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

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

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

CRA Y RESEARCH, INC.,
1440 Northland Drive,
Mendota Heights, Minnesota 55120

<table>
<thead>
<tr>
<th>Revision</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>January, 1981 - obsoletes other revisions.</td>
</tr>
<tr>
<td>C</td>
<td>September, 1981 - obsoletes other revisions. Supports 1.10 release.</td>
</tr>
</tbody>
</table>
OVERVIEW OF CRAY-1 OPERATING SYSTEM (COS)
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
- MANAGES FILE MAINTENANCE
- MANAGES PROGRAM MAINTENANCE
- CAPABLE OF MODIFICATION AT STARTUP
COS HARDWARE ELEMENTS

- MCU

STARTUP AND RECOVERY OF COS FRONT-END COMPUTER SYSTEM

- MASS STORAGE SUBSYSTEM

UTILITIES
DATA
LIBRARIES
COPY OF COS (OPTIONAL)
COS OVERLAYS (OPTIONAL)
COPIES OF CSP (OPTIONAL)

- CRAY-1 COMPUTER

COS RESIDENT
USER APPLICATIONS EXECUTION AREAS
USER STATION BUFFERS
COS Software Elements

- Resident System Programs
  
  Executive (EXEC)
  System Task Processor (STP)
  Control Statement Processor (CSP) - Optional

- Non-Resident Programs
  
  Libraries
  System Utilities
  User Applications
SYSTEM MEMORY ASSIGNMENTS

EXEC TABLE AREA

STP TABLE AREA

Copy of CSP (optional)

User Area 1

User Area 2

User Area 3

User Area 4

User Area N

CRAY-OS System LOG & STATION BUFFERS

Maximum Memory

JOB TABLE AREA (JTA)

JOB COMMUNICATION BLK.

USER PROGRAM AREA

I/O TABLES & DATASET BUFFERS

BA

BA+2008

JCHLM

User LA
SYSTEM EXECUTIVE
SYSTEM EXECUTIVE PROGRAM (EXEC)

- CONTROL CENTER FOR COS
- EXECUTES IN MONITOR MODE
- CAN ACCESS ALL OF MEMORY (BA=0, LA=1nMEM)

DISK RESIDENT

Utilities
CAL
CFT
Loader

CSP

INTERRUPT

INTERCHANGE

INTERRUPT HANDLERS

CHANNEL PROCESSORS

DISK DRIVER
FRONT-END DRIVER
EXEC REQUEST PROCESSOR

TASK SCHEDULER

USER 1

JTA

JTA n

to current job

Idle program

COMMON ROUTINES

Task 1
Task 2
Task 3
...
Task n

STP
- CONTROL CENTER FOR COS
- EXECUTES IN MONITOR MODE
- CAN ACCESS ALL OF MEMORY
FUNCTIONS OF EXEC

- INTERCHANGE ANALYSIS
- INTERRUPT HANDLERS
- CHANNEL MANAGEMENT
- TASK SCHEDULER
- EXECUTIVE REQUEST PROCESSOR
- DISK DRIVER
- FRONT-END DRIVER
SYSTEM EXECUTIVE PROGRAM (EXEC)

- Control center for COS
- Executes in monitor mode
- Can access all of memory

DISK RESIDENT

INTERCHANGE

INTERRUPT HANDLERS

EXEC

CHANNEL PROCESSORS

DISK DRIVER

FRONT-END DRIVER

EXEC REQUEST PROCESSOR

STP

COMMON ROUTINES

Task 1

Task 2

Task 3

...
**INTERCHANGE**

- Saves current RTC
- Updates accrued CPU time
- Calls I/O handler if I/O interrupt
- Senses for lost hardware interrupts of previous disk I/O instructions
- Calls interrupt handlers for:

  - Console I/O interrupt
  - Real time clock
  - Error exit
  - Normal exit
  - Memory error exit
  - MIDI external clock
  - Floating point error
  - Program range error
  - Operator range error
SYSTEM EXECUTIVE PROGRAM (EXEC)

- CONTROL CENTER FOR COS
- EXECUTES IN MONITOR MODE
- CAN ACCESS ALL OF MEMORY

DISK RESIDENT

INTERUPT

INTERCHANGE

INTERRUPT HANDLERS

EXEC

CHANNEL PROCESSORS

DISK DRIVER

FRONT-END DRIVER

EXEC REQUEST PROCESSOR

COMMON Routines

Task 1

Task 2

Task 3

Task n

STP
INTERRUPT HANDLERS

- EXECUTE ONE OF THE FOLLOWING INTERRUPT HANDLERS:
  
  I0I  I/O INTERRUPT HANDLER
  CII  CONSOLE INTERRUPT HANDLER
  RTI  REAL-TIME INTERRUPT HANDLER
  NEI  NORMAL EXIT INTERRUPT HANDLER
  EE1  ERROR EXIT INTERRUPT HANDLER
  MEI  MEMORY ERROR INTERRUPT HANDLER

- CLEAR THE INTERRUPT FLAG IN THE EXCHANGE PACKAGE OF THE INTERRUPTED PROGRAM

- DETERMINE THE CHANNEL ON WHICH THE INTERRUPT OCCURRED

- BRANCH TO THE APPROPRIATE CHANNEL PROCESSOR
SYSTEM EXECUTIVE PROGRAM (EXEC)

- CONTROL CENTER FOR COS
- EXECUTES IN MONITOR MODE
- CAN ACCESS ALL OF MEMORY

DISK RESIDENT

UTILITIES

Loader

CSP

INTERRUPT

INTERCHANGE

INTERCEPT HANDLERS

EXEC

CHANNEL PROCESSORS

DISK DRIVER

FRONT-END DRIVER

EXEC REQUEST PROCESSOR

COMMON ROUTINES

Task 1

Task 2

Task 3

...
CHANNEL PROCESSORS

• CHANNEL PAIRS ARE ASSIGNED NUMBERS AS FOLLOWS:

0  CONSOLE
1  MCU-CLOCK
2
4
6  -  12 I/O CHANNELS

• 24

26  NORMAL EXIT PSEUDO CHANNEL
28  ERROR EXIT PSEUDO CHANNEL
30  PROGRAMMABLE CLOCK PSEUDO CHANNEL
32  MEMORY ERROR PSEUDO CHANNEL

• CHANNEL PROCESSOR TABLE (CHT) HAS ENTRIES FOR BOTH THE INPUT AND OUTPUT SIDES OF EACH CHANNEL

• CHT POINTS TO THE INTERRUPT PROCESSOR

• CHT POINTS TO A PARAMETER AREA FOR EACH CHANNEL PROCESSOR

• CHT POINTS TO CONTROL TABLE ADDRESS
The Channel Table resides in EXEC memory and contains information for use by the interrupt handlers. There is one entry for each channel, physical or pseudo.

<table>
<thead>
<tr>
<th>Field</th>
<th>Word</th>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHTN</td>
<td>0</td>
<td>0-23</td>
<td>Table name; &quot;CHT&quot; in ASCII</td>
</tr>
<tr>
<td>CHTPB</td>
<td>0</td>
<td>0-15</td>
<td>Address of task parameter word</td>
</tr>
<tr>
<td>CHCTA</td>
<td>0</td>
<td>16-39</td>
<td>Control table address</td>
</tr>
<tr>
<td>CHIHA</td>
<td>0</td>
<td>40-63</td>
<td>Interrupt handler address</td>
</tr>
</tbody>
</table>
CLEAR FLAG BIT IN XP

UPDATE EXIT COUNT FOR INT. PROG.

OBTAIN NORMAL EXIT ADDR. FROM CHT

NORMAL PROCESSOR

UPDATE TASK QUEUE

OBTAIN DRIVER ADDRESS FROM CHT

NORMAL PROCESSOR

USER EXEC PROCESSOR

COMPARE A REQUEST FOR A TASK

UPDATE TASK QUEUE

COPY THE XP TO THE USER AREA

INDICATE USER XP IN JTA

DISK INT.

OBTAIN PHYSICAL UNIT TABLE ADDRESS

FRONT-END

DISK

DRIVER
TASK REQUESTS

- THE EXECUTIVE REQUEST PROCESSOR IS INITIATED BY THE NORMAL EXIT CHANNEL PROCESSOR.

- THE REQUEST IS PASSED TO EXEC IN REGISTERS S6 AND S7

- EXECUTIVE REQUESTS ARE:

  CREATE A TASK
  READY A TASK
  SELF SUSPEND TASK
  ASSIGN CHANNEL
  STATION I/O REQUEST
  DISK BLOCK I/O REQUEST
  READY TASK AND SUSPEND SELF
  GET TIME AND DATE
  CONNECT USER JOB TO CPU
  DISCONNECT USER JOB FROM CPU
  POST A MESSAGE IN HISTORY BUFFER
  SET MEMORY SIZE
  START/STOP OPERATING SYSTEM
  DISPLAY MEMORY/EXCHANGE PACKAGE
  ENTER MEMORY/EXCHANGE PACKAGE
  SET/CLEAR SYSTEM BREAKPOINT

2.15
CREATE A TASK

SET TO RETURN WITH ERROR OF FULL

SET TASK STATUS TO REQUESTED IN STT

SET TASK PRIORITY IN STT

SET TASK ID IN STT

CONSTRUCT XP IN SYSTEM TASK TABLE (STT)

SET SYSTEM DEFINED IN WORD 2 OF STT

FORCE TASK INTO EXECUTION

EXIT

<table>
<thead>
<tr>
<th>S6</th>
<th>S7</th>
</tr>
</thead>
<tbody>
<tr>
<td>status</td>
<td>priority</td>
</tr>
<tr>
<td>P-register for new task</td>
<td>01</td>
</tr>
</tbody>
</table>
SYSTEM EXECUTIVE PROGRAM (EXEC)

- CONTROL CENTER FOR COS
- EXECUTES IN MONITOR MODE
- CAN ACCESS ALL OF MEMORY

DISK RESIDENT

INTERCEPT

EXEC

INTERCHANGE

INTERRUPT HANDLERS

CHANNEL PROCESSORS

DISK DRIVER

FRONT-END DRIVER

EXEC REQUEST PROCESSOR

STP

COMMON ROUTINES

Task 1

Task 2

Task 3

Task n
CIRCULAR BUFFER BEGINS AT LOCATION DBF

LOCATION DBFP POINTS TO NEXT BUFFER ADDRESS

EACH TRACE MESSAGE 4 WORDS IN LENGTH

CIRCULAR BUFFER HOLDS THE 1024 MOST CURRENT MESSAGES.

FUNCTIONS MAY BE COLLECTIVELY ENABLED/DISABLED THROUGH SETTING OF LOCATION DBUGM AND ITS ASSOCIATED FUNCTION TABLE.

SETTING LOCATION DBUGM NONZERO ENABLES ALL FUNCTIONS

SETTING DBUGM+FUNCTION NUMBER ENABLES THE FUNCTION

** EXEC HISTORY TRACE **

### Function Number

<table>
<thead>
<tr>
<th>Function Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>'ALL'L+1</td>
<td>UNIVERSAL ENABLE FLAG</td>
</tr>
<tr>
<td>'TOI'L</td>
<td>I/O INTERRUPT</td>
</tr>
<tr>
<td>'UIN'E'L</td>
<td>USER NORMAL EXIT</td>
</tr>
<tr>
<td>'SPE'E'L</td>
<td>OTP NORMAL EXIT</td>
</tr>
<tr>
<td>'NE'E'L</td>
<td>EXEC NORMAL EXIT</td>
</tr>
<tr>
<td>'RTI'L</td>
<td>REAL TIME INTERRUPT</td>
</tr>
<tr>
<td>'XCPC'L</td>
<td>COPY USER XP TO JTA</td>
</tr>
<tr>
<td>'MCU'L</td>
<td>MCU COMMAND</td>
</tr>
<tr>
<td>'SCHJ'L</td>
<td>SCHEDULE USER JOB</td>
</tr>
<tr>
<td>'DIOC'L</td>
<td>DISK DRIVER REQUEST</td>
</tr>
<tr>
<td>'ITMS'L</td>
<td>INTER-TASK MESSAGE</td>
</tr>
<tr>
<td>'EE'I'L</td>
<td>ERROR EXIT</td>
</tr>
<tr>
<td>'XCHG'</td>
<td>XCHG PACKAGE TRACE CURRENT</td>
</tr>
<tr>
<td>'DBUGM+DBF'</td>
<td>DEBUG BUFFER CURRENT POSITION</td>
</tr>
</tbody>
</table>

### Field Word Bits Description

<table>
<thead>
<tr>
<th>Field</th>
<th>Word</th>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>0</td>
<td>0-6</td>
<td>Function number</td>
</tr>
<tr>
<td>P</td>
<td>0</td>
<td>27-50</td>
<td>Program register of interrupted exchange package</td>
</tr>
<tr>
<td>XP</td>
<td>0</td>
<td>51-63</td>
<td>Exchange package address</td>
</tr>
</tbody>
</table>
**DISK DRIVER (R011)**

- Does the physical I/O on disks
- Requests to R011 queued via the Request Table (RQT)
- Updates the Dataset Parameter Table (DSP)
- Examines the Disk Reservation Table (DRT) for useable disk space
- Performs servo offset and data strobe functions for error recovery
COMMON SUBROUTINES
SYSTEM TASK PROCESSOR

- **DEFINITION** - CONSISTS OF TABLES, COMMON SUBROUTINES, TASKS, AND I/O ROUTINES.

- **COMMON SUBROUTINES**
  - RE-ENTRANT ROUTINES
  - USED BY TASKS

- **TASKS**
  - PERFORMS A SPECIFIC OPERATION
  - CAN BE CALLED BY OTHER TASKS
  - HAVE THEIR OWN XP’s
  - HAVE THEIR OWN ID NUMBERS (0-358)
  - HAVE THEIR OWN PRIORITIES (0-3778)
  - BA AND LA THE SAME FOR ALL TASKS (BA=BASEP, LA=1MEM)
  - COMMUNICATE WITH EXEC, EACH OTHER AND WITH USER JOBS.

- **TABLES**
  - USED BY STP AND EXEC
STP COMMON ROUTINES

- USED BY TASKS TO PERFORM:

  TASK LOGICAL INPUT/OUTPUT (TIO)
  CIRCULAR INPUT/OUTPUT (CIO)
  MEMORY MANAGEMENT
  ITEM CHAINING/UNCHAINING
  TASK TO TASK COMMUNICATION
Overview of COS I/O
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
  - $REW - 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.

- 1 TIO routine ($WWDP/$WWDR) will be examined here. Refer to SM0040 for remainder of TIO routines.
Task's Data Area

SWDP
SWDR
WWDS
WEOF
WEOD

I/O BUFFER

DSP

CMCC
for
DQM

WDCS

disk queue manager

mass storage

TIO logical write
$WWD$ WRITES THE NUMBER OF WORDS SPECIFIED INTO THE I/O BUFFER. $WWD$ IS CALLED VIA A RETURN JUMP WITH THE CALLER PROVIDING THE FOLLOWING:

**ENTRY CONDITIONS:**
(A1) Address of DSP
(A2) FWA of task's data area
(A3) Word count
   If count is 0, no data is transferred
(A6) Address of DNT
(A7) Address of JXT
   (=0 if not job related)

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

$WWD

WAS PRECEDING A WRITE

NO

MOVE WORDS INTO BUFFER

YES

WRITE EOR

NO

MORE WORDS

YES

INSERT END RECORD WORD

AT 512 WORDS

NO

YES

$WBLK

UPDATE DATASET PARAMETER AREA

EXIT

WRITE DATA TO DISK (WDCS)

BUFFER > HALF FULL

UPDATE DATASET PARAMETER AREA

EXIT

$WBLK

INSERT BCW

$WBLK

3.9
* WRITE WORDS
* ENTRY:
  A1 @FCB
  A2 FWA
  A3 COUNT
  A6 DNT ADDRESS
  A7 JXT ADDRESS, =0 IF NOT JOB RELATED
* EXIT:
  A1 @FCB
  A2 FWA
  A3 COUNT
  A6 DNT ADDRESS
  A7 JXT ADDRESS
* MODIFIES:
  A0,4-5 S0,1,2,3,4,5,6,7
* WRITE WORDS, PARTIAL MODE
  $SWDP = *
  S6 = 0
  J WW01
* WRITE WORDS, RECORD MODE
  $SWDR = *
  S6 = >1
  WW01 = *
  S1 A6 DNT ADDRESS
  S2 A7 JXT ADDRESS
  S1 $1'X'D'24
  S1 $1'S2
  WW02 = *
  A7 B0
  S1 A7 B.ZA
  S2 A2 B.ZB
  S1 $1'X'D'24
  S1 $1'S2
  WW03 = *
  R TDSP CALCULATE A7 = DSP BASE
  S4 5.A1 PW
  S3 2.A1 IN
  S0 S4
  S2 <30
  JAN WW03 NOT FIRST WRITE
  S4 $3'X'S4'S2 SET POWA
  S2 <1
  A4 A3
  S3 $3'X'S2 IN + 1
  S2 164
  S2 2X71
  A4 A4'A7
  0.A4 A0 CLEAR FIRST BCW
  A4 A4'A7
  S4 $2'X'S4 INSERT RAW BITS
  S2 >4
  JAN WW22 ERROR
  R WW30 PROCESS WRITE AFTER READ
  JAN
A0 A2-A3
S0 0, A2
JAM WUI4 NOT AT EOC
A7 A7-A4
S0 W3DPTM.A1 T.ZA
* END OF COUNT
S1 W3DPTM.A1 PW (T.ZA)
JSP WUI2 PARTIAL MODE
* RECORD MODE
S1 W3DPTM.A1 T.ZA
S2 <43
S1 S1>17
S2 S2<11
S4 @PCW
S2 >4
S1 S4!S1&$2 BFI, BRI
S2 <4
S1 A7 A7-A4 INSERT MODE BITS
A7 A7-A4 STORE ROW
A7 A7-A4 PCW
A7 A7+6
A2 A5+1
S1 A4
A2 A4-A2 FWI
S2 <30
A4 A4+1
S4 S1!S4&S2 UPDATE PCW
S1 A2
S6 S6
A0 S1&A5 INSERT FWI
S6 S6
A7 A7-A6 NOT AT BCW
JAM WUI21 WRITE BLOCK
R WUI2 WRITE BLOCK
JAN WUI21 ERROR
* PARTIAL MODE
WUI21 = *
S1 A4
S3 A4 UPDATE PW
S2 <30
S3 S1!S3&S2 INSERT NWA
S2 <30
S3 S2&S3 CLEAR BP
S3 A1 S3 UPDATE IN
A2 W3DPTM+2.A1 B.ZS
A3 A3-A2 COUNT
A0 0 COMPLETE FLAG
J WUI22
WUI39 = *
S1 W3DPPRM.A1
S3 >1
S3 S3>S3&DDPERM WRITE PAST EOD
S1 S1!S3 SET ERROR FLAG
W3DPPRM.A1 S1
WUI22 = *
S1 W3DPTM+2.A1
S1 S1>D’24
* S1 43’, 24/80
* A1 DSP ADDRESS
* WUI9 = *
A7 S1
B0 A7
S1 W3DPTM+5.A1 JXT ADDRESS
S1 S1>D’24 INT ADDRESS
J B0

3.12
VECTOR MOVE FOR WRITE RECORD

A2: SOURCE ADDRESS
A4: OUT
A7: DESTINATION ADDRESS
A6: SHORT VECTOR LENGTH (NOT ZERO)
S1: NEGATIVE NUMBER OF WORDS TO BE MOVED

S2: A3 SAVE A3
C4: BICOMP: V0 SAVE AREA FOR BUFFERED I/O
A3: Z50 D'54
V0: A3 SAVE V0
A3: S1
A3: -A3 INCREMENT OUT

A0: SOURCE ADDRESS
A6: A5
V0: A0.1
A7: DESTINATION ADDRESS
A3: A3-A6 WORDS LEFT TO MOVE
A2: A2-A6
A7: A7-A6
A6: Z50 D'54
V0: A0.1
A3: A3
J: WMMV1 LOOP UNTIL ALL WORDS MOVED

**

** NAME TDSP
* CALCULATE DSP POINTER BASE
* = 0 IF JXT ADDRESS =0
* = JTA ADDRESS IF DNJTF =1
* = USER BA IF DNJTF =0

** ENTRY TDSP:
* A1: DSP ADDRESS
* TDSP1:
* S7: 16/0.24/DNT ADDRESS,24/JXT ADDRESS

* EXIT A1: DSP ADDRESS (TDSP ENTRY ONLY)
* A7: BASE OF DSP POINTERS
* REGISTERS MODIFIED:
* A0,A7 S0,S7

**

TDSP: S7: W3DPTHHE.A1
TDSP1 = k ENTRY W, S7 = JXT ADDRESS
A0: S7 JXT ADDRESS
A7: 0
JAZ: TDSP9 NOT JOB RELATED
A7: S7 JXT ADDRESS
A7: W3DNJTA.A7 JTA ADDRESS
S7: S7>D'24
TDSPA.0: A6 SAVE A6
A6: S7 DNT ADDRESS
S0: W3DNJTF.A6
S0: S0<S0=DNJTF
A6: L3JTA
JSM: TDSP3 IN JTA
A7: A8+H7 USER BA
TDSP8: A6 TDSPA.0
TDSP9 J B9 RETURN 3.13
TDSPA: 0
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.
A. Filling the buffer

B. Emptying the buffer

C. Concurrently filling and emptying the buffer

Physical I/O
Logical I/O requires the presence of a DSP for the dataset in the user's field. Refer to publication SR-0011 for details of DSP use.

