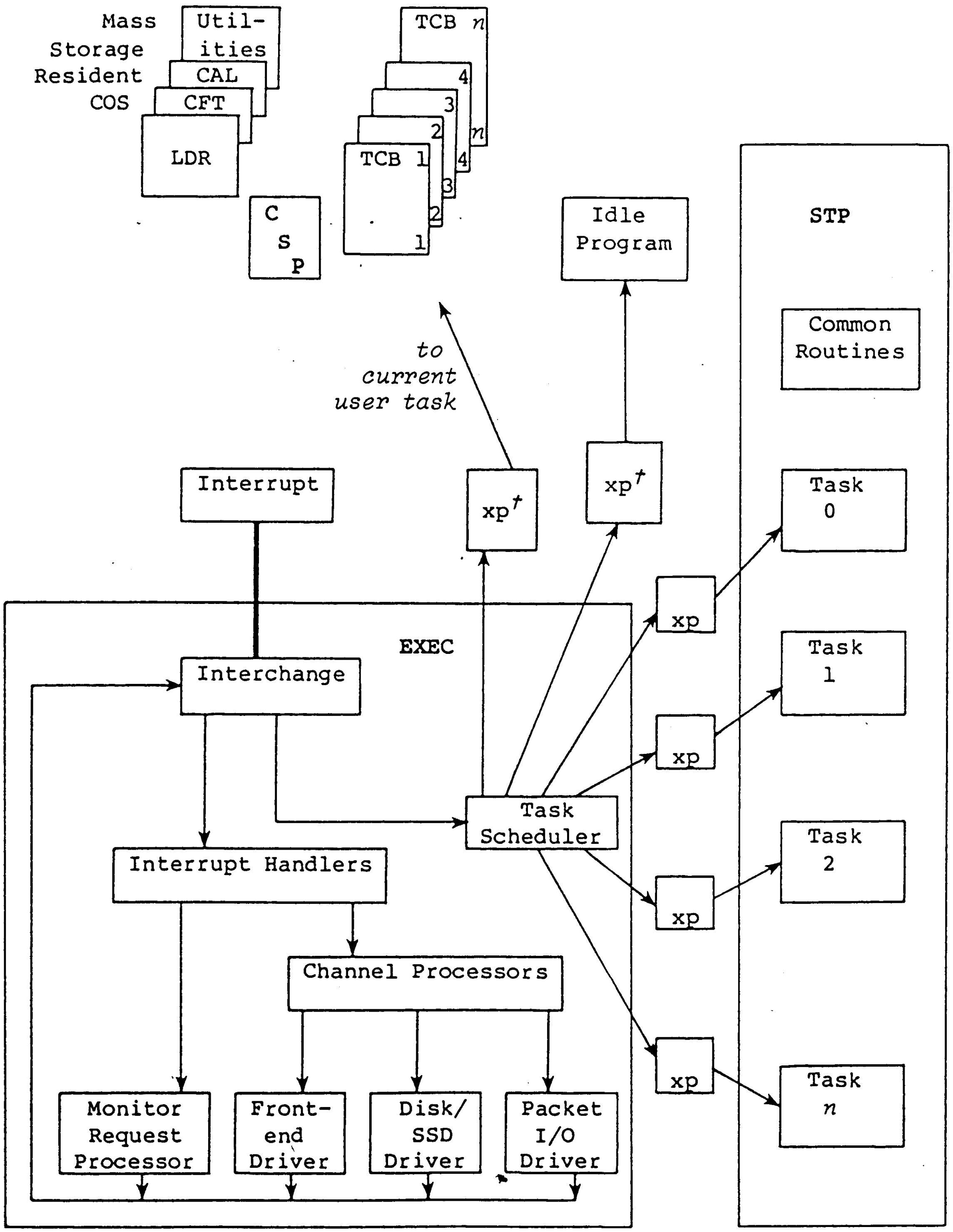
The task scheduler selects the highest priority system task which is ready to run and causes it to be executed.

If no system tasks are ready, the user task scheduler is invoked.

If no user task is currently connected, the idle task is selected 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.

INTERRUPT HANDLERS



System control

† One Exchange Package per CPU

INTERCHANGE ROUTINE

Each time the interchange analysis routine is entered:

* the interprocessor request queue is checked for an inter-processor message. If one is there,
  - it is processed,
  - cleared, and
  - control returns to the interachange analysis routine.

The routine next looks for pending **I/O Channel interrupts.** When an I/O channel is found to have an interrupt pending:

* control transfers to the **I/O Interrupt Handler (IOI)**
  - which clears the I/O interrupt bit in the active exchange package,
  - selects a processing routine based on the channel number, and
  - enters the routine.
  - The channel processor returns control to the interchange routine.

Next, the real-time clock and the time event table are examined. If a timer event is pending:

* Control is passed to the expired event interrupt handler (TEI).
  - After processing the timer event
  - control is returned to the interchange routine.

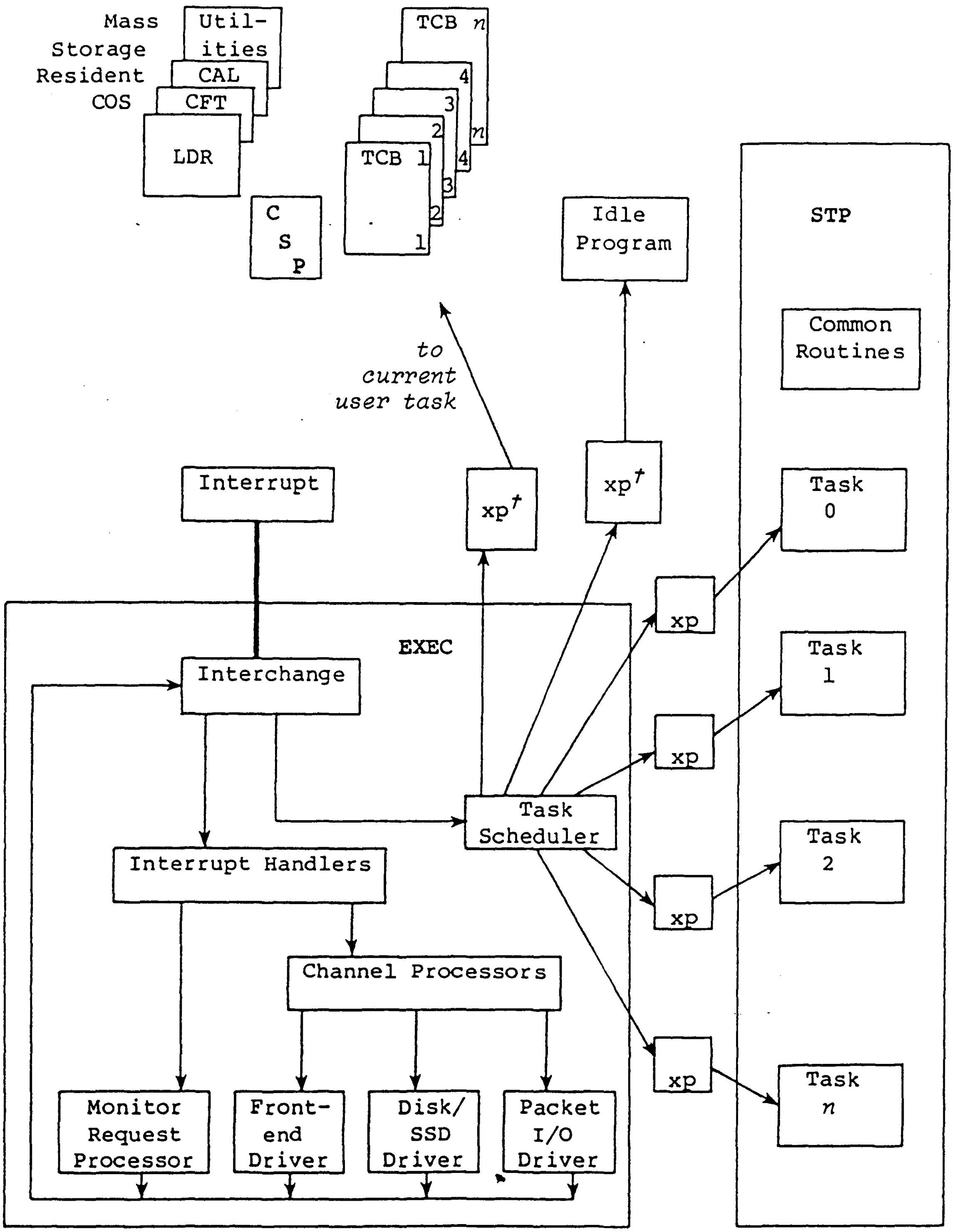
Finally, after the interchange routine has processed all of the above conditions:

* The flags in the interrupted exchange package are examined to determine the cause of the exchange.
  - The I/O Interrupt flag is ignored since the interchange routine has already processed pending I/O interrupts.

* The Interrupt Handler Table maps each flag into a handling routine.
  - When a flag is set, the corresponding interrupt handler is entered.

After a pass through the interchange routine with none of the above conditions encountered, the Task Scheduler (TSO) is invoked.

INTERRUPT HANDLERS



System control

† One Exchange Package per CPU

INTERRUPT HANDLERS

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

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.

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.

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 default time event.

MCU Interrupt Handler (CII)

CII clears the MCU Interrupt flag in the interrupted exchange package.

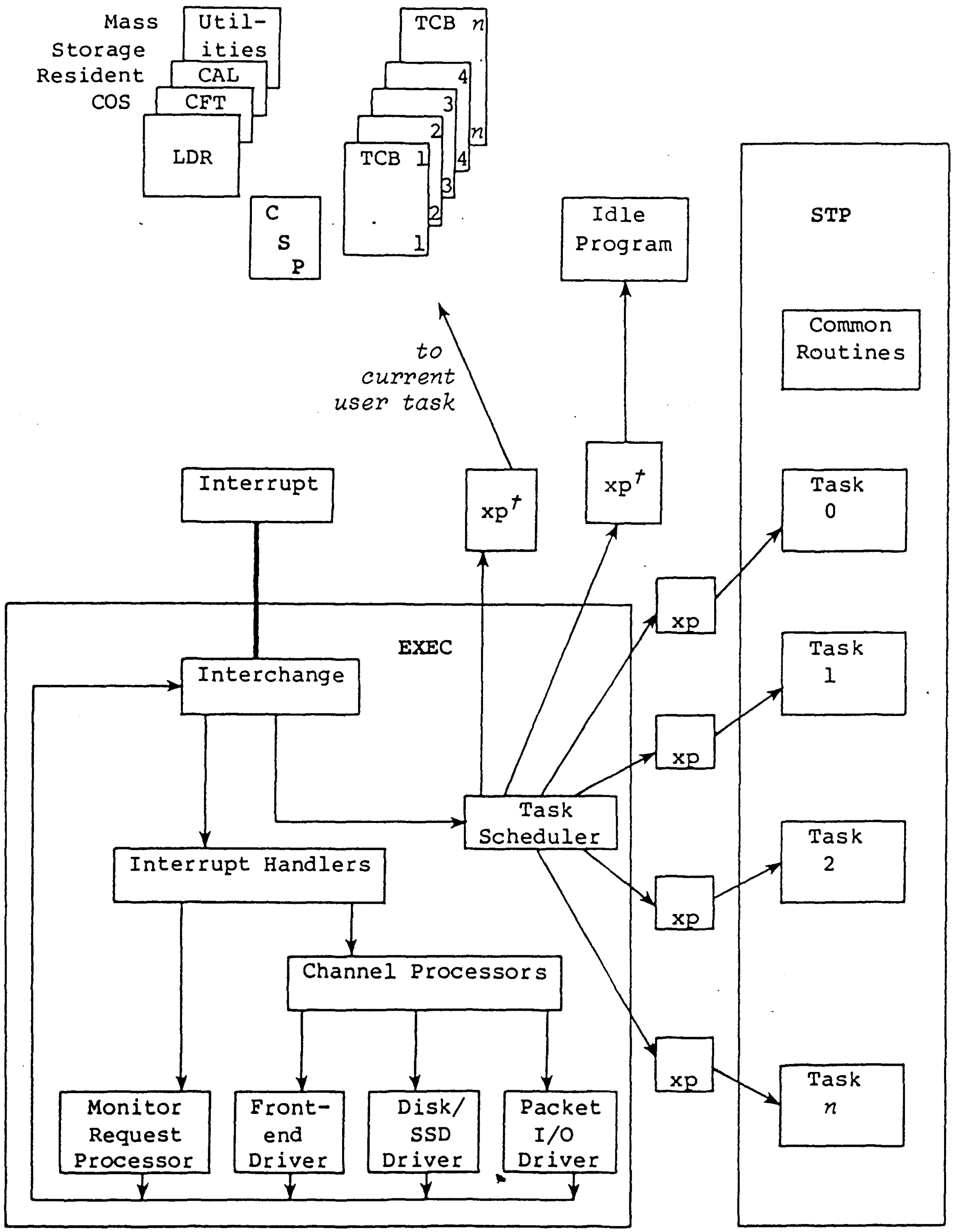
Error Interrupt Handler (EE)

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.

INTERRUPT HANDLERS



System control

† One Exchange Package per CPU

INTERRUPT HANDLERS (continued)

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 multi-bit error causes the system to halt if the error occurred in the operating system or by a channel read from an I/O buffer.

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.

Interprocessor Interrupt Handler (IPI)

IPI clears the interprocessor interrupt flag in the interrupted exchange package on a Cray X-MP.

Deadlock Interrupt Handler (DLI)

On the Cray X-MP, deadlock interrupts can occur that do not indicate that a programming error occurred. For instance, a deadlock interrupt occurs whenever a Test and 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.

10.16

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 configuration deck will be discussed in detail in Unit Two of this course.

Typical channel layouts are shown below:

CHANNEL	PAIR	DESCRIPTION
2,3	1	6 Mbyte channel to MCU (MIOP or DG)
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

10.18

CHANNEL MANAGEMENT TABLES

The following tables aid in channel management:

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

Channel Buffer Table (CBT)

EXEC assings one Channel Buffer Table (CBT) entry to each pair of Channel Table (CHT) entries during EXEC initialization. The Channel Buffer Table is the default processot 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 Channel Table. (Site configuration information is provided in unit 2 of this course.)

Each entry contains:

* A task parameter block address linking the channel to an STP task,
* A table address,
* and an interrupt handler address.

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 in the I/O Subsystem. (The Packet I/O Driver is discussed later in this unit.)

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 (LIT or CBT)

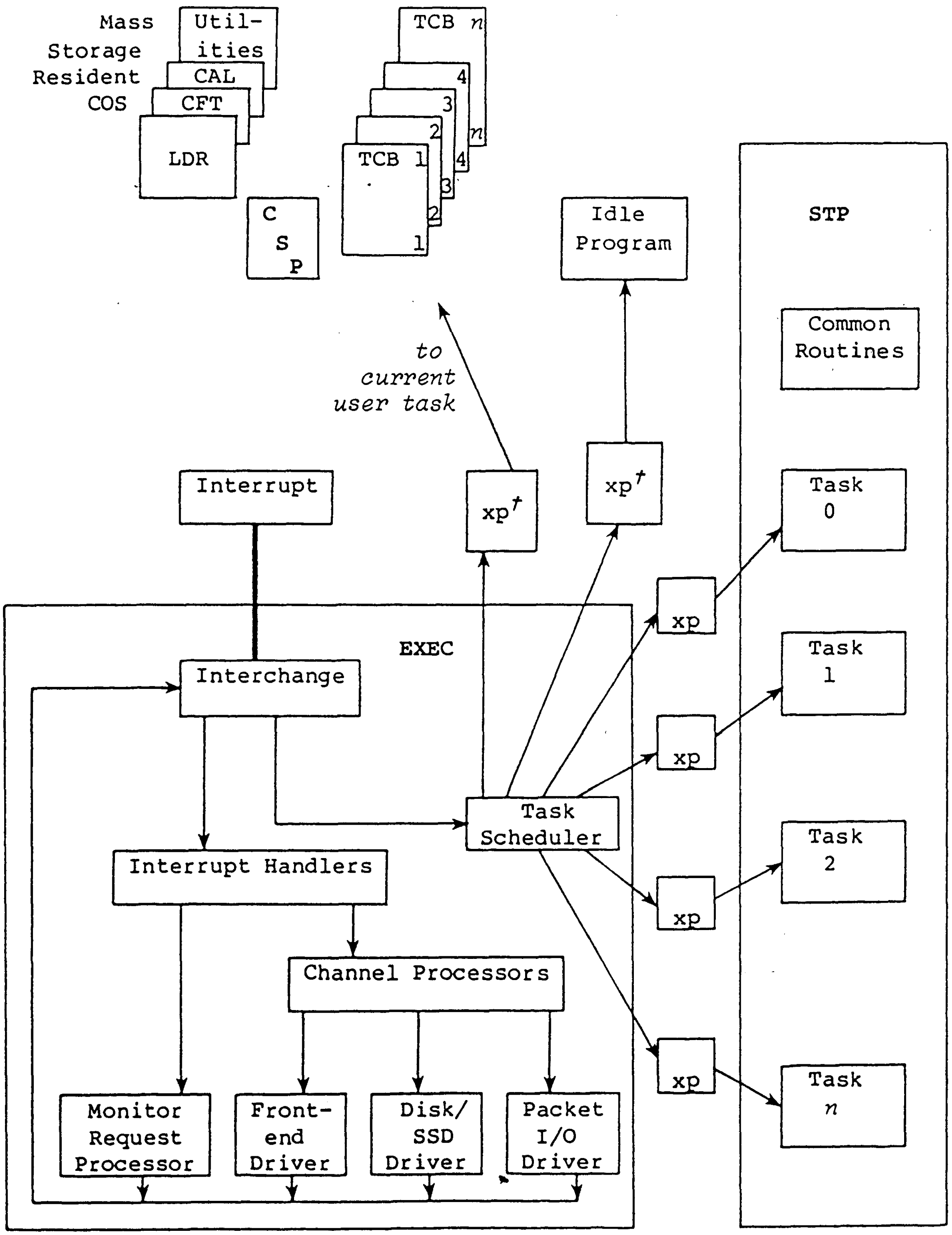
The I/O Service Processor tables contain information for control of the channel processor and can contain pointers to other tables.

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.

CHANNEL ASSIGNMENTS

When an STP task makes an I/O request for a specified channel pair, EXEC assigns the STP task that channel pair.

INTERRUPT HANDLERS



System control

† One Exchange Package per CPU

CHANNEL PROCESSORS

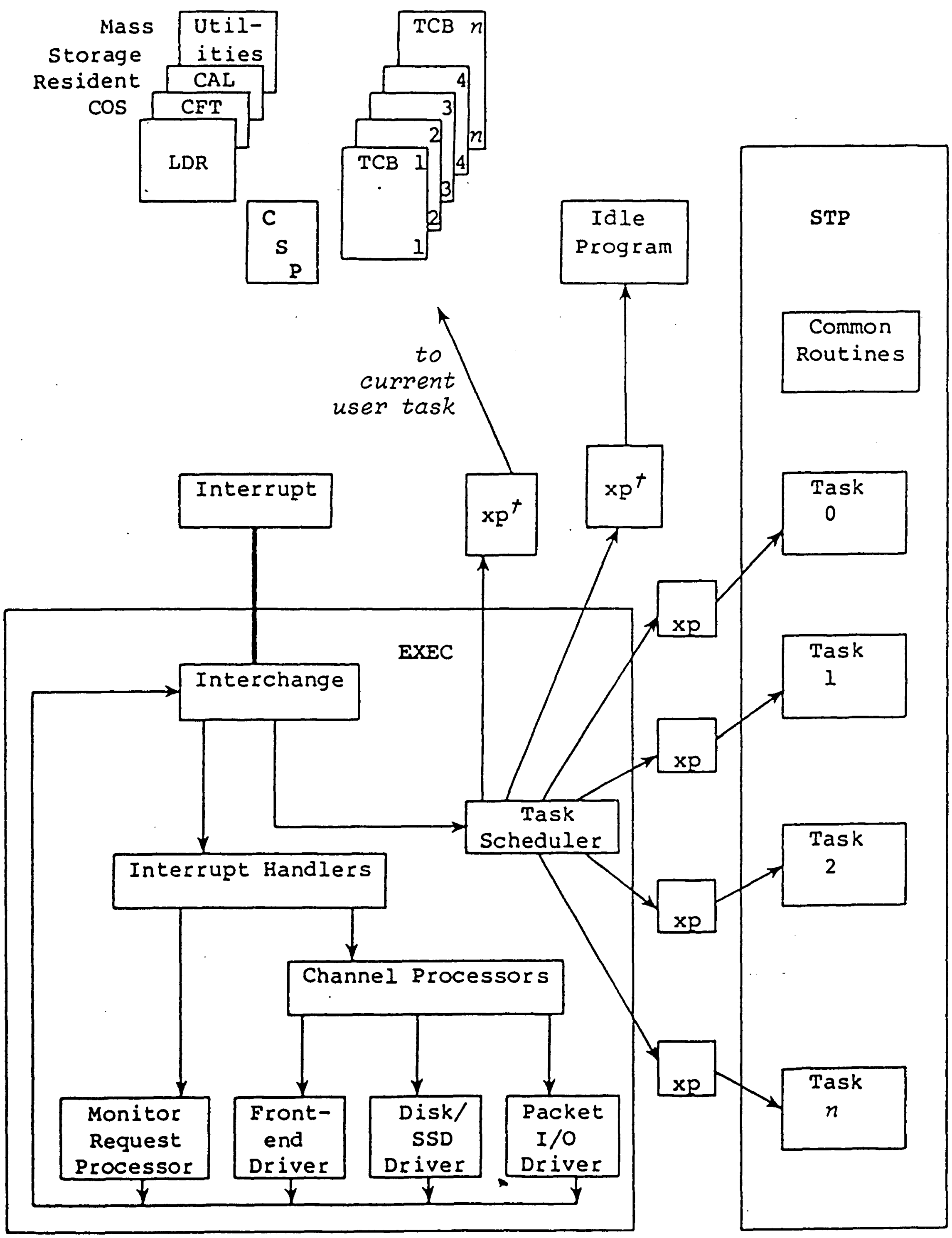
The Channel Table (CHT) has a processor address for each physical mainframe channel configured. By default, this channel processor is the reject (RJ) processor, which ignores all interrupts on the channel.

If the 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
* Disk/SSD Driver Interrupts
* MIOP Driver interrupts

INTERRUPT HANDLERS



System control

† One Exchange Package per CPU

MONITOR REQUEST PROCESSOR

The Executive (Monitor) 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.

FRONT-END DRIVER

The Front-end Driver (FED) physically controls I/O between the Cray mainframe and the front-end computers attached directly to the Cray.

In addition, it passes requests to the MIOP for I/O between the Cray mainframe and front-end computers attched to the I/O Subsystem.

The Front-end Driver is invoked by an Executive (monitor) Request. The Station Call Processor (discussed in Lesson 13 of this unit) is the only task to use FED.

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.

DISK/SSD DRIVER

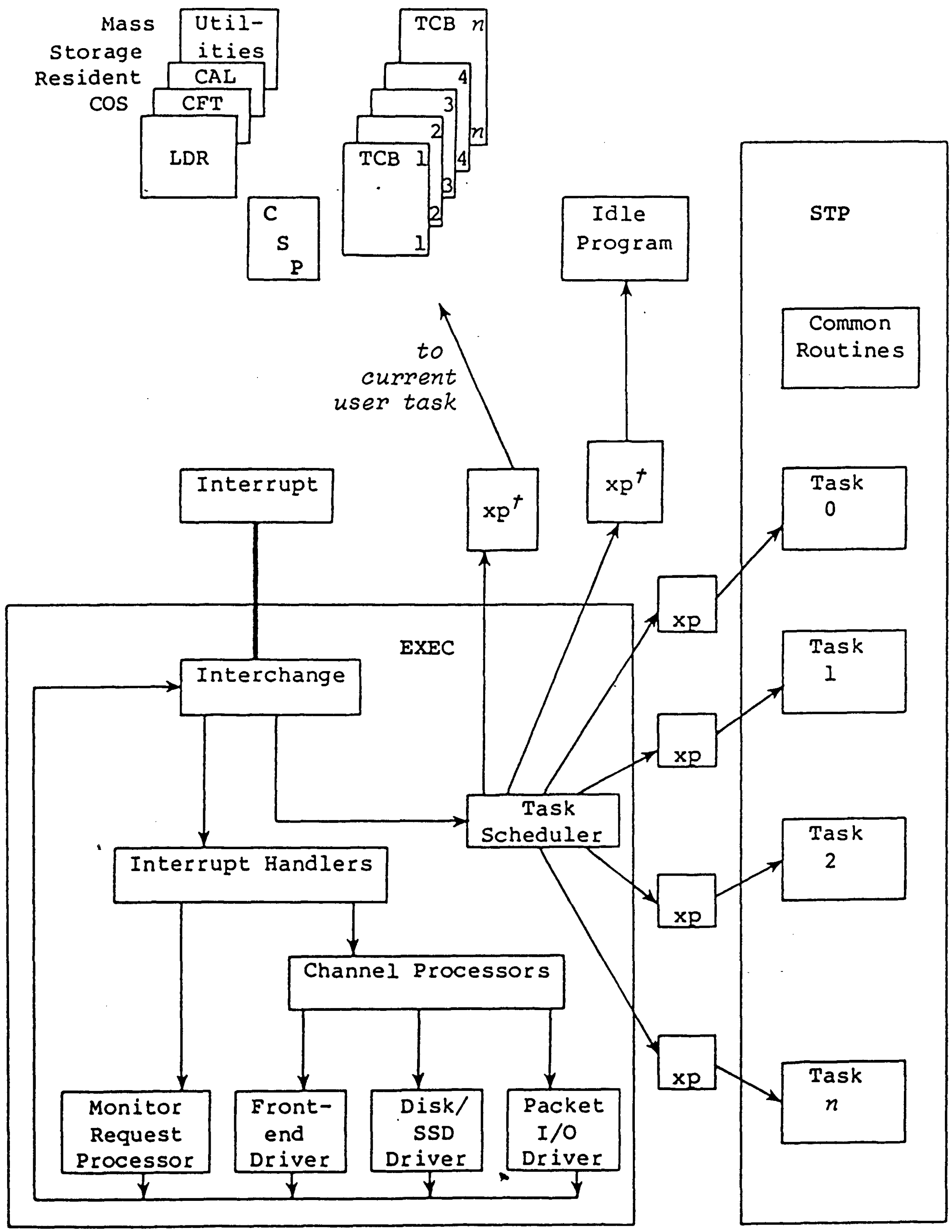
The Disk/SSD Driver controls the following devices connected to a mainframe I/O channel:

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

DISK

Each disk controller can drive from one to four disk storage units of either the DD-19 or DD-29 type.

