I/O SYSTEM STATUS RETURNS

0 - Pending

1	Waiting for completion	1
2	Doing Error recovery	2
3	Not ready wait	3
4	No Write Ring wait	4
5	New Paper Tape wait	5

1 - Successful

0	Normal	1
1	Read terminated with special character	%11
2	Tape retry for success required	%21
3	Low tape or End of tape after write	%31

2 - End of File

1	Physical End of File	%12
2	DATA	%22
3	EOD	%32
4	HELLO	%42
5	BYE	%52
6	JOB	%62
7	EOJ	%72

3 - Unusual Condition

1	Terminal parity error	%13
2	Terminal read timed out	%23
3	I/O aborted externally	%33
4	Data lost	%43
5	Data Set not Ready or disconnect or Unit not on line	%53
6	Aborted because of power fail	%63
7	BOT and BSR, BSF request	%73
10	Tape runaway	%103
11	EOT and Write request	%113
12	No Write Ring after request to operator	%123
13	End of Tape (Paper Tape Low)	%133
14	Plotter limit switch reached	%143
15	Enable sus sys Break and no Control Y pin	%153
16	Read time returned overflow	%163
17	BREAK stopped read	%173
20	Write and no card in wait station	%203

4 - Irrecoverable Error

0	- Invalid request	4
1	- Transmission	%14
2	- I/O time out	%24
3	- Timing error	%34
4	- SIO failure	%44
5	- Unit failure	%54
6	- Invalid disc address	%64
7	- Tape parity error	%74
11	- Paper Tape tape error	%114
12	- System Error	%124
13	- Invalid Sbuf index	%134

TERMINAL I/O SPECIFICATIONS

The general meaning of the parameters to ATTACHIO is described in the ATTACHIO description. Particular circumstances and limitations are described below.

0 - READ

```
P1.(13:3) - End of File Specification

P1.( 0:1) - No RETURN or LINEFEED is to be emitted at the
            end of a line.

P2. (0:8) - If not zero, use as special Read termination
            character.

  .(11:2) - IF zero then ASCII else Binary read. Binary
            reads are only completed on satisfaction of the
            read count and no carriage control is output.

  .(10:1) - User Mode. If set, the terminal subsystem does
            not issue a second XON when a DC2, CR is detected,
            indicating the "enter" key has been pressed. If
            clear, the terminal subsystem outputs a second
            XON when a DC2, CR is detected, to read in the
            block of data.
```

1 - WRITE

```
P1          - Vertical Format Specification

0           - Output and single space (CR/LF)
1           - Use first data byte as Vertical Format Specifications
%53 "+"     - Output and RETURN (no LF)
%60 "0"     - Output and double space (CR/LF/LF)
%61 "1"     - Output and do form feed if applicable or CR/LF
%2000-"277- Output, RETURN and N-%200 LF's
%320        - Output (no CR or LF)
               All others result in output and CR/LF.

P2.(15:1) - Prespace Flag. If odd, then do space operation
            before output of data. If even, then do space
            operation after output of data. If the request
            preceding a post space request was a prespace
            request, a CR/LF is output before the post space
            request.

P2.(11:2) - If zero the ASCII else Binary write. All 8 bits
            are transfer with no parity and no carriage
            control is appended for binary writes.
```

2 - FILE OPEN

A CR/LF is output if the last operation did not end with a LINE FEED and the unit is on line.

3 - FILE CLOSE

Resets Tape-mode, disables read timer and timeouts, desables parity checking, enables parity generation, turns echo on and does a CR/LF if the last opertion did not end with a CR/LF.

4 - DEVICE CLOSE

Resets End of File flags, clears all mode and capability flags and sets the device back to the speed sensing mode. If the device was connected through a data set, it is "Hung-Up".

5 - SET READ TIMEOUT INTERVAL

P1 = 0 - Disable read timeouts.
   = 0 - Timeout interval in seconds.

6 - SET INPUT SPEED AND RETURN PREVIOUS SPEED

P1 - New speed in characters/second.

7 - SET OUTPUT SPEED AND RETURN PREVIOUS SPEED

P1 - New speed in characters/second.

8 - ENABLE ECHO AND RETURN IF ECHO WAS ENABLED ELSE 0

9 - DISABLE ECHO AND RETURN 1 IF ECHO WAS ENABLED 0

10 - DISABLE BREAK

11 - ENABLE BREAK

12 - DISABLE SUBSYSTEM BREAK (CONTROL Y)

13 - ENABLE SUBSYSTEM BREAK (CONTROL Y)

14 - DISABLE TAPE-MODE

- ENABLE TAPE-MODE. INHIBITS LINE FEED AFTER RETURN, "\" AFTER

17 - ENABLE READ TIMER

18 - RETURN READ TIME IN ONE HUNDREDTH'S OF SECONDS (A LOGICAL QUANTITY)

19 - DISABLE PARITY CHECKING OF READS

20 - ENABLE PARITY CHECKING OF READS

21 - LOGGED ON

Indicates that the user has successfully logged on and that the log on timeout is to be aborted. BREAK is also enabled.

22 - NOT USED

23 - TERMINAL TYPE

Sets the terminal type in the DIT and sets the LF and CR sync delays to the proper value associated with the type of terminal and the current speed.

24 - ALLOCATE TERMINAL

Sets the device type, speed and carriage control sync counts and the character size. This function allows the terminal to be set up without going through the speed sensing and log on procedure.

P1 - Terminal type as specified in the MPE ERS.
p2 - Speed in characters per second.

25 - CLEAR FLUSH AND WRITE

This function is the same as the write function, except the BREAK Flushing flag is cleared before the write request is initiated.

26 - DISABLE CONTROL X ECHO

Disables the "!!!", Carriage Return, Line Feed response when a Control X deleting a line is detected.

27 - ENABLE CONTROL X ECHO

Enables the "!!!", Carriage Return, Lire Feed response when a control X is detected.

28 - NO OPERATION. Returns invalid request status.

29 - PTAPE READ. Paper tape spooling, read into spooling buffers.
P1 - High order disk address
P2 - Low order disk address

NOTE: The IOQ number (IOQ index/IOQ size) or zero of the last two spooled buffer srites are returned in the transmission log word.

30 - SET BREAK MODE

P1 = 1 Sets a flage causing any requests queued to be queued with a prempt level higher than a normal request. Also causes any "broken read" saved data, not to be passed back as part of any reads.

P1 = 0 Clears the flag causing "broken read" data to be saved so that next read begins with any saved data alread "read".

31 - SET CONSOLE MODE

P1 = 1 Sets a flag causing any requests queued to be queued with a prempt level higher than Break or Normal requests. Disables BREAK and Control Y.

P1 = 0 Clears the Console Mode Flag and reenables BREAK and Control Y.

32 - SET PARITY

P1 = 0 - No parity generated.
   = 1 - No parity generated. Bit 8 always one.
   = 2 - Even parity generated on write and expected on read (default).
   = 3 - Odd parity generated on write and expected on read.

33 - ALLOCATE TERMINAL. Sets the device type, speed and carriage control sync counts and the character size. This function allows terminal to be set up without going through the speed sensing and log on procedure.

P1 - Terminal type as specified in the MPE ERS.
P2 - Speed in characters per second.

34 - TERMINAL TYPE. Sets the terminal type in the DIT and sets the LF CR sync delays to the proper value associated with the type of terminal and the current speed.

35 - RETURN TERMINAL TYPE. Returns the terminal type number as defined the MPE ERS in count word.

This device supports Chained System Buffers.

36 - RETURN OUTPUT SPEED. Returns the output speed in characters per second in the count word.

37 - SET STOP CHARACTERS. Sets a special "unedited" mode, where all characters are passed to the user except for an End of Record character and a Sub System Break character. These two edited characters cause essentially the same action as CR and control Y in normal mode.

P1.(0:8) - Sub System Break character
  .(8:8) - End of Record character

STATUS RETURNS:

General (13:3)	Qualifying (8:5)	Overall (8:8)
0 - Pending	0 - Not started or writing	0
1 - Successful	0 - Normal completion	1
	1 - Special character read termination	%11
2 - End of File	See EOF Specifications	%X2
3 - Unusual Condition		
	1 - Parity error.	%13
	2 - Read time out.	%23
	3 - I/O aborted externally.	%33
	4 - Data lost. No buffer available.	%43
	5 - Data set not ready or disconnect or unit not on line.	%53
	15 - Enable Subsystem Break and no Control Y PIN	%153
	16 - Read Time overflow	%163
	17 - Read return when BREAK flush flag set.	%173
4 - Irrecoverable Error		
	0 - Invalid request	4
	3 - Timing error. Unit was not serviced in time.	%34

DISK I/O SPECIFICATIONS

The calling sequence and the returns for both Moving Head and Fixed Head Disc I/O are the same. Any differences in internal operation is described in more detail in the driver description.

The parameters to ATTACHIO have the general meaning outlined in the ATTACHIO description with the following exceptions. This device supports chained I/O operations.

FUNCTION CODE

0 - READ	An even number of bytes is transferred on all reads. Odd byte counts are rounded up.
1 - WRITE	A multiple of 128 words is always written. The last word of the buffer is used to fill out the sector.
2 - FILE OPEN	No operation.
3 - FILE CLOSE	No operation.
4 - DEVICE CLOSE	No operation.
5 - FILL with Zero's	
6 - FILL with Blanks	

The fill operations write the appropriate data in all words of the prescribed area. A multiple of 128 words is always filled.

COUNT - Word/byte count rounded up as described above.

Parameters P1 and P2 form a double word disc address, expressed as a sector number. Sector numbers begin with zero.

TARGET - Offset in data segment.

STATUS RETURNS:

General (13:3)	Qualifying (8:5)	Overall (8:8)
0 - Pending	1 - Waiting for completion	%10
1 - Successful	0 - Normal Completion	1
4 - Irrecoverable	0 - Invalid function.	4
	1 - Transmission error. CRC or track specific error.	%14
	3 - Transmission Error	%34
	4 - SIO not ready.	%44
	5 - Unit failure. All errors other than track specific	%54
	6 - Invalid disc address	%64

Irrecoverable error are returned only after the operation has been retried 10 times. In addition, a recalibrate and 10 more retries are issued to the Moving Head Disc.

Whenever an irrecoverable track specific error is detected, a track by track transfer is initiated. During this process if an error is detected, the track number is entered into the defective track table kept in sector one of the disc.

FUNCTION CODE -

0 - READ

P1.(13.3) - End of File Specification

1 - WRITE

P2.(13:1) - If set then write past End of Tape Mark. If clear return error if End of Tape has been detected.

2 - OPEN FILE - No operation.

3 - CLOSE FILE - Reset EOF flags.

4 - CLOSE DEVICE - Reset EOF flags and Rewind.

5 - Rewind.

6 - Write Tape Mark.

7 - Forward Space Mark.

8 - Back Space File.

9 - Rewind and Unload.

10 - Gap.

11 - Forward Space Record.

12 - Back Space Record.

This device supports chained System Buffers.

RETURN STATUS -

General (13:3)	Qualifying (8:5)	Overall (8:8)
0 - Pending	1 - Waiting for completion.	%10
	2 - Doing error recovery.	%20
	3 - Not Ready wait.	%30
	4 - No Write Ring wait.	%40
1 - Successful	0 - No errors.	1
	2 - Retry necessary for success.	%21
	3 - EOT after write.	%31
2 - End of File	See EOF Specifications	%X2
3 - Unusual Condition		
	3 - Request aborted externally	%33
	6 - Power fail Abort	%63
	7 - BOT and BSR or BSF request	%73
	10 - Tape runaway.	%103
	11 - EOT and write request	%113
4 - Irrecoverable Error		
	0 - Invalid request	4
	1 - Transmission error	%14
	3 - Timing error	%34
	4 - SIO failure	%44
	5 - Unit failure	%54
	7 - Tape parity error	%74
	12 - System Error	%124

CARD READER I/O SPECIFICATIONS

FUNCTION CODE -

0 - READ

P1.(13:3) - End of File specification.

P2.(11:2) - Data Mode
0 - ASCII
1 - Column Binary

COUNT - COUNT is truncated to produce a maximum of 80 bytes for ASCII reads and 80 words for Binary reads.

2 - FILE CLOSE - No Operation

3 - FILE CLOSE - No Operation.

4 - DEVICE CLOSE - Reset EOF flags.

STATUS RETURNS -

General (13:3)	Qualifying (8:5)	Overall (8:8)
0 - Pending or not started		0
1 - Successful		1
2 - End of File	See EOF Specifications	%X2
3 - Unusual Condition		
	3 - Request aborted externally	%33
	6 - Power fail abort	%63
4 - Irrecoverable Error		
	0 - Invalid Function	4
	4 - Start SIO Failure	%44
	12 - No message buffers	%124

LINE PRINTER I/O SPECIFICATIONS

The Line Printer specifications are designed to produce similar responses as output to a terminal.

1 - WRITE

P1 - Vertical Format Specifications

0 - Print and single space
1 - Use first data byte as Vertical Format Specifications

```
%53 "+" Print and no space
%60 "0" Print and double space
%61 "1" Print and space to top of form
%200-%277 Print and space N-%200 Lines
%300-%307 Print and space with channel N-%277 of the
          Format Tape
%320      Fill print buffer only
```

All others result in print and single space.

P2 - Space Mode Flags

```
(15:1) - 0 - Fill buffer then do space operation
             (Postspace)
         1 - Do space operation then fill buffer
             (Prespace)
(14:1) - 0 - Single and double space with auto page
             eject (60 lines/page)
         1 - No auto page eject (66 line/page)
```

COUNT - COUNT is truncated to produce a maximum of 132 printed bytes.

2 - FILE OPEN - Page eject

3 - FILE CLOSE - Page eject

4 - DEVICE CLOSE - Page eject

STATUS RETURNS:

General (13:3)	Qualifying (8:5)	Overall (8:8)
0 - Pending	- Not started or waiting	0
1 - Successful		1
3 - Unusual	3 - Request aborted externally	%33
	6 - Power failure abort	%63
4 - Irrecoverable		
	0 - Invalid function	4
	1 - Transmission error	%14
	2 - Operation timed out	%24
	4 - SIO failure	%44
	5 - Unit failure	%54
	12 - No message link buffers available	%124

When a Postspace mode follows a Prespace mode (i.e. when the Line Printer buffer contains data) a print and single space is executed before the Postspace request is processed.

Page ejects are suppressed whenever the last operation ended with a page eject.

PAPER TAPE READER SPECIFICATIONS

FUNCTION CODE -

0 - READ

P1.(13:3) - EOF Specification

P2.(0:8) - Special Read Termination character.
.(11:2) - 0 - ASCII otherwise Binary

2 - FILE OPEN - NOP

3 - FILE CLOSE - NOP

4 - DEVICE CLOSE - Reset EOF Flags

STATUS RETURNS -

General (13:3)	Qualifying (8:5)	Overall (8:8)
0 - Pending		0
1 - Successful		1
2 - EOF	X - See EOF Specifications	%X2
3 - Unusual Condition		
	3 - Aborted Externally	%33
	6 - Power fail abort	%63
4 - Irrecoverable		
	0 - Invalid Function	4
	1 - Transmission	%14
	2 - I/O Timeout	%24
	4 - SIO Failure	%44
	5 - Unit Failure	%54

Whenever the out of tape condition is detected or in ASCII mode if 30 null frames are detected an out of tape message is issued and the driver waits for the device to be made ready again. This device supports chained System Buffers.

PAPER TAPE PUNCH SPECIFICATIONS -

FUNCTION CODE -

1 - WRITE

    P1 - Vertical Format Specification if ASCII
        1 - Use first data byte as Vertical Foramt Specification
        %53 "+" Punch and terminate date with XOFF/CR
        %60 "0" Punch and doublespace (XOFF/CR/LF/LF)
        %61 "1" Punch and Form Feed (XOFF/CR/FF)
        %200-%277 Punch and N-%200 Spaces (XOFF/CR/N-%200 LF's)
        %320-   Punch (no CR/LF)

        All others - Punch and terminate with (XOFF/CR/LF)

    P2 - Space Mode Flags
        .(15:1) - 0 - Postspace. Punch data the punch spacing characters
                - 1 - Prespace. Punch spacing characters then punch data

        .(11:2) - 0 - ASCII
                - 1 - Binary

2 - FILE OPEN - Punch Leader

3 - FILE CLOSE - Punch trailer

4 - DEVICE CLOSE - No Operation

STATUS RETURNS -

General (13:3)	Qualifying (8:5)	Overall (8:8)
0 - Pending		0
1 - Successful	0 - No Errors	1
	3 - Tape low after punch	%31
2 - End of File	X - Set EOF Specifications	%X2
3 - Unusual Conditions		
	3 - Aborted Externally	%33
	6 - Power Fail Abort	%63
4 - Irrecoverable	0 - Invalid Function	4
	1 - Transmission Error	%14
	2 - I/O Timeout	%24
	4 - SIO Failure	%44
	5 - Unit Failure	%54

In Binary mode the data is punched as presented, no spacing characters or record delimiters are added. This device supports chained System Buffers.

PLOTTER INTERFACE SPECIFICATIONS

FUNCTION CODE

1 - WRITE

2 - FILE OPEN - Master Clear

3 - FILE CLOSE - No Operation

4 - DEVICE OPEN - No Operation

STATUS RETURNS:

General (13:3)	Qualifying (8:5)	Overall (8:8)
0 - Pending	1 - Waiting for completion	%10
	3 - Not Ready wait	%30
1 - Successful		1
3 - Unusual Condition		
	3 - I/O aborted externally	%33
	6 - Power Failure abort	%63
	14 - Limit swich reached	%143
4 - Irrecoverable	0 - Invalid function	4
	1 - Transmission error	%14
	3 - Timing error	%34
	4 - SIO failure	%44
	5 - Unit failure	%54

CARD READER/PUNCH SPECIFICATIONS

FUNCTION CODE

```
0 - READ

     P1.(13:3) - EOF specification

     P2.(11:2) - If 0 then ASCII else column binary.

1 - WRITE

     P2.(11:2) - If 0 then ASCII else column binary

2 - FILE OPEN

     P1 - 1 - Read access
        - 2 - Write access, Pick a card
        - 3 - Read/Write access

3 - FILE CLOSE - If read access, then stack last card

4 - DEVICE CLOSE - Reset EOF Flags

5 - CONTROL - Allowed only if FOPEN, P1 = 3

     P2.(6:1 ) - 0 - Sel no inhibit feed on WR
               - 1 - Sel Inhibit Feed On WR
       .( 7:1) - 0 - Sel Punch On Writes
               - 1 - Sel No Punch On Writes

       .( 8:1) - 0 - Sel Print On Writes
               - 1 - Sel No Print On Writes

       .( 9:1) - 0 - Sel Print & Punch Same Dat
               - 1 - Sel Print & Punch Sep Dat

       .(10:1) - 0 - Sel Primary Stacker
               - 1 - Sel Secondary Stacker

       .(11:1) - 0 - Sel Primary Hopper
               - 1 - Sel Secondary Hopper
```

STATUS RETURNS:

General (13:3)	Qualifying (8:5)	Overall (8:8)
0 - Pending	1 - Waiting for completion	%10
	3 - Not Ready Wait	%30
1 - Successful		1
2 - End of File	See EOF Specifications	%X2
3 - Unusual Condition		
	3 - I/O aborted externally	%33
	6 - Power fail abort	%63
4 - Irrecoverable	0 - Invalid Function	%14
	1 - Transmission Error	%14
	2 - I/O Timeout	%24
	5 - Unit Failure	%54

PAGE MODE TERMINAL SPECIFICATIONS

In General, the Page Mode Terminal Driver supports the same functions as the Asynchronous Terminal Driver. Listed below are the exceptions.

FUNCTION CODE -

0 - READ

P2.(10:1) - If 0 then System Mode else User Mode In System Mode, the cursor is positioned to the column where the last write terminated before a read is initiated. In Use Mode, the User is responsible for all cursor positioning and for reading the DC2 and issuing a second read to obtain the data.

.(9:1) - CR/LF transmission control. If 0 then CR, LF characters are not transmitted to the users buffer.

.(8:1) - Function Key Control. If 0 then F1 and Function keys are used for BREAK and Sub System Break respectively. If not 0 then the generated escape sequences are transmitted.

6/7 Change Input/Output speed. For this device, if either the input or output speed is change, the other speed is also changed.

8/9 Echo control. Not used. Invalid request.

14/15 Tape mode control. Not used. Invalid request.

23 Set Terminal Type. Only types 10 and 11 allowed.

26/27 Control X echo Control. Not used. Invalid request.

29 Ptape read. Not used. Invalid request.

STATUS RETURNS:

General (13:3)	Qualifying (8:5)	Overall (8:8)
0 - Pending	0 - Not started	0
	1 - Waiting completion	%10
1 - Successful	0 - Normal completion	1
	1 - Completed on special read stop character	%11
2 - End of File	X - See EOF Specifications	%X2
3 - Unusual Condition		
	1 - Parity Error on read	%13
	2 - Read Timeout	%23
	3 - I/O aborted externally	%33
	5 - Terminal not on line or not Ready or Data Set not Ready	%53
	6 - Power fail abort	%63
	%15 - Enable Sybsystem Bread and no Control Y PIN	%153
	%16 - Read Time overflow	%163
	%17 - Read stopped when Break detected.	%173
4 - Irrecoverable Error		
	0 - Invalid request, function or parameter	4
	1 - Transfer Error	%14
	2 - SIO program failed to complete and was time out	%24
	4 - SIO failure	%44
	%12 - System Error	%124

DECLARATION:

```
DOUBLE PROCEDURE ATTACHIO(LDEV,QMISC,DSTX,ADDR,FUNC,CNT,P1,P2,FLAGS);
  VALUE  LDEV,QMISC,DSTX,ADDR,FUNC,CNT,P1,P2,FLAGS;
  INTEGER LDEV,QMISC,DSTX,ADDR,FUNC,CNT,P1,P2,FLAGS;
  OPTION UNCALLABLE, PRIVILEGED;
```

FUNCTION:

THIS PROCEDURE CONSTRUCTS AN IOQ ELEMENT AND LINKS IT TO THE APPROPRIATE DEVICE QUEUE. IF THIS IS THE FIRST ELEMENT IN THE QUEUE OR THE REQUEST SPECIFIES PREMPTION, THE MONITOR IS CALLED TO INITIATE THE OPERATION. FOR BLOCKED REQUESTS, THE MONITOR MAY BE RECALLED BY ATTACHIO AFTER A "WAIT" IF THE REQUEST IS NOT COMPLETED WHEN THE CALLER IS AWOKEN.
IF NO IOQ ELEMENTS ARE AVAILABLE, IMPEDABLE REQUESTS ARE SUSPENDED UNT UNTIL AN IOQ ELEMENT BECOMES AVAILABLE.
REQUESTS WHICH SPECIFY NOT IMPEDABLE ARE NOT "WAITED" FOR ANY REASON.

INPUT:

LDEV - LOGICAL DEVICE NUMBER.

QMISC - MISCELLANEOUS PARAMETER SPECIFED FOR THE DEVICE. IF NOT SPECIFIED MUST BE ZERO.

DSTX - DST NUMBER OF DATA SEGMENT. IF ZERO THEN SPECIFIES THAT ADDR IS DB RELATIVE TO THE CALLERS STACK. MUST BE ZERO IF SYSTEM BUFFERS IS SPECIFICED.

ADDR - DEPENDING ON FLAGS.(14:1) AND DSTX THIS MAY BE:
- 1.) OFFSET TO DATA IN DATA SEGMENT.
- 2.) OFFSET TO DATA FROM DB IN CALLERS STACK.
- 3.) INDEX TO A SYSTEM BUFFER.

