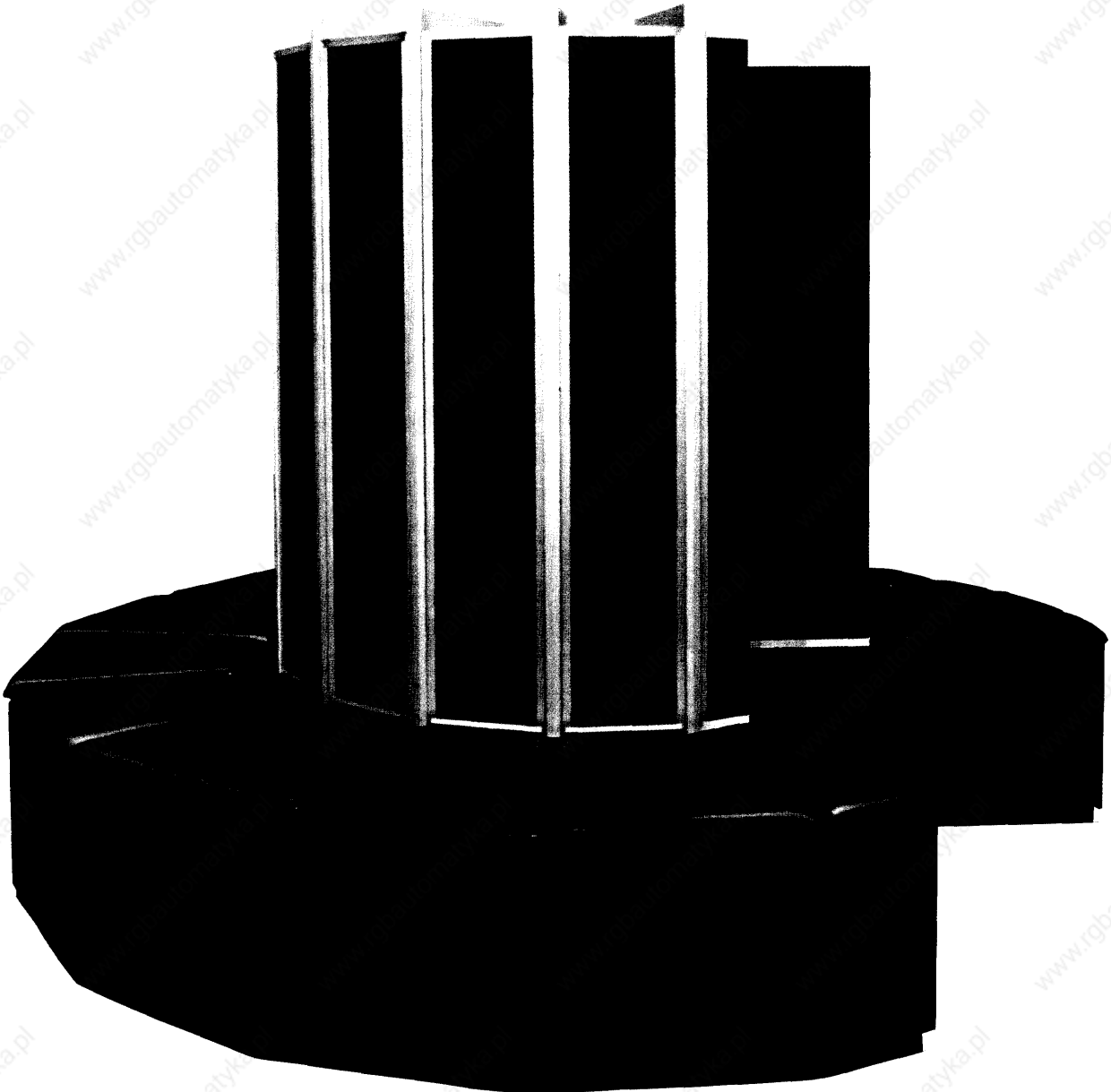


**CRAY**  
**RESEARCH, INC.**

**CRAY-1®**  
**COMPUTER SYSTEM**

OPERATING SYSTEM (OS)  
WORKBOOK

T-0103



FOR TRAINING PURPOSES ONLY



**CRAY-1<sup>®</sup>**

**COMPUTER SYSTEM**

**OPERATING SYSTEM (COS)  
WORKBOOK**

**T-0103**

Copyright© 1979 by CRAY RESEARCH, INC. This manual or parts thereof may not be reproduced in any form without permission of CRAY RESEARCH, INC.

**CRAY  
RESEARCH, INC.**

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

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

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

CRAY RESEARCH, INC.,

1440 Northland Drive,

Mendota Heights, Minnesota 55120

---

<u>Revision</u>	<u>Description</u>
	December, 1979 - Original printing.
A	June, 1980 - obsoletes other revision.
B	January, 1981 - obsoletes other revisions.
C	September, 1981 - obsoletes other revisions. Supports 1.10 release.

## 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



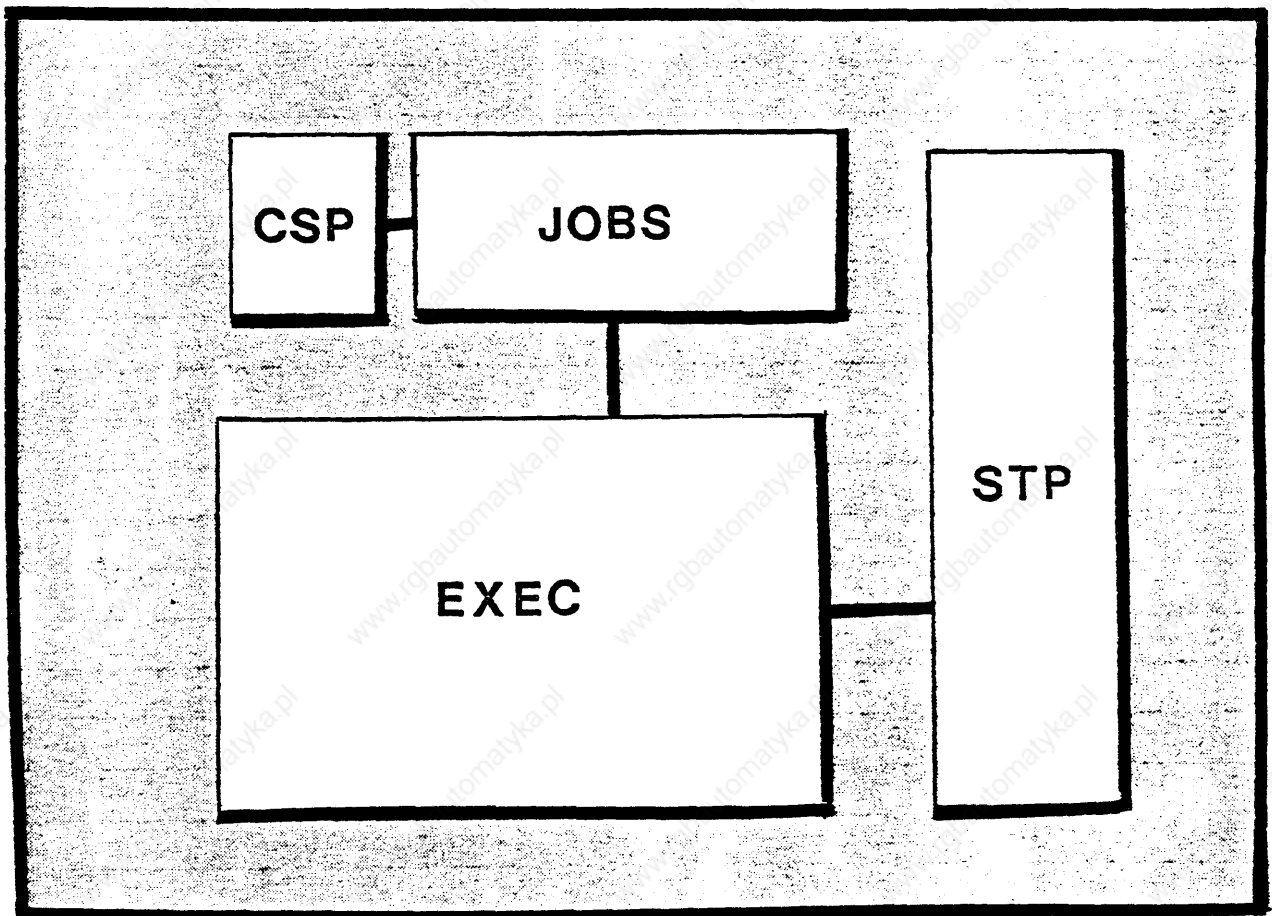
## COS SOFTWARE ELEMENTS

### ● RESIDENT SYSTEM PROGRAMS

EXECUTIVE (EXEC)

SYSTEM TASK PROCESSOR (STP)

CONTROL STATEMENT PROCESSOR (CSP) - OPTIONAL



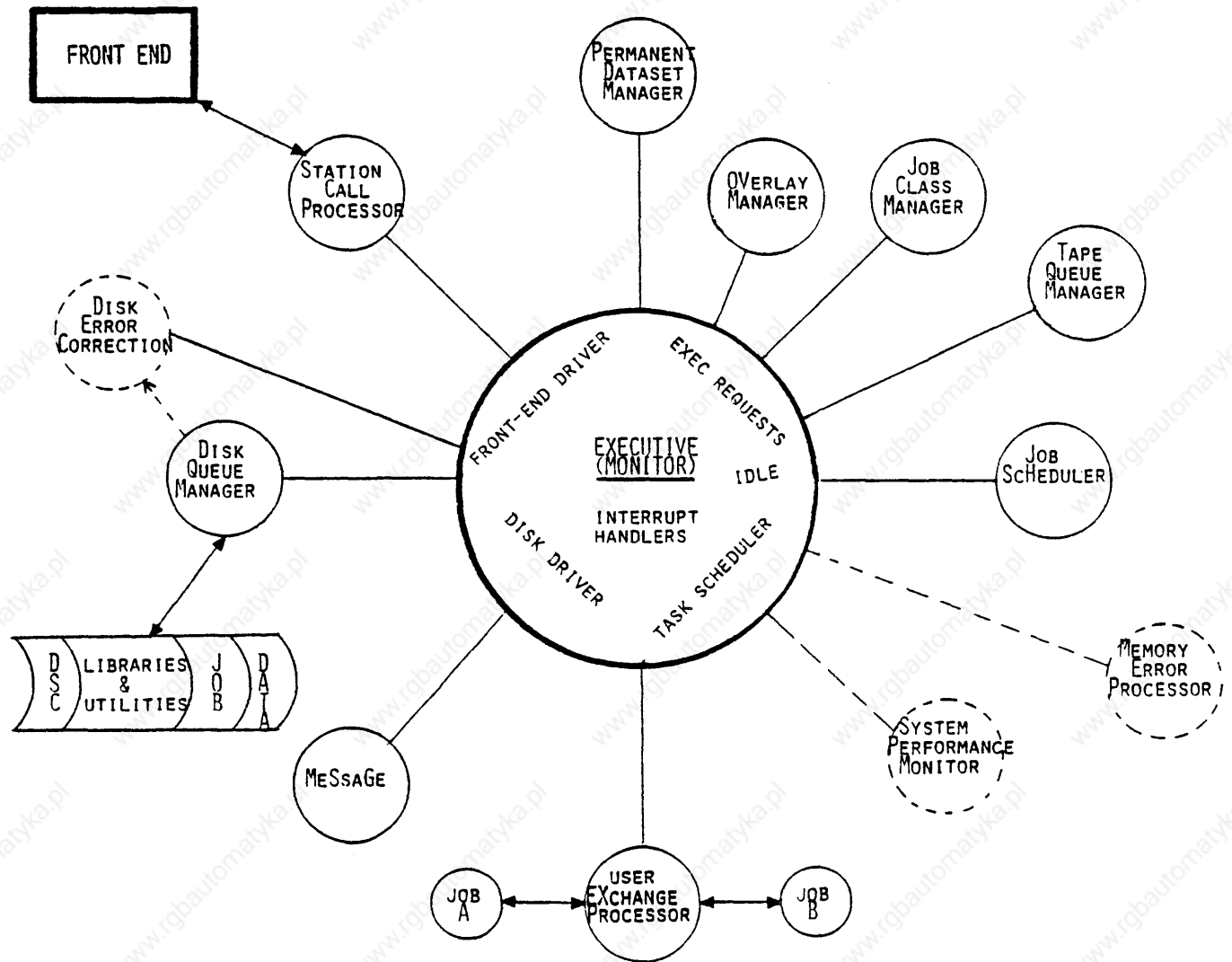
### ● NON-RESIDENT PROGRAMS

LIBRARIES

SYSTEM UTILITIES

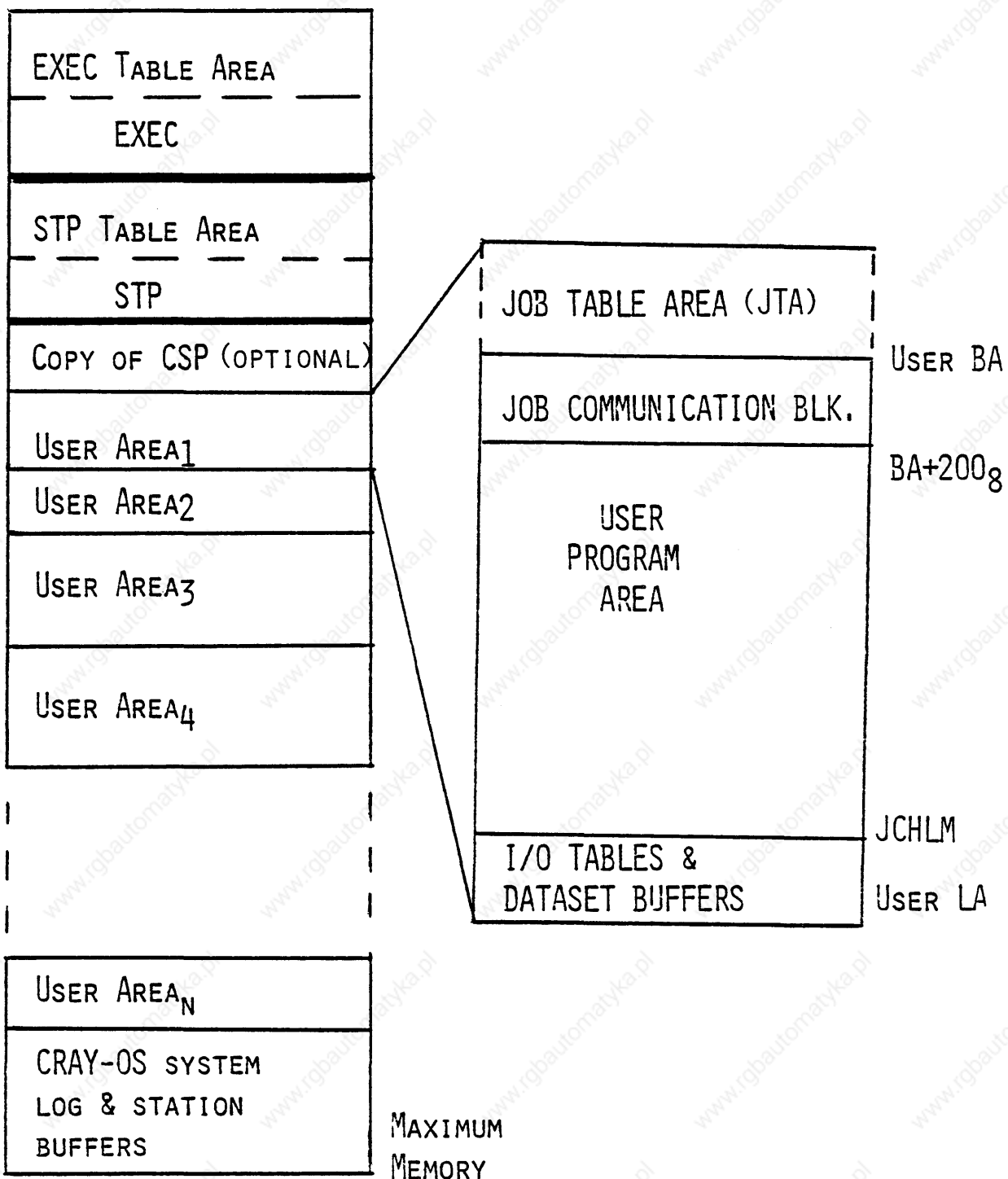
USER APPLICATIONS

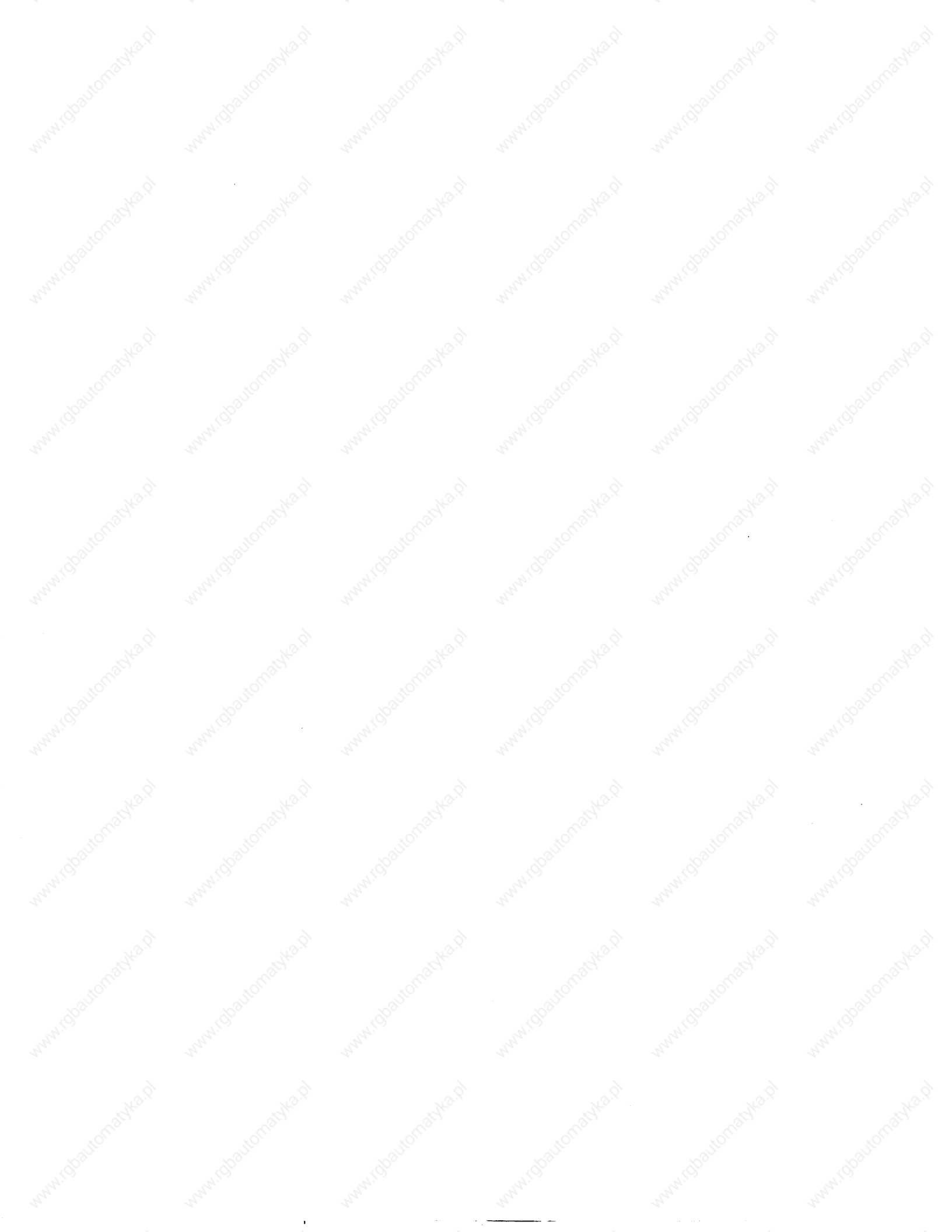






# SYSTEM MEMORY ASSIGNMENTS





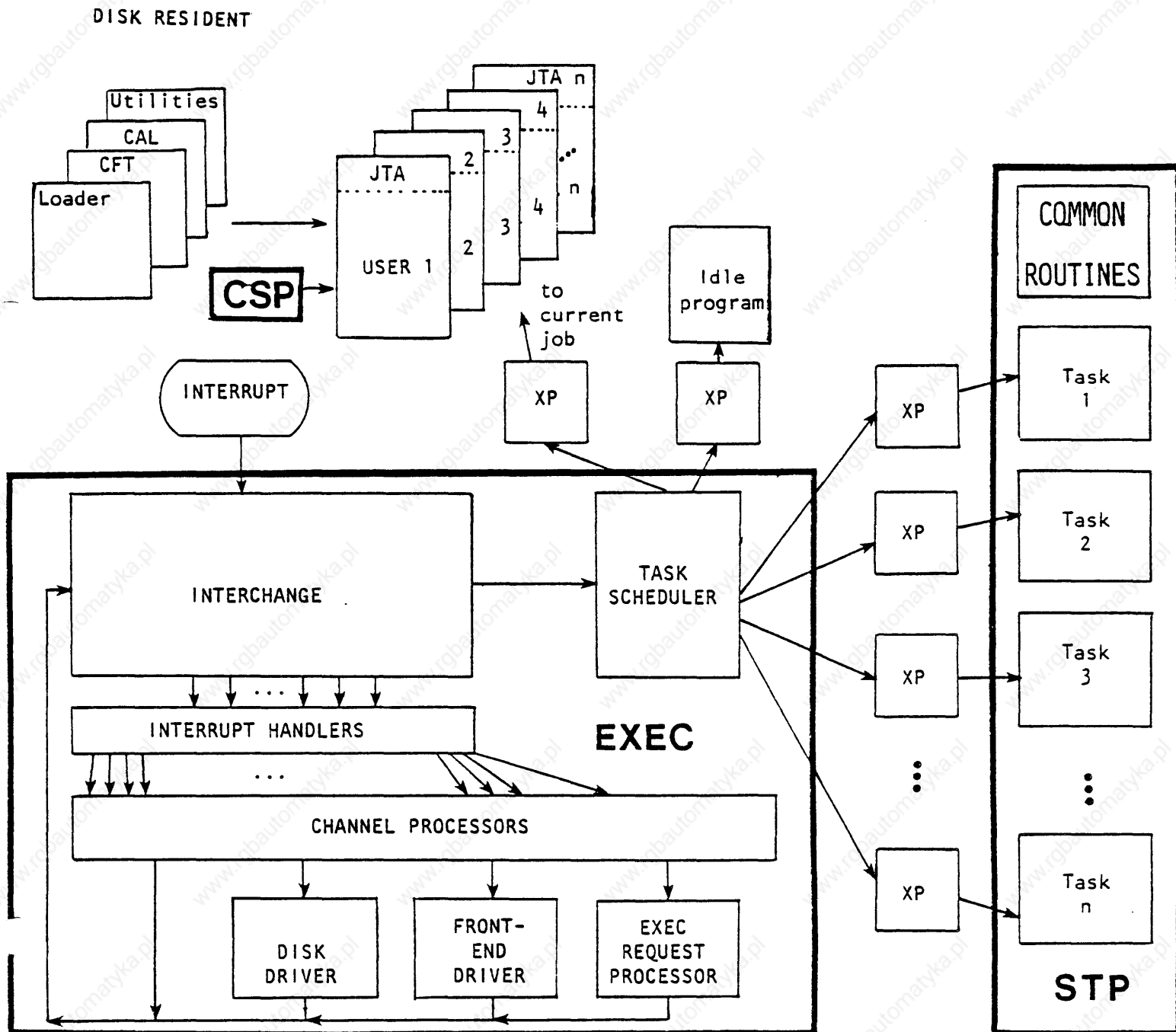


SYSTEM EXECUTIVE



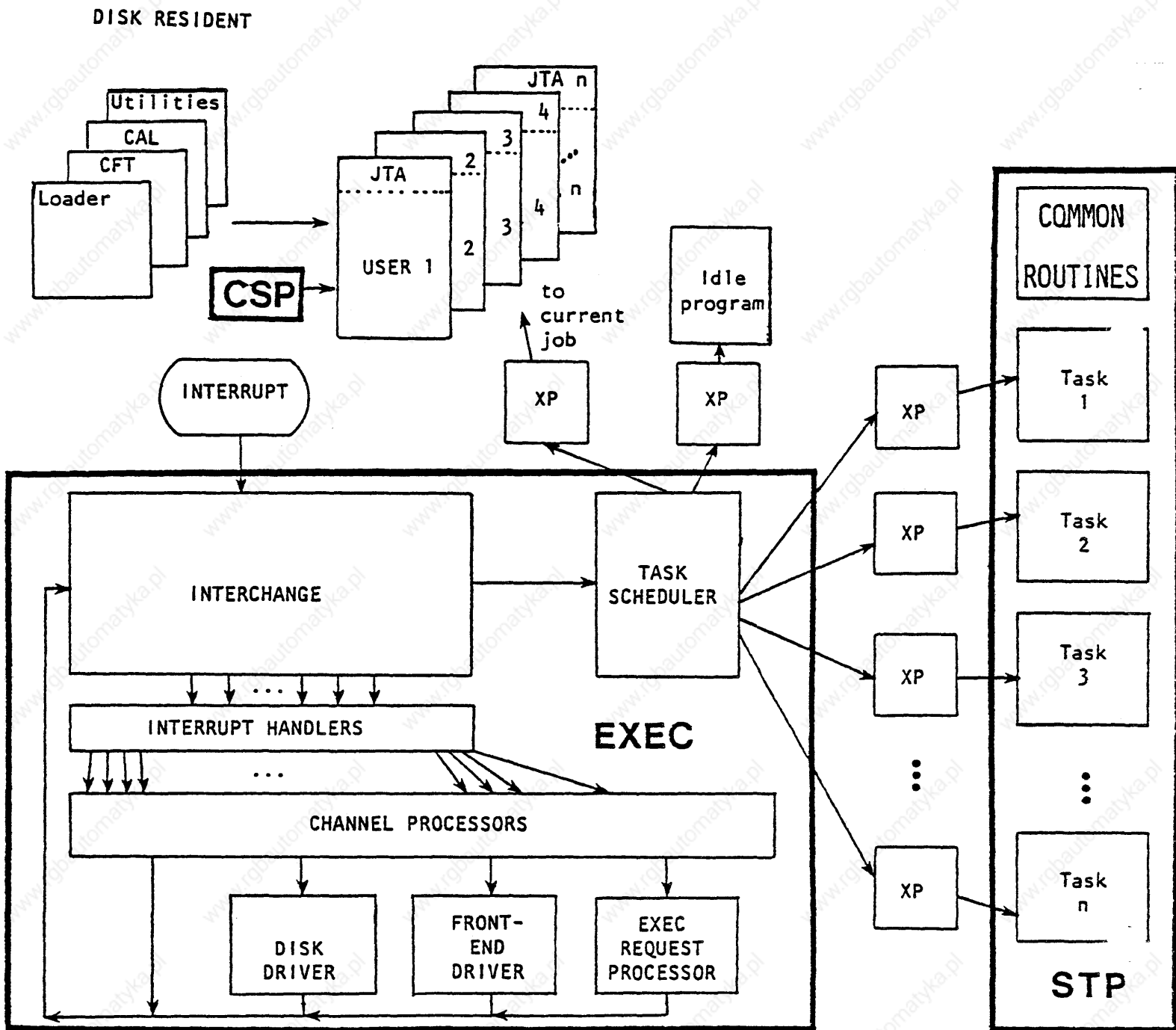
# SYSTEM EXECUTIVE PROGRAM (EXEC)

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



# SYSTEM EXECUTIVE PROGRAM (EXEC)

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



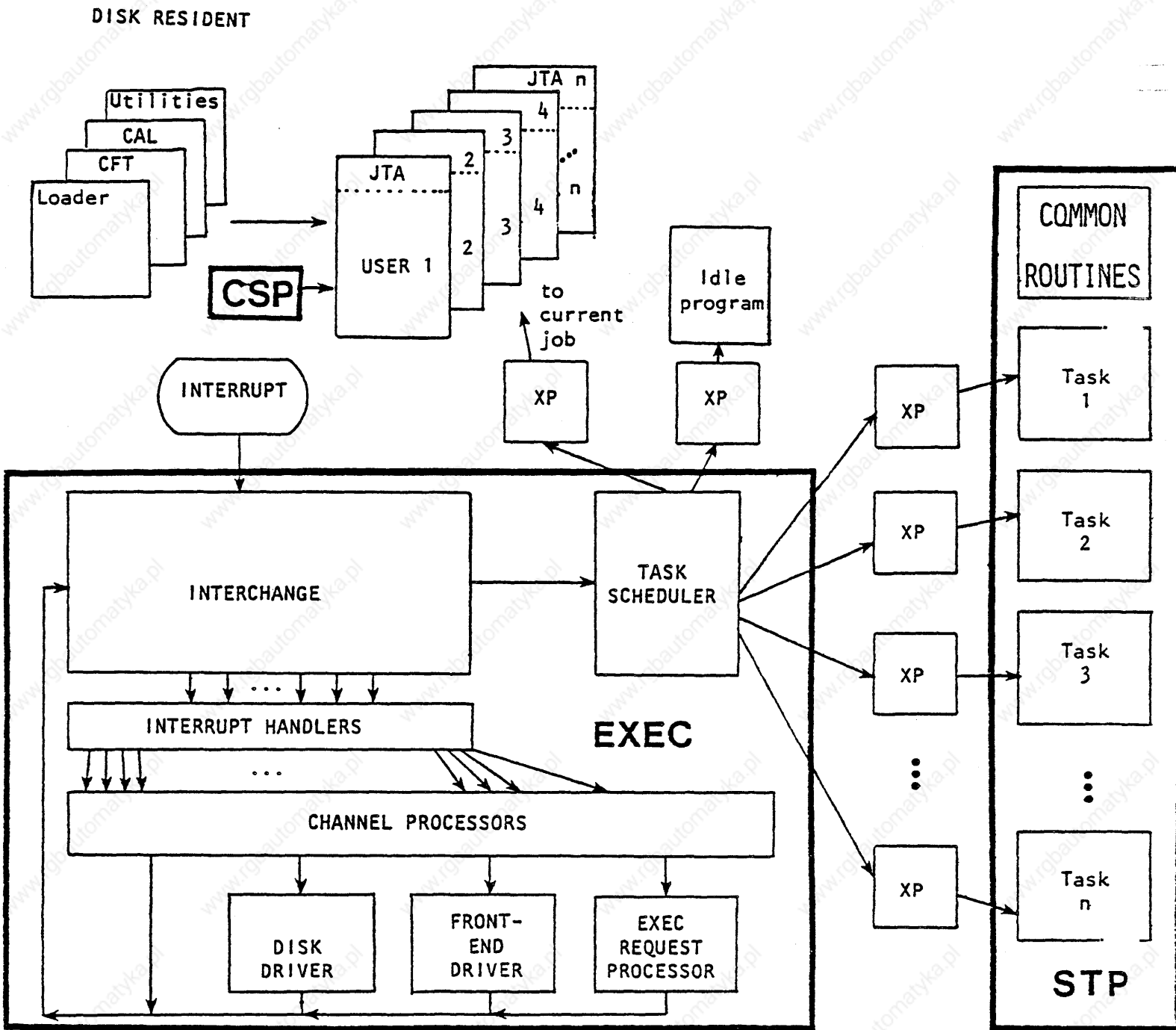
## FUNCTIONS OF EXEC

- 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

*Non Interruptible*



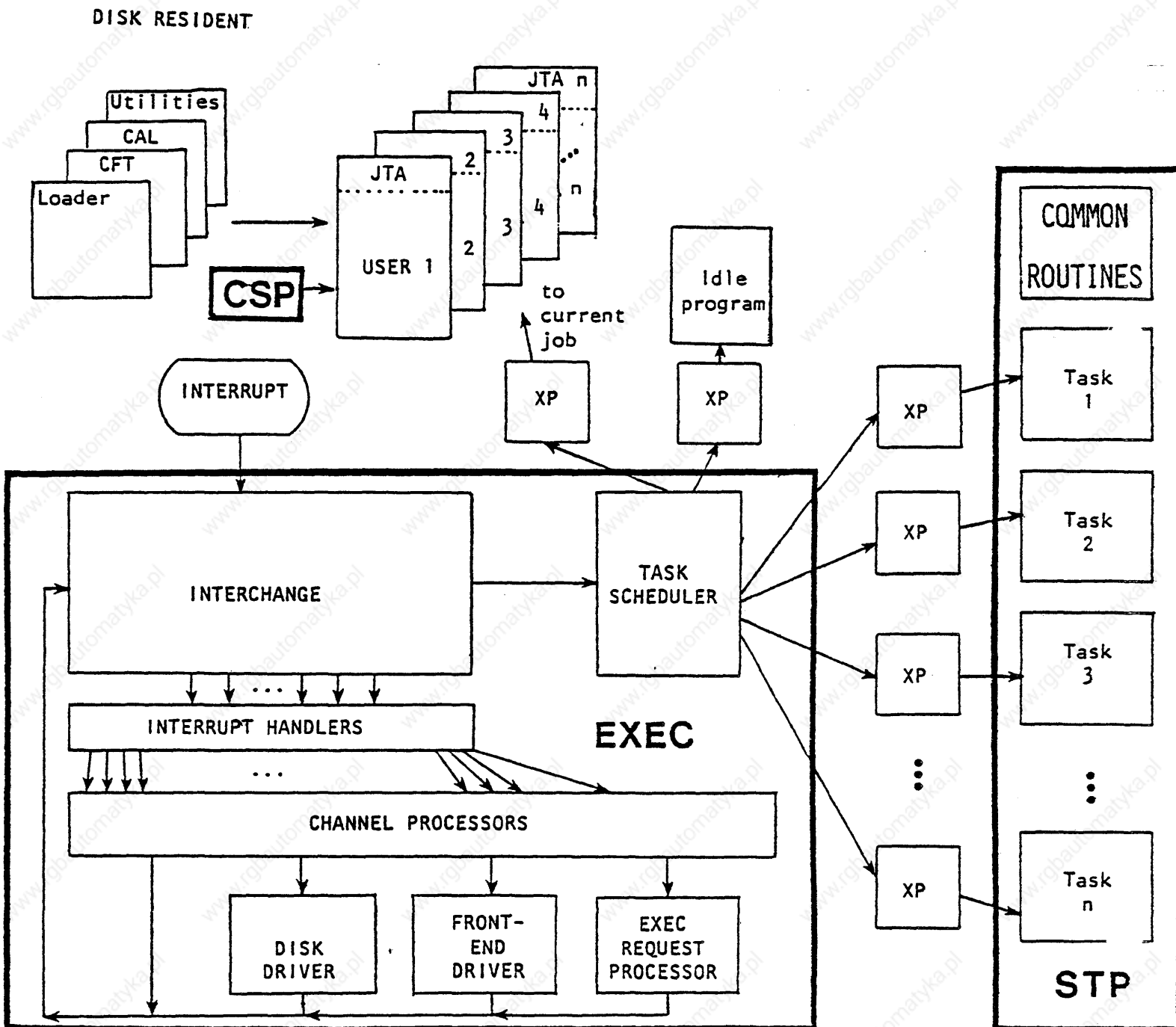
## INTERCHANGE

- 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  
m.u. external clock  
floating point err  
Prog range err  
Operand range err

# SYSTEM EXECUTIVE PROGRAM (EXEC)

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





## INTERRUPT HANDLERS

- EXECUTE ONE OF THE FOLLOWING INTERRUPT HANDLERS:

IOI I/O INTERRUPT HANDLER

CII CONSOLE INTERRUPT HANDLER

RTI REAL-TIME INTERRUPT HANDLER

NEI NORMAL EXIT INTERRUPT HANDLER

EEI ERROR EXIT INTERRUPT HANDLER

MEI MEMORY ERROR INTERRUPT HANDLER

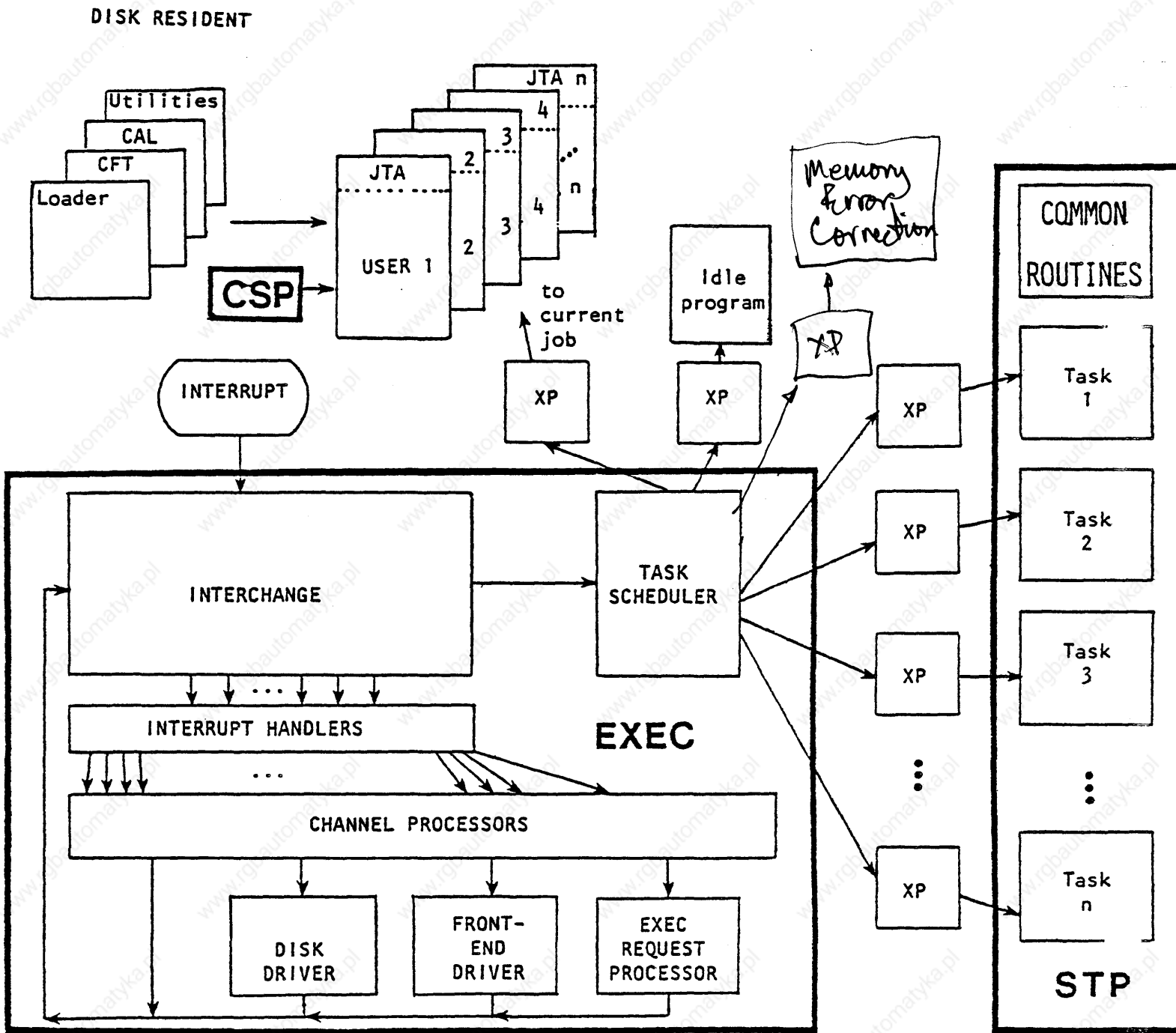
Not Used  
→

Programmable Clock Int. Hand.

- CLEAR THE INTERRUPT FLAG IN THE EXCHANGE PACKAGE OF THE INTERRUPTED PROGRAM
- DETERMINE THE CHANNEL ON WHICH THE INTERRUPT OCCURED
- BRANCH TO THE APPROPRIATE CHANNEL PROCESSOR

# SYSTEM EXECUTIVE PROGRAM (EXEC)

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



## CHANNEL PROCESSORS

- CHANNEL PAIRS ARE ASSIGNED NUMBERS AS FOLLOWS:

0	CONSOLE	<i>Interrupt for</i>	
1	<del>MCU CLOCK</del>	<i>Deadstart Signal</i>	
2	] - 12 I/O CHANNELS		
4			
6			
.			
.			
.			
24			
26		NORMAL EXIT PSEUDO CHANNEL	
28		ERROR EXIT PSEUDO CHANNEL	
30		PROGRAMMABLE CLOCK PSEUDO CHANNEL	
32		MEMORY ERROR PSEUDO CHANNEL	<i>→ Not Used</i>

Processor

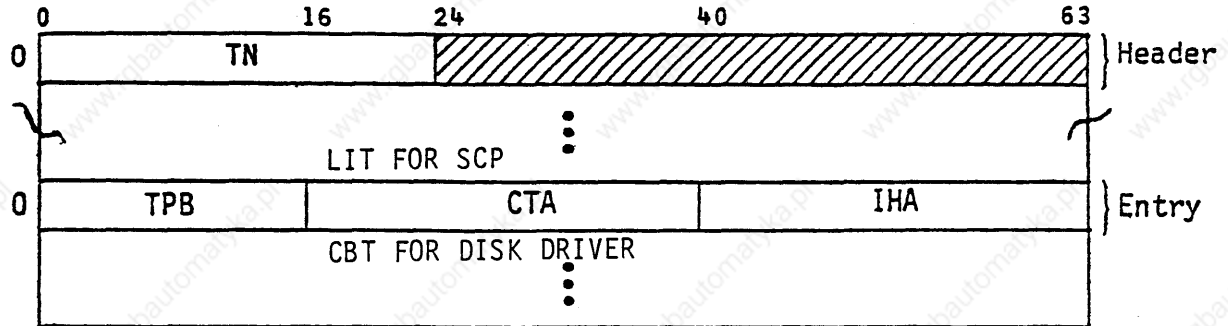
0 - CEP  
 1 - 14  
 2-24 RJ  
 Unused  
 26 - NE  
 (Create what rom)  
 task => For register  
 Create task table  
 28 - FE  
 Call error  
 30 - ME  
 (Not used)  
 30 - PCI  
 (Not used)

- 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



## CHANNEL TABLE - CHT

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.