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>3</th>
<th>5</th>
<th>10</th>
<th>16</th>
<th>25</th>
<th>40</th>
<th>56</th>
<th>63</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>BSY</td>
<td>STS</td>
<td>DN</td>
<td>TCS</td>
<td>TP</td>
<td>MEM</td>
<td>RDM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>ERR</td>
<td>BFI</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td>IPB</td>
<td></td>
<td>IBN</td>
<td>TRAN</td>
<td>IA</td>
<td>UDS</td>
<td>IN</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>RBC</td>
<td>OBP</td>
<td></td>
<td>OBN</td>
<td></td>
<td></td>
<td></td>
<td>OUT</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>RW</td>
<td>PPI</td>
<td></td>
<td>PRI</td>
<td></td>
<td>RCW</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>EOF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>LPW</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>BF</td>
<td>BUBC</td>
<td></td>
<td>BWC</td>
<td></td>
<td>BWA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>DPDIR</td>
<td>TPC</td>
<td>TPM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>DEL</td>
<td>BLNK</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>23</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Dataset Parameter Area (DSP)
MEMORY MANAGEMENT

- PROVIDES TEMPORARY MEMORY AREAS FOR TASKS (AREAS IN STP)
- MEMORY AREAS ARE OF VARIABLE-SIZE
- MEMORY POOLS ARE USED

MEMORY MANAGEMENT ROUTINES ARE:

  MEMAL-ALLOCATES A TASK MEMORY AREA

  MEMDE-DEALLOCATES A TASK MEMORY AREA
MEMAL ALLOCATES A VARIABLE-SIZE MEMORY AREA TO A TASK. MEMAL IS CALLED VIA A RETURN JUMP WITH THE CALLER PROVIDING:

INPUT REGISTERS:  
- (A6) = Number of memory pool from which to allocate  
- (A7) = Number of words desired

OUTPUT REGISTERS:  
- (A6) = Status:  
  0: Good status  
  1: Invalid memory pool number  
  2: Invalid number of words requested  
  3: Memory not available

- (A7) = Address of first usable word of memory to be allocated
EJECT

* PURPOSE MEMAL WILL ALLOCATE A VARIABLE SIZE AREA OF MEMORY FROM A
  MEMORY POOL.

* ENTRY A6 = NUMBER OF THE MEMORY POOL TO ALLOCATE FROM
A7 = NUMBER OF WORDS REQUIRED

* EXIT A6 = STATUS
0 = GOOD RETURN
1 = INVALID POOL NUMBER
2 = INVALID NUMBER OF WORDS REQUESTED
3 = MEMORY NOT AVAILABLE
A7 = ADDRESS OF FIRST USEABLE WORD ALLOCATED—MEANINGFUL
  ONLY IF A6 = 0

* MISTAKES
MISTAKE1 = 1
MISTAKE2 = 2
MISTAKE3 = 3

A5 = POOLTCL BASE ADDRESS OF POOL TABLE
GET.S5 S6&37.PTMAX,A5 MAX POOL NUMBER
A4 S5
A6 A1-A6
JAI MEMER1 JUMP IF POOL NUMBER GREATER THAN MAX
A5 A6+A6 ADDRESS OF POOL WORD
S0 0.AE
JSZ MEMER1
A6 A7
JAZ MEMER2 JUMP IF ZERO WORDS REQUESTED

* THE TOTAL SIZE TO BE ALLOCATED IS THE REQUESTED SIZE PLUS
  THE SIZE OF THE HEADER AND TRAILER

A4 MPHT
A7 A7+4A4 TOTAL SIZE REQUIRED

* VERIFY THAT THE REQUESTED SIZE IS NOT LARGER THAN THE POOL

GET.S1 S6&37.PTSIZE,A5
A4 S1
A6 A4-A7
JAI MEMER2
GET.S1 S6&37.PTBASE,A5
A4 S1 BASE ADDRESS OF MEMORY POOL

* LOCK OUT INTERRUPTS UNTIL THE SPACE HAS BEEN ALLOCATED

S1 1
STLK.A0 S1

MEMAL10 =

GET.S1 S6&37.MPID.A4 VALIDATE THE HEADER WORD
S2 1DURD.A6
S0 S1-S2
EPEN
GET.S0 S6&37.MPST.A4
JAI MEMAL50 JUMP IF AREA IN USE
GET.S1 S6&37.MPSIZE,A4 SIZE OF AVAILABLE AREA
A1 S1
A2 A1-A7
JAI MEMAL30 COUNT SIZE UNDIFFER TO AVAILABLE
AT THIS POINT THE SIZE OF THE REQUEST IS EQUAL TO THE
AVAILABLE AREA

S1  1
PUT.S1 SB&S7,MPST,A4 SET IN USE INDICATOR
A2  A4+1
A6  A2-1 ADDRESS OF TRAILER WORD
PUT.S1 SB&S7,MPST,A2

RELEASE INTERRUPT LOCKOUT

S1  0
UNLOCK A7  A4+1 RETURN ADDRESS FIRST USABLE WORD

CLEAR THE MEMORY ALLOCATED WITHOUT DESTROYING THE HEADER
AND TRAILER

MEMAL20 =  A6  A7
  0.A6  S1
  A6  A6+1
  A9  A9+H2
JAN  MEMAL20

RETURN TO REQUESTOR WITH GOOD STATUS

A6  0  J  B00

HERE AN AVAILABLE MEMORY AREA HAS BEEN LOCATED. IT IS UNEQUAL
IN SIZE TO THE REQUEST

MEMAL30 =  A6  A7
  0.A6  S1
  A6  A6+1
  A9  A9+H2
JAN  MEMAL50  AVAILABLE IS LESS THAN REQUEST

REPLACE THE EXISTING HEADER WORD AND CREATE A NEW TRAILER

A2  A4+47  TRAILER ADDRESS
A2  A2-1
S1  0  CLEAR OUT ANY GARBAGE
S1  A7
PUT.S1 SB&S7,MPST,A4
PUT.S1 SB&S7,MPST,A2 SIZE OF ALLOCATED AREA
S1  1
PUT.S1 SB&S7,MPST,A4
PUT.S1 SB&S7,MPST,A2 IN USE INDICATOR
S2  I0WRD,A6
PUT.S2 SB&S7,MP1D,A4
PUT.S2 SB&S7,MP1D,A2 POOL ID

CREATE A NEW HEADER AND TRAILER FOR THAT SPACE WHICH IS
STILL AVAILABLE.

A2  A2+1  ADDRESS OF NEW HEADER
A1  A1+47  SIZE OF NEW AREA
A6  A2+47
A6  A6+1  ADDRESS OF NEW TRAILER
S6  0
0.A2  S8  CLEAR NEW HEADER
S1  A1
PUT.S1 SB&S7,MPST,A2
PUT.S1 SB&S7,MPST,A6
S1  0
PUT.S1 SB&S7,MPST,A2
PUT.S1 SB&S7,MPST,A6
PUT.S2 SB&S7,MP1D,A2
PUT.S2 SB&S7,MP1D,A2
UNLOCK

3.24
CLEAR THE ALLOCATED AREA

A2  A2-1
A7  A4+1
A4  A4+1
MENAL40 = *
0 A4  S1
A4  A4+1
A0  A2-A4
JAN  MEMAL40
A6  0  RETURN GOOD STATUS
MENAL50 = *
GET, S1 SBLS7, MPSIZE, A4 SIZE OF CURRENT AREA
A2  S1
A4  A4+A2 ADDRESS NEXT HEADER WORD
GET, S1 SBLS7, PTBASE, A5 POOL BASE ADDRESS
GET, S2 SBLS7, PFSIZE, A5 POOL SIZE
S1  S1+S2 END ADDRESS OF POOL
A1  S1
A0  A1-A4 JAN  MENAL10 JUMP IN ENTRIE POOL NOT YET SEARCHED

THE MEMORY POOL DOES NOT CONTAIN ENOUGH AVAILABLE CONSECUTIVE SPACE TO FILL THE REQUEST

UNLOCK RELEASE INTERRUPT LOCK-OUT
A6  MISTAKE3
J  B00
MEMER1 = *
A6  MISTAKE1
J  B00
MEMER2 = *
A6  MISTAKE2
J  B00
MEMDE RETURNS (DEALLOCATES) MEMORY TO THE MEMORY POOL FOR REALLOCATION. MEMDE IS CALLED VIA A RETURN JUMP WITH THE CALLER PROVIDING:

**INPUT REGISTERS:**
- (A6) = Memory pool number
- (A7) = Address of first usable word of memory to be deallocated

**OUTPUT REGISTERS:**
- (A6) = Status:
  - 0: Good return
  - 1: Invalid address
  - 2: Area not currently allocated
  - 3: Invalid pool number
- (A7) = Address of memory released; meaningful only if status is 0.
**COMMON SUBROUTINES**

**CAL 1.03 72282**

10/09/78 12:36:17

**PAGE 68**

---

**NAME: MEMDE**

**PURPOSE:** MEMDE returns memory to the pool combining it with available adjacent areas to help prevent fragmentation.

**ENTRY:** A6 = pool number memory should be returned to.

**EXIT:** A6 = status

- 0 = good return
- 1 = invalid header word
- 2 = area not currently allocated
- 3 = invalid pool number

**NAME:** A7 = address of area released – meaningful only if good status.

---

**DEER1** = 1

**DEER2** = 2

**DEER3** = 3

**MEMDE** = X

**GET, S1** = SS & S7, MPID, A1

**S2** = IDWRD, A6

**S0** = S2 - S1

**JSN DUM1**

JUMP IF INVALID HEADER WORD

**GET, S0** = SS & S7, MPST, A1

**JS2 DUM2**

JUMP IF PREVIOUSLY RELEASED

**A5** = POOL_TBL.

BASE ADDRESS OF POOL TABLE

**GET, S1** = SS & S6, PTCNT, A5.

MAX VALID POOL NUMBER

**S2** = A6

**S0** = S1 - S2

**JSM DUM3**

JUMP IF INVALID POOL

**A5** = A5 + A6.

ADDRESS OF POOL WORD

**S0** = 0, A6

**JSZ DUM4**

JUMP IF INVALID POOL

**GET, S1** = SS & S7, PTBASE, A5

**GET, S2** = SS & S7, PTSIZE, A5.

POOL SIZE

**S2** = S2 + S1.

END ADDRESS OF POOL

**S3** = A1.

ADDRESS OF AREA TO RELEASE

**S0** = S3 - S1

**JSM DUM5**

JUMP IF ADDRESS OUTSIDE POOL RANGE

**S0** = S2 - S3

**JSM DUM5**

JUMP IF ADDRESS OUTSIDE POOL RANGE

---

THE VALIDATION IS COMPLETE. NOW ACTUALLY RELEASE THE SPACE

**SS** = 1

**STPJK, A0** = SS.

LOCK OUT INTERRUPTS

**GET, S4** = SS & S7, MPSIZE, A1.

SIZE OF AREA

**S5** = S3 + S4

**A2** = S5

**A2 = A2 - 1**

TRAILER ADDRESS

**S6** = 0

**PUT, S6** = SS & S7, MPST, A1

MARK IT AVAILABLE

**PUT, S6** = SS & S7, MPSY, A2

**S0** = S5 - S2

**JS** = MEMDE10.

JUMP IF LAST AREA IN POOL

**A4** = S5.

ADDRESS OF NEXT HEADER

**GET, S0** = SS & S7, MPST, A4

**JSN MEMDE10**

JUMP IF NEXT AREA NOT AVAILABLE
THE FOLLOWING MEMORY AREA IS AVAILABLE. COMBINE THE TWO INTO ONE LARGE AREA.

GET.S5 S6&S7, MPSIZE, A4 SIZE OF FOLLOWING AREA
S4 S5+S4 COMBINED SIZE
A5 S4
A2 A1+A5
A2 A2-1 TRAILER ADDRESS

MEMDE10 = *
S0 S1-S3
JSZ MEMDE20 JUMP IF AREA IS FIRST IN POOL
A4 A1-1 ADDRESS OF PRECEEDING TRAILER
GET.S0 S6&S7, MPS, A4
JSN MEMDE20 JUMP IF PRECEEDING AREA IN USE

THE PRECEEDING AREA IS AVAILABLE. COMBINE THE TWO INTO ONE LARGE AREA.

GET.S5 S6&S7, MPSIZE, A4 SIZE OF PRECEEDING AREA
S4 S4+S5 COMBINED SIZE
A4 S5
A1 A1-A4 BASE ADDRESS OF LARGEST AREA

CREATE NEW HEADER AND TRAILER WORDS

MEMDE20 = *
PUT.S4 S6&S7, MPSIZE, A1
PUT.S4 S6&S7, MPSIZE, A2
A6 0 GOOD STATUS
UNLOCK J B00

DEER1 = *
A6 DEERR1
J B00

DEER2 = *
A6 DEERR2
J B00

DEER3 = *
A6 DEERR3
J B00
The Pool Table is an STP-resident table used for memory pool management.

### Pool Table

**HEADER**

<table>
<thead>
<tr>
<th>Field</th>
<th>Word</th>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PTMAX</td>
<td>0</td>
<td>58-63</td>
<td>Maximum valid memory pool number in system</td>
</tr>
</tbody>
</table>

**ENTRY**

<table>
<thead>
<tr>
<th>Field</th>
<th>Word</th>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PTSIZE</td>
<td>0</td>
<td>16-39</td>
<td>Size of the memory pool</td>
</tr>
<tr>
<td>PTBASE</td>
<td>0</td>
<td>40-63</td>
<td>Base address of the memory pool</td>
</tr>
</tbody>
</table>

### MEMORY POOL - POOL1...n

Memory pool areas are surrounded by header and trailer words that control the allocation and deallocation of the areas.

**Memory Pool**

<table>
<thead>
<tr>
<th>Field</th>
<th>Word</th>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPST</td>
<td>0,n,etc.</td>
<td>0</td>
<td>Status of the memory area:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0 Available</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 In use</td>
</tr>
<tr>
<td>MPID</td>
<td>0,n,etc.</td>
<td>16-39</td>
<td>Memory pool identification:</td>
</tr>
<tr>
<td>MPSIZE</td>
<td>0,n,etc.</td>
<td>40-63</td>
<td>Size of the memory pool</td>
</tr>
</tbody>
</table>
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
**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
COMMON ROUTINES

EJECT

**
**
**NAME CHAIN
**
**PURPOSE CHAIN WILL ADD AN ITEM TO AN EXISTING QUEUE. ITEMS ARE
**ALWAYS ADDED AT THE END OF THE QUEUE.
**
**ENTRY A6 = ADDRESS OF CHAIN CONTROL WORD
**A7 = ADDRESS OF THE ITEM TO ADD TO THE CHAIN
**
**EXIT A6 = UNCHANGED
**A7 = UNCHANGED
**
**
CHAIN = *
S1 0
PUT.S1 S8&S7.CIFL,A7
PUT.S1 S8&S7.CIBL,A7 JUST TO BE SAFE
S1 'A6
PUT.S1 S8&S7.CICC.A7 SAVE CHAIN CONTROL ADDRESS
GET.S1 S8&S7.CCTAIL,A6
S2 A7
PUT.S2 S8&S7.CCTAIL,A6 NEW TAIL ADDRESS
PUT.S1 S8&S7.CIBL,A7 BACKWAR LINK = OLD TAIL
S0 S1
JSN CHAIN10

* THE CHAIN WAS EMPTY. THIS IS THE ONLY ITEM ON QUEUE
*
PUT.S2 S8&S7.CCHEAD.A6 HEAD = TAIL
J CHAIN20
CHAIN10 = *
A5 S1
PUT.S2 S8&S7.CIFL,A5 FORWARD LINK OLD TAIL = NEW TAIL
CHAIN20 = *
J R00
**COMMON SUBROUTINES**

**CHAINF**

**PURPOSE** CHAINF will add an item to the front of an existing queue.

**ENTRY**

- `A6` = Address of chain control word
- `A7` = Address of the item to be added

**EXIT**

- `A6` = UNCHANGED
- `A7` = UNCHANGED

**REGISTERS MODIFIED** - `S0, S1, S2, S6, S7, A6`

**CHAINF** = *

```
S1 1
SPLK, 0 S1 LOCK OUT INTERRUPTS
GET, S2 S6 & S7, CHEAD, A6
S0 S2
S1 A7
JSZ CHAINF1
SPUT, S1 S6 & S7, CHEAD, A6
A5 S2
PUT, S1 S6 & S7, CBL, A5
J CHAINF2
```

**CHAINF1** = *

```
SET, S1 S6 & S7, CHEAD
SET, S1 S6 & S7, CCTAIL
WCCHEAD, A6 S6
```

**CHAINF2** = *

```
S1 A6
PUT, S1 S6 & S7, CICL, A7 CHAIN CONTROL POINTER
S6 WACIFL, A7
SET, S2 S6 & S7, CIFL
S1 0
SET, S1 S6 & S7, CIBL
WACIFL, A7 S6
UNLOCK
J B0
ERR IF WACIFL, NE, W0CIBL
ERR IF WACHEAD, NE, W0CCTAIL
```

3.35
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
STP
COMMON SUBROUTINES

EJECT
******************************************************************************
**
**NAME. UNCHAIN
**
**PURPOSE UNCHAIN WILL REMOVE AN ITEM FROM AN EXISTING QUEUE. THE ITEM
** TO BE REMOVED MAY OCCUPY ANY POSITION IN THE CHAIN.
**
**ENTRY A7 = ADDRESS OF ITEM TO BE REMOVED
**
**EXIT A0 = STATUS
**
**K 0 = GOOD RETURN
**
**K 1 = ITEM IS NOT ON A CHAIN
**
**K A7 = ADDRESS OF ITEM REMOVED
**
**K
******************************************************************************
UNCHAIN1 = 1
UNCHAIN = *
GET, S1 8885,CICC,A7 CHAIN CONTROL WORD ADDRESS
S0 S1
A6 S1
A0 UNCHAIN1
JSZ UNCH50 JUMP IF ITEM NOT ON A CHAIN
S1 1
STPLK,A0 S1 LOCK OUT INTERRUPTS
GET, S1 8885,CHEAD,A6 HEAD OF CHAIN
S2 A7 ADDRESS OF ITEM TO REMOVE
GET, S2 8885,CIFL,A7 FORWARD LINK OF ITEM
GET, S4 8885,CIBL,A7 BACKWARD LINK OF ITEM
S0 S1-S2
JSN UNCH10 JUMP IF ITEM NOT HEAD OF CHAIN
PUT, S3 8885,CHEAD,A6 ITEM FORWARD LINK = NEW HEAD
UNCH10 = *
GET, S1 8885,CTAIL,A6 TAIL OF CHAIN
S0 S1-S2
JSN UNCH20 JUMP IF ITEM NOT TAIL OF CHAIN
PUT, S4 8885,CTAIL,A6 ITEM BACKWARD LINK = NEW TAIL
UNCH20 = *
S0 S3
JSZ UNCH30 JUMP IF FORWARD LINK IS ZERO
A2 S3
PUT, S4 8885,CIBL,A2 NEW BACKWARD LINK FOR FOLLOWING ITEM
UNCH30 = *
S0 S4
JSZ UNCH40 JUMP IF BACKWARD LINK IS ZERO
A2 S4
PUT, S3 8885,CIFL,A2 NEW FORWARD LINK FOR PRECEDING ITEM
UNCH40 = *
UNLOCK
PUT, S1 8885,CICC,A7 CLEAR CHAIN CONTROL
A0 0 GOOD STATUS
UNCH50 = *
J 800
Intertask communication requires chain control words in the format defined.