INTERRUPT HANDLERS



System control

† One Exchange Package per CPU

SSD

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

* On the Cray-1 machines, the SSD is contyrolled 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, the SSD is connected directly to the mainframe through a 1250 Mbyte channel.

PACKET I/O DRIVER

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

* The MIOP driver, which controls the 6-Mbyte channel to the Master I/O Processor in the I/O Subsystem,

* Packet Queueing, which routes packets among three areas of the system:

  - STP Tasks
  - EXEC
  - I/O Subsystem

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

10.28

Packet I/O Driver Tables

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

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

Any Packet Table (APT)

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

Channel Extensionm 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 ID's.

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 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.

PACKET DESCRIPTION

The unit of information passed is known as a packet and is always six 64-bit words long.

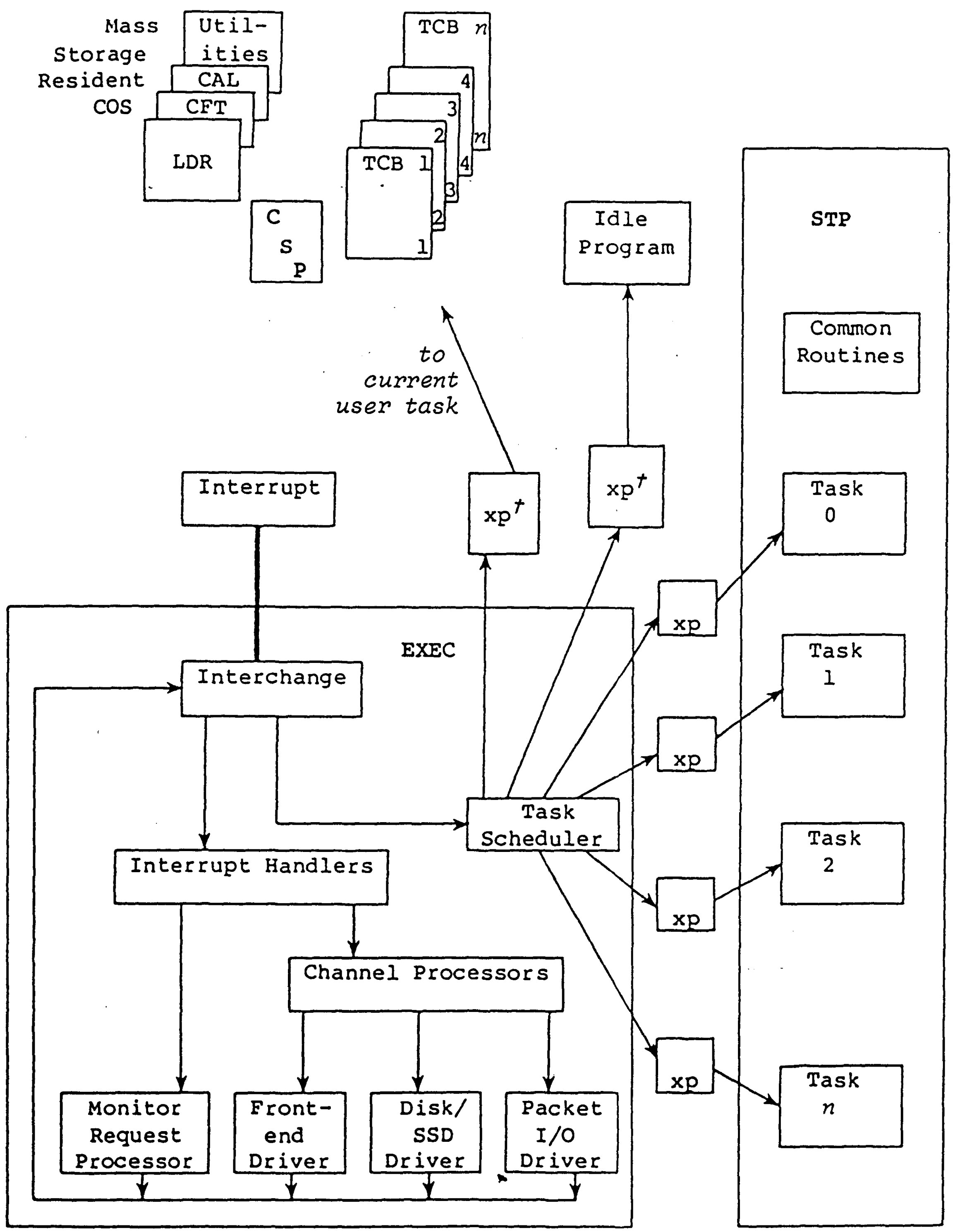
The Any Packet Table (APT) describes most of the formats the packet can take. The packet always has a 16-bit Destination ID (DID) and a 16-bit Source ID (SID) used by the Packet I/O Driver to route the packet to its destination.

The following ASCII identifiers are valid in the SID and DID fields.

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

Packet I/O 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.

INTERRUPT HANDLERS



System control

† One Exchange Package per CPU

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 is selected.

* If no user task (job) 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 (System Task Table) address of the previously active system task.

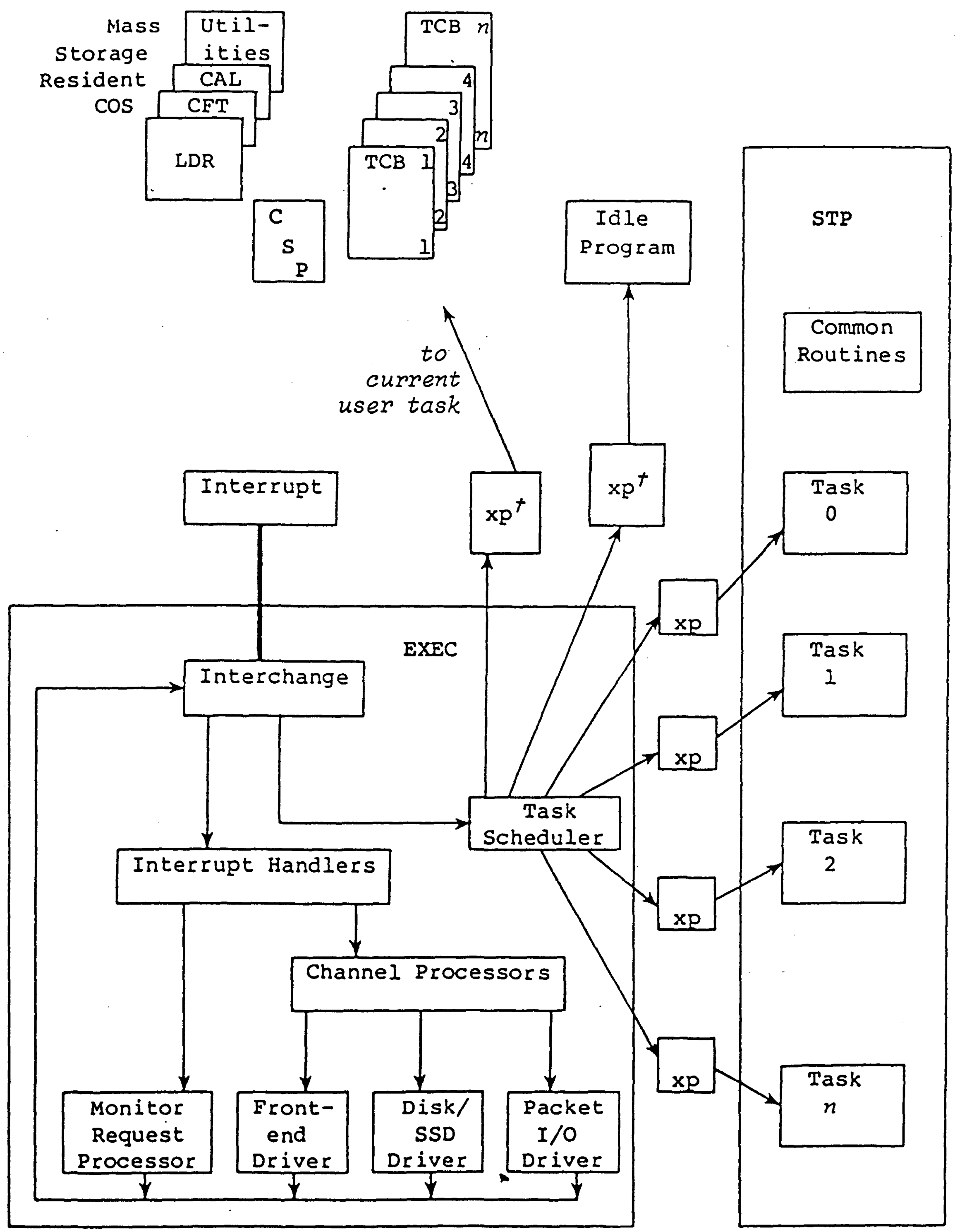
* STPLK, the STP lock indicator. When nonzero, the previously-executing STP task has disabled preemptive task scheduling, indicating that the task scheduler should return to the task.

* TBIDLE, a field in the Task Breakout 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 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.

INTERRUPT HANDLERS



System control

† One Exchange Package per CPU

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

1. 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.

2. 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 intitialized, or if it is suspended, select the idle package instead.

3. 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.)

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

EXEC RESOURCE ACCOUNTING

EXEC maintains the following performance information in EXEC tables:

* Accumulated CPU time for itself (in PWS)
* Accumulated CPU time for each task (in STT)
* Total time given to users (in PWS)
* Count of all channel interrupts for both real and pseudo channels.
* 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

LESSON 11: SYSTEM TASK PROCESSOR (STP)

Objective: State the purpose and function of the System Task Processor (STP) and its relationship to EXEC and the user's job.

INTRODUCTION

The System Task Processor (STP) runs in user mode and accesses all memory other than that occupied by EXEC. STP is responsible for processing all user requests.

STP consists of:

* A set of programs called TASKS
* A set of Tables used by the 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:

* It has its own ID (a number in the range 0-35 octal)
* It has an assigned priority
* It has its own exchange package area in the System Task table (STT),
* It has its own intertask communication control table which defines which tasks it is allowed to communicate with.

Each task will be described in detail in Lesson 12 of this unit.

SYSTEM TASKS

The 15 system tasks are:

STARTUP (STP)

STP handles the process of loading COS into central memory, beginning execution, and generating or recovering tables for the operating system.

STATION CALL PROCESSOR (SCP)

SCP handles functions for one or more front-end computer systems.

STAGER (STG)

STG is a subtask of SCP. It separates the disk I/O processing from the protocol processing in SCP. STG also initiates input jobs by processing the job card, assigning a job sequence number, and calling the Job Class Manager to assign a job class.

JOB CLASS MANAGER (JCM)

Before a job enters the input queue, it must be given a job class. JCM assigns a job to a class.

JOB SCHEDULER (JSH)

JSH is responsible for initiating the processing of a job, initiating processing of user tasks, selecting a user task to be active, managing job roll-in/roll-out, terminating user tasks, and terminating a job.

EXCHANGE PROCESSOR (EXP)

EXP processes all user system action requests and user error exits. EXP also handles requests from the Job Scheduler for initiating or aborting a job.

PERMANENT DATASET MANAGER (PDM)

PDM provides a means of creating, accessing, deleting, maintaining, and auditing disk-resident permanent datasets.

DISK QUEUE MANAGER (DQM)

DQM controls the simulataneous operation of disk storage units on CPU I/O channels or the I/O Subsystem.

TAPE QUEUE MANAGER (TQM)

TQM manages tape I/O between one or more user jobs and the I/O Subsystem.

MESSAGE PROCESSOR (MEP)

MEP exists so that EXEC and the I/O Subsystem can communicate with the system log.

LOG MANAGER (MSG)

MSG writes messages in the system and user log files in response to requests from other tasks.

DISK ERROR CORRECTION (DEC)

DEC is called by DQM, and attempts correction of a disk error by applying the CRC algorithm.

SYSTEM PERFORMANCE MONITOR (SPM)

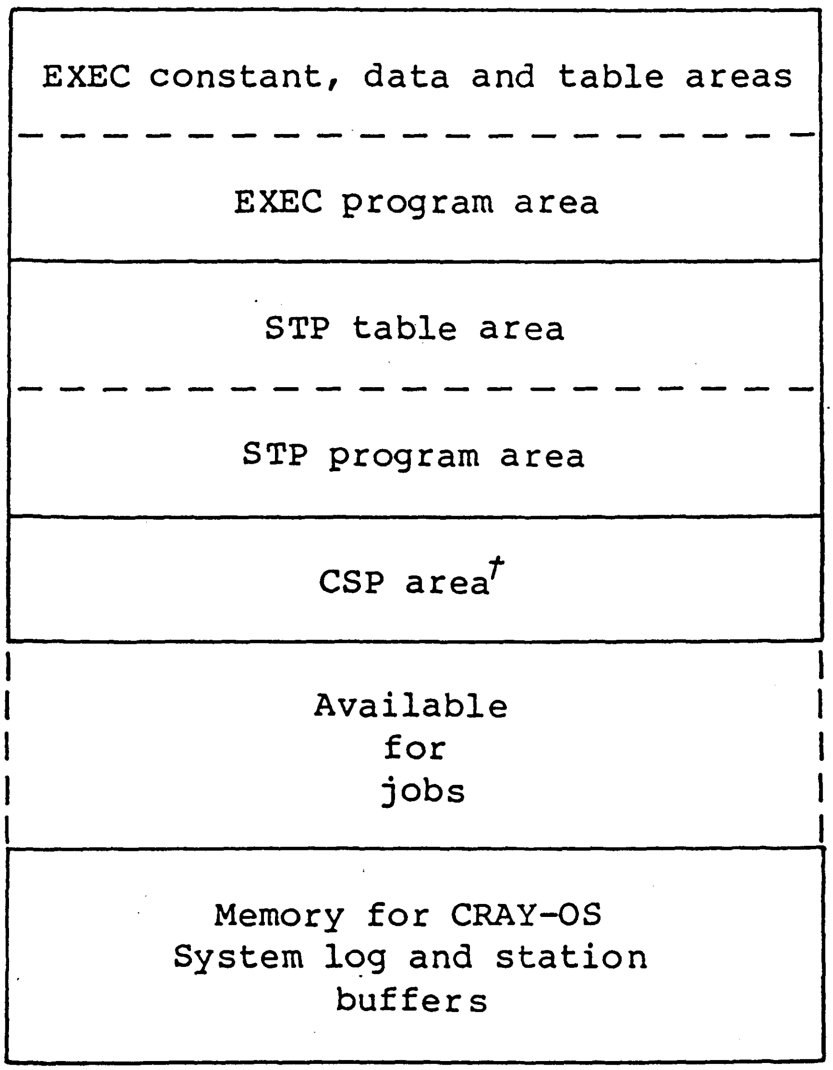
SPM is a low-priority task that collects system performance data and periodically sends it to the system log.

OVERLAY MANAGER (OVM)

OVM handles the loading and executing of system overlays.

FLUSH VOLATILE DEVICE (FVD)

FVD performs the backup of information contained on volatile devices (Buffer Memory and SSD).


EXEC constant, data and table areas
EXEC program area
STP table area
STP program area
CSP area†
Available for jobs
Memory for CRAY-OS System log and station buffers

Task Residence

The addresses in the Base Address (BA) register and the 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.

A create task request assigns an ID and a priority to a task through the task's parameter block in the System Task Table (STT).

Task Execution

Tasks execute in program mode and are therefore 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).

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.

Task Readying

Readying of a task occurs automatically or explicitly. Readying occurs automatically for tasks assigned to a channel when an interrupt occurs on that channel.

Readying also occurs as a result of an explicit EXEC request issued by one task for the execution of another task.

A task is also 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.

Self-suspension

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 (STT) to be set. When this bit is set, the next suspend request for the task causes the task to be re-readied rather than suspended. The task ready request bit is then cleared.

STP TABLES

The following 35 tables are accessible to all system tasks:

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 (tape).

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

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.

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 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 one-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.

Details of the STP tables are given in the COS Table Descriptions Internal Reference Manual, publication SM-0045, and will be addressed in Unit 3 of this course.

11. 12

TASK COMMUNICATION

Tasks communicate with:

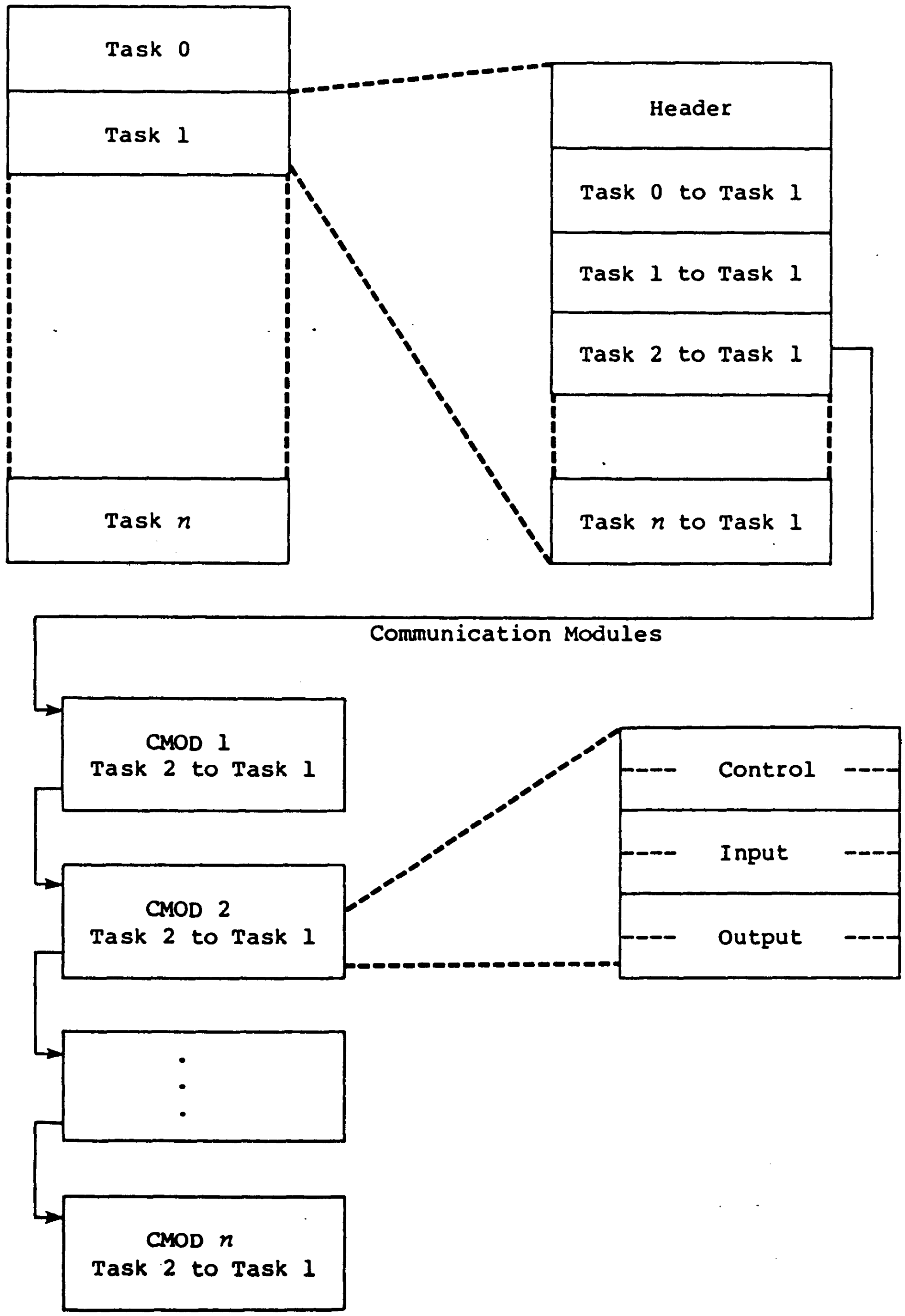
* EXEC
* Each Other
* User Jobs
* the Front-end computer

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.

A reply to the request is returned in registers S6 and S7. Executive requests are discussed in detail in section 2.6 of publication SM-0040.

Communication Module Chain Control



Task communication tables

TASK TO TASK COMMUNICATION

STP contains two areas used for intertask communication:

* Communication Module Chain Control (CMCC)
* Communication Module (CMOD)

CMCC

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 of task 1, etc. The task ID of the **requesting** task and the task ID of the **requested** task are the values that determine the appropriate CMCC entry.

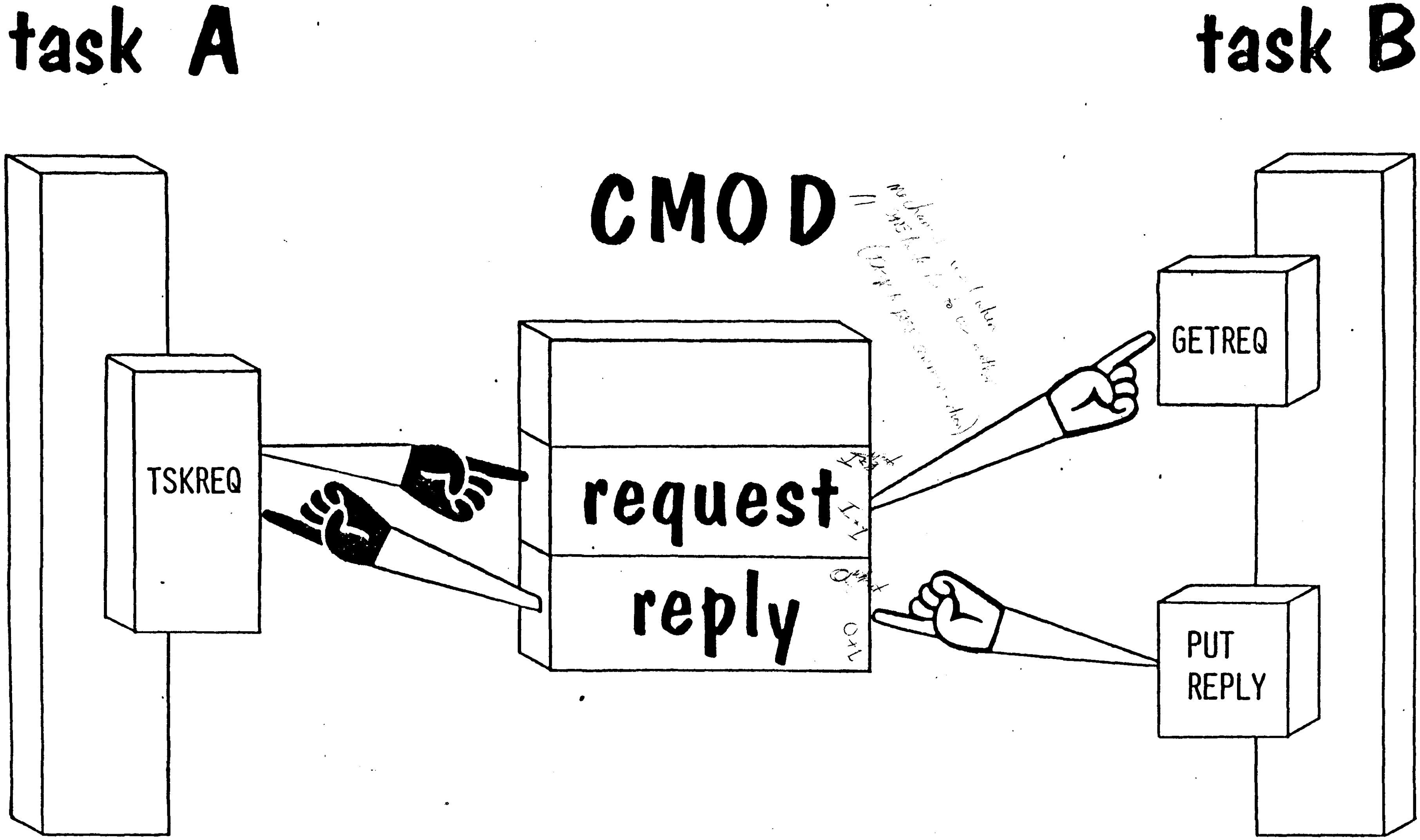
CMOD

CMODs are allocated from a pool as needed and, therefore, have no fixed location within STP.

A CMOD consists of six words:

* (2) for Control
* (2) for Input
* (2) for Output

A task receives all of its requests and makes all of its replies through a CMOD.



11.16

METHOD OF 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.