### HEADER

<u>Field</u>	<u>Word</u>	<u>Bits</u>	<u>Description</u>
CHTN	0	0-23	Table name; "CHT" in ASCII

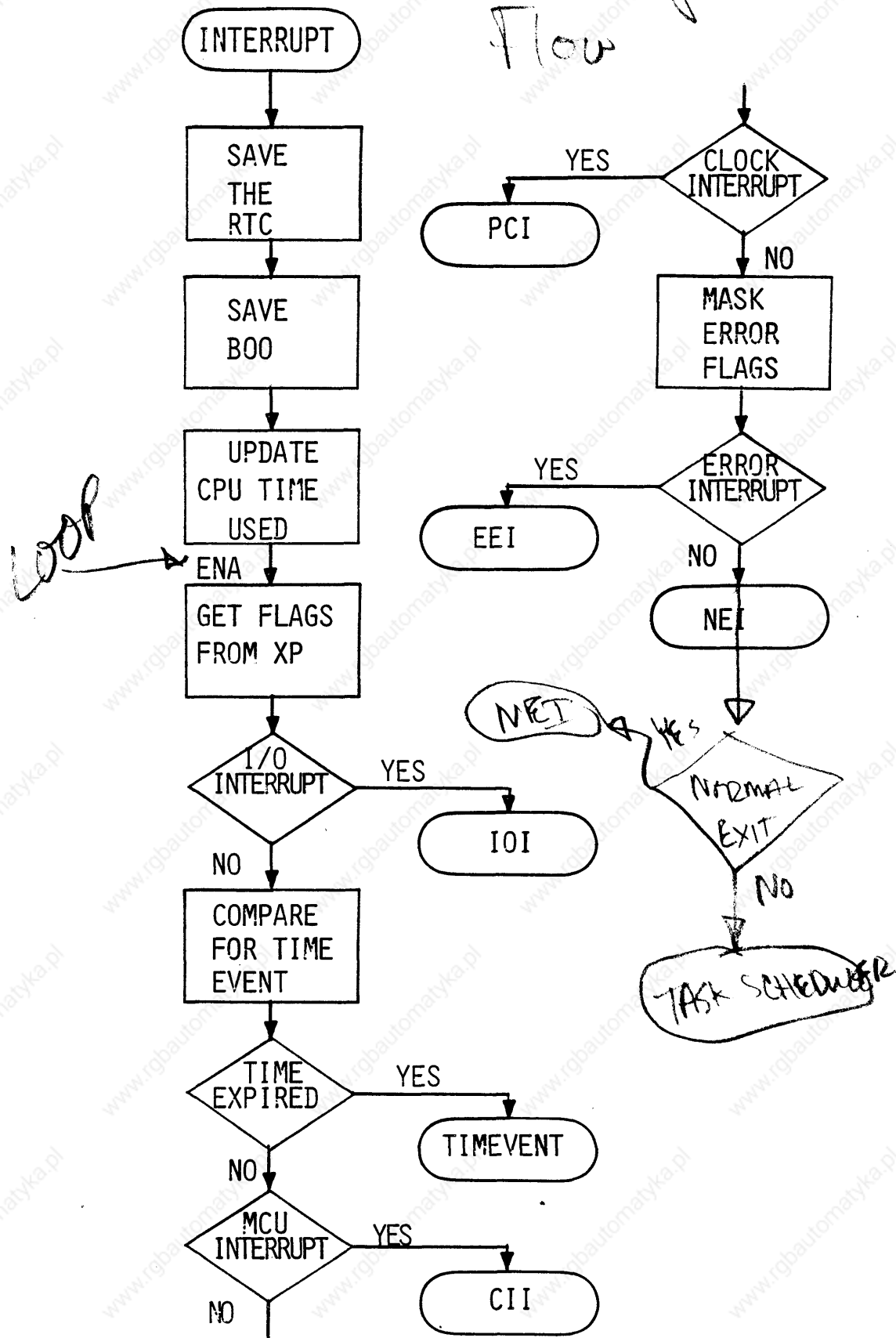
### ENTRY

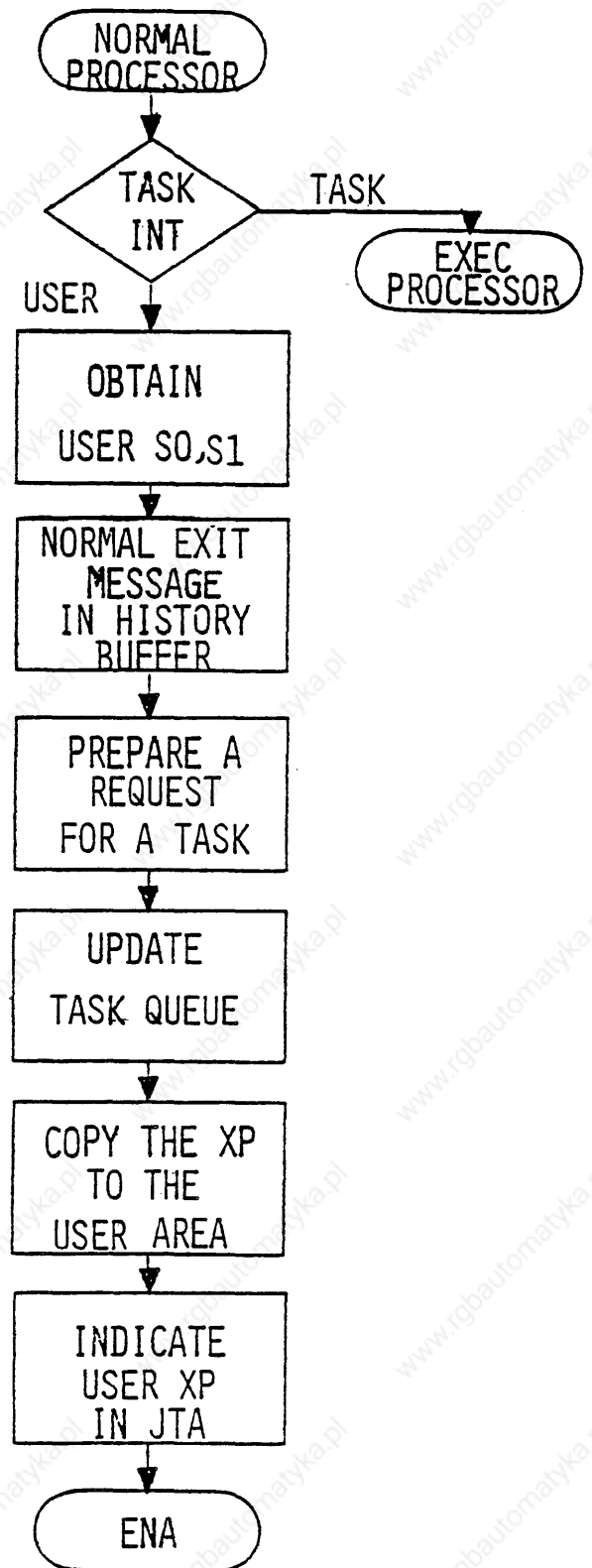
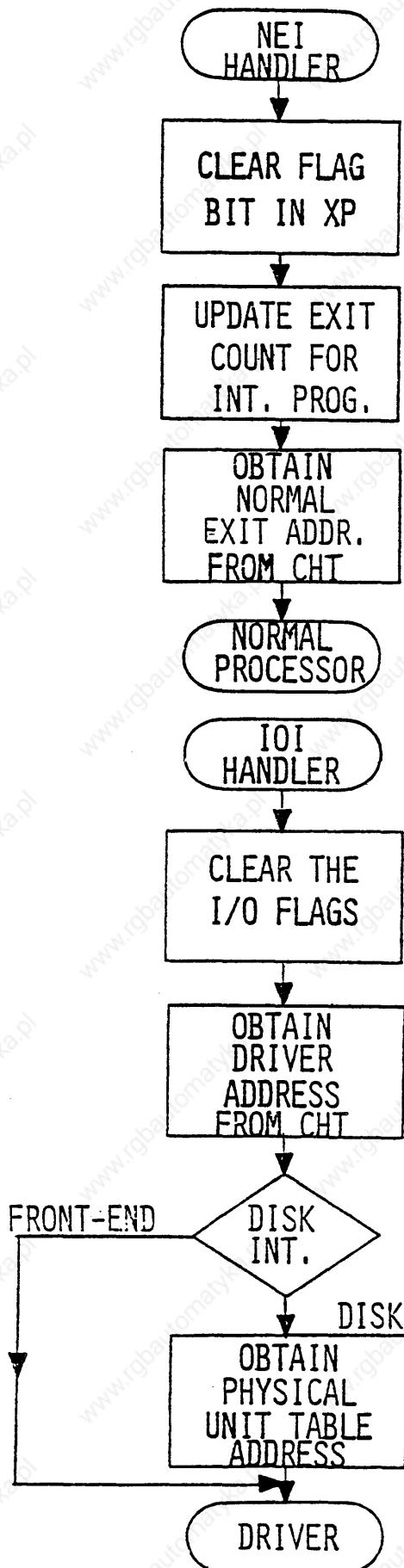
<u>Field</u>	<u>Word</u>	<u>Bits</u>	<u>Description</u>
CHTPB	0	0-15	Address of task parameter word
CHCTA	0	16-39	Control table address
CHIHA	0	40-63	Interrupt handler address

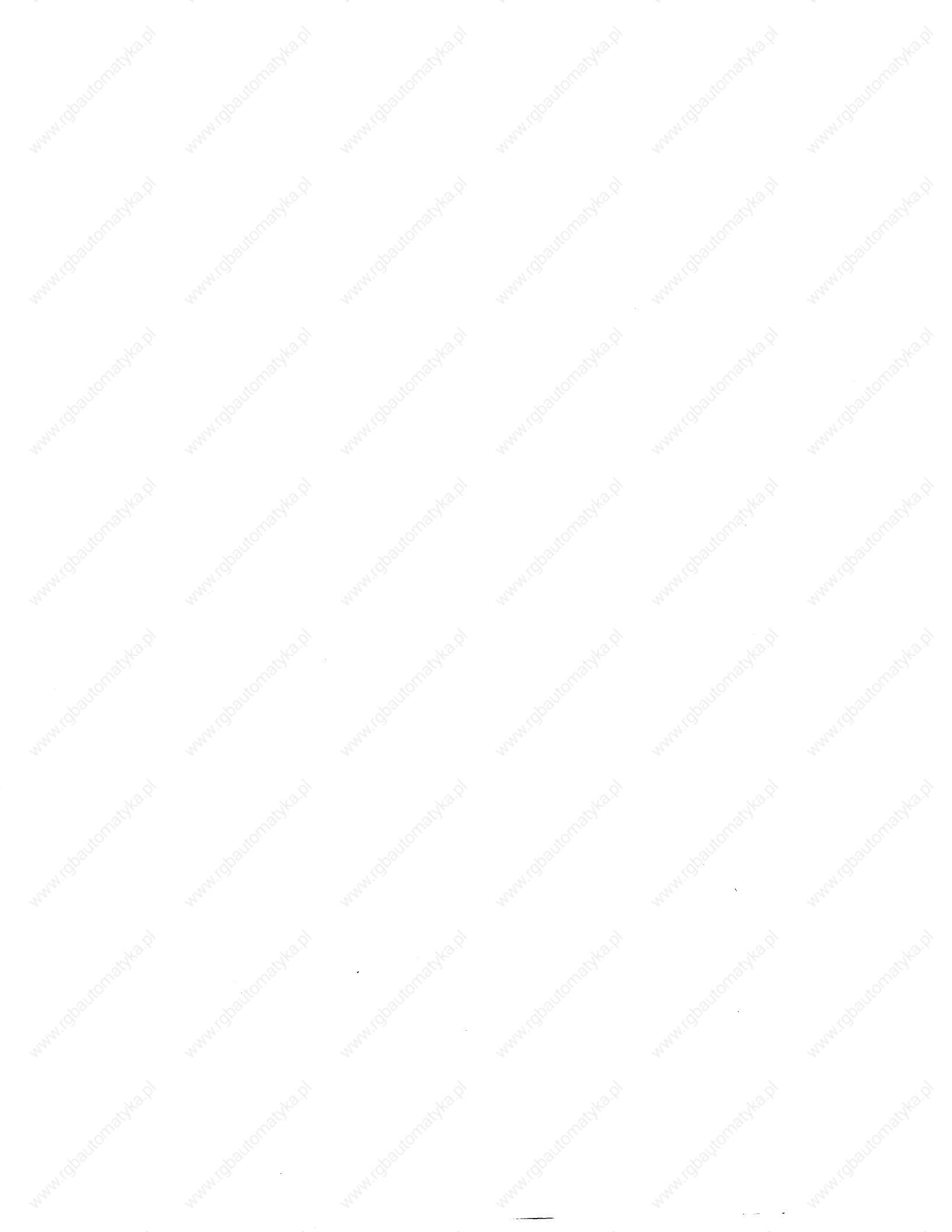
### CHANNEL PROCESSOR TABLE

00000000	04151025000000000000	0000000000230300064451	00000000000000000000	0016150007143200074370	CHT
00000001	0016150007143200075661	00000000002323000657324	00000000002323000657324	0000000000233300072623	
00000002	00000000002333000657324	00000000002343000657324	00000000002343000657324	00000000002353000687211	
00000003	00000000002353000657324	00000000002363000701131	00000000002363000657324	00000000002373000657324	
00000004	00000000002373000657324	00000000002383000657324	00000000002383000657324	0016150007157000074370	
00000005	0016150007157000075661	0016150007172600074370	0016150007172600075661	0016150007206400074370	
00000006	0016150007206400075661	0016150007222300074370	0016150007222300075667	0000000000245300064044	
00000007	00000000000000000000	0000000000246300064453	00000000000000000000	0000000000247300065135	
00000008	00000000000000000000	0000000000250300065215	00000000000000000000		

# Interchange Flow









## 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

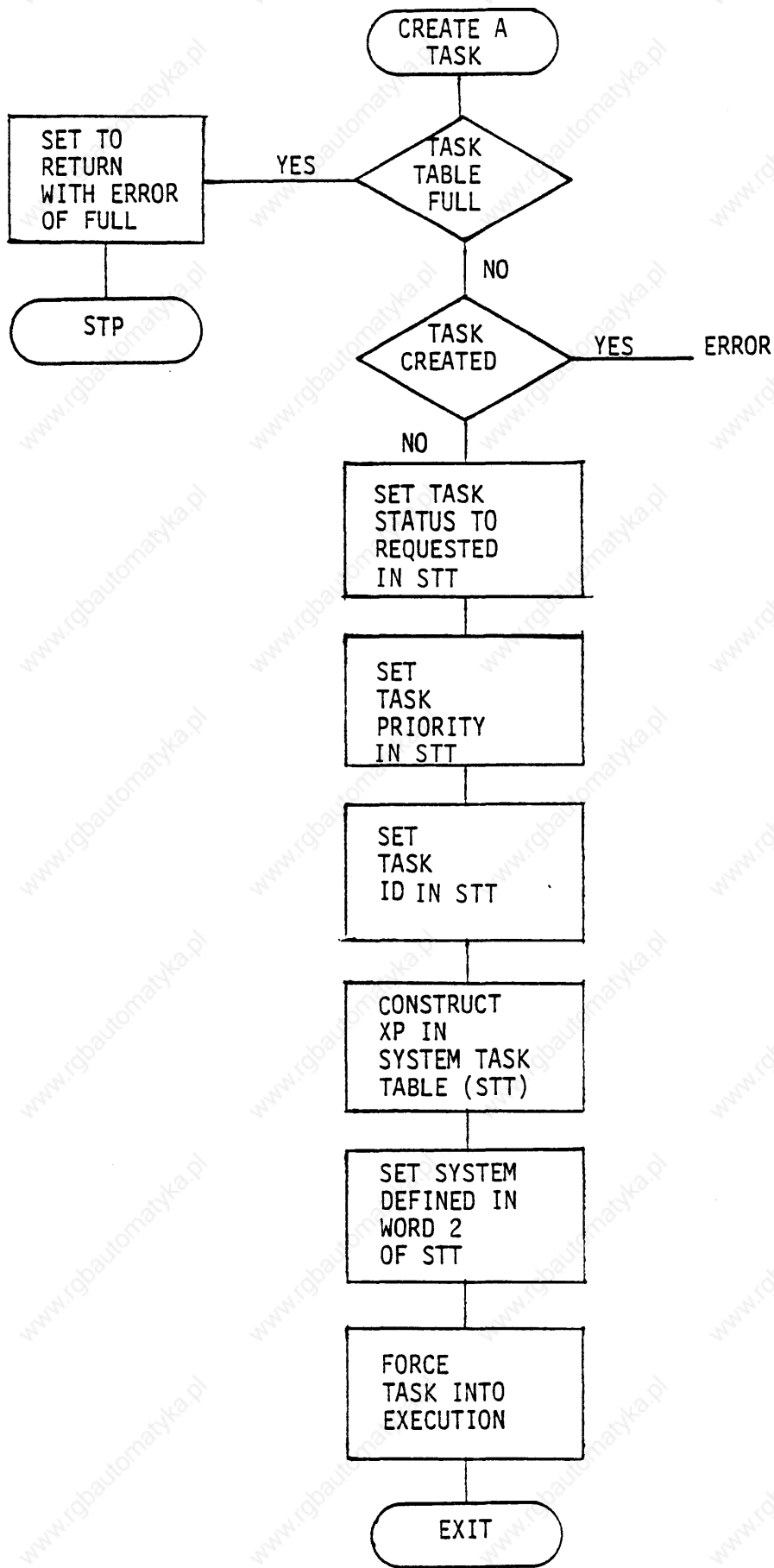
START/STOP OPERATING SYSTEM

DISPLAY MEMORY/EXCHANGE PACKAGE

ENTER MEMORY/EXCHANGE PACKAGE

SET/CLEAR SYSTEM BREAKPOINT



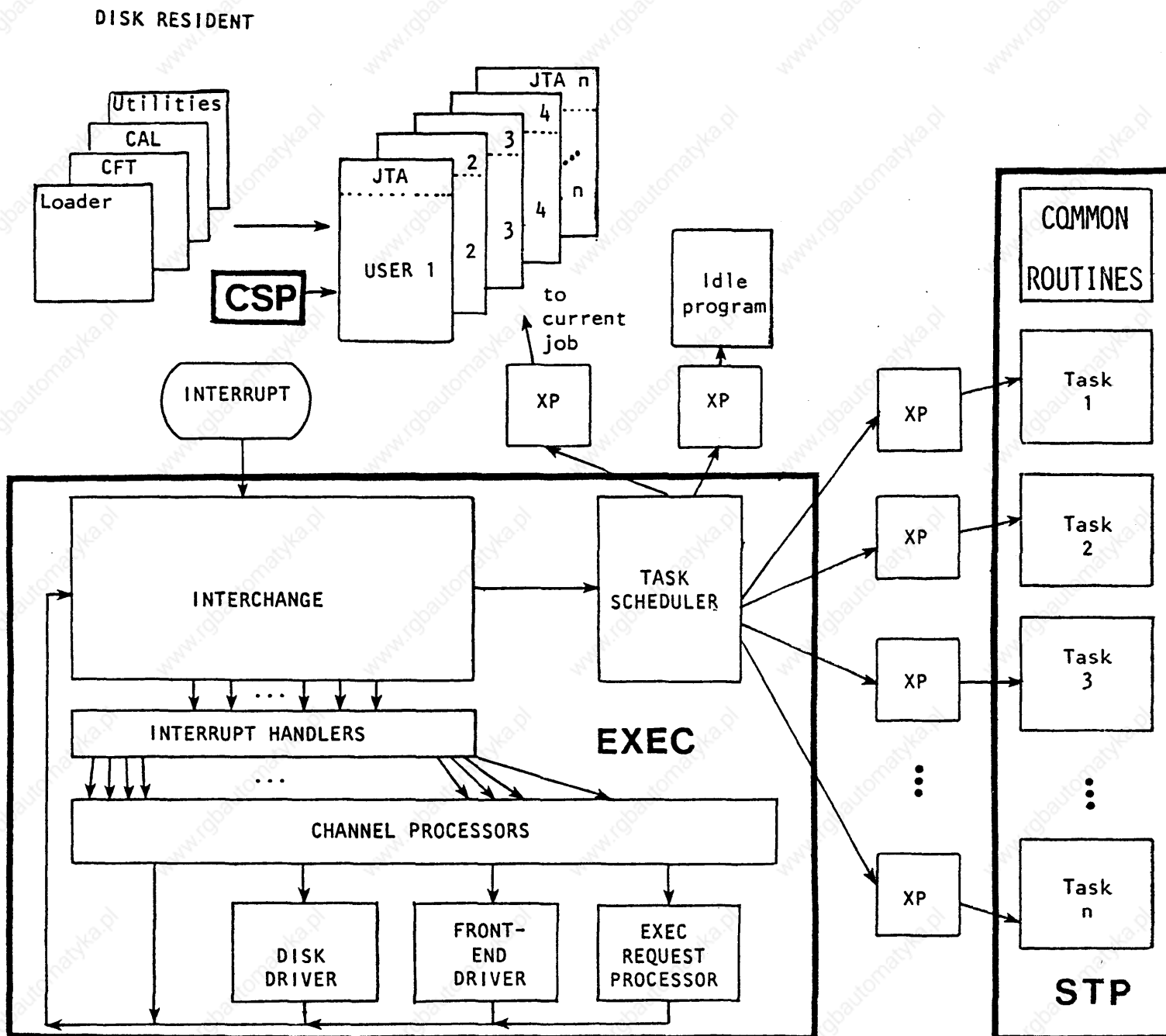


	0	16	40	48	56	63
S6				status	priority	task ID
S7				P-register for new task		01



# SYSTEM EXECUTIVE PROGRAM (EXEC)

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



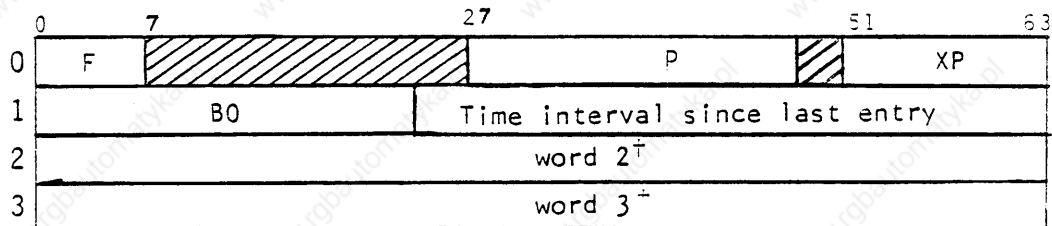


# EXEC HISTORY TRACE

- CIRCULAR BUFFER BEGINS AT LOCATION DBE
- LOCATION DBEP 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

\*  
\* HISTORY BUFFER FUNCTION FLAGS  
\*

04051423000000000000000001	DBUGM	CON	'ALL'L+1	UNIVERSAL ENABLE FLAG
04451722200000000000000000		CON	'IOI'L	I/O INTERRUPT
05251621200000000000000000		CON	'UNE'L	USER NORMAL EXIT
05151621200000000000000000		CON	'SNE'L	STP NORMAL EXIT
04251621200000000000000000		CON	'ENE'L	EXEC NORMAL EXIT
05112422200000000000000000		CON	'RTI'L	REAL TIME INTERRUPT
05412020600000000000000000		CON	'XPC'L	COPY USER XP TO JTA
04650325200000000000000000		CON	'MCU'L	MCU COMMAND
05150322045000000000000000		CON	'SCHJ'L	SCHEDULE USER JOB
04211123651000000000000000		CON	'DIOR'L	DISK DRIVER REQUEST
04452423251400000000000000		CON	'ITMS'L	INTER-TASK MESSAGE
04250522200000000000000000		CON	'EEI'L	ERROR EXIT
	17	BSSZ	DBUGM+DBUGN-W.*	
00000000000000000000000000	XCHGBP	CON	0	XCHG PACKAGE TRACE CURRENT POSITION
00000000000000000000000000	DBFP	CON	0	DEBUG BUFFER CURRENT POSITION



Field	Word	Bits	Description
F	0	0-6	Function number
P	0	27-50	Program register of interrupted exchange package
XP	0	51-63	Exchange package address





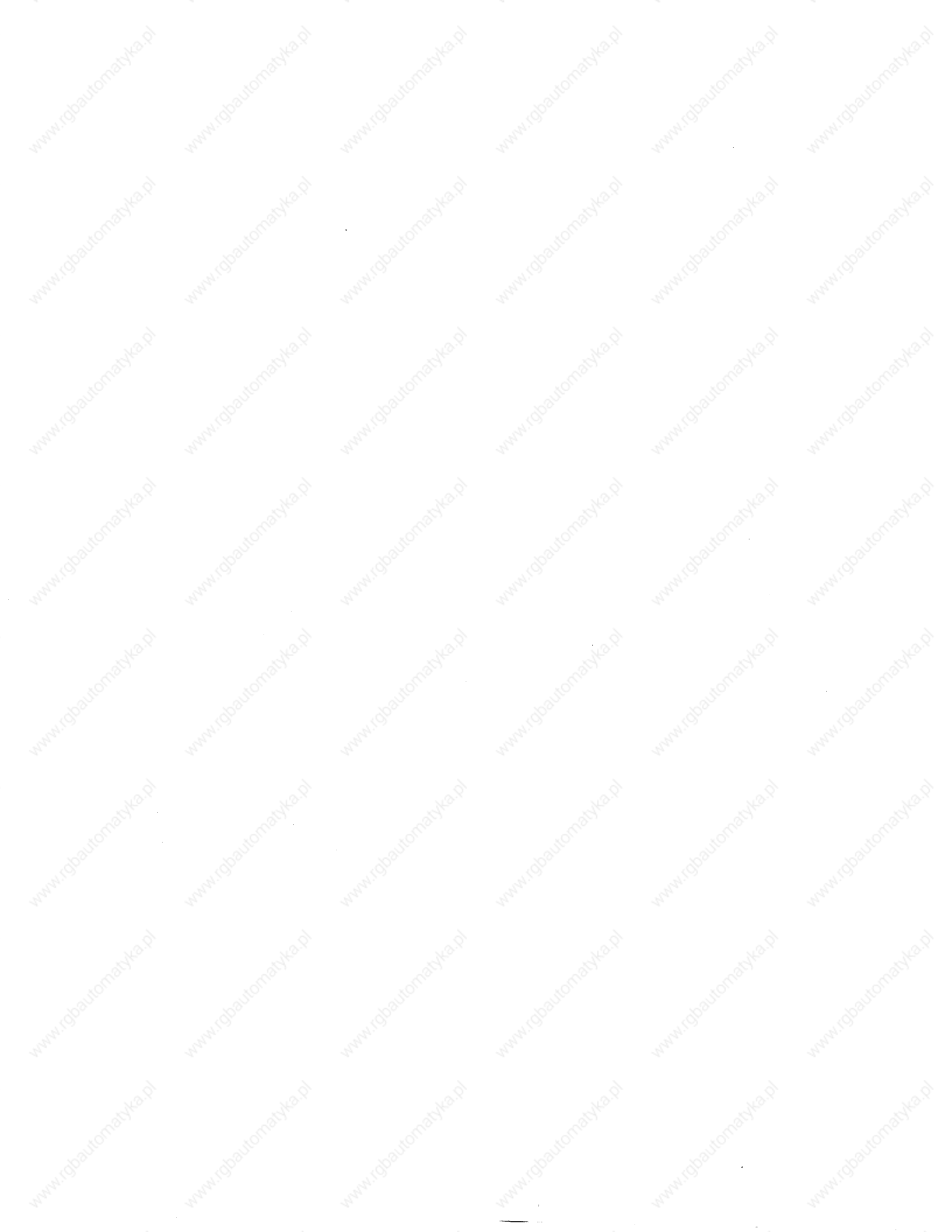
2.23

FUNCTION	P-REG	SAVED B0	XP	INTERVAL	WORDS 2 AND 3			
011	DQM	0001660A	0000001B	01500	00000000000472	0114400002027100070220	04000000000000000000	@
011	DQM	0001660A	0000001B	01500	00000000000276	0000010011110000145000	0000000000256000002557	@
001	I/O	0001660A	0377760A	01500	000000000003050	0000050000230300067053	0000000000000041777777	+
001	I/O	0001660A	0377760A	01500	000000000000673	0000050000230300067173	0000000000000041777777	
001	I/O	0001660A	0377760A	01500	000000000000501	0000040000230300067212	0000000000000041777777	
001	I/O	0001660A	0377760A	01500	000000000002340	0000050000230300067655	0100000000000041777777	
001	I/O	0001660A	0377760A	01500	000000000000714	0000050000230300066274	0100000000000041777777	
001	I/O	0001660A	0377760A	01500	000000000000612	0000040000230300066325	0100000000000041777777	
001	I/O	0001660A	0377760A	01500	000000000000774	0000050000230300067755	0100000000000041001177	
001	I/O	0001660A	0377760A	01500	000000000001122	0000050000230300070167	0100000000000041001177	
010	SCP	0001660A	0000000A	01500	000000000000777	0000000000000000000000	0013001265000777771500	+P @
004	ENE	0001660A	0001461C	01500	000000000000364	0000000000000000000000	0013001265000777771500	+P @
001	I/O	0001660C	0377760A	01500	000000000003235	0000040000230300070220	0100000000000041001177	
011	DQM	0001660C	0000001B	01500	00000000000472	0114400002027100070220	04000000000000000000	@
011	DQM	0001660C	0000001B	01500	00000000000276	0000010011110000145000	0000000000256000002557	@
001	I/O	0001660C	0377760A	01500	000000000003050	0000050000230300067053	0000000000000041777777	+
001	I/O	0001660C	0377760A	01500	000000000000673	0000050000230300067173	0000000000000041777777	
001	I/O	0001660C	0377760A	01500	000000000000501	0000040000230300067212	0000000000000041777777	
001	I/O	0001660C	0377760A	01500	000000000002340	0000050000230300067655	0100000000000041777777	
001	I/O	0001660C	0377760A	01500	000000000000714	0000050000230300066274	0100000000000041777777	
001	I/O	0001660C	0377760A	01500	000000000000612	0000040000230300066325	0100000000000041777777	
001	I/O	0001660C	0377760A	01500	000000000000774	0000050000230300067755	0100000000000041001177	
001	I/O	0001660C	0377760A	01500	000000000001122	0000050000230300070167	0100000000000041001177	
010	SCP	0001660C	0000000A	01500	000000000000777	0000000000000000000000	0013001265000777771500	+P @
004	ENE	0001660C	0001461C	01500	000000000000364	0000000000000000000000	0013001265000777771500	+P @
001	I/O	0001652C	0377760A	01500	000000000003270	0000040000230300070220	0100000000000041001177	
011	DQM	0001652C	0000001B	01500	00000000000472	0114400002027100070220	04000000000000000000	@
011	DQM	0001652C	0000001B	01500	00000000000276	0000010011110000145000	0000000000256000002557	@
001	I/O	0001652C	0377760A	01500	000000000003050	0000050000230300067053	0000000000000041777777	+
001	I/O	0001652C	0377760A	01500	000000000000673	0000050000230300067173	0000000000000041777777	
001	I/O	0001652C	0377760A	01500	000000000000501	0000040000230300067212	0000000000000041777777	
005	RTI	0001652C	0000000C	01500	000000000002025	1000000000000000000000	0013001265000777771500	+P @
004	ENE	0101636D	0101676A	02060	00000000001345	0000000000000000001650	0013001265000777771500	+P @
005	RTI	0050606B	0102432B	02060	000000000006456	0000000000000000001650	0013001265000777771500	+P @
003	SNE	0050606B	0102432B	02060	000000000000753	0000000202020200000714	000000000000001212020	
012	ITM	0050606B	0102432B	02060	000000000000250	1000000003030700033333	000000000000000036610	@ 6 =



FUNCTION	P-REG	SAVED B0	XP	INTERVAL	WORDS 2 AND 3			
004	ENE	0050606D	0102432B	02060	00000000000623	00000000000000001650	001300126500077771500	+P @
005	RTI	0050647D	0102432B	02060	00000000001554	00000000000000001650	001300126500077771500	+P @
003	SNE	0050647D	0102432B	02060	00000000000753	00000000000000000005	00000000000000000002	
004	ENE	0050650B	0102432B	02060	00000000001632	00000000000000001650	001300126500077771500	+P @
005	RTI	0050606B	0102512D	02060	00000000004210	00000000000000001650	001300126500077771500	+P @
003	SNE	0050606B	0102512D	02060	00000000000753	0000000202020200000704	0000000000000001212020	
012	ITM	0050606B	0102512D	02060	00000000000250	1000000001050000104075	0000000000000000000000	@ ==
004	ENE	0050606D	0102512D	02060	00000000000623	00000000000000001650	001300126500077771500	+P @
005	RTI	0051004A	0102165C	02060	00000000003561	00000000000000001650	001300126500077771500	+P @
003	SNE	0051004A	0102165C	02060	00000000000753	0400030000000000000000	0000000000000001212020 @	
012	ITM	0051004A	0102165C	02060	00000000000250	0000000000000000000007	0007347520054000064424 @	
004	ENE	0051004C	0102165C	02060	00000000000623	00000000000000001650	001300126500077771500	+P @
005	RTI	0051005C	0102165C	02060	00000000000733	00000000000000001650	001300126500077771500	+P @
003	SNE	0051005C	0102165C	02060	00000000000753	00000000000000000005	0000000000000000000002	
004	ENE	0051006A	0102165C	02060	00000000001630	00000000000000001650	001300126500077771500	+P @
005	RTI	0103051A	0101676A	02060	00000000003430	00000000000000001650	001300126500077771500	+P @
003	SNE	0103051A	0101676A	02060	00000000000753	0000000030107400045310	0000000005426200002011	< J X
011	DQM	0103051A	2646214A	02060	00000000000366	0000020207040000323200	0000000005426200002011	X
001	I/O	0103051C	0377760A	02060	00000000001715	0000050000230300067656	010000000000040777777	
001	I/O	0103051C	0377760A	02060	00000000000714	0000050000230300066274	010000000000040777777	
004	ENE	0103051C	0101676A	02060	00000000001012	00000000000000001650	001300126500077771500	+P @
001	I/O	0103051C	0377760A	02060	00000000000707	0000040000230300066325	010000000000040777777	
001	I/O	0103051C	0377760A	02060	00000000000774	0000050000230300067755	010000000000040020777	!
001	I/O	0103051C	0377760A	02060	00000000001156	0000050000230300070337	010000000000040020777	!
005	RTI	0103051C	0377760A	02060	00000000000640	00000000000000001650	001300126500077771500	+P @
004	ENE	0103051C	0101676A	02060	00000000000747	00000000000000001650	001300126500077771500	+P @
005	RTI	0101636B	0101676A	02060	00000000001120	00000000000000001650	001300126500077771500	+P @
003	SNE	0101636B	0101676A	02060	00000000001010	00000000000000000005	00000000000000000003	
004	ENE	0074442D	0074450B	02100	00000000001414	00000000000000001657	001300126500077771500	+P @
001	I/O	0050437C	0377760A	02100	000000000006010	0000040000230300070370	010000000000040020777	!
005	RTI	0050437C	0377760A	02100	00000000000753	00000000000000001657	001300126500077771500	+P @
004	ENE	0050437C	0050335B	02100	00000000000747	00000000000000001657	001300126500077771500	+P @
005	RTI	0050412A	0075103B	02100	00000000004246	00000000000000001657	001300126500077771500	+P @
013	ERR	0050412A	0075103B	02100	00000000000670	0515242500000000000000	00000200000000000000 STT	

2.25



## 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 (0-35<sub>g</sub>).  
HAVE THEIR OWN PRIORITIES (0-377<sub>g</sub>).  
BA AND LA THE SAME FOR ALL TASKS (BA=B@STP,  
LA=I@MEM)  
COMMUNICATE WITH EXEC, EACH OTHER AND WITH  
USER JOBS.