<table>
<thead>
<tr>
<th>Field</th>
<th>Word</th>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CCTM</td>
<td>0</td>
<td>0-7</td>
<td>Maximum number of items to be queued to a particular task</td>
</tr>
<tr>
<td>CCQM</td>
<td>0</td>
<td>0-7</td>
<td>Maximum number of items to be queued from one task to another</td>
</tr>
<tr>
<td>CCTL</td>
<td>0</td>
<td>8-15</td>
<td>Number of items queued to a particular task</td>
</tr>
<tr>
<td>CCQL</td>
<td>0</td>
<td>8-15</td>
<td>Number of items to be queued from one task to another</td>
</tr>
<tr>
<td>CCHEAD</td>
<td>0</td>
<td>16-39</td>
<td>Address of first item on the chain</td>
</tr>
<tr>
<td>CCTAIL</td>
<td>0</td>
<td>40-63</td>
<td>Address of last item on the chain</td>
</tr>
</tbody>
</table>

CHAIN ITEM - CI

Any item queued using the STP common routines CHAIN and UNCHAIN must reserve the first two words of the item to be used by the common routines.

<table>
<thead>
<tr>
<th>Field</th>
<th>Word</th>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CIEX</td>
<td>0</td>
<td>0</td>
<td>This bit, if set, indicates that the item is in execution.</td>
</tr>
<tr>
<td>CIFL</td>
<td>0</td>
<td>16-39</td>
<td>Forward link; address of next item on the chain</td>
</tr>
<tr>
<td>CIBL</td>
<td>0</td>
<td>40-63</td>
<td>Backward link; address of the preceding item on the chain</td>
</tr>
<tr>
<td>CICC</td>
<td>1</td>
<td>40-63</td>
<td>Address of the chain control word for this item</td>
</tr>
</tbody>
</table>
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)
   ALLOCATED AS NEEDED FROM A POOL
   ALL TASK REQUESTS ARE THROUGH A CMOD.
   ALL TASK REPLIES ARE THROUGH A CMOD.
   2 WORDS FOR SYSTEM 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

COMMUNICATION MODULES

CMOD #1
TASK 2 TO 1

CMOD #2
TASK 2 TO 1

CMOD N
TASK 2 TO 1

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

CHAIN
ITEM

- -

INPUT

- -

OUTPUT

3.43
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, TSKREQ 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

OUTPUT REGISTERS: None
EJECT

**COMMON SUBROUTINES**

PURPOSE PUTREQ QUEUES A REQUEST FOR PROCESSING TO ANOTHER TASK.

NAME PUTREQ

ENTRY A1 = THROW AWAY INDICATOR

A2 = DESTINATION TASK ID

A1 = NEGATIVE THROW AWAY REQUEST INSTEAD OF SUSP

A1 = POSITIVE SUSPEND UNTIL REQUEST QUEUED

S1 = INPUT+0

S2 = INPUT+1

EXIT

PUTREQ =

A6 = LE@SAVER

A7 = CTID,A0

AS = A7

A7 = SAVER

A6 = A7

SAVEA1,A6 A1

SAVEA2,A6 A2

SAVEA3,A6 A3

SAVEA4,A6 A4

SAVEA5,A6 A5

SAVE50,A6 S0

SAVE51,A6 S1

SAVE52,A6 S2

SAVE53,A6 S3

SAVE54,A6 S4

SAVE55,A6 S5

SAVE56,A6 S6

SAVE57,A6 S7

A7 = B00

SAVERT,A6 A7

S8 = 0

SAVEA8,A6 S8

SAVEA9,A6 S9

A3 = A6

S0 = S1

ERROR IF INPUT31 = 0

PUTREQ10 =

A6 = CNPOOL

A7 = CMSIZE

R = MEMAL

A6 = SAVERT,A3

B00 = AS

A0 = A6

J12 = PUTREL020

A7 = MISTAKE3

A0 = A6-A7

ERROR
MEMAL RETURNED A THREE STATUS--THERE IS NO MEMORY AVAILABLE.

SUSPEND AND TRY AGAIN LATER.

A0 SAVEA1.A3
JAM PUTREQ50 THROW AWAY THE REQUEST
S? SUBF
EX
R ERRORR0
J PUTREQ10

PUTREQ50 = *
S1 SAVEA1.A3
S2 SAVEA2.A3
CMIN.A7 S1
CMIN.A7 S2
POST 12.S1.S2

* * CALCULATE ADDRESS OF CHAIN CONTROL HEADER
* * CHAIN CONTROL HEADER = DESTINATION TASK ID * LENGTH OF CHAIN
* * CONTROL ENTRY + CHAIN CONTROL BASE
* *
A6 SAVEA2.A3
A5 LDA CMCC
A5 ASX4A6
A5 CMCC
A5 ASX4A6

CHAIN CONTROL HEADER ADDRESS

* * CALCULATE THE CHAIN CONTROL WORD ADDRESS
* * CHAIN CONTROL = CHAIN CONTROL HEADER + LENGTH OF HEADER +
* * CURRENT TASK ID
* *
A6 LH@CMCC
A6 ASX+45
A1 CTID.A0
A6 ASX4A6

CHAIN CONTROL ADDRESS

PUTREQ21 = *
S1 0
STPLK.A0 A1
GET,S1 S8&S7.CCTL.A6
GET,S2 S8&S7.CCTM.A6
S0 S2
JNZ PUTREQ22 JUMP IF QUEUE AT MAX
S4 1
S1 S4+S1
GET,S2 S8&S7.CCGL.A6
GET,S3 S8&S7.CCOMM.A6
S0 S3
JNZ PUTREQ22 JUMP IF QUEUE AT MAX
S2 S2+44

* * UPDATE THE QUEUE COUNTS
* * PUT,S1 S8&S7.CCTL.A6
* * PUT,S2 S8&S7.CCGL.A6
J PUTREQ25

* * THE QUEUE IS AT ITS MAX. EITHER SUSPEND OR THROW THE REQUEST
* * AWAY.
* *
PUTREQ22 = *
S1 0
DPLK.A0 A1
A0 SAVEA1.A3
JAM PUTREQ23 JUMP TO THROW THE REQUEST AWAY
S7 SUBP
EX
R ERRORR0
J PUTREQ21

3.48
GETREQ SEARCHES FOR AN ACTIVE REQUEST FOR THE CALLER. 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
(S2) = INPUT+1
SELECT

**Purpose**: GETREQ searches for an outstanding request to the task.

**Usage**: The highest priority request will be returned to the originating task. Priority corresponds to task ID.

**Entry**

- \( A_0 \) = FIND/NO FIND INDICATOR
- \( 0 \) = NO FIND
- \( 1 \) = FIND
- \( A_2 \) = TASK ID of the request located
- \( S_1 \) = INPUT-2
- \( S_2 \) = INPUT+1

**Exit**

- \( A_6 \) = ENTRY

**Variables**

- \( A_6 \) = LEBSAVE
- \( A_7 \) = CTID.A0
- \( A_9 \) = ASKP7
- \( A_7 \) = SAVR
- \( A_9 \) = ASKPV

**Note**: Save those registers not already destroyed.

**Assume no find will occur**

**Code**

1. Calculate the address of chain control header:
   
   \[ \text{CHAIN CONTROL HEADER} = \text{CURRENT TASK ID} \times \text{CHAIN CONTROL LENGTH} \]

   OF ENTRY + BASE OF CHAIN CONTROL

2. \( A_1 \) = CTID.A0
3. \( A_2 \) = LEBMCC
4. \( A_1 \) = A1K42
5. \( A_2 \) = CYCC
6. \( A_1 \) = A1K42

7. \( \text{GET7.S0} \) = SAVS7.CTL.A1
8. \( \text{JSE} \) = GETREQ30

3.51
SEARCH EACH CHAIN CONTROL WORD FOR AN OUTSTANDING REQUEST

A2 LH@CMCC
A1 A1+42
A2 0 WILL CONTAIN TASK ID OF FIND ON EXIT
A4 WAXTH

GETREQ10 = *
GET.S0 $8&57,CC00L.A1
JSN GETREQ20 JUMP IF ITEMS ON CHAIN

GETREQ15 = *
A1 A1+1 THIS CHAIN IS EMPTY GO TO THE NEXT
A2 A2+1
A0 A4-A2
JAN GETREQ10 JUMP IF ALL CHAINS NOT EXAMINED
J GETREQ30 THERE ARE NO REQUESTS FOR THE TASK

A CHAIN WITH ITEMS HAS BEEN LOCATED

GETREQ20 = *
GET.S1 $8&57.CHEAD.A1
GET.S2 $8&57.CTAIL.A1

GETREQ22 = *
AS S1 CMOD ADDRESS
GET.S0 $8&57.CIEX.A5
JSZ GETREQ25 JUMP IF REQUEST IS NOT IN EXECUTION

THE TASK IS EXECUTING THIS ITEM

S0 S1-S2
JSZ GETREQ15 JUMP IF ALL ITEMS IN EXECUTION
GET.S1 $8&57.CIFL.A5 ADDRESS OF NEXT CMOD
J GETREQ22

A REQUEST TO PROCESS HAS BEEN LOCATED

GETREQ25 = *
SAVEA2.A0 A2 SAVE TASK ID OF REQUEST
S1 1
SAVEA0.A0 S1 SET FIND INDICATOR
PUT.S1 $8&57.CIEX.A5 SET EXECUTE INDICATOR
S1 CMIN0.A5
S2 CMIN1.A5
S0 S1
EPRSZ ERROR IF INPUT.S1 = 0
SAVE.S1.A3 S1
SAVE.S2.A3 S2

RESTORE REGISTERS AND EXIT.

GETREQ30 = *
R8 A3
R9 SAVEA0.A6
R1 SAVEA1.A6
R2 SAVEA2.A6
R3 SAVEA3.A6
R4 SAVEA4.A6
R5 SAVEA5.A6
R6 SAVEA6.A6
R7 SAVEA7.A6
J B00 3.52
PUTREPLY PLACES A REPLY IN THE APPROPRIATE CMOD WHEN A TASK COMPLETES PROCESSING. 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
**COMMON SUBROUTINES**

EJECT

******************************************************************************

* NAME PUTREPLY

* PURPOSE PUTREPLY SAVES OUTPUT IN THE APPROPRIATE CMOD WHEN A TASK
  COMPLETES PROCESSING

* ENTRY A2 = ID OF THE TASK TO RECEIVE THE REPLY
  S1 = OUTPUT+6
  S2 = OUTPUT+1

* EXIT

* 

******************************************************************************

PUTREPLY =

A6 LEA SAVES
A7 CTID.A0 ORIGINATING TASK ID
A6 ASR A7
A7 SAV
A6 ASHA7 ADDRESS OF SAVE AREA FOR ORIGIN TASK

SAVE A0
SAVE A1 A2
SAVE A3 A4
SAVE A5 A6
SAVE A7 A8
SAVE A9 A10
SAVE A11 A12
SAVE A13 A14
SAVE A15 A16
SAVE A17 A18
SAVE A19 A20
SAVE A21 A22
SAVE A23 A24
SAVE A25 A26
SAVE A27 A28
SAVE A29 A30
SAVE A31 A32
SAVE A33 A34
SAVE A35 A36
SAVE A37 A38
SAVE A39 A40

A7 R00
SAVETR A7
S6 0
SAVEZ A6 A5
SAVE A9 A8
SAVE A3 A7
S0 S2

ERRSZ ERROR IF 2ND WORD OF REPLY IS ZERO

CALCULATE CHAIN CONTROL HEADER AND CHAIN CONTROL WORD ADR
CHAIN CONTROL HEADER = BASE OF CMOD CHAIN CONTROL + (LENGTH
OF AN ENTRY X CURRENT TASK ID)
CHAIN CONTROL WORD = CHAIN CONTROL HEADER + LENGTH OF HEADER
+ RECEIVING TASK ID

A1 CTID.A0
A6 LEA A1
A6 ASRA1
A1 CMCC
A6 ASRA1 CHAIN CONTROL HEADER
A7 LHRBMC
A7 A7 R4A6
A7 A7 R4A7 CHAIN CONTROL WORD
GET S0 S8 S7 CCOL A7
ERRSZ
GET S5 S8 S7 CHEAD A7
GET S4 S8 S7 CCTAIL A7

3.54
PUTREP10 = *
AS SS CMOD ADDRESS
SQ CMCOUT1,AS
JSZ PUTREP20 JUMP IF CMOD HAS NO REPLY
*
A REPLY EXISTS IN THIS CMOD. GO TO THE NEXT CMOD IF THERE IS ONE
*
S0 S4-S5
ERRSZ
GET.S5 S8&57.CIFL,AS NEXT CMOD ADDRESS
J PUTREP10
*
AT THIS POINT WE HAVE FOUND A CMOD FOR THE REPLY
*
PUTREP20 = *
A1 1
STPLK,A0 A1 SET INTERRUPT LOCK-OUT
CMOUT0,AS S1
CMOUT1,AS S2
*
DECREMENT THE NUMBER OF ITEMS ON QUEUE
*
GET,S1 S9&57.CPGL,A7
S4 1
S1 S1-S4
PUT.S1 S9&57.CPGL,A7
GET.S1 S9&57.CTTL,A6
S1 S1-S4
PUT.S1 S9&57.CPGL,A6
N1 0
STPLK,A0 A1
*
READY THE RECEIVING TASK
*
S1 CMOUT0,AS
POST 12.51,52
S6 A2
S7 RTSK
EX
R ERROR
*
RESTORE THE REGISTERS AND EXIT
*
A6 A3
A5 SAVEA8,A6
A4 SAVEA1,A8
A3 SAVEA2,A6
A2 SAVEA3,A6
A1 SAVEA4,A6
A0 SAVEA5,A6
S9 SAVE30,A6
S8 SAVE31,A6
S7 SAVE32,A6
S6 SAVE33,A6
S5 SAVE34,A6
S4 SAVE35,A6
S3 SAVE36,A6
S2 SAVE37,A6
S1 SAVE38,A6
A7 SAVEF7,A6
B0-9 A7
B1-7 SAVEA7,A6
A6 SAVEA8,A6
J 270
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:**
- \((A0) = \text{Find indicator. If } (A0) = 0, \text{ no reply was located; if } (A0) \neq 0, \text{ a reply is being returned to the caller.}\)
- \((A2) = \text{ID of replying task}\)
- \((S1) = \text{OUTPUT} + 0 \) \(\) \text{Reply}\)
- \((S2) = \text{OUTPUT} + 1 \) \(\) \text{Reply}\)
EJECT

*NAME GETREPLY

*PURPOSE GETREPLY LOCATES AND RETURNS THE HIGHEST PRIORITY REPLY TO
A TASK. THE PRIORITY STRUCTURE CORRESPONDS TO THE TASK ID

*ENTRY

*EXIT A0 = FIND: NO FIND INDICATOR
0 = NO FIND
1 = FIND
A2 = ID OF TASK WHERE REPLY FOUND
S1 = OUTPUT+0
S2 = OUTPUT+1

*SAVE THE REGISTERS CLEARING THOSE ALREADY DESTROYED

SAVEA1.A6 A1
SAVEA2.A6 A2
SAVEA3.A6 A3
SAVEA4.A6 A4
SAVEA5.A6 A5
SAVEA6.A6 A6
SAVEA7.A6 A7
SAVEA8.A6 A8
SAVEA9.A6 A9
SAVEA10.A6 A10
SAVEA11.A6 A11
SAVEA12.A6 A12
SAVEA13.A6 A13
SAVEA14.A6 A14
SAVEA15.A6 A15
SAVEA16.A6 A16
SAVEA17.A6 A17
SAVEA18.A6 A18
SAVEA19.A6 A19
SAVEA20.A6 A20
SAVEA21.A6 A21
SAVEA22.A6 A22
SAVEA23.A6 A23
SAVEA24.A6 A24
SAVEA25.A6 A25
SAVEA26.A6 A26
SAVEA27.A6 A27
SAVEA28.A6 A28
SAVEA29.A6 A29
SAVEA30.A6 A30
SAVEA31.A6 A31
SAVEA32.A6 A32
SAVEA33.A6 A33
SAVEA34.A6 A34
SAVEA35.A6 A35
SAVEA36.A6 A36
SAVEA37.A6 A37
SAVEA38.A6 A38
SAVEA39.A6 A39
SAVEA40.A6 A40
SAVEA41.A6 A41
SAVEA42.A6 A42
SAVEA43.A6 A43
SAVEA44.A6 A44
SAVEA45.A6 A45
SAVEA46.A6 A46
SAVEA47.A6 A47
SAVEA48.A6 A48
SAVEA49.A6 A49
SAVEA50.A6 A50
SAVEA51.A6 A51
SAVEA52.A6 A52
SAVEA53.A6 A53
SAVEA54.A6 A54
SAVEA55.A6 A55
SAVEA56.A6 A56
SAVEA57.A6 A57
SAVEA58.A6 A58
SAVEA59.A6 A59
SAVEA60.A6 A60
SAVEA61.A6 A61
SAVEA62.A6 A62
SAVEA63.A6 A63
SAVEA64.A6 A64
SAVEA65.A6 A65
SAVEA66.A6 A66
SAVEA67.A6 A67
SAVEA68.A6 A68
SAVEA69.A6 A69
SAVEA70.A6 A70
SAVEA71.A6 A71
SAVEA72.A6 A72
SAVEA73.A6 A73
SAVEA74.A6 A74
SAVEA75.A6 A75
SAVEA76.A6 A76
SAVEA77.A6 A77
SAVEA78.A6 A78
SAVEA79.A6 A79
SAVEA80.A6 A80
SAVEA81.A6 A81
SAVEA82.A6 A82
SAVEA83.A6 A83
SAVEA84.A6 A84
SAVEA85.A6 A85
SAVEA86.A6 A86
SAVEA87.A6 A87
SAVEA88.A6 A88
SAVEA89.A6 A89
SAVEA90.A6 A90
SAVEA91.A6 A91
SAVEA92.A6 A92
SAVEA93.A6 A93
SAVEA94.A6 A94
SAVEA95.A6 A95
SAVEA96.A6 A96
SAVEA97.A6 A97
SAVEA98.A6 A98
SAVEA99.A6 A99
SAVEAA.A6 A100

ASSUME NO FIND WILL OCCUR

*CALCULATE CHAIN CONTROL ADDRESS

CHAIN CONTROL = CURRENT TASK ID + CHAIN CONTROL BASE + LENGTH
OF CHAIN CONTROL HEADER

A6 CMCC
A1 CTID.A0
A6 A6+1
A5 L-3CMCC
A6 A6+5
A2 0
A4 MAXTN
A5 L2CMCC
GETREP10 =  *  
GET, S1 $3487.CHEAD,A6  
S0   $1  
JSZ  GETREP20  
A7   $1  ADDRESS OF THE CMOD  
S0   CHOUT1.A7  
JSN  GETREP30  JUMP IF A REPLY EXISTS  
*  
*  THERE IS NO REPLY FROM THIS TASK. UPDATE TO NEXT TASK  
*  
GETREP20 =  *  
A6   A6+A6  NEXT CHAIN CONTROL  
A2   A2+1  NEXT TASK ID  
A0   A1=A2  
JAN  GETREP10  JUMP IF ALL CHAINS NOT EXAMINED  
*  
*  AT THIS POINT THE SEARCH IS COMPLETE WITH NO REPLY FOUND  
*  
J   GETREP40  
GETREP30 =  *  
SAVEA2,A3 A2  SAVE TASK ID OF FIND  
A0   1  
SAVEA0,A3 A0  
S1   CHOUT0.A7  
S2   CHOUT1.A7  
SAVE1,A3 S1  
SAVE2,A3 S2  MOVE THE REPLY OUT OF THE CMOD  
R    UNCHAIN REMOVE IT FROM THE CHAIN  
A5   SAVERT.A3 RESTORE B00 FOR TRACE ENTRIES  
B00  A5  
ERRAN  
A6   CMPPOOL DEALLOCATE THE MEMORY  
R    MEMDE  
A5   SAVERT.A3 RESTORE B00 FOR TRACE ENTRIES  
B00  A5  
A0   A5  
ERRAN.  
*  
*  RESTORE THE REGISTERS AND EXIT  
*  
GETREP40 =  *  
R6   A3  
R0   SAVEA0.A6  
R1   SAVEA1.A6  
R2   SAVEA2.A6  
R3   SAVEA3.A6  
R4   SAVEA4.A6  
R5   SAVEA5.A6  
S0   SAVE00.A6  
S1   SAVE1.A6  
S2   SAVE2.A6  
S3   SAVE3.A6  
S4   SAVE4.A6  
S5   SAVE5.A6  
S6   SAVE6.A6  
S7   SAVE7.A6  
A7   SAVERT.A6  
B00  A7  
A7   SAVEA7.A6  
R6   SAVEA6.A6  
J   B30  

3.58
○ TSKREQ QUEUES A REQUEST TO ANOTHER TASK.

