1800

T$MT

UN AS MT0 -UNLOAD

PR1ME

PRIMOS INTERNALS

Revision 19.1

MARCUS

MARCUS

PRIMOS INTERNALS

Revision 19.1

MARCUS

Date: May 13, 1983

Revision: 1

Section 1 - Hardware Features

PRIMOS OPERATING SYSTEM

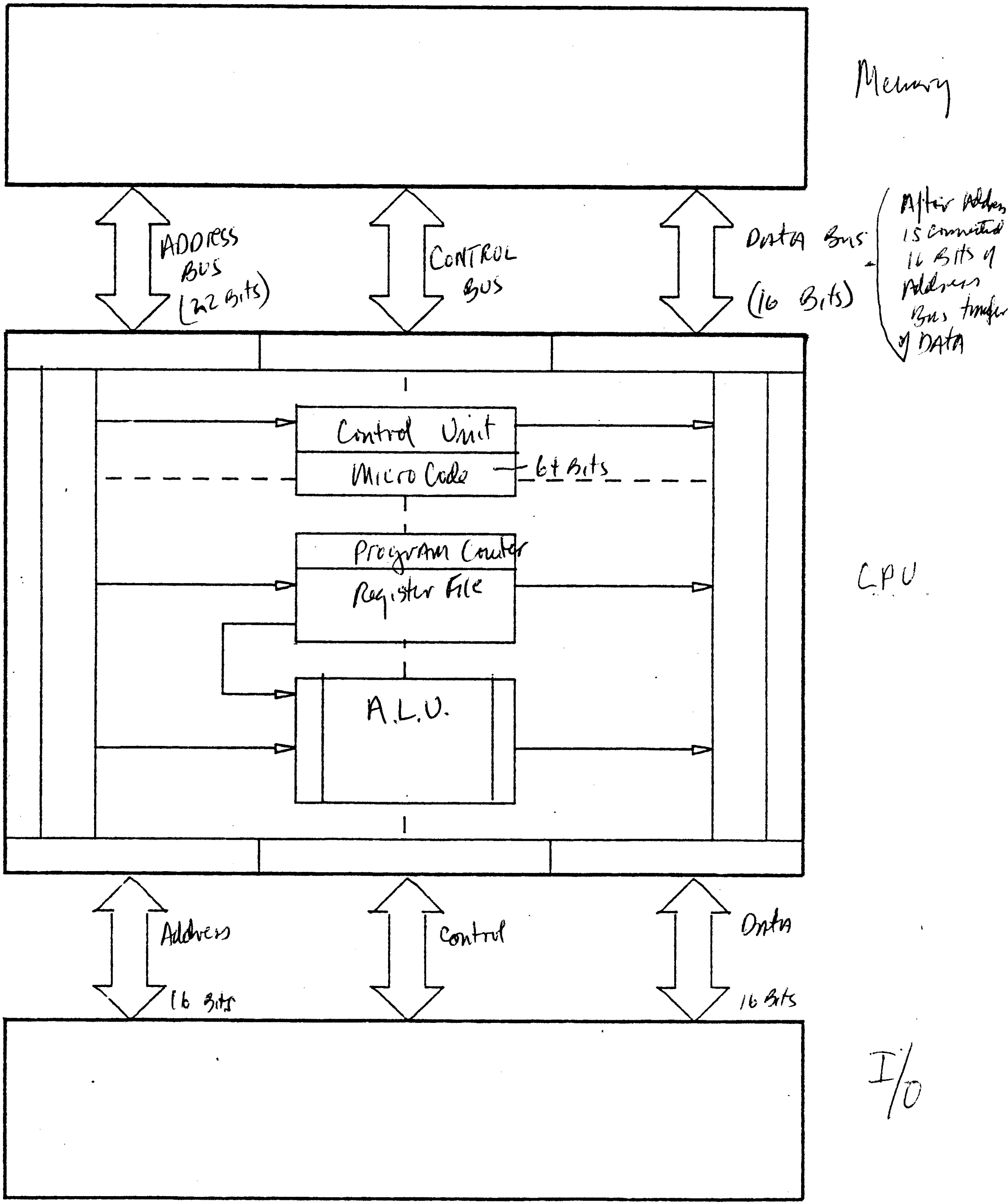
The chief features of the Primos operating system are:

1. INTERACTIVE - up to 128 user processes
   (14+ interrupt processes)
2. 32 MB maximum virtual address space per user (NOT Actually there At memory)
3. Users share the resources of the system

High speed memory

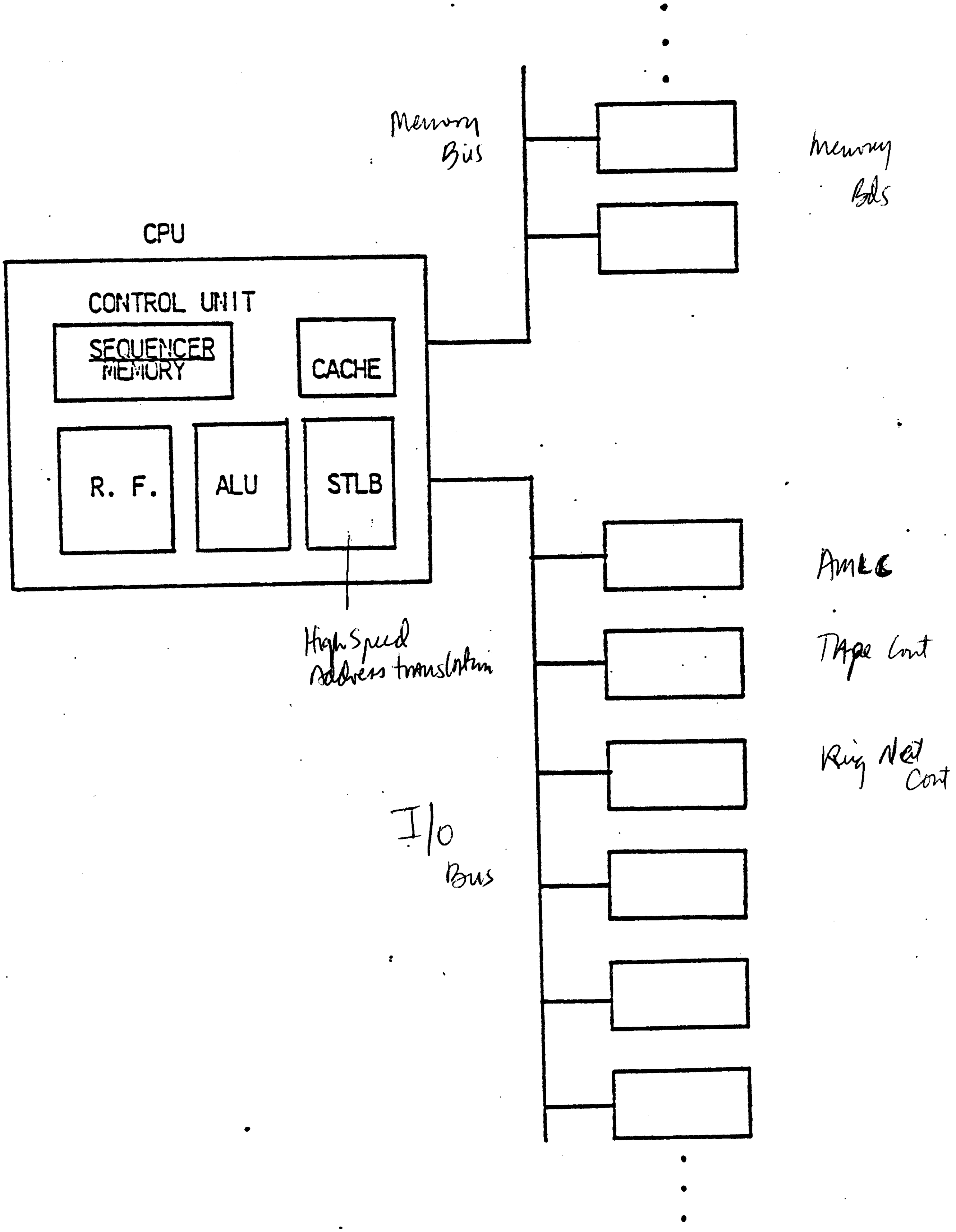
Peripherals and controllers

- System Console ✓
- Real Time Clock ✓
- Disk Drive(s) ✓ (At least 1 Disc)
- AMLC(s)/ICS1(s) ✓ Async Intelligent Communication
- SMLC(s)/MDLC(s) ✓ Sync Multi Data Link (Pt to Pt communication
- Ring Node Controller (PNC) ✓ Local Ring
- Magnetic Tape Drive(s) ✓
- Line Printer(s) ✓



All Registers
All Busses } 32 Bits

CENTRAL PROCESSOR UNIT



REGISTER FILE

128 - 32 Bit Registers

MICROCODE SCRATCH

	HIGH	LOW
0	TR0	
1	TR1	
2	TR2	
3	TR3	
4	TR4	
5	TR5	
6	TR6	
7	TR7	
10	RDMX1	
11	RDMX2	
12		RATMPL
13	RSGT1	
14	RSGT2	
15	RECC1	
16	RECC2	
17		REDIV
20	ZERO	ONE
21	PRSAVE	
22	RDMX3	
23	RDMX4	
24	C377	
25		
26	LREGSET	CHKREG
27	DSWPARITY	
30	PSWPB	
31	PSWKEYS	
32	PPA:PLA	PCBA
33	PPB:PLB	PCBB
34	DSWRMA	
35	DSWSTAT	
36	DSWPB	
37	RSAVPTR	

DMA

	HIGH	LOW
0		
1		
2		
3		
4		
5		
6		
7		
10		
11		
12		
13		
14		
15		
16		
17		
20	(20)	(21)
21		
22	(22)	(23)
23		
24	(24)	(25)
25		
26	(26)	(27)
27		
30	(30)	(31)
31		
32	(32)	(33)
33		
34	(34)	(35)
35		
36	(36)	(37)
37		

CURRENT REGISTER

	HIGH	LOW
0	GR0:DLT2	
1	GR1:PTS	
2	GR2(1,A,LH)	(2,B,LL)
3	GR3 (EH)	(EL)
4	GR4	
5	GR5 (3,S,Y)	
6	GR6	
7	GR7 (0,X)	
10	FAR0 (13)	
11	FLR0	
12	FAR1/FAC(4)	(5)
13	FLR1/FAC(6)	
14	PB	
15	SB (14)	(15)
16	LB (16)	(17)
17	XB	
20	DTAR3 (10)	
21	DTAR2	
22	DTAR1	
23	DTAR0 → Primos	
24	KEYS	MODALS
25	OWNER	
26	FCODE (11)	
27	FADDR	(12)
30	CPU TIMER	
31	MICROCODE SCRATCH	
32	"	
33	"	
34	"	
35	"	
36	"	
37	"	

THE USER REGISTER SET

A & B register together is called L Register

All in Octal # '1 → '16

All are 32 Bit Registers

	HIGH	LOW		
0			GR0	
1			GR1	
2	A	B	GR2	
3	EH (Extension reg H)	EL (Extension reg L)	GR3	
4			GR4	
5	S/Y		GR5	
6			GR6	
7	X		GR7	
10	FAR0			
11	FLR0			
12	FAR1/FAC			(Field Address } For Decimal Arithmetic
13	FLR1/FAC			(Field Length
14	PB			
15	SB			
16	LB			
17	XB			
20	DTAR3			} Virtual Address Space
21	DTAR2			
22	DTAR1			
23	DTAR0			
24	KEYS/MODALS			
25	OWNER			– Points BACK to Processor Control Block
26	FCODE			
27	FADDR			
30	CPU TIMER			← INTERVAL timer
31	MICROCODE SCRATCH			
.	"			
.	"			
37	"			

THE USER REGISTER SET

A	Accumulator Register
B	Accumulator Extension (A + B = L)
EH,EL	Accumulator Extension for long integers (64 bit)
S	Stack Register (R S Modes)
Y	Alternate Index Register (V Mode only)
X	Index Register (R, S, V Modes)
GR0-GR7	General Registers 0-7 (I Mode only)
FAR0	Field Address Register 0
FLR0	Field Length Register 0
FAR1	Field Address Register 1 (for block moves
FLR1	Field Length Register 1 char./dec. data)
FAC	Floating Point Accumulator
PB	Procedure Base Register
SB	Stack Base Register
LB	Link Base Register
XB	Auxiliary Base Register
OWNER	Address of User Register Set Owner's PCB
FCODE	Fault Code
FADDR	Fault Address
CPU TIMER	overflow of two's complement value ends timeslice

User programs may access the Register-file using LDLR and STLR (64V).
Only locations '0 - '17 are accessible.
Any attempt to access location '14 (PB) will give undefined results.
The first eight locations are interpreted for V-mode (default).

PROCEDURE/LINK/STACK ARCHITECTURE

PROCEDURE AREA

- 1 per system if shared
- contains pure code and literals
- pointed to by Procedure Base Register (PB)

LINKAGE AREA

- 1 per user
- contains local variables and pointers
- pointed to by Linkage Base Register (LB)

STACK FRAME

- 1 per invocation
- contains caller's saved state, argument pointers, and dynamic work space
- pointed to by Stack Base Register (SB)

CK

<	=		
0	0	-	>
0	1	-	=
1	0	-	<

To store pointer in seperate locations

KEYS

bit #	purpose S R Modes	 V I Modes
1	Arithmetic Error Cond.	C Bit (Carry Bit)
2	Double Precision Bit	reserved
3	reserved	Link Bit
4-6	Mode bits	Mode Bits (Can switch Between keys)
	000 16S mode	
	001 32S	
	011 32R	
	010 64R	
	110 64V	
	100 32I	
7	reserved	Floating Point Exception
8	reserved	Integer Exception
9	Bits 9-16 are bits 9-16	LT (less than) bit } Condition Bits
10	of address 6	EQ (equal) bit
11	"	DEX (decimal exception)
2-13	"	reserved
14	"	In CHECK bit (850 only) Allows only 1 cpu to use reg.
15	"	I bit - In Dispatcher
16	"	S bit - Save Done

MODALS

bit #	PURPOSE (V I modes only)	
1	Interrupts Enables	
2	Vectored Interrupt Mode	
3	Disable Prefetch Overlap (P750)	
4	Disable Indirect Overlap (P750)	
5 - 8	reserved - Must be zero	
9 -11	Current Register Set	
12	Mapped I/O Mode	Used at cold Start
13	Process-exchange Mode	
14	Segmentation Mode	
15 - 16	Machine Check Mode	

00 = Report no errors

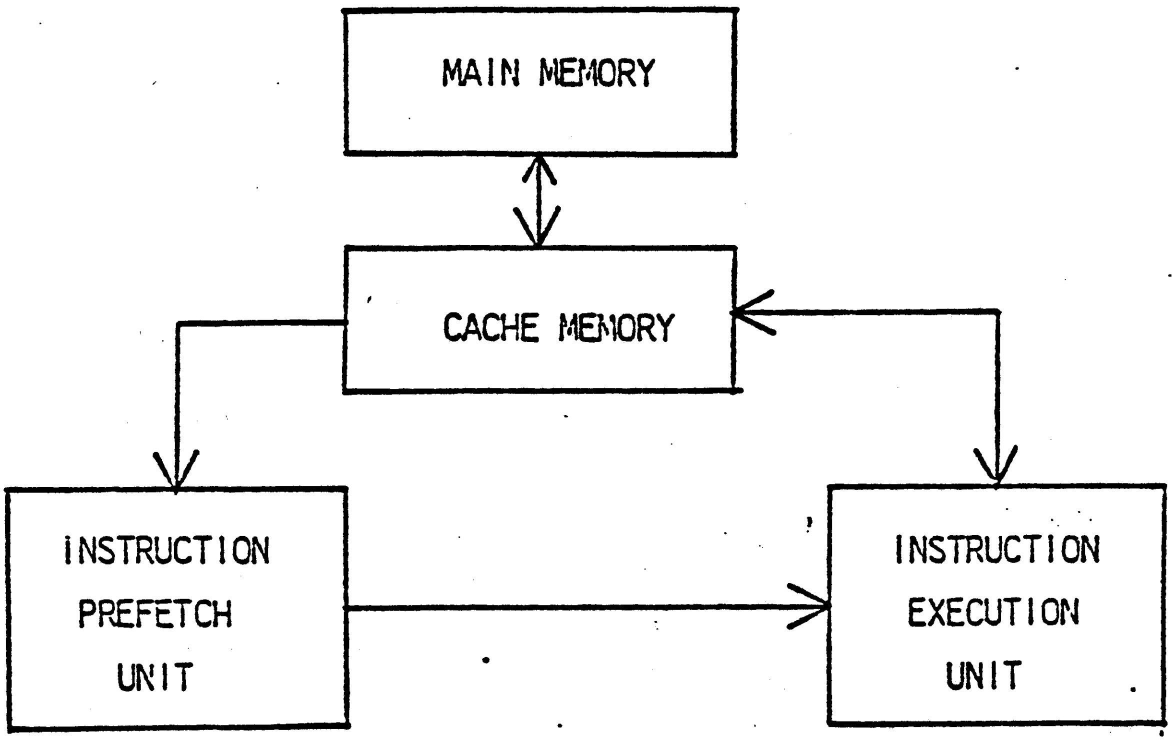
01 = Report ECCU errors only
(Error Checking and Correction Uncorrectable)

10 = Report all unrecoverable errors
(only ECCC errors are unrecorded)

11 = Report and record all errors

INSTRUCTION PREFETCH UNIT

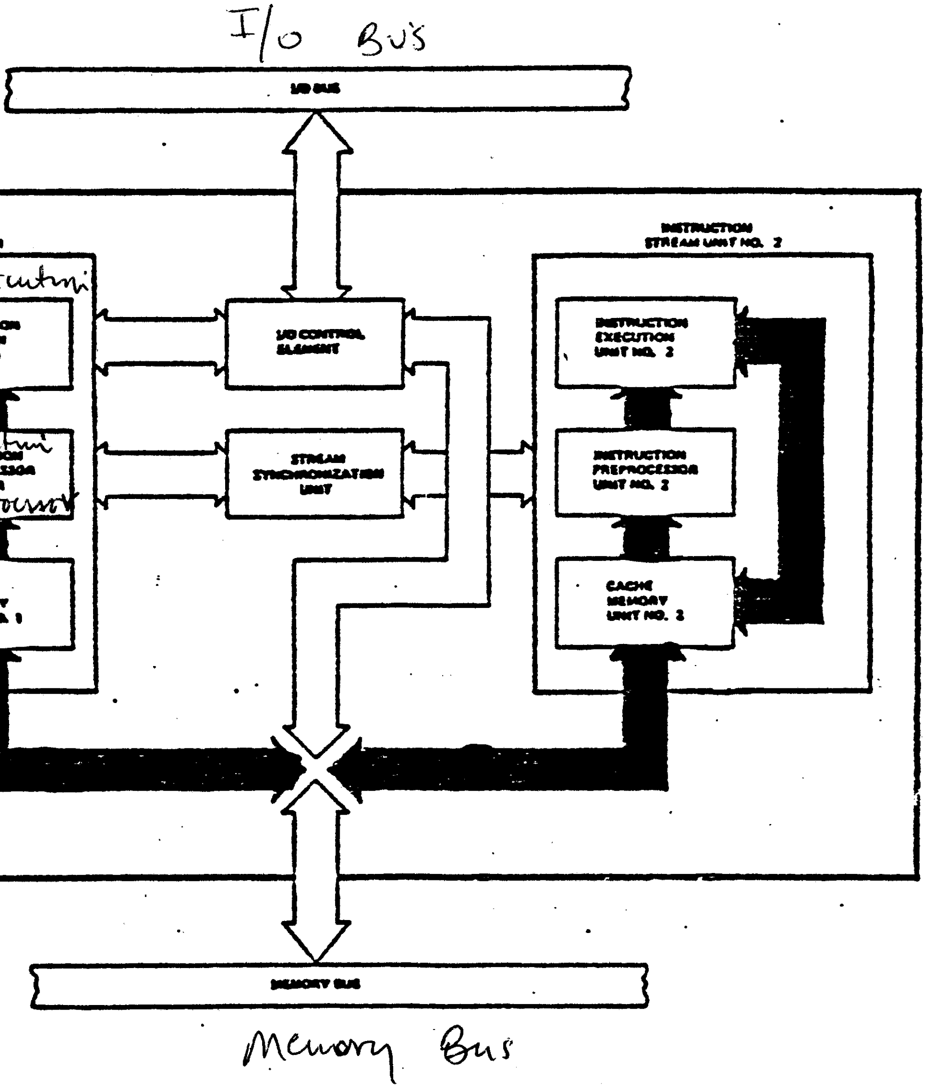
750/850



Pre fetch
(Concurrent Processing)

Read Program
What address?
Pre Load Cache

PRIME P850 FUNCTIONAL DIAGRAM


I/O Bus
I/O BUS
INSTRUCTION STREAM UNIT NO. 1
INSTRUCTION STREAM UNIT NO. 2
Instruction Execution
INSTRUCTION EXECUTION UNIT NO. 1
I/O CONTROL ELEMENT
INSTRUCTION EXECUTION UNIT NO. 2
Instruction
INSTRUCTION PREPROCESSOR UNIT NO. 1
Pre Processor
STREAM SYNCHRONIZATION UNIT
INSTRUCTION PREPROCESSOR UNIT NO. 2
CACHE MEMORY UNIT NO. 1
CACHE MEMORY UNIT NO. 2
MEMORY BUS
Memory Bus

5 Bd.
CPU

850 has Shared memory and I/O

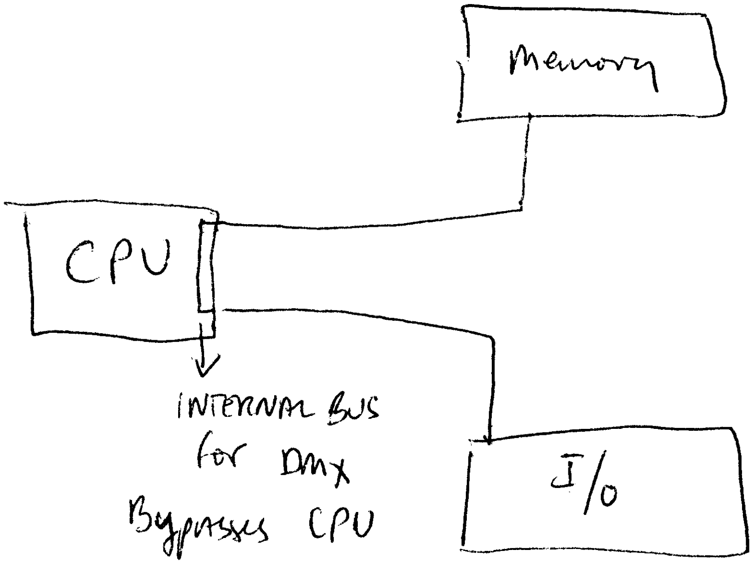
Stream Sync allots time between CPU
And invalidates Cache to
prevent overlapping Cache error

850
Master ISU - Does I/O
Slave ISU

DMx Operation

DMx is a method whereby an I/O data/memory transfer may occur without program intervention. To perform such operations a temporary diversion in the sequence of microcode from CPU instruction to DMx transfer routines occurs. This is called cycle stealing or a TRAP. At the end of the DMx/memory transfer, the CPU instruction microcode continues as though nothing had happened. The actual trap diversion occurs at the end of the micro step in which it was sensed. At the same time, information about the next CPU micro step is saved to effect a return to the original sequence.

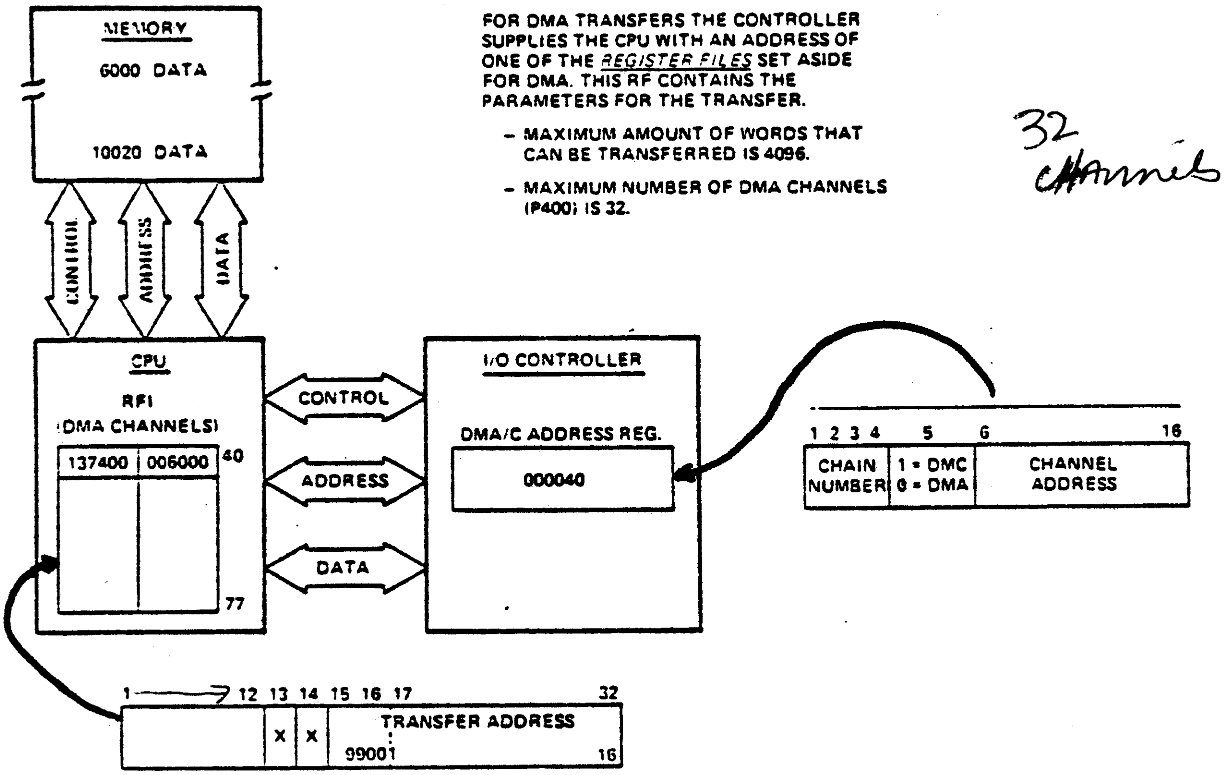
There are four types of DMx transfer: DMA, DMC, DMT, and DMQ. Each method has advantages and disadvantages in terms of speed, volume and control features and so form a comprehensive range of methods.



1). DIRECT MEMORY ACCESS (DMA)

DMA transfers are controlled by pairs of registers (channels) in the CPU register file. There are 32 such register channels, locations '40 - '77 in the register file (32 bit locations). The high 12 bits of each location govern the number of words to be transfered and the low 16 bits specify the start address of the buffer to be used.

DIRECT MEMORY ACCESS (DMA)*



FOR DMA TRANSFERS THE CONTROLLER SUPPLIES THE CPU WITH AN ADDRESS OF ONE OF THE *REGISTER FILES* SET ASIDE FOR DMA. THIS RF CONTAINS THE PARAMETERS FOR THE TRANSFER.

- MAXIMUM AMOUNT OF WORDS THAT CAN BE TRANSFERRED IS 4096.
- MAXIMUM NUMBER OF DMA CHANNELS (P400) IS 32.

32 channels

* Example shows parameters for a 1040 word transfer from to locations 6000 - 10020

ON 850 Burst mode is used with Wide word Inter. transfers 4X As much

Before transfers begin, the program must set up the channel registers in the CPU. Up to 4096 words per channel may be transfered. Successive channels may be chained by setting channel registers in the CPU and the chaining register in the controller (not all controllers have this capability)

2). DIRECT MEMORY CHANNEL (DMC)

DMC transfers are controlled by pairs of words (Channels) in main memory. The first (even) word controls the first and current address of the buffer, and the second word controls the last address of the buffer. There is potential for transferring 65536 words, but in practice transfers are usually very much smaller than this.

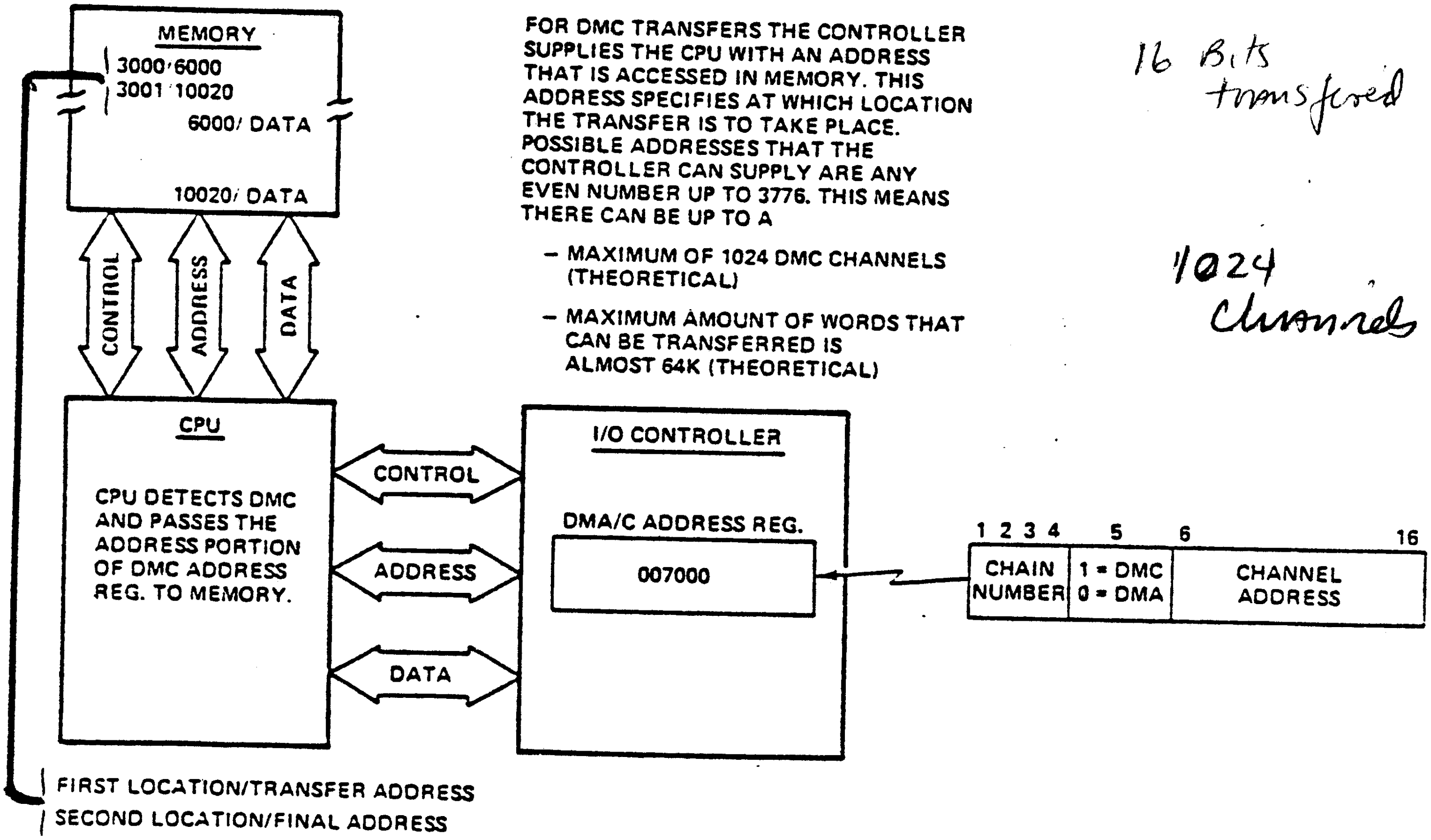
DIRECT MEMORY CHANNEL (DMC)*

FOR DMC TRANSFERS THE CONTROLLER SUPPLIES THE CPU WITH AN ADDRESS THAT IS ACCESSED IN MEMORY. THIS ADDRESS SPECIFIES AT WHICH LOCATION THE TRANSFER IS TO TAKE PLACE. POSSIBLE ADDRESSES THAT THE CONTROLLER CAN SUPPLY ARE ANY EVEN NUMBER UP TO 3776. THIS MEANS THERE CAN BE UP TO A

- MAXIMUM OF 1024 DMC CHANNELS (THEORETICAL)
- MAXIMUM AMOUNT OF WORDS THAT CAN BE TRANSFERRED IS ALMOST 64K (THEORETICAL)

16 Bits transfered

1024 Channels



CPU DETECTS DMC AND PASSES THE ADDRESS PORTION OF DMC ADDRESS REG. TO MEMORY.

FIRST LOCATION/TRANSFER ADDRESS
SECOND LOCATION/FINAL ADDRESS

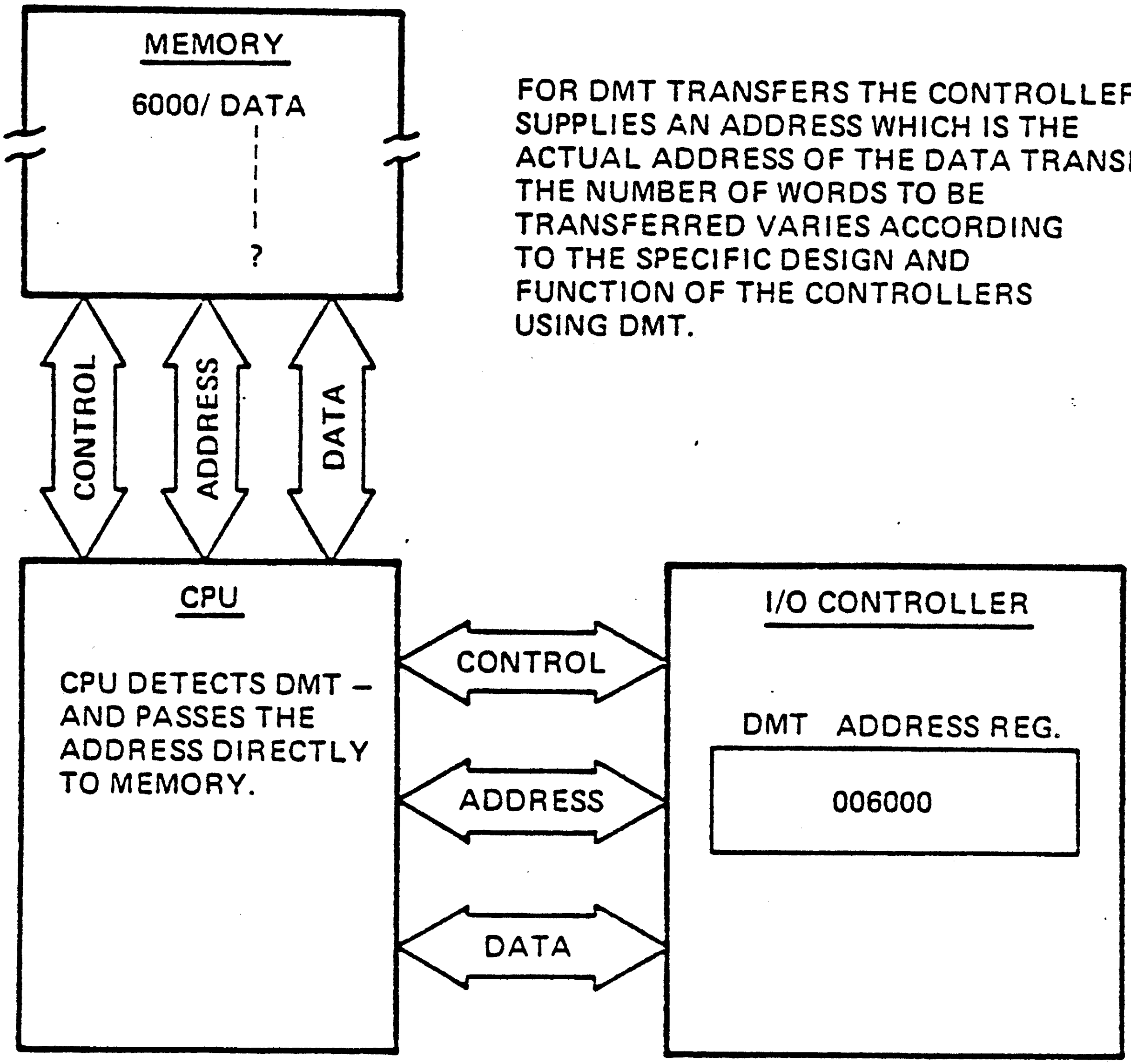
* Example shows parameters for a 1040 word transfer from/to locations 6000-10020.

1024 DMC channels are available in the system but the use of memory for control words makes it slower than DMA.

3). DIRECT MEMORY TRANSFERS (DMT)

DMT transfers are controlled by the device controllers themselves. The memory of the start and current location of the buffer, and the memory address of the last location of the buffer are held in the controller.

DIRECT MEMORY TRANSFER (DMT)*



FOR DMT TRANSFERS THE CONTROLLER SUPPLIES AN ADDRESS WHICH IS THE ACTUAL ADDRESS OF THE DATA TRANSFER. THE NUMBER OF WORDS TO BE TRANSFERRED VARIES ACCORDING TO THE SPECIFIC DESIGN AND FUNCTION OF THE CONTROLLERS USING DMT.

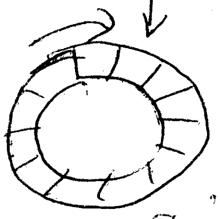
* *Example shows parameters for a transfer to/from location 6000.*

4). DIRECT MEMORY QUEUE (DMQ)

DMQ mode provides a ring-structured memory buffer for the reception and transmission of stream I/O. Stream I/O is a data transfer in which data is transfered in continuous streams of bits, characters or words rather than in discrete records

This mode allows the AMLQ driver to queue messages using queing instructions, without the need for extensive software management of character time interrupts on transmit. Therefore DMQ mode substantially reduces the system overhead in dealing with terminal output.

read pt

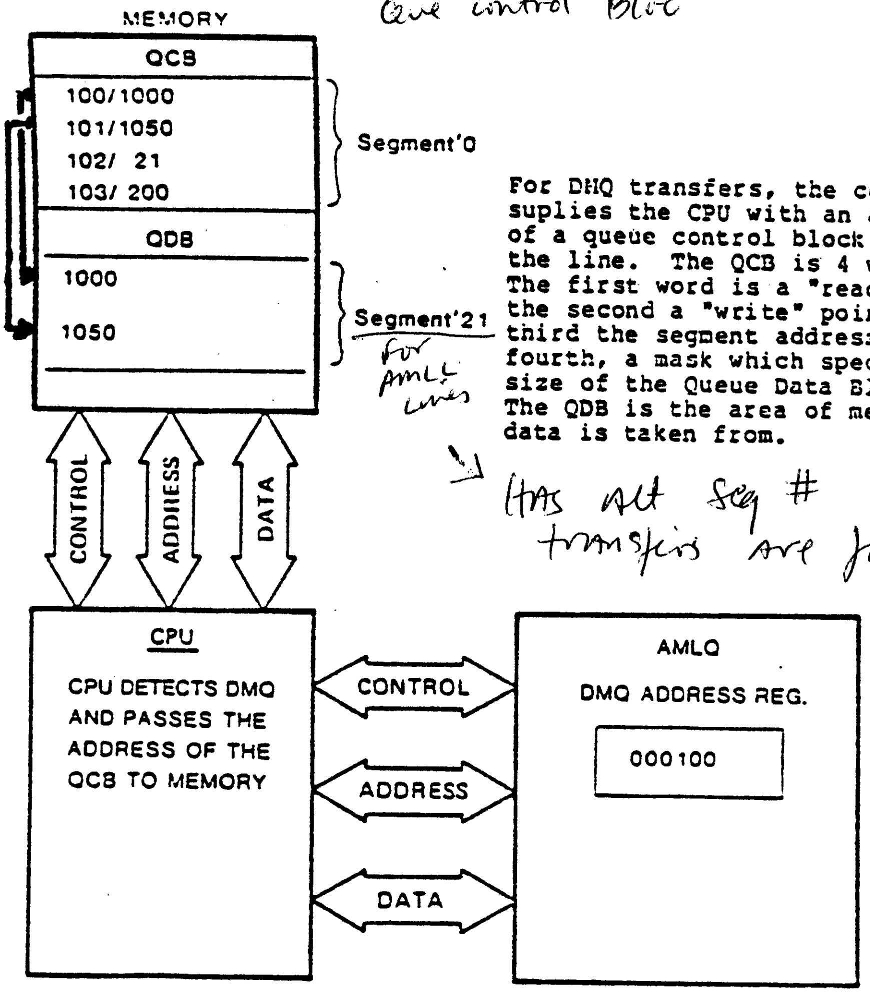


write pointer

Ring Structured Buffer Never gets full

Que control Bloc

Read pointer
Write pointer



for AMLL lines

For DHQ transfers, the controlle suplies the CPU with an address of a queue control block (QCB) for the line. The QCB is 4 words long. The first word is a "read" pointer, the second a "write" pointer, the third the segment address and the fourth, a mask which specifies the size of the Queue Data Block (QDB). The QDB is the area of memory the data is taken from.

Has all seg # All other transfers are from Q

DMQ Operation

The control information is held in segment 0 of memory in an area known as the Queue Control Block (QCB).

Each queue is implemented by an array of 2**N words where N is greater than or equal to 4, and less than or equal to 16.

Each QCB is a four word structure:

TOP POINTER (read)	word number of the head of the queue
BOTTOM POINTER (write)	word number of the tail of the queue
SEGMENT NUMBER or PHYSICAL ADDRESS	segment number or PPN of above pointers
MASK	2**N - 1 defines the size of the buffer

The instructions provided for DMQ and QUEUE manipulation are:

ATQ	: add to the top of the queue
ABQ or DMQ input	: add to the bottom of the queue
RTQ or DMQ output	: remove from top of the queue
RBQ	: remove from the bottom of the queue
TSTQ	: test the queue (# of items->A, ifemptyEQ->CC)

Section 2 - Lab Exercise 1

PRxxxx Files

PRIMOS SOURCES

FILES	RING0.MAP	Ring 0 SEG map
	RING3.MAP	Ring 3 SEG map
INPUT file to LOAD modules	RING0.LOAD	Ring 0 SEG load control file
	RING3.LOAD	Ring 3 SEG load control file
SUBDIRECTORIES		
	CPLS	CPL interpreter
	CS	Communications: synchronous
	ES	Emulators: dptx — to communicate with other computer systems
	FS	File system
	INSERT	Insert files
	KS	Kernel
	NPXS	NPX (slave)
	NS	Networks: FAM I, FAM II
	OBJ	Binaries
	PSD	Wired debugger
	R3S	Ring 3 and command processor
	RJES	Remote job entry
	FIND_OBJ	Utility to use a load control file and merge binaries from two separate ufds (LOAD from 2 Directories)
	PRMLD	Primos preloader
	MAPGEN	Program to generate initial page maps
	USAGE	Usage monitoring utility — Displays event meters (% IDLE)

PRIMOS BUILD - COMPILE.CPL

```
R COMPILE [<object>]
          [-FTN]   [-PLP]    [-PMA]
          [-Bin <treename>]  [-List <treename>]
          [-AFter <date>]    [-BeFore <date>]       (date = MM/DD/YY)
          [-No_COmo]         [-COMO <treename>]
```

The caller may specify a <source_tree> of an item, sub-dir or file, to be compiled. The default is to compile all languages in all dirs.

The user is also allowed to specify the -BEFORE and -AFTER arguments to compile only modules changed during a specified time interval.

If any of -FTN, -PMA or -PLP is given, then only modules written in those languages are compiled. If all are omitted, all languages are compiled.

If -AFTER and/or -BEFORE is given, only those modules which also have a date-time-modified within the bounds specified by -AFTER and -BEFORE, are compiled. If neither is given, dtm is not checked.

If -NO_COMO is given, a separate comoutput file is not produced. Otherwise, %dir%.como is produced.

PRIMOS BUILD - COMPILE.CPL examples

A file may be specified in a number of ways:

```
    ks>ainit.ftn , ks>ainit , ainit.ftn , ainit
```

If a sub-dir is omitted, each one one is searched for the file.

If the language suffix is omitted a search is done using PMA, FTN, and PLP until the file is found.

NOTE:::: Any unclaimed arguments will be used as compiler options!!!

Examples:

```
R COMPILE                  compiles everything, creates compile.como
R COMPILE -PLP -AF 0 -NCO  compiles only PLP modules modified after
                              midnight this morning; no como file.
R COMPILE ks -BF 1-1       compiles all modules in KS modified before
                              midnight on Jan. 1 of the current year.
R COMPILE ks>ainit.ftn     compiles *>ks>ainit.ftn
R COMPILE ks>ainit         searches ks for ainit.(PMA FTN PLP)
R COMPILE ainit.ftn        searches all sub-dirs, *>@@>ainit.ftn
R COMPILE ainit            compile  *>@@>ainit.(PMA FTN PLP)
```

LD *>EE>EE ~ EE

Does Ring & Load

PRIMOS BUILD - LOAD.CPL

```
R LOAD [<load_data_file>]
       [-LIBRARY <lib_path> | -LIB <lib_path>]
       [-OBJECT <obj_path> | -OBJ <obj_path>]
       [-RING <ring> | -R <ring>]
       [-VERSION <version> | -V <version>]
       [-NO_COMO | -NCO]
```

- installed BASE of Primos

```
    <load_data_file>  file with seg commands and name of files to load
    <lib_path>  dir containing binary files of installed (base) Primos
    <obj_path>  dir containing binary files that are new
    <ring>      ring to load (currently 0 or 3)
    <version>   char string of this version of Primos (e.g. 18.0.10)
```

```
defaults: load data file=     lib path=    obj path=  ring=  version=
           RINGring.LOAD     PRI19.CK>OBJ    *>OBJ      0      19.0
```

This CPL procedure accepts a load data file in the following format:

```
        /* comment line - ignored
        .SEG <command> - direct command to seg, passed as is
        file_name {optional seg numbers for seg}
```

When the line is a file_name,

```
   file_name.bin is searched for in the object directory;
      if found the object pathname is prepended to file_name
      else the library pathname is prepended to file_name
   (in both cases .BIN is appended to the filename)
```

PRIMOS BUILD - more CPL utilities

— recompiles & loads everything

PRIMOS.BUILD.CPL

```
R PRIMOS.BUILD {version_number} {-LOAD}
```

Compiles and/or load all of PRIMOS.

MOVSYS.CPL (in PRIRUN) - copies files to prirun

```
R MOVSYS <source_tree> <target_tree> [ -OPSYS ] (default)
                                     [ -ALL ]
                                     [-HELP | -USAGE ]
```

Copy primos and/or prirun modules between ufds.

VERSION_STAMP.CPL - Gives version type to file

Type out version number and creation date of this PRIMOS.

Builds file *COLD for initialization

COLD.CPL

Build *colds and run mapgen.

MOTIVATION

- Allows Primos to be booted in two steps:

  New BOOT command to the VCP
  SETIME command to Primos

- Or in three steps:

  Old or New BOOT command to VCP
  PRIMOS command to DOS (Primos II)
  SETIME command to Primos

IMPLEMENTATION

- Software required for new BOOT command:

  New BOOT file from rev 19 Master Disk or rev 19 MAKE
  Rev 19.0 *DOS64 in DOS
  PRIMOS command in CMDNC0
  COMDEV must be first partition on device

NEW BOOTSTRAP

NEW BOOT COMMAND:

- Uses switches 4 and 5.

Primos
(is command
that replace)
VA PRIRUN
R Primos

```
4:  down - prompt for 'Physical Device='
    up   - use first partition on device specified in BOOT
           command

5:  down - prompt for user input in Primos II via 'OK:'
    up   - execute PRIMOS command for user
```

- PRIMOS command defaults to booting out of PRIRUN.

- Must re-issue PRIMOS command to change default directory.

- Note, PRIMOS command will work without new BOOT/DOS. (However, if the command device is rev 19 format, ONLY the new DOS will recognize the disk.)

Installing a RING 0 GATE

This lab exercise consists of two distinct parts: modifying PRIMOS to add the gate and writing an application routine to take advantage of the new gate.

TO modify Primos

Adding a Gate to PRIMOS

PRIMOS RING 0 Gates are defined in PRIMOS>KS>SEG5.PMA

GATE SRCH $$

Each Gate takes the form:

```
                    GATE      <ring 3 name>,<ring 0 name>
```

where

<ring 3 name> is the routine name the application will use

<ring 0 name> is the actual routine name in ring 0

if only one argument is present, then <ring 3 name> = <ring 0 name>

Add your new gate, being careful to place it at the end of the list, after all the other gates. Also be sure that the name you use is unique.

The next step is to invoke COMPILE.CPL in order to re-compile the appropriate module(s). (Hint--Look at source comments)

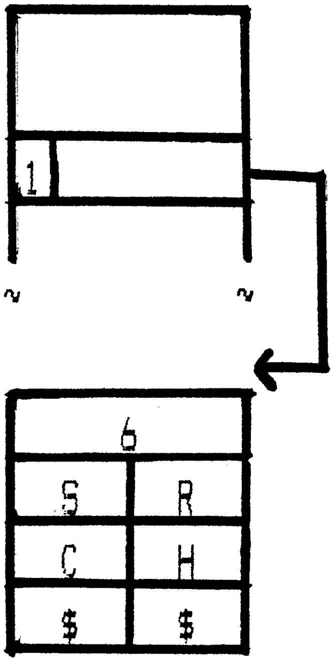
The newly compiled modules need to be re-loaded.
Use PRIMOS.BUILD.CPL or LOAD.CPL, remember to set the version number.

Add gate to END of List

Calling a RING 0 GATE

The application program should be kept as simple as possible, and must contain a "CALL <ring 3 name>" with arguments as required.
In order to get a LOAD COMPLETE message from SEG, you will need to write a short PMA program as follows:

```
        SEG
        DYNT      <ring 3 name>
        END
```



```
      SEG
e.g.  DYNT     SRCH$$
      END
```

TNOU -(USRNUM, STRING, COUNT)
STNOU -(STRING , COUNT)

TNOU , STNOU

Gate My gate, TNOU

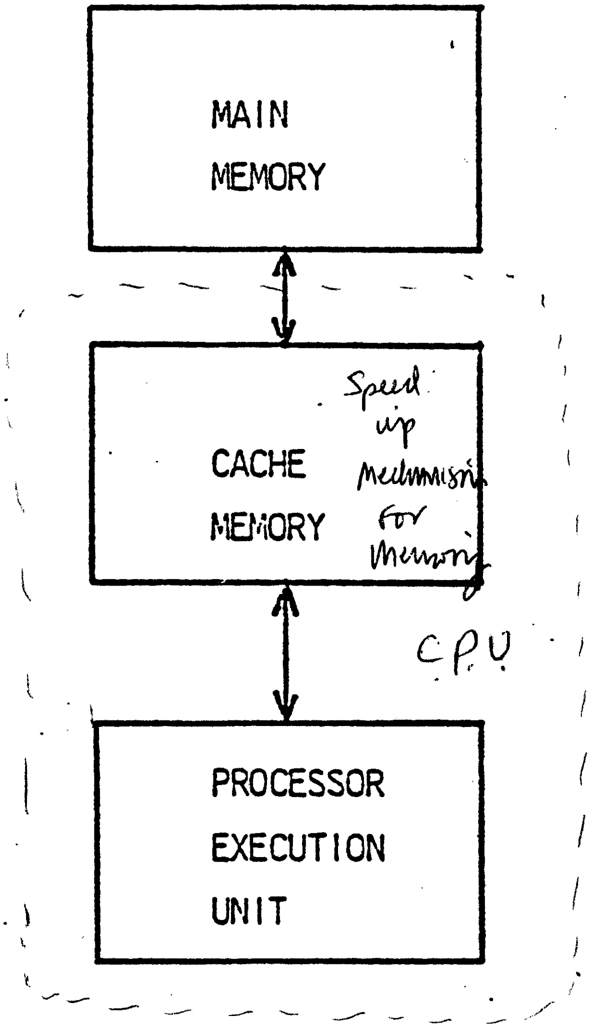
Testing the program

First try executing your application program under standard PRIMOS.
REBOOT the system with your modified PRIMOS.
Try executing your application program again.

Section 3 - MEMORY

1 K word on 2250
1024 entries

CACHE FUNCTIONAL DIAGRAM



DATA
AND
INSTRUCTIONS

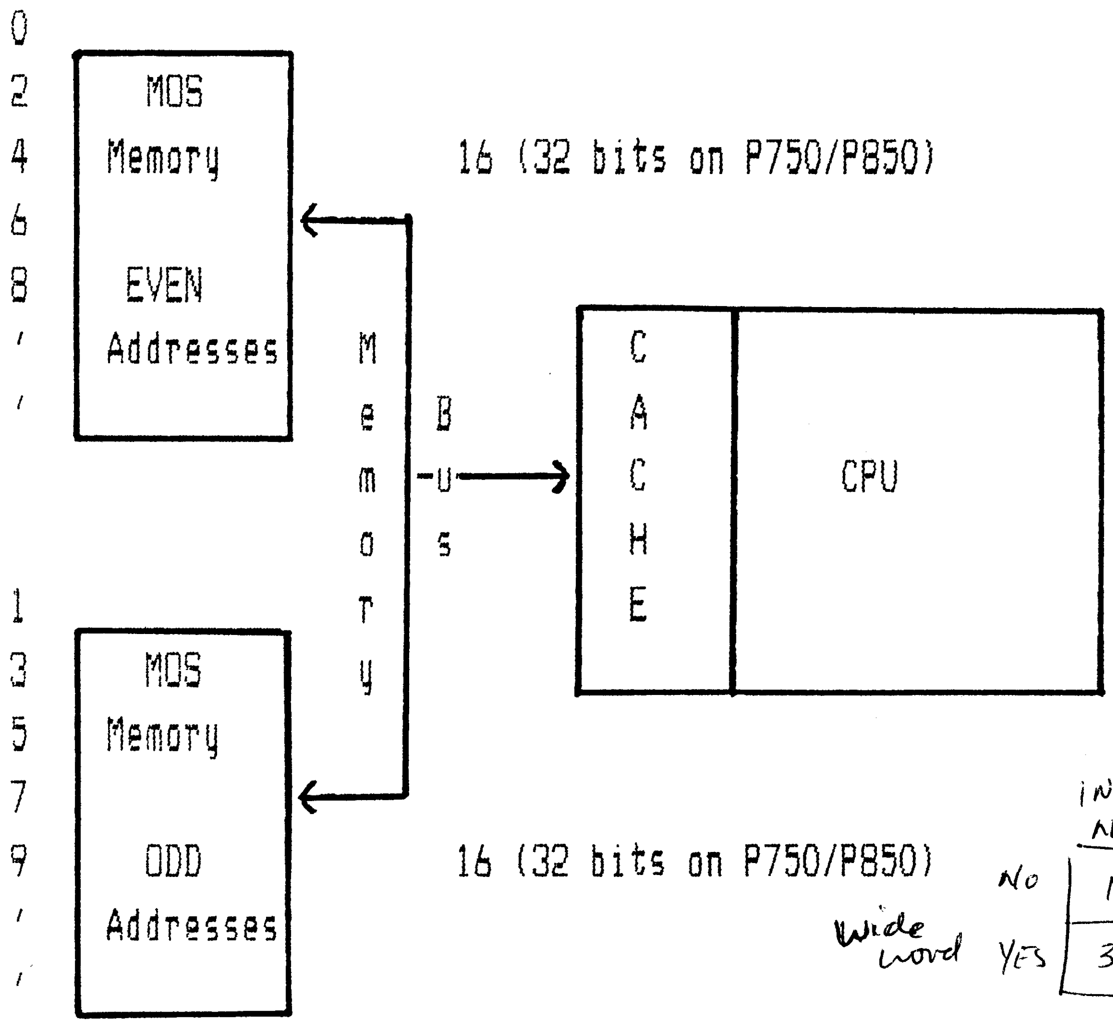
1 word at a time
Data

Having greater Cache space &
Loop Back in program
Cache gives most benefit

Cache Hit rate 80%

INTERLEAVING

Memory 600μ sec
cycle this 800 μsec



		INTERLEAVED	
		NO	YES
Wide word	NO	16	32
	YES	32	64

# of Bits

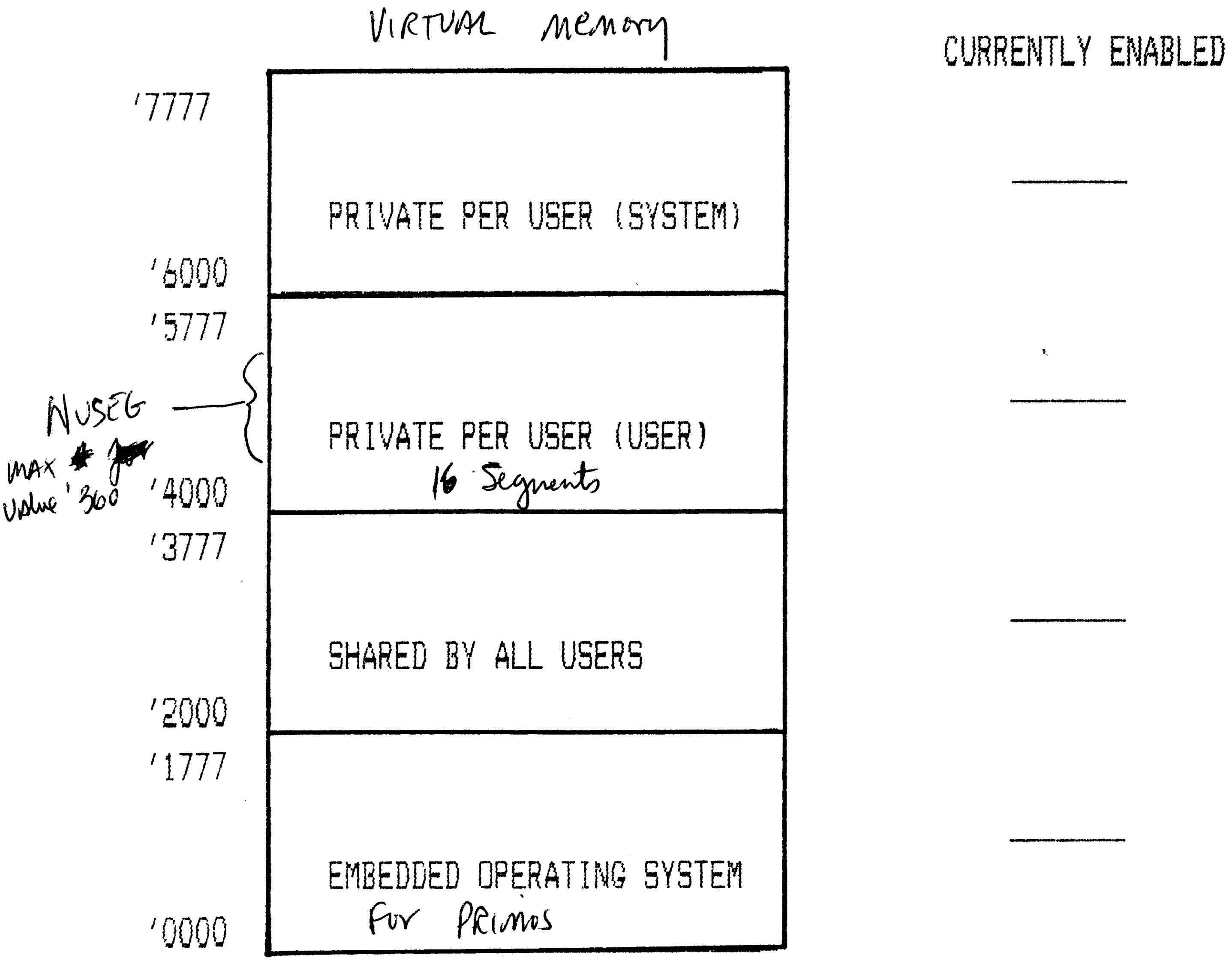
Interleaving is implemented using two identical boards.

One board contains the even addresses, the other board contains the odd addresses.

This has the effect of speeding up sequential access and increasing the cache hit rate.

SEGMENTATION - Dividing up of Virtual Memory

Virtual Memory is divided into variable length SEGMENTS (64K words max)
4096 SEGMENTS of 64K words define 512 MB of Virtual Memory.
The Virtual address space is divided into 4 areas (DTARs),
each area consisting of 1024 ('2000) segments.

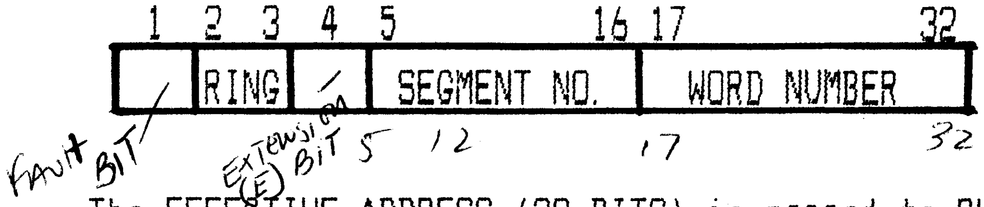


Virtual ~~memory~~ is determined by the # of Bits Available

EFFECTIVE ADDRESS FORMAT

PROGRAM INSTRUCTIONS GENERATE AN EFFECTIVE ADDRESS (EA).

- 2 Bits RING NUMBER (defines privileges)
- 12 Bits SEGMENT NUMBER
- 16 Bits WORD NUMBER (within SEGMENT)



The EFFECTIVE ADDRESS (28 BITS) is mapped to PHYSICAL MEMORY.

- 22 Bits PHYSICAL ADDRESS
- Up to 8M Bytes of PHYSICAL MEMORY.

RING NUMBER

There are 3 RINGS which define the privileges of access to the SEGMENT.

RING 0 is the most privileged and allows unrestricted access to all segments. Ring 0 is the only ring that can execute restricted instructions. PRIMOS runs in RING 0.

RING 1 Not currently used by software

RING 3 The least privileged. USERS run in RING 3.

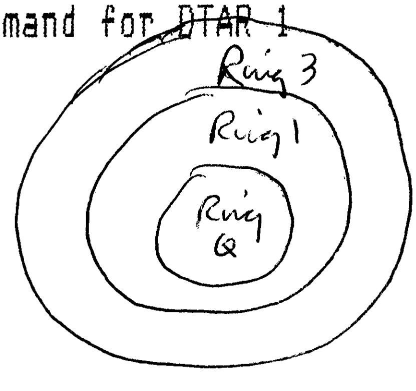
Seg 5 - only Routines that can be called

Hardware defines access rights of:

Inner ring accessing memory in an outer ring.

Outer ring accessing memory in an inner ring.
GATE access

The SHARE command for ~~DTAR 1~~



only (WATCH can SHARE to lower Ring) command can allow ring protection to be overridden

Ring
Share
Detar } 3 separate interrelated functions

MEMORY MANAGEMENT TECHNIQUES

8192 seg on Rev 19.1

The total number of segments available is currently 1022.
All 1022 segments cannot be contained in physical memory.
Virtual Memory is divided into two parts:

1) the part in physical memory
2) the part on the paging disk

Note: each user has a different Segment with max addressing of 4

Certain information is too critical to be on the paging disk, it is "WIRED" ("LOCKED") into physical memory.
At COLD START, PRIMOS "wires" critical information, this area will grow as PRIMOS requires certain per-user data to be wired.
When user segments are allocated, paging space is allocated.

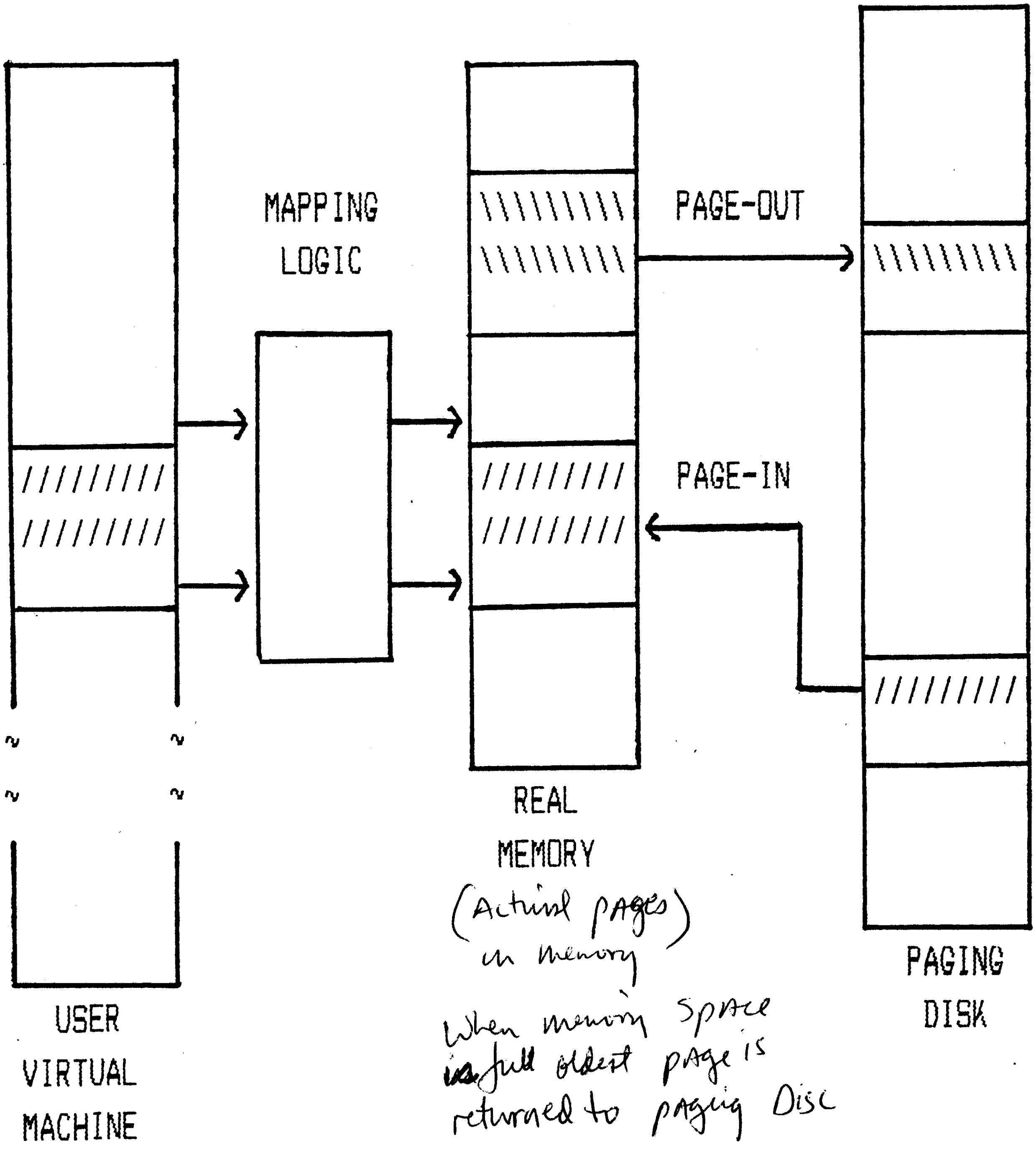
Virtual Memory on memory or on disc

Programs generate VIRTUAL ADDRESSES.
The VIRTUAL ADDRESS is translated (mapped) to a main memory address.
If the required physical address is resident within physical memory, the access may proceed without interruption.
If not in physical memory, a PAGE FAULT will occur.

When a PAGE FAULT does occur, the program is suspended while the required page is moved from the PAGING DISK into main memory.
This is called PAGING IN.
If there is no physical memory page available, PRIMOS will use a Approximately-Least-Recently-Used algorithm to determine which page in physical memory will be PAGED OUT to allow space for the in-coming page.

Segment divides up Virtual Memory

MEMORY MANAGEMENT



(Actual pages) in memory

When memory space is full oldest page is returned to paging Disc

PAGE FAULT (Access then proceeds)

Page = 1024 words (1K word)

Paging divids up physical memory

ADDRESS TRANSLATION

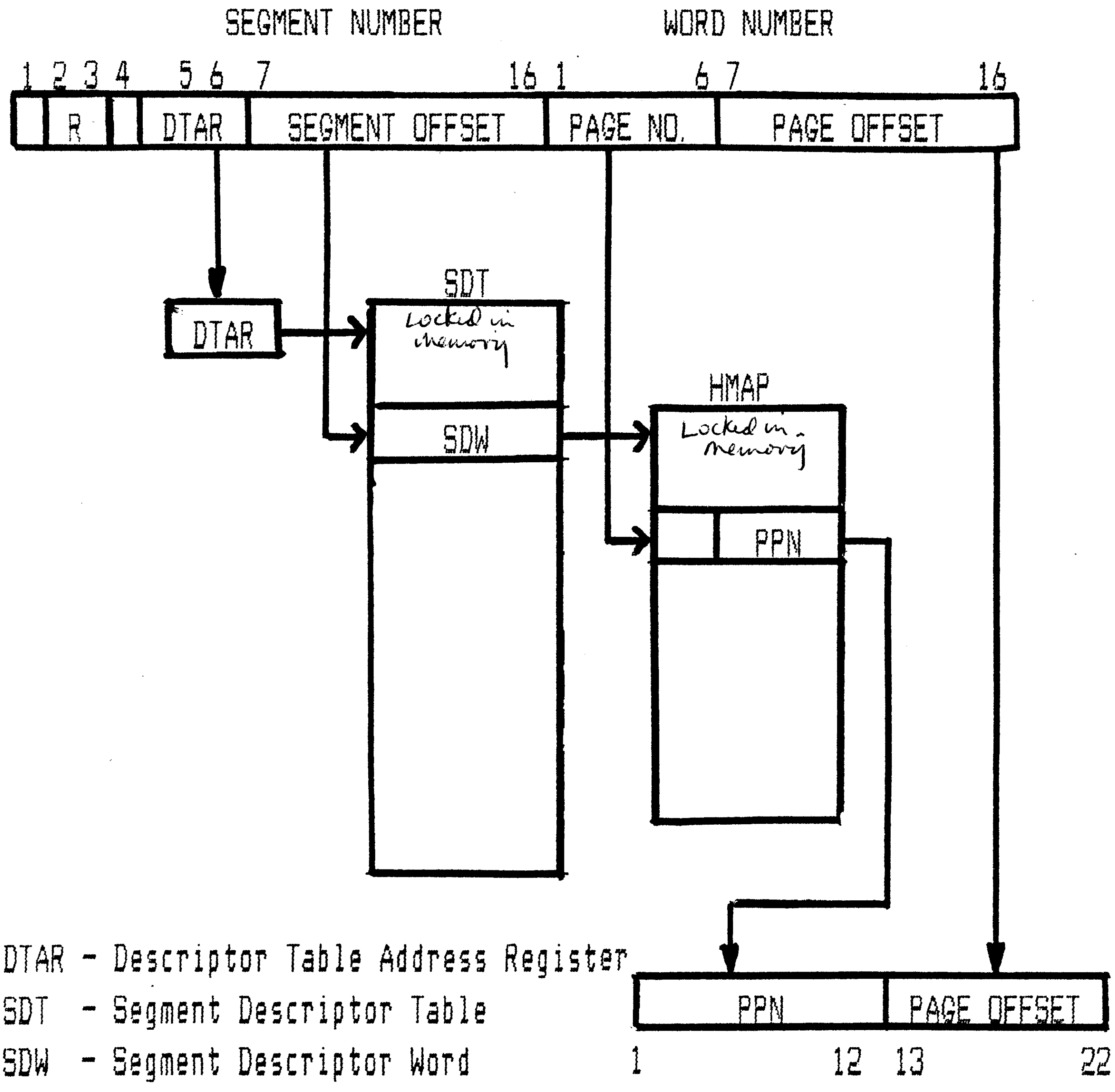
Every VIRTUAL ADDRESS is translated (mapped) to a physical address by accessing the STLB (Segmentation Translation Lookaside Buffer). The STLB holds the 64 most recent virtual to physical address translations. When the STLB does not have a valid entry for the virtual address to be translated, hardware calculates the address translation using Descriptor Table Address Registers, Segment Descriptor Tables and Hardware Page Maps. The STLB is accessed again, this time being sure to get a STLB hit. During the translation, a page fault will occur if the desired page is not in physical memory.

Simultaneous to the STLB access, hardware starts a CACHE access. If the word from cache is from the correct physical page, then the access is complete. If the word sought is not a valid cache entry, then the information is brought into cache from physical memory.