FUNC - FUNCTION CODE. DEVICE DEFINED BUT USUALLY:
- 0 - READ
- 1 - WRITE
- 2 - OPEN FILE
- 3 - CLOSE FILE
- 4 - CLOSE DEVICE

CNT - DATA TRANSFER COUNT:
WORDS IF POSITIVE, BYTES IF NEGATIVE.

P1 - PARAMETER 1. DEVICE DEPENDANT.

P2 - PARAMETER 2. DEVICE DEPENDANT.

```
FLAGS - CONTROL AND SPECIFICATION FLAGS.
  .( 0:7) - 0
  .( 7:2) - PREMPTION FLAGS
             1 - SOFT PREMPTION
             2 - HARD PREMPTION
  .( 9:1) - 0
  .(10:1) - SPECIAL REQUEST. DEVICE DEFINED. IF SET THEN SPECIAL
            HANDLING IS TO BE APPLIED TO THIS REQUEST.
  .(11:1) - IF SET THEN THIS IS A DIAGNOSTIC REQUEST.
  .(12:1) - SYSTEM BUFFER FLAG. IF SET THE ADDR IS AN INDEX RELATIVE
            TO THE SBUF TABLE. FOR DEVICES WHICH SUPPORT CHAINING
            THE DATA IS TRANSFERED TO AND FROM A SET OF CHAINED
            BUFFERS, UP TO A MAXIMUM OF 1024 WORDS.
            IF CLEAR THEN ADDR IS A DATA SEGMENT RELATIVE
            ADDRESS.
  .(13:3) - REQUEST TYPE:
             0 - UNBLOCKED, NO WAKE ON COMPLETION. IMPEDE IF NO IOQ
                 ELEMENT IS AVAILABLE.
             1 - BLOCKED. CALLER IS TO BE WAITED UNTIL REQUEST
                 IS COMPLETED.
             2 - UNBLOCKED, WAKE CALLER WHEN REQUEST IS COMPLETED.
                 IMPEDE IF NO IOQ AVAILABLE.
             3 - UNBLOCKED AND NO PROCESS IS TO BE ASSOCIATED
                 WITH THIS REQUEST. IMPEDE IF NO IOQ AVAILABLE.
             4 - UNBLOCKED, NO WAKE ON COMPLETION BUT DO NOT
                 IMPEDE IF NO IOQ AVAILABLE.
             5 - RESERVED.
             6 - UNBLOCKED, WAKE ON COMPLETION BUT DO NOT IMPE
                 IF NO IOQ IS AVAILABLE.
             7 - SAME AS 3 BUT DO NOT IMPEDE IF NO IOQ IS AVAILABLE.


RETURN:

 BLOCKED -

          0         7 8         12 13      15
          *************************************
  S-1     *    PCB    * QUALIFYING * GENERAL *
          *   NUMBER  *   STATUS   *  STATUS *
          *************************************
  S-0     * TRANSMISSION LOG / CONTROL RETURNS*
          *************************************


NOTE - TRANSMISSION LOG IS RETURNED WITH THE SAME SENSE AS THE COUNT
       PARAMETER IN ATTACHIO (+=WORDS, -=BYTES).
```

UNBLOCKED -

```
          0            7 8          12 13      15
          ************************************
  S-1     *       IOQ INDEX OF REQUEST         *
          ************************************
  S-0     *             0                      *
          ************************************
```

NOTE - THE IOQ INDEX RETURN ABOVE IS USED AS THE PARAMETER TO IOSTATUS TO DETERMINE THE COMPLETION STATUS OF THE REQUEST. IF THE REQUEST TYPE IN FLAGS SPECIFIED THAT THIS REQUEST IS NOT IMPEDABLE THEN THE IOQ INDEX RETURN WILL BE ZERO IF NO IOQ ELEMENTS ARE AVAILABLE.

FOR TYPE 3 REQUESTS, IF ADR IS NOT ZERO THEN IT IS ASSUMED TO BE A SYSTEM BUFFER INDEX. AT THE COMPLETION OF A REQUEST, THE SYSTEM BUFFER(S) POINTED TO BY ADR ARE RETURNED TO THE FREE LIST BY THE I/O SYSTEM.

DECLARATION:

```
PROCEDURE ABORTPROCIO(PCBN);
  VALUE PCBN;  INTEGER PCBN;
  OPTION PRIVILEGED, UNCALLABLE;
```

FUNCTION:

THIS PROCEDURE ABORTS ALL I/O REQUESTS ASSOCIATED WITH A PROCESS.
IF PCBN IS ZERO THEN THE PROCESS IS THE CALLER OTHERWISE PCBN
SPECIFIES THE PROCESS WHOSE I/O IS TO BE ABORTED.
THE I/O IS ABORTED BY SETTING AN ABORT FLAG IN THE IOQ ELEMENT.
THE MONITOR/DRIVER ASSOCIATED WITH THE LOGICAL DEVICE SPECIFIED
IN THE REQUEST IS THEN CALLED TO ABORT THE I/O.
THIS PROCEDURE RETURNS WHEN NO FURTHER REQUESTS ARE FOUND
ASSOCIATED WITH THE SPECIFIED PROCESS.

INPUT:

PCBN - IF ZERO THEN ABORT I/O ASSOCIATED WITH THE CALLING PROCESS.

IF NOT ZERO, THEN SPECIFES THE PCB NUMBER WHOSE I/O IS TO
BE ABORTED.

TABLES ACCESSED:

DIT IOQ LPDT

DECLARATION:

[redacted]
VALUE LDEVN; INTEGER LDEVN;
OPTION EXTERNAL, UNCALLABLE;

FUNCTION:

THIS PROCEDURE ABORTS ALL I/O REQUEST QUEUED FOR THE DEVICE SPECIFIED BY LDEVN. THE REQUESTS ARE ABORTED BY SETTING AN ABORT FLAG IN THE REQUEST AND CALLING THE MONITOR/DRIVER.

INPUT:

LDEVN - LOGICAL DEVICE NUMBER ON WHICH THE I/O IS TO BE ABORTED.

RETURN:

CCE - REQUEST ABORTED
CCG - NO REQUEST PENDING
CCL - INVALID LOGICAL DEVICE NUMBER

DECLARATION:

```
PROCEDURE AWAKEIO(DITP,FLAGS);
  VALUE DITP, FLAGS;  INTEGER POINTER DITP;  INTEGER FLAGS;
  OPTION PRIVILEGED, UNCALLABLE;
```

FUNCTION:

THIS PROCEDURE CAUSES THE MONITOR/DRIVER ASSOCIATED WITH THE DEVICE SPECIFIED BY DITP TO BE EXECUTED. THE MONITOR/DRIVER PROCEDURES MAY BE EXECUTED ON THE CALLING STACK OR IN AN I/O PROCESS DEPENDING ON THE TYPE OF DRIVER AND CONDITIONS SPECIFIED IN FLAGS.

INPUT:

DITP - A SYSDB RELATIVE POINTER TO THE DIT OF THE DEVICE FOR WHICH SERVICE IS REQUESTED.

FLAGS.(0:10) - DST NUMBER OF THE CALLERS STACK OR ZERO. IF ZERO IS SPECIFIED CERTAIN DATA MOVES TO THE TERMINALBUFFERS WILL NOT BE EXECUTED ON THE CALLERS STACK.

.(13:1) - IMPEDABLE FLAG. IF SET THEN EXECUTION OF THIS ROUTINE CAN BE IMPEDED BECAUSE OF CODE ABSENCE OR FOR OTHER REASONS. IF CLEAR THEN THIS ROUTINE WILL NOT BE IMPEDED OR WAIT FOR ANY REASON. IF SET THEN ONE LEVEL OF PSEUDODISABLE MUST BE IN EFFECT WHEN AWAKEIO IS CALLED.

.(14:2) - MUST BE ZERO

DB MUST BE SET TO SYSDB.

TABLES ACCESSED:

DIT CST DLT

DECLARATION:

```
PROCEDURE BCONVERT(BN);
  VALUE BN;  INTEGER BN;
  OPTION PRIVILEGED, UNCALLABLE;
```

FUNCTION:

THIS PROCEDURE CONVERTS THE NUMBER BN TO OCTAL ASCII AND OUTPUTS IT DIRECTLY TO THE SYSTEM CONSOLE, NOT USING THE TERMINAL I/O SYSTEM.

INPUT:

BN - NUMBER TO BE OUTPUT IN OCTAL FORMAT.

DB - SET TO SYSDB.

INTERRUPTS MUST BE DISABLED BEFORE THE CALL.

NOTE:

INTERRUPTS ARE NOT ENABLED BY THIS PROCEDURE.

DECLARATION:

```
PROCEDURE CHECKLDEV(LDEV);
  VALUE LDEV;  INTEGER LDEV;
  OPTION PRIVILEGED, UNCALLABLE;
```

FUNCTION:

THIS PROCEDURE CHECKS THAT LDEV IS A VALID LOGICAL DEVICE AND WHETHER IT IS A TERMINAL, DISC OR OTHER SIO DEVICE.

INPUT:

LDEV - LOGICAL DEVICE NUMBER

DB SET TO SYSDB.

RETURN:

CONDITON CODE -

CCE - IF CARRY THEN A DISC OTHERWISE OTHER SIO DEVICE

CCL - INVALID LOGICAL DEVICE NUMBER.

CCG - DEVICE IS A TERMINAL

DECLARATION:

```
  PROCEDURE CLEARWAKE(IOQX);
    VALUE IOQX;  INTEGER IOQX;
    OPTION PRIVILEGED, UNCALLABLE;
```

FUNCTION:

THIS PROCEDURE CLEARS THE WAKE FLAG FOR UNBLOCKED REQUESTS.

INPUT:

IOQX - THE INDEX,RELATIVE TO THE BASE OF THE IOQ TABLE,OF THE IOQ ELEMENT ASSOCIATED WITH THE REQUEST. THIS INDEX IS RETURNED BY ATTACHIO WHEN A REQUEST IS UNBLOCKED.

RETURN:

CONDITON CODE -

CCE - WAKE FLAG CLEARED

CCL - REQUEST HAS ALREADY COMPLETED.

NOTE:

INTERRUPT ENABLED/DISABLED STATUS IS MAINTAINED OVER THE CALL.

DECLARATION:

```
PROCEDURE DCONVERT(DN);
  VALUE DN;  INTEGER DN;
  OPTION PRIVILEGED, UNCALLABLE;
```

FUNCTION:

THIS PROCEDURE CONVERTS THE NUMBER DN TO DECIMAL ASCII AND OUTPUTS IT DIRECTLY TO THE SYSTEM CONSOLE, NOT USING THE TERMINAL I/O SYSTEM.

INPUT:

DN - NUMBER TO BE OUTPUT IN DECIMAL FORMAT.

DB - SET TO SYSDB.

INTERRUPTS MUST BE DISABLED BEFORE THE CALL.

NOTE:

INTERRUPTS ARE NOT ENABLED BY THIS PROCEDURE.

DECLARATION:

```
LOGICAL PROCEDURE DEVICESTATUS(LDEV);
  VALUE LDEV;  INTEGER LDEV;
  OPTION PRIVILEGED, UNCALLABLE;
```

FUNCTION:

THIS PROCEDURE RETURNS THE 16 BIT HARDWARE STATUS OF THE PHYSICAL DEVICE SPECIFIED BY THE LOGICAL DEVICE, LDEV. THE STATUS IS CURRENT TO THE LAST TIME THE DEVICE INTERRUPTED. NO STATUS IS AVAILABLE FOR ASYNCHRONOUS TERMINALS.

INPUT:

LDEV - LOGICAL DEVICE NUMBER

TABLES ACCESSED:

DIT LPDT

RETURN:

HARDWARE STATUS FOR THE DEVICE SPECIFIED BY LDEV

CONDITION CODE -

CCE - SUCCESSFUL

CCL - DEVICE WAS A TERMINAL OR INVALID LOGICAL DEVICE NUMBER

DECLARATION:

```
PROCEDURE DSETCONTROL(FUNCTION,DITP);
  VALUE FUNCTION;  INTEGER FUNCTION;
  ARRAY DITP;
  OPTION PRIVILEGED, UNCALLABLE;
```

FUNCTION:

THIS PROCEDURE OUTPUTS CONTROL WORDS TO THE DATA SET CONTROL BOARDS TO INITIALIZE, HANGUP, SET THE DATA SET TO READING AND SET THE DATA SET TO WRITING. IT CHECKS TO SEE IF CONTROL SHOULD BE OUTPUT TO NONE, ONE OR TWO DATA SET CONTROL BOARDS.

INPUT:

FUNCTION - CODE TO SPECIFY THE OPERATION TO BE PERFORMED.

0 - INITIALIZE THE DATA SET. SETS DATA TERMINAL READY AND ENABLES INTERRUPT ON CC FOR 202/2002 OR CC AND CF FOR A 103.

1 - TURN TO WRITE. SETS CA AND CLEARS SA. INTERRUPT ON SB = 1 AND CB = 1.

2 - TURN TO READ. CLEARS CA AND SETS SA. INTERRUPT ON CB = 0 AND SB = 1 FOR 202 OR SB = 0 FOR 2002.

3 - HANGUP. CLEAR DATA TERMINAL READY. DISABLE INTERRUPTS ON ALL STATUS.

DITP - SYSDB RELATIVE POINTER TO DIT FOR DEVICE.

DB - SET TO SYSDB ON CALL.

TABLES ACCESSED:

DIT

DECLARATION:

```
INTEGER PROCEDURE GETSYSBUF(NUMB,IFLAG);
  VALUE NUMB,IFLAG;  INTEGER NUMB;   LOGICAL IFLAG;
  OPTION PRIVILEGED, UNCALLABLE;
```

FUNCTION:

THIS PROCEDURE GETS THE NUMBER OF ENTRIES NUMB FROM THE SYSTEM BUFFER TABLE AND RETURNS THE INDEX OF THE HEAD ENTRY. THE FIRST WORD OF EACH ENTRY IS A LINK TO THE NEXT. IF THE NUMBER OF ENTRIES IS UNAVAILABLE, THE PROCESS WILL BE IMPEDED IF IFLAG IS SET TRUE. NOTE THAT A SECONDARY ENTRY POINT FGETSYSBUF IS PROVIDED. THIS CALL IS USED TO PREVENT HOGGING OF ALL THE SYSTEM BUFFERS BY THE FILESYSTEM.

INPUT:

NUMBER - NUMBER OF BUFFERS TO BE ALLOCATED

IFLAG - SET TRUE IF PROCESS MAY BE IMPEDED UNTIL BUFFERS ARE AVAILABLE.

TABLES ACCESSED:

SBUF

RETURN:

INDEX OF THE HEAD ENTRY OF A LINKED LIST OF THE ENTRIES REQUESTED. THE LINKS ARE INDEXES RELATIVE TO THE TABLE BASE AND ARE CONTAINED IN THE FIRST WORD OF EACH ENTRY. THE INDEX RETURNED AND THE LINKS POINT TO THE SECOND WORD OR THE BEGINNING OF THE DATA AREA IN EACH ENTRY.

CONDITION CODE -
- CCE - SUCCESSFUL
- CCG - UNABLE TO FILL REQUEST AND CANNOT IMPEDE
- CCL - INVALID NUMBER OF BUFFERS REQUESTED

DECLARATION:

```
LOGICAL PROCEDURE IOCONTROL(LDEV,FUNCTION);
  VALUE LDEV, FUNCTION;  INTEGER LDEV, FUNCTION;
  OPTION PRIVILEGED, UNCALLABLE;
```

FUNCTION:

THIS PROCEDURE GENERATES AN I/O REQUEST FOR THE DEVICE LDEV WITH A FUNCTION CODE SPECIFIED BY FUNCTION. THE REQUEST IS BLOCKED WITH ALL OTHER PARAMETERS IN THE ATTACHIO CALL ZERO.

INPUT:

LDEV - LOGICAL DEVICE NUMBER

FUNCTION - FUNCTION CODE DEFINED BY THE DEVICE SPECIFIED BY LDEV.

RETURN:

TRUE - REQUEST SUCCESSFULLY COMPLETED.

FALSE - ERROR OCCURED IN PERFORMING REQUEST.

DECLARATION:

```
DOUBLE PROCEDURE IOSTATUS(IOQX);
  VALUE IOQX;  INTGER IOQX;
  OPTION PRIVILEGED, UNCALLABLE;
```

FUNCTION:

THIS PROCEDURE RETURNS THE STATUS AND THE TRANSFER COUNT OF THE REQUEST IDENTIFIED BY IOQX. IF THE REQUEST HAS COMPLETED THE IOQ ELEMENT IDENTIFIED BY IOQX IS RETURNED TO THE FREE POOL.

INPUT:

IOQX - THE INDEX,RELATIVE TO THE BASE OF THE IOQ TABLE,OF THE IOQ ELEMENT ASSOCIATED WITH THE REQUEST. THIS INDEX IS RETURNED BY ATTACHIO WHEN A REQUEST IS UNBLOCKED.

TABLES ACCESSED:

IOQ

RETURN:

```
         0           7 8         12 13       15
         *************************************
   S-1   *    PCB     * QUALIFYING * GENERAL *
         *   NUMBER   *   STATUS   *  STATUS *
         *************************************
   S-0   * TRANSMISSION LOG / CONTROL RETURNS*
         *************************************
```

NOTE - TRANSMISSION LOG IS RETURNED WITH THE SAME SENSE AS THE COUNT PARAMETER IN ATTACHIO (+=WORDS, -=BYTES) AND BUFFER OFFSET IS THE SAME AS THE TARGET PARAMETER.

CONDITION CODE -

CCE - REQUEST COMPLETED. IOQ ELEMENT RETURNED.

CCG - REQUEST NOT COMPLETED

CCL - PCB NUMBER IS ZERO, INDICATING THE REQUEST IS NO LONGER ASSOCIATED WITH A PROCESS.

DECLARATION:

```
  DOUBLE PROCEDURE IOSTATUSX(IOQX);
    VALUE IOQX;  INTGER IOQX;
    OPTION PRIVILEGED, UNCALLABLE;
```

FUNCTION:

THIS PROCEDURE RETURNS THE STATUS, TRANSFER COUNT AND THE QMISC WORD OF THE REQUEST IDENTIFIED BY IOQX. IF THE REQUEST HAS COMPLETED THE IOQ ELEMENT IDENTIFIED BY IOQX IS RETURNED TO THE FREE POOL.

INPUT:

IOQX - THE INDEX,RELATIVE TO THE BASE OF THE IOQ TABLE,OF THE IOQ ELEMENT ASSOCIATED WITH THE REQUEST. THIS INDEX IS RETURNED BY ATTACHIO WHEN A REQUEST IS UNBLOCKED.

TABLES ACCESSED:

IOQ

RETURN:

```
           0          7 8         12 13      15
           *************************************
   S-1     *    PCB    * QUALIFYING * GENERAL *
           *   NUMBER  *   STATUS   *  STATUS *
           *************************************
   S-0     * TRANSMISSION LOG / CONTROL RETURNS*
           *************************************
```

NOTE - THIS PROCEDURE RETURNS THE SAME DATA AS IOSTATUS WITH THE EXCEPTION THAT THE QMISC WORD OF THE IOQ ELEMENT FOR THE REQUEST IS RETURNED IN THE X REGISTER.

CONDITION CODE -

CCE - REQUEST COMPLETED. IOQ ELEMENT RETURNED.

CCG - REQUEST NOT COMPLETED

CCL - PCB NUMBER IS ZERO, INDICATING THE REQUEST IS NO LONGER ASSOCIATED WITH A PROCESS.

DECLARATION:

```
PROCEDURE MEASIO(LDEV,SBUFI,CNT,FUNC);
VALUE LDEV,SBUFI,CNT,FUNC;
INTEGER LDEV,SBUFI,CNT,FUNC;
OPTION PRIVILEGED,UNCALLABLE;
```

FUNCTION:

THIS PROCEDURE PROVIDES MAGNETIC TAPE OUTPUT OF INFORMATION BY MEANS OTHER THEN THE NORMAL I/O SYSTEM. IT WILL TRANSFER DATA FROM SYSTEM BUFFERS TO A MAGNETIC TAPE DRIVE SPECIFIED BY LDEV.

INPUT:

LDEV - THE LOGICAL DEVICE NUMBER OF THE TAPE DRIVE.

SBUFI - THE TABLE INDEX OF THE FIRST SYSTEM BUFFER.

CNT - THE NUMBER OF WORDS TRANSFERRED (1 <= CNT <= 1024)

FUNC - THE REQUEST FUNCTION
- 0 - WRITE
- 1 - REWIND
- 2 - WRITE EOF AND UNLOAD

OUTPUT:

CONDITION CODE

- CCE - SUCCESSFUL COMPLETION
- CCG - RETRY NECESSARY FOR SUCCESS
- CCL - REQUEST UNSUCCESSFUL

NOTE:

DB MUST BE SET AT SYSDB AND THE INTERRUPT SYSTEM MUST BE DISABLED. SINCE I/O IS DONE WITH INTERRUPTS DISABLED, NO OTHER TAPE UNIT ON THE SELECTED CONTROLLER MAY BE USED BY THE SYSTEM.

DECLARATION:

```
MPXCONTROL(OPERATION,DITP);
  VALUE OPERATION, DITP;  INTEGER OPERATION;  POINTER DITP;
  OPTION PRIVILEGED, UNCALLABLE;
```

FUNCTION:

THIS PROCEDURE OUTPUTS CONTROL INFORMATION TO THE ASYNCHRONOUS TERMINAL MULTIPLEXOR TO INITIALIZE THE INPUT CHANNEL AND TO TURN ECHOING ON AND OFF. IT ALSO INITIALIZES THE SPEED AND PARITY GENERATION CONTROL ON THE OUTPUT CHANNEL. WHEN EVER THIS PROCEDURE IS CALLED, THE SPEED FOR THE REFERENCED CHANNEL IS UPDATED TO THE CURRENT SPEED SPECIFIED IN THE DIT.

INPUT:

```
OPERATION - 0 - TURN ECHO OFF AND SET OR CLEAR DIAGNOSTIC MODE IF
                SPEED SENSING ON OR OFF.

            1 - TURN ECHO ON IF ECHO ENABLED AND SET OR CLEAR
                DIAGNOSTIC MODE IF SPEED SENSING ON OR OFF.

           -1 - INITIALIZE OUTPUT CHANNEL AND SET WRITE PARITY
                CONTROL ACCORDING TO THE P1YCONTROL BIT.

           -2 - DISABLE INTERRUPTS ON READ CHANNEL AND TURN ECHO OFF

DITP      - SYSDB RELATIVE POINTER TO THE DIT.

DB SET TO SYDB.
```

TABLES ACCESSED:

DIT

DECLARATION:

```
PROCEDURE PTAPE(TFILE,DFILE);
  VALUE TFILE,DFILE;  INTEGER TFILE, DFILE;
  OPTION PRIVILEGED;
```

FUNCTION:

THIS PROCEDURE IS USED TO READ PAPER TAPES WHICH DO NOT CONTAIN X-OFF CHARACTERS, OR WHEN THE PAPER TAPE READER DOES NOT RECOGNIZE THE X-OFF CHARACTER. THE TAPE IS READ FROM THE DEVICE ASSOCIATED WITH THE FILE TFILE AND SPOOLED INTO THE DISC FILE SPECIFIED BY DFILE.
THE MAXIMUM NUMBER OF BYTES WHICH CAN BE SPOOLED IN ONE CALL TO PTAPE IS 32,767.

INPUT:

TFILE - FILE NUMBER OF THE TERMINAL FROM WHICH THE DATA IS TO BE INPUT.
DFILE - FILE NUMBER OF THE DISC FILE TO WHICH THE DATA IS TO BE SPOOLED.