○ TSKREQ IS CALLED VIA A RETURN JUMP WITH THE CALLER PROVIDING THE FOLLOWING:

INPUT REGISTERS:  
(\(a_2\)) = ID OF REQUESTED TASK  
(\(s_1\)) = input+0  
(\(s_2\)) = input+1  
\{ Request

OUTPUT REGISTERS:  
(\(s_1\)) = output+0  
(\(s_2\)) = output+1  
\{ Reply

○ ONCE THE REQUEST HAS BEEN PROCESSED, THE CALLER MAY EXAMINE ITS \(s_1,s_2\) REGISTERS FOR A REPLY. CONVENTIONALLY, \(s_1=0\) WHEN THERE IS NO ERROR, OTHERWISE \(s_1=\text{ERR CODE}\). \(s_2=\text{THE CALLING TASKS INPUT+0 REGISTER (S1) INFORMATION.}\)
**EJECT**

```
**NAME** TSKREQ

**PURPOSE** TSKREQ QUEUES A REQUEST FOR PROCESSING TO ANOTHER TASK. THE ORIGINATING TASK IS SUSPENDED UNTIL THE DESTINATION TASK HAS COMPLETED PROCESSING.

**ENTRY**
A2 = DESTINATION TASK ID
S1 = INPUT+0
S2 = INPUT+1

**EXIT**
S1 = OUTPUT+0
S2 = OUTPUT+1

**EXIT**

**SOURCE**

TSKREQ = x
A2 = LEASER
A4 = CTID.A0
A3 = A3+A4
A4 = SAVC
A3 = A3+A4

SAVEA0.A0 A0
SAVEA1.A0 A1
SAVEA2.A0 A2
SAVEA5.A0 A5
SAVEA6.A0 A6
SAVEA7.A0 A7
SAVEA0.A1 A0
SAVEA1.A1 A1
SAVEA2.A1 A2
SAVEA5.A1 A5
SAVEA6.A1 A6
SAVEA7.A1 A7
SAVEA0.A2 A0
SAVEA1.A2 A1
SAVEA2.A2 A2
SAVEA5.A2 A5
SAVEA6.A2 A6
SAVEA7.A2 A7
SAVEA0.A5 A0
SAVEA1.A5 A1
SAVEA2.A5 A2
SAVEA5.A5 A5
SAVEA6.A5 A6
SAVEA7.A5 A7
SAVEA0.A6 A0
SAVEA1.A6 A1
SAVEA2.A6 A2
SAVEA5.A6 A5
SAVEA6.A6 A6
SAVEA7.A6 A7
SAVEA0.A7 A0
SAVEA1.A7 A1
SAVEA2.A7 A2
SAVEA5.A7 A5
SAVEA6.A7 A6
SAVEA7.A7 A7
SAVEA0.S0 S0
SAVEA1.S0 S1
SAVEA2.S0 S2
SAVEA5.S0 S5
SAVEA6.S0 S6
SAVEA7.S0 S7
SAVEA0.S1 S0
SAVEA1.S1 S1
SAVEA2.S1 S2
SAVEA5.S1 S5
SAVEA6.S1 S6
SAVEA7.S1 S7
S6 = 0
SAVEA3.S8 S8
SAVEA4.S8 S9
SAVEA5.A4 B00
SAVEA6.A4 S0
SAVEA7.A4 S1
ERROR A5 A4

ERRS2 = x
ERROR IF INPUT31 = 0

TSKREQ10 =
A6 = CMPOOL
A7 = CMSIZE
R = MEMAL
A5 = SAVCRA
B00 = A5

**CHECK STATUS RETURNED FROM MEMAL**
A0 = A6
J 2 TSKREQ20
J 3
A3 = A3-A7
```
AT THIS POINT THERE IS NO MEMORY AVAILABLE. SUSPEND AND
TRY AGAIN LATER.

S7     SUSP
EX
R     ERR:R0
J     TSKREQ10

A CMOD HAS BEEN ALLOCATED BUT THE INPUT MESSAGE IN THE CMOD

TSKREQ20 = *
S1     SAVES1.A3
S2     SAVES2.A3
CMIN0.A7 S1
CMIN1.A7 S2
POST   12.31.S2

CALCULATE THE ADDRESS OF THE CHAIN CONTROL HEADER
CHAIN CONTROL HEADER = DESTINATION TASK ID * LENGTH OF CHAIN
  CONTROL ENTRY + BASE OF CHAIN CONTROL

A5     SAVEA2.A3
A4     LE@CMCC
A5     A4+AS
A4     CVCC
A5     A4+AS     CHAIN CONTROL HEADER ADDRESS

CALCULATE CHAIN CONTROL WORD ADDRESS
CHAIN CONTROL WORD = CHAIN CONTROL HEADER + LENGTH OF CHAIN
  CONTROL HEADER + CURRENT TASK ID

A6     LE@CMCC
A6     A6+A6
A4     CTID.A0
A6     A6+A4     CHAIN CONTROL WORD ADDRESS

TSKREQ30 = *
A1     1
STPLK.A0 A1     LOCK OUT INTERRUPTS
GET,S1 SS&L7.COCL.A5
GET,S2 SS&L7.COCL.A5
S0     S2
ERRSZ   S1-S2     ERROR IF QUEUE MAX IS ZERO
S0     S2
JSZ    TSKREQ70     JUMP IF QUEUE AT LIMIT
S4     1
S1     S1+S4
GET,S2 SS&L7.COCL.A6
GET,S3 SS&L7.COCL.A6
S9     S3
ERRSZ   S2-S3
S0     S2-S3
JSZ    TSKREQ70     JUMP IF QUEUE AT LIMIT
S2     S2+S4
PUT,S1 SS&L7.COCL.A5
PUT,S2 SS&L7.COCL.A6
R     CHAIN
A5     SAVERT.A3     RESTORE B00 FOR TRACE ENTRIES
B00   A5
A1     0
STPLK.A0 A1     ENABLE INTERRUPTS

3.61
THE REQUEST HAS BEEN QUEUED TO THE DESTINATION TASK READY THE TASK AND SUSPEND UNTIL A REPLY IS RECEIVED

TSKREQ40 = X
S6 SAVEA2.A3
S7 RTSS
EX R ERROR0

TSKREQ50 = X
S1 CMOUT0.A7
S2 CMOUT1.A7
S0 $2
JSN TSKREQ60

THE DESTINATION TASK DID NOT REACTIVATE ME

S7 SUSP
EX R ERROR0
J TSKREQ50

S6, S7 SAVEA3
S0, S1 SAVEA3
POST 12.S1.S2
R UNCHAIN UNCHAIN THE CMOD
A5 SAVERT.A3 RESTORE B00 FOR TRACE ENTRIES
B00 A6 ERRAN

A6 CMPOOL RELEASE THE CMOD MEMORY
R MEMDE
A5 SAVERT.A3 RESTORE B00 FOR TRACE ENTRIES
B00 A6 ERRAN
A6 A6 STATUS CHECK

RESTORE THE REGISTERS AND EXIT

A0 SAVEA0.A3
A1 SAVEA1.A3
A2 SAVEA2.A3
A5 SAVEA5.A3
A6 SAVEA6.A3
A7 SAVEA7.A3
S0 SAVEA0.A3
S1 SAVEA1.A3
S2 SAVEA2.A3
S3 SAVEA3.A3
S4 SAVEA4.A3
S5 SAVEA5.A3
S6 SAVEA6.A3
S7 SAVEA7.A3
A4 SAVERT.A3
B00 A4
A4 SAVEA4.A3
A3 SAVEA3.A3
J B00

TSKREQ70 = X

THE QUEUE FOR THE DESTINATION TASK IS AT ITS LIMIT THE REQUESTOR WILL HAVE TO WAIT.

A1 0
STPLK.A0 A1 RELEASE INTERRUPT LOCKOUT
S7 SUSP
EX R ERROR0
J TSKREQ30
USER TO STP CALLS

- Advance Job
- Abort Job
- Get Current Date
- Get Current Time
- Enter Logfile Message
- Dataset Recall
- Terminate Job
- Set Sense Switch
- Open Dataset
- Request Memory
- Close Dataset
- Create DNT
- Set Exchange Package Mode
- Get Next Control Statement
- Load Binary Dataset
- Return Dataset
- Permanent Dataset Management Request
- Read Disc Circular
- Write Disc Circular

- Get System Revision Numbers
- Dispose Dataset
- Get Current Julian Date
- Return Accumulated CPU Time
- Return Accounting Information
- Set P register & Suspend User
- Clear Sense Switch
- Test Sense Switch
- Delay Job

A user communicates with STP through macro calls.
STARTUP (Z)
CURRENT TASKS ARE:

- CRAY-OS STARTUP
- STATION CALL PROCESSOR
- DISK QUEUE MANAGER
- PERMANENT DATASET MANAGER
- JOB CLASS MANAGER
- JOB SCHEDULER
- EXCHANGE PACKAGE PROCESSOR
- MESSAGE PROCESSOR
- MEMORY ERROR PROCESSOR
- DISK ERROR CORRECTION
- SYSTEM PERFORMANCE MONITOR
- OVERLAY MANAGER
- TAPE QUEUE MANAGER
CRAY-OS STARTUP

- LOADS CRAY-OS OPERATING SYSTEM (COS) INTO MEMORY.
- BEGINS EXECUTION OF THE OPERATING SYSTEM.
- GENERATES OR RECOVERS SYSTEM TABLES.
- THREE METHODS OF STARTUP.

INSTALL-INITIATION OF COS THE FIRST TIME

DEADSTART-CONTINUATION OF COS FOLLOWING A NORMAL SYSTEM SHUTDOWN

RESTART-CONTINUATION OF COS FOLLOWING A SYSTEM INTERRUPTION.
INSTALL

- LOADS THE CRAY-1 OPERATING SYSTEM (COS) INTO CRAY-1 MEMORY.
- CRAY-1 MASS STORAGE IS INITIALIZED FOR THE VERY FIRST TIME.
  A DEVICE LABEL (DVL) IS WRITTEN ON EACH DISK UNIT.
  SPACE IS ZEROED AND RESERVED ON THE MASTER DEVICE
  SUFFICIENT TO HOLD CRAY-1 MEMORY SIZE.
  A ROLL JOB INDEX DATASET IS INITIALIZED ($ROLL).

- SYSTEM TABLES ARE INITIALIZED FOR THE VERY FIRST TIME.
  REFLECT HOW MUCH USEABLE DISK SPACE IS AVAILABLE (DRT).
  CREATES A DISK DATASET CATALOG (DSC) AND WRITES THE DSC TO THE MASTER DEVICE.
  MAKES ENTRIES IN THE DSC FOR $ROLL

Mass Storage Organization
SYSTEM DIRECTORY INITIALIZATION

- The system directory (SDR) must be initialized after any install (automatic in deadstart and restart).

- Initialization occurs through submission of a high priority job.

- The job's priority should enable it to execute before any other job in the input queue.

- Once a dataset name is entered in the SDR it is accessible to other jobs.

- An 'access' control statement with parameter 'enter' is specified for each dataset being entered in the SDR.

Sample job:

```
JOB,JN=SYSDIR,T=2,M=20,P=15.
.*
* SDR initialization job
.*
ACCESS,ENTER,DN=AUDIT.
ACCESS,ENTER,DN=BUILD.
ACCESS,ENTER,DN=CAL.
ACCESS,ENTER,DN=CFT.
ACCESS,ENTER,DN=COMPARE.
ACCESS,ENTER,DN=COPYD.
ACCESS,ENTER,DN=COPYF.
ACCESS,ENTER,DN=COPYR.
ACCESS,ENTER,DN=DS DUMP.
ACCESS,ENTER,DN=DUMP.
ACCESS,ENTER,DN=EXTRACT.
ACCESS,ENTER,DN=FDUMP.
ACCESS,ENTER,DN=LDR.
ACCESS,ENTER,DN=PDS DUMP.
```

ACCESS,ENTER,DN=PDSLOAD.
ACCESS,ENTER,DN=SKIPD.
ACCESS,ENTER,DN=SKIPF.
ACCESS,ENTER,DN=SKIPR.
ACCESS,ENTER,DN=UNB.
ACCESS,ENTER,DN=UPDATE.
ACCESS,ENTER,DN=WRI TED S.
(EOF)
DEADSTART

- CONTINUES THE CRAY-1 OPERATING SYSTEM (COS) FOLLOWING A NORMAL SYSTEM SHUTDOWN.

- DELETES DSC ENTRIES FOR INPUT AND OUTPUT DATASETS. (SDT)

- PRESERVES DSC ENTRIES FOR PERMANENT DATASETS.

- PRESERVES DISK SPACE OCCUPIED BY THE SYSTEM DUMP.

- COPIES SYSTEM DUMP TO ANOTHER AREA IF USED AND MAKES IT A PERMANENT DATASET.

- REBUILDS THE SYSTEM DIRECTORY FROM DISK IF DESIRED.
RESTART

- CONTinues the cray-1 operating system (cos) following an abnormal system interruption.
- preserves dsc entries for input and output datasets.
- preserves dsc entries for permanent datasets.
- preserves disk space occupied by the system dump.
- copies system dump to another area if used and makes it a permanent dataset.
- preserves rolled jobs and associated datasets if required.
- rebuilds the system directory from disk if desired.
RECOVERY OF THE SYSTEM DIRECTORY DURING A RESTART

- An access of the $SDR permanent dataset is requested.

- If $SDR does not exist then the system creates $SDR as a permanent dataset.

- If $SDR exists a dataset name table (DNT) is created and the dataset is accessed.

- If the *SDR command is in the restart parameter file the previous $SDR dataset is not recovered and a new edition of $SDR is created as a permanent dataset.

- Once initialization of the $SDR dataset is complete, additional entries may be added to the system directory and the $SDR dataset.
RECOVERY OF ROLLED JOBS DURING A RESTART

A validation of the rolled job index table entry for a job ensures the job is not marked irrecoverable and the job resides on an available device.

The jobs job table area (JTA) is read into memory and the jobs dat's are verified as are all dat's within the system.

If a job's dat is invalid and the job is rerunnable the system clears the index entry and the job remains on the input queue in the system dataset table (SDT).

If all dat's for a job are valid the system sets the device reservation table (DRT) entries for the dat's.
DATASET ALLOCATION TABLE (DAT) VERIFICATION

- A DATASET ALLOCATION TABLE (DAT) MUST RESIDEentirely in the system for system datasets, and within a jobs job table area (JTA) for a job.

- THE DAT MUST POINT TO THE CORRECT JOB EXECUTION TABLE (JXT) ENTRY FOR A JOB (LOCAL) DATASET.

- THE DEVICES USED BY THE DATASETS MUST BE AVAILABLE.

- FOR A DAT THE LOGICAL TRACK ADDRESSES MUST NOT HAVE THE CORRESPONDING BIT SET IN THE DEVICE RESERVATION TABLE (DRT).

- WHEN THE LAST LOGICAL TRACK ADDRESS IS VERIFIED AGAINST THE DRT BIT THE REMAINING ADDRESS COUNT MUST BE ZERO.

- DAT VERIFICATION OCCURS IN TWO PASSES
  
  FIRST PASS SCANS FOR ERRORS AS ABOVE
  SECOND PASS SETS THE DRT BITS

- PERMANENT DATASETS ARE FURTHER VERIFIED AGAINST THE DISK RESIDENT DATASET CATALOG (DSC) AND EACH DATASET DAT IS COMPARED AGAINST THE DISK COPY.
STATION CALL PROCESSOR (SCP)
STATION CALL PROCESSOR (SCP)

- PROVIDES COMMUNICATIONS WITH FRONT-END SYSTEMS.
- MANAGES I/O TRANSMISSION POOLS CREATED AT STARTUP.
- PROVIDES FOR OPERATOR GUIDANCE.
- CHANNEL DISCIPLINE IS TWO WAY ALTERNATE.
- INDEPENDENT OF FRONT-END TYPE.
- MULTIPLEXES STREAMS (8 INPUT AND 8 OUTPUT).
GENERAL INTERFACE PROTOCOL