● TABLES  
USED BY STP AND EXEC



## STP COMMON ROUTINES

- USED BY TASKS TO PERFORM:

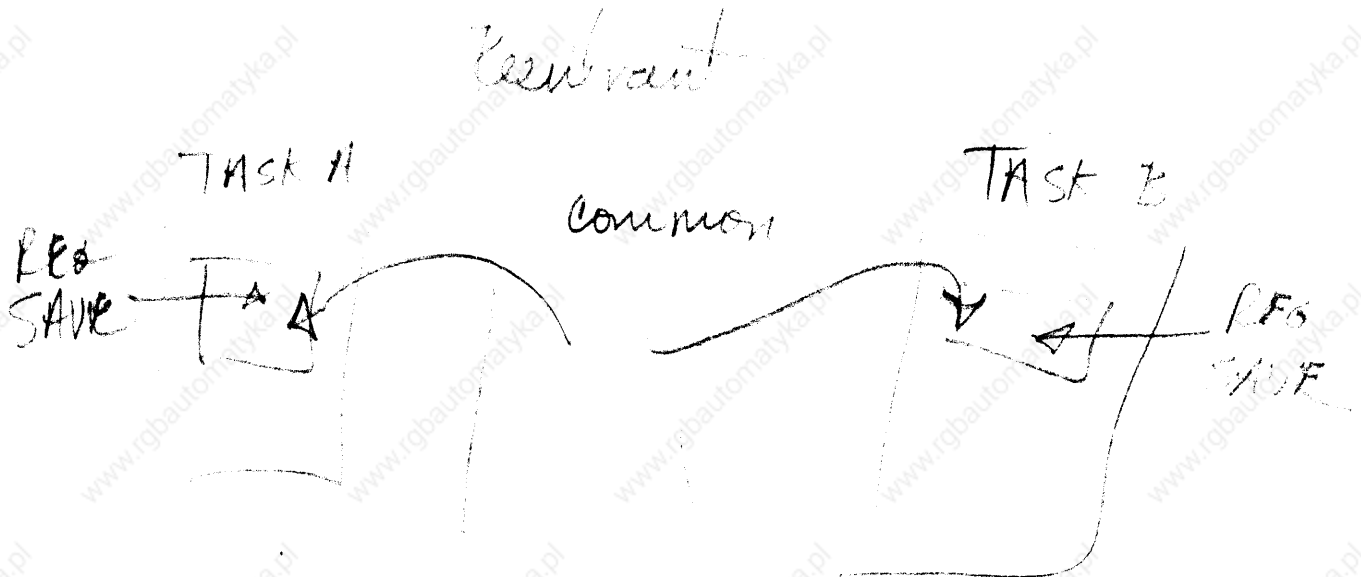
TASK LOGICAL INPUT/OUTPUT (TIO)

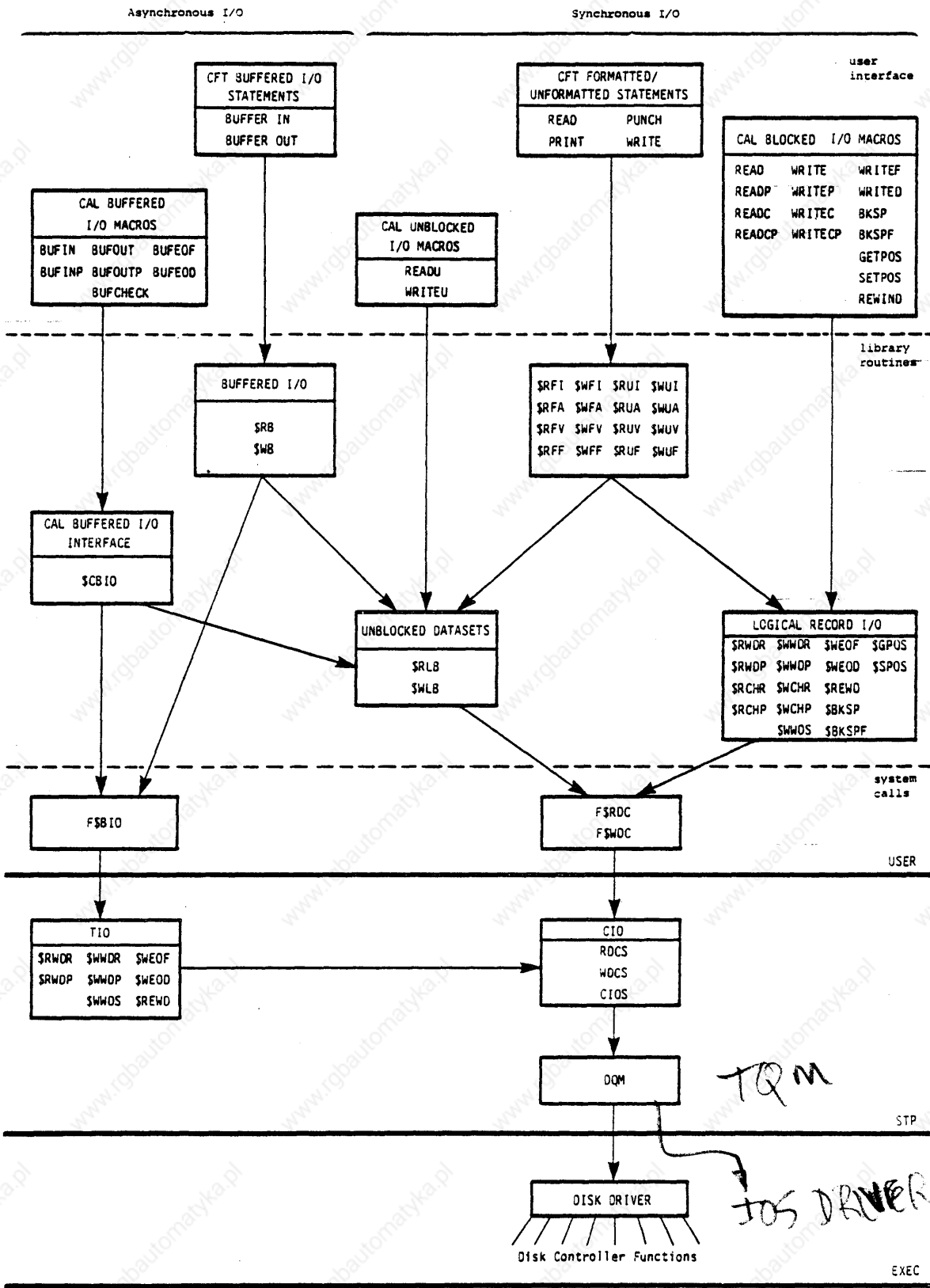
CIRCULAR INPUT/OUTPUT (CIO)

MEMORY MANAGEMENT

ITEM CHAINING/UNCHAINING

TASK TO TASK COMMUNICATION



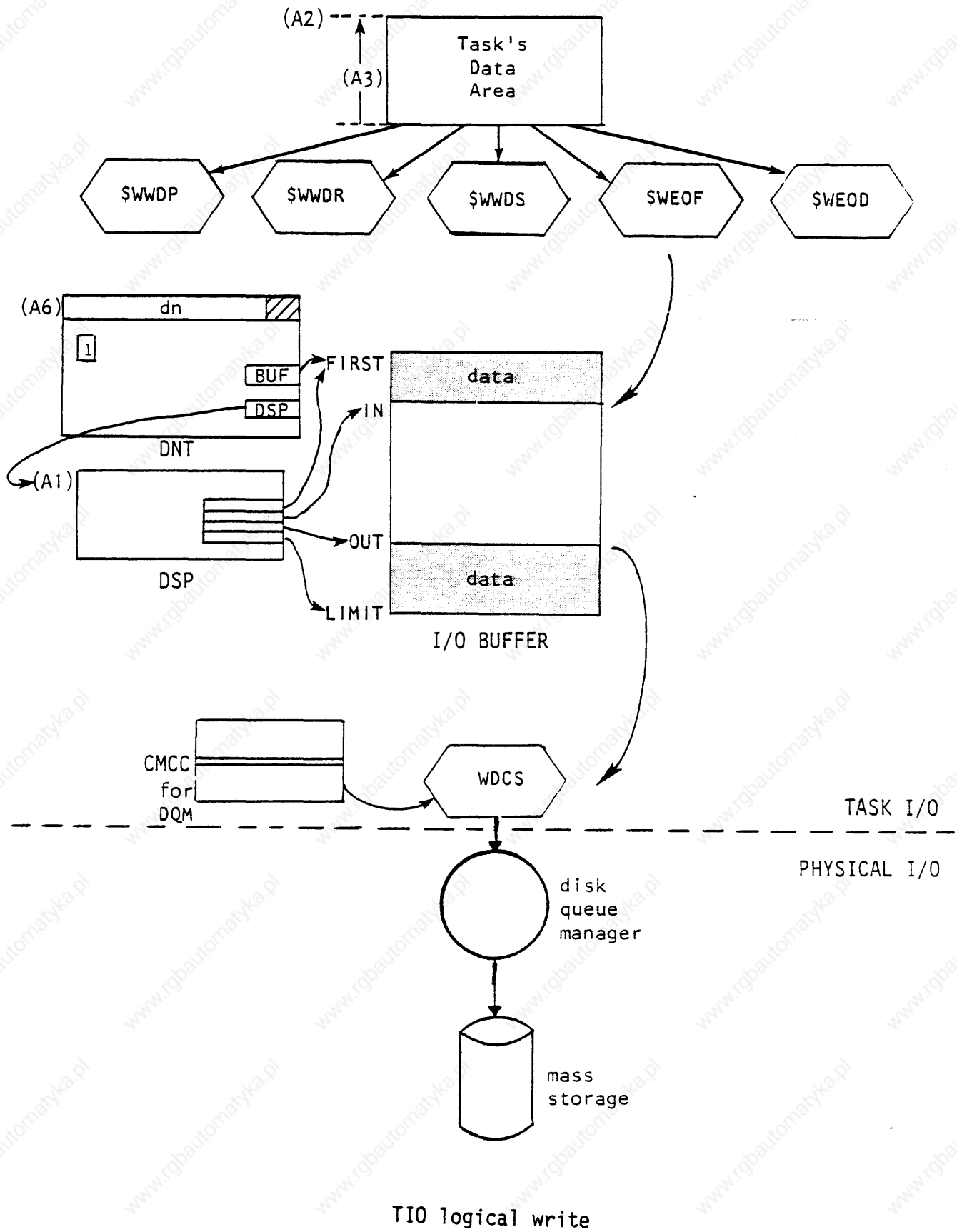


Overview of COS I/O

## 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
  - \$WWDW/\$WWDW-WRITE WORDS PARTIAL/FULL RECORD
  - \$WEOF-WRITE END OF FILE
  - \$WEOD-WRITE END OF DATA
  - \$REWDR-REWIND A DATASET
  - \$WWDW-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 (\$WWDW/\$WWDW) WILL BE EXAMINED HERE. REFER TO SMO040 FOR REMAINDER OF TIO ROUTINES.



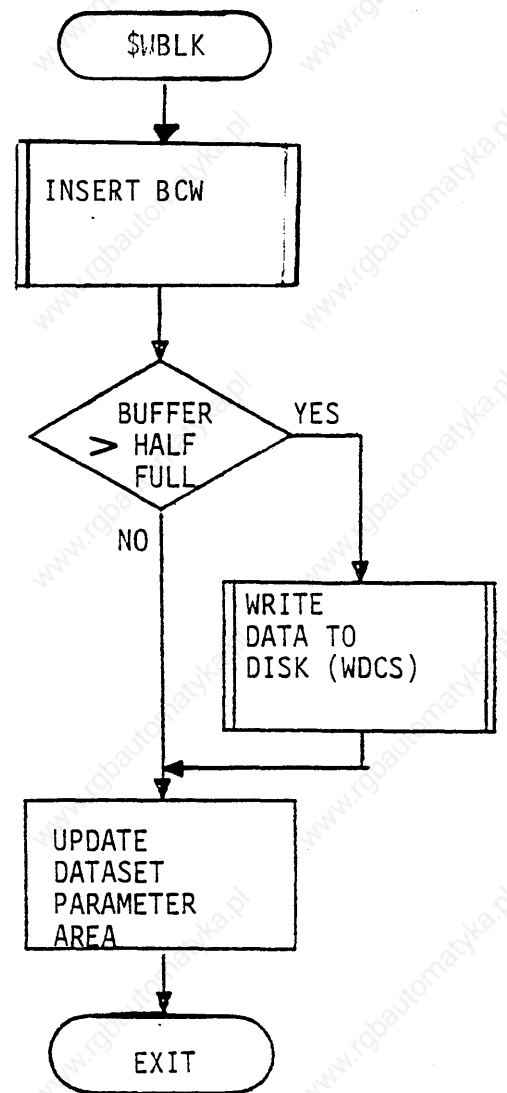
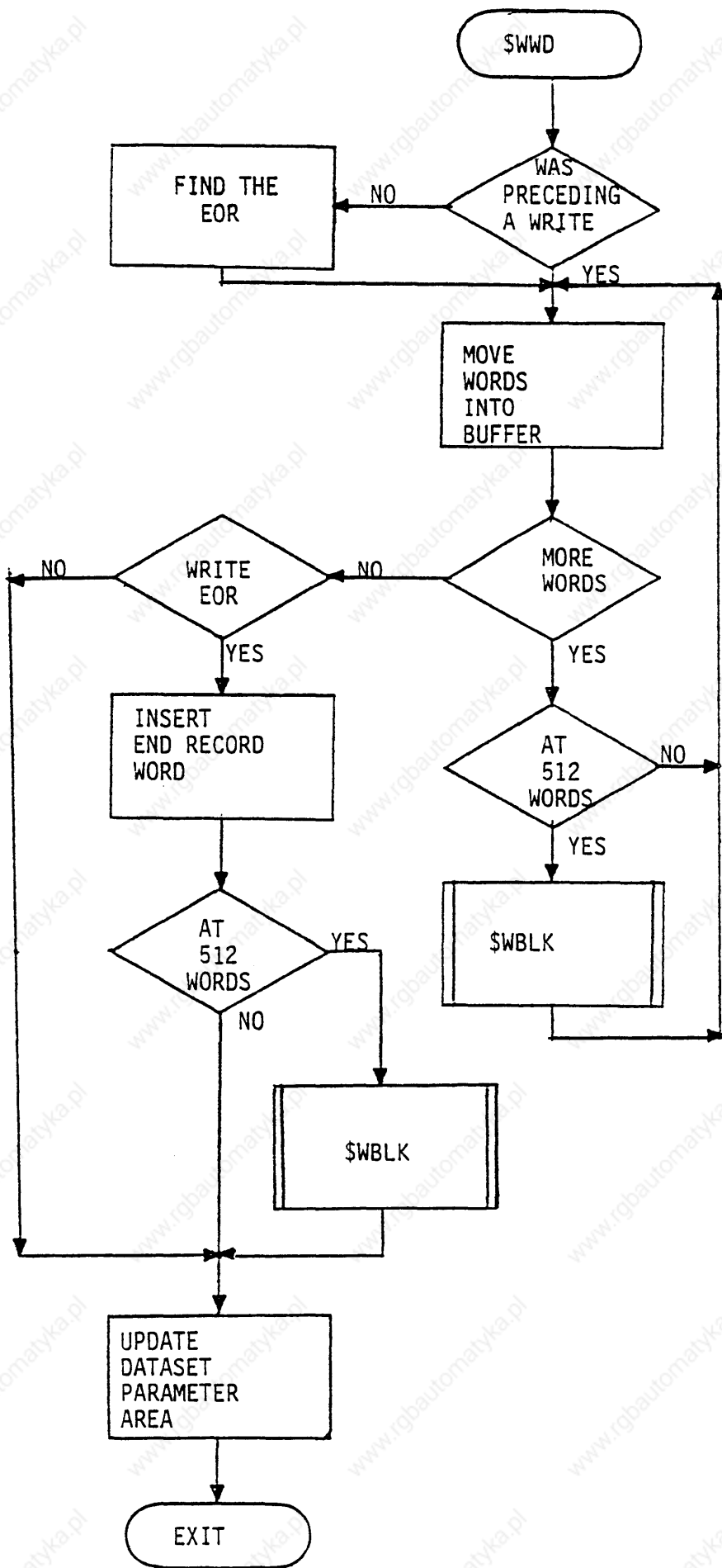


● \$WWDP/\$WWD R WRITES THE NUMBER OF WORDS SPECIFIED INTO THE I/O BUFFER. \$WWD IS CALLED VIA A RETURN JUMP WITH THE CALLER PROVIDING THE FOLLOWING:

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





```

*      WRITE WORDS                               S. 16418
*      ENTRY:                                     S. 16419
*      A1 @FCB                                    S. 16420
*      A2 FWA                                     S. 16421
*      A3 COUNT                                  S. 16422
*      A6 DNT ADDRESS                             S. 16423
*      A7 JXT ADDRESS, =0 IF NOT JOB RELATED     S. 16424
*      EXIT:                                       S. 16425
*      A1 @FCB                                    S. 16426
*      A2 FWA                                     S. 16427
*      A3 COUNT                                  S. 16428
*      A6 DNT ADDRESS                             S. 16429
*      A7 JXT ADDRESS                             S. 16430
*      MODIFIES:                                   S. 16431
*      A0,4,5  S0,1,2,3,4,5,6,7                 S. 16432
*      DPTH WORDS IN DSP                          S. 16433
*      WRITE WORDS, PARTIAL MODE                  S. 16434
$WWDP = *                                       S. 16435
      S6 0                                       S. 16436
      J   WW01                                    S. 16437
*      WRITE WORDS, RECORD MODE                  S. 16438
$WWDR = *                                       S. 16439
      S6 >1                                       S. 16440
WW01 = *                                       S. 16441
      S1 A6                                       DNT ADDRESS   S. 16442
      S2 A7                                       JXT ADDRESS   S. 16443
      S1 S1<D'24                                  S. 16444
      S1 S1!S2                                     S. 16445
      W@DPTH+5,A1 S1                               S. 16446
WW02 = *                                       S. 16447
      A7 B0                                       S. 16448
      S1 A7                                       B.ZA         S. 16449
      S2 A2                                       B.ZB         S. 16450
      S1 S1<D'24                                  S. 16451
      S1 S1!S2                                     S. 16452
      W@DPTH+2,A1 S1                               S. 16453
      R TDSP                                       CALCULATE A7 = DSP BASE S. 16454
      S4 S,A1                                       PW           S. 16455
      S3 2,A1                                       IN           S. 16456
      A0 S4                                       S. 16458
      S2 <30                                       S. 16459
      JAN WW03                                       NOT FIRST WRITE S. 16460
      S4 S3!S4&S2                                       SET POWA     S. 16461
      S2 <1                                       S. 16462
      A4 S3                                       S. 16463
      S3 S3+S2                                       IN + 1      S. 16464
      S2 164                                       S. 16465
      S2 S2<71                                       S. 16466
      A4 A4+A7                                       S. 16467
      0,A4 A0                                       CLEAR FIRST BCW S. 16468
      A4 A4-A7                                       S. 16469
      S4 S2!S4                                       INSERT RFW BITS S. 16470
WW03 = *                                       S. 16471
      S0 S4<4                                       S. 16472
      S2 >4                                       S. 16473
      JSM WW04                                       PRECEDING WRITE S. 16474
      R WW00                                       PROCESS WRITE AFTER READ S. 16475
      JAN WW22                                       ERROR        S. 16476

```

	R	TDSP	CALCULATE A7 = DSP BASE ADDRESS	S.16477
WW04	=	*		S.16478
	S1	1,A1	FIRST	0001.102
	S0	S4		S.16479
	S4	S0!S4&S2	CLEAR RFD BITS	S.16480
	S2	0		S.16481
	JSP	WW05	NOT BOR	S.16482
	S0	S0<2		S.16483
	S2	<17		S.16484
	JSP	WW05	NOT BOF	S.16485
	S0	S0<1		S.16486
	S2	<43		S.16487
	JSM	WW09	WRITE AFTER EOD	S.16488
WW05	=	*		S.16489
	S2	S2<30		S.16490
	A3	A2+A3	LWA + 1	S.16491
	S4	#S2&S4	CONDITIONAL CLEAR BFI, BRI	S.16492
	S1	S1-S3	F - I	S.16493
	S2	<11		S.16494
	A4	S3	I	S.16495
	S1	S2&S1		S.16496
	S1	S3+S1		S.16497
	A5	S1	NOWA	S.16498
	W@DPTM,A1	S4	T.ZA	S.16499
WW10	=	*		S.16500
	S7	W@DPBIO,A1		S.16501
	S6	A1		S.16502
	A1	A2-A3		S.16503
	A6	A4-A5		S.16504
	S1	A1		S.16505
	A0	A1-A6		S.16506
	S2	<6		S.16507
	A1	S6		S.16508
	JAP	WW11	COUNT FIRST	S.16509
WW11	S1	A6		S.16510
	=	*		S.16511
	S0	S1		S.16512
	S2	#S1&S2		S.16513
	A6	S2		S.16514
	A6	A6+1	FIRST SEGMENT LENGTH	S.16515
	A7	A4+A7		S.16516
	JSZ	WW15	ZERO LENGTH	S.16517
	S0	S7		S.16518
	ERRIF	S@DPBIO,NE,0		S.16519
	JSM	WWVMV	USE VECTOR MOVE IF BUFFERED I/O	S.16520
	S0	0,A2		S.16521
WW14	=	*		S.16522
	A6	A6-1		S.16523
	A2	A2+1		S.16524
	A0	A6-1	DONE?	S.16525
	A7	A7+1		S.16526
	A4	A4+1		S.16527
	-1,A7	S0		S.16528
	S0	0,A2		S.16529
	JAP	WW14	NO	S.16530
	A6	D'64	MOVE IN 64 WORD CHUNKS	S.16531
WW15	=	*		S.16532
	A0	A4-A5		S.16533
	JAM	WW16	NOT AT BCW	S.16534
	A7	A7-A4		S.16535
	R	\$WBLK	PROCESS BCW	S.16536
	JAN	WW22	ERROR	S.16537
	J	WW10		S.16538

	A0	A2-A3		S. 16540
	S0	0, A2		S. 16541
	JAM	WW14	NOT AT EOC	S. 16542
	A7	A7-A4		S. 16543
	S0	W@DPTM, A1	T. ZA	S. 16544
*	END OF COUNT			S. 16545
	S4	W@DPTM, A1	PW (T. ZA)	S. 16546
	JSP	WW21	PARTIAL MODE	S. 16547
*	RECORD MODE			S. 16548
	S1	W@DPTM, A1	T. ZA	S. 16549
	S2	<43		S. 16550
	S1	S1>17		S. 16551
	S2	S2<11		S. 16552
	A6	S4	@ PCW	S. 16553
	S1	S2&S1	BFI, BRI	S. 16554
	S2	>4		S. 16555
	S1	S4!S1&S2	INSERT MODE BITS	S. 16556
	A7	A7+A4		S. 16557
	0, A7	S1	STORE RCW	S. 16558
	A7	A7-A4		S. 16559
	A7	A7+A6		S. 16560
	S6	0, A7	PCW	S. 16561
	A2	A6+1		S. 16562
	S1	A4		S. 16563
	A2	A4-A2	FWI	S. 16564
	S2	<30		S. 16565
	S4	S1!S4&S2	UPDATE PCWA	S. 16566
	A4	A4+1		S. 16567
	S1	A2		S. 16568
	A0	A4-A5		S. 16569
	S6	S6!S1	INSERT FWI	S. 16570
	0, A7	S6		S. 16571
	A7	A7-A6		S. 16572
	JAM	WW21	NOT AT BCW	S. 16573
	R	WB01	WRITE BLOCK	S. 16574
	JAN	WW22	ERROR	S. 16575
*	PARTIAL MODE			S. 16576
WW21	=	*		S. 16577
	S1	A4		S. 16578
	S, A1	S4	UPDATE PW	S. 16579
	S2	<30		S. 16580
	S3	S1!S3&S2	INSERT NWA	S. 16581
	S2	<60		S. 16582
	S3	S2&S3	CLEAR BP	S. 16583
	Z, A1	S3	UPDATE IN	S. 16584
	A2	W@DPTM+2, A1	B. ZB	S. 16585
	A3	A3-A2	COUNT	S. 16586
	A0	0	COMPLETE FLAG	S. 16587
	J	WW22		S002. 1447
WW39	=	*		S. 16592
	S1	W@DPERR, A1		S. 16593
	S3	>1		S. 16594
	S3	S3>S@DPER!	WRITE PAST EOD	S. 16595
	S1	S1!S3	SET ERROR FLAG	S. 16596
	W@DPERR, A1	S1		S. 16597
WW22	=	*		S002. 1448
	S1	W@DPTM+2, A1		S002. 1449
	S1	S1>D'24		S002. 1450
*				S. 16598
*	S1	43/, 24/B0		S. 16599
*	A1	DSP ADDRESS		S. 16600
*				S. 16601
WW99	=	*		S. 16602
	A7	S1		S. 16603
	B0	A7		S. 16604
	S1	W@DPTM+5, A1		S. 16605
	A7	S1	JXT ADDRESS	S. 16606
	S1	S1>D'24		S. 16607
	A6	S1	DNT ADDRESS	S. 16608
	J	B0		S. 16609

```

* WJVMV VECTOR MOVE FOR WRITE RECORD S.16611
*
* A2 SOURCE ADDRESS S.16612
* A1 OUT S.16613
* A7 DESTINATION ADDRESS S.16614
* A6 SHORT VECTOR LENGTH (NOT ZERO) S.16615
* S1 NEGATIVE NUMBER OF WORDS TO BE MOVED S.16616
WJVMV S2 A3 SAVE A3 S.16617
A0 BIOV0SV V0 SAVE AREA FOR BUFFERED I/O S.16618
A3 ZS0 D'64 S.16619
VL A3 S.16620
,A0,1 V0 SAVE V0 S.16621
S.16622
A3 S1 S.16623
A3 -A3 S.16624
A4 A4+A3 INCREMENT OUT S.16625
S.16626
WJVMV1 A0 A2 SOURCE ADDRESS S.16627
VL A6 S.16628
V0 ,A0,1 S.16629
A0 A7 DESTINATION ADDRESS S.16630
A3 A3-A6 WORDS LEFT TO MOVE S.16631
A2 A2+A6 S.16632
A7 A7+A6 S.16633
A6 ZS0 D'64 S.16634
,A0,1 V0 S.16635
A0 -A3 S.16636
JAM WJVMV1 LOOP UNTIL ALL WORDS MOVED S.16637
S.16638
VL A5 S.16639
A0 BIOV0SV S.16640
V0 ,A0,1 RESTORE V0 S.16641
A3 S2 RESTORE A3 S.16642
J WJ15 S.16643
S.16644
S.16645
S.16646

```

```

**
* NAME TDSP S.16647
* CALCULATE DSP POINTER BASE S.16648
* = 0 IF JXT ADDRESS =0 S.16649
* = JTA ADDRESS IF DNJTF =1 S.16650
* = USER BA IF DNJTF =0 S.16651
* ENTRY TDSP: S.16652
* A1 DSP ADDRESS S003.737
* S003.738
* TDSP1: S003.739
* S7 16/0,24/DNT ADDRESS,24/JXT ADDRESS S003.740
* S003.741
* S003.742
* EXIT A1 DSP ADDRESS (TDSP ENTRY ONLY) S003.743
* A7 BASE OF DSP POINTERS S.16656
* S.16657
* REGISTERS MODIFIED - S.16658
* A0,A7 S0,S7 S.16659
* S.16660
* S.16661
TDSP S7 W@DPTH+5,A1 S.16662
TDSP1 = * ENTRY W/ S7 = JXT ADDRESS S.16663
A0 S7 JXT ADDRESS S.16664
A7 0 S.16665
JAZ TDSP9 NOT JOB RELATED S.16666
A7 S7 JXT ADDRESS S.16667
A7 W@JXTA,A7 JTA ADDRESS S.16668
S7 S7>D'24 S.16669
TDSPA,0 A6 SAVE A6 S.16670
A6 S7 DNT ADDRESS S.16671
S0 W@DNJTF,A6 S.16672
S0 S0<S@DNJTF S.16673
A6 L@JTA S.16674
JSM TDSP3 IN JTA S.16675
A7 A5+A7 USER BA S.16676
TDSP3 A6 TDSPA,0 S.16677
TDSP9 J B0 RETURN S.16678
TDSPA 0 S.16679

```



## CIRCULAR I/O

- PERFORMS PHYSICAL I/O ON A DATASET  
ACCESSIBLE TO TASKS THROUGH TIO AND DIRECT CALLS.

CIO ROUTINES ARE:

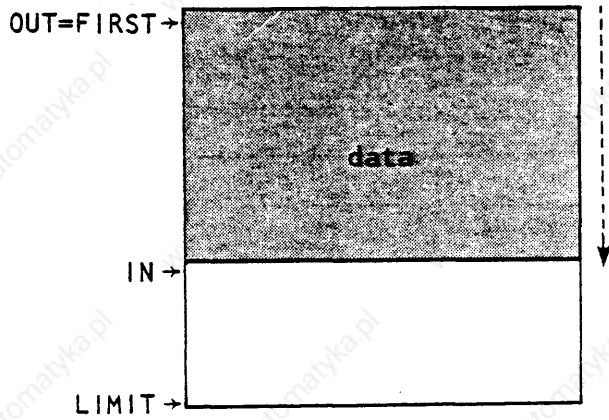
RDCS-READ CIRCULAR REQUEST

WDCS-WRITE CIRCULAR REQUEST

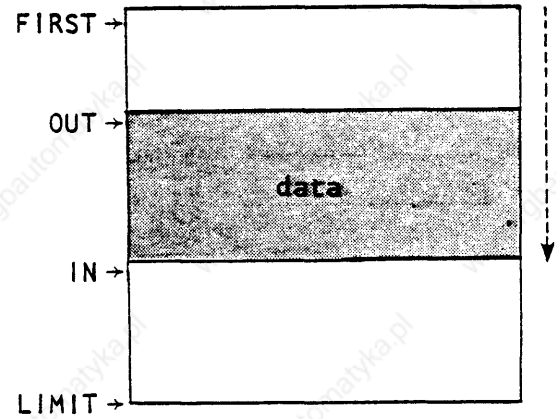
- TASKS CALL CIO BY PLACING REQUIRED PARAMETERS IN 'A' REGISTERS AND EXECUTING A RETURN JUMP TO THE ROUTINE.
- CIO READS/WITES 512 WORD BLOCKS. THE CALLER HAS THE RESPONSIBILITY OF MAINTAINING THE BUFFER IN/OUT POINTER IN THE DSP. AS SHOWN IN THE PREVIOUS \$WWD FLOW DIAGRAM.
- THE CALLER SENSES COMPLETION 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.



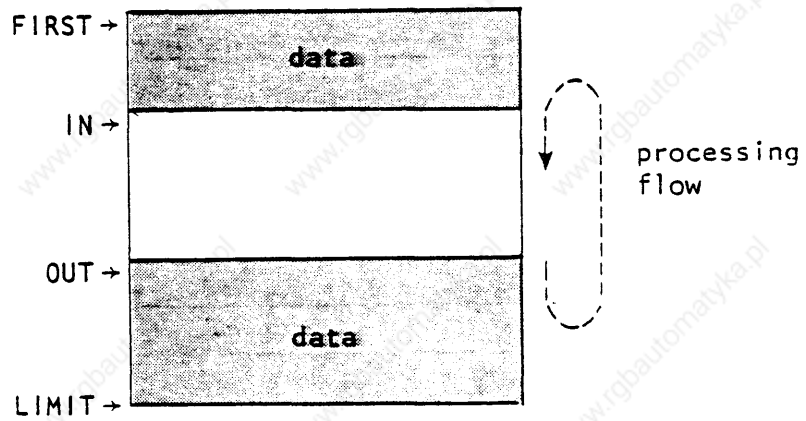




A. Filling the buffer



B. Emptying the buffer



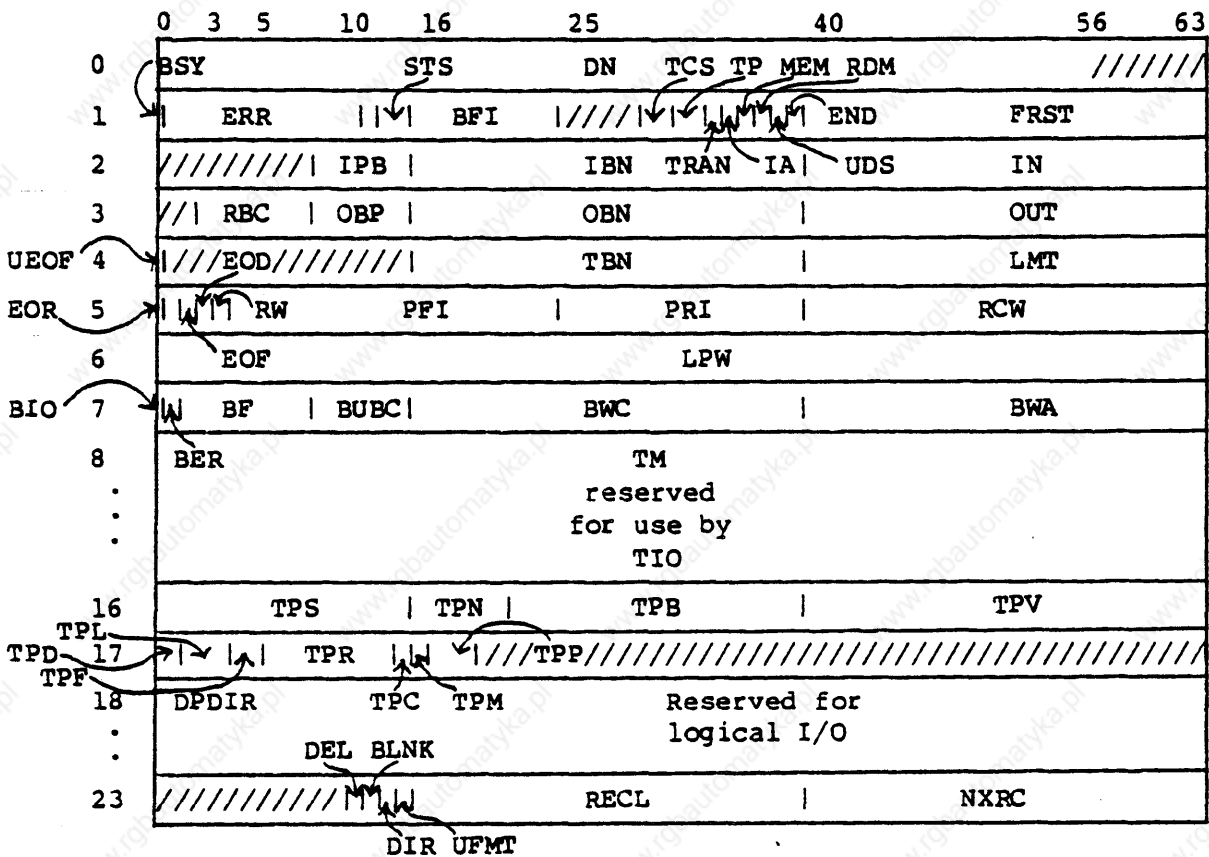
C. Concurrently filling and emptying the buffer

Physical I/O



DATASET PARAMETER AREA - DSP

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.



TAPE I/O

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

```

*****
**
*
*NAME      MEMAL
*
*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 AVAILABE
*          A7 = ADDRESS OF FIRST USEABLE WORD ALLOCATED—MEANINGFUL
*              ONLY IF A6 = 0
*
*****
MISTAKE1 =      1
MISTAKE2 =      2
MISTAKE3 =      3
MEMAL      =      *
          A5      POOLTBL          BASE ADDRESS OF POOL TABLE
          GET,S5  S6&S7,PTMAX,A5  MAX POOL NUMBER
          A4      S5
          A0      A4-A6
          JAM     MEMER1          JUMP IF POOL NUMBER GREATER THAN MAX
          A5      A5+A6          ADDRESS OF POOL WORD
          S0      0,A5
          JSZ    MEMER1
          A0      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+A4          TOTAL SIZE REQUIRED
*
*          VERIFY THAT THE REQUESTED SIZE IS NOT LARGER THAN THE POOL
*
          GET,S1  S6&S7,PTSIZE,A5
          A4      S1
          A0      A4-A7
          JAM     MEMER2
*
          GET,S1  S6&S7,PTBASE,A5
          A4      S1          BASE ADDRESS OF MEMORY POOL
*
*          LOCK OUT INTERRUPTS UNTIL THE SPACE HAS BEEN ALLOCATED
*
          S1      1
          STRLK,A0 S1
MEMAL10   =      *
          GET,S1  S6&S7,MPID,A4  VALIDATE THE HEADER WORD
          S2      IDWRD,A6
          S0      S1-S2
          EPRSN
          GET,S0  S6&S7,MPST,A4
          JSN    MEMAL50          JUMP IF AREA IN USE
          GET,S1  S6&S7,MPSIZE,A4 SIZE OF AVAILABLE AREA
          A1      S1
          A0      A1-A7
          JAM     MEMAL30          REQUEST SIZE UNEQUAL TO AVAILABLE

```

A0255.933  
A0255.934  
A0255.935  
A0255.936  
A0255.937  
A0255.938  
A0255.939  
A0255.940  
A0255.941  
A0255.942  
A0255.943  
A0255.944  
A0255.945  
A0255.946  
A0255.947  
A0255.948  
A0255.949  
A0255.950  
A0255.951  
A0255.952  
A0255.953  
A0255.954  
A0255.955  
A0255.956  
A0255.957  
A0255.958  
A0255.959  
A0255.960  
A0255.961  
A0255.962  
A0255.963  
A0255.964  
A0255.965  
A0255.966  
A0255.967  
A0255.968  
A0255.969  
A0255.970  
A0255.971  
A0255.972  
A0255.973  
A0255.974  
A0255.975  
A0255.976  
A0255.977  
A0255.978  
A0255.979  
A0255.980  
A0255.981  
A0255.982  
A0255.983  
A0255.984  
A0255.985  
A0255.986  
A0255.987  
A0255.988  
A0255.989  
A0255.990  
A0255.991  
A0255.992  
A0255.993  
A0255.994  
A0255.995  
A0255.996  
A0255.997  
A0255.998  
A0255.999

```

*
* AT THIS POINT THE SIZE OF THE REQUEST IS EQUAL TO THE
* AVAILIABLE AREA
*
S1 1
PUT,S1 S6&S7,MPST,A4 SET IN USE INDICATOR
A2 A4+A1
A2 A2-1 ADDRESS OF TRAILER WORD
PUT,S1 S6&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
*
A6 A7
MEMAL20 = *
0,A6 S1
A6 A6+1
A0 A6-A2
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 = *
JAM MEMAL50 AVAILABLE IS LESS THAN REQUEST
*
* REPLACE THE EXISTING HEADER WORD AND CREATE A NEW TRAILER
*
A2 A4+A7
A2 A2-1 TRAILER ADDRESS
S1 0
0,A2 S1 CLEAR OUT ANY GARBAGE
S1 A7
PUT,S1 S6&S7,MPST,A4
PUT,S1 S6&S7,MPST,A2 SIZE OF ALLOCATED AREA
S1 1
PUT,S1 S6&S7,MPST,A4
PUT,S1 S6&S7,MPST,A2 IN USE INDICATOR
S2 IDWRD,A6
PUT,S2 S6&S7,MPID,A4
PUT,S2 S6&S7,MPID,A2 POOL ID
*
* CREATE A NEW HEADER AND TRAILER FOR THAT SPACE WHICH IS
* STILL AVAILBLE.
*
A2 A2+1 ADDRESS OF NEW HEADER
A1 A1-A7 SIZE OF NEW AREA
A6 A2+A1
A6 A6-1 ADDRESS OF NEW TRAILER
S6 0
0,A2 S6 CLEAR NEW HEADER
S1 A1
PUT,S1 S6&S7,MPST,A2
PUT,S1 S6&S7,MPST,A6
S1 0
PUT,S1 S6&S7,MPST,A2
PUT,S1 S6&S7,MPST,A6
PUT,S2 S6&S7,MPID,A2
PUT,S2 S6&S7,MPID,A6
UNLOCK

```



* * *	CLEAR THE ALLOCATED AREA	A0255.1071 A0255.1072 A0255.1073 A0255.1074 A0255.1075 A0255.1076 A0255.1077 A0255.1078 A0255.1079 A0255.1080 A0255.1081 A0255.1082 A0255.1083 A0255.1084 A0255.1085 A0255.1086 A0255.1087 A0255.1088 A0255.1089 A0255.1090 A0255.1091 A0255.1092 A0255.1093 A0255.1094 A0255.1095 A0255.1096 A0255.1097 C0763.1 A0255.1098 A0255.1099 A0255.1100 A0255.1101 A0255.1102 A0255.1103 A0255.1104 A0255.1105
MEMAL40	A2 A2-1 A7 A4+1 A4 A4+1 = * 0, A4 S1 A4 A4+1 A0 A2-A4 JAN MEMAL40 A6 0 J B00 RETURN GOOD STATUS	
MEMAL50	= * GET, S1 S6&S7, MPSIZE, A4 SIZE OF CURRENT AREA A2 S1 A4 A4+A2 ADDRESS NEXT HEADER WORD GET, S1 S6&S7, PTBASE, A5 POOL BASE ADDRESS GET, S2 S6&S7, PTSIZE, A5 POOL SIZE S1 S1+S2 END ADDRESS OF POOL A1 S1 A0 A1-A4 JAN MEMAL10 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	
MEMER1:	A6 MISTAKE3 J B00 = * 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.

EJECT

```

*****
**
**
**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
**          A7 = ADDRESS OF THE MEMORY TO BE RELEASED
**
**EXIT     A6 = STATUS
**          0 = GOOD RETURN
**          1 = INVALID HEADER WORD
**          2 = AREA NOT CURRENTLY ALLOCATED
**          3 = INVALID POOL NUMBER
**          A7 = ADDRESS OF AREA RELEASED--MEANINGFUL ONLY IF GOOD STATUS
**
**
*****
DEERR1    =      1
DEERR2    =      2
DEERR3    =      3
MEMDE     =      *
A1        A7-1
GET, S1   S6&S7, MPID, A1
S2        IDWRD, A6
S0        S2-S1
JSN       DEER1          JUMP IF INVALID HEADER WORD
GET, S0   S6&S7, MPST, A1
JSZ       DEER2          JUMP IF PREVIOUSLY RELEASED
A5        POOLTBL       BASE ADDRESS OF POOL TABLE
GET, S1   S5&S6, PTMAX, A5 MAX VALID POOL NUMBER
S2        A6
S0        S1-S2
JSM       DEER3          JUMP IF INVALID POOL
A5        A5+A6         ADDRESS OF POOL WORD
S0        0, A5
JSZ       DEER3          JUMP IF INVALID POOL
GET, S1   S6&S7, PTBASE, A5
GET, S2   S6&S7, PTSIZE, A5 POOL SIZE
S2        S2+S1         END ADDRESS OF POOL
S3        A1            ADDRESS OF AREA TO RELEASE
S0        S3-S1
JSM       DEER3          JUMP IF ADDRESS OUTSIDE POOL RANGE
S0        S2-S3
JSM       DEER3          JUMP IF ADDRESS OUTSIDE POOL RANGE
**
**          THE VALIDATION IS COMPLETE.  NOW ACTUALLY RELEASE THE SPACE
**
S6        1
STPLK, A0 S6            LOCK OUT INTERRUPTS
GET, S4   S6&S7, MPSIZE, A1 SIZE OF AREA
S5        S3+S4
A2        S5
A2        A2-1          TRAILER ADDRESS
S6        0
PUT, S6   S0&S7, MPST, A1 MARK IT AVAILABLE
PUT, S6   S0&S7, MPST, A2
S0        S5-S2
JSP       MEMDE10       JUMP IF LAST AREA IN POOL
A4        S5            ADDRESS OF NEXT HEADER
GET, S0   S6&S7, MPST, A4
JSN       MEMDE10       JUMP IF NEXT AREA NOT AVIALABLE

```

A0255.1106  
A0255.1107  
A0255.1108  
A0255.1109  
A0255.1110  
A0255.1111  
A0255.1112  
A0255.1113  
A0255.1114  
A0255.1115  
A0255.1116  
A0255.1117  
A0255.1118  
A0255.1119  
A0255.1120  
A0255.1121  
A0255.1122  
A0255.1123  
A0255.1124  
A0255.1125  
A0255.1126  
A0255.1127  
A0255.1128  
A0255.1129  
A0255.1130  
A0255.1131  
A0255.1132  
A0255.1133  
A0255.1134  
A0255.1135  
A0255.1136  
A0255.1137  
A0255.1138  
A0255.1139  
A0255.1140  
A0255.1141  
A0255.1142  
A0255.1143  
A0255.1144  
A0255.1145  
A0255.1146  
A0255.1147  
A0255.1148  
A0255.1149  
A0255.1150  
A0255.1151  
A0255.1152  
A0255.1153  
A0255.1154  
A0255.1155  
A0255.1156  
A0255.1157  
A0255.1158  
A0255.1159  
A0255.1160  
A0255.1161  
A0255.1162  
A0255.1163  
A0255.1164  
A0255.1165  
A0255.1166  
A0255.1167  
A0255.1168  
A0255.1169  
A0255.1170

\*  
\*  
\*  
\*

THE FOLLOWING MEMORY AREA IS AVAILABLE. COMBINE THE TWO INTO ONE LARGE AREA

GET, S5 S6&S7, MPST, 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 PRECEDING TRAILER  
GET, S0 S6&S7, MPST, A4  
JSN MEMDE20 JUMP IF PRECEDING AREA IN USE

\*  
\*  
\*  
\*

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

GET, S5 S6&S7, MPST, A4 SIZE OF PRECEDING AREA  
S4 S4+S5 COMBINED SIZE  
A4 S5  
A1 A1-A4 BASE ADDRESS OF LARGEST AREA

\*  
\*  
\*

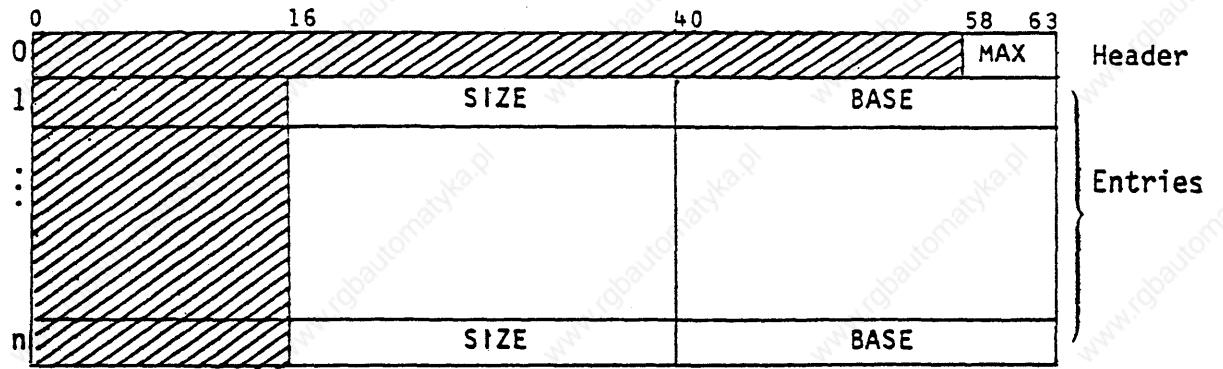
CREATE NEW HEADER AND TRAILER WORDS

MEMDE20 = \*  
PUT, S4 S6&S7, MPST, A1  
PUT, S4 S6&S7, MPST, A2  
A6 0 GOOD STATUS  
UNLOCK  
J B00  
DEER1 = \*  
A6 DEERR1  
J B00  
DEER2 = \*  
A6 DEERR2  
J B00  
DEER3 = \*  
A6 DEERR3  
J B00

A0255.1171  
A0255.1172  
A0255.1173  
A0255. 4  
A0255.1175  
A0255.1176  
A0255.1177  
A0255.1178  
A0255.1179  
A0255.1180  
A0255.1181  
A0255.1182  
A0255.1183  
A0255.1184  
A0255.1185  
A0255.1186  
A0255.1187  
A0255.1188  
A0255.1189  
A0255.1190  
A0255.1191  
A0255.1192  
A0255.1193  
A0255.1194  
A0255.1195  
A0255.1196  
A0255.1197  
A0255.1198  
A0255.1199  
A0255.1200  
C0583.31  
A0255. 2  
A0255.1203  
A0255.1204  
A0255.1205  
A0255.1206  
A0255.1207  
A0255.1208  
A0255.1209  
A0255.1210  
A0255.1211

POOL TABLE - POOLTBL

The Pool Table is an STP-resident table used for memory pool management.