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

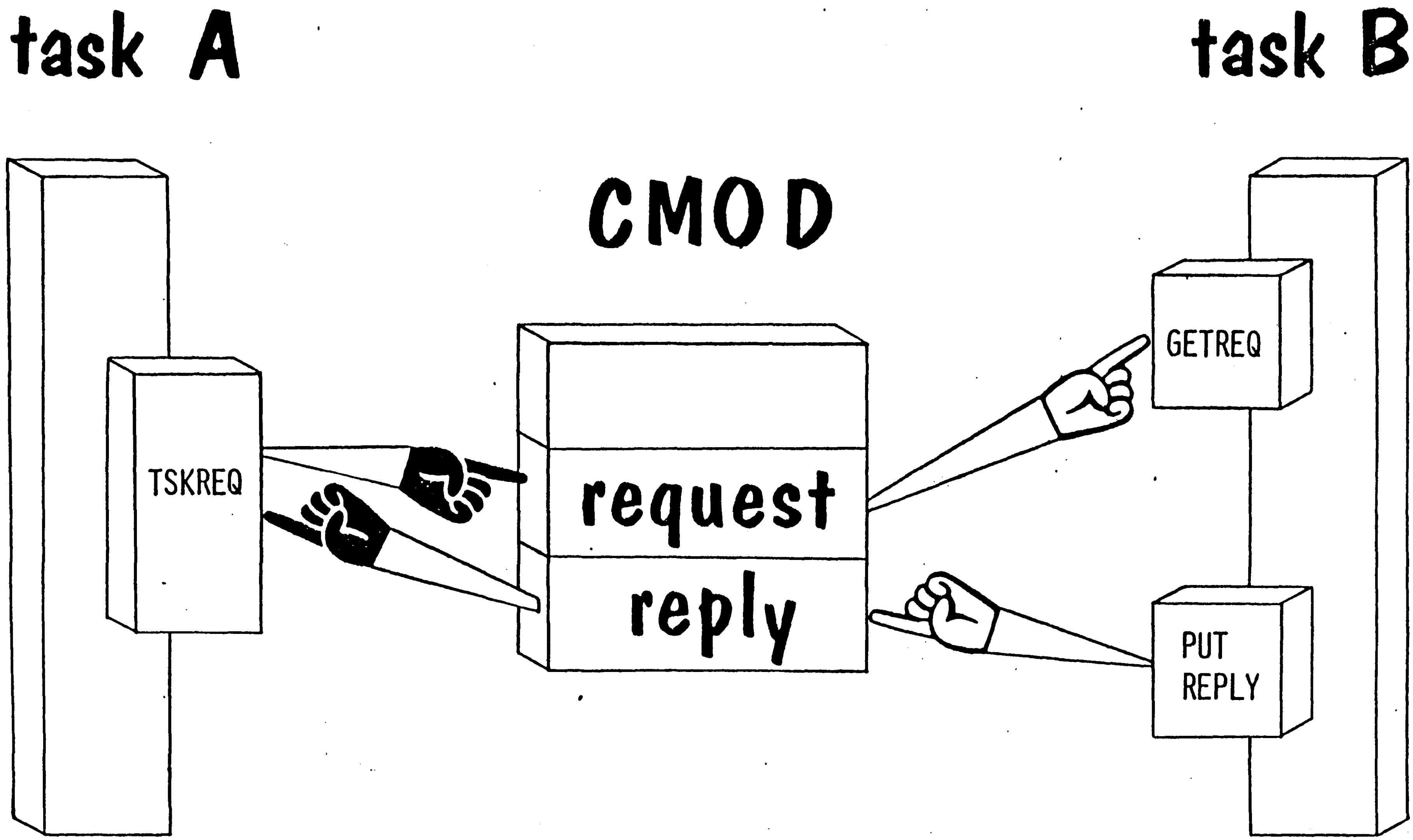
PUTREQ	Put Request routine
GETREQ	Get Request routine
PUTREPLY	Put Task Reply routine
GETREPLY	Request Status routine
TSKREQ	Task Request routine
REPLIES	Queues Unrequested Reply

Tasks call these routines through return jumps.

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.

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



11.1.8

PUTREQ

PUTREQ 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 requestor 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 requestor.

GETREQ

GETREQ locates any outstanding request for the caller.

Using the CMCC, GETREQ searches for a CMOD representing a request not yet given to the requestor. GETREQ begins the CMCC search with the lowest numbered task and returns the first request encountered to the caller.

PUTREPLY

PUTREPLY 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. PUTREQ readies the task where the reply is directed and returns to the requestor.

GETREPLY

GETREPLY searches for a reply to the calling task.

The searches 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.

TSKREQ

TSKREQ 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. 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.

REPLIES

REPLIES 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.

USER - STP COMMUNICATION

User tasks initiate user/STP communication.

A user program request to STP is performed when the user task loads register S0 (or 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 manual, 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 Exchange Processor task (EXP), which is discussed in detail in Lesson 12.

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.

STP COMMON ROUTINES

Certain reentrant routines resident in STP are called by return jumps rather than by a call to another tasks.

These common routines 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

TASK I/O ROUTINES

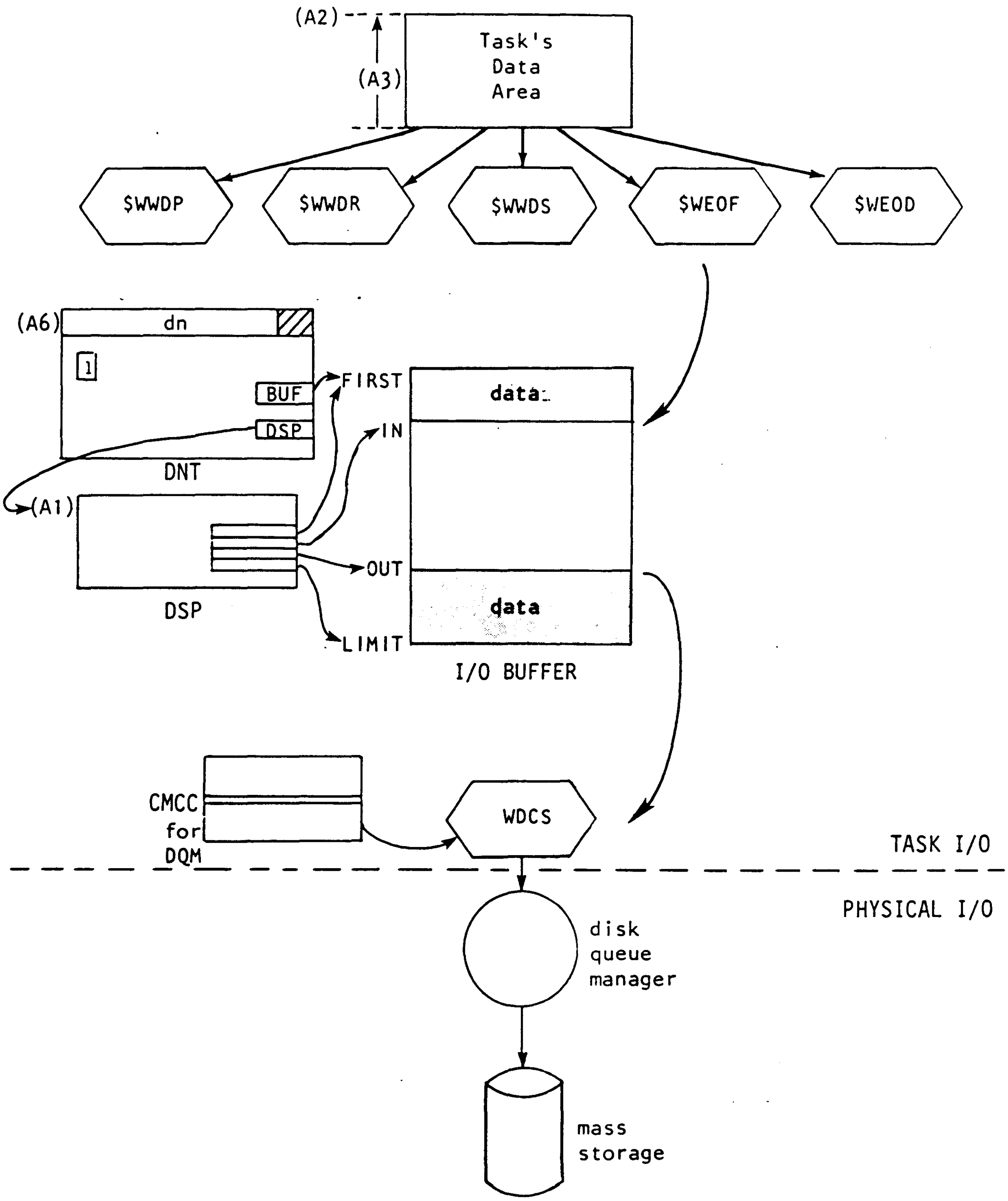
Task I/O (TIO) is a set of reentrant common routines in STP logically considered part of any system task that calls it.

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 call TIO:

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



TIO logical write

TIO (continued)

Primary inputs to TIO consist of:

* a Task Execution Table (TXT) address,
* a Dataset Name Table (DNT) address,
* a Dataset Parameter Table (DSP) address,
* 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.

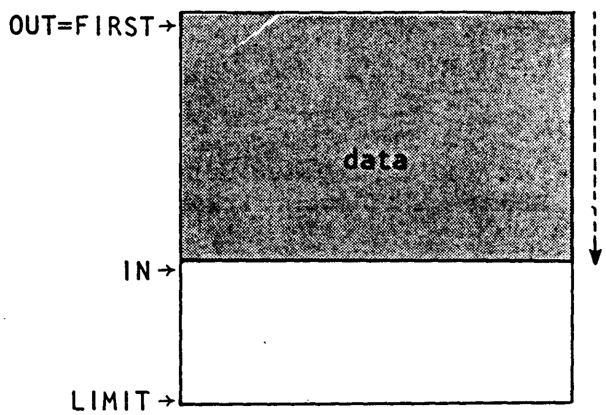
TIO does not allocate or deallocate any of the control structures or buffers for the request, but assumes all control structures and buffers are set up correctly before the request by the system task.

TIO FLOW

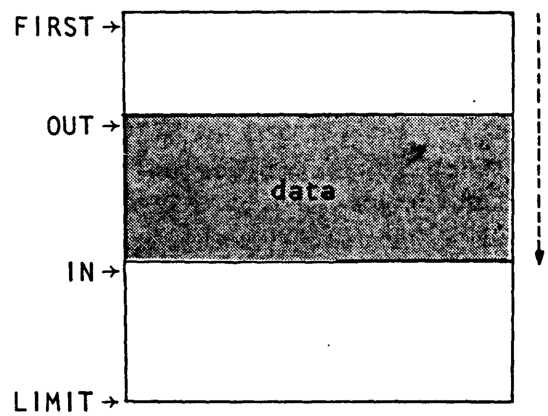
1. System task calls TIO with proper input parameters

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

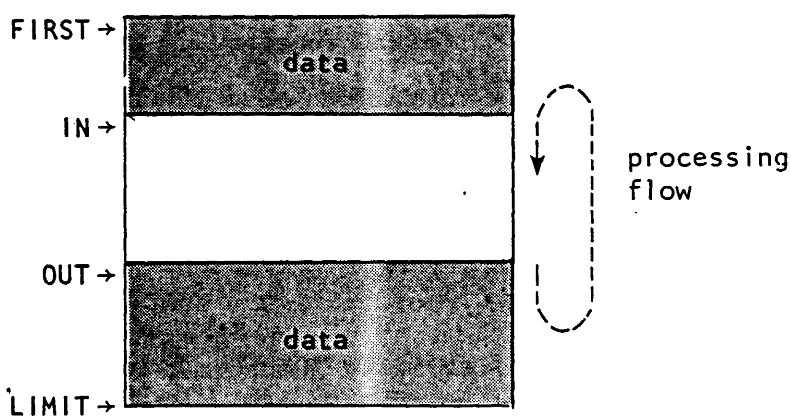
3. If necessary, TIO calls CIO to perform a physical read/write. CIo exits to the calling task's main interrupt loop.



A. Filling the buffer



B. Emptying the buffer



C. Concurrently filling and emptying the buffer

Physical I/O

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.

CIO routines are directly callable from system tasks.

The following system tasks directly call CIO within COS:

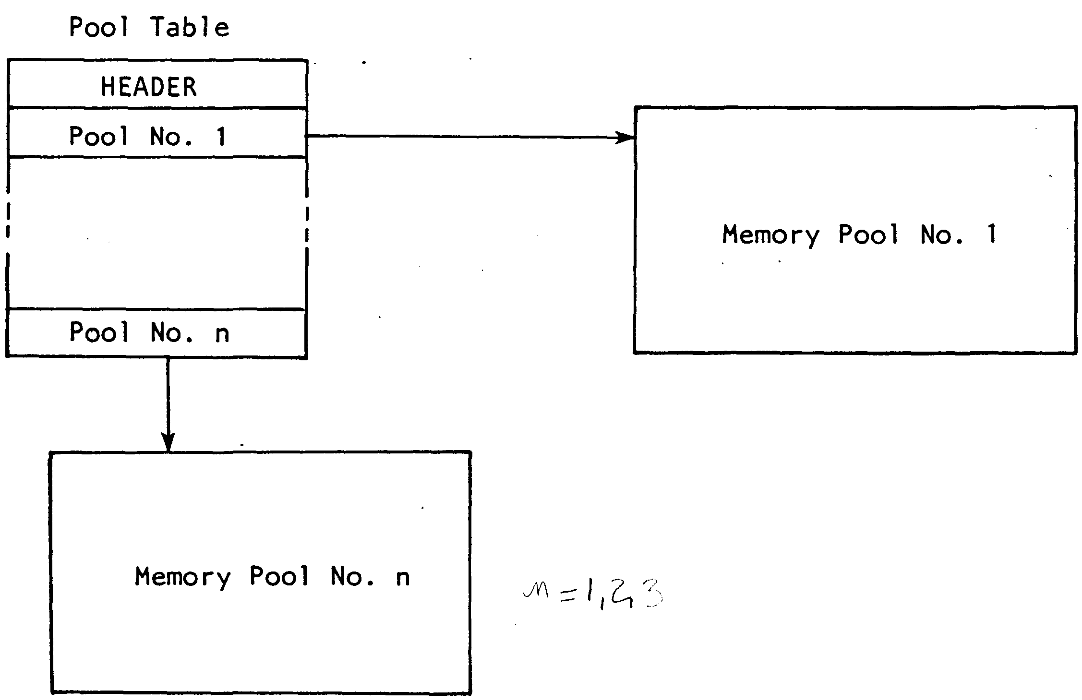
* Exchange Processor (EXP)
* Log Manager (MSG)
* Permanent Dataset Manager (PDM)

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.



Memory allocation tables

MEMORY MANAGEMENT ROUTINES

STP common subroutines provide for allocation an deallocation of variable size memory areas for temporary use by a task.

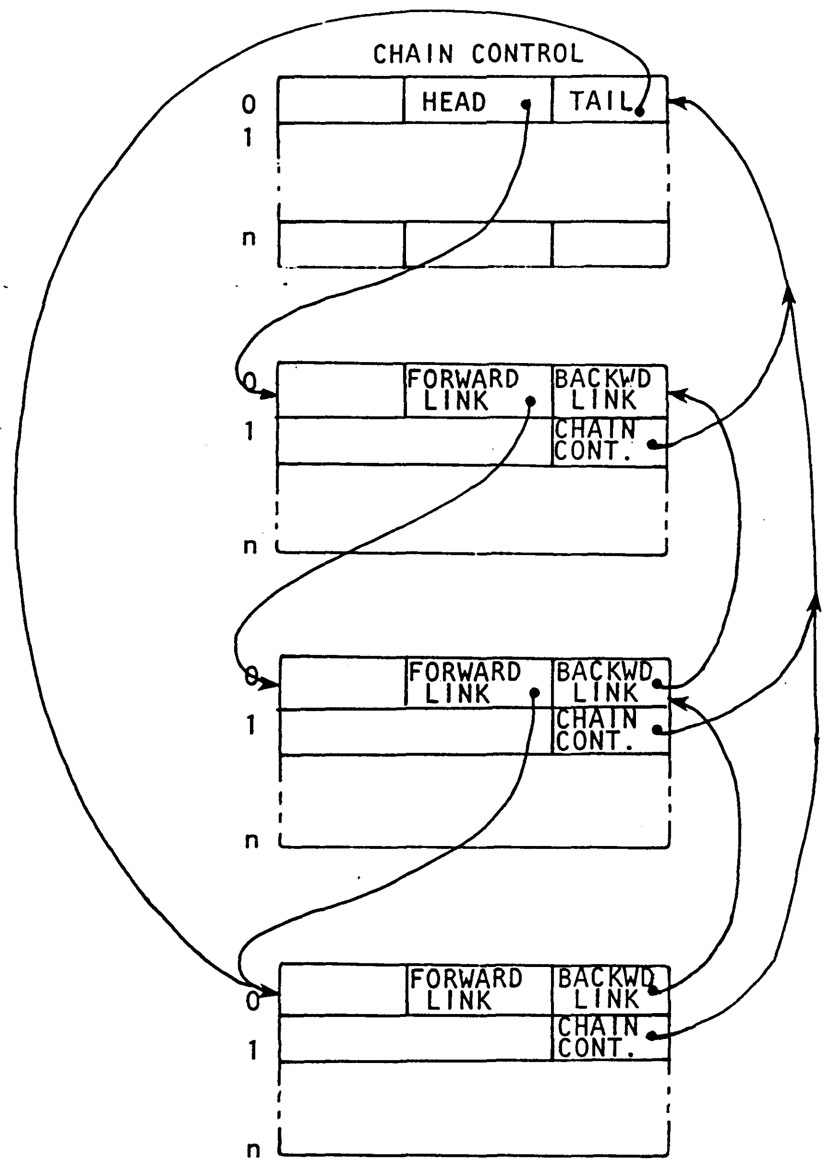
Allocation and deallocation are from memory pools. The number and size of the pools are determined when the operating system is generated.

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 size of the memory pool.



Chain tables

CHAINING/UNCHAINING SUBROUTINES

The CHAIN and UNCHAIN common subroutines provide tasks with a means of linking data.

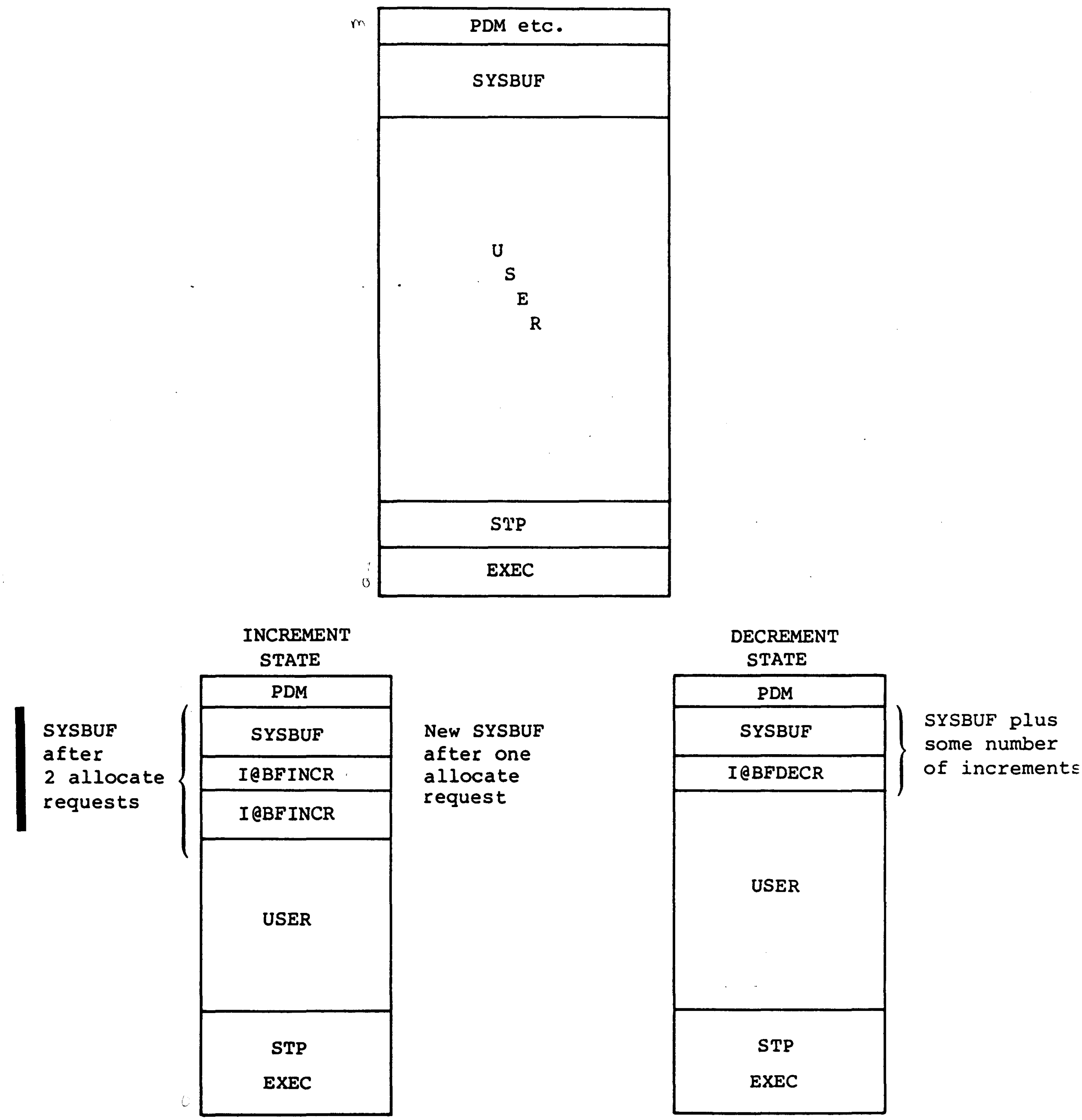
Each piece of data is termed an **item** and consists of two words of header information followed by the information being added to the chain.

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 inter-task 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.

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.

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.



System Buffer memory management

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.

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.

The original buffer is allocated by the Job Scheduler (JSH) and is the size of the installation parameter I@SYSBUF.

As more space is needed, the buffer manager, a common subroutine called BFMAN, requests JSH for an increase in words to be added to the buffer.

Memory is added or removed from the end of the buffer adjacent to user space, which means that availability of user space memory space is affected by fluctuations in communication load.

LESSON 12: SYSTEM TASKS: Purpose & Function

Objective: State the purpose and function of the various System Tasks, and the role they play in the user's job.

INTRODUCTION

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:

* It has its own ID (a number in the range 0-35 octal)
* It has an assigned priority
* It has its own exchange package area in the System Task table (STT),
* It has its own intertask communication control table which defines which tasks it is allowed to communicate with.

COS STARTUP

* INSTALL
* DEADSTART
* RESTART

COS STARTUP

System startup is the process of loading COS into central memory, beginning execution, and generating or recovering tables for the operating system.

The COS initialization task (Startup) is created by EXEC. Startup executes only once ... when the operating system is loaded and started up.

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

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 Startup logic resides in the station software of the station from which startup occurs (such as the I/O Subsystem) and in EXEC.

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).

When the Install option is selected, the Startup task:

* Searches for EFT,s if they exist
* Writes a device label (DVL) on each mass storage unit.
* Accumulates Flaw Information
* Processes Mass Storage Groups
* Creates the Dataset Catalog on the Master Device
* Sets up the DSC and tables in memory
* Reserves space on the master device for system dumps
* Reserves space for the datasets maintained by IOS
* Initializes the Rolled Job Index dataset and enters it into the DSC.
* Optionally creates the Dataset Catalog Extension Table on the master device and enters it into the DSC
* Initializes the Job Class Structure and System Directory datasets and enters them into the DSC.
* Allocates disk space for volatile device backup dataset.

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 in the Dataset Catalog are deleted.

When the Deadstart option is selected, the Startup task:

* Searches for the Engineering Flaw Table (EFT)
* Finds device label on each mass storage unit
* Preserves flaw information
* Preserves mass storage groups
* Reserves Dataset Catalog on master device and the disk space allocated for system dump; initializes DNT and DAT for the DSC
* Preseves the allocated space for the datasets maintained by the IOS
* Restores all data on volatile devices from the backup datasets
* 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 the 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

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.

When the Restart option is selected, the Startup task:

* Attempts to preserve the area reserved for system dumps
* Restores information on volatile devices from their associated backup
* 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.
* If specified, recovers rolled out jobs through call to Recover Rolled Jobs routine (RRJ)
* Preserves or allocates space for the datasets maintained by the IOS
* Copies system dump if necessary and saves the copy as a permanent dataset

Input to Startup

Input to Startup may consist of a parameter file, the Dataset Catalog Extension Table, and the $SDR and $ROLL datasets.

Startup may also receive configuration and status changes to devices from the system operator.

Configuration Changes

Startup can receive configuration information from any of the following sources:

* Information assembled into tables at system genration 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.

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

STATION CALL PROCESSOR (SCP)

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

* Establishing communications with the front-end
* 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

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

PROCESSING FLOW FOR SCP

Upon receipt of each message from a front-end, SCP checks for illegal code or illegal parameters.

SCP then processes the message code as follows:

1. Log on causes SCP to save log on parameters and to initialize the buffer pool.

2. The incoming dataset header causes a System Dataset Table entry to be assigned and the header parameters to be saved in the SDT.

3. A start request is issued to the Stager (STG) task via the Stager Stream Table (SST).

4. SCP trades the input buffer for the empty buffer pointed to by the SST. The STG task is then activated with a process buffer code.

5. Status messages are sent by the front-end and verified by SCP.

6. Memory pool buffer is aquired

7. SCP processes the input stream control bytes:

    - Request to send from Front-end

    - SCP responds with receiving

    - Front-end sends data

    - STG processes incoming data (mass storage)

    - End Data

    - SCP responds with Dataset Saved to front-end

STAGER (STG)

Stager is a subtask of SCP. The purpose of STG is to separate the disk I/O processing from the protocol processing in SCP.

STG:

* Writes data segment buffer contents received from front-end systems to mass storage.

* and fills data segment buffers destined for front-end systems with data from 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

Tables Used By Stager

STG uses the following tables:

* PDD Permanent Dataset Definition

* SDT System Dataset Table

* SST Stager Stream Table

Permanent Dataset Definition

STG uses the PDD to create and release permanent datsets.

System Datset Table

STG places information in the SDT for datasets being transferred to or from a front-end system concerning block size, processing direction, etc.

Stager Stream Table

The SST is used for communications between STG and SCP.

Overview of STG Processing

STG is activated for dataset transfers taking place between the Cray and the front-end systems.

The STG task is dormant when no datasets are being transferred.

SCP requests STG processing for active data streams.

Input Processing

The input startup phase is entered when a Start message request code is received by STG.

1. If a 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 alloctaed, 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:

1. Move data from the segment buffer to the disk buffer.

   When the disk buffer is full, a write to disk is initiated.

2. 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, the segment buffer is release and reallocated.