- Each message is headed by a LINK control package
- Subsegment size varies with front-end

```
TRANSMISSION_1  LCP
TRANSMISSION_2  SUBSEGMENT_1
TRANSMISSION_3  SUBSEGMENT_2
TRANSMISSION_4  SUBSEGMENT_3
    ...  ...  ...
TRANSMISSION_N  SUBSEGMENT_{N-1}
    ...  ...  ...
TRANSMISSION   L T P --- OPTIONAL
```
**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     | 16    | 24    | 32    | 40    | 48    | 56    | 63    |
+-------+-------+-------+-------+-------+-------+-------+-------+
| DID   | SID   | NSSG  | MN    | MC    | HSC   |
+-------+-------+-------+-------+-------+-------+-------+-------+
| STN   | SGN   | SGBC  |
+-------+-------+-------+-------+-------+-------+-------+-------+
| ISCB1 | ISCB2 | ISCB3 | ISCB4 | ISCB5 | ISCB6 | ISCB7 | ISCB8 |
+-------+-------+-------+-------+-------+-------+-------+-------+
| OSCB1 | OSCB2 | OSCB3 | OSCB4 | OSCB5 | OSCB6 | OSCB7 | OSCB8 |
+-------+-------+-------+-------+-------+-------+-------+-------+
```

5.5
<table>
<thead>
<tr>
<th>Code</th>
<th>Function</th>
<th>Sender</th>
<th>CRAY-1</th>
<th>Segment</th>
<th>Stream Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>Logon</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>002</td>
<td>Relog</td>
<td>x+++</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>003</td>
<td>Logoff</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>004</td>
<td>Start</td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>005</td>
<td>Restart</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>006</td>
<td>Dataset header</td>
<td>x</td>
<td>x</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>007</td>
<td>Dataset segment</td>
<td>x</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>011</td>
<td>Control</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>012</td>
<td>Message error</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>013</td>
<td>Dataset transfer request</td>
<td>x+++</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>014</td>
<td>Dataset transfer reply</td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>015</td>
<td>Enter logfile request</td>
<td>x+++</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>016</td>
<td>Enter logfile reply</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>020</td>
<td>Logfile information request</td>
<td>x+++</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>021</td>
<td>Job status request</td>
<td>x+++</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>022</td>
<td>System status request</td>
<td>x+++</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>023</td>
<td>Dataset status request</td>
<td>x+++</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>024</td>
<td>Link status request</td>
<td>x+++</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>025</td>
<td>Mass storage status request</td>
<td>x+++</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>026</td>
<td>Operator function request</td>
<td>x+++</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>027</td>
<td>Debug function request</td>
<td>x+++</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>030</td>
<td>Logfile information reply</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>031</td>
<td>Job status reply</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>032</td>
<td>System status reply</td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>033</td>
<td>Dataset status reply</td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>034</td>
<td>Link status reply</td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>035</td>
<td>Mass storage status reply</td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>036</td>
<td>Operator function reply</td>
<td></td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>037</td>
<td>Debug function reply</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>040</td>
<td>Diagnostic echo request</td>
<td>x+++</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>041</td>
<td>Diagnostic echo reply</td>
<td>x+++</td>
<td>x</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

++ Optional; front end not required to send
STREAM CONTROL BYTES

- 8 INPUT AND 8 OUTPUT STREAMS

- ALTHOUGH EACH MESSAGE IS ASSIGNED TO ONLY ONE STREAM, THE LCP MUST CARRY STREAM CONTROL BYTES FOR ALL 16 STREAMS.

<table>
<thead>
<tr>
<th>Octal Code</th>
<th>Mnemonic</th>
<th>Request/Response</th>
<th>Sender</th>
<th>Receiver</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>IDL</td>
<td>Idle</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>01</td>
<td>RTS</td>
<td>Request to send</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>02</td>
<td>PTR</td>
<td>Preparing to receive</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>03</td>
<td>SND</td>
<td>Sending</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>04</td>
<td>RCV</td>
<td>Receiving</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>05</td>
<td>SUS</td>
<td>Suspend</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>06</td>
<td>END</td>
<td>End dataset</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>07</td>
<td>SVG</td>
<td>Saving dataset</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>10</td>
<td>SVD</td>
<td>Dataset saved</td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>11</td>
<td>PPN</td>
<td>Postpone</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>12</td>
<td>CAN</td>
<td>Cancel</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>13</td>
<td>MCL</td>
<td>Master clear</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>SENDER SCB SENT</td>
<td>IDL</td>
<td>PTR</td>
<td>RCV</td>
<td>SUS</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
<td>-----</td>
</tr>
<tr>
<td>IDL</td>
<td>N</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RTS</td>
<td>N</td>
<td>C</td>
<td>C</td>
<td>A</td>
</tr>
<tr>
<td>SND</td>
<td></td>
<td></td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>END</td>
<td></td>
<td></td>
<td></td>
<td>N</td>
</tr>
<tr>
<td>PPN</td>
<td>C</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CAN</td>
<td>C</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

N = Normal receiver SCB response
C = Normal receiver SCB response which requires change in sender SCB
A = Abnormal receiver SCB response

<table>
<thead>
<tr>
<th>RECEIVER SCB SENT</th>
<th>IDL</th>
<th>RTS</th>
<th>SND</th>
<th>END</th>
<th>PPN</th>
<th>CAN</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDL</td>
<td>N</td>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PTR</td>
<td></td>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RCV</td>
<td></td>
<td></td>
<td>N</td>
<td>C</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>SUS</td>
<td></td>
<td></td>
<td></td>
<td>N</td>
<td>C</td>
<td>A</td>
</tr>
<tr>
<td>SVG</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>SVD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>C</td>
</tr>
<tr>
<td>PPN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>C</td>
</tr>
<tr>
<td>CAN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>C</td>
</tr>
</tbody>
</table>

N = Normal sender SCB response
C = Normal sender SCB response which requires change in receiver SCB
A = Abnormal sender SCB response

5.9
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).
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

5.11
SCP JOB STREAM FLOW

- Examines the dataset header message and determines the dataset is a job dataset.

- Builds disk buffers from station buffers and when buffers are full has them written to disk.

- 'Cracks' the job card assigning defaults when appropriate.

- Makes appropriate entries in the job queue (system dataset table (SDT)), such as:

  - Dataset name
  - Job name and priority
  - Source and destination ID
  - Disposition code set to job dataset (IN)

- Has the dataset made permanent for the life of the job. (DSC)
EXEC RO05

INITIALIZE CHT ENTRIES FOR FRONT-END.

INITIALIZE LIT ENTRY FOR NEW O/I PAIR

LSTART

OUTPUT

NO

YES

Set up output LCP. Increment message, segment counts.

LRETRY

Set output active in LIT entry

Set interrupt address (WLCP)

Set CL from the LIT entry

Set CA from the LIT entry

Set input active in LIT entry

Set interrupt address (RLCP)

Set CL from the LIT entry

Set CA from the LIT entry

Set CL from the LIT entry

Set CA from the LIT entry

Set input active in LIT entry

Obtain link ID from LXT table

RLCP55

Set up for input of first subsegment

Set interrupt address (RSSEG)

RLCP65

Set CL from the LIT entry

Set CA from the LIT entry

Set CA from the LIT entry

Set CL from the LIT entry

Clear any output in progress

Station is now the master

Station is now the master

Logon LCP

ENO

NO

YES

ENO
DISK QUEUE MANAGER (DQM)
Device Allocation

by logical device name

1/0, 1/1, 1/2, 1/3

File system table

Data Progress

- Per Available Space
- Data Traversed
- DE Available Space

Modes:

- Static
- Dynamic

Size in tracks

Only size is track
DISK QUEUE MANAGER (DQM)

- MANAGES ALLOCATION/DEALLOCATION OF MASS STORAGE (DISKS)
- MANAGES MASS STORAGE REQUEST QUEUES
- MANAGES MASS STORAGE CHANNELS, CONTROLLERS AND DISK UNITS.
- MAXIMUM OF 11 DISK CONTROLLERS (44 DSU's)
- UPDATES THE DEVICE RESERVATION TABLE (DRT) FOR EACH DSU.
- CALLED BY EXEC (UPON TRANSFER OF AN I/O SEGMENT) OR BY ANOTHER TASK (FOR DISK REQUEST).
A DAT exists for each dataset in the system. A DAT defines where a dataset logically resides on mass storage, i.e., on which logical device or devices, and what portions of each device.

The DAT page and entry header contains general dataset information.

The DAT partition header contains general information concerning a particular partition of the DAT. A partition represents a portion of a single logical device. Each allocation index (AI) in a partition is a bit number in the respective Device Reservation Table (DRT).

The DAT is composed of as many 16-word DAT pages as necessary to represent the mass storage occupied by the dataset. Additional DAT pages continue from the point at which the last DAT left off.
Dataset Allocation Table (DAT)
FUNCTIONS:

DQMTAB

DQM

ALLOCATE DISK

NO

YES

DIS100

GET A DAT PAGE

ERR

NO

DAT SPACE AVAIL

YES

INSERT DAT ADDRESS INTO DNT

ERR

NO

AVAIL. DEVICE

YES

DIS125

MOVE LOGICAL DEVICE NAME TO DAT ENTRY

MOVE ALLOCATION SIZE AND STYLE TO DAT

DETERMINE NUMBER OF AI's NEEDED

DIS155

RESERVE DISK SPACE IN THE DRT

INSERT AI INTO THE DAT

MORE AI's

NO

EXIT

NO

YES

1

DEALLOCATE DISK

NO

10

DIS200

GET DAT ADDRESS

DIS203

GET DRT ADDRESS FOR THIS DAT

CLEAR AI's IN THE DRT

INCREMENT AVAILABLE AI's COUNT IN THE DRT

MORE PAGES

YES

NO

MORE PARTITIONS

YES

NO

DIS260

RELEASE THE DAT PAGES FOR THIS DATASET

EXIT

6.8
GET THE REQUEST TABLE ENTRY

STORE TASK ID, DNT ADDRESS, TASK REPLY WORD IN THE RQT ENTRY

SET THE DNT ENTRY ACTIVE

STORE REQUEST AND TRANSFER DIRECTION IN THE RQT ENTRY

TRANSFORM REQUEST FROM LOGICAL TO PHYSICAL

INCREMENT REQUEST COUNT IN THE EQUIPMENT TABLE

SET QUEUE BIT ACTIVE IN THE DEVICE CHANNEL TABLE

A JOB REQUEST

INSERT REQUEST IN SECOND ENTRY ON QUEUE

INSERT REQUEST AT END OF QUEUE

UPDATE QUEUE POINTERS IN THE EQU

UPDATE QUEUE POINTERS IN THE EQT

UPDATE QUEUE HEAD IN THE EQT

UPDATE QUEUE LENGTH IN THE EQT

CLEAR QUEUE BIT IN THE DEVICE CHANNEL TABLE

EXIT

EXIT

EXIT

DEQUEUE I/O

SET TO ERROR STATUS IN RETURN MESSAGE

EXIT

6.9
PERMANENT DATASET MANAGER (PDM)
PERMANENT DATASET MANAGER (PDM)

- PROVIDES FOR CREATING, ACCESSING, DELETING, MAINTAINING, AND AUDITING PERMANENT DATASETS.

- PERMANENT DATASETS MAY EXIST FOR USERS OR THE SYSTEM.

- USER PERMANENT DATASETS ARE CREATED UPON REQUEST BY A USER JOB. (SAVE)

- SYSTEM PERMANENT DATASETS ARE CREATED FOR SPOOLED INPUT AND OUTPUT DATASETS.

- A SYSTEM PERMANENT DATASET IS CREATED FOR JOB DATASETS ARRIVING FROM A FRONT-END. (SDT)

- A PERMANENT JOB DATASET IS DELETED UPON COMPLETION OF THE JOB.
FUNCTIONS: FCTBL
JOB SCHEDULER (JSH)
JOB SCHEDULER (JSH)

- INITIATES JOB SCHEDULING
- MANAGES USER AREA MEMORY RESOURCES. (JTA-LA)
- MANAGES USER JOB SWAPPING.
- MANAGES ROLL-IN AND ROLL-OUT OF USER JOBS.
- INITIATED THROUGH CALLS BY OTHER TASKS.
**JOB SCHEDULING**

- JSH SELECTS JOBS FROM THE SYSTEM DATASET TABLE (DC=IN).
- JOBS ARE ENTERED ON THE JOB EXECUTION TABLE (JXT) BY CLASS, PRIORITY AND TIME OF SUBMISSION.
- A PRIORITY OF 0-15 MAY BE SPECIFIED ON THE JOB CARD.
- THERE CAN BE UP TO 63 ENTRIES ON THE JXT.
- JOBS ON THE JXT CONTEND FOR MEMORY BASED ON PRIORITY AND JOB SIZE. (EXCLUDING PRIORITY 0 JOBS).
- ONCE A WAITING JOB BECOMES THE HIGHEST MEMORY PRIORITY IT IS MOVED FROM THE DISK TO MEMORY.
USER JOB SWAPPING

● Once in memory a job must contend with other jobs in memory for CPU time.

● CPU priority falls when a job is connected to the CPU.

● CPU priorities are recomputed after each scheduling interval.

● CPU priority is given to jobs that are streaming (doing I/O).

● The algorithm for computing CPU priority is adjustable.
ROLL-OUT AND ROLL-IN OF JOBS

- MEMORY PRIORITY RISES WHILE A JOB IS WAITING FOR MEMORY

- MEMORY PRIORITY FALLS WHEN A JOB IS IN MEMORY AND EXECUTING

- HIGHER PRIORITY JOBS MAY PRE-EMPT MEMORY FROM LOWER PRIORITY JOBS, THEN:

  - THE LOWER PRIORITY JOB IS WRITTEN TO DISK (ROLLED-OUT).
  - HIGHER PRIORITY JOB USES THE VACATED MEMORY.
  - MEMORY MAY HAVE TO BE 'COMPACTED' SO HIGH PRIORITY JOB HAS A CONTIGUOUS AREA.

- ROLLED-OUT JOBS PRIORITY IS RISING SO EVENTUALLY IT WILL BE WRITTEN BACK TO MEMORY (ROLLED-IN).

- THE ALGORITHM FOR COMPUTING MEMORY PRIORITY IS ADJUSTABLE.

- ROLLED JOBS MAY BE CONTINUED OR RERUN AFTER A RESTART
Note: In all the examples, jobs are swapped when the difference between their priorities equals or exceeds $db$.

Showing two jobs which are too large to share memory. Their initial priorities (from the job cards) are 3 and 5. Note that the higher-priority job runs first and consistently enjoys a longer stay in memory, because its priority has asymptotes at $P=2$ and 8 (as opposed to 0 and 6 for the other job.)
Showing three jobs, only two of which can share memory at any one time. Except for the presence of the priority-7 job, this graph is identical to figure 3.8-2. The priority-7 job can never be forced out of memory because its priority never will be even as much as one unit below $P=5$, which is the maximum attainable by either of the other two jobs. (If its initial priority were 6 instead of 7, it still would never be forced out by the other two jobs.)
SUBROUTINE ROLLJOB

* PURPOSE:
* TO MAKE A REQUEST OF THE DISK QUEUE MANAGER, EITHER TO COPY
* A JOB OUT INTO ITS ROLLOUT DATASET OR TO READ THE JOB'S IMAGE
* BACK INTO MEMORY.

* ENTRY:
* A4 = JXT-ENTRY ADDRESS.
* A5 = JTA ADDRESS.
* DNT (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 | WJHINT.A4 |
| A6 | WJXCSJ.S4 |
| S1 | A5 |
| PUT,S1 | $S&S7.INBUF,A7 |
| S2 | A6 |
| PUT,S2 | $S&S7.INRKB,A7 |
| S1 | A4 |
| S1 | $S&D40 |
| S2 | A7 |
| S1 | $S1S2 |
| A1 | JSHID.0 |
| A2 | DOMID.0 |
| S2 | TRANSFER |
| J | PUTREQ |

LET PUTREQ RETURN TO THE CALLER.
USER AREA MANAGEMENT

- PROVIDES FOR ROLL-IN, ROLL-OUT RESOURCES
- PROVIDES FOR INITIAL JOB ENTRY TO MEMORY
- PROVIDES DYNAMIC ALLOCATION OF MEMORY TO USER JOBS
- PROVIDES DYNAMIC DEALLOCATION OF MEMORY TO USER JOBS
- USES A TABLE (MST) FOR DETERMINING USEABLE MEMORY SPACE
- USES A 'FIRST-FIT' METHOD FOR GAINING MORE MEMORY

'FIRST-FIT' METHOD:
ALLOCATES MEMORY (BY BLOCK) BEGINNING AT LOW END OF MEMORY

MOVES JOBS (IF REQUIRED) BEGINNING AT HIGH END OF MEMORY

ENCOURAGES LARGE BLOCKS OF FREE MEMORY AT THE HIGH END OF MEMORY
SUBTITLE 'JOB SCHEDULER (LIB0INIT)'

SUBROUTINE LIB0INIT

PURPOSE:

TO FIND THE MST ENTRY FOR AN ALLOCATED SEGMENT WHOSE ADDRESS
IS KNOWN, OR TO FIND THE MST ENTRY FOR THE FIRST ALLOCATED
SEGMENT ABOVE A GIVEN ADDRESS.

ENTRY:

A3 = ADDRESS OF AN ALLOCATED SEGMENT WHICH IS TO BE FREED,
OR ANY VALUE LESS THAN THAT ADDRESS AS LONG AS IT IS
GREATER THAN THE ADDRESS OF ANY LOWER ALLOCATED SEGMENT.

EXIT:

S0 = 0 IF NO ALLOCATED SEGMENT EXISTS WITH AN ADDRESS
THAT IS EQUAL TO OR GREATER THAN (A3).

S2 = (IF S0 IS NONZERO) THE OFFSET OF THE DESIRED MST ENTRY.

S0 = (IF S0 IS NONZERO) THE MST ENTRY ITSELF.

REGISTERS:

(A0-A2), (S0-S2) ARE DESTROYED.

LIB0INIT =

A2 = -1 LET A2=0 TO FETCH THE FIRST MST ENTRY.

LIB1 =

A2 = x

A2 = A2+1 ADVANCE THE MST OFFSET.

S2 = B'MST.A2 FETCH AN MST ENTRY.

S0 = S2

JSZ LIB2 RETURN S0=0 IF NO MSADDR IS >= A3.

S0 = S2+64-N'MSTYPE

A1 = S2

A0 = A1-A3 COMPARE MSADDR TO THE CALLER'S ADDRESS.

JSZ LIB1 RELLOOP IF THIS SEGMENT ISN'T ALLOCATED.

JAN LIB1 LOOP UNTIL THE DESIRED SEGMENT IS FOUND.

LIB2 =

J = S0 RETURN.
MEMORY SEGMENT TABLE - MST
The MST in STP memory contains a one-word entry for each segment of memory
that has been allocated by the Job Scheduler plus additional entries that
describe free segments. MST entries are stored in ascending order according
to the beginning address of the segment (MSADDR). Any free space between
two allocated segments ;s consolidated into a single entry. The last entry
in the table is always followed by a zero word. To provide for the case
where every allocated segment is surrounded by a free segment, the MST must
have twice as many words in it as the maximum number of allocated segments,
plus two more.

o

40

16

TYPE

SIZE

63

ADDR

]

Memory Segment Table (MST) entry
Field

Word

Bits

Description

MSTYPE

a

0-15

Contains a if the segment is free;
otherwise, it contains the JXT ordinal
of the job to which the segment is
allocated.

MSSIZE

a

16-39

Number of words in the segment.
is always a multiple of lOOOa.

MSADDR

o

40-63

STP-relative address of the first word
in the segment. This is always a
multiple of 1000 a -

This

MEMORY SEGMENT ALLOCATION TABLE
0040753 0000010014500000116000
004~754 0000020014500000263000 0000000121000000430000 0000000000000000000000 0000000000000000000000
LOCATIONS 00040760 THROUGH 0(~41053 CONTAIN 0000000'~OO00e000000000

0041054 000000.0000000000000000

8.15


SUBTITLE 'JOB SCHEDULER (MOVEMENT)'

**SUBROUTINE MOVEMENT**

**PURPOSE:**
TO MOVE A BLOCK OF WORDS IN MEMORY.

**ENTRY:**
A3 = ADDRESS OF THE DESTINATION AREA.
A5 = ADDRESS OF THE FIRST WORD TO BE MOVED.
A6 = NUMBER OF WORDS IN THE DESTINATION AREA.
A7 = NUMBER OF WORDS TO BE MOVED.

**EXIT:**
THE NUMBER OF WORDS GIVEN BY MIN (A6,A7) IS MOVED FROM (A5) TO (A3). A2 AND S2 ARE SET UP FOR A SUBSEQUENT 'ERASEMEM' CALL (WHICH WILL HAVE NO EFFECT IF S2 <= 0).

**REGISTERS:**
(V0), (B10-B11), (A0-A2), (S0-S3) ARE DESTROYED.