Pool Table

HEADER

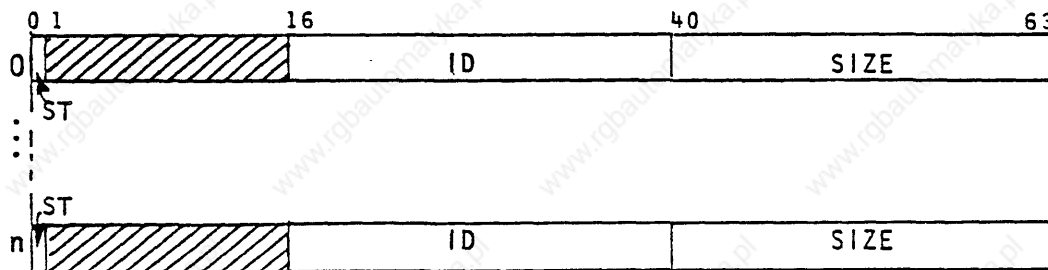
<u>Field</u>	<u>Word</u>	<u>Bits</u>	<u>Description</u>
PTMAX	0	58-63	Maximum valid memory pool number in system

ENTRY

<u>Field</u>	<u>Word</u>	<u>Bits</u>	<u>Description</u>
PTSIZE	0	16-39	Size of the memory pool
PTBASE	0	40-63	Base address of the memory pool

MEMORY POOL - POOLI...n

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



Memory Pool

<u>Field</u>	<u>Word</u>	<u>Bits</u>	<u>Description</u>
MPST	0,n,etc.	0	Status of the memory area: 0 Available 1 In use
MPID	0,n,etc.	16-39	Memory pool identification: 01010101 Pool 1 0x0x0x0x Pool x
MPSIZE	0,n,etc.	40-63	Size of the memory pool



## 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

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     S6&S7,CIFL,A7
           PUT,S1     S6&S7,CIBL,A7  JUST TO BE SAFE
           S1         A6
           PUT,S1     S6&S7,CICC,A7  SAVE CHAIN CONTROL ADDRESS
           GET,S1     S6&S7,CCTAIL,A6
           S2         A7
           PUT,S2     S6&S7,CCTAIL,A6  NEW TAIL ADDRESS
           PUT,S1     S6&S7,CIBL,A7  BACKWAR LINK = OLD TAIL
           S0         S1
           JSN       CHAIN10
*
*           THE CHAIN WAS EMPTY.  THIS IS THE ONLY ITEM ON QUEUE
*
           PUT,S2     S6&S7,CCHEAD,A6  HEAD = TAIL
           J         CHAIN20
CHAIN10    =          *
           A5         S1
           PUT,S2     S6&S7,CIFL,A5  FORWARD LINK OLD TAIL = NEW TAIL
CHAIN20    =          *
           J         B00

```

A0255.1212  
A0255.1213  
A0255.1214  
A0255.1215  
A0255.1216  
A0255.1217  
A0255.1218  
A0255.1219  
A0255.1220  
A0255.1221  
A0255.1222  
A0255.1223  
A0255.1224  
A0255.1225  
A0255.1226  
A0255.1227  
A0255.1228  
A0255.1229  
A0255.1230  
A0255.1231  
A0255.1232  
A0255.1233  
A0255.1234  
A0255.1235  
A0255.1236  
A0255.1237  
A0255.1238  
A0255.1239  
A0255.1240  
A0255.1241  
A0255.1242  
A0255.1243  
A0255.1244  
A0255.1245  
A0255.1246  
A0255.1247  
A0255.1248  
A0255.1249  
A0255.1250

```

*****
**
**NAME      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
            STPLK,0  S1          LOCK OUT INTERRUPTS
            GET,S2   S6&S7,CCHEAD,A6
            S0       S2
            S1       A7
            JSZ      CHAINF1
            SPUT,S1  S6&S7,CCHEAD,A6
            A5       S2
            PUT,S1   S6&S7,CIBL,A5
            J        CHAINF2
CHAINF1     =      *
            GET,S1   S6&S7,CCHEAD
            SET,S1   S6&S7,CCTAIL
            W@CCHEAD,A6 S6
CHAINF2     =      *
            S1       A6
            PUT,S1   S6&S7,CICC,A7  CHAIN CONTROL POINTER
            S6       W@CIFL,A7
            GET,S2   S6&S7,CIFL
            S1       0
            SET,S1   S6&S7,CIBL
            W@CIFL,A7 S6
            UNLOCK
            J        B0
            ERRIF   W@CIFL,NE,W@CIBL
            ERRIF   W@CCHEAD,NE,W@CCTAIL

```

B0492.124  
B0492.125  
B0492.126  
B0492.127  
B0492.128  
B0492.129  
B0492.130  
B0492.131  
B0492.132  
B0492.133  
B0492.134  
B0492.135  
B0492.136  
B0492.137  
B0492.138  
B0492.139  
B0492.140  
B0492.141  
B0492.142  
B0492.143  
B0492.144  
B0492.145  
B0492.146  
B0492.147  
B0492.148  
B0492.149  
B0492.150  
B0492.151  
B0492.152  
B0492.153  
B0492.154  
B0492.155  
B0492.156  
B0492.157  
B0492.158  
B0492.159  
B0492.160  
B0492.161  
B0492.162  
C0983.32  
B0492.164  
B0492.165  
B0492.166



- 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



```

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.
*
*KEYWORD   A7 = ADDRESS OF ITEM TO BE REMOVED
*
*EXIT      A0 = STATUS
*          0 = GOOD RETURN
*          1 = ITEM IS NOT ON A CHAIN
*          A7 = ADDRESS OF ITEM REMOVED
*
**
*****
UNCHER1 = 1
UNCHAIN = *
GET,S1  S6&S7,CICC,A7  CHAIN CONTROL WORD ADDRESS
S0      S1
A6      S1
A0      UNCHER1
JSZ     UNCH50          JUMP IF ITEM NOT ON A CHAIN
S1      1
STP_LK,A0 S1          LOCK OUT INTERRUPTS
GET,S1  S6&S7,CCHEAD,A6 HEAD OF CHAIN
S2      A7              ADDRESS OF ITEM TO REMOVE
GET,S3  S6&S7,CIFL,A7  FORWARD LINK OF ITEM
GET,S4  S6&S7,CIBL,A7  BACKWARD LINK OF ITEM
S0      S1-S2
JSN     UNCH10         JUMP IF ITEM NOT HEAD OF CHAIN
PUT,S3  S6&S7,CCHEAD,A6 ITEM FOWARD LINK = NEW HEAD
UNCH10 = *
GET,S1  S6&S7,CCTAIL,A6 TAIL OF CHIAN
S0      S1-S2
JSN     UNCH20         JUMP IF ITEM NOT TAIL OF CHIAN
PUT,S4  S6&S7,CCTAIL,A6 ITEM BACKWARD LINK = NEW TAIL
UNCH20 = *
S0      S3
JSZ     UNCH30         JUMP IF FOWAD LINK IS ZERO
A2      S3
PUT,S4  S6&S7,CIBL,A2  NEW BACKWARD LINK FOR FOLLOWING ITEM
UNCH30 = *
S0      S4
JSZ     UNCH40         JUMP IF BACKWARD LINK IS ZERO
A2      S4
PUT,S3  S6&S7,CIFL,A2  NEW FORWARD LINK FOR PRECEEDING ITEM
UNCH40 = *
UNLOCK
PUT,S1  S6&S7,CICC,A7  CLEAR CHAIN CONTROL
A0      0              GOOD STATUS
UNCH50 = *
J       B00

```

A0255.1251  
A0255.1252  
A0255.1253  
A0255.1254  
A0255.1255  
A0255.1256  
A0255.1257  
A0255.1258  
A0255.1259  
A0255.1260  
A0255.1261  
A0255.1262  
A0255.1263  
A0255.1264  
A0255.1265  
A0255.1266  
A0255.1267  
A0255.1268  
A0255.1269  
A0255.1270  
A0255.1271  
A0255.1272  
A0255.1273  
A0255.1274  
A0255.1275  
A0255.1276  
A0255.1277  
A0255.1278  
A0255.1279  
A0255.1280  
A0255.1281  
A0255.1282  
A0255.1283  
A0255.1284  
A0255.1285  
A0255.1286  
A0255.1287  
A0255.1288  
A0255.1289  
A0255.1290  
A0255.1291  
A0255.1292  
A0255.1293  
A0255.1294  
A0255.1295  
A0255.1296  
A0255.1297  
A0255.1298  
A0255.1299  
A0255.1300  
C0983.33  
A0255.1303  
C0983.34  
A0255.1305  
A0255.1306





## CHAIN CONTROL - CC

Intertask communication requires chain control words in the format defined.

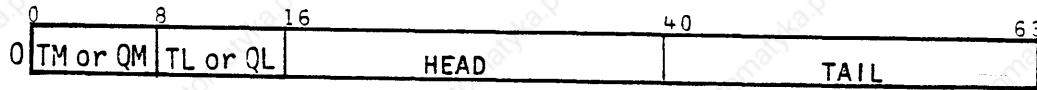
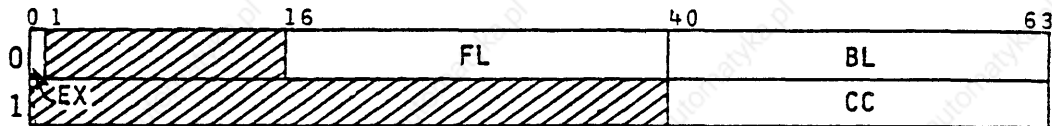


Figure 1.CC-1. Chain Control Word

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

## 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.



Chain Item

<u>Field</u>	<u>Word</u>	<u>Bits</u>	<u>Description</u>
CIEX	0	0	This bit, if set, indicates that the item is in execution.
CIFL	0	16-39	Forward link; address of next item on the chain
CIBL	0	40-63	Backward link; address of the preceding item on the chain
CICC	1	40-63	Address of the chain control word for this item

## 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

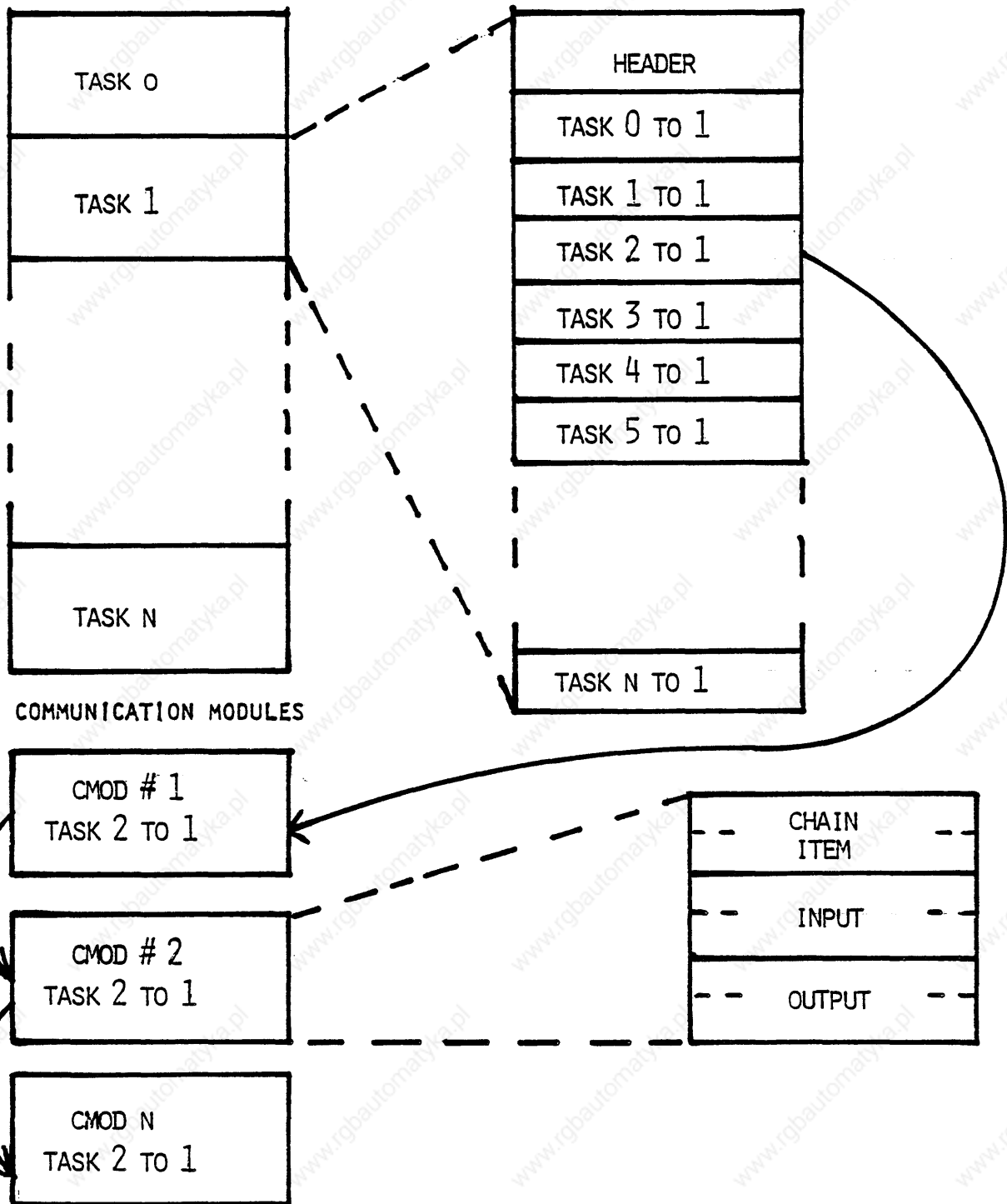
2 WORDS AS TASK OUTPUT REGISTERS

- 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





- A TASKS 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
*****
**
*
*NAME      PUTREQ
*
*PURPOSE  PUTREQ QUEUES A REQUEST FOR PROCESSING TO ANOTHER TASK.
*         ONCE THE REQUEST HAS BEEN CHAINED PUTREQ RETURNS TO THE
*         THE ORIGINATOR
*
*ENTRY    A1 = THROW AWAY INDICATOR
*         A1 = NEGATIVE THROW AWAY REQUEST INSTEAD OF SUSP
*         A1 = POSITIVE SUSPEND UNTIL REQUEST QUEUED
*
*         A2 = DESTINATION TASK ID
*
*         S1 = INPUT+0
*         S2 = INPUT+1
*
*EXIT
*
**
*****
PUTREQ    =      *
          A6      LE@SAVER
          A7      CTID,A0
          A6      ASXA7
          A7      SAVER
          A6      SAVER          ADDRESS OF SAVE AREA FOR ORIGIN TASK
          SAVEA1,A6 A1
          SAVEA2,A6 A2
          SAVEA3,A6 A3
          SAVEA4,A6 A4
          SAVEA5,A6 A5
          SAVEA6,A6 A6
          SAVEA7,A6 A7
          A3      A6
          S0      S1
          ERRSZ   ERROR IF INPUT&1 = 0
PUTREQ10 =      *
          A6      CNPOOL        MEMORY POOL NUMBER
          A7      CMSIZE        CMOD SIZE
          R       MEMAL         ALLOCATE A CMOD
          A5      SAVERT,A3     RESTORE B00 FOR TRACE ENTRIES
          B00     A5
          A0      A6
          JAZ     PUTREQ20      JUMP IF GOOD STATUS RETURNED
          A7      MISTAKE3
          A0      AS-A7
          ERRAN

```

A0255.379  
A0255.380  
A0255.381  
A0255.382  
A0255.383  
A0255.384  
A0255.385  
A0255.386  
A0255.387  
A0255.388  
A0255.389  
A0255.390  
A0255.391  
A0255.392  
A0255.393  
A0255.394  
A0255.395  
A0255.396  
A0255.397  
A0255.398  
A0255.399  
A0255.400  
A0255.401  
A0255.402  
A0255.403  
A0255.404  
A0255.405  
A0255.406  
A0255.407  
A0255.408  
A0255.409  
A0255.410  
A0255.411  
A0255.412  
A0255.413  
A0255.414  
A0255.415  
A0255.416  
A0255.417  
A0255.418  
A0255.419  
A0255.420  
A0255.421  
A0255.422  
A0255.423  
A0255.424  
B0492.7  
B0492.8  
A0255.425  
A0255.426  
A0255.427  
A0255.428  
A0255.429  
A0255.430  
A0255.431  
A0255.432  
A0255.433  
A0255.434  
A0255.435

```

*
* MEMAL RETURNED A THREE STATUS--THERE IS NO MEMORY AVAILABLE.
* SUSPEND AND TRY AGAIN LATER
*
A0      SAVEA1,A3
JAM     PUTREQS0      THROW AWAY THE REQUEST
S7      SUSP
EX
R        ERROR0
J        PUTREQ10
PUTREQ20 = *
S1      SAVE31,A3
S2      SAVE32,A3
CMIN0,A7 S1
CMIN1,A7 S2      PUT THE MESSAGE IN THE CMOO
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
A6      LE@CMCC
A6      ASKA6
A6      CMCC
A5      A5+A6      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      A6+A6
A4      CTID,A0
A6      A6+A4      CHAIN CONTROL ADDRESS
PUTREQ21 = *
A1      1
STPLK,A0 A1      LOCK OUT INTERRUPTS
GLT,S1  S6&S7,CCTL,A5
GET,S2  S6&S7,CCTM,A5
S0      S2
ERRSZ   ERROR IF QUEUE MAX IS ZERO
S0      S1-S2
JSZ     PUTREQ22   JUMP IF QUEUE AT MAX
S4      1
S1      S4+S1
GET,S2  S6&S7,CCQL,A6
GET,S3  S6&S7,CCQM,A6
S0      S3
ERRSZ   ERROR IF QUEUE AT MAX
S0      S2-S3
JSZ     PUTREQ22   JUMP IF QUEUE AT MAX
S2      S2+S4

*
* UPDATE THE QUEUE COUNTS
PUT,S1  S6&S7,CCTL,A5
PUT,S2  S6&S7,CCQL,A6
J        PUTREQ25

*
* THE QUEUE IS AT ITS MAX. EITHER SUSPEND OR THROW THE REQUEST
* AWAY
*
PUTREQ22 = *
A1      0
STPLK,A0 A1      RELEASE INTERRUPT LOCKOUT
A0      SAVEA1,A3
JAM     PUTREQ23   JUMP TO THOW THE REQUEST AWAY
S7      SUSP
EX
R        ERROR0
J        PUTREQ21

```





- 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        } request

```

EJECT
*****
**
*
*NAME      GETREQ
*
*PURPOSE  GETREQ SEARCHES FOR AN OUTSTANDING REQUEST TO THE TASK.
*          THE HIGHEST PRIORITY REQUEST WILL BE RETURNED TO THE
*          ORIGINATING TASK. PRIORITY CORRESPONDS TO TASK ID
*
*ENTRY
*EXIT     A0 = FIND/NO FIND INDICATOR
*          0 = NO FIND
*          1 = FIND
*          A2 = TASK ID OF THE REQUEST LOCATED
*          S1 = INPUT+0
*          S2 = INPUT+1
**
*****
GETREQ    =      *
          A6      LE@SAVER
          A7      CTID,A0
          A6      ASKA7
          A7      SAVER
          A6      AS+A7          ADDRESS OF SAVE AREA FOR ORIGIN TASK
*
*          SAVE THOSE REGISTERS NOT ALREADY DESTROYED
*
          SAVEA1,A6 A1
          SAVEA2,A6 A2
          SAVEA3,A6 A3
          SAVEA4,A6 A4
          SAVEA5,A6 A5
          SAVES0,A6 S0
          SAVES1,A6 S1
          SAVES2,A6 S2
          SAVES3,A6 S3
          SAVES4,A6 S4
          SAVES5,A6 S5
          SAVES6,A6 S6
          SAVES7,A6 S7
          A7      B00
          SAVERT,A6 A7
          S6      0
          SAVEA6,A6 S6
          SAVEA7,A6 S6
          SAVEA0,A6 S6          ASSUME NO FIND WILL OCCUR
          A3      A6
*
*
*          CALCULATE THE ADDRESS OF CHAIN CONTROL HEADER
*          CHAIN CONTROL HEADER = CURRENT TASK ID * CHAIN CONTROL LENGTH
*          OF ENTRY + BASE OF CHAIN CONTROL
*
          A1      CTID,A0
          A2      LE@CMCC
          A1      A1KA2
          A2      CMCC
          A1      A1+A2          CHAIN CONTROL HEADER ADDRESS
          GET,S0  S6&S7,OCTL,A1
          JEZ     GETREQ00      JUMP IF THERE IS NOTHING ON QUEUE

```

```

*
* SEARCH EACH CHAIN CONTROL WORD FOR AN OUTSTANDING REQUEST
*
A2      LH@CMCC
A1      A1+A2
A2      0                WILL CONTAIN TASK ID OF FIND ON EXIT
A4      MAXTN
GETREQ10 = *
GET, S0  S6&S7, CQCL, 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  S6&S7, CHEAD, A1
GET, S2  S6&S7, COTAIL, A1
GETREQ22 = *
A5      S1              CMOD ADDRESS
GET, S0  S6&S7, 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  S6&S7, CIFL, A5 ADDRESS OF NEXT CMOD
J       GETREQ22
*
* A REQUEST TO PROCESS HAS BEEN LOCATED
*
GETREQ25 = *
SAVEA2, A3 A2          SAVE TASK ID OF REQUEST
S1      1
SAVEA0, A3 S1          SET FIND INDICATOR
PUT, S1  S6&S7, CIEX, A5 SET EXECUTE INDICATOR
S1      CMIN0, A5
S2      CMIN1, A5
S0      S1
EPRSZ                   ERROR IF INPUT&1 = 0
SAVE S1, A3 S1
SAVE S2, A3 S2
*
* RESTORE REGISTERS AND EXIT.
*
GETREQ30 = *
A6      A3
A0      SAVEA0, A6
A1      SAVEA1, A6
A2      SAVEA2, A6
A3      SAVEA3, A6
A4      SAVEA4, A6
A5      SAVEA5, A6
S0      SAVE S0, A6
S1      SAVE S1, A6
S2      SAVE S2, A6
S3      SAVE S3, A6
S4      SAVE S4, A6
S5      SAVE S5, A6
S6      SAVE S6, A6
S7      SAVE S7, A6
A7      SAVERT, A6
B00     A7
A7      SAVEA7, A6
A6      SAVEA6, A6
J       B00

```

```

A0255. 612
A0255. 613
A0255. 614
A0255. 615
A0255. 616
A0255. 617
A0255. 618
A0255. 619
A0255. 620
A0255. 621
A0255. 622
A0255. 623
A0255. 624
A0255. 625
A0255. 626
A0255. 627
A0255. 628
A0255. 629
A0255. 630
A0255. 631
A0255. 632
A0255. 633
A0255. 634
A0255. 635
A0255. 636
A0255. 637
A0255. 638
A0255. 639
A0255. 640
A0255. 641
A0255. 642
A0255. 643
A0255. 644
A0255. 645
A0255. 646
A0255. 647
A0255. 648
A0255. 649
A0255. 650
A0255. 651
A0255. 652
A0255. 653
A0255. 654
B0492. 9
B0492. 10
A0255. 655
A0255. 656
A0255. 657
A0255. 658
A0255. 659
A0255. 660
A0255. 661
A0255. 662
A0255. 663
A0255. 664
A0255. 665
A0255. 666
A0255. 667
A0255. 668
A0255. 669
A0255. 670
A0255. 671
A0255. 672
A0255. 673
A0255. 674
A0255. 675
A0255. 676
A0255. 677
A0255. 678
A0255. 679
A0255. 680

```

- 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

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+0
* S2 = OUTPUT+1
*
* EXIT
*
**
*****
PUTREPLY = *
          A6 LE@SAVER
          A7 CTID,A0 ORIGINATING TASK ID
          A6 ASXA7
          A7 SAVER
          A6 AS+A7 ADDRESS OF SAVE AREA FOR ORIGIN TASK
*
* SAVE THE REGISTERS CLEARING THOSE ALREADY DESTROYED.
*
          SAVEA0,A6 A0
          SAVEA1,A6 A1
          SAVEA2,A6 A2
          SAVEA3,A6 A3
          SAVEA4,A6 A4
          SAVEA5,A6 A5
          SAVES0,A6 S0
          SAVES1,A6 S1
          SAVES2,A6 S2
          SAVES3,A6 S3
          SAVES4,A6 S4
          SAVES5,A6 S5
          SAVES6,A6 S6
          SAVES7,A6 S7
          A7 B00
          SAVERT,A6 A7
          S6 0
          SAVEA7,A6 S6
          SAVEA6,A6 S6
          A3 A6
          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 * CURRENT TASK ID)
* CHAIN CONTROL WORD = CHAIN CONTROL HEADER + LENGTH OF HEADER
* + RECEIVING TASK ID
*
          A1 CTID,A0
          A6 LE@CMOC
          A6 ASXA1
          A1 CMOC
          A6 AS+A1 CHAIN CONTROL HEADER
          A7 LH@CMOC
          A7 A7+A6
          A7 A2+A7 CHAIN CONTROL WORD
          GET,S0 S6&S7,CCOHL,A7
          ERRSZ
          GET,S5 S6&S7,CCHEAD,A7
          GET,S4 S6&S7,CCTAIL,A7

```

A0255.681  
A0255.682  
A0255.683  
A0255.684  
A0255.685  
A0255.686  
A0255.687  
A0255.688  
A0255.689  
A0255.690  
A0255.691  
A0255.692  
A0255.693  
A0255.694  
A0255.695  
A0255.696  
A0255.697  
A0255.698  
A0255.699  
A0255.700  
A0255.701  
A0255.702  
A0255.703  
A0255.704  
A0255.705  
A0255.706  
A0255.707  
A0255.708  
A0255.709  
A0255.710  
A0255.711  
A0255.712  
A0255.  
A0255.714  
A0255.715  
A0255.716  
A0255.717  
A0255.718  
A0255.719  
A0255.720  
A0255.721  
A0255.722  
A0255.723  
A0255.724  
A0255.725  
A0255.726  
B0492.11  
B0492.12  
A0255.727  
A0255.728  
A0255.729  
A0255.730  
A0255.731  
A0255.732  
A0255.733  
A0255.734  
A0255.735  
A0255.736  
A0255.737  
A0255.738  
A0255.739  
A0255.741  
A0255.742  
A0255.743  
A0255.744  
A0255.745

```

PUTREP10 = *
AS SS CMOD ADDRESS
SC CMOUT1,A5
JSZ PUTREP20 JUMP IF CMOD HAS NO REPLY

A REPLY EXISTS IN THIS CMOD. GO TO THE NEXT CMOD IF THERE IS
ONE
SO S4-SS
ERRSZ
GET,SS SB&S7,CIFL,A5 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,A5 S1
CMOUT1,A5 S2

DECREMENT THE NUMBER OF ITEMS ON QUEUE
GET,S1 SB&S7,CCOL,A7
S4 1
S1 S1-S4
PUT,S1 SB&S7,CCOL,A7
GET,S1 SB&S7,CCTL,A6
S1 S1-S4
PUT,S1 SB&S7,CCOL,A6
A1 0
STPLK,A0 A1

READY THE RECEIVING TASK
S1 CMOUT0,A5
POST 12,S1,S2
S6 A2
S7 RTSK
EX R
R ERROR0

RESTORE THE REGISTERS AND EXIT
A6 A3
A0 SAVEA0,A6
A1 SAVEA1,A6
A2 SAVEA2,A6
A3 SAVEA3,A6
A4 SAVEA4,A6
A5 SAVEA5,A6
S0 SAVE$0,A6
S1 SAVE$1,A6
S2 SAVE$2,A6
S3 SAVE$3,A6
S4 SAVE$4,A6
S5 SAVE$5,A6
S6 SAVE$6,A6
S7 SAVE$7,A6
A7 SAVERT,A6
B00 A7
A7 SAVEA7,A6
A6 SAVEA6,A6
J B00

```

```

A0255.746
A0755.747
A0255.748
A0255.749
A0255.750
A0255.751
A0255.752
A0255.753
A0255.754
A0255.755
A0255.756
A0255.757
A0255.758
A0255.759
A0255.760
A0255.761
A0255.762
A0255.763
A0255.764
A0255.765
A0255.766
A0255.767
A0255.768
A0255.769
A0255.770
A0255.771
A0255.772
A0255.773
A0255.774
A0255.775
A0255.776
A0255.777
A0255.778
A0255.779
A0255.780
A0255.781
A0255.782
A0255.783
A0255.784
A0255.785
A0255.786
A0255.787
A0255.788
A0255.789
A0255.790
A0255.791
A0255.792
A0255.793
A0255.794
A0255.795
A0255.796
A0255.797
A0255.798
A0255.799
A0255.800
A0255.801
A0255.802
A0255.803
A0255.804
A0255.805
A0255.806
A0255.807
A0255.808
A0255.809

```

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

INPUT REGISTERS:     None

OUTPUT REGISTERS:   (A0) = Find indicator. If (A0) = 0, no reply was located; if (A0) ≠ 0, a reply is being returned to the caller.  
                      (A2) = ID of replying task  
                      (S1) = OUTPUT+0    } Reply  
                      (S2) = OUTPUT+1   }



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
*
**

```

\*\*\*\*\*

```

GETREPLY = *
          A6      LE@SAVER
          A7      CTID, A0
          A3      A6+A7
          A7      SAVER
          A6      A6+A7                ADDRESS OF SAVE AREA FOR ORIGIN TASK
*
*
*

```

```

*
*      SAVE THE REGISTERS CLEARING THOSE ALREADY DESTROYED
*

```

```

          SAVEA1, A6 A1
          SAVEA2, A6 A2
          SAVEA3, A6 A3
          SAVEA4, A6 A4
          SAVEA5, A6 A5
          SAVES0, A6 S0
          SAVES1, A6 S1
          SAVES2, A6 S2
          SAVES3, A6 S3
          SAVES4, A6 S4
          SAVES5, A6 S5
          SAVES6, A6 S6
          SAVES7, A6 S7
          A7      B00
          SAVERT, A6 A7
          S6      0
          SAVEA7, A6 S6
          SAVEA0, A6 S6                ASSUME NO FIND WILL OCCUR
          SAVEA6, A6 S6
          A3      A6
*
*
*

```

```

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

```

```

          A6      CMCC
          A1      CTID, A0
          A6      A6+A1
          A5      LH@CMCC
          A6      A6+A5
          A2      0
          A4      MAXTN
          A5      LE@CMCC

```

A0255. 310  
A0255. 311  
A0255. 312  
A0255. 313  
A0255. 314  
A0255. 315  
A0255. 316  
A0255. 317  
A0255. 318  
A0255. 319  
A0255. 320  
A0255. 321  
A0255. 322  
A0255. 323  
A0255. 324  
A0255. 325  
A0255. 326  
A0255. 327  
A0255. 328  
A0255. 329  
A0255. 330  
A0255. 331  
A0255. 332  
A0255. 333  
A0255. 334  
A0255. 335  
A0255. 336  
A0255. 337  
A0255. 338  
A0255. 339  
A0255. 340  
A0255. 341  
A0255. 342  
A0255. 343  
A0255. 344  
A0255. 345  
A0255. 346  
A0255. 347  
A0255. 348  
A0255. 349  
A0255. 350  
A0255. 351  
A0255. 352  
A0255. 353  
A0255. 354  
A0255. 355  
A0255. 356  
A0255. 357  
A0255. 358  
A0255. 359  
A0255. 360  
A0255. 361  
A0255. 362  
A0255. 363  
A0255. 364  
A0255. 365  
A0255. 366  
A0255. 367  
A0255. 368  
A0255. 369  
A0255. 370  
A0255. 371

```

GETREP10 = *
GET, S1 SS&S7, CCHEAD, A6
S0 S1
JSZ GETREP20
A7 S1 ADDRESS OF THE CMOD
S0 CMOUT1, A7
JSN GETREP30 JUMP IF A REPLY EXISTS
*
* THERE IS NO REPLY FROM THIS TASK. UPDATE TO NEXT TASK
*
GETREP20 = *
A6 A6+A5 NEXT CHAIN CONTROL
A2 A2+1 NEXT TASK ID
A0 A4-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 CMOUT0, A7
S2 CMOUT1, A7
SAVE S1, A3 S1
SAVE S2, A3 S2 MOVE THE REPLY OUT OF THE CMOD
R UNCHAIN REMOVE IT FROM THE CHAIN
AS SAVERT, A3 RESTORE B00 FOR TRACE ENTRIES
B00 AS
ERRAN
A6 CMP00L DEALLOCATE THE MEMORY
R MEMDE
AS SAVERT, A3 RESTORE B00 FOR TRACE ENTRIES
B00 AS
A0 AS
ERRAN
*
* RESTORE THE REGISTERS AND EXIT
*
GETREP40 = *
A6 A3
A0 SAVEA0, A6
A1 SAVEA1, A6
A2 SAVEA2, A6
A3 SAVEA3, A6
A4 SAVEA4, A6
A5 SAVEA5, A6
S0 SAVE S0, A6
S1 SAVE S1, A6
S2 SAVE S2, A6
S3 SAVE S3, A6
S4 SAVE S4, A6
S5 SAVE S5, A6
S6 SAVE S6, A6
S7 SAVE S7, A6
A7 SAVERT, A6
B00 A7
A7 SAVEA7, A6
A6 SAVEA6, A6
J B00

```

```

A0255 372
A0255 373
A0255 374
A0255 375
A0255 376
A0255
A0255 377
A0255 378
A0255 380
A0255 381
A0255 382
A0255 383
A0255 384
A0255 385
A0255 386
A0255 387
A0255 388
A0255 389
A0255 390
A0255 391
A0255 392
A0255 393
A0255 394
A0255 395
A0255 396
A0255 397
A0255 398
A0255 399
A0255 900
A0255 901
A0255 902
A0255 903
A0255 904
A0255 905
A0255 906
A0255 907
A0255 908
A0255 909
A0255 910
A0255 911
A0255 912
A0255 913
A0255 914
A0255 915
A0255 916
A0255 917
A0255 918
A0255 919
A0255 920
A0255 921
A0255 922
A0255 923
A0255 924
A0255 925
A0255 926
A0255 927
A0255 928
A0255 929
A0255 930
A0255 931
A0255 932

```

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

INPUT REGISTERS:      (A2) = ID OF REQUESTED TASK  
                           (s1) = INPUT+0      } REQUEST  
                           (s2) = INPUT+1

OUTPUT REGISTERS:    (s1) = OUTPUT+0      } REPLY  
                           (s2) = OUTPUT+1

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

```

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
**
**
*****
TSKREQ      =      *
            A3      LE@SAVER
            A4      CTID,A0
            A3      A3*A4
            A4      SAVER
            A3      A3+A4          ADDRESS OF SAVE AREA FOR ORIGIN TASK
*
*           SAVE THE REGISTERS AND CLEAR OUT THOSE THAT HAVE ALREADY
*           BEEN DESTROYED
*
            SAVEA0,A3 A0
            SAVEA1,A3 A1
            SAVEA2,A3 A2
            SAVEA5,A3 A5
            SAVEA6,A3 A6
            SAVEA7,A3 A7
            SAVES0,A3 S0
            SAVES1,A3 S1
            SAVES2,A3 S2
            SAVES3,A3 S3
            SAVES4,A3 S4
            SAVES5,A3 S5
            SAVES6,A3 S6
            SAVES7,A3 S7
            S6      0
            SAVEA3,A3 S6
            SAVEA4,A3 S6
            A4      B00
            SAVERT,A3 A4          SAVE THE RETURN ADDRESS
            S0      S1
            ERRSZ   B0492.5      ERROR IF INPUT&1 = 0
TSKREQ10    =      *
            A6      CMPPOOL      MEMORY POOL NUMBER
            A7      CMSIZE       NUMBER OF WORD REQUIRED
            R       MEMAL        ALLOCATE A CMOD
            A5      SAVERT,A3    RESTORE B00 FOR TRACE ENTRIES
            B00     A5
*
*           CHECK STATUS RETURNED FROM MEMAL
*
            A0      A5
            JAZ     TSKREQ20     JUMP IF GOOD RETURN
            A7      3
            A0      A5-A7
            ERRAN

```

```

*
* AT THIS POINT THERE IS NO MEMORY AVAILABLE. SUSPEND AND
* TRY AGAIN LATER
*
S7      SUSP
EX
R      ERROR0
J      TSKREQ10
*
* A CMOD HAS BEEN ALLOCATED PUT THE INPUT MESSAGE IN THE CMOD
*
TSKREQ20 =      *
S1      SAVES1,A3
S2      SAVES2,A3
CMIN0,A7 S1
CMIN1,A7 S2
POST    12,S1,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*A5
A4      CMCC
A5      A4+A5      CHAIN CONTROL HEADER ADDRESS
*
* CALCULATE CHAIN CONTROL WORD ADDRESS
* CHAIN CONTROL WORD = CHAIN CONTROL HEADER + LENGTH OF CHAIN
* CONTROL HEADER + CURRENT TASK ID
*
A6      LH@CMCC
A6      A6+A5
A4      CTID,A0
A6      A6+A4      CHAIN CONTROL WORD ADDRESS
TSKREQ30 =      *
A1      1
STPLK,A0 A1      LOCK OUT INTERRUPTS
GET,S1  S6&S7,CCTL,A5
GET,S2  S6&S7,CCTM,A5
S0      S2
ERRSZ   ERROR IF QUEUE MAX IS ZERO
S0      S1-S2
JSZ     TSKREQ70  JUMP IF QUEUE AT LIMIT
S4      1
S1      S1+S4
GET,S2  S6&S7,CCQL,A6
GET,S3  S6&S7,CCQM,A6
S0      S3
ERRSZ
S0      S2-S3
JSZ     TSKREQ70  JUMP IF QUEUE AT LIMIT
S2      S2+S4
PUT,S1  S6&S7,CCTL,A5  UPDATE QUEUE COUNTS
PUT,S2  S6&S7,CCQL,A6
R      CHAIN
A5      SAVERT,A3  RESTORE B00 FOR TRACE ENTRIES
B00    A5
A1      0
STPLK,A0 A1      ENABLE INTERRUPTS

```

```

A0255.251
A0255.252
A0255.253
A0255.254
A0255.255
A0255.256
A0255.257
A0255.258
A0255.259
A0255.260
A0255.261
A0255.262
A0255.263
A0255.264
A0255.265
A0255.266
A0255.267
A0255.268
A0255.269
A0255.270
A0255.271
A0255.272
A0255.273
A0255.274
A0255.275
A0255.276
A0255.277
A0255.278
A0255.279
A0255.280
A0255.281
A0255.282
A0255.283
A0255.284
A0255.285
A0255.286
A0255.287
A0255.288
A0255.289
A0255.290
A0255.291
A0255.292
A0255.293
A0255.294
A0255.295
A0255.296
A0255.297
A0255.298
A0255.299
A0255.300
A0255.301
A0255.302
A0255.303
A0255.304
A0255.305
A0255.306
A0255.307
A0255.308
A0255.309
A0255.310