In summary fastest to slowest:

```
                              STLB 'hit'  +  CACHE 'hit'
                              STLB 'hit'  +  MEMORY 'hit', CACHE 'hit'
                                            (CASHE MISS)
            full translation, STLB 'hit'  +  CACHE 'hit'
            full translation, STLB 'hit'  +  MEMORY 'hit', CACHE 'hit'
full translation (PAGE FAULT), STLB 'hit'  +  MEMORY 'hit', CACHE 'hit'
```

FULL ADDRESS TRANSLATION



DTAR - Descriptor Table Address Register

SDT - Segment Descriptor Table

SDW - Segment Descriptor Word

HMAP - Hardware page MAP

PPN - Physical Page Number

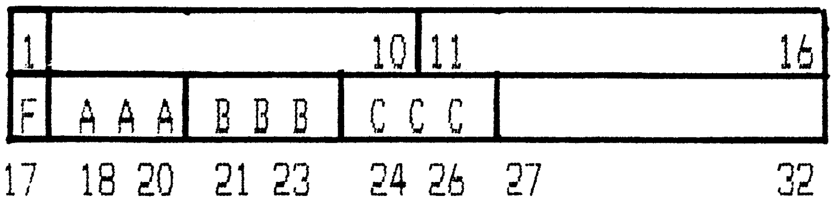
DTAR - DESCRIPTOR TABLE ADDRESS REGISTER

<table>
<tr><td>1</td><td>10</td><td>11 16</td></tr>
<tr><td>17</td><td colspan="2">18 32</td></tr>
</table>

Bits 1-10 = 1024 minus number of entries in SDT
11-16 = High order 21 bits of physical address
18-32 of SDT origin
17 = must be zero

Physical pointers for # of entries

SDW - SEGMENT DESCRIPTOR WORD



Bits 27-32 = Physical address of Page Map Table (HMAP)

1-16 = (Bits 11-16 must be zero)

H map defines the segment
It must be zero

17 = Fault Bit

18-20 = (AAA) Access rights from RING 1

000	no access
001	Gate access only
010	Read access only
011	Read and write access
100	reserved
101	reserved
110	Read and execute access
111	Read, write, and execute access

21-23 = (BBB) reserved for future use

24-26 = (CCC) Access rights from RING 3
same as RING 1 access bits

HMAP - HARDWARE PAGE MAP ENTRY

22 Bits of address
in First 8 MB memory

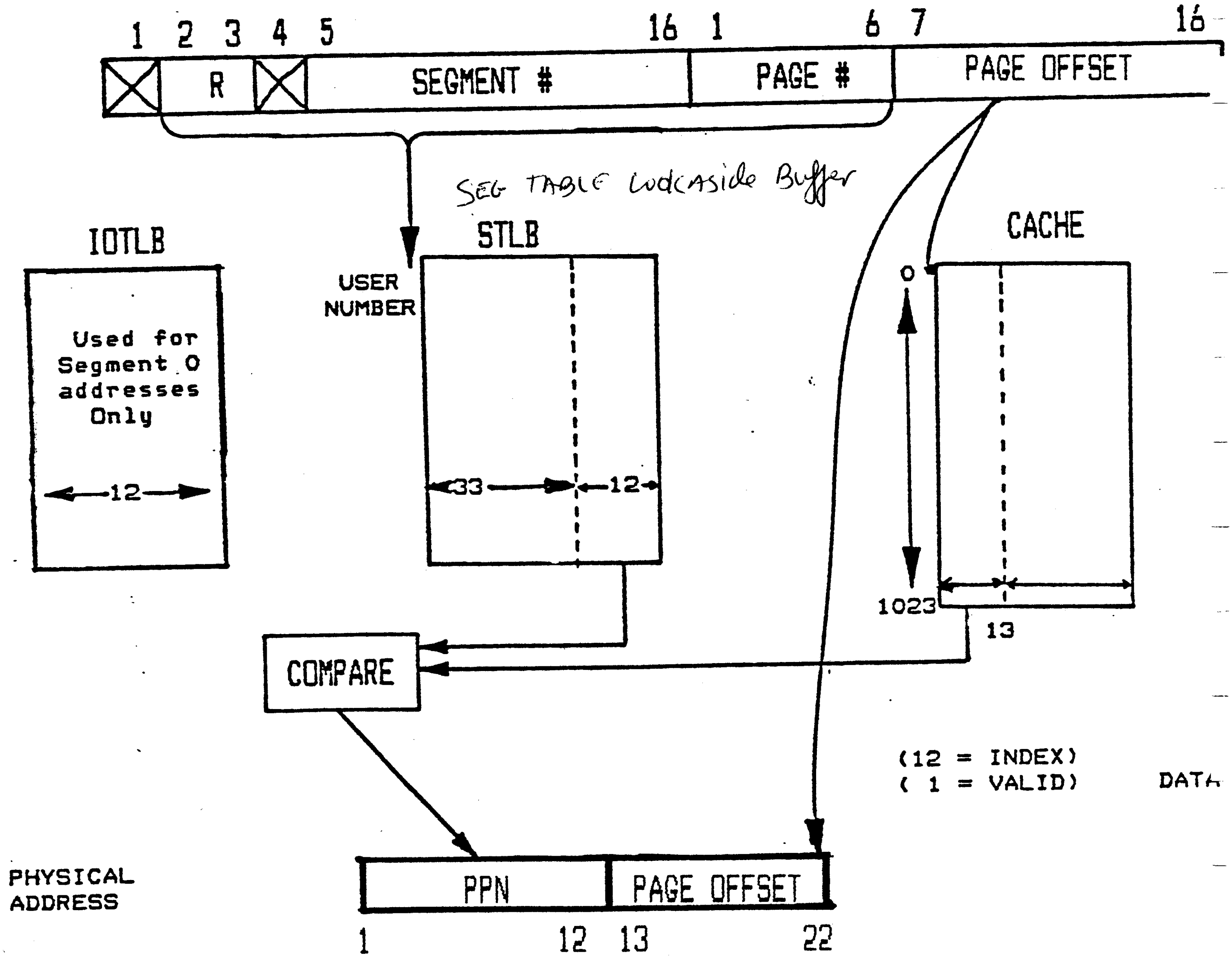
1	2	3	4	5 16
V	R	U	S	PPN

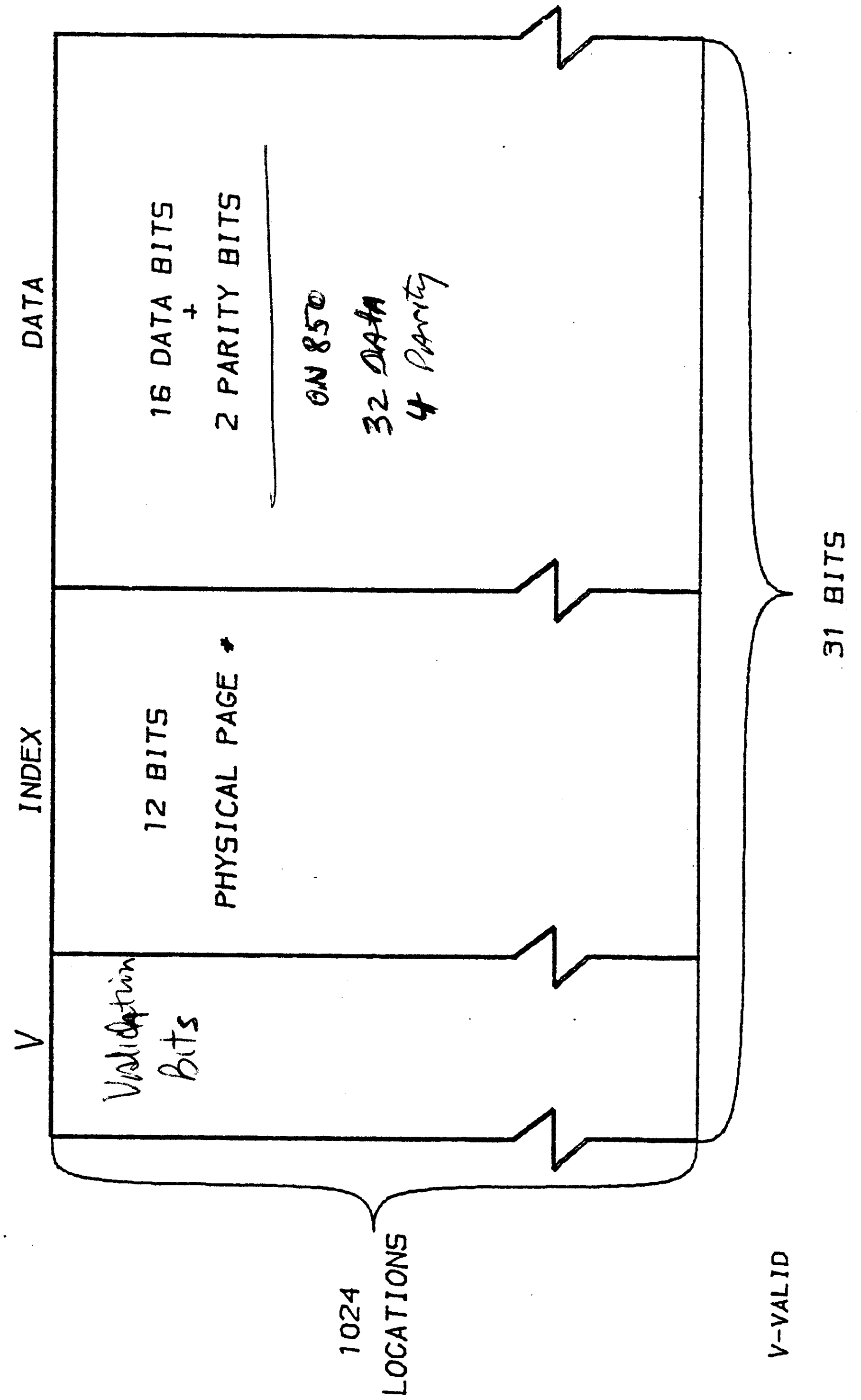
Bis 1 (V) = VALID Bit, set when page is in physical memory.
2 (R) = REFERENCED Bit, set by PAGTUR when the page is brought in.
3 (U) = UNMODIFIED Bit, reset by hardware whenever the page is modified.
4 (S) = SHARED Bit, set at cold start for memory pages, so that each location in the page is not put in cache.
5-16 = Physical Page Number (PPN)

(bits 3,5 indicate page status if Valid bit is reset)

See Handout
pg 3-13

VIRTUAL ADDRESS

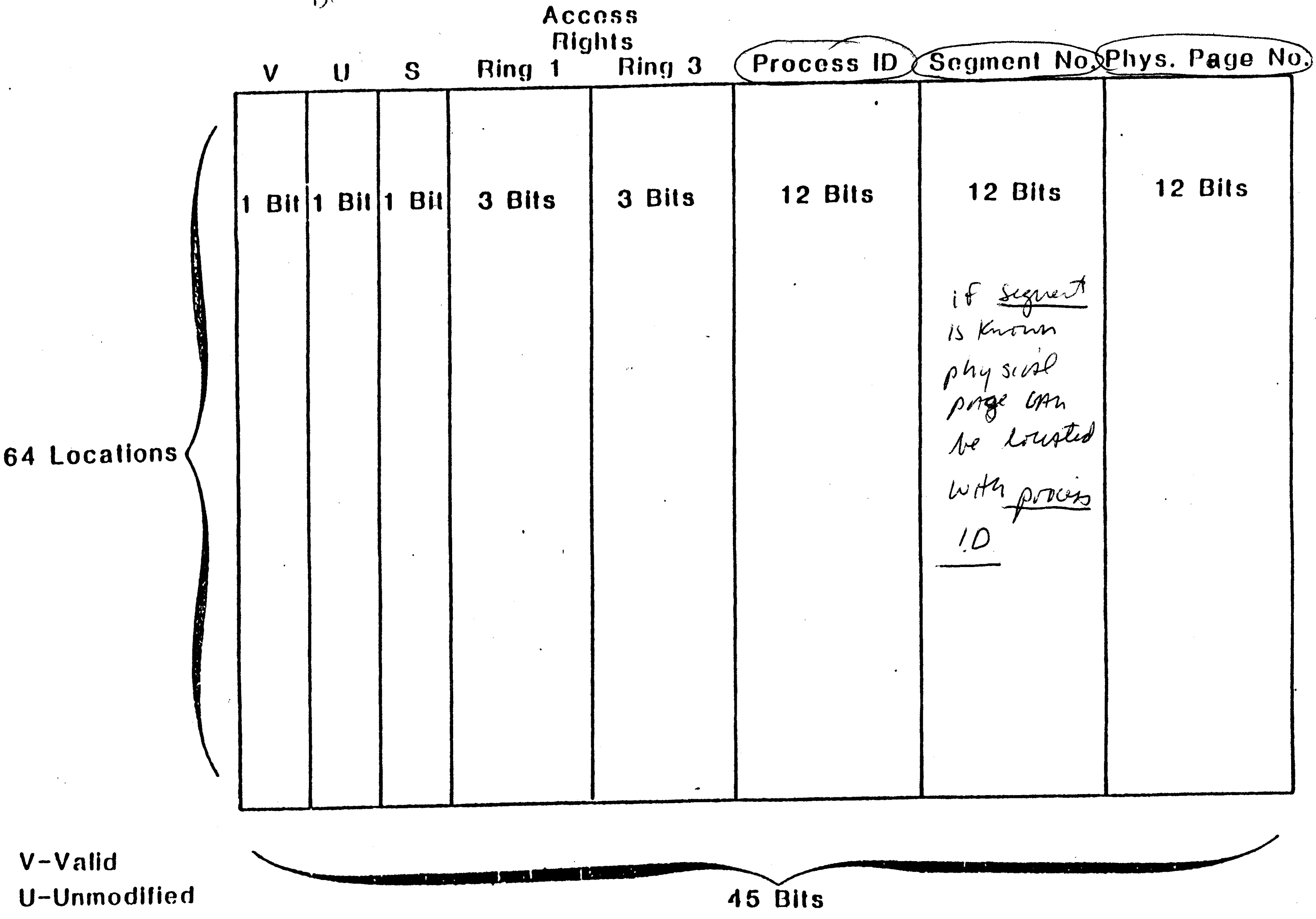



DATA
16 DATA BITS
+
2 PARITY BITS
ON 850
32 Data
4 Parity
INDEX
12 BITS
PHYSICAL PAGE #
V
Validation
Bits
31 BITS
1024
LOCATIONS
V-VALID

2 KB Cashe

STLB

Must Have Validation BIT

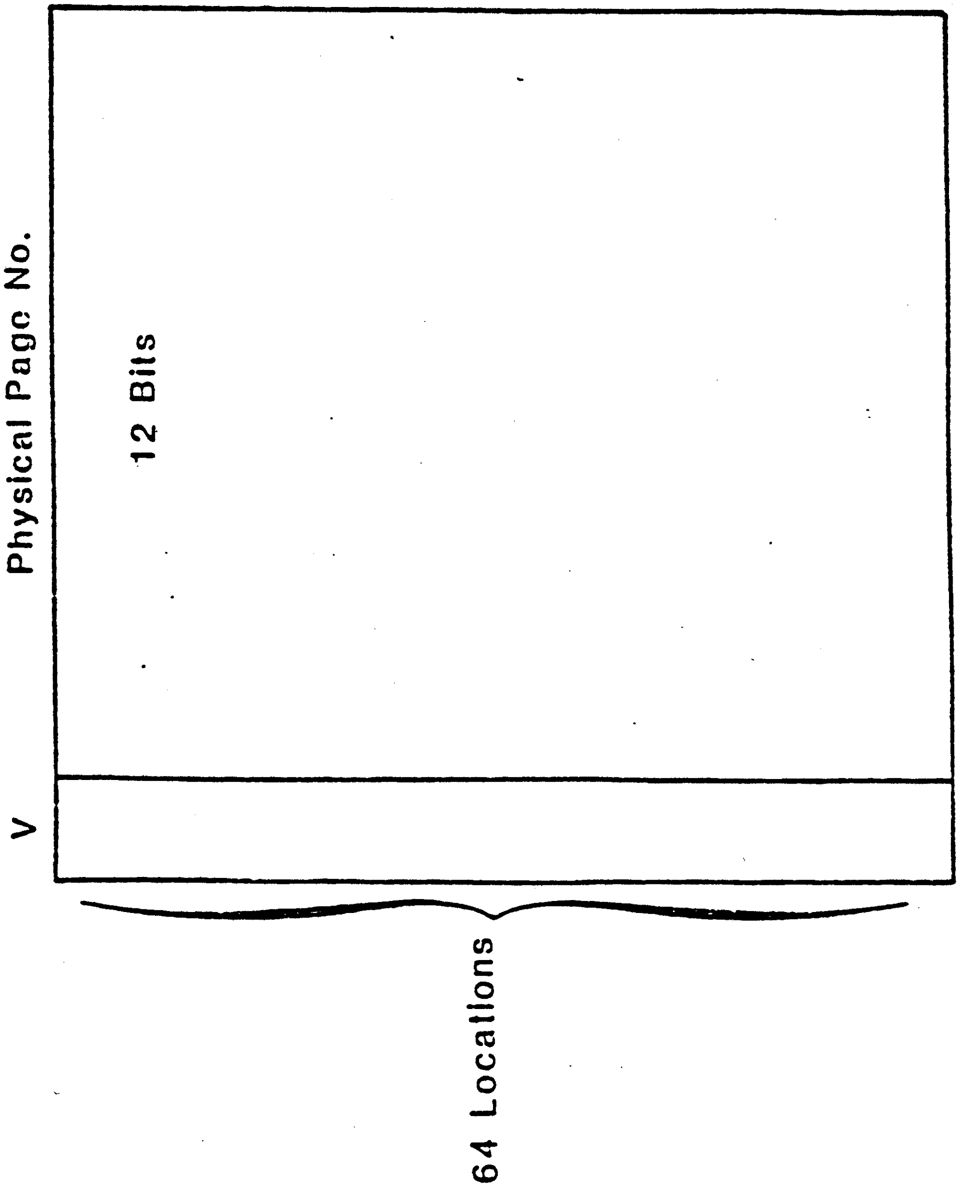


			Access Rights				
V	U	S	Ring 1	Ring 3	Process ID	Segment No.	Phys. Page No.
1 Bit	1 Bit	1 Bit	3 Bits	3 Bits	12 Bits	12 Bits	12 Bits

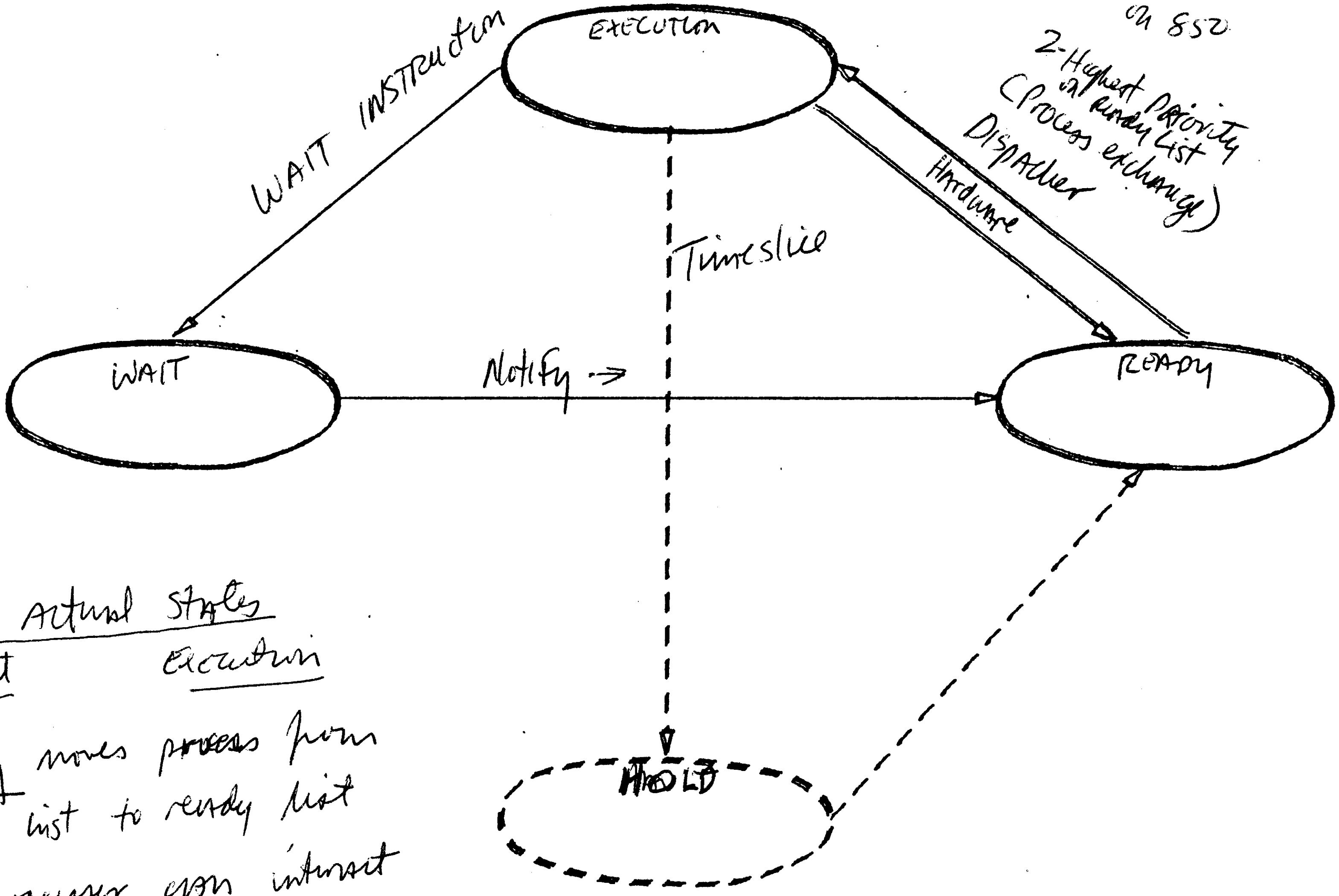
if segment is known physical page can be located with process ID

V-Valid
U-Unmodified
S-Shared Page

IOTLB



Section 4 - Process Exchange



2 Actual States

Wait Execution

Notify moves process from wait list to ready list so processor can interact

PROCESS EXCHANGE

Process Exchange is the hardware/firmware mechanism used to switch the CP from being used by one user to being used by a different user.

A context switch occurs whenever a higher priority user or system requires the use of the CP. The context switch involves saving the registers and state of the currently running process and placing the needed information in the current register set for the new user or system. This is accomplished by the firmware/hardware and the two user register sets in the High Speed Register File.

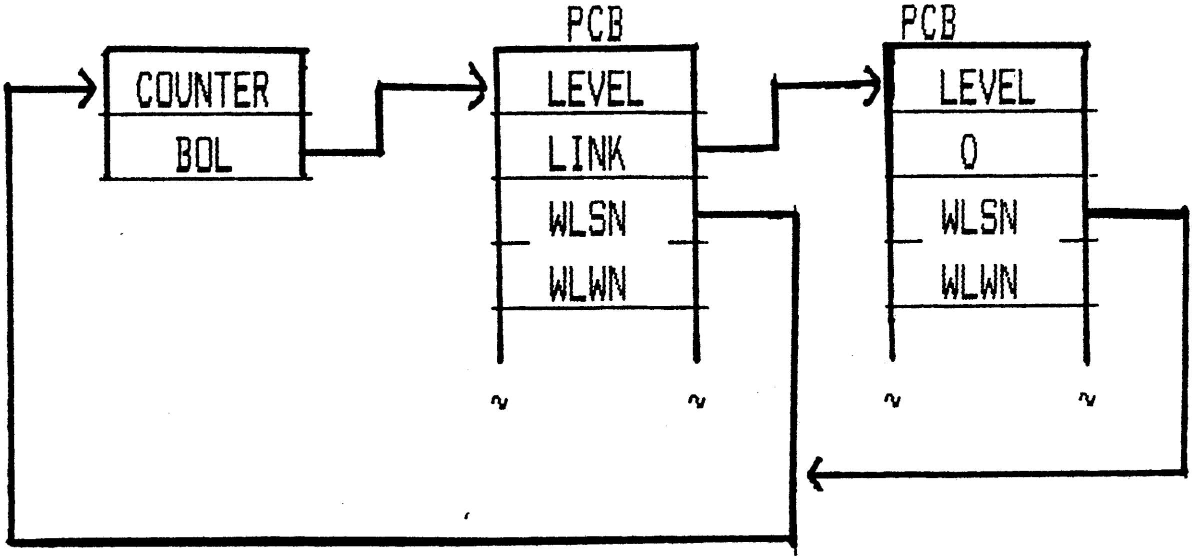
A process is a sequential flow of execution (a user, an I/O driver). The process is described to PRIMOS by a PCB (Process Control Block). Each process has its own PCB. A proces must be in one of two states:

1). waiting for an event or non-CP resource

2). ready to execute.

When the process has all the resources required to run and is only waiting for the CP, the process' PCB is placed on the READY LIST. If the process is waiting, its PCB is threaded onto a semaphore or wait list.

WAIT LIST (Semaphore)



Note: Queuing is priority order with FIFO for equal priority. However, there are different flavors of NOTIFY, Notify end or Notify beginning.

WAIT <semaphore name>

```
access semaphore
count = count + 1
if count > 0
     then PCB --> Wait List
OR else process continues
```

NOTIFY <semaphore name>

```
access semaphore
count = count - 1
first PCB --> Ready List
```

PROCESS CONTROL BLOCK

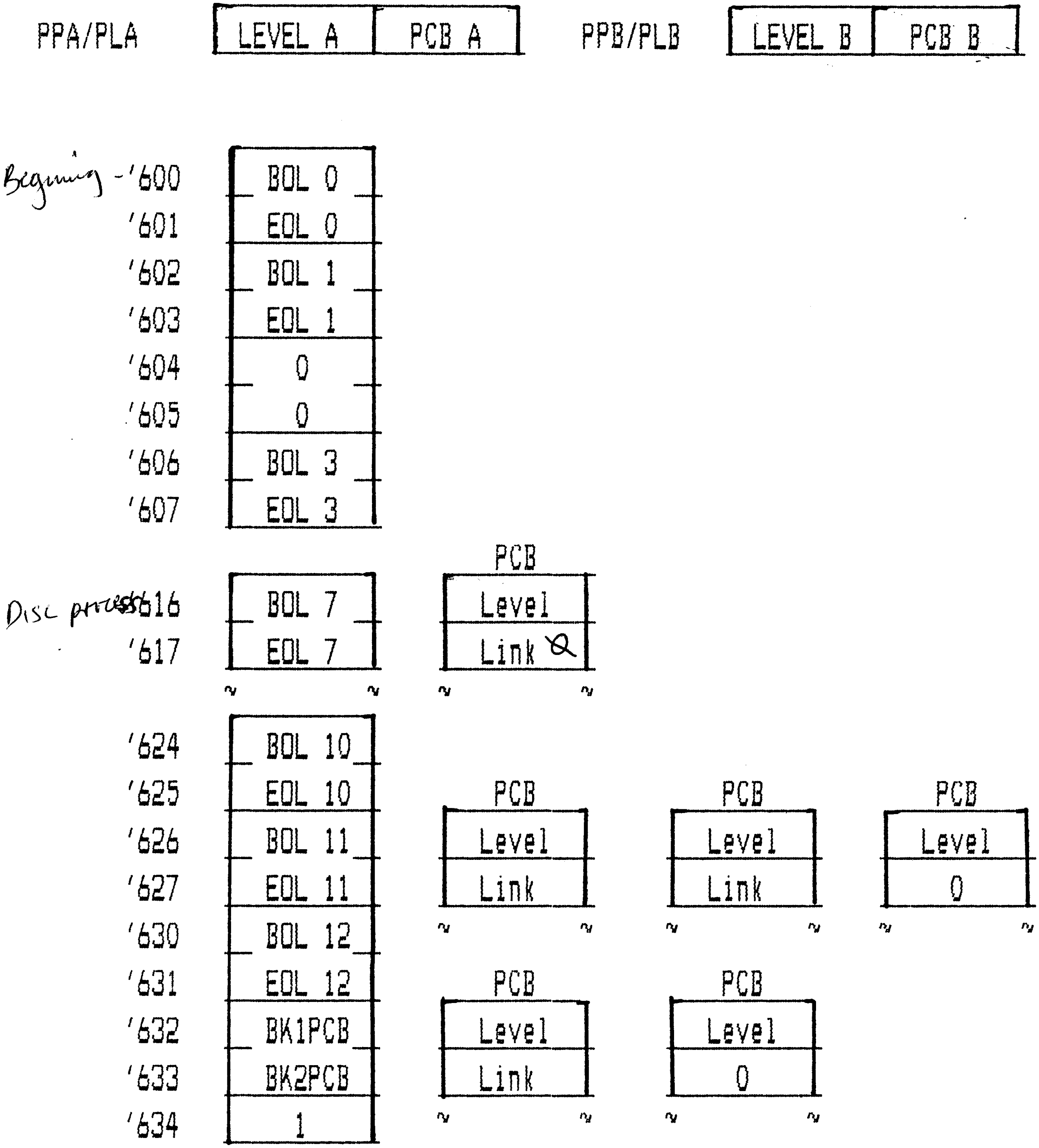
0	LEVEL (PRIORITY)	
1	LINK	Pts. to next process
2	POINTER TO WAIT LIST	
3	"	
4	ABORT FLAGS	
5	MULTISTREAM CONTROL	
6	RESERVED	
7	"	
'10	PROCESS ELAPSED TIMER	*Note DTAR 0,1 Are same for Everyone
'11	"	
'12	DTAR 2	
'13	"	
'14	DTAR 3	
'15	"	
'16	PROCESS INTERVAL TIMER	
'17	PROCESS INTERVAL TIMER	
'20	REGISTER SAVE MASK	
'21	KEYS	
'22		
..	REGISTER SAVE AREA	
'61		
'62	RING 0 FAULT VECTOR	
'63	"	
'64	RING 1 FAULT VECTOR	
'65	"	
'66	NOT USED	
'67		
'70	RING 3 FAULT VECTOR	
'71	"	
'72	PAGE FAULT VECTOR	Where you go to correct fault
'73	"	
'74	CONCEALED STACK FIRST FRAME PTR	
'75	CONCEALED STACK NEXT FRAME PTR	
'76	CONCEALED STACK LAST FRAME PTR	
'77	RESERVED	

READY LIST

Based on How long it takes to Run

LEVEL		
0	CLOCK PROCESS/FNTSTOP(clock 2)	
1	AMLC PROCESS (Character in/output	
2	SMLC PROCESS	
3	MPC PROCESS, MP2 (Parallel Printer	
4	VERSATEC PROCESS, MPC-4	
6	RING NET CONTROLLER PROCESS –	Node Controller
7	SPARE	
D	DISK PROCESS	
8	SUPERVISOR PROCESS	
9	USER LEVEL 3	
10	USER LEVEL 2	
11	USER LEVEL 1 (DEFAULT LEVEL) –	User Default Level
12	USER LEVEL 0	
13	BK1PCB (BACKSTOP 1) CPU #1	
	BK2PCB (BACKSTOP 2) CPU #2	
14	END OF READY LIST = 1	

READY LIST EXAMPLE #1

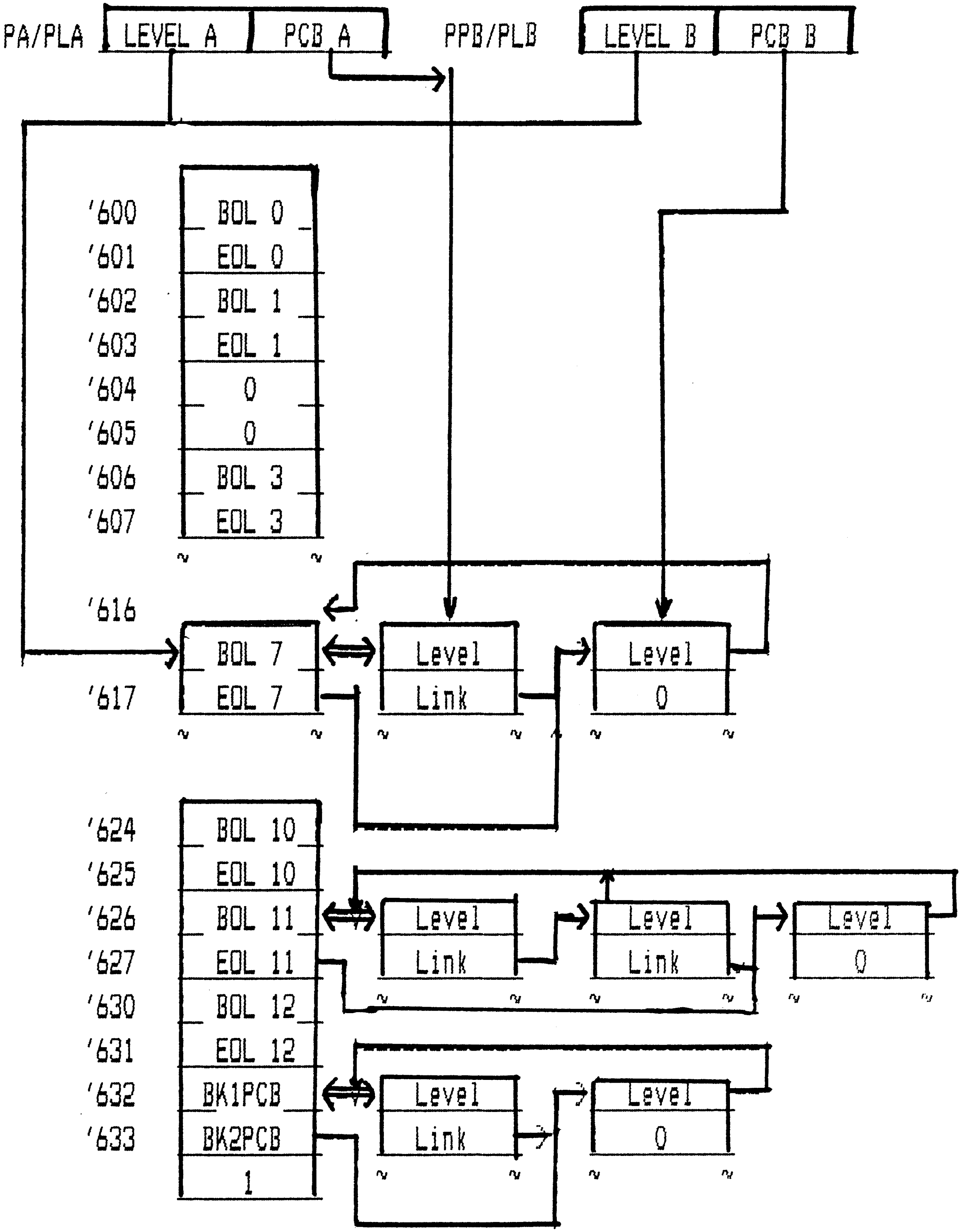


To move a PCB from the Ready List to a Wait List, the WAIT instruction is used. The NOTIFY instruction will move a process from a wait list to the Ready List. Both instructions must always reference a semaphore or wait list. The NOTIFY removes the first PCB from the semaphore and places it onto the Ready List at the proper level. When the process has completed execution or requires another resource, a WAIT is executed and the process moves from the Ready List to the specified Wait List or semaphore. PCBs are placed in the Wait List queue in priority level order.

READY LIST

The firmware dispatcher uses two locations in the High Speed Register File Group 0. The first location is called PPA/PLA. PPA holds the pointer to the PCB of the currently running process. PLA contains the Ready List level of the currently running process. The currently running process will be the highest priority process on the Ready List. PPB contains the PCB address of the next process to run. PLB has the level of the next process. This allows the User Register Set for the next process to be set up while still running another process at a higher level.

READY LIST EXAMPLE #2



The Ready List and the PCBs are all in Segment 4. This is one of the 'wired' segements of PRIMOS. This means it never gets paged out to the paging disc. The Ready List begins at Segment 4, address '600 and extends through address '634.

The PCB address and User Number bear a direct relationship to one another. For example; the address for User 1's PCB is 100100. The address for User 7's PCB is 100700. The PCB at address 101200 belongs to User 10. Addresses are in octal, user numbers are decimal. All PCBs are 64 ('100) words long so the least significant two octal digits of any PCB address is '00.

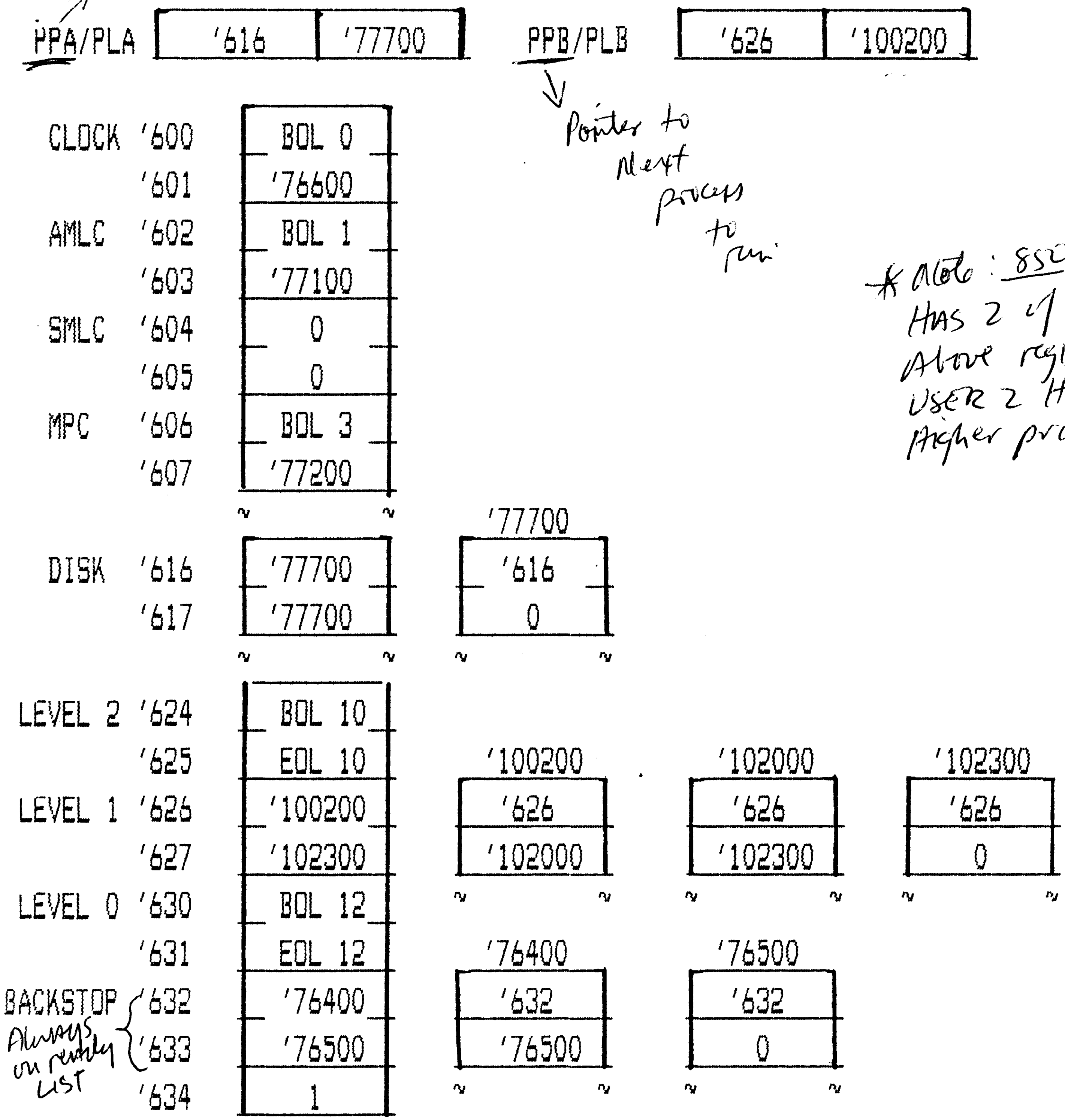
Ready list

Begining of list pointer
End of list pointer

PCB 100100
100200
100300

these 3 #'s give
you

READY LIST EXAMPLE #3



* Note: 850 Has 2 of Above registers USER 2 Has Higher priority

This example shows actual addresses found using VPSD on Rev 18.2

The contents pf PPA/PPB are calculated.

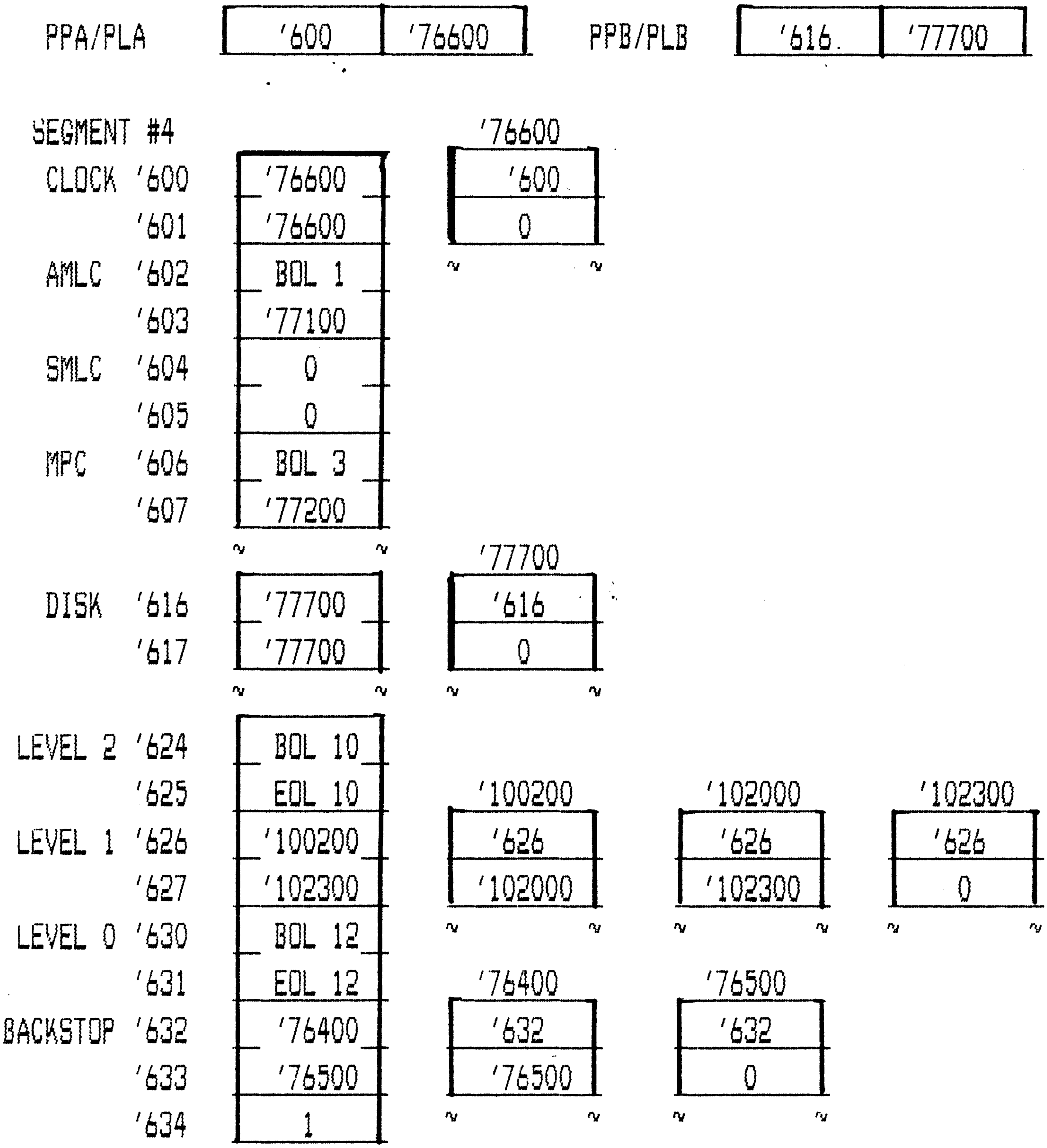
(on Micro Code)

PPA is currently Running

In Example #3, PLA points to the currently active level (Disk) and PPA points to the PCB of the currently running process. The Disk Driver is now the highest priority process on the Ready List. PLB and PPB contain the level and PCB address of the next process to run. In our example, the next process happens to be User 2.

A CLOCK interrupt occurs. The interrupting controller places its address on the CPU bus. The currently running process is suspended at the completion of the current instruction. The firmware uses the controller address as an index or vector into the interrupt segment which is also segment 4. At this address is a pointer to the Interrupt Response Code (IRC) which handles the interrupts from this particular controller. This code is not associated with any specific process and cannot have a PCB of its own. The IRC can do no more than acknowledge the interrupt and schedule the device driver to actually handle the event. This code is called the PHANTOM INTERRUPT CODE or PIC. The PIC will acknowledge the interrupt and execute an INEC (Interrupt Notify to End of list and Clear active interrupt). For a clock interrupt, the INEC will reference the semaphore CLKSEM. The INEC causes the clock to be scheduled on the READY LIST by moving the PCB from the Wait List to the appropriate level on the Ready List. PRIMOS has assigned the Clock the highest priority and all clock interrupts are placed on the Ready List at address '600 or level 0. If location '600 contains a zero, the address of the PCB is placed into location '600. If '600 is not zero, the firmware will access '601 and thread the new PCB onto the end of the chain.

READY LIST EXAMPLE #4



PPA/PLA	'600	'76600
PPB/PLB	'616	'77700

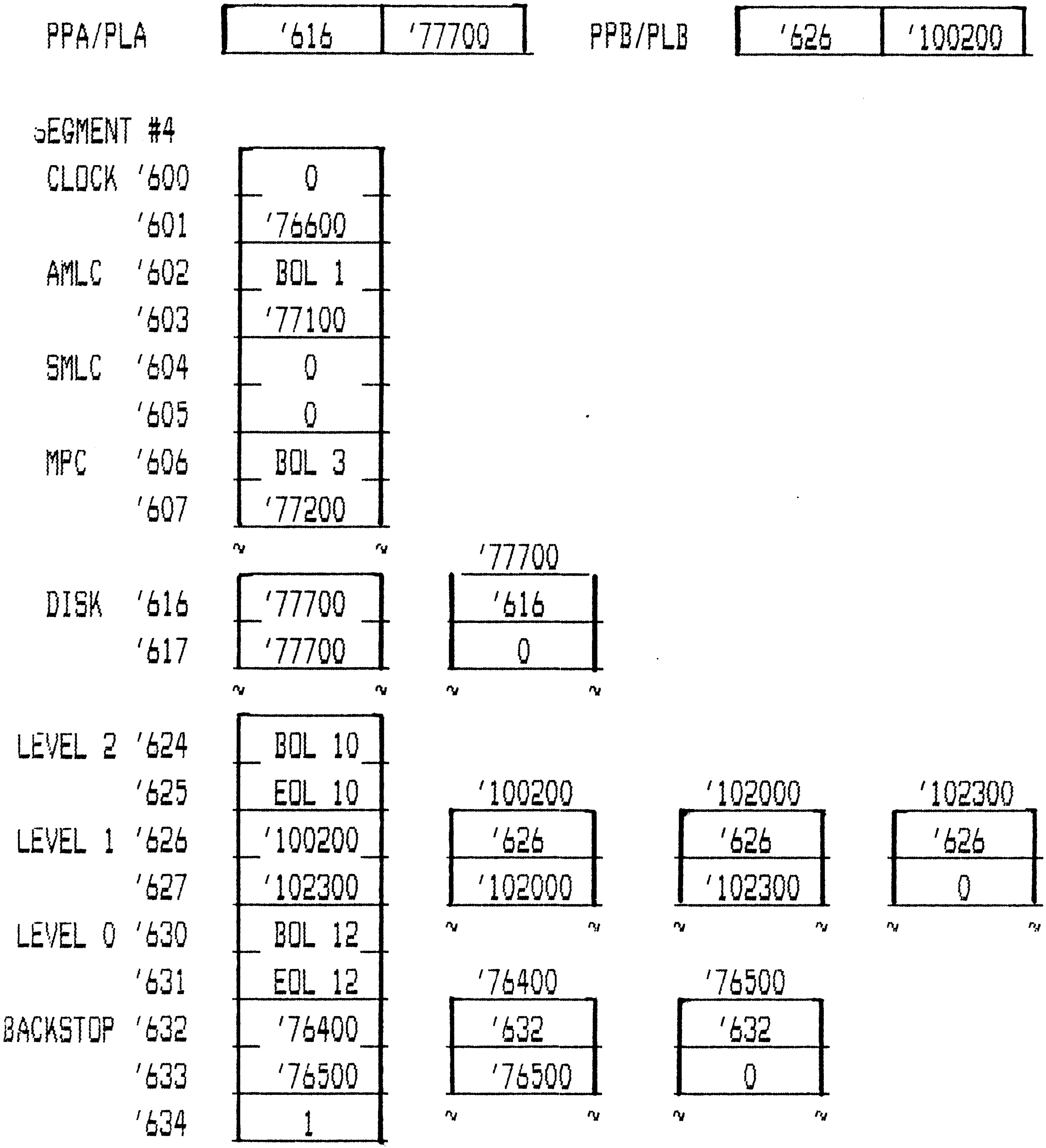
SEGMENT #4		
CLOCK	'600	'76600
	'601	'76600
AMLC	'602	BOL 1
	'603	'77100
SMLC	'604	0
	'605	0
MPC	'606	BOL 3
	'607	'77200
DISK	'616	'77700
	'617	'77700
LEVEL 2	'624	BOL 10
	'625	EOL 10
LEVEL 1	'626	'100200
	'627	'102300
LEVEL 0	'630	BOL 12
	'631	EOL 12
BACKSTOP	'632	'76400
	'633	'76500
	'634	1

'76600	'77700	'100200	'102000	'102300	'76400	'76500
'600	'616	'626	'626	'626	'632	'632
0	0	'102000	'102300	0	'76500	0

The NOTIFY instruction causes the firmware dispatcher to update the contents of PPA and PPB. As the clock interrupt is a higher priority than that of the currently running process, the contents of PPA/PLA is moved to PPB/PLB and the Clock's PCB address and level are placed into PPA/PLA.

The clock driver will now run to completion. At the completion of the driver routine a WAIT CLKSEM will be executed. This removes the clock's PCB from the Ready List, places it on the CLKSEM Wait List, and allows the dispatcher to move PPB/PLB to PPA/PLA and update PPB/PLB for the next ready process. PPB/PLB is updated by the dispatcher performing a scan of the Ready List. This is done by comparing the BOL (Beginning Of List) and EOL (End Of List) for this level. If they are not equal, the next process is on the same level and PPB/PLB are updated. If they are equal, the next word (BOL for the next level) is checked. If this value is not zero, then the next process is on this level and PPB/PLB are updated. If BOL is zero, there is no ready processes on this level and the next level's BOL will be checked. This procedure will continue until PPB/PLB are updated with a PCB address and a process' level.

READY LIST EXAMPLE #5



READY LIST EXAMPLE #6

PPA/PLA			PPB/PLB		
	'626	'100200		'626	'102000

SEGMENT #4

CLOCK	'600	0
	'601	'76600
AMLC	'602	BOL 1
	'603	'77100
SMLC	'604	0
	'605	0
MPC	'606	BOL 3
	'607	'77200
~		~
DISK	'616	0
	'617	'77700
~		~
LEVEL 2	'624	BOL 10
	'625	EOL 10
LEVEL 1	'626	'100200
	'627	'102300
LEVEL 0	'630	BOL 12
	'631	EOL 12
BACKSTOP	'632	'76400
	'633	'76500
	'634	1

'100200	'102000	'102300
'626	'626	'626
'102000	'102300	0

'76400	'76500
'632	'632
'76500	0

The process at the head of User Level 1 will now run until it completes execution, requires another resource, does an I/O operation, a fault occurs, or the process' time slice is used up. All of these conditions cause the PCB to be removed from the Ready List and placed on the appropriate Wait List. The firmware then dispatches the next PCB to PPB/PLB.

When a process terminates "normally" (runs to completion), PRIMOS places the process' PCB on that User's BUFSEM Wait List. BUFSEM is the semaphore the User waits on while entering commands and typing at the terminal.

If a process is terminated because of a time-slice end, the process' PCB is placed on a lower priority queue dependent upon which how much CP time the process has used and the User priority level.

READY LIST EXAMPLE #7

PPA/PLA	'626	'102000	PPB/PLB	'626	'102300

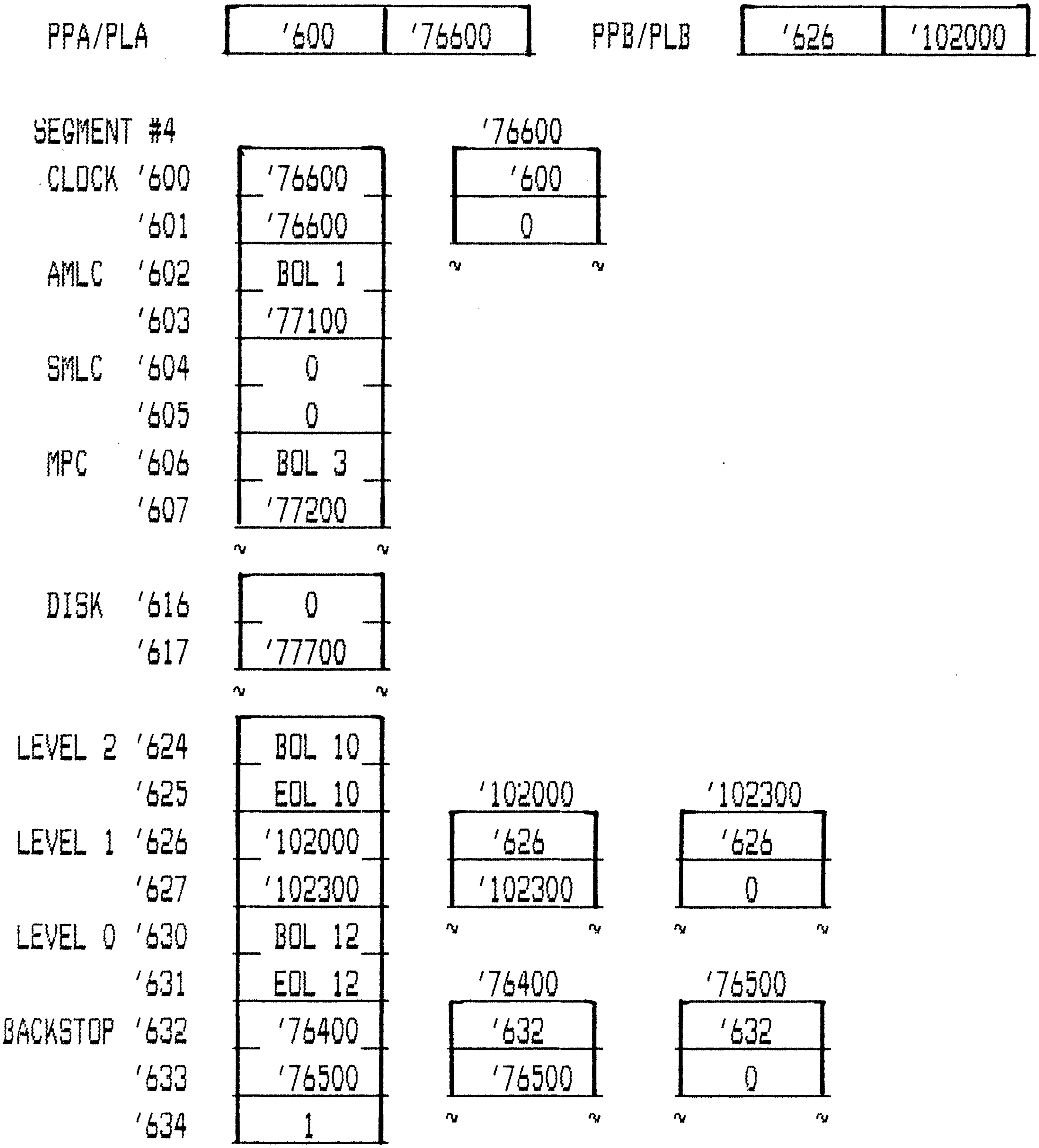
SEGMENT #4

CLOCK	'600	0
	'601	'76600
AMLC	'602	BOL 1
	'603	'77100
SMLC	'604	0
	'605	0
MPC	'606	BOL 3
	'607	'77200
DISK	'616	0
	'617	'77700
LEVEL 2	'624	BOL 10
	'625	EOL 10
LEVEL 1	'626	'102000
	'627	'102300
LEVEL 0	'630	BOL 12
	'631	EOL 12
BACKSTOP	'632	'76400
	'633	'76500
	'634	1

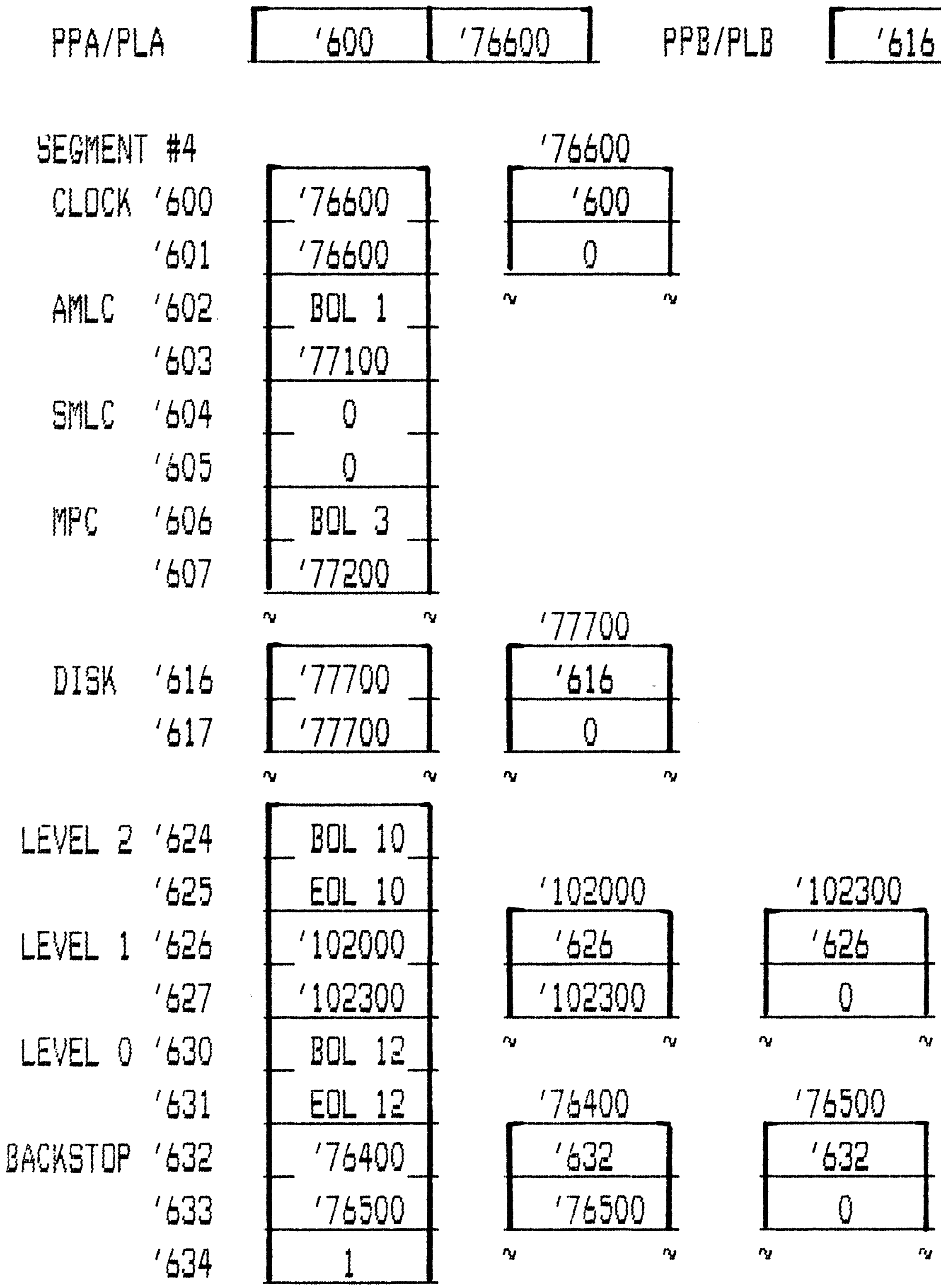
'102000	'102300
'626	'626
'102300	0

'76400	'76500
'632	'632
'76500	0

READY LIST EXAMPLE #8



READY LIST EXAMPLE #9



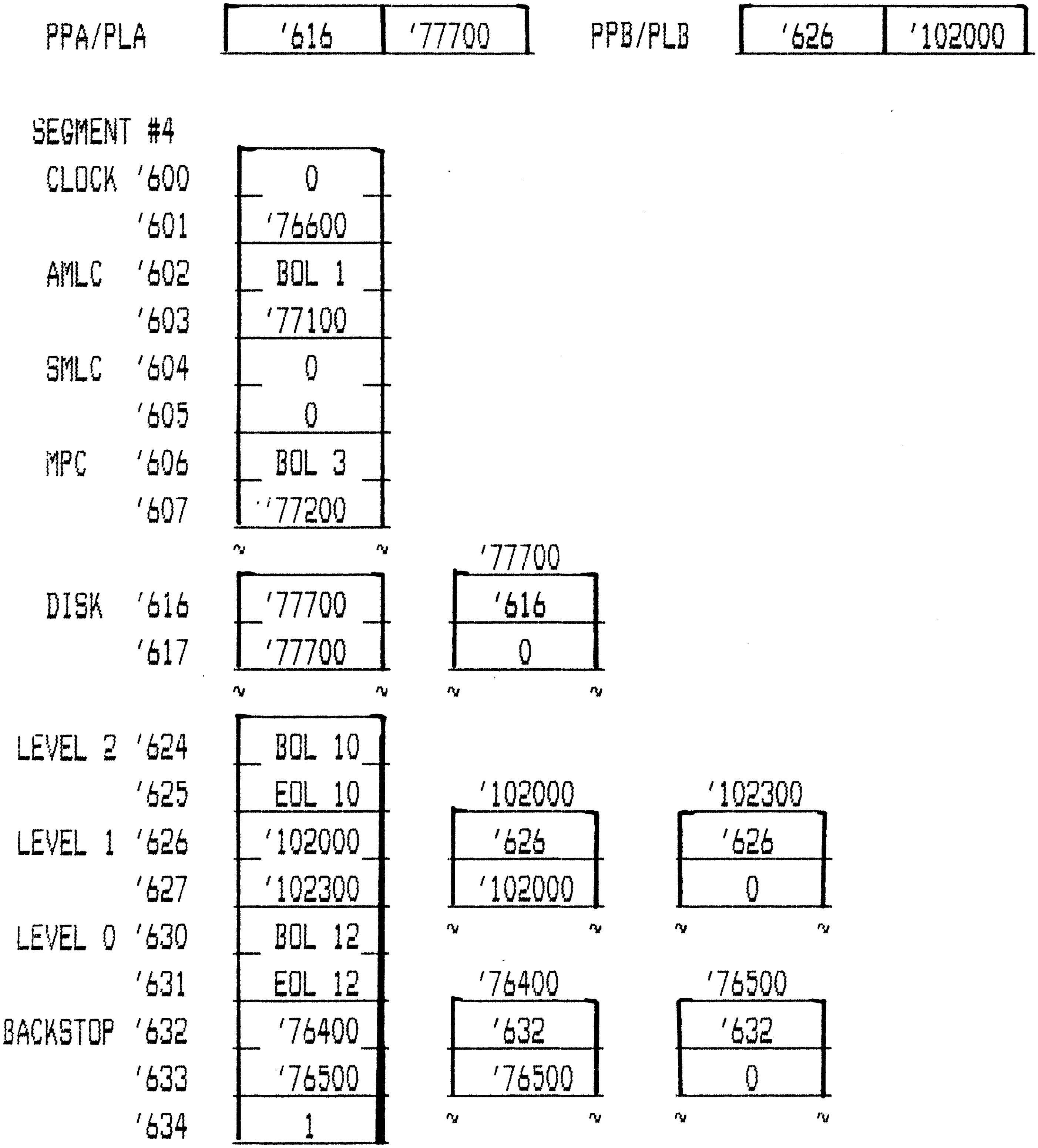
PPA/PLA	'600	'76600
PPB/PLB	'616	'77700

SEGMENT #4

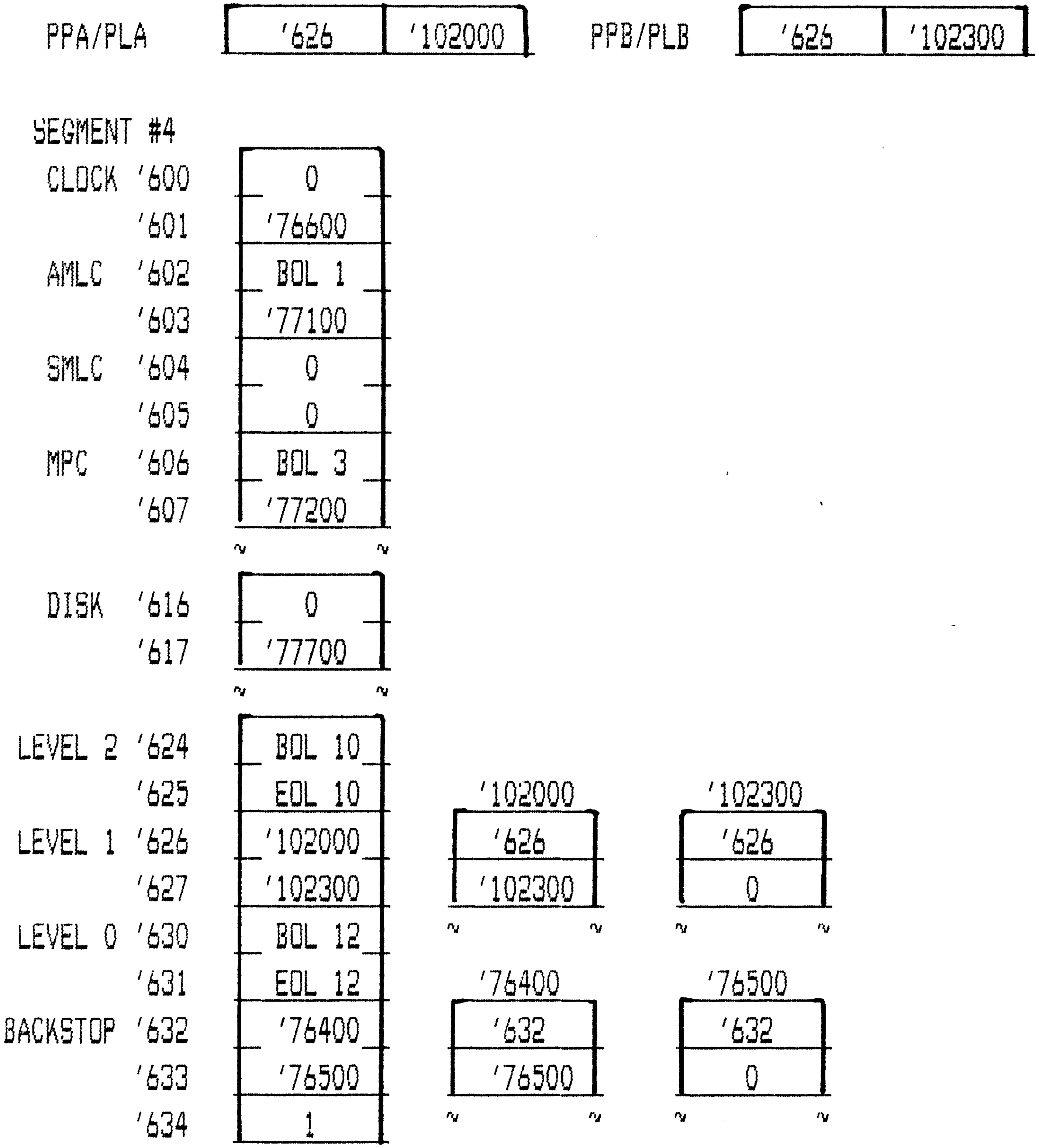
CLOCK	'600	'76600
	'601	'76600
AMLC	'602	BOL 1
	'603	'77100
SMLC	'604	0
	'605	0
MPC	'606	BOL 3
	'607	'77200
DISK	'616	'77700
	'617	'77700
LEVEL 2	'624	BOL 10
	'625	EOL 10
LEVEL 1	'626	'102000
	'627	'102300
LEVEL 0	'630	BOL 12
	'631	EOL 12
BACKSTOP	'632	'76400
	'633	'76500
	'634	1

'76600	'77700	'102000	'102300	'76400	'76500
'600	'616	'626	'626	'632	'632
0	0	'102300	0	'76500	0

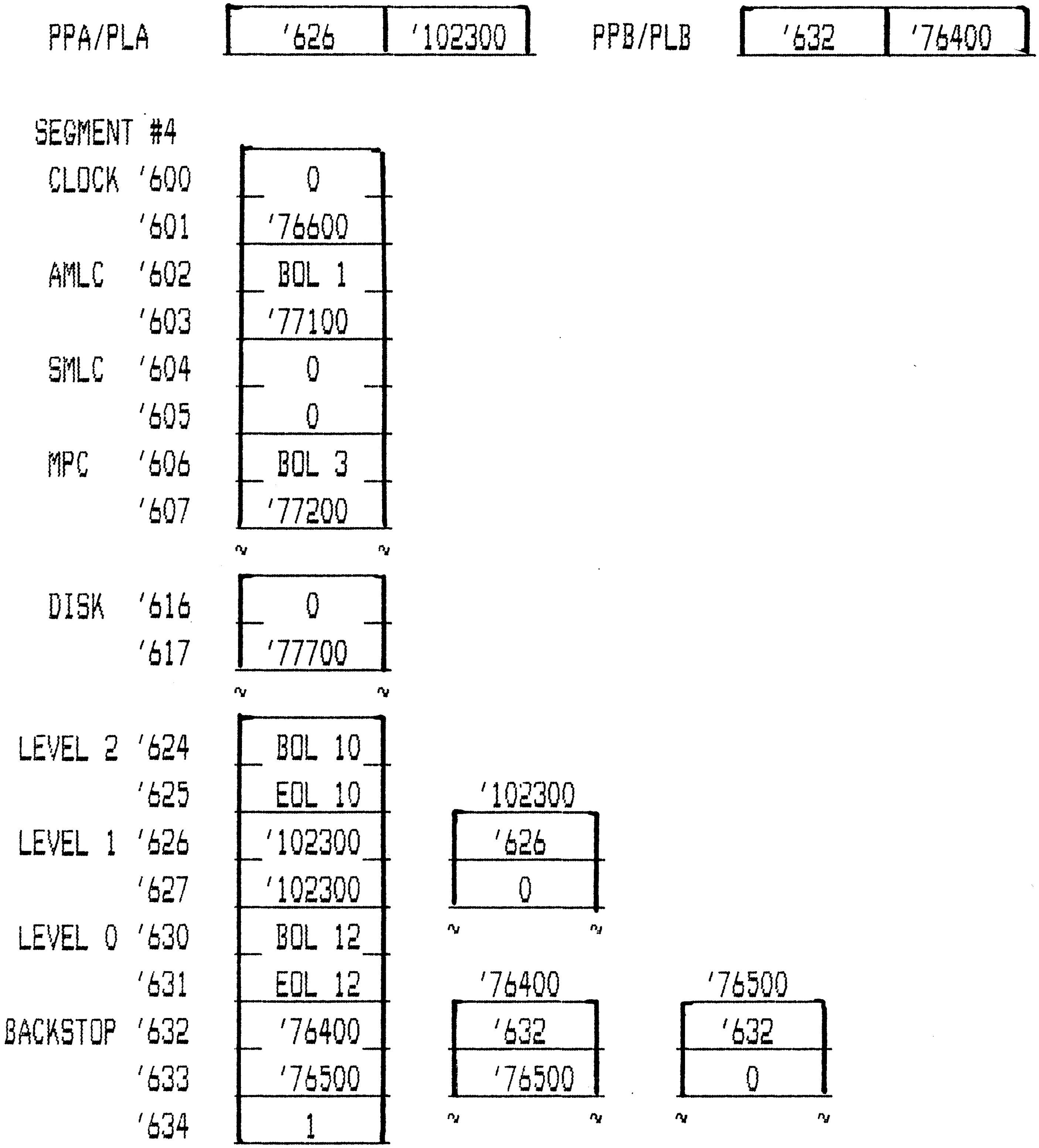
READY LIST EXAMPLE #10



READY LIST EXAMPLE #11



READY LIST EXAMPLE #12



READY LIST EXAMPLE #13

PPA/PLA	'632	'76400
PPB/PLB	'632	'76500

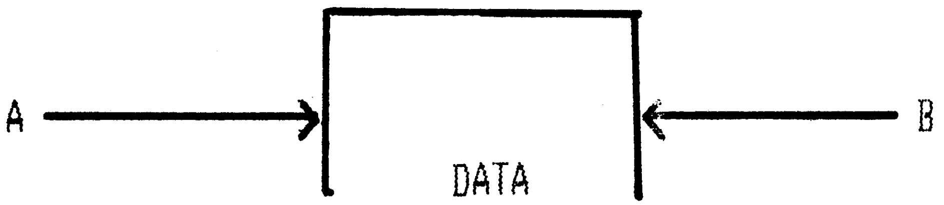
SEGMENT #4

CLOCK	'600	0
	'601	'76600
AMLC	'602	BOL 1
	'603	'77100
SMLC	'604	0
	'605	0
MPC	'606	BOL 3
	'607	'77200
DISK	'616	0
	'617	'77700
LEVEL 2	'624	BOL 10
	'625	EOL 10
LEVEL 1	'626	0
	'627	'102300
LEVEL 0	'630	BOL 12
	'631	EOL 12
BACKSTOP	'632	'76400
	'633	'76500
	'634	1

'76400	'76500
'632	'632
'76500	0

The BACKSTOP processes PCBs are ALWAYS on the Ready List. The purpose of BACKSTOP is to call the SCHEDULER. The SCHEDULER is used to move any process which has taken a time-slice end or is on the 'HI-PRI' queue to Ready List with another time-slice. There are two BACKSTOPs as the P850 requires one BACKSTOP for each CP. .ej

USE OF LOCK SEMAPHORES - Simple Lock



Two processes are sharing the same data area. Process A could be changing data at the same time as Process B is reading the data. B may read incorrect data.

To prevent this, use a Simple Lock Semaphore (initial count = -1).

In order to access the data

   Process A must wait on the semaphore (count = 0)

   Process A proceeds

If Process B attempts to access the data it must first wait on the semaphore. (count = 1)

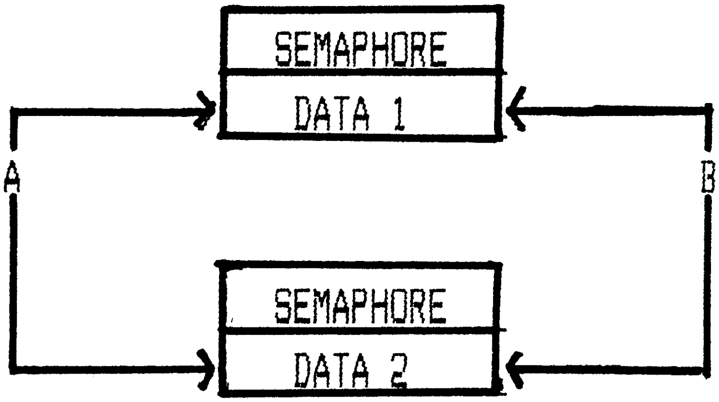
   Process B goes onto the Wait List for that semaphore

   Process A must NOTIFY the semaphore. (count = 0)

   Process B returns to the Ready List and proceeds

All processes that access the data must first WAIT on the semaphore and NOTIFY the semaphore when access is completed.

USE OF LOCK SEMAPHORES - Ordered Locks



Two processes are sharing two data areas.

If using simple locks;

Process A WAIT on semaphore 1
Process B WAIT on semaphore 2
Process B WAIT on semaphore 1
Process A WAIT on semaphore 2

A "Deadly Embrace" situation will be the result.

Semaphores

64 numbered
64 Named

To avoid the "Deadly Embrace", it is vital that all processes that share data areas order their locks. The WAITs on the various semaphores must occur in the same order for each process.

Process A WAIT on semaphore 1
Process A WAIT on semaphore 2
Process A NOTIFY semaphore 1
Process A NOTIFY semaphore 2

Process B WAIT on semaphore 1
Process B WAIT on semaphore 2
Process B NOTIFY semaphore 1
Process B NOTIFY semaphore 2

Section 5 - Traps, Interrupts, Faults and Checks

There are 3 categories of software breaks in program execution:

1). INTERRUPTS
2). FAULTS
3). CHECKS

} Breaks in Execution

Ex. DMX

TRAP refers to a break in execution on the microcode level. TRAPS can occur for many reasons, some of which may directly or indirectly cause breaks in software execution. Not all software breaks are a result of a TRAP.

1). INTERRUPT (External Interrupt)

A signal has been received from a device in the external world (including clocks) indicating that the device either requires service or has completed an operation.

2). FAULT

A FAULT is a condition which has been detected as a result of the currently running software and which requires software intervention. A FAULT may be handled by the current software though most frequently common supervisor code will handle the FAULT (e.g. Page Fault).

3). CHECK

A CHECK is an internal CP consistency problem that requires software intervention. The problem may be an integrity violation, reference to a non-existent memory module or a power failure.

PIC - Phantom Interrupt Code (Seg 4 is interrupt segment)

EXTERNAL INTERRUPTS

When an EXTERNAL INTERRUPT is generated by a controller, the controller places a 16 bit interrupt vector address onto the bus. This address is used as an index into the interrupt segement (Seg 4) Segment 4 is "wired memory" and will, therefore, always be present in physical memory. The PB and Keys are saved in the microcode scratch registers PSWPB and PSWKEYS.

Further interrupts are then inhibited and the Interrupt Response Code (IRC) begins execution in 64V mode. It is the responsibility of the IRC to issue a CAI (Clear Active Interrupt) to the interrupting controller.

The IRC is Segment 4 does not belong to any specific process and has no PCB assigned to it. As it has no PCB, the IRC cannot save its registers and context. Clearly, there is little the IRC can do. It returns to PROCESS EXCHANGE as quickly as possible. The IRC is generally referred to as the PIC (Phantom Interrupt Code).

The PIC must perform one of two operations:

1). If the interrupt is very simple, the PIC will handle the interrupt
2). in the case of a more complex handling routine, PIC will reset the interrupt and NOTIFY the remainder of the PIC.

1). Simple Case

The IRTN (Interrupt Return) will be executed. This will restore the PB and KEYS and enter the dispatcher.

2). NOTIFY IRC Case

In order to NOTIFY a process, PIC must ensure that the PB and KEYS are restored before issuing the NOTIFY. The INOTIFY instruction will do both the restore and the Notify.

There are two ways by which the PIC can issue a CAI.

1). CAI instruction

2). Set bit 15 of the IRTN/INOTIFY instruction.

In practice, the PIC combines all of the above steps with a single instruction INEC.

CLOCK INTERRUPTS (on VCP)

Most current Prime systems use a device called the Programable Interval Clock (PIC). The PIC is a counter that is initialized or loaded by system software and once it is loaded it counts up at a rate of 3.2 us. until it overflows. The overflow is used to generate an interrupt via location '63 to wake up the clock interrupt handler (and hence the clock process). The counter is located on the controlller itself and can be counted independently of CPU operation.

The PIC counter is initialized at cold start to a -947.
    947 * 3.2 us. = 3.0303 ms.
After the PIC counts up 947 times at a 3.2 us. rate it overflows and generates an interrrupt via location '63 at a 3.0303 ms. rate. The PIC need only be preset once, thereafter it will reinitialize itself to a -947 after each time it overflows.

Earlier systems used a hardware controller called an Option A instead of a Diagnostic Processor (DP), System Option Controller (SOC), or Virtual Control Panel (VCP). The Option A board contains a Real Time Clock (RTC) which depends on the CPU to increment a memory location, which results in greater CPU overhead.

FAULTS

FAULTs are CPU events which are synchronous with and caused by software. Program caused itself to Stop

Two data areas are used:

1). PCB FAULT VECTORs and concealed stack pointers

2). the FAULT TABLEs pointed to by the PCB vectors.} Ring 0 Ring 3

Therefore each process can define its own fault handlers and the concealed stack allow FAULTS to be stacked. The PAGE FAULT has its own vector and only one system-wide handler is used so all PAGE FAULT vectors point to the same place.

Each FAULT TABLE entry consists of 4 words, of which the first 3 must be a CALF instruction. The CALF (CAL1 Fault) instruction is essentially a PCL (Procedure CaLl) instruction for the various Fault handling routines. The PB and KEYS from the concealed stack are placed in the Fault Handler's stack frame along with other base registers. The Fault Code and Fault Address are placed in words '12,'13, '14 of the Fault Handler's stack. The first word of the new stack frame is set to a value of 1. This is to distinguish the CALF stack frame from the normal PCL stack frame. The ECB (Entry Control Block) addressed by the CALF must not specify any arguments. Return from the fault handler is by normal PRTN instruction.

Arguments that get passed to Fault Handler

FAULT PROCESSING

TYPE	Octal # OFFSET	RING	Procedure Based Register SAVED PB	FCODE	FADDR
RESTRICTED INSTRUCTION	0	CURRENT	BACKED	--	--
PROCESS	4	0	CURRENT	ABORT	--
PAGE	10	0	BACKED	FLAGS --	ADDRESS
SVC For Calling Pieces operating sys.	14	CURRENT	CURRENT	--	--
UNIMPLEMENTED INSTRUCTION	20	CURRENT	BACKED	CURRENT P COUNTER	EFF ADDRESS
ILLEGAL INSTRUCTION	40	CURRENT	BACKED	CURRENT P COUNTER	EFF ADDRESS
ACCESS (To Rings) VIOLATION	44	0	BACKED	--	ADDRESS
ARITHMETIC EXCEPTION	50	CURRENT	CURRENT	EXCEPTION CODE	OPERAND ADDRESS
STACK OVERFLOW	54	0	BACKED	--	LAST STACK SEGMENT
SEGMENT- Caused By	60	0	BACKED	# too large or Fault Bit	ADDRESS
POINTER	64	CURRENT	BACKED	PTR 1st word	ADDRESS OF PTR

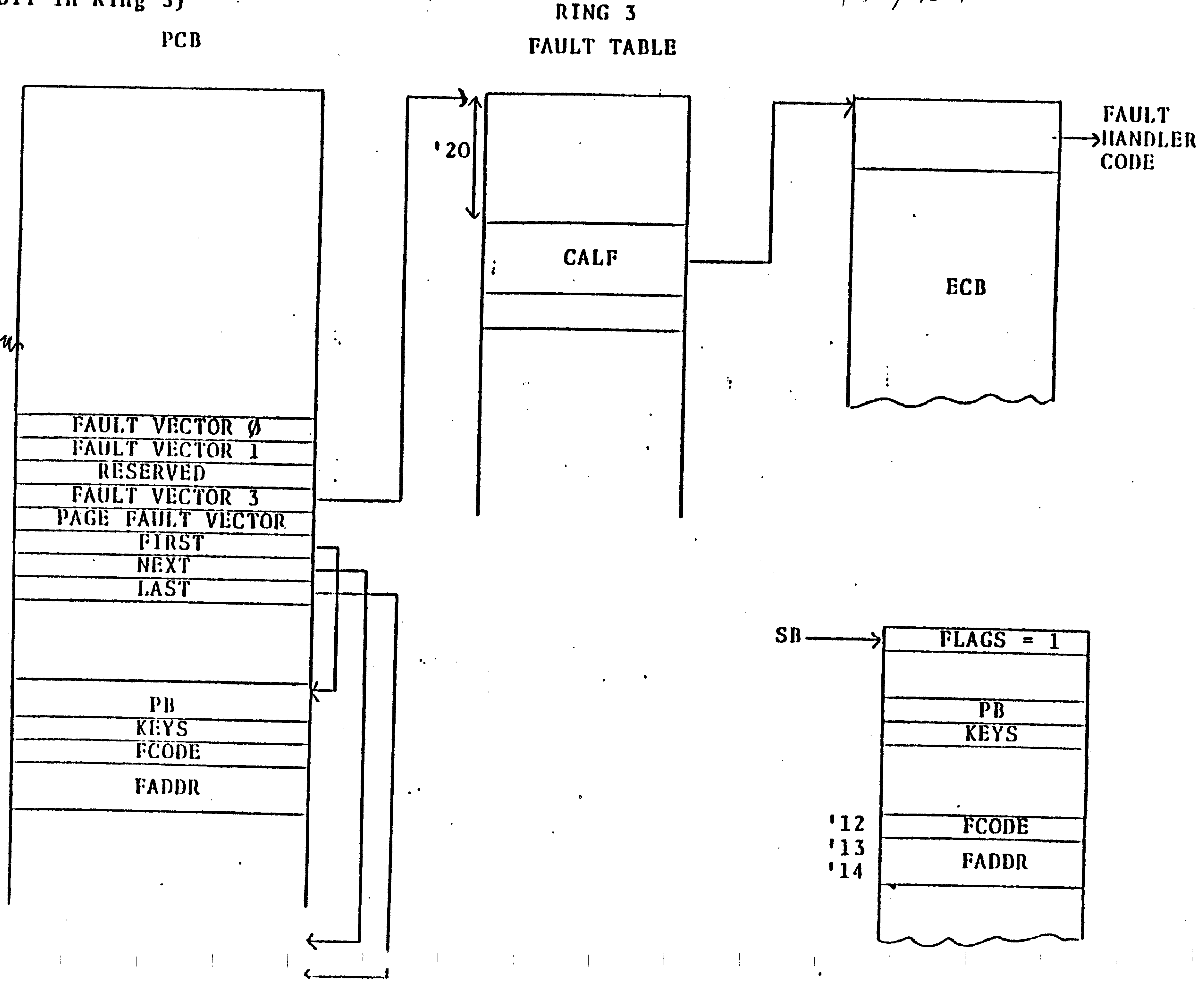
FAULT OPERATION

(eg UII in Ring 3)

to CAll system sub routine to resolve software fault

R3S>R3 FALT.PMA
KS>R0 FALT.PMA

the sub routine takes you to the Handler as if it were in the program



ACTION ON FAULT

1). Create an entry in the Concealed Stack (Firmware).

2). Transfer control to the Fault Table at the correct offset, in 64V Mode, with interrupts enabled.

3). Execute the Fault Handling routine as a part of the current process. The entry in the Fault Table will a CALF instruction. This creates a Stack Frame and transfers the Fault Code and Fault Address into this Stack Frame. The Fault Handling routine (software) is now called.

4). The Fault Handling routines executes a Procedure Return to exit the Fault processing and resume "normal" program execution.

REFALT

A page fault can not happen on top of An existing page fault - IT IS Delayed till 1st IS finished

1). Mechanism for deferring faults until the return from PGFSTK.

2). REFALT modifies the return (PB) in a stack frame and pushes a frame in the concealed stack so that a simulated fault may be taken when leaving PGFSTK.