(T10,T11) ARE DESTROYED WHILTE THE POST CALL IS PRESENT.

**METHOD:**
IF A3=A5, NO MOVING IS DONE, BUT A2 AND S2 ARE SET FOR ERASEMEM ANYWAY. IF A3<A5, MOVING COMMENCES AT THE LOWER ADDRESSES, IF A3>A5, IT BEGINS AT THE HIGHER ONES.
S4 ELEMENTS ARE MOVED AT A TIME, UNLESS THE STORE WOULD OVERFLOW THE DESTINATION AREA --- IN WHICH CASE A FINAL PASS IS MADE WITH A REDUCED VL.

MOVEMENT =

B10 = A3
B11 = A5
A0 = A5-A7
A1 = A3-A7
S3 = A7
JAP = MVM1
S3 = A5

MVM1 =

S2 = A1
A2 = A3+A7
A0 = A3-A5
S1 = KS
A1 = Z30
S0 = 4516-53
JAZ = MW89
JSM = MW89

T10 = S2
T11 = S7
S1 = A5
S1 = SLD'24
A2 = A3
S2 = S3'D'24
S2 = A7
S1 = S1'57
S2 = S2'S7
S3 = 57'D'43
S4 = 57'S4

POST = 32,S1,S2
S2 = T10
S7 = T11

TOTAL MOVE LENGTH (IN S3) = MIN (A6,A7).
SET UP S2 AND A2 FOR A SUBSEQUENT CALL TO ERASEMEM (WILL BE A NO-OP IF S2<=0).
COMPARE THE TWO GIVEN ADDRESSES.
INITIAL VECTOR LENGTH = 64.
DO NOTHING IF A3<=A5.
DO NOTHING IF S3<=0.
BUILD ARGUMENTS FOR A POST CALL.
FROM-ADDRESS
TO-ADDRESS
FROM-LENGTH
TO-LENGTH
IDENTIFIER
POST THE MOVE ARGUMENTS AND RESTORE REGISTERS.
JS2  MVN4  DO ONLY THE FINAL PASS IF S3<64.
JSN  MVN6  MOVE FROM THE TOP DOWN IF A3>A5.

MVN2 = *  SET VL TO 64 (OR. ON FINAL PASS. TO RESIDUE).
        VL   A1
        S1   A1

MVN3 = *  SET ADDRESS FOR VECTOR LOAD.
 A0   A5  S3 = WORDS STILL TO BE MOVED AT LOOP's END.
 S3   S3-S1  A5=A1
 A6   A5+A1  ADVANCE VECTOR-LOAD POINTER.
 V0   A0-A1  LOAD
 A0   A3  SET ADDRESS FOR VECTOR STORE.
 S0   S3-S1  S0 WILL BE NEGATIVE WHEN S3<64 (OR. ON FINAL
 A3   A3-A1  PASS, WHEN S3=0).
        ,A0.1  V0  STORE
        JSP  MVN8  RELOOP WHILE S3 >= 64.

MVN4 = *  HERE IF MOVING FROM THE TOP DOWN INSTEAD OF FROM THE BOTTOM UP.
        S0   S3  0 <= S3 < 64; WE ARE DONE IF S3=0.
        A1   S3
        JSM  MVN2  IF S3>0, GO BACK TO MAKE A FINAL PASS.
        J   MVN9  ELSE RETURN.

MVN6 = *  START A5 AND A3 AT THEIR HIGHEST VALUES.
        A1   S3
        A6   A5+A1
        A3   A3+A1
        A1   Z50  RESTORE A1=64.

MVN7 = *  SET VL TO 64 (OR. ON FINAL PASS. TO RESIDUE).
        VL   A1
        S1   A1

MVN8 = *  SET ADDRESS FOR VECTOR LOAD.
 A0   A5-A1  S3 = WORDS STILL TO BE MOVED AT LOOP's END.
 S3   S3-S1  A5=A1
 A6   A5-A1  BACK UP VECTOR-LOAD POINTER.
 V0   A0-A1  LOAD
 A0   A3-A1  SET ADDRESS FOR VECTOR STORE.
 S0   S3-S1  S0 WILL BE NEGATIVE WHEN S3<64 (OR. ON FINAL
 A3   A3-A1  PASS, WHEN S3=0).
        ,A0.1  V0  STORE
        JSP  MVN8  RELOOP WHILE S3 >= 64.

MVN9 = A3   B10  S002.1355
        A5   B11  S002.1996
        J   B0   S002.1997
        RETURN.
JOB SCHEDULER JOB FLOW

- JOB DATASET ENTERS THE SYSTEM

- STATION CALL PROCESSOR (SCP) HAS THE JOB SAVED ON DISK

- SCP HAS AN ENTRY MADE IN THE SYSTEM DATASET TABLE (SDT), AND DATASET CATALOG (DSC)

- JOB IS PLACED IN JOB EXECUTION TABLE (JXT) ACCORDING TO CLASS/PRIORITY AND TIME OF SUBMISSION

- ONCE ON THE JXT THE JOB CONTENTS FOR MEMORY

- JOB MEMORY IS ERASED WHEN JOB INITIATED (I@ERASE)

- JOB IN MEMORY CONTENTS FOR CPU TIME

- JOB IN MEMORY MAY BE ROLLED OUT

- WHEN A JOB COMPLETES, ITS OUTPUT DATASETS ARE PLACED IN THE OUTPUT QUEUE IN SDT AND MADE PERMANENT

- WHEN FRONT-END ACKNOWLEDGES RECEIPT OF A DATASET THE DATASET CATALOG ENTRY IS REMOVED
Calling Sequence

JSH can be invoked from any other task by calling either TSKREQ or PUTREQ with the following instruction sequence:

<table>
<thead>
<tr>
<th>Location</th>
<th>Result</th>
<th>Operand</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>calling task's ID</td>
<td></td>
</tr>
<tr>
<td>A2</td>
<td>JSHID,0</td>
<td></td>
</tr>
<tr>
<td>S1</td>
<td>function code (already shifted)</td>
<td></td>
</tr>
<tr>
<td>S2</td>
<td>job's offset in JXT</td>
<td></td>
</tr>
<tr>
<td>S1!S2</td>
<td>address if any</td>
<td></td>
</tr>
<tr>
<td>S2</td>
<td>S2&lt;0'16</td>
<td></td>
</tr>
<tr>
<td>S1!S2</td>
<td>auxiliary information if any</td>
<td></td>
</tr>
<tr>
<td>S2</td>
<td>S2&lt;0'40</td>
<td></td>
</tr>
<tr>
<td>S1!S2</td>
<td>TSKREQ or PUTREQ</td>
<td></td>
</tr>
</tbody>
</table>

Input register format:

<table>
<thead>
<tr>
<th>INPUT+0</th>
<th>INPUT+1</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUX</td>
<td>ADDR</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>AUX</th>
<th>unused</th>
</tr>
</thead>
</table>

AUX Auxiliary information; unused by JSH. (Any value the caller places in INPUT+0 is returned verbatim in OUTPUT+1.)

ADDR Word address relative to the beginning of STP of an additional word or list of words if needed to fully specify the call.

FC Function code; use equated labels of the form JSXXXX, selected from table

JXO JXT offset for the job in question.

Output register format:

<table>
<thead>
<tr>
<th>OUTPUT+0</th>
<th>OUTPUT+1</th>
</tr>
</thead>
<tbody>
<tr>
<td>STATUS</td>
<td>AUX</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>STATUS</th>
<th>Status of requested function.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Requested function completely accomplished.</td>
</tr>
<tr>
<td>≠0</td>
<td>Error or system is unable to fulfill request completely.</td>
</tr>
<tr>
<td>Function Code</td>
<td>Input Parameters</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>J$ABORT</td>
<td>JXO, CODE</td>
</tr>
<tr>
<td>J$DELETE</td>
<td>JXO</td>
</tr>
<tr>
<td>J$RERUN</td>
<td>JXO</td>
</tr>
<tr>
<td>J$ALLOC</td>
<td>JXO, ADDR</td>
</tr>
<tr>
<td>J$IOSUSP</td>
<td>JXO</td>
</tr>
<tr>
<td>J$IODONE</td>
<td>JXO</td>
</tr>
<tr>
<td>J$DELAY</td>
<td>JXO, ADDR</td>
</tr>
<tr>
<td>J$DELAYK</td>
<td>JXO, ADDR</td>
</tr>
<tr>
<td>J$WAIT</td>
<td>JXO, ADDR</td>
</tr>
<tr>
<td>J$SUSP</td>
<td>JXO</td>
</tr>
<tr>
<td>J$SUSPK</td>
<td>JXO</td>
</tr>
<tr>
<td>J$REMK</td>
<td>JXO</td>
</tr>
<tr>
<td>J$RESUME</td>
<td>JXO</td>
</tr>
<tr>
<td>J$STOP</td>
<td>JXO</td>
</tr>
<tr>
<td>J$STPALL</td>
<td>JXO</td>
</tr>
<tr>
<td>J$START</td>
<td>JXO</td>
</tr>
<tr>
<td>J$STRALL</td>
<td>JXO</td>
</tr>
<tr>
<td>J$CLEAR</td>
<td>JXO</td>
</tr>
<tr>
<td>J$INDEX</td>
<td>JXO</td>
</tr>
<tr>
<td>J$SHUTDOWN</td>
<td>JXO</td>
</tr>
<tr>
<td>J$RCVR</td>
<td>JXO</td>
</tr>
</tbody>
</table>

PARTIAL JOB SCHEDULER FUNCTIONS

FUNCTIONS:
J$REQTAB

---

8.23
JOB STATUS AND STATE CHANGES

The status field

The 23-bit status field (JXSTAT) in each job's JXT entry is described in table 3.8-2. The bits labeled Q, R, X, I, U, L, S, O, and M are mutually exclusive. Generally, whichever one of them is set determines the job's state. The other bits modify the job's state.

If all of bits 3 through 22 are zero, the job is said to be waiting to be connected to the CPU (state W).

Table 3.8-2. Status bit assignments

<table>
<thead>
<tr>
<th>Bit position in JXSTAT</th>
<th>Bit name</th>
<th>Corresponding job state</th>
<th>Interpretation (when bit is set)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>K</td>
<td>S</td>
<td>Keep this job in memory; don't roll it out.</td>
</tr>
<tr>
<td>1</td>
<td>A</td>
<td>any</td>
<td>Abort pending; reason given in JXEPC.</td>
</tr>
<tr>
<td>2</td>
<td>H</td>
<td>O</td>
<td>Holding operator or shutdown suspension until RN is set.</td>
</tr>
<tr>
<td>3</td>
<td>O</td>
<td>S</td>
<td>Suspended (indefinitely) by operator.</td>
</tr>
<tr>
<td>4</td>
<td>S</td>
<td>S</td>
<td>Suspended (momentarily) by system.</td>
</tr>
<tr>
<td>5</td>
<td>T</td>
<td>S</td>
<td>Suspended until a given time elapses.</td>
</tr>
<tr>
<td>6</td>
<td>E</td>
<td>S</td>
<td>Suspended until a given event occurs.</td>
</tr>
<tr>
<td>7</td>
<td>M</td>
<td>M</td>
<td>Memory allocation is pending.</td>
</tr>
<tr>
<td>8</td>
<td>Q</td>
<td>Q</td>
<td>Queued up; waiting to be initiated.</td>
</tr>
<tr>
<td>9</td>
<td>R</td>
<td>R</td>
<td>Rolled out. The M bit may also be set.</td>
</tr>
<tr>
<td>10</td>
<td>X</td>
<td>X</td>
<td>Executing.</td>
</tr>
<tr>
<td>11</td>
<td>I</td>
<td>I</td>
<td>Dormant pending recall on I/O completion.</td>
</tr>
<tr>
<td>12</td>
<td>C</td>
<td>any</td>
<td>Rerun request in progress.</td>
</tr>
<tr>
<td>13</td>
<td>D</td>
<td>any</td>
<td>Delete request in progress.</td>
</tr>
<tr>
<td>14</td>
<td>U</td>
<td>U</td>
<td>Unloading from memory to roll file.</td>
</tr>
<tr>
<td>15</td>
<td>L</td>
<td>L</td>
<td>Loading into a new memory area.</td>
</tr>
<tr>
<td>16</td>
<td>P</td>
<td>U or L</td>
<td>Unload or load initiation is pending.</td>
</tr>
<tr>
<td>17</td>
<td>Y</td>
<td>M, Q or R</td>
<td>Waiting for memory liberation.</td>
</tr>
<tr>
<td>18</td>
<td>Z</td>
<td>M, Q or R</td>
<td>Waiting for memory compaction.</td>
</tr>
<tr>
<td>19</td>
<td>B</td>
<td>S</td>
<td>Suspended (indefinitely) by recovery.</td>
</tr>
<tr>
<td>20</td>
<td>V</td>
<td>I</td>
<td>Waiting on INDEX write completion.</td>
</tr>
<tr>
<td>21</td>
<td>F</td>
<td>I</td>
<td>Waiting on rollfile write completion.</td>
</tr>
<tr>
<td>22</td>
<td>N</td>
<td>M, Q or R</td>
<td>Not in memory.</td>
</tr>
</tbody>
</table>
CPU SWAPPING

Q → X  A JOB IS QUEUED IN THE JOB EXECUTION TABLE (JXT) WAITING MEMORY. THE JOB IS BROUGHT INTO MEMORY AND BEGINS EXECUTION

X → I  THE JOB IS SUSPENDED PENDING COMPLETION OF I/O. THE CPU IS AVAILABLE FOR USE BY ANOTHER JOB.

I → W  THE I/O COMPLETES. THE JOB MAY WAIT FOR MORE CPU TIME.

W → X  JSJ RECONNECTS THE WAITING JOB BECAUSE OF ITS HIGH PRIORITY

X → W  THE JOB'S TIME SLICE EXPIRES. THE CPU IS AVAILABLE FOR USE BY ANOTHER JOB.
MEMORY SWAPPING

W → U  I/O IS INITIATED FOR ROLL-OUT OF THE WAITING JOB.

U → R  THE JOB IS ROLLED-OUT. THE JOB’S MEMORY IS AVAILABLE.

R → L  I/O IS INITIATED FOR ROLL-IN OF THE JOB. MEMORY HAS BEEN ALLOCATED.

L → W  THE JOB IS ROLLED-IN. THE JOB MAY WAIT FOR CPU TIME.

---

Roll-in is done

W ← L

Roll-out forced by another job

Roll-out is done

U ← R

Preempts memory from another job

8.29
JOB SUSPENSION

X → M  THE EXECUTING JOB MAKES A REQUEST FOR MORE MEMORY. IF REQUEST CANNOT BE SATISFIED THE JOB MAY BE ROLLED-OUT.

M → W  THE ALLOCATION REQUEST WAS SATISFIED. THE JOB WAS NOT ROLLED OUT SO NOW MAY WAIT FOR CPU TIME.

M → U  I/O IS INITIATED FOR ROLL-OUT OF THE JOB REQUESTING MORE MEMORY.

X → O  X → S  THE EXECUTING JOB REQUESTS SUSPENSION. JOB IS DISCONNECTED FROM CPU AND LIABLE TO BE ROLLED OUT.

O → OU  S → SU  I/O IS INITIATED FOR ROLL-OUT OF THE SUSPENDED JOB.

OU → OR  SU → SR  THE SUSPENDED JOB IS ROLLED-OUT. THE JOB'S MEMORY IS AVAILABLE.

OR → R  SR → R  THE SUSPENDED JOB IS REACTIVATED.

O → W  S → W  THE SUSPENDED JOB IN MEMORY IS REACTIVATED.
Operator SUSPEND command

Allocate request

Operator RESUME command

Request satisfied

Request cannot be satisfied soon enough

Memory is in demand

Roll-out is done

Operator RESUME command

8.31
OPERATOR SUSPEND

ALLOCATE REQUEST

MEMORY IS IN DEMAND

MUST WAIT FOR I/O

DISCONNECTION (TIME SLICE EXPIRED)

I/O IS COMPLETE

RECONNECTION (NEW TIME SLICE)

REQUEST SATISFIED

REQUEST CANNOT BE SATISFIED SOON ENOUGH

ROLL-IN IS DONE

ROLL-OUT FORCED BY ANOTHER JOB

ROLL-OUT IS DONE

PREEMPT MEMORY FROM ANOTHER JOB

ROLL-OUT IS DONE
USER EXCHANGE PROCESSOR (EXP)
USER EXCHANGE PROCESSOR (EXP)

- Processes all user normal action requests.
- Processes all user error exits.
- Processes JSH requests to initiate or abort a user job.
- All requests made through the exchange processor request word (JTEP) in a user JTA. JTEP is word 6780.

<table>
<thead>
<tr>
<th>Field</th>
<th>Bits</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JTEPX</td>
<td>0-1</td>
<td>User exit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2 Normal exit</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 Error exit or execution error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 Not user exit</td>
</tr>
<tr>
<td>JTEPC</td>
<td>2</td>
<td>Continuation flag</td>
</tr>
<tr>
<td>JTEPJ</td>
<td>3</td>
<td>Job Scheduler request flag</td>
</tr>
<tr>
<td>JTEPM</td>
<td>4</td>
<td>JTA expansion request flag</td>
</tr>
<tr>
<td>JTEPF</td>
<td>7-15</td>
<td>Exchange package flags</td>
</tr>
<tr>
<td>JTEPP</td>
<td>16-39</td>
<td>P register for errors</td>
</tr>
<tr>
<td>JTEPA</td>
<td>40-63</td>
<td>EXP Continuation address</td>
</tr>
</tbody>
</table>
USER NORMAL EXIT

- USER ISSUES A SYSTEM ACTION REQUEST WHICH SETS 8 REGISTERS AND ISSUES AN EX INSTRUCTION

  SO-DESIRED CALL TABLE OFFSET (FUNCTION CODE)
  S1-CONVENTION -- AN ADDRESS OF A USER TABLE (OPTIONAL)
  S2-FURTHER OPTIONAL ARGUMENT

- WHEN THE REQUEST IS COMPLETE THE USER'S 80-REGISTER IS SET SO-ZERO IF REQUEST COMPLETED NORMALLY 
  SO-ERROR CODE OR JOB ABORTS

- EXP EXAMINES THE USERS JXT ENTRIES. A JXT ENTRY POINTS TO THE USERS JTA

- THE USERS JTA CONTAINS A COPY OF THE EXCHANGE PACKAGE

- EXP EXAMINES JTEP (WORD 673) AND THE USERS 80 REGISTER TO DETERMINE REQUEST
NORMAL EXCHANGE CALLS AND PROCESSING ADDRESSES

<table>
<thead>
<tr>
<th>CALL</th>
<th>NORMAL EXCHANGE CALLS AND PROCESSING ADDRESSES</th>
</tr>
</thead>
<tbody>
<tr>
<td>CON 0×24+ADV</td>
<td>00 ADVANCE PROGRAM</td>
</tr>
<tr>
<td>CON 0×24+ABT</td>
<td>01 ABO RT JOB</td>
</tr>
<tr>
<td>CON 1×24+DAT</td>
<td>02 GET DATE - MM/DD/YY</td>
</tr>
<tr>
<td>CON 1×24+TIM</td>
<td>03 GET TIME - HH:MM:SS</td>
</tr>
<tr>
<td>CON D×3×24+MSG</td>
<td>04 PUT MESSAGE INTO USERS DAYFILE</td>
</tr>
<tr>
<td>CON 2×24+RCL</td>
<td>05 SUSPEND IF I/O ACTIVE</td>
</tr>
<tr>
<td>CON 0×24+TRM</td>
<td>06 TERMINATE JOB</td>
</tr>
<tr>
<td>CON 1×24+SSW</td>
<td>07 SET SENSE SWITCHES</td>
</tr>
<tr>
<td>CON 2×24+OFN</td>
<td>10 OPEN DATASET</td>
</tr>
<tr>
<td>CON 1×24+MEM</td>
<td>11 MEMORY REQUEST</td>
</tr>
<tr>
<td>CON 1×24+LEN</td>
<td>12 GET LAST BLOCK NUMBER</td>
</tr>
<tr>
<td>CON 2×24+CLS</td>
<td>13 CLOSE DATASET</td>
</tr>
<tr>
<td>CON D×3×24+DNT</td>
<td>14 CREATE/SENSE DNT</td>
</tr>
<tr>
<td>CON 1×24+HDE</td>
<td>15 SET MODE</td>
</tr>
<tr>
<td>CON D×3×24+GNS</td>
<td>16 GET NEXT CONTROL CARD</td>
</tr>
<tr>
<td>CON 0×24+HEX</td>
<td>17 EXECUTE AN OBJECT CODE FILE</td>
</tr>
<tr>
<td>CON D×3×24+RLS</td>
<td>20 RELEASE DATASET</td>
</tr>
<tr>
<td>CON 0×24+PIM</td>
<td>21 PERMANENT DATASET MANAGER REQUEST</td>
</tr>
<tr>
<td>CON D×3×24+RDC</td>
<td>22 READ DISK CIRCULAR</td>
</tr>
<tr>
<td>CON D×3×24+UOC</td>
<td>23 WRITE DISK CIRCULAR</td>
</tr>
<tr>
<td>CON 2×24+GRC</td>
<td>24 GET SYSTEM REVISION LEVELS</td>
</tr>
<tr>
<td>CON D×3×24+DPS</td>
<td>25 DISPOSE DATASET</td>
</tr>
<tr>
<td>CON 1×24+JDA</td>
<td>26 GET JULIAN DATE - YYDDD</td>
</tr>
<tr>
<td>CON 1×24+JTI</td>
<td>27 ACCUMULATE JOB CPU TIME</td>
</tr>
<tr>
<td>CON D×3×24+ACT</td>
<td>30 GET ACCOUNTING INFORMATION FROM JTA</td>
</tr>
<tr>
<td>CON 0×24+SFS</td>
<td>31 SET P AND SUSPEND USER</td>
</tr>
<tr>
<td>CON 1×24+CSW</td>
<td>32 CLEAR SENSE SWITCH</td>
</tr>
<tr>
<td>CON 1×24+TSW</td>
<td>33 TEST SENSE SWITCH</td>
</tr>
<tr>
<td>VWD D×40-LEBDSP.D×24+B10</td>
<td>034 BUFFERED I/O REQUEST</td>
</tr>
<tr>
<td>CON 1×24+DLY</td>
<td>35 DELAY JOB X NUMBER OF MILLISECONDS</td>
</tr>
<tr>
<td>CON 0×24+AGR</td>
<td>36. ACQUIRE DATASET FROM FRONT END</td>
</tr>
<tr>
<td>CON 0×24+RNN</td>
<td>37 DISABLE NO-RERUN CHECK</td>
</tr>
<tr>
<td>CON 0×24+RRN</td>
<td>40 OVERRIDE NOT-RERUNNABLE FLAG</td>
</tr>
</tbody>
</table>

CALMAX = W×CALL×1 CALL LIMIT
CALMAX1 = CALMAX+1

EXCHANGE PROCESSOR CALL TABLE

| CALMAX | 000000000000000000031302 | 00000000000000000100321101 |
| CALMAX1 | 000000000000000000031302 | 00000000000000000100321101 |

9.5
USER ERROR EXIT

- A job executes an error exit (ERR) instruction or encounters a hardware detected error
- Exp issues appropriate error messages
- Exp may initiate job abort or reprieve processing
- Exp may search the remaining job control statements for an exit or end-of-file
- If exit is found exp examines the next statement
- If the next statement is 'DUMPJOB' a dataset named $DUMP is written to disk in binary
- $DUMP is a binary unblocked dataset and contains the JTA and entire User field (BA-LA) of the aborted job
- $DUMP may then be used by the 'DUMP' control statement to output various areas of the aborted program
- If reprieve processing, exp transfers control back to the user job at the address specified in the reprieve macro
MESSAGE PROCESSOR (MSG)
MESSAGE PROCESSOR (MSG)

- Writes messages in the system and user log files.
- Can be called by other tasks and users (through EXP).
- MSG uses TIO when performing I/O.
- The system log can be analyzed by the 'extract' program.
- Users can enter messages in the system and/or user log.
THE SYSTEM LOG IS A PERMANENT DATASET NAMED $SYSTEMLOG.

THE SYSTEM LOG BUFFER AND DSP ARE ALLOCATED IN HIGH MEMORY.

A TASK CALLS MSG THROUGH THE TSKREQ ROUTINE WITH THE FOLLOWING REGISTERS SET:

(A1) ID of task requesting message entry
(A2) MSGID,0
(S1) INPUT+0 } request
(S2) INPUT+1 }

Request format:

Field | Word | Bits | Description
--- | --- | --- | ---
FC | INPUT+0 | 0-1 | Function code:
 | | | 1 Write message in user's logfile only
 | | | 2 Write message in SYSTEMLOG only
 | | | 3 Write message in both user's logfile and in SYSTEMLOG
TYPE | INPUT+0 | 19-27 | Major class of log record
SUB | INPUT+0 | 28-33 | Subtype of log record
LENGTH | INPUT+0 | 34-39 | Length in words of message. If length is 0, the message is a character string terminated by a zero byte in the low-order eight bits.
ADDR | INPUT+0 | 40-63 | Starting address relative to STP of message to be written
JXT | INPUT+1 | 46-63 | JXT address if message associated with job; otherwise 0

Reply format:

Field | Word | Bits | Description
--- | --- | --- | ---
DONE | OUTPUT+1 | 0-63 | Characters 'MSG DONE' in ASCII
USER LOG PROCESSING

- The user log ($LOG) is created by JSH for each user job.
- The user log buffer is in the job's JTA.
- The user calls MSG via the 'MESSAGE' macro:

MESSAGE - ENTER MESSAGE IN LOGFILE

The printable ASCII message at the location specified in the macro call is entered in the job and system logfile. The message must be 1-80 characters terminated by a zero byte. A flag, loc, indicates the destination for the message.

Format:

<table>
<thead>
<tr>
<th>Location</th>
<th>Result</th>
<th>Operand</th>
</tr>
</thead>
<tbody>
<tr>
<td>MESSAGE</td>
<td></td>
<td>address, loc</td>
</tr>
</tbody>
</table>

- address: A symbol or an A, S, or T register that contains the address.
- loc: Destination for message. May be any of the following:
  - U: User logfile only
  - S: System logfile only
  - US: User and system logfiles; default, if loc is blank.
MEMORY ERROR PROCESSOR (MEP)

DISK ERROR CORRECTION (DEC)
**MEMORY ERROR PROCESSOR (MEP)**

- **PURPOSE IS TO RELAY MEMORY ERROR INFORMATION FROM EXEC TO MSG.**
- **MESSAGES FROM EXEC TO MSG ARE IN THE FORMAT:**