RETURN:

CONDITION CODE -

CCE - SUCCESSFUL

CCG - INSUFFICIENT SYSTEM RESOURCES, OR THE TAPE BEING READ CONTAINS MORE DATA THAN CAN BE READ IN ONE PTAPE CALL OR AN ERROR OCCURRED WHILE WRITING TO DFILE OR READING FROM TFILE.

CCL - THE DEVICE SPECIFIED BY TFILE IS NOT A TERMINAL.

DECLARATION:

```
PROCEDURE RESETBREAKBITS(LDEV,BREAKFLAG);
  VALUE LDEV, BREAKFLAG;  INTEGER LDEV;  LOGICAL BREAKFLAG;
  OPTION PRIVILEGED, UNCALLABLE;
```

FUNCTION:

THIS PROCEDURE RESETS THE BREAKFLAG OR THE CONTROL Y (SUBSYSTEM BREAK) FLAG IN THE LOGICAL PHYSICAL DEVICE TABLE FOR THE DEVICE SPECIFIED BY LDEV. IF THE DEVICE SPECIFIED IS NOT A TERMINAL, THEN NO ACTION IS TAKEN.

INPUT:

```
LDEV       - LOGICAL DEVICE NUMBER

BREAKFLAG  - IF TRUE THEN RESET BREAK FLAG
             IF FALSE THE RESET CONTROL Y FLAG
```

TABLES ACCESSED:

LPDT DIT

DECLARATION:

```
PROCEDURE RETURNSYSBUF(HEAD);
  VALUE HEAD;  INTEGER HEAD;
  OPTION PRIVILEGED, UNCALLABLE;
```

FUNCTION:

THIS PROCEDURE RETURNS THE LIST OF SYSTEM BUFFERS POINTED TO BY HEAD. THE LIST IS LINKED WITH INDEXES RELATIVE TO THE BASE OF THE SBUF TABLE IN THE FIRST WORD OF EACH ELEMENT. THE LIST IS TERMINATED WITH A ZERO FOR THE LAST LINK.

INPUT:

HEAD - INDEX OF THE FIRST ELEMENT IN THE LIST RELATIVE TO THE BASE OF THE SBUF TABLE.

TABLES ACCESSED:

SYSBUF

DECLARATION:

```
PROCEDURE SETWAKE(IOQX);
  VALUE IOQX;  INTEGER IOQX;
  OPTION PRIVILEGED, UNCALLABLE;
```

FUNCTION:

THIS PROCEDURE SETS THE WAKE FLAG FOR UNBLOCKED REQUESTS, CAUSING THE PROCESS ASSOCIATED WITH THE REQUEST TO BE AWAKENED,IF IT WAS WAITING ON I/O, WHEN THE REQUEST IS COMPLETED.

INPUT:

IOQX - THE INDEX,RELATIVE TO THE BASE OF THE IOQ TABLE,OF THE IOQ ELEMENT ASSOCIATED WITH THE REQUEST. THIS INDEX IS RETURNED BY ATTACHIO WHEN A REQUEST IS UNBLOCKED.

RETURN:

CONDITION CODE -

CCE - WAKE FLAG SET

CCL - REQUEST HAS ALREADY COMPLETED

NOTE:

NORMALLY THIS PROCEDURE IS CALLED WITH INTERRUPTS DISABLED TO PREVENT REQUEST COMPLETION BETWEEN THE SETTING OF THE WAKE FLAG AND RETURN TO THE CALLER. INTERRUPT STATUS IS MAINTAINED OVER THE CALL.

DECLARATION:

```
PROCEDURE SUDDENDEATH(N);
  VALUE N;  INTEGER N;
  OPTION PRIVILEGED, UNCALLABLE;
```

FUNCTION:

THIS PROCEDURE OUTPUTS A SYSTEM HALT MESSAGE WITH THE NUMBER N TO THE SYSTEM CONSOLE. THE MESSAGE IS OUTPUT DIRECTLY AND NOT THROUGH THE I/O SYSTEM. AFTER THE MESSAGE IS OUTPUT THE MACHINE IS HALTED WITH A HALT 17.

ALSO THERE MAY BE A HALT 16 IF THE CONSOLE IS NOT CONFIGURED AND SET ON LINE WHEN SUDDENDEATH IS CALLED OR HALT 15 IF AN I/O INSTRUCTION FAILURE OCCURS DURING THE OUTPUT OF THE SYSTEM FAILURE MESSAGE.

THE MESSAGE OUTPUT HAS THE FOLLOWING FORMAT:

```
**** SYSTEM FAILURE # XX; STATUS YYYYYY; DELTA P ZZZZZZ
```

INPUT:

N - ERROR NUMBER TO BE PRINTED IN DECIMAL.

DECLARATION:

```
PROCEDURE TERMINIT(DITP);
  INTEGER ARRAY DITP;
  OPTION PRIVILEGED, UNCALLABLE;
```

FUNCTION:

THIS PROCEDURE INITIALIZES THE ASYNCHRONOUS CONTROLLER CHANNEL IDENTIFIED BY DITP. THE DIAGNOSTIC OR SPEED SENSING CHANNELS OF THE CONTROLLER ARE ALSO SET UP ON EACH CALL. THE CHANNEL MAY BE SET TO THE SPEED SENSING MODE OR INITIALIZED TO THE SPEED SET IN THE DIT. IF THE DEVICE WAS IN THE PROCESS OF WRITING A CHARACTER A SYNC IS SENT TO RESTART THE WRITE. IF THE DEVICE IS ON A DATA SET, THE DATA SET IS SET TO LISTEN FOR DATA SET READY.

INPUT:

DITP - SYSDB RELATIVE POINTER TO THE DIT FOR THE DEVICE.

DB - SET TO SYSDB ON THE CALL.

TABLES ACCESSED:

DIT LPDT ILT

DECLARATION:

```
  PROCEDURE WAITFORIO(IOQX);
    VALUE IOQX;   INTEGER IOQX;
    OPTION PRIVILEGED, UNCALLABLE;
```

FUNCTION:

THIS PROCEDURE CHECKS IF THE I/O REQUEST ASSOCIATED WITH IOQX HAS BEEN COMPLETED. IF IT HAS NOT, THE PCB INDEX OF THE CALLER IS SET INTO THE THE IOQ ELEMENT AND THE CALLER IS WAITED UNTIL THE I/O IS COMPLETED. IF THE I/O IS COMPLETED OR WHEN IT DOES COMPLETE AFTER THE CALLER IS WAITED, THE COMPLETION STATUS AND TRANSFER COUNT/CONTROL RETURN ARE RETURNED TO THE CALLER AND THE IOQ ELEMENT IS RETURNED TO THE FREE POOL.

INPUT:

IOQX - THE INDEX,RELATIVE TO THE BASE OF THE IOQ TABLE,OF THE IOQ ELEMENT ASSOCIATED WITH THE REQUEST. THIS INDEX IS RETURNED BY ATTACHIO WHEN A REQUEST IS UNBLOCKED.

TABLES ACCESSED:

IOQ

RETURN:

```
          0           7 8          12 13       15
          *************************************
  S-1     *    PCB     * QUALIFYING * GENERAL *
          *   NUMBER   *   STATUS   *  STATUS *
          *************************************
  S-0     * TRANSMISSION LOG / CONTROL RETURNS*
          *************************************
```

NOTE - TRANSMISSION LOG IS RETURNED WITH THE SAME SENSE AS THE COUNT PARAMETER IN ATTACHIO (+=WORDS, -=BYTES) AND BUFFER OFFSET IS THE SAME AS THE TARGET PARAMETER.

CONDITION CODE -

CCE - REQUEST COMPLETED. IOQ ELEMENT RETURNED.

CCL - PCB NUMBER IS ZERO, INDICATING THE REQUEST IS NO LONGER ASSOCIATED WITH A PROCESS.

DECLARATION:

```
PROCEDURE WAITFORIOX(IOQX);
  VALUE IOQX;   INTEGER IOQX;
  OPTION PRIVILEGED, UNCALLABLE;
```

FUNCTION:

THIS PROCEDURE CHECKS IF THE I/O REQUEST ASSOCIATED WITH IOQX HAS BEEN COMPLETED. IF IT HAS NOT, THE PCB INDEX OF THE CALLER IS SET INTO THE THE IOQ ELEMENT AND THE CALLER IS WAITED UNTIL THE I/O IS COMPLETED. IF THE I/O IS COMPLETED OR WHEN IT DOES COMPLETE AFTER THE CALLER IS WAITED, THE COMPLETION STATUS, TRANSFER COUNT/CONTROL AND QMISC WORD OF THE IOQ ELEMENT ARE RETURNED TO THE CALLER AND THE IOQ ELEMENT IS RETURNED TO THE FREE POOL.

INPUT:

IOQX - THE INDEX,RELATIVE TO THE BASE OF THE IOQ TABLE,OF THE IOQ ELEMENT ASSOCIATED WITH THE REQUEST. THIS INDEX IS RETURNED BY ATTACHIO WHEN A REQUEST IS UNBLOCKED.

TABLES ACCESSED:

IOQ

RETURN:

```
           0          7 8          12 13      15
           **************************************
   S-1     *    PCB     * QUALIFYING * GENERAL *
           *   NUMBER   *   STATUS   *  STATUS *
           **************************************
   S-0     * TRANSMISSION LOG / CONTROL RETURNS*
           **************************************
```

NOTE - THIS PROCEDURE RETURNS THE SAME DATA AS WAITFORIO WITH THE EXCEPTION THAT THE QMISC WORD OF THE IOQ ELEMENT FOR THE REQUEST IS RETURNED IN THE X REGISTER.

CONDITION CODE -

CCE - REQUEST COMPLETED. IOQ ELEMENT RETURNED.

CCL - PCB NUMBER IS ZERO, INDICATING THE REQUEST IS NO LONGER ASSOCIATED WITH A PROCESS.

DECLARATION:

```
PROCEDURE WRITE2(TWOCHARS);
  VALUE TRWOLCHARS;  INTEGER TWOCHARS;
  OPTION PRIVILEGED, UNCALLABLE;
```

FUNCTION:

THIS PROCEDURE WRITES TWO CHARACTERS TO THE SYSTEM CONSOLE DEVICE. THE CHARACTERS ARE WRITTEN DIRECTLY WITHOUT USING THE TERMINAL I/O SYSTEM.

INPUT:

TWOCHARS - CHARACTERS TO BE WRITTEN. THE FIRST CHARACTER OUTPUT IS CONTAINED IN THE LEFT BYTE AND THE SECOND CHARACTER IS CONTAINED IN THE RIGHT BYTE.

DB - SET TO SYSDB

INTERRUPTS MUST BE DISABLED BEFORE THE CALL.

NOTE:

INTERRUPTS ARE NOT ENABLED BY THIS PROCEDURE.

DECLARATION:

```
PROCEDURE ADDHEAD(NEW,LINKX,QN);
  VALUE LINKX, QN;  INTEGER LINKX, QN;
  INTEGER ARRAY NEW;
  OPTION PRIVILEGED, UNCALLABLE;
```

FUNCTION:

THIS PROCEDURE ADDS THE ELEMENT NEW TO THE BEGINNING OF THE SPECIFIED QUEUE.

INPUT:

NEW - SYSDB RELATIVE POINTER TO WORD ZERO OF THE ELEMENT TO BE ADDED.

LINKX - INDEX INTO THE DIT OF THE WORD USED TO HOLD THE RESOURCE QUEUE LINK.

QN - THE NUMBER OF THE I/O RESOURCE QUEUE TO WHICH THE NEW ELEMENT IS TO BE ADDED.

DB - SET TO SYSDB ON CALL

TABLES ACCESSED:

HEAD SYSDB

DECLARATION:

```
PROCEDURE ADDTAIL(NEW,LINKX,QN);
  VALUE LINKX, QN;  INTEGER LINKX, QN;
  INTEGER ARRAY NEW;
  OPTION PRIVILEGED, UNCALLABLE;
```

FUNCTION:

THIS PROCEDURE ADDS THE ELEMENT NEW TO THE END OF THE SPECIFIED QUEUE.

INPUT:

NEW - SYSDB RELATIVE POINTER TO WORD ZERO OF THE ELEMENT TO BE ADDED.

LINKX - INDEX INTO THE DIT OF THE WORD USED TO HOLD THE RESOURCE QUEUE LINK.

QN - THE NUMBER OF THE I/O RESOURCE QUEUE TO WHICH THE NEW ELEMENT IS TO BE ADDED.

DB - SET TO SYSDB ON CALL

TABLES ACCESSED:

TAIL SYSDB

NOTE:

INTERRUPTS ARE NOT ENABLED BY THIS PROCEDURE.

DECLARATION:

```
PROCEDURE AWAKETERMINAL(DITP);
  INTEGER ARRAY DITP;
  OPTION PRIVILEGED, UNCALLABLE;
```

FUNCTION:

THIS PROCEDURE SEARCHES THE IOQ OF THE DEVICE SPECIFIED BY DITP AND IF ANY BLOCKED REQUESTS ARE FOUND, THAT PROCESS IS AWAKEN TO RUN THE TERMINAL MONITOR. IF NO BLOCKED REQUESTS ARE FOUND, AWAKEIO IS CALLED TO RUN THE MONITOR IN THE SYSTEM I/O PROCESS. THE ATTEMPT TO RUN THE MONITORS IN A BLOCKED USER PROCESS IS DONE SO THAT LONG DELAYS CAN BE AVOIDED BETWEEN I/O OPERATIONS WHERE A PROCESS SWITCH IS REQUIRED TO COMPLETE A SEQUENCE, SUCH AS A PROMPT/READ INITIATION SEQUENCE.

INPUT:

DITP - SYSDB RELATIVE POINTER TO DIT.

DB - SET TO SYSDB.

TABLES ACCESSED:

DIT IOQ

DECLARATION:

```
LOGICAL PROCEDURE BREAKOK(DITP);
  VALUE DITP;  POINTER DITP;
  OPTION PRIVILEGED, UNCALLABLE;
```

FUNCTION:

THIS PROCEDURE CHECKS IF THE BREAK ENABLED BIT IS SET IN THE DIT AND THAT A FLUSH IS NOT IN PROGRESS. IT ALSO CHECKS THAT THE TERMINAL IS NOT IN THE CONSOLE MODE. IF THE CONDITIONS ARE SATISFACTORY A TRUE RETURN IS INDICATED.

INPUT:

DITP - SYSDB RELATIVE POINTER TO THE TERMINAL DIT.

DB - SET TO SYSDB ON CALL.

RETURN:

TRUE - BREAK ENABLED AND NOT FLUSHING AND TERMINAL NOT IN CONSOLE MODE.

FALSE - TERMINAL IN CONSOLE MODE OR BREAK NOT ENABLED OR ALREADY OCCURED.

TABLES ACCESSED:

DIT   LPDT

NOTE:

SSBREAKOK IS AN ENTRY POINT TO THIS PROCEDURE.

DECLARATION:

```
PROCEDURE CHECKINDEX(INDX,TBASE);
  VALUE INDX;  INTEGER INDX;
  INTEGER ARRAY TBASE;
  OPTION PRIVILEGED,UNCALLABLE;
```

FUNCTION:

THIS PROCEDURE CHECKS THAT INDX IS VALID INDEX IN THE TABLE WHOSE BASE IS TBASE. IF INDX IS OUT OF THE TABLE BOUNDS OR NOT A VALID INDEX THEN SUDDENDEATH 249 OCCURS.

INPUT:

INDX - TABLE RELATIVE INDEX OF THE ELEMENT TO BE CHECKED. IF INDX IS NEGATIVE AND TBASE IS THE SBUF TABLE THEN ONLY THE RANGE TEST IS PERFORMED ON INDX.

TBASE - POINTER TO THE BSE OF THE TABLE CONTAINING THE ELEMENT.

DB SET TO SYSDB.

DECLARATION:

```
PROCEDURE CHKCHANNELQUE(ILTO);
   VALUE ILTO;
   INTEGER ILTO;
   OPTION PRIVILEGED,UNCALLABLE;
```

FUNCTION:

THIS PROCEDURE MANAGES THE I/O CHANNEL RESOURCE SPECIFIED IN WORD 0 OF THE ILT TABLE. IT RELEASES THE CURRENT CONTROLLER FROM THE CHANNEL AND ASSIGNS THE FIRST QUEUED CONTROLLER TO THE CHANNEL RESOURCE. THE SIO PROGRAM FOR THAT CONTROLLER IS STARTED.

INPUT:

ILTO - WORD 0 OF THE ILT TABLE FOR THIS CONTROLLER. IT CONTAINS THE I/O CHANNEL RESOURCE QUEUE NUMBER.

NOTE:

DB MUST BE SET TO SYSDB.

TABLES ACCESSED:

HEAD BUSY DIT

DECLARATION:

```
PROCEDURE CHECKTQUEUE(DITP);
  INTEGER ARRAY DITP;
  OPTION PRIVILEGED, UNCALLABLE;
```

FUNCTION:

THIS PROCEDURE DECREMENTS THE TERMINAL READ AND WRITE COUNTERS AS NECESSARY. FOR ANY DEVICES WAITING TO BE STARTED, IT SETS DSTATE TO THE STATE STOPED IN NXTDSTATE AND CAUSES AN INTERRUPT SO THAT TIP WILL STARTED THE REQUESTED OPERATION.

INPUT:

DITP - SYSDB RELATIVE POINTER TO DIT.

DB - SET TO SYSDB.

TABLES ACCESSED:

DIT I/O RESOURCE QUEUES SYSDB

DECLARATION:

```
INTEGER PROCEDURE DEQUEUE(LINKX, QN);
  VALUE  LINKX, QN;   INTEGER LINKX, QN;
  OPTION PRIVILEGED, UNCALLABLE;
```

FUNCTION:

THIS PROCEDURE REMOVES THE FIRST ELEMENT, IF ANY IN THE I/O RESOURCE QUEUE NUMBER QN AND RETURNS IT. IF THE QUEUE IS EMPTY A -1 IS RETURNED. THE LINK WORD IN THE RETURNED ELEMENT IS SET TO ZERO AND THE REST OF THE QUEUE IS LINKED TO THE HEAD OF THE LIST.

INPUT:

LINKX - INDEX INTO THE DIT OF THE WORD USED TO HOLD THE RESOURCE QUEUE LINK.

QN - THE NUMBER OF THE I/O RESOURCE QUEUE TO WHICH THE NEW ELEMENT IS TO BE ADDED.

DB - SET TO SYSDB ON CALL

RETURN:

SYSDB RELATIVE POINTER TO THE DIT OF THE FIRST ELEMENT IN THE QUEUE. IF THE QUEUE WAS EMPTY, A -1 IS RETURNED.

TABLES ACCESSED:

DIT HEAD

DECLARATION:

```
PROCEDURE DMONITOR(DITP, ID, P1, P2);
  VALUE ID, P1, P2;  INTEGER ID, P1, P2;
  INTEGER ARRAY DITP;
  OPTION PRIVILEGED, UNCALLABLE;
```

FUNCTION:

DEPENDING ON THE MONITORING CODE FIELD IN DLAST OF THE DIT, THIS PROCEDURE WILL CREATE A FOUR WORD MONITOR ENTRY AND PLACE IT IN A TABLE (2 SYSBUFS WHOSE BASE IS IN DB+ 66). THE FIRST WORD OF THE ENTRY CONTAINS ID.(12:4) AND THE TIME FROM THE LAST EVENT. THE 2ND WORD HOLDS THE UNIT NUMBER, DSTATE, DSET STATUS BITS AND OTHER FLAGS. THIS ROUTINE CAN ALSO BE DIRECTED TO FORM A TIME HISTOGRAM OR CALL HELP.

INPUT:

DITP - SYSDB RELATIVE POINTER TO THE DIT

ID - IDENTIFIES THE CALLING PLACE . USED WITH MONITORING CODE TO DETERMINE IF A MONITORING FUNCTION IS TO BE EXECUTED.

P1 - PARAMETER 1, TO BE SAVED IN THE 3RD WORD OF THE ENTRY.

P2 - PARAMETER 2, TO BE SAVED IN THE 4TH WORD OF THE ENTRY. IF -1 THEN A DEFAULT SET OF FLAGS IS SAVED.

DB - SET TO SYSDB.

DECLARATION:

```
PROCEDURE DOCIO(ORDER,DEVICE);
  VALUE ORDER, DEVICE;   INTEGER ORDER, DEVICE;
  OPTION PRIVILEGED, UNCALLABLE;
```

FUNCTION:

THIS PROCEDURE OUTPUTS THE CONTROL WORD ORDER TO THE DEVICE WITH A CIO INSTRUCTION. IF THERE IS AN NON RESPONDING I/O INSTRUCTION ERROR, THE PROCEDURE IOFAILURE IS CALLED, RESULTING IN A SUDDENDEATH.

INPUT:

ORDER - THE WORD TO BE OUTPUT WITH A CIO INSTRUCTION.

DEVICE - IF GREATER THAN 128, THEN IT IS ASSUMED TO BE A DIT POINTER AND THE DRT NUMBER IS OBTAINED FROM THE ILT ASSOCIATED WITH THE DEVICE POINTED TO BY THE DIT POINTER.
IF EQUAL OR LESS THAN 128, DEVICE IS ASSUMED TO BE A DRT NUMB NUMBER AND THE I/O IS DONE DIRECTLY TO THAT DRT.

RETURN:

IF CIO INSTRUCTION ACCEPTED THEN A NORMAL RETURN. IF AN I/O INSTRUCTION ERROR, THEN NO RETURN.

TABLES ACCESSED:

DIT ILT

DECLARATION:

```
PROCEDURE DSET1;
  OPTION UNCALLABLE, PRIVILEGED
```

FUNCTION:

THIS PROCEDURE HANDLES INTERRUPTS OF THE FIRST DATA SET CONTROLLER. THIS BOARD HAS THE CA AND CD CONTROL SIGNALS AND MONITORS THE CF AND CC STATUS LINES. IF CA RISES A DATA SET READY SERVICE REQUEST GENERATED FOR THE TERMINAL MONITOR. IF CA FALLS, A DISCONNECT REQUEST IS GENERATED. WHEN CF FALLS, A CARRIER FAIL TIME OUT IS INITIATED AND WHEN CF RAISES ANY CARRIER FAIL TIME OUTS ARE ABORTED.

INPUT:

WHEN EXECUTION BEGINS, DB IS SET TO THE ILT FOR THE ASYNCHRONOUS TERMINAL CONTROLLER ASSOCIATED WITH THIS DATA SET CONTROLLER.

DECLARATION:

```
PROCEDURE DSET2;
  OPTION UNCALLABLE, PRIVILEGED
```

FUNCTION:

THIS PROCEDURE HANDLES INTERRUPTS OF THE SECOND DATA SET CONTROLLER. THIS BOARD HAS THE CH AND SA CONTROL SIGNALS AND MONITORS THE CB AND SB STATUS LINES. IF CB AND SB ARE BOTH UP AND THE DEVICE IS IN THE TURN TO WRITE STATE, AN INTERRUPT INDICATING THE TURN TO WRITE IS COMPLETE IS CAUSED ON THE TERMINAL CONTROLLER. IF THE TERMINAL IS DOING NOTHING OR WRITING AND SB FALLS, A BREAK SERVICE REQUEST IS SENT TO THE TERMINAL MONITOR.

INPUT:

WHEN EXECUTION BEGINS, DB IS SET TO THE ILT FOR THE ASYNCHRONOUS TERMINAL CONTROLLER ASSOCIATED WITH THIS DATA SET CONTROLLER.