The Input Termination phase:

Upon receipt from SCP of an End message code (end-of-data):

1. Any data in the segment buffer is copied to the disk buffer and a write is issued to flush the buffer.

2. The disk buffer and segment buffer are released.

3. If the dataset transfer is an ACQUIRE or FETCH, exit.

4. A Permanent Dataset Definition (PDD) entry is allocated.

5. If the dataset is a job, assign a job sequence number, and call the Job Class Manager to assign a class.

6. If the dataset is a job, PDM saves the input dataset.

7. When PDM is complete, SCP is notified.

STG Output Processing

Startup phase

The output startup phase is initiated by a Start message request code from SCP.

1. Allocate a segment buffer

2. Set parameters in the SDT for reading the dataset

3. Allocate the disk buffer

4. Initiate the disk read.

Transfer phase

1. Reallocate a segment buffer if the current buffer is empty.

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 disk buffer is empty, reallocate it.

Output Termination phase

1. Release disk and segment buffers

2. Allocate a PDD

3. PDM deletes the output dataset

4. Release the PDD used to delete the output dataset

JOB CLASS MANAGER (JCM)

Before a job enters the input queue, it must be given a job class assignment.

The Job Class Manager task (JCM) assigns a job to a class.

JCM uses the job class structure currently in effect based on installation parameters to determine the class assignment.

The Job Class Manager task is created with all other system tasks by the Startup procedure.

A task can call JCM by setting the appropriate input registers and calling PUTREQ and TSKREQ. JCM replies to each request by setting the appropriate output registers.

Job Class Assignment

A job can only belong to 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 the use of the CL parameter on the job control statement, but the job must still meet the qualifications of the specified class. If the job does not qualify for any class, it is assigned to the class defined using CHAR=ORPH (orphan).

See JCSDEF in the COS Operational Aids Reference Manual (SM-0044) for a detailed description of a job class structure.

JOB SCHEDULER (JSH)

The Job Scheduler (JSH) task is responsible for:

* Initiating processing of a job
* Initiating processing of a user task
* Selecting a user task to be active
* Managing job roll-in and roll-out
* Terminating user tasks
* Terminating a job

The staging task (STG) builds a System Dataset Table (SDT) entry containing the job card parameters and information to find the dataset.

The Job Scheduler then performs:

* JXT allocation
* Initial TXT allocation
* Memory Allocation
* CPU connection

JXT allocation

JSH allocates a Job Execution Table (JXT) entry for each job.

The information in the JXT contains:

- current status of the job
- location in memory or on a roll file,
- working values of priorities

TXT allocation

The TXT contains working values of concerning CPU use.

The TXT includes:

- The most recent job logfile and
- most recent control statement message

to enable the operator to determine the current job step.

Memory allocation

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 the priority considerations, a memory allocation can be taken away from a job, and the job can be written out to the roll file.

CPU allocation

JSH allocates the CPU(s) 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.

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 then lower priority jobs
* Responsiveness should be available to those jobs that require it.

EXCHANGE PROCESSOR (EXP)

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** for **initiating** or **aborting** a job.

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 flag
TCEPM	4	JTA Expansion Request flag
TCEPA	40-63	Continuation address; EXP address if TCEPC=1.

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.

JSH sets the **TCEP** field to 1, indicating a JSH request.

EXEC, recognizing the TCEP field has been set to 1, readies the Exchange Processor, initiating the job.

System Action Requests (Normal Exit)

Sequence Of Events:

1. 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.

   (See page 8-3 thru 8-22 in manual SM-0040 for a list of system action request codes)

   If an error is encountered, the job normally aborts with appropriate messages issued in the logfile. For some errors, however, an error code is placed in the user's S0 and the user is allowed to continued processing.

2. EXEC sets the TCEPN field in the TCEP word and readies the Exchange Processor (EXP).

3. When EXP is readied, it detects the user request because of the TCEPN field being set.

4. EXP then processes the system action request by using the function code in S0 as an index into the CALL table (discussed later in this lesson) to obtain the address of the routine to process this request.

5. After EXP processes a request, it clears TCEP to allow EXEC to return to the user job.

   If EXP cannot process 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.

   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.

User Error Exit

When a user program executes an error exit instruction or encounters a hardware error (floating-point error, operand range error, or program range error), an exchange to EXEC occurs.

EXEC readies EXP after setting the following fields in the Task Control Block in the job's JTA:

* TCEPX is set to 1
* TCEPF is set to the exchange package flags in the user exchange package.

EXP either initiates **reprieve processing** or issues appropriate error messages and aborts the job.

ABORT

If the job is not reprievable, EXP skips through the control statements to the one following the next EXIT statement or to the end of file.

If the statement is DUMPJOB, a dataset named $DUMP is created which contains the job image, including the JTA and the entire user field.

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.

Reprieve processing is enabled by issuing the SETRPV macro instruction in a CAL program, or by calling the SETRPV library routine in CFT.

Sequence

1. When a job step is terminated, 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 as reprievable. If so:

2. The reprieve processing routine clears the current reprieve values,

3. Copies the exchange package, vector mask register, error class code, and actual error code contents to the user-specified area,

4. Sets up the user-specified reprieve routine to receive control when the job is selected for execution, by placing its address in the P-register of the exchange package.

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.

  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.

The job will become recoverable as soon as the Job Scheduler rolls the job out to disk again.

A job is declared irrecoverable by a call from EXP to the Job Scheduler. 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, and writes the modified index to disk.

When the modified index is successfully written, JSH resumes the job.

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, and can be requested using the RERUN macro or RERUN control statement.

When a job is rerun, the results should be the same as those obtained if the original execution had continued to a normal termination.

However, after a job has performed certain functions, 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.**

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

If the job is ineligible for rerun, it aborts with an informative message when the Job Scheduler attempts to reinitiate the job.

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).

System tables usually accessed by the Exchange Processor are:

* CALL Call Table
* JXT Job Execution Table
* QDT Queued Dataset Table
* SDT System Dataset Table

Call Table

The CALL table is composed of a 1-word entry for each user system action request. The contents of the user's register S0 serves as an index into the call table to obtain the address of the routine that processes the request.

Job Execution Table

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.

Queued Dataset Table

EXP modifies the QDT when a job releases a local scratch dataset having related disposes.

System Dataset Table

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.

It also allocates an SDT if a dataset is submitted to the input queue.

Task Control Block

The TCB 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, and CPU timimg information.

PERMANENT DATASET MANAGER

The Permanent Dataset Manager task (PDM) provides a means of creating, accessing, deleting, maintaining, and auditing disk-resident permanent datasets.

The Permanent Dataset Manager is called by the Exchange Processor (EXP) for:

*	SAVE	Creates user permanent dataset
*	ACCESS	Associates a user permanent dataset with a job.
*	DISPOSE	Stages a CRAY permanent dataset to a front-end computer system
*	RELEASE	Relinquishes access to the named dataset for the 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

and to perform functions for PDSDUMP, PDSLOAD, and AUDIT.

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

PDM is also called by **SCP** to:

* create Dataset Catalog entries for spooled input datasets,
* delete DSC entries for spooled output datasets,
* perform permanent dataset name (PDN) requests,
* SAVE datasets staged from front-end stations

PDM is called by **EXP** to:

* create DSC entries for splooed output datasets,
* delete DSC entries for spooled input datasets,
* rewrite spooled input dataset entries

PDM is called by **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.

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

Functions

A task calls the Permanent Dataset Manager by placing a message in the PDM CMCC.

The layout of the CMCC is shown below:

	0–15	16–39	40–63
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 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 call
DAT	INPUT+1	16-39	Dataset Allocation Table, if system call
JTA	INPUT+1	40-63	Base address of the associated job's JTA. If the system flag is not set, the JTA must be specified.

The FC field of the PDD indicates the function to be performed.

The function codes processed by PDM are:

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

CRAY OPERATING SYSTEM

INTERNALS

COS INTERNALS

- EXEC
- SYSTEM TASK PROCESSOR
- Control STATEMENT PROCESSOR
- USER ⟷ COS INTERACTION
- DUMP ANALYSIS

SECTION 2

SYSTEM EXECUTIVE

EXEC FUNCTIONS

+ INTERRUPT HANDLING
+ PHYSICAL I/O
+ SYSTEM TASK SCHEDULING
+ EXECUTIVE REQUESTS
+ MEMORY ERROR CORRECTION
+ IDLE
+ RESOURCE ACCOUNTING
+ EXCHANGE MANAGEMENT

EXEC COMPONENTS

- EXCHANGE PROCESSOR
- INTERCHANGE
- INTERRUPT HANDLERS
- CHANNEL PROCESSORS
- I/O DRIVERS
  - DISK
  - SSD
  - FRONT END
  - I/O SUBSYSTEM
- EXEC ("MONITOR") REQUEST PROCESSOR
- TASK SCHEDULER
- MEMORY ERROR CORRECTION
- IDLE LOOP

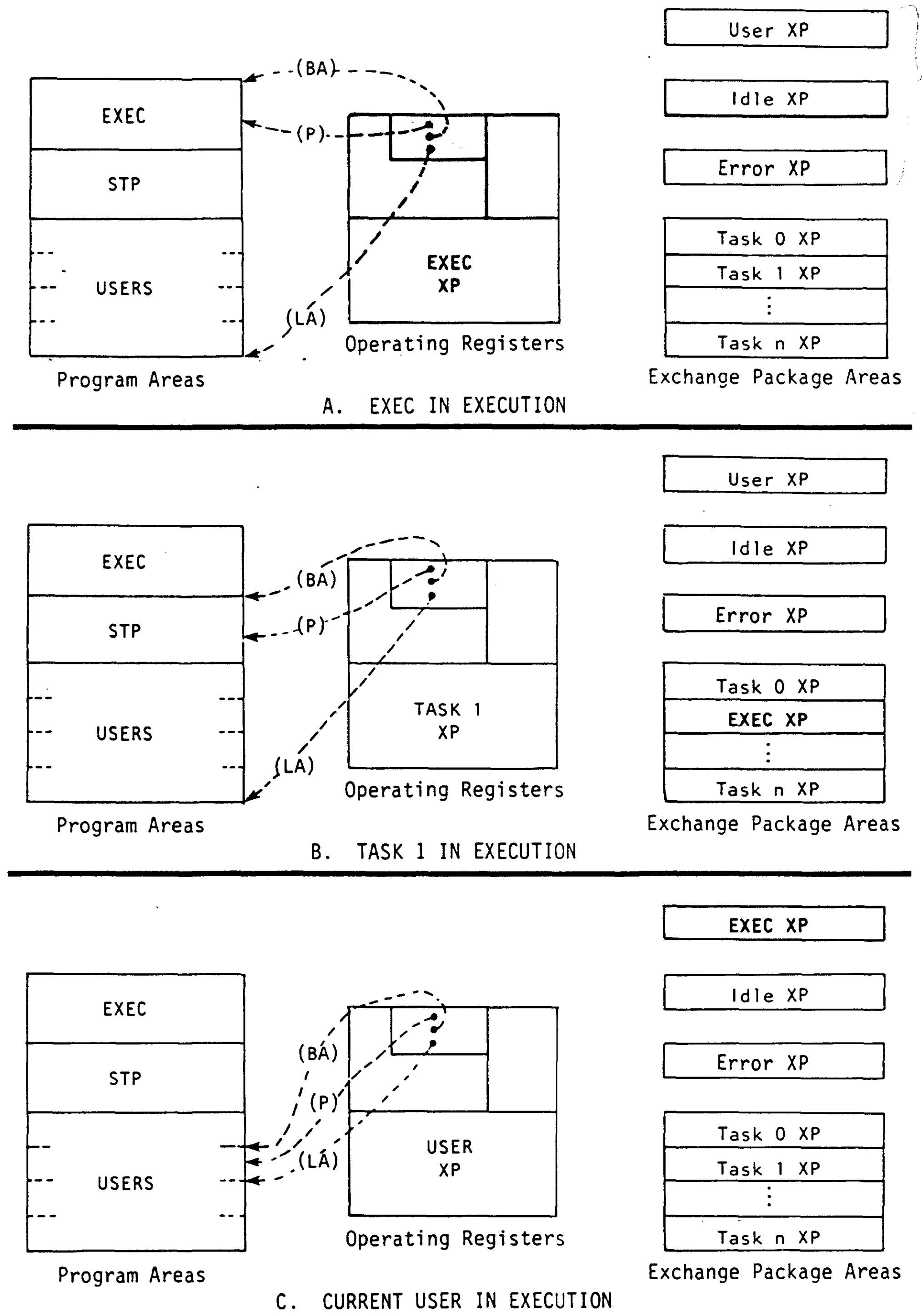


Figure 1-8. Exchange Package management

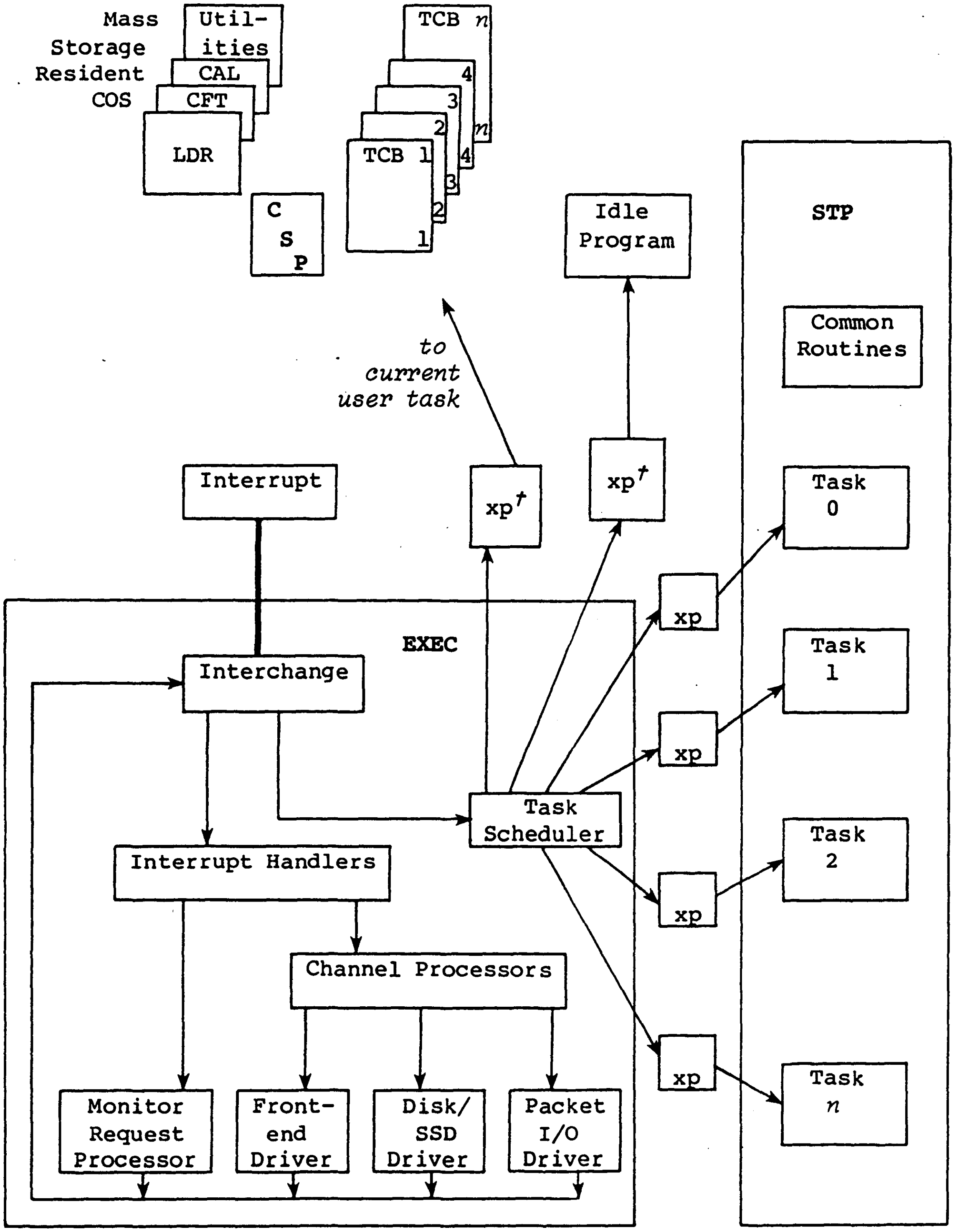
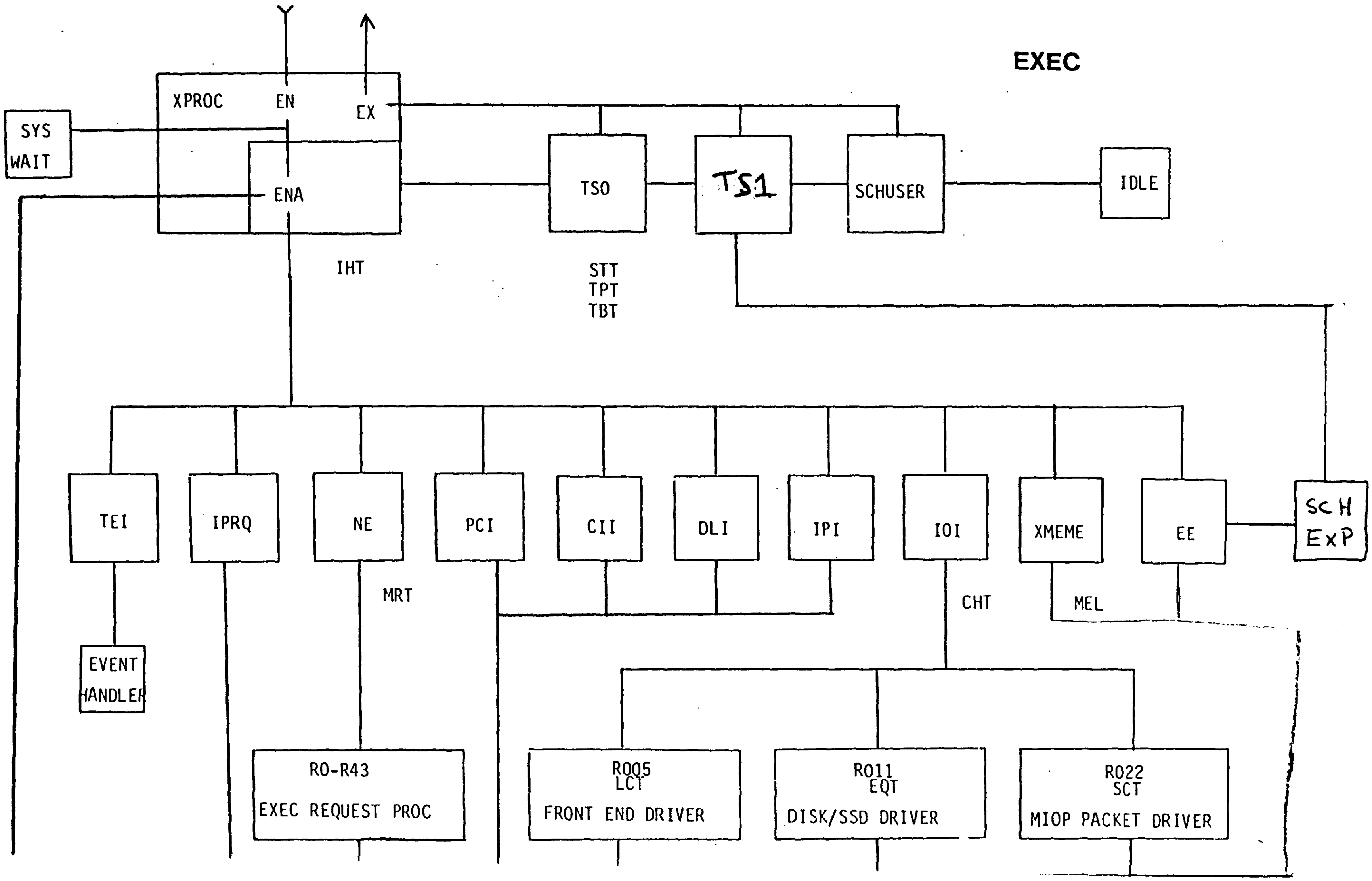


Figure 2-2. System control

† One Exchange Package per CPU



EXEC
XPROC
EN
EX
ENA
SYS WAIT
TSO
TS1
SCHUSER
IDLE
IHT
STT
TPT
TBT
TEI
IPRQ
NE
PCI
CII
DLI
IPI
IOI
XMEME
EE
ScH ExP
EVENT HANDLER
MRT
CHT
MEL
RO-R43
EXEC REQUEST PROC
R005
LCT
FRONT END DRIVER
R011
EQT
DISK/SSD DRIVER
R022
SCT
MIOP PACKET DRIVER

EXCHANGE PROCESSOR

FUNCTION: ENTRY AND EXIT ROUTINE FOR EXEC

ENTRY:

- ENTERED ON ANY EXCHANGE TO EXEC
- ASSURES THAT ONLY 1 CPU IS IN THE OPERATING SYSTEM
- UPDATES STATISTICS
- CHECKS FOR IOS REQ. HALT

EXIT:

- EXIT WHEN ALL EXEC WORK IS FINISHED
- CHECKS FOR I/O INTERRUPTS
- SETS XA
- EX

INTERCHANGE

INTERRUPT ANALYSIS

- ENTERED FROM EXCHANGE PRO ON AN EXCHANGE AND FROM INTERRUPT PROCESSI ROUTINES

- DETERMINES WHAT CAUSED THE EXCHANGE

- BRANCHES TO APPROPRIATE INTERRUPT HANDLER

EXEC - INTERRUPT HANDLERS

- IOI - I/O INTERRUPTS
- NE - NORMAL EXCHANGES
- CII - MCU INTERRUPTS
- PCI - PROGRAMMABLE CLOCK INT.
- TEI - TIMED EVENTS
- EE - ERROR EXCHANGE/INT.
- XMEME - MEMORY ERRORS
- IPREQST - INTER-PROCESSOR REQUESTS
- IPI - INTER-PROC NO-OPS
- DLI - DEADLOCK INTERRUPTS

Z	SCP	STG	DQM	PDM	JCM	JSH	EXP	TQM	MEP	MSG	SPM	OVM	FVD	DEC
AUT	-AUT	PDD	-DAT	DAT	-CSD	CSD	-CALL	CNT	-AEM	AUT	CSD	-ODT	EQT	EQT
CNT	-IBT	SDT	DCT	CSD	SDT	-JXT	(DDL)	-DEX		DSP	DCT	-OCS	DRT	
DAT	-LCT	-SST	DNT	(DNT)		-MST	-DNT	(DNT)		JTA		-OCT		
(DNT)	-LIT		-DRT	DRT		-RJI	-DSP	(DSP)		JXT	°MCT	-OLL		
DRT	-LXT		DSP	-DSC		SDT	JXT	-DUX		-LGJ	°STT			
-DSC	PDD		-EQT	DSP		-TXT	-LFT	-FSH		PDD	° IC			
(DSP)	-SDT		GRT	DXT		JTA	-ODN	GRT		SDT				
-DVL	-SST		JXT	EQT			(PDD)	JXT						
DXT			-RQT	JXT			QDT	-LDT						
-EFT			°SCT	PDD			SDT	-SM						
EQT				-PDI			-SWT	-TDT						
GRT				-PDS			TXB	-VAX						
JTA				-QDT			-UPT	-VUX						
JXT				SDT			-KTA	JTA						
(ODT)				-XAT			-TXT							
PDI														
QDT														
-RJI														
SDT														
TDT														

MIOP Driver	FED Driver	SSD/ DISK Driver	RO- R43	USER	CSP
°SCT	°CHT	-DCT	°MRT	(BAT)	(DSP)
°FIQ	°CXT	EQT	°MCT	(DDL)	
°FOQ	LIT	°CHT		(DSP)	(LFT)
°CHT	LXT	°STT		(DNT)	(DNT)
°CAT	LCT	°CAT		(JAC)	
°CLT	°CHT	°CLT		(JCB)	
°CIT	°CAT	°CIT		(LFT)	
	°CLT	°CBT		(ODN)	
	°CIT			(PDD)	
	°STT			(JTA)	
	AUT				
	IBT				

o Exec

( ) User

\- Task Controlled this table

EXECUTIVE REQUEST PROCESSOR
(MONITOR)

- PROCESSES REQUESTS FROM STP TASKS

- TO MAKE AN EXEC REQUEST, A TASK PUTS THE REQUEST INTO ITS S6 AND S7 REGISTERS AND DOES A NORMAL EXIT

- THE NORMAL EXIT INTERRUPT HANDLER DETECTS THAT IT WAS A TASK THAT DID THE NORMAL EXIT AND JUMPS TO THE EXECUTIVE REQUEST PROCESSOR

- S7 CONTAINS A FUNCTION CODE THAT IS USED TO INDEX INTO THE MONITOR REQ TABLE TO OBTAIN THE ADDRESS OF THE ROUTINE TO PROCESS THE REQUEST

- FOR A LIST OF EXECUTIVE REQUESTS SEE SM-0040, PAGE 2-16

EXEC - PHYSICAL I/O DRIVERS

- FRONT END DRIVER
- DISK/SSD DRIVER
- IOS PACKET DRIVER

TASK SCHEDULER

IF STRTS IS SET

THE TASK SCHEDULER FINDS THE HIGHEST PRIORITY TASK THAT IS READY TO EXECUTE AND SELECTS ITS EXCHANGE PACKAGE

ELSE

THE EXCHANGE PACKAGE OF THE CURRENTLY EXECUTING TASK IS SELECTED

IF NO TASK IS READY SELECT THE EXCHANGE PACKAGE OF THE CURRENT USER JOB

IF NO JOB IS READY SELECT THE EXCHANGE PACKAGE OF THE IDLE LOOP

STPLOCK - ALLOWS A TASK TO RUN IN NON PRE-EMPTIVE MODE

SYSTEM TASK TABLE

FUNCTION - FOR SCHEDULING AND CONTROLLING STP TASKS

STT HEADER

- STRTS BIT - REQUEST TASK SCHEDULER FLAG
- ACTIVE TASK ID
- ACTIVE TASK EXCHANGE PACKAGE ADDRESS
- ACTIVE TASK PARAMETER BLOCK ADDRESS