```

```

*
* THE REQUEST HAS BEEN QUEUED TO THE DESTINATION TASK READY THE
* TASK AND SUSPEND UNTIL A REPLY IS RECEIVED
*
TSKREQ40 = *
S6 SAVEA2, A3
S7 RTSS
EX
R ERROR0
TSKREQ50 = *
S1 CMOUT0, A7
S2 CMOUT1, A7
S0 S2
JSN TSKREQ60
*
* THE DESTINATION TASK DID NOT REACTIVATE ME
*
S7 SUSP
EX
R ERROR0
J TSKREQ50
TSKREQ60 = *
SAVES1, A3 S1
SAVES2, A3 S2
POST 12, S1, S2
R UNCHAIN UNCHAIN THE CMOD
A5 SAVERT, A3 RESTORE B00 FOR TRACE ENTRIES
B00 A5
ERRAN
A6 CMPPOOL RELEASE THE CMOD MEMORY
R MEMDE
A5 SAVERT, A3 RESTORE B00 FOR TRACE ENTRIES
B00 A5
A0 A5 STATUS CHECK
ERRAN
*
* RESTORE THE REGISTERS AND EXIT
*
A0 SAVEA0, A3
A1 SAVEA1, A3
A2 SAVEA2, A3
A5 SAVEA5, A3
A6 SAVEA6, A3
A7 SAVEA7, A3
S0 SAVES0, A3
S1 SAVES1, A3
S2 SAVES2, A3
S3 SAVES3, A3
S4 SAVES4, A3
S5 SAVES5, A3
S6 SAVES6, A3
S7 SAVES7, A3
A4 SAVERT, A3
B00 A4
A4 SAVEA4, A3
A3 SAVEA3, A3
J B00
TSKREQ70 = *
*
* 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)



## TASK PROCESSORS

### ● 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



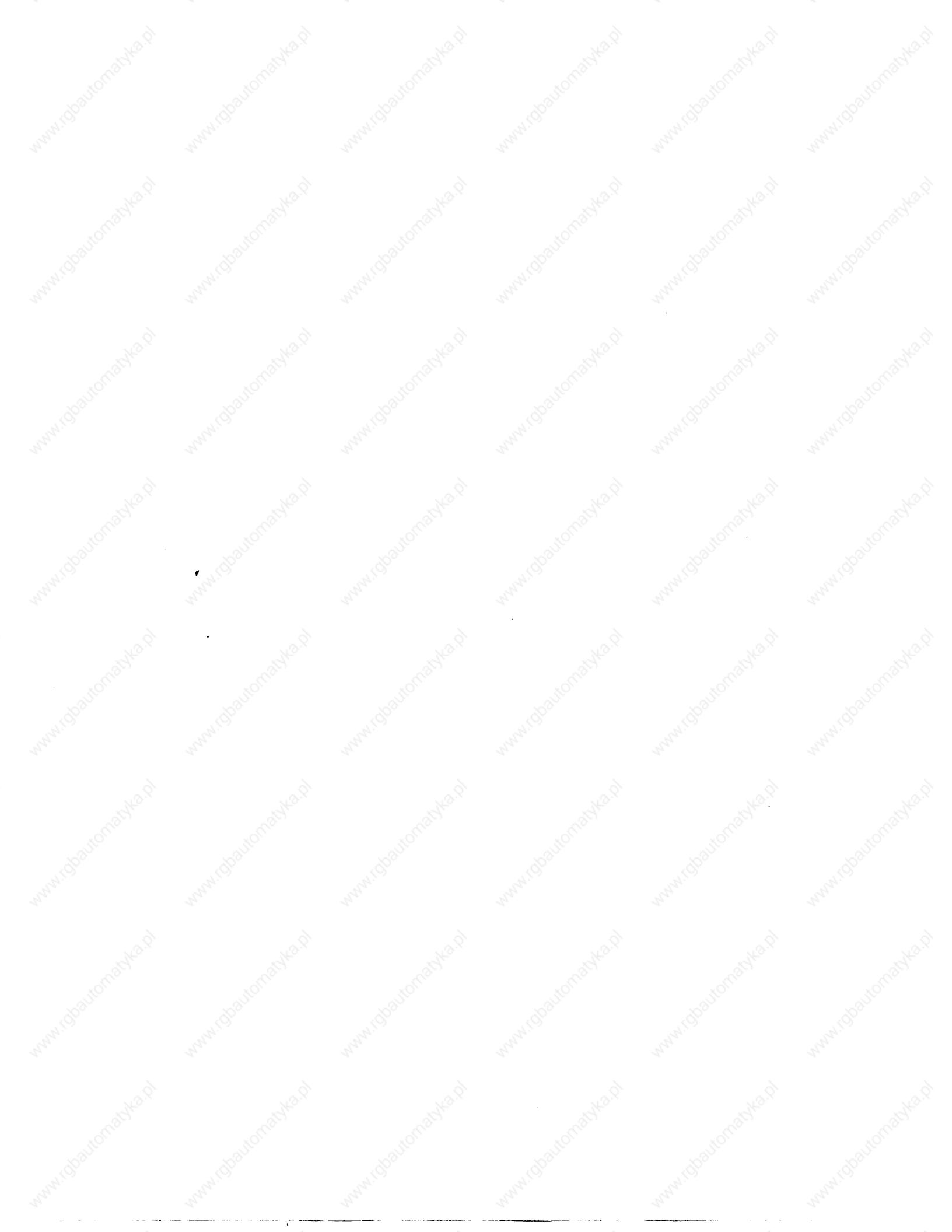
## CRAY-OS STARTUP

- 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.

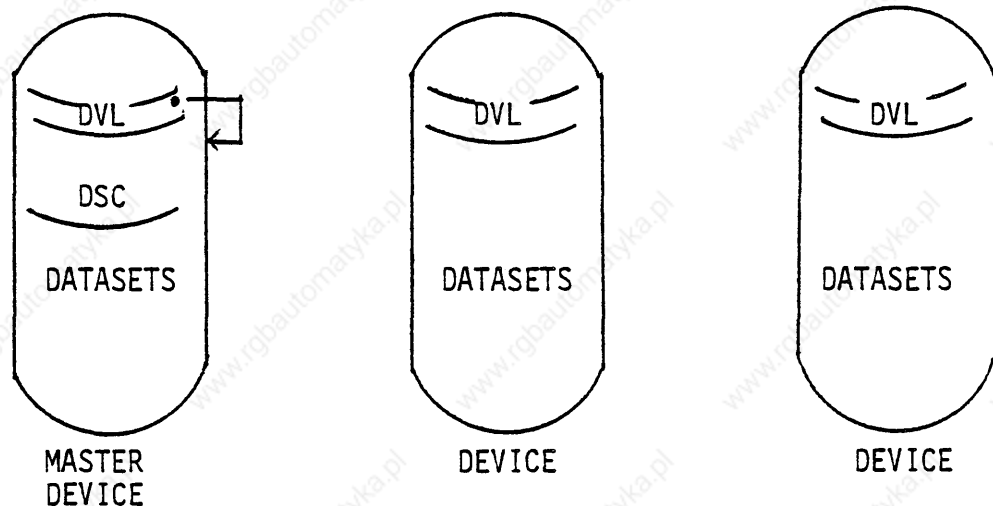
A ROLL JOB INDEX DATASET IS INITIALIZED (\$ROLL).

- 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 ACCESSABLE 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.      ACCESS, ENTER, DN=PDSLOAD.
*.                                       ACCESS, ENTER, DN=SKIPD.
*. SDR initialization job             ACCESS, ENTER, DN=SKIPF.
*.                                       ACCESS, ENTER, DN=SKIPR.
ACCESS, ENTER, DN=AUDIT.              ACCESS, ENTER, DN=UNB.
ACCESS, ENTER, DN=BUILD.              ACCESS, ENTER, DN=UPDATE.
ACCESS, ENTER, DN=CAL.                ACCESS, ENTER, DN=WRITEDS.
ACCESS, ENTER, DN=CFT.                (EOF)
ACCESS, ENTER, DN=COMPARE.
ACCESS, ENTER, DN=COPYD.
ACCESS, ENTER, DN=COPYF.
ACCESS, ENTER, DN=COPYR.
ACCESS, ENTER, DN=DSDUMP.
ACCESS, ENTER, DN=DUMP.
ACCESS, ENTER, DN=EXTRACT.
ACCESS, ENTER, DN=FDUMP.
ACCESS, ENTER, DN=LDR.
ACCESS, ENTER, DN=PDSDUMP.
```



## 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 DATSET 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 DATS ARE VERIFIED AS ARE ALL DATS 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 DATS FOR A JOB ARE VALID THE SYSTEM SETS THE DEVICE RESERVATION TABLE (DRT) ENTRIES FOR THE DATS.



## DATASET ALLOCATION TABLE (DAT) VERIFICATION

A DATASET ALLOCATION TABLE (DAT) MUST RESIDE ENTIRELY 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)

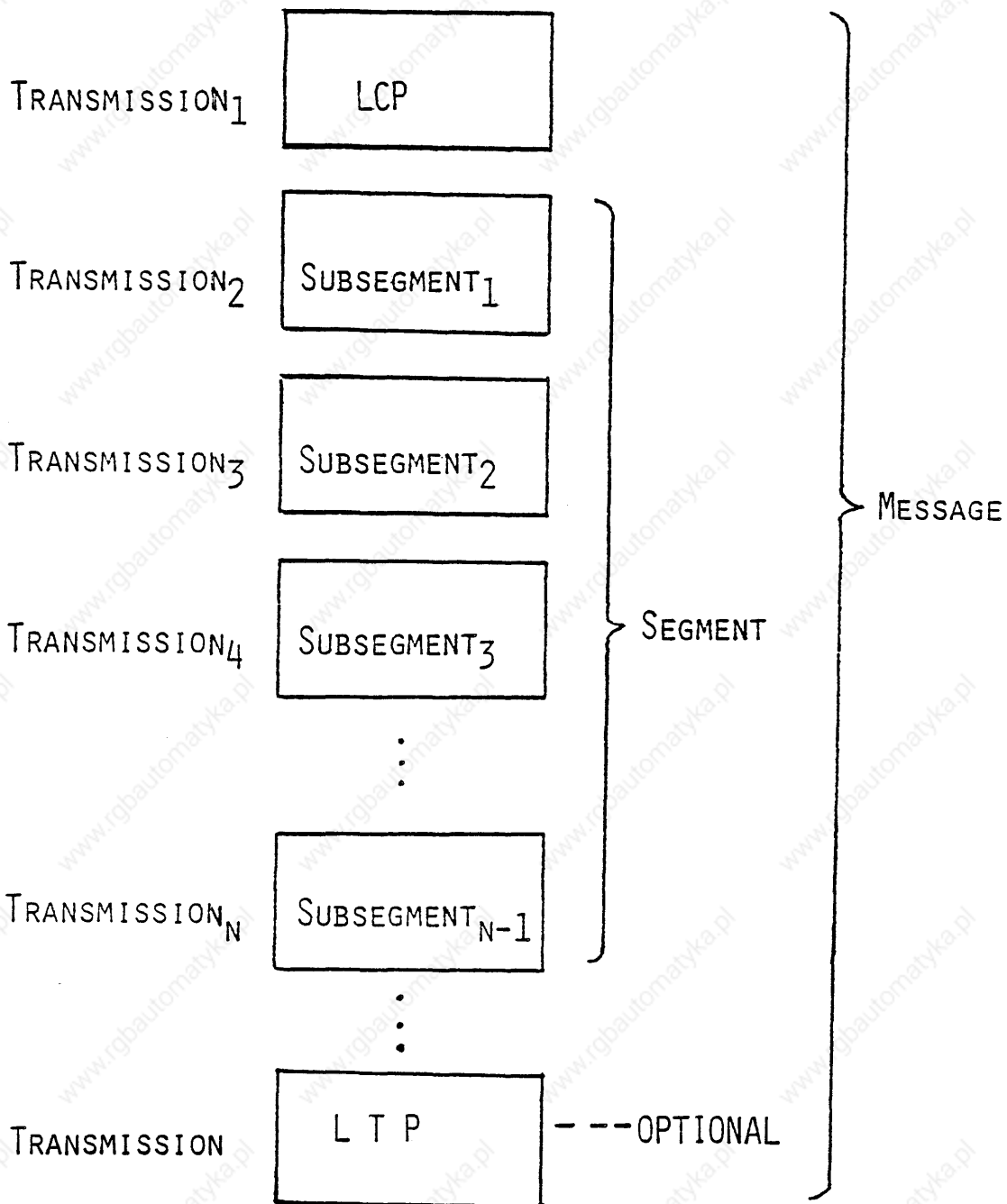
- PROVIDES COMMUNICATIONS WITH FRONT-END SYSTEMS.
- MANAGES I/O TRANSMISSION POOLS CREATED AT STARTUP.
- PROVIDES FOR OPERATOR GUIDANCE.
- CHANNEL DISCIPLINE IS TWO WAY ALTERNATE.
- INDEPENDENT OF FRONT-END TYPE.
- MULTIPLEXES STREAMS (8 INPUT AND 8 OUTPUT).





# GENERAL INTERFACE PROTOCOL

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



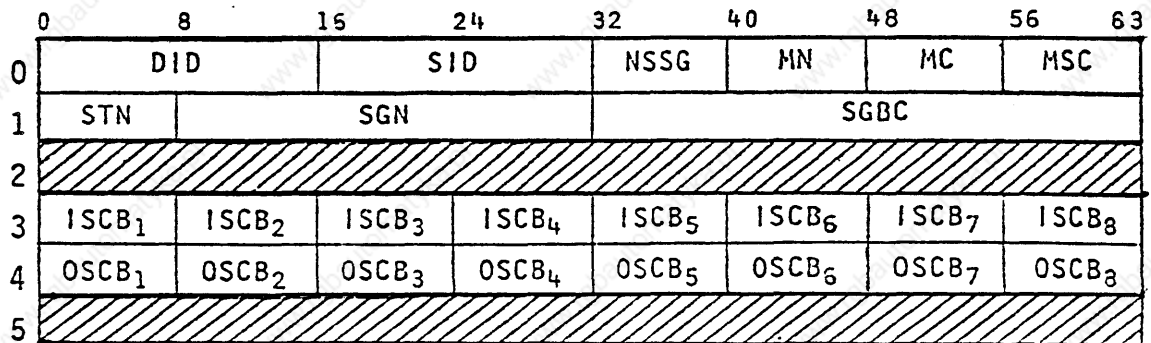


## LINK CONTROL PACKAGE

● 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)





Message codes

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

<sup>††</sup> 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.

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

RECEIVER SCB RESPONSE

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

N = Normal receiver SCB response

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

A = Abnormal receiver SCB response

SENDER SCB RESPONSE

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

N = Normal sender SCB response

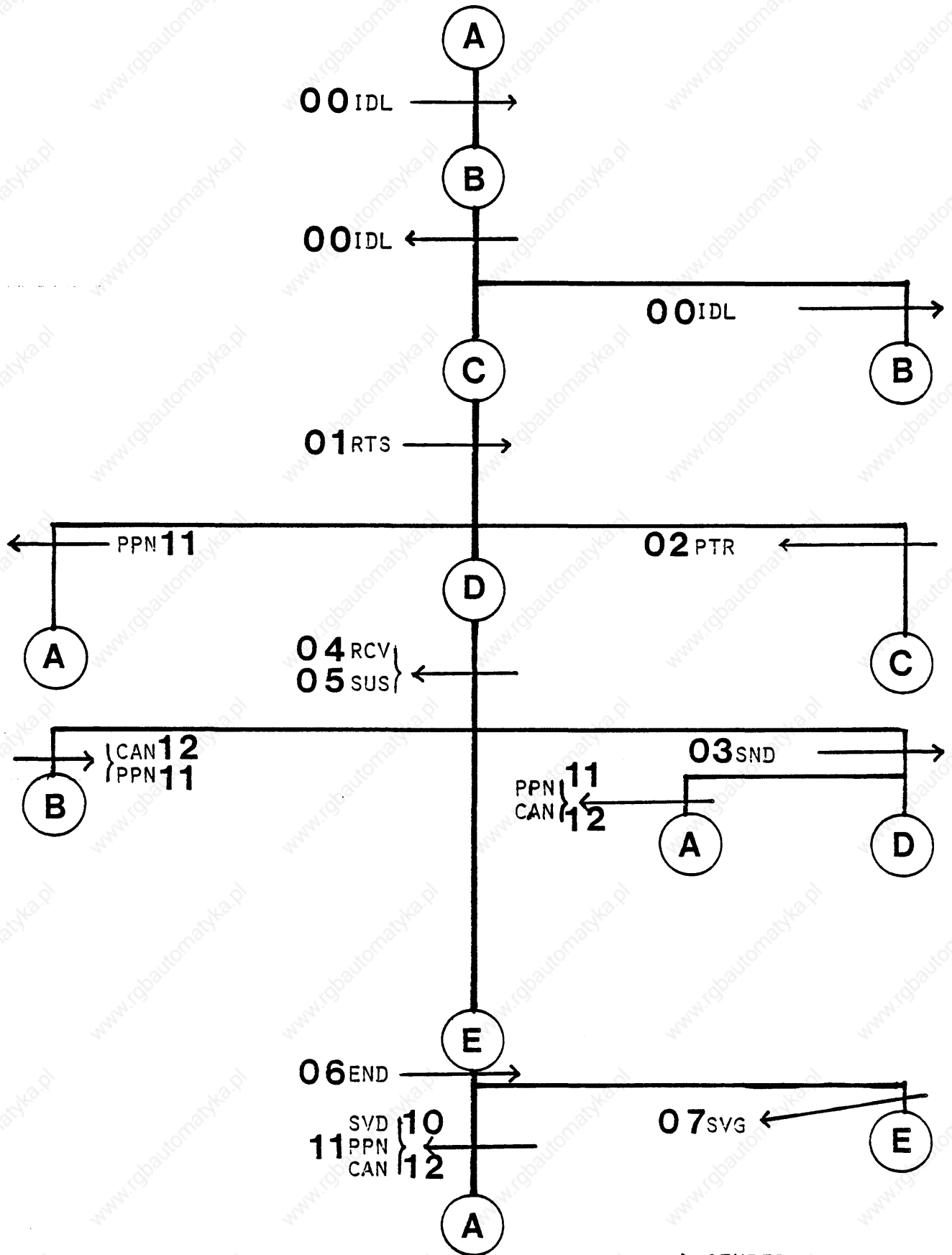
C = Normal sender SCB response which requires change in receiver SCB

A = Abnormal sender SCB response

## BASIC STREAM FLOW

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





**13**

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

STREAM CONTROL BYTE FLOW



## 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

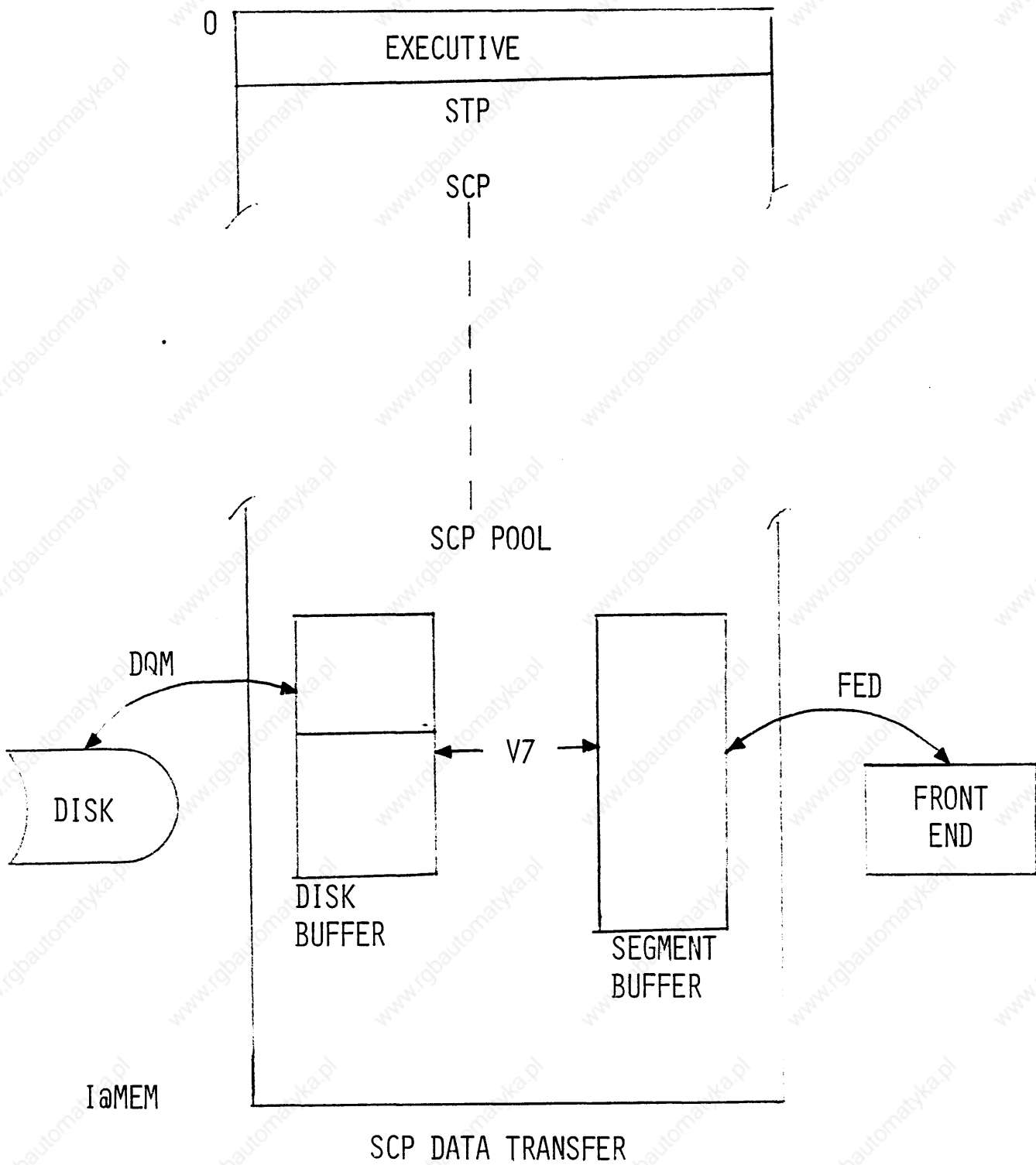
JOB NAME AND PRIORITY

SOURCE AND DESTINATION ID

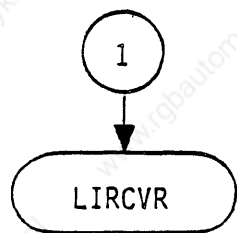
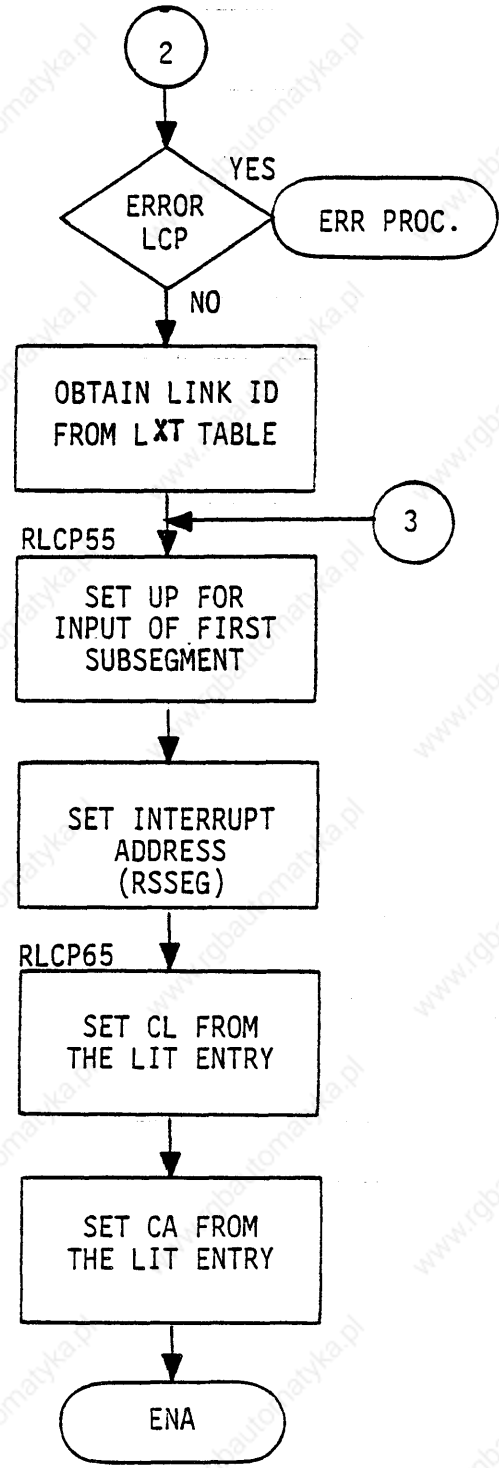
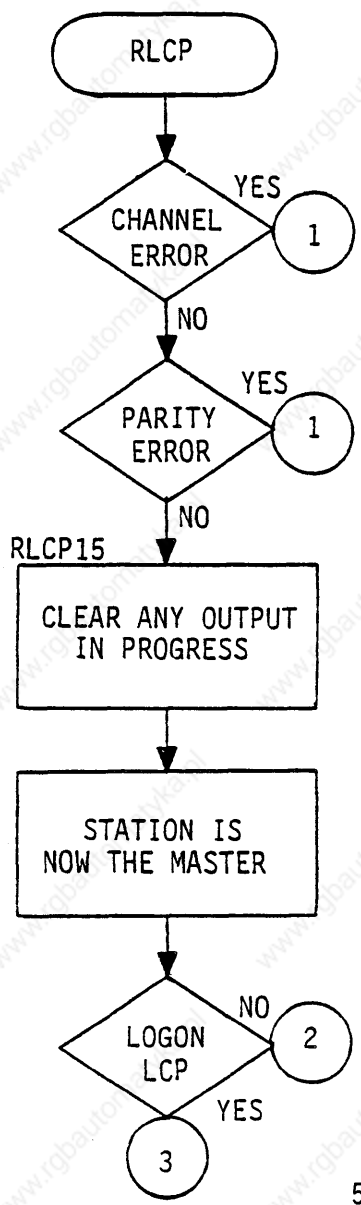
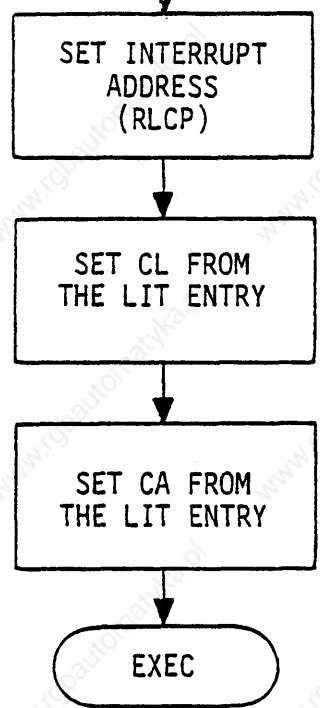
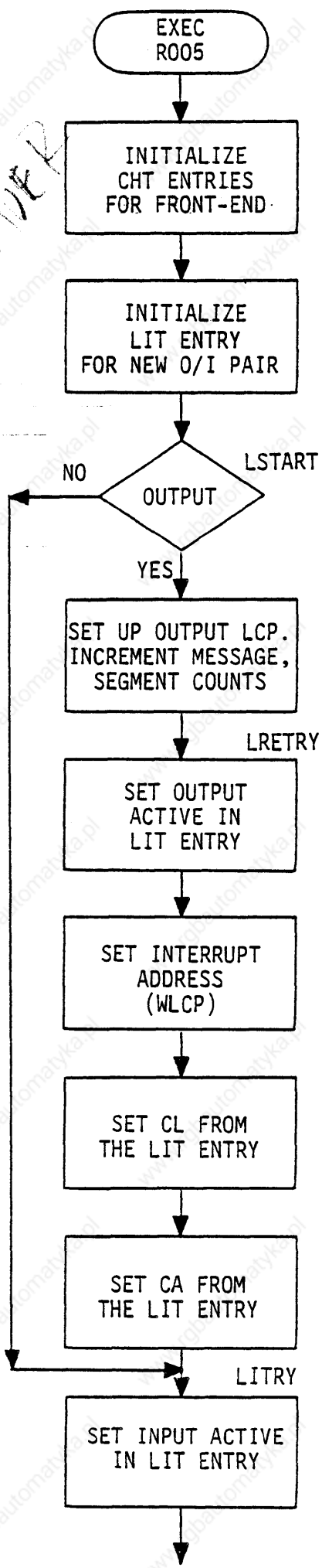
DISPOSITION CODE SET TO JOB DATASET (IN)

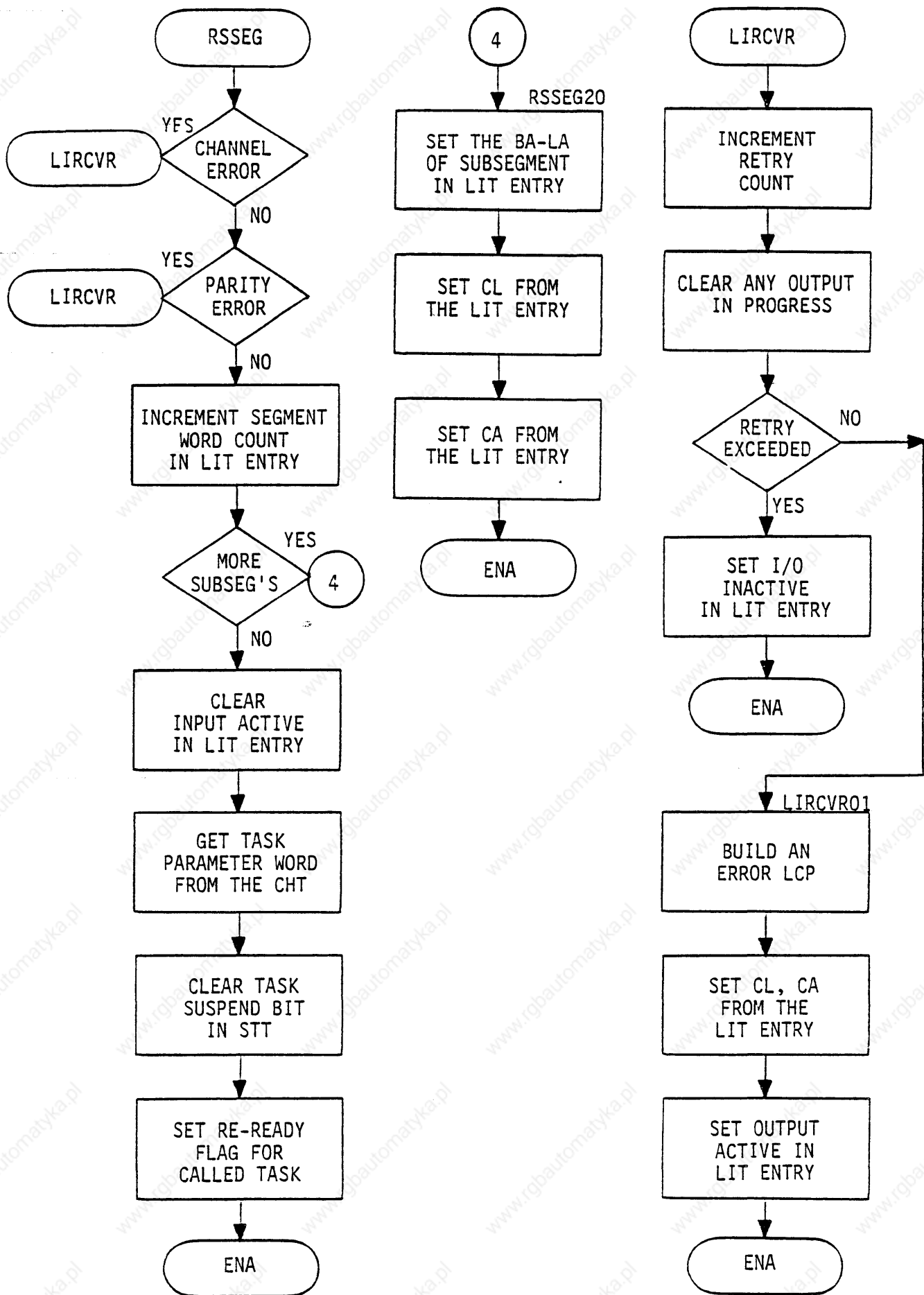
- HAS THE DATASET MADE PERMANENT FOR THE LIFE OF THE JOB. (DSC)





FE  
LAYER









## DISK QUEUE MANAGER (DQM)

## DQM REQUESTS

- PRE ALLOCATE SPACE
- DATA TRANSFER
- DE ALLOCATE SPACE

## MODES

· STATIC

· DYNAMIC

SIZE IN TRACKS

ONLY SIZE IS TRACK

## DEVICE ALLOCATION

BY LOGICAL DEVICE NAME

FOR ELEMENT TABLE

DISK, UNIT, X



## DATASET ALLOCATION TABLE - DAT

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.

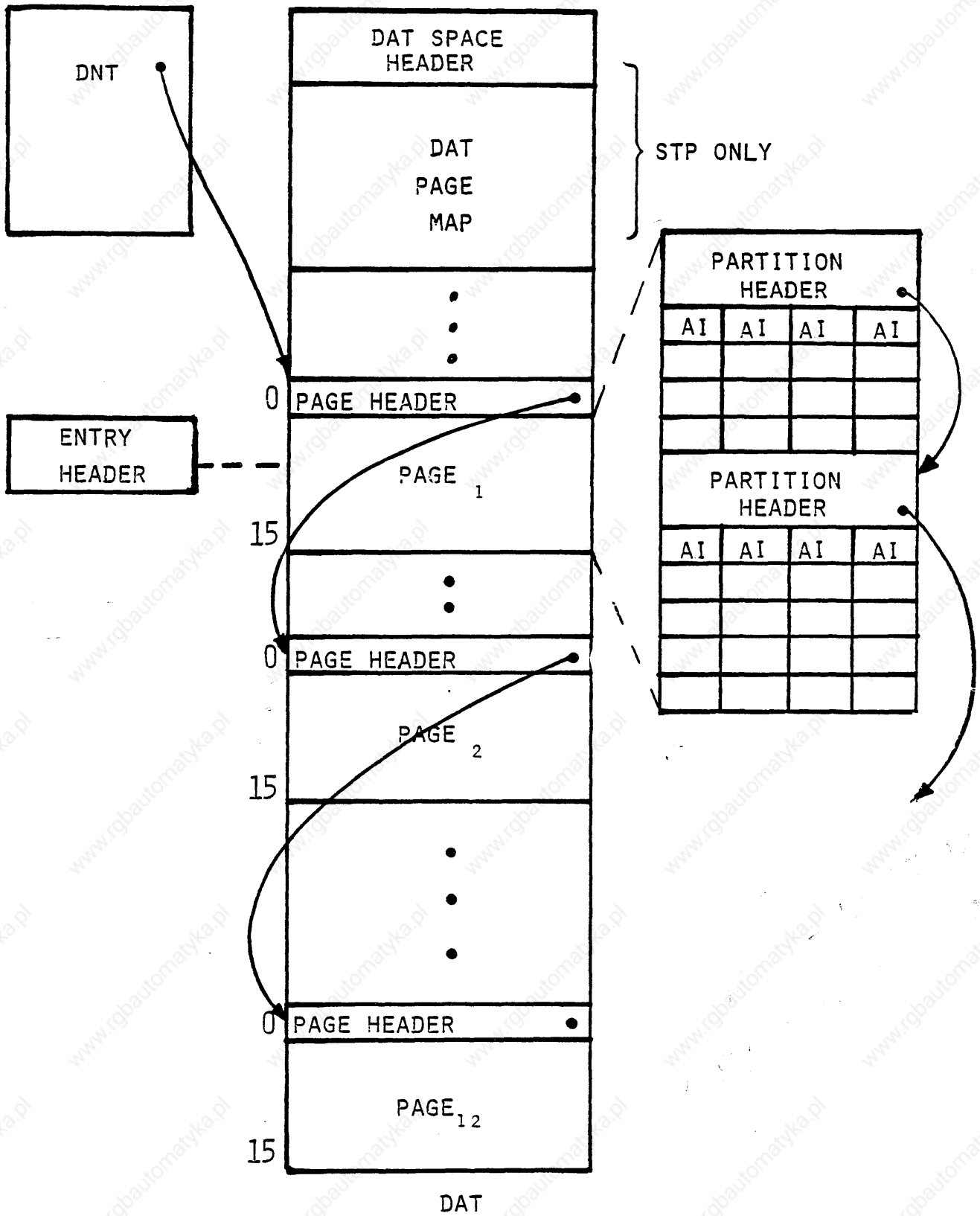
The DAT partition header contains general information concerning a particular partition of the DAT. A partition represents a portion of a single logical device. Each allocation index (AI) in a partition is a bit number in the respective Device Reservation Table (DRT).

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.