TABLES ACCESSED:

HEAD   TAIL   SYSDB

DECLARATION:

```
PROCEDURE EOFCHECK(IOQP,BUF,CNT,HARDCHK);
VALUE IOQP,BUF,CNT,HARDCHK;
POINTER IOQP;
DOUBLE BUF;
INTEGER CNT,HARDCHK;
OPTION PRIVILEGED,UNCALLABLE;
```

FUNCTION:

THIS PROCEDURE CHECKS FOR AN END-OF-FILE CONDITION BASED EITHER ON THE EOF CONDITION WHEN THE LAST RECORD WAS READ OR THE EOF CONDITION OF THE CURRENT READ DATA. IF THE BUFFER POINTER IS ZERO, THE EOF CONDITION IS BASED ON THE LAST CONDITION STORED IN THE LOGICAL TO PHYSICAL DEVICE TABLE (LPDT). IF THE BUFFER POINTER IS NON-ZERO, THE CHECK IS BASED ON THE CURRENT HARDWARE EOF CONDITION IN THE LPDT AND THE DATA CONTAINED IN THE BUFFER. THE EOF CONDITION FOUND IS INSERTED INTO THE LPDT. IF AN EOF CONDITION IS DISCOVERED, THE EOF TYPE WILL BE RETURNED IN THE QUALIFYING STATUS OF THE IOQ, WHILE THE GENERAL STATUS WILL BE SET TO TWO AND THE COUNT IN THE IOQ WILL BE SET TO ZERO.

A HARDWARE EOF IS RETURNED IF THE HARDWARE EOF CONDITION IF FOUND IN THE LPDT OR A ":EOF:" IS FOUND IN THE BUFFER.

THE FOLLOWING TWO TABLES DEFINE THE EOF CONDITION RETURNED IN THE IOQ FOR EACH LEVEL OF CHECKING. THE LEVEL IS TAKEN FROM PARAMETER ONE BITS 13-15 OF THE IOQ.

PREREAD CHECK (BUF = 0)

		CURRENT LPDT CONDITION		
		NONE (0)	HARDWARE (1)	DATA DEPENDENT EOF (2-7)
C	L 0	-	-	-
H	E 1	-	EOF(1)	-
E	V 2	-	EOF(1)	EOF(2-7)
C	E 3	-	EOF(1)	EOF(2-7)
K	L 4	-	EOF(1)	EOF(2-7)
	5	-	-	-

POST READ CHECK (BUF <> 0)

CHECK LEVEL	EOF CONDITION FOUND NONE (0)	HARD. (1)	DATA (2)	EOD (3)	HELLO (4)	BYE (5)	JOB (6)	EOJ (7)
0	-	EOF(1)	-	-	-	-	-	-
1	-	EOF(1)	-	-	-	-	-	-
2	-	EOF(1)	EOF(2)	EOF(3)	-	-	EOF(6)	-
3	-	EOF(1)	EOF(2)	-	EOF(4)	EOF(5)	EOF(6)	-
4	-	EOF(1)	EOF(2)	-	-	-	EOF(6)	EOF(7)
5	-	-	-	-	-	-	-	-

NOTE THAT NO EOF CHECK IS MADE IS THE IOQ FUNCTION CODE IS NOT OF TYPE READ (FUNCT = 0). NO DATA EOF CHECK IS MADE IF CNT = 0.

INPUT:

IOQP - SYSDB RELATIVE POINTER TO THE I/O QUEUE ELEMENT ASSOCIATED WITH THIS REQUEST. THE CHECKING LEVELS IN IOQP(QPAR1).(13:3) ARE AS FOLLOWS:

- 0 - RESET AND READ
- 1 - CHECK FOR HARDWARE EOF, INCLUDING ":EOF:"
- 2 - CHECK FOR DATA READ EOF TYPES
  (EOF ON ":DATA",":EOD", AND ":JOB";
  BACKSPACE ":DATA" AND ":JOB".)
- 3 - CHECK FOR SESSION READ EOF TYPES
  (EOF ON "DATA","HELLO","BYE", AND "JOB;
  BACKSPACE "DATA","JOB", AND "HELLO".)
- 4 - CHECK FOR JOB READ EOF TYPES
  (EOF ON ":DATA",":JOB", AND ":EOJ";
  BACKSPACE ":DATA" AND ":JOB".)
- 5 - SKIP ALL EOF CHECKS

BUF - THE DOUBLE WORD ABSOLUTE ADRESS OF THE START OF THE DATA BUFFER.

CNT - THE NUMBER OF WORDS (+) OR BYTES (-) IN THE DATA BUFFER.

HARDCHK - THE TYPE OF HARDWARE EOF ":EOF:" CHECK TO BE MADE.

0 - OMIT ":EOF:" CHECK
> 0 - CHECK FOR ASCII ":EOF".
< 0 - CHECK FOR COLUMN BINARY ":EOF:".
NOTE: THE COLUMN BINARY CHECK OMITS ALL OTHER DATA CHECKS

OUTPUT:

THE EOF CONDITION IS RETURNED TO THE CALLER VIA THE STATUS WORD CONDTION CODE.

CCE - NO EOF WAS FOUND.
CCL - AN EOF CONDITION WAS FOUND, BACKSPACE THE BUFFER.
CCG - AN EOF CONDITION WAS FOUND, DO NOT BACKSPACE THE BUFFER.

THE QUALIFYING STATUS RETURNS ARE:

0 - NO EOF
1 - HARDWARE EOF, INCLUDING ":EOF:"
2 - ":DATA" TYPE EOF
3 - ":EOD" EOF
4 - "HELLO" EOF
5 - "BYE" EOF
6 - ":JOB" TYPE EOF
7 - ":EOJ" EOF

THE ABOVE CODES ARE ALSO INSERTED INTO THE LPDT ON A POST-READ CHECK

NOTE:

DB MUST BE SET TO SYSDB.

TABLES ACCESSED:
IOQ LPDT

DECLARATION:

```
INTEGER PROCEDURE GETSBUF(TYPE);
  VALUE TYPE;  INTEGER TYPE;
  OPTION PRIVILEGED, UNCALLABLE;
```

FUNCTION:

THIS PROCEDURE GETS AN ELEMENT FROM THE SBUF TABLE AND PETURNS A SYSDB RELATIVE POINTER TO THE ELEMENT. IF THE PRIMARY TABLE IS EMPTY THEN TYPE SPECIFIES WHETHER THE PROCESS IS TO BE IMPEDED OR AN ELEMENT IS TO BE GOTIEN FROM THE SECONDARY TABLE. IF THE PROCESS IS NOT TO BE IMPEDED AND THE SBUF TABLE IS EMPTY, A ZERO POINTER IS RETURNED.

INPUT:

TYPE - ACTION IF ELEMENT IS UNAVAILABLE
- 0 - IMPEDE CALLER
- 1 - GET ONLY FROM PRIMARY TABLE. RETURN ZERO IF TABLE EMPTY
- 2 - GET FROM SECONDARY AREA IF PRIMARY TABLE EMPTY. IF BOTH EMPTY RETURN ZERO.

DB SET TO SYSDB

RETURN:

SYSDB RELATIVE POINTER TO THE GOTTEN ELEMENT OR ZERO IF NONE IS AVAILABLE AND THE CALLER WAS NOT TO BE IMPEDED.

TABLES ACCESSED:

SBUF SYSDB

NOTE:

THIS PROCEDURE IS AN ENTRY POINT TO GETTBUF.

DECLARATION:

```
INTEGER PROCEDURE GETTBUF(TYPE);
  VALUE TYPE;  INTEGER TYPE;
  OPTION PRIVILEGED, UNCALLABLE;
```

FUNCTION:

THIS PROCEDURE GETS AN ELEMENT FROM THE TBUF TABLE AND RETURNS A SYSDB RELATIVE POINTER TO THE ELEMENT. IF THE PRIMARY TABLE IS EMPTY THEN TYPE SPECIFIES WHETHER THE PROCESS IS TO BE IMPEDED OR AN ELEMENT IS TO BE GOTTEN FROM THE SECONDARY TABLE. IF THE PROCESS IS NOT TO BE IMPEDED AND THE TBUF TABLE IS EMPTY, A ZERO POINTER IS RETURNED.

INPUT:

TYPE - ACTION IF ELEMENT IS UNAVAILABLE
- 0 - IMPEDE CALLER
- 1 - GET ONLY FROM PRIMARY TABLE. RETURN ZERO IF TABLE EMPTY
- 2 - GET FROM SECONDARY AREA IF PRIMARY TABLE EMPTY. IF BOTH EMPTY RETURN ZERO.

DB SET TO SYSDB

RETURN:

SYSDB RELATIVE POINTER TO THE GOTTEN ELEMENT OR ZERO IF NONE IS AVAILABLE AND THE CALLER WAS NOT TO BE IMPEDED.

TABLES ACCESSED:

TBUF SYSDB

NOTE:

GET100 AND GETSBUF ARE ENTRY POINTS TO THIS PROCEDURE.

DECLARATION:

PROCEDURE GIP;
OPTION PRIVILEGED,UNCALLABLE,INTERRUPT;

FUNCTION:

THIS PROCEDURE PROCESSES INTERRUPTS FOR ALL SIO DEVICES. IT SAVES THE HARDWARE STATUS IN THE DIT AND AWAKENS THE MONITOR FOR THIS DEVICE. IF THE DEVICE IS NOT OWNED, GIP WILL SET STATE 6 FOR SIODM. THIS WILL CAUSE THE DEVICE RECOGNITION SEQUENCE TO BEGIN.

INPUT:

NONE

NOTE:

DB MUST BE SET TO THE ILT TABLE ON ENTRY

TABLES ACCESSED

DIT ILT SYSDB

DECLARATION:

```
INITCHANNEL(DITP);
  INTEGER ARRAY DITP;
  OPTION UNCALLABLE, PRIVILEGED;
```

FUNCTION:

THIS PROCEDURE INITIALIZES THE INPUT, OUTPUT AND DATASET CONTROLLER CHANNELS FOR THE DEVICE SPECIFIED BY DITP. THE INPUT CHANNEL IS INITIALIZED WITH ECHO OFF AND TO SPEED SENSING IF THE SPEED SENSING FLAG IS SET IN THE DIT OTHERWISE IT IS SET TO THE SPEED SPECIFIED IN IN THE DIT. THE WRITE CHANNEL IS INITIALIZED TO THE PARITY CONTROL STATE AND SPEED SET IN THE DIT. IF THE SPEED WAS UNDEFINED, IT IS SET TO 2400 BAUD.

INPUT:

DITP - SYSDB RELATIVE POINTER TO DIT.

DB - SET TO SYSDB.

DECLARATION:

```
PROCEDURE IOFAILURE( DRTN, DITP );
VALUE DRTN;    INTEGER DRTN;
ARRAY DITP;
  OPTION UNCALLABLE, PRIVILEGED;
```

FUNCTION:

THIS PROCEDURE IS CALLED WHENEVER AN I/O INSTRUCTION FAILURE IS DETECTED. IF THE CONSOLE HAS BEEN INITIALIZED A SUDDENDEATH 201 OCCURS, OTHERWISE AN IRRECOVERABLE HALT OCCURS. IN ADDITION, THE DRT AND LOGICAL DEVICE NUMBER OF THE DEVICE ARE OUTPUT ON THE SYSTEM CONSOLE. IF THE FAILURE OCCURS ON THE SYSTEM CONSOLE, A HALT 15 OCCURS AND NO MESSAGE IS OUTPUT.

INPUT:

DRTN - DRT NUMBER TO WHICH THE I/O INSTRUCTION WAS DIRECTED.

DITP - IF NOT ZERO THEN A SYSDB RELATIVE POINTER TO THE DIT OF THE DEVICE TO WHICH THE I/O INSTRUCTION WAS DIRECTED. IF NOT ZERO THEN THE LOGICAL DEVICE NUMBER WILL ALSO BE PRINTED BEFORE THE SYSTEM FAILURE 201 OCCURS.

DECLARATION:

```
PROCEDURE IOIMPEDE(TABLE);
VALUE TABLE; POINTER TABLE;
OPTION PRIVILEGED,UNCALLABLE;
```

FUNCTION:

THIS PROCEDURE IMPEDES THE CURRENT PROCESS AND APPENDS THIS PROCESS TO THE LIST WAITING FOR THE RESOURCE SPECIFIED IN THE CALL.

INPUT:

TABLE - SYSDB RELATIVE POINTER TO THE RESOURCE TABLE THAT THIS PROCESS IS TO BE APPENDED TO.

TABLES ACCESSED:

PCB IOQ TBUF SBUF

DECLARATION:

```
LOGICAL PROCEDURE IOMESSAGE(DEST,CATN,P1,P2,P3,P4,DITP,BUF,CONT);
VALUE DEST,CATN,P1,P2,P3,P4,DITP,BUF,CONT;
INTEGER DEST,CATN,P1,P2,P3,P4,BUF,CONT;
INTEGER POINTER DITP;
OPTION VARIABLE,PRIVILEGED,UNCALLABLE;
```

FUNCTION:

THIS PROCEDURE FORMS THE INTERFACE BETWEEN THE I/O SYSTEM AND THE MESSAGE SYSTEM OF MPE. ITS MAJOR FUNCTION IS TO PROVIDE A BUFFER BETWEEN THE SYSTEMS SINCE THE I/O DOES NOT RUN IN A PROCESS. THE PROCEDURE GETS AN IOQ ELEMENT, CONSTRUCTS THE FOLLOWING BLOCK OF DATA, LINKS THE BLOCK TO THE I/O MESSAGE LIST, AND WAKES THE I/O MESSAGE PROCESS:

WORD USE

0 - LINK
1 - DEST
2 - CATN
3 - P1
4 - P2
5 - P3
6 - P4
7 - DITP
8 - BUF
9 - CONT
10 - VARIABLE MASK

INPUT:

THE INPUT PARAMETERS CORRESPOND TO THE PARAMETERS IN PUTMESSAGE EXCEPT AS FOLLOWS:
IF A REPLY IS EXPECTED, THEN

DITP - A SYSDB RELATIVE POINTER TO THE DEVICE INFOMATION TABLE.

BUF - THE WORD OFFSET FROM WORD 0 OF THE DIT TO WORD 0 OF THE REPLY BUFFER.

RETURN:

FALSE - NO IOQ ELEMENT AVAILABLE.

TRUE - MESSAGE HAS BEEN QUEUED FOR OUTPUT

NOTE:

DB MUST BE SET TO SYSDB.

DECLARATION:

PROCEDURE IOMESSPROC;
OPTION PRIVILEGED,UNCALLABLE;

FUNCTION:

THIS PROCEDURE IS THE PROCESS INTERFACE BETWEEN IOMESSAGE AND PUTMESSAGE, AND LOGERROR AND LOG. IT DOES THIS BY TRANSFERRING THE INFOMATION CONTAINED IN THE ELEMENT TO THE APPPOPRIATE ROUTINE AND RETURNING THE ELEMENT TO THE FREELIST. IF A MESSAGE REPLY IS EXPECTED, THE PROCEDURE WILL CALL IOWAKE TO AWAKEN THE I/O WHEN THE REPLY HAS BEEN RECEIVED.

INPUT:

NONE

OUTPUT:

NONE

TABLES ACCESSED:

DIT IOQ

DECLARATION:

```
PROCEDURE IOUNIMPEDE(TABLE);
VALUE TABLE; POINTER TABLE;
OPTION PRIVILEGED,UNCALLABLE;
```

FUNCTION:

THIS PROCEDURE REMOVES THE I/O IMPEDIMENT ASSOCIATED WITH THE HEAD PROCESS WAITING FOR THE TABLE REFERENCED IN THE CALL.

INPUT:

TABLE - SYSDB RELATIVE POINTER TO THE RESOURCE TABLE.

TABLES ACCESSED:

PCB   IOQ   TBUF   SBUF

DECLARATION:

```
PROCEDURE LDEVNOTRDY(DITP);
VALUE DITP;
POINTER DITP;
OPTION PRIVILEGED,UNCALLABLE;
```

FUNCTION:

THIS PROCEDURE SENDS A "NOT READY" MESSAGE TO THE SYSTEM CONSOLE FOR THE DEVICE SPECIFIED BY DITP.

INPUT:

DITP - A SYSDB RELATIVE POINTER TO THE DEVICE INFORMATION TABLE.

NOTE:

DB MUST BE SET TO SYSDB

TABLES ACCESSED:

NONE

DECLATATION:

```
PROCEDURE LOGERROR(DITP,IOQP,ERRSTAT);
VALUE DITP,IOQP,ERRSTAT;
INTEGER ERRSTAT;
INTEGER POINTER DITP,IOQP;
OPTION PRIVILEDEG,UNCALLABLE;
```

FUNCTION:

THIS PROCEDURE PROVIDES THE INTERFACE BETWEEN THE I/O SYSTEM AND THE LOG PROCEDURE FOR I/O ERROR LOGGING. IT OBTAINS A TBUF, CONSTRUCTS THE FOLLOWING BLOCK OF DATA, AND AWAKENS IOMESSPROC:

```
WORD USE
  0 - LINK                          1 - DITP
  2 - HARDWARE ERROR STATUS         3 - QFLAG
  4 - QLDEV                         5 - QMISC
  6 - QDSTN                         7 - QADDR
  8 - QFUNC                         9 - QWBCT
 10 - QPAR1                        11 - QPAR2
 12 - QSTAT
 13 - (4:4) - DEVICE SUBTYPE, (8:8) - DEVICE TYPE
 14 - (0:8) - UNIT NUMBER, (8:8) - DRT NUMBER
```

INPUT:

DITP - A SYSDB RELATIVE POINTER TO THE DEVICE INFORMATION TABLE

IOQP - A SYSDB RELATIVE POINTER TO THE I/O QUEUE ELEMENT

ERRSTAT - THE I/O HARDWARE STATUS AT THE TIME OF THE ERROR

NOTE:

DB MUST BE SET TO SYSDB.

TABLES ACCESSED:

DIT IOQ TBUF

NOTE:

DB MUST BE SET TO SYSDB.

DECLARATION:

```
PROCEDURE MASTERCLEAR(DITP);
  INTEGER ARRAY DITP;
  OPTION UNCALLABLE, PRIVILEGED;
```

FUNCTION:

THIS PROCEDURE ISSUES A MASTER CLEAR CONTROL FOLLOWED BY A CLEAR INTERRUPTS CONTROL TO THE CONTROLLER IDENTIFIED BY DITP. THE SIO PROGRAM FLAGS AND COUNTS ARE CLEANED UP AS IF A VALID SIO COMPLETION HAD OCCURED.

INPUT:

DITP - SYSDB RELATIVE POINTER TO DIT.

DB - SET TO SYSDB.

DECLARATION:

```
MPXWRITE( DATA, DITP);
  VALUE FUNCTION;  INTEGER FUNCTION;
  INTEGER ARRAY DITP;
  OPTION UNCALLABLE, PRIVILEGED;
```

FUNCTION:

THIS PROCEDURE OUTPUTS THE WORD DATA TO THE DEVICE IDENTIFIED BY DITP. IF IT IS A SYNC CHARACTER, DATA IS MODIFED TO BE THE PROPER SYNC CHARACTER FOR THE CURRENT PARITY BEING GENERATED.
IF A SYNC CHARACTER IS TO BE OUTPUT AND THE DEVICE IS ALREADY WAITING FOR A WRITE INTERRUPT THEN NO ACTION IS TAKEN.

INPUT:

DITP - SYSDB RELATIVE POINTER TO DIT.

DATA - WORD TO BE OUTPUT TO THE UNIT IDENTIFIED BY DITP.

DB - SET TO SYSDB.

DECLARATION:

```
PROCEDURE RETURNIOQ(IOQP);
  VALUE IOQP;  POINTER IOQP;
  OPTION PRIVILEGED, UNCALLABLE;
```

FUNCTION:

THIS PROCEDURE RETURNS THE ELEMENT POINTED TO BY IOQP TO THE FREE LIST OF THE IOQ TABLE. IF A PROCESS HAS BEEN IMPEDED WAITING FOR AN ELEMENT FROM THE TABLE, THE IMPEDEMENT IS REMOVED.

INPUT:

IOQP - SYSDB RELATIVE POINTER TO THE ELEMENT TO BE RETURNED

DB SET TO SYSDB

TABLES ACCESSED:

SYSDB IOQ

NOTE:

THIS PROCEDURE IS AN ENTRY POINT TO GETTBUF.

DECLARATION:

```
PROCEDURE RETURNSBUF(SBUFP);
  VALUE SBUFP;   POINTER SBUFP;
  OPTION PRIVILEGED, UNCALLABLE;
```

FUNCTION:

THIS PROCEDURE RETURNS THE ELEMENT POINTED TO BY SBUFP TO THE FREE LIST OF THE SBUF TABLE. IF A PROCESS HAS BEEN IMPEDED WAITING FOR AN ELEMENT FROM THE TABLE, THE IMPEDEMENT IS REMOVED.

INPUT:

SBUFP - SYSDB RELATIVE POINTER TO THE ELEMENT TO BE RETURNED

DB SET TO SYSDB

TABLES ACCESSED:

SYSDB SBUF

NOTE:

THIS PROCEDURE IS AN ENTRY POINT TO RETURNTBUF.

DECLARATION:

```
PROCEDURE RETURNTBUF(TBUFP);
  VALUE TBUFP;  POINTER TBUFP;
  OPTION PRIVILEGED, UNCALLABLE;
```

FUNCTION:

THIS PROCEDURE RETURNS THE ELEMENT POINTED TO BY TBUFP TO THE FREE LIST OF THE TBUF TABLE. IF A PROCESS HAS BEEN IMPEDED WAITING FOR AN ELEMENT FROM THE TABLE, THE IMPEDEMENT IS REMOVED.

INPUT:

TBUFP - SYSDB RELATIVE POINTER TO THE ELEMENT TO BE RETURNED

DB SET TO SYSDB

TABLES ACCESSED:

SYSDB   TBUF

NOTE:

GETIOQ AND GETTBUF ARE ENTRY POINTS TO THIS PROCEDURE.

DECLARATION:

```
PROCEDURE SENDSYNC(NEWDS,DITP);
  VALUE NEWDS;   INTEGER NEWDS;
  INTEGER ARRAY DITP;
  OPTION PRIVILEGED, UNCALLABLE;
```

FUNCTION:

THIS PROCEDURE SETS THE DEVICE DSTATE TO NEWDS AND SENDS A SYNC CHARACTER OUT, CAUSING AN INTERRUPT ON THE ASYNCHRONOUS MPX. THE INTERRUPT HANDLER TIP, THEN TAKES APPROPRIATE ACTION

INPUT:

NEWDS - NEW DSTATE TO BE SET INTO DSTATE OF THE DEVICE.

DITP - SYSDB RELATIVE POINTER TO DIT.

DB - SET TO SYSDB.

TABLES ACCESSED:

DIT

DECLARATION:

```
SETREADERROR(IOQP,ENUMB);
  VALUE ENUMB;    INTEGER ENUMB;
  INTEGER ARRAY IOQP;
  OPTION PRIVILEGED, UNCALLABLE;
```

FUNCTION:

THIS PROCEDURE SETS THE ERROR NUMBER ENUMB INTO THE READERRORS FIELD OF THE IOQ ELEMENT POINTED TO BY IOQP. THE ERROR NUMBER IS NOT CHANGED IF ENUMB IS LESS THAN THE CURRENT VALUE OF READERRORS.

INPUT:

IOQP - SYSDB RELATIVE POINT ER TO THE IOQ ELEMENT.

ENUMB - ERROR NUMBER.