STT PART A - TPB's

- ONE ENTRY FOR EACH TASK
- READY BIT
- SUSPEND BIT
- TASK ID

STX

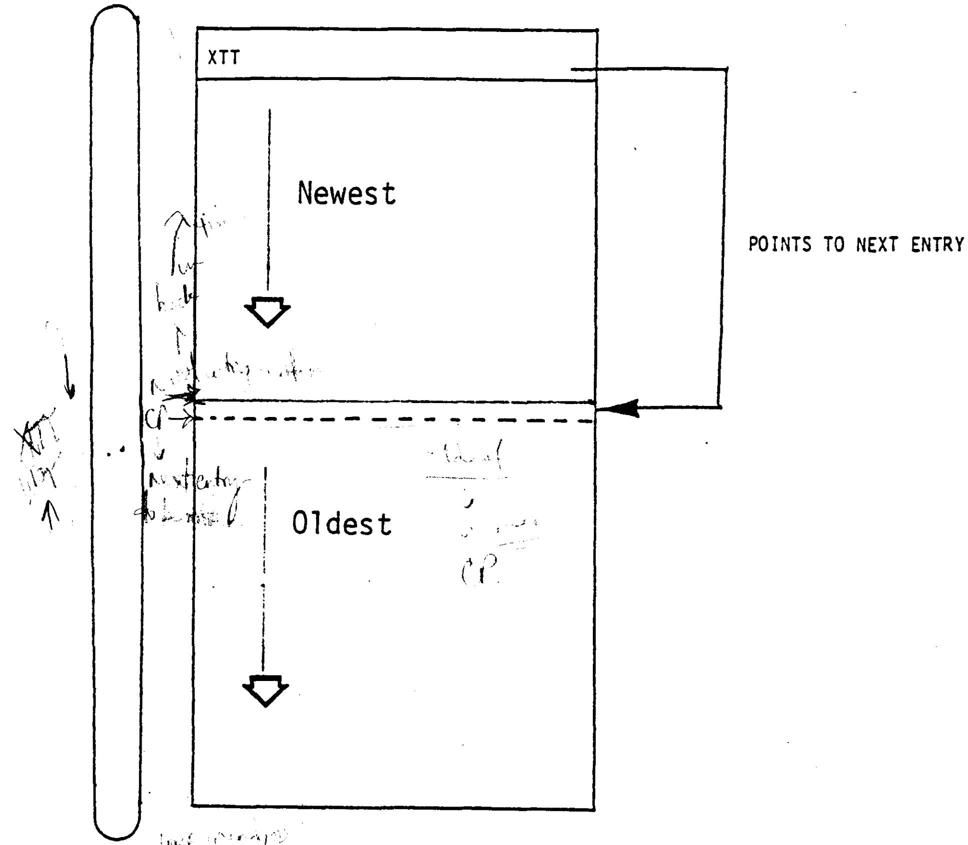
- ONE ENTRY FOR EACH TASK
- CONTAINS THE EXCHANGE PACKAGE FOR THE TASK
- LOCATED IN THE LOW MEMORY XP AREA

IDLE LOOP

-- EXECUTES WHEN THERE IS NOTHING ELSE TO DO

-- SCANS EXEC'S MEMORY IN INTERRUPTIBLE MODE - ATTEMPTS TO DETECT MEMORY ERRORS IN EXEC

HISTORY TRACE TABLE



1024 ENTRIES

```
DMEM,FWA=0,LWA=100000.              RAW DUMP                         FDUMP 1.13        05/08/84    16:38:50     PAGE
                                                                     SYSDUMP           04/20/84    15:13:59

0005750 0501032220000000000000 0541202060000000000000 0431052220000000000000 0515032200000000000000 PCI     XPC     FEI
0005754 0421112360000000000000 0445242320000000000000 0425052220000000000000 0431052360000000000000 DIO     ITM     EEI
0005760 0515052160000000000000 0515032220000000000000 0431052120000000000000 0515032360000000000000 SEG     SCI     FEE
0005764 0521232600000000000000 0451112500000000000000 0451222120000000000000 0451232060000000000000 TSX     JIT     JRE
0005770 0435052500000000000000 0461112040000000000000 0415202520000000000000 0451232200000000000000 GET     LIB     CPU
0005774 0515232100000000000000 0465052320000000000000 0465032520000000000000 0445202220000000000000 SSD     MEM     MCU
0006000 0421142220000000000000 0515312460000000000000 0471272500000000000000 0445202440000000000000 DLI     SYS     NWT
0006004 0445202020000000000000 0405232060000000000000 0465052060000000000000 0000000000000000000000 IPA     ASC     MEC
0006010 0364751723647510044111 0515242365113110052122 0405032122012420241114 0424401723647517236475 ===== HISTORY TRACE TABLE
0006014 0541242500000000000000 0002052743124445574460 0000000000000000000530 0000000000000000000000 XTT        ^2   O      X
0006020 0031250000362530005360 0036250600000000003050 0000000000000000161436 0000000000000001224020  U              (
0006024 0241250000362530005360 0036250600000000000273 0534000000000000026476 0540000000000000000005 (U              W    ->>
0006030 0041250000362530405360 0036250600000000000711 0000000000000025047472 0451232200000000000000  U                   TO:
0006034 0031250000353530605360 0035363620000000002426 0451232202652325251520 0000000000000000000003  U  ]    ^     JSH-SUSP
0006040 0041250001005254604400 0100622060000000001372 0000000000000025047472 0525232125100000161436  U  *    2C          TO:
0006044 0021250001005254604400 0100622060000000003211 0000000000000000000005 0000000000000001713500  U  *    2C
0006050 0041250000252543605200 0025254460000000001364 0000000000000025047472 0425302400000000000000  U  U    U           TO:E
0006054 0031250000231733005200 0027544620000000020073 0000000000000000000011 1000000000000000000014  U              ;
0006060 0121250000231733005200 0027544620000000001003 0026622040000000054005 0000000000000000000005  U                B   X
0006064 0421250000231733005200 0027544620000000000276 0511042622012325251440 0425302402647622451510 DU              RDY SUS
0006070 0041250000353531205360 0035363620000000000676 0000000000000025047472 0451232200000000000000  U  ]    ^           TO:
0006074 0031250000356470205360 0035641040000000004720 1000000000000000161436 0000000000000000000017  U       "
0006100 0041250000356470605360 0035641040000000001637 0000000000000025047472 0451232200000000000000  U       "           TO:
0006104 0031250000362530005360 0036250600000000002252 0000000000000000161436 0000000000000001224020  U
0006110 0241250000362530005360 0036250600000000000265 0540000000000000026476 0444000000000000000005 (U              X    ->
0006114 0041250000362530405360 0036250600000000000677 0000000000000025047472 0451232200000000000000  U                   TO:
0006120 0031250000232222605360 0035353640000000002323 0000000000000000000002 1000000000000000000002  U  K    ]     .
0006124 0121250000232222605360 0035353640000000000525 0000000000000000000000 0026622040000000054005  U  K    ]    U
0006130 0421250000232222605360 0035353640000000000334 0511052024213110020040 0451232202647621254120 DU  K    ]     READY
0006134 0041250000231733405200 0027544620000000000704 0000000000000025047472 0425302400000000000000  U                   TO:E
0006140 0031250000252543205200 0025254460000000005503 0425302402652325251520 0000000000000000000003  U  U    U    CEXP-SUSP
0006144 0041250000232223205360 0035353640000000001107 0000000000000025047472 0451232200000000000000  U  M    ]    G      TO:
0006150 0011250000353617005360 0521172400000000003661 0000100000447100103315 0414610004200100000000  U ^<   TOP        9   C
0006154 0041250000307440605420 0031427540000000003401 0000000000000025047472 0521212320000000000000  U  <                TO:T
0006160 0031250000307450605420 0030744220000000003230 0000000000000000335540 0000000000000000002022  U  <    <
0006164 0041250000307451205420 0030744220000000003517 0000000000000025047472 0521212320000000000000  U  <    <    O      TO:T
0006170 0031250000332770605420 0033276740000000063134 0000000000000000335540 0000000000000000001022  U            \
0006174 0041250000332771205420 0033276740000000005117 0000000000000025047472 0521212320000000000000  U            O      TO:T
0006200 0011250000332774405420 0521172400000000001225 0000110000447100102660 0001042063040100000000  U       TOP        9
0006204 0031250000332774405420 0521172400000000001251 0002052743124437654465 0000000000000000000006  U       TOP       ^2  Y5
0006210 0051250000332775005420 0000000000000000003203 0000000000004300117541 0521112324252621247124  U                   #   T
0006214 0041250000332775005420 0033276740000000001071 0000000000000025047472 0521212320000000000000  U  ,         9      TO:T
0006220 0031250000307440205420 0031427540000000004604 0002052743127546027037 0000000000000000000006  U  <                ^2  .
0006224 0041250000353617005360 0035360600000000001627 0000000000000025047472 0451232200000000000000  U ^<    ^           TO:
0006230 0031250000353530605360 0035363620000000003141 0451232202652325251520 0000000000000000000003  U  ]    ^     JSH-SUSP
0006234 0041250000017663204420 0000000000000000001165 0000000000000025047472 0445042304244010020040  U                   TO:
0006240 0011250000017664404420 0521172400000000260310 0000100000447100103315 0414610004200100000000  U       TOP.       9   C
006244 0041250000307440605420 0031427540000000003445 0000000000000025047472 0521212320000000000000  U  <          %      TO:T
0006250 0031250000307450605420 0030744220000000003332 0000000000000000335540 0000000000000000002022  U  <    <
0006254 0041250000307451205420 0030744220000000003570 0000000000000025047472 0521212320000000000000  U  <    <             TO:T
0006260 0031250000232222605420 0033176640000000033046 0000000000000000000002 1000000000000000000002  U  K          6&
0006264 0121250000232222605420 0033176640000000000535 0000000000614400000000 0000000000000000022001  U  K           ]
0006270 0421250000232222605420 0033176640000000000335 0511052024213110020040 0521212322647621254120 DU  K            READY  T
0006274 0041250000232223205420 0033176640000000000702 0000000000000025047472 0521212320000000000000  U  M                 TO:T
0006300 0031250000307440205420 0031427540000000004527 0002052743127546372041 0000000000000000000006  U  <          W  ^2  ]
0006304 0041250000252543605200 0025254460000000001572 0000000000000025047472 0425302400000000000000  U  U    U             TO:E
0006310 0031250000252543205200 0025254460000000010457 0425302402652325251520 0000000000000000000003  U  U    U    /EXP-SUSP
```

SECTION 3

COMMON SUBROUTINES

STP COMMON ROUTINES

1) Common routines are used by STP tasks to perform certain utility functions.

2) The common routine can be considered to be logically part of the task which is executing it (it uses the task's A- and S-registers).

3) Some common routines are re-entrant (more than one task may be executing the same common routine simultaneously).

RE-ENTRANCY CONSIDERATIONS

1) The task's A- and S-registers are preserved while executing common routine code (except output registers).

2) Local storage is not used (provided by the caller).

3) If global data must be changed, STP is LOCKED.

STP COMMON ROUTINES

MODULE	ENTRY POINTS	PURPOSE
STPUTIL	BTO, $OTB, $DTB, SFN, $NOCV	utility routines
STPDATS	GETDAT, RELDAT	DAT management
JMEM	JMEMAL, JMEMDE	JTA memory pool management
JTADNT	GETDNT, GETLFT, RELDNT	JTA DNT management
FIXJXPR	FIXJXO, FIXPRI	job pri. calculations
CRACKER	IND	JOB control stmt. cracker
GETPARM	GETPARM	parameter cracker
CONFIG	CONFIG	configuration changes

STP COMMON ROUTINES

MODULE	ENTRY POINTS	PURPOSE
ERROR	ERROR0, ERROR1	hang the system
REQRPLY	TSKREQ, PUTREQ, GETREQ, PUTREPLY, REPLIES, GETREPLY	task-to-task communications
STPMEM	MEMAL, MEMDE, PMEMDE, SSLDE	memory pool management
CHAINS	CHAIN, CHAINF, UNCHAIN, JCHAIN, JCHAINF, JUNCHAIN	chain management
STPTIME	RQST2, RT2JD, JD2RT	date/time calculations
QUEUES	DQSD2, EQSD2	SDT queue management
QMSG	NXTMSG, FREEMSG, ENQMSG	interactive station message management
MSGQUE	MSGQUE	SCP/operator message processing

MEMORY POOLS

- memory pools provide temporary data areas for tasks
- memory is allocated from a pool when needed and returned when the task is finished with it
- memory areas are variable-sized
- currently, 4 memory pools are defined: * 3 IN 1.13

POOL 1 - miscellaneous

POOL 2 - task to task communication modules (CMODS)

POOL 3 - TQM storage

POOL 4 - OVM storage — ELIMINATED IN 1.13

POOL TABLE

Header:

Word	0–15	16–39	40–57	58–63
0	//////	//////	//////	MAX

Figure 1.PT-1. Pool Table (PT) header

Field	Word	Bits	Description
PTMAX	0	58-63	Maximum valid memory pool number in system

mem pools

Entry:

Word	0–15	16–39	40–63
1	//////	SIZE	BASE
	//////		
	//////		
.	//////		
.	//////		
.	//////		
	//////		
n	//////		

Figure 1.PT-2. Pool Table (PT)

Field	Word	Bits	Description
PTSIZE	1- *n*	16-39	Size of the memory pool
PTBASE	1- *n*	40-63	Base address of the memory pool

MEMORY POOL

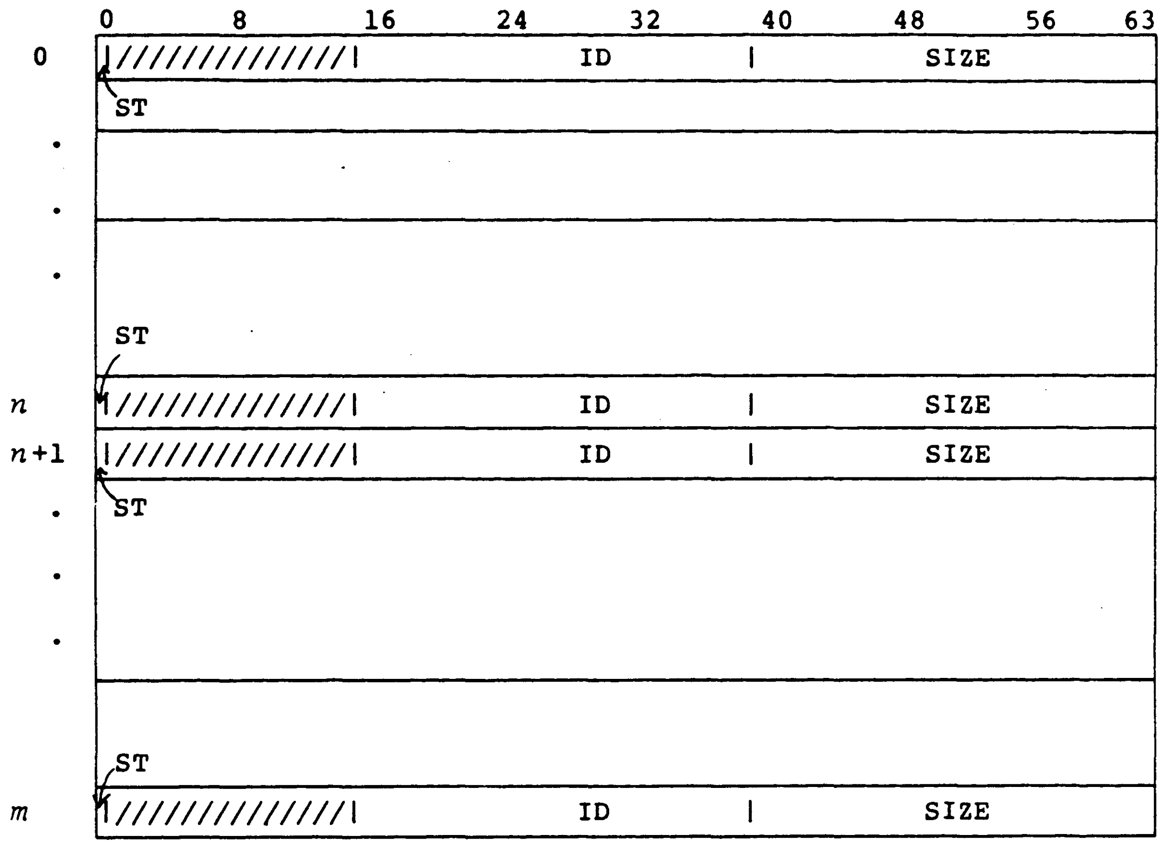


Figure 1.MP-1. Memory Pool

Field	Word	Bits	Description
MPST	0,*n*,etc.	0	Status of the memory area: 0 Available 1 In use
MPID	0,*n*,etc.	16-39	Memory pool identification: 01010101_8 Pool 1 $0x0x0x0x_8$ Pool x. Current values are 1, 2, 3, or 7.
MPSIZE	0,*n*,etc.	40-63	Size of the memory area

MEMAL – memory allocation

- example: allocate memory from the TXTPOOL (POOL 1)

```
           A7          S1              NUMBER OF TEXT BLOCKS
           A6          LE@DAT-L@DATPH  LENGTH IN WORDS OF EACH TEXT BLOCK
           A7          A7*A6
           A6          TXTPOOL
           R           MEMAL
           A0          A5
           ERRAN
ZMOVTXT2   =           *
           Z1TXTFST,0  A7              SAVE FWA OF TEXT BLOCK IN POOL
```

- inputs –
  - (A6) is pool number
  - (A7) is number of words to allocate

- outputs –
  - (A6) is return status:
    - 0 – OK
    - 1 – invalid pool number
    - 2 – invalid word count
    - 3 – memory not available
  - (A7) is fwa of area allocated if (A6) is 0

- allocated memory is zeroed for the caller

MEMDE - memory deallocation

- example: deallocate memory from TXTPOOL

```
A7        ZTXTFST,0       FWA OF TEXT
A6        TXTPOOL
R         MEMDE           REMOVE THE RESERVATION ON THAT MEMORY
A0        A5
ERRAN
```

- inputs -

  (A6) is pool number

  (A7) is fwa to deallocate

- outputs -

  (A6) is return status:

  0 - OK

  1 - invalid fwa

  2 - area not allocated

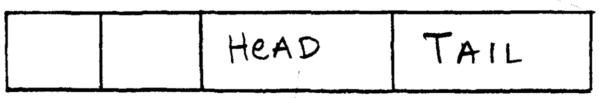
  3 - invalid pool number

  (A7) is fwa of memory deallocated if (A6) is 0

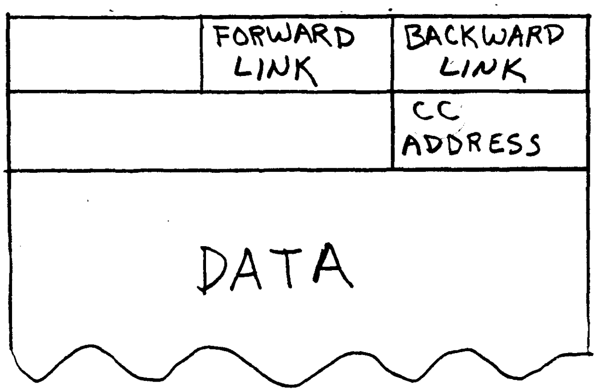
ITEM CHAINING/UNCHAINING

- PROVIDES MEANS FOR TASKS TO LINK DATA
- AMOUNT OF DATA TO LINK IS DEFINED BY THE TASKS
- MAY BE USED TO LINK REGISTER DATA OR POOL DATA
- DATA IS CONSIDERED AN ITEM

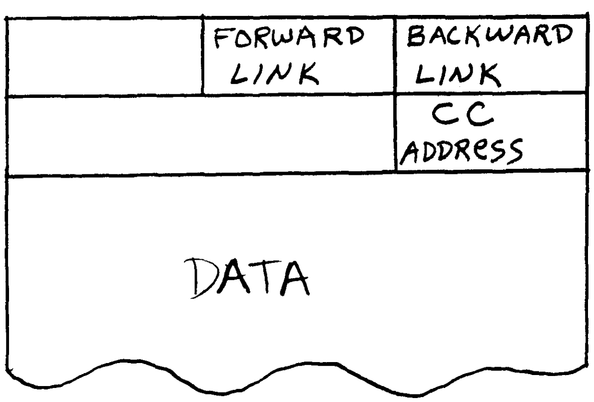
CHAINS



CHAIN CONTROL WORD



CHAIN ITEM



CHAIN ITEM

STP COMMON ROUTINES

- CHAIN/CHAINF PLACE AN ITEM ON A CHAIN. CHAIN WILL PLACE AN ITEM ON THE END WHEREAS CHAINF WILL PLACE AN ITEM ON THE FRONT OF A CHAIN. CHAIN/CHAINF ARE CALLED VIA A RETURN JUMP WITH THE CALLER PROVIDING THE FOLLOWING:

INPUT REGISTERS:	(A6) = Address of chain control word
	(A7) = Address of the item to be chained
OUTPUT REGISTERS:	(A6) = Unchanged from input
	(A7) = Unchanged from input

CHAIN → queue (FIFO)

CHAINF → stack (LIFO)

- UNCHAIN REMOVES AN ITEM FROM ANYWHERE ON THE CHAIN. THE CALLER MUST UPDATE THE COUNT OF THE NUMBER OF ITEMS REMAINING ON THE CHAIN. UNCHAIN IS CALLED VIA A RETURN JUMP WITH THE CALLER PROVIDING THE FOLLOWING:

INPUT REGISTER:	(A7) = Address of item to be unchained
OUTPUT REGISTER:	(A7) = Unchanged from input

EQSD2 – enqueue SDT entry

- re–entrant common routine
- entry parameter –

  S6: 1/EQSEQ, 15/–, 24/EQSQH, 24/EQSEA

  EQSEQ: 0 – FIFO enqueuing
  1 – priority enqueuing

  EQSQH: SDT queue header address

  EQSEA: SDT entry address

- returns to (B0) plus 2 if no error, else to (B0) with (A0) error status
- priority enqueuing:
  1. job class rank
  2. job priority
  3. time of job submission

DQSD2 - dequeue SDT entry

re-entrant common routine

entry parameter -

S6: 1/DQSDQ, 15/-, 24/DQSQH, 24/DQSEA

DQSDQ: 0 - FIFO dequeuing
1 - entry dequeuing

DQSQH: SDT queue header address

DQSEA: SDT entry address (for entry dequeuing)

returns to (B0) plus 2 with (S6) SDT entry address if FIFO dequeuing

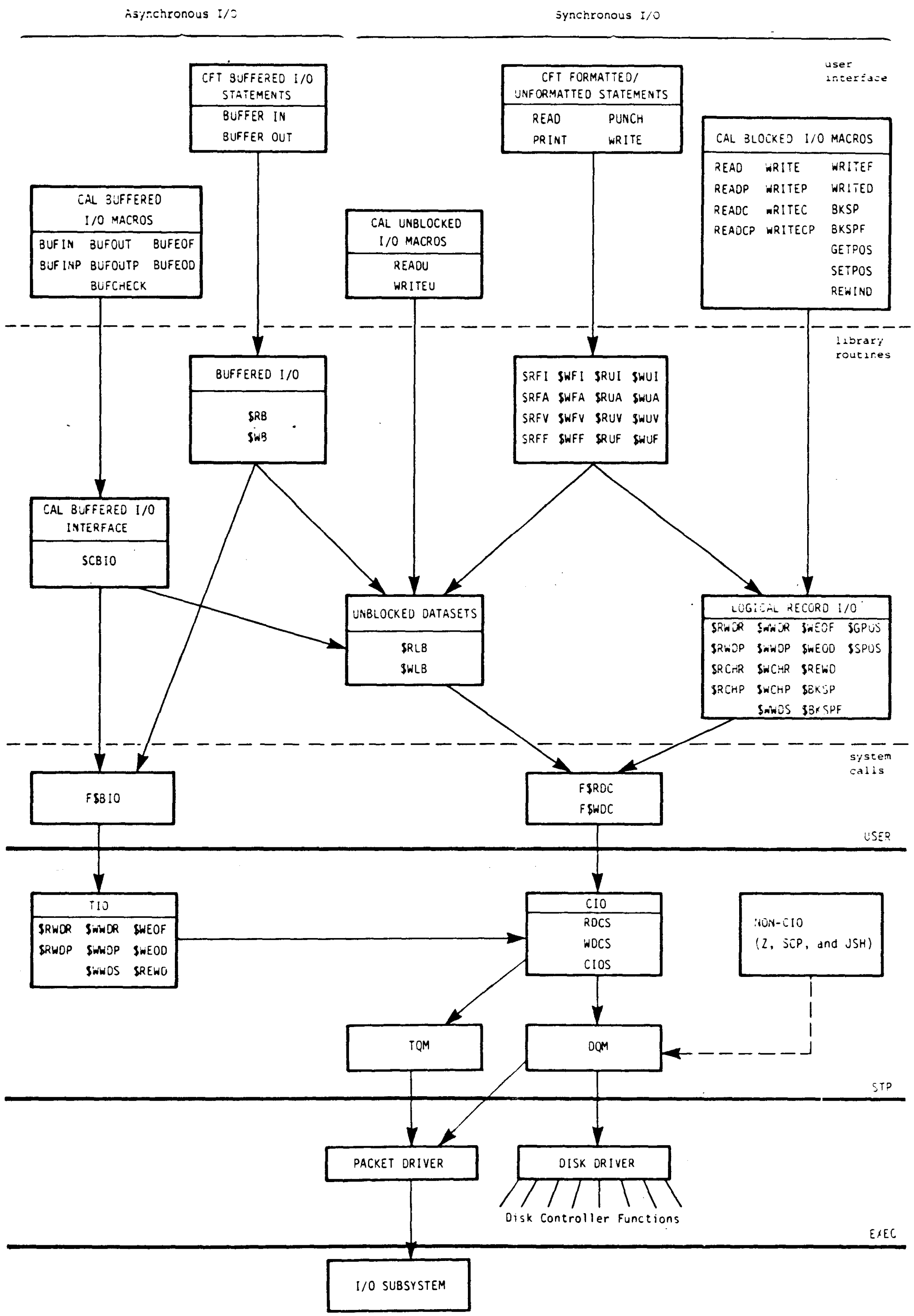
error return to (B0) with (S6) error status