- 1 DRT / DRIVE      10 MAR 1974      0 - 1 8 1 2  
1 - 1 2 3 4

EQT      1 entry / unit      ptr to 1st

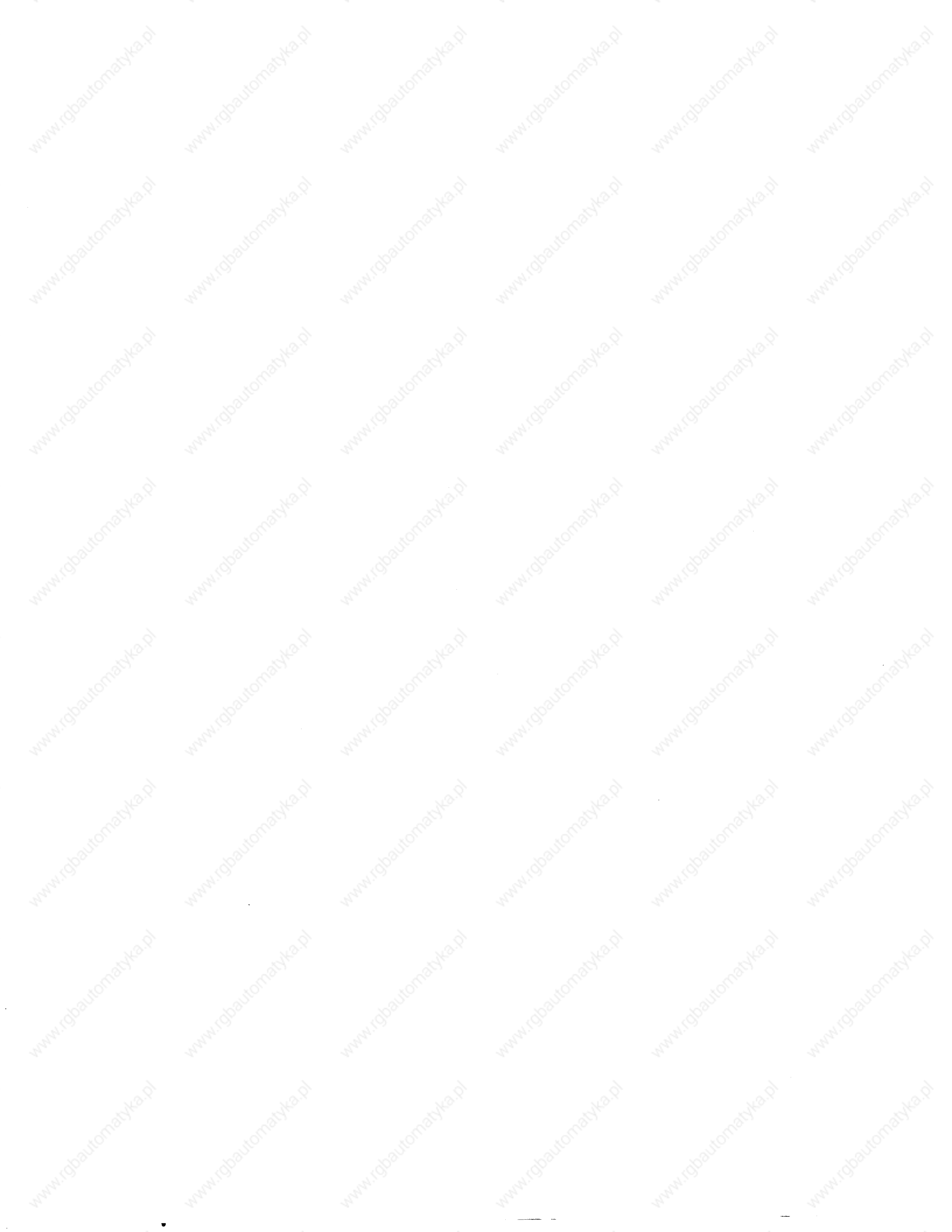
DAT      1 entry / page  
200 13 g@tail  
200 13 g@head+1





	0	16	24	28	32	40	48	63	STP	
	0	TN		WCT		PA			DAT Space Header	
	1	MAP								
		⋮								
DAT Page	0	PN				JORD	DAT			DAT Page Header
	1	DSC			AS		PDS			
	2	CA				DSZ				DAT Entry Header
	3	LDV								
	4			BPB		NPA		NAI		DAT Partition
	5	AI	AI	AI	AI	AI	AI	AI		
	6	AI	AI	AI	AI	AI	AI	AI	DAT Partition	
	7	AI	AI	AI	AI	AI	AI	AI		
	8	LDV							DAT Partition Header	
	9			BPB		NPA		NAI		
	10	AI	AI	AI	AI	AI	AI	AI	DAT Partition	
	11	AI	AI	AI	AI	AI	AI	AI		
	12	AI	AI	AI	AI	AI	AI	AI		
	13	AI	AI	AI	AI	AI	AI	AI		
	14	AI	AI	AI	AI	AI	AI	AI		
15	AI	AI	AI	AI	AI	AI	AI			
	0	PN				JORD	DAT			DAT Page Header
	1	AI	AI	AI	AI	AI	AI			
		⋮								
	15	AI	AI	AI	AI	AI	AI	AI		

Dataset Allocation Table (DAT)





DATASET ALLOCATION TABLE AREA

0020372	042101250000000000412	1777773777774000000000		
0020374	0000000000000000000000	0000000000000000000000	0000000000000000000000	
0020400	0004000000000000000000	0000000000000100000000	0000001620000001223000	0421041323047113231060
0020404	00000000001100000000045	0000020000140001000005	0000060000340002000011	0000120000600003200016
0020410	0000170001000004200022	0000230001200005200026	0000270001400006200032	0000330001600007200036
0020414	0000370002000010200042	0000430002200011200046	0000470000000000000000	0000000000000000000000
0020420	0004000000000000020440	0000000000000100000000	000000000000004026000	0421041323047113231060
0020424	0000000000110000000163	0000500002440012400053	0000540002640013400057	0000600003040014400063
0020430	0000640003240015400067	0000700003440016400073	0000740003640017400077	0001000004040020400103
0020434	0001040004240021400107	0001100004440022400113	0001140004640023400117	0001200005040024400123
0020440	0010000000000000020460	0001240005240025400127	0001300005440026400133	0001340005640027400137
0020444	0001400006040030400143	0001440006240031400147	0001500006440032400153	0001540006640033400157
0020450	0001600007040034400163	0001640007240035400167	0001700007440036400173	0001740007640037400177
0020454	0002000010040040400203	0002040010240041400207	0002100010440042400213	0002140010640043400217
0020460	0014000000000000000000	0002200011040044400223	0002240011240045400227	0002300011440046400000
67	464	0000000000000000000000	0000000000000000000000	0000000000000000000000
	470	0000000000000000000000	0000000000000000000000	0000000000000000000000
	474	0000000000000000000000	0000000000000000000000	0000000000000000000000
0020500	0004000000000000000000	0000013340000100000000	0000000520000000001000	0421041323047113231060
0020504	00000000001100000000001	0000010000000000000000	0000000000000000000000	0000000000000000000000
0020510	0000000000000000000000	0000000000000000000000	0000000000000000000000	0000000000000000000000
0020514	0000000000000000000000	0000000000000000000000	0000000000000000000000	0000000000000000000000
0020520	0004000000000000020540	0000021604000100000000	00000043000000003720000	0421041323047113231060
0020524	00000000001100000000160	0023010114100470202352	0024040145000624603171	0031730151540702203416
0020530	0034210161100704603424	0034250161400713203456	0034700163501012004102	0041640207241035604334
0020534	0043760222341174605002	0050060243341216005071	0064240321541507006437	0064430322201511206446
0020540	0010000000000000020560	0064470322401512206453	0064540322641513406457	0064600323041514406463
0020544	0064640323241515406477	0065020324141521006505	0065060327341741407607	0076100370441742407613
0020550	0076140370641743407617	0076200371101744607624	0076250371341746007631	0076320371541747007635
0020554	0076360371741750007641	0076420372141751007645	0076460372341752007651	0076520372541753007655
0020560	0014000000000000000000	0076560372741754007661	0076620373141755007666	0000000000000000000000
	LOCATIONS 00020564 THROUGH 00020573 CONTAIN 00000000000000000000			
0020574	0000000000000000000000	0000000000000000000000	0000000000000000000000	0000000000000000000000
0020600	0004000000000000000000	0000001064000100000000	0000000520000000001000	0421041323047113231463
0020604	00000000001100000000001	0012010000000000000000	0000000000000000000000	0000000000000000000000

DAT

9 & DD-19-20

%

! " # \$ % & ' ( ) \* + , - . / 0 1 2 3

4 5 6 7 8 9 : ; < = > ? @ A B C

D E F G H I J K L M N O P Q R S

T U V W X Y Z [ \ ] ^ \_

! 0 T U V W X Y Z [ \ ] ^ \_

DD-19-20

DD-19-20

P S

- . 8 : ( B

7 8 9 # \$ % &

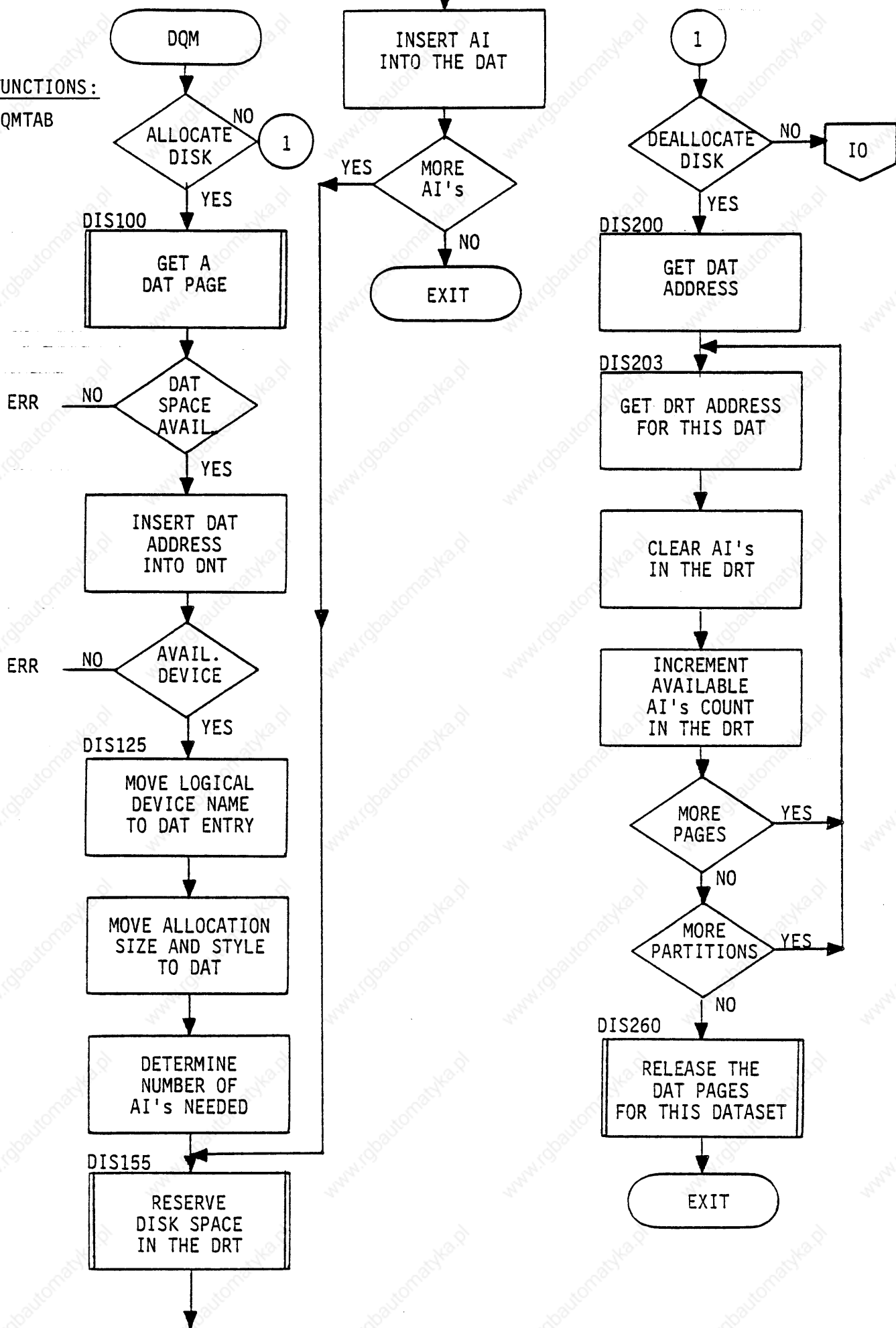
! ' ( ) + , - . / 0 1 2 3

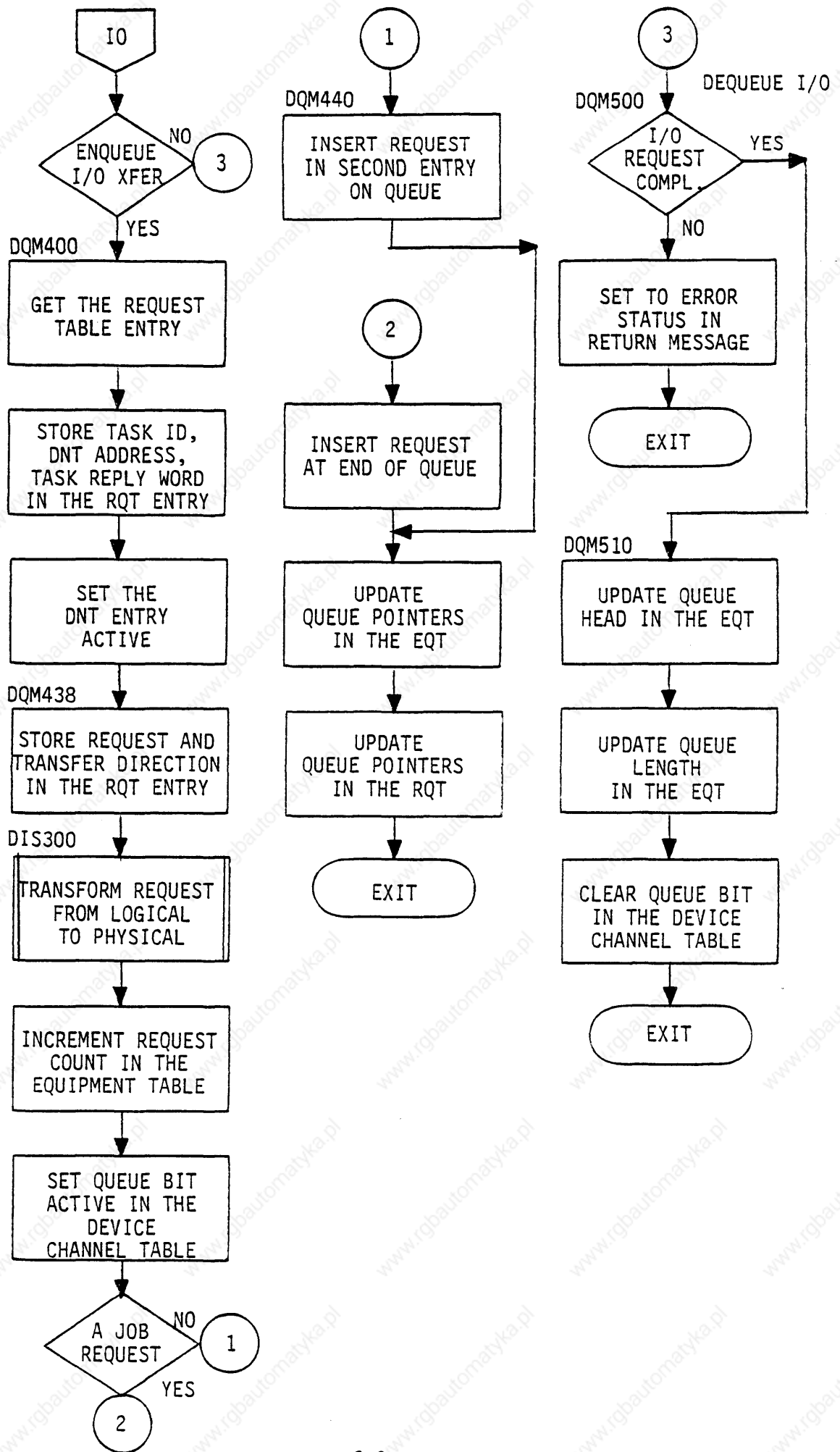
4 5 6 7 B C D E F

# @

DD-19-33

FUNCTIONS:  
DQMTAB







PERMANENT DATASET MANAGER (PDM)

Widziałem  
pal save  
Cieszę się

5

## 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.

Y R  
- P V  
- S C P  
- M S P

Y R  
- P V  
- S C P  
- M S P

DSC - contains DATs

50 words/page

ECW  
Requester  
SAS header

Entry DATs

63 words

7/page

USES HASH into

ARGO

No tree structure

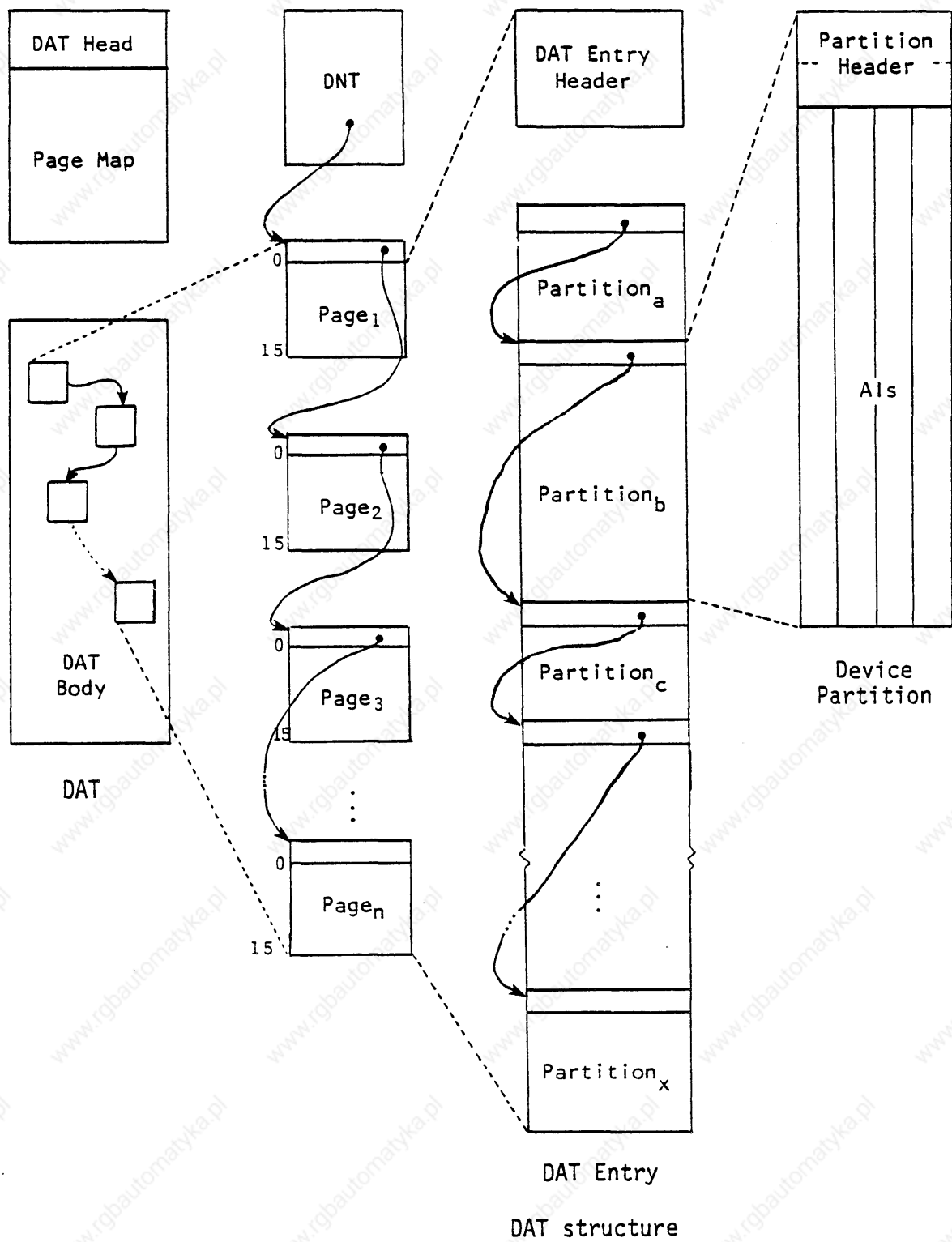
Media Drive label contains

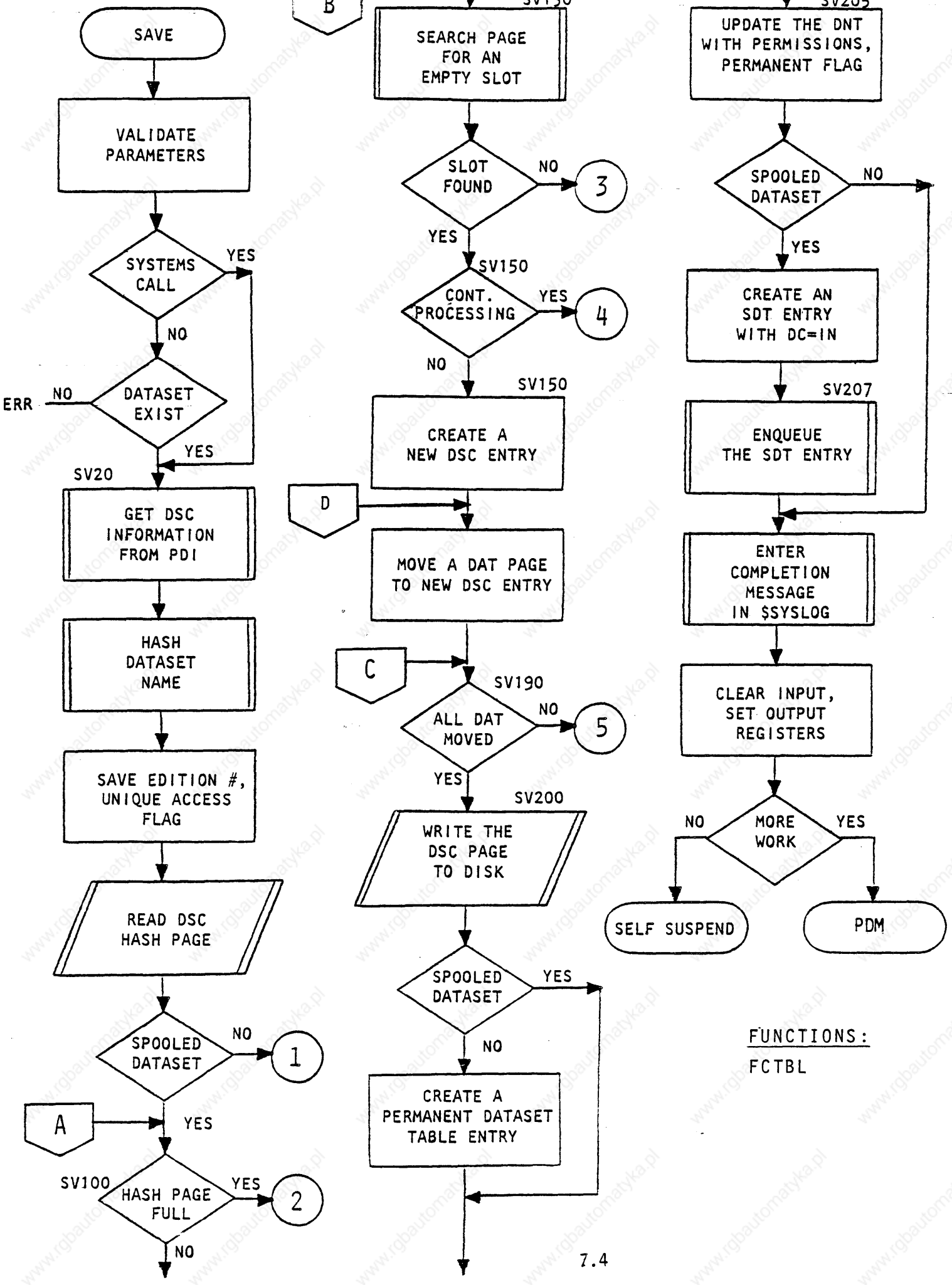
DSC -> DATs => Max DSC 18 M words

NEW MEMORY - SAR (NEW STRUCT)

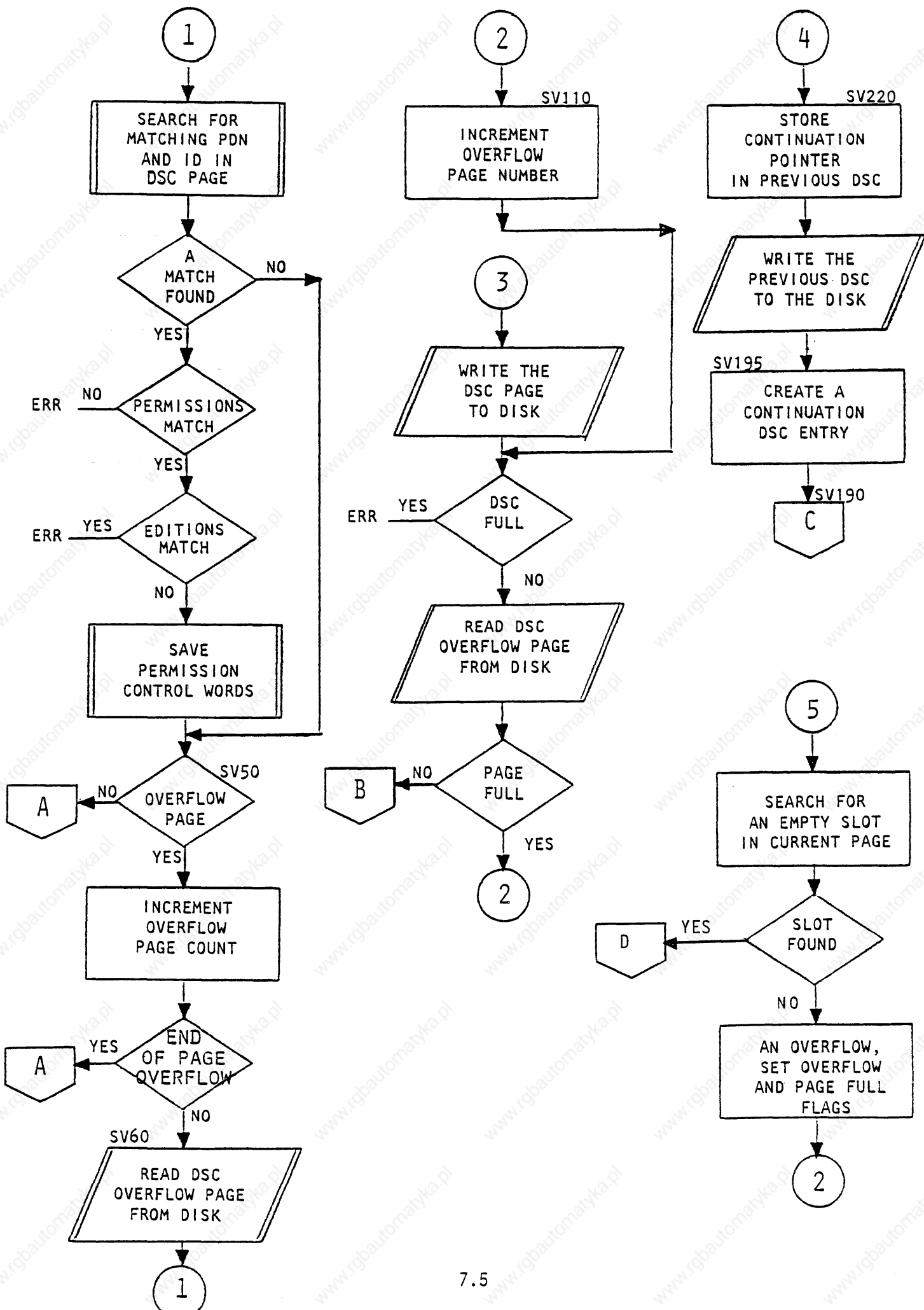
PDB, PDB, PDB -> NEW REQUESTER



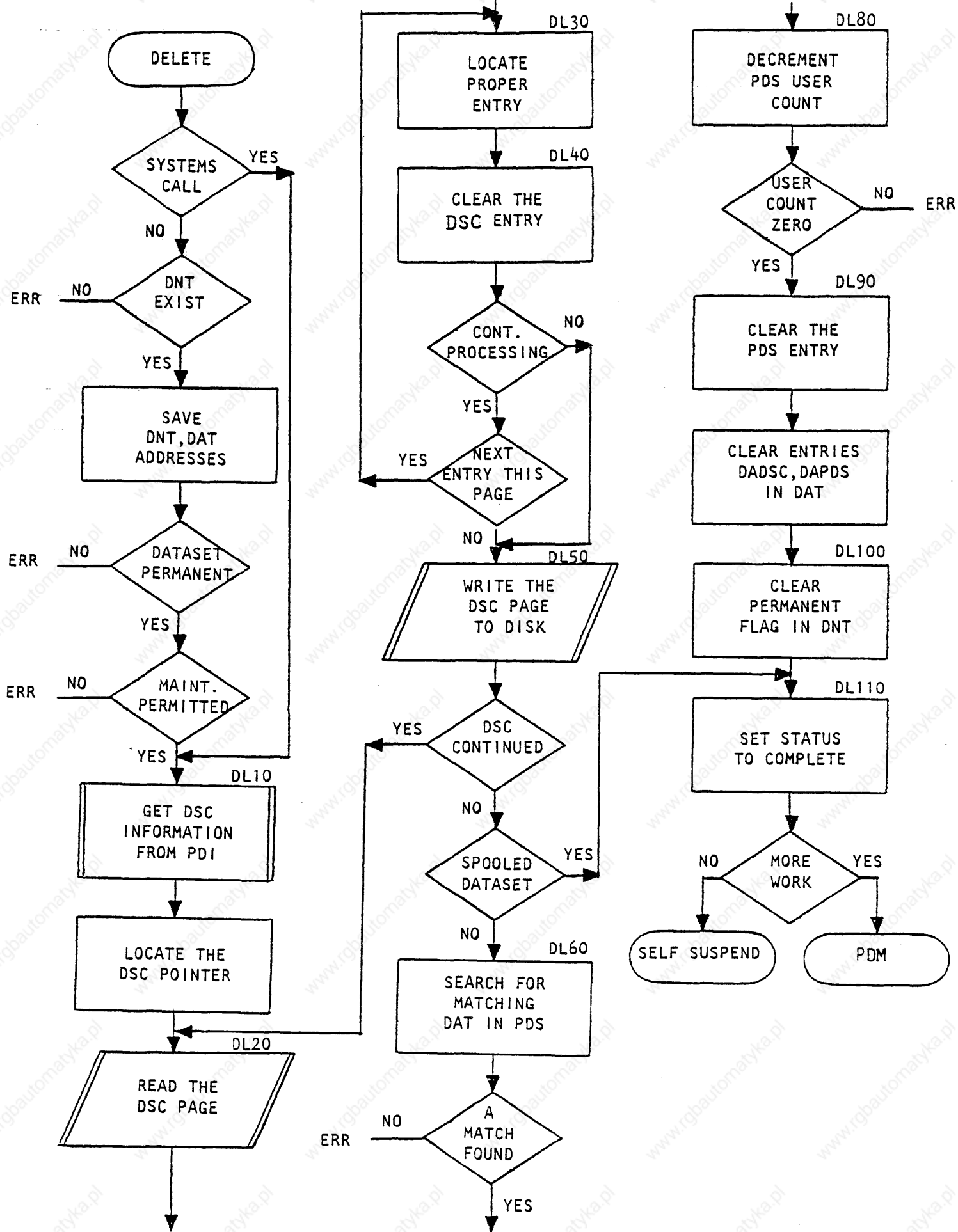




FUNCTIONS :  
FCTBL

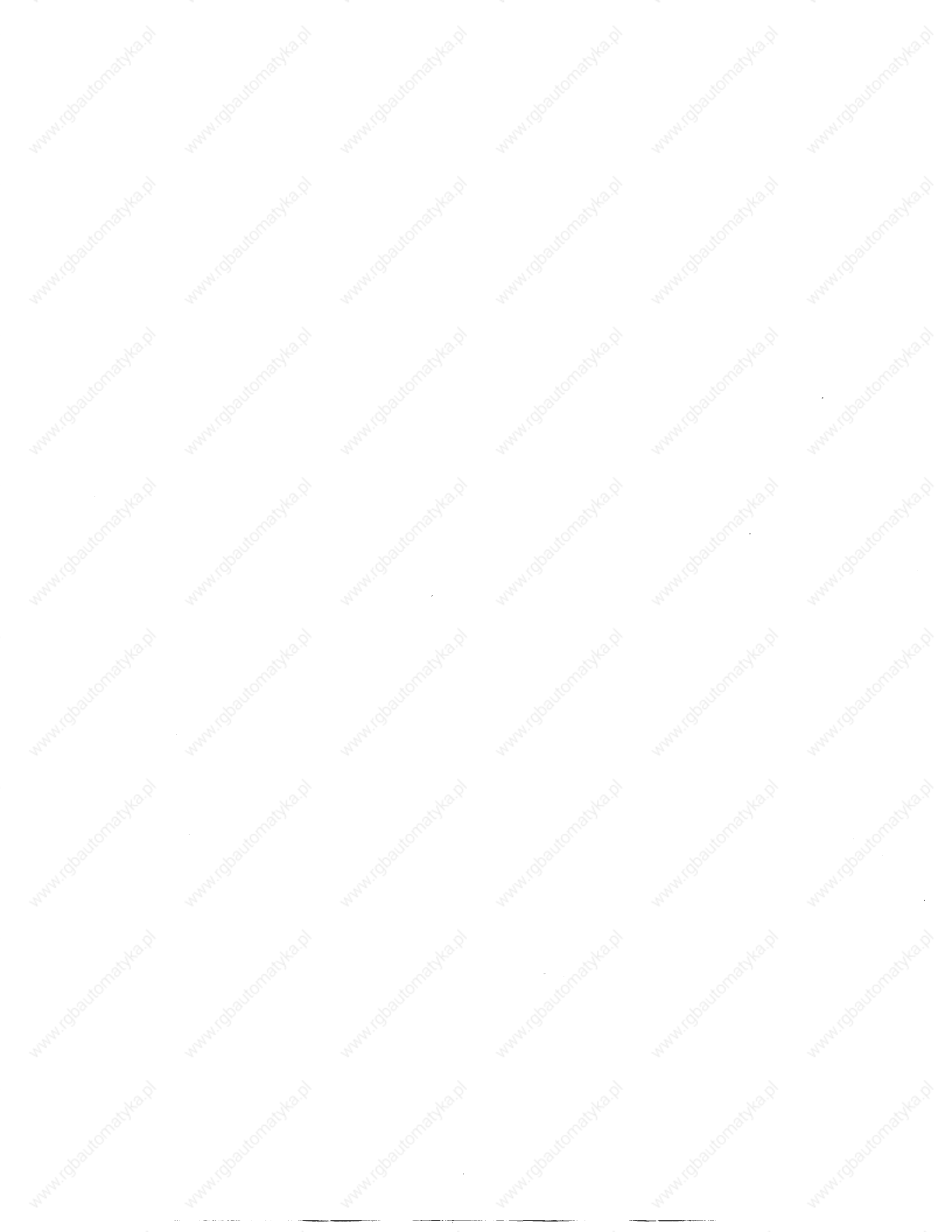








**JOB SCHEDULER (JSH)**





## 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

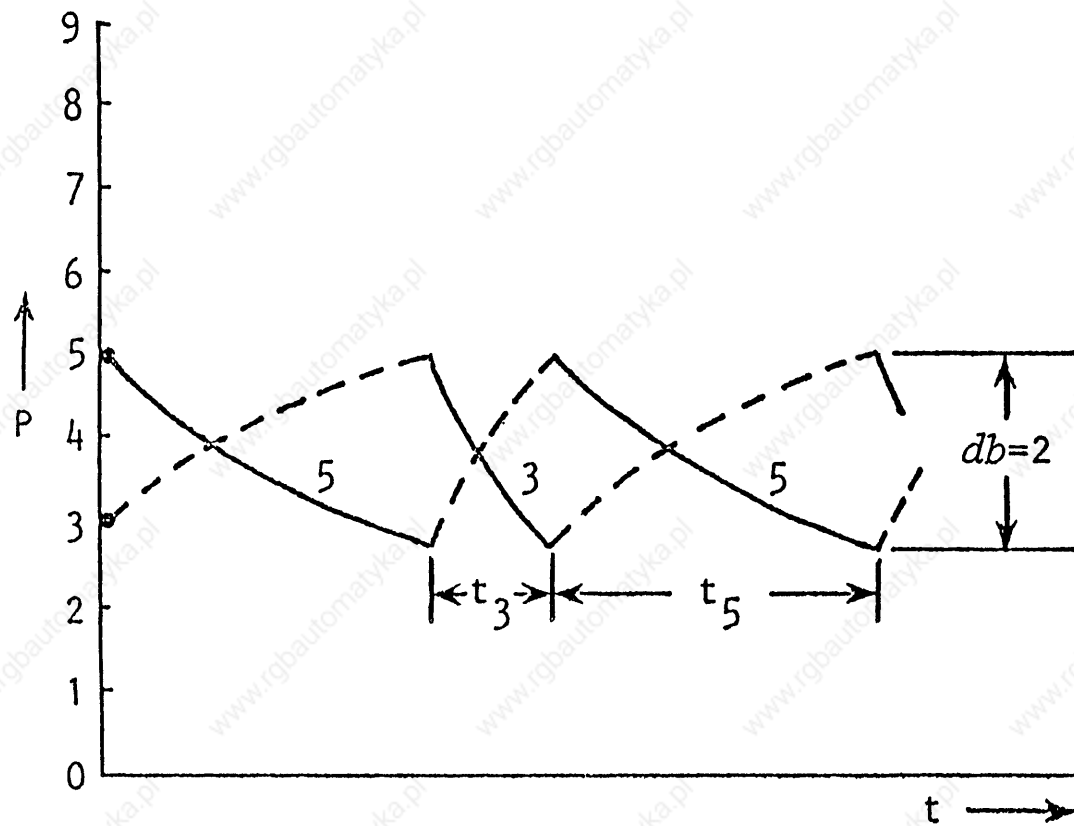
- MEMORY PRIORITY RISES WHILE A JOB IS WAITING FOR MEMORY
- MEMORY PRIORITY FALLS WHEN A JOB IS IN MEMORY AND EXECUTING
- HIGHER PRIORITY JOBS MAY PRE-EMPT MEMORY FROM LOWER PRIORITY JOBS, THEN:

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

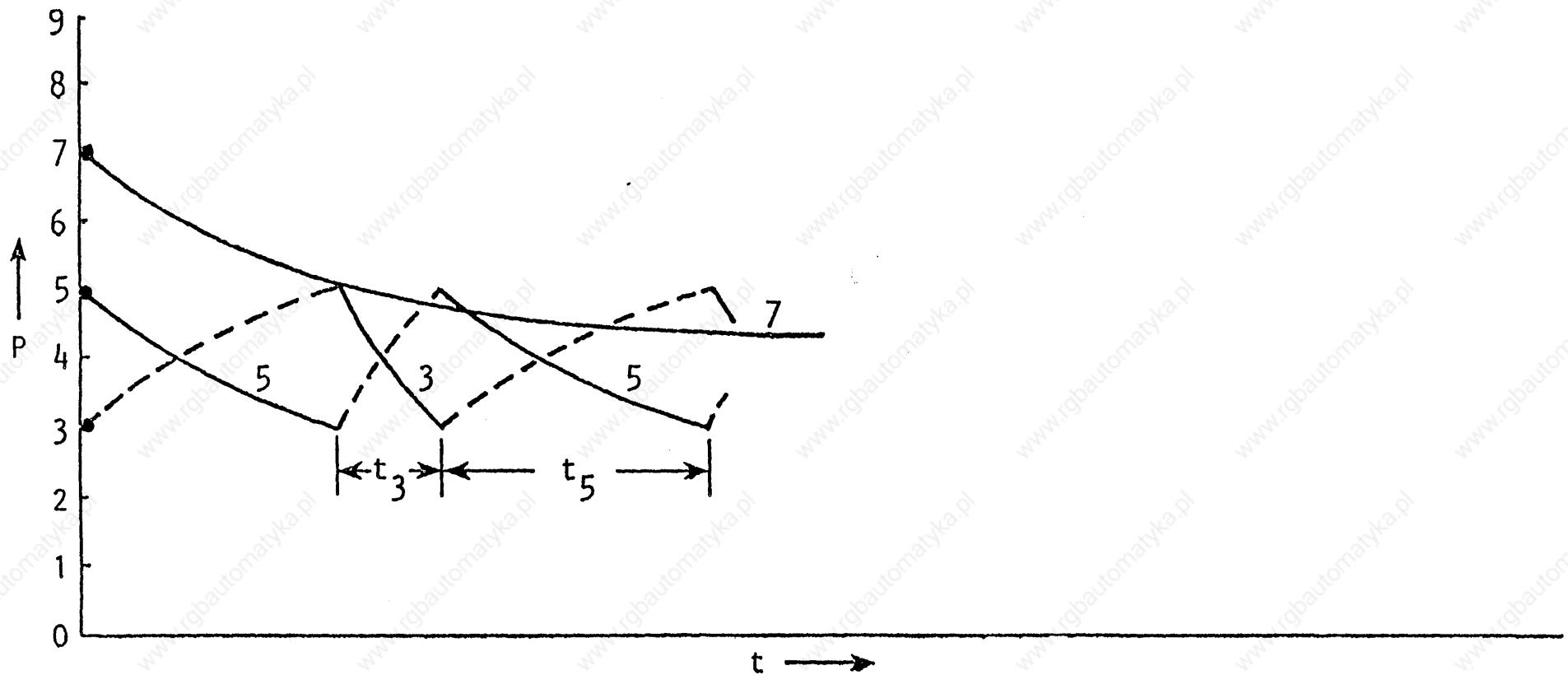






Note: In all the examples, jobs are swapped when the difference between their priorities equals or exceeds  $db$ .

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.)



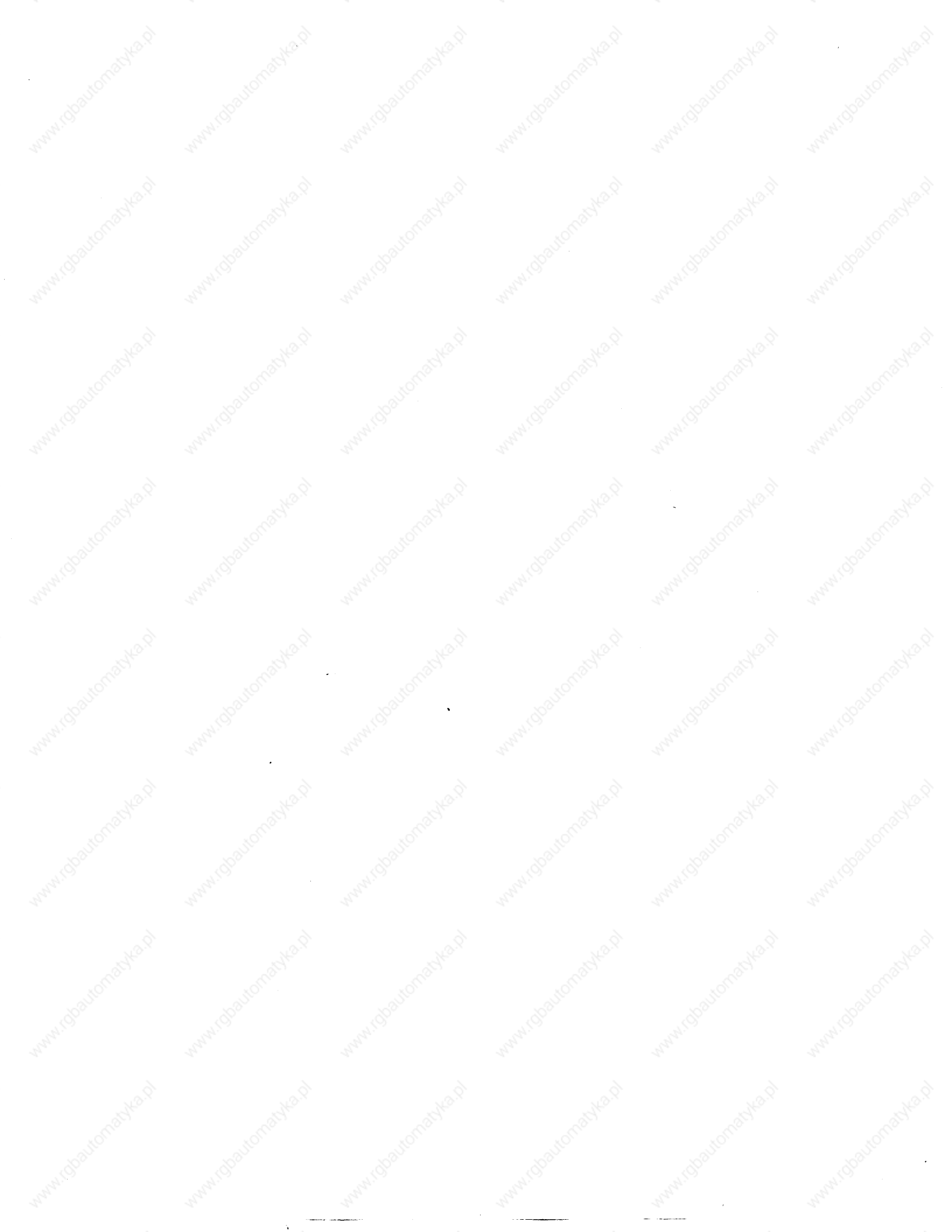
8.10

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.)

```

SUBTITLE 'JOB SCHEDULER (ROLLJOB)'
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
**
*
*           SUBROUTINE ROLLJOB
*
* PURPOSE:
*           TO MAKE A REQUEST OF THE DISK QUEUE MANAGER, EITHER TO COPY
*           A JOB OUT ONTO ITS ROLLOUT DATASET OR TO READ THE JOB'S IMAGE
*           BACK INTO MEMORY.
*
* ENTRY:
*           A4 = JXT-ENTRY ADDRESS.
*           A5 = JTA ADDRESS.
*           DNP (PROCESSING DIRECTION IN THE ROLLFILE'S DNT) IS ALREADY
*           SET -- TO 0 IF ROLLING IN, OR TO 1 IF ROLLING OUT.
*
* EXIT:
*           I/O IS IN PROGRESS.
*
* REGISTERS:
*           (A0-A2), (A6-A7), (S0-S2), (S6-S7) ARE DESTROYED
**
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
ROLLJOB = *
A7      W@JX:DNT,A4
A6      W@JXCJS,A4
S1      A5          SET UP THE DNT:
PUT,S1  S6&S7,INBUF,A7  BUFFER ADDRESS = JTA ADDRESS,
S2      A6          NUMBER OF BLOCKS = JOB SIZE/S12.
S2      S2>D'9
PUT,S2  S6&S7,DNNBK,A7
S1      A4          SUBMIT THE I/O REQUEST:
S1      S1<D'40      LEFT-ADJUST THE JXT ADDRESS.
S2      A7
S1      S1!S2       INSERT THE DNT ADDRESS.
A1      JSHID,0
A2      DQMID,0
S2      TRANSFER
J       PUTREQ      LET PUTREQ RETURN TO THE CALLER.