DB - SET TO SYSDB

TABLES ACCESSED:

IOQ

DECLARATION:

```
PROCEDURE SETTERMTYPE(TYPE,INSPD,OUTSPD,DITP);
  VALUE TYPE, INSPD, OUTSPD;  INTEGER TYPE, INSPD, OUTSPD;
  ARRAY DITP;     OPTION PRIVILEGED, UNCALLABLE;
```

FUNCTION:

THIS PROCEDURE CHECKS FOR A VALID TYPE SPECIFICATION AND CHECKS THAT THE SPEED IS ALLOWED. IF THE SPEED AND TYPE ARE CORRECT, IT SETS THE TYPE INTO THE DIT AND SETS THE SPEED AND SYNC COUNTS

INPUT:

```
TYPE    - TERMINAL TYPE AS SPECIFIED IN THE MPE ERS

INSPD   - INPUT SPEED NUMBER
            0 - 240                      4 - 30 CPS
            1 - 240 CPS                  5 - 15 CPS
            2 - 120 CPS                  6 - 10 CPS
            3 -  60 CPS                  7 - 14 CPS

OUTSPD  - OUTPUT SPEED NUMBER
              VALUES SAME AS INSPD

DITP    - SYSDB RELATIVE POINTER TO DIT.

DB      - SET TO SYSDB.
```

RETURN:

CCL - SPEED OR TYPE INCORRECT OR INCOMPATABLE

CCG - NEW TYPE, SPEED AND SYNC COUNTS SET

TABLES ACCESSED:

DIT

DECLARATION:

```
LOGICAL PROCEDURE SSBREAKOK(DITP);
  VALUE DITP;  POINTER DITP;
  OPTION PRIVILEGED, UNCALLABLE;
```

FUNCTION:

THIS PROCEDURE CHECKS IF THE SSBREAK ENABLED BIT IS SET IN THE DIT AND IF THE SSBREAK OCCURED BIT IS CLEAR IN THE LPDT. IT ALSO CHECKS THAT THE TERMINAL IS NOT IN THE CONSOLE MODE. IF THE CONDITIONS ARE SATISFACTORY A TRUE RETURN IS INDICATED.

INPUT:

DITP - SYSDB RELATIVE POINTER TO THE TERMINAL DIT.

DB - SET TO SYSDB ON CALL.

RETURN:

TRUE - SSBREAK ENABLED AND NOT ALREADY OCCURED AND TERMINAL NOT IN CONSOLE MODE.

FALSE - TERMINAL IN CONSOLE MODE OR SSBREAK NOT ENABLED OR ALREADY OCCURED.

TABLES ACCESSED:

DIT LPDT

NOTE:

THIS PROCEDURE IS AN ENTRY POINT IN BREAKOK.

DECLARATION:

```
PROCEDURE SIODM(DITP,FLAGS);
VALUE DITP,FLAGS; INTEGER POINTER DITP; LOGICAL FLAGS;
OPTION PRIVILEGED,UNCALLABLE;
```

FUNCTION:

THE SIO DEVICE MONITOR CONTROLLS ALL SYSTEM RELATED ACTIVITY FOR ALL SIO DEVICES IN THE SYSTEM. IT USES A STATE DRIVEN EVENT ALGORITHM TO DETERMINE WHAT ACTION TO TAKE WHEN IT IS CALLED. THE STATES ARE AS FOLLOWS:

```
 0  -  START REQUEST
 1  -  NOT USED
 2  -  CALL DRIVER INITIATOR
 3  -  CALL DRIVER COMPLETOR
 4  -  UNUSED
 5  -  COMPLETE REQUEST
 6  -  UNEXPECTED INTERRUPT
 7  -  START OPERATOR INTERVENTION WAIT
10  -  WAIT FOR OPERATOR INTERVENTION (RESTART AT STATE 0)
11  -  WAIT FOR DATA MAKEPRESENT AND FREEZE
12  -  WAIT FOR INITIATOR CODE MAKEPRESENT AND FREEZE
13  -  WAIT FOR I/O COMPLETION
14  -  WAIT FOR DEVICE CONTROLLER AVAILABLE
15  -  UNUSED
16  -  WAIT FOR INITIATOR CODE MAKEPRESENT
17  -  WAIT FOR COMPLETOR CODE MAKEPRESENT
```

INPUT:

DITP - SYSDB RELATIVE POINTER TO THE DIT REQUIRING SERVICE

```
FLAGS.(0:10) - MUST BE ZERO
     .(12:1) - NOT USED
     .(14:2) - MUST BE ZERO
```

DB MUST BE SET AT SYSDB

TABLES ACCESSED:

DIT IOQ CST DST DLT ILT LPDT

DECLARATION:

```
PROCEDURE STARTIO(DITP,SIOP,QUEUE);
VALUE DITP,SIOP,QUEUE; INTEGER POINTER DITP,SIOP;
LOGICAL QUEUE;
OPTION PRIVILEGED,UNCALLABLE;
```

FUNCTION:

THIS PROCEDURE STARTS AN I/O PROGRAM DIRECTED TO THE PHYSICAL DEVICE SPECIFIED BY DITP. IF THE I/O CANNOT BE STARTED BECAUSE THE DEVICE CONTROLLER OR CHANNEL IS BUSY, THE CALLER IS INFORMED VIA THE CONDITION CODE AND, IF QUEUE IS SET TRUE, THE DIT WILL BE APPENDED TO THE CHANNEL REQUEST QUEUE.

INPUT:

DITP - SYSDB RELATIVE POINTER TO THE DIT

SIOP - SYSDB RELATIVE POINTER TO THE START OF THE I/O PROGRAM

QUEUE - IF TRUE, THE DITP WILL BE APPENDED TO THE CHANNEL REQUEST QUEUE IF THE I/O PROGRAM CANNOT BE STARTED. NOTE THAT SIOP MUST BE THE SAME VALUE AS SIOP IN THE ILT TABLE IF THIS OPTION IS USED.

DB MUST BE SET TO SYSDB

RETURN:

CONDITION CODE -

CCE - I/O PROGRAM STARTED
CCG - I/O PROGRAM NOT STARTED, DEVICE CONTROLLER BUSY
CCL - I/O PROGRAM NOT STARTED, CHANNEL BUSY

TABLES ACCESSED:

DIT ILT

DECLARTION:

```
PROCEDURE SYSIOPROC;
  OPTION PRIVILEGED, UNCALLABLE;
```

FUNCTION:

THIS PROCEDURE IS USED BY THE INITIAL TO GENERATE A PROCESS, WHICH PROCESSES REQUESTS QUEUED BY AWAKEIO TO RUN MONITORS WHEN THEY CAN NOT NOT BE EXECUTED IN THE ENVIRONMENT OF THE CALLER OF AWAKEIO. THIS PROCESS DEQUEUES THE REQUEST AND CALLS THE APPROPRIATE MONITOR. WHEN NO FURTHER REQUESTS ARE QUEUED, IT "WAITS".

INPUT:

QUEUED REQUESTS.

TABLES ACCESSED:

DIT   HEAD

DECLARATION:

```
PROCEDURE STARTTIMEOUT( TYPE, DITP );
  VALUE TYPE;     INTEGER TYPE;
  ARRAY DITP;
  OPTION UNCALLABLE, PRIVILEGED;
```

FUNCTION:

THIS PROCEDURE STARTS A TIME OUT REQUEST OF THE TYPE SPECIFIED. IN SOME CASES, IF A REQUEST IS ALREADY IN PROGRESS THE CURRENT REQUEST WILL BE ABORTED AND A NEW TIME OUT STARTED. OTHERWISE IF A REQUEST OF THE TYPE SPECIFIED IS IN PROGRESS, NO ACTION IS TAKEN.

INPUT:

```
TYPE   - SPECIFIES THE TYPE OF TIME OUT TO BE STARTED.
           0 - 2640 BLOCK MODE READ OR WRITE ENQ/ACK WAIT
           1 - CARRIER FAILED
           2 - TURNING 202 MODEM TO WRITE
           3 - TIME OUT A READ OPERATION
           4 - LOGON TIME OUT
           5 - HANG UP SEQUENCE TIMEOUT
           6 - SPEED SENSING TIME OUT
           7 - DISCONNECT SPEED SENSING TIMEOUT

DITP   - SYSDB RELATIVE POINTER TO THE DIT

DB     - SET TO SYSDB
```

TABLE ACCESSED:

DIT

DECLARATION:

```
PROCEDURE STOPTIMEOUT( TYPE, DITP );
  VALUE TYPE;     INTEGER TYPE;
  ARRAY DITP;
  OPTION UNCALLABLE, PRIVILEGED;
```

FUNCTION:

THIS PROCEDURE ABORTS ANY OUTSTANDING TIME OUT REQUESTS IN PROGRESS OF THE TYPE SPECIFIED FOR THE DEVICE IDENTIFIED BY DITP. IT ALSO CLEARS THE ASSOCIATED TIME OUT EXPIRED SERVICE REQUEST BIT IN THE DRQST WORD OF THE DIT.

INPUT:

TYPE - SPECIFIES THE TYPE OF TIME OUT TO BE STOPPED.

- 0 - 2640 BLOCK MODE READ OR WRITE ENQ/ACK WAIT
- 1 - CARRIER FAILED
- 2 - TURNING 202 MODEM TO WRITE
- 3 - TIME OUT A READ OPERATION
- 4 - LOGON TIME OUT
- 5 - HANG UP SEQUENCE TIMEOUT
- 6 - SPEED SENSING TIME OUT
- 7 - DISCONNECT SPEED SENSING TIMEOUT

DITP - SYSDB RELATIVE POINTER TO THE DIT

DB - SET TO SYSDB

TABLE ACCESSED:

DIT

DECLARATION:

```
PROCEDURE TERMIOM(DITP,FLAGS);
  VALUE FLAGS;  INTEGER FLAGS;
  INTEGER ARRAY DITP;
  OPTION UNCALLABLE, PRIVILEGED;
```

FUNCTION:

THIS PROCEDURE IS THE ASYNCHRONOUS TERMINAL CONTROLLER MONITOR AND DRIVER. IT PROCESSES REQUESTS QUEUED VIA ATTACHIO, REQUESTS FOR I/O REQUEST ABORTION AND REQUESTS GENERATED BY THE TERMINAL AND DATASET INTERRUPT HANDLERS, TIP, DSET1 AND DSET2.

INPUT:

DITP - SYSDB RELATIVE POINTER TO THE DIT.

FLAGS.(0:10) - DST NUMBER OF THE CALLERS STACK OR ZERO. IF ZERO IS SPECIFIED CERTAIN DATA MOVES TO THE TERMINALBUFFERS WILL NOT BE EXECUTED ON THE CALLERS STACK.

.(13:1) - IMPEDABLE FLAG. IF SET THEN EXECUTION OF THIS ROUTINE CAN BE IMPEDED BECAUSE OF CODE ABSENCE OR FOR OTHER REASONS. IF CLEAR THEN THIS ROUTINE WILL NOT BE IMPEDED OR WAIT FOR ANY REASON. IF SET THEN ONE LEVEL OF PSEUDODISABLE MUST BE IN EFFECT WHEN AWAKEIO IS CALLED.

DB - SET TO SYSDB

DECLARATION:

```
PROCEDURE TIPX;
  OPTION UNCALLABLE, PRIVILEGED;
```

FUNCTION:

THIS PROCEDURE CONTAINS THE ROUTINES FOR HANDLING INTERRUPTS GENERATED BY THE ASYNCHRONOUS TERMINAL CONTROLLER. IT DOES EDITING ON BOTH INPUT AND OUTPUT, GENERATES BREAK AND SUBSYSTEM BREAK REQUESTS. IT ALSO DOES SPEED SENSING AND GENERATES CONSOLE INTERRUPT REQUESTS. ALL READS ARE TERMINATED AND CLEANED UP BY THIS PROCEDURE. THE ACTION TAKEN ON AN INTERRUPT IS A FUNCTION OF THE CAUSE OF THE INTERRUPT AND THE CURRENT DEVICE STATE. THE INTERRUPT HANDLER BEGINS AT THE ENTRY POINT TIP IN ORDER THAT CODE CAN PRECEED THE BEGINING OF THE ROUTINE TO REDUCE THE NEED FOR INDIRECT BRANCHES AND HENCE INCREASE THE CHARACTER PROCESSING SPEED.

INPUT:

WHEN EXECUTION BEGINS, DB IS SET TO THE ILT FOR THE ASYNCHRONOUS TERMINAL CONTROLLER ASSOCIATED WITH THIS DATA SET CONTROLLER.

NOTE:

THE ENTRY POINT OF THIS PROCEDURE IS TIP TO ALLOW FOR CODE BEFORE THE ENTRY POINT TO ELIMINATE INDIRECT BRANCHES. TIPX IS NOT CALLED.

DECLARATION:

```
PROCEDURE WRITECHAR(CHAR);
  VALUE CHAR;  INTEGER CHAR;
  OPPTION PRIVILEGED, UNCALLABLE;
```

FUNCTION:

THIS PROCEDURE WRITES ONE CHARACTER TO THE SYSTEM CONSOLE DEVICE. THE CHARACTER IS WRITTEN DIRECTLY WITHOUT USING THE TERMINAL I/O SYSTEM.

INPUT:

CHAR - CHARACTER TO BE OUTPUT TO THE SYSTEM CONSOLE.

DB   - SET TO SYSDB.

INTERRUPTS MUST BE DISABLED BEFORE THE CALL.

NOTE:

INTERRUPTS ARE NOT ENABLED BY THIS PROCEDURE.

MPE/30

I/O SYSTEM

I.M.S.*

Tom Ellestad
12/12/75

MPE/30 I/O SYSTEM

The I/O System is organized to make the handling of each device or logical unit as independent from one another as possible. This philospny is particularly suited for terminals, where each device is independent from a software standpoint. For devices which must compete for common resources such as controllers, or SIO bandwidth, a queue is formed for each resource. Thus if a device desires to use a resource and it is busy, a request is queued and when the resource is released the System assigns it to the next device in the request queue.

The I/O system provides for three types of Monitors:

1.) Able to execute on any stack including the ICS.

2.) Able to execute in a user process or in the I/O process.

3.) Executes only in its own process with no restrictions.

The standard system monitors are normally type 1 or type 2. The SIO device monitors are generally type 1 and the terminal monitors are type 2. These monitors are procedures which execute to completion, that is they have no explicit waits. Type 1 monitors may not have explicit or implicit waits.

Normally I/O is initiated in the users process and completed on the ICS for SIO devices. For terminals, the driver is normally executed in the users process for both initiation and completion. Drivers which are not resident when called by the SIO monitor are executed in the memory management process when they become resident.

I/O TABLES

There are six I/O Tables not including the resource request queu management tables. Except for the Driver Linkage Table (DLT), any tabl can be reached from any other if the unit number is known.

a.) Logical to Physical Device Table - LPDT

This table consists of two word entries. The first word is SYSDB relative pointer to the DIT and the second contains variou flags. This table maps a logical device into a physical devic through the DIT for the device.

b.) Device Reference Table - DRT

Defines the interrpt handler and contains a pointer to th Interrupt Linkage Table.

c.) Interrupt Linkage Table - ILT

There one ILT for each DRT. This table contains unit extractio information and a set of DIT pointers; one for each device on th controller connected to this DRT. Knowing the unit number, th DIT for the unit interrupting can be accessed through this ta

d.) Device Information Table - DIT

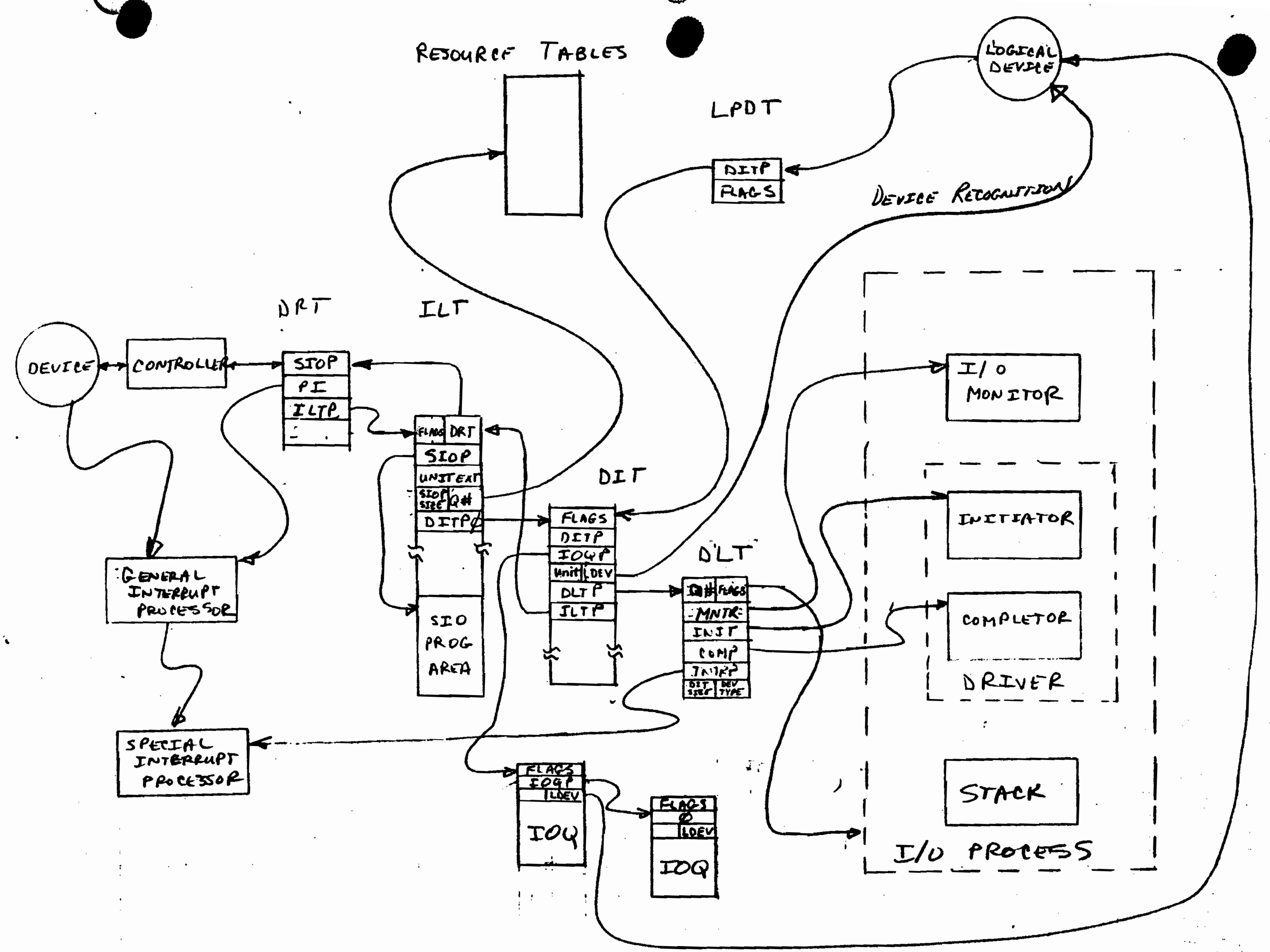
There is one DIT for each device on the system. This table link the requests to the device and contains information defining th current state of the device. In addition, the DIT contain pointers to a Driver Linkage Table and an Interrpt Linkage Table Linked DITs form request queues for I/O System Resources and I/ process requests.

e.) Driver Linkage Table - DLT

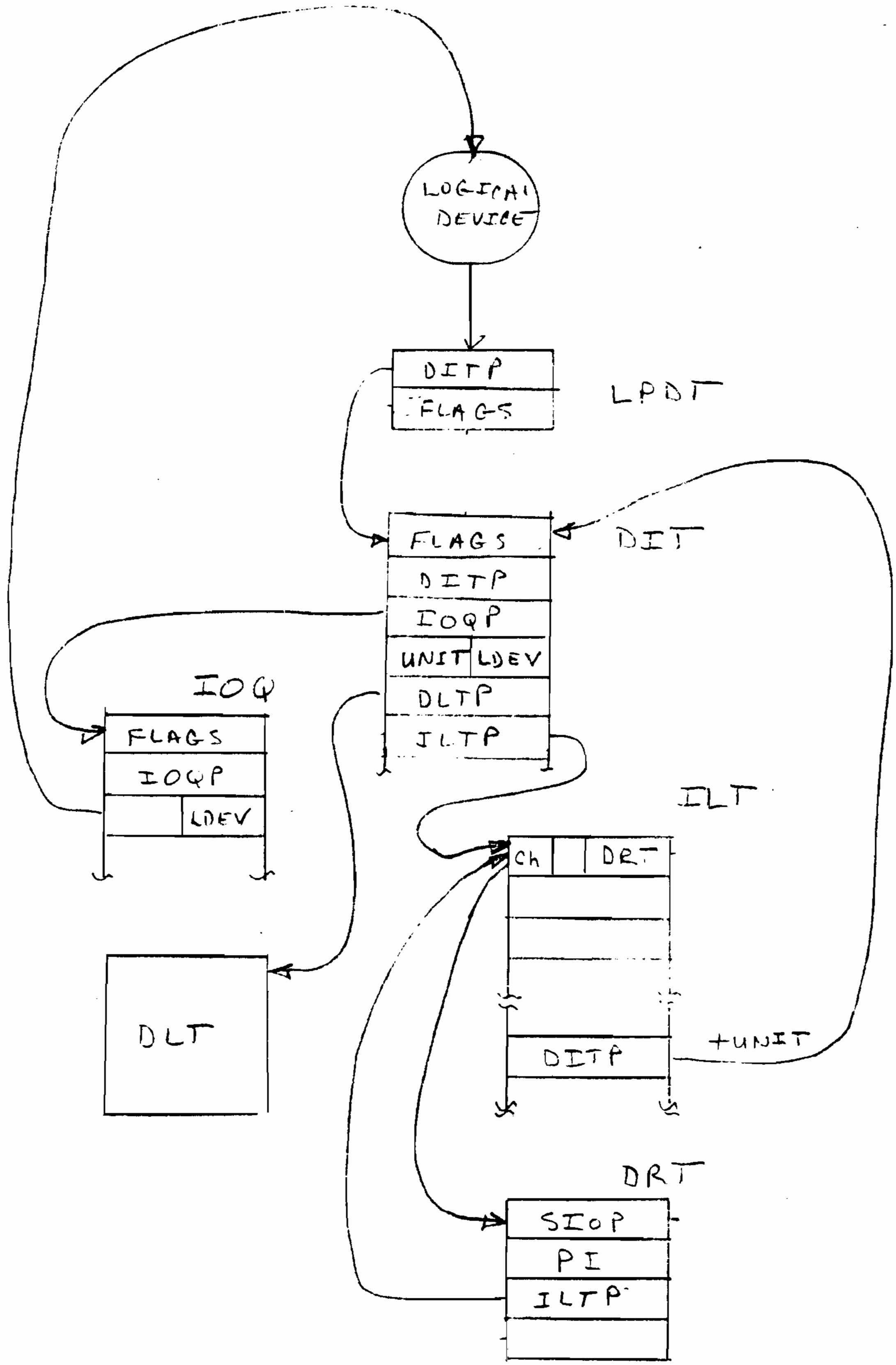
This table contains the Plabels for the Monitor, Initiation Completor and Interrupt Handler procedures ot be used in sevicin this device. There is one DLT for each type of driver in th system. A pointer in the DIT allows different devices on a controller to be serviced by different drivers.

f.) I/O Request Block - IOQ

One IOQ element is associated with each I/O request. The IO element contains all the necessary parameters to perform the I/ and information defining the current state of the request. Linke entries form the request queue for a device.



I/O TABLE RELATIONSHIPS



I/O TABLE LINKAGE

TABLE ELEMENT ALLOCATION

The allocatin of the elements in the IOQ, terminal buffer (TBUF) and system buffer (SBUF) tables is of concern to the I/O System.

These tables are in the form of a linked list of the free elements. For the SBUFs, the -1 word of entry is the link to the next element. For the TBUFs, word zero is the link and word 1 is the link for the IOQ elements.