SDT queue manipulation

- **Example: Move SDT entry from INPUT queue to EXECUTE queue**

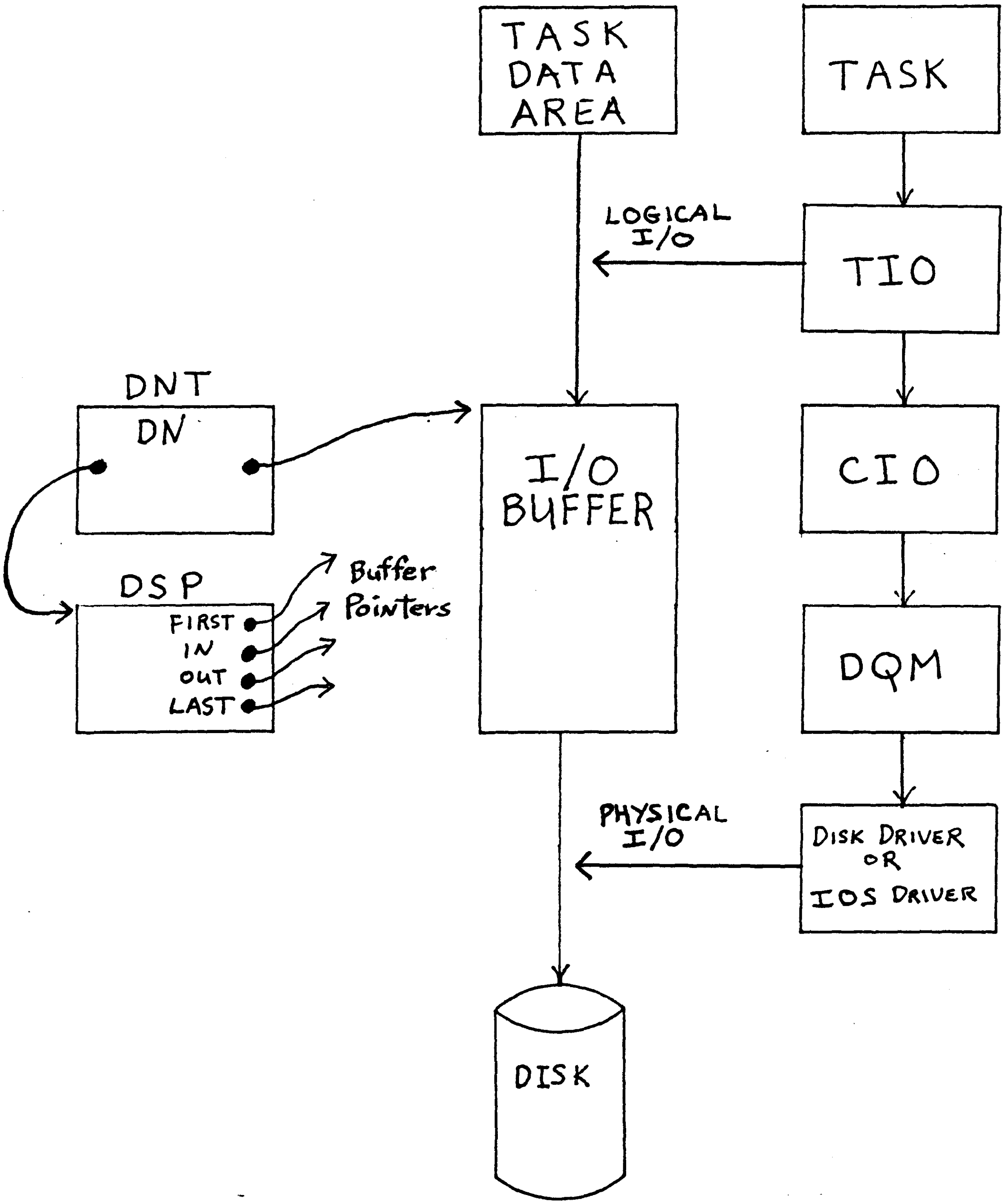
```
S6      Q@INPUT
S7      RJSDT,0
S6      S5<D'24
S6      S5!S7
S6      S5!SB
R       DQSD2           DEQUEUE SDT ENTRY
R       ERROR0
S6      Q@EXCUTE
S7      RJSDT,0
S6      S5<D'24
S6      S5!S7
S6      S5!SB           PRIORITY ENQUEUE
R       EQSD2           ENQUEUE SDT ENTRY
R       ERROR0
```

calls to DQSD2 (...) EQSD2 (en ...)



Overview of COS I/O

I/O OVERVIEW


TASK DATA AREA
TASK
LOGICAL I/O
TIO
DNT
DN
I/O BUFFER
CIO
DSP
FIRST
IN
OUT
LAST
Buffer Pointers
DQM
PHYSICAL I/O
DISK DRIVER OR IOS DRIVER
DISK

TASK LOGICAL I/O (TIO)

- ALLOWS A SYSTEM PROGRAMMER TO DO LOGICAL I/O AT THE TASK LEVEL.

- TIO ROUTINES ARE:
  - $RWDP/$RWDR-READ WORDS PARTIAL/FULL RECORD
  - $WWDP/$WWDR-WRITE WORDS PARTIAL/FULL RECORD
  - $WEOF-WRITE END OF FILE
  - $WEOD-WRITE END OF DATA
  - $REWD-REWIND A DATASET
  - $WWDS-WRITE WORDS--UNUSED BIT COUNT

- TASKS CALL TIO BY PLACING REQUIRED PARAMETERS IN 'A' REGISTERS AND EXECUTING A RETURN JUMP TO THE ROUTINE.

CIRCULAR I/O

- PERFORMS PHYSICAL I/O ON A DATASET

  ACCESSIBLE TO TASKS THROUGH TIO AND DIRECT CALLS.

  CIO ROUTINES ARE:

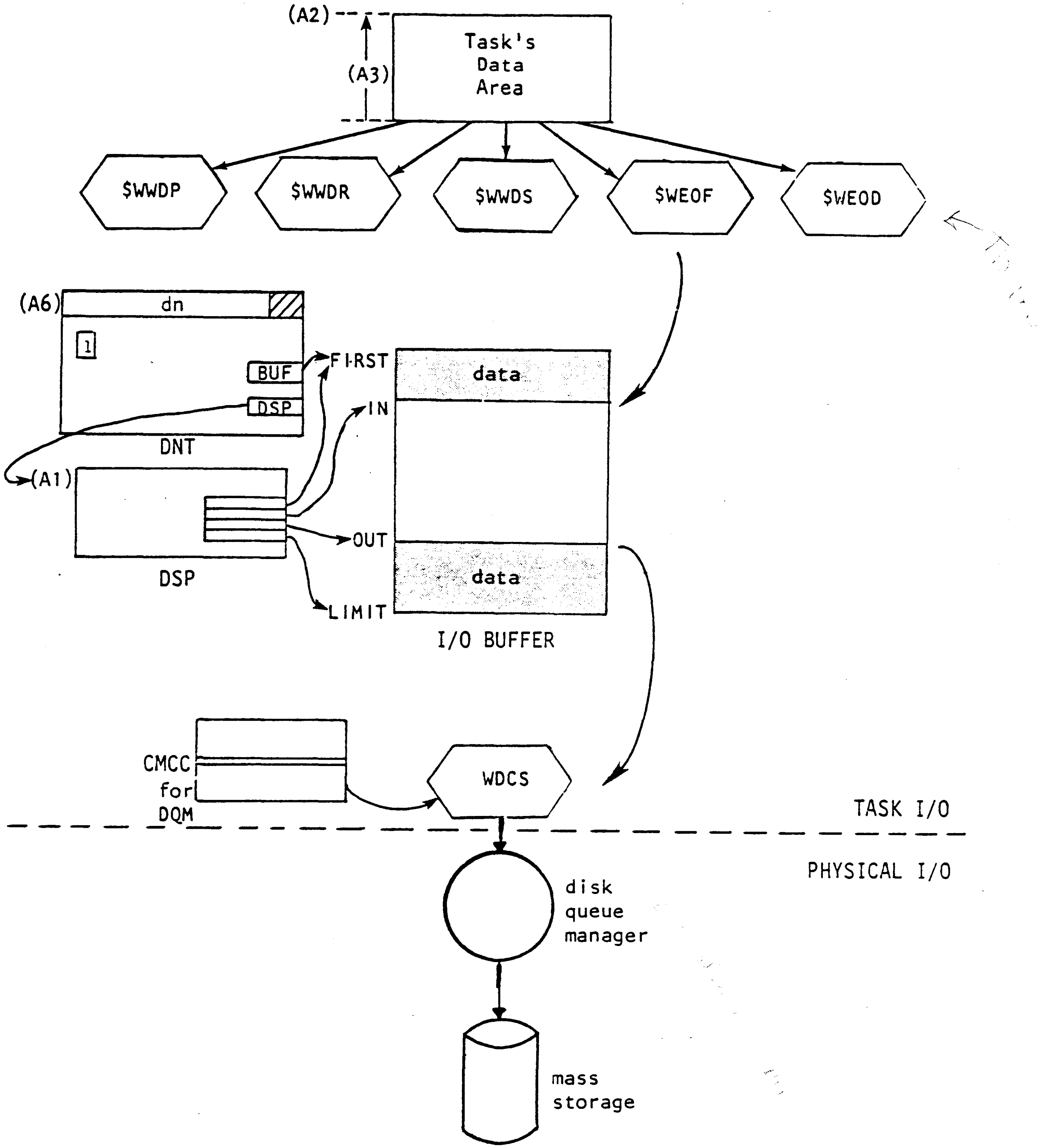
  RDCS-READ CIRCULAR REQUEST

  WDCS-WRITE CIRCULAR REQUEST

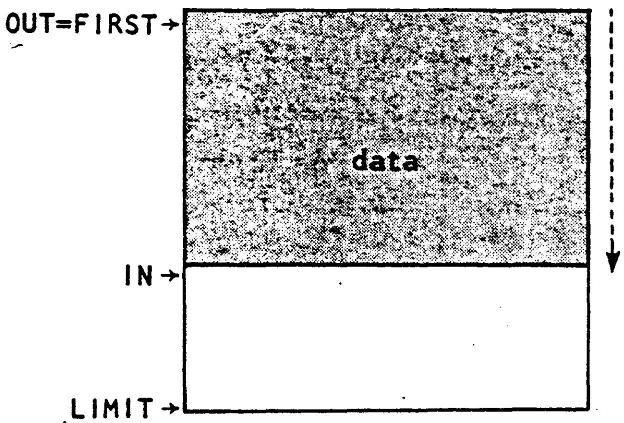
- TASKS CALL CIO BY PLACING REQUIRED PARAMETERS IN 'A' REGISTERS AND EXECUTING A RETURN JUMP TO THE ROUTINE.

- CIO READS/WRITES 512 WORD BLOCKS. THE CALLER HAS THE RESPONSIBILITY OF MAINTAINING THE BUFFER IN/OUT POINTER IN THE DSP. AS SHOWN IN THE PREVIOUS $WWD FLOW DIAGRAM.

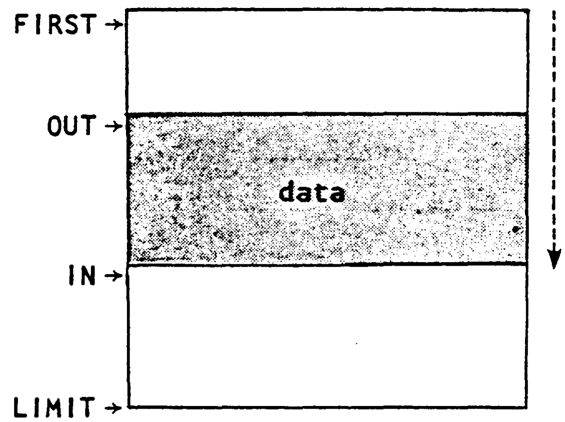
- THE CALLER SENSES COMPLETION OF PHYSICAL I/O BY CALLING GETREPLY. IF A REPLY IS FOUND THE CALLER SHOULD CALL ROUTINE REPCIO WITH S1 AND S2 INTACT FROM GETREPLY.



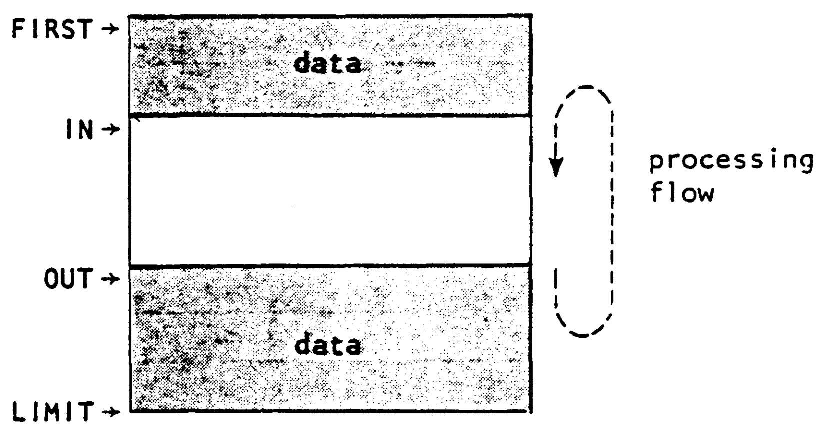
TIO logical write



A. Filling the buffer



B. Emptying the buffer

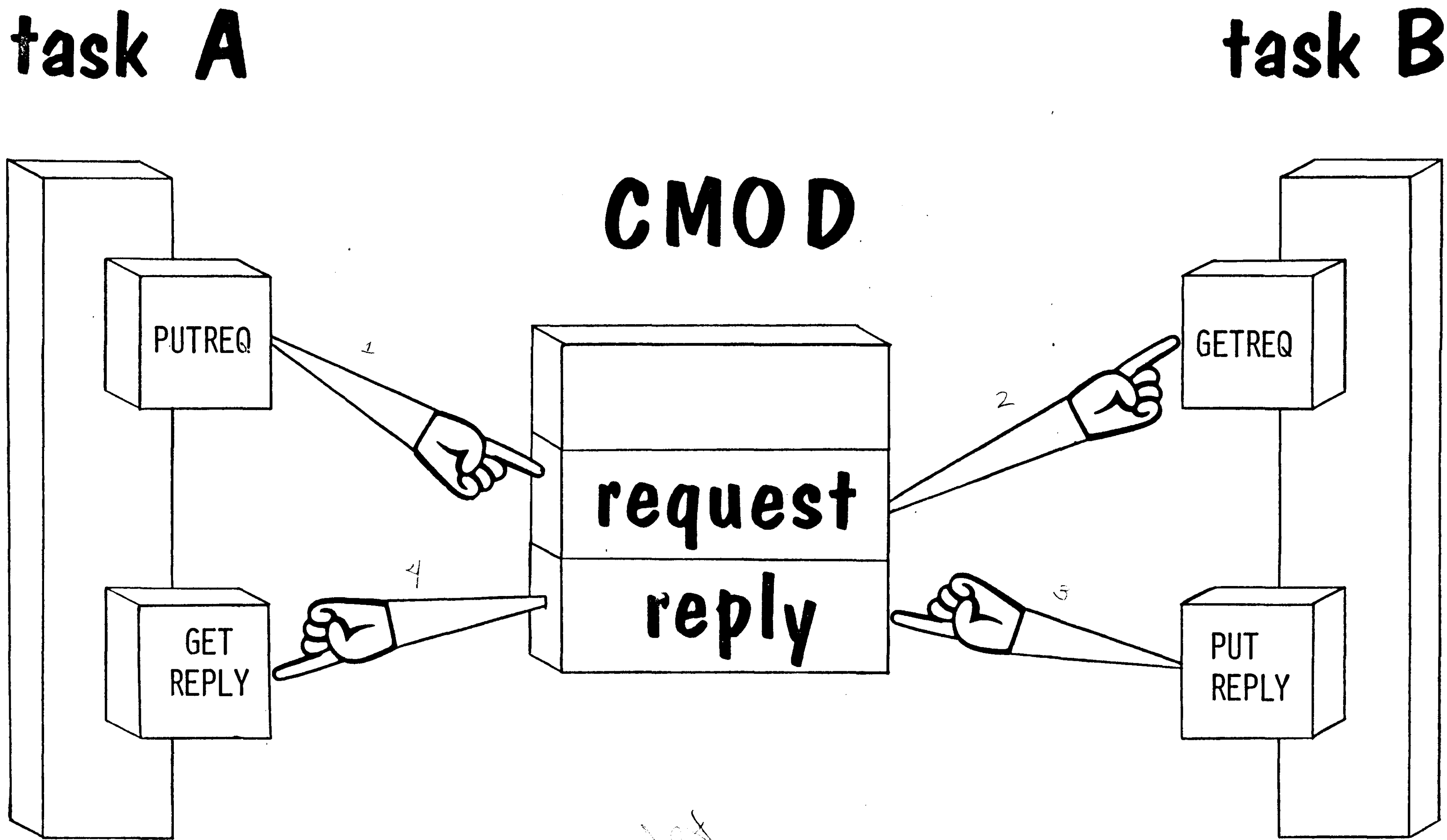


C. Concurrently filling and emptying the buffer

Concurrent Logical & Physical I/O (i.e. CIO)

SECTION 4

TASK TO TASK COMMUNICATIONS



task A
task B
CMOD
PUTREQ
GETREQ
request
reply
GET REPLY
PUT REPLY
1
2
4
5

4.1

TASK TO TASK COMMUNICATION

- THERE ARE 2 AREAS FOR INTERTASK COMMUNICATION

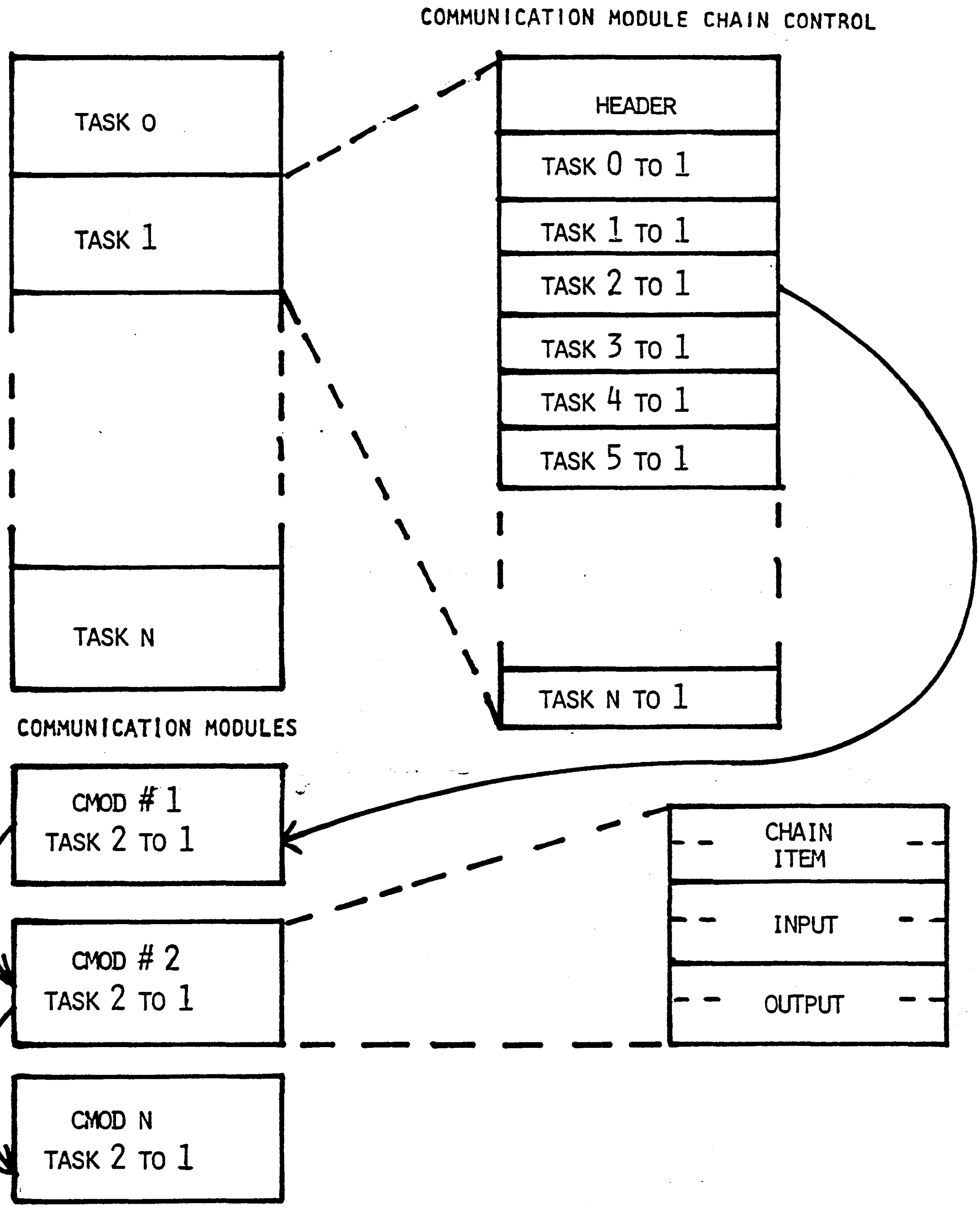
1. COMMUNICATION MODULE CHAIN CONTROL (CMCC).

   CONTIGUOUS AREA
   ENTRY FOR EACH POSSIBLE TASK COMBINATION
   ARRANGED IN TASK NUMBER SEQUENCE
   POINT TO THE COMMUNICATION MODULES (CMOD's)

2. COMMUNICATION MODULE (CMOD) → 6 wds long

   ALLOCATED AS NEEDED FROM A POOL
   ALL TASK REQUESTS ARE THROUGH A CMOD.
   ALL TASK REPLIES ARE THROUGH A CMOD.
   - 2 WORDS FOR SYSTEM (Chain) CONTROL
   - 2 WORDS AS TASK INPUT REGISTERS
   - 2 WORDS AS TASK OUTPUT REGISTERS

- TASKS PLACE REQUESTS IN THE INPUT WORDS OF A CMOD.
- TASKS RECEIVE REPLIES IN THE OUTPUT WORDS OF THEIR CMOD
- FORMAT OF A REQUEST IS DEFINED BY THE CALLED TASK


COMMUNICATION MODULE CHAIN CONTROL
TASK 0
TASK 1
TASK N
HEADER
TASK 0 TO 1
TASK 1 TO 1
TASK 2 TO 1
TASK 3 TO 1
TASK 4 TO 1
TASK 5 TO 1
TASK N TO 1
COMMUNICATION MODULES
CMOD # 1
TASK 2 TO 1
CMOD # 2
TASK 2 TO 1
CMOD N
TASK 2 TO 1
CHAIN ITEM
INPUT
OUTPUT

CMCC HEADER

	0	8	16	24	32	40	48	56	63
0	TM	TL	NOT USED						

Figure 1.CC-2. Chain Control Word header format

Field	Bits	Description
CCTM	0-7	Maximum number of items to be queued to a particular task†
CCTL	8-15	Number of items queued to a particular task†

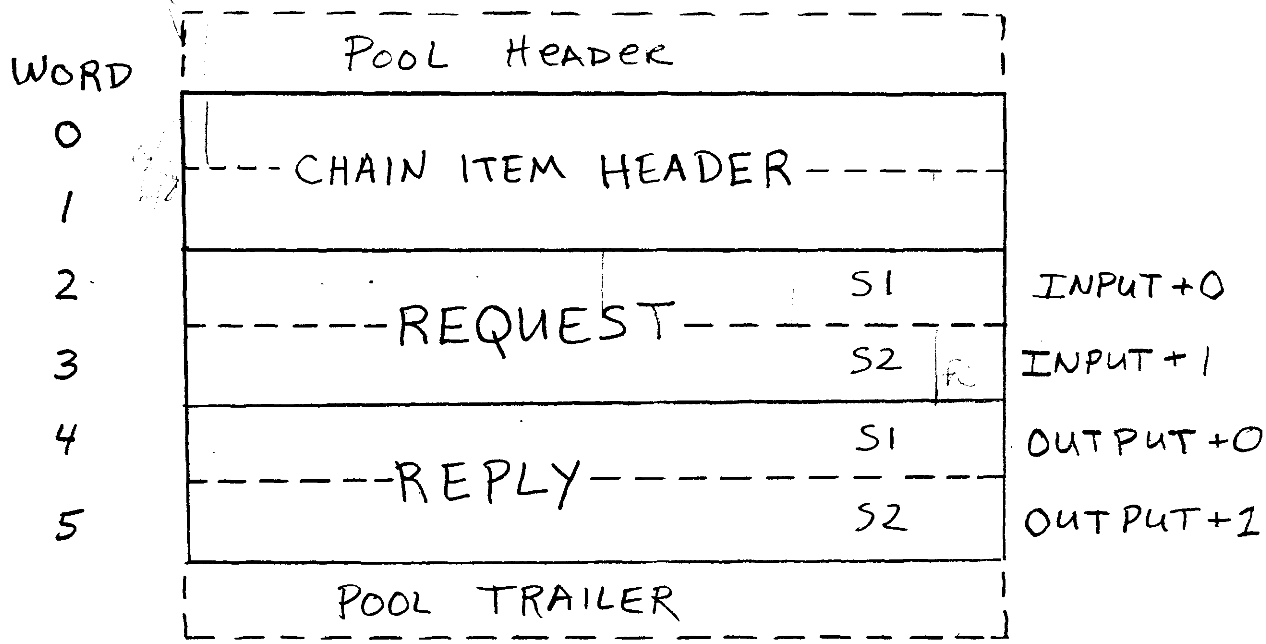
CMCC CHAIN CONTROL WORD

	0	8	16	24	32	40	48	56	63
0	QM	QL	HEAD			TAIL			

Figure 1.CC-3. Chain Control Word entry format

Field	Bits	Description
CCQM	0-7	Maximum number of items to be queued from one task to another† ≤ TM)
CCQL	8-15	Number of items currently queued from one task to another†
CCHEAD	16-39	Address of first item on the chain
CCTAIL	40-63	Address of last item on the chain

CMOD


POOL HEADER
WORD
0
1
2
3
4
5
CHAIN ITEM HEADER
REQUEST
S1
S2
INPUT+0
INPUT+1
REPLY
S1
S2
OUTPUT+0
OUTPUT+1
POOL TRAILER

° A TASK CALLS EXEC TO ACTIVATE ANOTHER TASK

° THE TASK SCHEDULER IN EXEC EXAMINES THE SYSTEM TASK TABLE TO DETERMINE THE HIGHEST PRIORITY TASK READY TO EXECUTE.