```



## 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 (LIB@INIT)'

\*\*\*\*\*

\*\*

\*

\*

\*

\*

\* 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).  
 \* A2 = (IF S0 IS NONZERO) THE OFFSET OF THE DESIRED MST ENTRY.  
 \* S2 = (IF S0 IS NONZERO) THE MST ENTRY ITSELF.

\* REGISTERS:

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

\*\*

\*\*\*\*\*

LIB@INIT =	*		
LIB1	A2	-1	LET A2=0 TO FETCH THE FIRST MST ENTRY.
	=	*	
	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>D'64-N@MSTYPE	
	A1	S2	A1 = MSADDR.
	A0	A1-A3	COMPARE MSADDR TO THE CALLER'S ADDRESS.
	JSZ	LIB1	RELOOP IF THIS SEGMENT ISN'T ALLOCATED.
	JAM	LIB1	LOOP UNTIL THE DESIRED SEGMENT IS FOUND.
LIB2	=	*	
	J	B0	RETURN.

S001.457  
 S001.458  
 S001.459  
 S001.460  
 S001.461  
 S001.462  
 S001.463  
 S001.464  
 S001.465  
 S001.466  
 S001.467  
 S001.468  
 S001.469  
 S001.470  
 S001.471  
 S001.472  
 S001.473  
 S001.474  
 S001.475  
 S001.476  
 S001.477  
 S001.478  
 S001.479  
 S001.480  
 S003.1071  
 S001.482  
 S001.483  
 S001.484  
 S001.485  
 S001.486  
 S001.487  
 S001.488  
 S001.489  
 S001.490  
 S001.491  
 S001.492  
 S001.493  
 S001.494  
 S001.495  
 S001.496  
 S001.497  
 S001.500







	JSZ	MVM4	DO ONLY THE FINAL PASS IF S3<64.	S002.1303
	JAP	MVM6	MOVE FROM THE TOP DOWN IF A3>A5.	S.20531
MVM2	=	*		S.20532
	VL	A1	SET VL TO 64 (OR, ON FINAL PASS, TO RESIDUE).	S.20533
	S1	A1		S.20534
MVM3	=	*		S.20535
	A0	A5	SET ADDRESS FOR VECTOR LOAD.	S.20536
	S3	S3-S1	S3 = WORDS STILL TO BE MOVED AT LOOP'S END.	S.20537
	A5	A5+A1	ADVANCE VECTOR-LOAD POINTER.	S.20538
	V0	,A0,1	LOAD.	S.20539
	A0	A3	SET ADDRESS FOR VECTOR STORE.	S.20540
	S0	S3-S1	S0 WILL BE NEGATIVE WHEN S3<64 (OR, ON FINAL	S.20541
	A3	A3+A1	PASS, WHEN S3=0).	S.20542
	,A0,1	V0	STORE.	S.20543
	JSP	MVM3	RELOOP WHILE S3 >= 64.	S.20544
MVM4	=	*		S.20545
	S0	S3	0 <= S3 < 64: WE ARE DONE IF S3=0.	S.20546
	A1	S3		S.20547
	JSN	MVM2	IF S3>0, GO BACK TO MAKE A FINAL PASS.	S.20548
	J	MVM3	ELSE RETURN.	S.20549
*				S.20550
			HERE IF MOVING FROM THE TOP DOWN INSTEAD OF FROM THE BOTTOM UP.	S.20551
MVM6	=	*		S.20552
	A1	S3	START A5 AND A3 AT THEIR HIGHEST VALUES.	S.20553
	A5	A5+A1		S.20554
	A3	A3+A1		S.20555
	A1	ZS0	RESTORE A1=64.	S.20556
MVM7	=	*		S.20557
	VL	A1	SET VL TO 64 (OR, ON FINAL PASS, TO RESIDUE).	S.20558
	S1	A1		S.20559
MVM8	=	*		S.20560
	A0	A5-A1	SET ADDRESS FOR VECTOR LOAD.	S.20561
	S3	S3-S1	S3 = WORDS STILL TO BE MOVED AT LOOP'S END.	S.20562
	A5	A5-A1	BACK UP VECTOR-LOAD POINTER.	S.20563
	V0	,A0,1	LOAD.	S.20564
	A0	A3-A1	SET ADDRESS FOR VECTOR STORE.	S.20565
	S0	S3-S1	S0 WILL BE NEGATIVE WHEN S3<64 (OR, ON FINAL	S.20566
	A3	A3-A1	PASS, WHEN S0=0).	S.20567
	,A0,1	V0	STORE.	S.20568
	JSP	MVM8	RELOOP WHILE S3 >= 64.	S.20569
	S0	S3	0 <= S3 < 64: WE ARE DONE IF S3=0.	S.20570
	A1	S3		S.20571
	JSN	MVM7	IF S3>0, GO BACK TO MAKE A FINAL PASS.	S.20572
MVM9	=	*		S.20573
	A3	B10		S.20574
	A5	B11		S.20575
	J	B0	RETURN.	S.20576



## 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 CONTENDS FOR MEMORY
- JOB MEMORY IS ERASED WHEN JOB INITIATED (I@ERASE)
- JOB IN MEMORY CONTENDS 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:

Location	Result	Operand
	A1	calling task's ID
	A2	JSHID,0
	S1	function code (already shifted)
	S2	job's offset in JXT
	S1	S1!S2
	S2	address if any
	S2	S2<D'16
	S1	S1!S2
	S2	auxiliary information if any
	S2	S2<D'40
	S1	S1!S2
	R	TSKREQ or PUTREQ

Input register format:

	00	24	48	53	63
INPUT+0	AUX		ADDR		JXO
INPUT+1	unused				

**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 J\$XXXX, selected from table

**JXO** JXT offset for the job in question.

Output register format:

	00	24	48	53	63
OUTPUT+0	STATUS				
OUTPUT+1	AUX		ADDR		JXO

**STATUS** Status of requested function.

0 Requested function completely accomplished.

≠0 Error or system is unable to fulfill request completely.



PARTIAL JOB SCHEDULER FUNCTIONS

FUNCTIONS:  
JSREQTAB

Function Code	Input Parameters	Function
—no input required—		Fill up JXT with jobs from SDT.
J\$ABORT	JXO, CODE	Abort a job.
J\$DELETE	JXO	Release all space allocated to a job.
J\$RERUN	JXO	Same as J\$DELETE but reinitiate a job.
J\$ALLOC	JXO, ADDR	Allocate or release memory for a job.
J\$IOSUSP	JXO	Suspend a job until an I/O request is done.
J\$IODONE	JXO	Resume an I/O suspended job.
J\$DELAY	JXO, ADDR	Suspend a job for a given time.
J\$DELAYK	JXO, ADDR	Same as J\$DELAY but keep the job in memory.
J\$AWAIT	JXO, ADDR	Suspend a job until a given event occurs.
J\$SUSP	JXO	Suspend a job momentarily; system initiated.
J\$SUSPK	JXO	Same as J\$SUSP but keep the job in memory.
J\$REMK	JXO	Lift the "keep in memory" restriction.
J\$RESUME	JXO	End momentary suspension.
J\$STOP	JXO	Suspend a job indefinitely; operator action.
J\$STPALL	JXO	Suspend all jobs indefinitely; operator action.
J\$START	JXO	End an indefinite suspension for a job; operator action.
J\$STRALL	JXO	End an indefinite suspension for all jobs; operator action.
J\$CLEAR	JXO	Force the end of a job's suspension.
J\$INDEX	JXO	Mark the job irrecoverable.
J\$SHTDWN	JXO	Idle down job activity in preparation for a system interruption.
J\$RCVR	JXO	Lift the suspension from jobs suspended by a J\$SHTDWN or system interruption.





## 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

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

NO ROLL  
NO PUSH



## CPU SWAPPING

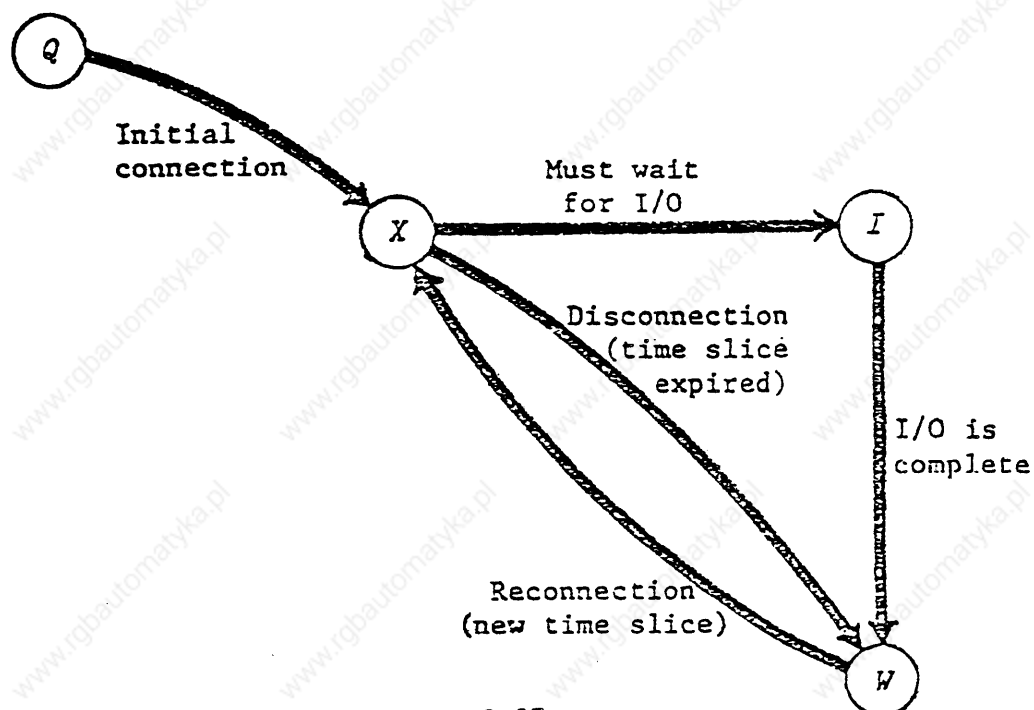
Q → X      A JOB IS QUEUED IN THE JOB EXECUTION TABLE (JXT) WAITING MEMORY. THE JOB IS BROUGHT INTO MEMORY AND BEGINS EXECUTION

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      JSH RECONNECTS THE WAITING JOB BECAUSE OF ITS HIGH PRIORITY

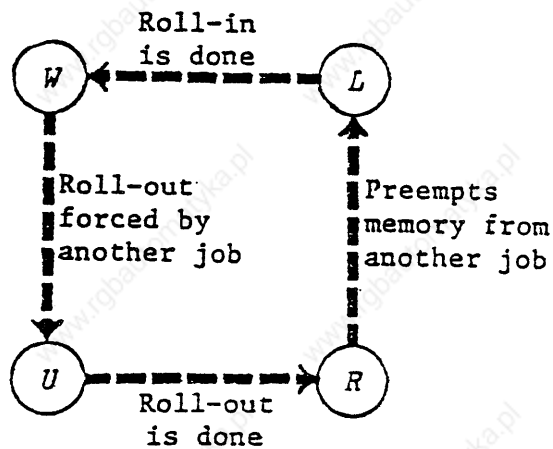
X → W      THE JOB'S TIME SLICE EXPIRES. THE CPU IS AVAILABLE FOR USE BY ANOTHER JOB.





## 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.



## 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

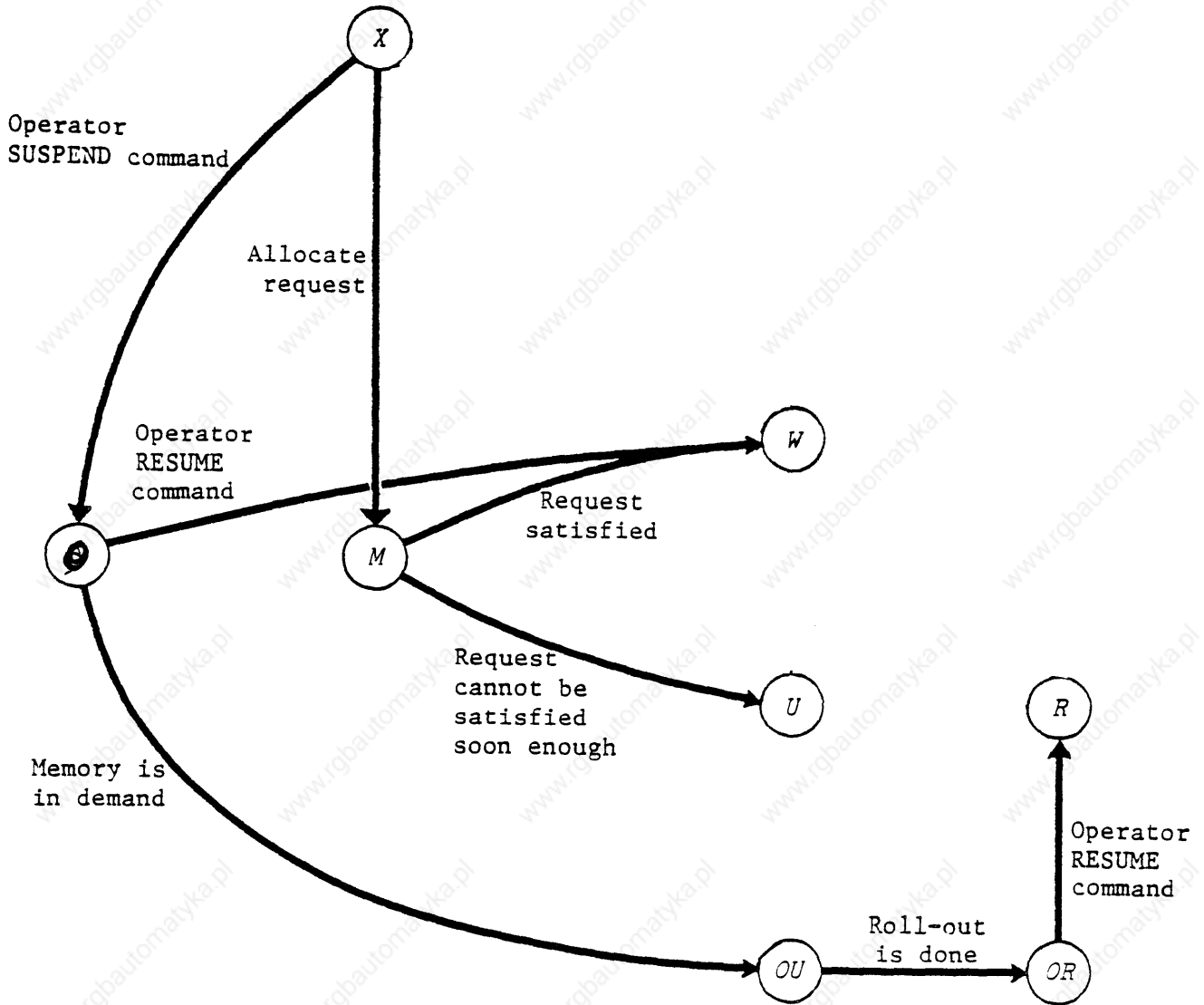
X → O X → S THE EXECUTING JOB REQUESTS SUSPENSION. JOB IS DISCONNECTED FROM CPU AND LIABLE TO BE ROLLED OUT.

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.

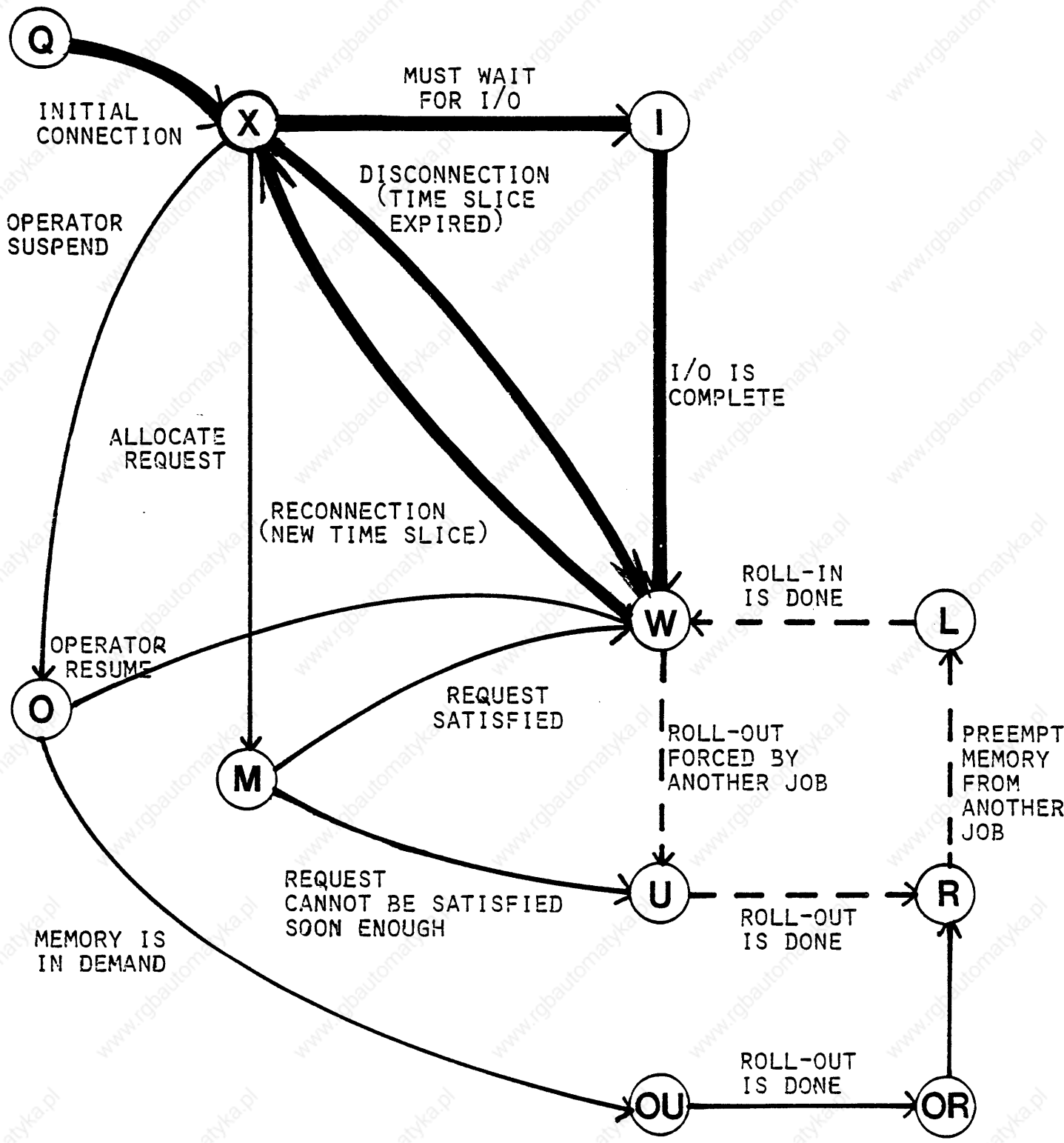
OR → R SR → R THE SUSPENDED JOB IS REACTIVATED.

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











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 67<sub>8</sub>.



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



## USER NORMAL EXIT

- USER ISSUES A SYSTEM ACTION REQUEST WHICH SETS S 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 SO-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 67<sub>g</sub>) AND THE USERS SO REGISTER TO DETERMINE REQUEST







```

0136000 0411252465440000000000 000000000063551645625 0000000000023663324313 000000000000073314630 BUSY      K
0136004 000000000000000000000055 0000000000000000000007 0000000000000000000000 0000000000000000000000
      LOCATIONS 00136010 THROUGH 00136017 CONTAIN 0000000000000000000000
0136020 000000000104100000212 0000000014500000000000 0000000030301600000000 0000001500000000000000  !          4
0136024 000000000000000000000000 0000000000000000000000 0000000000000000000000 000000000000000000213
0136030 000000000000000000000000 0000000000000000000000 0000000000000000000000 0000000000000000000000
0136034 000000000000000000000000 0000000000000000000000 0000000000000000000000 0000000000000000000001
0136040 177777777777777777777777 0000000000000000000000 0000000000000000000000 0000000000000000000000
      LOCATIONS 00136044 THROUGH 00136057 CONTAIN 0000000000000000000000
0136060 000000000000000000000000 0000000000000000000000 0000000000000000000000 000000000063551645625      K
0136064 000000000000000000000000 0425322255254201321403 00000000000000000324715 0000000000000000000000      EZJ
0136070 000000000000000000000000 0000000000000000000000 0000000000000000000000 0421072104340000036026      DGDG <
0136074 0475202125110125047522 0400615063140064344400 0400505667546174000000 0000040000000000000000 OPERATOR@1 3 @ ( 1
0136100 00000000000000000001020 0000000000000000000213 0000000000000000000000 0000000000000000000000
      LOCATIONS 00136104 THROUGH 00137317 CONTAIN 0000000000000000000000
0137320 022103246000000000000001 00000200000000000002000 0000000000000100002000 00000000000000000002010 $CS
0137324 0000000000000000000003000 10000000000000000002011 0000000000000000000000 0000400000000000000000
0137330 10000000000000000002007 0000000000000000000000 0000000033651400117660 0000000000000000000000      L
0137334 0000000000000000000352545 0000000012240600036026 0020000000700500007017 1000000000000000000000      <
0137340 02211423643400000000001 00000100000000000003000 00000000000000000003152 00000000000000000003000 $LOG
0137344 000000000000000000004000 10400000000000000003151 0000000000000000000000 0000000000000000000000
0137350 10400000000000000003140 0000000000000000000000 0000000035636000000712 0000000000000000000000
0137354 00000000000000000000000 0000000012242400036026 0000000000000000000000 0000000000000000000000      <
0137360 04610424400000000000000 0100000000000000000200 0000000000001100000200 0000000000001000010450 LDR @ ( (
0137364 00000000000000000011200 1000000100001000010450 0000000000000000000000 0000000000000000000000
0137370 0000000100001000010447 0000000000000000000000 0000000033630700125200 0000000000000000000000
0137374 00000000000000000352615 0000000011742000036026 0062000001515000017414 0221202044700000000000      < $PBN
0137400 0000000000000000000000 0000000000000000000000 0000000000000000000000 0000000000000000000000
      LOCATIONS 00137404 THROUGH 00137417 CONTAIN 0000000000000000000000
0137420 04610424400000000000000 00000024641140100021140 0000040000000500132200 1000200000000000000000 LDR SC ""
0137424 0000000035213400001360 0000000000000000000000 0000000100000001100004 00000000000000000000011
0137430 0000000001100000116100 0000000000000000000000 0000000000000000000000 0000000000000000000000      e
0137434 00000000000000000000000 0000000000000000000000 0000000000000000000000 0000000000000000000000
0137440 00000000000000000343413 00000000000000000116000 0000000000000000036026 00000000000000000333175      <
0137444 0000000000000000004520 00000000000000000122000 0000000000000000004000 0000000000000077777777 P
0137450 00000000000000000000000 0000000000000000000000 0007070260011751604001 0000000000000000000001      0
0137454 0000010000000000000000 0377777777777777700000 0000000000000000000000 00000000000000000333175      ?
0137460 00000000000000000000000 0000000000000000000000 0000000000000000000000 0000000000000000000000
      LOCATIONS 00137464 THROUGH 00137657 CONTAIN 0000000000000000000000
0137660 04610424427000000000000 0000000000000000000000 0000000000000000000000 0000000000000000000000 LDR..
      LOCATIONS 00137664 THROUGH 00137673 CONTAIN 0000000000000000000000
0137674 0000000001451600136000 0000030012200000121600 00000600000000000121564 0000000000000000000000      N

```





0142554 0000000000000100000000 0000000520000000001000 0421041323047113231463 0000000001100000000001  
 0142560 0037100000000000000000 0000000000000000000000 0000000000000000000000 0000000000000000000000  
 0142564 0000000000000000000000 0000000000000000000000 0000000000000000000000 0000000000000000000000  
 0142570 0000000000000000000000 0000000000000000000000 0000000000000000000000 10000007070700000022  
 0142574 1000000707070700000022 0004000000000100000000 0000000000000100000000 0000000520000000010000  
 0142580 0421041323047113231463 0000000001100000000001 0037110000000000000000 0000000000000000000000 DD-19-33  
 LOCATIONS 00142604 THROUGH 00142613 CONTAIN 0000000000000000000000  
 0142614 0000000000000000000000 10000007070700000022 000000070707000002162 0000000000000000000000  
 LOCATIONS 00142620 THROUGH 00144773 CONTAIN 0000000000000000000000  
 0144774 0000000000000000000000 0000000000000000000000 0000000000000000000000 000000070707000002162

DD-19-33

DD-19-33

\*\*\* DMEM.FWA=303000,L=7000.

```

0303000 041522202515100000000 0000000000000000072535 0000000000000217262754 0000000000000006340325 CRASH
0303004 000000000000000000004 00000000000000000003 00000000000000000000 00000000000000000000
LOCATIONS 00303010 THROUGH 00303017 CONTAIN 00000000000000000000
0303020 0000000000133300000000 0000000003120000000021 00000000045001600000001 000000150210000000000
0303024 000000000000000000000 0000000000000000000657 000000000000000000020 000000000000000000006
0303030 0000000000000000000017 00000000000000000020 000000000000000000000 000000000000000000000
0303034 020000000000000000000 00000000000000000017 00000000000000000040 000000000000000000176
0303040 000200000000000000000 00000000000000000000 00000000000000000000 00000000000000000000
LOCATIONS 00303044 THROUGH 00303057 CONTAIN 00000000000000000000
0303060 000000000000000000000 00000000000000000000 00000000000000000000 000000000000000006657
0303064 000000000000000000000 00000000000000000000 00000000000000000324715 00000000000000000343636
0303070 000000000000000000000 00000000000000000001 00000000000000000001 0421072104340000036102
0303074 0475202125110125047522 040037532140540000000 040045505105004400000 000004000000000000000 OPERATOR@
0303100 0000000000000000004561 00000000000000000200 000000000000000001074 00000000000000000000
LOCATIONS 00303104 THROUGH 00303167 CONTAIN 00000000000000000000
0303170 000000000000000000000 00000000000000000000 00000000000000000000 0000000000000000001616
0303174 000000000000000000000 00000000000000000000 00000000000000000000 00000000000000000000
LOCATIONS 00303200 THROUGH 00303277 CONTAIN 00000000000000000000
9.10 300 0425162504252200000000 0124120644252200000000 0060122504252200000000 0000122504252200000000 ENTER ER TER TER
304 0135162504252200000000 0111162504252200000000 0041162504252200000000 0100372504252200000000 NTER NTER NTER TER
310 000000000000000000000 00000000000000000000 00000000000000000000 00000000000000000000
LOCATIONS 00303314 THROUGH 00303377 CONTAIN 00000000000000000000
0303400 042116000000000000000 050104234000000000000 044504000000000000000 042504000000000000000 DN PDN ID ED
0303404 051000000000000000000 053400000000000000000 046400000000000000000 052521000000000000000 R W M UQ
0303410 042516250425220000000 006014016060250240000 0074140420000101447000 0114060360002200042400 ENTER N E
0303414 011406042100352400000 011406050060350560000 011405036425232460000 011002052001232460000 S ESS SS
0303420 011024024104200660000 00000000000000000000 00000000000000000000 00000000000000000000
LOCATIONS 00303424 THROUGH 00303477 CONTAIN 00000000000000000000
0303500 100000000000000000000 00000000000000000000 00000000000000000000 030000000000000000000
0303504 000000000000000000000 00000000000000000000 00000000000000000000 000000000000000000000
0303510 000000000000000000000 03000000000000000000 00000000000000000000 000000000000000000000
LOCATIONS 00303514 THROUGH 00303577 CONTAIN 00000000000000000000
0303600 100000000000000000000 10000000000000000000 10000000000000000000 100000000000000000000
0303604 10000000000000000000 10000000000000000000 10000000000000000000 000000000000000000001
0303610 000000000000000000001 10000000000000000000 00000000000000000001 000000000000000000000
LOCATIONS 00303614 THROUGH 00303677 CONTAIN 00000000000000000000
0303700 0000000000000000000650 000000000000000000660 000000000000000000662 0000000000000000001230
0303704 000000000000000000673 00000000000000000674 00000000000000000675 0000000000000000001231
0303710 0000000000000000001232 00000000000000000651 000000000000000001642 000000000000000000000
LOCATIONS 00303714 THROUGH 00303777 CONTAIN 00000000000000000000
0304000 000000000000000000657 00000000000000000660 00000000000000000662 0000000000000000001230
0304004 000000000000000000673 00000000000000000674 00000000000000000675 0000000000000000001231

```

0304010 0000000000000000001232 0000000000000000000650 0000000000000000001642 0000000000000000000000  
 LOCATIONS 00304014 THROUGH 00304077 CONTAIN 0000000000000000000000

0304100 0000000000000000000657 0000000000000000000661 0000000000000000000662 0000000000000000001230  
 0304104 0000000000000000000673 0000000000000000000674 0000000000000000000675 0000000000000000001231  
 0304110 0000000000000000001232 0000000000000000000650 0000000000000000001642 0000000000000000000000  
 LOCATIONS 00304114 THROUGH 00304177 CONTAIN 0000000000000000000000

0304200 0405232464450723400000 0000000000000000000017 0421160000000000000000 000000000000000000007 ASSIGN  
 0304204 0221112340000000000000 0000000000000000000017 0401000000000000000000 000000000000000000007 \$IN  
 0304210 0431241403240000000000 0000000000000000000017 0461150000000000000000 000000000000000000007 FT05  
 0304214 0300000000000000000000 0000000000000000000037 0000000000000000000000 000000000000000000000 0

DN  
A  
LM

LOCATIONS 00304220 THROUGH 00304317 CONTAIN 0000000000000000000000

0304320 0221032460000000000001 0000000000000000000200 000000000000000000100002000 00000000000000000002012 \$CS  
 0304324 0000000000000000000300 10000000000000000002013 0000000000000000000000 0000200000000000000000  
 0304330 10000000000000000002011 0000000000000000000000 0000000033651400264660 0000000000000000000000  
 0304334 00000000000000000352545 0000000026740600036102 0020000000700500007017 1000000000000000000000  
 0304340 0221142364340000000001 00000100000000000003000 00000000000000000003130 00000000000000000003000 \$LOG  
 0304344 00000000000000000004000 10400000000000000003127 0000000000000000000000 00000000000000000000000

L  
<B  
W  
<B  
X

0304350 10400000000000000003120 0000000000000000000000 0000000035636600000712 0000000000000000000000 P  
 0304354 0000000000000000000000 0000000026742400036102 0000000000000000000000 0000000000000000000000  
 6 000 0415172405452200000000 1000000000000000000200 0000000000000000000200 1777777777777700000200 COPYR  
 11 001 00000000000000000004200 0000000000000000000000 0000000000000000000000 0000000000000000000000  
 370 0000000000000000000000 0000000000000000000000 0000000033662000007200 0000000000000000000000

COPYR

0304374 00000000000000000352545 17777777777777521200 0002000000720000007210 0000000000000000000000  
 LOCATIONS 00304400 THROUGH 00304437 CONTAIN 0000000000000000000000

0304440 0000000000000000000000 0000000000000000000263000 000000000000000000036102 000000000000000000272000  
 0304444 00000000000000000004520 000000000000000000335713 0000000000000000000010 0000000000000000000005  
 0304450 0000000000000000000000 0415172405452200000000 0415172405452200000000 0000000000000000000000  
 0304454 0000000000001400177700 1777777776000000000000 000000000000000000177700 0000000000001400177700  
 0304460 0000000000000000000000 0000000000000000000000 0000000000000000000000 0000000000000000000000

<B  
P  
COPYR COPYR

LOCATIONS 00304464 THROUGH 00304657 CONTAIN 0000000000000000000000

0304660 0415172405452213400000 0000000000000000000000 0000000000000000000000 0000000000000000000000 COPYR.  
 LOCATIONS 00304664 THROUGH 00304673 CONTAIN 0000000000000000000000

0304674 0000000000647500136000 0000020012600000125600 000004000000000125570 0000000000000000000000 =  
 LOCATIONS 00304700 THROUGH 00304737 CONTAIN 0000000000000000000000

0304740 00000000000000000334257 0000000000000000000000 0000000000000000000000 00000000000000000007643  
 0304744 00000000000000000267442 0000000000000000000000 0000000000000000000000 0000000000000000000000  
 LOCATIONS 00304750 THROUGH 00304777 CONTAIN 0000000000000000000000

COPYR.

=

"

0305000 0000000000000000000002 0451172042411223436503 0511012464405100000000 1030000000000000000003  
 0305004 0405032064252324624104 0470752064752026251054 0425162504252212200000 1020000000000000000001 ACCESS(DN=COPYR,ENTER)  
 0305010 0415172405452213400000 1020000000000000000001 0425302225205600000000 1030000000000000000001 COPYR. EXIT:  
 0305014 0425302225205600000000 1030000000000000000000 1600000000000000000000 1700000000000000000000 EXIT:  
 0305020 0000000000000000000000 0000000000000000000000 0000000000000000000000 0000000000000000000000

JOB(JN=CRASH)  
ACCESS(DN=COPYR,ENTER)  
COPYR. EXIT:  
EXIT:

LOCATIONS 00305024 THROUGH 00305777 CONTAIN 00000000000000000000

0306000	000000000000000000000016	0200401002004010020040	0310611643046216431062	0200401002004010030056				21:12:22	0
0306004	0300601403004010020040	0415232402004010020040	0200401002004010020040	0415222025445514220040	0000	CSP			CRAY-1
0306010	0515052444450123026461	0310401002004010020040	041522222005510046505	0471042365210110044105		SERIAL-12			CRI - MENDOTA HE
0306014	0445072205212313020115	0445162342704010020040	0300661363106113633471	1000000000000000000006		IGHTS, MINN.			06/21/79
0306020	0200401002004010020040	0310611643046216431062	0200401002004010030056	0300601403004010020040			21:12:22	0.0000	
0306024	0415232402004010020040	0200401002004010020040	1000000000000000000016	0200401002004010020040		CSP			
0306030	0310611643046216431062	0200401002004010030056	0300601403004010020040	0415232402004010020040	21:12:22		0.0000		CSP
0306034	0200401002004010020040	0415222025445514220040	0475202125110125044516	0434402465452325042515		CRAY-1 OPERATING SYSTEM			
0306040	0200401002004010020040	0415172462006113430066	0200402025152321246502	0461311004210125042440		COS 1.06 ASSEMBLY DATE			
0306044	0300661363106013633471	1000000000000000000006	0200401002004010020040	0310611643046216431062	06/20/79			21:12:22	
0306050	0200401002004010030056	0300601403004010020040	0415232402004010020040	0200401002004010020040	0.0000	CSP			
0306054	1000000000000000000006	0200401002004010020040	0310611643046216431062	0200401002004010030056			21:12:22	0.	
0306060	0300601403004010020040	0415232402004010020040	0200401002004010020040	1000000000000000000007	0000	CSP			
0306064	0200401002004010020040	0310611643046216431062	0200401002004010030056	0300601403004010020040			21:12:22	0.0000	
0306070	0415232402004010020040	0451172042411223436503	0511012464405110020040	1000000000000000000010		CSP			JOB (JN=CRASH)
0306074	0200401002004010020040	0310611643046216431062	0200401002004010030056	0300601403104010020040			21:12:22	0.0002	
0306100	0415232402004010020040	0405032064252324624104	0470752064752026251054	0425162504252212220040		CSP			ACCESS (DN=COPYR, EHTER)
0306104	1000000000000000000013	0200401002004010020040	0310611643046216431062	0200401002004010030056			21:12:22	0.	
0306110	0300601403144010020040	0501042322004010020040	0501041403006110026440	0405032064252324620040	0003	PDM			PD001 - ACCESS
0306114	0415172405452210020040	0200401002004010020040	0425041723006014031440	0415172325011421252105		COPYR			ED=0003 COMPLETE
0306120	1000000000000000000006	0200401002004010020040	0310611643046216431062	0200401002004010030056			21:12:22	0.	
0306124	0300601403144010020040	0415232402004010020040	0415172405452213420040	1000000000000000000000	0003	CSP			COPYR.
0306130	0000000000000000000000	0000000000000000000000	0000000000000000000000	0000000000000000000000					

LOCATIONS 00306134 THROUGH 00306777 CONTAIN 00000000000000000000

0307000	00177700000000000007000	0000000000000000000000	0000000000000000000000	0000000000000000000000					
---------	-------------------------	------------------------	------------------------	------------------------	--	--	--	--	--

LOCATIONS 00307004 THROUGH 00307403 CONTAIN 00000000000000000000

0307404	1000000707070700000016	0000000000000000004423	0221032460000000000001	0400002224700200021400					\$CS @ IN #
0307410	0000010000000000265000	1001600000000000000000	0100000035222400001320	0000000000000000000000					
0307414	0000000100000000100001	0000000000000000000000	0000000000000000000000	0000000000000000000000					
0307420	0000000000000000000000	1000000707070700000016	1000000707070700000016	000000000440500004441					!
0307424	0221142364340000000001	020000240510027773321	0000000000000000000000	0001600000000000000063	\$LOG	FR			3
0307430	0100000000000000000000	0000000000000000000000	0000000000000000000001	0000000000000000000000					
0307434	0000000000000000000063	0000000000000000000000	0000000000000000000000	1000000707070700000016		3			
0307440	1000000707070700000016	0000000000442300004501	0221112340000000000000	0610002464140200021400			A\$IN		SC #
0307444	000001000000000420000	1000200000000000000000	0000000000000000134640	0000000000000000000000					
0307450	000000010000000100004	0000000000000000000000	0000000000000000000000	0000000000000000000000					
0307454	0000000000000000000000	1000000707070700000016	1000000707070700000022	0000000000000000000000					
0307460	0000000000000100000000	0000000560000000000000	0421041323047113231463	0000000001100000000003					DD-19-33
0307464	003720017504076400000	0000000000000000000000	0000000000000000000000	0000000000000000000000					
0307470	0000000000000000000000	0000000000000000000000	0000000000000000000000	0000000000000000000000					
0307474	0000000000000000000000	0000000000000000000000	0000000000000000000000	1000000707070700000022					



```

0307500 1000000707070700000016 0000000000444100004517 0221172525200000000000 0600002405100200000000
0307504 0000000000000000000000 0001600000000000000000 00000000000000000000 00000000000000000000
0307510 00000000000000000000004 00000000000000000000 0000000000000000003720 00000000000000000000
0307514 0000000000000000000000 1000000707070700000016 1000000707070700000016 0000000000450100000000
0307520 0415172405452200000000 100000246414017773243 0000000000000000272200 1000200000000000000000 COPYR SC
0307524 004000000000000000001360 0000000000000000000000 0000000100000000200004 0000000000000000000004
0307530 0000000000400000116100 0000000000000000000000 0000000000000000000000 1000000707070700000016 e
0307534 1000000707070700000022 0004000000000200000000 0000007424000100035405 0000000520000000004000 @ ;
0307540 0421041323047113231463 0000000001100000000001 0020130000000000000000 0000000000000000000000 DD-19-33
      LOCATIONS 00307544 THROUGH 00307553 CONTAIN 0000000000000000000000
0307554 0000000000000000000000 1000000707070700000022 0000000707070700002222 0000000000000000000000
      LOCATIONS 00307560 THROUGH 00311773 CONTAIN 0000000000000000000000
0311774 0000000000000000000000 0000000000000000000000 0000000000000000000000 0000000707070700002222

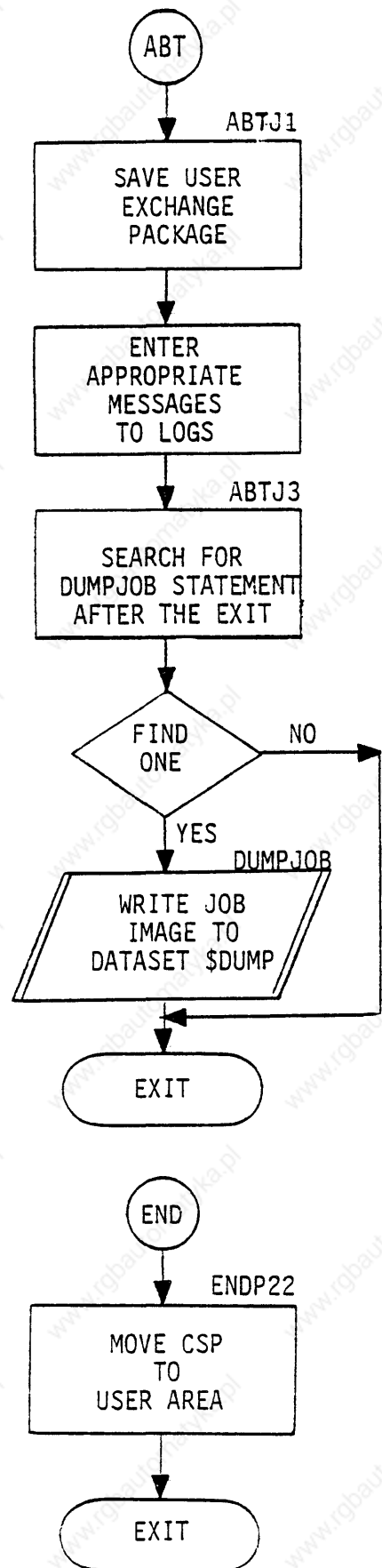
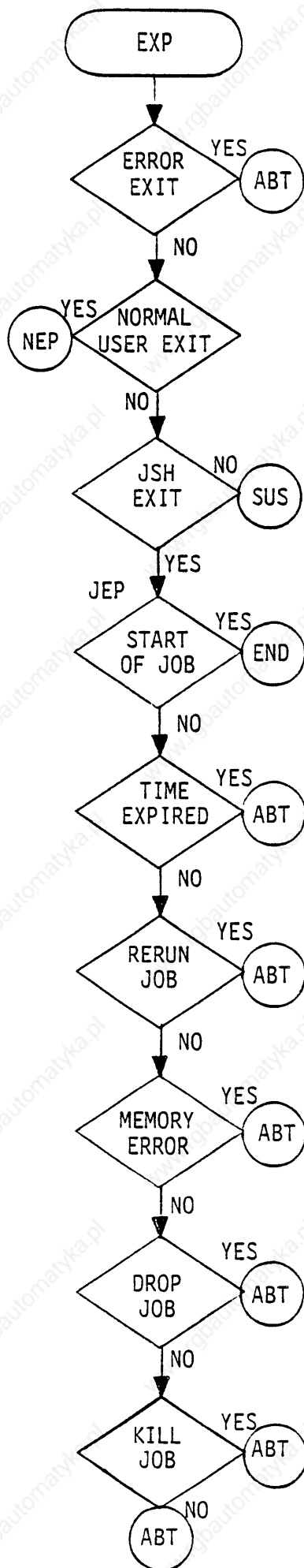
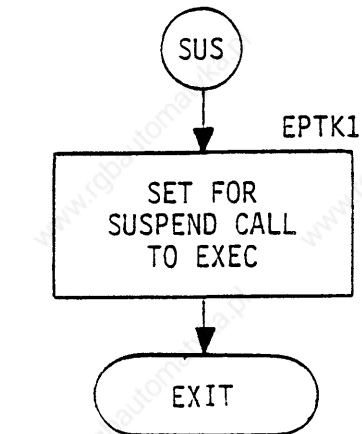
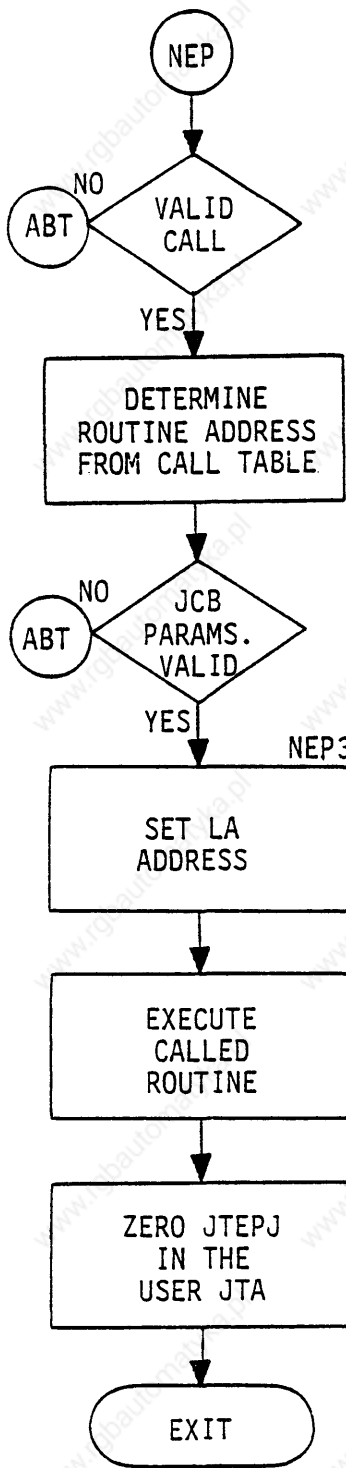
```