CHECKS

A CHECK is a CPU event which is asynchronous with and not caused by normal instruction execution. CHECKs can most easily be classified as some sort of hardware physical failure.

There are four types of CHECKS:

CHECK	HEADER LOC	FIRST INSTRUCTION	DSW
		OF HANDLER	SET
Power Failure	4/'200	4/'204	No
Memory Parity	4/'270	4/'274 - Single Bit is corrected	Yes
Machine Check	4/'300	4/'304 - error on A Bus	Yes
Missing Memory	4/'310	4/'314	Yes

Each CHECK class has a single save area consisting of 8 words in the interrupt segment; in which the PB and KEYS are saved in the first 4 locations and the remaining 4 locations contain software codes.

Three 32 bit registers are used as a Diagnostic Status Word (DSW) to help a software Check Handler determine the cause of the CHECK. Check Handling software has the responsibility of clearing the DSW after every CHECK.

Event Logger - transfers registers to memory and files them

Section 6 - System Initialization

SYSTEM INITIALIZATION

PRIMOS is initiated from PRIMOS II by atteching to the UFD PRIRUN (Normally found on the command disk) and resuming PRIMOS. The routine PRMLD.FTN is then entered and the following actions are performed:

1). Attach to CMDNCO and open the file C_PRMO for command input.
PRIMOS.COMI (Rev20)
2). If the file is not found, output the message
'PLEASE ENTER CONFIG' and return to console input. (OLD STYLE)
3). Read in the first command from the file or read the
command from the console.
4). If the first command is not a CONFIG, output the message
'FIRST COMMAND MUST BE CONFIG' and return to the
message in 2).
5). Close the C_PRMO file and proceed with configuration.
configuration.

NEW STYLE CONFIGURATION

1). Open CONFIG data area
2). Read in commands and check legality.
3). When 'GO' command is inputted, close data file and proceed
as "OLD STYLE CONFIGURATION" from step 1).
4). If no 'GO' is inputted and the end of file is reached,
output the message 'MISSING GO'.

OLD STYLE CONFIGURATION

1). Check, configure, and start-up the main and alternative paging devices (if applicable).

2). If the device is illegal, output the message 'ILLEGAL PAGDEV'.

3). If the device contains normal file formats rather than paging formats, output the message 'USE DISK FOR PAGING'. A 'YES' or 'NO' answer must be given. THINK TWICE OR THRICE BEFORE ANSWERING 'YES'. BY ANSWERING 'YES' THE SURFACE IS MADE INTO A PAGING SURFACE AND ALL FILE DATA IS DESTROYED AND LOST.

4). Check, configure and start-up the command device.

5). If the device is illegal, output the message 'ILLEGAL COMDEV'.

6). Check the paging devices for split disk. If the name is 'PAGING', it can contain a 'BADSPT' file.

7). Read in the page maps from *COLDS.

8). If there is a BADSPT file, adjust the page maps accordingly.

9). Pre-page all PRXXXX files as necessary.

10). Resume *COLDS.

There are two possible entry points to the system:

1). COLD START - enter at SEG '14 '3000

2). WARM START - enter at SEG '14 '1000.

COLD START

PHASE 1

1). Enter 64V mode.

2). Set up CPU model number, u-code revision number, and write PRIMOS version into LOGBUF.

3). Set up controls for OPTION A or SOC is ASRDIM.

4). perform memory scan to size memory, check parity, and find bad pages.

5). Invalidate the STLB.

6). Clear the DSW.

7). Set up the interrupt processes PCBs.

8). Set up and start the clock.

9). Enter PROCESS EXCHANGE mode.

10). Set up Stack Base Register for USER 1.

11). Call AINIT.

AINIT

1). Turn off input from system console until I/O buffers are configured.

2). Set up system console baud rate if necessary.

3). Print the system ID and memory size.

4). Set up 'MAXSCH' based on available memory.

5). Check that 'CONFIG' information is available.

6). Check NUSR, PAGDEV, COMDEV, MAXPAG, ALTDEV, NAMLC, NPUSR, NRUSR, and SMLC.

7). Set up PAGREL for PAGDEV and ALTDEV (split disks only).

8). Unlock pages not needed for MMAP and adjust page maps.

9). Allow PAGE FAULTs.

10). Initialize USRCOMs.

11). Set login name for USER 1.

12). Attach to CMDNCO.

13). Establish terminal buffers for configured lines.

14). Call CINIT to process CONFIG commands.

15). Allow input from system console.

16). Initialize and wire PCBs for configured USERs 2 and up.

17). Calculate NSEG as follows:

A). Segments that will fit into specified paging space.

B). Specified NSEG command.

C). Default NSEG setting (Pre-Rev.18).

AINIT - continued

18). Initialize DTAR2 and DTAR3 for users.

19). Set page maps for RINGO Stacks.

20). Invalidate all except first two pages.

21). Set up templates for USER's PUDCOM and RING 0 Stacks.

22). Set up PUDCOM and USRCOM for configured users.

23). Lock network code if networks configured.

24). Lock SMLC driver if configured.

25). Initialize ECBs in Gate (Segment 5).

26). Initialize USER priority level.

27). Open C_PRMO if found, and skip the first executable statement.

28). Turn on AMLC and networks (if configured).

29). Calculate and print wired memory if WIRMEM directive is found.

30). Print message 'PLEASE ENTER DATE'.

31). Call FATAL$ to exit command for USER 1.

Once the date and time have been entered by the SE command, USERs may LOGIN. The form of the SE command is: SE -MMDDYY -HHMM.

32). Process other commands in C_PRMO

WARM START

1). Enter 64V mode.

2). Set up DTARs, Link Base, and enter Segemented Mode.

3). Initialize IOTLB.

4). Save registers on interrupted USER.
NOTE: WARM START cannot be done if no registers have been saved. If this is the case, HALT.

5). LOG if power fail.

6). Move registers from save area to PCBs.

7). Correct PB/KEYS for process that was running. This is necessary if the HALT was in Phantom Interrupt Code or after a Machine Check.

8). Reset PCBs for device driver processes.

9). Initialize various flags and control registers for device controllers and device drivers.

10). Reset USER 1 Stack; reset Clock; and enter PROCESS EXCHANGE mode.

11). Handle UPS (Uniterruptable Power Supply) if present.

12). Log WARM START in LOGBUF.

13). Reset critical state variables and semaphores.

14). NOTIFY DSKSEM if user waiting.

15). Set WARMALM for USER 1. Other USERs should continue normally.

16). Exit into clock process.

Section 7 - Condition Mechanism

Faults signal conditions

CONDITION MECHANISM

conditions work off the stack

MOTIVATION

- system software error handling
- manage reentrant/recursive command environment
- user program error (and event) handling
- support ANSI PL/1 condition mechanism

IMPLEMENTATION

- extended stack header
- on-unit descriptor block (on stack)
- condition frame header (on stack)
- fault frame header (on stack)

CONDITION MECHANISM-definitions

CONDITION - an unscheduled event (Asyacronaus)

ON-UNIT - a procedure to handle an event

SIGNAL - telling the world the event happened

RAISE - procedure which searches the stack for the ON-UNIT

CRAWL_ - procedure which switches from inner ring to ring 3 stack
(out of Ring Ø into Ring 3)

MAKE ON-UNIT - turn on event handler for this activation

REVERT ON-UNIT - turn off event handler for this activation

NON-LOCAL-GOTO - a goto to a predefined label not in this activation
(-GOTO transfers out to previous

DEFAULT ON-UNIT - one example of system use of condition mech.

```
ok.e, seg sleep
This is SLEEP.FTN, going to sleep for one minute      /* normal
This is SLEEP.FTN, finished sleeping, exiting         /* execution

ok.e, seg sleep
This is SLEEP.FTN, going to sleep for one minute      /* control P
                                                      /* typed
QUIT.
ok.e, dmstk -all -on_units
Backward trace of stack from frame 1 at 6002(3)/7642.

STACK SEGMENT IS 6002.

(1) 007642: Owner=  (LB= 13(0)/13062).
    Called from 13(3)/101525; returns to 13(3)/101531.

(2) 006564: Owner=  (LB= 13(0)/103240).
    Called from 13(3)/100723; returns to 13(3)/100727.

(3) 004330: Owner=  (LB= 13(0)/103240).
    Called from 13(3)/10234; returns to 13(3)/10254.
```

```
(4) 003576: Owner=  (LB= 13(0)/13062).                      /* STD$CP
    Called from 13(3)/2717; returns to 13(3)/2731.
    Onunit for "CLEANUP$" is 13(3)/14063.
    Onunit for "STOP$" is 13(3)/13663.
    Onunit for "SUBSYS_ERR$" is 13(3)/13703.

(5) 003260: Owner=  (LB= 13(0)/3700).                       /* LISTEN_
    Called from 13(3)/75556; returns to 13(3)/75562.
    Onunit for "CLEANUP$" is 13(3)/4432.
    Onunit for "ANY$" is 13(3)/70446.
    Onunit for "LISTENER_ORDER$" is 13(3)/4472.
    Onunit for "SETRC$" is 13(3)/4452.
    Onunit for "REENTER$" is 13(3)/4512.

(6) 003234: Owner=  (LB= 13(0)/75172).                      /* COMLV$
    Called from 13(3)/55364; returns to 13(3)/55366.

(7) 002544: Owner=  (LB= 13(0)/57774).                      /* DF_UNIT_
    Called from 13(3)/45217; returns to 13(3)/45223.

(8) 002444: Owner=  (LB= 13(0)/44734).                      /* RAISE
    Called from 13(3)/44267; returns to 13(3)/44301.
```

```
(9) 002316: CONDITION FRAME for "QUIT$"; returns to 13(3)/51247.
    Condition raised at 6(0)/3435; LB= 6(0)/3314, Keys= 014000
    (Crawlout to 4001(3)/1043; LB= 4002(0)/177400. )
    Inner ring fault: type "PROCESS" (4); code= 000200; addr= 0(0)/0
    Registers at time of fault in inner ring:
                    Save Mask= 000000;  XB= 6(0)/1372
        GR0      0      0           0   GR1      0      0           0
      L,GR2      0      0           0 E,GR3      0      0           0
        GR4      0      0           0 Y,GR5      0      0           0
        GR6      0      0           0 X,GR7      0      0           0
     FAR0 0(0)/0             FLR0           0 FR0    0.00000000E 00
     FAR1 0(0)/0             FLR1           0 FR1    0.00000000E 00

(10) 002114: Owner=  (LB= 13(0)/50660).               /* CRFIM_
    Called from 4001(3)/1043; returns to 4001(3)/1043.

STACK SEGMENT IS 4001.                 /* control P typed here

(11) 001174: Owner=  (LB= 4002(0)/177400).            /* SLEEP.FTN
    Called from 4000(3)/56547; returns to 4000(3)/56551.
```

```
STACK SEGMENT IS 4000.

(12) 150062: Owner=  (LB= 4000(0)/56234).               /* SEG (VRUNIT)
     Called from 4000(3)/1723; returns to 4000(3)/1725.
     Proceed to this activation is prohibited.

(13) 150012: Owner=  (LB= 4000(0)/5130).                /* SEG (MAIN)
     Called from 4000(3)/1100; returns to 4000(3)/1102.
     Onunit for "CLEANUP$" is 4000(3)/57340.

(14) 150000: Owner=  (LB= 4002(0)/177400).              /* invalid frame
     Called from 0(0)/177776; returns to 0(0)/0.        /* set up by SEG
```

```
STACK SEGMENT IS 6002.

(15) 001652: Owner=  (LB= 13(3)/31260).                        /* INVKSM_
     Called from 13(3)/12610; returns to 13(3)/12632.
     Onunit for "CLEANUP$" is 13(3)/31745.
     Onunit for "ANY$" is 13(3)/31725.
```

Runs program SEG

```
(16) 001472: Owner=  (LB= 13(0)/13062).
     Called from 13(3)/11632; returns to 13(3)/11636.

(17) 000750: Owner=  (LB= 13(0)/13062).                        /* STD$CP
     Called from 13(3)/2717; returns to 13(3)/2731.
     Onunit for "CLEANUP$" is 13(3)/14063.
     Onunit for "STOP$" is 13(3)/13663.
     Onunit for "SUBSYS_ERR$" is 13(3)/13703.
```

command line

```
(18) 000432: Owner=  (LB= 13(0)/3700).                         /* LISTEN_
     Called from 13(3)/142374; returns to 13(3)/142400.
     Onunit for "CLEANUP$" is 13(3)/4432.
     Onunit for "ANY$" is 13(3)/70446.
     Onunit for "LISTENER_ORDER$" is 13(3)/4472.
     Onunit for "SETRC$" is 13(3)/4452.
     Onunit for "REENTER$" is 13(3)/4512.
```

(waits for you to type command)

```
(19) 000424: Owner=  (LB= 13(0)/142014).                       /* INFIM_
     Called from 0(0)/142376; returns to 0(0)/0.
```

The condition mechanism is activated whenever a condition is raised by the PL/1 <SIGNAL STATEMENT> or by a call to SIGNL$ or SGNL$F. It scans the stack backwards in sequence until an activation is found with an on-unit the condition or for ANY$ is found.

POSSIBLE ACTIONS OF AN ON-UNIT

1). Perform application specific tasks (e.g. closing files, updating files).

2). Repair cause of condition and resume execution.

3). Decide that the normal flow can be interrupted and the program re-entered at a known point by performing a non-local GOTO to some previously defined label.

4). Signal another condition.

5). Transfer user to command level.

6). Continue the search for more on-units.

7). Run diagnostic routines.

CONDITIONS

1). A name (Up to 32 characters).

2). Machine state at the time the condition occured.

3). Auxiliary information (e.g. file control block of PL/1 I/O condition).

4). Continue switch (continue to signal)

5). Return switch (on-unit may return)

6). Inaction switch (on-unit may return without taking any action)

ON-UNIT

1). Name of condition to be handled.

2). A pointer to the procedure to handle the condition.

3). Reverted switch (the on-unit is no longer active if set)

4). Specifier (set if more than the condition name is required to completely describe the condition)

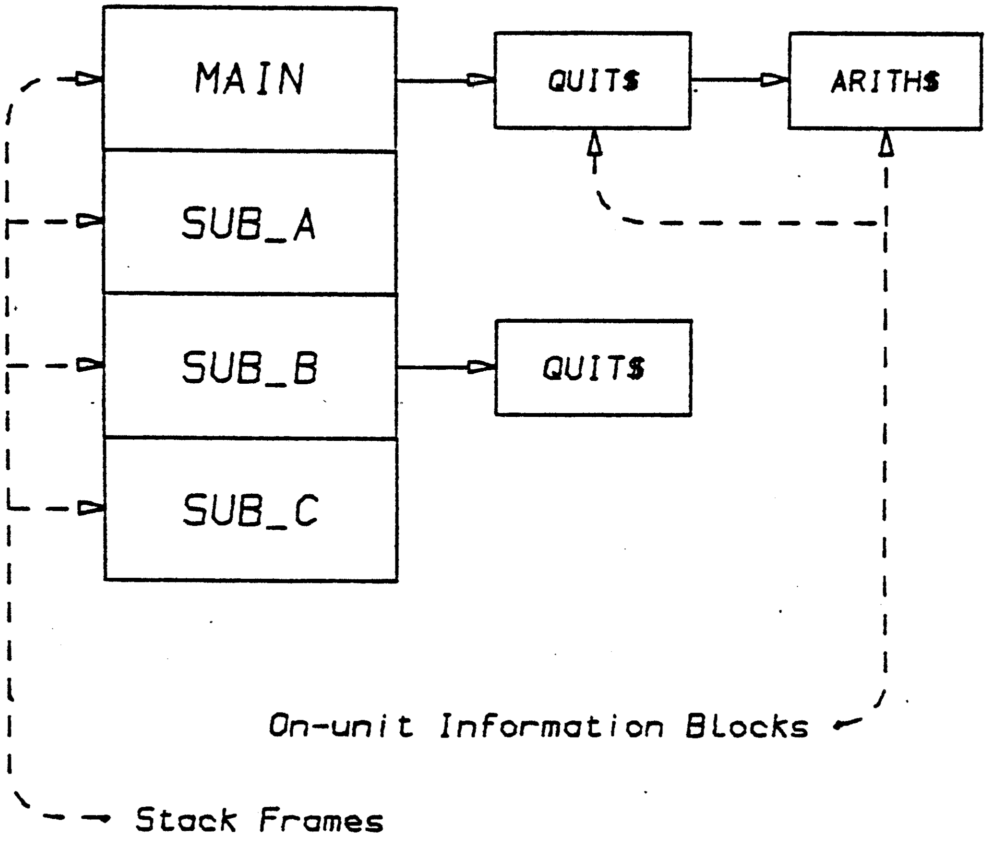
5). Specifier pointer (to file descripter if required)

CLEANUP.FTN

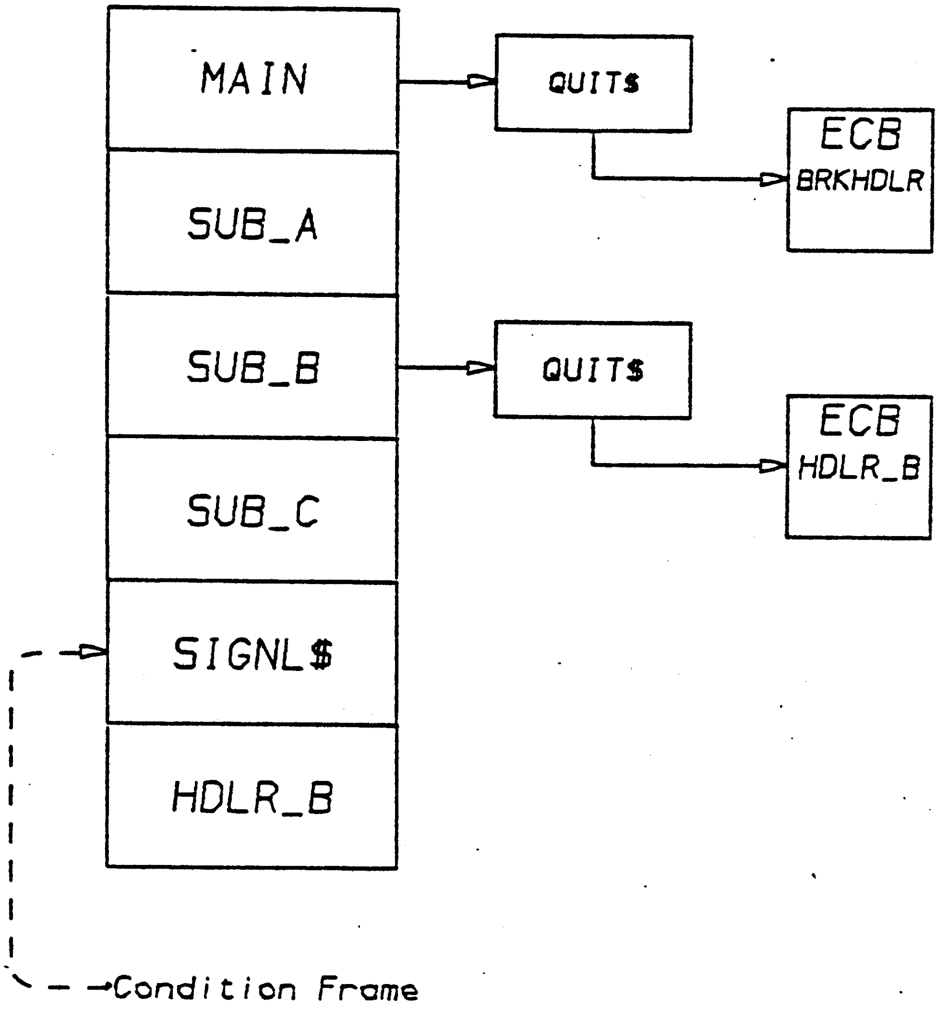
```
      EXTERNAL BKHDLR
      INTEGER DUMMY
      REAL*8 BRKRTN
      COMMON /BRKLBL/ BRKRTN
      LOGICAL*2 MAINBK
      COMMON /BRKCOM/ MAINBK
      MAINBK = .FALSE.                    /* BKHDLR NOT YET ENTERED
      CALL MKON$F ('QUIT$', 5, BKHDLR)       /* MAKE ON-UNIT FOR MAIN
      CALL MKLB$F ($1000, BRKRTN)            /* LABEL FOR NON-LOCAL GOTO
      PRINT 10
10    FORMAT ('Entering MAIN after invocation from SEG')
      PRINT 20
20    FORMAT ('Type <RETURN> to call SUBA, <BREAK> to test on-unit')
      READ (1,25) DUMMY
25    FORMAT (A2)
      IF (MAINBK) GOTO 100
      CALL SUBA
      PRINT 30
30    FORMAT ('Returned to MAIN normally from SUBA')
      CALL EXIT
100   PRINT 110
110   FORMAT ('Returned to MAIN from BKHDLR')
      CALL EXIT
1000  PRINT 1010
1010  FORMAT ('Returned to MAIN via NON-LOCAL go to')
      CALL EXIT
      END
      SUBROUTINE SUBA
      PRINT 10
10    FORMAT ('Entering SUBA called by MAIN, call SUBB')
      CALL SUBB
      PRINT 20
20    FORMAT ('Returned to SUBA normally from SUBB'
      RETURN
      END
      SUBROUTINE SUBB
      EXTERNAL HDLRB
      CALL MKON$F ('QUIT$', 5, HDLRB)
      PRINT 10
10    FORMAT ('Entering SUBB called by SUBA, call SUBC')
      CALL SUBC
      PRINT 20
20    FORMAT ('Returned to SUBB normally from SUBC')
      RETURN
      END
      SUBROUTINE SUBC
      INTEGER DUMMY
      EXTERNAL CLHDLR
      CALL MKON$F ('CLEANUP$', 8, CLHDLR)
      PRINT 10
10    FORMAT ('Entering SUBC called by SUBB')
      PRINT 20
20    FORMAT ('Type <RETURN> to EXIT, <BREAK> to test on-unit')
      READ (1,25) DUMMY
25    FORMAT (A2)
      PRINT 30
30    FORMAT ('SUBC exiting normally')
      RETURN
```

CONDITION MECHANISM--CLEANUP.FTN.

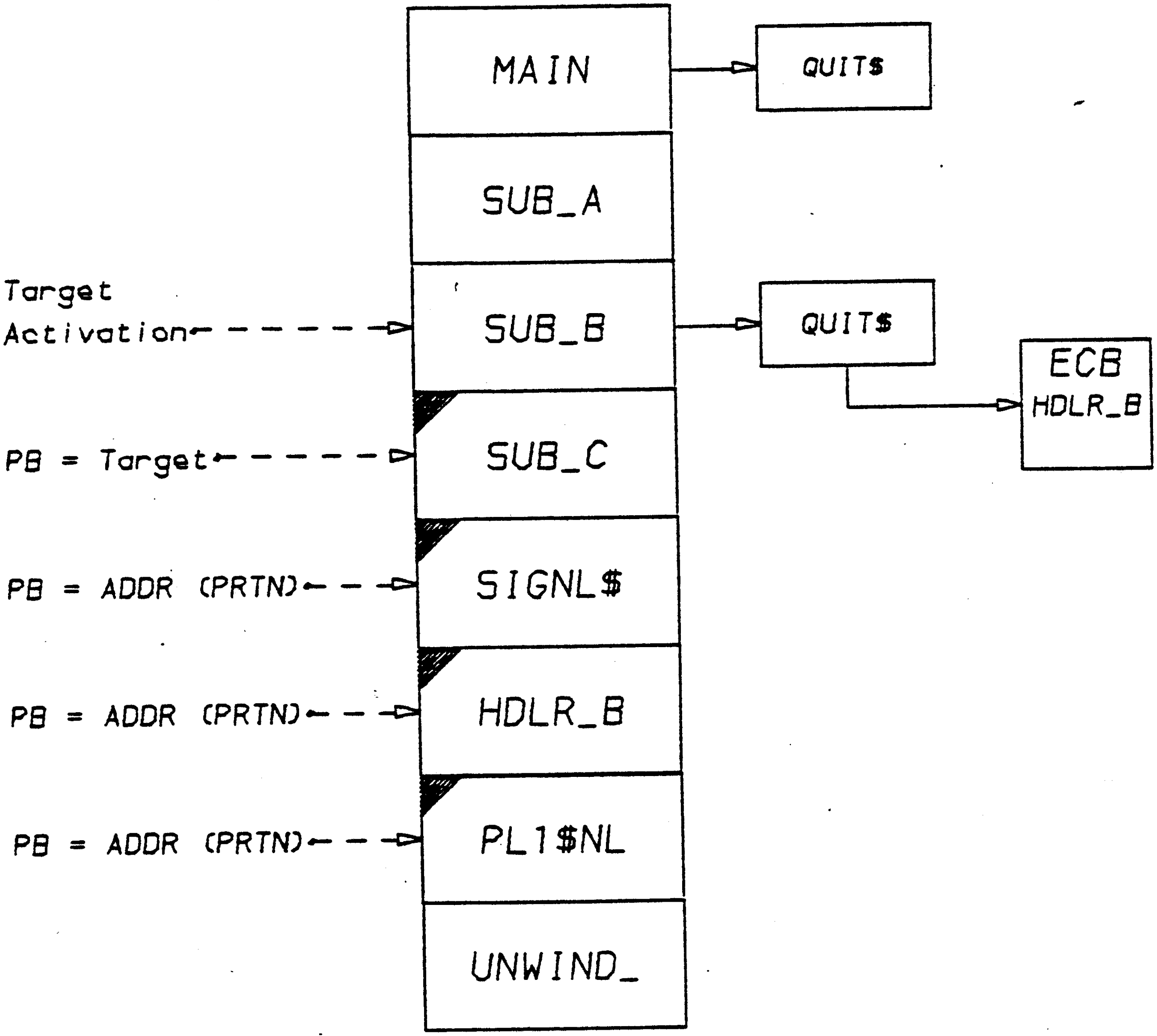
```
      SUBROUTINE BKHDLR (PNTR)
      INTEGER*4 PNTR
      LOGICAL*2 MAINBK
      COMMON /BRKCOM/ MAINBK
      CALL TNOU('BKHDLR called by condition QUIT$, return',40)
      PAUSE 1                          /* needed since I/O on return
      MAINBK = .TRUE.                  /* BKHDLR now entered
      RETURN
      END
      SUBROUTINE HDLRB (PNTR)
      INTEGER*4 PNTR
      REAL*8 BRKRTN
      COMMON /BRKLBL/ BRKRTN
      PRINT 10
10    FORMAT ('Entering HDLRB called by condition QUIT$, call PL1$NL')
      CALL PL1$NL (BRKRTN)
      RETURN
      END
      SUBROUTINE CLHDLR (PNTR)
      INTEGER*4 PNTR
      PRINT 10
10    FORMAT ('Entering CLHDLR called by condition CLEANUP$, return')
      RETURN
      END
```



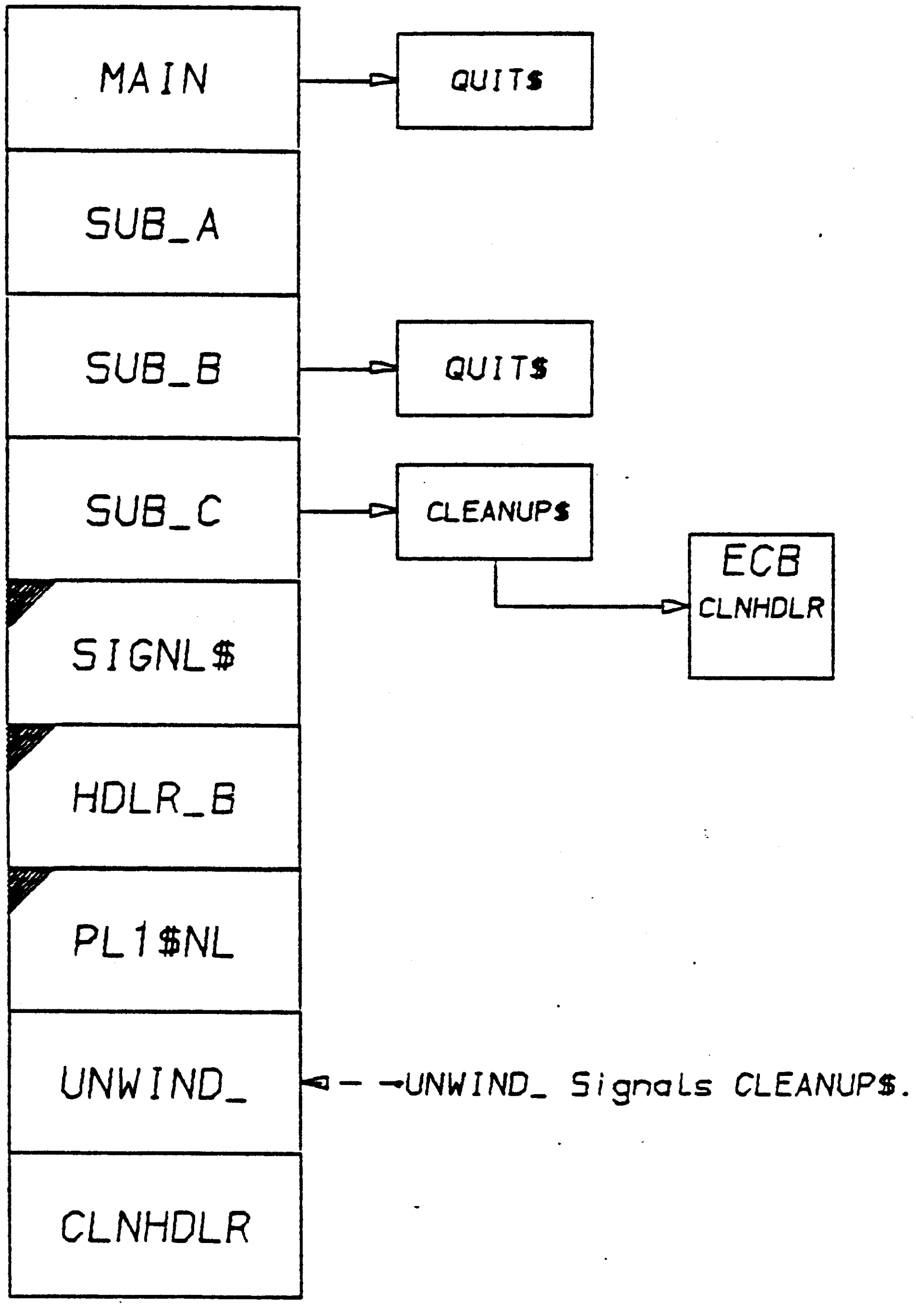
MAKING ON-UNITS



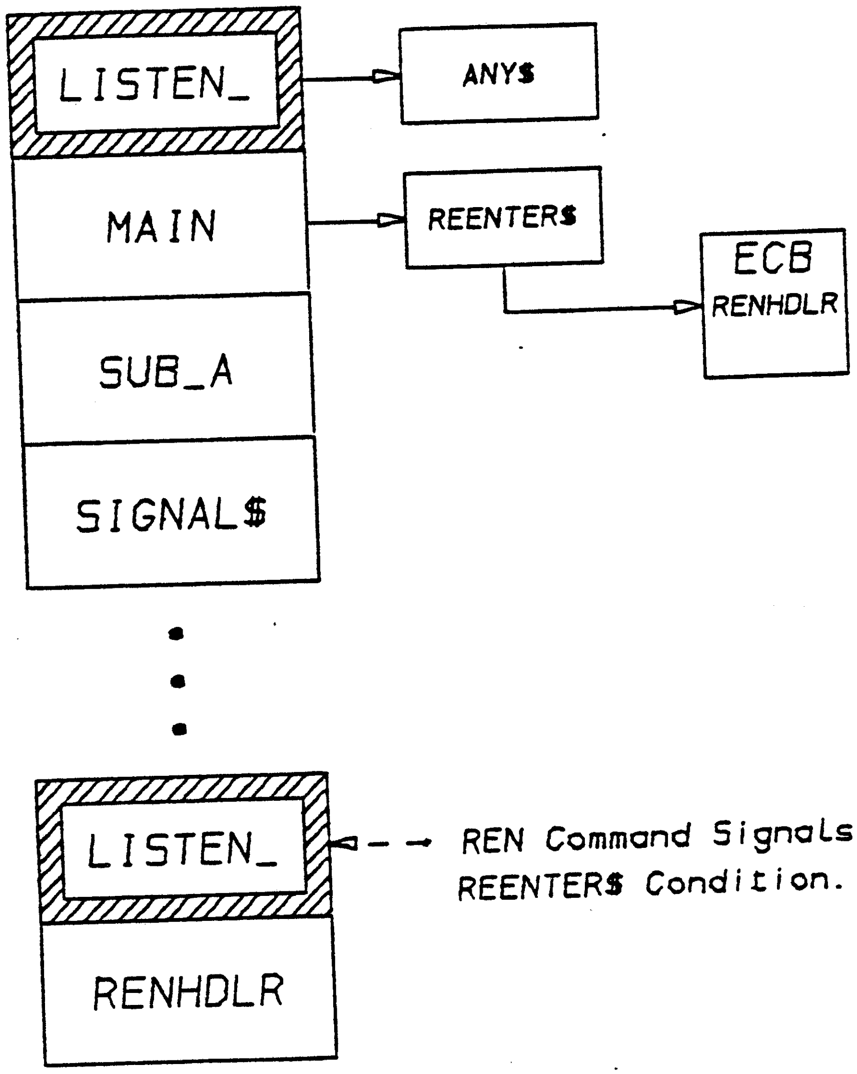
SIGNALING A CONDITION



NONLOCAL GOTO


MAIN
QUIT$
SUB_A
SUB_B
QUIT$
SUB_C
CLEANUP$
ECB
CLNHDLR
SIGNL$
HDLR_B
PL1$NL
UNWIND_
UNWIND_ Signals CLEANUP$.
CLNHDLR

CLEANUP



SUBSYSTEM REENTRY

CRAWLOUT

Crawlout occurs when the end of an inner ring stack has been reached by the condition mechanism without handling the condition.

Control always orginates in an outer ring, the end of an inner ring stack is threaded to an outer ring stack. The condition mechanism continues the stack search across the connection and back down the outer ring stack. Crawlout is the mechanism which copies the information describing the condition to the outer ring and resignals.

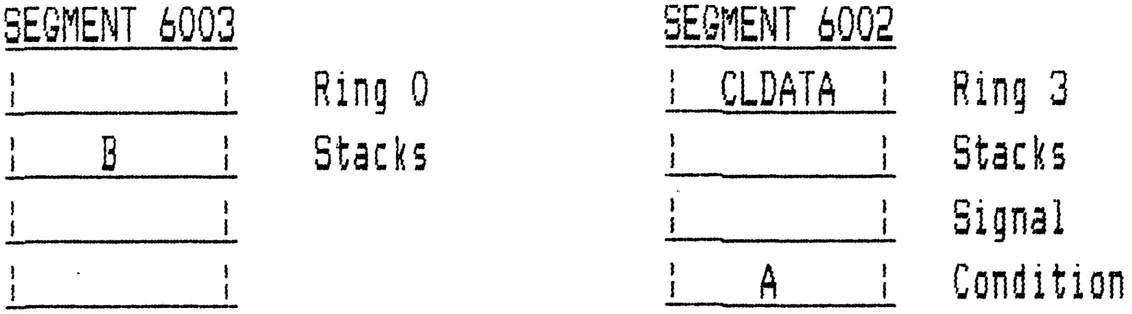
When RAISE reaches the end of the inner ring stack, it returns to SIGNL$ with the CRAWLOUT_NEEDED flag set, a pointer to the last stack frame on the inner ring (CRAWL_FRAME) and a pointer to the most recent inner ring stack frame in which the registers are saved.

SIGNL$ calls CRAWL_ defining the crawlout fault interceptor module (CRFIM_). The stack frame on the outer ring is the target frame.

CRAWL_ checks the space needed in the outer ring stack for the target ring stack and copies the neccessary information into the target stack. The return information in CRAWL_FRAME is adjusted to appear as though it was called from the target frame.

UNWIND is called to unwind the stacks and RO locks are released. A procedure return is then invoked to CRFIM_.

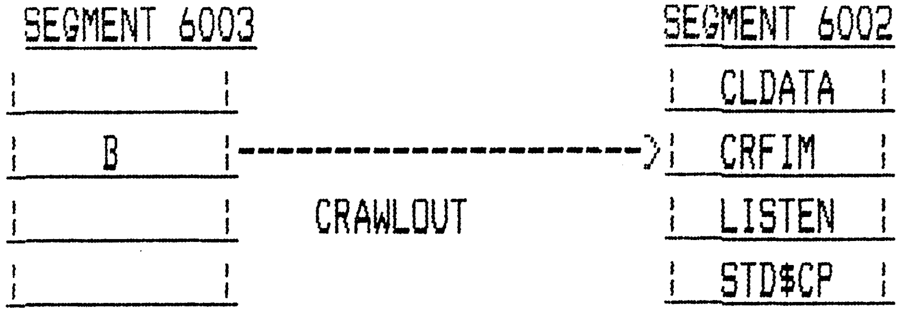
CRFIM_ calls SIGNL$ to signal the condition in the outer ring and the on-unit will invoke the first LISTEN_ level.



Procedure B signals a condition. The stacks are searched but a suitable on-unit cannot be found.

B is the last inner ring stack.

(CRAWL_FRAME)



Section 8 - Fault Handling

FAULTS are handled in two ways:

1). Those handled in RING 0 and

2). Those handled in the current RING (RING 3).

1). RING 0 FAULTS

The Fault Vector in the user's PCB for RING 0 points to a fault table called FAULT in Segment 6. The fault table is defined in PRIMOS>KS>PABORT.FTN The Fault Handlers are found in PRIMOS>KS>R0FALT.PMA

The following Fault Handlers exist in Segmment 6:

PROCESS FAULT
PAGE FAULT
UII (UnImplemented Instruction)
ACCESS VIOLATION
STACK OVERFLOW
SEGMENT FAULT
POINTER FAULT

Any other Fault occurring in RING 0 (e.g. SVC, restricted instruction) will cause the system to HALT.

PROCESS FAULT

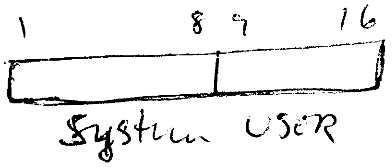
1. Check Abort Flags
2. If any Abort Flag is set and aborts are enabled, call PABORT.

SYSTEM ABORT FLAGS

– (Primos does the processes it needs to Do As user 1)

PABORT bit number

1	MINALM	One minute update
2	SMLALM	SMLC alarm
3	NETALM	Network Alarm
4	LGIALM	LOGIN Alarm
5	WRMALM	Warm Start
6	MSGALM	SUSR Message Alarm
7,8	- - -	Not Used



USER 1

1 ONE MINUTE (MINABT)

Dump any entries in LOGBUF to LOGREC

Update all disk buffers

Decrement auto-logout clocks and logout any USERs out of time.

2 SMLC (SMLCEX) Process SMLC requests

3 NETWORK Process network requests (done by NETUSR at Revision 19)

4 LOGIN ALARM (WIRSTK) Lock USER stack, notify user (LOGLCK)

5 WARM START (WRMABT)

Initialize MPC, VERSATEC, and Magnetic Tape

Initialize network and AMLCs, Output message 'WARM START'

6 SUPERVISOR MESSAGE ALARM (T1OU) Process USER 1 message buffer.

USER ABORT FLAGS

PABORT bit number

16	TSEALM	Time Slice End (set by microcode)
14	TMOALM	Time-out LOGOUT
13	DISALM	AMLC disconnect LOGOUT or Operator LOGOUT
10	IOALM	I/O done (Magtape, MEGATEK)
9	SWIALM	SoftWare Interrupt Alarm (formerly QUTALM)
15,12,11	- - -	Not Used

FOR EACH USER

16 TIME SLICE END (SCHED)

Place process on low priority or eligibility queue

14,13 FORCED LOGOUT (LOGABT)

Output message 'TIMEOUT', or 'FORCE LOGOUT', Signal 'LOGOUT$'

10 I/O ALARM Call MTDONE

9 SoftWare Interrupt (SW$ABT)

SOFTWARE INTERRUPT HANDLING

MOTIVATION

- Due to increased frequency of asynch events at rev 19; more pressure on quit mechanism.

- Ring 0 code had to explicitly inhibit process aborts. Unexpected exit from many ring 0 routines before completion produces non-reliable results.

- Inhibiting quits would disable multiple process abort events.

IMPLEMENTATION

- BREAK$ code reduced to only handle QUIT$.

- SoftWare Interrupt modules for rest of process aborts.

- SWITYP flag word defines which event.

- New mechanism defaults to inhibiting process aborts in ring 0. Enabling quits in ring 0 must now be explicitly performed.

SOFTWARE INTERRUPT HANDLING - Routines and Variables

BREAK$ - enable/disable QUIT$ aborts in ring 0

SW$INT - process abort interrupt enable/disable control

SETSWI - store event bit in PUDCOM.SWITYP } Used both for Abort

SETABT - set user's abort flags

SW$ABT - fault handler for process aborts

(Fault interface mgmnt)

SWFIM_ - handles deferred ring 0 aborts on return to outer ring

SW$RST - called by SWFIM_ to reset ROSWIN, ROQUIT

Variables SWITYP

Variables	SWITYP	1 = quit
		2 = logout notification (LON)
		4 = real time watchdog
		'10 = cpu time watchdog
		'20 = Cross Process Signalling (CPS)
		'40 = forced logout

ROSWIN - ring 0 software interrupt enable counter
ROQUIT - ring 0 quit enable counter

SOFTWARE INTERRUPT HANDLING

When process abort happens while inhibited in ring 0, SW$ABT detects need to defer process and does following:

1. Turn current frame into pseudo condition frame as indicated by SWITYP.

2. Check concealed stack to see if outstanding faults.

3. Call CRAWL_ to build SWFIM_ frame on outer ring stack; but do not execute crawlout.

4. Set ROSWIN (or ROQUIT) to -1 (process abort deferred).

5. Mark SWFIM_ frame if concealed stack frames outstanding.

When execution returns from ring 0, SWFIM_ is entered.

1. Cleanup concealed stack if needed.

2. Invoke SW$RST to reset ROSWIN and ROQUIT; if SWITYP non-zero call SETABT (multiple events)

3. Signal condition.

Ring & Faults

UII FAULT

XVRY, ZMV, ZMVD, ZFIL, and ZCM are simulated in a routine called ROUII in segment 6. (only if operating on a P400/350) All other UII faults in ring 0 HALT the machine.

ACCESS VIOLATION

SIGNAL$ called to output the message "ACCESS VIOLATION RAISED AT ...."

STACK OVERFLOW

Call STKOVF, SIGNAL$ 'STACK_OVF$', message 'STACK-OVF$ RAISED AT ...."

SEGMENT FAULT

GETSEG called to either allocate a segment or SIGNAL$ called to output the message "ILLEGAL SEGNO$ RAISED AT ...."

POINTER FAULT - Ring 0

1). Save user state
2). Pick up faulting pointer
3). Return if pointer is greater or equal 0
4). Erase fault bit
5). Error message if pointer is equal 0, or invalid
6). Call SNAP$3 to get new pointer
7). Snap link
8). If not found error message

POINTER FAULT outputs the message "POINTER-FAULT$ RAISED AT ...."

PAGE FAULT

Whenever a user program issues a virtual address the hardware translates this address into physical memory using the STLB. An STLB 'miss' may be caused by failure to find the desired entry, or by a reset valid bit for the desired entry. During full translation, the HMAP entry will indicate if the desired page is not in memory.

The page map entry contains a marker bit (bit 1) indicating whether or not the required page is held in memory. If the page is in physical memory, translation proceeds but if the page is not in memory, a PAGE FAULT occurs.

This fault causes a branch in execution through the user's page fault vector to the fault table code. A CALF is then executed in the page fault catcher. (All page faults are handled by this routine).

The page fault catcher will:

1). Save the user state (Rends PB, Keys)
2). Check recursive page fault. If so HALT
    Allow warm start but process takes fatal error.
3). Call PAGTUR
4). Increment page fault counter

See Handout

PAGTUR

The routine PAGTUR handles the page management in PRIMOS. Page-in is on demand, page-out is based on an approximate least-recently-used algorithm with pre-paging.

PAGTUR uses the page-maps as follows:

1). HMAP segment 22

1	2	3	4	5 … 16
V	R	U	S	PRN

(V) Valid Bit. Page in memory (1 = yes)
(R) Referenced bit
(U) Unmodified bit
(S) Inhibit CACHE for this page
5-16 Physical page number

if the page is not in memory bits 3,5 define

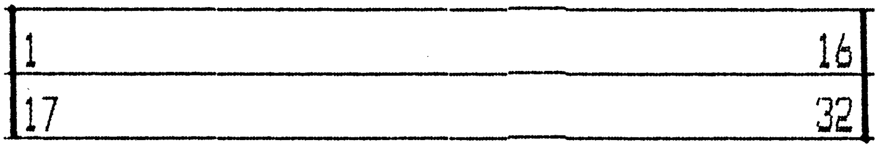
00 not in, copy on disk
10 not in, no copy on disk
01 in transition, coming in
11 in transition, going out

2). <u>LMAP</u> segment 33

1	2	3	4	5	16
Lock		F	A	RECORD INDEX	

BITS	
1,2	lock number (0 = unlocked)
3	First time bit (to keep page in memory longer)
4	Use alternative paging disk
5-16	Record index (Address of a track containing 8 pages)

3). MMAP (segment 14)

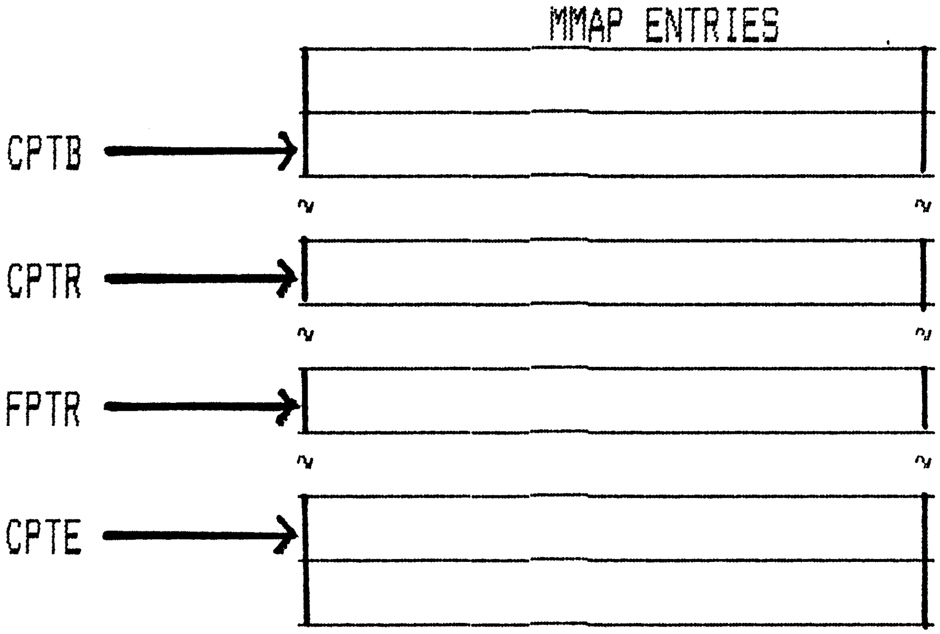


If entry LT 0 page does not exist (missing memory)

If entry EQ 0 page is available

If entry GT 0 page is in use (indicates the owner of the page)

PAGTUR uses four pointers to MMAP



CPTR is stepped during page-out

FPTR is stepped during page-in

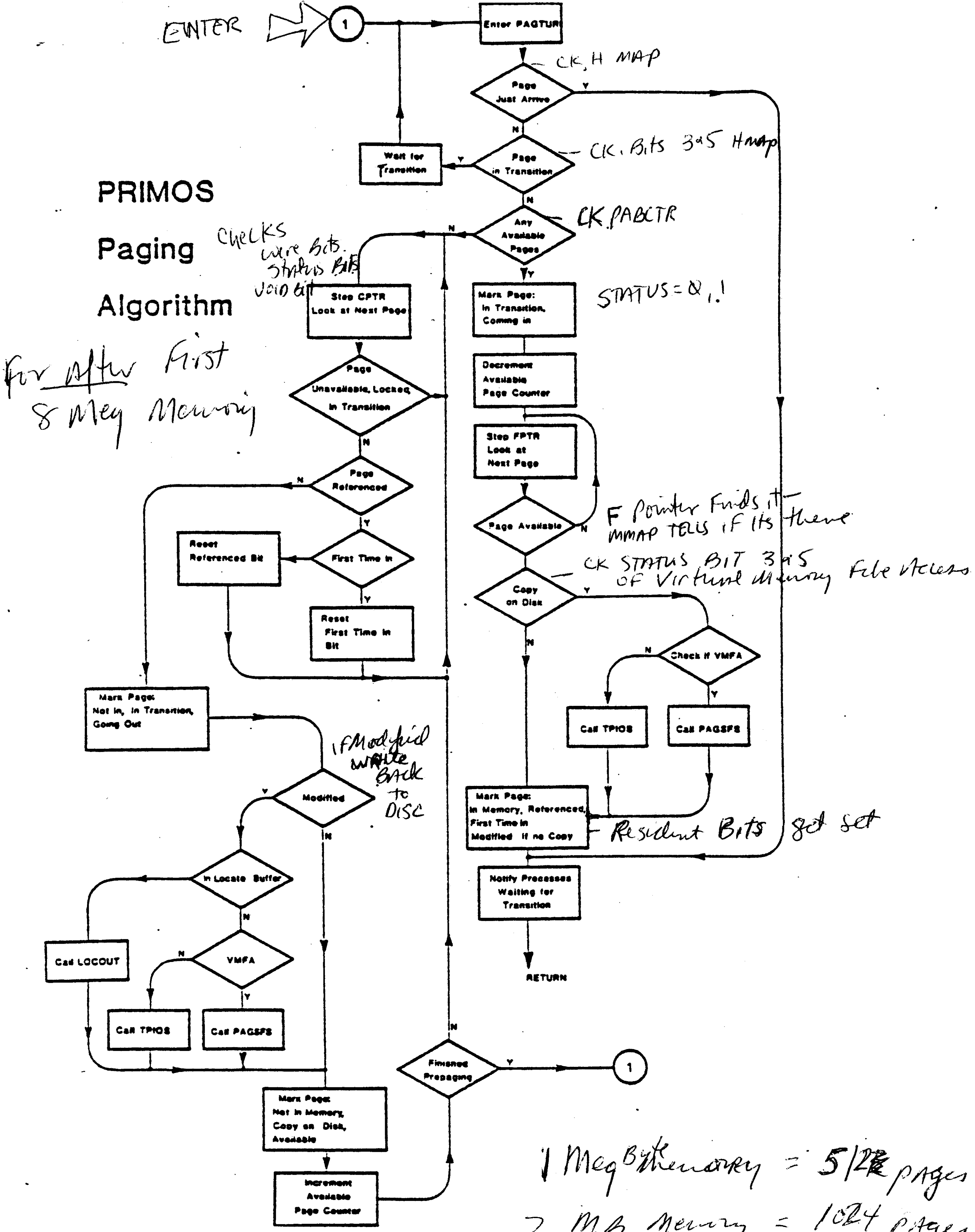
CPTB pointer to first pageable page

CPTE pointer to last pageable page

PAGTURN

PRIMOS Paging Algorithm

For After First 8 Meg Memory


ENTER
1
Enter PAGTURN
CK. H MAP
Page Just Arrive
Y
N
Page in Transition
CK. Bits 3&5 Hmap
Y
Wait for Transition
N
Any Available Pages
CK PABCTR
N
Y
Checks were Bits. Status Bits Void Bit
Step CPTR Look at Next Page
Mark Page: In Transition, Coming in
STATUS = 0, 1
Page Unavailable, Locked, In Transition
Decrement Available Page Counter
N
Page Referenced
N
Step FPTR Look at Next Page
Y
Reset Referenced Bit
First Time In
Page Available
N
F Pointer Finds it - MMAP TELLS if its there
Y
Reset First Time In Bit
Copy on Disk
CK STATUS BIT 3&5 OF Virtual Memory File Access
Y
N
Check if VMFA
N
Y
Mark Page: Not In, In Transition, Going Out
Call TPIOS
Call PAGSFS
If Modified write Back to DISC
Modified
Y
N
Mark Page: In Memory, Referenced, First Time In Modified If no Copy
Resident Bits get set
In Locate Buffer
Notify Processes Waiting for Transition
N
Call LOCOUT
VMFA
N
Y
RETURN
Call TPIOS
Call PAGSFS
N
Finished Prepaging
Y
1
Mark Page: Not In Memory, Copy on Disk, Available
Increment Available Page Counter

1 Meg Byte memory = 512 pages

2 MB memory = 1024 pages

RING 3 FAULTS

The fault vector in the user's PCB for ring 3 points to a fault table called R3FALT in segment 13.

The following fault handlers exist in segment 13:

- RESTRICTED INSTRUCTION FAULT
- SVC FAULT
- UII FAULT
- ILLEGAL INSTRUCTION FAULT
- ARITHMETIC FAULT
- STACK OVERFLOW FAULT
- POINTER FAULT

Any other fault occuring in ring 3 is handled by the ring 0 fault handlers.

RESTRICTED INSTRUCTION FAULT

Call PTRAP in ring 0

1). Read violating instruction and analyze.

2). If illegal or HALT instruction call SIGNAL$ to output the message 'PROGRAM HALT AT .....'

3). Simulate trapped I/O instructions for
   - System console, CRTs
   - Paper tape reader/punch
   - Card reader
   - Control panel

SVC

Enter SVC fault handler to initiate SVC and pass arguments.

UII FAULT (Same as Ring 0)

Enter UII routine in segment 13 to software emulate the instruction.

ILLEGAL INSTRUCTION FAULT

Enter illegal instruction fault handler which signals 'ILLEGAL-INST$'.

ARITHMETIC FAULT (gets error msg printed out)

Enter arithmetic fault handler which signals ARITH$ condition.

STACK OVERFLOW FAULT

Call STKOVF. (Automatic Ring 3 Stack Extension)

Examine stack frame prior to fault frame and determine stack root segment.

If root is '6002 then STK_EX is called.

Otherwise condition 'STACK_OVF$' is signalled as before.

STK_EX

Attempts to get a DTAR 3 dynamic segment.

If not possible calls FATAL$.

Otherwise fixes up stack extension ptr to point to new segment, and returns.

Address

FAULT Bit		SEG #	WORD #

POINTER FAULT

SEE pg 8-8

1). Save user state

2). Clear fault bit

3). If bad pointer - signal POINTER-FAULT$ (Must Be in Ring 3 or Bad Pointer)

4). Loop through library table (LIBTBL). Call the handler if it exists, if not signal 'LINKAGE-FAULT$'. The first entry in the table is a pointer to the ECB for HCS$ in seg 5. This routine scans seg 5 for the Direct Entry Call.

The second entry in the table is a pointer to the ECB for SNAP$3. This routine scans a list of ring 3 direct callable ECB'S.

Further entries in the table are pointers to the ***ECBs*** for the shared library fault handlers.

5). The fault handlers return the address of the ECB for the original call. The link is then snapped. If the handlers fail to find the ECB then signal 'LINKAGE-FAULT$'.

6). In the case of shared libraries the fault handler checks location 4 of the stack segment to make sure the local data of the library package has been loaded into the users segment '6001.

linkage

DIRECT ENTRANCE CALLS

The direct entrance call mechanism provides a form of dynamic linking using the standard Procedure Call (PCL) instruction (V - Mode only) and the indirect memory address pointer. The purpose of the direct entrance call is to provide an efficient mechanism that allows application programs (also system programs) to make calls to procedures that are part of the operating system or shared libraries without the overhead normally associated with other methods such as the Supervisor Call (SVC) instruction. The advantages of the direct entrance call are; first the same procedure can be shared by all users on the system without the need to have a unique copy for each, thus wasting valuable memory space, second, since the address linkage to the procedure is not made until execute time a program that makes use of these procedures does not have to be relinked for a different revision of PRIMOS where the location of the procedure may change.

Part of the implementation of this mechanism requires a special form of object module be loaded into the library that is searched when doing the program load. This object module is created by assembling a PMA program that has the form

```
SEG
DYNT procedure name
END
```

This object module triggers special action by the SEG loader when it is resolving the address linkages for called routines. When SEG encounters this structure it puts an indirect pointer in the link frame of the calling procedure that has the fault bit set and points to a location in the procedure area where SEG has put the name of the direct entrance call and the number of characters. That is all that happens at load time.

At execute time when the call is made to the procedure the fault bit causes the hardware to detect a pointer fault and the pointer fault handler is entered. The pointer fault handler attempts to resolve the address linkage to the called procedure by searching through various lists of ECBs or entry points to the direct entrance callable routines. If it finds the one it wants it puts the address pointer to the procedure back in the address pointer that originally caused the pointer fault, erases the fault bit and reexecutes the call which now proceeds as usual. If it doesn't find it or finds that the pointer is bad it raises a condition and returns

Direct Entrance Calls

I. Ring 0

Entry point definitions - PRIMOS>INSERT>GATES.INS.PMA

Entry points reside in - PRIMOS>KS>SEG5.PMA

List Name - SEG5

Memory Location - Segment 5

Search routine - HCS$ (PRIMOS>KS>HCS$.PMA) (first entry in SEG5)

I. Ring 3

Entry point definitions - PRIMOS>INSERT>R3ENTS.INS.PMA

Entry points reside in - PRIMOS>R3S>SNAP$3.PMA

List name - LIST

Memory location - Segment 13

Search routine - SNAP$3 (PRIMOS>R3S>SNAP$3.PMA)

..I. Shared Library

Entry point definitions - HTAB (Each library that is to be shared has a table called HTAB in it's source file UFD)

Entry points reside in - DIRECV>R3POFH.PMA (there will be a copy of this procedure, each with it's own HTAB, for each shared library installed.)

List name - HTAB

Memory Location - Segment 2xxx (same segment library resides in)

Search Routine - R3POFH (DIRECV>R3POFH.PMA)

LIBTBL

LIBTBL is a table that contains address pointers to the search roucines for the various direct entrance callable "packages". It is used by the Ring 3 fault handler in attempting to resolve the direct entry link. The fault handler does a PCL indirect through each of the entries in LIBTBL whicn invokes each of the various search routines in order until the link is made. The order of search is Ring 0 DECs first, then Ring 3, then shared lioraries. A typical LIBTBL is shown below (this is a Rev. 18.3 version).

In Segment 13/1434

```
1434/ 5        Pointer to SEG5 (first ECB is HCS$)
1435/ 0
1436/ 13       Pointer to SNAP$3
1437/ 400
1440/ 62050    Pointer to R3POFH
1441/ 1170
1442/ 62014          "
1443/ 41170
1444/ 62014          "
1445/ 1170
1446/ 62021          "
1447/ 1165
1450/ 62001          "
1451/ 1170
1452/ 62057          "
1453/ 1170
1454/ 62071          "
1455/ 1170
1456/ 62121          "
1457/ 1170
1460/ 62026          "
1461/ 0
1462/ 0        End of LIBTBL
```

Section 9 - Interrupt Handling

CLOCK PROCESS

The clock interrupt is treated like any other device interrupt. An address ('63) is presented by the controller. The hardware interprets this location as the address of the Phantom Interrupt Code (PIC) in Segement 4 for this device.

The PIC executes an INEC which acknowledges the interrupt, clears the Active Interrupt flag, and does a NOTIFY to CLKSEM.

The clock process will then be entered.

```
     1).  Handle PBHIST.
     2).  Reset location '61.
     3).  Display memory location selected by switches.
     4).  Increment ONE-MINUTE timer.
          If timer equals 0, then
              A). reset timer
              B). set USER 1 MINALM Abort Flag and NOTIFY ASRSEM
     5).  Increment timer 2 (Paper Tape Punch) (1/75 second).
             If zero, reset clock and call BRPDIM (if chars in buffer).
     6).  Increment Timer 3 (Digital input)
             If zero, reset timer and enter DIGDIM
     7).  Increment timer 4 (ASR) (1/30 or 1/10 second).
             If zero, reset clock and call ASRDIM.
```

CLOCK PROCESS

8). Increment timer 5 (1/10 second).

If zero , doing the following:

A). Reset clock

B). Display Segment number in lights

C). Update clock ring

D). Handle USER timer semaphores

E). Increment Timer 9 (DISK)

If zero, reset clock and NOTIFY DSKSEM

F). Increment Timer 10 (SMLC) 1/2 second, if zero

1. Reset clock
2. Set USER 1 SMLALM Abort Flag

G). Increment Timer 11 (Gross Network) 10 second, if zero

1. Reset clock
2. Set USER 1 NETALM Abort Flag

H). Increment Timer 12 (PNC) 1 second. If zero,

1. Reset clock
2. Set USER 1 NETALM Abort Flag.

I). Increment Timer 13 (Remote USER I/O) 1/2 second

If zero,

1. Reset clock
2. Set USER 1 NETALM Abort Flag

J). Increment Timer 14 (4 second). If zero,

1. Reset clock
2. Update Date and Time for TIMMOD

9). Wake up PNCDIM if PNC configured

10). Call CENDIM, CENDIM2, PTRDIM if there are chars in buffer(s).

11). WAIT CLKSEM.

THE QAMLC/ICS Driver (AMLDIM/ASYDIM)

The AMLQ will configure itself to drive up to eight controllers using device addresses '54, '53, '52, '35, '15, '16, '17 and '32. The default configuration can be changed using the AMLC command at the system console or in PRIMOS.COMI

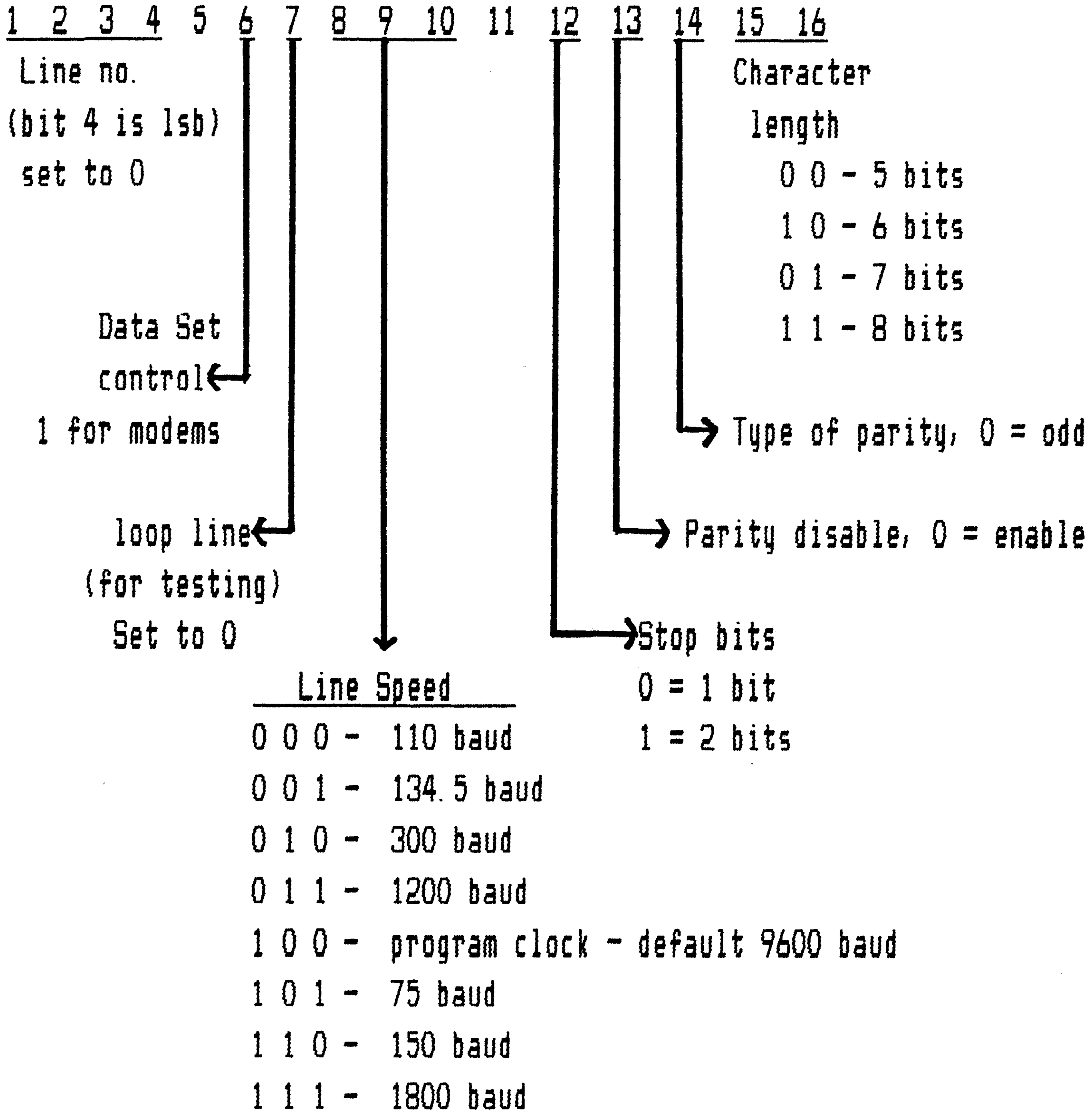
```
AMLC  [PROTOCOL]  LINE  [CONFIG]  [LWORD]
```

PROTOCOL

TTY	terminal protocol (default protocol)
TRAN	transparent protocol
TTYUPC	upper case output protocol
TTYNOP	ignore this line (used for assigned lines)

LINE	The AMLC line number (octal)
CONFIG	See line configuration table.
LWORD	See LWORD table.

LINE CONFIGURATION TABLE



1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

Line no.
(bit 4 is lsb)
set to 0

Character
length
0 0 - 5 bits
1 0 - 6 bits
0 1 - 7 bits
1 1 - 8 bits

Data Set
control
1 for modems

Type of parity, 0 = odd

loop line
(for testing)
Set to 0

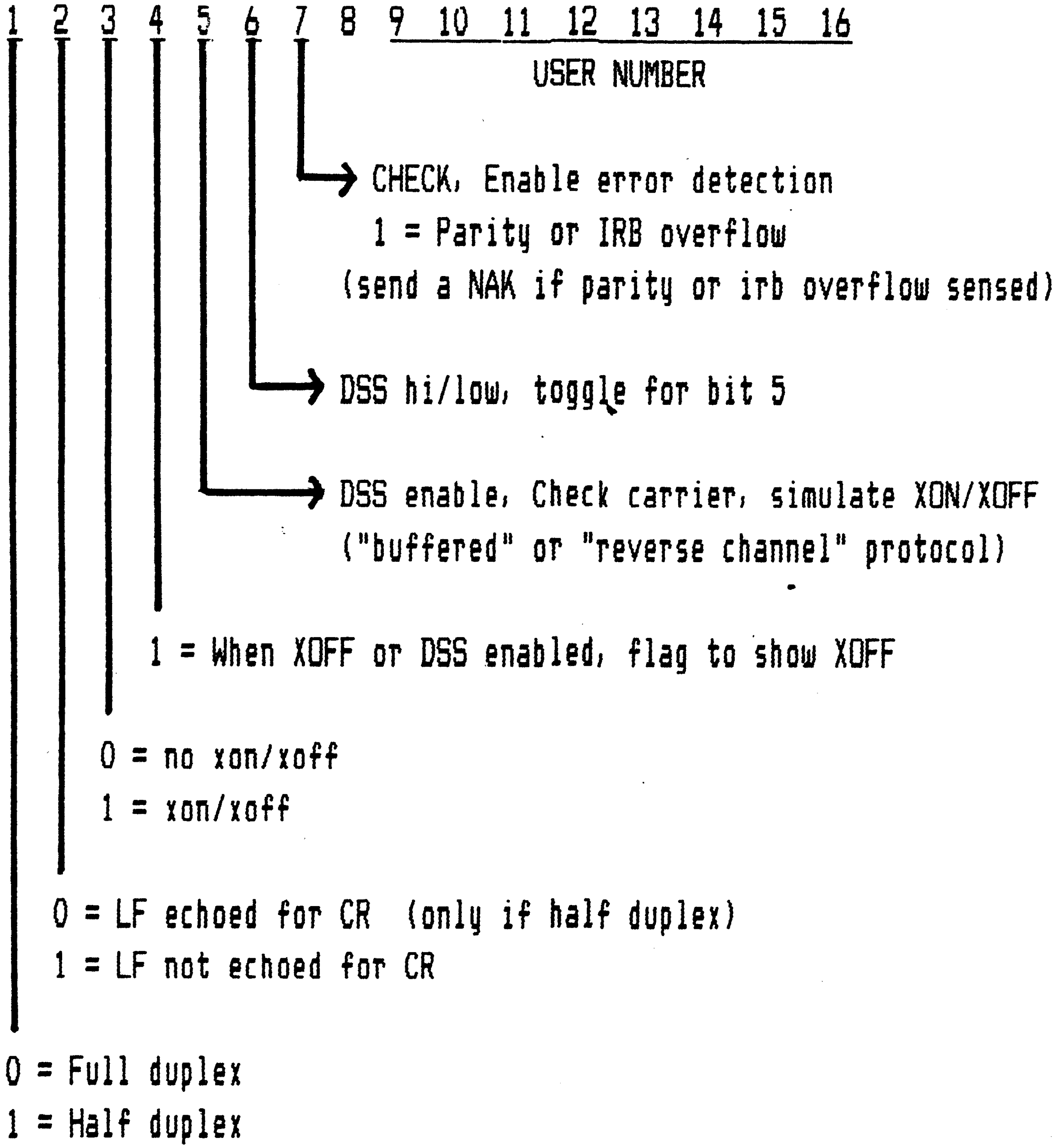
Parity disable, 0 = enable

Stop bits
0 = 1 bit
1 = 2 bits

Line Speed

0 0 0 -	110 baud
0 0 1 -	134.5 baud
0 1 0 -	300 baud
0 1 1 -	1200 baud
1 0 0 -	program clock - default 9600 baud
1 0 1 -	75 baud
1 1 0 -	150 baud
1 1 1 -	1800 baud

LWORD TABLE



1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

USER NUMBER

CHECK, Enable error detection
1 = Parity or IRB overflow
(send a NAK if parity or irb overflow sensed)

DSS hi/low, toggle for bit 5

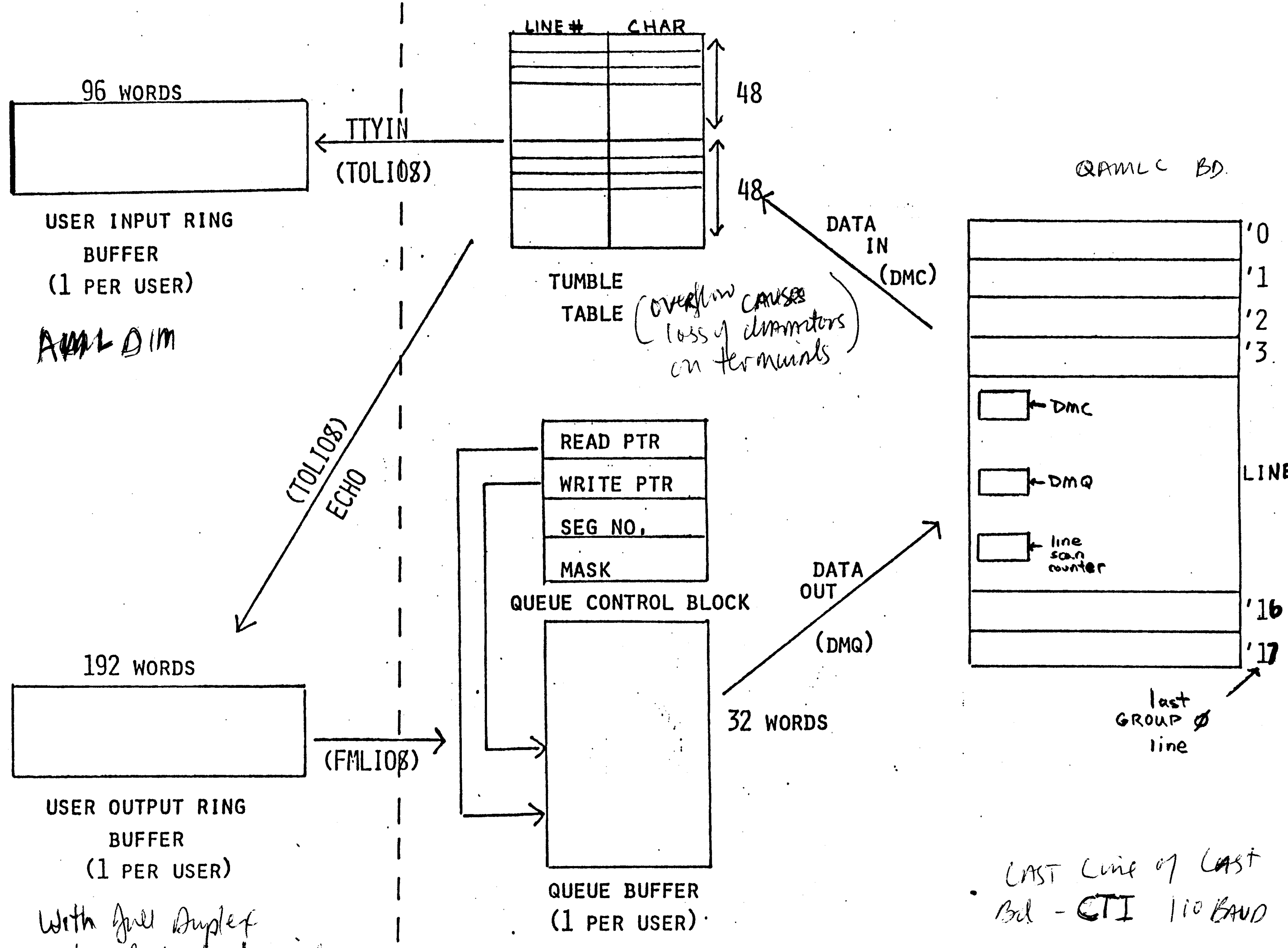
DSS enable, Check carrier, simulate XON/XOFF
("buffered" or "reverse channel" protocol)

1 = When XOFF or DSS enabled, flag to show XOFF

0 = no xon/xoff
1 = xon/xoff

0 = LF echoed for CR (only if half duplex)
1 = LF not echoed for CR

0 = Full duplex
1 = Half duplex



BLOCK DIAGRAM OF AMLDIM (AMLQ)

THE AMLQ - Notes on the diagram

1). There can be up to 8 boards.

2). All lines are configured into group 0.

3). The speeds of the lines are set by default as follows:
All lines except the last line on the last board
- 1200 baud, Normal TTY protocol
Last line - 110 baud, TTYNOP

4). The last line defines the rate at which all lines are scanned for both input and output. The default is 10 times per second.

ICS

1). There is no special line to determine the line scan rate. The rate is fixed at 10 times per second.

2). The ICS boards use DMQ for input instead of tumble tables.

CONFIG DIRECTIVES for Lines

Creates Assignable Buffers

Terminal users

NAMLC, NTUSR

Set Programmable clock

AMLCLK baudrate

Timing for chkg carrier on lines

AMLTIM [ticks] [disctime] [gracetime] → Amt of time befor line Drops

(default = 2, 3410, 0)

DTR (DATA terminal Ready)

DTRDRP When terminals log out drop DTR

Default

DISLOG { NO ¦ YES } Auto line Log out if cable Disconnect (default = NO)

AMLIBL - Changes tumble table (default = '60)

ICS INPQSZ [size] (default = '77)

Has no last line Always 10 times per sec

ICS JUMPER [speeda] [speedb] [speedc]

Software initated jumpers

CONFIG DIRECTIVES - For User Buffers

1 command 3 Different forms

```
NAMLC   number-of-buffers                      (default = 0)
```

3 types

input Buffer Size

output Buffer Size

```
AMLBUF  amlc-line           0              0             dmq-size
AMLBUF  user-buff-no        in-buff-size   out-buff-size
AMLBUF  assigned-buff-no    in-buff-size   out-buff-size
```

1) 2) combine these for Terminals

3)

DMQ does not go thru processor there is no ck on when buffer is full

~~[illegible]~~ DESIRED Baud Rate / # of Bits per char.