Each table has an 8 word header beginning at the base of the table. The first four words of the header are for managing the table and the second four are for monitoring table activity.

The entries follow the header beginning at word eight.

Elements will be gotten from the beginning of the free list, pointed to by the head and returned to the end of the free list pointed to by the tail.

When the free list is empty, the head index is zero and the tail index is set to point at the head index.

The tables are divided into two areas; a primary area and a secondary area. Most requests are gotten from the primary area. The secondary area is used only for critical requirements when the primary area is exhausted. These areas are logical areas determined by parameters in the header.

The utility of these core resident tables is seriously reduced if their use is not restricted to Dynamic situations. Since the file system may tie up system buffers for extended periods, it is not allowed to get any system buffers if the number requested would leave less than 8 buffers in the primary area.

One of the three responses must be specified to the routines which allocate elements from the I/O system tables.

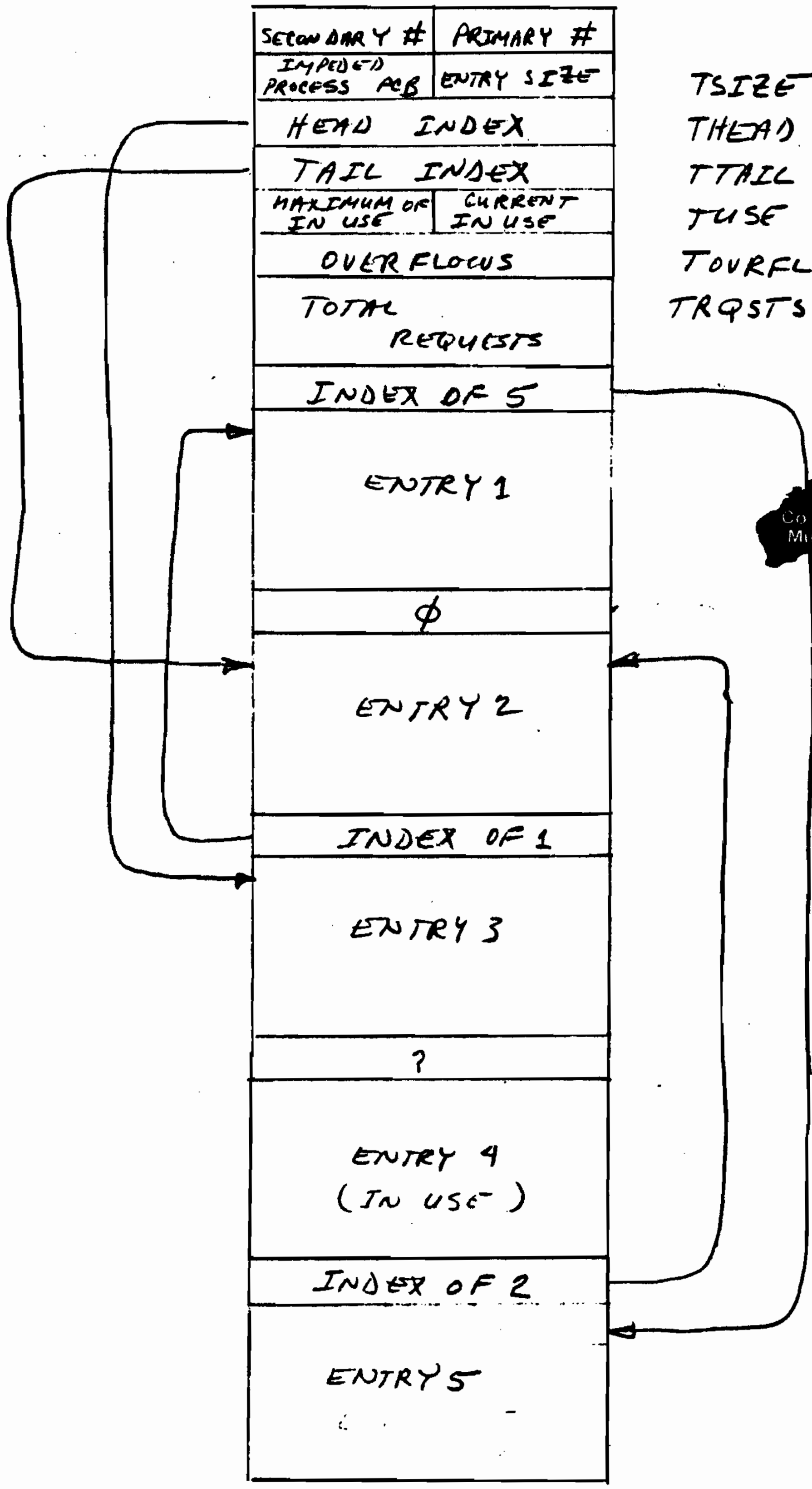
1.) Impede caller if primary area is empty.
2.) Get from primary area only.
3.) Get from secondary area if primary area is empty.

Request types 2 and 3 return an indication to the caller if the request could not be satisfied. The following table specifies the the types of calls for element allocation and the action if an element is not available.

BUFFER USER	CALL TYPE	FINAL ACTION
SBUF'S		
PTAPE	Impede	-
Bad Track	Primary	Forget Request
TBUF'S		
Terminal write (impedable)	Impede	-
Terminal write (not impedable)	Primary	I/O Error
Terminal Read on ICS	Secondary	I/O Error
LOGERROR	Secondary	Forget Request
IOQ'S		
ATTACHIO (not impedable)	Primary	Return IOQX=0
ATTACHIO (impedable)	Impede	-
SIODM	Secondary	Sudden Death
IOMESSAGE	Secondary	I/O Error

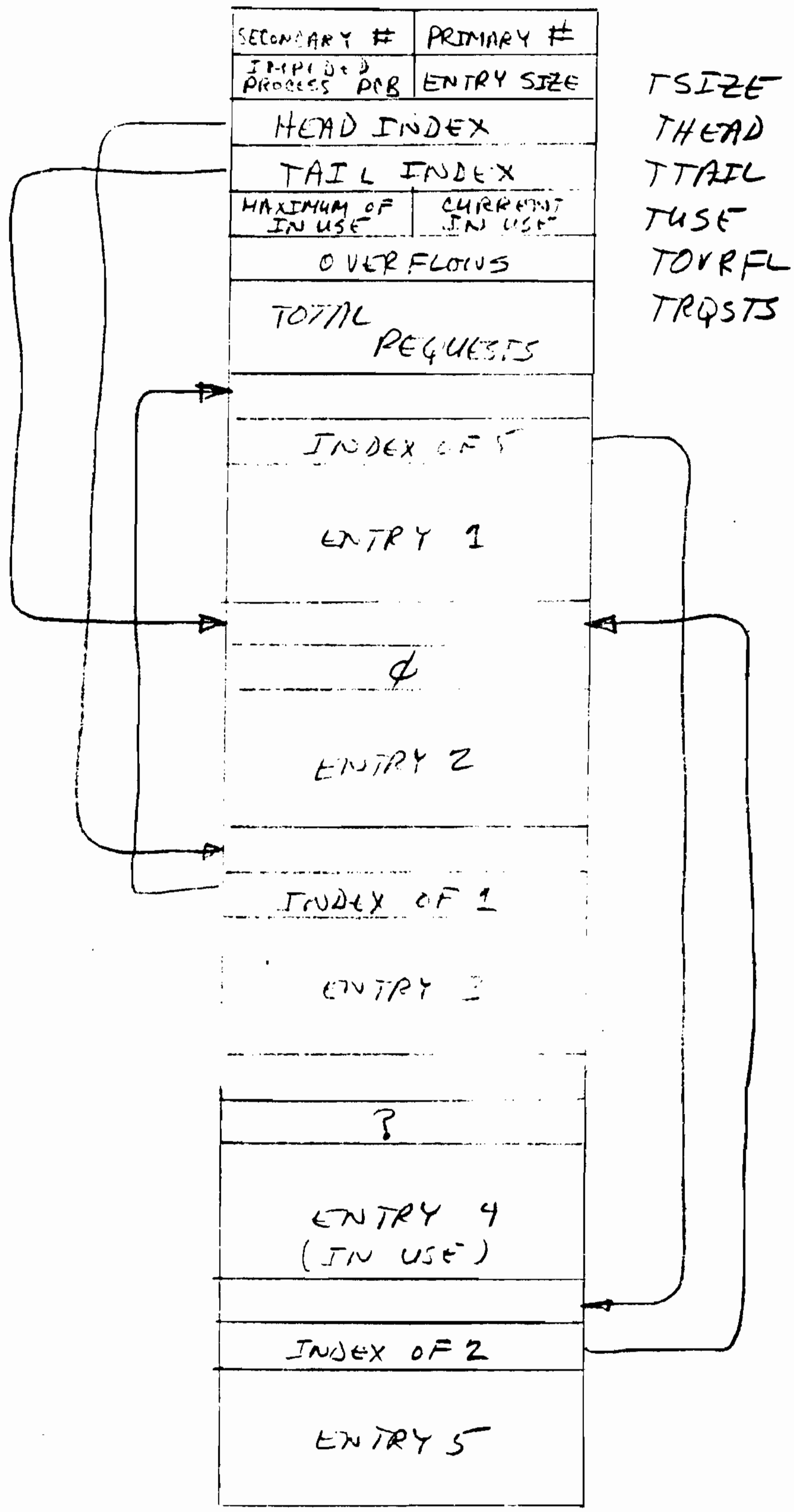
HEADER DEFINITION:

PRIMARY #	- Number of elements in the primary area
SECONDARY #	- Total # of elements in the table
SIZE #	- Size in words of each element
IMPEDED PCB	- If not zero then contains the PCB number of the first process waiting for an element in this table
HEAD INDEX	- Index of 1st free element
TAIL INDEX	- Index of last free element
MAX IN USE	- Maximum value of in use field
IN USE	- Current number not in free list
OVERFLOWS	- Number of requests made for an element when the free list was empty
TOTAL REQUESTS	- Total number of elements requested

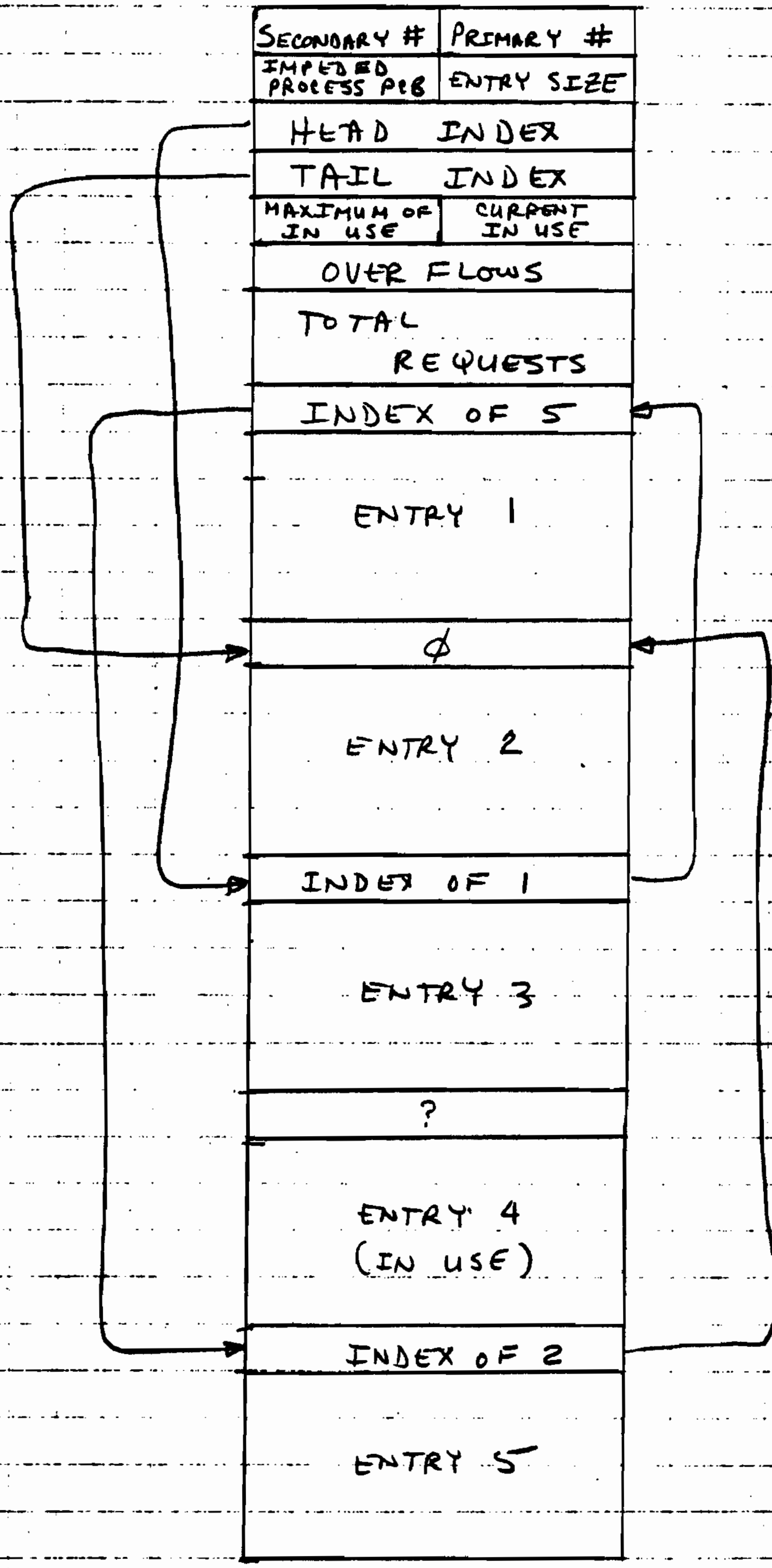


SBUF TABLE LAYOUT

PAGE 7



IOQ TABLE LAYOUT



TBUF TABLE LAYOUT

RESOURCE AND REQUEST QUEUES

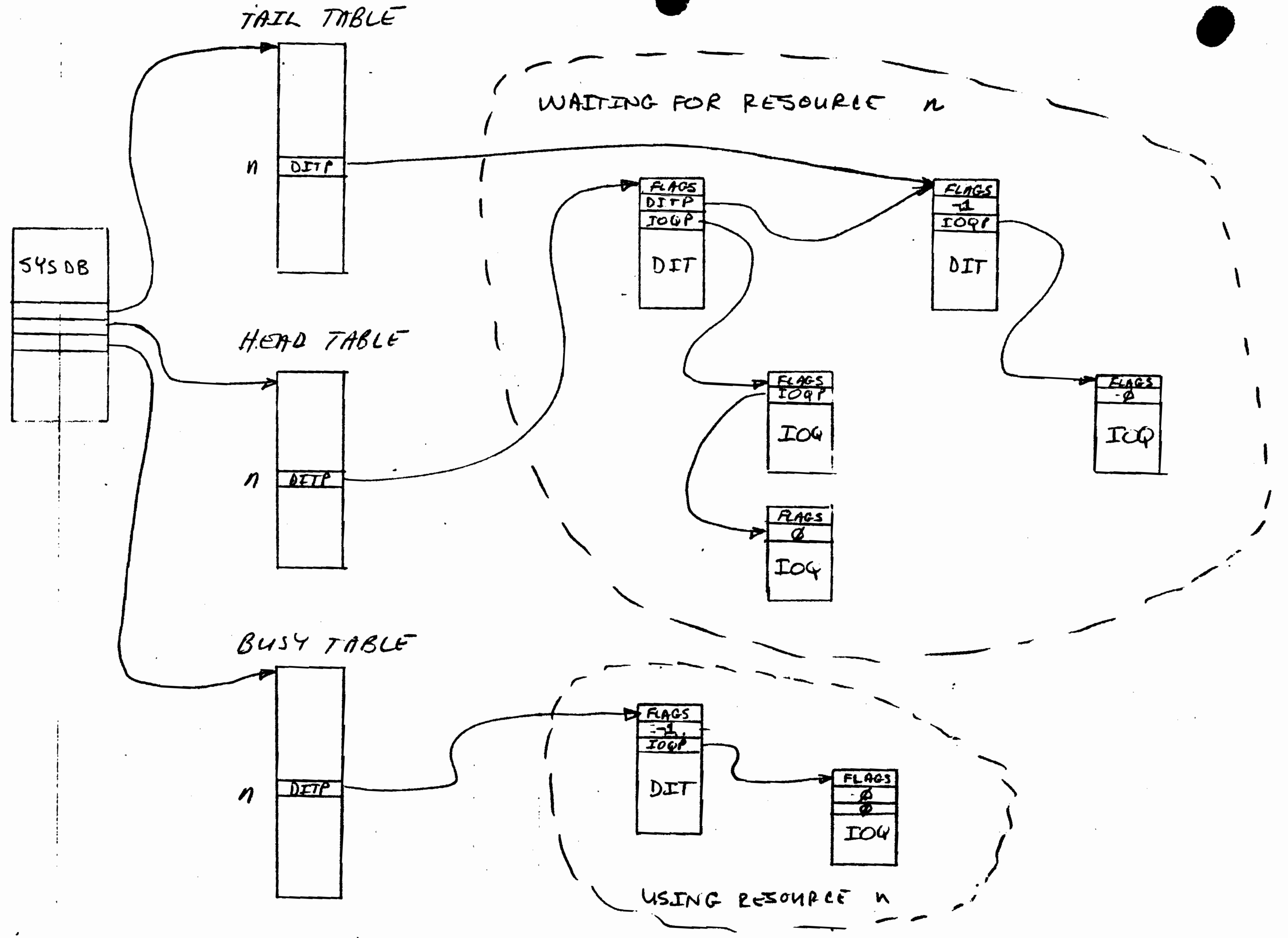
In order for a device to perform I/O it must acquire all the necessary resources. The resources are acquired and set busy according to the hierarchy shown, with acquisition beginning at the lowest level. Where a resource is always available to a device. Such as a controller which only services a single unit, the resource acquiring mechanism proceeds directly to the next necessary resource.

At the device or logical device level, the request are queued as a linked list of IOQ elements and this list attached to the DIT for the physical device. The device is effecively set busy because only the first element in the IOQ list is serviced.

Above the device level, the requests are queued as a linked list of DITs. Since more than one resource may be required by the device, a DIT could appear in more than one list. Link words would be required for each resource if the first element in a resource list is used to signify the device is busy. Instead, a busy flag is maintained for each resource. The resource request list contains only request and no references to the device currently using the resource. A zero link indicates that the device is not on a queue. A link of -1 indicates that the element is the last device on a resource queue. A non zero link indicates the device is in a queue.

The resource queuing mechanism is also used to queue request to run monitors for the type 2 drivers in their associated I/O process. An I/O process request queue is linked list of DIT's. The monitor to be run is specified through a pointer in the DIT to a Driver Linkage Table (DLT). The BUSY table entry for an I/O process request queue contains the PCB wake index to expedite waking the process. When a resource queue is empty, the head pointer is -1.

The resource queues are maintained by set of procedures which delete a request, add a request to the head or add a request to the tail of a list. These procedures reference by queue number and base base pointers in SYSDB two tables, the HEAD and TAIL tables. In addition a parameter is provided to specify the location of the link word in the DIT. More than one link word is necessary for devices which may appear on more than are request queue simultaneously. For example, a terminal may be waiting TBUF's to become available and at the same time a request may be queued to run the terminal monitor in the system I/O process.



TAIL TABLE
HEAD TABLE
BUSY TABLE
SYSDB
n
DITP
WAITING FOR RESOURCE n
FLAGS
-1
IOQP
DIT
IOQ
Ø
USING RESOURCE n

The three queue control tables are:

These queues are a linked list of DIT's and are maintained by three tables.

1. HEAD TABLE - A set of SYSDB relative pointers to the first element in the queue.

2. TAIL TABLE - A set SYSDB relative pointers to the first element in the queue.

3. BUSY TABLE -
   a) If the resource is busy then contains a SYSDB relative pointer to the DIT which has the resource. Zero if the resource is available.
   b) For I/O process request queues, these words contain the PCB index of an I/O process.

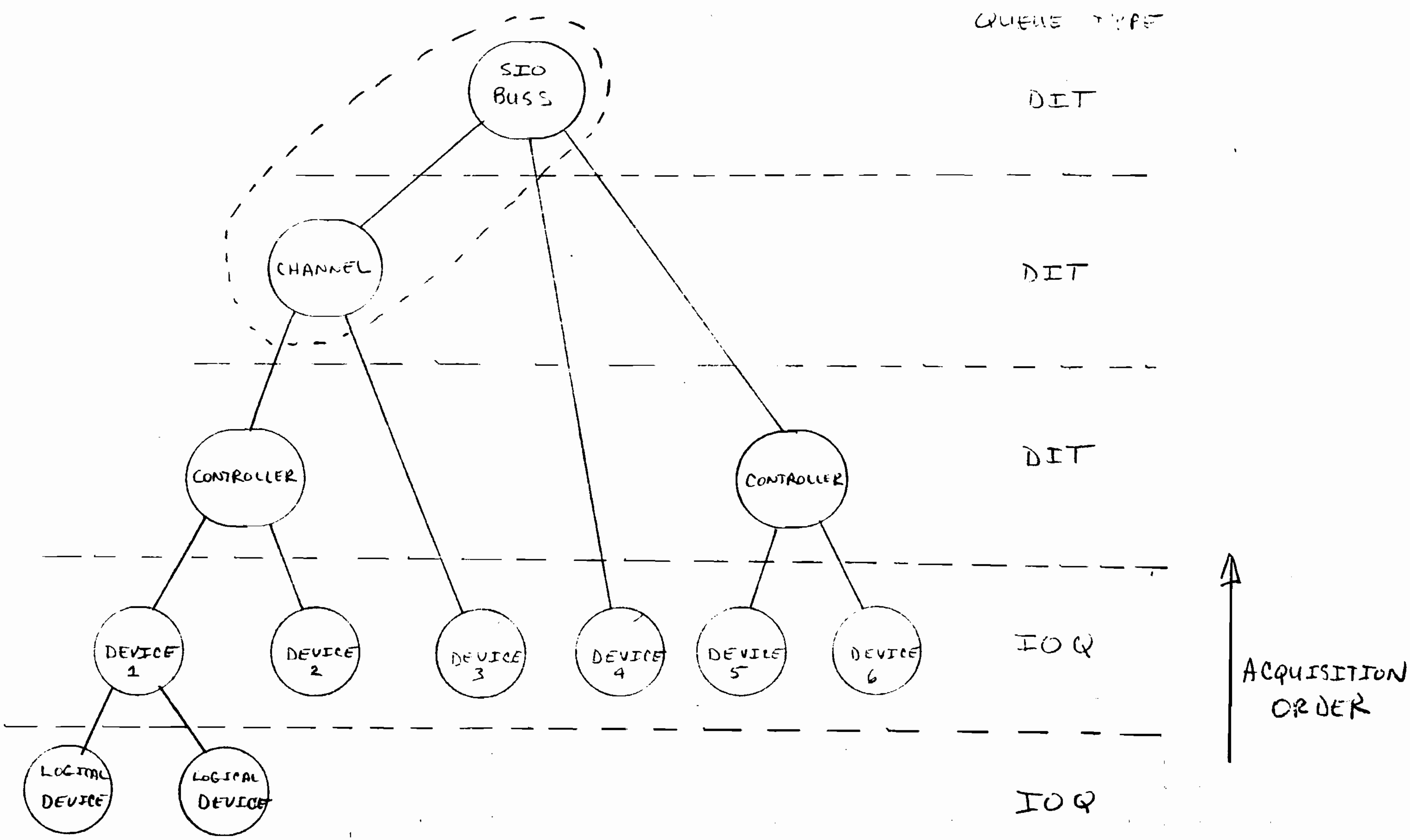
These tables are addressed by three SYSDB relative pointers to the base of each table indexed by the queue number.