## 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.

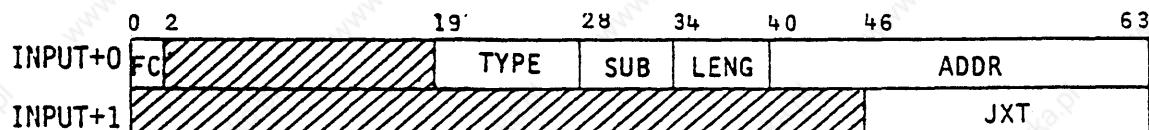


## SYSTEM LOG PROCESSING

- 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:

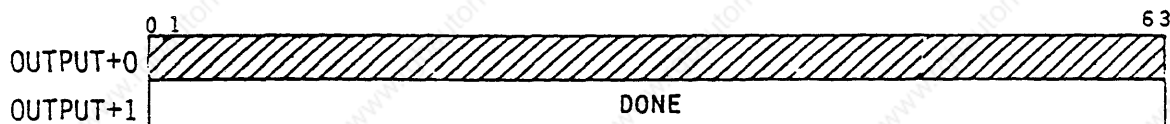
(A1) ID of task requesting message entry  
 (A2) MSGID,0  
 (S1) INPUT+0 } request  
 (S2) INPUT+1 }

Request format:



<u>Field</u>	<u>Word</u>	<u>Bits</u>	<u>Description</u>
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:



<u>Field</u>	<u>Word</u>	<u>Bits</u>	<u>Description</u>
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:

Location	Result	Operand
	MESSAGE	<i>address, loc</i>

*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)

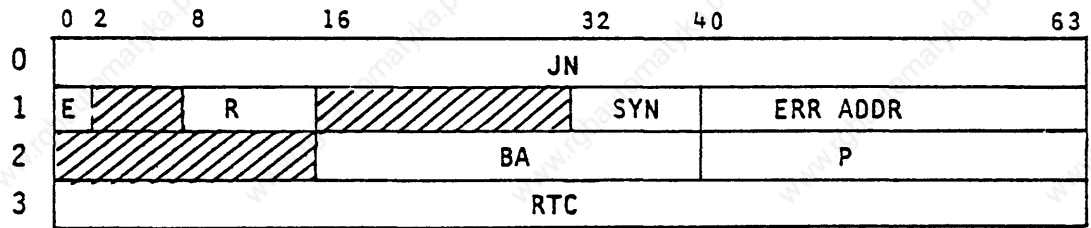
DISK ERROR CORRECTION (DEC)





## MEMORY ERROR PROCESSOR (MEP)

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

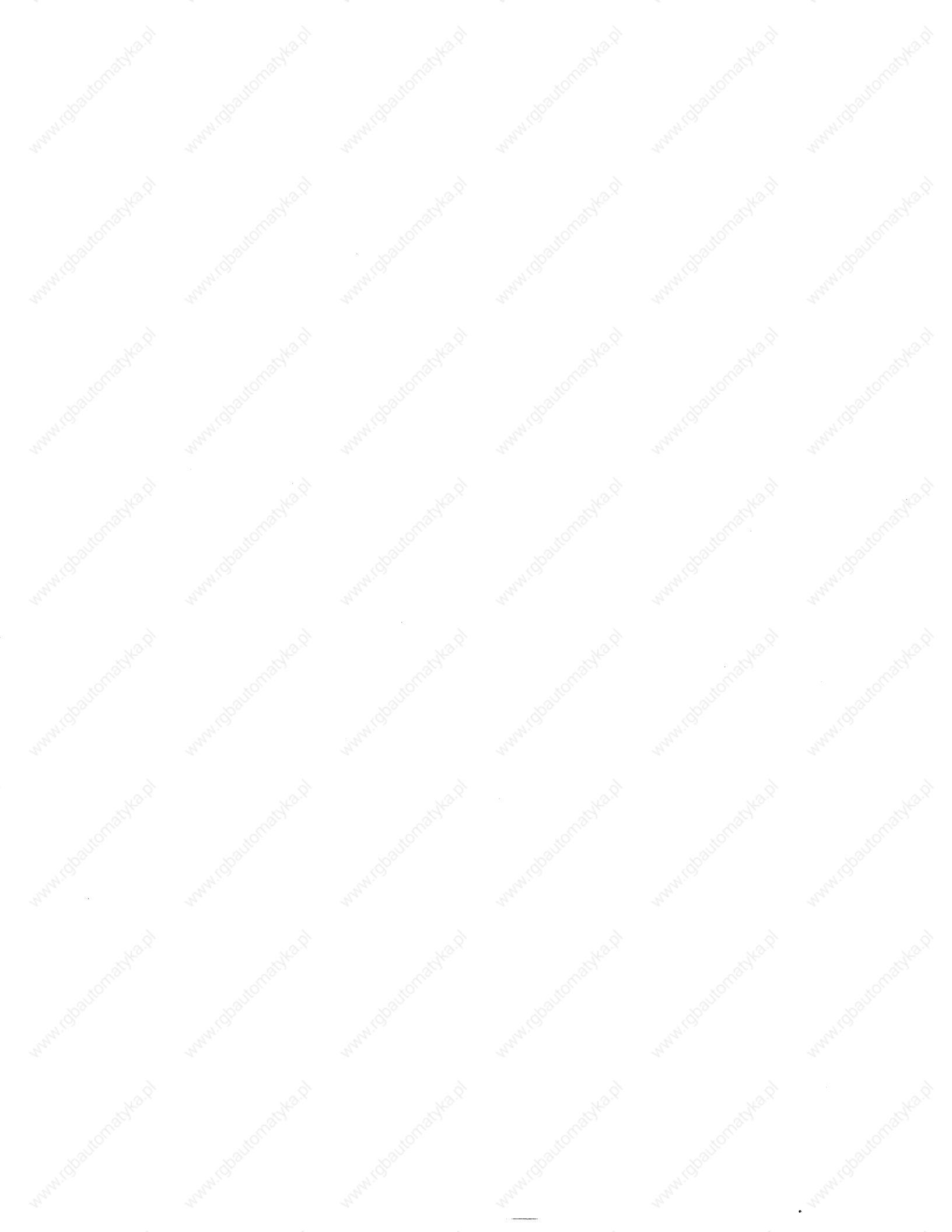


<u>Field</u>	<u>Word</u>	<u>Bits</u>	<u>Significance</u>
JN	0	0-63	Jobname or 'STP'
E	1	0,1	Error type (binary): 10 Uncorrectable memory error 01 Correctable memory error
R	1	8-15	Read mode: 0 Scalar 1 I/O 2 Vector 3 Fetch
SYN	1	32-39	Syndrom bits
ERR ADDR	1	40-63	Error address
BA	2	26-39	Base address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



## 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 (I@SPMDLY)
- 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





11:12:02.9094

```
EXECUTIVE REQUESTS          TIME INTERVAL = 60012.18 MILLISECONDS
NUMBER OF TASK 0 REQUESTS = 0
NUMBER OF TASK 1 REQUESTS = 313
NUMBER OF TASK 2 REQUESTS = 1370
NUMBER OF TASK 3 REQUESTS = 266
NUMBER OF TASK 4 REQUESTS = 0
NUMBER OF TASK 5 REQUESTS = 715
NUMBER OF TASK 6 REQUESTS = 576
NUMBER OF TASK 7 REQUESTS = 0
NUMBER OF TASK 8 REQUESTS = 21
NUMBER OF TASK 9 REQUESTS = 2239
NUMBER OF TASK 10 REQUESTS = 0
```

11:12:02.9096

```
TASK UTILIZATION          TIME INTERVAL = 60012.18 MILLISECONDS
NUMBER OF TASK 0 READIES = 0
NUMBER OF TASK 1 READIES = 129
NUMBER OF TASK 2 READIES = 488
NUMBER OF TASK 3 READIES = 86
NUMBER OF TASK 4 READIES = 0
NUMBER OF TASK 5 READIES = 297
NUMBER OF TASK 6 READIES = 156
NUMBER OF TASK 7 READIES = 0
NUMBER OF TASK 8 READIES = 3
NUMBER OF TASK 9 READIES = 410
NUMBER OF TASK 10 READIES = 0
```

11:12:02.9099

```
CPU UTILIZATION          TIME INTERVAL = 60012.18 MILLISECONDS
USER TIME = 60134.40 MSEC 100.20%
IDLE TIME = 0.00 MSEC 0.00%
BLOCKED TIME = 0.00 MSEC 0.00%
EXEC TIME = 151.90 MSEC 0.25%
TASK 0 TIME = 0.00 MSEC 0.00%
TASK 1 TIME = 16.42 MSEC 0.03%
TASK 2 TIME = 25.84 MSEC 0.04%
TASK 3 TIME = 4.39 MSEC 0.01%
TASK 4 TIME = 0.00 MSEC 0.00%
TASK 5 TIME = 14.25 MSEC 0.02%
TASK 6 TIME = 17.61 MSEC 0.03%
TASK 7 TIME = 0.00 MSEC 0.00%
TASK 8 TIME = 0.41 MSEC 0.00%
TASK 9 TIME = 48.38 MSEC 0.03%
TASK 10 TIME = 0.00 MSEC 0.00%
TIME NOT ACCOUNTED FOR = -402.02 MSEC -0.67%
```

----- END OF EXTRACT REPORT

43892 RECORDS READ FROM #SYSLOG

49 RECORDS WRITTEN ON #OUT

Listing of all subtypes of SPM records

11:12:02.9087

CHANNEL INTERRUPT COUNTS

CHANNEL 0 =	0
CHANNEL 1 =	0
CHANNEL 2 =	118
CHANNEL 3 =	116
CHANNEL 4 =	611
CHANNEL 5 =	684
CHANNEL 6 =	1819
CHANNEL 7 =	1090
CHANNEL 8 =	0
CHANNEL 9 =	0
CHANNEL 10 =	0
CHANNEL 11 =	0
CHANNEL 12 =	0
CHANNEL 13 =	0
CHANNEL 14 =	0
CHANNEL 15 =	0
CHANNEL 16 =	0

TIME INTERVAL = 60012.18 MILLISECONDS

CHANNEL 17 =	0
CHANNEL 18 =	0
CHANNEL 19 =	0
CHANNEL 20 =	0
CHANNEL 21 =	0
CHANNEL 22 =	0
CHANNEL 23 =	0
CHANNEL 24 =	0
CHANNEL 25 =	0
CHANNEL 26 =	5627
CHANNEL 27 =	0
CHANNEL 28 =	0
CHANNEL 29 =	0
CHANNEL 30 =	486
CHANNEL 31 =	0
CHANNEL 32 =	0
CHANNEL 33 =	0

11:12:02.9090

EXEC CALL USAGE

NUMBER OF TYPE 0 CALLS =	0
NUMBER OF TYPE 1 CALLS =	0
NUMBER OF TYPE 2 CALLS =	905
NUMBER OF TYPE 3 CALLS =	1172
NUMBER OF TYPE 4 CALLS =	0
NUMBER OF TYPE 5 CALLS =	104
NUMBER OF TYPE 6 CALLS =	370
NUMBER OF TYPE 7 CALLS =	0
NUMBER OF TYPE 8 CALLS =	24
NUMBER OF TYPE 9 CALLS =	176
NUMBER OF TYPE 10 CALLS =	24
NUMBER OF TYPE 11 CALLS =	0
NUMBER OF TYPE 12 CALLS =	0
NUMBER OF TYPE 13 CALLS =	108
NUMBER OF TYPE 14 CALLS =	307
NUMBER OF TYPE 15 CALLS =	307
NUMBER OF TYPE 16 CALLS =	1997

TIME INTERVAL = 60012.18 MILLISECONDS

NUMBER OF TYPE 17 CALLS =	0
NUMBER OF TYPE 18 CALLS =	0
NUMBER OF TYPE 19 CALLS =	0
NUMBER OF TYPE 20 CALLS =	0
NUMBER OF TYPE 21 CALLS =	0
NUMBER OF TYPE 22 CALLS =	0
NUMBER OF TYPE 23 CALLS =	0
NUMBER OF TYPE 24 CALLS =	0
NUMBER OF TYPE 25 CALLS =	0
NUMBER OF TYPE 26 CALLS =	0
NUMBER OF TYPE 27 CALLS =	0
NUMBER OF TYPE 28 CALLS =	1
NUMBER OF TYPE 29 CALLS =	1
NUMBER OF TYPE 30 CALLS =	1
NUMBER OF TYPE 31 CALLS =	1
NUMBER OF TYPE 32 CALLS =	1

11:12:02.9092

DISK CHANNEL UTILIZATION

CHANNEL 0 =	0.00 MSEC	0.00%
CHANNEL 1 =	0.00 MSEC	0.00%
CHANNEL 2 =	2539.73 MSEC	4.27%
CHANNEL 3 =	1426.73 MSEC	2.32%
CHANNEL 4 =	0.00 MSEC	0.00%
CHANNEL 5 =	0.00 MSEC	0.00%
CHANNEL 6 =	0.00 MSEC	0.00%
CHANNEL 7 =	0.00 MSEC	0.00%
CHANNEL 8 =	0.00 MSEC	0.00%
CHANNEL 9 =	0.00 MSEC	0.00%
CHANNEL 10 =	0.00 MSEC	0.00%
CHANNEL 11 =	0.00 MSEC	0.00%

TIME INTERVAL = 60012.18 MILLISECONDS

Listing of all subtypes of SPM records (continued)

08:54:24.8568

MEMORY UTILIZATION

TIME INTERVAL = 300006.12 MILLISECONDS

EXECUTE MEMORY INTEGRAL = 0.10355550E+02 MWS 8.944 % UTILIZATION  
 I/O WAIT MEMORY INTEGRAL = 0.13730810E+02 MWS 11.950 % UTILIZATION  
 CPU WAIT MEMORY INTEGRAL = 0.28333448E+02 MWS 24.473 % UTILIZATION  
 AVAILABLE MEMORY INTEGRAL = 0.11577656E+03 MWS

08:54:24.8570

DISK UTILIZATION

TIME INTERVAL = 300006.12 MILLISECONDS

LDU	RATE	%SEEK	%TRANS	% TOTAL	# REQS	# ON-CYL	% FREE	% PERM	BLKS/REQ
DD-19-20	0.103957 MWS/S	38.34	29.50	67.84	5348	773	6.76	0.00	11.38
DD-19-30	0.064985 MWS/S	9.09	14.75	23.85	3476	290	12.26	0.00	10.95
DD-19-40	0.104673 MWS/S	11.78	27.04	38.82	3203	376	23.22	0.00	13.12
DD-19-50	0.000000 MWS/S	0.00	0.00	0.00	0	0	100.00	0.00	0.00
DD-29-21	0.032098 MWS/S	9.72	7.91	17.63	1533	266	56.03	0.00	12.22
DD-19-31	0.040775 MWS/S	9.40	11.62	21.03	2426	289	32.09	0.00	9.85
DD-19-41	0.007004 MWS/S	8.39	4.73	13.12	2183	713	20.15	0.00	1.88
DD-19-51	0.000000 MWS/S	0.00	0.00	0.00	0	0	100.00	0.00	0.00
DD-19-22	0.000000 MWS/S	0.00	0.00	0.00	0	0	100.00	0.00	0.00
DD-19-32	0.000000 MWS/S	0.00	0.00	0.00	0	0	100.00	0.00	0.00
DD-19-42	0.000000 MWS/S	0.00	0.00	0.00	0	0	100.00	0.00	0.00
DD-19-52	0.000000 MWS/S	0.00	0.00	0.00	0	0	100.00	0.00	0.00
DD-19-23	0.000000 MWS/S	0.00	0.00	0.00	0	0	100.00	0.00	0.00
DD-19-33	0.000000 MWS/S	0.00	0.00	0.00	0	0	100.00	0.00	0.00
DD-19-43	0.000000 MWS/S	0.00	0.00	0.00	0	0	100.00	0.00	0.00
DD-19-53	0.000000 MWS/S	0.00	0.00	0.00	0	0	100.00	0.00	0.00

0.053392 MWS/S

71.61 0.00

08:54:24.8577

LINK UTILIZATION

TIME INTERVAL = 300006.12 MILLISECONDS

LINK	TOTAL #	GROSS RATE	-OUTPUT DATA NET RATE--	--INPUT DATA NET RATE--
IDNT	MESSAGES	(MBITS/S)	WORDS SENT (MBITS/S)	WORDS RCVD (MBITS/S)
MS	152	0.001105	3239	0.000704
US	1372	0.014410	34197	0.007295
IC	2918	0.009560	4903	0.001046
EB	0	0.000000	0	0.000000
DG	796	0.003017	4496	0.000957
JB	63	0.000220	264	0.000056

Listing of all subtypes of SPM records (continued)

```

08:54:24.8591      USER CALL USAGE                      TIME INTERVAL =    300006 12 MILLISECONDS
NUMBER OF TYPE 0 CALLS =                      0                      NUMBER OF TYPE 25 CALLS =                      0
NUMBER OF TYPE 1 CALLS =                      1                      NUMBER OF TYPE 26 CALLS =                      0
NUMBER OF TYPE 2 CALLS =                      92                     NUMBER OF TYPE 27 CALLS =                      8
NUMBER OF TYPE 3 CALLS =                      83                     NUMBER OF TYPE 28 CALLS =                    13832
NUMBER OF TYPE 4 CALLS =                      691                    NUMBER OF TYPE 29 CALLS =                      0
NUMBER OF TYPE 5 CALLS =                      2720                   NUMBER OF TYPE 30 CALLS =                      0
NUMBER OF TYPE 6 CALLS =                      7                       NUMBER OF TYPE 31 CALLS =                      0
NUMBER OF TYPE 7 CALLS =                      0                       NUMBER OF TYPE 32 CALLS =                      0
NUMBER OF TYPE 8 CALLS =                      656                    NUMBER OF TYPE 33 CALLS =                    778
NUMBER OF TYPE 9 CALLS =                      26                     NUMBER OF TYPE 34 CALLS =                    21
NUMBER OF TYPE 10 CALLS =                     178                    NUMBER OF TYPE 35 CALLS =                      0
NUMBER OF TYPE 11 CALLS =                     593                    NUMBER OF TYPE 36 CALLS =                      0
NUMBER OF TYPE 12 CALLS =                     480                    NUMBER OF TYPE 37 CALLS =                      1
NUMBER OF TYPE 13 CALLS =                      0                       NUMBER OF TYPE 38 CALLS =                    204
NUMBER OF TYPE 14 CALLS =                    2099                   NUMBER OF TYPE 39 CALLS =                      9
NUMBER OF TYPE 15 CALLS =                      0                       NUMBER OF TYPE 40 CALLS =                      0
NUMBER OF TYPE 16 CALLS =                      38                     NUMBER OF TYPE 41 CALLS =                      0
NUMBER OF TYPE 17 CALLS =                      46                     NUMBER OF TYPE 42 CALLS =                    88
NUMBER OF TYPE 18 CALLS =                    2338                   NUMBER OF TYPE 43 CALLS =                      0
NUMBER OF TYPE 19 CALLS =                    2105                   NUMBER OF TYPE 44 CALLS =                      0
NUMBER OF TYPE 20 CALLS =                    825                    NUMBER OF TYPE 45 CALLS =                    312
NUMBER OF TYPE 21 CALLS =                      3                       NUMBER OF TYPE 46 CALLS =                    203
NUMBER OF TYPE 22 CALLS =                      2                       NUMBER OF TYPE 47 CALLS =                      0
NUMBER OF TYPE 23 CALLS =                    118773                NUMBER OF TYPE 48 CALLS =                      0
NUMBER OF TYPE 24 CALLS =                    11
    
```

```

08:49:24.8529      JOB CLASS STATISTICS                      TIME INTERVAL =    300006.14 MILLISECONDS
CLASS NAME =                                      OPERATR
NUMBER OF ACTIVE JXTS =                              0
NUMBER OF JOBS WAITING FOR JXTS =                      0
NUMBER OF RESERVED JXTS =                            1
MAXIMUM NUMBER OF JXTS =                            64
STATUS =                                              ON
    
```

```

08:54:24.8586      JOB SCHEDULER STATISTICS                      TIME INTERVAL =    300006 12 MILLISECONDS
NUMBER OF MEMORY COMPACTS =                      6756
NUMBER OF ROLLS =                                    1268
NUMBER OF EXPANDS =                                14
NUMBER OF REDUCES =                                2
NUMBER OF INITIATES =                              9
NUMBER OF TERMINATES =                            6
NUMBER OF SCHEDULING INTERVALS =                      60
NUMBER OF INDEX WRITES =                            212
CLASS STRUCTURE NAME =                              DAY
NUMBER OF JOBS IN SYSTEM =                            38
NUMBER OF ACTIVE JXTS =                            22
MAXIMUM # JXTS (LIMIT) =                            30
NUMBER OF AVAILABLE POOL JXTS =                      3
NUMBER OF DEFINED CLASSES =                        14
NUMBER OF CLASSES WAITING FOR JXTS =                      65535
    
```

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+200<sub>g</sub>).
- 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

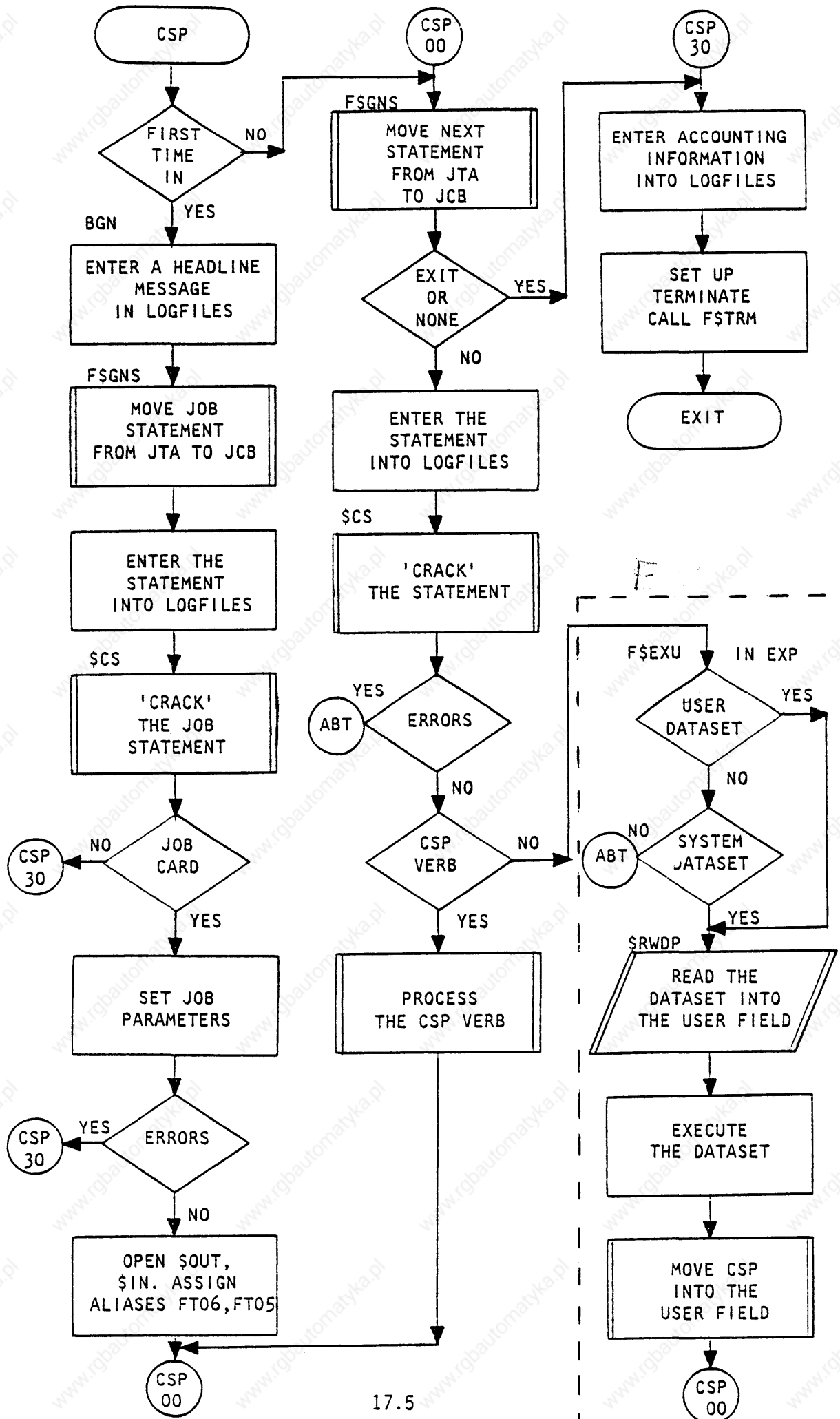
THE CONTROL STATEMENT

ASSIGN(DN=MYFILE,BS=25)

IS STORED IN THE JCB AS FOLLOWS:

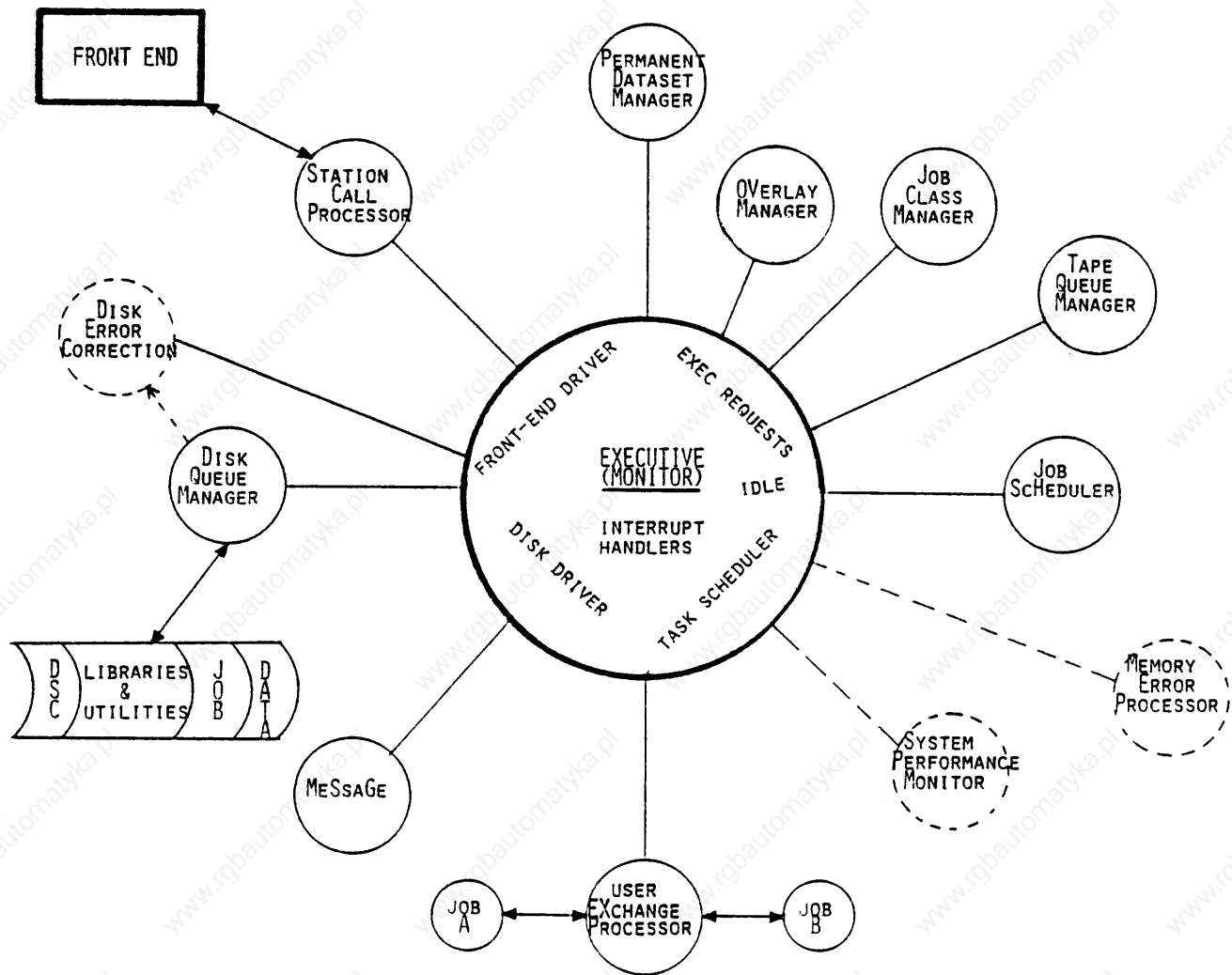
0								
5	A	S	S	I	G	N	(	D
	N	=	M	Y	F	I	L	E
	,	B	S	=	2	5	)	0
	0	0	0	0	0	0	0	0
	0	0	0	0	0	0	0	0
	⋮							
15	0	0	0	0	0	0	0	0
16	A	S	S	I	G	N	0	0
	0	0	0	0	0	0	0	17 <sub>8</sub>
	D	N	0	0	0	0	0	0
	0	0	0	0	0	0	0	07
	M	Y	F	I	L	E	0	0
	0	0	0	0	0	0	0	17 <sub>8</sub>
	B	S	0	0	0	0	0	0
	0	0	0	0	0	0	0	07
	2	5	0	0	0	0	0	0
	0	0	0	0	0	0	0	37 <sub>8</sub>











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) <sup>JOB CARD</sup> 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 THUS MAKING 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.

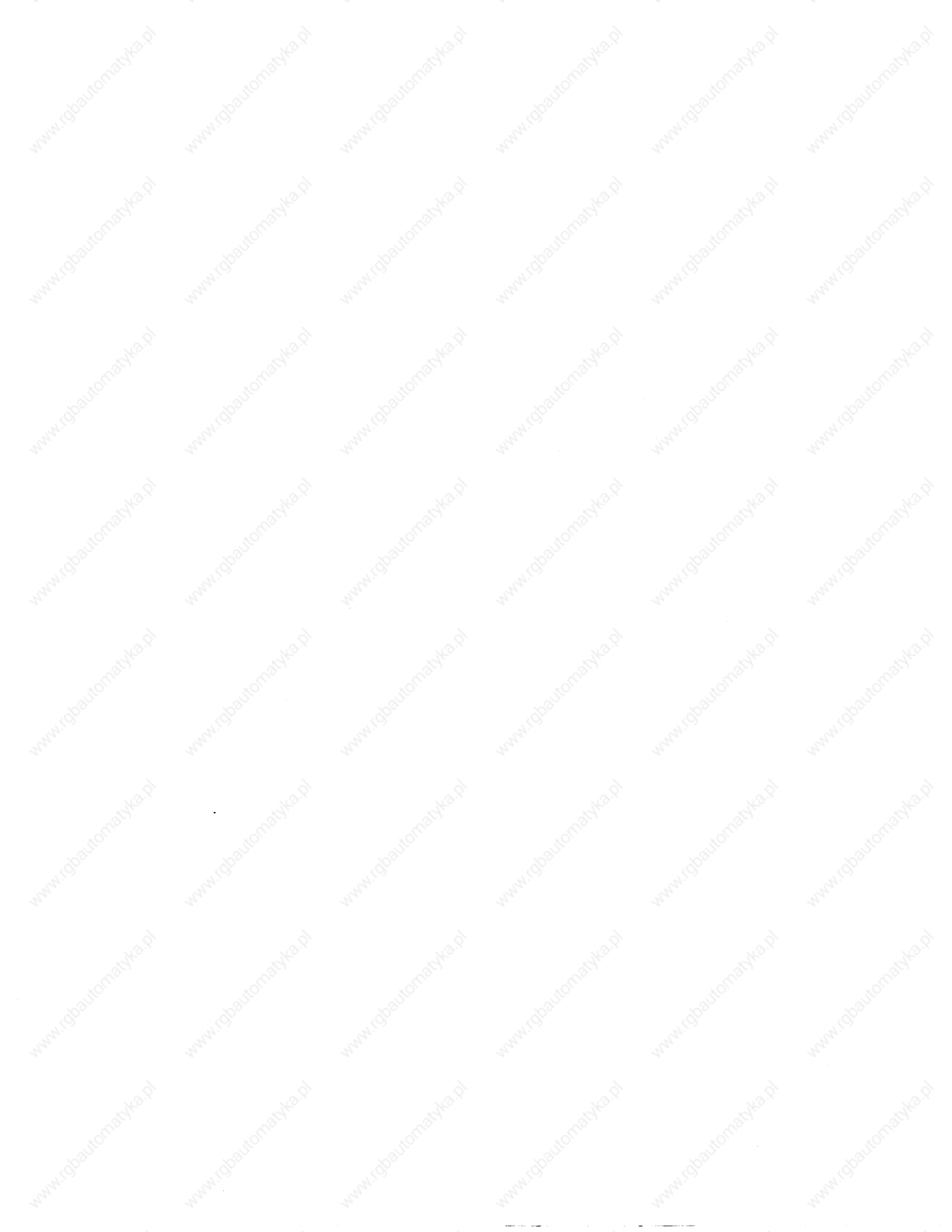
EXP REQUESTS THE JOB SCHEDULER (JSH) TASK FOR THE ACTUAL ALLOCATE/DEALLOCATE OF MEMORY.

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 I@ERASE 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.

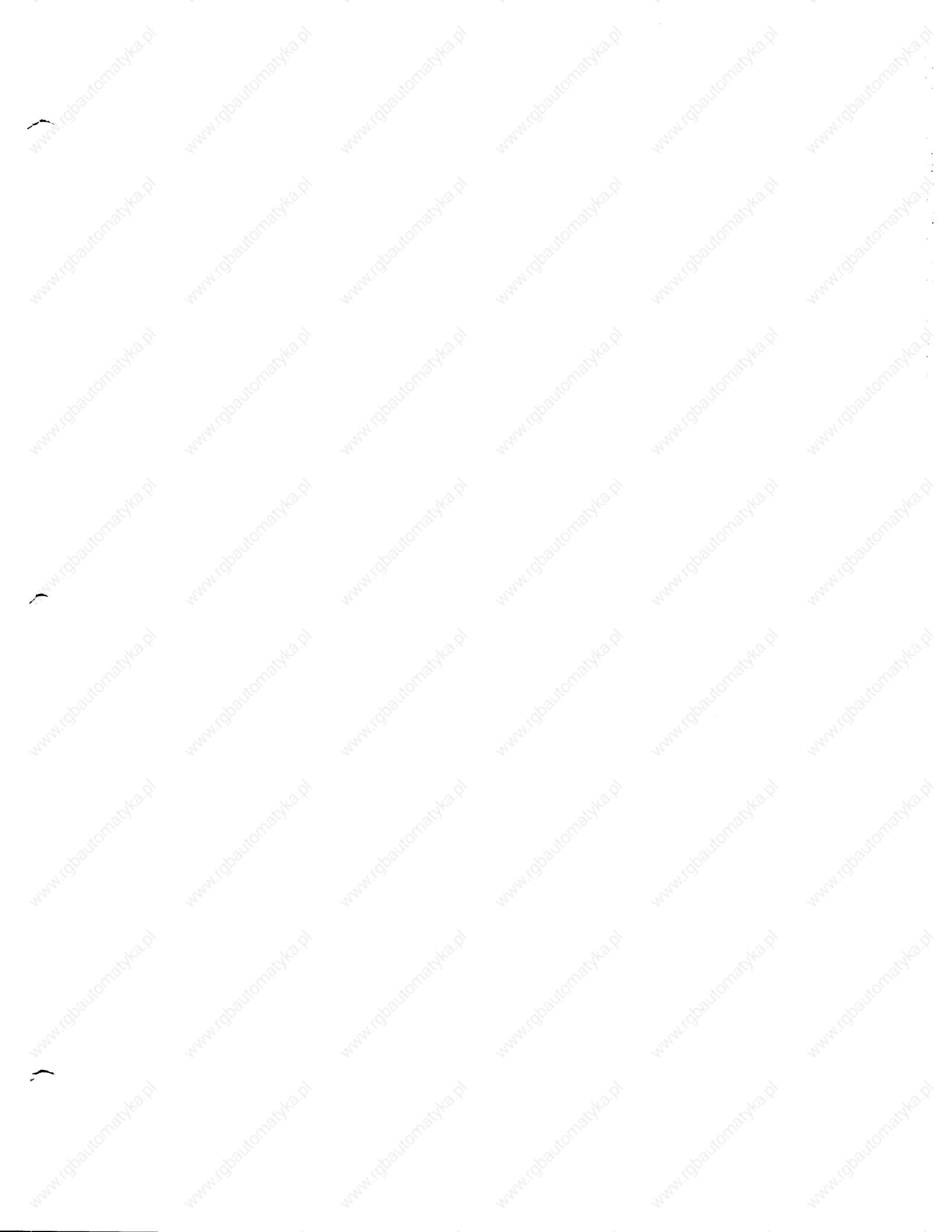
EXP CHANGES \$LOG TO SCRATCH AND RELEASES SCRATCH DATASETS AS WELL AS \$IN, \$CS.

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.





**Cray Research, Inc.  
Corporate Addresses**

**CORPORATE HEADQUARTERS**

1440 Northland Drive  
Mendota Heights, MN 55120  
Tel: 612-452-6650  
TLX 243444

≈

**THE CHIPPEWA FACILITIES**

Manufacturing:  
Highway 178 North  
Chippewa Falls, WI 54729  
Tel: 715-723-2221  
TWX 910285 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

Hallie Lab  
P.O. Box 169  
Chippewa Falls, WI 54729  
Tel: 715-723-0266

**SALES OFFICES**

Eastern Regional Sales Office  
1075C 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: 713-877-8053

Austin Sales Office  
3415 Greystone, Suite 201  
Austin, TX 78731  
Tel: 512-345-7034

≈

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 Colby Avenue  
Everett, WA 98201  
Tel: 206-259-5075

**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

**CRAY**  
**RESEARCH, INC.**