```
<table>
<thead>
<tr>
<th>Field</th>
<th>Word</th>
<th>Bits</th>
<th>Significance</th>
</tr>
</thead>
<tbody>
<tr>
<td>JN</td>
<td>0</td>
<td>0-63</td>
<td>Jobname or 'STP'</td>
</tr>
<tr>
<td>E</td>
<td>1</td>
<td>0,1</td>
<td>Error type (binary):</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>10 Uncorrectable memory error</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>01 Correctable memory error</td>
</tr>
<tr>
<td>R</td>
<td>1</td>
<td>8-15</td>
<td>Read mode:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>0 Scalar</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1 I/O</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2 Vector</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>3 Fetch</td>
</tr>
<tr>
<td>SYN</td>
<td>1</td>
<td>32-39</td>
<td>Syndrome bits</td>
</tr>
<tr>
<td>ERR ADDR</td>
<td>1</td>
<td>40-63</td>
<td>Error address</td>
</tr>
<tr>
<td>BA</td>
<td>2</td>
<td>26-39</td>
<td>Base address</td>
</tr>
<tr>
<td>P</td>
<td>2</td>
<td>40-63</td>
<td>Program address</td>
</tr>
<tr>
<td>RTC</td>
<td>3</td>
<td>0-63</td>
<td>Real-time clock</td>
</tr>
</tbody>
</table>
```

11.1
DISK ERROR CORRECTION (DEC)

- Attempts correction of a data error from disk.
- Uses cyclic redundancy checkwords (CRC) for error correction.
- DQM schedules DEC if normal recovery procedures are unsuccessful.
- DQM passes the EQT address which contains the valid checkwords.
- DEC processes one disk error at a time.
- Refer to mass storage subsystem reference manual 2240630 for the CRC algorithm.
SYSTEM PERFORMANCE MONITOR
SYSTEM PERFORMANCE MONITOR

- MEASURES OVERALL SYSTEM PERFORMANCE
- INVOKED ON A TIME DELAY BASIS
- DELAY INTERVAL BETWEEN COLLECTIONS IS (LETASPMDFLY)
- USES MESSAGE TASK TO WRITE SYSTEM TIMES TO THE SYSTEMLOG
- USES POOL 1 AS A BUFFER FOR THE SYSTEM TIMES
- MEASUREMENTS COLLECTED BY SPM INCLUDE
  
  CPU UTILIZATION
  NUMBER OF TASK READIES
  NUMBER OF EXEC REQUESTS BY TASK
  NUMBER OF EXEC REQUESTS BY TYPE
  MEMORY UTILIZATION
  CHANNEL INTERRUPT COUNT
  DISK CHANNEL UTILIZATION
  DISK UNIT UTILIZATION
  LINK UTILIZATION
  USER CALL USAGE
  JOB SCHEDULER STATISTICS
  JOB CLASS STATISTICS
### EXECUTIVE REQUESTS

<table>
<thead>
<tr>
<th>TASK</th>
<th>REQUESTS</th>
<th>TIME INTERVAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>313</td>
<td>60012.18 MS</td>
</tr>
<tr>
<td>2</td>
<td>1290</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>256</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>715</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>576</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>71</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>2226</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

### TASK UTILIZATION

<table>
<thead>
<tr>
<th>TASK</th>
<th>READIES</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>129</td>
</tr>
<tr>
<td>2</td>
<td>482</td>
</tr>
<tr>
<td>3</td>
<td>83</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>2597</td>
</tr>
<tr>
<td>6</td>
<td>156</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>410</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
</tr>
</tbody>
</table>

### CPU UTILIZATION

<table>
<thead>
<tr>
<th>TASK</th>
<th>EXECUTION TIME</th>
<th>EXECUTION BLOCKED TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.00 MSEC</td>
<td>0.00 MSEC</td>
</tr>
<tr>
<td>1</td>
<td>151.96 MSEC</td>
<td>0.00 MSEC</td>
</tr>
<tr>
<td>2</td>
<td>0.00 MSEC</td>
<td>0.00 MSEC</td>
</tr>
<tr>
<td>3</td>
<td>25.56 MSEC</td>
<td>0.00 MSEC</td>
</tr>
<tr>
<td>4</td>
<td>0.00 MSEC</td>
<td>0.00 MSEC</td>
</tr>
<tr>
<td>5</td>
<td>4.29 MSEC</td>
<td>0.01 MSEC</td>
</tr>
<tr>
<td>6</td>
<td>14.42 MSEC</td>
<td>0.05 MSEC</td>
</tr>
<tr>
<td>7</td>
<td>10.76 MSEC</td>
<td>0.03 MSEC</td>
</tr>
<tr>
<td>8</td>
<td>0.00 MSEC</td>
<td>0.00 MSEC</td>
</tr>
<tr>
<td>9</td>
<td>5.41 MSEC</td>
<td>0.00 MSEC</td>
</tr>
<tr>
<td>10</td>
<td>42.78 MSEC</td>
<td>0.03 MSEC</td>
</tr>
<tr>
<td></td>
<td>0.00 MSEC</td>
<td>0.00 MSEC</td>
</tr>
<tr>
<td></td>
<td>0.00 MSEC</td>
<td>0.00 MSEC</td>
</tr>
</tbody>
</table>

- - - - - END OF EXTRACT REPORT

Listing of all subtypes of SPM records

---

12.3
### Listing of all subtypes of SPM records (continued)

#### Channel Interrupt Counts

<table>
<thead>
<tr>
<th>Channel</th>
<th>Calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>118</td>
</tr>
<tr>
<td>3</td>
<td>116</td>
</tr>
<tr>
<td>4</td>
<td>811</td>
</tr>
<tr>
<td>5</td>
<td>684</td>
</tr>
<tr>
<td>6</td>
<td>1219</td>
</tr>
<tr>
<td>7</td>
<td>1889</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
</tr>
<tr>
<td>13</td>
<td>0</td>
</tr>
<tr>
<td>14</td>
<td>0</td>
</tr>
<tr>
<td>15</td>
<td>0</td>
</tr>
<tr>
<td>16</td>
<td>0</td>
</tr>
</tbody>
</table>

#### DEC Call Usage

<table>
<thead>
<tr>
<th>Type</th>
<th>Calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>985</td>
</tr>
<tr>
<td>3</td>
<td>1172</td>
</tr>
<tr>
<td>4</td>
<td>104</td>
</tr>
<tr>
<td>5</td>
<td>379</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>24</td>
</tr>
<tr>
<td>8</td>
<td>126</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>24</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
</tr>
<tr>
<td>13</td>
<td>197</td>
</tr>
<tr>
<td>14</td>
<td>367</td>
</tr>
<tr>
<td>15</td>
<td>367</td>
</tr>
<tr>
<td>16</td>
<td>1977</td>
</tr>
</tbody>
</table>

#### Disk Channel Utilization

<table>
<thead>
<tr>
<th>Channel</th>
<th>SEC</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.00</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0.00</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>2358.79</td>
<td>4.27%</td>
</tr>
<tr>
<td>3</td>
<td>1426.73</td>
<td>2.35%</td>
</tr>
<tr>
<td>4</td>
<td>0.00</td>
<td>0.00%</td>
</tr>
<tr>
<td>5</td>
<td>0.00</td>
<td>0.00%</td>
</tr>
<tr>
<td>6</td>
<td>0.00</td>
<td>0.00%</td>
</tr>
<tr>
<td>7</td>
<td>0.00</td>
<td>0.00%</td>
</tr>
<tr>
<td>8</td>
<td>0.00</td>
<td>0.00%</td>
</tr>
<tr>
<td>9</td>
<td>0.00</td>
<td>0.00%</td>
</tr>
<tr>
<td>10</td>
<td>0.00</td>
<td>0.00%</td>
</tr>
<tr>
<td>11</td>
<td>0.00</td>
<td>0.00%</td>
</tr>
</tbody>
</table>

**Time Interval:** 60012.18 Milliseconds
Listing of all subtypes of SPM records (continued)
<table>
<thead>
<tr>
<th>USER CALL USAGE</th>
<th>TIME INTERVAL = 3000000 12 MILLISECONDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUMBER OF TYPE 0 CALLS = 1042</td>
<td>NUMBER OF TYPE 26 CALLS = 0</td>
</tr>
<tr>
<td>NUMBER OF TYPE 1 CALLS = 1</td>
<td>NUMBER OF TYPE 27 CALLS = 0</td>
</tr>
<tr>
<td>NUMBER OF TYPE 2 CALLS = 2</td>
<td>NUMBER OF TYPE 28 CALLS = 0</td>
</tr>
<tr>
<td>NUMBER OF TYPE 3 CALLS = 83</td>
<td>NUMBER OF TYPE 29 CALLS = 13832</td>
</tr>
<tr>
<td>NUMBER OF TYPE 4 CALLS = 691</td>
<td>NUMBER OF TYPE 30 CALLS = 0</td>
</tr>
<tr>
<td>NUMBER OF TYPE 5 CALLS = 2729</td>
<td>NUMBER OF TYPE 31 CALLS = 0</td>
</tr>
<tr>
<td>NUMBER OF TYPE 6 CALLS = 7</td>
<td>NUMBER OF TYPE 32 CALLS = 0</td>
</tr>
<tr>
<td>NUMBER OF TYPE 7 CALLS = 0</td>
<td>NUMBER OF TYPE 33 CALLS = 77876</td>
</tr>
<tr>
<td>NUMBER OF TYPE 8 CALLS = 26</td>
<td>NUMBER OF TYPE 34 CALLS = 21</td>
</tr>
<tr>
<td>NUMBER OF TYPE 9 CALLS = 179</td>
<td>NUMBER OF TYPE 35 CALLS = 0</td>
</tr>
<tr>
<td>NUMBER OF TYPE 10 CALLS = 5399</td>
<td>NUMBER OF TYPE 36 CALLS = 0</td>
</tr>
<tr>
<td>NUMBER OF TYPE 11 CALLS = 480</td>
<td>NUMBER OF TYPE 37 CALLS = 1</td>
</tr>
<tr>
<td>NUMBER OF TYPE 12 CALLS = 0</td>
<td>NUMBER OF TYPE 38 CALLS = 284</td>
</tr>
<tr>
<td>NUMBER OF TYPE 13 CALLS = 0</td>
<td>NUMBER OF TYPE 39 CALLS = 8</td>
</tr>
<tr>
<td>NUMBER OF TYPE 14 CALLS = 20999</td>
<td>NUMBER OF TYPE 40 CALLS = 0</td>
</tr>
<tr>
<td>NUMBER OF TYPE 15 CALLS = 0</td>
<td>NUMBER OF TYPE 41 CALLS = 0</td>
</tr>
<tr>
<td>NUMBER OF TYPE 16 CALLS = 333</td>
<td>NUMBER OF TYPE 42 CALLS = 98</td>
</tr>
<tr>
<td>NUMBER OF TYPE 17 CALLS = 46</td>
<td>NUMBER OF TYPE 43 CALLS = 0</td>
</tr>
<tr>
<td>NUMBER OF TYPE 18 CALLS = 2239</td>
<td>NUMBER OF TYPE 44 CALLS = 0</td>
</tr>
<tr>
<td>NUMBER OF TYPE 19 CALLS = 21955</td>
<td>NUMBER OF TYPE 45 CALLS = 312</td>
</tr>
<tr>
<td>NUMBER OF TYPE 20 CALLS = 825</td>
<td>NUMBER OF Type 46 CALLS = 233</td>
</tr>
<tr>
<td>NUMBER OF TYPE 21 CALLS = 3</td>
<td>NUMBER OF TYPE 47 CALLS = 0</td>
</tr>
<tr>
<td>NUMBER OF TYPE 22 CALLS = 2</td>
<td>NUMBER OF TYPE 48 CALLS = 0</td>
</tr>
<tr>
<td>NUMBER OF TYPE 23 CALLS = 110773</td>
<td>NUMBER OF TYPE 49 CALLS = 0</td>
</tr>
<tr>
<td>NUMBER OF TYPE 24 CALLS = 11</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>JOB CLASS STATISTICS</th>
<th>TIME INTERVAL = 3000000 14 MILLISECONDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLASS NAME = OPERATE</td>
<td>NUMBER OF ACTIVE JATS = 0</td>
</tr>
<tr>
<td></td>
<td>NUMBER OF JOBS WAITING FOR JATS = 0</td>
</tr>
<tr>
<td></td>
<td>NUMBER OF RESERVED JATS = 1</td>
</tr>
<tr>
<td></td>
<td>MAXIMUM NUMBER OF JATS = 64</td>
</tr>
<tr>
<td></td>
<td>STATUS = JB</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>JOB SCHEDULER STATISTICS</th>
<th>TIME INTERVAL = 3000000 12 MILLISECONDS</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUMBER OF MEMORY COMPACTS = 97E8</td>
<td></td>
</tr>
<tr>
<td>NUMBER OF ALLOCATIONS = 1288</td>
<td></td>
</tr>
<tr>
<td>NUMBER OF ALLOCATIONS = 14</td>
<td></td>
</tr>
<tr>
<td>NUMBER OF REDUCES = 2</td>
<td></td>
</tr>
<tr>
<td>NUMBER OF INITIATES = 3</td>
<td></td>
</tr>
<tr>
<td>NUMBER OF TERMINATES = 6</td>
<td></td>
</tr>
<tr>
<td>NUMBER OF SCHEDULING INTERVALS = 80</td>
<td></td>
</tr>
<tr>
<td>NUMBER OF INDEX WRITES = 212</td>
<td></td>
</tr>
<tr>
<td>CLASS STRUCTURE NAME = DAY</td>
<td></td>
</tr>
<tr>
<td>NUMBER OF JOBS IN SYSTEM = 38</td>
<td></td>
</tr>
<tr>
<td>NUMBER OF ACTIVE JATS = 22</td>
<td></td>
</tr>
<tr>
<td>MAXIMUM JATS LIMIT = 30</td>
<td></td>
</tr>
<tr>
<td>NUMBER OF AVAILABLE POOL JATS = 6</td>
<td></td>
</tr>
<tr>
<td>NUMBER OF DEFINED CLASSES = 14</td>
<td></td>
</tr>
<tr>
<td>NUMBER OF CLASSES WAITING FOR JATS = 95595</td>
<td></td>
</tr>
</tbody>
</table>

Listing of all subtypes of SPM records (continued)
OVERLAY MANAGER
OVERLAY MANAGER (OVM)

- CONTROLS THE LOADING OF TASK OVERLAYS.

- EACH TASK USING OVM MUST RESERVE A MEMORY AREA SUFFICIENT TO HOLD ONE OR MORE DISK BLOCKS (512 WORDS) AS AN OVERLAY AREA.

- OVERLAYS MAY BE NESTED UP TO TEN DEEP, THAT IS OVERLAY X MAY CALL OVERLAY Y, ETC.

- AN OVERLAY MAY CALL ANOTHER OVERLAY WITH OR WITHOUT RETURN REQUESTED TO ITSELF.

- ONCE IN MEMORY AN OVERLAY MAY OR MAY NOT BE RELOADED FROM DISK DEPENDENT ON CHANGES IN INTERNAL CODING OR TABLES WITHIN THE OVERLAY.

- EACH CALLED OVERLAY OVERWRITES THE TASKS OVERLAY AREA WITH THE REQUESTED OVERLAY.
JOB CLASS MANAGER
JOB CLASS MANAGER (JCM)

- Validates job class qualifications and assigns a job to a class.

- Links job input queue entries in the SDT by class rank, job priority, and time submitted.

- Optionally replaces each job statement priority with the class rank priority.

- Assigns the number of available job execution table (JXT) entries for each specified job class.

- Once a job has a job execution table (JXT) entry, its class assignment does not change unless the job is rerun.

- A user may lower a job class through a parameter in the job control statement.

- Enabled/disabled through operator commands.
TAPE QUEUE MANAGER
TASK TO BE PROVIDED
CONTROL STATEMENT PROCESSOR (CSP)
**CONTROL STATEMENT PROCESSOR (CSP)**

- Binary copy of CSP resides in memory immediately above STP or optionally on disk(s).

- When a user job is submitted, JSH calls EXP to copy CSP into the user field (at BA+200g).

- CSP executes in the user field and is subject to user job states.

- CSP shares the user tables such as the JTA, JCB, DSP's, etc.

- CSP 'cracks' user control statements.

- CSP enters messages into the logs through the message macro.
**Example**

The control statement