° THE RE-ENTRANT ROUTINES:

PUTREQ GETREQ PUTREPLY GETREPLY	ASYNCHRONOUS	* REPLIES
TASKREQ —	SYNCHRONOUS	

ARE USED FOR INTERTASK COMMUNICATION

° THE REQUEST FOR INTERTASK COMMUNICATION IS PASSED IN REGISTERS S1 AND S2

- PUTREQ PLACES THE REQUEST IN THE INPUT REGISTERS OF A CMOD AND LINKS THE CMOD TO THE APPROPRIATE CMCC. PUTREQ IS CALLED VIA A RETURN JUMP WITH THE CALLER PROVIDING THE FOLLOWING:

INPUT REGISTERS:	(A1) =	"Throw-away" indicator. If (A1) is positive, control is not returned to caller until request is queued. If (A1) is negative, control returns with no action taken if the request cannot be queued without suspending the caller.
	(A2) =	Requested task's ID
	(S1) =	INPUT+0 } request
	(S2) =	INPUT+1 } request
OUTPUT REGISTERS:	None	

PUTREQ

-- ALLOCATES A CMOD

-- PUTS REQUEST (S1 AND S2) IN CMOD

-- LINKS CMOD TO CMCC

-- INCREMENTS COUNTS IN HEADER

-- MAKES AN EXECUTIVE REQUEST TO READY THE REQUESTED TASK

- GETREQ SEARCHES FOR AN ACTIVE REQUEST FOR THE ~~CALLER~~ CALLEE. GETREQ IS CALLED VIA A RETURN JUMP AND REPLIES WITH THE FOLLOWING:

INPUT REGISTERS: None

OUTPUT REGISTERS:

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

(A2) = ID of task that generated the request.

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

GETREQ

-- SEARCHES EACH CMCC FOR A REQUEST

-- SETS EXECUTING BIT IN CMOD

-- GIVES THE REQUEST FROM THE CMOD TO THE TASK IN S1 AND S2

PUTREPLY

PUTREPLY PLACES THE REPLY IN THE OUTPUT REGISTERS OF A CMOD. PUTREPLY IS CALLED VIA A RETURN JUMP WITH THE CALLER PROVIDING THE FOLLOWING:

INPUT REGISTERS: (A2) = ID OF TASK TO RECEIVE THE REPLY

(S1) = OUTPUT+0
REPLY
(S2) = OUTPUT+1

OUTPUT REGISTERS: NONE

PUTREPLY

-- THE REPLY GOES ON THE SAME CHAIN AS THE REQUEST

-- PUTREPLY LOOKS FOR THE FIRST AVAILABLE CMOD ON THE CHAIN

-- THE REPLY (S1 AND S2) IS PUT INTO THE CMOD

-- COUNTS ARE DECREMENTED

-- AN EXEC REQUEST IS MADE TO READY THE TASK THAT IS TO RECEIVE THE REPLY

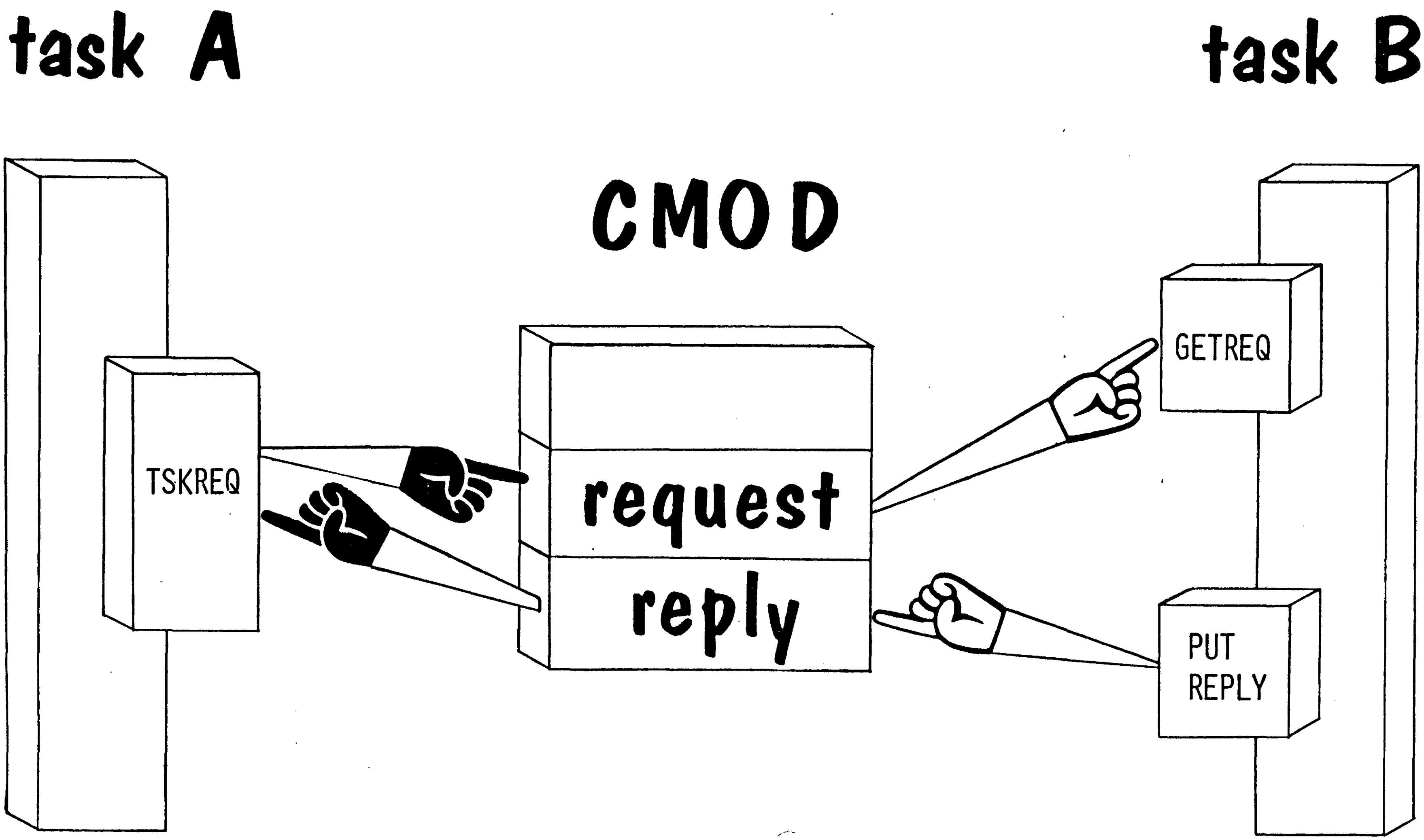
- GETREPLY SEARCHES FOR A REPLY TO THE CALLING TASK. GETREPLY ALSO RELEASES THE APPROPRIATE CMOD WHEN A REPLY IS FOUND. GETREPLY IS CALLED VIA A RETURN JUMP AND REPLIES WITH THE FOLLOWING:

INPUT REGISTERS:	None
OUTPUT REGISTERS:	(A0) = Find indicator. If (A0) = 0, no reply was located; if (A0) ≠ 0, a reply is being returned to the caller.
	(A2) = ID of replying task
	(S1) = OUTPUT+0 } Reply
	(S2) = OUTPUT+1 } Reply

GETREPLY

-- THE REPLY FROM THE CMOD IS PLACED INTO S1 AND S2

-- THE CMOD IS UNCHAINED AND DEALLOCATED.


task A
task B
CMOD
TSKREQ
request
reply
GETREQ
PUT
REPLY

TSKREQ

-- SYNCHRONOUS EQUIVALENT OF PUTREQ AND GETREPLY

-- ALLOCATES A CMOD

-- PUTS S1 AND S2 INTO CMOD

-- ACTIVATES REQUESTED TASK AND SELF SUSPENDS

-- AWAKENED BY REPLYING ROUTINE

PUTREPLY

- TSKREQ QUEUES A REQUEST TO ANOTHER TASK.
- TSKREQ IS CALLED VIA A RETURN JUMP WITH THE CALLER PROVIDING THE FOLLOWING:

INPUT REGISTERS:	(A2) = ID OF REQUESTED TASK	
	(S1) = INPUT+0	} REQUEST
	(S2) = INPUT+1	
OUTPUT REGISTERS:	(S1) = OUTPUT+0	} REPLY
	(S2) = OUTPUT+1	

- ONCE THE REQUEST HAS BEEN PROCESSED, THE CALLER MAY EXAMINE ITS S1,S2 REGISTERS FOR A REPLY. CONVENTIONALLY, S1=ZERO WHEN THERE IS NO ERROR, OTHERWISE S1=ERR CODE. S2=THE CALLING TASKS INPUT+0 REGISTER (S1) INFORMATION.

REPLIES

Reply to a non-specific req.

(Multiple replies to a non-specific req.)

-- QUEUES A REPLY FOR WHICH NO REQUEST WAS MADE

-- USED BY DQM ONLY

-- ALLOCATES A CMOD

-- SETS EXECUTING BIT SO IT IS NOT TAKEN AS A REQUEST

-- PUTS REPLY ON BEGINNING OF CHAIN (CHAIN F)

SECTION 5

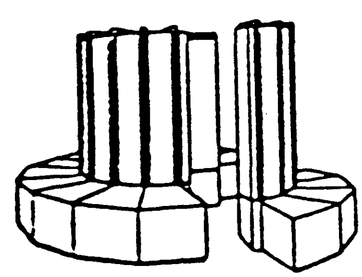
SYSTEM TASK PROCESSOR - TASKS

SYSTEM TASKS

A system task is a COS system program which performs one or more specific functions

Tasks have the following characteristics:

- Tasks are memory-resident following EXEC
- Each task is a separate program module and has it's own XP in EXEC
- BA is the end of EXEC, LA is the end of machine's memory
- Tasks operate in user mode
- Each task has a priority (0-[redacted] 77 octal)
- Each task has a unique ID (0-[redacted] ~~octal~~) 13



1.13 STP TASK IDs and Priorities - (defined in startup)

```
SCP     ITASK    ID=D'01,PRI=0'10,PREG=SCPINIT
EXP     ITASK    ID=D'02,PRI=0'12,PREG=EPTK
PDM     ITASK    ID=D'03,PRI=0'14,PREG=PDMGR
DEC     ITASK    ID=D'04,PRI=0'20,PREG=DEC
DQM     ITASK    ID=D'05,PRI=0'02,PREG=DIS
MSG     ITASK    ID=D'06,PRI=0'04,PREG=LOGINIT
MEP     ITASK    ID=D'07,PRI=0'05,PREG=MEP
SPM     ITASK    ID=D'08,PRI=0'24,PREG=SPM
JSH     ITASK    ID=D'09,PRI=0'13,PREG=JSH
JCM     ITASK    ID=D'10,PRI=0'11,PREG=JCM
TQM     ITASK    ID=D'11,PRI=0'03,PREG=TQM
STG     ITASK    ID=D'12,PRI=0'06,PREG=STG
FVD     ITASK    ID=D'13,PRI=0'15,PREG=FVD
```

STARTUP ID=0, PRI=0'77

TASK STATES

SUSPENDED - not ready to execute

READY - ready to execute

waiting - waiting for CPU

running - actually executing

Each task's state is known to EXEC, but not to individual tasks

A task is READIED (moved from SUSPENDED to READY state) by EXEC. This can occur 2 ways:

1) EXEC readies tasks based on certain events

2) One task can request that another task be readied

A task is suspended by EXEC request. One task may not suspend another task, only itself.

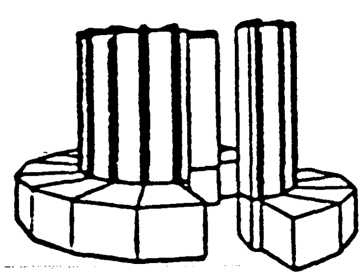
TASK PREEMPTION

Tasks are preemptable

Task preemption can occur anytime EXEC executes

Exceptions:

- **A task may become temporarily non-preemptable**
- **Task breakpointing**



TASK CREATION

A task may create another task with an EXEC request

The STARTUP task is responsible for creating the other system tasks

The created task is readied by EXEC and forced to execute regardless of relative task priorities. This allows the task to perform it's initialization.

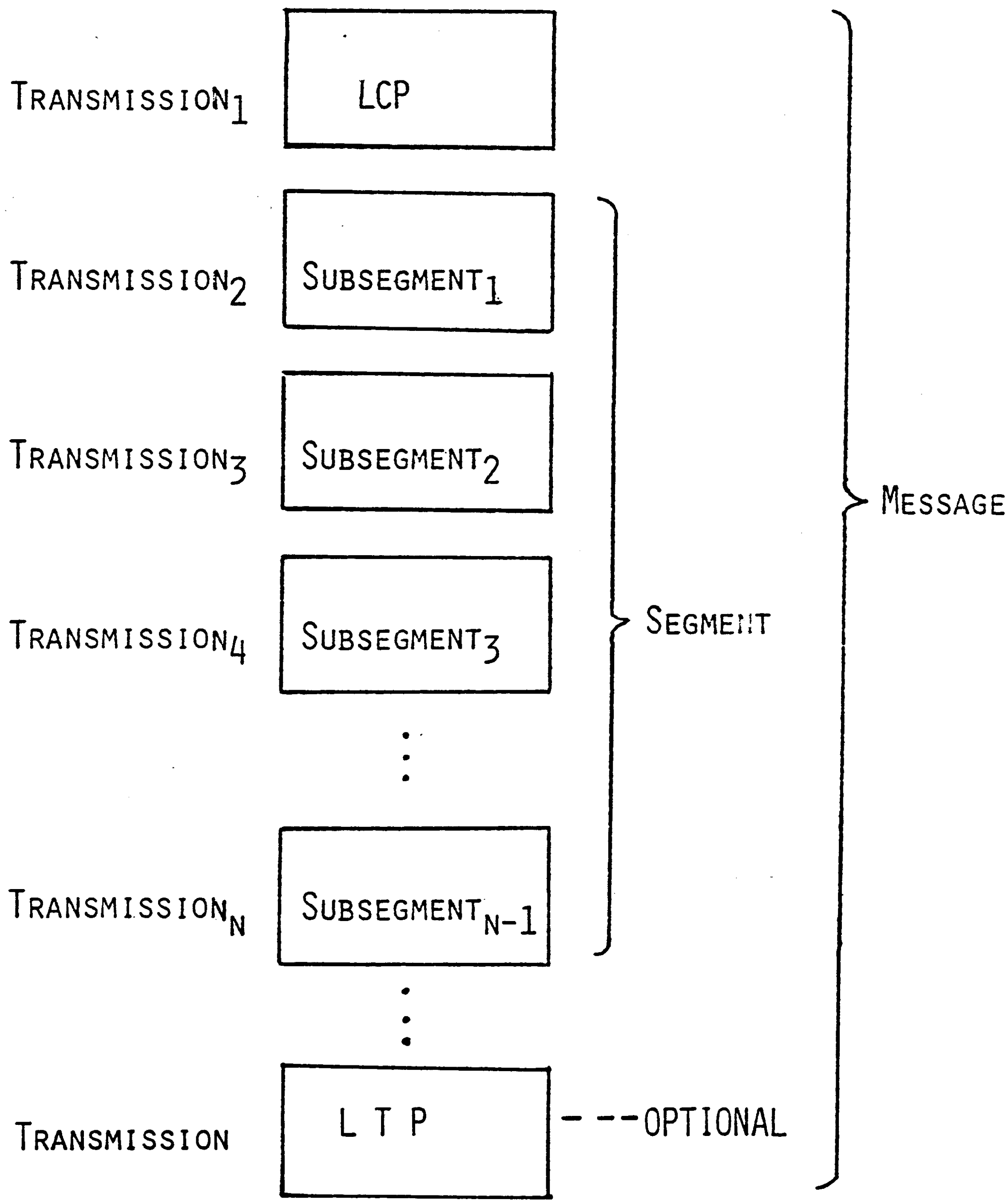
SECTION 6

STATION CALL PROCESSOR (SCP)

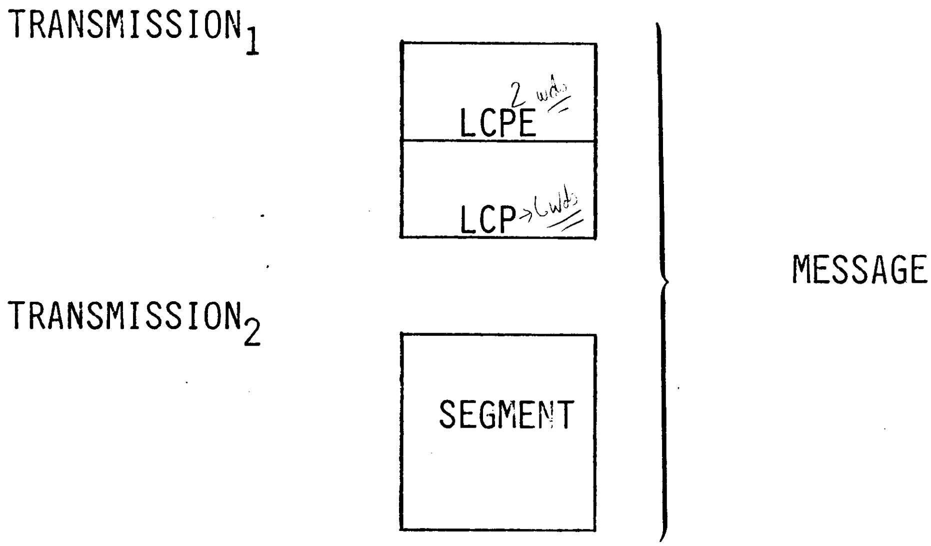
STAGER (STG)

GENERAL INTERFACE PROTOCOL

- EACH MESSAGE IS HEADED BY A LINK CONTROL PACKAGE
- SUBSEGMENT SIZE VARIES WITH FRONT-END



HYPERCHANNEL PROTOCOL



- LTP IS NOT SUPPORTED
- ONLY 1 SUBSEGMENT PER SEGMENT

LINK CONTROL PACKAGE

- EACH LCP CONSISTS OF SIX 64-BIT WORDS
- LCP CONTAINS:
  - SOURCE MAINFRAME ID (SID)
  - DESTINATION MAINFRAME ID (DID)
  - NO. OF SUBSEGMENTS (NSSG)
  - MESSAGE NUMBER (MN)
  - MESSAGE CODE (MC)
  - MESSAGE SUB CODE (MSC)
  - STREAM NO. (STN)
  - SEGMENT NUMBER (SGN)
  - SEGMENT LENGTH (SGBC)
  - STREAM CONTROL BYTES (ISCB, OSCB)

	0	8	15	24	32	40	48	56 63
0	DID		SID		NSSG	MN	MC	MSC
1	STN	SGN			SGBC			
2								
3	$ISCB_1$	$ISCB_2$	$ISCB_3$	$ISCB_4$	$ISCB_5$	$ISCB_6$	$ISCB_7$	$ISCB_8$
4	$OSCB_1$	$OSCB_2$	$OSCB_3$	$OSCB_4$	$OSCB_5$	$OSCB_6$	$OSCB_7$	$OSCB_8$
5								

Table 4-1. Message codes

Code	Function	Sender Station	Sender COS	Segment	Stream Required	Synchronous Request
001	Logon	X		X		
003	Logoff	X				
004	Start		X	X		
005	Restart		X			
006	Dataset header	X	X	X	X	

Table 4-1. Message codes (continued)

Code	Function	Sender Station	Sender COS	Segment	Stream Required	Synchronous Request
007	Dataset segment	X	X	X	X	
011	Control	X	X			
012	Message error	X	X			
013	Dataset transfer request		X	X		
014	Dataset transfer reply	X		X		
015	Enter logfile request	X[†]		X		X
016	Enter logfile reply		X	X		
021	Job status request	X[†]		X		X
022	System status request	X[†]		X		X
023	Dataset status request[§§]	X[†]		X		X
024	Link status request	X[†]		X		X
025	Mass storage status request	X[†]		X		X
026	Operator function request	X[†]		X		X
027	Debug function request	X[†]		X		X
031	Job status reply		X	X		
032	System status reply		X	X		
033	Dataset status reply[§§]		X	X		
034	Link status reply		X	X		
035	Mass storage status reply		X	X		
036	Operator function reply		X	X		
037	Debug function reply		X	X		
040	Diagnostic echo request	X[†]		X		
041	Diagnostic echo reply		X	X		
042	Interactive request	X[†]		X		X
043	Interactive reply		X	X		
044	Statclass request	X[†]		X		X
045	Statclass reply		X	X		
046	Station message[††]		X	X		
047	Station reply	X[†]		X		
050	Tape configuration request	X[†]		X		X
051	Tape configuration reply		X	X		
052	Tape job status request	X[†]		X		X
053	Tape job status reply		X	X		
054	Configure request	X[†]		X		X
055	Configure reply		X	X		
056	Dataset status request (ownership)[§§]	X[†]		X		X
057	Dataset status reply (ownership)[§§]		X	X		
060	Job information request	X[†]		X		X
061	Job information reply		X	X		
062	Stream status request	X[§]		X		X
063	Stream status reply		X	X		
064	Generic Resource Status Request	X[†]		X		
065	Generic Resource Status Reply		X	X		
070–077	Reserved for site use[†††]					

† Optional; the front-end station is not required to send.

†† COS does not send if the front-end station logged on with message receive disabled (Logon field MRE=0).

††† Message codes 070-077 are reserved for site use, and are maintained exclusively by the site. COS prevents COS products from using these codes, but is otherwise unaffected by them.

§ Reserved for CRI

§§ Codes 056 and 057 replace codes 023 and 033 for implementation of the security features introduced in COS 1.12. Codes 023 and 033 are still supported.

STREAMS

- A STREAM IS ALL THE MESSAGES RELATING TO A PARTICULAR DATA SET
- 8 INPUT AND 8 OUTPUT STREAMS — MAXIMUM
- ALTHOUGH EACH MESSAGE IS ASSIGNED TO ONLY ONE STREAM, THE LCP MUST CARRY STREAM CONTROL BYTES FOR ALL 16 STREAMS.

STREAM CONTROL BYTES

Octal Code	Mnemonic	Request/Response	Sender	Receiver
00	IDL	Idle	x	x
01	RTS	Request to send	x	
02	PTR	Preparing to receive		x
03	SND	Sending	x	
04	RCV	Receiving		x
05	SUS	Suspend		x
06	END	End dataset	x	
07	SVG	Saving dataset		x
10	SVD	Dataset saved		x
11	PPN	Postpone	x	x
12	CAN	Cancel	x	x
13	MCL	Master clear	x	x

RECEIVER SCB RESPONSE

SENDER SCB SENT

	IDL	PTR	RCV	SUS	SVG	SVD	PPN	CAN
IDL	N							
RTS		N	C	C			A	
SND			N	N			A	A
END					N	C	A	A
PPN	C							
CAN	C							

N = Normal receiver SCB response

C = Normal receiver SCB response which requires change in sender SCB

A = Abnormal receiver SCB response

SENDER SCB RESPONSE

RECEIVER SCB SENT

	IDL	RTS	SND	END	PPN	CAN
IDL	N	C				
PTR		N				
RCV			N	C	A	A
SUS			N	C	A	A
SVG				N		
SVD	C					
PPN	C					
CAN	C					

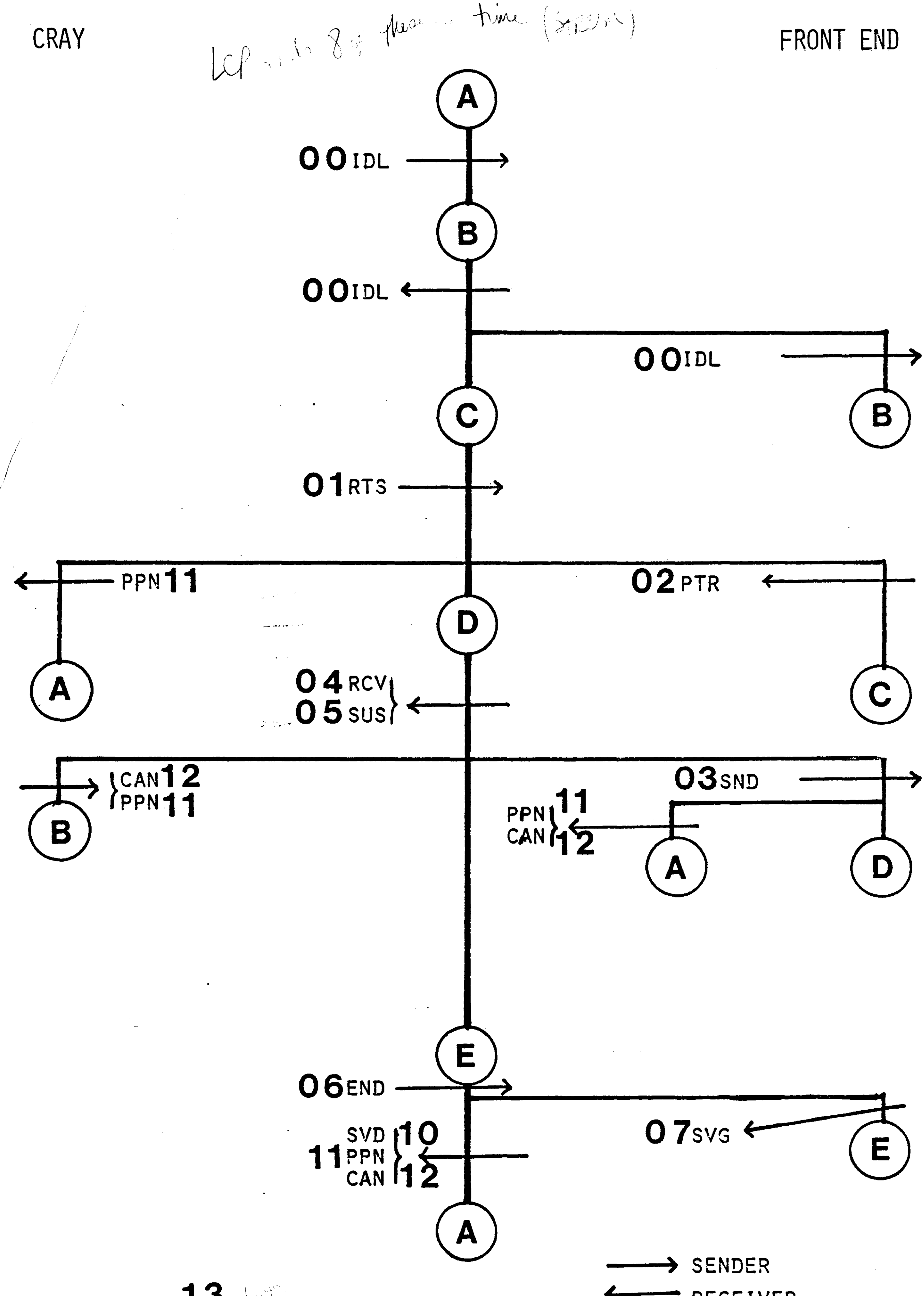
N = Normal sender SCB response

C = Normal sender SCB response which requires change in receiver SCB

A = Abnormal sender SCB response

BASIC STREAM FLOW

- FRONT-END IS LOGGED ON
- COMMUNICATIONS IN AN IDLE STATE
- FRONT-END SENDS RTS(01) TO THE CRAY-1
- CRAY-1 SENDS RCV (04) TO THE FRONT END.
- FRONT-END SENDS SND (03) TO THE CRAY-1 ALONG WITH THE JOB DATASET
- CRAY-1 SENDS RCV (04) TO THE FRONT-END WHILE DECODING THE MESSAGE AND SAVING THE JOB DATASET
- FRONT-END SENDS END (06) TO THE CRAY-1 UNTIL CRAY-1 HAS SAVED THE DATASET.
- CRAY-1 SENDS SVD (10) TO THE FRONT-END ONCE DATASET HAS BEEN SAVED.
- FRONT-END AND CRAY-1 THEN KEEP COMMUNICATIONS OPEN BY ALTERNATELY SENDING AND RECEIVING IDL(00).