Above devided By Last line Baud Rate / # Bits per char.

```
AMLBUF  amlc-line           200 (128)      300 (192)     40 (32)
    default: user-no      =  amlc-line + 2
    SINCE: user-buff-no =  user-no   - 2
    THEN: amlc-line = user-buff-no (if user-no is default)
    assigned-buff-no-1 = NTUSR + NRUSR - 1 (rotating pool)
```

Always true

```
REMBUF  in-buff-size  out-buff-size       (default = 200, 300)
```

First user = # 2

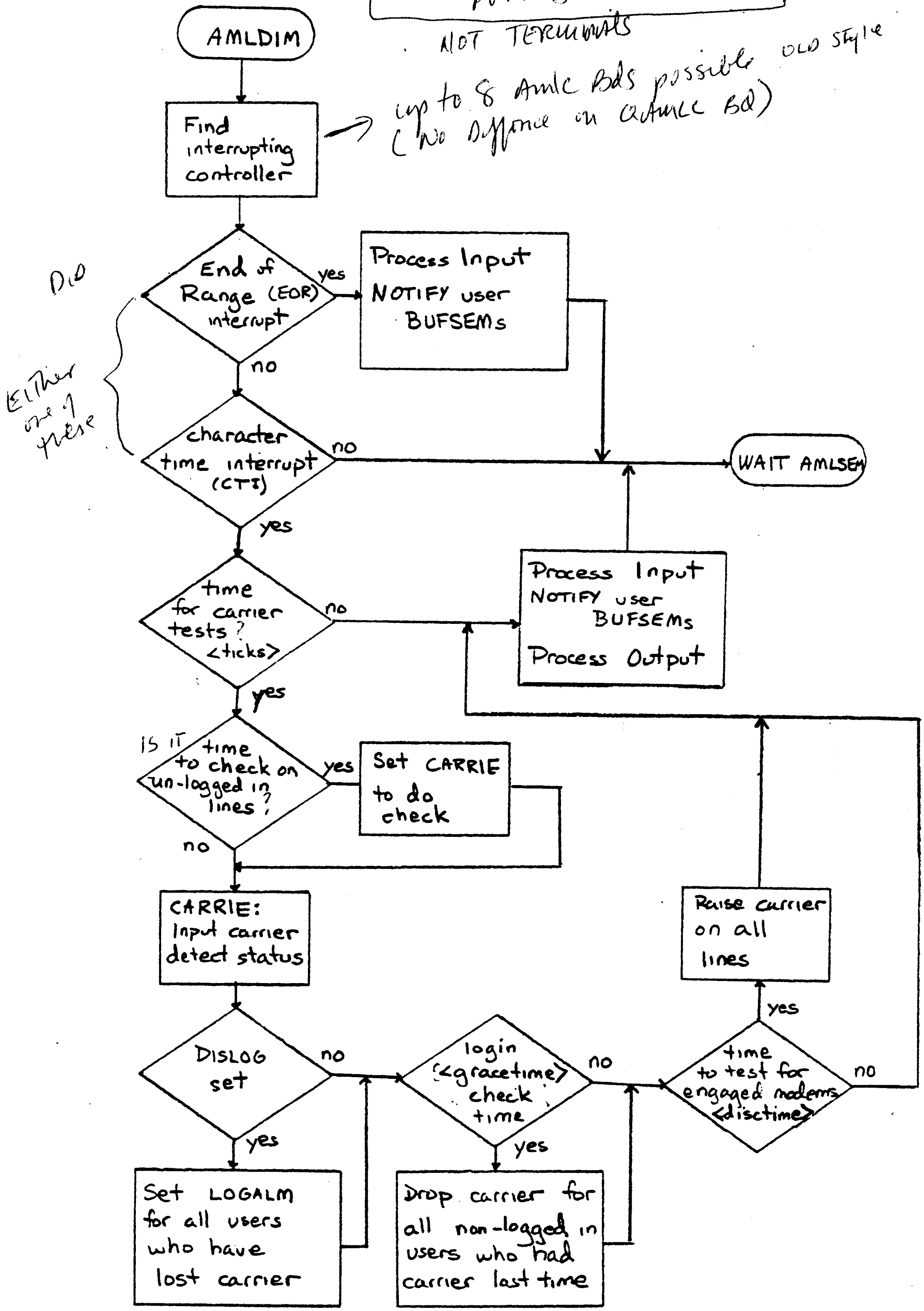
First AMLC Buffer = # 0

No buffer for Sys console

To increase Buffer

1. increase Last line Baud to 300 (utilizes processor)
2. increase DMQ Size

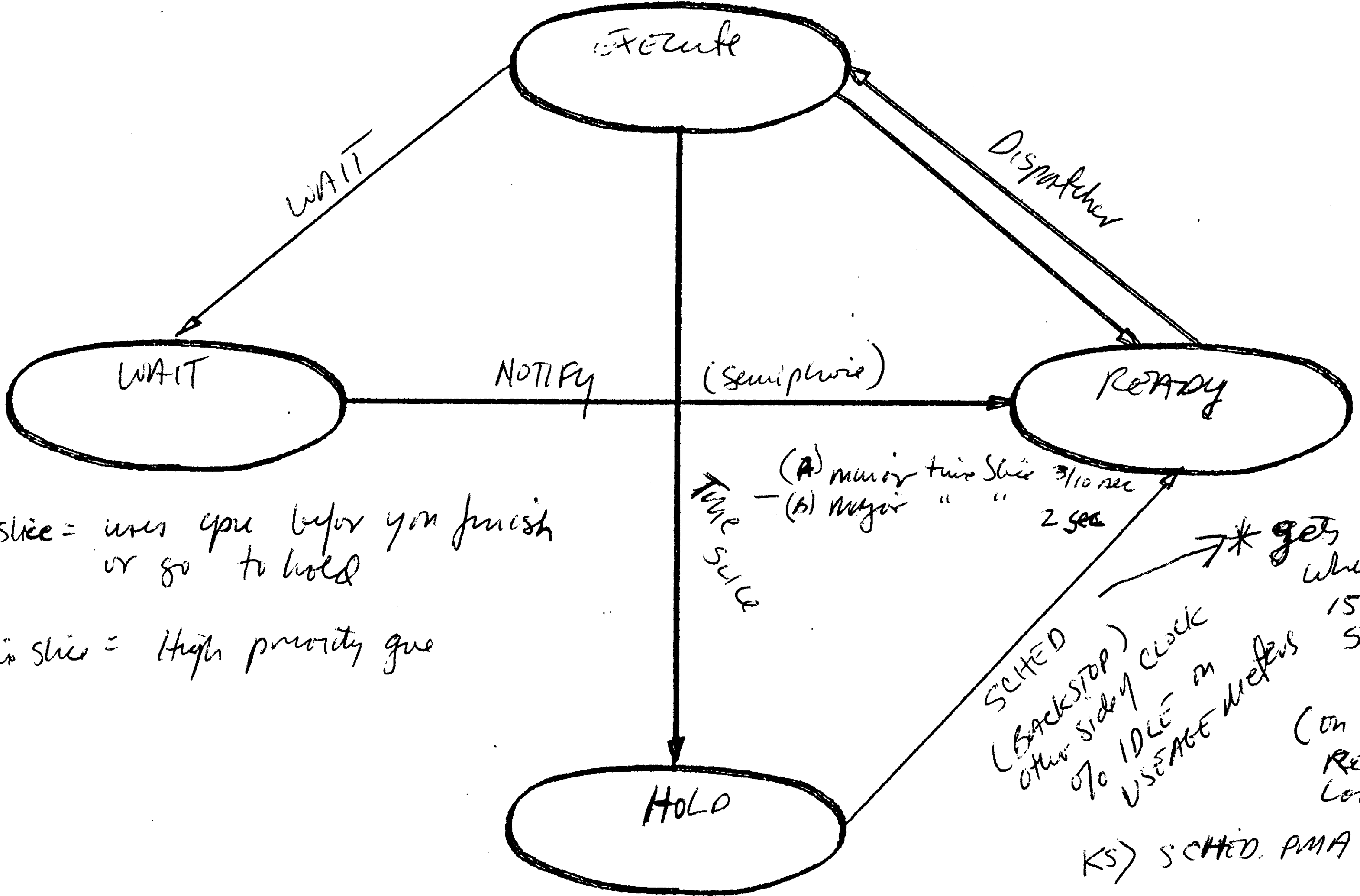
For things going thru Port Selectors


NOT TERMINALS
AMLDIM
Find interrupting controller
up to 8 Amlc Bds possible OLD Style
(No Difference on QAMLC Bd)
DID
End of Range (EOR) interrupt
yes
Process Input
NOTIFY user BUFSEMs
no
Either one of these
character time interrupt (CTI)
no
WAIT AMLSEM
yes
time for carrier tests? <ticks>
no
Process Input
NOTIFY user BUFSEMs
Process Output
yes
IS IT
time to check on un-logged in lines?
yes
Set CARRIE to do check
no
CARRIE: Input carrier detect status
Raise carrier on all lines
yes
DISLOG set
no
login (<gracetime>) check time
no
time to test for engaged modems <disctime>
no
yes
Set LOGALM for all users who have lost carrier
yes
Drop carrier for all non-logged in users who had carrier last time

Section 10 - Scheduling of Users

Process Exchange

Hi Priority Que
ELIGIBILITY
4 L PRI Que
3 L PRI Que
2 L PRI Que
1 L PRI Que
Ø L PRI Que



Stored in PCB

* gets to run when everyone is on WAIT state or Hold state (on Bottom of Ready List Lowest Priority)

KS> SCHED. PMA

Minor time slice = uses cpu befor you finish or go to hold

Major time slice = High priority que

Favors interactive user
5 levels to keep lower priorities on Hold state longer befor they get on Ready List

SCHEDULING OF USERS

Sched takes people off ques - on

Listen - puts people on Hi Priority Que

PRIMOS scheduling is based on two criteria.

1). PROCESS EXCHANGE - See Apendix B
2). BACKSTOP PROCESS (SCHED)

(5) Low priority Que (1 for every user level

The process exchange mechanism is implemented in firmware and uses the ready list/wait list philosophy described earlier.

SCHED, also known as the backstop process:

1). Responding to requests for users to be placed on one of three queues and allocating a time-slice.
2). Deciding the sequence of processes placed on the READY LIST.

SCHED maintains three basic queues using semaphores.

A). High priority (interactive users)
B). Eligibility
C). Low priority (compute bound users)

When a user process returns to command level, the listener is called to a invoke a new command level and CL$GET is called to read in the command line. C1IN$ is then called to read in the characters. C1IN$ will wait on BUFSEM (there is one BUFSEM semaphore per user) and when a character is input into the user ring buffer the AMLC driver will notify BUFSEM. The user will continue to use C1IN$ to input characters until a <CR> character is detected.

On detecting <CR> CL$GET calls SCHED to place the user process on the HIGH priority queue and to allocate a full time-slice. SCHED scans for high priority users before any others and a user in the high priority queue will be placed on the ready list and scheduled to run with a timeslice of 3/10 sec. At the end of this period the process will fault and be placed on the elgibility queue. The backstop process scans the elgibility queue after the high priority queue and eventually the user will be notified and moved on to the ready list with another timeslice of 3/10 sec.

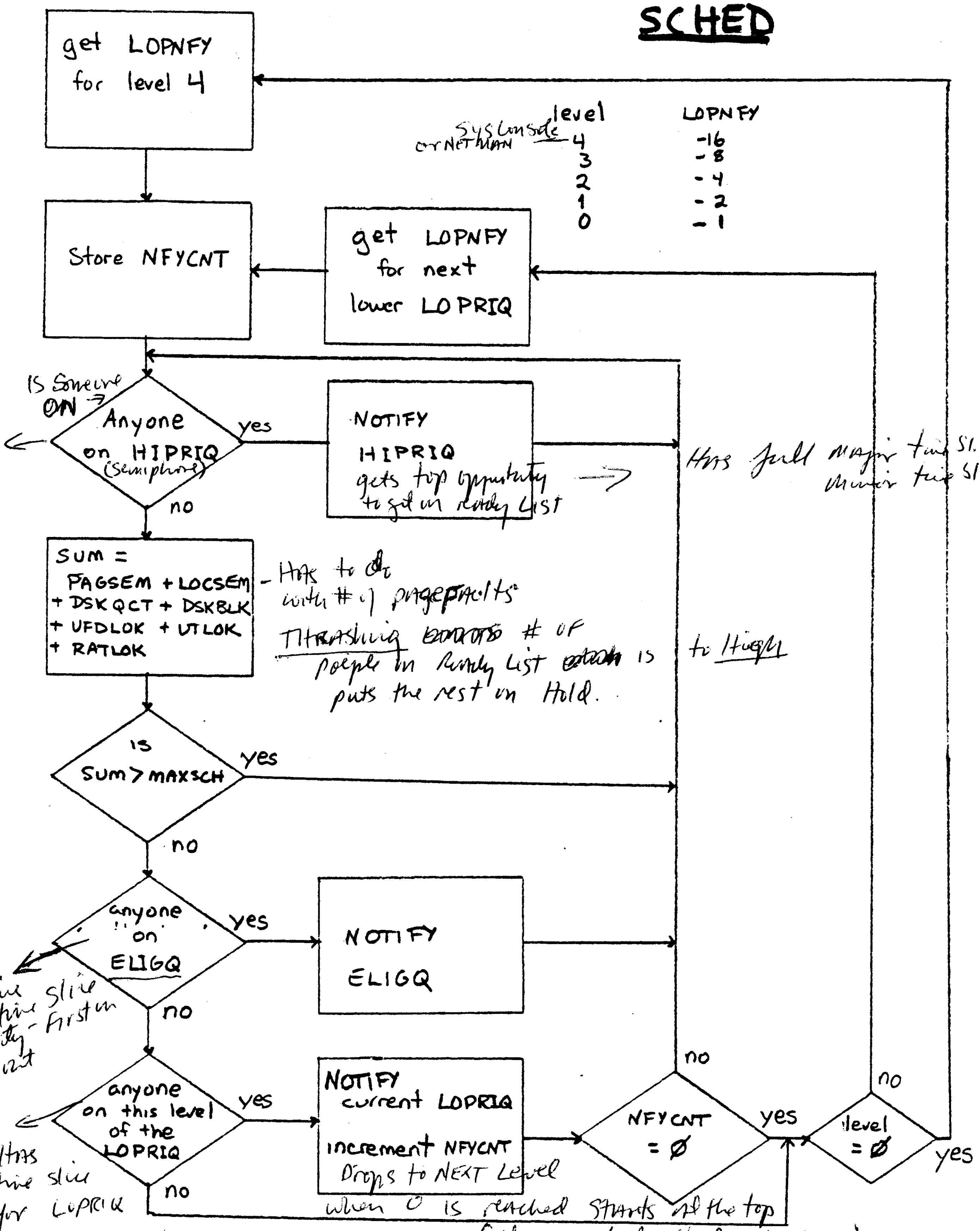
This sequence of events continues until the full 2 second time-slice has elapsed. The process is then placed on the low priority queue appropriate to its priority level. The backstop process maintains five semaphores in the low priority queue for this purpose:

Supervisor level (level 4)
User level 3
User level 2
User level 1 (default user level)
User level 0

The backstop process will schedule users on the low priority queue after both the high priority and the elgibility queues have been exhausted according to the following flowchart.

Semiphore - gets PCB chained on to List
this count field

SCHED



level	LOPNFY
4	-16
3	-8
2	-4
1	-2
0	-1

Sys Console or NETMAN

Is Someone ON →

gets top opportunity to get on Ready List →

Has full Major time Sl. Minor time Sl.

- Has to do with # of pagefaults

Thrashing ~~[illegible]~~ # of people in Ready List ~~each~~ is puts the rest on Hold.

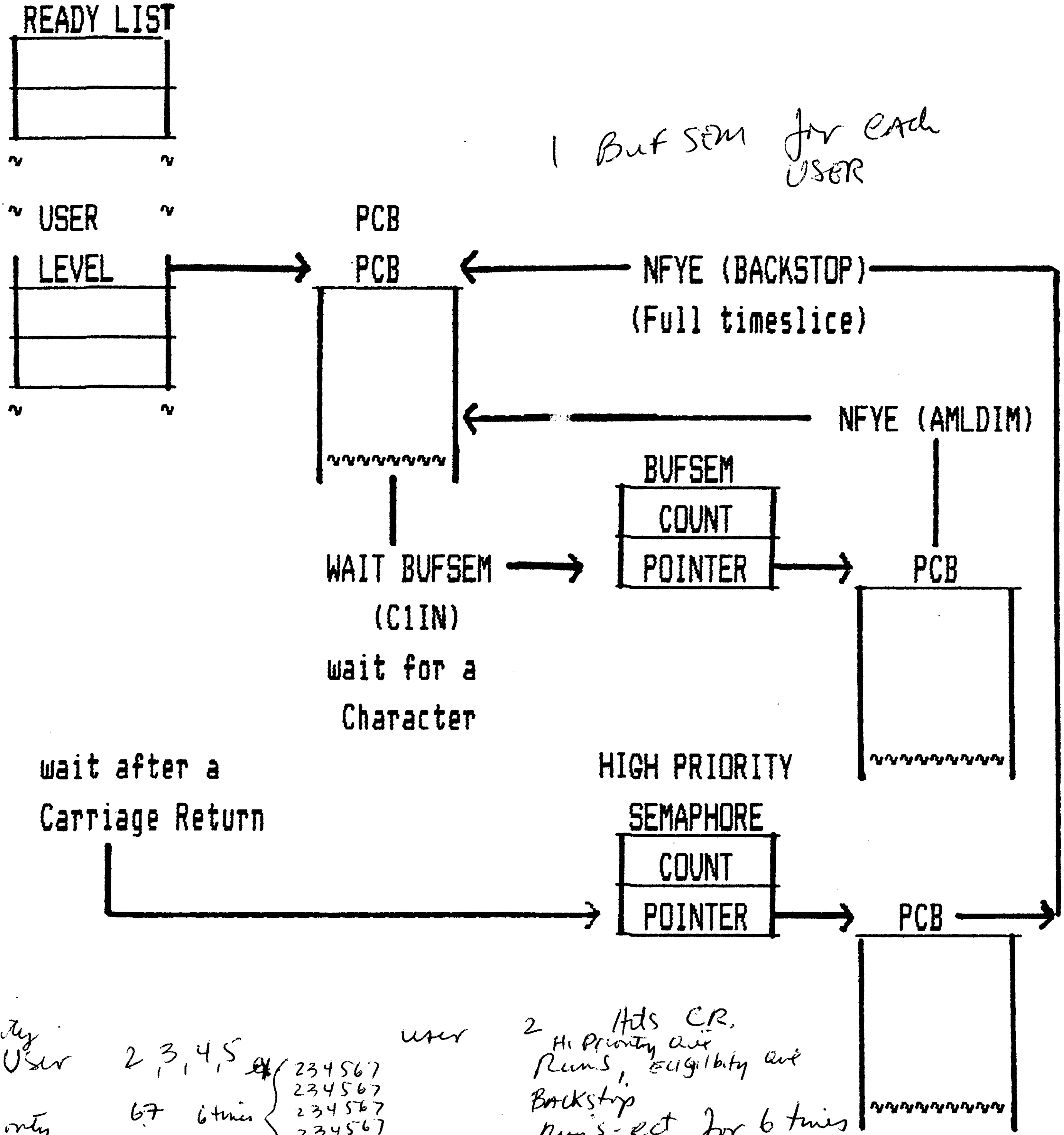
to High

USERS who require Minor time slice No Priority - First in First out

No one Has Major time slice left befor LOPRIQ is Run - ~~Runs~~ 1/6 of High priority Que

Drops to NEXT Level when 0 is reached Starts at the top (Always checks Hi Priority Que) while it Runs

INTERACTIVE USER



1 Buf Sem for each USER

Hi Priority User 2, 3, 4, 5

Lo priority 6 7 6 times

234567
234567
234567
234567
234567
234567

user 2 Hits CR, Hi Priority que Runs, eligibity que Backstop Run's-ect for 6 times

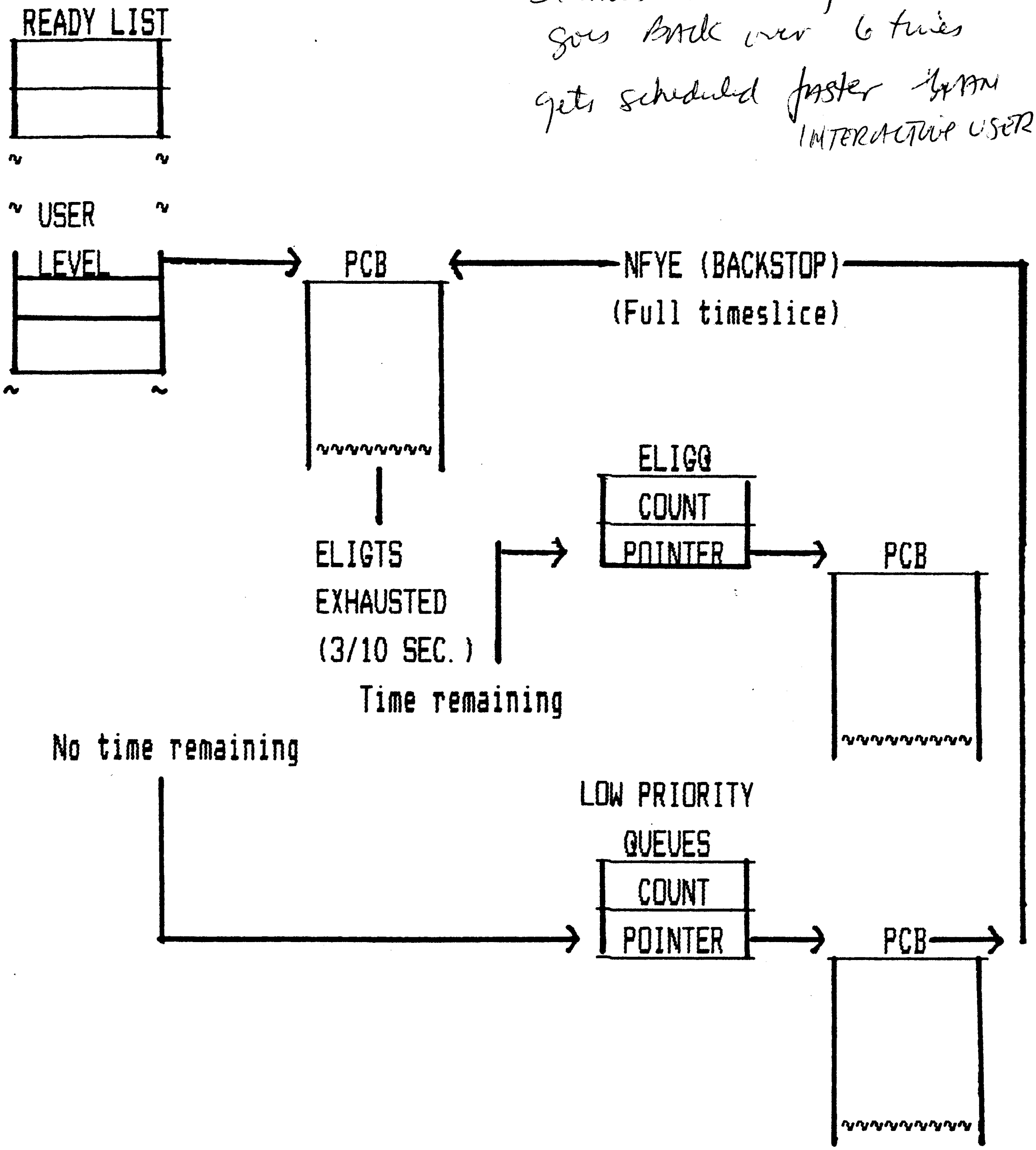
① L PRI que - 2 3 4 5
② L PRI que 6, 7

Scheduler gives Benefit of Priorities

222222
333333
666666
444444
555555
777777
222222
333333
666666
444444

COMPUTE BOUND USER

Exhausts ELIGIBility time slice
goes Back over 6 times
gets scheduled faster than
INTERACTIVE USER



READY LIST

USER LEVEL

PCB

NFYE (BACKSTOP)
(Full timeslice)

ELIGTS EXHAUSTED (3/10 SEC.)

Time remaining

ELIGQ
COUNT
POINTER

PCB

No time remaining

LOW PRIORITY QUEUES
COUNT
POINTER

PCB

USER PRIORITIES AND TIME-SLICE

The following operator command is available for changing user priorities and time-slice.

Chap user up if it is to run faster

```
CHAP  [-USERNO/ALL] [PRIORITY] [TIME-SLICE]
      USERNO        Is in the form -nn or ALL
      PRIORITY      Integer 0 to 3 (default = 1)
      TIME-SLICE    Length of time-slice in tenths of seconds.
                    0 means reset to the system default (2 sec.)
                    If omitted the time-slice is unchanged.
   If both priority and timeslice are omitted, then priority and
   time-slice are set to the system default values.
```

Makes more of a diff than priority level

(can sometimes speed things up more ~~than~~ changing priority level)

STAT US Displays the priority of users not at user level 1.

LOGOUT Resets priority and timeslice to defaults.

ELIGTS Is used to modify the elgibility time-slice from the system console. This will affect all users equally.

```
ELIGTS [<eligibility_timeslice>]   (default = 3/10 sec.)
```

MAXSCH

IN OCTAL

Previously, MAXSCH was determined by indexing into an array of values; 0,0,1,2,3,4,4. The value of the index was the memory size in 32K units. If there was more than 256K then MAXSCH would be 4.

MAXSCH is now calculated as follows:

on 850
MAXSCH = 7

MAXSCH = (megabytes_of_memory + 3) * x + y
5 *1 *1

where, x is 1.2 if there exists an alternate device on a different controller than the primary device, otherwise it is 1.
y is 1 if CPU is a P850, otherwise it is 0.

IF MAXSCH is to high thrashing will result Causing page faults

if greater than 12 page faults per sec profomance is degraded

The optimal value of MAXSCH is application dependent, hence there is no hard and fast formula to determine its value. Therfore, it is a configurable parameter.

rule of thumb:

$$\text{MAXSCH} = \frac{\text{Physical-Memory-Size} - \text{PRIMOS-locked-memory}}{\text{average-job-size}}$$

Section 11 - User Profiles

USER PROFILES

Security Protection

MOTIVATION

- To provide secure user registration.
- Provide central database to store per user attributes.
- Provide mechanism to define a group of users with similar attributes.

IMPLEMENTATION

- Rev. 19 PRIMOS validates users at login; all users must be registered BEFORE they can login.
- All profile information stored in the System Administrator Database (SAD ufd). (Manipulated By) ↓
- SAD is manipulated by EDIT_PROFILE utility.
- Access to SAD controlled by ACLs.

USER PROFILES - DEFINITIONS

User-id -- A 32 character name uniquely identifying user.

Login Password -- A 16 character string known only to the owning user. Supplied at login to validate user-id Stored on the disk encrypted.

Project -- A collection of users with similar system attributes.

System Administrator (SA) -- The user resposible for administering the profile database.

Project Administrator (PA) -- A user delegated administrative powers over a particular project.

Initial Attach Point (ORIGIN) -- UFD where a user is attached after successful login. Need not be a top-level ufd.

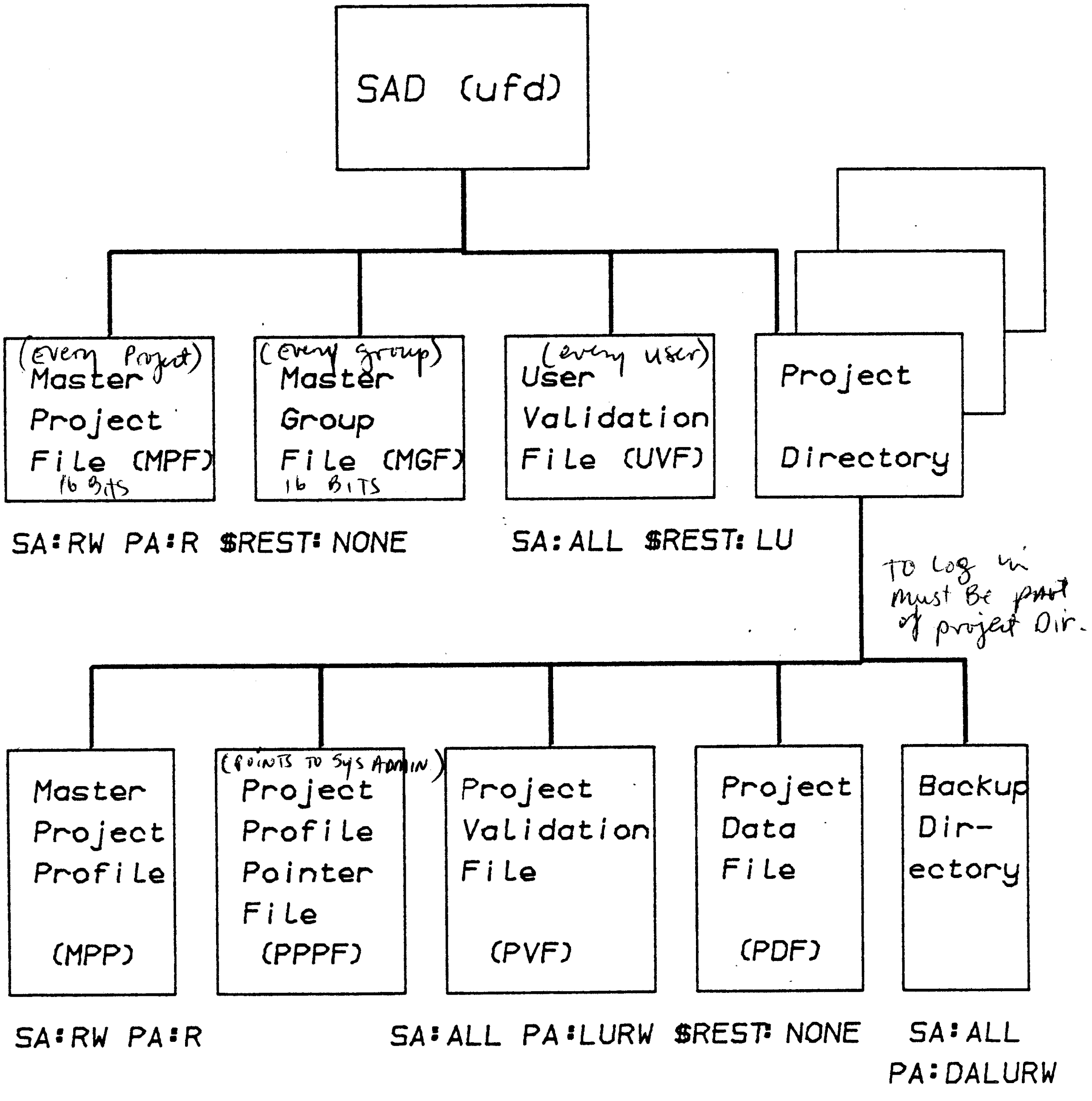
ACL group -- A symbolic name which may be used in an ACL. The user's profile defines group membership.

Project 'Limits' -- The set of parameters which the PA is allowed to administer. Currently a list of ACL groups only.

Profile -- The set of parameters defining per user or per project attributes. Currently a list of ACL groups and ORIGIN.

USER PROFILES - SAD FILES

On Rev 19.2 SAD
must be rebuilt



SAD (ufd)

(every project) Master Project File (MPF) 16 bits

(every group) Master Group File (MGF) 16 bits

(every user) User Validation File (UVF)

Project Directory

SA:RW PA:R $REST: NONE

SA:ALL $REST: LU

To log in
must be part
of project Dir.

Master Project Profile (MPP)

(points to sys admin) Project Profile Pointer File (PPPF)

Project Validation File (PVF)

Project Data File (PDF)

Backup Dir- ectory

SA:RW PA:R

SA:ALL PA:LURW $REST: NONE

SA:ALL
PA:DALURW

USER PROFILES - SAD FILES

MPF - MASTER PROJECT FILE

Contains one 16 word entry for each project on system
(not ordered) (32 Char)
ACCESS: SA:RW PA:R $REST:NONE
dcl project_id char (32) based;

MGF - MASTER GROUP FILE

Contains a 16 word entry for each ACL group on system
(not ordered)
ACCESS: SA:RW PA:R $REST:NONE
dcl group_name char (32) based;

UVF - USER VALIDATION FILE

Contains a 16 word header.
Contains a 48 word entry for each user on system.
User entries are hashed by User I.D.
ACCESS: SA:ALL $REST:LU
RWLOCK: NONE

USER PROFILES - SAD FILES

```
dcl 1 vf_header based,  /* Header for validation files(UVF,PVF) */
        2 free_ptr fixed bin (31),     /* Current length of file */
        2 oflo_ptr fixed bin (31), /* Location of overflow area */
        2 admin_ptr fixed bin (31),/* Pointer to entry of SA/PA */
        2 entry_size fixed bin,
        2 table_size fixed bin,    /* Size of prime hash table  */
        2 bucket_size fixed bin,   /* Size of a bucket in table */
        2 entries_used fixed bin,
        2 overflows fixed bin,   /* Current number of overflows */
        2 bits,
            3 ggrps bit (1),   /* System supports global groups */
            3 pgrps bit (1),  /* Project supports groups */
            3 projects bit (1),        /* Projects exist */
            3 no_acls bit (1),       /* SAD is not ACL-protected */
            3 no_null_pw bit (1), /* Null passwords not allowed */
            3 force_pw bit (1),/* Don't allow password on login line */
            3 mbz bit (10),
        2 version fixed bin,      /* EDIT_PROFILE version number */
        2 reserved (3) fixed bin;
```

USER PROFILES - SAD FILES

```
dcl 1 uvf_entry based,
        2 user_id char (32),
        2 password char (16),
        2 dft_project_ptr bit (16) aligned, /* Pointer into MPF */
        2 site_rsvd (4) fixed bin,    /* Reserved for site use */
        2 last_login_date,            /* Date of last login */
            3 year bit (7) unal,      /* Year (mod 100) */
            3 month bit (4) unal,     /* Month */
            3 day bit (5) unal,       /* Day */
        2 last_login_time fixed bin,  /* Quadseconds since midnight */
        2 rsvd fixed bin,             /* Reserved for future use */
        2 group_ptr (up_maxgrp) bit (16) aligned;/* Pointers to MGF */
```

USER PROFILES - PROJECT FILES

MPP - MASTER PROJECT PROFILE

This file defines the project 'limits'.

Currently valid groups for this project.

One 48 word entry.

ACCESS: SA:RW PA:R $REST:NONE

```
/* Master Project Profile (MPP) */

dcl 1 mpp_entry based                   /* Only one of these per project */
        2 limit_rsvd_1 (16) fixed bin,    /* Reserved for accounting */
        2 limit_rsvd_2 (16) fixed bin,    /*     "         "      "      */
        2 group_ptr (mpp_maxgrp) bit (16) aligned;/* Pointers to MGF */
```

USER PROFILES - PROJECT FILES

PVF - PROJECT VALIDATION FILE (aka. User Profile Pointer File - UPPF)

Contains a 16 word header (like UVF header).
Contains a 48 word entry for each user in the project.
All pointers point to the Project Data File (PDF).
Entries hashed by User I.D.
ACCESS: SA:ALL PA:LURW $REST:NONE

PPPF - PROJECT PROFILE POINTER FILE

This file defines the Project Administrator,
and the 'Default Project Profile'.
There is one 48 word entry like the PVF entry.
ACCESS: SA:ALL PA:LURW $REST:NONE

```
/* Project and User Profile Pointer Files (PPPF and UPPF [PVF]) */

dcl 1 ppf_entry based, /* One in PPPF, one per user in PVF */
      2 user_id char (32),
      2 origin_ptr bit (16) aligned,             /* Pointer into PDF */
      2 process_dir_ptr bit (16) aligned,        /* Pointer into PDF */
      2 site_rsvd (8) fixed bin,               /* Reserved for site use */
      2 rsvd (6) fixed bin,                  /* Reserved for future use */
      2 group_ptr (up_maxgrp) bit (16) aligned;  /* Pointer into PDF */
```

USER PROFILES - PROJECT FILES

PROJECT DATA FILE (PDF)

Used for initial attach point and project based group names.

Contains the actual data pointed to by the PPPF and PVF.

Consists of one 16 word header followed by data blocks.

There are two types of data blocks:

Name block - 16 word (group_name or name_of_one_pathname_level).

Pathname pointer block - A 16 word array of 1 word pointers to name blocks elsewhere in file. Each array describes one pathname. Each pointer points to name of 1 level of pathname. Max. of 16 levels. Used for origin. Null ptr at end-of-list.

ACCESS: SA:ALL PA:LURW $REST:NONE

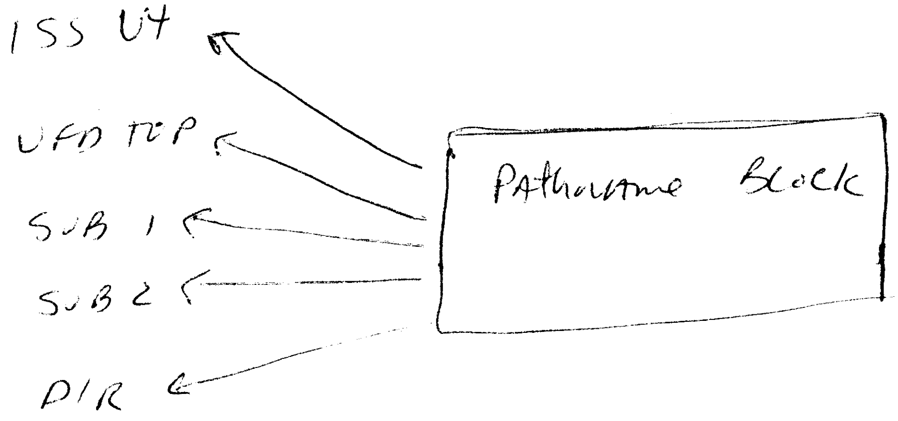
```
dcl 1 pdf_header based,
        2 free_ptr bit (16) aligned, /* Current length of file */
        2 pathname_count fixed bin,  /* Number of pathname blocks */
        2 group_count fixed bin,     /* Number of group name blocks */
        2 limit_count fixed bin,     /* Number of limit blocks */
        2 reserved (12) fixed bin;
```

BACKUP SUB-UFD

This sub-ufd is used to store copies of all project files while project is being 'rebuilt'

ACCESS: SA:ALL PA:DALURW $REST:NONE

<ISSU4> UFD TOP > SUB1 > SUB2 > DIR



Section 12 - Login/Logout

NEW LOGIN MECHANISM

MOTIVATION

- Support user registration
- Old login poorly structured
- Old login code difficult to maintain

ADVANTAGES

- User registration
- Login/Logout code separated
- DOSSUB no longer involved
- Re-coded in PLP

OLD LOGIN MECHANISM

TERMINAL USERS

```
              LISTEN (ring 0)
                 |
              DOSSUB
               /     \
          LOGIN      RLOGIN
            |
          INIT$3
```

PHANTOM USERS

```
              UNLOAD (in TMAIN or PHMSEM)
                |
              LOGIN
                |
              INIT$3
```

NEW LOGIN MECHANISM

TERMINAL USERS

```
           STD$CP                    LISTEN (ring 0)
   login      \                        /              normal
   over        LISTEN_          LOGO$CP (ring 3)   login
   login      (Ring 3) \          /
                       LOGIN$
         (Remote Login)/      \
               RLOGIN       NLOGIN
      remote                  |
      login                 INIT$U
                              |
                            INIT$3
```

PHANTOM USERS

```
        UNLOAD (in TMAIN)
          |
        PHLOGIN
          |
        INIT$U
          |
        INIT$3
```

NEW LOGIN MECHANISM

NPX SLAVES

- Started up from BINIT.

- NLOGIN used to perform validation for different naming spheres.

NETMAN (gets logged in during cold start - really is system)

- Started from NETON during initialization.

NEW LOGIN MECHANISM

LISTEN

- ring zero listener
- collects characters to form line

LOGO$CP

- logged-out command processor
- parses command line
- calls LOGOCM_ to lookup commands in LOGOCMT - the logged-out command table
- executes commands or types 'Login please.'

LOGOCMT

- logged-out command table

  valid commands: login, delay, usrasr,
  date, dropdtr

LOGIN$

- validates login
- login over login allowed, not sysusr
- calls CL$PIX to parse login command
- calls RLOGIN if going remote
- calls NLOGIN if local

NEW LOGIN MECHANISM

NLOGIN

- main login routine For Validation
- makes 'any$' handler
* calls logout if login over login
- allocates unit table (UTALOC)
* checks maxusr
* prompts for user_id, password, project, if required
- reads 'SAD' files
- validates user_id, password, project
- setup upcom data
* setup utype
- setup ACL groups
* setup initial attach point
* initialize cpu, i/o counters, etc.
* build dummy login line for external login
* call LOG_INIT
* call INIT$U
- special checks for FAM I

* These steps are NOT performed for NPX slaves

LOG_INIT

- initialize PUDCOM variables:
  limits, watchdogs, erase, kill, time-slice, priority
  terminal characteristics

NEW LOGIN MECHANISM

INIT$U - initialize PUDCOM variables:
date, vrtssw, asrcwd, famsem, in_grace_period
- initialize NPX databases
- setup unique i.d. for logout notification (UID$BT)
- open logout notification queue
- send login message to user/console
- return all segments
- allocate segments 4000, 6002
- restore external login (EXTLOG)
- call INIT$3

INIT$3 Ring 0
- initialize ring 3 stack root
- setup CLDATA variables
- initialize static on-units (INSOU$)
- turn my frame into condition frame
- crawlout

NEW LOGIN MECHANISM

INIT$3 Ring 3

- NPX slaves call SLAVER
- make special 'any$' handler
- run external login
- revert 'any$' handler
- if logging out, call FATAL$(e$logo)
- if CPL phantom start CPL program
- call INIT$P for tty users

INIT$P

- attach to I.A.P.
- find LOGIN. (.run, .cpl, .comi, .save)
- execute LOGIN.

NEW LOGIN MECHANISM

PHLOGIN
- main phantom login routine
- if slave, netman and date is set or if login over login call boot
- if top level ufd of cominput treename = FAM switch lognam to FAM
- reset cpu, i/o, etc.
- apply suffix rules to treename (SRPHAN)
- setup CPL arguments
- attach home
- release phantom lock
- setup utype
- call INIT$U

LOGIN SECURITY VALIDATION

See Handout pg 12-10

The system will prompt for a password even if the user id provided is invalid. If either the user id or the password is invalid, the user will be told that one of them is incorrect, but not which one.

If the SAD is set to force passwords, users who provide the password on the login command line will not be permitted to login, even if the password supplied is the correct one.

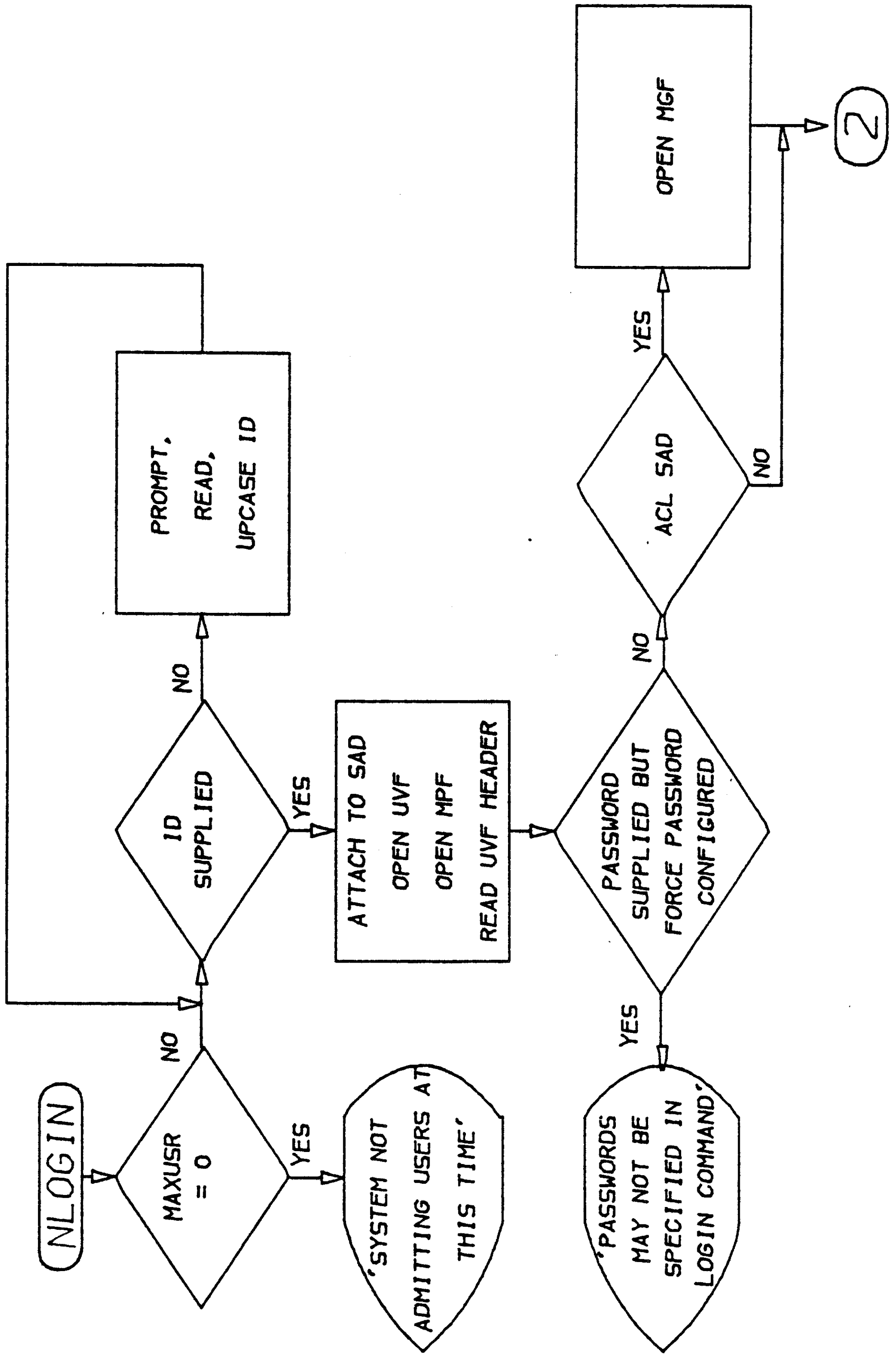
The password supplied in response to the prompt is not echoed on the terminal. It is stored in the PVF in encrypted form.

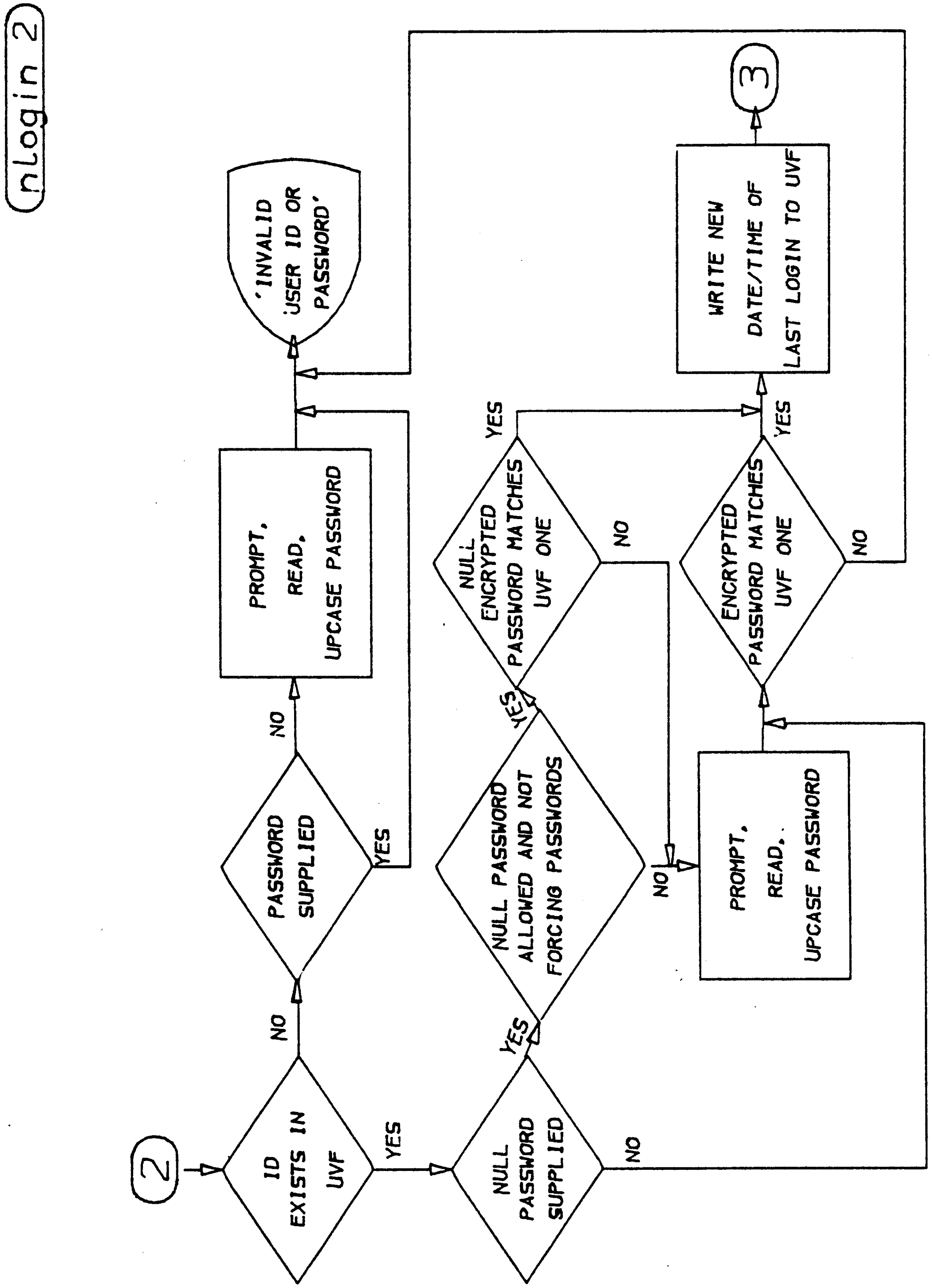
The SAD must be an ACL directory in order to enable active ACL groups.

The user will be prompted for a project if either s/he is not specified as having a default project, or s/he is not a registered member in the default project that is listed for that user.

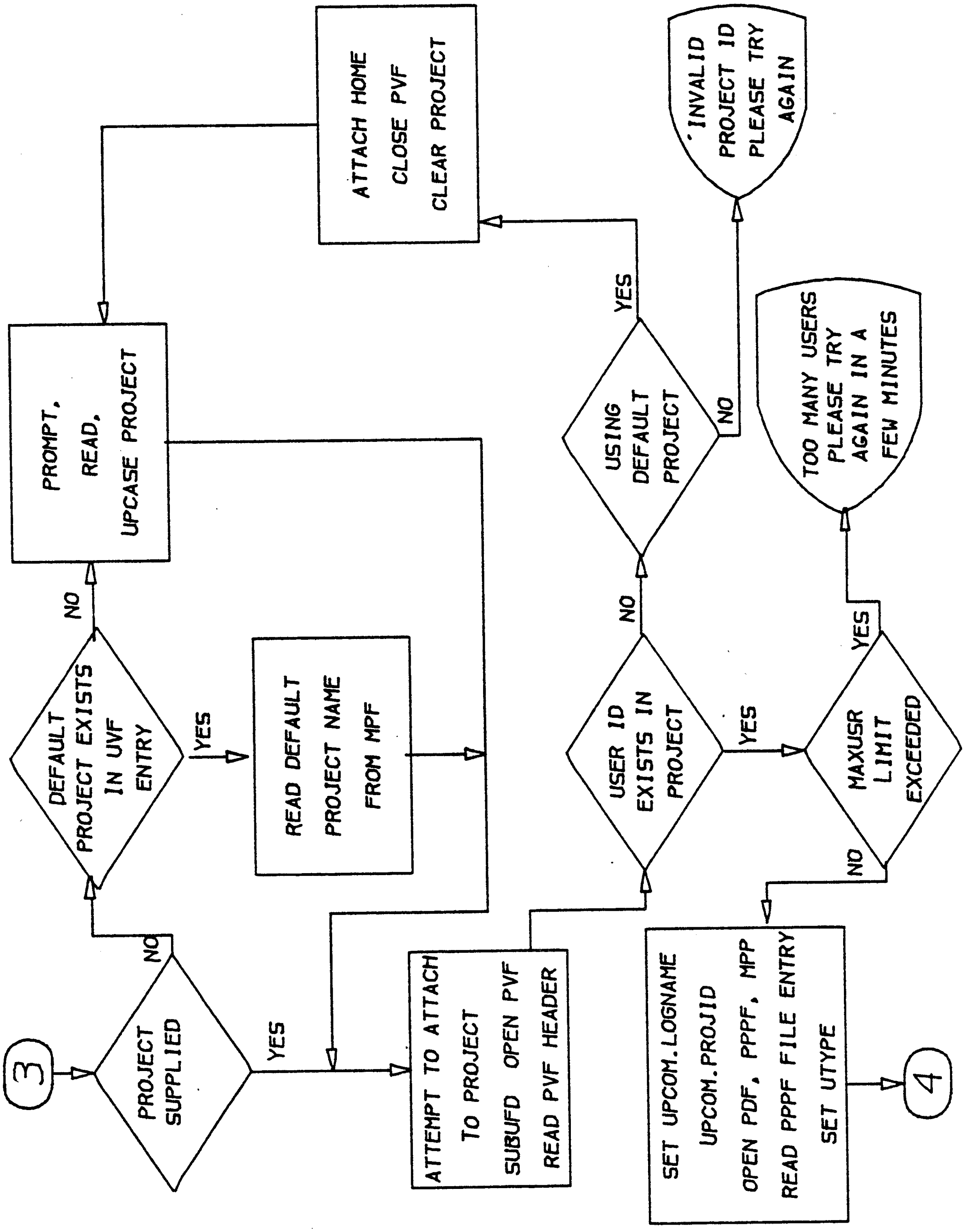
A user's project based ACL groups will only become active if they are in the MPP 'limit list.'

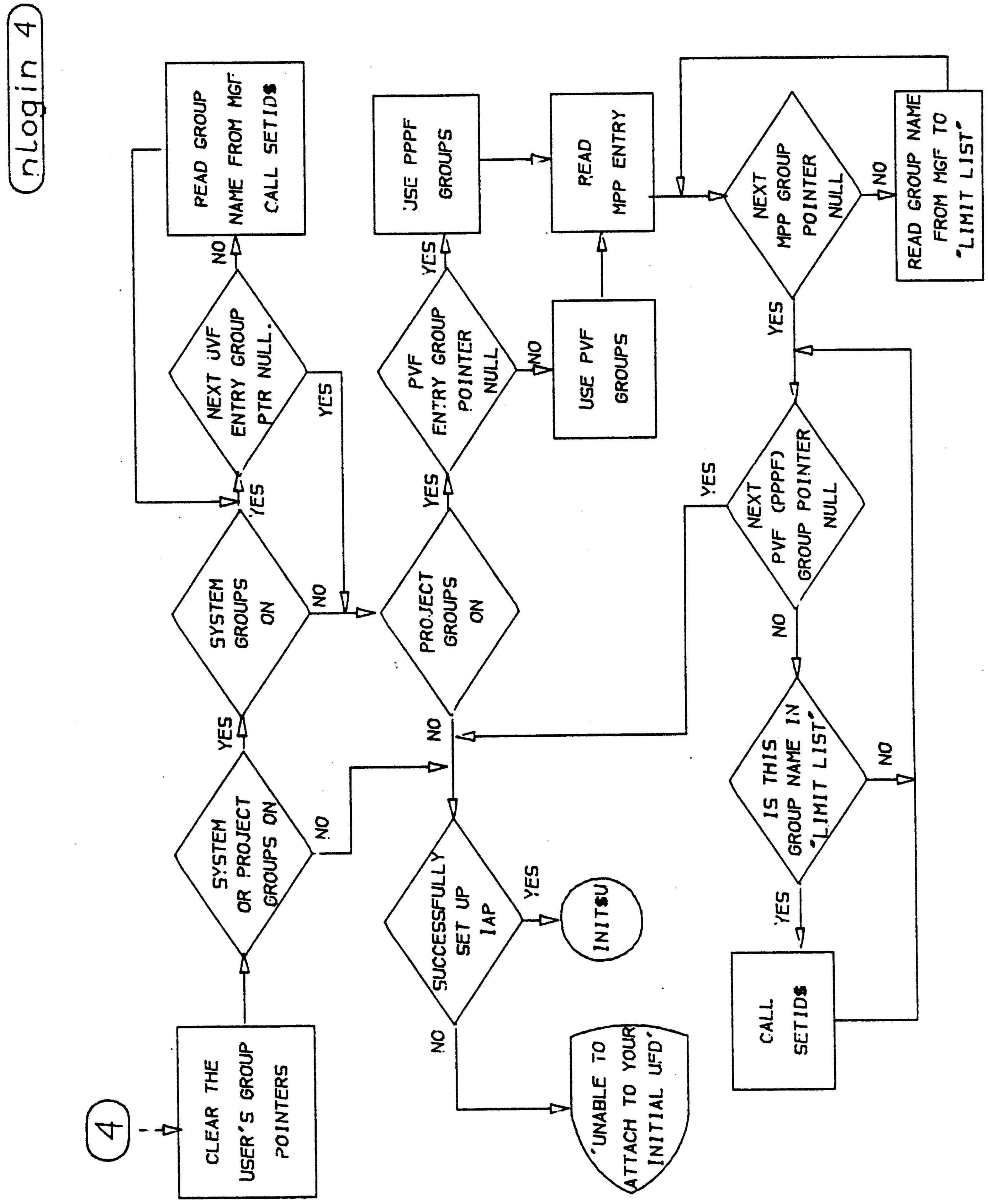
nLogin 1


NLOGIN
MAXUSR = 0
YES
'SYSTEM NOT ADMITTING USERS AT THIS TIME'
NO
ID SUPPLIED
NO
PROMPT, READ, UPCASE ID
YES
ATTACH TO SAD
OPEN UVF
OPEN MPF
READ UVF HEADER
PASSWORD SUPPLIED BUT FORCE PASSWORD CONFIGURED
YES
'PASSWORDS MAY NOT BE SPECIFIED IN LOGIN COMMAND'
NO
ACL SAD
YES
OPEN MGF
NO
2

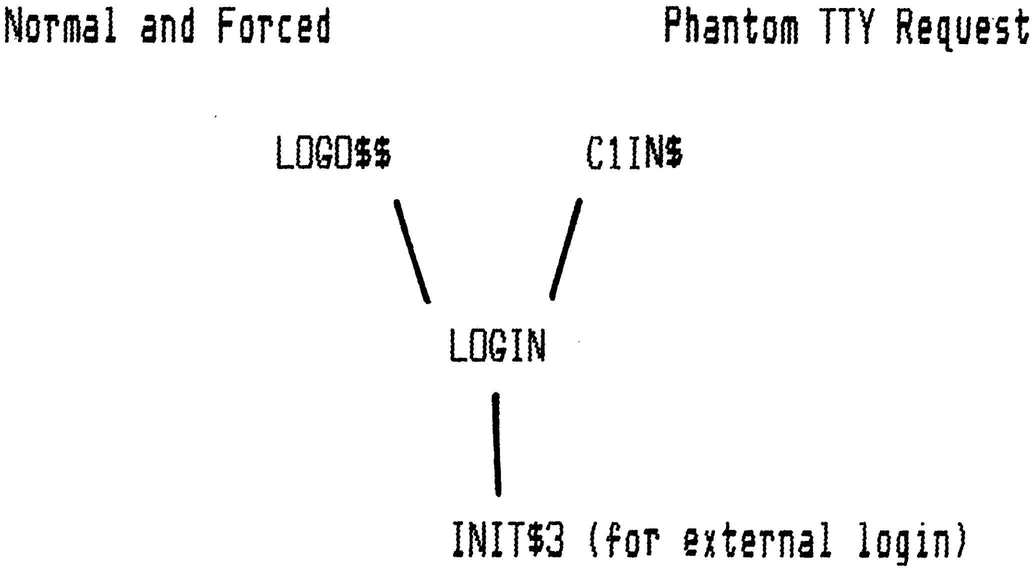

nLogin 2
2
ID EXISTS IN UVF
YES
NO
PASSWORD SUPPLIED
YES
NO
PROMPT, READ, UPCASE PASSWORD
'INVALID USER ID OR PASSWORD'
NULL PASSWORD SUPPLIED
YES
NO
NULL PASSWORD ALLOWED AND NOT FORCING PASSWORDS
YES
NO
NULL ENCRYPTED PASSWORD MATCHES UVF ONE
YES
NO
PROMPT, READ, UPCASE PASSWORD
ENCRYPTED PASSWORD MATCHES UVF ONE
YES
NO
WRITE NEW DATE/TIME OF LAST LOGIN TO UVF
3

nLogin 3


3
PROJECT SUPPLIED
NO
YES
DEFAULT PROJECT EXISTS IN UVF ENTRY
NO
YES
READ DEFAULT PROJECT NAME FROM MPF
PROMPT, READ, UPCASE PROJECT
ATTACH HOME CLOSE PVF CLEAR PROJECT
ATTEMPT TO ATTACH TO PROJECT SUBUFD OPEN PVF READ PVF HEADER
USER ID EXISTS IN PROJECT
NO
YES
USING DEFAULT PROJECT
YES
NO
'INVALID PROJECT ID PLEASE TRY AGAIN
MAXUSR LIMIT EXCEEDED
YES
NO
TOO MANY USERS PLEASE TRY AGAIN IN A FEW MINUTES
SET UPCOM.LOGNAME UPCOM.PROJID OPEN PDF, PPPF, MPP READ PPPF FILE ENTRY SET UTYPE
4


nLogin 4
4
CLEAR THE USER'S GROUP POINTERS
SYSTEM OR PROJECT GROUPS ON
YES
NO
SYSTEM GROUPS ON
YES
NO
NEXT UVF ENTRY GROUP PTR NULL.
NO
YES
READ GROUP NAME FROM MGF CALL SETID$
PROJECT GROUPS ON
YES
NO
PVF ENTRY GROUP POINTER NULL
YES
NO
USE PPPF GROUPS
USE PVF GROUPS
READ MPP ENTRY
NEXT MPP GROUP POINTER NULL
NO
YES
READ GROUP NAME FROM MGF TO "LIMIT LIST"
NEXT PVF (PPPF) GROUP POINTER NULL
YES
NO
IS THIS GROUP NAME IN "LIMIT LIST"
YES
NO
CALL SETID$
SUCCESSFULLY SET UP IAP
YES
NO
INIT$U
'UNABLE TO ATTACH TO YOUR INITIAL UFD'

OLD LOGOUT MECHANISM



Normal and Forced

Phantom TTY Request

LOGO$$

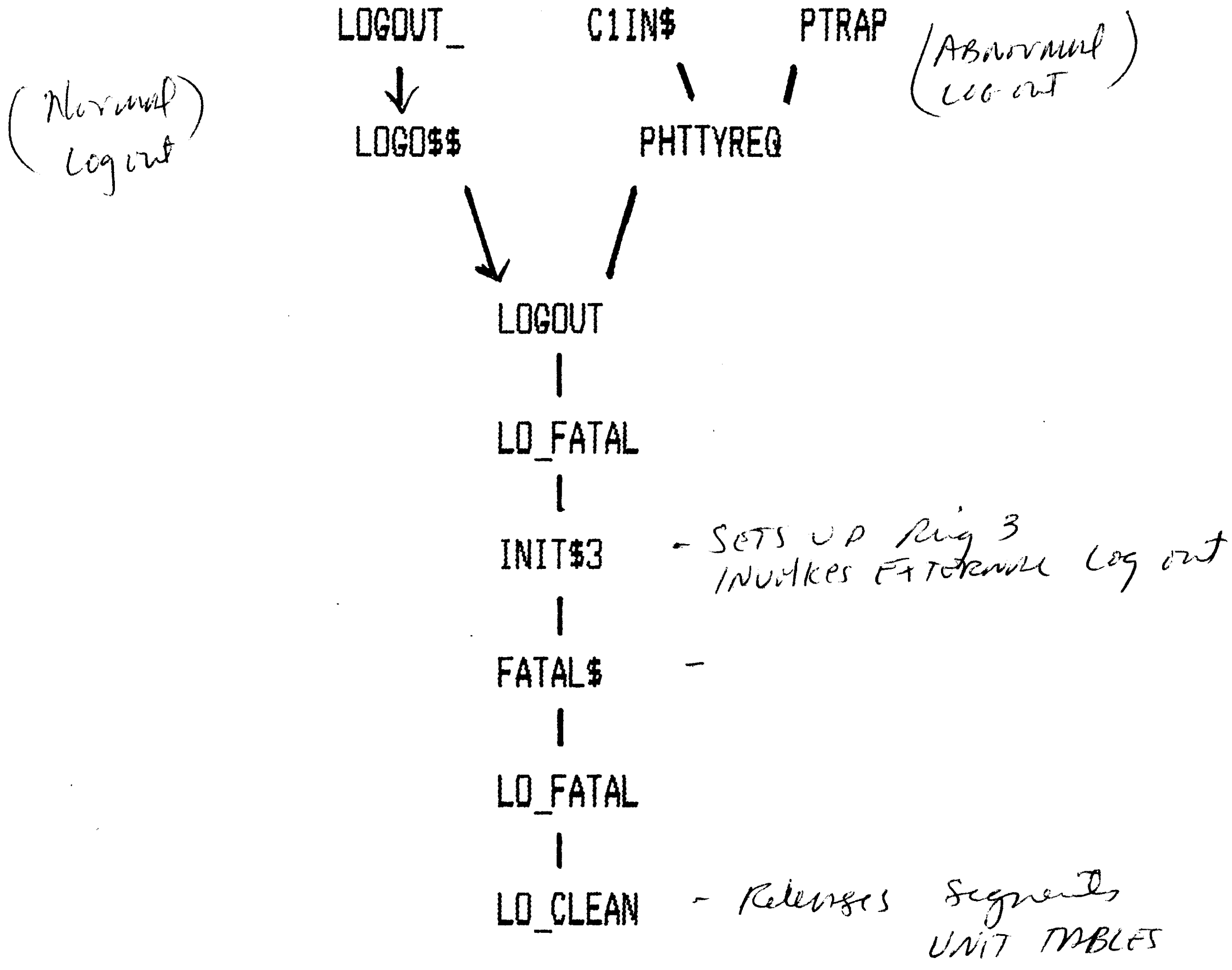
C1IN$

LOGIN

INIT$3 (for external login)

NOTE: Login over login handled internally within LOGIN (tricky!)

NEW LOGOUT MECHANISM



(Normal Logout)

LOGOUT_ → LOGO$$

C1IN$ PTRAP → PHTTYREG

(Abnormal Logout)

LOGOUT

LO_FATAL

INIT$3 - Sets up Ring 3 Invokes External Log out

FATAL$ -

LO_FATAL

LO_CLEAN - Releases Segments UNIT TABLES

NEW LOGOUT MECHANISM

LOGOUT_
- ring 3 command moved from DOSSUB
- handles normal and forced logout commands
- parses command line
- calls LOGO$$

LOGO$$
- for forced logout
- validates and calls SETABT
- for normal logout calls LOGOUT

LOGOUT
- if logged out return
- don't allow phantom login over login
- force tty output on, comi off
- reset tty characteristics
- pass any outstanding messages to user
- build logout message
- if phantom put message in l.o.n. queue
  otherwise close l.o.n. queue
- type message at user/console
- call LO_FATAL

PHTTYREQ
(PHTTYR)
- send message to console
- call LOGOUT

NEW LOGOUT MECHANISM

LO_FATAL

- make any$ handler
- close file units
- unattach home, current, origin (LO_NATCH)
- free semaphores
- free dptx devices (ODUNDO)
- free rje devices (RJUNDO)
- free assigned devices
- if netman call NETDWN
- if FAM I do special cleanup

Normal, Forced, Phantom Abort

- 'wait...' for remote users
- return all segs
- allocate segs 6002, 4000
- restore external login (EXTLOG)
- inhibit r3 quits
- call INIT$3 (never returns)

Login over Login

- close como
- if using FAM I tell FAM I
- disconnect from network

FATAL$ LO Key

- call LO_CLEAN
- disconnect from network (XCLRA)

Action determined by key passed in as argument.

NEW LOGOUT MECHANISM

FATAL$

- unwind r0 stack
- rebuild our frame
- unlock all r0 locks (UNLKF$)
- r3 quits off
- if e$logo key call LO_FATAL - doesn't return
- if logged out call r0 LISTEN
- if phant_err key call PHTTYREQ
  otherwise call INIT$3 with error key

LO_CLEAN

- return segs (not dynamic ones for slave)
- free attach points (LO_NATCH)
- switch comi and como off
- if using FAM I tell FAM I
- send logout notification if message is built (LON$S)
- close l.o.n. queue (LON$C)
- close CPS down (CPSRG, CPSCA)
- clear user_id, project
- set utype = -utype
- clear groups
- reset per user parameters (LOG_INIT)
- if remote user clear v.c. (X$LOGO)
- deallocate unit table (not slave)
- clear pending quits
- drop dtr if configured (DRPDTR)

'LOGOUT$' CONDITION - grace period

PABORT - Takes a process abort SWIALM.
If SWITYP = '40 (forced logout) then call LOGABT

LOGABT (cases)
1) force logout, and process is remote
2) force logout (either by operator or amlc disconnect)
3) cpu time limit exceeded
4) inactivity time limit exceeded
5) login time limit exceeded
6) in grace period, abort <u>not</u> login time limit exceeded
7) in grace period, abort is login time limit exceeded

When (1) tell network to send logout message to remote end
When (6) ignore abort
When (7) log the process out immediately
Otherwise

```
    inhibit process aborts
    set login time limit to (grace_period)
    clear pcb.abort_flags, pudcom.absave login time limit abort flag
    call SETSWI(LOGINT) Set Software interrupt
    enable process aborts
    call SW$ABT directly to process LOGINT
```

SW$ABT - signal the condition 'LOGOUT$'

'LOGOUT$' CONDITION - grace period

(actual condition that logs you out)

The user could 'make' an on-unit for 'LOGOUT$' to ensure a clean exit before the actual logout.

Otherwise DF_UNIT_ will simply print the error message call LOGOU$.

```
    when (login_limit)
         call ioa$ ('login time limit exceeded.
    when (cpu_limit)
         call ioa$ ('cpu time limit exceeded.
    when (timeout)
         call ioa$ ('maximum inactive time limit exceeded.
    otherwise
         call ioa$ ('forced logout.
    end;
    call logou$;
```

```
LOGOU$ (LOGOUT)
    call internal routine LOGMSG to
         print message to system console and user terminal.
    If a phantom, queue Logout Notification (LON) message to spawner.
```

LOGOUT NOTIFICATION

- Mechanism to pass message to spawner when phantom logs out.

- Simple IPC mechanism.

- At login LON queue opened for user.

- When phantom logs out - message added to spawner's queue. Spawner takes SoftWare Interrupt abort (type LONINT).

- If LON not inhibited, then 'PH_LOGO$' is signalled.

- Default on-unit prints LON message.

- At logout LON queue is closed.

- Lon database in segment 35 manipulated by area management package.

COMMAND -- enable/disable immediate notification

```
  LOgout_Notification  -ON ¦ -OFF
```

LOGOUT NOTIFICATION

DATABASE

- 8192 words reserved in segment 35.

- LON$SEM - semaphore used to single thread all access to database.

- Database consists of receiver blocks and message blocks.

- LON$STA points to start of receiver block chain. (Null if nobody has queue open.)

- Receiver block chain is doubly linked list.

- Message blocks are doubly linked lists starting at a receiver block.

LOGOUT NOTIFICATION - Data Structures

```
dcl lon$adr pointer ext;               /* address first word of lon$
                                        area*/

dcl 1 lon$_rcvr based,                 /* receiver node structure*/
        2 length fixed bin(15),        /* length of header*/
        2 id,                          /* unique id*/
            3 uno char(6),             /* unique number*/
            3 usrno fixed bin(15),     /* user no*/
        2 nextrcvr pointer,            /* next receiver*/
        2 lastrcvr pointer,            /* last receiver*/
        2 cnt fixed bin(15),           /* number of messages associated
                                          with this rcvr*/
        2 size fixed bin(15),          /* total size of messages for
                                          this rcvr*/
        2 notify bit(1),               /* notify flag
                                          1-notify
                                          0-don't notify*/
        2 headmsg pointer;             /* head of message list*/
```

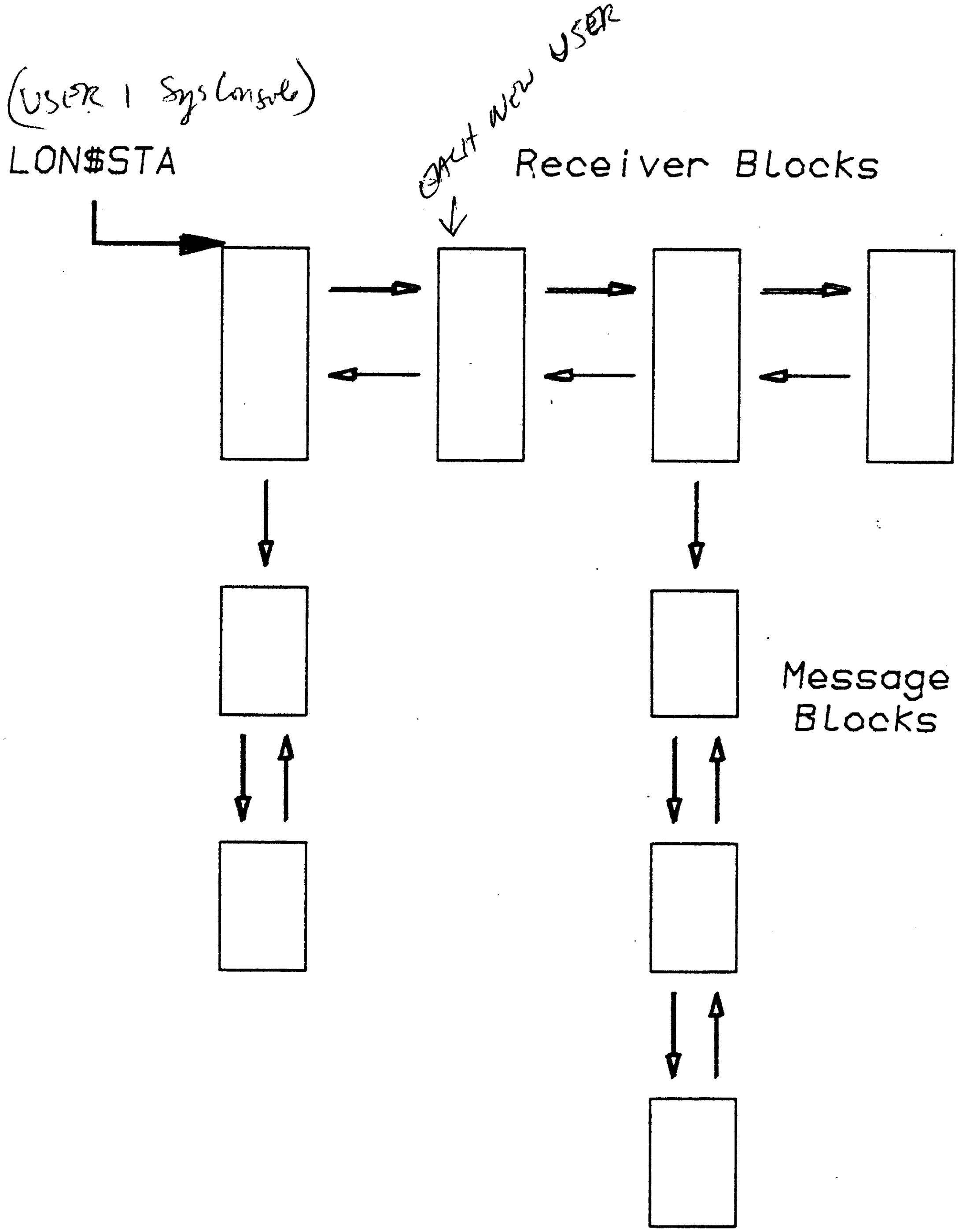
LOGOUT NOTIFICATION - Data Structures

```
dcl 1 lon$_msg based,                /* message node*/
        2 length fixed bin(15),      /* length of this message
                                        including header info.*/
        2 next pointer,              /* pointer to next message*/
        2 last pointer,              /* pointer to last message*/
        2 info(1) fixed bin(15);     /* message information*/
```

```
log_msg (1) = pudcom.cusr
        (2) = time in mins since midnight
        (3) = connect time mins
        (4) = cpu secs
        (5) = i/o secs
        (6) = normal/abnormal logout flag
```

LOGOUT NOTIFICATION

DATABASE



GETTING INTO THE COMMAND LOOP

It is not apparent how one gets into the command loop initially, thi writeup is an attempt to trace the path of the user process from cold sta to login and then into the basic command loop.

All PCBs for the system processes including user 1 are initially defined in KS>SEG4.PMA. In addition a PCB is defined for user 2, this PCB is called U02PCB, it will be used as a template for building all other us PCBs needed at cold start time. Initially the stored PB value for U02PCB (and hence all others) is set to a value called CLDPB which is a pointer to location CLDPB in the module KS>FATAL$.PMA. In addition, the pointer t the WAIT list that the PCB is waiting on is initially set to point to a semaphore called CLDSEM (KS>SEG4>PMA). At cold start time KS>AINIT.FTN makes as many copies of U02PCB as needed according to the number of users that are configured by the CONFIG file directives, each one of these PCBs for terminal users having it's initial stored PB pointing to CLDPB and it WAIT list pointer pointing to CLDSEM.

When the SETIME MAXUSR 19.2 command is issued at the system console the CLDSEM semaphore is NOTIFYed for the number of terminal users and each user is sent the 'LOGIN PLEASE' message. When each terminal user process is notified it moves to the READY list to await execution, when it gets it's turn it starts to run from location CLDPB. The instruction at CLDPB is a procedure call to FATAL$ with an argument value of zero.

FATAL$ initializes stack pointers via a call to UNWIND (KS>TMAIN.PMA), quits are disabled for Ring 3 and enabled for Rings 0 and 1, and finally a call to LISTEN (KS>LISTEN.PLP) is made passing it the current user number and an argument specifying whether that user is a phantom (bit 1 set) or terminal user (all zeros).

LISTEN checks to see if the user is a phantom or a terminal user, if it's a phantom LISTEN calls UNLOAD (KS>TMAIN.PMA)

If the user is a terminal user the 'OK' prompt is printed at the user terminal and CL$GET (KS>CL$GET.PLP) is called to read a command from the terminal. CL$GET calls C1IN$ (KS>C1IN$.PLP) to read the characters in.

C1IN$ uses a function called TF$ANY in KS>TFLIO$.PMA to see if there are any characters in the input buffer, if not it does a WAIT on the BUFSEM (   ) appropriate to that user. C1IN$ also checks for and handle special characters such as ERASE and KILL and the carriage return character. It just keeps reading in characters (moving back and forth between the READY list and BUFSEM until a carriage return character is detected at which point it calls SCHED (KS>SCHED.PMA) to get that user put on the HIPRIQ.

When the user runs, ClIN$ returns to CL$GET which returns to LISTEN, LISTEN calls DOSSUB (KS>DOSSUB.~~PMA~~ PLP) and passes it the command line which contains the LOGIN command. DOSSUB processes the LOGIN command and calls LOGIN (KS>LOGIN.~~PMA~~ .PLP). LOGO$CP

LOGIN; attaches to the login UFD, prints the login messages on the system console and at the user terminal, calls RTNSEG to return all segmants except the Ring 3 stack, calls GETSEG to allocate the Ring 3 stack ('6002) and Static Mode ('4000) segments, disables Ring 3 Quits, attaches to CMDNC0 and executes the external LOGIN program if there is one and returns to the login UFD in either case. Finally LOGIN calls INIT$3 to get the user from the Ring 0 to the Ring 3 environment.

INIT$3 has two phases, a Ring 0 phase and a Ring 3 phase. The Ring 0 phase initializes the users Ring 3 stack and command line data (CLDATA) structures, makes itself into a condition frame and dummies the return PB ring bits to be Ring 3, then calls CRAWL _ (R3S>CRAWL _ .PLP), passing as arguments INFIM _ , pointers to the condition frame just built and a zero to indicate the depth of the concealed stack???

CRAWL _ ; forces Quits to be inhibited, calls MKONU$ to make an on-unit for ANY$, selects a stack segment for the target ring (Ring3), copies the condition frame from Ring 0 (which would be for INIT$3), to the target ring stack, and eventually returns which passes control to the routine that we passed as an argument to CRAWL _ , which is INFIM _ .

INFIM _ (R3S>INFIM _ .PMA) is the fault interceptor module for getting to INIT$3 again, this time in Ring 3. It adjusts a few pointers, enables Quits, and calls INIT$3.

INIT$3 is now entered to perform it's Ring 3 phase operation, it will do nothing more than return to INFIM _ for the simple case of a terminal user logging in.

INFIM _ finally calls the Ring 3 listener LISTN _ (R3S>LISTEN _ .PLP) and sit in a loop calling it forever, so that when the listener returns it is just called again (and again and again).

Section 13 - Command Processor

EXTENDED FEATURES

- Command processor enhanced to support following *extended* features:

  simple iteration - Delete (File1 File2), Acts inside parenthesis. 1 command does many
  wildcard expansion -
  treewalking
  name generation
  special reserved arguments - TREEWALKING Commands

- All above are processed by c.p. itself.