```
ASSIGN(DN=MYFILE,BS=25)
```

is stored in the JCB as follows:

```
<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>A</td>
<td>S</td>
<td>S</td>
<td>I</td>
<td>G</td>
<td>N</td>
<td>(</td>
</tr>
<tr>
<td></td>
<td>N</td>
<td>=</td>
<td>M</td>
<td>Y</td>
<td>F</td>
<td>I</td>
<td>L</td>
</tr>
<tr>
<td></td>
<td>,</td>
<td>B</td>
<td>S</td>
<td>=</td>
<td>2</td>
<td>5</td>
<td>)</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
```

```
<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>16</td>
<td>A</td>
<td>S</td>
<td>S</td>
<td>I</td>
<td>G</td>
<td>N</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>D</td>
<td>N</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>M</td>
<td>Y</td>
<td>F</td>
<td>I</td>
<td>L</td>
<td>E</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>S</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
```

17.3
JOB, JN=EXAMPLE, P=3.
ACCESS, DN=TEMP, PDN=MASTERFILE, R=SECRET.
ASSIGN, DN=TEMP, BS=12.
COPYD, I=TEMP, O=COPY.
DISPOSE, DN=COPY, SDN=BACKUP, DC=ST, MF=A.
EXIT.
EOF.

1. JOB IS STAGED FROM FRONT-END TO CRAY-1.

2. STATION CALL PROCESSOR (SCP) RECEIVES JOB AND PLACES IT IN THE SYSTEM DATASET TABLE (SDT) WHICH IS THE INPUT QUEUE FOR JOB SCHEDULER. SCP ALSO SAVES JOB IN THE DATASET CATALOG (DSC).

3. JOB SCHEDULER PICKS THE JOB FOR EXECUTION FROM SDT AND THEN MAKES AN ENTRY FOR THE JOB IN THE JOB EXECUTION TABLE AND ALLOCATES MEMORY.

4. JOB SCHEDULER CONNECTS THE JOB TO CPU.

5. CONTROL STATEMENT PROCESSOR (CSP) CRACKS THE ACCESS CARD, CALLS THE EXCHANGE PROCESSOR (EXP) TO VERIFY PERMISSIONS TO ACCESS MASTERFILE.

6. CSP CRACKS THE ASSIGN CARD, CALLS EXP TO OPEN TEMP, AND CHANGE THE BUFFER SIZE TO 12 INSTEAD OF 4 BLOCKS.
7. CSP CRACKS THE COPYD CARD, CALLS EXP TO LOAD PROGRAM COPYD FROM THE SYSTEM DIRECTORY.

8. COPYD EXECUTES, I/O BOUND, STREAMING DATA FROM TEMP TO COPY.

9. JOB SCHEDULER RECOGNIZES THAT JOB'EXAMPLE' IS STREAMING AND CONNECTS EXAMPLE TO THE CPU WHENEVER A BLOCK IS AVAILABLE IN THE INPUT CIRCULAR BUFFER.

10. CSP CRACKS THE DISPOSE STATEMENT, CALLS EXP.

11. EXP WRITES AN ENTRY FOR BACKUP IN THE SDT OUTPUT QUEUE.

12. SCP STAGES BACKUP TO MAINFRAME A.

13. DEPENDING ON MAINFRAME A, SCP DELETES ENTRIES FOR BACKUP IN THE SDT AND DSC.

14. CSP CRACKS EXIT STATEMENT, CALLS EXP TO TERMINATE THE JOB.

15. EXP SEARCHES FOR $OUT, COPIES $LOG TO $OUT (EXTENDING $OUT), CHANGES $OUT's DATASET NAME TO JOBNAME.

16. JOB TERMINATION CHANGES ENTRIES IN THE DSC AND SDT OUTPUT QUEUE FOR DATASET EXAMPLE.

17. SCP STAGES EXAMPLE TO MAINFRAME A, ETC.

18. JOB TERMINATION RELEASES ALL LOCAL DATASETS, DELETES $CS, AND TERMINATES JOB.
SYSTEM INTERACTION
JOB ENTRY

1. THE STATION CALL PROCESSOR (SCP) TASK EXAMINES A DATASET HEADER MESSAGE AND DETERMINES A JOB DATASET IS ENTERING THE SYSTEM.

SCP MOVES THE SEGMENT BUFFER TO THE DISK BUFFER EACH TIME THE SEGMENT BUFFER BECOMES FULL FROM INPUT OF THE SUBSEGMENTS.

SCP REQUESTS THE JOB DATASET DISK BUFFER BE WRITTEN TO DISK BY THE DISK QUEUE MANAGER (DQM) TASK.

2. THE DISK QUEUE MANAGER (DQM) TASK ALLOCATES DISK SPACE FOR THE JOB DATASET ON THE INITIAL CALL, AND HAS THE DISK BUFFER WRITTEN TO DISK VIA THE DISK DRIVER.

3. UPON REQUEST FROM THE STATION CALL PROCESSOR (SCP) TASK, THE JOB CLASS MANAGER (JCM) TASK DETERMINES WHICH JOB CLASS QUEUE ON THE SYSTEM DATASET TABLE (SDT) THE JOB SHOULD BE ENTERED IN.

4. UPON REQUEST FROM THE STATION CALL PROCESSOR (SCP) TASK, THE PERMANENT DATASET MANAGER (PDM) TASK CREATES A PERMANENT DATASET STRUCTURE FOR THE JOB AND SETS THE DISPOSITION CODE (DC) EQUAL TO ASCII CHARACTERS 'IN'.

5. THE JOB SCHEDULER (JSH) TASK SELECTS THE HIGHEST PRIORITY JOB FROM THE INPUT QUEUE WHOSE JOB CLASS HAS A JOB EXECUTION TABLE (JXT) ENTRY AVAILABLE.

JSH CONSTRUCTS THE JOBS JXT ENTRY AND CHANGES THE SYSTEM DATASET TABLE (SDT) INPUT QUEUE ENTRY TO EXECUTE QUEUE AND SETS THE JOB STATE TO THE QUEUED (Q) STATE IN THE JXT.
INITIAL JOB PREPARATION

1. THE JOB SCHEDULER (JSH) TASK SELECTS A JOB FROM THE JOB EXECUTING TABLE (JXT) THAT HAS BEEN GIVEN ITS INITIAL MEMORY ALLOCATION.

2. THE JOB SCHEDULER (JSH) ZEROES OUT THE JOB TABLE AREA (JTA) AND JOB COMMUNICATION BLOCK (JCB) OF THE JOB.

3. THE JOB SCHEDULER (JSH) INITIALIZES THE JOB COMMUNICATION BLOCK (JCB) POINTERS SUCH AS HLM, FL, DSP, ETC. AND SAVES THE POINTER TO THE JCB IN THE JOB TABLE AREA (JTA).

4. THE JOB SCHEDULER CREATES DATASET NAME TABLE (DNT) ENTRIES FOR $CS, $LOG, $IN, AND $OUT IN THE USERS JOB TABLE AREA (JTA).

5. THE JOB SCHEDULER INITIALIZES THE JOBS ROLLFILE DATASET NAME TABLE (DNT) ENTRY AT THE END OF THE JOB EXECUTION TABLE (JXT) ENTRY.

6. THE JOB SCHEDULER INITIALIZES THE ROLL JOB INDEX (RJI) TABLE ENTRY FOR THE JOB FOR POSSIBLE RECOVERY.

7. THE JOB SCHEDULER (JSH) SETS THE JOBS STATE TO WAITING (W) IN THE JXT THUS ALLOWING THE JOB TO CONTEND FOR THE CPU ALONG WITH OTHER ACTIVE JOBS IN THE JXT.
JOB ROLLOUT

1. THE JOB SCHEDULER (JSH) TASK DETERMINES A NEED BY A JOB FOR USER MEMORY AND CREATES A REQUEST TO HAVE ANOTHER JOB IN MEMORY ROLLED-OUT.

JSH SEARCHES THE JOB EXECUTION TABLE (JXT) FOR A CANDIDATE FOR JOB ROLL-OUT.

JSH AFTER LOCATING A CANDIDATE MAY HAVE TO COMPACT MEMORY TO OBTAIN A LARGE ENOUGH FREE SEGMENT OF MEMORY.

JSH SETS THE PROPER JOB STATES OF THE WAITING JOB AND THE JOB ABOUT TO BE ROLLED-OUT.

2. THE DISK QUEUE MANAGER (DQM) TASK ALLOCATES DISK SPACE IF NEEDED, AND HAS THE MEMORY RESIDENT JOB AREA WRITTEN TO DISK VIA THE DISK DRIVER.

3. THE JOB SCHEDULER (JSH) TASK UPDATES THE MEMORY SEGMENT TABLE (MST).

4. THE JOB SCHEDULER (JSH) TASK LIFTS THE SUSPENSION ON THE JOB WAITING FOR MEMORY LIBERATION.
ACCESS A PERMANENT DATASET

1. A JOB MAKES A REQUEST FOR PERMISSION TO USE A PERMANENT DATASET.

2. THE EXCHANGE PACKAGE PROCESSOR (EXP) TASK EXAMINES THE REQUEST FOR VALID PARAMETERS AND ISSUES A REQUEST TO THE PERMANENT DATASET MANAGER (PDM) TASK FOR ACCESS AND THE JOB SCHEDULER (JSH) TASK FOR JOB SUSPENSION.

3. THE PERMANENT DATASET MANAGER (PDM) TASK VALIDATES ACCESS TO THE SELECTED DATASET FOR THE USER JOB.

   PDM MANAGES THE NECESSARY SYSTEM TABLES SO THAT THE USER HAS THE DESIRED PERMISSIONS GRANTED.

   PDM CREATES A PERMANENT DATASET TABLE (PDS) ENTRY FOR THE ACTIVE DATASET.

   PDM CREATES A DATASET NAME TABLE (DNT) ENTRY FOR THE USER AND INSURES THE DATASET ALLOCATION TABLE (DAT) FOR THE DATASET IS MADE RESIDENT IN THE REQUESTING JOB AREA THEM AKING THE DATASET LOCAL FOR THAT JOB.

4. THE JOB SCHEDULER (JSH) TASK SETS THE JOBS STATUS BIT TO SUSPK (K) THUS DISALLOWING THE JOB FROM ROLL-OUT AND MEMORY MOVEMENT UNTIL THE JOBS DATASET ALLOCATION TABLE (DAT) ENTRY IS READ INTO THE USER AREA FROM THE DISK DATASET CATALOG (DSC).

   JSH RESETS THE JOBS STATUS TO THE WAIT (W) STATE FOLLOWING COMPLETION OF GENERATION OF THE LOCAL DATASET BY THE PERMANENT DATASET MANAGER (PDM) TASK.
DATASET ACQUISITION

1. A JOB MAKES A REQUEST TO ACQUIRE A PERMANENT DATASET.

2. THE EXCHANGE PACKAGE PROCESSOR (EXP) TASK EXAMINES THE REQUEST FOR VALID PARAMETERS AND ISSUES A REQUEST TO THE PERMANENT DATASET MANAGER (PDM) TASK TO SEARCH FOR AN EXISTANT PERMANENT DATASET.

   IF THE DATASET ALREADY EXISTS IN THE SYSTEM, THE FLOW Follows ACCESS PROCESSING.

   IF THE DATASET DOES NOT EXIST WITHIN THE SYSTEM EXP ISSUES A REQUEST TO THE JOB SCHEDULER (JSH) TASK TO SUSPEND THE JOB AND EXP ENQUEUES THE DATASET NAME AND PLACES IT ON THE REQUEST QUEUE IN THE SYSTEM DATASET TABLE (SDT).

3. THE PERMANENT DATASET MANAGER (PDM) TASK SEARCHES THE DATASET CATALOG (DSC) FOR AN EXISTANT DATASET AND IF SO TO VERIFY THE PERMISSION WORDS.

   PDM MONITORS NECESSARY TABLES AND USER AREAS ENABLING THE DATASET TO BECOME LOCAL TO THE JOB THAT ISSUED THE ACQUIRE AS IN THE ACCESS FLOW.

4. IF THE JOB SCHEDULER (JSH) TASK WAS REQUESTED IT SUSPENDS THE JOB AWAITING THE DATASET TRANSFER FROM THE FRONT-END.

5. IF THE DATASET IS NOT CRAY RESIDENT THE STATION CALL PROCESSOR (SCP) TASK SELECTS THE REQUEST ENTRY FROM THE SYSTEM QUEUE AND COMMUNICATES WITH THE FRONT-END VIA THE FRONT-END DRIVER TO RETRIEVE THE NON-RESIDENT DATASET.

   SCP THEN COMMUNICATES WITH THE JOB SCHEDULER (JSH), THE PERMANENT DATASET MANAGER (PDM) AND THE DISK QUEUE MANAGER (DQM) TO INSURE PROPER SAVING OF THE DATASET AND JOB ADVANCEMENT.
LOCAL I/O

1. A JOB ISSUES A WRITE REQUEST THROUGH A SYSTEM MACRO CAUSING PHYSICAL I/O.

2. THE EXCHANGE PACKAGE PROCESSOR (EXP TASK EXAMINES THE REQUEST FOR VALID PARAMETERS AND FORWARDS THE REQUEST FOR I/O.

3. THE TASK I/O (TIO) ROUTINES TRANSFER DATA, INSERTING RCW's WHERE NEEDED, FROM THE JOB DATA AREA TO THE JOB DISK BUFFERS ON A WRITE.

   TIO DETERMINES THE DISK BUFFERS ARE GREATER THAN HALF FULL AND PERFORMS CIRCULAR I/O (CIO).

   CIRCULAR I/O (CIO) ISSUES A WRITE REQUEST OF THE DISK QUEUE MANAGER (DQM) TASK AND HAS THE JOB SUSPENDED, STATE I, BY THE JOB SCHEDULER (JSH).

4. THE DISK QUEUE MANAGER (DQM) TASK ALLOCATES DISK ON A WRITE IF NEEDED, AND PERFORMS THE I/O VIA THE DISK DRIVER.
JOB MEMORY REQUEST

1. A JOB MAKES A REQUEST TO ALLOCATE/DEALLOCATE MEMORY THROUGH AN RFL CONTROL STATEMENT.

2. THE EXCHANGE PACKAGE PROCESSOR (EXP) EXAMINES THE USERS JOB COMMUNICATION BLOCK (JCB) FOR VALID PARAMETERS SUCH AS HLM, FL, ETC.

   EXP REQUESTS THE JOB SCHEDULER (JSH) TASK FOR THE ACTUAL ALLOCATE/DEALLOCATE OF MEMORY.

3. THE JOB SCHEDULER (JSH) TASK UPON A DEALLOCATE REQUEST MUST UPDATE USER FIELDS SUCH AS LA, HLM, DSP POINTERS, ETC.

   JSH RETURNS THE RELEASED MEMORY TO THE SYSTEM BY UPDATING THE MEMORY SEGMENT TABLE (MST) AND FILLING THE RETURNED MEMORY WITH THE CONTENTS OF IaERASE SHIFTED LEFT 24 BITS.

4. THE JOB SCHEDULER (JSH) TASK UPON AN ALLOCATE REQUEST MUST SEARCH THE MEMORY SEGMENT TABLE (MST) FOR FREE MEMORY.

5. IF FREE MEMORY IS FOUND ADJACENT TO THE USER JOB THE MEMORY MAY BE GIVEN TO THE USER BY UPDATING VARIOUS USER FIELDS SUCH AS LA, HLM, ETC. AND UPDATING THE MEMORY SEGMENT TABLE (MST).

6. IF THE JOB SCHEDULER (JSH) TASK IN ITS SEARCH OF THE MEMORY SEGMENT TABLE (MST) CANNOT FIND MEMORY ADJACENT TO THE USER JOB, MEMORY COMPACTION OR ROLL-OUT MAY HAVE TO BE USED.

   IF SUCH IS THE CASE THE REQUESTING JOB IS SET TO THE WAITING (W) STATE AND THE REQUESTING JOB ITSELF BECOMES SUBJECT TO ROLL-OUT.
JOB TERMINATION

1. A JOB TERMINATES WITH AN EXIT CONTROL STATEMENT INTERPRETED BY
   THE CONTROL STATEMENT PROCESSOR (CSP).

2. CSP POSTS TERMINATION MESSAGES TO THE LOGFILES VIA THE MESSAGE
   (MSG) TASK.

3. THE EXCHANGE PACKAGE PROCESSOR (EXP) TASK COPIES THE JOBS
   $LOG TO $OUT.

   EXP CHANGES $LOG TO SCRATCH AND RELEASES SCRATCH DATASETS AS
   WELL AS $IN, $CS.

4. THE PERMANENT DATASET MANAGER (PDM) CREATES A DATASET CATALOG
   ENTRY (DSC) FOR EACH JOB OUTPUT DATASET THUS MAKING THE DATASET
   PERMANENT.

5. EXP HAS CREATED ENTRIES FOR THE JOBS OUTPUT DATASETS IN THE SDT.

6. THE DISK QUEUE MANAGER (DQM) TASK ALLOCATES DISK FOR $OUT IF
   NEEDED, AND HAS THE BUFFER WRITTEN TO DISK VIA THE DISK DRIVER.

7. THE JOB SCHEDULER (JSH) TASK TERMINATES THE JOB BY RELEASING
   THE EXECUTING QUEUE ENTRY IN THE JOB EXECUTION TABLE (JXT).

   JSH RELEASES THE JOBS MEMORY AND RELEASES THE JOBS DATASET
   AND UPDATES THE MEMORY SEGMENT TABLE (MST).

8. THE STATION CALL PROCESSOR (SCP) TASK MANAGES SYSTEM MEMORY
   BUFFERS AND TRANSMITS THE JOBS OUTPUT DATASETS TO THE FRONT-
   END VIA THE FRONT-END DRIVER.

9. UPON RECEIPT OF EACH OUTPUT DATASET BY THE FRONT-END, THE
   STATION CALL PROCESSOR (SCP) REQUESTS THE PERMANENT DATASET
   MANAGER (PDM) TO DELETE THE DATASET CATALOG (DSC) ENTRY FOR
   THE DATASET AND DQM RELEASES THE DATASETS DISK SPACE.
CORPORATE HEADQUARTERS
1440 Northland Drive
Mendota Heights, MN 55120
Tel: 612-452-6650
TLX 24344

THE CHIPPEWA FACILITIES
Manufacturing:
Highway 178 North
Chippewa Falls, WI 54729
Tel: 715-723-2221
TWX 0285 1699

Engineering:
Highway 178 North
Chippewa Falls, WI 54729
Tel: 715-723-5501

CRAY LABORATORIES
Cray Labs Headquarters
5311 Western Avenue
Boulder, CO 80301
Tel: 303-449-3351

Hall lab
P.O. Box 169
Chippewa Falls, WI 54729
Tel: 715-723-0296

SALES OFFICES
Eastern Regional Sales Office
10750 Columbia Pike, Suite 602
Silver Spring, MD 20901
Tel: 301-681-9626

Central Regional Sales Office
5330 Manhattan Circle, Suite F
Boulder, CO 80303
Tel: 303-499-3055

Houston Sales Office
3121 Buffalo Speedway, Suite 400
Houston, TX 77098
Tel: 713-677-8053

Austin Sales Office
3415 Keystone, Suite 201
Austin, TX 78731
Tel: 512-2245-7034

Western Regional Sales Office
Sunset Office Plaza
1874 Holmes Street
Livermore, CA 94550
Tel: 415-447-0201

Los Angeles Sales Office
101 Continental Boulevard, Suite 456
El Segundo, CA 90245
Tel: 310-640-2351

Seattle Sales Office
536A Medical Dental Building
2728 Dolby Avenue
Everett, WA 98201
Tel: 425-299-5075

INTERNATIONAL SUBSIDIARIES
Cray Research (UK) Limited
James Glaisher House
Grenville Place
Bracknell, UK
Tel: 44-344-21515
TLX: 848841

Cray Research GmbH
Wartburgplatz 7
8000 Munich 40
West Germany
Tel: 49-89-3630-76
TLX: 05213211

Cray Research Japan, Limited
Shin Aoyama Building, West 1661
1-1 Minami-Aoyama 1-chome
Minato-ku, Tokyo 107 Japan
Tel: 81(03)403-3471