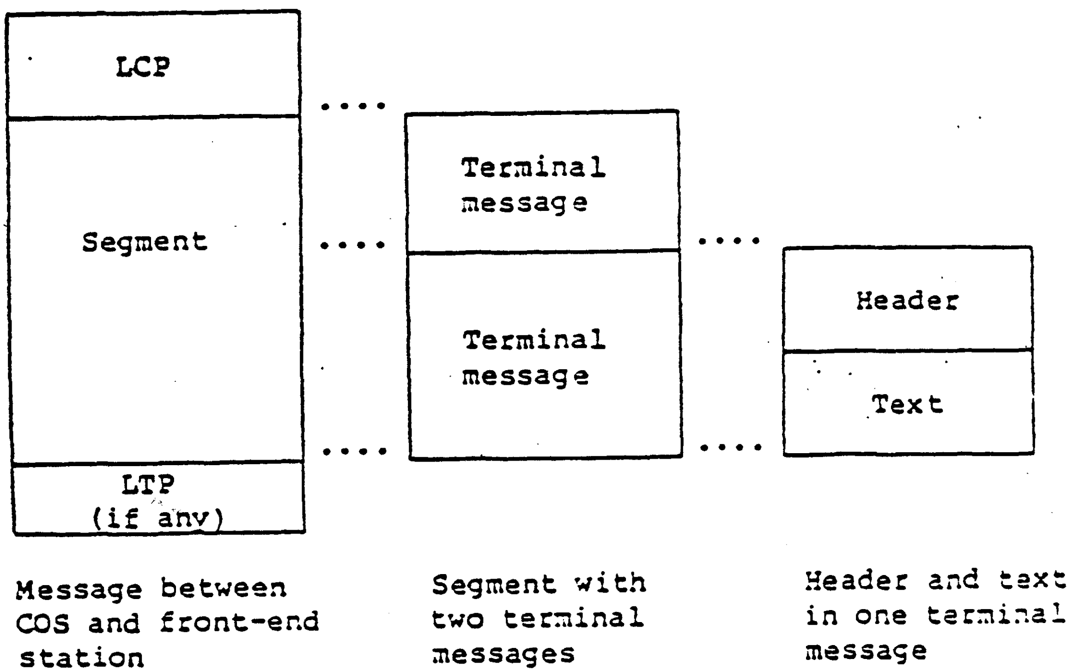


13

NOTE: A MCL SCB IS A LEGAL REQUEST OR RESPONSE AT ANY TIME. THE ONLY LEGAL REPLY TO MCL IS IDL.

STREAM CONTROL BYTE FLOW

INTERACTIVE MESSAGE

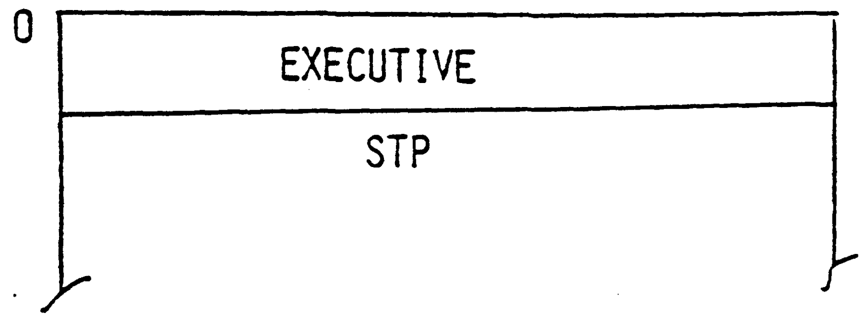


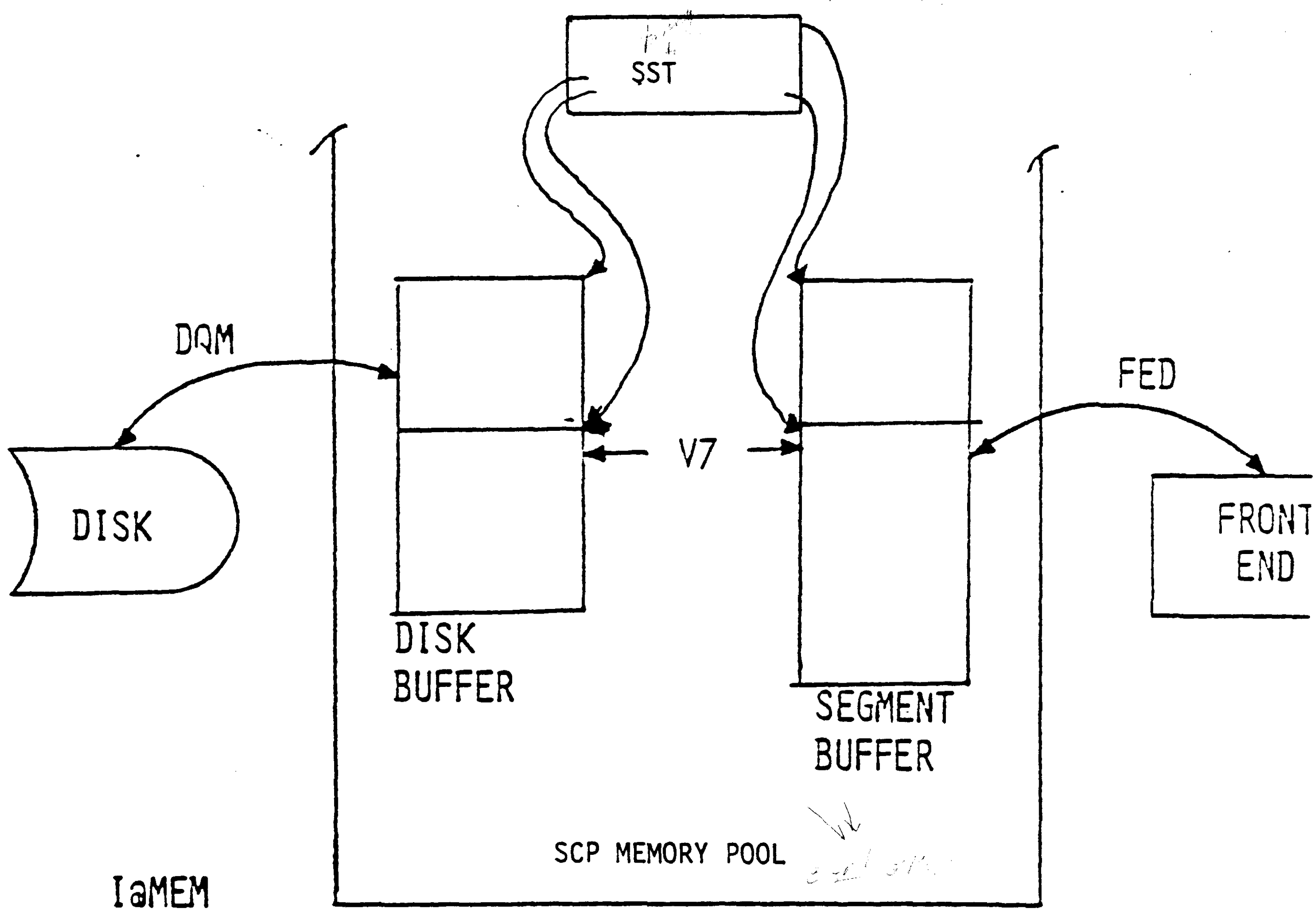
LINK TABLES

- LINK CONFIGURATION TABLE - LCT
  - DEFINES THE CONFIGURATION OF EACH PHYSICAL CHANNEL PAIR USED FOR FRONT END COMMUNICATION

- LINK INTERFACE TABLE - LIT
  - ONE ENTRY FOR EACH PHYSICAL CHANNEL
  - HOLDS INPUT LINK CONTROL PACKAGE FOR PHYSICAL CHANNEL
  - POINTS TO SEGMENT BUFFERS

- LINK INTERFACE EXTENSION TABLE - LXT
  - ONE ENTRY FOR EACH LOGICAL ID
  - HOLDS LINK CONTROL PACKAGE FOR THE LOGICAL ID
  - CONTAINS STAGER STREAM TABLE (SST) ENTRIES


0
EXECUTIVE
STP


SST
DQM
FED
V7
DISK
FRONT
END
DISK
BUFFER
SEGMENT
BUFFER
SCP MEMORY POOL
I@MEM

SYSTEM DATASET TABLE - SDT

- CONTAINS INFORMATION ON ALL DATASETS THAT ARE SENT BACK AND FORTH FROM THE FRONT END

- SEVEN QUEUES

- AN ENTRY ON A QUEUE REPRESENTS ONE DATASET

SDT QUEUES

- AVAILABLE QUEUE - CONTAINS AVAILABLE MEMORY FOR ALL SDT QUEUES
- INPUT QUEUE - JOBS WAITING TO BE INITIATED
- EXECUTE QUEUE - JOBS ALREADY INITIATED
- OUTPUT QUEUE - DATASETS WAITING TO BE SENT TO FRONT END
- SENDING QUEUE - DATASETS IN PROCESS OF BEING SENT
- RECEIVING QUEUE - DATASETS IN PROCESS OF BEING RECEIVED
- REQUEST QUEUE - FOR DATASET ACQUIRE REQUEST

INTER-TASK CALLS
SCP CALLS
* * *

SCP/any task

- via RTSK:

    1) INIT n operator command (n=task id)

SCP/DQM

- via PUTREQ:

    TRANSFER: 1) write RCV datasets to mass storage
                2) read SND datasets from mass storage

    *** SCP does not use CIO for DQM transfer requests ***

    DEALLOCATE: 1) cancel SND or RCV datasets
                2) deallocate job input dataset
                   - job termination
                   - operator KILL
                3) idle active LXT entry
                   - release all RCV dataset space

    ALLOCATE: 1) interactive job input dataset
                   - allocate a dummy dataset

SCP/JSH

- via RTSK:

    1) notify JSH that a new job in on input Q
    2) change PRI of executing job
    3) rewrite CSD dataset when job class is turned ON/OFF
    4) alter number of JXTs available with LIMIT

- via TSKREQ:

    1) job debugging from operator console
    2) operator control (DROP, KILL, RESUME, etc.)
    3) interactive job control
       - attention
       - abort
    4) ACQUIRE, DISPOSE failures (abort job)

INTER-TASK CALLS
SCP CALLS
* * *

SCP/JCM

- via TSKREQ:

  1) assign job to a class
     - job input dataset has arrived
  2) reassign job class; operator changed ...
     - front-end ID & TID (ENTER or ROUTE command)
     - priority, time limit (ENTER command)
  3) assign class to a job
     - ENTER CLASS command

*** JCM is called ONLY for jobs on the input Q ***

SCP/MSG

- via TSKREQ:

  1) record operator type-ins (system and user logs)
  2) error message for DISPOSE disk read failure
  3) log dataset transmission and reception messages

SCP/PDM

- via PUTREQ:

  1) see if dataset to ACQUIRE is on CRAY
  2) operator DATASET command processing
  3) save spooled input datasets
  4) delete spooled output datasets

SCP/TQM

- via PUTREQ:

  1) process operator CONFIG command for XIOP tapes

SECTION 7

DISK QUEUE MANAGER (DQM)

DD-19, DD-29 DISKS

1 SECTOR = 512 WORDS

1 TRACK = 18 SECTORS

1 CYLINDER = 10 TRACKS

1 DD-19 = 411 CYLINDERS

1 DD-29 = 823 CYLINDERS

DATASET ALLOCATION

- ALLOCATION MODES

  - PRE-ALLOCATION
  - DYNAMIC ALLOCATION

- ALLOCATION UNITS (ALLOCATION STYLE)

  - DISK SPACE IS ALLOCATED BY TRACK

- DEVICE ALLOCATION

  - IF SPECIFIED BY REQUEST, THE LOGICAL DEVICE NAME FROM THE DNT IS USED

  - OTHERWISE IT ROTATES AMONG THE CONTROLLERS AND DISKS AS SPECIFIED BY THE ORDER OF THE EQUIPMENT TABLE (EQT)

DISK QUEUE MANAGER (DQM)

- MANAGES ALLOCATION/DEALLOCATION OF MASS STORAGE (DISKS)
- MANAGES MASS STORAGE REQUEST QUEUES
- MANAGES MASS STORAGE CHANNELS, CONTROLLERS AND DISK UNITS.

DQM REQUESTS

- PRE-ALLOCATE DISK SPACE
- QUEUE I/O REQUESTS
- DEALLOCATE DISK SPACE

DEVICE RESERVATION TABLE - DRT

- ONE DRT FOR EACH DEVICE (DISK, SSD, BMR)

- CONTAINS A BIT MAP INDICATING WHICH TRACKS ARE ALLOCATED

- BIT POSITIONS IN THE DRT CORRESPOND TO THE ALLOCATION INDEX (LOGICAL TRACK ADDRESS)

DAT - Dataset Allocation Table

* * *

A Dataset Allocation Table defines the mass storage logical location of a dataset.

DAT format:

```
+---------------------------------------+
|           DAT  entry header           |
+---------------------------------------+
|                                       |
|                                       |
|                                       |
|               DAT entry               |
|                                       |
|                                       |
|                                       |
|                                       |
+---------------------------------------+
```

The DAT entry header contains general information about the dataset, such as dataset size and the DSC entry pointer. The DAT entry is divided into partitions. Each DAT partition describes a portion of the dataset for a single logical device. That is, if a dataset is spread over two logical devices, it has two DAT partitions.

DAT - Dataset Allocation Table

* * *

DAT partition format:

```
+--------------------------------------+
|           DAT  entry header          |
+--------------------------------------+ ---
|        DAT partition 1 header        |  :
+--------------------------------------+  :
|                                      |  :
|                                      |  :
|                                      |  :
|                                      |  :
|                                      |  :
|                                      |  :
|                                      |  partition 1
|         DAT partition 1 entry        |  :
|                                      |  :
|                                      |  :
|                                      |  :
|                                      |  :
|                                      |  :
|                                      |  :
+--------------------------------------+ ---
|        DAT partition 2 header        |  :
+--------------------------------------+  :
|                                      |  :
|                                      |  :
|                                      |  :
|                                      |  partition 2
|         DAT partition 2 entry        |  :
|                                      |  :
|                                      |  :
|                                      |  :
|                                      |  :
+--------------------------------------+ ---
```

The DAT partition headers contain general information concerning the partitions, such as the logical device name. The partition entries are a list of logical track addresses referred to as allocation indicies (AIs). Each AI is a bit index into the Disk Reservation Table (DRT).

DAT - Dataset Allocation Table
* * *

DAT partition format:

```
+--------------------------------------+ ---
|         DAT partition header         |  :
+--------+---------+---------+---------+  :
|  AI1   |  AI2    |  AI3    |  AI4    |  :
|  AI5   |  ...    |         |         |  :
|                                      |
|                                      |  partition
|                                      |
|                                      |  :
|                                 ...  |  :
| AIn-3  |  AIn-2  |  AIn-1  |   AIn   |  :
+--------+---------+---------+---------+ ---
```

A DAT is a segmented table. It actually consists of one or more fixed-size DAT pages, which are not necessarily contiguous in memory. Each DAT page is 16 words long. The first word of each page is the DAT page header. The remaining 15 words contain the DAT itself. DAT pages are numbered consecutively from 1, and each DAT page header contains a pointer to the next page.

DAT - Dataset Allocation Table
* * *

DAT page format:

```
+-----------------------------------------+
|            DAT page 1 header            |
+-----------------------------------------+
|                                         |
|                                         |
|                  DAT                    |
|                                         |
|                                         |
+-----------------------------------------+

+-----------------------------------------+
|            DAT page 2 header            |
+-----------------------------------------+
|                                         |
|                                         |
|              DAT (cont'd)               |
|                                         |
|                                         |
+-----------------------------------------+

+-----------------------------------------+
|            DAT page 3 header            |
+-----------------------------------------+
|                                         |
|              DAT (cont'd)               |
|                                         |
+-----------------------------------------+
|                (unused)                 |
+-----------------------------------------+
```

DAT - Dataset Allocation Table

* * *

DAT pages are allocated from the STP DAT area for system datasets (user dataset DATs are allocated from the dynamic portion of the job's JTA). The STP DAT area consists of a DAT page space and a space header.

DAT - Dataset Allocation Table

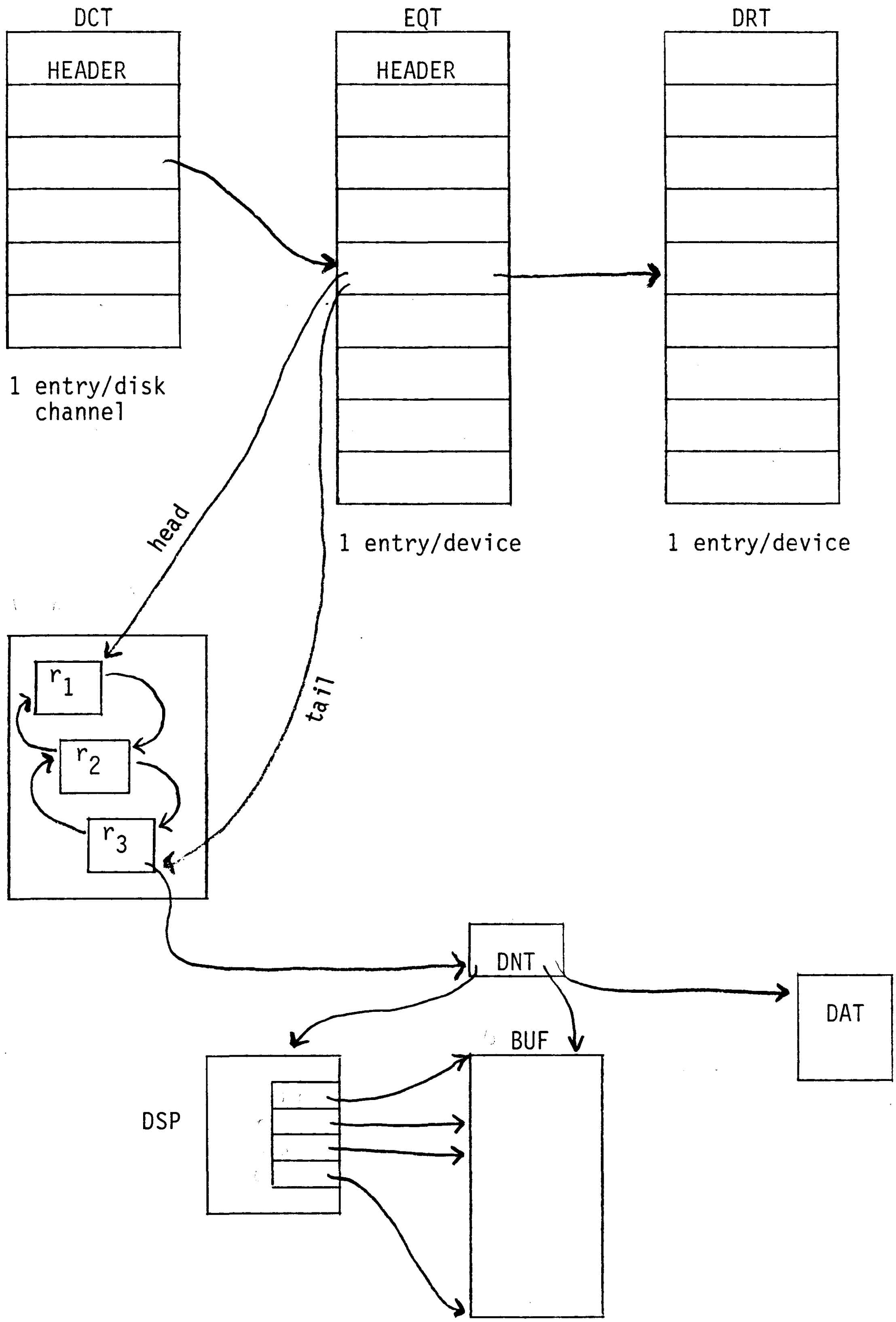
* * *

STP DAT area format:

```
+--------------------------------------+
|           DAT space header           |
+--------------------------------------+
|                                      |
|                                      |
|            DAT page space            |
|                                      |
|                                      |
|                                      |
+--------------------------------------+
```

The DAT space header contains a counter for the number of DAT pages available for allocation and a bit map for flagging currently allocated DAT pages.

DQM TABLE LINKAGE



DCT - points to current active EQT entry for each channel.

EQT - contains current request from request queue. Has a chain control word for the request chain. Also points to the DRT entry for the device.

RQT - request queue. Entries are a doubly linked list. Points to the DNT for the dataset.

A DQM TRANSFER REQUEST

```
***********************************************************************************
**
*
*                                   SUBROUTINE ROLLJOB
*
*
*  PURPOSE:
*            TO MAKE A REQUEST OF THE DISK QUEUE MANAGER, EITHER TO COPY
*            A JOB OUT ONTO ITS ROLLOUT DATASET OR TO READ THE JOB'S IMAG
*            BACK INTO MEMORY.
*
*  ENTRY:
*            A4 = JXT-ENTRY ADDRESS.
*            A5 = JTA ADDRESS.
*            DNP (PROCESSING DIRECTION IN THE ROLLFILE'S DNT) IS ALREADY
*                 SET -- TO 0 IF ROLLING IN, OR TO 1 IF ROLLING OUT.
*
*  EXIT:
*            I/O IS IN PROGRESS.
*
*  REGISTERS:
*            (A0-A2), (A6-A7), (S0-S2), (S6-S7) ARE DESTROYED.
**
***********************************************************************************

ROLLJOB  =         *
         A7        W@JXDNT,A4
         A6        W@JXCJS,A4
         S1        A5                 SET UP THE DNT:
         PUT,S1    S6&S7,DNBUF,A7       BUFFER ADDRESS = JTA ADDRESS,
         S2        A6                   NUMBER OF BLOCKS = JOB SIZE/512.
         S2        S2>D'9
         PUT,S2    S6&S7,DNNBK,A7
         S1        A4                 SUBMIT THE I/O REQUEST:
         S1        S1<D'40              LEFT-ADJUST THE JXT ADDRESS.
         S2        A7
         S1        S1!S2                INSERT THE DNT ADDRESS.
         A1        JSHID,0
         A2        DQMID,0
         S2        TRANSFER
         J         PUTREQ             LET PUTREQ RETURN TO THE CALLER.
```

DATASET NAME TABLE

0			
1	flags		DAT
2	NBK	SBK	BUF
3			
4			
5			
6			
7			
8			
9			
10			
11			
12			
13			

Field	Word	Bits	Description
FLAGS:	1	0-15	
DNP		3	Type of processing; used by Disk Queue Manager: 0 Read 1 Write
DNDAT	1	40-63	Dataset allocation table address: =0 No DAT assigned >0 DAT in STP <0 DAT in job's JTA
DNNBK	2	0-15	Number of blocks to be read or written; number of words in last block to be written if (DNEND)=1.
DNSBK	2	16-39	Starting block number
DNBUF	2	40-63	I/O buffer address

EQUIPMENT TABLE - EQT

- ONE ENTRY FOR EACH DISK
- CONTAINS STATUS AND ERROR INFORMATION
- POINTS TO DRT AND I/O REUQEST QUEUE

REQUEST TABLE - RQT

- ONE QUEUE FOR EACH DISK
  (QUEUE HEADER IS IN EQT ENTRY)
- CONTAINS PHYSICAL I/O REQUESTS
- USER REQUESTS PLACED ON END OF QUEUE
- SYSTEM REQUESTS PLACED SECOND ON QUEUE

FOR I/O REQUEST FLOW SEE SM-0040

CRAY RESEARCH, INC.

TECHNICAL TRAINING & DEVELOPMENT

Cray Research, Inc.
Software Training
2520 Pilot Knob Road, Suite 300
Mendota Heights, MN 55120

Cray Research, Inc.
Hardware Training
21 East Grand Avenue
Chippewa Falls, WI 54729