- Enabling of individual features may be selected in various ways:

  CPL - defined to have c.p. do simple iteration only

  Static Programs - all features enabled unless special names:
  NW$ - no wildcard or equalname
  NX$ - only simple iteration

  EPF - enabled features specified at BIND time and stored in file

  Internal Commands - enabled features specified in internal command table

EXTENDED FEATURES

CP_ITER

- main routine which processes extended features
- makes three passes over command line to verify syntax, expand iteration, process options

Pass I

- parses command line into 2 level tree
- each node represents a token
- 2nd level for simple iteration tokens

Pass II

- repeated while iteration in progress
- convert tree into simple threaded list
- expand dot products
- call DCOD_ITR to find type of token (e.g. wildcard, wildtree, control, equalname)

Pass III

- repeated while iteration in progress
- verify only one wildcard/tree per line
- find location of wild tokens
- if wildtree call ITR_WLDT
- if wildcard call ITR_WLDC
- if no wilds call LIGASE
- free all temporary storage

EXTENDED FEATURES

ITR_WLDT
- expands wild trees
- uses control args if supplied
- calls ITR_WLDC if wilcards, or 'executer' to execute each match
- recurses when required

ITR_WLDC
- expands wild cards
- uses control args if supplied
- asks user for verification if reqd
- calls 'executer' to execute each match

EQUAL$P
- special routine for c.p.
- splits pathnames into dir and entry
- calls EQUAL$ to match names

EQUAL$
- parse generation pattern components
- process 'commands' in components
- build generated name by concatenation

EXTENDED FEATURES

LIGASE (internal to CP_ITER)

- follows assembled node list concatenating tokens to form command line
- calls EQUAL$P to process name generation
- call 'executer' routine to execute line

SM_EXECUTER (internal to STD$CP)

- executes static mode command
- calls INVKSM_

CPL_EXECUTER (internal to STD$CP)

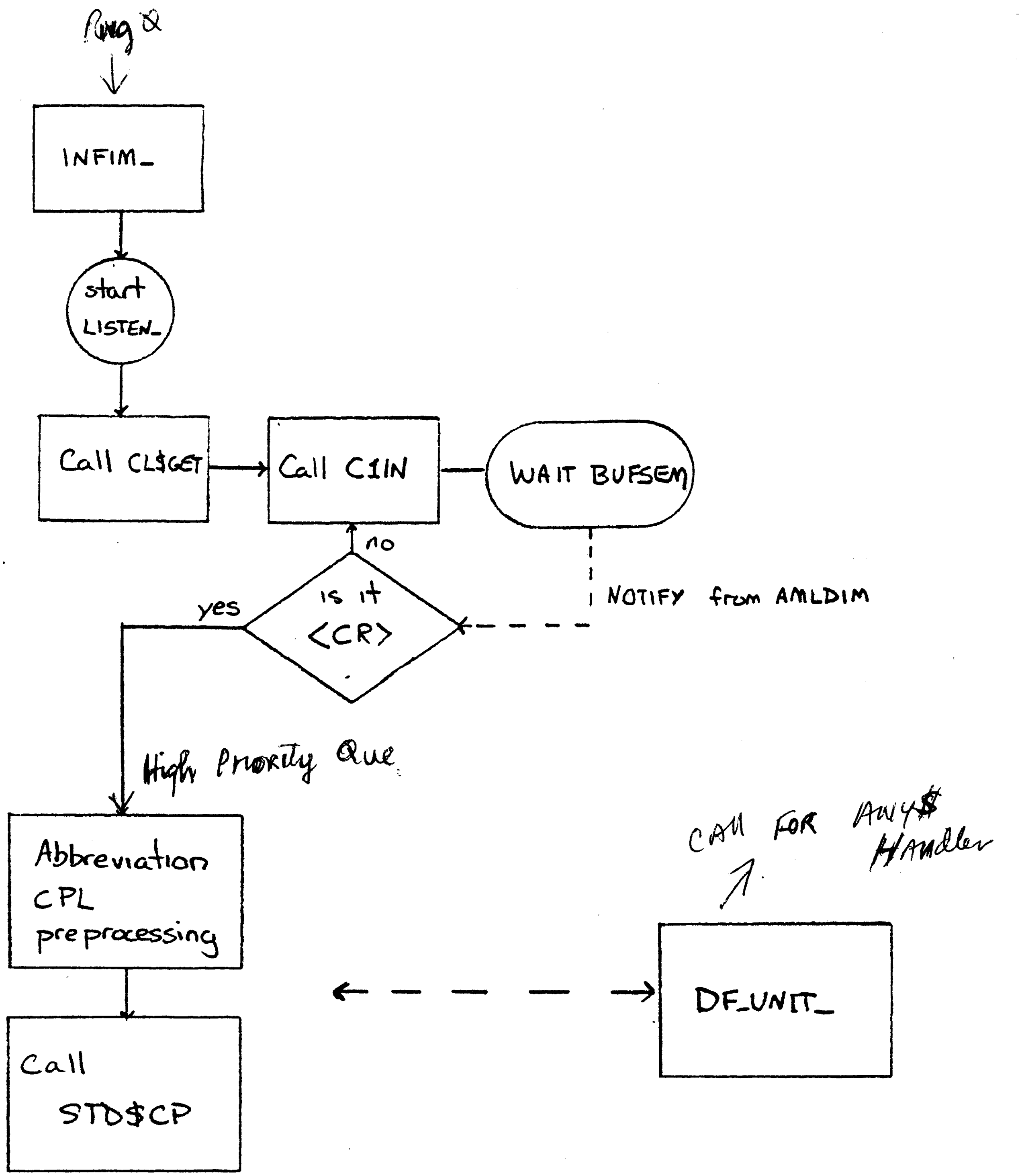
- executes CPL command
- calls ICPL_

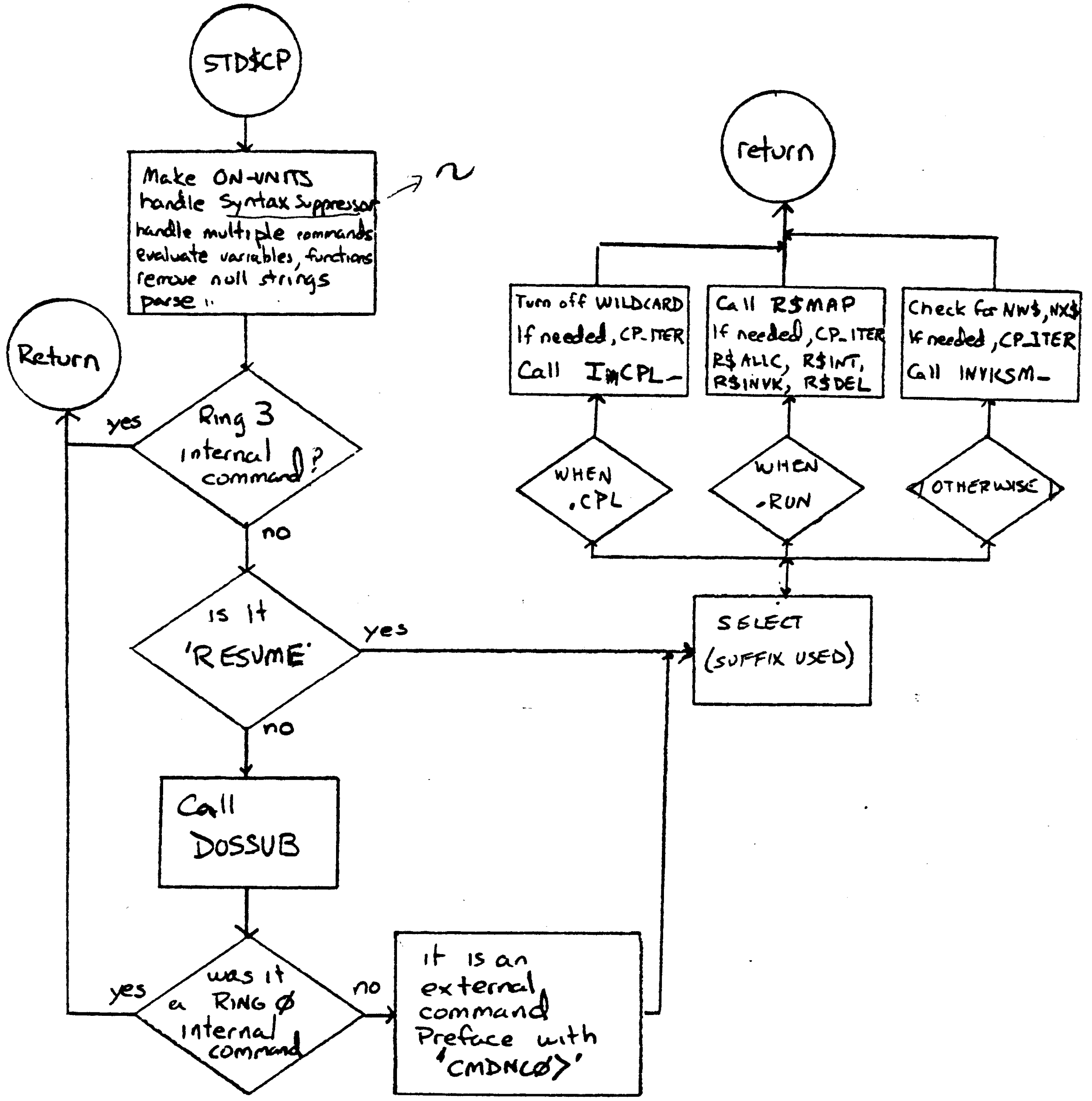
INTERNAL_EXECUTER

- executes an internal command
- calls appropriate routine directly

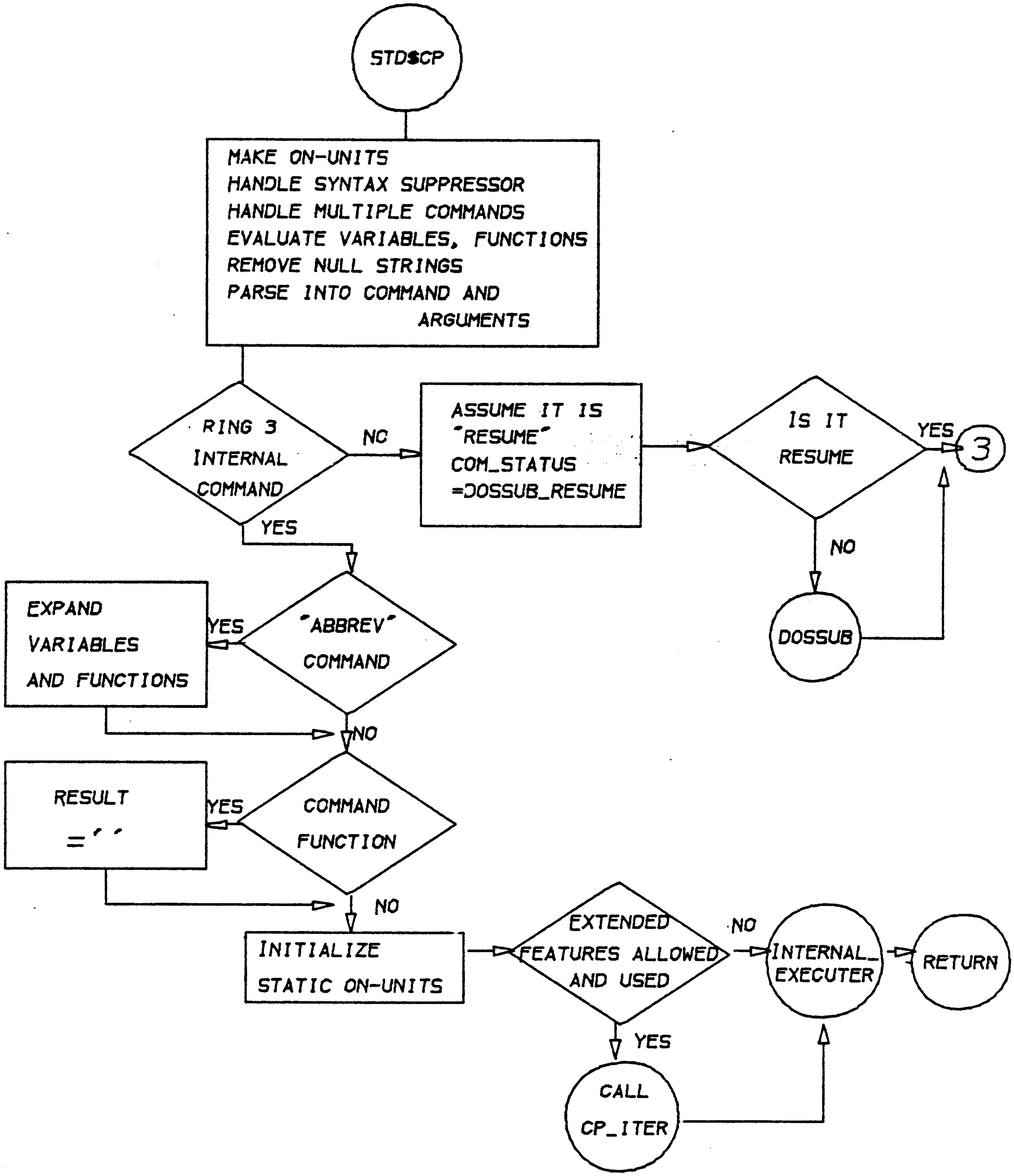
RUN_EXECUTER (internal to STD$CP)

- executes an EPF
- calls R$ALLC to allocate linkage
  R$INIT to initialize linkage
  R$INVK to execute EPF

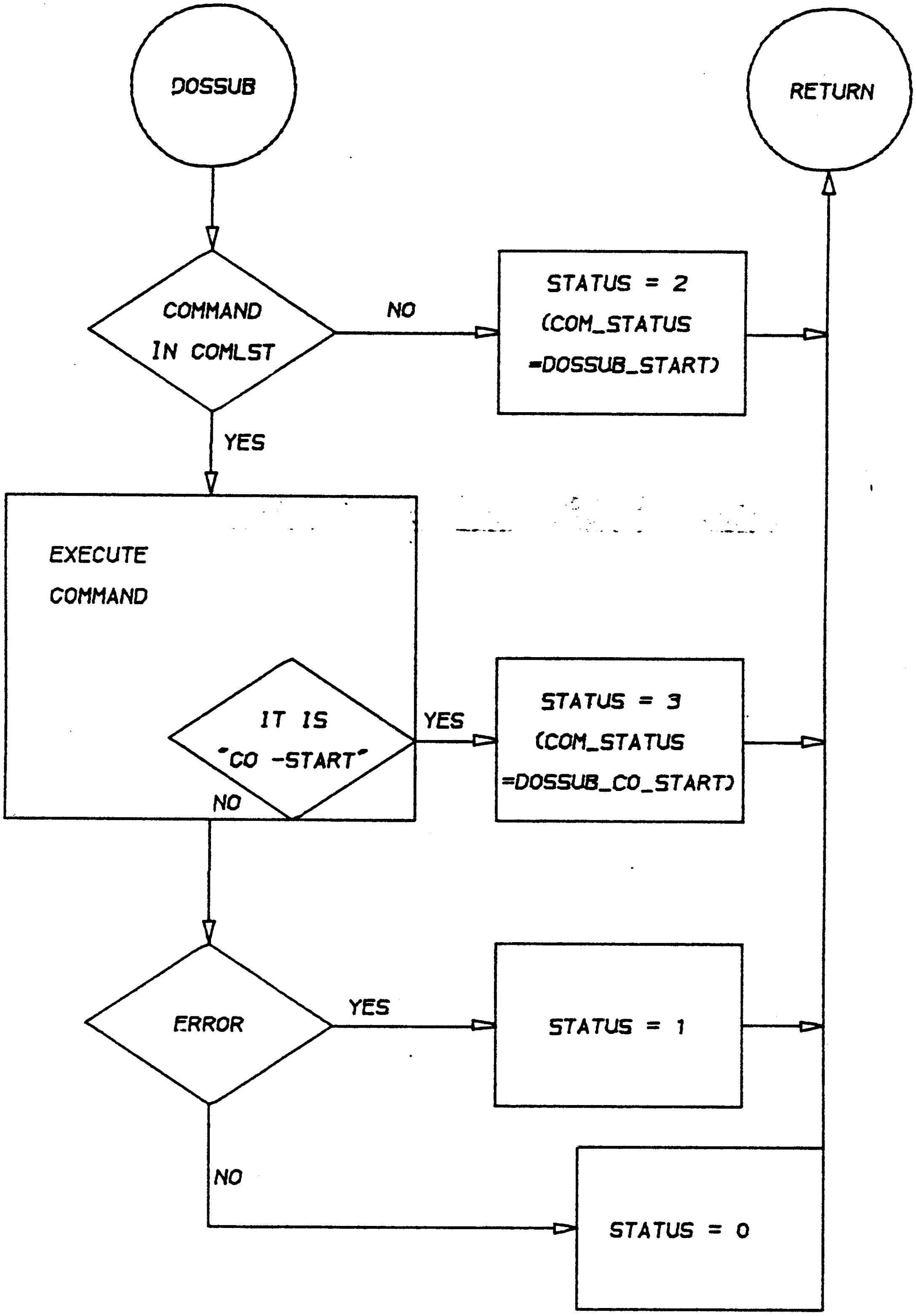

Ring 0
INFIM_
start LISTEN_
Call CL$GET
Call C1IN
WAIT BUFSEM
no
is it <CR>
yes
NOTIFY from AMLDIM
High Priority Que
Abbreviation CPL preprocessing
CALL FOR ANY$ Handler
DF_UNIT_
Call STD$CP


STD$CP
Make ON-UNITS
handle Syntax Suppressor
handle multiple commands
evaluate variables, functions
remove null strings
parse
~
Return
yes
Ring 3 internal command?
no
is it 'RESUME'
yes
no
Call DOSSUB
yes
was it a RING Ø internal command
no
it is an external command Preface with 'CMDNCØ>'
SELECT (SUFFIX USED)
WHEN .CPL
WHEN .RUN
OTHERWISE
Turn off WILDCARD
If needed, CP_ITER
Call I$CPL_
Call R$MAP
If needed, CP_ITER
R$ALLC, R$INT,
R$INVK, R$DEL
Check for NW$, NX$
If needed, CP_ITER
Call INVKSM_
return

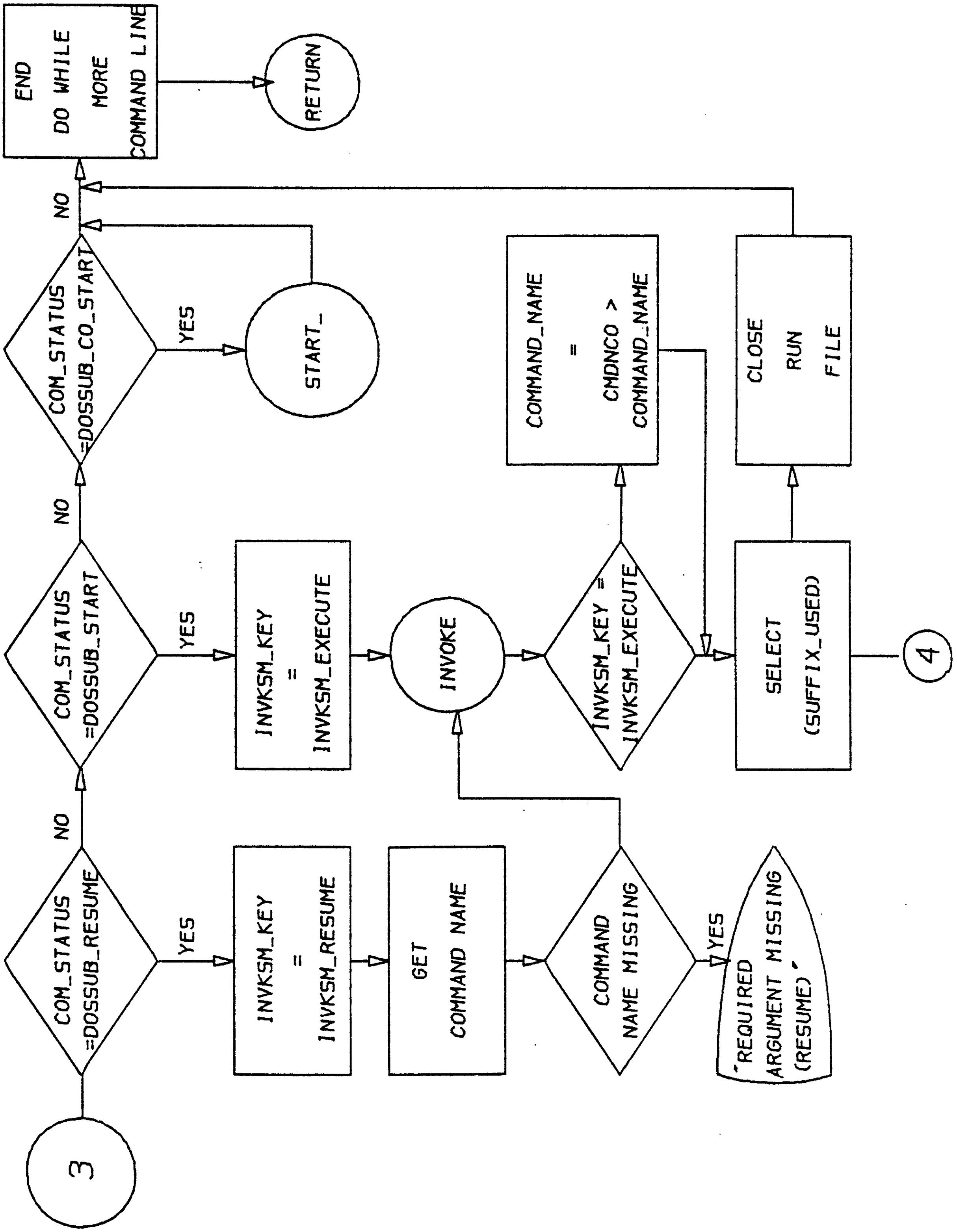
COMMAND 1


STD$CP
MAKE ON-UNITS
HANDLE SYNTAX SUPPRESSOR
HANDLE MULTIPLE COMMANDS
EVALUATE VARIABLES, FUNCTIONS
REMOVE NULL STRINGS
PARSE INTO COMMAND AND ARGUMENTS
RING 3 INTERNAL COMMAND
NO
YES
ASSUME IT IS "RESUME" COM_STATUS =DOSSUB_RESUME
IS IT RESUME
YES
3
NO
DOSSUB
EXPAND VARIABLES AND FUNCTIONS
YES
"ABBREV" COMMAND
NO
RESULT =''
YES
COMMAND FUNCTION
NO
INITIALIZE STATIC ON-UNITS
EXTENDED FEATURES ALLOWED AND USED
NO
INTERNAL_ EXECUTER
RETURN
YES
CALL CP_ITER

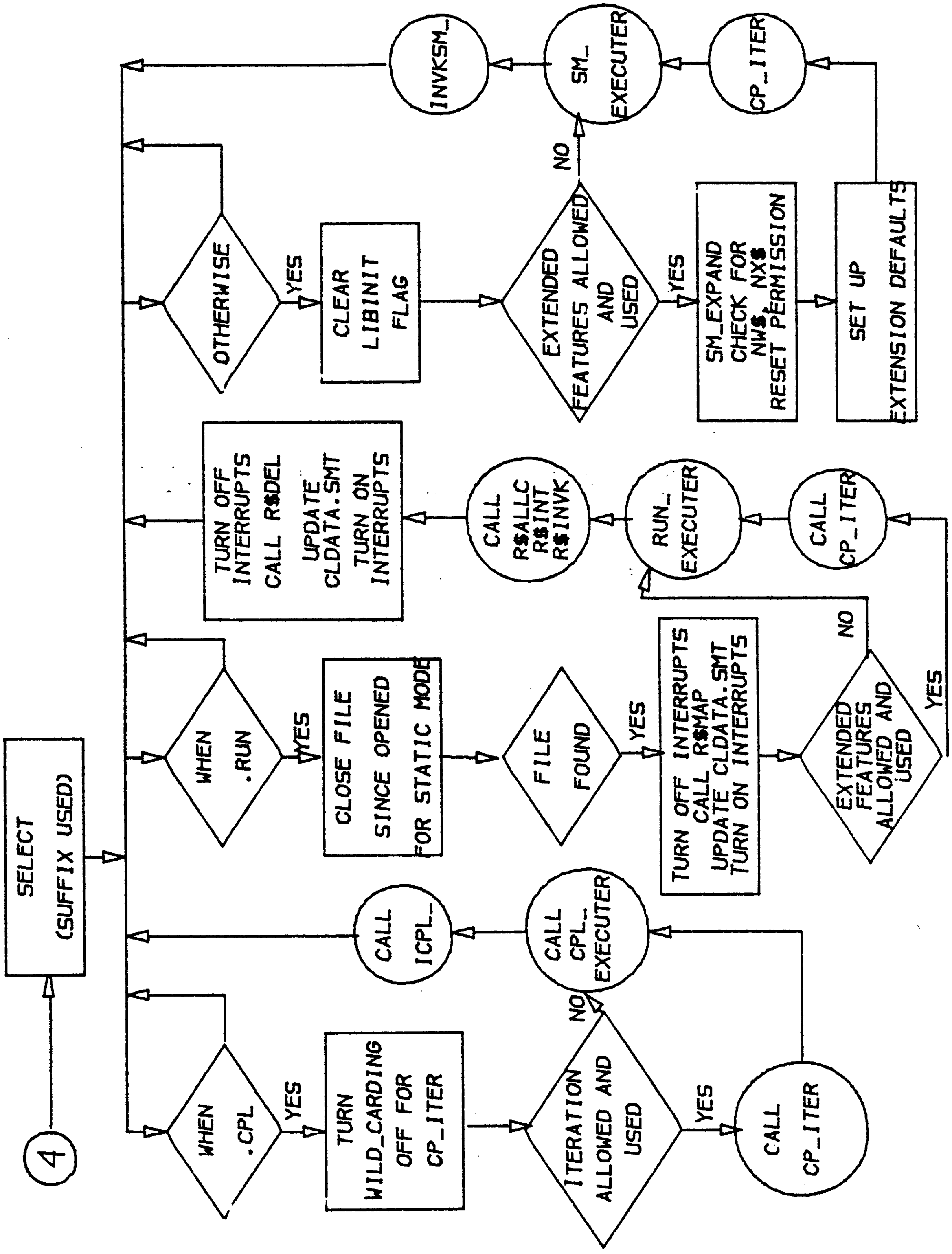
COMMAND 2


DOSSUB
RETURN
COMMAND IN COMLST
NO
STATUS = 2 (COM_STATUS =DOSSUB_START)
YES
EXECUTE COMMAND
IT IS "CO -START"
YES
STATUS = 3 (COM_STATUS =DOSSUB_CO_START)
NO
ERROR
YES
STATUS = 1
NO
STATUS = 0

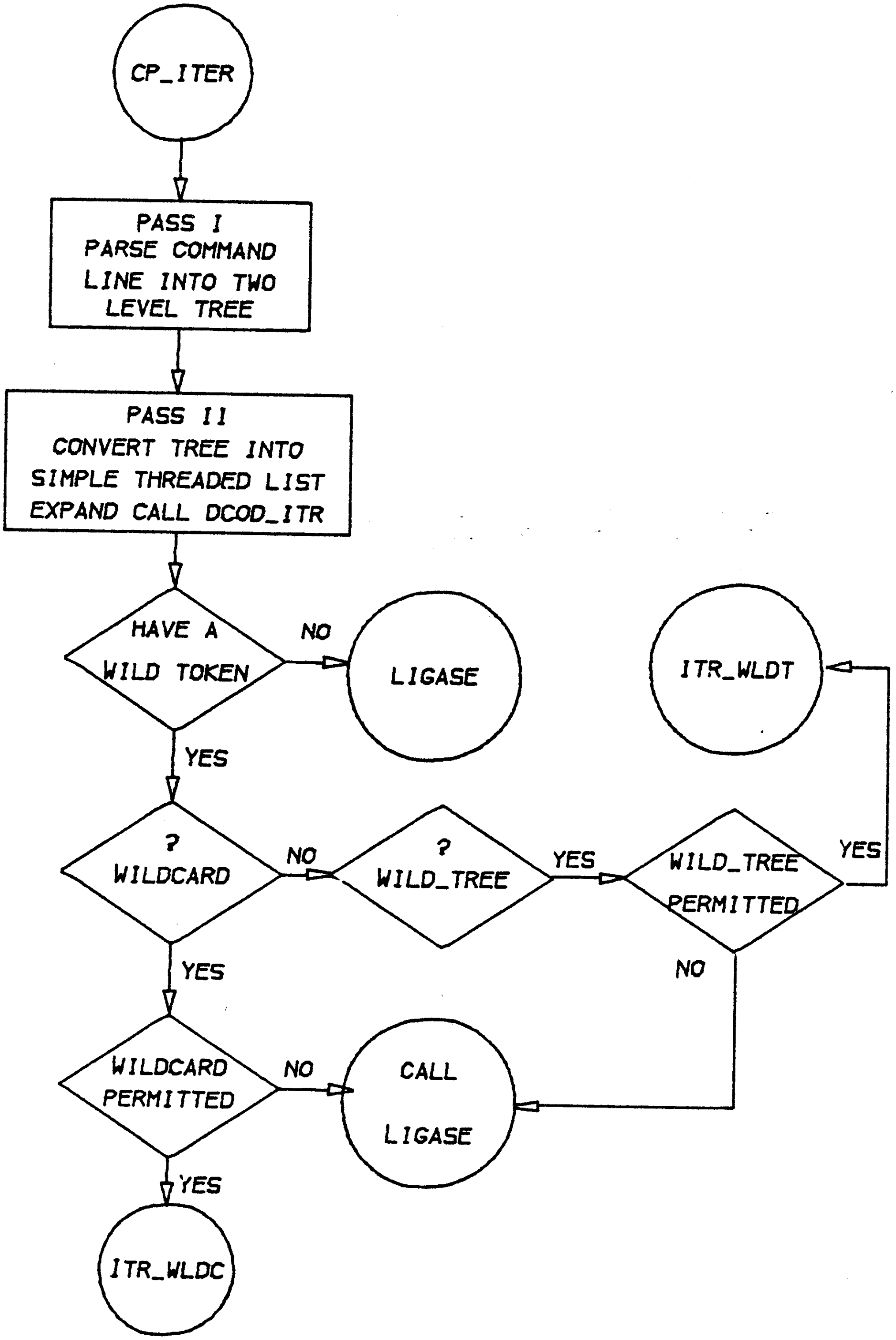
COMMAND 3


3
COM_STATUS =DOSSUB_RESUME
YES
NO
INVKSM_KEY = INVKSM_RESUME
GET COMMAND NAME
COMMAND NAME MISSING
YES
'REQUIRED ARGUMENT MISSING (RESUME)'
COM_STATUS =DOSSUB_START
YES
NO
INVKSM_KEY = INVKSM_EXECUTE
INVOKE
INVKSM_KEY = INVKSM_EXECUTE
COMMAND_NAME = CMDNCO > COMMAND_NAME
SELECT (SUFFIX_USED)
4
CLOSE RUN FILE
COM_STATUS =DOSSUB_CO_START
YES
NO
START_
END DO WHILE MORE COMMAND LINE
RETURN

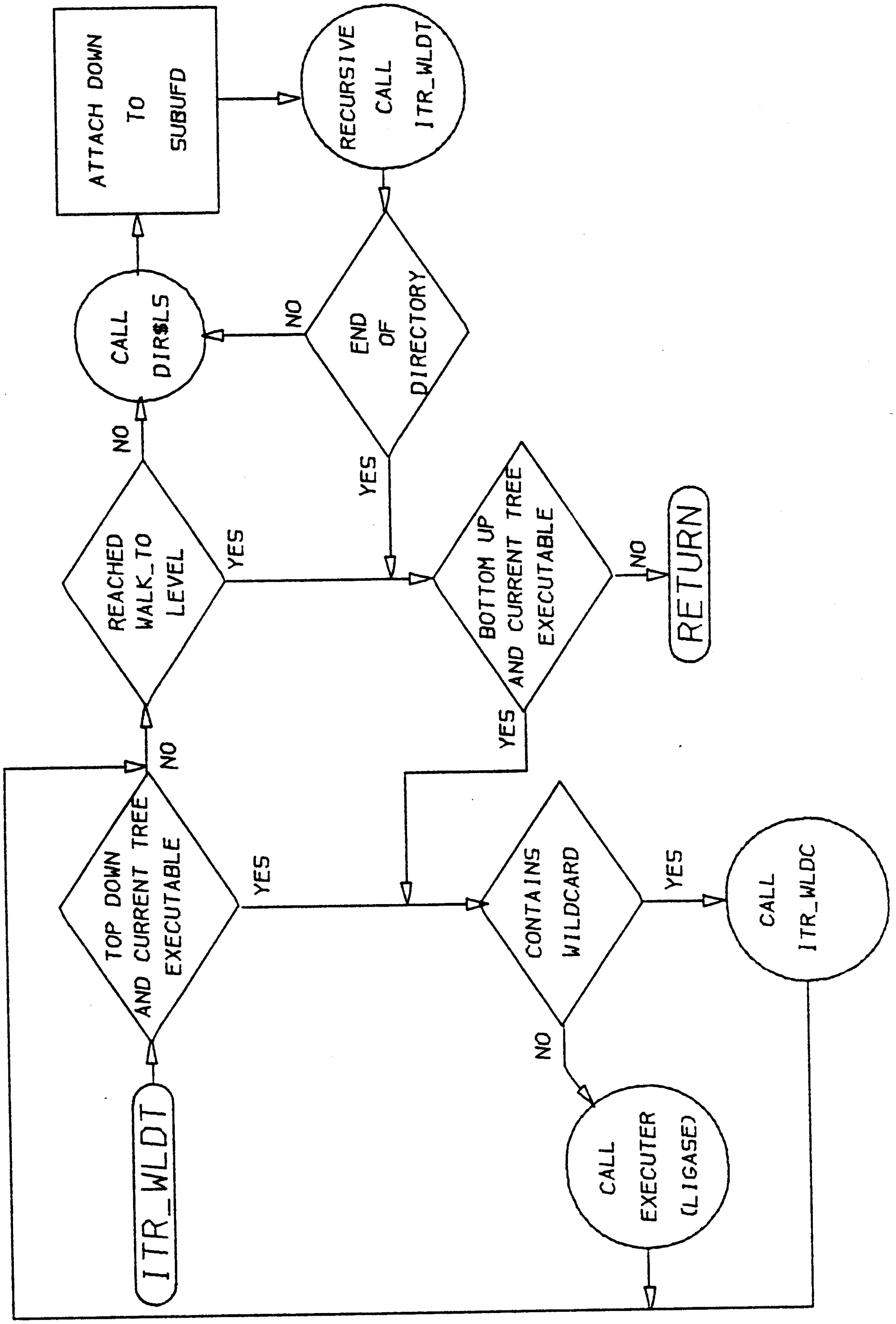
COMMAND 4


4
SELECT (SUFFIX USED)
WHEN .CPL
YES
TURN WILD_CARDING OFF FOR CP_ITER
ITERATION ALLOWED AND USED
YES
CALL CP_ITER
NO
CALL CPL_ EXECUTER
CALL ICPL_
WHEN .RUN
YES
CLOSE FILE SINCE OPENED FOR STATIC MODE
FILE FOUND
YES
TURN OFF INTERRUPTS CALL R$MAP UPDATE CLDATA.SMT TURN ON INTERRUPTS
EXTENDED FEATURES ALLOWED AND USED
YES
NO
CALL CP_ITER
RUN_ EXECUTER
CALL R$ALLC R$INT R$INVK
TURN OFF INTERRUPTS CALL R$DEL UPDATE CLDATA.SMT TURN ON INTERRUPTS
OTHERWISE
YES
CLEAR LIBINIT FLAG
EXTENDED FEATURES ALLOWED AND USED
YES
NO
SM_EXPAND CHECK FOR NW$, NX$ RESET PERMISSION
SET UP EXTENSION DEFAULTS
CP_ITER
SM_ EXECUTER
INVKSM_

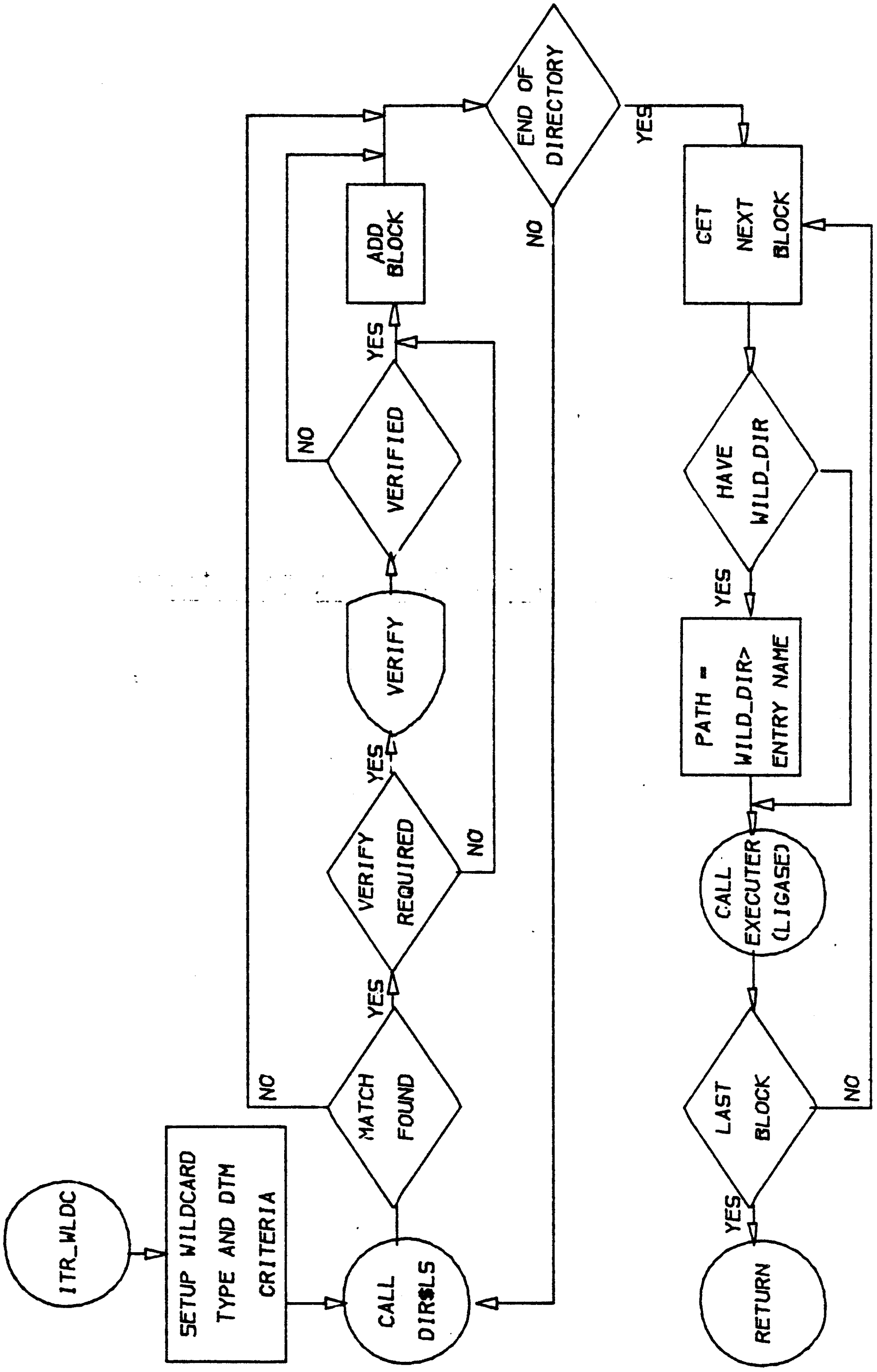
COMMAND 5


CP_ITER
PASS I
PARSE COMMAND
LINE INTO TWO
LEVEL TREE
PASS II
CONVERT TREE INTO
SIMPLE THREADED LIST
EXPAND CALL DCOD_ITR
HAVE A
WILD TOKEN
NO
LIGASE
ITR_WLDT
YES
?
WILDCARD
NO
?
WILD_TREE
YES
WILD_TREE
PERMITTED
YES
YES
NO
WILDCARD
PERMITTED
NO
CALL
LIGASE
YES
ITR_WLDC

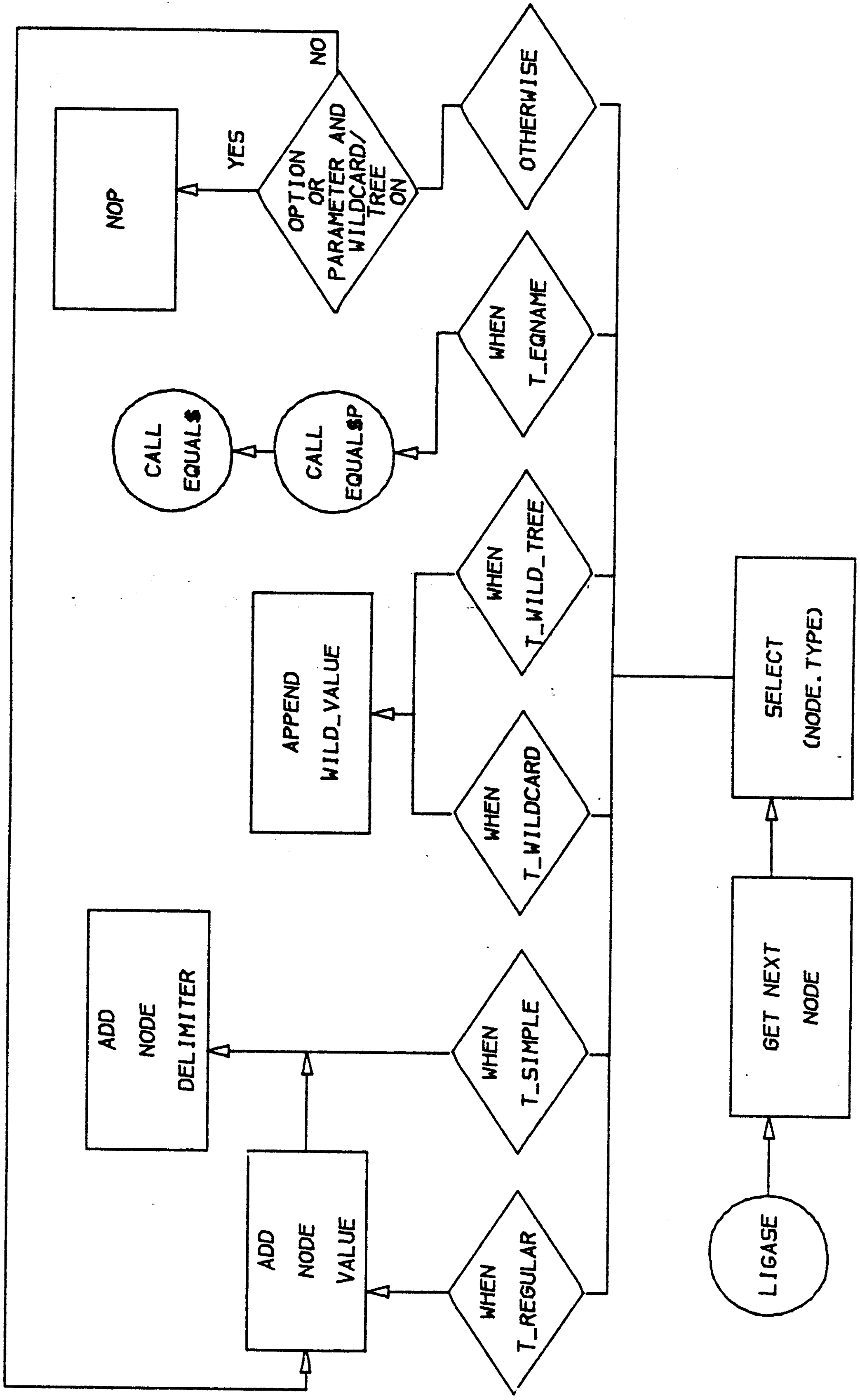
COMMAND 6


ITR_WLDT
TOP DOWN AND CURRENT TREE EXECUTABLE
YES
NO
REACHED WALK_TO LEVEL
YES
NO
CALL DIR$LS
ATTACH DOWN TO SUBUFD
RECURSIVE CALL ITR_WLDT
END OF DIRECTORY
NO
YES
BOTTOM UP AND CURRENT TREE EXECUTABLE
NO
RETURN
YES
CONTAINS WILDCARD
YES
CALL ITR_WLDC
NO
CALL EXECUTER (LIGASE)

COMMAND 7


ITR_WLDC
SETUP WILDCARD TYPE AND DTM CRITERIA
CALL DIR$LS
MATCH FOUND
NO
YES
VERIFY REQUIRED
NO
YES
VERIFY
VERIFIED
NO
YES
ADD BLOCK
END OF DIRECTORY
NO
YES
GET NEXT BLOCK
HAVE WILD_DIR
YES
PATH = WILD_DIR> ENTRY NAME
CALL EXECUTER (LIGASE)
LAST BLOCK
NO
YES
RETURN

COMMAND 8


LIGASE
GET NEXT NODE
SELECT (NODE.TYPE)
WHEN T_REGULAR
WHEN T_SIMPLE
WHEN T_WILDCARD
WHEN T_WILD_TREE
WHEN T_EQNAME
OTHERWISE
ADD NODE VALUE
ADD NODE DELIMITER
APPEND WILD_VALUE
CALL EQUAL$P
CALL EQUAL$
OPTION OR PARAMETER AND WILDCARD/ TREE ON
YES
NO
NOP

Section 14 - Static On-Units

STATIC ON-UNITS

- Static On-Units (SOU) are similar to dynamic on-units.
  Handle asynchronous conditions regardless of the stack state.

- SOUs are not condition name specific.
  All SOUs are invoked for all conditions.
  SOU must determine it's action by examining the condition name.

- Ring limiting feature. (Stops normal flow of error processing)

- SOUs must return cannot use non-local goto.

- SOUs exist for duration of command.

- SOUs may signal conditions.

- If an SOU sets the 'crash' flag, condition 'CRASH$' is signalled.

- SOU has count associated. May be 'made' multiple times.
  Only removed when count = 0.

STATIC ON-UNITS - Routines

USER ROUTINES

MKSON$ (sou_ecb, code)	- make a SOU
RVSON$ (sou_ecb, code)	- revert a SOU

INTERNAL ROUTINES

WRL$ (list_ptr, nent)	- return pointer to SOU list
SOUR3_ (list_ptr)	- return pointer to ring 3 SOUs
SOR0$	- invoke ring 0 SOUs
SOR3$	- invoke ring 3 SOUs
INSOU$ (key)	- mark both SOU lists empty or clear down SOU list

STATIC ON-UNITS - Data Structures

```
2 cflags                          /* Condition Frame CFLAGS extended */
  3 crawlout bit(1),
  3 continue_sw bit (1),
  3 return_ok bit (1),
  3 inaction_ok bit (1),
  3 specific bit (1),
  3 ring_limit bit (2),         /* Stop handling condition at this ring
                                   1 = ring 1, 2 = ring 0, 3 = ring 3,
                                   0 = no limit                          */
  3 sou_crash bit (1),          /* set if sub-system unrecoverable      */
  3 sou_comp_hndld bit (1),     /* set if completely handled by SOU     */
  3 mbz bit (7),


PUDCOM now includes:  2 static_on_units (4),        /* ring 0 SOUs  */
                        3 sou_ecb ptr,
                        3 sou_status fixed bin(15),

CLDATA now includes:  2 static_on_units (10),       /* ring 3 SOUs  */
                        3 sou_ecb ptr,
                        3 sou_status fixed bin(15),
```

STATIC ON-UNITS - Modified Routines

```
DOSSUB, STD$CP - Mark SOU lists empty

                                 → Ring0 to Ring 3
SIGNL$         - If crawlout_needed   ring_limit = 2
                      Invoke all ring 0 SOUs        /* ring 0 limit */
                      If SOU_CRASH = 1 signal 'CRASH$'
                 Else call CRAWL_

                                        INVOKE
DF_UNIT_       - invoke all SOUs     - All ring 0 & Ring 3 Static on Units
                 If SOU_CRASH = 1 signal 'CRASH$'
                 If SOU_COMP_HNDLD = 1 return
                 If ring_limit = 3 return           /* ring 3 limit */
                 otherwise handle condition
```

MAKE command - Formats Disc (At REV 19 changed)
options

Tries Failures 10 times - now it can be specified
# of times

MAKE creates MFD
(MFD is its own owner)

Section 15 - File System

Boot_CREATE (Creates a Boot tape)
When Booting from a tape BOOT 505 (See Admin Guide)

√DISK STRUCTURES

A disk drive is divided into one or more partitions where a partition is one or more pairs of heads. Each partition must contain:

1).	MFD	(Master file directory)
2).	DSKRAT	(Disk record availability table)
3).	BOOT	(For initial loading)
4).	UFD DOS	(Initially empty - not actually required)
5).	BADSPT	(If badspots on the disk)

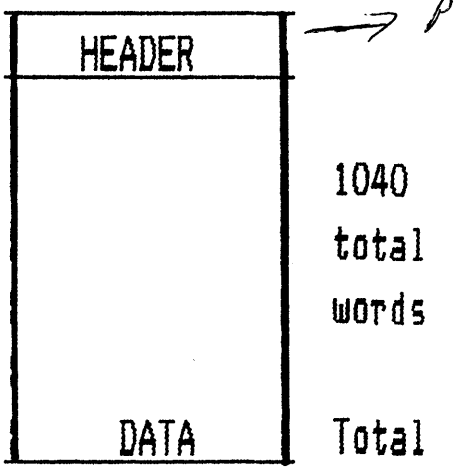
Dos> * Dos 64

LOG REC
EVENT Log File
(Logs Bad Spots)

Each partition is divided into 1040 word records.
(16 Bit words)

The record header is-16 words for storage modules devices.

The remainder of the record holds data (1024 words).



→ physical Disc record Header

RECORD HEADER FORMAT - 1040 WORD

0		
1	REKCRA	RECORD ADDRESS OF THIS RECORD
2		
3	REKPOP	RA OF DIRECTORY ENTRY OF THIS RECORD
4	REKDCT	NUMBER OF DATA WORDS IN RECORD
5	REKTYP	TYPE OF FILE (Only on first record)
6		(SAM File DAM File)
7	REKFPT	RA OF NEXT SEQUENTIAL RECORD
8		
9	REKBPT	RA OF PREVIOUS RECORD
10	REKLVL	INDEX LEVEL FOR DAM FILES
11		
12		
13		
14	Reserved	
15		

RECORD HEADER - Notes

1). REKPOP, The beginning record address (also known as REKBRA) of the first record in the file points to the beginning record address of the directory in which the file entry appears. In all other records, REKPOP points to the first record in the file.

2). REKFPT contains the address of the next sequential record in the file or, if this is the last record in the file REKFPT is zero.

3). REKBPT contains the address of the previous record in sequence or, if this is the first record in the file REKBPT is set to zero.

4). REKTYP is valid only in the first record of a file.
Possible values are:

0	SAM file
1	DAM file
2	SAM segment directory (Sub files with # not Names)
3	DAM segment directory
4	UFD user file directory (Password)
5	ACL directory
6	Access category

If the file is BOOT (Record 0) or DSKRAT bit 1 of REKTYP will be set.

NEW DSKRAT FORMAT

CHANGES TO THE DSKRAT: 1040 Words

- CYLS: number of cylinders (tracks) on this device
- REV_NUM: revision stamp

```
dcl 1 disk_rat based,             /* Usually found in LOCATE buffer */
      2 len fixed bin,            /* no. of words in DSKRAT header  */
      2 rec_size fixed bin,       /* phys. record size (~~448~~ or 1040)*/
      2 disk_size fixed bin(31),  /* number of records in partition */
      2 heads fixed bin,          /* number of heads in partition   */
      2 spec_bits,
         3 dummy bit(14),           only indicates it was shut down improperly
         3 crash bit(1),          /* improperly shut down last time */
         3 dos bit(1),            /* DOS modified or perm. broken   */
      2 cyls fixed bin,           /* number of cylinders (tracks)   */
      2 rev_num fixed bin,        /* Rev. number */
      2 rat(0:1015) bit (16) aligned; /* The RAT itself */
                                        0-1015 Records
```

OLD BADSPOT FILE FORMAT   create By make or Fixed Disc

- Save memory image. Can be RESTored, then modified with VPSD.
- N entries in the file. One for each badspot.
- Each entry consists of: track number and head number.

NEW BADSPOT FILE FORMAT - MOTIVATION

- Single record badspots, instead of mapping out a whole track.
- Allows remapping of bad records (COPY_DISK, PHYRST).

IMPLEMENTATION

- Created by MAKE, or FIX_DISK with -CONVERT_19.
- COPY_DISK and PHYRST do not understand file system structures. Create an 'equivalence' block to a goodspot.
- FIX_DISK and MAKE understand file system structures. Adjust the DSKRAT to include remapped badspot entries.
- PRIMOS does not create badspot entries, nor remap badspots.
- Primos preloader will use new BADSPT file to avoid badspots on the paging surface.

NEW BADSPOT FILE FORMAT - Data Structures

- BADSPT file header:

```
dcl 1 badspt_file_header,
      2 bad_blk_off fixed bin,  /* offset of the 1st badspt blk */
      2 MBZ fixed bin,          /* must be zero                 */
      2 file_size fixed bin,    /* size of the badspt file      */
      2 reserve(5) fixed bin;
```

- Badspot entry:

```
dcl 1 badspt_blk_header,
      2 bcw,                 /* block control word                  */
        3 type bit(4),       /* block type (badspt blk type = 0)    */
        3 length bit(12),    /* length of this block                */
      2 badspt_blk((badspt_blk_header.bcw.length-1)/2)
        3 track fixed bin,   /* track number                        */
        3 sector bit(8),     /* sector number+1, 0 for whole track*/
        3 head bit(8);       /* head number                         */
```

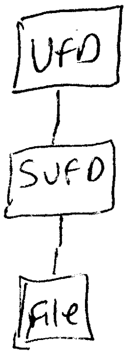
NEW BADSPOT FILE FORMAT

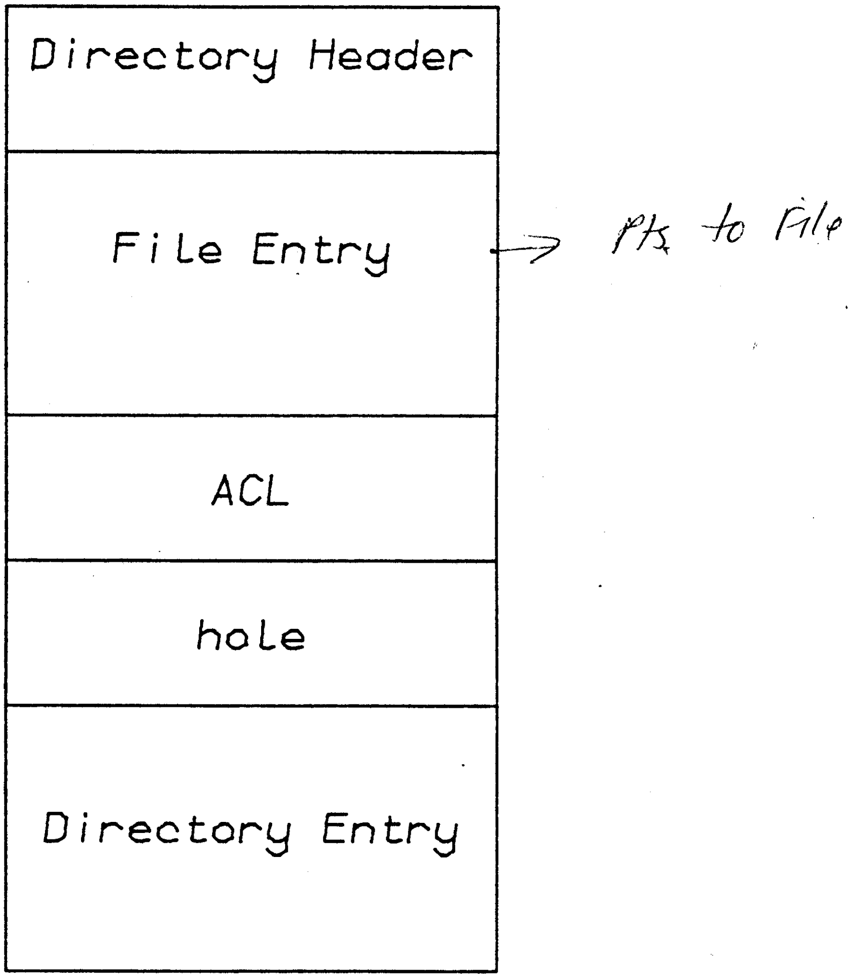
- Remapped badspot entry:

```
dcl 1 eqv_blk_header,
      2 bcw,                     /* block control word         */
        3 type bit(4),           /* type of this block
                                    (eqv blk type = 1)         */
        3 length bit(12),        /* length of this block       */
      2 eqv_blk((eqv_blk_header.bcw.length-1)/2)
        3 bad_track fixed bin,   /* bad track number           */
        3 bad_sector bit(8),     /* bad sector number+1        */
        3 bad_head bit(8),       /* bad head number            */
        3 eqv_track fixed bin,   /* equivlant track number     */
        3 eqv_sector bit(8),     /* equivlant sector number+1 */
        3 eqv_head bit(8);       /* equivlant head number      */
```

DIRECTORY STRUCTURE

-A directory is a header followed by a bunch of entries.





Directory Header

File Entry

Pts to File

ACL

hole

Directory Entry

-Note: ACLs are embedded in the directory itself.

DIRECTORY STRUCTURE

```
dcl 1 dir_hdr based,                  /* dir header entry structure */
      2 ecw like ecw,
      2 owner_password char(6),       /* Owner password             */
      2 non_owner_password char(6),   /* Nonowner password          */
      2 spare1 fixed bin,
      2 max_quota fixed bin (31),     /* Max Quota                  */
      2 dir_used fixed bin (31),      /* Quota used in this dir     */
      2 tree_used fixed bin (31),     /* Quota used in whole subtree*/
      2 rec_time_prod fixed bin (31), /* Record/time product        */
      2 prod_dtm like fsdate,         /* DTM of record/time product */
      2 spare2(5) fixed bin;

                                         (What type of ENTRY)
dcl 1 ecw based,                      /* Entry control word         */
      2 type bit(8),                  /* Type of entry              */
      2 len bit(8);                   /* Length of entry            */

replace dir_hdr_ecwt by '01'b4,       /* ECW types: directory header*/
        vacant_ecwt  by '02'b4,       /*            vacant entry    */
        file_ecwt    by '03'b4,       /*            file entry      */
        acc_cat_ecwt by '04'b4,       /*            access category */
        acl_ecwt     by '05'b4;       /*            ACL itself      */
```

DIRECTORY STRUCTURE - Entry Types

- Directory Header

- Vacant Entry: Unused entry (hole) in the directory.

			file_ent.file_info.type
- Normal Entry:	Describes a file:	SAM	0
		DAM	1
		SEGSAM	2
		SEGDAM	3
	or a directory:	Password	4
		ACL	5

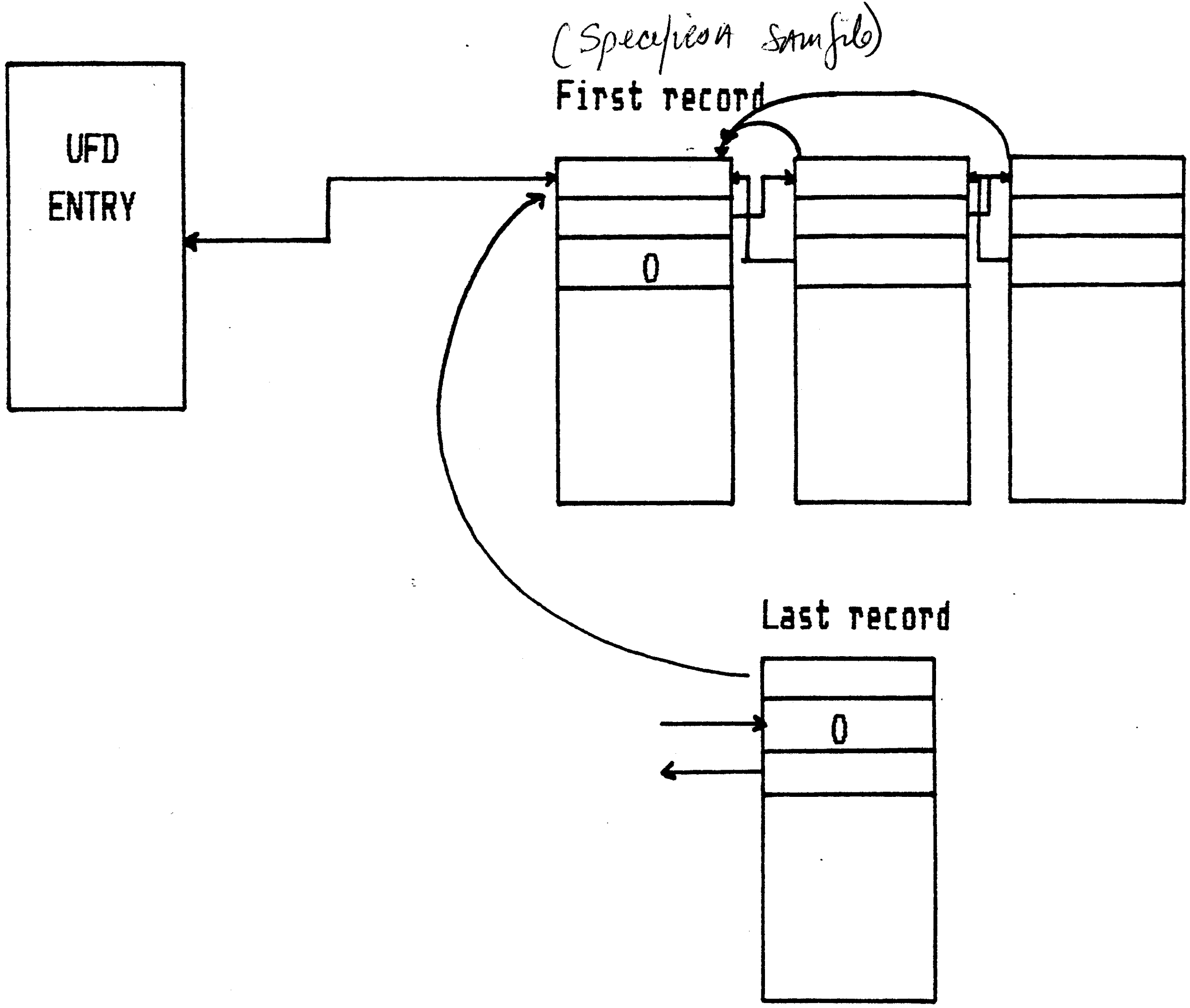
- ACL Entry: Set of access pairs.

- Access Category: Named ACL. Always points to an ACL entry.

SEGMENT DIRECTORY FORMAT (Form for organizing files)

0	BRA 0	Beginning record address
1		of the first file in the directory
2	BRA 1	Beginning record address
3		of second file in directory
4	0	Null entry
5		
2n	BRA n	Beginning record address
2n+1		of the last file in the directory

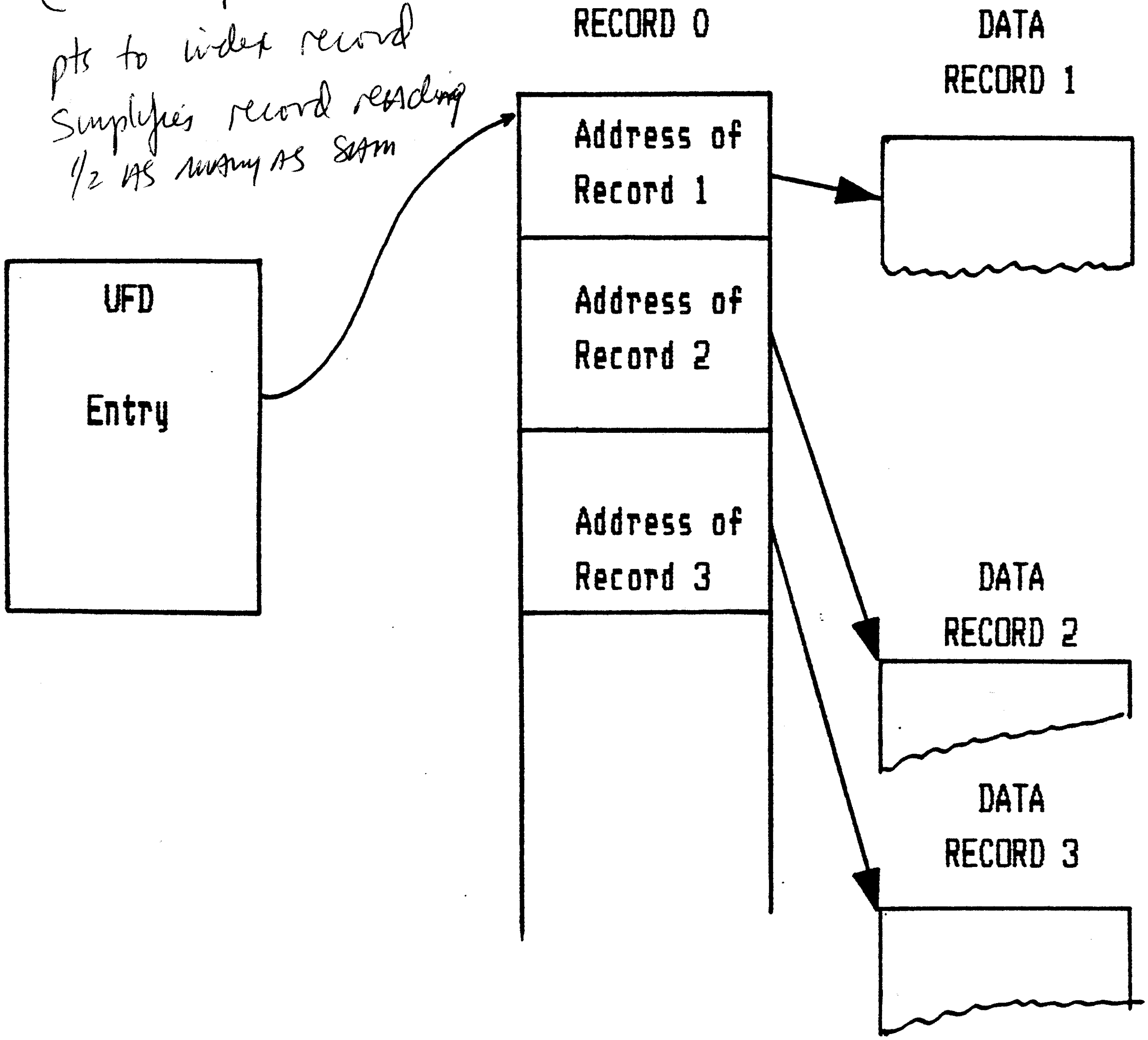
SAM FILE — set up to read sequential
uses forward & backward pointers



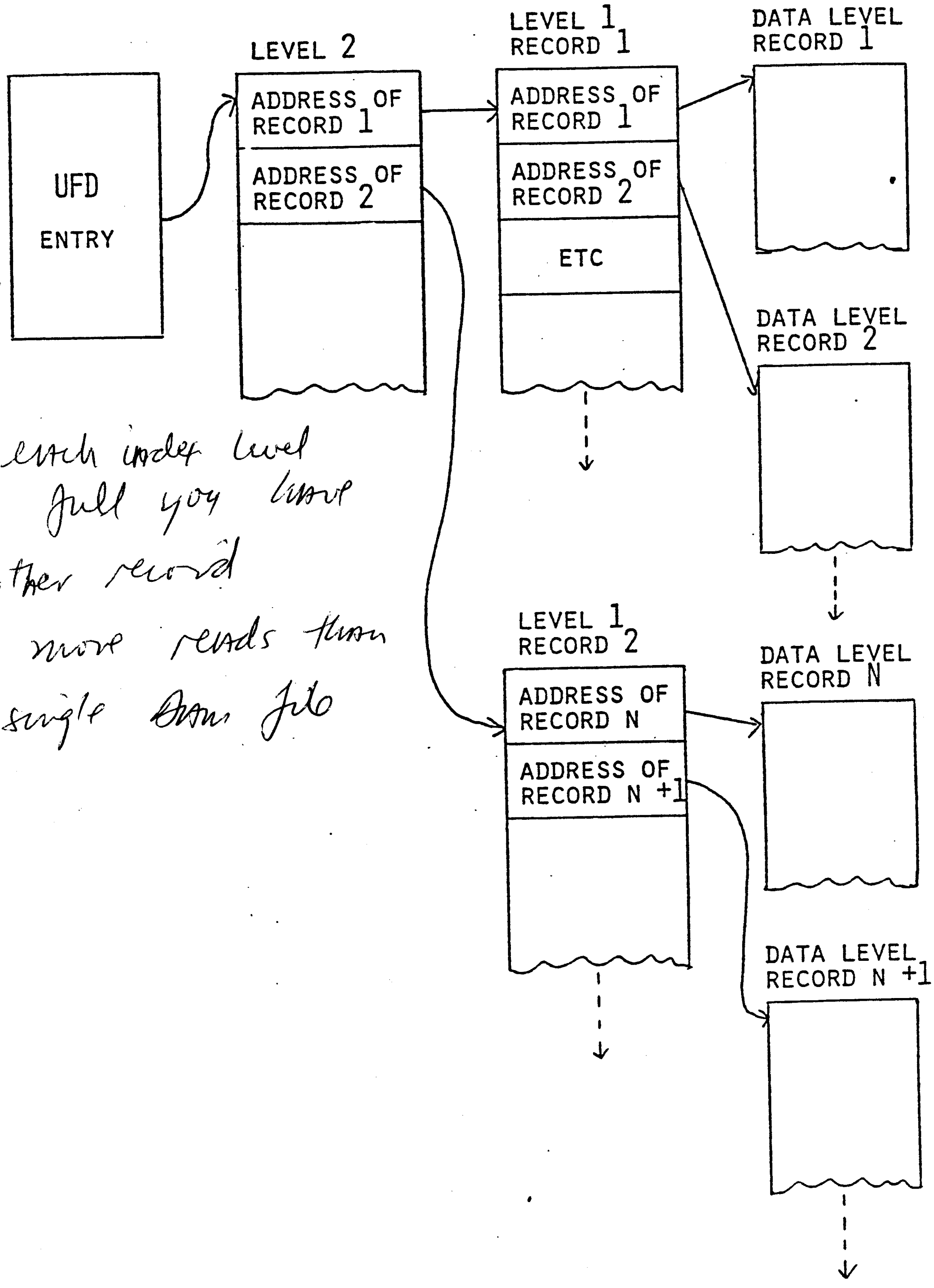
Limited to 512 Records For Non Sequential reading

DAM FILE (single level)

(intervening record) pts to index record Simplifies record reading 1/2 AS many AS SAM



DAM FILE (MULTILEVEL)



When each index level gets full you have Another record takes more reads than A single Dam file

DIRECTORY STRUCTURE

Normal Entry

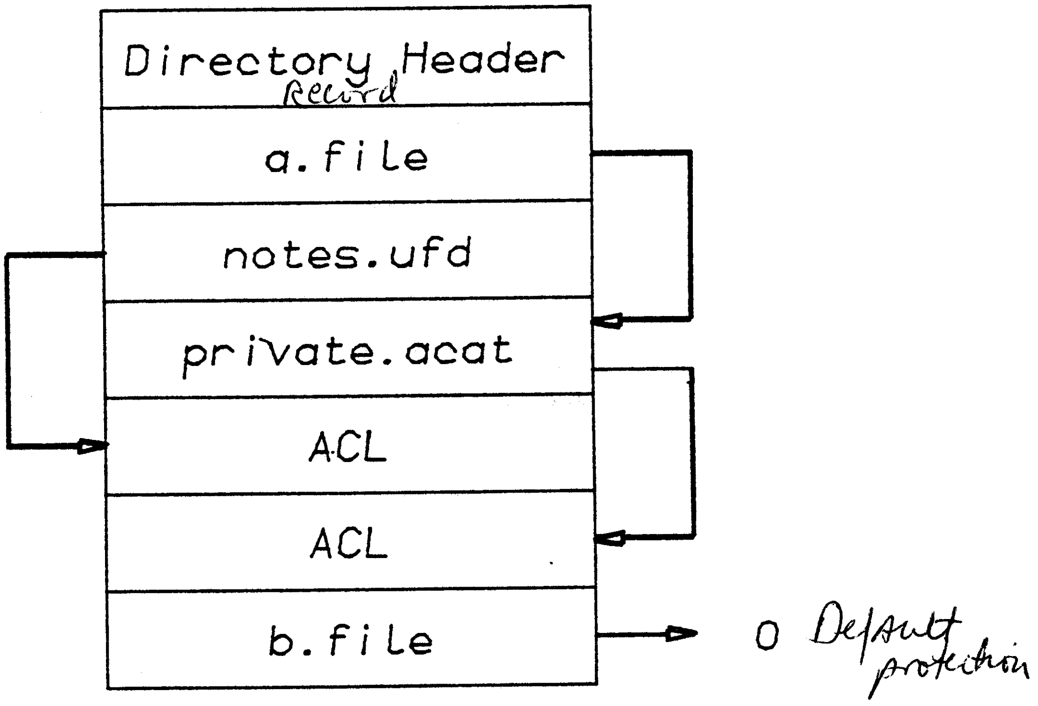
-ACL_POS

Position in the directory of the ACL protecting this object

if specific protection then pointer is to an ACL.
if category protection then pointer is to access category.
if default protection then pointer is zero.

Access category

group of files
All connected by
the same Accle



-Note. the ACL protecting this directory Lives in the directory along with the entry describing this directory.

DIRECTORY STRUCTURE - Normal Entry

See Handout for 19.2

- Normal entry for a file or directory:

```
dcl 1 file_ent based,               /* Structure of file entry        */
      2 ecw like ecw,
      2 bra fixed bin (31),         /* bra of file                    */
      2 spare1(3) fixed bin,
      2 protec bit (16),            /* Protection keys                */
      2 acl_pos fixed bin,          /* Position of ACL, assumes
                                       dir <= 64k                     */
      2 dtm like fsdate,
      2 file_info,
        3 long_rat_hdr bit (1),     /* '8000'b4:  file is a long RAT */
        3 dumped bit (1),           /* '4000'b4:  has been backed up */
        3 dos_mod bit (1),          /* '2000'b4:  modified under DOS */
        3 special bit (1),          /* '1000'b4:  Special file        */
        3 rwlock bit (2),           /* Bits 5-6:  Concurrency lock    */
        3 spare bit (2),            /* Bits 7-8:  Unused              */
        3 type bit (8),             /* Bits 9-16: File type           */
      2 scw fixed bin,              /* Length of name subentry        */
      2 name char (32);             /* Name of object                 */
```

DIRECTORY STRUCTURE - ACL Entry

FORMAT OF AN ACL:

- An ACL consists of three parts:

  A user_id section
  An ACL groups section
  A $rest section

- Each section is a set of access pairs.

- An ACL may be up to 255 words in length.

- Each access pair specifies ACL rights for:

  ~~Ring 1~~ (not implemented)
  Ring 3

DIRECTORY STRUCTURE - ACL Entry

- Directory entry for an ACL:

```
dcl 1 acl_ent based,                    /* Dir entry for an ACL         */
      2 ecw like ecw, (ENTRY control word) /* See above                 */
      2 user_count fixed bin,           /* Number of user entries       */
      2 group_count fixed bin,          /* Number of group entries      */
      2 version fixed bin,              /* Version number of structure */
      2 spare1 fixed bin,
      2 group_offset fixed bin,         /* Relative position of first
                                           group entry                  */
      2 rest_accesses like accesses,    /* Rights for $REST             */
      2 owner_pos fixed bin,            /* Position of owner in dir     */
      2 dtm like fsdate,                /* Date/time last modified      */
      2 spare2 fixed bin,
      2 entry like coded_access;        /* See below */
```

DIRECTORY STRUCTURE - ACL Entry

- Format of a single access pair:

```
dcl 1 coded_access based,       /* Entry in an ACL       */
      2 scw fixed bin,          /* Length only           */
      2 access like accesses,   /* <access>              */
      2 spare(2) fixed bin,
      2 id char(32) var;        /* <id> */

dcl 1 accesses based,           /* A 16-bit access word */
      2 ring1 like acc_bits,
      2 ring3 like acc_bits;

                                                                  BiT
dcl 1 acc_bits based,           /* Access bit definition       */
      2 protect bit(1),         /* Directory accesses -- Protect */  9
      2 delete bit(1),                              /* Delete  */  10
      2 add bit(1),                                 /* Add     */  11
      2 list bit(1),                                /* List    */  12
      2 use bit(1),                                 /* Use     */  13
      2 execute bit(1),         /* File accesses --    Execute */  14
      2 write bit(1),                               /* Write   */  15
      2 read bit(1);                                /* Read    */  16
```

DIRECTORY STRUCTURE - Access Category Entry

- An access category is a named ACL.

- It is a pointer to an ACL entry.

- Each file system object protected by the category points to the access category entry, not the ACL itself.

- The name field of an access category is always padded to 32 characters in order to reduce directory fragmentation.

```
dcl 1 acc_cat_ent based,      /* access category directory entry     */
      2 ecw like ecw,
      2 spare1(6) fixed bin,
      2 acl_pos fixed bin,    /* Position of ACL itself              */
      2 dtm like fsdate,      /* Date/time last modified             */
      2 file_type fixed bin, /* For compatibility with normal entry */

      2 scw fixed bin,        /* Length of name subentry             */
      2 name char (32);       /* Name of object, (padded to 32 chars)*/
```

Section 16 - Unit Tables

UNIT TABLES (File units)

OLD METHOD

- Unit tables statically allocated at cold start (AINIT).

- 2048 file units per system.

NEW METHOD

- Per-User unit tables allocated/deallocated dynamically.

- Constrains working set of unit table databases to what is actually being used.

- Vital statistics:

  3247 file units available per system

16	guaranteed per user (default)
1	system unit per user (unit #0)
3	attach points (home,current,initial) per user
127	maximum 'usable' file units per user

UNIT TABLES - Definitions

- A unit table (ut) is a list of pointers to unit table entries.

- A hash table is a set of pointers to linked lists of unit table entries.

- A unit table entry (ute) desribes a file system object that is currently in use via the file system.

- A file system object is a data file, directory or access category. These objects may reside on a local or a remote system.

- UTBTMP is the unit table bit map, 128 bits (8 words).

- UTBITS is the unit table entries bit map, 3247 bits (203 words)

  Each ut or ute has one bit corresponding to it:
      = 0   in use
      = 1   available

  The first available ut or ute is always allocated.

UNIT TABLES

The following steps are performed in order to use a file system object:

- Allocate a unit table:
  for system user at cold start (BINIT)
  for terminal users during login (NLOGIN)
  for phantom users by spawner (PHNTM$)
  for slaves when they are awoken (NPXPRC)

- Allocate a unit table entry when a file system object is 'opened'.

- Access the ute:
  by the file system via the hash table.
  by a user program via the unit table.

- Deallocate the ute when the object is 'closed'.