QUEUE NUMBER ASSIGNMENTS

Queue Number	
0	A list of terminal devices waiting to obtain TBUF's in order to do output.
1	A list of 2640/44 terminals waiting for sufficient Asynchronous terminal Multiplexor bandwidth in order to do block mode input.
2+	I/O process service request queues. These queues are a list of devices, with type 2 drivers, waiting to be run by the associated I/O process. The requests are queued by AWAKE IO. The queue number associated with the process is passed to the process through the X register when it is first run. The queue number is contained in the left byte of the first of the DIT.
N+	Multi unit controller queues. When multi-units are on a controller, a queue number is assigned to manage the controller. The queue number is set in the right byte of word 3 of the ILT.
M+	Channel Queues. A queue number is assigned to each channel which has more than one controller connected to it. The queue number is contained in bits 1 through 6 of the first word of the ILT.

EXPLANATION OF RESOURCE HIERARCHY DIAGRAM (SEE NEXT PAGE)

Device 1	has multiple logical units on a single device, such as a two platter disc.
Device 1 and 2	are on a multiunit controller.
Device 3	is on a single unit controller. The controller is connected to a channel which services more than one controller.
Device 4	is on a single unit controller which is connected to the SIO multiplexor.
Device 5 and 6	are on a multiunit controller which is connected to the SIO multiplexor.



RESOURE ACQUISITION HIERARCHY

I/O MONITOR TYPES

There are three types of monitors supported by the System:

Type 1-

This type of monitor is executable on any stack including the ICS. They must always execute to completion. The monitor itself must be core resident but it may call non resident segments if it insures that the called segment is resident before actually executing the call. Generally SIO Device Monitors are of this type. All segments called by these drivers must be core resident.

Type 2-

This type of monitor is executable in any process. The monitor does not have to be core resident, nor do the segments it calls have to be resident. If the call to AWAKEIO specifies the process is not impedable, AWAKE IO queues a request to run the monitor in its associated I/O process and wakes that process. There may be implicit "waits" such as segment absences but this type of monitor should have no explicit waits which would impede the I/O process and thus possibly effecting the response of other devices. The Asynchronous Terminal monitor is of this type.

Type 3-

This type of monitor runs only its own process. There are no restrictions on the residency or operation of this type.

I/O DRIVERS

Drivers may consist of the following 5 components. The first item listed below is necessary. Items 2 and 3 are necessary but may be provided thru the use of GIP, the general interrupt processor and SIODM, the SIO Device Monitor. Item 4 is not necessary for type 1 monitors and may be provided by SYSIOPROC, the system I/O process for type 2 monitors. Item 5 is completely optional.

DRIVER COMPONENTS

1) Driver Program. This program file contains a linkage area specifying initiator, completor monitor and interrupt plabels. Its DB area contains information used to initialize tables (eg. DIT initial values, DIT size, SIO program area and size, etc.). See the SIO Driver Interfacing guide for a complete description of the required fields and their format. Normally the driver program file will contain the initiator and completor procedures.

2) Interrupt Procedure. This must be a core resident procedure. either in SL.PUB.SYS or in the Driver program file. There are two types of interrupt handlers:

   a) primary interrupt handler. If any logical device for a given DRT has an interrupt handler of this type, its plabel will be put in the DRT. This handler must be capable of checking for other units on this DRT which have secondary interrupt handlers in order to PCAL them to process an interrupt for that particular unit. If two drivers for different units on the same DRT specify different primary interrupt handlers, it is an error.

   b) Secondary interrupt handler. This type of handler is not expected to be capable of checking for other handlers to PCAL. If a secondary interrupt handler is specified on one or more devices on a DRT and no primary handler is specified for any device on that DRT, INITIAL will hook up a known one (GIP, for instance).

3) Monitor Procedure. This procedure which may be in SL.PUB.SYS or in the driver program file determines what action is necessary to service the controller, including freezing data, calling the initiator or completor, completing the request and starting the request.

4) I/O Process Procedure. This procedure constitutes the "outer block" of an I/O process. This procedure then is the program run when the I/O process is woken. Type one monitors do not need an I/O process since they always run on the stack of the caller of AWAKEIO.

5) Initialization Procedure. This procedure is called by PROGEN to initialize the logical device. It is called once for each logical device associated with the driver. Different units on a controller may have different drivers, therefore a controller may be initialized by more than one initialization procedure. These procedures may be in the driver program file or SL.PUB.SYS. Initial places the plabel of the initialization procedure, if any, in the 2nd word of the DIT. Progen then scans the DIT's, via the LPDT and calls the initialization procedure for any DIT with a non zero plabel. The link word is then cleared. When the initialization procedure is called, DB is set to SYSDB.

SIO DEVICE CONTROL

The control of SIO devices, even though they are simpler to program and have less nuances than terminals is complicated by the sharing of common facilities such as controllers and channels.

An I/O monitor and driver may be considered a routine to process I/O requests. The monitors perform common functions such as freezing data, handling on line interrupts, calling appropriate driver routines, completing requests, etc. The driver perform device specific functions such as generating I/O programs and checking completion status.

In the following discussion the monitor/driver routines are considered a single entity called a monitor.

When a monitor is executing, an active flag is set for the device. If a request to execute a monitor for a device which is already active occurs (i.e. from an interrupt) a request flag is set but the monitor is not executed. when a monitor finishes execution and before clearing the active flag the request flag is checked. If the request flag is set, it is cleared and the monitor is executed again.

when a monitor begins execution, it checks the controllers busy fla If it is busy, a request for the controller is queued; otherwise, t controller is set busy. The controller is set busy allowing t monitor to have unrestricted access to the SIO program area while it is executing. There is only one SIO program area per controller, since only one I/O program may be executing at a time. While the monitor is executing, the controller is busy and thus impedes other requests which share the same controller, therefore it is important that the monitor execute with a high priority. This is accomplished by setting the process structure disabling flag which allows only the current process executing the monitor to run.

when a monitor attempts to start an I/O program on a device whose controller shares a channel and the channel is busy, a request for the channel is queued. when an interrupt occurs and the device is a channel, the channel request list is checked and if any devices are waiting the interrupt processor removes that device from the queue and calls STARTIO to start the SIO program for the device. All SIO programs are required to interrupt upon completion even though further interrupts from the operation initiated by the SIO program are expected! Interrupts at the completion of the SIO program allows the controllers and channels to be released and assigned to other devices so that overlapping I/O operations may take place.

No explicit provisions for eliminating data overruns have been incorporated into the I/O system. It is envisioned that devices wou be assigned to pseudo-channels to limit the concurrent activity produce an acceptable level of overruns.

Only one I/O monitor procedure is required for the disc and other SIO devices. A flag indicating the device is a disc allows for special processing of disc requests.

Before any new disc requests are selected for initiation, the disc monitor examines a memory management request list. If this is not empty an I/O request for the disc is generated which preempts all requests for this disc. These memory management requests are considered special high priority requests and they will be initiated immediately if no requests are pending or at the completion of any current request. When one of these special requests is completed, a memory management table is updated and the memory management process is awakened.

The following applies to terminal I/O and control through calls to ATTACHIO. All terminal I/O is done into intermediary buffers. Writes are considered completed when the data is moved into the terminal buffers. Reads are completed when data is moved into the users data area.

Whenever possible, the terminal driver is executed in a user process. The process chosen is usually associated with the first blocked request. If no BLOCKED requests are queued, the driver is run in the System I/O process. Normally this results in no process waking between "prompt", read requests and eliminates the annoying pause between the issue of a prompt and the initiation of the read. Also, reads which are queued when a write completion occurs are initiated on the ICS.

The terminal driver uses the fact that ATTACHIO recalls the driver if an awake occurs on a blocked request and it is not completed. In the case where the driver must wait for addition resources, such as TUFs, a driver state is set and the driver exits back to ATTACHIO where the BLOCKED request is waited. When the resource becomes available, the process is awoken and the driver recalled by ATTACHIO to continue or complete the request. Terminal read completions are handled in similar manner so that they are completed in the callers process.

When a BREAK is detected and accepted, the current read or write operation is stopped and a CR/LF is output if the carriage is not on a new line. The data input preceeding a BREAK during a read is saved. When a read is started and the device is not in Break Mode, the read is started with this data already "read". BREAK also causes a Flush flag to be set and as long as this flag is set, all writes are returned as if successfully completed although no I/O is done and all reads are returned with a status of %173. The Flush flag is reset by a clear flush and write request, function code 25.

When a Control Y is detected and allowed, the current request is terminated and marked successfully completed. All non-preemptive requests currently queued are flushed, that is, they are returned with a successful completion indication even though no I/O took place. Flushing is terminated whenever the queue is empty. If the current request is a read, the count when control Y is detected is returned. In addition, the Subsystem Break bit is set in the PCB.

A hard or soft preemption may be specified with terminals requests. Write requests which have started (i.e. data move to terminal buffers) are allowed to complete independent of the preemption type.

Preemption of a given level are queued behind any requests at the same level. Hard preemptions preempt all writes which have not been state and all reads. A read which has been preempted is restarted from the beginning.

Soft preemptions preempt normal writes which have not been started and reads on which no data has been input. If data has been input, soft preemption is held off until the read is completed.

Preemption is specified by setting FLAG.(7:1) for hard preemption and FLAGS.(8:1) for soft preemption.

Several additional preemption conditions are also possible. The order of handing preemptive requests is given below from the least preemptive to the most preemptive.

Preempt Level	Meaning
0	Normal request
1	Request queued with device in Break Mode
2	Request queued with device in Console Mode
3	Soft preemptive request
4	Hard preemptive request

I/O SYSTEM BUFFERING

The I/O system will provide buffering only for the terminals. Up to 270 characters per terminal will be buffered into a core resident terminal buffer area, although the maximum byte count per request is not limited to 270. Terminal writes are considered completed, from the users view point, when the data is moved to the terminal buffers. If sufficient buffer space is not available, the user will be suspended at the point where buffering is performed. When the current write nears completion, so that sufficient buffer space is available, the user will be awakened, thus allowing the user to present a continous string of characters for output. This continous buffering will only occur where no other requests are queued with an IOQ element. Requests will be queued with a separate IOQ element whenever a mode change has occurred or if the request is preemptive.

All devices capable of transmitting more than 128 words in a request will have the capability to transfer from a linked list of System Buffers. The SIO devices will only have the capability to transfer a maximum of 1024 words from or to a set of 8 disjoint 128 System Buffers.

Zero Word Count Requests

The following action will take place when a zero word count request is processed.

DEVICE	READ	WRITE
Terminal	XON/CR/LF	XOFF/CR/LF
Mag Tape	Forward Space Record	NOP
Card Reader	Pick a Card	--
Reader/Punch	Pick a Card	Pick a Card
DISC	NOP	NOP
Paper Tape Reader	Skip a Record	--
Paper Tape Punch	--	XOFF/CR/LF
Plotter	NOP	NOP
Printer	--	Print a Blank Line

FILE OPEN, CLOSE AND DEVICE CLOSE

These control functions are invoked by the following function codes:

2 - FILE OPEN
3 - FILE CLOSE
4 - DEVICE CLOSE

DEVICE	FUNCTION		
	FILE OPEN	FILE CLOSE	DEVICE CLOSE
Disc	NOP	NOP	NOP
Magnetic Tape	NOP	NOP	Reset EOF flags Rewind and Unload
Line Printer	Page Eject	Page Eject	Page Eject
Card Reader/Punch	If output, Pick	If input, Stack	Reset EOF flags
Paper Tape Punch	Punch Leader	Punch Trailer	NOP
Paper Tape Reader	NOP	NOP	Reset EOF flags
Plotter	NOP	NOP	NOP
Terminal	CR/LF	CR/LF	Reset EOF flags Initialize to speed sense

END OF FILE SPECIFICATIONS

End of File (EOF) may be caused by data records with one of the attributes listed in the table below. The I/O System maintains in the LPDT a 3 bit field which indicates the EOF condition caused by the last record read.

EOF Code	Meaning
0	No EOF
1	Hardware or :EOF: detected
2	:DATA or DATA detected
3	:EOD detected
4	HELLO detected
5	BYE detected
6	:JOB or JOB detected
7	:EOJ detected

The ":EOF:" record is used on devices which do not have hardware EOF facilities. Magnetic tape and disc devices do not recognize the :EOF: as an EOF indicator.

Binary data is delimited by the same indicators as ASCII data. A binary pattern which results in an apparent EOF record will cause an EOF indication on all devices except terminals.

In order that a Job Control error in a preceeding Job does not effect succeeding JOB's, the I/O System will "backspace" certain types of EOF. If a Job, Data Hello or Bye is detected the record is saved by the driver. When the requested length is less than the device-dependant critical length (usually 128 words or the maximum physical record size which ever is less) the driver reads the critical length into an auxillary buffer so that no data is lost. If no EOF is detected, the count requested is moved into the users buffer. If a backspace EOF is detected, the data is saved in the auxilary buffer and passed to the next read request which would not result in an EOF. The terminal system saves data as noted above with the exception that all reads are initiated with the count specified.

Except for a tape mark, which always results in an EOF return, the EOF condition is only returned if the EOF specification in the read request enables the class of EOF detected. The EOF specification is passed in QPAR1.(31:3) of the request.

Reads with an EOF specification of zero are always initiated. Before any physical read is initiated, the driver passes back any saved data as if a physical read had taken place.

The EOF specifications are:

0 - Reset EOF condition and read
1 - Check for hardware EOF including :EOF:
2 - Check for :DATA, :EOD, :EOJ
3 - Check for DATA, HELLO, BYE, JOB

4 - Check for :DATA, :JOB, :EOJ
5 - Read but do not reset EOF, pass back saved data or check read data for an EOF

Before reads with an EOF specification other than 0 or 5 are initiated , the previous EOF condition is checked against the EOF specification. If the previous EOF condition was 1 (hardware) or the EOF specification greater than 1 and the previous EOF greater than 1, then the previous EOF condition is passed back to the user and no read takes place.

When a read is initiated, the previous EOF condition is cleared. The data is checked upon proper completion of the read to see if an EOF of the type specified occurred. See EOFCHECK procedure description for particular EOF conditions versus EOF read specifications.

If an EOF of the type specified occurred, an EOF indication is returned to the user and the type of EOF which occurred is saved in the LPDT.

DRIVER INPUT AND OUTPUT SPECIFICATIONS

The input and output specifications of all the drivers has been made as consistent as possible to make calls to ATTACHIO reasonably device independent.

Six function codes have been reserved to have the following meaning:

Function Code	Meaning
0	Read
1	Write
2	File Open
3	File Close
4	Device Close
28	General Device Function

The count parameter is negative for byte counts and positive for word counts.

Parameter One and Parameter Two have the device dependent meanings diagramed in the following table.

DEVICE	PARAMETER 1	PARAMETER 2

READ

Device	Parameter 1			Parameter 2					
DISC	SECTOR ADDRESS (HI ORDER)			SECTOR ADDRESS (LO ORDER)					
MAG TAPE			EOF						
CARD READER			EOF						
PAPER TAPE			EOF	READ STOP CHAR				BINARY	
PUNCH / READER			EOF						
TERMINAL	NO CRLF		EOF	READ STOP CHAR	FUNC KEYS	PASS CR/LF	USER MODE	BINARY	

WRITE

Device	Parameter 1	Parameter 2			
DISC	SECTOR ADDRESS (HI ORDER)	SECTOR ADDRESS (LO ORDER)			
MAG TAPE			EOT		
LINE PRINTER	VERTICAL FORMAT SPECIFICATION				
PAPER TAPE	END OF RECORD SPECIFICATION		BINARY		
PUNCH /READER			BINARY		
TERMINAL	CARRIAGE CONTROL SPECIFICATION		BINARY		PRE SPACE
PLOTTER					

DEVICE OPERATION	FUNC	P1	P2
DISK			
READ	0	DISK	ADDRESS
WRITE	1	DISK	ADDRESS
FILE OPEN	2	---	---
FILE CLOSE	3	---	---
DEVICE CLOSE	4	---	---
FILL WITH ZEROS	5	DISK	ADDRESS
FILL WITH BLANKS	6	DISK	ADDRESS
MAG TAPE			
READ	0	EOF	---
WRITE	1	---	EOT FLAG
FILE OPEN	2	---	---
FILE CLOSE	3	---	---
DEVICE CLOSE	4	---	---
REWIND	5	---	---
WRITE TAPE MARK	6	---	---
FORWARD SPACE FILE	7	---	---
BACK SPACE FILE	8	---	---
REWIND & UNLOAD	9	---	---
GAP	10	---	---
FORWARD SPACE	11	---	---
BACK SPACE RECORD	12	---	---
CARD READER/PUNCH			
READ	0	EOF	MODE
WRITE	1	---	MODE
FILE OPEN	2	---	---
FILE CLOSE	3	---	---
DEVICE CLOSE	4	---	---
CONTROL	5	---	FUNCTION
PRINTER			
WRITE	1	SPACE CONTROL	MODE
FILE OPEN	2	---	---
FILE CLOSE	3	---	---
DEVICE CLOSE	4	---	---
PAPER TAPE READER			
READ	0	EOF	MODE, STOP CHARACTER
FILE OPEN	2	---	---
FILE CLOSE	3	---	---
DEVICE CLOSE	4	---	---

DEVICE OPERATION	FUNC	P1	P2
PAPER TAPE PUNCH			
WRITE	1	SPACE CONTROL	MODE
FILE OPEN	2	---	---
FILE CLOSE	3	---	---
DEVICE CLOSE	4	---	---
SWITCH TAPE	5	---	---
PLOTTER			
WRITE	1	---	---
FILE OPEN	2	---	---
FILE CLOSE	3	---	---
DEVICE CLOSE	4	---	---

TERMINAL FUNCTINS AND RETURNS

OPERATION	FUNC	P1	P2	XLOG
READ	0	EOF FLAGS	STOPCHAR	COUNT
WRITE	1	SPACE CONTROL	PRESPACE	COUNT
FILE OPEN	2	--	--	--
FILE CLOSE	3	--	--	--
DEVICE CLOSE	4	--	--	--
SET TIMEOUT	5	SECONDS	--	--
SET INSPEED	6	CHAR/SEC	--	OLDSPEED
SET OUTSPEED	7	CHAR/SEC	--	OLDSPEED
ECHO ON	8	--	--	OLD ECHO
ECHO OFF	9	--	--	OLD ECHO
DISABLE BRK	10	--	--	--
ENABLE BRK	11	--	--	--
DISABLE ESC	12	--	--	--
ENABLE ESC	13	--	--	--
DISABLE TAPEMODE	14	--	--	--
ENABLE TAPE MODE	15	--	--	--
DISABLE TIMER	16	--	--	--
ENABLE TIMER	17	--	--	--
READ TIMER	18	--	--	--
DISABLE PARITY	19	--	--	--
ENABLE PARITY	20	--	--	--
LOGGED ON	21	TYPE	--	--
SET TERM TYPE	23	TERM TYPE	--	--
ALLOCATE TERMINAL	24	TERM TYPE	SPEED	--
CLR FLUSH & WRITE	25	SPACE CONTROL	PRESPACE	COUNT
CNTRL X ECHO ON	26	--	--	--
CNTRL X ECHO OFF	27	--	--	--
INVALID REQUEST	28	--	--	--
PTAPE READ	29	DISC ADDRESS (DOUBLE)		--
SET BREAK MODE	30	ON/OFF	--	--
SET CONSOLE MODE	31	ON/OFF	--	--
SET PARITY	32	PARITY	--	--
ALLOCATE TERMINAL	33	TERM TYPE	--	--
SET TERM TYPE	34	TERM TYPE	--	--
GET TERM TYPE	35	--	--	--
GET OUTPUT SPEED	36	--	--	OUTPUT SPEED
SET STOP CHARS	37	STOP CHARS	--	--

Interrupt Command Interpreter Overview

1A

Terminal Interrupt
↓
TIP executes on ICS
handle interrupt
[redacted]
IXIT
↓
DISPATCH

Launch SYSIOPROC

SYSIOPROC is changed to an IO process by Richard

1A1

↓
SYSIOPROC
PCAL monitor (IOTERMØ)

no more TERMIOPM

↓
IOTERMØ
process unexpected
interrupt
wake up DEVREC
process suspends
↓
DISPATCH

1B

SIO device Interrupt

C.R. or Job accepting may Take

↓
GIP executes on ICS
handle interrupt
[redacted]

SIODM is a procedure

↓
SIODM
process unexpected
interrupt
wake up DEVREC
IXIT
↓
DISPATCH

↓

2

DEVREC
build job tables
wake up UCOP
process suspends
↓
DISPATCH
↓

3

UCOP
build C.I. stack
set up system tables
set C.I. alive
process suspends

C.I. is not created (precreated)

↓
DISPATCH
↓
C.I. executes

4

1A

Interrupt on job/data accepting terminal (non-SIO device)

TIP
  if unanticipated interrupt, check device for initialized and STATE = Ø
  speed sense the device
  set a request for "online service"

  AWAKETERMINAL
     check for unanticipated interrupt

     AWAKEIO
        check type of I/O monitor
        queue a request for type 2 monitor (non-SIO)
        awake type 2 if caller is not impedable

        AWAKE
           set PCB active bit for SYSIOPROC process
           place PCB in ready list
           notify Dispatcher if SYSIOPROC has a higher priority than CPCB

     AWAKEIO returns to AWAKETERMINAL

  AWAKETERMINAL returns to TIP

TIP exits ICS (IXIT)

Type

1 SIO
Can run on - ICS — PCAL to Monitor
- USER
- sys I/O Process

2 Terminal type - User
- sys I/o process

Dispatched monitor

3 <none> - only on I/O process

for Terminal — contd.

1A1

INDENTATION represents PCALS

```
SYSIOPROC
  dequeue request for I/O monitor

  AWAKEIO
     check type of I/O monitor
     PCAL type 2 if caller is impedable — Can not PCAL on ICS hence the IXIT to get out of ICS hence now it is impedable

     IOTERMØ
        test for service requests
        if "online request" device must be unowned and data accepting
        set DEVREC service request in LPDT
        set "logging on" state in the DIT
        start logon timeout
        set default termtype
        initialize the device channel

        AWAKE
           [redacted]
           place PCB in ready list
           notify Dispatcher if DEVREC is higher priority than CPCB

        check for more service requests
        check for more user requests

  AWAKEIO returns to SYSIOPROC

SYSIOPROC
  attempt to dequeue more requests and call the monitor
  if no monitor requests remain call WAIT to suspend
  when reactivated, loop back and start operations again
```

13

Interrupt on job/data accepting SIO device

GIP
  check for unanticipated interrupt: new STATE, no SIO prog. in progress
  set STATE = 6 in DIT (device recognition state)

  AWAKEIO
    check type of I/O monitor
    PCAL type 1 (SIO device monitor)

    SIODM run on ICS
      process unanticipated interrupt STATE
      check for no I/O pending and system up
      check for job/data accepting device
      device recognition state should be unowned

      AWAKE
        set PCB active bit for DEVREC
        place PCB in the ready list
        notify Dispatcher if DEVREC is higher priority than CPCB

      reset STATE to Ø in DIT
      check for more monitor requests and process them
      if no requests remain, return

  AWAKEIO returns to GIP

GIP exits ICS (IXIT)

2

DEVREC
check for job accepting & available device
get a system buffer
output a LF to a session terminal
output a : prompt to a session terminal
read a command (may have continuation images)
combine continuation images & upshift lower case characters
if command is not JOB, DATA, or HELLO, print error
is device configuration compatible with command - no:print error

STARTDEVICE ← Big procedure
parse the JOB, DATA, or HELLO command
XDD ≡ VDD
error check the command parameters
build JMAT and IDD entries in DEVREC's stack
verify account and user passwords
if DATA command, post IDD entry in XDD
turn off timeout for :DATA and return
[redacted]
check executing priority (PRI) - it must be lower than acct. & user MAXPRIORITY
if no logon group is specified, set home group into JMAT
verify group password if command specified a group
if HIPRI was used, acct. and user must have SM or OP capability
if OUTCLASS was used, acct. and user must have ND capability
post JMAT entry to system table from DEVREC's stack
post IDD entry for $STDIN to XDD from DEVREC's stack
if device is a terminal and command is :JOB turn off timeout

SCHEDULEJOB
link JMAT into scheduling queue
change state of entry from Introduced to Wait

AWAKE
set PCB active bit for UCOP
place PCB in the ready list
notify Dispatcher if UCOP is higher priority than CPCB

return system buffer
If no further work can be done, DEVREC calls WAIT to suspend.
Reactivation causes DEVREC to loop back to its beginning.