- Deallocate the unit table:
  for terminal/phantom users during logout (LO_CLEAN)
  for slaves when they go to sleep (NPXPRC)

UNIT TABLES (1 for each user)

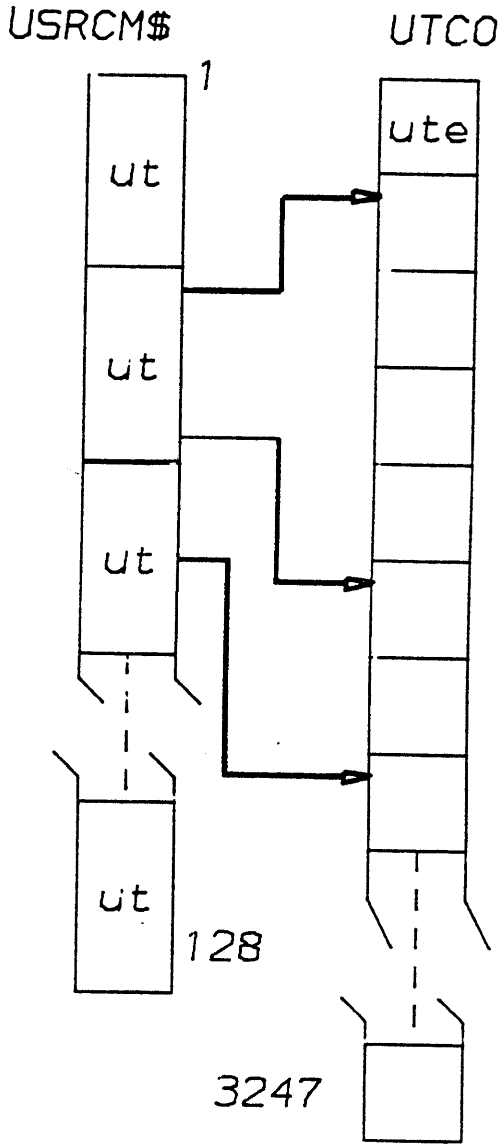
(File UNITS)

Data Structures

pudcom.Lusr indexed_by unit

Unit Table (ut)

```
  0:  system unit
  1:
      usable file units

127:
128: current attach point
129: home attach point
130: initial attach point
```

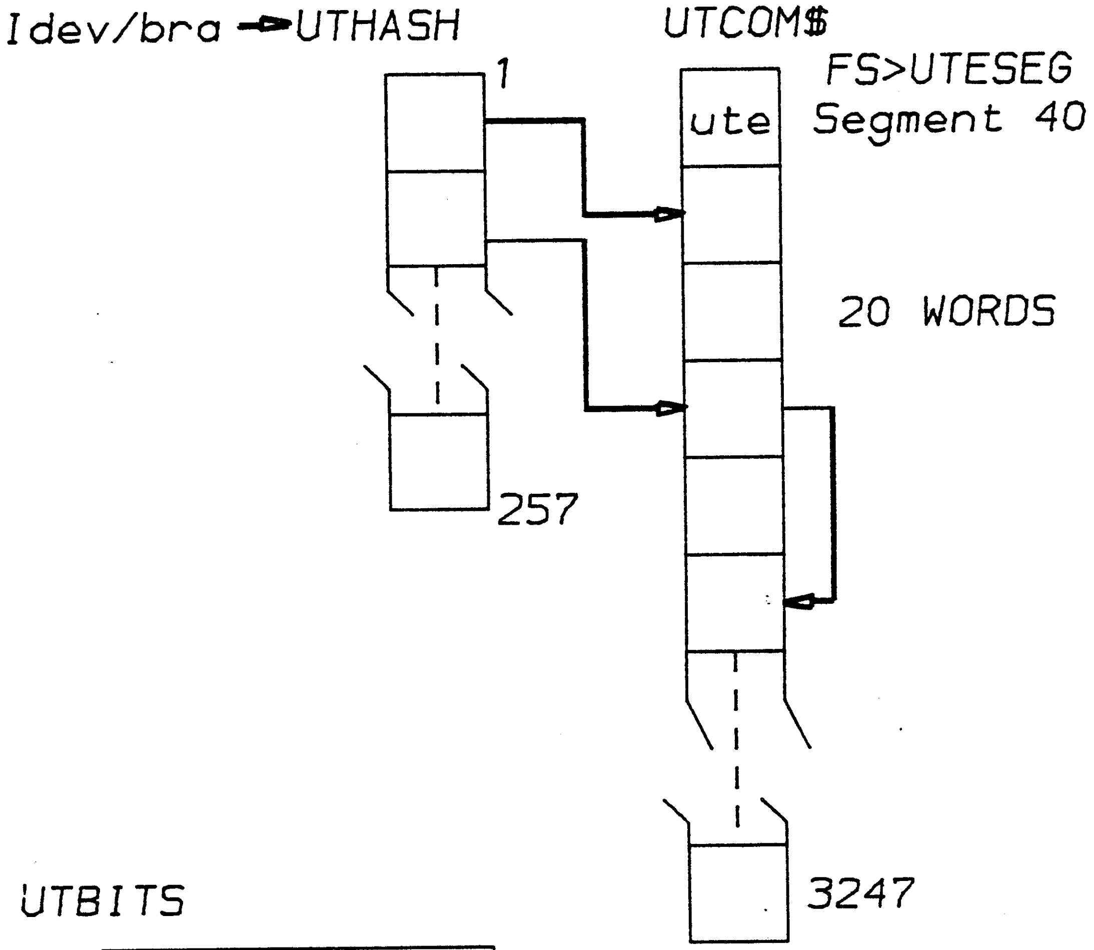


UTBTMP

0	1001111111111111
1	1111111111111111
2-6	
7	1111111111111111

UNIT TABLES

Data Structures



UTBITS

```
0        1010101111111111
1        1111111111111111
2-202
203      1111111111111111
```

UNIT TABLES - Types of UTEs

IN Memory

Files: SAM, DAM, SEGSAM, SEGDAM

Directories: Password protected
ACL protected

Attach Points: Password protected
ACL protected

Access Categories

Remote Units (of any type)

New Elements of a File/Directory UTE

ACCESS ACL access allowed for this user on this file/dir. (Owner/Non-owner access is mapped to ACL access)

QUOTA_BLK_PTR Pointer to the quota block chain for this file/directory to maintain quota information.

DIR_BLK_PTR Pointer to the directory block for the parent of this file/directory to maintain record usage information.

We only can attach to Directories

UNIT TABLES - Data Structures

- Files and directories (not opened as attach points):

```
Dcl 1 utcme based,                 /* File/Directory Unit Table Entry */
      2 vstat like status_bits,     /* See below                       */
      2 bra fixed bin (31),         /* BRA of file                     */
      2 ldevno fixed bin,           /* logical device number           */
      2 cur_ra fixed bin (31),      /* current r.a. in file            */
      2 rel_wordno fixed bin,       /* position within current record*/
      2 rel_recno fixed bin (31),   /* ordinal record no. in file      */
      2 rwlock bit(8),              /* Read/write concurrency lock     */
      2 access like access_bits,    /* Accesses allowed on file        */
      2 parent_bra fixed bin (31),  /* BRA of parent directory  owner of file */
      2 pos_in_parent fixed bin,    /* position in parent              */
      2 hash_thread fixed bin,      /* hash thread                     */
      2 quota_blk_ptr fixed bin,    /* Quota block pointer             */
      2 dir_blk_ptr fixed bin,      /* Directory block pointer         */
      2 dam_idx_ra fixed bin (31),  /* current r.a. in DAM index       */
      2 spare(2) fixed bin;
```

UNIT TABLES - Data Structures

```
dcl 1 dir_utcme based,              /* attach point Unit Table Entry */
      2 vstat like status_bits,     /* See definition below          */
      2 bra fixed bin(31),          /* BRA                           */
      2 ldevno fixed bin,           /* Logical device number         */
      2 cur_ra fixed bin(31),       /* current r.a. in file          */
      2 rel_wordno fixed bin,       /* position within current record*/
      2 rel_recno fixed bin(31),    /* ordinal record no. in file    */
      2 access,                     /* Access rights                 */
         3 ring1 like access_bits,  /*    in ring 1                  */
         3 ring3 like access_bits,  /*    and ring 3                 */
      2 parent_bra fixed bin (31),  /* BRA of parent directory       */
      2 pos_in_parent fixed bin,    /* position in parent            */
      2 hash_thread fixed bin,      /* hash thread                   */
      2 quota_blk_ptr fixed bin,    /* Quota block pointer           */
      2 dir_blk_ptr fixed bin,      /* Quota directory block pointer */
      2 acl_bra fixed bin (31),   /* BRA of directory containing ACL */
      2 acl_pos fixed bin,        /* Position of ACL in dir          */
      2 spare fixed bin;
```

New Elements of an Attach Point UTE

ACCESS.RING1 ACL access available under ring 1. (not implemented)

ACCESS.RING3 ACL access available under ring 3.
(Access from ring 0 is ALL).

QUOTA_BLK_PTR Pointer to the quota block chain for this directory.

DIR_BLK_PTR Pointer to the directory block for this directory
(not the parent).

ACL_BRA
and ACL_POS BRA and word offset pointing to the ACL protecting
this directory.

Remote Units

- Remote units are a 'pointer' to a remote ute.

```
Dcl 1 rem_ute based,                    /* UTCOM$ entry for remote units */
      2 vstat like status_bits,
      2 master_to_slave fixed bin,  /* NPX Master-Slave Mapping       */
      2 real_ldevno fixed bin,      /* Ldev (normally in ldevno)      */
      2 negative_node fixed bin,    /* -(node no. of remote system)  */
      2 packname char (32);         /* NPX Packname                   */
```

UNIT TABLES - Data Structures

```
dcl 1 status_bits based,      /* VSTAT definition                    */
      2 modified bit (1),     /* modified                            */
      2 sysuse bit (1),       /* open for system use                 */
      2 shtbit bit (1),       /* device shut down                    */
      2 no_close bit (1),     /* special file, not closed by C -ALL */
      2 spare bit (1),
      2 file_type bit (3),    /* Defined below                       */
      2 open_mode bit (8);    /* Accesses which file is opened with */

      file_type:
        sam_ftype     by 0,       /* File types: SAM file  */
        dam_ftype     by 1,       /* DAM file              */
        samseg_ftype  by 2,       /* SAM segment directory */
        damseg_ftype  by 3,       /* DAM segment directory */
        dir_ftype     by 4,       /* Directory             */
        acl_dir_ftype by 5,       /* ACL directory         */
        acc_cat_ftype by 6;       /* Access category       */
```

Section 17 - Locate Mechanism

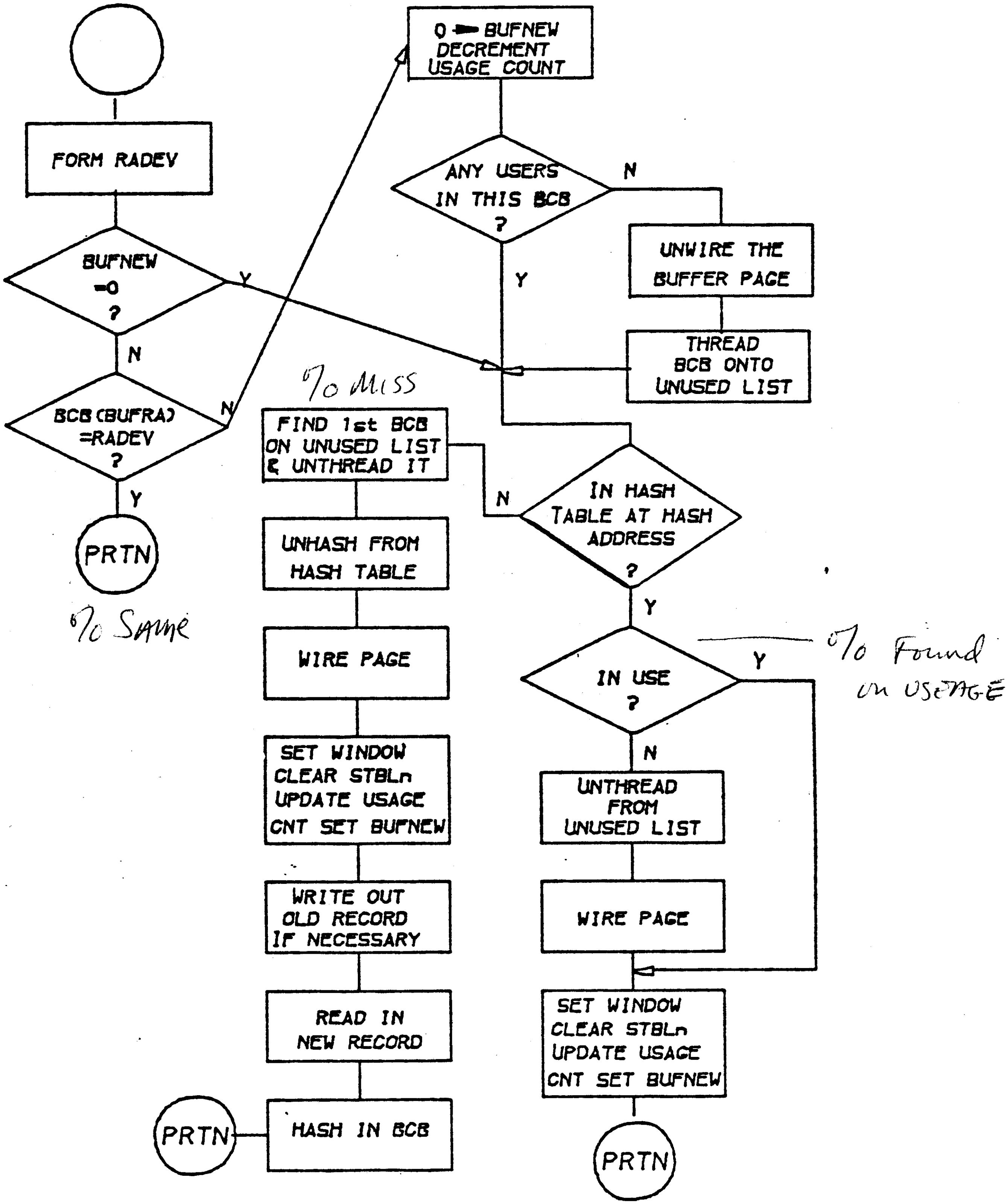
BUFFER CONTROL BLOCK (BCB)

0	HASH THREAD		BUFLNK	
1	Logical dev	Record	BUFRA	
2	ADDRESS			
3	BRA of file record is in		BUFBRA	
4				
5	Process no.	Hash index	BUFUSR	
6	User count	Flag bits	BUFLAG	
7			REKCRA	disk record header
8				
9			REKPOP	
10				
11			REKDCT	
12			REKTYP	
13			REKFPT	
14				
15			REKBPT	
16				
17			REKLVL	
18	ADDRESS OF PTW FOR BUFFER		BUFPMP	
19	LRU THREAD FOR		BUFTHD	
20	UNUSED BUFFERS			
21	length of BCB		BFCLEN	

All I/O is done thru Locate Buffers

	IN HASH TABLE	
UNUSED LIST	No	Yes
No	X	Wired Memory
Yes	on Disc	Memory

FLAG BITS 16 = BUFFER MODIFIED
15 = BUFFER IN TRANSITION
14 = UPDATE MISSED


FORM RADEV
BUFNEW =0 ?
Y
N
BCB (BUFRA) =RADEV ?
N
Y
PRTN
% Same
0 ➡ BUFNEW DECREMENT USAGE COUNT
ANY USERS IN THIS BCB ?
N
Y
UNWIRE THE BUFFER PAGE
THREAD BCB ONTO UNUSED LIST
% Miss
FIND 1st BCB ON UNUSED LIST & UNTHREAD IT
N
IN HASH TABLE AT HASH ADDRESS ?
Y
UNHASH FROM HASH TABLE
WIRE PAGE
IN USE ?
Y
% Found in USAGE
N
SET WINDOW CLEAR STBLn UPDATE USAGE CNT SET BUFNEW
UNTHREAD FROM UNUSED LIST
WRITE OUT OLD RECORD IF NECESSARY
WIRE PAGE
READ IN NEW RECORD
SET WINDOW CLEAR STBLn UPDATE USAGE CNT SET BUFNEW
PRTN
HASH IN BCB
PRTN

ASSOCIATIVE BUFFERS - CONFIG DIRECTIVE

Previously- there were always 64 associative buffers which resided
in segment 1.

Now there can be any where from 8 to 256 associative buffers.

New CONFIG directive: NLBUF n
where n = the octal number of LOCATE buffers to use.

The buffers will reside in segments 50 - 53.

The 21 word Buffer Control Block (BCB) is wired at cold start.
The LOCATE buffer is only wired when it is in use.

The optimal number of associative buffers depends on the system.
If the LOCATE miss rate is greater than 10 percent,
NLBUF should be increased until
However, if PF/S is greater than 10, do not increase NLBUF.

Section 18 - Disk Quotas

DISK QUOTAS

MOTIVATION

- Provides administrative control over disk usage.
- Quota limits the number of records a single directory or directory sub-tree can use.

IMPLEMENTATION

- Specifed on a per-ufd basis.
- Units are physical disk records (2kb).
- Quota of zero means unlimited record usage is allowed.
- Quota may not be set on an MFD.
- Requires rev 19 disk format.

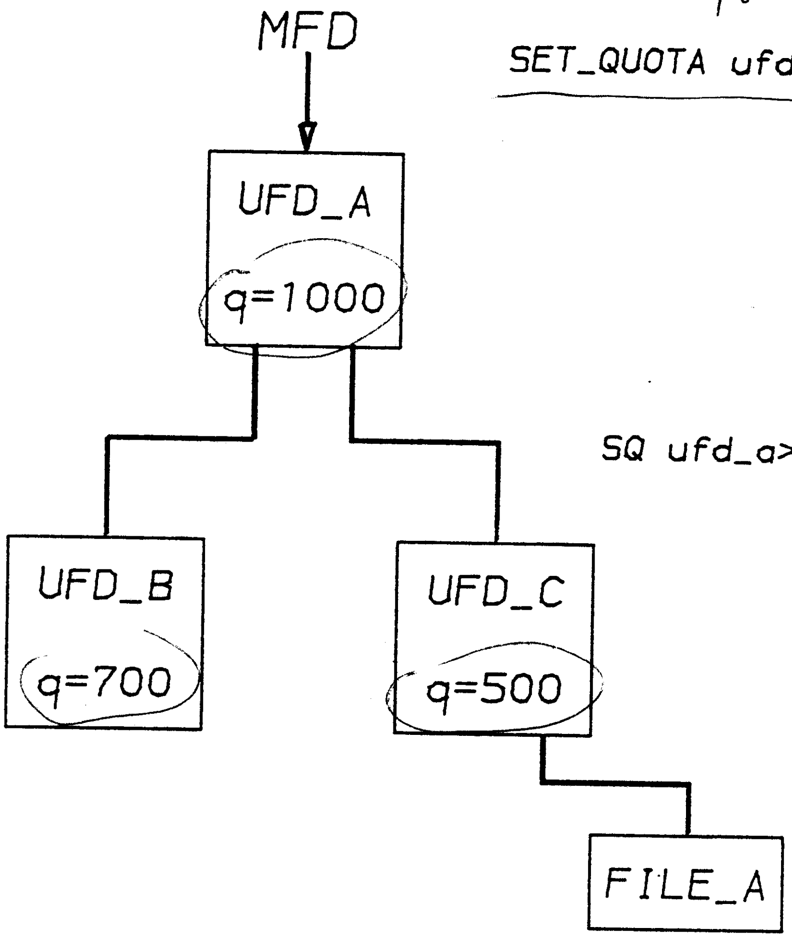
Note: No temporary file allowance, nor login/out quota.

DISK QUOTAS

Example

TO SET MAX QUOTA

SET_QUOTA ufd_a -MAX 1000



SQ ufd_a>ufd_c -MAX 500

The quota set on UFD_B is 700 records.
The quota set on UFD_C is 500 records.
The parent directory UFD_A has a quota of 1000 records.

The total records that can be used by
the entire sub-tree (UFD_A, UFD_B, UFD_C)
is 1000.

DISK QUOTAS

- Quota and non-quota directories may be intermixed in the same subtree.

- A quota directory can be subordinate to a non-quota directory, and vice versa.

- Two counters are maintained:

  DIR_USED: number of records used by this directory.
  QUOTA_LEFT: number of records still available to this subtree.

- Each time the DIR_USED count changes for any directory, the quota for that directory must be updated (if there is one).

- Each time the QUOTA_LEFT count changes for a quota directory, any superior quota directories must have their quotas updated.

DISK QUOTAS - Data Structures

DIRECTORY BLOCKS (DB)

- One directory block is maintained for each open attach point on the system.

- The dir_block contains:

USE_COUNT: number of open attach points using this block.
DIR_USED: number of records used by this directory.
NOT_MODIFIED: flag indicating if DIR_USED has changed (and info must be written back to disk).

DISK QUOTAS - Data Structures

Should only be used at the level you need it - Never on MFD Level

QUOTA BLOCKS (QB)

- A quota block is maintained for each open attach point which has a quota.

- A quota block is maintained for each superior directory of an attach point which has a quota.

- These quota blocks are chained together.

- If two open attach points are constrained by the same quota directory(s), then they will share the quota block chain.

- The quota_block contains:

  USE_COUNT: number of open attach points using this block.
  QUOTA_LEFT: the number of records still available under the quota at this directory level.
  PARENT_PTR: pointer to any superior quota directory (zero if none).

DISK QUOTAS - Data Structures

```
dcl 1 quota_block based,
      2 use_count fixed bin,         /* Use count                     */
      2 ldevno fixed bin,            /* Ldev of directory             */
      2 bra fixed bin (31),          /* BRA of directory              */
      2 hash_thread fixed bin,       /* Hash thread link to next block*/
      2 parent_ptr fixed bin,        /* Pointer to superior block     */
      2 quota_left fixed bin (31);   /* Amount left in tree           */


dcl 1 dir_block based,
      2 use_count fixed bin,         /* Use count                     */
      2 ldevno fixed bin,            /* Ldev                          */
      2 first_ra fixed bin (31),     /* BRA                           */
      2 hash_thread fixed bin,       /* Link to next block            */
      2 dtype,
        3 type bit (15),             /* Type of block                 */
        3 not_modified bit (1),      /* Quota not modified if on      */
      2 dir_used fixed bin (31);     /* Amount used in this dir       */
```

The type of the block is maintained in the DTYPE (PARENT_PTR) field.
The value is -1 for dir_blocks (-2 if modified).
All other values indicate quota_blocks.

DISK QUOTAS

MAINTAINING DIRECTORY/QUOTA BLOCKS:

- Since directory and quota blocks are the same size, they are stored in a common area (QBCOM$).

- Directory/quota blocks are allocated/deallocated in a manner similar to unit table entries.

  The hash table is QBHASH.
  The bit map is QBBITS.

- Quota_blocks are chained (threaded) together according to directory level (PARENT_PTR).

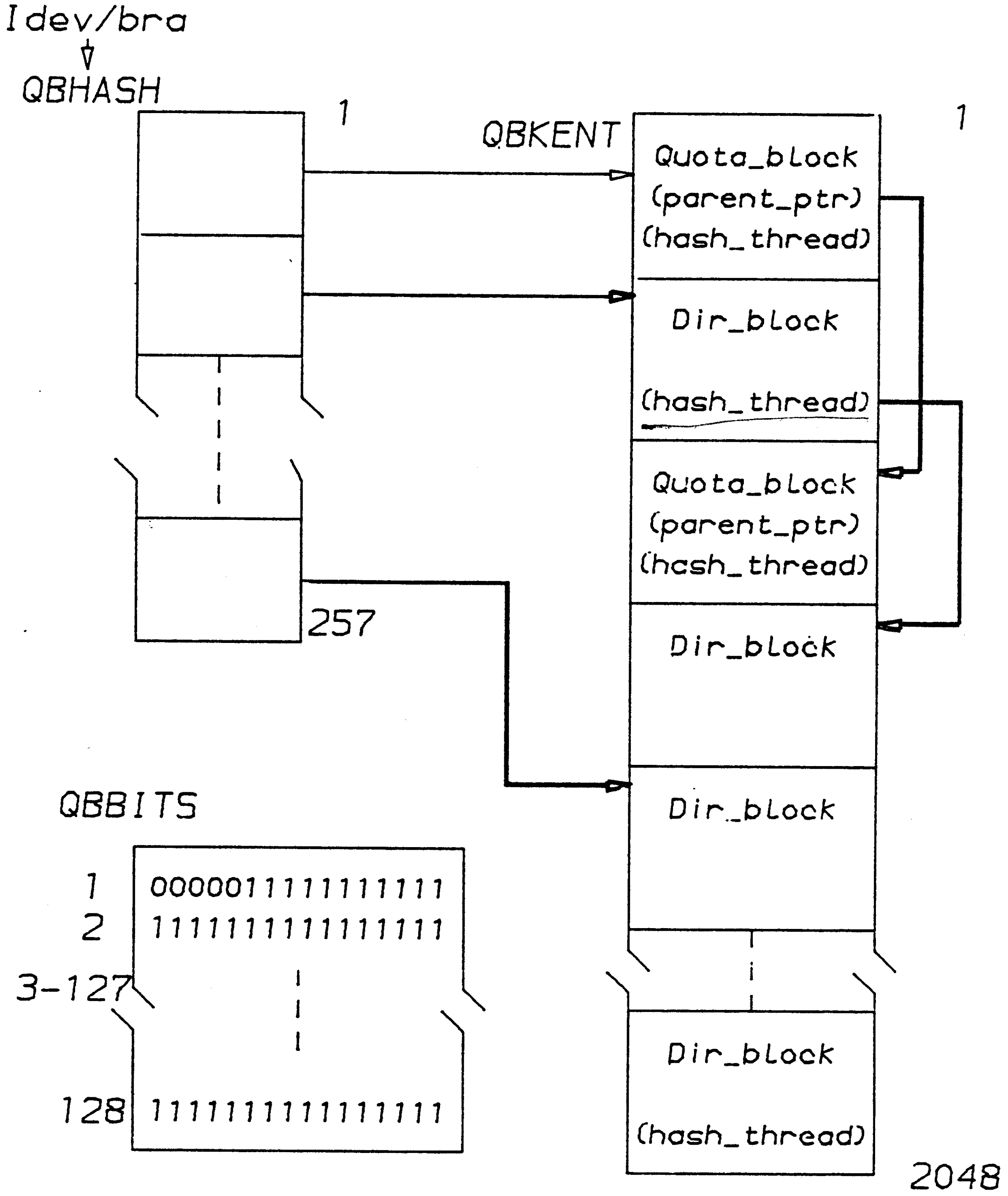
- QBCOM$ (QBHASH, QBKENT and QBBITS) are protected by the UTLOK.

- Up to 2048 quota/directory blocks may be in use at any one time.

- The hash table (QBHASH) has 257 entries which point (up) to 2048 quota/dir_blocks. Therefore both quota and directory blocks are independently threaded together in hash chains (HASH_THREAD).

DISK QUOTAS

QBCOM$ - fs>seg10.pma -Segment 10



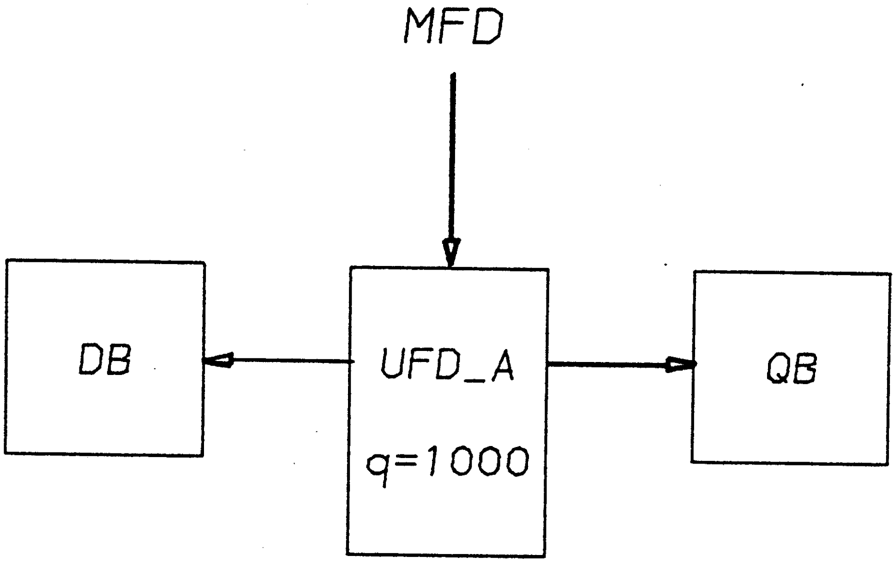
```
1     0000011111111111
2     1111111111111111
3-127
128   1111111111111111
```

DISK QUOTAS

Example

ATTACH to top-level UFD_A -AT$ABS calls AT_CLEAN:

```
if UFD_A = quota_dir
 then allocate QB

allocate DB
```

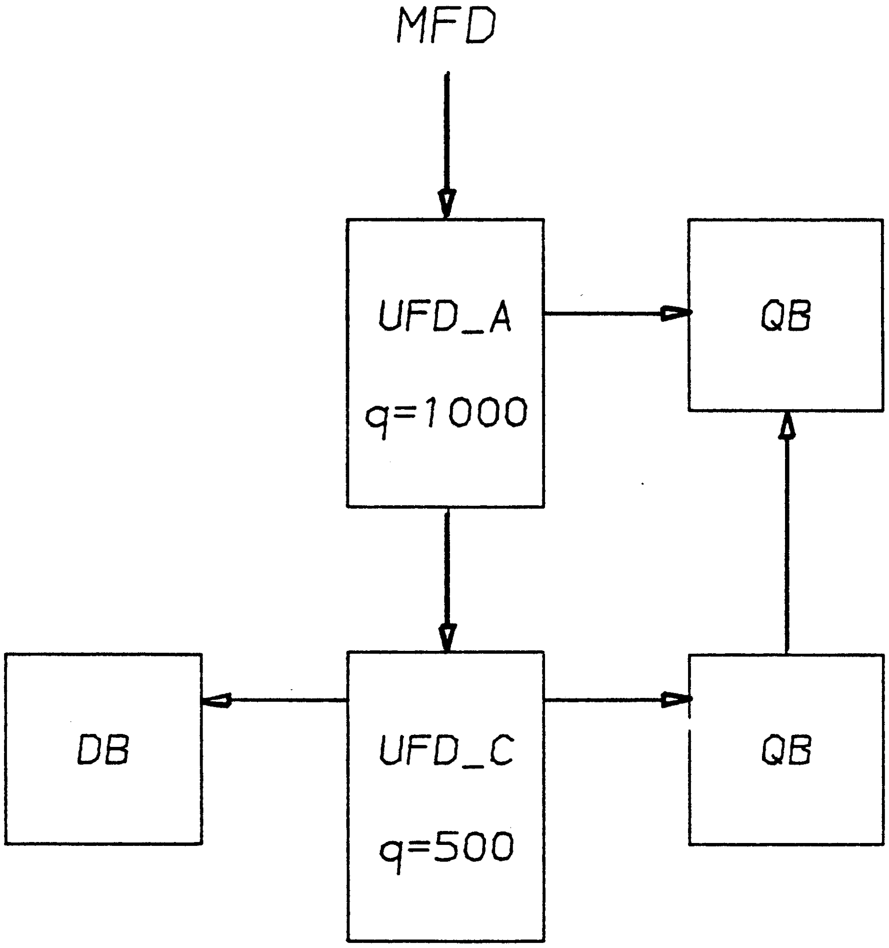


DISK QUOTAS

Example

ATTACH to subufd UFD_C -AT$REL calls AT_CLEAN:

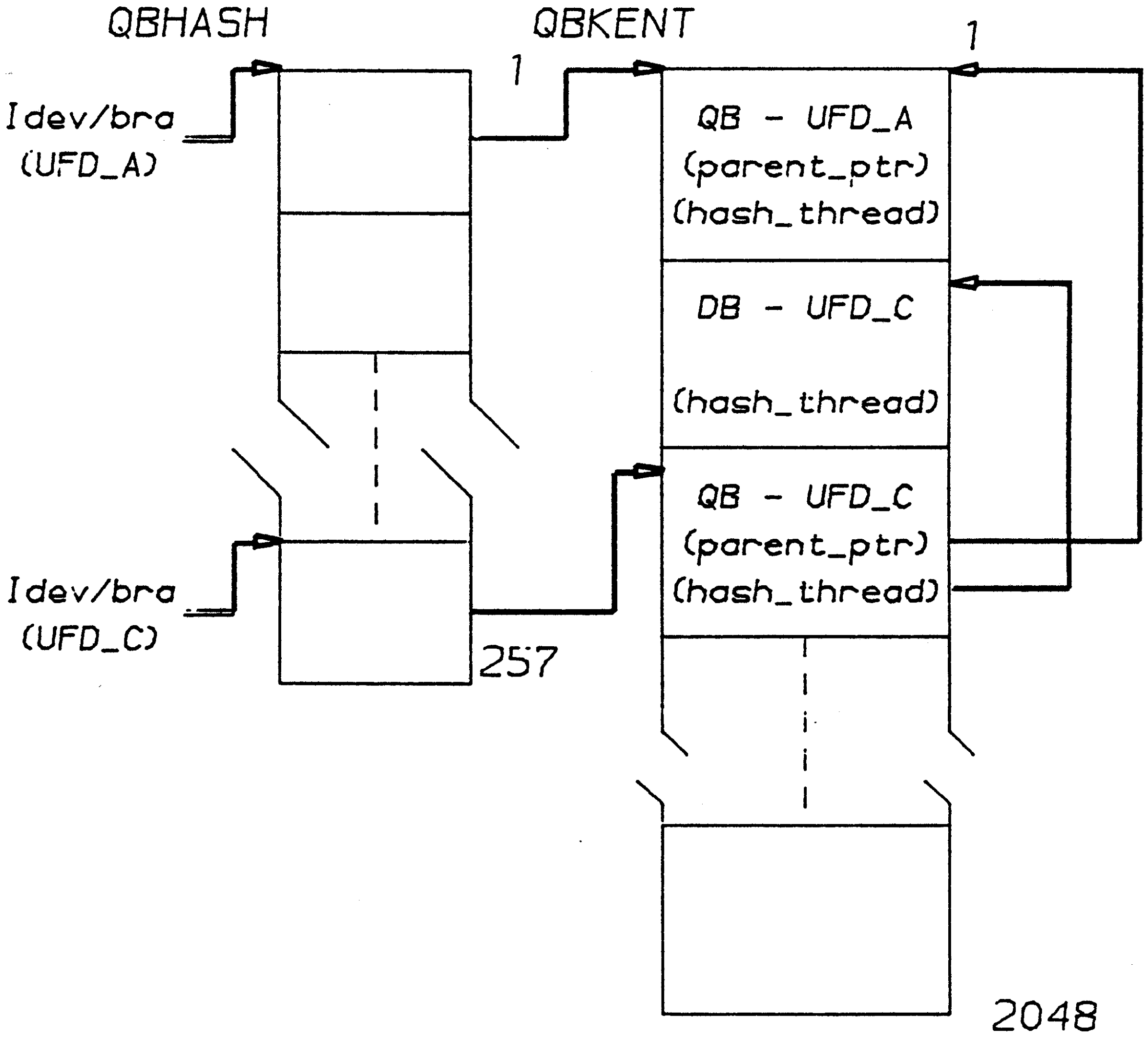
```
  if UFD_C=quota_dir
    then allocate QB
  if UFD_C=new attach point
    then deallocate old DB
  allocate DB
(QB for UFD_A is still in use by our new attach point)
```



DISK QUOTAS

Example

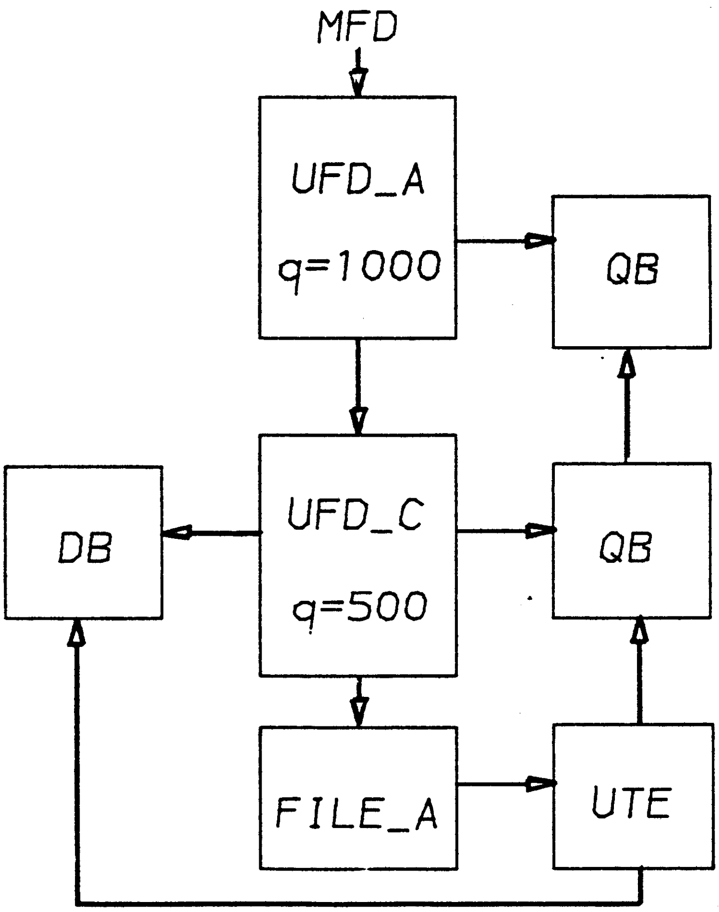
Here is what QBCOM$ looks like after the two attaches:



DISK QUOTAS

Example

```
OPEN FILE_A -SRCH$$
                        allocate unit table entry
                        set UTE.DIR_BLK_PTR to
                            parent (UFD_C)
                        set UTE.QUOTA_BLK_PTR to
                            first quota parent (UFD_C)
                        increment USE_COUNT
                            for DB (UFD_C)
                        increment USE_COUNT
                            for QB chain (UFD_C, UFD_A)
                            (USE_COUNT is now 2;
                            1 for attach + 1 for open)
```



DISK QUOTAS - Example

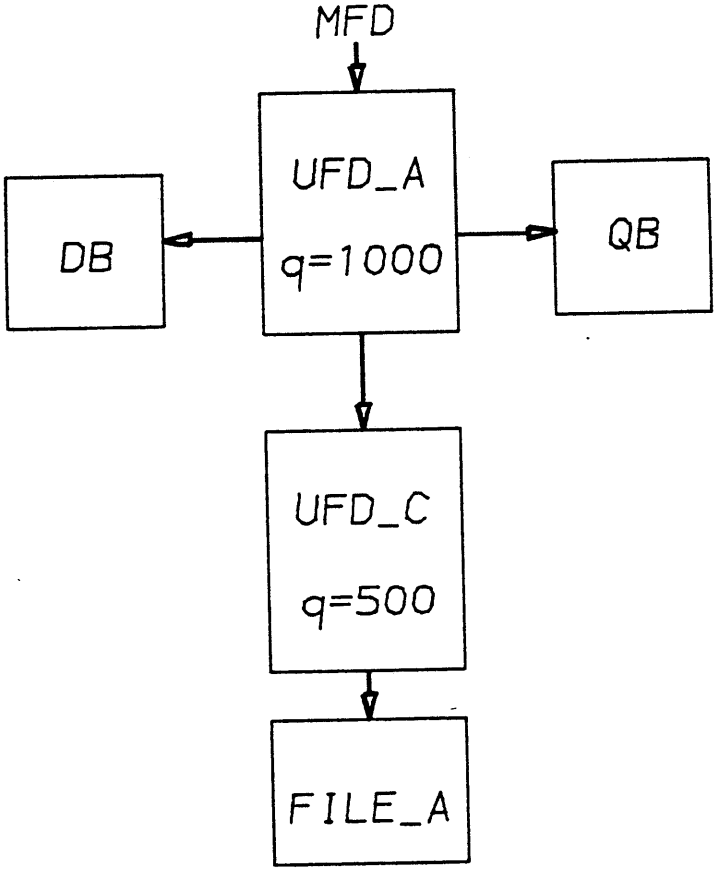
```
WRITE TO FILE_A - PRWF$$ calls GETREC:
    DIR_USED = DIR_USED + 1
    reset NOT_MODIFIED bit
    if UFD_C = quota_dir then QUOTA_LEFT = QUOTA_LEFT - 1


TRUNCATE FILE_A - PRWF$$ calls TRUNC$ calls RTNREC:
    DIR_USED = DIR_USED - 1
    reset NOT_MODIFIED bit
    if UFD_C = quota_dir then QUOTA_LEFT = QUOTA_LEFT + 1


CLOSE FILE_A - SRCH$$ calls CLOSE:
    if dir_block.NOT_MODIFIED = false
         then update DIR_USED on disk (UFD_C)
              update QUOTA_LEFT on disk (UFD_C)
              do while parent_ptr <> 0
                   update QUOTA_LEFT on disk (UFD_A)
    decrement USE_COUNT for DB (UFD_C)
    decrement USE_COUNT for QB (UFD_C)
    if USE_COUNT = 0 then deallocate dir/quota block
         (The USE_COUNT = 1 because we are still attached to UFD_C)
```

DISK QUOTAS

Example

ATTACH TO UFD_A -AT$ calls AT_CLEAN:

```
if UFD_A = quota_dir then
    increment USE_COUNT for QB (UFD_A)
if UFD_B = new attach point then
     decrement USE_COUNT for old DB (UFD_C)
if USE_COUNT = 0 then
    deallocate old DB (UFD_C)
    decrement USE_COUNT for QB (UFD_A)
    (this USE_COUNT is still 1
    because we are attached to UFD_A)
allocate DB (UFD_A)
```



Section 19 - Attach

ATTACH

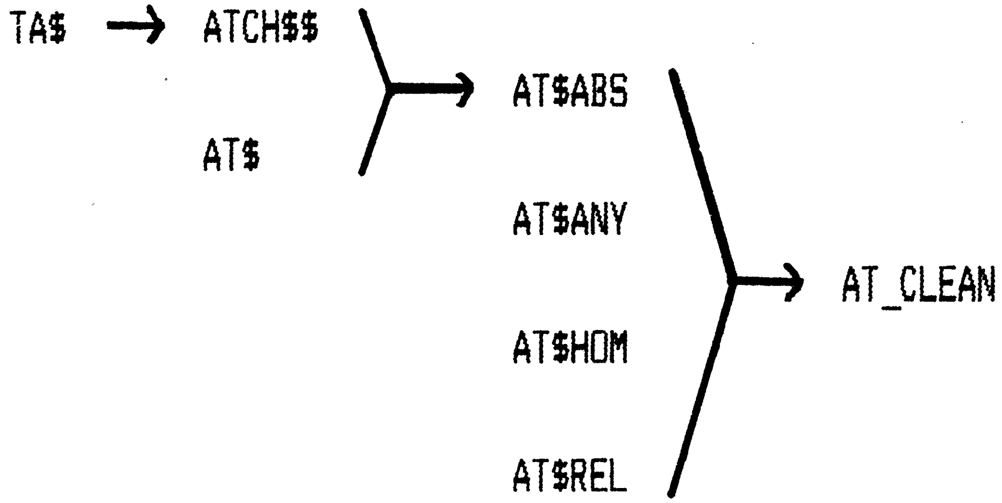
- Functionality has changed due to ability to completely exclude a user from an MFD with ACLs.

- Duplicate packnames no longer allowed.

- Passwords no longer converted to upper case by attach routines.

- Attach routines allow ring 0 callers to override access priviledges.

- New routines:



TA$ → ATCH$$

AT$

AT$ABS

AT$ANY

AT$HOM

AT$REL

AT_CLEAN

AT$OR

ATTACH - AT$ANY attach scan

```
Do (for each local partition) While (not found)
    ("open" MFD of this partition)
    If (have rights to this MFD)
         Then (search for entry with given name)
              If (directory found)
                   Then If (have access to directory)
                             Then (set new current)
                                  If (requested to set home)
                                       Then (set new home)
                             Else (insufficient access rights)
                   Else (go on to next partition)
   End  /* Do While

If (not found locally)
    Then Do (for each disk in the disk list) While (not found)
         If (disk is remote)
              Then (start remote search list)
                   Do While (next disk is on same node)
                        (next disk in list)
                        (add next disk to list)
                   (search remote system with ATLIST through R$CALL)
                   If (found)
                        Then (set up remoteness by At_adrem)
         End  /* Do While
```

ATTACH

AT_CLEAN - Common clean up for AT$ routines.

- Validates new attach point.

- Releases current attach point.

- Sets up new current (and possibly home) attach point(s)

- Allocates new unit table entry.

- Allocates dir_block to maintain records used info.

- If a quota dir, allocates quota_block to maintain quota info.

- Sets up pointers to the ACL protecting this directory.

CALCULATING ACCESS

WHO IS THIS USER?

- A user is identified via:
  a unique user_id
  a set of ACL groups the user_id is a member of

User Id:

- Stored in the process' UPCOM.

ACL Groups:

- Stored in the Active Group Table (AGT).

- A user may be a member of up to 32 ACL groups.

- All active ACL group names are stored in the AGT.

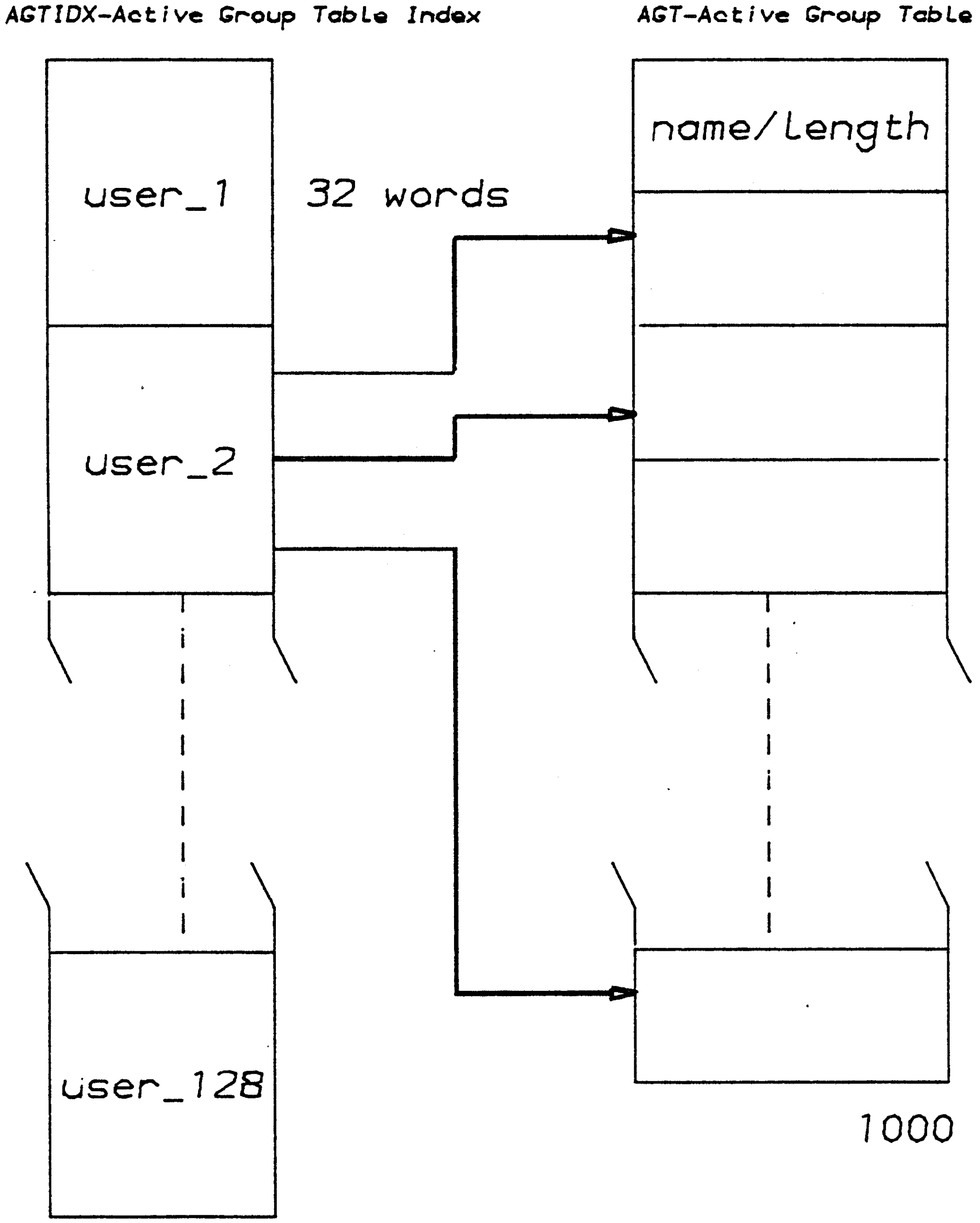
- For each user, there is a 32 word index table.

- The index table points to the names of the ACL groups that process is a member of.

ACCESS CONTROL LISTS

Data Structures

-ACL Database, Segment 37:



PRIORITY ACLS - Data Structures

- One priority ACL per ldev.

- Table of pointers to the ACL, PA_PTR.

- ACL is stored in PA_AREA.

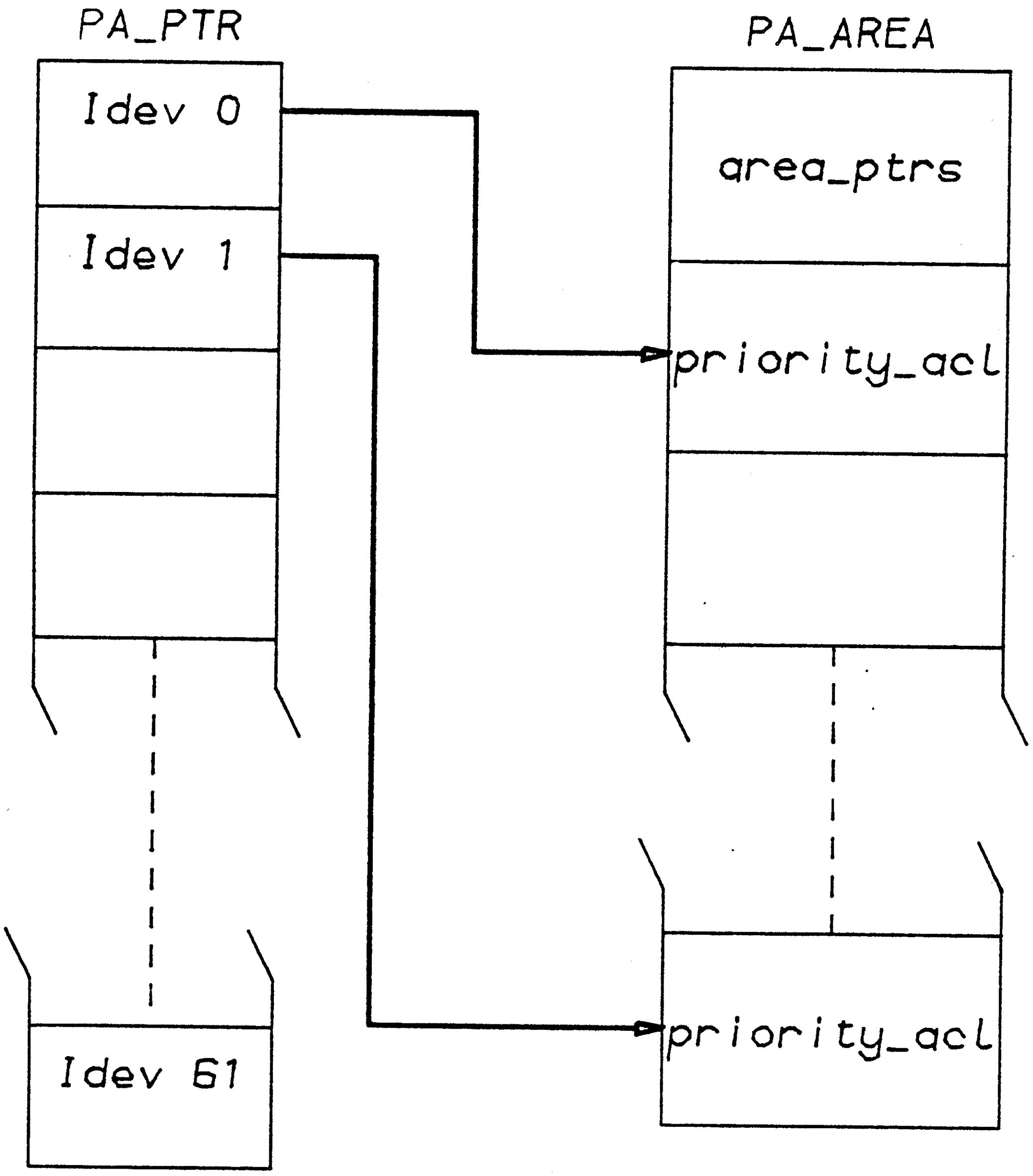
- Space is dynamically allocated/deallocated by area manager.

```
dcl 1 pacl_ based,                          /* Priority ACL (PACL)         */
      2 ecw like ecw,
      2 user_count fixed bin,               /* Number of user entries      */
      2 group_count fixed bin,              /* Number of group entries     */
      2 version fixed bin,                  /* Version no. of structure    */
      2 use_count fixed bin,                /* Number of LDEVs using this
                                               PACL (not implemented)      */
      2 group_offset fixed bin,             /* Relative position of first
                                               group entry                 */
      2 rest_accesses like accesses,        /* Rights for $REST            */
      2 rest_acc_valid bit (1) aligned,/* SET if $REST rights valid  */
      2 dtm like fsdate,                    /* Date/time created           */
      2 spare2 fixed bin,
      2 entry like coded_access;            /* like ACLs (ring1/ring3)     */
```

PRIORITY ACLS

Data Structures

-ACL Database, Segment 37:



150000

CALCULATING ACCESS

WHEN?

- During an attach operation (AT$ABS, AT$ANY, AT_CLEAN).
- During a file open operation (SRCH$$).

HOW?

- Password owner/non-owner access rights are mapped to ACL rights

Owner:	PDALU
Non-owner:	LU
Read:	R
Write:	W
Delete:	D

set According to protect Bits

```
Priority Access:   if priority_acl then
                         if user_in_pacl then
                             get access from pacl
User Id:           else if user_id_in_acl then
                             get access from acl
ACL Groups:             else if user_member_of_group(s) then
                                   get access for each member_group
                                   logical-or these accesses together
$Rest:                       else if $rest then
                                        get access from $rest pair
                                   else no access
```

Section 20 - Miscellaneous

FILE SYSTEM LOCKS

The following locks are used by the FILING system and allow a certain amount of concurrent access to the FILE system (in priority order):

FSLOK	Global file system locks
UFDLOK	UFD lock
UTLOK	Unit tables lock
TRNLOK	Transaction lock
RATLOK	Record availability lock

Each lock consists of the following data structure:

COUNTER
POINTER

READER'S Semaphore

COUNTER
POINTER

WRITER'S Semaphore

USAGE Counter

PRIORITY

Different Unit table pointers for each reader of same file But same buffer

FILE SYSTEM LOCKS

Locks will allow N readers or 1 writer.

A writer will wait on the writers semaphore if there are any active readers or an active writer.

A reader will wait on the readers semaphore if there is an active writer or if a writer is waiting.

When the USAGE counter is equal to

0	the lock is free (available)
+N	there are N active readers
-1	there is one active writer

Priority is used to force an order to avoid deadly embrace situations. In general locks are not recursively lockable and an attempt to re-lock one already locked by the calling process is disallowed. FSLOK is, however, an exception and may be recursively locked for reading only. The system maintains for each process a bitmap of the locks owned by that process. The depth of recursion for FSLOK is maintained. This information is held in PUDCOM (LOKOWN and OWNFS).

OTHER LOCKS

LOCKS (following on from file system locks in priorty order).

DEVLCK	DEVICE table in PBDIOS
SP1LCK	Spare locks
SP2LCK	
SP3LCK	
NETLCK	Network data
SLCLCK	Smlc driver data
MOVLCK	MOVUTU usage
SEGLCK	Segment tables
PAGLCK	Page tables and data bases
DSKLCK	Disk driver

PRIMOS SEGMENTS - DTAR 0

Hand out
Rev 19.2

[In Ring 0 or 3 MAp]

0	clock, i/o windows, DMx control blocks		[KS>SEG0.PMA]
1	(GEN$BUF)		
2	movutu		
3	movutu		
4	PIC, PCBs, fault handlers, checks, SEMCOM, vpsd		[KS>SEG4.PMA]
5	ring 0 gate segment	(GATSG$)	[KS>SEG5.PMA]
6	kernel code and linkage		
7	TFLIOB buffers	(TFLSN1)	
10	per-user unit tables, directory/quota blocks, usrcom		[SEG10.PMA]
11	file system code and linkage	(LCSEG$)	
12	network system code and linkage	(NETSG$)	
13	command loop and CPL code and linkage		[R3S]
14	PAGCOM, HDRBUF, config, RSAV, FIGCOM, MMAP, tape-dump, warm/cold start code		
15	additional kernel code and linkage		
21	DMQ buffers	(DMQBUF)	
22	HMAPs		
23	SMLC map segment		
24	SMLC map segment		
25	SMLC map segment		
26	SMLC map segment		

PRIMOS SEGMENTS - DTAR 0 continued

27	network buffers	(NETBF$)
30	network queues	(NETBH$)
31	network (not used)	
32	additional command loop and CPL code and linkage	[R3S]
33	LMAPs	
34	named semaphores data area	
35	logout notification queues, CPS	
36	additional TFLIOB buffers	(TFLSN2)
37	active group table, per-user group list, priority acl table	
40	unit table entries	(UTBSEG)
.		
50	associative buffers	(BUFSEG)
51	associative buffers	
52	associative buffers	
53	associative buffers	
.		
67	RJE code and linkage	
70	RJE code and linkage	
71		
.	RJE buffers	
100		

PRIMOS SEGMENTS - DTAR 0 continued

101		
.	32 network mapped segments	
140		
141	DPTX code and linkage	
142	additional DPTX code and linkage	
143		(DPTCOM)
.	DPTX buffers	
200		
201		(PUDCM$)
.	mapped per-process ring 0 stacks	
400		
401		
.	dynamically allocated by GETSN$	
477		

PRIMOS SEGMENTS - DTAR 1

```
2000
  .     shared code
  .
2030
  .     8 user segments
  .
2040
  .     shared code
  .
2170
  .     8 user segments
  .
2200
  .    shared code
  .
2300
  .     dynamically allocated by GETSN$
  .
2377
```

PRIMOS SEGMENTS

DTAR 2

4360		
.	dynamically allocated by GETSN$	
.		
4377		

DTAR3

6000	user profile stuff, UPCOM, page fault (wired ring 0) stack, SDTs for DTARS 2 and 3, mapped LOCATE buffer ('17600)	
6001	abbrevs, shared library linkage	
6002	CLDATA, ring 3 stack	(PUSTAK)
6003	unwired ring 0 stack	
6004	CPL work area	
6005	global variables	
6006	additional shared library linkage	
6007		(DYSNBG)
.	dynamically allocated by GETSN$	
6010		(DYSNED)

PRIMOS LOCKED MEMORY REQUIREMENTS

SEGNO	LOCKED
0	3KW
4	4
6	16
14	4
22	2
33	2
6000	1 (2 IF NETWORKS)

PLUS: SEG 4 100 WORDS FOR EACH CONFIGURED USER
(PCB'S AND CONCEALED STACKS)

SEG 7 TERMINAL I/O BUFFERS FOR EACH CONFIGURED USER
(DEFAULT 96 AND 192 WORDS RESPECTIVELY).

PAPER TAPE, CENTRONICS BUFFERS AS REQUESTED (1KW)

SEG 12 6K WORDS FOR MDLC
18K WORDS FOR PNC
23K WORDS FOR MDLC PNC

SEG 14 SEGMENT DESCRIPTOR TABLES (DTAR'S 0 and 1 only)
MMAP 2K WORDS FOR EACH 2MB OF PHYSICAL MEMORY

PRIMOS LOCKED MEMORY REQUIREMENTS

SEG 21 Q DATA BLOCKS FOR EACH CONFIGURED LINE (DEFAULT 32 WORDS/LINE)

SEG 22 HARDWARE PAGE MAPS, 64 WORDS FOR EACH USER SEGMENT IN USE ABOVE '1777

SEG 33 LOGICAL PAGE MAPS, 64 WORDS FOR EACH USER SEGMENT IN USE ABOVE '1777

SEG 6000 PAGE FAULT STACK, 1K WORDS FOR EACH LOGGED IN USER.

19.1 I/O ENHANCEMENTS

- New LOCATE Mechanism, NLBUF

- Balancing Primary and Alternate Paging Devices, PRATIO

- Default Value of MAXSCH, MAXSCH = (m+3) * x + y

- Reduce Active Users Working Set
  (CPLIM, LOGLIM from UPCOM to PUDCOM)

- Using Z-move Instructions

- Gate Access MOV32P, (MOVEW$)

- More Disk Queue Control Blocks (17 instead of 7)

- Hashed Transaction Locks (1 TRNLOK -to 67 LOCKRH, LOCKWH)

- No Page-in on page-aligned page-sized reads

- SEG Enhancements

- FORCEW Changes

19.1 I/O ENHANCEMENTS - Using Z-move Instructions

MOV32P moves N words of data from source to destination.

Previously, if the length specified is greater than 8 words then MOV32P would loop on: double floating loads stores, double loads stores, and single loads stores, depending on the length.

Now, for those CPUs on which the Z-move instructions are more efficient (a P750 or a P850) the ZMVD instruction is used.

MOV32P has been made available to the user from Ring 3 by adding a Gate to Seg 5. The name has been changed to MOVEW$, move words.

The calling sequence:

```
    CALL MOVEW$(ADDR(SOURCE),ADDR(DESTINATION), LENGTH)
         where LENGTH is the number of words to be moved.
```

SYSTEM LIMIT EXTENSIONS

NEW

- New INITIAL ATTACH POINT per user.
- 16 Remote_ids per user.
- 16 character login passwords.
- Maximum number of user_ids in a system or project is 7516.
- Number of DYNAMIC SEGMENTS is 148.

SEGMENTS

- Maximum value for NUSEG is now 240, due to 16 NVMFS segments.
- Number of shared segments (DTAR1) is now 192 ('2000 - '2277)
- Number of shared user segments is now 16. ('2030 - '2037, '2170 - '2177)
- Effective increase in maximum number of segments, paging space now allocated in 16KB blocks (1/8th segment) instead of 128KB (entire segment).

FILE SYSTEM

- Number of file units is now 3147
- Utilities do not convert lowercase passwords to uppercase.
- Maximum number of LOCATE buffers is 256, minimum is 8.

AREA MANAGEMENT

MOTIVATION

- Provide a single mechanism for allocating/freeing data blocks of varying sizes.

- Area manager automatically relocates blocks (if needed).

- Used for:

  CPL Variables
  CPL String Management
  Phantom Logout Notification Queues
  Priority ACLs

AREA MANAGEMENT

IMPLEMENTATION

- Uses Knuth's Boundary Tag Algorithm.

- Define an area of virtual memory to contain the data blocks.

- AR$IN to initialze the area.
  AR$ALC to allocate a block of a given size.
  AR$FRE to free a given block in an area.

- Condition 'AREA' is raised if:
  the area being initialized is too small/large
  the block being allocated is too small/large
  the area does not begin on an even word boundary
  an allocate or free request is made in an unitialized area
  the area is defective

Appendix A

Programmed Input/Output (PIO) Device Drivers

MPC-4

PROGRAMMED INPUT/OUTPUT (PIO)

1 2	3 6	7 10	11 16
zz	1100	function	device addr
	PIO	What is to be done	

	ZZ
OCP	0 0
SKS	0 1
INA	1 0
OTA	1 1

The purpose of the PIO instruction is to provide one-word Input/Output to or from a device.

1). OUTPUT CONTROL PULSE (OCP)

The OCP instruction normally performs a control function within the selected device control unit. These control functions are mandatory for such purposes as:

A). Starting a clock

B). Forcing an Input-only mode

C). Initializing a device (Device Master Clear)

PROGRAMMED INPUT/OUTPUT (PIO)

2). SKIP ON CONDITION SATISFIED (SKS)

The SKS instruction tests a condition on the selected device and if the condition is TRUE, skips the next instruction.

e.g. Skip if ready to input/output a character

3). INPUT TO REGISTER A (INA)

The INA instruction will input one word into Register A from the specified device (if the device is ready). If the operation is successful, the next instruction is skipped. The word may contain only one byte of valid data. In these cases the INA will input the byte into the least significant eight bits of Register A and leave the more significant byte of Register A unaltered.

4). OUTPUT FROM REGISTER A (OTA)

The OTA instruction will output the contents of registr A to the selected device if that device is ready to accept the data. If the output operatin is successful, the next instruction is skipped. The A register may contain only one byte of valid data.

PROGRAMMED INPUT/OUTPUT (PIO)

The FUNCTION CODE further defines the purpose of a PIO instruction.

OCP Function Code indicates control operation.

SKS Function Code indicates that a condition is to be tested.

OTA Function Code selects destination for word from Register A.

INA Function Code selects source of data word into Register A.

DEVICE

The 6 bit device number selects one of the 64 possible devices. The PIO instruction is recognized by the device with the selected address. Normally each control unit has a unique address but some respond to as many as four device addresses.

NOTE: The OCP, SKS, OTA, and INA instructions are restricted and are available only in R and S modes.

The EIO instruction (used in V mode) replaces the PIO instructions.

The effective address of the EIO is executed as one of the PIO instructions. EIO is a restricted instruction and sets the condition codes to indicate the success or failure of the specified operation. The EIO should be followed by a BCNE *-2 instruction. The EIO instruction is always two words long and never skips.

DEVICE DRIVERS

PRINCIPLES INVOLVED IN WRITING DRIVERS

1). ASSIGN/UNASSIGN Logic

A). Add device name to DEVCOM

B). Fix table sizes and indices

2). INITIALIZATION ROUTINE (Cold Start?)

A). Lock driver and buffers

B). Turn on the device

3). USER INTERFACE

A). Add SVC front end

B). Fix SVC class tables

C). Add direct entrance call (seg 5)

4). VALIDITY CHECKS

A). Assigned

B). Authorized user

C). Initialization of device

DEVICE DRIVERS

5). I/O CONSIDERATIONS

A). DMA, DMC, DMQ, DMT

B). DMX channel assignment

C). Buffer allocation - Mapped or not

D). Interrupt Phantom in seg 4 - Communication with driver

6). STRUCTURE

A). Separate process - synchronous or asynchronous with user execution.

B). Need for buffering.

7). WARM START REQUIREMENTS.

A). Initialization

B). PABORT logic

8). I/O COMPLETION

A). Unmap I/O

B). Release locks

C). Release user

EXAMPLE DRIVER (MTDIM)

(called by user and runs as part of the user's process)

1). Validate unit number
2). Validated user, if not same as present wait on semaphore
3). Lock controller if serial reusable
4). Set up information for phantom interrupt code.
5). Initialize controller if not already done.
6). Validate arguments.
7). Set up DMA/DMC channels
8). Call MAPIO
9). Start up operation
10). Return to user.

INTERRUPT PHANTOM

1). Reset mask
2). Set MTDONE abort flag
3). Notify other users if waiting on controller lock semaphore.

MTDONE

1). Called from PABORT
2). Get controller status
3). Return information to user
4). Call UMAPIO
5). Notify same user if waiting on MAG TAPE semaphore
6). Return to PABORT

MPC4 SUPPORT

MOTIVATION

- Provides a standard PIO/DMx interfacing mechanism.

- Device independent driver in Primos (ring 0), T$GPPI/GPIDIM.

- Device dependent code in ring 3, Primos rev independent.

IMPLEMENTATION

- MPC4 is a hardware implementation of the GPPI concept.

- User microcodable controller:
  Microcode maintained in ROM, or
  Downloaded to RAM from disk at system coldstart.

- Primos support for two MPC4 controllers, addresses '75 and '76.

- Each controller can support up to four devices.

MPC4 - General Purpose Parallel Interface

FUNCTIONS

1 - Read block
2 - Write block
3 - Read word
4 - Write word
5 - Wait/poll for interrupt
6 - Load interrupt mask register
7 - Load communication region address register
8 - Execute device-dependent OTA
9 - Reset device
10 - Load device timeout register
11 - Release communication region
:100001 - Execute OCP. (Restricted)
:100002 - Execute SKS. (Restricted)
:100003 - Execute INA. (Restricted)
:100004 - Execute OTA. (Restricted)

MPC4 - General Purpose Parallel Interface

USER CODE

- Assign/Unassign logic. (GPIONF)

  Assign device GPn, n = 0..7
  Wires GPIDIM.
  Initializes controller status.

- Subroutine interface to DIM, T$GPPI.

  Builds a unit data block (UDB).
  Notifies GPIDIM to process it.

MPC4 - General Purpose Parallel Interface

PRIMOS CODE

- Initialization code. (GPIINI)

  Check for controller and verify it.
  Loads microcode.
  Sets up controller data block (CDB).
  Allocate segment 0 i/o windows.

- Device Interrupt Manager. (GPIDIM)

  Notified by T$GPPI and PIC.
  Performs tasks specified by UDBs.

- Warm Start Code. (GP1PBW)

  Re-initializes controller status.
  Cleans up any DMA transfers in progress.
  Fixes up UDB servicing.

Appendix B - Process Exchange

DATE: March 29, 1976 PE-T-232

TO: Programming and Engineering Staff

FROM: M. L. Grubin

SUBJECT: P-400 PROCESS EXCHANGE AND NEW PROTOCOLS

I. Process Exchange

A. Data Bases
1. Ready List
2. WAIT Lists
3. Process Control Block (PCB)

B. Instruction Primitives
1. WAIT
2. NOTIFY

C. Dispatcher and Register File Management
1. Ready List Maintenance
2. Register Set Assignment
3. Fetch Cycle Trap

II. Traps, Interrupts, Faults, Checks

A. External Interrupts
1. Operation
2. Special Instructions (IRTN, INOTIFY)

B. Faults
1. Data Bases
2. CALF
3. Fault Handler

C. Checks

III. Register Files

IV. Control Panel

V. CP Timer

I. PROCESS EXCHANGE

The Process Exchange mechanism is composed of three data bases and two basic instruction primitives. The data bases are the ready list, wait lists, and Process Control Blocks (PCB). The basic instruction primitives are WAIT and NOTIFY. In addition, there is an independent mechanism for controlling the usage of two register sets which is related to, but not part of, the ready list data base.

A. Data Bases

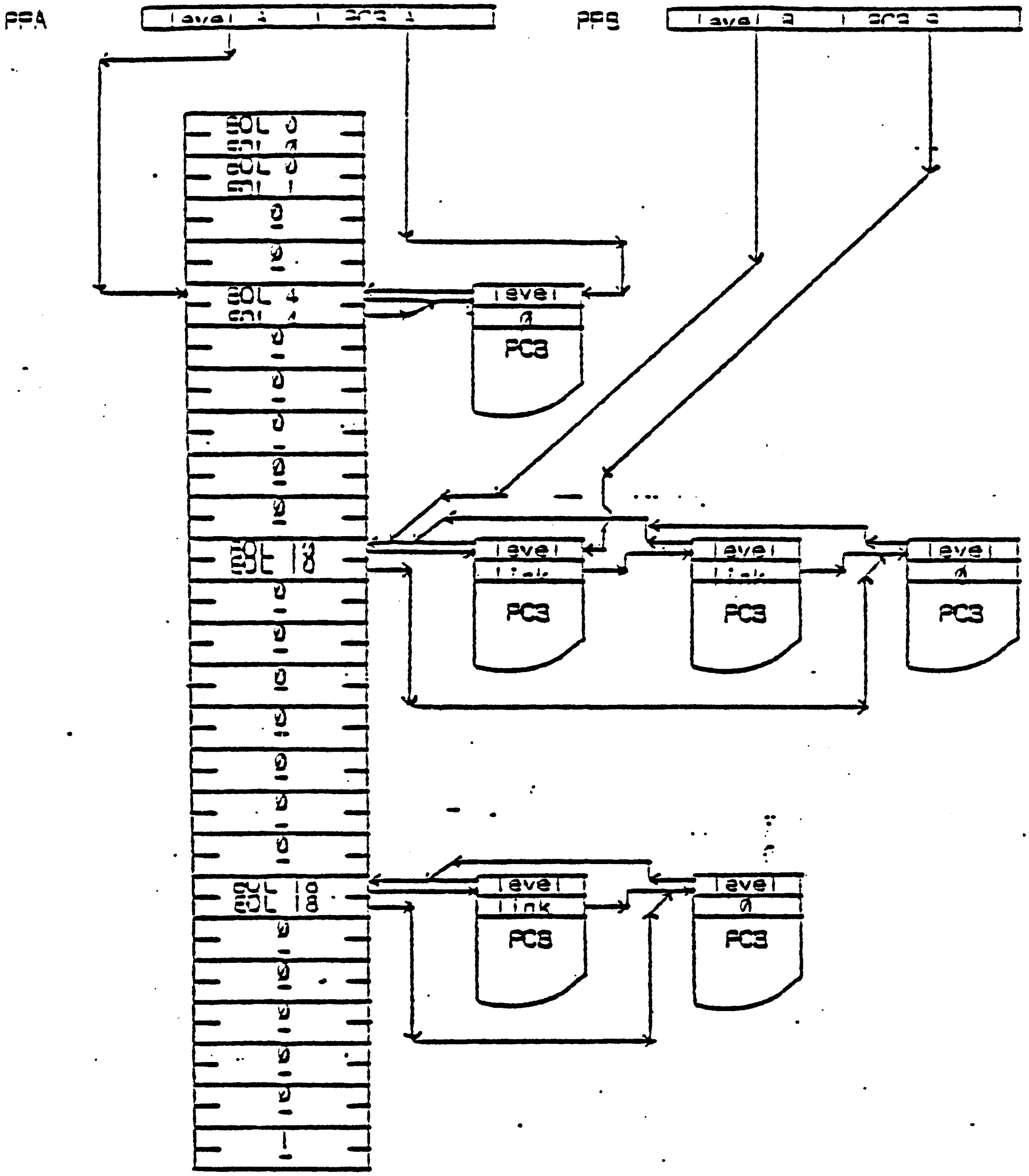
1. Ready List

The ready list is a two-dimensional list structure used for priority scheduling and dispatching of processes. The entire ready list data base (excluding live registers) and all PCB's are contained in a single segment. The segment number of this segment is contained in a 16-bit register called OWNERH. Within the segment, all pointers and addresses (except fault vectors and wait list pointers) are 16-bit word number quantities.

The two-dimensionality of the ready list is achieved with a linear array of list headers for each priority level composed of a Beginning of List (BOL) pointer and an End of List (EOL) pointer.

Each pointer is the 16-bit word number address of a PCB (in the same segment as the ready list). The PCB's on each priority level list are forward-threaded through a 16-bit link word, and as many PCB's as desired can be threaded together on each priority level to form the ready list. A process' priority level is both determined by and encoded as the address of a BOL pointer in the ready list. Priority order is determined by arithmetic comparison, i.e., smaller numbers (addresses) are higher priorities. As a result, priority level list headers must be allocated in contiguous memory at system startup time.

The end of the ready list is determined by a BOL containing a 1 (PCB addresses must be even). An empty level is indicated by a BOL containing 0. Figure 1 is a picture of the ready list structure. The 32-bit registers PPA (Pointer to Process A) and PPB (Pointer to Process B) are a speed-up mechanism for locating the next process to dispatch. PPA always contains both the level (BOL pointer) and PCB address (designated level A and PCBA) of the currently active process. PPB points to the NEXT process to be run when process A 'goes away'. PPA not only points to the currently active process, but, by definition, level A is the highest level in the system. It is possible for PPB and PPA to be 'invalid'. This condition is indicated by a PCB address of



Ready List: All pointers are 16-bit word number pointers within the PCB segment. The segment number is contained in the high portion of the OWNER pointer within each register set.

All PCB start addresses must be even (bit 16 = Ø). The end of the ready list is marked with a EOL entry = 1.

FIGURE 1.

0. It is important NOT to disturb the level portions, especially level A since, even if invalid, level A indicates the highest level that WAS in the system and therefore determines where in the ready list to begin a scan, if necessary (PPB invalid), for the next process to run. In a completely idle system, both PPA and PPB will be invalid and, upon completion of the ready list scan, the u-code will go into a 'wait for interrupt' loop.

It is important to notice that there is no word number pointer to the first priority level in the ready list. The ready list allocator, which starts the process exchange mechanism, knows where the list begins and, during execution, level A (in PPA) will always point to either the highest level currently in the system or the last known highest level and, hence, acts as an effective ready list begin pointer. In addition, level B will always be higher than the second level to run. That is, a PCB can never be on a level higher than level B unless it is the only PCB higher than level B (i.e., level A).

Two 'queuing' algorithms will be implemented for the ready list, either FIFO or LIFO queuing.

2. WAIT Lists

Every PCB in the system will always be somewhere. If it is not on the ready list, then, by definition, it will be on a wait list. A wait list is defined by a 32-bit semaphore consisting of a 16-bit counter (C) and a 16-bit word number BOL pointer. Since the ready list and all PCB's reside in one segment (OWNERH), and only PCB's go onto wait lists, a segment number is not needed in the semaphore. However, semaphores themselves can be anywhere and, in general, are NOT in the PCB segment. The structure of a wait list is shown in Figure 2. Notice that the last block on the wait list contains a 0 link word. Notice also that the semaphore contains only a BOL pointer.

The 'queuing' algorithm for wait lists is process priority queuing. That is, the priority level of a PCB will determine where on the wait list the PCB will be queued. For PCB's of equal priority, the algorithm becomes FIFO.

3. Process Control Block (PCB)

The contents of the PCB are shown in Figure 3. The PCB can be broken into the following logical sections which are ordered as shown:

WAIT LIST STRUCTURE

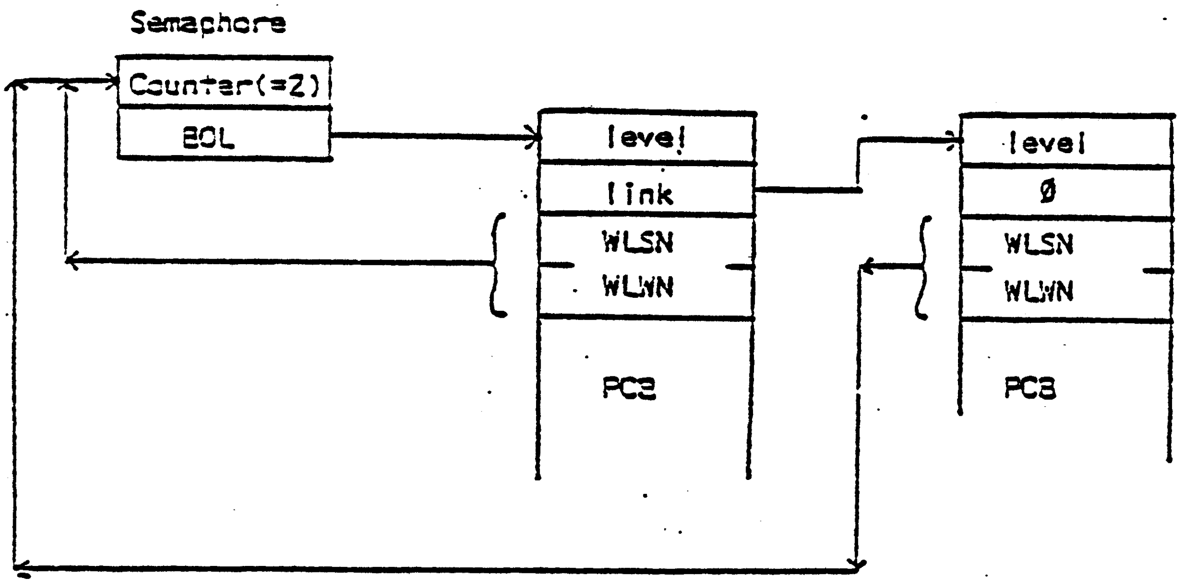


Figure 2.

Process Control Block (PCB)

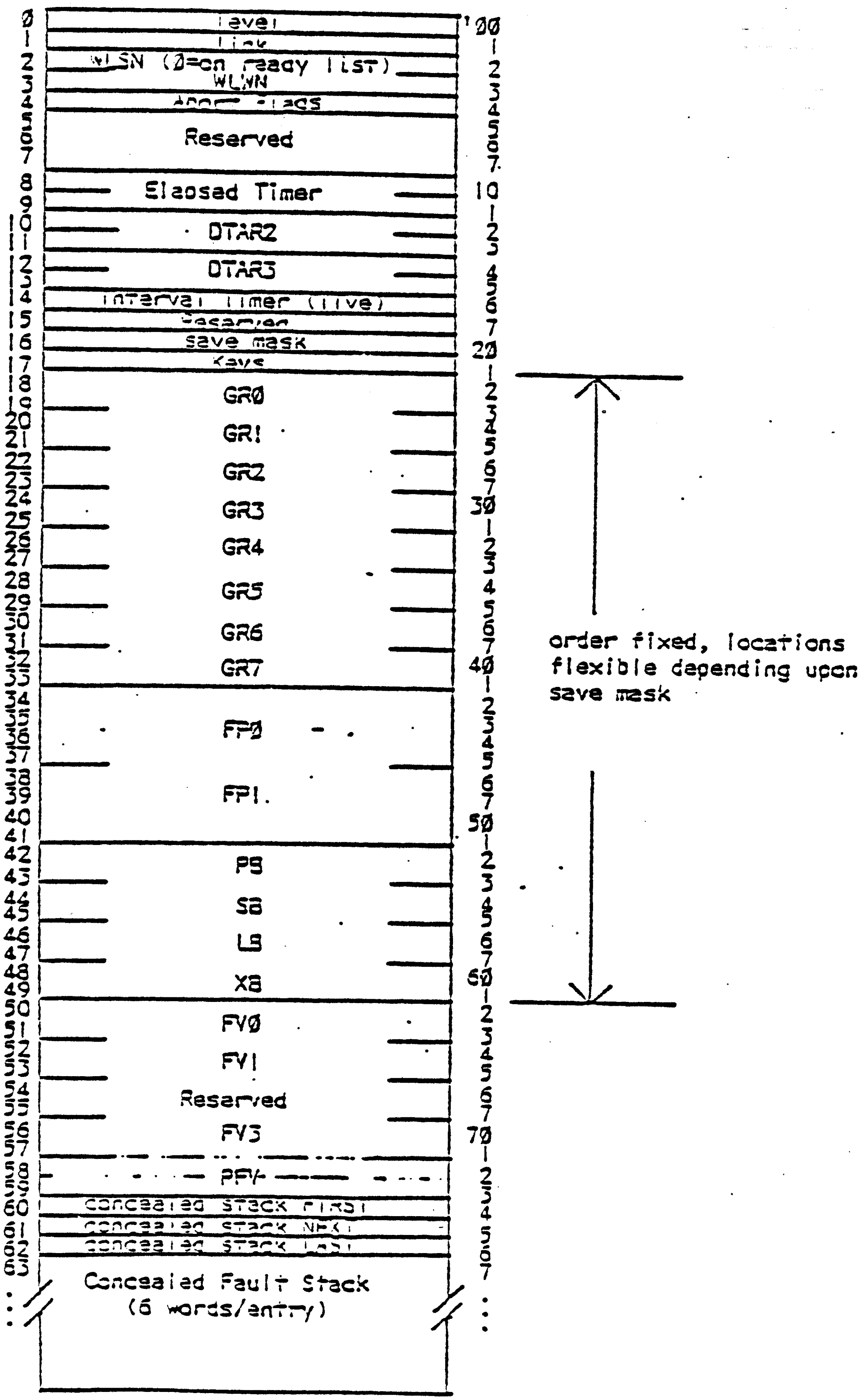


Figure 3.

a. Control

0 - level (pointer to BOL in ready list)
1 - link (pointer to next PCB or 0)
2,3 - SN/WN of Wait List this block is currently on (SN=0 indicates on ready list)
4 - abort flags used to generate Process Fault when PCB is dispatched.

Current bit assignments 1-15: MBZ
16: process i
nterval
timer ove
rflow

5,7 - reserved

b. Process State

8,9 - Process elapsed timers (must be maintained by software that resets the interval timer)
10,13 - DTAR2 and DTAR3 (never saved, always restored)
14 - Process Interval Timer with 1.024 msec resolution
15 - Reserved
16 - Save mask - used to avoid saving and restoring registers = 0

Bits 1- 8: GR0-GR7 (2 words each)
9-12: FP0-FP1 (4 registers, 2 words each)
13-16: Base
Registers(PB,SB,LB,XB)

17 - Keys
18,33 - GR0-GR7
34,41 - FP0-FP1
42,49 - Base Registers (PB,SB,LB,XB)

Note that although all the registers are assigned locations within the PCB, only non-zero registers will actually be saved which results in a compacted list which can only be determined by the bits in the save mask. In general, the saved registers (those not equal to 0) will be between words 18 and 49. The order of the registers, however, is fixed as above.

c. Fault (See section on Faults for a description of the use of this portion of the PCB)

50,59 - Fault Vectors: SN/WN pointers to fault tables for Ring 0, Ring 1, Page Fault and Ring 3 fault handlers
60,62 - Concealed Fault Stack Header
63,.. - Concealed Stack - 6 word entries. (This stack need not start at word 63).

B. Instruction Primitives

There are two basic instruction primitives for the process exchange mechanism: NOTIFY and WAIT. In addition, NOTIFY has two variants. These instructions, similar to Djikstra's P and V operators, are essentially 'interlock' mechanisms. These instructions are three-word (48-bit) 'instructions' as follows:

Instruction (16-bit universal generic)
32-bit pointer to semaphore address

As suggested by the names, WAIT is used to wait for an event (CP, time, unit record device available, whatever) and NOTIFY is used to signal that an event has occurred. In particular, a WAIT is used to wait for a NOTIFY and a NOTIFY is used to alert a process which is waiting.

Coordination is achieved by means of a semaphore containing a counter and a BOL pointer. The semaphore and the PCB's waiting for the event of that semaphore constitute a wait list. The counter, if greater than 0, indicates the number of PCB's on the wait list. If negative, it indicates the number of processes that can obtain the resource. Semaphores fall into two categories: public and private. A public semaphore is used to coordinate several processes together or control a system resource. Private semaphores are used by a single process to coordinate its own activities. For example, if a disk request is made, a process will wait on a private semaphore for the disk operation to complete. The disk process will then notify the semaphore upon completion. The distinguishing characteristics of a private semaphore is that only 1 PCB can ever be on that wait list. A public semaphore can have many different PCB's on its wait list.

1. WAIT

The operation of wait is as follows: the semaphore counter is incremented and, if greater than 0, (resource not available/event has not occurred), the PCB is removed from the ready list and added to the specified wait list. If the counter is less than or equal to 0, the process continues. If the PCB is put on the wait list, the general registers are NOT saved and the register set is made available. Therefore, a process can NEVER depend on the general registers being intact after a WAIT. In fact, from the point of view of an executing process, a WAIT appears as a NOP which destroys the registers. In addition, WAIT will turn off the process timer. Figure 4 is a detailed flow chart of the WAIT instruction.

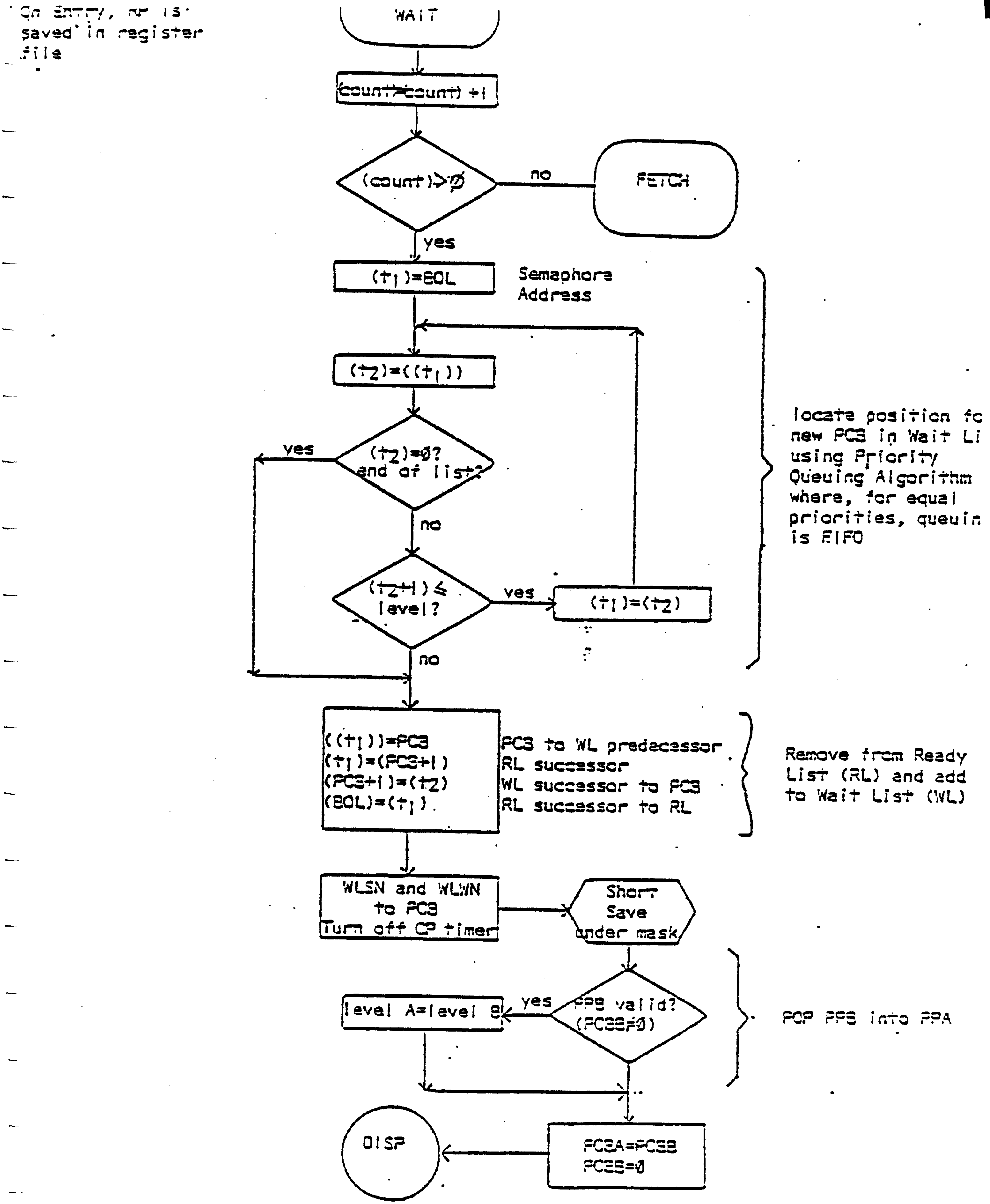

On Entry, AP is saved in register file
WAIT
(count)=(count)+1
(count)>∅
no
FETCH
yes
(t1)=BOL
Semaphore Address
(t2)=((t1))
yes
(t2)=∅? end of list?
no
(t2+1)≤ level?
yes
(t1)=(t2)
no
locate position fo new PCB in Wait Li using Priority Queuing Algorithm where, for equal priorities, queuin is FIFO
((t1))=PCB
(t1)=(PCB+1)
(PCB+1)=(t2)
(BOL)=(t1)
PCB to WL predecessor
RL successor
WL successor to PCB
RL successor to RL
Remove from Ready List (RL) and add to Wait List (WL)
WLSN and WLWN to PCB
Turn off CP timer
Short Save under mask
PPB valid? (PCBB≠∅)
yes
level A=level B
POP PPB into PPA
DISP
PCBA=PCBB
PCBB=∅

Figure 4.

2. NOTIFY

The NOTIFY instruction has two flavors:

NFYE: use FIFO queuing op code Bit 16 = 0
NFYB: use LIFO queuing op code Bit 16 = 1

The instructions differ ONLY in the ready list queuing algorithm used. The operation of NOTIFY is as follows: the semaphore counter is decremented and the notifying process continues. If the counter is less than 0, no action is taken, but if greater than or equal to 0, a PCB is removed from the top of the wait list and added to the ready list. No explicit action is ever taken on the notifying process, only the notified semaphore. If a notified process is of higher priority than the notifying process, the latter will be effectively 'interrupted', but will remain on the ready list. Figure 5 is a detailed flow chart of the NOTIFY instruction.

C. Dispatcher and Register File Management

The dispatcher is the root of the process exchange mechanism and is responsible for determining the next process to run (be dispatched), and assigning that process a register set. There is considerable overlap with NOTIFY and WAIT in functionality related to maintaining the ready list. For example, both NOTIFY and WAIT update PPA and PPB as appropriate, but the dispatcher scans the ready list if PPA is invalid. Register file management, including any necessary saves and restores, are the sole province of the dispatcher. Figures 6 and 7 are detailed flow charts of the dispatcher.

1. Ready List Maintenance

Upon entry, the dispatcher first asks if PPA is valid. If it is, the process is assigned a register set and dispatched. If PPA is not valid, a scan of the ready list is initiated. If a PCB is found, PPA is adjusted and the process dispatched. If the ready list is empty, the dispatcher idles. Whenever a process is dispatched the process timer is turned on.

2. Register Set Assignment

In each register set, a register, designated OWNER, contains a pointer to the PCB of the process which owns the set. OWNER is a full 32-bit pointer and OWNERH is used throughout the system to determine the segment number of the ready list and PCB's. Obviously, OWNERH must be the same in both

On Entry, RP is saved in register file

OP code Bit 16 = Ø end
1 beginnin

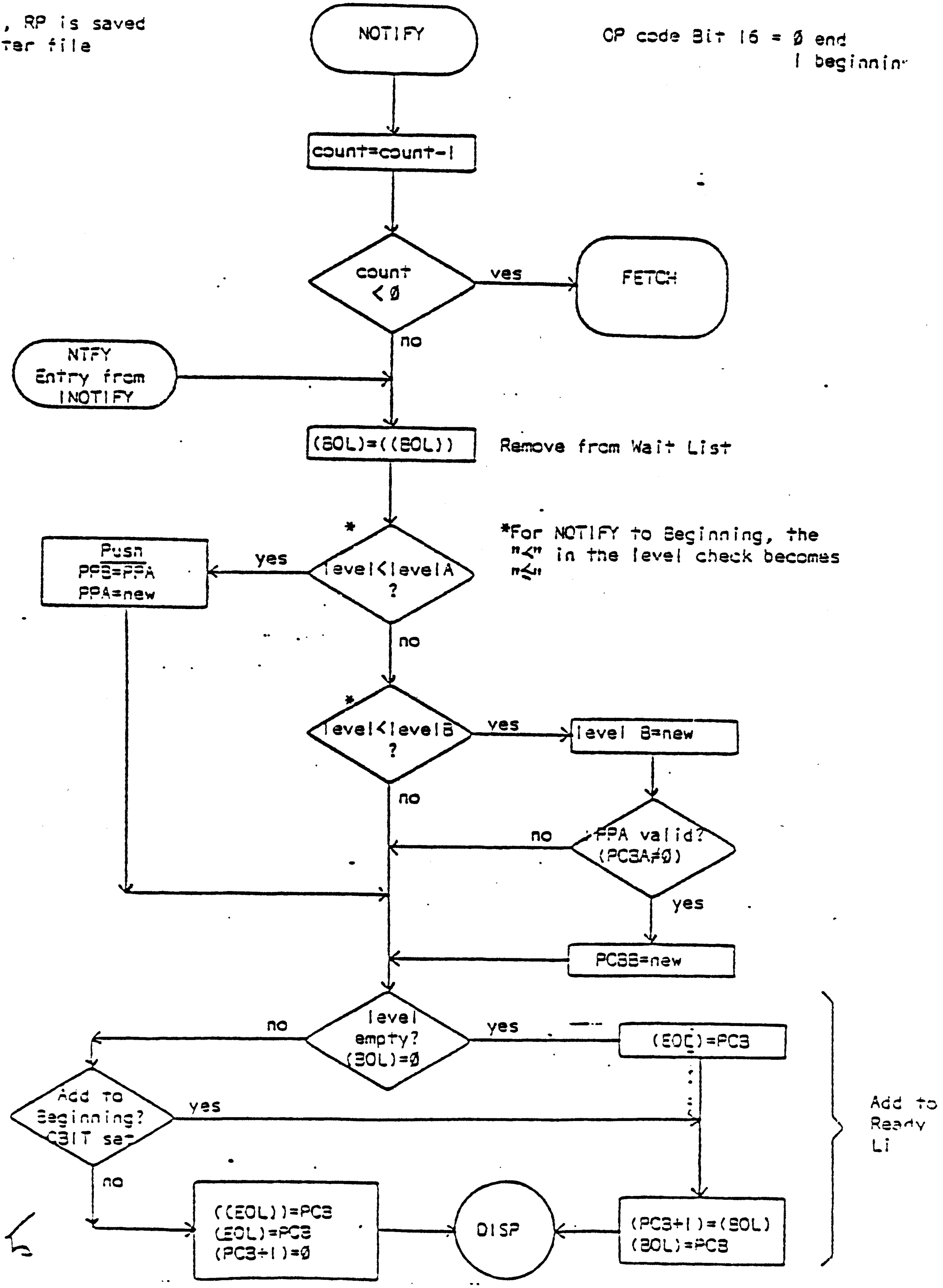


Fig. 5

On Entry: RB and KEYS registers are valid and RP and live keys are invalid

Note: All interrupt aks result in a return the top of the dispatcher

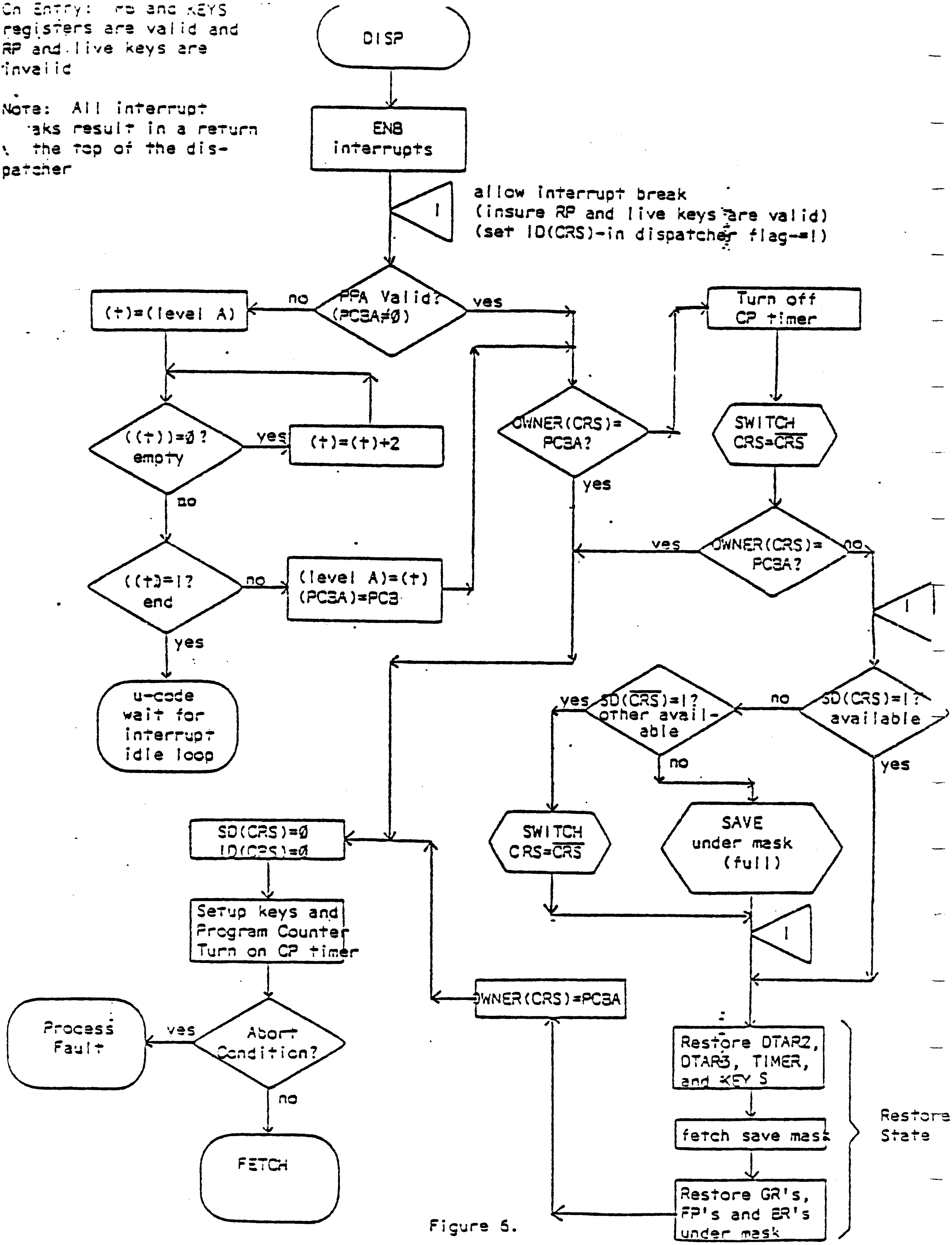


Figure 5.

*The registers to be saved are a parameter passed as a starting RF address in (TRØ,L)

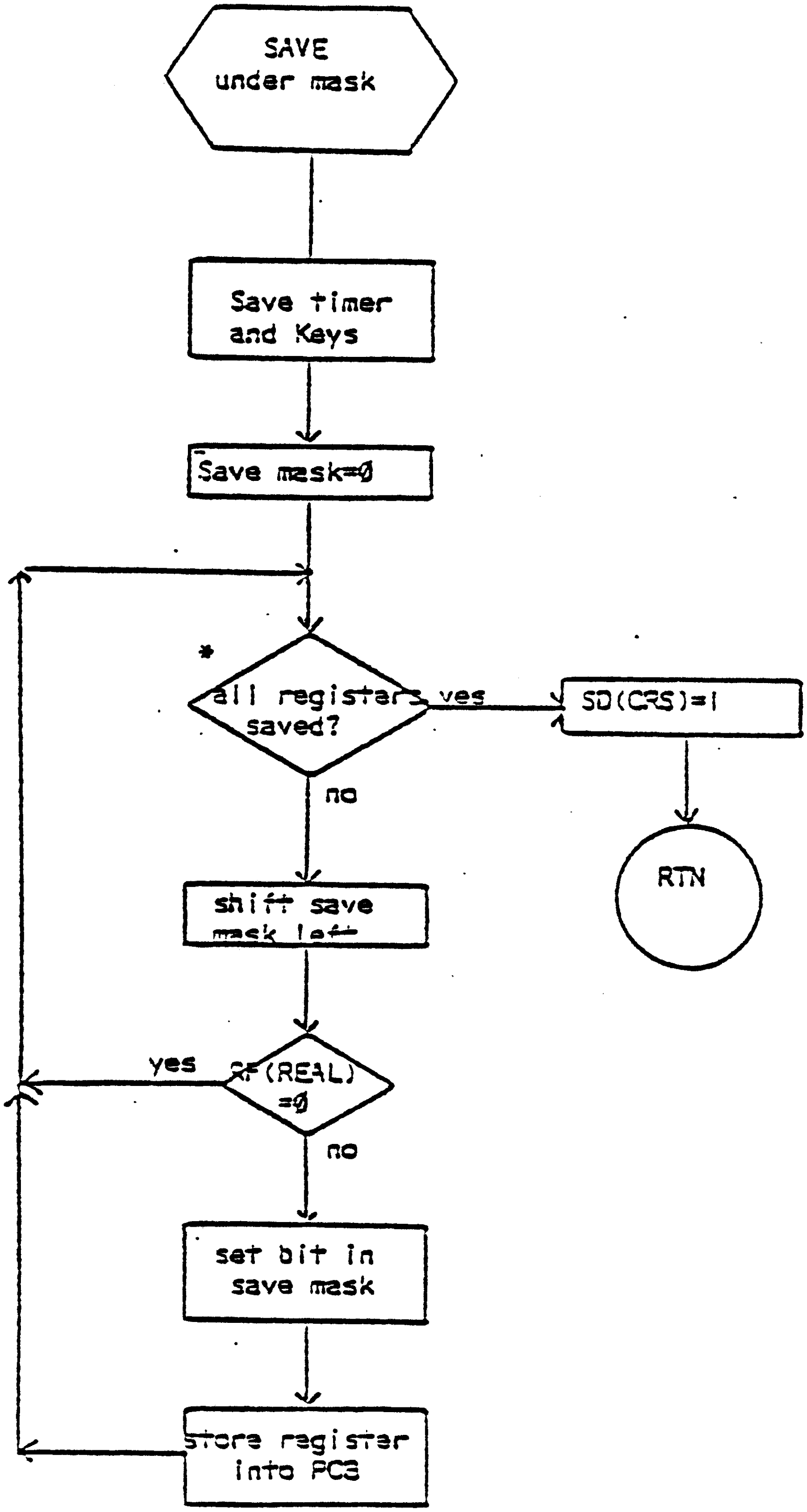


Figure 7.

register sets. In addition, the low order bit of the keys register (KEYSH) is used to indicate whether the register set is available. The bit is called the Save Done bit and, if set, indicates that the register set and its copy in the owner's PCB are identical (a save has been done). This bit is set by the save routine (called from WAIT or the dispatcher) and reset when a process is dispatched. Whether a register set is available (SD=1) or not, it is always owned. Therefore, if a process goes away (either as a result of a WAIT or the notification of a higher level process) and comes back again immediately and, if that process still owns the register set, a restore operation is not necessary. If a register set switch is necessary, the process timer is turned off. The details of selecting which register set to assign to a process being dispatched is shown on the right of Figure 6. The dispatcher is the only code which switches register sets.

3. Fetch Cycle Trap

At various points in the dispatcher (indicated by I on the flow chart) a check for interrupt pending (fetch cycle trap) is made. As a result, interrupts can occur either in the fetch cycle or in the dispatcher. The possible Fetch Cycle traps are:

1. External Interrupt (See Part II-A)
2. CP-timer increment and possible overflow (See Part V)
3. Power Failure (See Part II-C)
4. Halt switch on control panel (See Part IV)
5. End-of-Instruction Trap

The end-of-instruction trap occurs either from an ECC corrected error or from a missing memory module, memory parity, or machine check during I/O. In all cases, if the check handling software returns (via LPSW instruction), the possible destinations are either the fetch cycle or the dispatcher (PB in PSW not a real program counter). In order to guarantee the proper destination, bit 15 of the keys (KEYSH) is used to indicate if the trap was detected by the dispatcher (bit 15=1). This bit is set by the dispatcher upon detecting a trap and is reset when a process is actually dispatched (return to fetch cycle).

II. TRAPS, INTERRUPTS, FAULTS, CHECKS

Four words used frequently are 'trap', 'interrupt' (or 'external interrupt'), 'fault', and 'check'. The meanings of these terms are carefully distinguished for the P-400/500. Software breaks in execution are divided into three main categories referred to as 'interrupts', 'faults', and 'checks'. The word 'trap', on the other hand, refers to a break in execution flow on the u-code level.

Traps can occur for many reasons, not all of which cause software visible action, and are always processed on the u-code level. Some traps may directly or indirectly cause breaks in software execution, but not all software breaks are the result of a trap.

On the PRIME 300, interrupts, faults, and checks used the same protocol to get to their respective software handlers, namely they caused a vector through a dedicated sector 0 location (JST* vector). On the P-400/500, when process exchange mode is enabled, the three categories use different protocols both from the P-300 and each other. Roughly, the three terms are used to describe:

1. Interrupt - a signal has been received from a device in the external world (including clocks) indicating that the device either needs to be serviced or has completed an operation. In general, an interrupt is not the result of an operation initiated by the currently executing software and will not be processed by that software (though, of course, it may).

2. Fault - a condition has been detected that requires software intervention as a direct result of the currently executing software. In general, faults can be handled by the current software, though in many cases common supervisor code within the current process handles the fault. Also, in general, an external device in the real world is not directly involved in either the cause or cure of a fault condition. Often, however, external devices are involved indirectly as, for example, in performing a page turn operation as a result of a page fault.

3. Check - an internal CP consistency problem has been detected which requires software intervention. The condition could be either an integrity violation, reference to a memory module which does not exist, or a power failure. By contrast, a reference to a page which is not resident or an arithmetic operation which causes an exception is a FAULT condition.

A. External Interrupts

1. Operation

External Interrupts operate in either of two modes depending upon whether process exchange is turned on. If process exchange is off, all interrupts are treated as P-300 interrupts. In all cases, except memory increment, the address presented by the controller (or '63 if in standard interrupt mode) is used as the address in segment 0 of a 16-bit vector. This vector, in turn, points to interrupt response code (IRC), also in segment 0, which is entered via a simulated JST* through the vector. Thus, the current P-counter (RPL) is stored in (vector) and execution begins at location (vector) +1 with interrupts inhibited, but with no other keys or modals changed. If in vectored interrupt mode, it is the responsibility of the software to do a CAI. In either mode, the full RP is saved in the register PSWPB.

If process exchange mode is on, an entirely different mechanism operates. In all cases, except memory increment, the address presented by the controller is used as a 16-bit word number offset into the interrupt segment (#4). This segment is guaranteed to be in memory, but STLB misses may occur. The current PB (actually RP) and KEYS (keys and modals) are saved in the u-code scratch registers PSWPB and PSWKEYS. The machine is then inhibited and the IRC begins execution in 64V mode. It is the responsibility of the IRC to issue a CAI. It is important to note that the IRC in the interrupt segment does not belong to any process. PPA points to the PCB of the interrupted process and, in fact, no PCB exists for the IRC. Also, except for PB and KEYS, no registers are saved. In fact, even PSWPB and PSWKEYS are in the register file and not in memory. As a result, the IRC cannot do an enable and must return to the process exchange mechanism (i.e., the dispatcher) as soon as possible. Because of all these restrictions on what the immediate IRC can do, as well as the fact that it does not belong to any process, it is referred to as phantom interrupt code. Unless the job of servicing an interrupt is very simple, phantom interrupt code can do little more than turn off the controller's interrupt mask, issue a CAI, and NOTIFY the real IRC.

A memory increment interrupt is handled the same regardless of the state of process exchange. The address presented by the controller is used as the 16-bit word number in segment 0 (I/O segment) of a 16-bit memory cell to be incremented. If the counter does not overflow (-1->0), the u-code simply returns. With process exchange off, the return is always to the fetch cycle. With process exchange on, the return is either to the fetch cycle or the dispatcher, depending upon where the interrupt was detected. When detecting an interrupt, the dispatcher always insures that RP=PB and that all live keys=KEYS. If memory increment returns, it does so to the top of the dispatcher without having touched PB or KEYS. In this way, memory increment is guaranteed not to destroy any vital information needed by the dispatcher. If the memory cell counter does overflow, an End-of-Range interrupt is generated and then memory increment returns. The subsequent EOR interrupt will then be treated like any other external interrupt. Figure 8 is a detailed flow chart of the external interrupt handler.

2. Special Instructions (IRTN, INOTIFY)

Phantom interrupt code has two options for the actions it can take. If the servicing required by the interrupt is very simple, phantom code can completely process the interrupt and return to the dispatcher. If the servicing required is more complex, the phantom code must turn off the controller's interrupt mask and NOTIFY the remainder of the IRC. In the first case, PB and KEYS must be restored from PSWPB and PSWKEYS and then the dispatcher must be entered directly. Since PB cannot be restored in phantom code (the P-counter will point to the instruction in phantom code) and the dispatcher cannot be entered directly (no such instruction exists), the special instruction, IRTN, a 16-bit generic, is executed to perform these functions. After entering the dispatcher via an IRTN, the dispatcher does not know that an interrupt occurred.

In order to NOTIFY a process, phantom code must insure that PB and KEYS are restored before issuing the NOTIFY. The special instruction, INOTIFY, performs the restore and then does the NOTIFY. As NOTIFY, INOTIFY is a three-word generic with two flavors, INOTIFYB and INOTIFYE where the beginning of list option has bit 16=1 and the end of list option has bit 16=0 in the opcode.

Phantom Interrupt code can issue a CAI in one of two ways. Either an explicit CAI instruction may be issued or the IRTN/INOTIFY instructions can issue it. Bit 15 of the IRTN/INOTIFY instructions is interpreted as follows:

```
Bit 15 = 0 do not issue CAI
         1 issue CAI
```

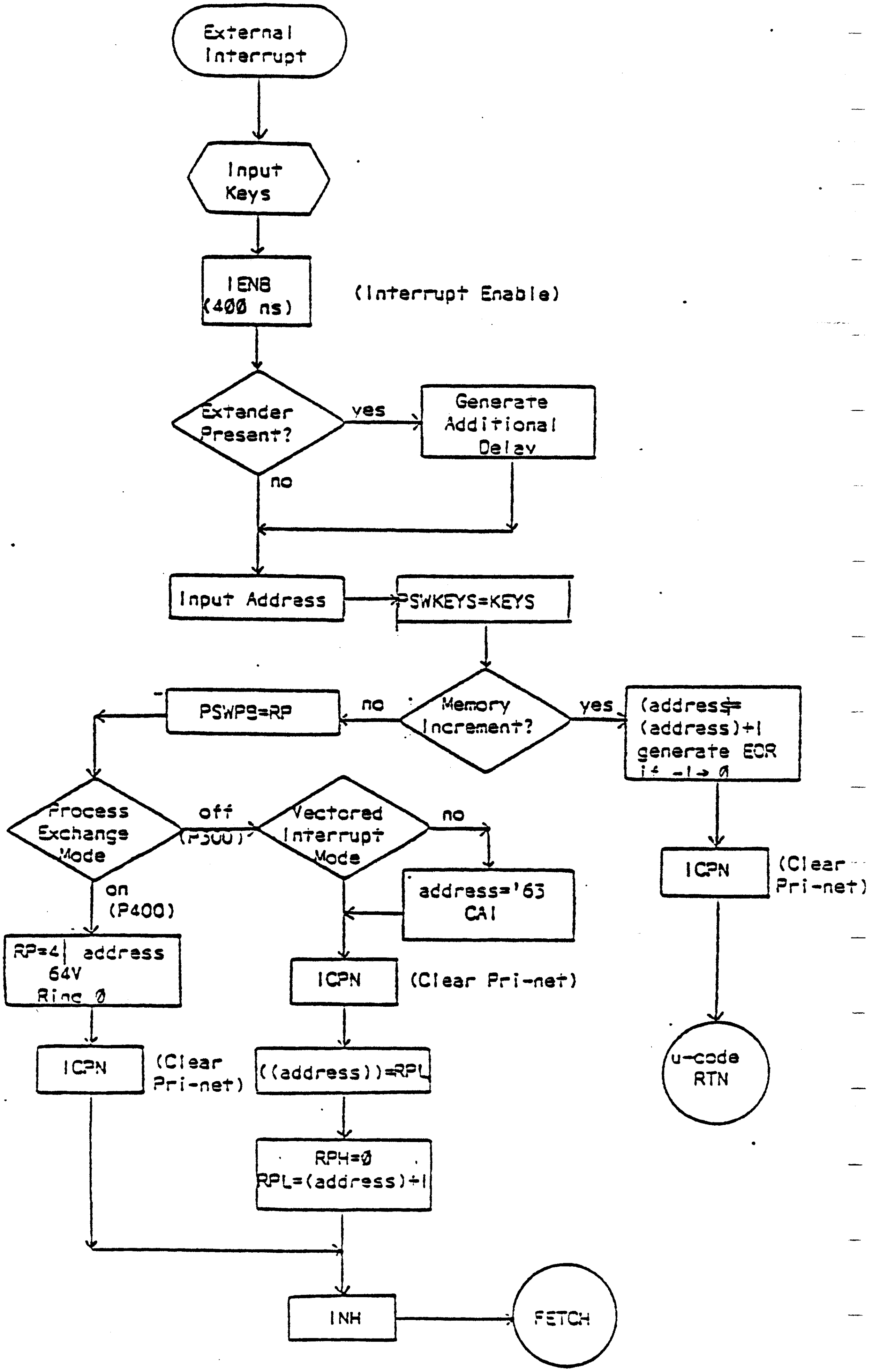

External Interrupt
Input Keys
IENB (400 ns)
(Interrupt Enable)
Extender Present?
yes
Generate Additional Delay
no
Input Address
PSWKEYS=KEYS
Memory Increment?
no
yes
PSWPB=RP
(address)=
(address)+1
generate EOR
if -1→0
Process Exchange Mode
off
(P300)
Vectored Interrupt Mode
no
on
(P400)
address='63
CAI
RP=4 address
64V
Ring 0
ICPN
(Clear Pri-net)
ICPN
(Clear Pri-net)
ICPN
(Clear Pri-net)
((address))=RPL
RPH=0
RPL=(address)+1
u-code RTN
INH
FETCH

Figure 8.

In all, there are four INOTIFY instructions as follows:

Name	Bit 15	16	Function
INEC	1	0	End + CAI
INEN	0	0	End + no CAI
INBC	1	1	Beginning + CAI
INBN	0	1	Beginning + no CAI

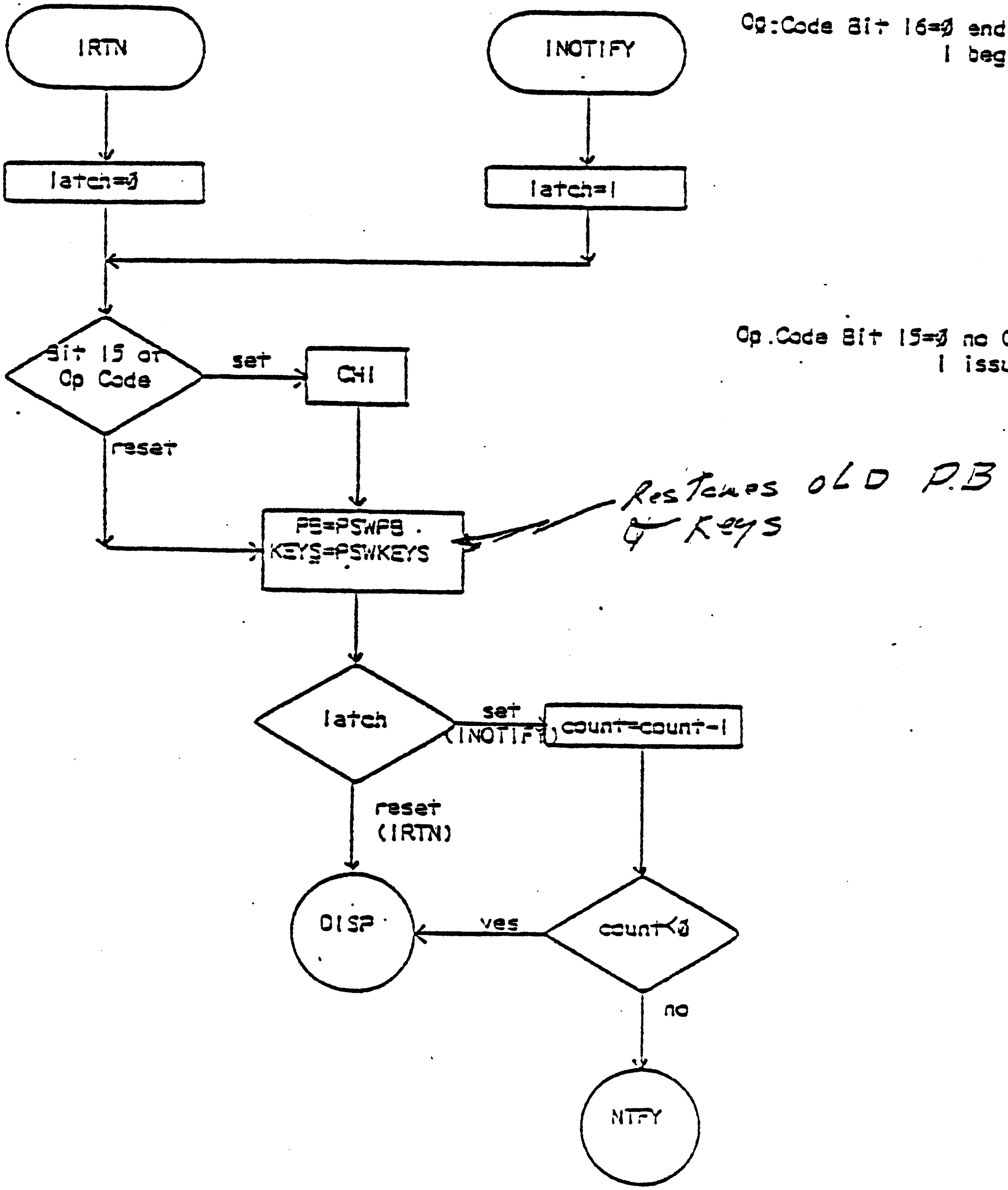
Figure 9 is a detailed flow chart of the IRTN and INOTIFY instructions.

B. Faults

Faults are CPU events which are synchronous with and, in a loose sense, caused by software. Eleven fault classes have been defined for the P-400. Several of these classes are further subdivided into distinct types. Of the eleven, three are completely new for the P-400 and, of the other eight, three have expanded meaning when in P-400 mode. The eleven fault classes and their meanings are:

Fault	P-400	P-300
RXM	Restrict mode violation	same
Process	Abort flags word .NE. 0 in PCB on dispatch	N. A.
Page	Page Fault (Page not in memory)	same
SVC	N. A.	Supervisor Call
UII	Unimplemented instruction	same
ILL	Illegal instruction	same
Access	Violation of segment	Page write viola
tion		
	access rights	
Arithmetic	All FLEX + IEX (Integer Exception)	FLEX
Stack	Stack overflow/underflow	Procedure Stack(
S-Reg)		
		Underflow
Segment	1: Segment # too big	N. A.
	2: Missing segment (SDW fault bit set)	N. A.
Pointer	Fault bit in pointer set	N. A.

The fault handling mechanism consists of two data bases and the CALF instruction. The u-code is in turn divided into a set of 'front-ends' for each fault class and a common fault handler.



Op:Code Bit 16=Ø end
1 beginn' g

Op.Code Bit 15=Ø no CAI
1 issue C I

Figure 9.

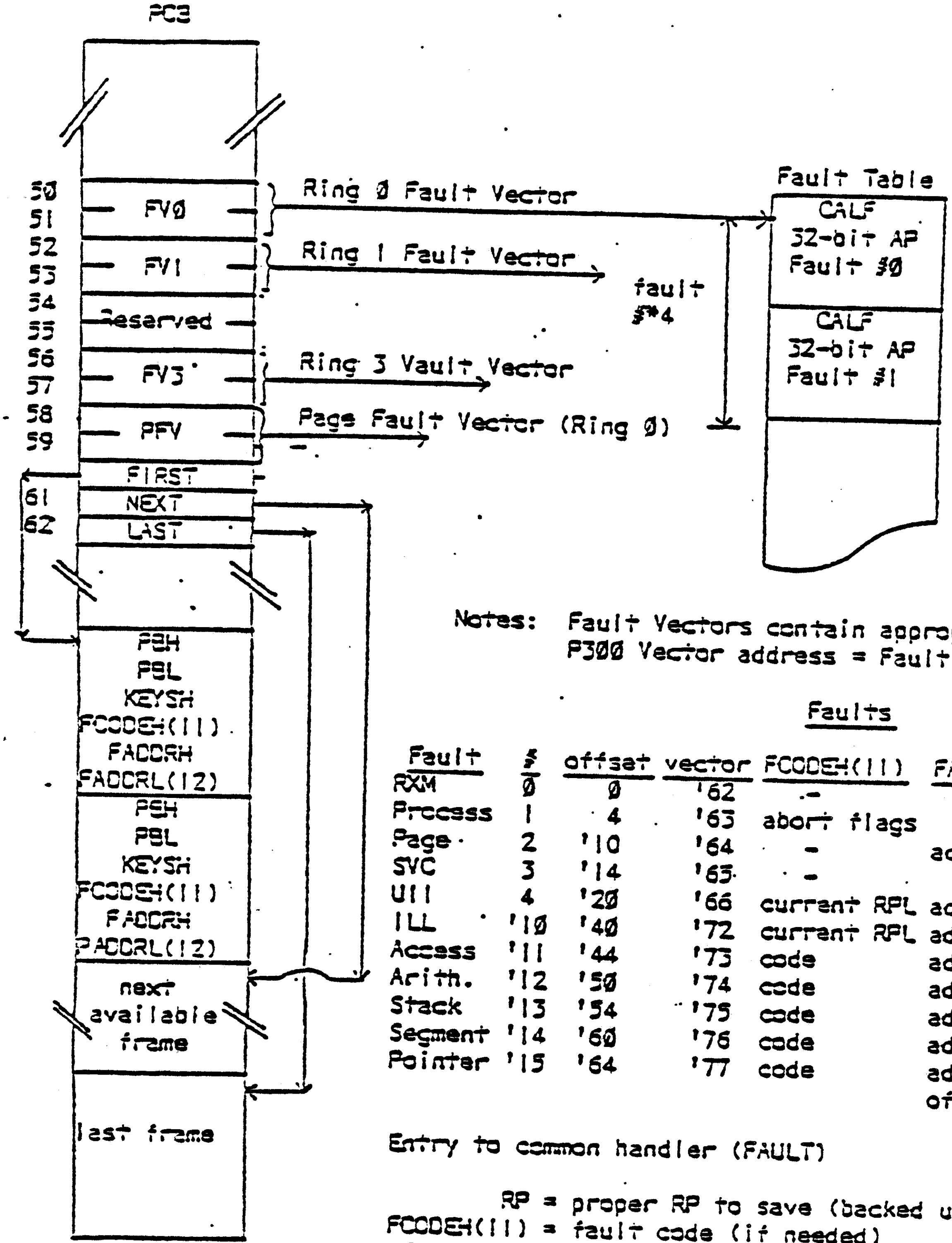
1. Data Bases

The fault data bases consist of the fault vectors and concealed stack in the PCB and the fault tables pointed to by the PCB vectors. Figure 10 shows these data bases as well as the mapping of P-300 faults to P-400 faults. Also shown in this figure is the differential action taken according to fault class (e.g., what ring to process the fault in) and the set up the u-code 'front end' must do before going to the common fault handler.

The underlying philosophy of the four fault vectors is that while some faults may need to be processed by ring 0 code, others may be adequately handled in the current ring of the faulting process. The vectors are in the PCB to allow different processes to have different fault handlers. For example, process A may wish to use a system fault routine to handle pointer faults (dynamic linker) while process B may wish to define its own algorithms for resolving pointer faults. Notice that it is always possible for a 'current ring' fault handler to call a ring 0 procedure if the need arises. Note also that page fault has its own vector despite the fact that ring 0 is entered. For the special case of page fault, only a single, system-wide processor will be used and all PCB page fault vectors will point to the same place.

The concealed stack, also in the PCB, is used to allow fault on fault conditions. For example, it is quite possible to get a segment fault while processing a segment fault. The only fault which cannot cause another fault of any type is page fault. Each frame of the concealed stack contains the PB and keys (KEYSH) of the faulting procedure as well as a fault code (to distinguish different types within each class) and a fault address, if appropriate. The stack itself is circular and must have allocated sufficient frames to handle the longest possible sequence of fault on fault that can occur in ring 0. Such a sequence might be: Pointer (link) fault -> Segment fault -> Stack fault -> Segment fault -> Page fault. Note that this particular sequence occurs before any software fault handler is entered. Also, the first segment fault enters ring 0, so at least a five-level stack is necessary if the original link fault is to be processed correctly.

The second data base consists of four distinct fault tables, each pointed to by a PCB fault vector. Each entry in the table consists of four words of which the first three must be a CALF instruction. Only the page fault table must be locked to memory and only the ring 0 table must be in a pre-defined (SDW exists) segment (otherwise, segment fault might recurse infinitely). Naturally, the ring 0 table, as well as the PCB, is carefully audited by ring 0 procedures.



Notes: Fault Vectors contain appropriate ring numbers
P300 Vector address = Fault # +'62

Faults

Fault	#	offset	vector	FCODEH(11)	FADDR(12)	Ring	Saved
RXM	Ø	Ø	'62	-	-	current	bac
Process	1	4	'63	abort flags	-	Ø	cur
Page	2	'10	'64	-	address	Ø	backe
SVC	3	'14	'65	-	-	current	cur
UII	4	'2Ø	'66	current RPL	address	current	bac
ILL	'1Ø	'4Ø	'72	current RPL	address	current	backe
Access	'11	'44	'73	code	address	Ø	backe
Arith.	'12	'5Ø	'74	code	address	current	cur
Stack	'13	'54	'75	code	address	Ø	backe
Segment	'14	'6Ø	'76	code	address	Ø	backe
Pointer	'15	'64	'77	code	address of pointer	current	bac

Entry to common handler (FAULT)

```
         RP = proper RP to save (backed up if necessary)
 FCODEH(11) = fault code (if needed)
  FADDR(12) = address (if needed)
     FCODEL = fault #*4=P400 fault table offset
     LATCH6 = Ø fault
              1 page fault (LATCH7 must=Ø)
     LATCH7 = Ø go to ring Ø
              1 use current ring
```

Figure 1Ø.

2. CALF

The CALF instruction has two major functions. First, to avoid holding off interrupts for too long, the CALF instruction defines a restart point in fault handling since it has a PB (i.e., it is a macro-machine instruction). As a result, it is quite possible to suspend a process in the middle of getting to a software fault handler. Second, it allows a straightforward mechanism to simulate a procedure call from the faulting procedure (at the instruction causing the fault) to the fault handler.

The instruction itself is a three-word generic in which the second and third words are a 32-bit pointer to the fault handler. To simulate the procedure call, the PB and KEYS from the concealed stack are placed in the fault handler's stack frame along with the other base registers (only the PB and KEYS have been changed to point to the CALF and to enter 64V addressing mode) to be used by the standard procedure return (PRTN) instruction. In addition, the fault code and address are placed in the fault handler's stack as if they were arguments passed by a standard procedure call (PCL) instruction. After the information is moved from the concealed stack it is popped. In all other respects, CALF is identical to PCL.

3. Fault Handler

The fault handler is a u-code routine that is entered from the various fault class 'front ends' and, based on process exchange mode, either simulates a P-300 type fault (JST* through segment 0 fault vectors) or performs the P-400 fault protocol which includes setting up a concealed stack frame, switching to 64V mode, and determining, on the basis of information provided by the 'front end', which fault vector to use and setting PB to point to the proper CALF in the fault table. Figure 11 is a detailed flow chart of the fault handler and Figure 10 contains a table of the necessary setup performed by each fault class 'front end'. Note that for P-300 faults, the full RP is also saved in the u-code scratch register PSWPB and the machine is inhibited for one instruction if in Ring 0.

C. Checks

Checks, unlike faults, are CPU events which are asynchronous with, and are not caused by, normal instruction execution. Rather, they are events which are either invisible (e.g., an ECC corrected error) or fatal (e.g., missing memory module) to the currently executing procedure and perhaps the CPU entirely (e.g., machine check). Checks essentially represent

: Entry:

RP = proper RP to save
:ODEH(11) = fault code
FCODEL = fault#*4
-FADDH = address(SN)
·DL .(12) = address(WN)
.TCH6 = Ø fault
1 page fault
LATCH7 = Ø use ring Ø
1 use current ring

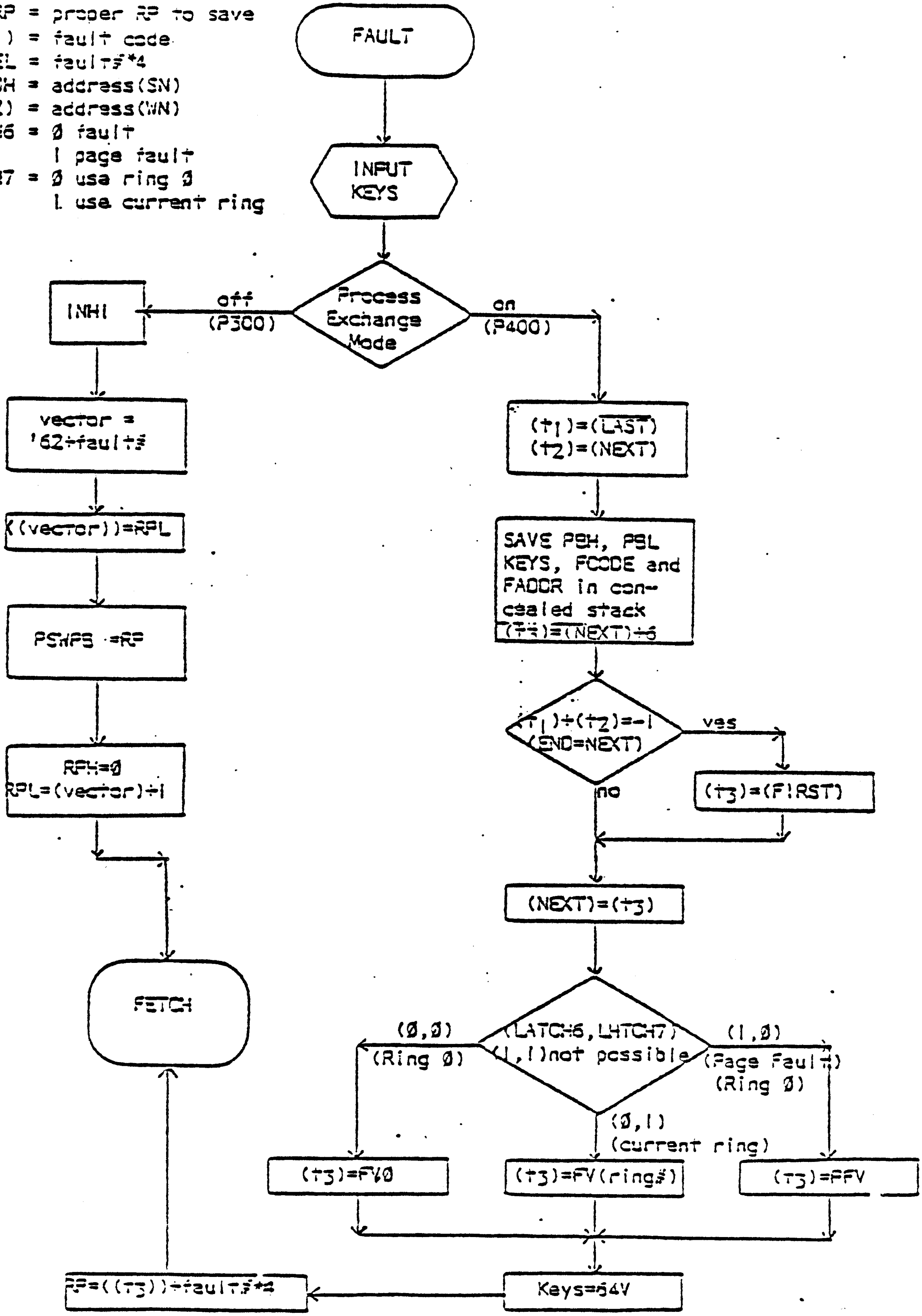


Figure 11.

processor faults as opposed to process or procedure faults. Four check classes have been defined as follows:

Check	P-400	P-300
Power Fail	Power Failure	same
Memory Parity	ECC corrected ECC uncorrected	Memory Parity
Machine Check	Fatal CPU error	same
Missing Memory Module	Memory module does not exist	same

Unlike faults which can be stacked and interrupts which cause a process to be suspended, each check class has a single save area (check block) consisting of eight words in the interrupt segment (#4) in which PB and KEYS (high and low) are saved in the first four locations (check header) and the remaining four locations contain software code (probably a JMP). Figure 12 is a picture of the check data base as well as a description of the necessary u-code setup required before going to the common check handler. In addition to the memory data base, three 32-bit registers are used as a diagnostic status word (DSW) to help a software check handler sort out what happened. Figure 13 shows the format of the DSW.

Check reporting (traps) is controlled by the two low order bits in the modals (KEYSL). The possible modes are:

MCM =	0	no reporting
	1	report memory parity (uncorrected) only
	2	report unrecovered errors only
	3	report all errors

The check trap can result in two possible actions depending upon the type of check that occurred and the type of u-code which was trapped. If the trapped code was either DMX, PIO, or external interrupt processing (unless the error was a machine check for RCM parity), or if the check was for an ECC corrected (ECCC) error, the end-of-instruction flag is set, REOIV is set to the proper offset/vector, MCM is set to 0 (except ECCC which sets it to 2), and a u-code RTN to the trapped step is executed. In this way, the IO bus is always left in a clean state. In all other cases, the check to software occurs immediately. Figure 14 is a detailed flow chart showing the operation of the check trap handlers.

The common check handler is entered from various check 'front

Check Handling (Data Base)

Software check catchers reside in the interrupt segment (4) and are 8 words each. The first 4 words are used as a PSW save area as:

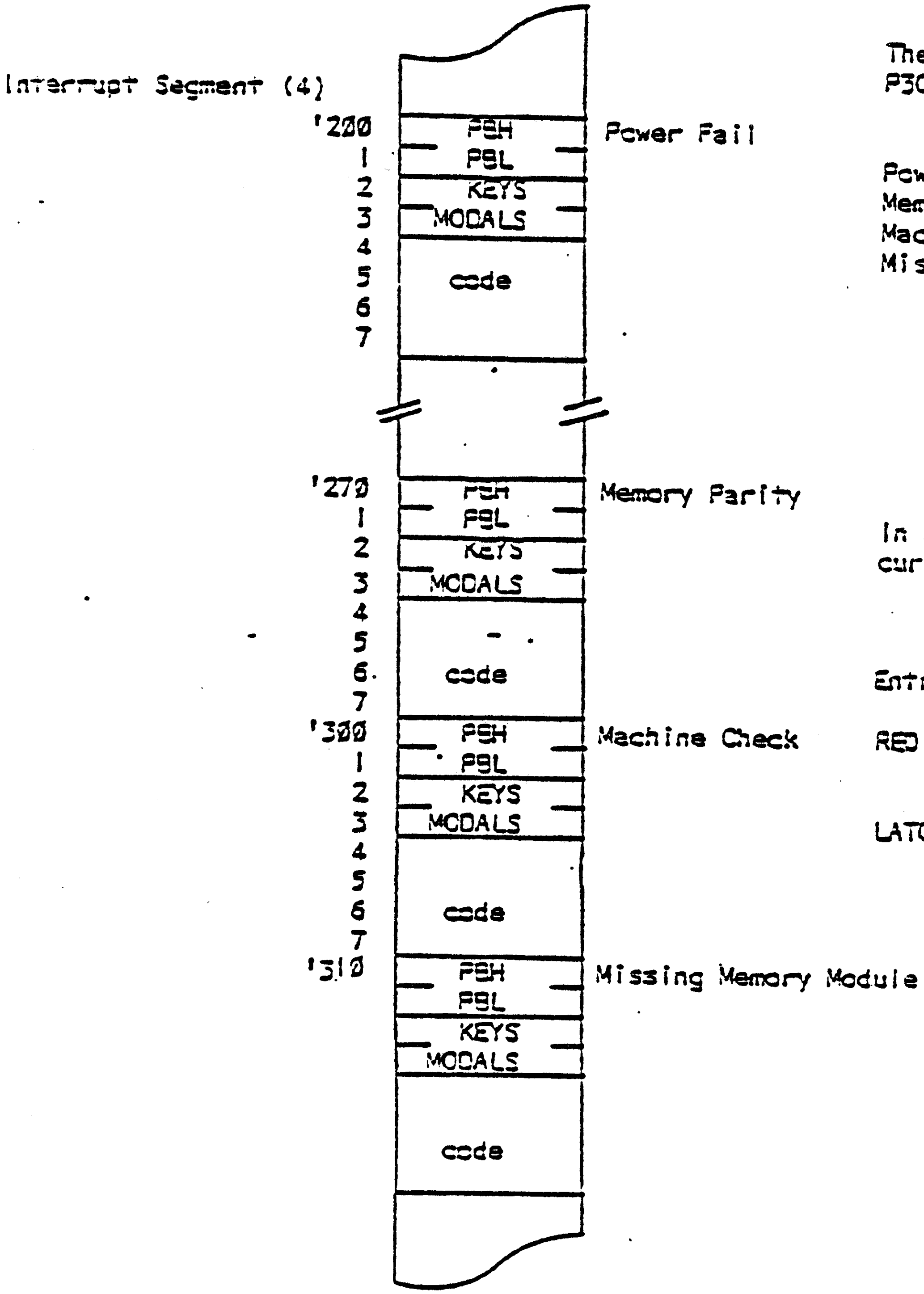


Figure 12.

The check offsets and correspon i
P300 vectors are:

Check	Offset	Vec o
Power Fail	'200	'6
Memory Par.	'270	'67
Machine Chk.	'300	'7
Missing Mem.	'310	'7

In all cases, the saved PB is t e
current PB when the check occur ac

Entry to common handler (CHECK)

```
REDIV = P400 offset
        P300 vector=(offset-'200)

LATCHS = 0 RP is proper RP to sa e
       = 1 proper RP is in PBSAVE
        (Note: PBSAVE=0 implies in
               dispatcher)
```

Diagnostic Status Word (DSW)

80 bits, Registers '34,'35&'36 (named DSWRMA, DSWSTAT, and DSWPB)
Bits 1,32: DSWRMA
33,48: DSWSTATH
49,64: DSWSTATL
65,80: DSWPB

Valid on all checks except Power Fail as follows:

1 33	2 34	3 35	4 36	5 37	6 38	7 39	8 40	9 41	10 42	11 43	12 44	13 45	14 46	15 47	16 48	DSWSTATH
C I	M C	M P	M M	Machine Check Code			$\overline{R}$ C M	E C C U	E C C C	Bup Inv	RP Backup Count			D M X	IO Bus	

1 17 49	2 18 50	3 19 51	4 20 52	5 21 53	6 22 54	7 23 55	8 24 56	9 25 57	10 26 58	11 27 59	12 28 60	13 29 61	14 30 62	15 31 63	16 32 64	DSWSTATL
RMA Inv	Res- erv- ed	ECCC Syndrome					Mod #	Reserved		u-Verify test #						

33: CI=Check Immediate
34: MC=Machine Check
35: MP=Memory Parity (ECC)
36: MM=Missing Memory
39: Machine Check Code
0=Peripheral Data (EPD) Output
1=Peripheral Address (EPA) Input
2=Memory Data (EMD) Output
3=Cache Data (RCD)
4=Peripheral Address (EPA) Output
5=RDX-EPD Input
6=Memory Address (EMA)
7=Register File
40: Not RCM Parity (Reset for RCM Parity error - XCS only)
41: ECCU=ECC Uncorrectable Error
42: ECCC=ECC Corrected Error
43: Bup Inv=RP backup count (44-46) invalid
44,46: RP Backup Count-amount RPL (DSWPB) was incremented in current instruction
47: DMX, set if check occurred during DMX
48: IO Bus, set if check occurred during DMX, PIO or Interrupt u-code
49: RMA Inv=DSWRMA invalid (Possible from ECCU and MM only)
50: Reserved
51,55: ECCC Syndrome=5 syndrome bits on a corrected error
56: Mod #=Low order address bit of memory module causing the error
57,58: Reserved
59,64: u-Verify test # set on failure during Master Clear or VIRY instruction

Validity:

```
     Always         :1-33,43,47-48,59-80
     If bit 34 set  :37-40
            35      :41-42,56  If bit 42 set:51-55
            36      :56
     If bit 43 reset:44-46
```

It is the responsibility of the check handling software to clear the DSW after a check has been processed.

Figure 13.

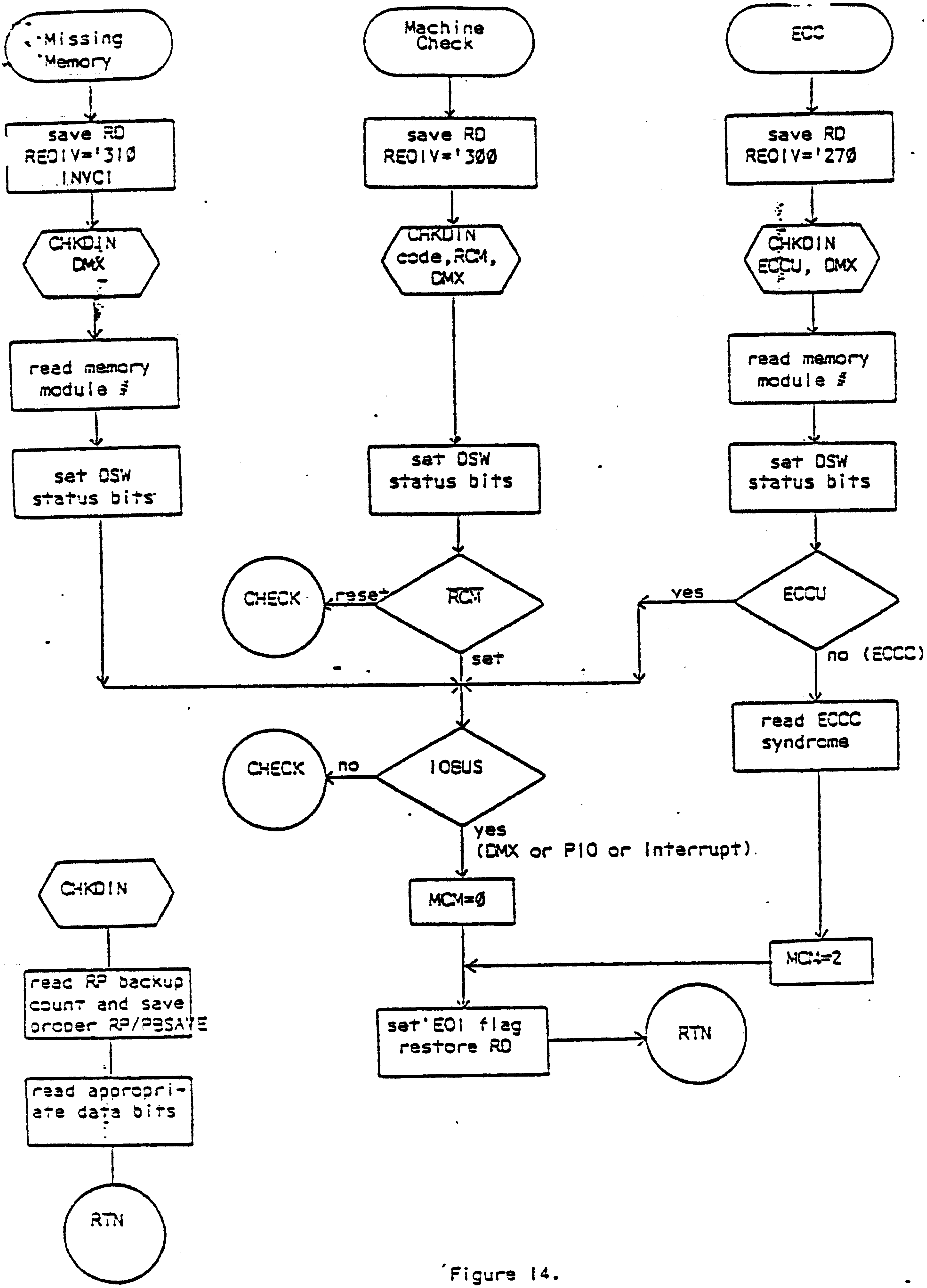

Missing Memory
save RD
REDIV='310
INVCI
CHKDIN
DMX
read memory module #
set DSW status bits
Machine Check
save RD
REDIV='300
CHKDIN
code, RCM, DMX
set DSW status bits
RCM
reset
CHECK
set
ECC
save RD
REDIV='270
CHKDIN
ECCU, DMX
read memory module #
set DSW status bits
ECCU
yes
no (ECCC)
read ECCC syndrome
IOBUS
no
CHECK
yes
(DMX or PIO or Interrupt)
MCM=0
MCM=2
set EOI flag
restore RD
RTN
CHKDIN
read RP backup count and save proper RP/PBSAVE
read appropriate data bits
RTN

Figure 14.

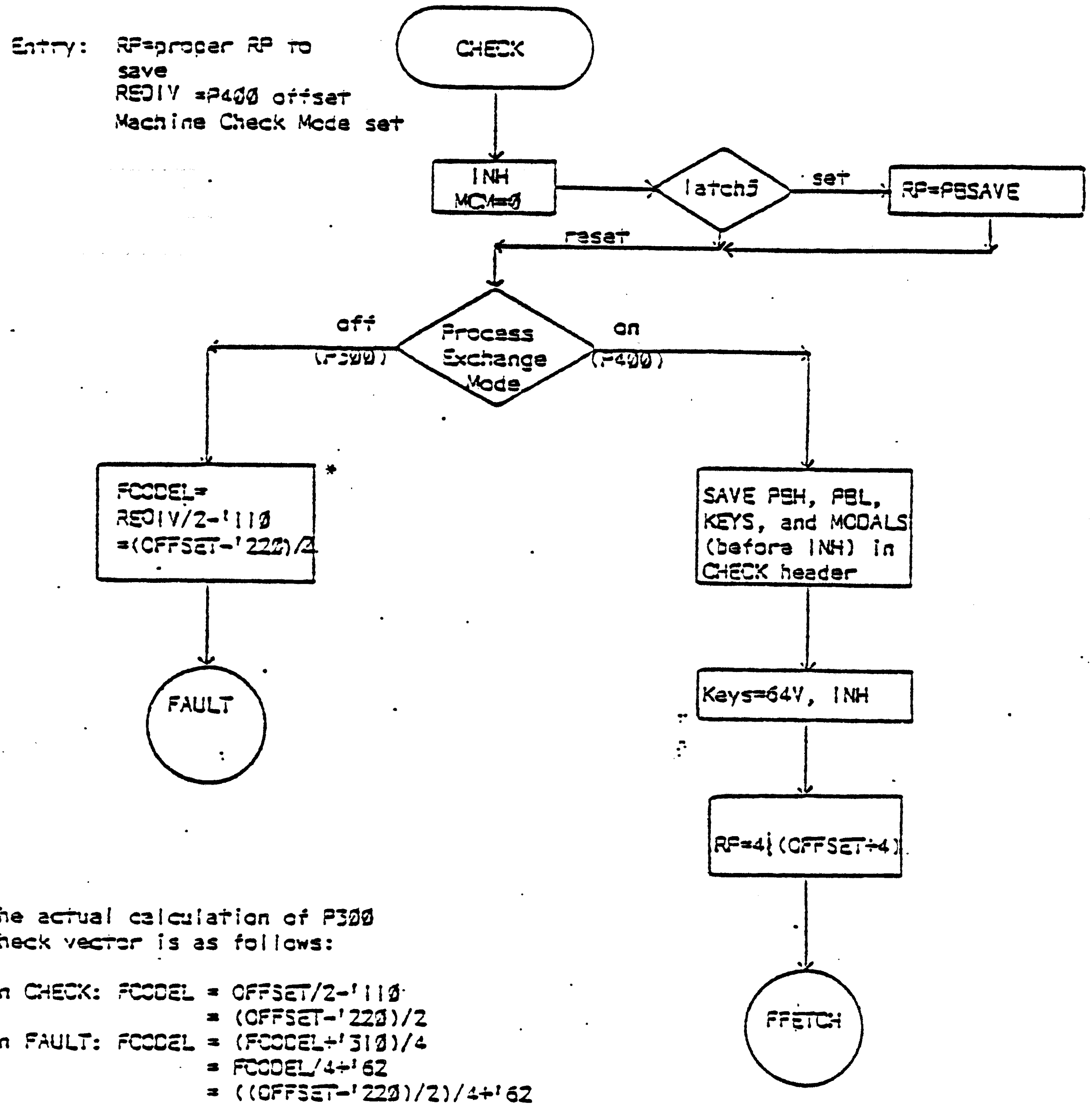
ends' and, based on process exchange mode, either simulates a P-300 type check (JST* through segment 0 check vectors) or performs the P-400 check protocol which includes setting up the check header, inhibiting the machine, and switching to 64V addressing mode. In either mode, MCM is set to 0 before going to software. Figure 15 is a detailed flow chart of the check handler and Figure 12 contains a table of the necessary setup performed by each check class 'front end'.

III. REGISTER FILES

The PRIME 400/500 contains four distinct register files. Each file is further divided into halves, each 32 locations (registers) long, and each 16 bits wide. One half is referred to as the high half and the other as the low half. Since both halves are addressed together, each register file contains 32, 32-bit register or 64, 16-bit registers. The register files, numbered from 0, are used as follows:

RF0 - u-code scratch and system registers
RF1 - 32 DMA channels
RF2 - User register set
RF3 - User register set

This layout of register files allows easy expansion to eight register files, thus adding four new user register sets. All user register sets have the same internal format and the DMA register file simply consists of 32 channel registers. Channel register '20 within RF1 is equivalent to the P-300 DMA registers '20 and '21. Channel register '22 is mapped to '22 and '23. In this way, the mapping proceeds for each even register in RF1 to channel register '36, mapped to '36 and '37. All other RF1 registers represent additional DMA channels over the P-300. Figure 16 shows the internal structure (usage) of RF0 and the user register sets (RF2, RF3). Note that all user register sets contain the segment number of the Ready List/PCB segment (OWNERH) and a cell for the modals (KEYSL). It is necessary, before entering process exchange mode, to set OWNERH in ALL register sets to the proper value and to NEVER alter it thereafter. Although all register sets contain a cell for the modals, only the current register set (CRS) contains the valid modals. It is therefore necessary, whenever register sets are switched, to copy the modals into the new register set. Currently, only the Dispatcher switches register sets. CRS is defined and specified by the three bit field labeled 'CRS' in the modals. Since this field can span up to eight register files, but two are used for u-code scratch and DMA, user register sets are numbered from 2 - 7. Of course, only 2 and 3 are currently implemented. Thus, for the P-400/500, the CRS field must always have bit 9 off, bit 10 on, and bit 11 selects the register set (as if 0 and 1 were the numbers). In fact, the u-code will only look at bit 11.



Entry: RP=proper RP to save
REDIV =P4ØØ offset
Machine Check Mode set

*The actual calculation of P3ØØ check vector is as follows:

```
In CHECK: FCODEL = OFFSET/2-'11Ø
                 = (OFFSET-'22Ø)/2
In FAULT: FCODEL = (FCODEL+'31Ø)/4
                 = FCODEL/4+'62
                 = ((OFFSET-'22Ø)/2)/4+'62
                 = (OFFSET-'22Ø)/8+'62
                 = (OFFSET-'2ØØ-'2Ø)/8+'62
                 = (OFFSET-'2ØØ)/8-2+'62
                 = (OFFSET-'2ØØ)/8+'6Ø
```

This circuitous calculation is used to oid dividing a negative number on a wer fail check.

Note: '2ØØ (Power fail offset)-'22Ø = -'2Ø.

Figure 15

u-code scratch

: 2 ar	High	Low
(	TRØ	-
	TR1	-
2	TR2	-
3	TR3	-
4	TR4	-
5	TR5	-
6	TR6	-
7	TR7	-
0	RDMX1	-
1	RDMX2	-
12		RATMPL
3	RSGT1	-
4	RSGT2	-
15	RECC1	-
16	RECC2	-
7		REDIV
2Ø	ZERO	ONE
21	PBSAVE	-
22		
23		
24		
25		
26		
27		
30	PSWPB	-
31	PSWKEYS	-
	PPA:PLA	PCBA
33	PPB:PLB	PCBB
34	DSWRMA	-
35	DSWSTAT	-
36	DSWPB	-
37		

DMA

Cell	High	Low	RF1 Addr
0			40
1			41
2			42
3			43
4			44
5			45
6			46
7			47
10			50
11			51
12			52
13			53
14			54
15			55
16			56
17			57
2Ø	(2Ø)	(21)	60
21			61
22	(22)	(23)	62
23			63
24	(24)	(25)	64
25			65
26	(26)	(27)	66
27			67
3Ø	(3Ø)	(31)	70
31			71
32	(32)	(33)	72
33			73
34	(34)	(35)	74
35			75
36	(36)	(37)	76
37			77

Current Register Set (CRS)

CRS Cell	High	Low	RF2 Addr	RF3 Addr
0	GRØ	-	100	140
1	GR1	-	101	141
2	GR2(I,A,LH)	-(2,B,LL)	102	142
3	GR3(EH)	-(EL)	103	143
4	GR4	-	104	144
5	GR5(3,S,Y)	-	105	145
6	GR6	-	106	146
7	GR7(Ø,X)	-	107	147
10	FRØ(13)	-	110	150
11	-	-	111	151
12	FR1(4)	-(5)	112	152
13	-(6)	-	113	153
14	PB	-	114	154
15	SB(14)	-(15)	115	155
16	LB(16)	-(17)	116	156
17	XB	-	117	157
20	DTAR3(1Ø)	-	120	160
21	DTAR2	-	121	161
22	DTAR1	-	122	162
23	DTARØ	-	123	163
24	KEYS	(modals)	124	164
25	OWNER	-	125	165
26	FCODE(11)	-	126	166
27	FADDR	-(12)	127	167
30	TIMER	-	130	170
31	-		131	171
32			132	172
33			133	173
34			134	174
35			135	175
36			136	176
37			137	177

KEYSH

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
CBIT	OF	LINK	Adr Mode			FLEX	IEX	CCLT	CCEQ					ID	SD

KEYSL (Modals)

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
ENB	VIM							CRS			MIO	PXM	SEG	MCM	

Adr.	Mode
Ø	16S
1	32S
2	64R
3	32R
4	32I
5	
	64V

FLEX=Ø allows FLEX Faults

ENB: Set=enable interrupts
VIM: Set=Vectored interrupt mode
CRS: Current Register Set
MIO: Set=mapped I/O
PXM: Set=Process Exchange Mode
SEG: Set=Segmentation Mode
MCM: Machine Check Mode

ID: In Dispatcher
SD: Save Done

Figure 16.

Direct register file addressing (not using CRS) is accomplished either with the LDLR/STLR instructions or via the control panel. The Register Files are ordered sequentially with an absolute address of 0 addressing RF0-register 0 (u-code scratch/system file), '40 addressing RF1-register 0 (DMA file), '100 addressing RF2-register 0 (user set 2), and '140 addressing RF3-register 0 (user set 3).

Beside each register name, where appropriate, is the PRIME-300 mode mapping from address traps to registers (e.g., the X register is the high half of GR7).

IV. CONTROL PANEL

The control panel for the P-400/500 is the same physical panel used for the P-100/200/300. It's functionality was enhanced by improving the u-code in the CP. All switches and selectors operate exactly as for the P-300 with the exception of the sense switches in the up position. Figure 17 is a diagram of the functionality of the switches. Notice that with all switches down, any FETCH/STORE operations are to/from memory-mapped. As long as segmentation mode is not turned on, mapped and absolute are the same, thus preserving compatibility. If SS4 down were absolute, address traps could not occur and would thus be incompatible. Notice also that SS5-16 in the up position changes meaning depending upon SS4. When mapped, all 12 switches are read as a 12-bit segment number. When absolute, SS11-16 are used as the 6 high order bits of the 22-bit physical address. To address any P-300 registers, all sense switches should be placed in the down position and addresses between 0 and '37 specified..

P-400/500 registers are accessed by raising SS1. Then, if SS2 is down, the low order 5 bits of the address are used to access 32-bit registers 0-'37 within CRS. If SS2 is raised, the full 7 bit address is used to access any register in any register file. The addresses, as shown in Figure 16, are 0-'37=u-code scratch/system, '40-'77=DMA, '100-'137=User set 2, and '140-'177=User set 3. SS4 is used to access either the high half (up) or the low half (down) of the selected register. For all register accesses, the Y+1 functions will advance the register address before the access, exactly as for memory accesses. Wrap around will occur on the appropriate number of bits, since any bits of higher order are ignored for the access.

The control panel data register is TR2H and the address register is TR3. Upon entering the control panel routine, RP is saved in TR3 and (RP) is saved in TR2H. In addition, the keys (KEYSH) are updated to reflect accurately the live keys. Thereafter, TR3H is not altered by the control panel itself so RPH is always remembered. However, on exit, PBH is used to update RPH and KEYS is used to update all the keys. As a

SS1		SS2	SS3	SS4	SS5-16
register	up	absolute		high half	SS11-16
	down	CRS		low half	
memory	up			absolute	Physical Address 95-ØØ
	down			mapped	Segment #

Notes: With all switches down, control panel works exactly as for the P-3ØØ following either a Master Clear or a HALT if not running in segmented mode. It is necessary to make mapped memory accesses if address traps are to be generated. If running segmented, memory accesses will be mapped to segment Ø unless an explicit segment number is entered in SS5-16.

Registers: Register address is in address register (switches down) For CRS, only low order 5 bits are used; for absolute, only low order 8 bits are used Y+1 (STORE/FETCH) operates exactly as for memory with the address being pre-incremented.

Null Vector: In P-3ØØ mode, if an external interrupt, fault, or check attempts to vector through a memory location containing a Ø, the following action is taken:

HALT
data <u>and</u> address lights cleared
RP = address trapped
PBH = RPH
TR2L = address of vector

Figure 17.

result, single stepping can change segments as well as keys and modals. Figure 18 is a detailed flow chart of the control panel routine.

The only exception to the control panel entry protocol is that if a Fault, Check, or external Interrupt attempts to vector through a vector containing 0 in P-300 mode, the following registers will contain:

RP: address of 'trapped' instruction
PBH: SN of 'trapped' instruction
KEYSH: proper keys
TR2H: (data) 0
TR3: (address) 0!0
TR2L: address, in segment 0, of the 'vector' containing 0

V. CP TIMER

Resolution = 1024 u-sec

Turned on by DISPATCHER before dispatch.

Turned off by:
WAIT after/during save
DISP before changing CRS

On tick, u-code increments the interval timer (TIMER) in RF(CRS). When that overflows, bit 16 in the PCB abort flags (memory) is set to cause a process fault.

It is the responsibility of software that resets the interval timer to maintain the elapsed timer.

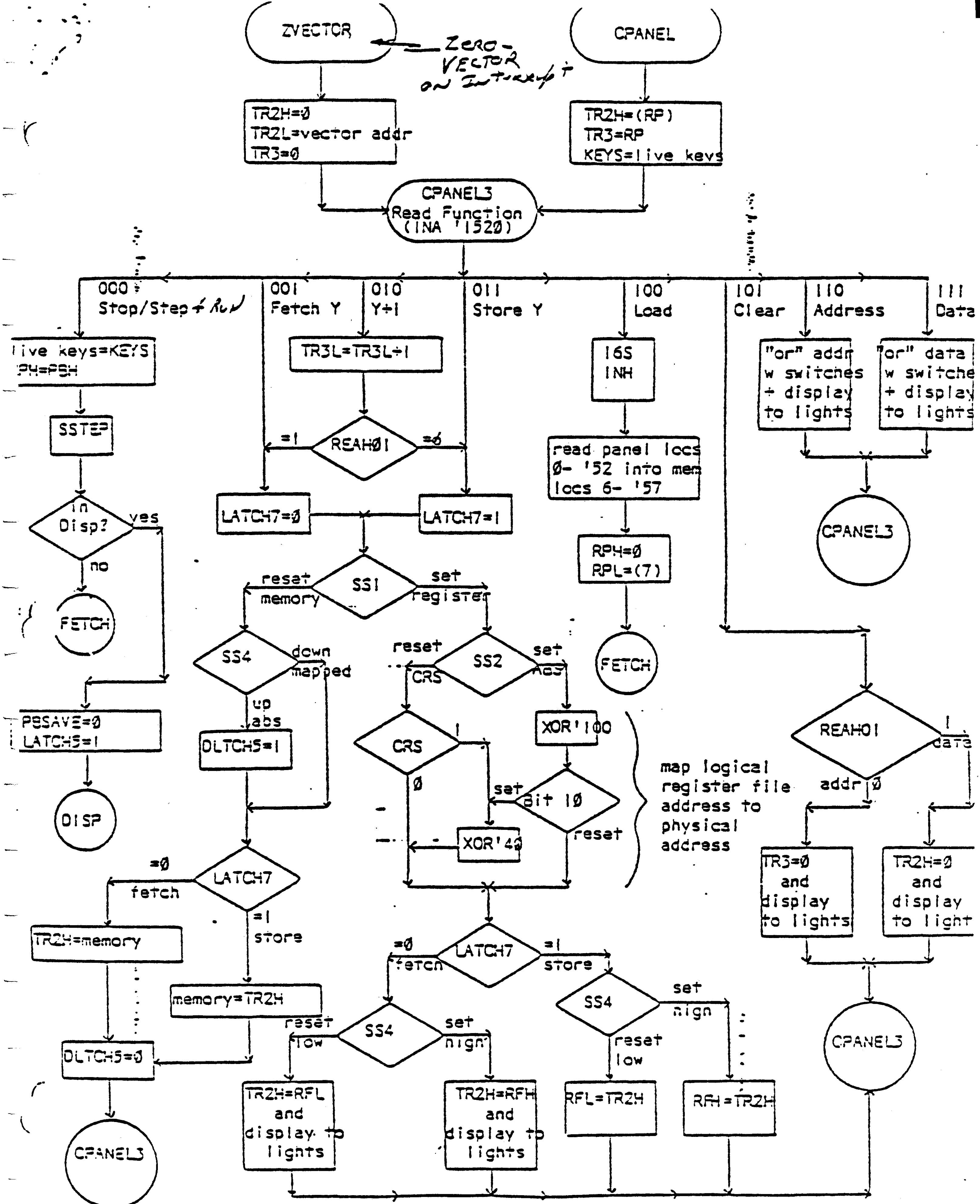

ZVECTOR
ZERO-VECTOR on Interrupt
CPANEL
TR2H=Ø
TR2L=vector addr
TR3=Ø
TR2H=(RP)
TR3=RP
KEYS=live keys
CPANEL3
Read Function
(INA '152Ø)
000
Stop/Step + Run
001
Fetch Y
010
Y+1
011
Store Y
100
Load
101
Clear
110
Address
111
Data
live keys=KEYS
PH=PSH
SSTEP
in Disp?
yes
no
FETCH
PBSAVE=Ø
LATCH5=1
DISP
TR3L=TR3L+1
=1
REAHØ1
=Ø
LATCH7=Ø
LATCH7=1
reset memory
SS1
set register
SS4
down mapped
up abs
DLTCH5=1
LATCH7
=Ø
fetch
=1
store
TR2H=memory
memory=TR2H
DLTCH5=Ø
CPANEL3
reset
CRS
SS2
set
ACS
CRS
1
Ø
XOR'100
set
Bit 1Ø
reset
XOR'4Ø
map logical register file address to physical address
LATCH7
=Ø
fetch
=1
store
SS4
reset
low
set
high
TR2H=RFL and display to lights
TR2H=RFH and display to lights
SS4
set
high
reset
low
RFL=TR2H
RFH=TR2H
16S
INH
read panel locs Ø- '52 into mem locs 6- '57
RPH=Ø
RPL=(7)
FETCH
"or" addr w switches + display to lights
"or" data w switches + display to lights
CPANEL3
REAH01
1
data
addr
Ø
TR3=Ø and display to lights
TR2H=Ø and display to light
CPANEL3

Figure 18.

Appendix C - Procedure Call Mechanism

SUBROUTINE CALLS

(Procedure CALL)
Calculates Ring #

(1) CALLING PROGRAM

CALL

- CALLS A SUBROUTINE
- GENERATES PCL (procedure call)

ENTRY Control Block
(ECB)

PCL

- ADDRESSES AN ECB THROUGH A LINK
- CALCULATES THE RING NUMBER
- ALLOCATES THE STACK FRAME

(Stack = Last in First out)
One

- INITIALIZES THE STATE OF THE CALLED PROCEDURE
- TRANSFERS THE ARGUMENT POINTERS

AP

- GENERATES THE ARGUMENT POINTERS FOR THE PCL
- FOLLOWS THE PCL INSTRUCTION
- FORMAT

```
    AP  ARG,TAG
     where TAG modifier can be:
       - S   variable is an argument
       - SL  variable is the last argumnet
       - *S  the argument is an indirect address
       - *SL the argument is an indirect and the last
```

EXAMPLE:

```
        CALL      SUB1
        AP        ARG1,S
        AP        ARG2,SL

        LINK
ARG1    DATA       0
ARG2    DATA       0
```

(2) THE SUBROUTINE

ARGT

- DOES THE LAST STEP OF THE PCL INSTRUCTION
- EXECUTED ONLY IF A FAULT OCCURS DURING THE CALL ARGUMENT TRANSFER
- MUST BE PRESENT IF THE SUBROUTINE REQUIRES ARGUMENTS

ECB

- GENERATES AN ENTRY CONTROL BLOCK (ECB) TO DEFINE A PROCEDURE ENTRY
- GOES INTO A LINK FRAME
- FORMAT

```
    LABEL   ECB     PFIRST,,ARGDISP,NARGS,SFSIZE,KEYS
```

WHERE:

```
  PFIRST  - pointer to the first executable statement
  ARGDISP - displacement in the stack frame of the
            argument list (default is '12)
  NARGS   - number of arguments to be passed
  SFSIZE  - stack frame size, the default is given
            by the DYMN
  KEYS    - keys, the default is 64V
```

(3) ARGUMENT TEMPLATE

1 4	5	6	7 8	9	10	11 16
B	I	0	base reg	L	S	0 ------------ 0
W						

B = BIT NUMBER

I = INDIRECT BIT

L = LAST BIT, LAST TEMPLATE FOR THIS PCL

S = STORE BIT, LAST TEMPLATE FOR THIS ARGUMENT

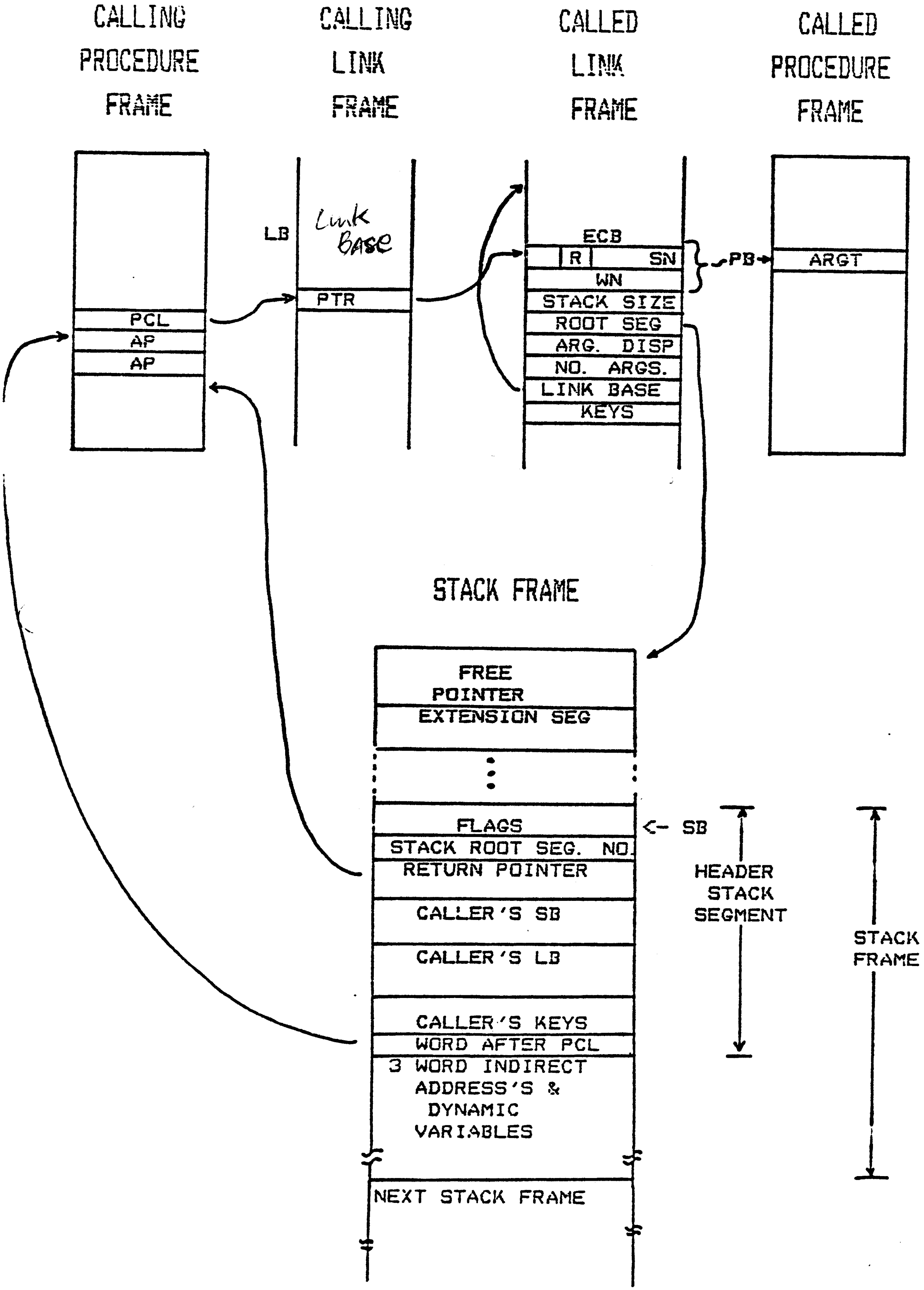
(4) ENTRY CONTROL BLOCK

0 1	POINTER TO THE FIRST EXECUTABLE STATEMENT OF THE CALLED PROGRAM
2	SIZE OF STACK FRAME
3	STACK ROOT SEGMENT NO.
4	ARGUMENT DISPLACEMENT
5	NUMBER OF ARGUMENTS
6 7	LINKAGE BASE ADDRESS OF THE CALLED PROGRAM
8	KEYS FOR THE CALLED PROGRAM
9 15	RESERVED MUST BE ZERO

(5) STACK FRAME (Has multiply segments)

0	POINTER TO THE NEXT FREE FRAME
1	
2	POINTER TO THE EXTENSION SEGMENT
3	
	⋮
0	FLAGS
1	STACK ROOT SEGMENT NO.
2	RETURN POINTER
3	
4	CALLER'S STACK BASE
5	
6	CALLER'S LINK BASE
7	
8	CALLER'S KEYS
9	WORD NUMBER AFTER PCL
10	POINTERS TO THE ARGUMENTS (3 WORD INDIRECT ADDRESSES) AND DYNAMIC VARIABLES

Procedure Call Mechanisim



Appendix D - Revision 19.0 Routine List

Index of files in PRIMOS>KS - Primos kernel code.

'*' in column 1 indicates file did not exist at Rev. 18

	AB$SW$.PLP	Routine to read ABBRSW in FIGCOM for ring 3.
	ACCOM$.PLP	Access cominput information in pudcom for ring 3 procedures
	ADDISK.FTN	ADDS DISKS TO THE SYSTEM DISK TABLE
	AINIT.FTN	COLD START INITIALIZATION (PART 1)
	AMINIT.PMA	INITIALIZES AMLC CONTROLLER(S)
	AMLC$.FTN	PROCESS INTERNAL COMMAND AMLC
	AMLDIM.PMA	PROCESSES AMLC INPUT AND OUTPUT
	ASNDE$.FTN	ASSIGN DISK AND OTHER PERIPHERAL DEVICES EXCEPT MAGTAPE.
*	ASNLN$.PLP	ASSIGN AND UNASSIGN AMLC LINES
*	ASNMT$.PLP	Assign magnetic tape drive units.
	ASRDIM.PMA	CLOCK DRIVEN ASR DRIVER (OPTION-A)
*	ASSUR$.PLP	Ensures a user has specified amount of cpu time left
	BADDSK.FTN	CHECK FOR LEGAL PRIMOS DISK NUMBER
*	BADIX$.FTN	MAP OUT BAD PAGING DEVICE RECORDS.
*	BCKUPB.PLP	Back Up Return PB For Ring 3 QUIT FIM.
	BFGETR.PMA	BUFFERING PACKAGE USED BY MPCDIM, VERDIM
*	BINIT.FTN	COLD START INITIALIZATION (PART 2).
	BREAK$.PMA	Manage Quit Inhibit Counters for all rings.
	BRPDIM.FTN	PAPER TAPE PUNCH DIM
	C1IN$.PLP	Single Character Command Input
	C1IN.PLP	User Version Of C1IN$
*	CHG$PW.PLP	Change the user's login password.
*	CHG$SA.PLP	Change System Administrator.
	CINIT.FTN	COLD START CONFIGURATION
	CMREA$.FTN	OLD STYLE COMMAND LINE PARSER
	CNEQV.PMA	NAMEQV-COMEQVCOMPARE ASCII NAMES
	CNFLCT.FTN	CHECK FOR CONFLICTING PRIMOS PARTITIONS
*	CPS$.PLP	Cross Process Signalling Send Signal Routine
*	CPS$CA.PLP	Cross Process Signalling Clear A User From All ACLs
*	CPS$CN.~~PLP~~	Cross Process Signalling Control Routine
*	CPS$CU.PLP	Cross Process Signalling Clear A User's USL.
*	CPS$DF.PLP	Cross Process Signalling Defer Signal Routine
*	CPS$IN.PLP	Cross Process Signalling Initialization Routine
*	CPS$NA.PLP	Cross Process Signalling Name Routine
*	CPS$RC.PLP	Cross Process Signalling Signal Received Routine
*	CPS$RG.PLP	Cross Process Signalling Registration Routine
*	CPS$SN.PLP	Cross Process Signalling Who Signalled Routine
*	CPS$ST.PLP	Cross Process Signalling Status Routine
	CRDDIM.PMA	CARD READER DRIVER
	CSTAK$.PLP	Manipulate/examine the calling process' concealed stack.
*	DATE$.PLP	Return the standard (FS) format date and time.
	DELAY.PMA	SET SLOPE OF DELAY CURVE FOR TERMINAL
	DEVCHK.FTN	CHECK EXTERNAL DEVICE ASSIGNMENT.
	DISKIO.PMA	DISK I/O FOR Primos.
	DMGSET.FTN	SET-UP DMQ CONTROL BLOCKS AND BUFFERS.
	DOSSUB.FTN	COMMAND LINE PROCESSOR FOR PRMOS4.
*	DROPD_.PLP	Invoke the DROPDTR command from ring3
	DRPDTR.PLP	Drop the amlc line dtr for a desired user
	DSKCHN.PMA	DISK CONTROLLER CHANNEL PROGRAMS.
	DSKEQV.FTN	CHECK FOR SAME PARTITION OR OVERLAPPING PARTITIONS
	DUPLX$.FTN	SET/RETURN TERMINAL CONFIGURATION WORD
	DYNSGS.PMA	DYNAMIC SEGMENT ALLOCATION DATA BASE
*	ENCRYPT$.PLP	Encrypt a user's login password.
	ERKL$$.FTN	SET ERASE AND KILL CHARACTERS FOR USER
	ERRRTN.FTN	ERROR RETURN HANDLER FOR PRMOS4.
*	EXTLOG.PLP	Restore the external login/logout program.

	FATAL$.PMA	FATAL PROCESS ERROR
	FILPAG.PMA	FILL PAGE WITH ZEROES
*	FIND_SEG.PLP	Return a vector of free segment numbers
*	GATINI.FTN	RING 0 GATE SEGMENT INITIALIZATION.
	GCHAR.PMA	GET CHAR FROM ARRAY, STEP CHAR PTR
	GETSEG.FTN	ADD A SEGMENT TO A USER
	GETSN$.PLP	Return a vector of allocated segment numbers
*	GET_SANAME.PLP	Read SA name from SAD into SUPCOM.
	GMETR$.PLP	Get metering data of various sorts and flavours.
	GPGREC.FTN	Allocate a paging device index.
*	GPIDIM.PMA	INTERRUPT PROCESS FOR T$GPPI INTERFACE
	GTWNDO.PMA	ROUTINE TO ALLOCATE SEG-0 WINDOWS FOR MAPPED I/O.
	HCS$.PMA	FIND GATE ENTRY POINT FOR POINTER FAULT HANDLER.
	HMAPS.PMA	SEGMENT 22 MODULE
*	INITSU.PLP	Initialize a new user.
*	INSON$.PLP	INSON$ initializes static on unit lists
*	IOA$SY.PMA	Ioa$ call for system console.
*	IOWIRE.PMA	Wire/unwire pages for performing I/O.
*	IOWNDW.PMA	Open mapped I/O windows.
*	JOB$O.PLP	Accesses on Batch queue control file.
*	LGINI$.PLP	Turn on and off OS and network logging.
	LIMIT$.FTN	SET/READ CPU AND LOGIN TIME LIMITS.
	LISTEN.PLP	Ring Zero (logged out) Listener.
	LMAPS.PMA	SEGMENT 33 MODULE
	LOCKPG.FTN	WIRE AN AREA OF THE VIRTUAL MEMORY.
*	LOGABT.PLP	Handle Logout Process Aborts (forced and timeouts).
	LOGEV1.PMA	FIRST-LEVEL EVENT LOGGING.
	LOGEV2.FTN	SECOND-LEVEL EVENT LOGGER
*	LOGIN$.PLP	Ring zero LOGIN command processor.
	LOGO$$.FTN	SUBROUTINE TO LOG OUT A USER OR USERS
*	LOGO$CP.PLP	Logged out command processor
*	LOGOCMT_.PMA	Logged out command table.
*	LOGOCM_.PLP	Decide whether command is a valid logged out command.
*	LOGOUT.PLP	Logout interface (r3 to r0) and message sender.
*	LOG_INIT.PLP	Reset parameters after logout or before login.
*	LON$C.PLP	Closes a user's logout notification message queue.
*	LON$CN.PLP	Logout Notification Instant Notify Control Routine.
*	LON$O.PLP	Logout Notification Receiver Message Queue Opener
*	LON$R.PLP	Logout Notification Message Receive Module
*	LON$S.PLP	Logout Notification Phantom Message Send Module
*	LOV$SW.PLP	Routine to read LOGOVR in FIGCOM for ring 3
*	LO_CLEAN.PLP	Clean up after external logout or login error.
*	LO_FATAL.PLP	Main logout processor, called by LOGOUT and FATAL$.
*	LO_NATCH.PLP	Unhash and close all attach points during logout.
	MAPIO.PMA	LOCK AND MAP (AND UNLOCK) USER BUFFERS INTO SEGMENT 0
	MAPNDX.PMA	ROUTINES TO FIND SDW AND PAGE MAP.
	MAPSEG.FTN	MAPS A SEGMENT ALREADY DEFINED IN DTAR 0 TO ANOTHER SEGMENT
	MESSAG.FTN	HANDLE MESSAGE COMMAND.
	MGSET$.FTN	SETS MSG RCV STATE FOR USER
	MINABT.FTN	HANDLE 1 MINUTE PROCESS ABORT.
	MOVE$.PMA	DATA MOVEMENT SUBROUTINES.
	MOVUTU.FTN	MOVE WORDS FROM ONE USER'S VIR. ADDR SPACE TO ANOTHER USER'
	MP2DIM.PMA	DRIVES LINE-PRINTER, CARD-READER, CARD-PUNCH VIA MPC#2.
	MPCDIM.PMA	DRIVES LINE-PRINTER, CARD-READER, CARD-PUNCH VIA MPC.
	MSG$.FTN	Send a message to a user on an arbitrary node.
	MSG$ST.FTN	RETURN MSG STATUS TO CALLER
	MSGCOM.PMA	MESSAGE COMMON
	MSGOUT.PLP	Message facility -- output message to user.
	MTDIM.PMA	DRIVES MAG-TAPE VIA MPC.
	N1LOCK.PMA	LOCKING ROUTINES FOR PRIMOS

	NLKCOM.PMA	NON-WIRED COMMON
*	NLOGIN.PLP	Main login routine for Normal users.
	OERRTN.FTN	OLD-STYLE ERROR HANDLING
	ORGO.PMA	SETS LOADER WDNO TO ZERO
	PABORT.FTN	HANDLE PROCESS ABORT CONDITIONS (NEE SCHED)
	PAG$FS.PLP	Page (to)/from the file system (1040wd-record devices).
*	PAGINI.FTN	PRIMOS PAGING MECHANISM COLD START INITIALIZATION.
	PAGTUR.FTN	TURN PAGE(S) IN RESPONSE TO A PAGE FAULT.
	PBDIOS.PMA	PAPER TAPE READER, PUNCH, PRINTER I/O RELATED ROUTINES
*	PBH$ON.PLP	PB Histogram Facility Startup/Access entries.
	PBTABL.PMA	Data area for PB Histogram.
*	PCBINI.FTN	PCB INITIALIZATION FOR COLD START.
*	PCBPTR_.PLP	Return ptr to a specified user's PCB.'
	PGFSTK.PMA	PUDCOM AND PAGE FAULT STACK FOR USER 1.
*	PHLOGIN.PLP	Log in a phantom user.
	PHNTM$.FTN	START UP PHANTOM USER (SVC AND DOSSUB COMMAND)
*	PHTTYREQ.PLP	Force a phantom to log out after an illegal TTY request.
	PMSG$.FTN	PRINT INTER USER MESSAGE.
	PRERR.FTN	PRINT NAME AND/OR MESSAGE FROM USER'S ERRVEC
	PRN$ST.FTN	PRINT SYSTEM STATUS ON USER TERMINAL.
	PTRAP.FTN	RESTRICTED MODE TRAP HANDLER
	PTRDIM.FTN	PAPER TAPE READER DIM
*	QUTABT.PLP	Handle QUIT Process aborts for the current process.
*	QUTRST.PLP	Reset Ring 0 QUIT Enable Mechanism.
	R0BASE.PMA	GET A POINTER TO THE FIRST FRAME ON THE RING 0 STACK.
*	R0FALT.PMA	RING 0 FAULT HANDLERS, RING 0 UTILITY SUBRS.
	R0UII.PMA	SPECIAL (QUICK, SMALL STACK FRAME) UII F.I.M. FOR RING 0.
	R3CALL.PMA	CALLS FROM RING 0 TO RING 3 ENVIRONMENT.
*	REMLI$.FTN	Process the REMLIN command.
	REPLY$.FTN	Operator/user communication facility.
	RMSGD$.FTN	RETURNS CONTENTS OF PER USER MSG BUFFER TO CALLER.
	RTIME$.PMA	Return real-time as 48 bit value in PIC counts
	RTNSEG.PMA	INTERLUDE TO RTNSG1.
	RTNSG1.FTN	Returns one segment or all segments in a unser's process.
*	SANAM$.PLP	Return the name of the System Administrator
	SCHAR.PMA	STORE CHAR INTO ARRAY, STEP CHAR PTR
	SCHED.PMA	PRIMOS 4 SCHEDULING ROUTINES
	SEG0.PMA	SEGMENT 0 MODULE
	SEG14.PMA	Segment 14 module
	SEG4.PMA	SEGMENT 4 MODULE
	SEG5.PMA	SEGMENT 5 -- SUPERVISOR DYNAMIC LINK TABLE (GATE SEGMENT)
	SEGAC$.FTN	SUBROUTINE TO SET SEGMENT ACCESS
	SEM$CA.PLP	Named semaphore - close all semaphores at LOGOUT time.
	SEM$CL.PLP	Named semaphore - close an open semaphore.
	SEM$DR.PLP	Named semaphore - drain a semaphore.
	SEM$NF.PLP	Named semaphore - notify a semaphore.
	SEM$OP.PLP	Named semaphore - open a semaphore associated with filename.
*	SEM$OU.PLP	Named semaphore - open and initialize a semaphore.
	SEM$ST.PLP	Named semaphore - report status of semaphores.
	SEM$TN.PLP	Named semaphore - set a timer for a semaphore.
	SEM$TS.PLP	Named semaphore - test value of a semaphore.
	SEM$TW.PLP	Named semaphore - wait on a semaphore and timer.
	SEM$WT.PLP	Named semaphore - to wait on a semaphore.
	SEMUTL.PLP	Named semaphore - utility routines.
	SEMVQA.PLP	Named semaphore - add a process to a virtual sem queue.
	SEMVQR.PLP	Named semaphore - remove a random process from a sem VQ.
	SEMVQS.PLP	Named semaphore - remove top process from virtual sem que.
	SETCPU.PMA	LOCK/UNLOCK PROCESS TO MASTER CPU.
*	SHDN$$.FTN	SHUTDN DISK LOCALLY AND REMOTELY.
	SHRLIB.FTN	INSTALL SHARED LIBRARY (RESTRICTED TO USER <SUSR>)

SHUTDN.FTN	SHUTDOWN COMMAND PROCESSING FOR PRIMOS IV.
* SID$GT.PLP	Get Spawner's Id
SMSG$.FTN	Send a message to a user on an arbitrary node.
* SORO$.PLP	INVOKES LIST OF RING ZERO STATIC ON-UNITS
* SPAWN$.PLP	Spawn a new process(some attributes specified by spawner).
SRPHAN.PLP	Apply suffix search conventions for phantom logins
SRWREC.FTN	SVC HANDLER FOR RREC,WREC SVC.
* STKINI.FTN	INITIALIZATION OF RING 0 STACK SEGMENTS.
STNOU.PMA	SVC-PCL INTERLUDES TO TNOU, TNOUA
SUPSTK.PMA	UNWIRED RING 0 STACK FOR USER 1.
SVCAL$.PMA	MISCELLANEOUS SUPERVISOR ENTRIES.
T$AMLC.PLP	Raw data mover for amlc lines.
T$CMPC.FTN	I/O TO CARD READER/PUNCH VIA MPC
* T$GPPI.PLP	General purpose parallel interface routine.
T$GS.PMA	DRIVER FOR VECTOR GENERAL GRAPHICS TERMINALS
T$LMPC.FTN	LINE PRINTER OUTPUT VIA MPC
T$MG.PMA	DRIVER FOR SOC-MEGRAPHIC 7000 INTERFACE
T$PMPC.FTN	CARD PUNCH I/O VIA MPC
T$TM.PMA	PRIMOS DIRECT-CALL HANDLER FOR TAG MONITOR
T$VG.FTN	VERSATEC-GOULD PLOTTER I/O
TA$.FTN	SUBROUTINE TO ATTACH TO A DIRECTORY CHAIN
* TDUMPC.PMA	Define the symbol TDUMPC and cause seg to allocate space.
TFLADJ.PLP	Adjust size of tfliob buffers
TFLIO$.PMA	LOGICAL I/O BUFFERING ROUTINES.
* TI$MSG.PLP	Print connect, cpu, and i/o time utilization.
TIMDAT.PMA	DATE AND TIME CONVERSION ROUTINES.
TMAIN.PMA	CLOCK PROCESS, RING 0 UTILITY SUBRS.
* TP$CON.PLP	Terminal-Process connect amlc line
* TP$DIS.PLP	Terminal-Process disconnect for amlc lines
TPIOS.FTN	PAGE TURNING INTERLUDE TO DISK I/O.
* TTY$IN.PLP	Check if there are any characters in input buffer for user.
TTY$RS.FTN	RESET TTY BUFFERS OF USER PROCESS
TTYPER.PMA	TYPERS FOR PRMOS4
TUTILS.PMA	RANDOM SUBROUTINES
UID$BT.PLP	Generate unique id as a bit string.
UID$CH.PLP	Generate a unique identifier as a character string.
ULOKPG.FTN	UNWIRE AN AREA OF THE VIRTUAL MEMORY.
* UNO$GT.PLP	Get the id's associated with this user.
USER$.FTN	Retreive ring0 data.
* USNMT$.PLP	Unassign magnetic tape drive units.
* USRAS$.FTN	Process the USRASR command.
UTIL$.PMA	UTILITY SUBROUTINES FOR FORTRAN PROGRAMS.
* UTYPE$.PLP	Function to return type of user (normal, remote, phantom)
VERDIM.PMA	PRIMOS 4 DRIVER FOR SOC INTERFACE
WAITIN.PMA	WAIT WITH PROCESS EXCHANGE INHIBITTED.
* WARMST.PMA	IS A WARM STARTABLE HALT ROUTINE.
WIRSTK.FTN	Procedure to wire the page fault stack for a process.
* WRL$.PLP	Get ptr to SOU lists.
WRMABT.FTN	HANDLE WARM START PROCESS ABORT.

Index of files in PRIMOS>FS - Primos file system.

'*' in column 1 indicates file did not exist at Rev.18

*	AC$CAT.PLP	Place an object into an access category.
*	AC$DFT.PLP	Protect an object with default access rights.
*	AC$LST.PLP	Return the contents of an ACL in logical format.
*	AC$RVT.PLP	Revert an ACL directory to password protection.
*	AC$SET.PLP	Create an ACL.
*	ACC_CHK.PLP	Handle access checking for access-setting routines.
*	ACDECODE.PLP	Decode a physical ACL entry into a logical one.
*	ACENCODE.PLP	Encode logical <id>:<access> pair into physical ACL entry.
*	ACLSEG.PMA	ACL system databases.
*	AC_CLEAN.PLP	Common cleanup for ACL gates.
*	AC_DELPA.PLP	Delete a priority ACL for a specified logical device.
*	AC_NEWPA.PLP	Add a new priority ACL to the specified LDEV.
*	ADD_ENT.PLP	Add a new entry to a directory.
*	ADD_REC.PLP	Extend a file.
*	ALC_REC.PLP	Allocate record(s) for new directory entry.
*	AT$.PLP	Attach to the specified pathname.
*	AT$ABS.PLP	Attach to a top-level directory on a specified partition.
*	AT$ANY.PLP	Do an attach scan.
*	AT$HOM.PLP	Set current attach point to be same as home.
*	AT$OR.PLP	Set home and/or current attach points to be same as initial.
*	AT$REL.PLP	Attach relative to the current attach point.
*	ATCH$$.PLP	Writearound for new attach modules.
*	ATLIST.PLP	Do a local attach scan on a specified list of disks.
*	AT_ADREM.PLP	Set unit table entry for attach point just gone remote.
*	AT_CLEAN.PLP	Common cleanup for attach modules.
*	AT_UNREM.PLP	Invalidate remote attach point(s).
*	AT_VALPAR.PLP	Validate key and directory name for AT$ routines.
*	BENSHT.PLP	Handle a unit on a device which has been shut down.
*	CALAC$.PLP	Calculate accesses available on a named object.
*	CALACS.~~PLP~~	Calculate accesses.
*	CAT$DL.PLP	Delete an access category.
	CLOSE.FTN	CLOSE A FILE BY NAME OR UNIT
*	CNAM$$.PLP	Change the name of a file system object.
*	CO$GET.FTN	Get ring0 data for invoking CLOSE and COMOUTPUT commands.
	COMI$$.FTN	COMINP-UT COMMAND AND SVC HANDLING
	COMO$$.FTN	SWITCH COMMAND OUTPUT ON/OFF
*	COPY_AP.PLP	Copy one attach point to another(handles hashing and quotas)
*	COPY_UTE.PLP	Copy one unit table entry to another.
*	CREA$$.PLP	Create a directory in the current directory.
*	DEL_ENT.PLP	Remove a directory entry.
*	DIR$RD.PLP	Read physical directory entries.
*	EMPTY_CK.PLP	Make sure the object whose BRA is passed may be deleted.
*	ENTINDIR.PLP	Attach to directory, return entry name in it.
	ERRCOM.PMA	STD. SYSTEM ERROR MESSAGE TABLE.
	ERRPR$.FTN	PRINT SYSTEM ERROR MESSAGE
*	FIL$DL.PLP	Delete a file or directory.
*	FIND_ENT.PLP	Find entry in directory specified by the unit table entry.
*	FIND_HOLE.PLP	Find first available hole of required size in a directory.
	FORCEW.FTN	FORCES DISK UPDATE.
*	FREE_REC.PLP	Free a file's records when it is deleted.
*	FSAHSH.PLP	Add unit table entry to file system and/or ACL hash threads.
	FSHASH.PMA	Calculate the hash index for the unit table
*	FSUHSH.PLP	Remove unit table entry from FS and/or ACL hash threads.
*	GETDV$.PLP	Return logical device number given unit number.
*	GETID$.PLP	Returns a user's complete ID (user id plus group ids).
*	GETQB.FTN	FUNCTION TO RETURN POINTER TO FREE QUOTA BLOCK.

	GETREC.FTN	GET A RECORD FROM DISK RAT.
*	GETUN.PLP	Allocate a unit table entry from the system-wide pool.
*	GET_LDEV.PLP	Convert partition name to logical device number.
*	GPAS$$.PLP	Read passwords on named directory.
*	GPATH$.PLP	Return a pathname given a unit or attach point.
*	GPDEV$.PLP	Return a physical device number given logical device number
	GSG$RA.FTN	Return segdir entry number by matching BRA in record LOCATEd b
*	GTUTBL.FTN	Allocate a unit table.
*	GUF$RA.PLP	Get dir entry from BRA in dir defined by LOCATE buf.
*	ISACL$.PLP	Indicates whether specified unit is an ACL directory.
*	KICKQB.PLP	Increment quota block use count for a subtree.
*	LDISK$.PLP	Return a list of disk names.
*	LDSKU$.PLP	List all users using a given ldev.
	LISTF.FTN	LIST DIRECTORY DRIVER
	LISTFT.FTN	LOAD A BUFFER WITH LISTF TEXT
	LOCATE.PMA	PRIMOS FILE SYSTEM ASSOCIATIVE BUFFERING.
*	LUDSK$.PLP	Return a list of all disks in use by a given user.
	M2SMA$.FTN	Return Master-to-Slave mapping for remote file unit.
	MARKUT.FTN	MARKS UNIT TABLE ENTRIES ON A DISK ERROR.
*	MKUTEPTR.PLP	Make a pointer to the unit table entry of the given unit.
	MOVNAM.PMA	Move names between two fields
	NAMEQ$.FTN	COMPARE TWO NAMES FOR EQUIV (RET TRUE IF SAME)
	NEWDAM.FTN	ADD RECORD TO NEW PARTITION DAM FILE.
*	NEW_ACL.PLP	Process addition of a new ACL to a directory.
*	OPEN_CHK.PLP	Check to see whether or not a file unit is open.
*	PA$DEL.PLP	Delete a priority ACL.
	PK2LDV.FTN	Convert disk pack name, node number in to an LDEV
	PRWF$$.FTN	READ, WRITE, POSITION SAM OR DAM FILES
*	Q$READ.PLP	Read quota information for current directory.
*	Q$SET.PLP	Set quota fields on specified directory.
*	Q$TRWK.PLP	Count records used in a subtree.
*	Q$UPDT.FTN	UPDATES DIRECTORY HEADERS WITH QUOTA DATA
*	R/W_ENT.PLP	Read or write the directory entry at the specified position.
*	RA2PTH.FTN	Return PATHNAME : <disk_name>tree_name based on BRA and LDEV.
*	RDEN$$.PLP	Writearound for RDEN$$ gate.
	RDLIN$.FTN	READ A LINE FROM A FILE.
	RDLN$X.PMA	SUBROUTINE TO EXPAND LINE READ FROM FILE.
	REST$$.FTN	RESTORE SAVED MEMORY IMAGE FILE.
*	RTNQB.FTN	SUBROUTINE TO RETURN QUOTA BLOCK.
	RTNREC.FTN	RETURN A RECORD TO DISK RAT.
*	RTNUN.PLP	Return a unit table entry to the global pool.
*	RTUTBL.FTN	Return a user unit table to the system free pool.
*	RVKID$.PLP	Revokes indices AGTIDX into AGT for given user.
	RWLKCK.FTN	CHECK UNIT TABLES FOR CONFLICT WITH SPECIFIED FILE
*	SATR$$.PLP	Set attributes for specified file.
	SAVE$$.FTN	Save memory image
	SEG10.PMA	USER COMMON AND FILE UNIT TABLES.
*	SEMSEG.PMA	NAMED SEMAPHORE DATA AREA
*	SETID$.PLP	Adds a group into the specified user's Active Group List.
*	SET_DTM.PLP	Set date/time modified of file entry to current date/time.
*	SET_OR.PLP	Set initial attach point (origin).
*	SET_QMOD.PLP	Set modified bit in a quota directory block.
*	SGD$DL.PLP	Delete a segment directory entry.
	SGDR$$.FTN	MANIPULATE SEGMENT DIRECTORY (OPEN STATUS DEMANDED):
*	SPAS$$.PLP	Set passwords on current directory.
	SRCH$$.FTN	Open,close,delete,change access,check existence of files.
	SRCH$R.FTN	FAM II FS CODE FOR OPEN-CLOSE-DELETE FILE SYSTEM PRIMITIVE
*	SYS_OPEN.PLP	Open a directory on the system unit or some other unit.
	TEXTCK.PMA	TESTS FOR A VALID 6-CHARACTER FILE NAME
	TRUNC$.FTN	TRUNCATE FILES.

```
  TRWRAT.FTN        STARTUP/SHUTDN FILE DEVICE
* UKCKGB.PLP        Decrement quota block use count for a subtree.
* UTALOC.FTN        Initial set up of unit table and other units for a user.
* UTDALC.FTN        Initial set up of unit table and other units for a user.
* UTESEG.PMA        Unit table entries and common area.
* VINIT$.PLP        Subroutine to initiate a VMFA segment.
  WTLIN$.FTN        WRITE A LINE TO A FILE.
  WTLN$C.PMA        SUBROUTINE TO COMPRESS LINE WRITTEN TO FILE.
```

Index of files in PRIMOS>R3S - Primos Ring 3 code.

'*' in column 1 indicates file did not exist at Rev.18

```
  $CALLS.FTN         Interludes to old_style calls
  ABBREV.PLP         This is the internal command for abbreviations.
  AB_FILE_.PLP       This is the routine to handle file i/o for abbreviations.
  AB_GET_.PLP        Get next whole token from command line, processing abbrevs.
  AB_PCS_.PLP        This is the routine to expand abbrevs.
* AC$CHG.PLP         Modifies the contents of an existing ACL
* AC$LIK.PLP         Set ACL on one file to be like that on another.
* AC$PAR.PLP         Parse an access control list.
* ADD_REMID_.PLP     Process the add_remote_id command.
  ALOC$S.PMA         ALLOCATE STORAGE ON THE STACK (FREE ONLY BY PRTN).
  APPEND.PMA         APPEND --- CONCATENTATE TO VARING STRING
  APSFX$.PLP         Append suffix to a pathname according to standards
  AREA_MAN.PLP       This is a general PL/I Area Manager.
  ASTRSK$.PLP        * Command
* ATCH_.PLP          Invoke the ATTCH command from ring3.
  BIN$SR.PLP         Do a binary search using pointers in a single segment.
* BINARY_.PLP        BINARY Command.
  CH$FX1.PMA         CHARACTER TO FIXED BIN(15,0) AND FIXED BIN(31,0) CONVERTERS.
  CH$OC2.PMA         CHARACTER (OCTAL) TO FIXED BIN(31,0) CONVERTER.
* CHANGE_PW.PLP      Command to allow a user to change his/her login password.
  CL$GET.PLP         Gets A Command Line Into User's Buffer
  CL$PAR.PLP         Parse string according to basic "command line" rules.
  CL$PIX.PLP         Parse command line according to a picture specifier.
* CLOSE_.PLP         Check cmdl syntax and call SRCH$$ to close file units.
* CLRLV_.PLP         Clear the existing level.
* CNAME_.PLP         Invoke the CNAME command from RING3...via GATE CNAM$$.
  CNIN$.PLP          Reads A Number Of Characters From Command Input Device
  CNSIG$.PLP         Set continue_sw on in most recent fault frame.
  COMANL.PLP         Writearound To CL$GET.
  COMLV$.PLP         Call a new command level.
  COMO$.PLP          COMOUTPUT Command.
  COND_CALLS.PMA     ADDITIONAL ENTRY POINTS FOR THE CONDITION MECHANISM.
  CP$.PLP            Invoke the user's currently specified command processor.
* CP_ITER.PLP        Command language iteration processor.
  CRAWL_.PLP         Perform crawlout from inner ring, rejoin signl$ or fim_.
* CREATE_.PLP        Invoke the CREATE command from RING3...via GATE CREA$$.
  CRFIM_.PMA         CRAWLOUT FAULT INTERCEPTOR RE-SIGNL$ IN THE OUTER RING.
  DB$MOD.PLP         Set/reset debugger-mode switch and static on-unit.
  DBG_.PLP           Internal command writearound to the DBG external command.
* DCOD_ITR.PLP       Decode command language extended feature token type.
  DEF_GV.PLP         Command to define global variables file to command env.
* DELAY_.PLP         Invoke the DELAY command from ring3.
  DELETE_VAR.PLP     Delete global variables
* DELSEG_.PLP        Process the DELSEG command.
  DET$GET.PLP        Get msg from a Diagnostic Error Table.
  DF_UNIT_.PLP       System Default On-Unit (includes PL/I runtime support).
* DISLV_.PLP         Display the current contents of a user's level.
  DUMPS_.PLP         Dump stack in a pretty format.
* EDIT_ACC_.PLP      Process the edit_access command.
  EDIT_CL.PMA        EDIT COMMAND LINE TO REMOVE EXPLICIT NULL STRINGS.
  ENDPAGE_.PLP       PL/I runtime support for ENDPAGE condition
* EQUAL$.PLP         Generate name from an object (source) name and a pattern.
* EQUAL$P.PLP        Append pathname generated from equalname to a given string.
  ERRSET.PMA         ERRSET INTERLUDE FOR SEGMENTED MODE
  EXIT.PLP           Exit from Static Mode, and return to Recursive Mode.
  FATAL_.PMA         GENERATE FATAL PROCESS ERROR.
```

FILL$A.FTN	FILL ARRAY WITH LITERAL
FINDPROC.PMA	FIND NAME AND ADDR FOR DF_UNIT_ PL/I CONDITION MESSAGES
* FIND_UID.PLP	Find a <user_id> in a validation file.
* FNCHK$.PLP	Check the string passed for validity as a file system name.
FNDCF$.PLP	Find most recent condition frame.
FNONU$.PLP	Find onunit in specified stack frame.
GATEQU.PMA	EQU'S INTO SEG5 (GATE SEGMENT)
* GET_FR.PMA	Get field address registers and floating point registers.
GS_FAC.PMA	GET/SET FP ACCUMULATOR FROM A FAULT FRAME REGISTER BLOCK.
GT$PAR.PLP	Parse string according to four types of characters.
GV$GET.PLP	Get the value of a global variable
GV$SET.PLP	Set the value of a global variable
* HASH_UID.PLP	Hash a <user_id>.
ICMTB_.PMA	INTERNAL (OLD AND NEW) COMMAND TABLE.
* IDCHK$.PLP	Check a (user or project) id for legality.
INFIM_.PMA	CRAWLOUT "FIM" FOR INIT$3 (INITIALIZE RING 3 ENVIRONMENT).
INIT$3.PLP	Initialize ring 3 environment
* INIT$P.PLP	Invoke initial routine (cominput, CPL, EPF, etc.) at login
INPUT$.PLP	INPUT Command.
INTCM_.PLP	Fetch local command table entry if any, else check system's ta
INVKSM_.PLP	Invoke (or restore) static mode program image.
IOA$.PMA	INTERLUDE TO CALL THE IOA$ FORMATTER. (IOA$, IOA$RS, IOA$ER).
IOAFM$.FTN	FORMATTING PACKAGE FOR IOA$.
IOAGA$.PMA	IOAGA$- GET ARGUMENT ROUTINE FOR IOAFM$
* IOAGD$.PMA	This module does an unsigned long divide.
* ITR_WLDC.PLP	Perform command language Wildcard Iteration.
* ITR_WLDT.PLP	Perform command language Treewalk Iteration.
LIBTBL.PMA	LIBRARY TABLES.
LISTEN_.PLP	Primos command loop standard Listener module.
* LIST_ACC_.PLP	Process the list_access command.
* LIST_ACL.PLP	Print the contents of an ACL on the terminal.
* LIST_GROUP.PLP	List the user's active and/or inactive groups.
* LIST_PA_.PLP	Process the List_priority_access command.
* LIST_QUOTA.PLP	Process the LIST_QUOTA command.
* LIST_REMID_.PLP	List one or all ID's used by this user on remote nodes.
LIST_VAR.PLP	List global variables and their values.
* LOGIN_.PLP	Handle LOGIN command from ring 3 (user already logged in).
* LOGOUT_.PLP	Logout command processor.
* LON$.PLP	Logout Notification Command
MISSIN.PMA	HANDLE MISSING ARGS IN V-MODE.
MKON$F.PLP	FTN interface to make an on-unit in caller's frame.
MKONU$.PMA	MAKE AN ON-UNIT IN THE CALLER'S STACK FRAME.
* MKSON$.PLP	Make a static on-unit for either ring.
* MOVWDS.PMA	DATA MOVEMENT SUBROUTINES.
* NEWLV$.PLP	Module to create a new level within the command environment
OCALLS.FTN	OLD PRIMOS SUBROUTINE CALLS
ONDISP.PLP	Display onunit data in a specific frame.
* OPEN_.PLP	OPEN Command.
* ORIGIN_.PLP	Command to return to initial attach point.
P$EPAGE.PLP	Write end of page text to a PL/I file(PL/I runtime support).
PHANTOM$.PLP	PHANTOM Command.
PL1$NL.PLP	Nonlocal goto processor.
PM$.PLP	Post Mortem command.
PRERR$.PLP	PRERR Command
PREVSB_.PLP	Find previous stack frame, given ptr to current.
PRTN_.PMA	VARIOUS FLAVOURS OF "RETURN" FOR USE BY THE UNWIND_ ROUTINE.
* PWCHK$.PLP	Check a password for legality.
* Q$SIZE.PLP	Return tree used for a directory subtree.
* QUTFIM.PMA	Ring 3 QUIT FIM-Invoke QUIT Condition In Ring 3.
* R$ALLC.PLP	EPF linkage allocation routine

*	R$CPF.PLP	Get command processor flags from an epf.
*	R$DEL.PLP	Delete an epf program.
*	R$INFO.PLP	return info about a desired epf file.
*	R$INIT.PLP	EPF linkage initialization routine
*	R$INVK.PLP	Routine to start the execution of an EPF
*	R$MAP.PLP	EPF file mapping routine
*	R$RELC.PLP	ERP: Epf Relative Pointer relocation routine
*	R$RUN.PLP	Run an EPF : Executable Program Format file
	R3FALT.PMA	RING 3 FAULT CATCHER.
	RAISE_.PLP	Search stack for onunit for condition, and invoke it.
	RDTK$$.PLP	Writearound to rdtk$p for use by static mode programs.
	RDTK$P.FTN	READ NEXT TOKEN FROM COMMAND LINE
	RDTKN$.FTN	USER CALLABLE ENTRY FOR RDTK$$ (OLD STYLE)
	RDY_.PLP	Set user's ready message mode(s).
	READY$.PLP	Print "ready" message on terminal.
	REENT_.PLP	Signal the condition REENTER$ for subsystem reentry.
*	REM_PA_.PLP	Process the Remove_priority_access command.
	RESTO_.PLP	Internal command "restore": load memory image of SM program.
	RESU$$.PMA	WRITEAROUND FOR RESU$$ CALL.
*	RLSLV$.PLP	Module to restore a level within the command environment
	RLSTK_.PLP	Generate the Listener Order "release stack".
	RMODE_.PLP	Return into Static Mode program, as defined by an "rvec".
	RSTERM.PLP	Command interface to reset terminal i/o buffer(s).
	RVONU$.PLP	Revert an onunit in caller's or given activation.
*	RVSON$.PLP	Remove static on-unit.
	SAVE$.PLP	Save a portion of memory as a file.
	SETRC$.PLP	Set Static Mode error code.
	SETREG.PMA	SETREG, GETREG -- SET, RETRIEVE REGS IN SVEC
*	SET_ACC_.PLP	Process the set_access command.
*	SET_PA_.PLP	Process the Set_priority_access command.
*	SET_QUOTA.PLP	Command to change quota or create a quota directory.
	SET_VAR.PLP	Internal command equivalent of &set_var CPL directive
	SIGNL$.PLP	Signal a specific condition.
	SNAP$3.PMA	FIND RING 3 ENTRY POINT FOR POINTER FAULT HANDLER.
*	SOR3$.PLP	Invoke ring 3 static on-unit.
*	SOUR3_.PLP	Find static on-unit list for ring 3.
	SRSFX$.PLP	Perform tree search, with or without suffix standard
	SRVEC_.PLP	Set Static Mode "rvec" from a fault frame.
	SS$ERR.PLP	Used by subsystems when they have run into an error.
	START_.PLP	Internal command "start": restart recursive or static mode.
	STD$CP.PLP	Standard Command Processor.
*	STK_EX.PLP	Handle auto stack extension.
*	STR$AL.PLP	Temporary storage allocation routine
*	STR$FR.PLP	Temporary storage free routine
	TALOC.PLP	Allocate large storage area
	TEMP$A.FTN	OPEN UNIQUE TEMPORARY FILE ON CURRENT UFD
*	TEXTO$.PLP	Check a character string for validity as a filename.
*	TIME_.PLP	Process the TIME command.
*	TNCHK$.PLP	Checks a character string for being a legal treename.
	TSRC$$.FTN	OPENS FILE WITH SPECIFIED TREENAME
	TYPE.PLP	Type text at a user's terminal.
	UNWIND_.PLP	Prepare the stack for nonlocal-goto-induced unwinding.
	USERS$.PLP	USERS Command
	VLIST.PMA	VLIST
	WILD$.PLP	Match wildcard name.
	XIS.PMA	XIS UNIMPLEMENTED INSTRUCTION EMULATOR

Index of files in PRIMOS>CPLS - Primos Command Procedure language.

'*' in column 1 indicates file did not exist at Rev. 18

	AFTER_AF.PLP	'after' active function for CPL.
	ALLOC_VAR.PLP	Allocate an extension area for variables
	ATTRB_AF.PLP	Get certain file attributes (command function).
	BEFORE_AF.PLP	'before' active function for CPL.
	CALC.PLP	CALC.PLP, PRIMOS>CPLS, PRIMOS GROUP, 01/07/82
	CH$HX2.PMA	CHARACTER (HEX) TO FIXED BIN(31,0) CONVERTER.
	CND_INFO_AF.PLP	condition_info a.f.: retrieve selection cond. info.
	COM_ABRV.PLP	Interlude to invoke command abbreviation processor.
	CPL.PLP	Interface CPL interpreter to command level.
	CPL_.PLP	Command Procedure Language Interpreter.
	CPL_ET_.PLP	Return pointer to CPL Error Table pathname.
*	CV$DQS.PLP	Convert FS format date/time to quadseconds since Jan1 1901.
	CV$DTB.PLP	Convert Date from ASCII to Binary (file system) format.
	CV$FDA.PLP	Standard fs date-time-mod converted to format mm/dd/yy hhmm.t
	DATE_AF.PLP	Date Command (Function).
	DIR$LS.PLP	Retrieve info about selected entries in a given directory.
	DIR_AF.PLP	'dir' active function for CPL.
	ENTRY_AF.PLP	'entry' active function for CPL.
	EVAL_AF.PLP	Active function evaluator for CPL
	EVAL_AN_EXPR.PLP	Evaluate expression containing variables, functions
	EVAL_VBL.PLP	Evaluate character string containing local/global variables
	EXISTS_AF.PLP	EXISTS command function for CPL.
	EXTR$A.PLP	Extract pathname components.
	EXT_VBL_MAN.PLP	External Variable Manager for Primos Command Loop.
	FROM_DECIMAL.PLP	Convert a decimal integer to an integer in a given base.
	GET_EXPR.PLP	Accumulate the next expression from the current line.
	GET_LINE.PLP	Get a new logical line from file on cpl_unit
	GET_REPLY.PLP	Fetch a yes/no/null/next reply from command input stream.
	GET_TOKEN.PLP	Get next token from CPL program
	GET_VAR_AF.PLP	get_var command function for CPL.
*	GVPATH_AF.PLP	Return pathname of current global variable file.
	GV_PTR_.PLP	Get pointer to global variable area.
	HEX_AF.PLP	Convert hexadecimal integer to decimal integer
	ICPL_.PLP	Invoke CPL interpreter on given file, processing suffix.
	ID_CHECK.PLP	Check a string for valid command var identifier format.
	INDEX_AF.PLP	'index' active function for CPL
	LENGTH_AF.PLP	'length' active function for CPL.
	MOD_AF.PLP	Implement mod function for CPL.
	NULL_AF.PLP	'null' active function for CPL.
	OCTAL_AF.PLP	Convert octal integer to decimal integer
	OPEN$B.PLP	Open a branch by tree name (nonstandard)
	OPEN_FILE_AF.PLP	open_file command function for CPL.
	PATHNAME_AF.PLP	Pathname command function for CPL.
	QUERY_AF.PLP	Query command function - get yes/no answer.
	QUOTE_.PLP	Perform a quote operation on a given string.
	QUOTE_AF.PLP	Perform quote operation for CPL active function.
	READ_FILE_AF.PLP	read_file command function for CPL.
	RESCAN_AF.PLP	Rescan command function for CPL.
	RESPONSE_AF.PLP	Response command function - get textual answer.
	SEARCH_AF.PLP	'search' active function for CPL
	SET_A_VAR.PLP	Set local and global user variables
	SIZE$B.PLP	Return the size of a branch in WORDS.
	SUBSTR_AF.PLP	'substr' active function for CPL
	SUBST_AF.PLP	Substitute command (function).
	TEST_EQUALS.PLP	Test expression equality for CPL.
	TO_HEX_AF.PLP	Convert a decimal integer to a hexadecimal integer.

TO_OCTAL_AF.PLP	Convert a decimal integer to a octal integer.
TRANSLATE_AF.PLP	'translate' active function for CPL.
TRIM_AF.PLP	'trim' active function for CPL.
UNQUOTE_AF.PLP	Perform unquote active function for CPL.
VBL_MAN.PLP	Variable manager for dynamically allocated string vars.
VERIFY_AF.PLP	'verify' active function for CPL
WILD_AF.PLP	"wild" command function, get list of files by wildcard name.
WRITE_FILE_AF.PLP'	'write_file function for CPL.

Index of files in PRIMOS>NS - Primos network code.

'*' in column 1 indicates file did not exist at Rev. 18

	ALCHCB.PLP	Allocate & initialize (to all zeros) a.host control block
	ALCMYL.PLP	Allocate & initialize my node's line definition table entry
	ALCNAM.PLP	Allocate & initialize (to all zeros) a name table entry
	ALCRNG.PLP	Allocate & initialize a ring line definition table entry
	ALCSLC.PLP	Allocate & initialize an SMLC line definition table entry
	COMDEF.PMA	NETWORK COMMON DEFINITIONS
	FAMMSG.FTN	INVOKE FAM IN THIS PROCESS
	FAMPRC.FTN	PRIVILEGED SVC FOR FAM
	FCPYRG.PMA	ARGUMENT COPYING AND RETURNING FOR FAMMSG
*	FNSID$.PLP	Search the DIFNS id structure for the id for a given node.
	GETVCIX.PLP	GETS AN INDEX INTO THE VCDATA FOR THIS USER
*	INIPNC.FTN	INITIALIZE RING, COLD START TIMER AND LINE TIMERS
	LKFA.PMA	LOCKFA
	LKTA.PMA	LOCKTA
*	N$LOGO.FTN	TELL NETWORK TO SEND FORCED LOGOUT MESSAGE TO REMOTE USER .
	NBKDEF.PMA	NETWORK NEW BLOCK AND QUEUE DEFINITIONS
	NBKINI.FTN	ROUTINE TO INITIALIZE NETWORK BLOCKS AND QUEUES
	NCMSUB.FTN	Initiates a HDX Primenet link.
*	NETABT.FTN	Main "work" loop for network process
	NETCM$.FTN	Handles 'NET' commands for HDX operator interface.
	NETDMP.PMA	USED TO TRACE ILLOGICAL SYSTEM FAILURES DURING PRIMOS OPERATION
*	NETDWN.PLP	Shuts down networks
	NETEV1.PMA	FIRST-LEVEL EVENT LOGGER (PCL-ABLE VERSION)
	NETEV2.FTN	SECOND-LEVEL EVENT LOGGER
	NETFIG.FTN	NETWORK COLD START CONFIGURATION MODULE
	NETMAP.PLP	Subroutine to manage segment mapping for networks
*	NETON.PLP	Turn network on
*	NETPRC.PLP	NETWORK PROCESS RUNNING IN RING 0
	NETRTN.PLP	Subroutine to invalidate network cache on RTNSEG
	NETSGS.PMA	COMMON DEFINITION FOR NETWORK MAPPED DATA MOVEMENT SUBROUTINES
	NETUTU.PLP	Subroutine to copy from Networks to user space
	NNITL.PMA	ALL THAT'S LEFT HERE IS A HALT (FOR FORTRAN STOPS)
	NPXPRC.FTN	THE RING 0 CALLS TO SUPPORT NPX (ANALOGOUS TO FAMSVC, FAMPRC)
*	NTINIT.FTN	Initialize the network
*	NTWMAB.PLP	Warm start code executed by the network process
	OLDFAM.FTN	CALLED BY R$CALL TO INVOKE FAM 1.
	OLDLSF.FTN	PROCESS 'LISTF' COMMAND FOR DOSSUB
	PNCDIM.PMA	HARDWARE INTERFACE FOR PRIMENET NODE CONTROLLER
	PRFTMR.FTN	TIMER FOR RING NETWORK PROTOCOL
*	PROALM.PMA	Indicate protocol required and notify network server process
	PRSMLC.FTN	LEVEL SMLC PROTOCOL FOR NETWORK, X.25
	R$ALOC.PLP	ALLOCATES A VCIX SLOT FOR NODE XRNODE
	R$CALL.PLP	USER CALLABLE INTERFACE TO NPX TO MAKE REMOTE PROCEDURE CALLS
*	R$CKVC.PLP	CALLED BY LOGABT TO CHECK NPX VIRTUAL CIRCUIT.
	R$RLS.PLP	DECREMENTS A PERNODE ALLOCATION COUNT FOR NPX.
	R$WHER.PLP	Return information on location of a file.
	REMOTE.FTN	DENY/PERMIT FOR DISKS, CALLED FROM DOSSUB
	RLOGIN.FTN	CONTROL USER PROCESS ON TERMINAL SIDE OF REMOTE LOGIN
	RNGRCV.FTN	LEVEL II PROTOCOL RECEIVE LOGIC FOR RING NETWORK
	RNGSND.FTN	LEVEL II PROTOCOL XMIT FOR HIGH SPEED RING NETWORK
	SLCNET.PMA	SMLC INTERRUPT STATUS HANDLER FOR X.25 LEVEL 2
	TRNRCV.PLP	TRANSMIT/RECEIVE MESSAGES TO AND FROM SLAVES IN ONE OPERATION.
	UPUS1.PLP	Subroutine to update.user status words
	UPUS2.PLP	Subroutine to update user status words
	UPUS3.PLP	Subroutine to update user status words
	X$ADCL.FTN	ROUTINE TO ADD DECLARATION TO DCL LIST

X$ADR.FTN	Modules to decode addresses from incoming calls
X$AGFI.FTN	ROUTINE TO DECLARE INTEREST IN GFI
X$CACP.FTN	ROUTINE TO ACCEPT A CALL
X$CLOK.FTN	BACKGROUND CLOCK FOR LEVEL 3 X.25 - SHOULD RUN EVERY 10 SECONDS
X$CLRA.FTN	ROUTINE THAT CAN BE USED TO CLEAR ALL CONNECTIONS A USER OWNS
X$COPY.FTN	ROUTINE TO COPY PACKET INTO AN UNWIRED BUFFER
X$CREQ.FTN	PROCESS AN INCOMING CALL REQUEST
* X$FCTY.PLP	Facilities parsing for call request/incoming call packets
X$FLDS.FTN	X$FLDS - Get all of the fields in a CREQ or ACCEPT packet
X$GBCD.FTN	X$GBCD - ROUTINE TO COPY BCD DIGIT STRING TO ASCII STRINGS
X$GETU.FTN	ROUTINE TO HANDLE OUTPUT PACKETIZING
X$GIVU.FTN	X$GIVU - ROUTINE TO TRY TO GIVE DATA PACKETS TO USER LEVEL
X$GVVC.FTN	PASS CONTROL OF A VIRTUAL CIRCUIT TO ANOTHER USER
X$HDWN.FTN	ROUTINE TO SHUTDOWN X.25 LEVEL 3 FOR A GIVEN HOST
X$IDNT.FTN	Routine to build a restart ID packet (rev 17.3+)
X$IPKT.FTN	TAKE INCOMING PACKETS FROM LEVEL II PROTOCOLS
X$LINK.FTN	Links network table entries for HDX on-the-fly configuration.
X$LOOP.FTN	ROUTINE TO PROCESS PKTS THAT START AND END IN THE SAME MACHINE
X$MAP.PMA	POINTRS TO IMPORTANT NETWORK STRUCTURES.
X$NORM.FTN	DECODE CMND BYTE AND DO ROUTINE WINDOW UPDATES
X$NTFY.FTN	WAIT ON AND KICK USER'S NETWAIT SEMAPHORE
X$PRIM.FTN	NETWORK PRIMITIVES
X$RLG.FTN	HANDLE USER SIDE OF REMOTE LOGIN
X$RLT.FTN	LOG-THRU MODULES - TERMINAL SIDE OF REMOTE LOGIN
X$RSET.FTN	ALLOW A USER TO CAUSE A RESET ON ON OF HIS VIRTUAL CIRCUITS
X$STAT.FTN	ROUTINE TO RETURN STATUS INFORMATION TO USER SPACE
X$USRQ.FTN	ROUTINE TO PUT VCB IN A USER'S QUEUE OF VCBS
X$UTIL.FTN	ALL OF THE NETWORK SOFTWARE UTILITY ROUTINES
X$XGFI.FTN	MOVE GFI'S TO AND FROM PACKETS
X25DEF.PMA	X.25 NETWORK COMMON DEFINITIONS (UNWIRED)
XLGC$.FTN	XLGC$ - GET ALL OF THE FIELDS IN A CONNECT REQUEST PACKET

Index of files in PRIMOS>NPXS - Primos Network Process Extension.

'*' in column 1 indicates file did not exist at Rev. 18

	ALLOC. PMA	ALLOCATES SPACE FOR TEMPS ON THE FLY FOR SLAVES
	CALLIT. PMA	THIS SUBR MAKES A DYNT AND CALLS IT(GIVEN PCL+ARGS).
	CIRLOG. PLP	STUFFS CIRCULAR BUFFER FOR DEBUG OF NPX
	EXTRAC. PLP	EXTRACTS A SPARE DATA FIELD FROM A REQ OR RESP MESSAGE
	MOVB. PMA	MOVES N BYTES FROM SRC 32 BIT POINTER TO DST POINTER
	NPXDNT. PMA	NPXDNT - THE DYNT TO GET NPXPRC DEFFINED FOR R$CALL
	R$CVT. PLP	CONVERTS A NODE NAME TO A NODE NUMBER
	SLAVE. PLP	GIVEN REQUEST MESSAGE, SLAVE CALLS TARGET SUBR, SENDS RESPONSE
	SLAVER. PLP	ROOT OF ALL SLAVE INVOKATIONS, ACCEPTS CALL, DEFS. 1ST MESS.
*	SLAVE_CK. PLP	Called by DF_UNIT_ to check usr type, U$NPX goto SLAVE_ON_UNIT
	STOPME. FTN	PRINTS ERROR AND STOPS NPX PHANTOM

ndex of files in PRIMOS>CS - Primos synchronous communications.

'*' in column 1 indicates file did not exist at Rev. 18

BSCMTR.PMA	PROTOCOL-SENSITIVE DIM CODE FOR THE 'BSCMAN' AND 'XBM' PROCESS.
CRFP.FTN	INTEGER*2 FUNCTION TO CREATE A FREE POOL
CRQ.FTN	INTEGER*4 FUNCTION TO CREATE A QUEUE
DMCDYN.FTN	RESERVES AND FREES DMC CHANNELS DYNAMICALLY FOR THE SLC USERS
FLSHFS.FTN	SUBROUTINE TO FLUSH FREE STORE
G$ALOC.PLP	Perform heap storage allocation for queueing routines
G$DALC.PLP	Perform heap storage deallocation for queueing routines
GSUBS.PMA	QUEUEING ROUTINES FOR NETWORK AND COMMUNICATION PRODUCTS
QUEDEF.PMA	QUEUEING ROUTINES COMMON DEFINITION
S0MAN.FTN	ALLOCATES 1-PAGE WINDOWS IN SEG. 0 FOR COMMUNICATIONS PROCESSES
SLABRT.FTN	ABORTS SMLC ACTIVITY FOR A GIVEN LINE
SLBSMR.FTN	INITIALIZES "BSCMR" WORKSPACE BEFORE A RECEIVE.
SLCCMP.PMA	UNPACKS SMLC STATUSES TO LINE PAIR BUFFERS HANDLES INT STATUS
SLCDIM.PMA	DISTRIBUTES SYNCHRONOUS CONTROLLER STATUS - HAS I/O CALLS
SLCLDB.FTN	LOADS DRIVER TABLES FROM A CONTROL BLOCK
SLCNFG.FTN	CONFIGURES HSSMLC CONTROLLER AND SINGLE-BOARD SUCCESSORS
SLCTOP.PMA	LOCATES TOP OF HSSMLC DRIVER MODULES
SLERF.FTN	HANDLES SMLC ERROR MESSAGES
SLSCH.FTN	SETS UP DMC CHANNELS FOR A LOGICAL SMLC LINE
SMLCEX.FTN	TRANSFERS SMLC STATUS DATA FROM BASE TO USER LEVEL FOR 5300
T$SLC1.FTN	CONTROL BLOCK INTERPRETER FOR HSSMLC AND MDLC CONTROLLERS

Index of files in PRIMOS>RJES - Primos Remote Job Entry code.

'*' in column 1 indicates file did not exist at Rev. 18

*	GETCP. PLP	PH/WRK - returns pointer to area used to pass PH config
*	HASP. PLP	HASP protocol specific RJPROC code
*	HASPCK. PLP	HASP Protocol Specific Check module
*	PHDBG. PLP	PH - returns addresses of common area for protocol handler
*	READQT. PLP	routine reads entry off primos queue
*	RJ$ATT. PLP	RJI interface routine - allows process to attach for line
*	RJ$I. PLP	RJI routines return info to user from the protocol handler
*	RJ$MSG. PLP	RJPROC message returning routine
*	RJ$O. PLP	RJI routines will output blocks, control messages, detach line.
*	RJCDF. PMA	COMMON DECLERATIONS FOR RJE EMULATORS
*	RJCMTR. PLP	Configure MTR sub-process for protocol handler
*	RJCPY. PLP	RJI-PH - routine copies xmit blocks into wired xmit buffers
*	RJDBG. PLP	Debug gate returns pointer to RJI common blocks for worker RJI
*	RJDLIN. PLP	Deconfigure line
*	RJEVNT. PLP	Event handler for the Rjproc system
*	RJGBDQ. PLP	RJI-PH routine - get a data block off a device queue
*	RJINI. PLP	Cold start code for RJE emulators
*	RJLINE. PLP	Low level routines for Rjproc
*	RJPCDF. PMA	protocol handler common declerations for rje emulators
*	RJPHFS. PLP	rje emulators - routine manages the dim free store area
*	RJPHLC. PLP	rje emulators - routine assigns a line control block
*	RJPHS. PLP	Modify protocol handler state in Worker RJI database
*	RJPLO. PLP	Logout code for protocol handlers
*	RJPMSG. PLP	RJPROC message printing routine
*	RJPROC. PLP	Main driver for RJE emulator process
*	RJQ. PLP	RJI queueing routines using RQCB
*	RJRBRQ. PLP	Copy contents of receive block and queue for the worker
*	RJRECV. PLP	Receive routines for RJPROC
*	RJRQST. PLP	Worker request processor for RJPROC
*	RJRTRY. PLP	Routines supporting RJPROC retry mechanism
*	RJSLCFG. PLP	Configure HSSMLC and MDLC for RJE use
*	RJTIM. PLP	Timer routines for the Rjproc system
*	RJTWKR. PLP	Send Messages to Ring3 Workers via RJI
*	RJUNDO. PLP	Logout code for RJE emulators.
*	RJWLO. PLP	Logout code for RJI workers.
*	RJWRFS. PLP	rje emulators - routine manages RJI system free store
*	RJWRLC. PLP	Routines assign and unassign control blocks for line
*	RJXMIT. PLP	Transmit routines for RJPROC
*	X80. PLP	X80 protocol handler
*	X80CK. PLP	X80 Protocol Specific Check module
*	XBM. PLP	XBM line events and timeouts
*	XBMCK. PLP	Determine type of message from MTR (XBM Link level processing)
*	XBMCOM. PMA	ALLOCATE SPACE FOR XBM CAT QUEUES

ndex of files in PRIMOS>DES - Primos DPTX code.

```
'*' in column 1 indicates file did not exist at Rev. 18

  ASSIST.PMA    SUBROUTINES TO MOVE AND CLEAR VIRTUAL BUFFERS FOR DPTX
  BD$ATT.FTN    BLOCK DEVICE 'ATTACH' SUBROUTINE
  BD$DET.FTN    BLOCK DEVICE DETACH SUBROUTINE
  BD$INF.FTN    BLOCK DEVICE INFORMATION & STATUS SUBROUTINE
  BD$INP.FTN    BLOCK DEVICE INPUT SUBROUTINE
  BD$LST.FTN    BLOCK DEVICE INTERFACE DESCRIPTION ROUTINE
  BD$OUT.FTN    BLOCK DEVICE OUTPUT SUBROUTINE
  BD$SET.FTN    BLOCK DEVICE ATTRIBUTE-SETTING SUBROUTINE
  BDFLSH.FTN    FLUSH BLOCK INPUT/OUTPUT QUEUES FOR A DPTX DEVICE
  BDICHR.FTN    INPUT CHARACTER FROM BLOCK DEVICE QUEUE ELEMENT
  BDIWRD.FTN    INPUT WORDS FROM BLOCK DEVICE QUEUE ELEMENT
  BDLDSO.FTN    LOAD 3270 SUPPORT OUTPUT INTO A QUEUE ELEMENT
  BDOWRD.FTN    OUTPUT WORDS TO BLOCK DEVICE QUEUE ELEMENT
  BDQUIT.FTN    QUIT PROCESSING FOR A DPTX COMMAND DEVICE
  BDUNDO.FTN    UNDOES ALL DPTX ATTACHMENTS OF A PROCESS
  BDVBIF.FTN    LOADS VB AND SOME PARAMETERS, AS PART OF BD$INF CALL
  BLDMSG.FTN    BUILDS CANNED MESSAGES FOR TRAFFFIC MANAGER
  BNDAID.FTN    AID BYTE ANALYSIS ROUTINE FOR TRAFFIC MANAGER
* BSCCDF.PMA    BSCMAN QUEUEING AND FREE STORAGE ALLOCATION
  BSCINI.FTN    CREATES FREE STORAGE POOLS AND QUEUES FOR BSCMAN AND DPTX
* BSCMAN.FTN    BSCMAN SENDS AND RECEIVES TEXT IN THE BSC PROTOCOL ... MORE OR LESS
* BSCMOV.PMA    MOVES CHARACTERS IN 64V MODE
* BSCSEM.FTN    OBTAIN SEMAPHORE FOR BSCMAN TO USE IN NOTIFYING A MATE
* BSCSHR.PMA    DEFINES STORAGE FOR BSCMAN VARIABLE INITIALIZED AT COLD-START ONLY
* BSCSLC.FTN    INITIALIZE THE SYNC CONTROLLER FOR BSCMAN
  CFI.FTN       PROGRAM TO CHECK IF ANY CHARACTER IN TERMINAL BUFFER
* CHAP.FTN      SETS A USER PROCESS TO A SPECIFIED PRIORITY LEVEL
  CHKTAT.FTN    CHECK TAT FLAGS FOR A DEVICE
  CKHOLD.FTN    MANAGES TAT HOLDING AREA FOR VBE
  CLNRB.FTN     CLEAN THE RB HEADER
  COPY.FTN      COPY COMMAND PROCESSING
  DH3270.FTN    DATA HANDLER INTERFACE TO TFLIOB BUFFERS FOR DPTX/TSF
  DHDBSC.PMA    DH3270 SPECIFIC SHORTCALL SCHAR EQUIVALENT
* DPSTAT.PMA    DEFINE COMMON AREA FOR DPTX STATISTICS MONITORING
* DPT$QM.PLP    QUEUE MONITOR SUBROUTINE FOR DPTX QUEUES
* DPT$ST.FTN    RETRIEVE RINGO INFORMATION FOR DPTX MONITOR
  DPTCDF.PMA    DEFINE COMMON AREAS FOR DPTX TABLES/VARIABLES
  DPTINI.FTN    SUBROUTINES TO INITIALIZE OR SHUT DOWN DPTX
* DPTNAM.FTN    DPTNAM CHANGES THE LOG NAME FOR DPTX PROCESSES
  EAU.FTN       ERASE ALL UNPROTECTED (EAU) COMMAND PROCESSING
  ECHONL.FTN    ECHO A "NEW LINE" TO A 3277 MOD 2 TERMINAL
  EM3270.FTN    MAIN PROGRAM FOR 3270 VIRTUAL BUFFER EMULATION
  EMCFGB.FTN    CONFIGURE DPTX/DSC SMLC LINE
  ERROR.FTN     SAVE INFO AND STOP ACTION (BSCMAN)
  FIXELM.FTN    INSERT APPROPRIATE KEYS IN A QUEUE STRUCTURE
  FMTSCR.FTN    REFORMAT AND CLEAR (OPTIONAL) 3277 SCREEN
  FNMONT.FTN    OUTPUTS ERROR AND STATUS MESSAGES FOR TM3270
  GETELM.FTN    BUILDS EMPTY QUEUE ELEMENT CHAIN
  HOLD.FTN      SAVE RESULTS FOR USER IN TAT
  LDTMQ1.FTN    LOADS A DATA BUFFER INTO A PREALLOCATED QUEUE ELEMENT
  LNKELM.FTN    LINK DB'S OF A QUEUE STRUCTURE (ROOT2) TO QUEUE STRUCTURE (ROOT1)
  LOADQ1.FTN    LOADS A DATA BUFFER INTO A PREALLOCATED QUEUE ELEMENT
  LOADQE.FTN    LOAD A DATA BUFFER INTO A QUEUE ELEMENT
  MESFAL.FTN    SEND MESSAGE FAILED STATUS TO USER FOR TM3270
  MSGVLD.FTN    MESSAGE VALIDATION FUNCTION FOR BSCMAN ROBUSTNESS
  RDBUFR.FTN    READ BUFFER COMMAND PROCESSING
```

	RDMODR.FTN	READ MODIFIED COMMAND PROCESSING
*	RETCDF.PMA	BSCMAN RETRY COMMON STORAGE ALLOCATION
*	RETRY.FTN	RETRY SUBROUTINES FOR BSCMAN
*	ROBCDF.PMA	BSCMAN ROBUSTNESS COMMON STORAGE ALLOCATION
	RTNELM.FTN	RETURNS ALL OR PART OF A QUEUE ELEMENT
	SENBDI.FTN	ENQUEUES A QUEUE ELEMENT FOR BLOCK USER INTERFACE
	SENBSC.FTN	ENQUEUES A QUEUE ELEMENT FOR BSCMAN
	SENDPH.FTN	ENQUEUE MESSAGE FOR PROTOCOL HANDLER
*	SETNOW.FTN	SETS TIMER USING VCLOCK(1) (BSCMAN)
	SS3270.FTN	ANALYZES SENSE AND STATUS BYTES FOR TRAFFIC MANAGER
	STTSND.FTN	SEND A STATUS MESSAGE TO A BLOCK DEVICE FOR TM3270
	TABLES.FTN	DATA FOR DPTX TABLE TRANSLATIONS
*	TBLINI.FTN	INITIALLIZES BSCMAN'S MESSAGE VALIDATION TABLE
	TM3270.FTN	MANAGES SYNCHRONOUS LINE TRAFFIC FOR PRIMOS 3270 TERMINALS
	TMCFGB.FTN	CONFIGURE TM3270'S BSC LINE
*	TMCLOK.FTN	RETURNS THE VALUES OF GCLOK, KUSR AND MPXSEM TO TM3270
	TMINIT.FTN	TM3270 INITIALIZATION ROUTINE
	TMRRE.FTN	DEVICE RECOVERY ROUTINE FOR TM3270
*	TMSTMP.FTN	PRINTS OUT A TIME STAMP WITHOUT A FOLLOWING CARRAGE RETURN
	TRCDEF.PMA	TM3270 COMMON AREA (DPTX)
	UNLDQE.FTN	UNLOADS A QUEUE ELEMENT INTO A DATA BUFFER
	VALBUF.FTN	CHECK USER'S OUTPUT BUFFER FOR ILLEGAL CONTROL CHARACTERS.
	VBGBDI.FTN	GET OUTPUT ELEMENT FROM BDI
	VBGBK.FTN	PERFORM 'GETBKC' CALLS FOR VBE
	VBINIT.FTN	INITIALIZES VIRTUAL BUFFERS FOR DPTX/DSC
	VBTMPL.FTN	BUILDS A VB UPDATE TEMPLATE FROM USER DATA
	VBUPDA.FTN	UPDATES VB FROM USER-SUBMITTED TEMPLATE
	VBVTAC.FTN	TACKS A VB COPY ONTO INPUT DATA
*	WORKRY.PMA	ALLOCATES WORKR$ AND ERRCTL COMMON AREAS
	WRITE.FTN	WRITE COMMAND GROUP PROCESSING
	XLATBF.PMA	ASCII-EBCDIC BUFFER TRANSLATION ROUTINE FOR DPTX
	XLCALL.PMA	CALLS XLATBF WITH BIT OFFSETS