UCOP 3

process pending requests to delete processes or abort jobs

GETJOB

examine entries in JMAT scheduling queue
attempt to launch waiting JOBs or SESSIONs

LAUNCHJOB

make certain that the new logon will not exceed maximum number of concurrent JOB/SESSION limits except for HIPRI case
check that INPRI is greater than JOBFENCE
get a PCB entry for the new process
allocate a DST entry and stack
get an extra data segment for the JIT
get an extra data segment for the JDT
get an entry in the JPCNT table

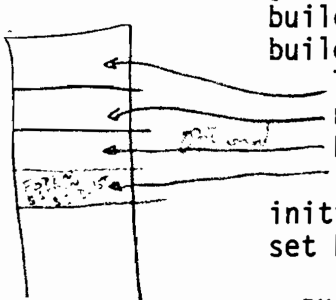
ALLOCATE

allocate the standard input & list devices

place input & list device information in the JMAT
set up the new process PXGLOB area in UCOP's stack

PROCREATE

get a PCB entry if none is provided
build PCBX in new process stack
build the required stack markers in the new stack:



TERMINATE marker (delete process)
normal initial launch marker (exit to C.I.)
DEBUG launch marker (if required)
INITIATE marker (opens $STDIN and $STDLIST for <u>user</u> process)

initialize the new process PCB entry
set PCB process Alive bit.

move PXGLOB from UCOP stack to new process stack

AWAKE

set PCB Active bit for new Command Interpreter Process
place PCB in ready list
notify Dispatcher if new C.I. is higher priority than CPCB

GETJOB returns to UCOP

UCOP

continues to call GETJOB until a false return value indicates that no waiting job or session can be launched. UCOP then checks again to make certain that all available tasks have been completed and then calls WAIT to suspend.

4

Command Interpreter

    move the JMAT entry into the C.I. stack
    if CPU time limit exists, store it in JCUT

    DIRECLOGON
        copy account, user, and group directory entries into C.I. stack
        update number of accessors in directory account group index
        update number of accessors in directory group file index
        increment logon count in directory user entry

    check user capability does not exceed account capability
    if it does limit user to account capability
    check requested priority (PRI) does not exceed account maximum
    if it does limit user to account MAXPRI
    request any required passwords omitted from HELLO command — a session
    verify the passwords supplied against directory entries in stack
    build JIT on C.I. stack and initialize it
    move JIT to its extra data segment
    change JMAT state to executing
    build JDT on C.I. stack and initialize it
    move JDT to its extra data segment
    turn off logon timeout for HELLO command

    FJOPEN
        open $STDIN and $STDLIST files

    output forms message to operator (if forms message used)
    wait until forms are mounted on joblist device
    if job in/list are not duplicative:
        top of form the list device
        print :JOB or :HELLO line
        print PRI, INPRI, TIME information
    print job or session number
    print date and time
    print MPE version, update, and fix level
    check for bad password supplied -illegitimate access
    make certain that account and group time limits have not been exceeded

    LOG
        check if logging is enabled for record type (job initiation)
        format the log record and place it in a buffer
        activate LOG process if the buffer is full

    print messages if capability or max priority were exceeded
    print WELCOME message if one exists

    [all initialization is done - C.I. begins main loop]

prompt + [redacted]
    identify the command
    command allowed under present circumstances (break etc.)
    PCAL command executor procedure
    continue loop

Command Interpreter CXRUN executor [:RUN command]

[redacted]

error check number of parameters and file name format
extract information from the parameters

CREATE

get a PCB entry for the new process
set priority class (if omitted) to that of CPCB
check validity of priority class
set values for MAXSTACK, STACKSIZE, DLSIZE from parameters

LOAD

truncate entry name (if any) to 15 characters
check for illegal library search (FLAGS parameter)
FOPEN the program file - execute access
FLOCK the program file
FGETINFO on program file
verify file code is type program
read program file record Ø
verify program file has only 1 extent
make certain program can run with access used (IA or BA)
locate PB address and STT number of entry point
use defaults for MAXDATA, STACKSIZE, DLSIZE (from PREP) if not already set
check DLSIZE not greater than 32K words
check STACKSIZE not less than 512 words

LOADPROGRAM

build SHARER entry for program in LST
check program not already loaded in opposite mode (PRIV/NON)
(program can't execute priv & no priv. at the same time)
check program not already loaded in same mode
(if loaded - share allocated code)
check program not LOADING in opposite mode
(error - can't share code if modes are different)
check program not LOADING in same mode
(if loading build waiting entry in LST & suspend)
build LOADING entry for program in LST

LOADER

get LOAD process SIR
set up Loader Communication Table in stack
move LCT to SYSGLOB

AWAKE

set PCB active bit for LOAD
place PCB in ready list
WAIT - suspend

[The LOAD Process executes] A

LOADER (continued)

check LCT for any returned load error
if any exists, save it for later ABORT message
Check LCT for list file flag (LMAP or UNDEF EXTS)
if listing is required:
FOPENDA - address of Load Process list File (LCT)
* FOPEN - LOADLIST - $STDLIST
copy disc to list device
FCLOSE both files
release LOAD process SIR

LOADPROGRAM (continued)

if any LOADER error, find & delete LST LOADING entry

LOAD (continued)

place new CST in entry point plabel
initialize PBX and WSP pointers in PCB using PROGRAM entry of LST
if program being traced, reset entry point to TRACE
check stack size (PCBX + DL + GLOB + STACK + OVFL < 32k words)
check MAXDATA not greater than system maximum

GETSTACK

include 1536 more words in stack V.M. (for PCBX & ovfl. abort)
allocate a DST entry
allocate virtual memory space for stack
initialize DST entry - mark segment absent

* change here for LOADLIST-SIR problem.

LOAD (continued)

zero PCBX and DL area of the stack

read program file global initialization record into the stack

set up DL reserved area (DB negative)

FCLOSE the program file

CREATE (continued)

if original parameters have been changed by LOAD, set up CCG for return

PROCREATE

get a PCB entry if none is provided

build PCBX in new process stack

build the required stack markers in the new stack:

TERMINATE marker (delete process)

normal initial launch marker (exit to process/TRACE)

DEBUG launch marker (if required)

INITIATE marker (opens $STDIN and $STDLIST)

initialize the new process PCB entry

set PCB process Alive bit

CXRUN (continued)

check for error from CREATE

warn user if original parameters were changed

AWAKE

set PCB active bit for new process

place PCB in the ready list

WAIT - suspend

[The new process executes]

FINISH

any command flush operation requested (no)

output "END OF PROGRAM"

CXRUN returns to the outer block of the Command Interpreter.

A

LOAD PROCESS (simplified)

FOPEN a disc file for LMAP listings "LOADLIST" (old file)
if LOADLIST does not exist, build it - then open old } done once
move Loader Communication Table (LCT) from SYSGLOB to stack
rewind LOADLIST file
LCT specifies program load (vs. procedure load) & dynamic proc load (LOADPROC)

SATISFYPROG

open and lock the program file
read first 2 sectors of program file info. and store in DL
read External List record into DL (unsatisfied external table)

SATISFY

open each SL to be searched (one at a time)
search the SL for entry points which match externals
list externals and entry points (if LMAP requested)
transform unsatisfied to satisfied external table entries
repeat the above until all requested SLs have been processed
list the remaining unsatisfied externals

LOADEXTERNALS

determine which segments are already allocated
allocate CST entries for SL segments
allocate a CST block for program segments
assign CST numbers to referenced SL segments
load each referenced segment (SL by SL)

LOADSEGMENT

read STT for the segment from the SL file
examine all STT entries
fix external type STT entries with new CST numbers
write updated STT back to the segment in the SL file
build 4 word CST entry
place the entry in the CST

cont'd.

A

LOAD Process (continued)

LOADPROGRAM

assign CST numbers to program segments
build a table to map previous CST numbers into new CSTs
save the new CST numbers (last assigned) in remapping table
list physical CST numbers in logical segment order (for LMAP)
write CST remapping table to program file record Ø
using CST remap, load each program segment

LOADSEGMENT

read STT for the segment from the program file
examine all STT entries
fix external type STT entries with new CST numbers
write updated STT back to the program file segment
build 4 word CST entry
place entry in XCST

set loaded bit in program file label

UPDATESEGTAB

allocate a working set table entry (WSTAB)
build a PROGRAM entry in the LST (program loaded)
build SL entries in LST where required
increment "number of processes sharing" (PROGRAM entry)

find all LST WAITING entries (waiting for this Load)
change WAITING to LOADED entries in LST
return any load error to waiting process in LOADED entries
unimpede PCBs of waiting processes
find and delete the LOADING entry in LST
close program file, group, and public SLs as required
set any error and address of LOADLIST file into LCT
write eof on LOADLIST disc file

con'td

(A)

LOAD Process (contd.)

AWAKE

set PCB active bit for load originating process

place PCB in ready list

WAIT - suspend

when reawakened, LOAD resets any errors & restarts assuming LOADLIST is open.

Termination of the executing process (son of main)

TERMINATE

turn off traps

turn off live bit in PCB

reset Q and S to initial Q (1 word before TERMINATE marker)

EXPIRE

set STOP bit in PCB of each son of the terminating process

REMITENTRY - remove any Reply Information Table entry

REMOVESTOP - clear any DEBUG breakpoints

if a data base is open, do IMAGE clean up with DBPROCTERM

set C.I. working set pointer into CPCB

CLEARWS - clear working set table entries - place on overlay select queue

UNLOAD

find and delete SHARER entry in LST

find PROGRAM entry in Loader Segment Table

decrement reference counts for program file segments

unload segments with Ø reference counts

delete PROGRAM entry if all segments are unloaded

unload any dynamically loaded procedures

if Logging is enabled, Log process termination

FPROCTERM

scan AFT for open CS lines

CCLOSE - close all communications lines

scan AFT for open files

FCLOSE (disp: no change, acc: unrestricted) all open files

ABORTDSEG - return all process extra data segments

ABORTMAIL - clean up any pending mail transfers

ABORTRIN - unlocks all local/global rins which the process had locked

EXPIRE (continued)

check all existing sons of terminating process

if son's PCB is ALIVE:

set SOFT KILL bit in son's PCB

ABORTPROCIO - abort all of son's pending I/O requests

if DEAD bit set in son's PCB, call WAIT (mourning)

BURRYPROC

remove son's PCB from scheduling queue

adjust father-son and brother linkages

if FAC is set in PCB, AWAKE father process

CLEARWS - clear WS table entries place on overlay select queue

place any active extra data segment on overlay select queue

release the son process stack

return the son process PCB

if son's PCB is not ALIVE:

remove the STOP bit in son's PCB

set FAC in son's PCB

WAIT for son to terminate

continue to eliminate sons until none remain (repeat above)

deallocate all local RINs allocated to the job

get process CPU time and execution time remaining from PCBX

compute: process CPU time + job CPU time + system clock

save cumulative time in JIT (-1D if greater than 23 days)

update job CPU time in JCUT

disable control Y (if session and enabled) reset PIN in LDT

if pseudo-interrupt mode in CPCB is SOFT KILL:

set DEAD bit in CPCB

AWAKE mourning father and suspend self

WAIT if AWAKE fails to suspend

if pseudo-interrupt mode in CPCB is not SOFT KILL:

REQUCOP

make a process termination entry in UCOP Request List

AWAKE - UCOP process

WAIT - UCOP wait set

At this point, UCOP runs to finish killing the son of main process.

UCOP - son of main process termination operation

get next entry in UCOP request list (process termination request)

BURRYPROC

remove PCB from scheduling queue
adjust father-son linkages
if FAC is set in PCB, AWAKE father process (C.I.)
CLEARWS - clear WS tab.e entries - place on overlay select queue
place any active extra data segment on overlay select queue
release son of main process stack
return son of main PCB entry

loop back and continue to process request queue entries
if the request queue is empty and no jobs can be started - WAIT

[At this point the process executed by the RUN command has terminated and control passes back to the Command Interpreter.]

Command Interpreter termination [BYE/EOJ command]

CXJOB

    if EOJ (CXEOJ entry) set TERMINATE parameter to code 6
    if BYE (CXBYE entry) set TERMINATE parameter to code 5
    store TERMINATE parameter in initial Q location (below TERMINATE marker)

    TERMINATE
        turn off traps
        turn off live bit in PCB
        reset Q and S to initial Q (1 word before TERMINATE marker)
        EXPIRE

            REMITENTRY - remove any Reply Information Table entry
            FUNBREAK - abort broken terminal read if any
            set C.I. working set pointer into CPCB
            CLEARWS - clear C.I. working set entries - place on overlay select queue

            UNLOAD

                find and delete SHARER entry in LST
                find PROGRAM entry in Loader Symbol Table (for C.I.)
                decrement reference counts for program file segments
                unload segments with Ø reference counts
                delete PROGRAM entry if all segments are unloaded
                if Logging is enabled, Log process (C.I.) termination

            get process CPU time and execution time remaining from PCBX
            compute: process CPU time + job CPU time + system clock
            save cumulative time in JIT (-1D if greater than 23 days)
            update job CPU time in JCUT
            if job input device is a terminal:
                reestablish normal terminal state if in break
                reset control Y bit in LPDT
                reset break flush (throw away broken read input)
            if C.I. was aborted - PCB hard kill or soft kill bits set
                format JOB/SESSION ABORT/TIMEOUT message
                PRINT abort message

EXPIRE (continued)

CLEANUPJOB

check TERMINATE parameter set by C.I. for origin of request
use PXGLOB to locate the JMAT
set JMAT entry to terminating
copy the JMAT entry to the stack
CLEANUPFILES - delete job temp. files and return disc space
ABORTDSEG - release main process extra data segments
get accounting information from the JIT
PRINT "CPU (SEC)=" cputime
get final time stamp (CALENDAR and CLOCK)
compute elapsed time in minutes
PRINT CONNECT/ELAPSED (MIN) message for session/job
PRINT DATE'LINE message - date and time
output END OF SESSION or END OF JOB

DIRECLOGOFF

copy account, user, and group directory entries into stack
decrement number of accessors in account group index
decrement number of accessors in group file index
decrement logon count in directory user entry
update CPU and connect times in account and group entries

FJCLOSE

close $STDLIST AND $STDIN files
release JCUT entry if any exists
release JPCNT entry
release JMAT entry

REQUCOP

make a process termination entry in UCOP Request List
AWAKE - UCOP process
WAIT - UCOP wait set

UCOP - Command Interpreter termination (main process)

get next entry in UCOP request list (process termination request)

BURRYPROC

remove PCB from scheduling queue
adjust father son and brother linkages
if UCOP has no remaining sons:
CLEARWS - clear WS table entries for C.I. - place on overlay select queue
place any active extra data segment on overlay select queue
release main process stack
return main process PCB entry

if UCOP has no remaining sons, AWAKE PROGEN (possible LOGOFF)
loop back and continue to process request queue entries
if the request queue is empty and no jobs can be started - WAIT

At this point the Command Interpreter (job/session main process) has terminated and the JOB or SESSION has logged off.

Note: When particular steps in the process termination operation were not required to terminate a job or session main process, they were omitted from the above discussion. These omitted steps are included in the material on terminating a son of main process.

HEWLETT hp PACKARD

GENERAL SYSTEMS · 5303 Stevens Creek Blvd., Santa Clara, California 95050, Telephone 408 249-7020

FROM: John Dieckman

DATE: April 8, 1976

TO: Distribution

SUBJECT: Software Channels on the HP3000 Series II

The software channel provides a means of preventing concurrent operations of two or more device controllers. This may become necessary when the aggregate demand of all concurrent devices exceeds the capability of the I/O system. The problem is caused by the fact that some devices cannot wait longer than a specified period of time for data service. If this time is exceeded, the device misses a data word transfer, which results in an overrun. Any device which exhibits this property is termed a synchronous device. Note that the allowable delay of service, or latency, is not the inverse of the device transfer rate. For this reason, the aggregate transfer rate of all synchronous devices may be less than the channel transfer rate, and yet some devices may overrun.[1] Asynchronous devices may wait indefinitely for channel service. It is not possible for an asynchronous device to overrun. The operation of an asynchronous device, however, extends the latency that synchronous devices must endure, which may cause overruns. Operation of devices on a selector channel also impact devices on the multiplexor channel since both channels are competing for memory access.

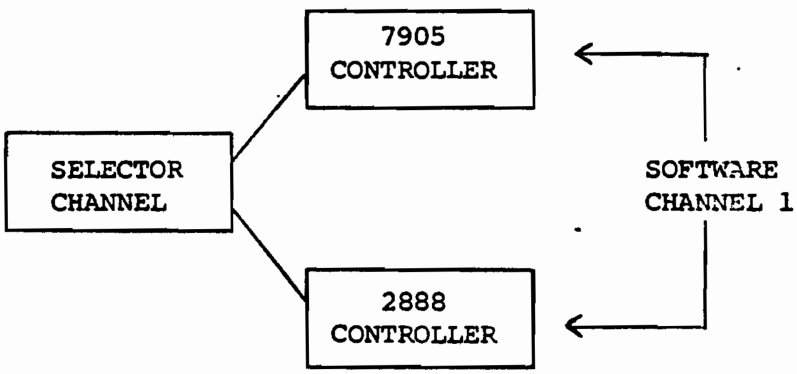
The effect of a data overrun can range from a slight performance degradation to a total system failure. The effect depends on the type of device, the overrun detection capability of the device, and the recovery ability of both the device and the driver software. All synchronous devices detect an overrun condition reliably, except for the 2660 Fixed Head Disc. The 2660 disc controller may fial to detect an overrun that occurs. This means either a CRC error on the disc or improper information in the CPU memory, depending on the transfer direction. If either of these conditions happen to the operating system, a system failure will usually result. For this reason, overruns must be prevented on this controller. A data overrun on the calcomp plotter interface will result in an error, with no recovery possible. The card reader must re-read the card image on an overrun, which necessitates operator intervention. For these reasons, overruns should be prevented on the card reader and plotter. The software recovery in other synchronous device drivers is adequate to recover from an occasional overrun. Frequent overruns are bad for two reasons. The software device driver may exhaust its retry capability and return an error, and excessive retries detracts from system performance. A rule of thumb is that overruns

1. 30036A Multiplexor Channel External Reference Specifications, Section 3.5.

should not occur in more than 1% of the total transfers done to a device, using a worst case test load running under MPE. (Slueth tests are not very realistic, and may place much greater demand on the system than a similar MPE test) the retry rate may be determined by using I/O error logging of the logging facility.

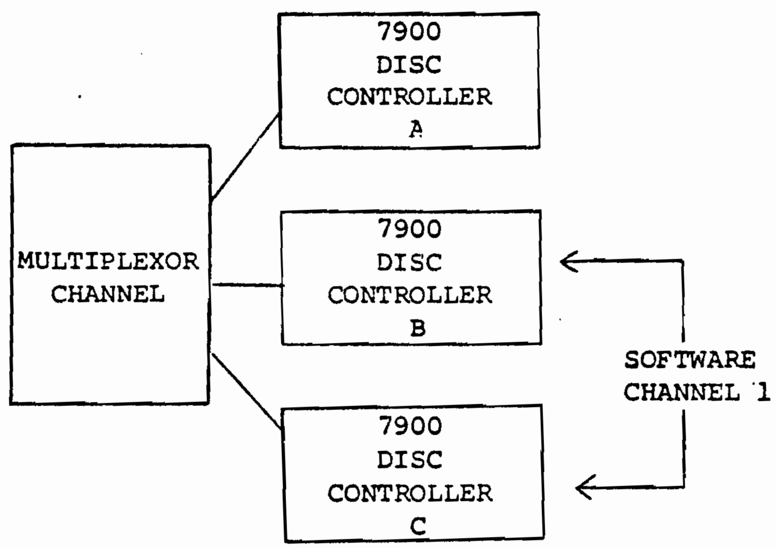
The only means available to reduce or prevent overruns on the Series II is to use a software channel. The advantage of software channels is that they may be applied as needed, and do not impose arbitrary constraints on device overlap. A disadvantage is that each situation must be analyzed before the channels are applied. Thus, a person who configures the system must be more knowledgeable. The current CX series imposes the concurrency constraint through the use of static bandwidth factors for each device driver. This has the advantage of being simple and automatic. A disadvantage is that, since it cannot be configured, it does not allow performance enhancements through the use of hardware means, such as selector channels. More than one software channel may be used to break devices into groups, if this is the desired result. Some examples may be used to demonstrate the use of software channels.

EXAMPLE 1 - Two device controllers on a single selector channel



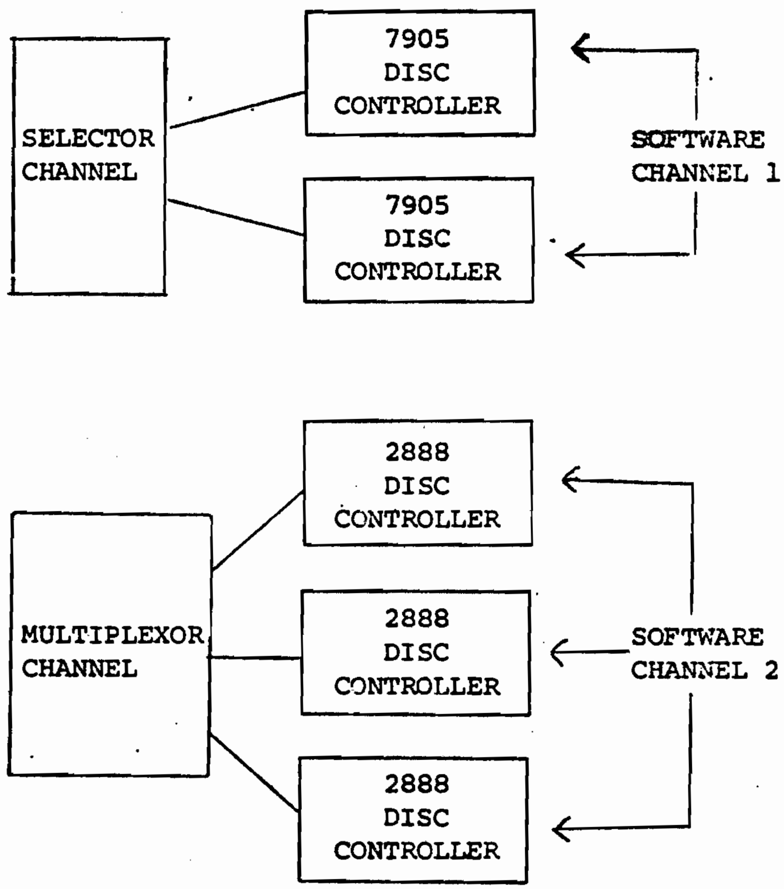
These two device controllers must be placed on a software channel since the selector channel can only service one device at a time. This is done by entering "1" in response to the channel question in device configuration.

EXAMPLE 2 - Three disc controllers on the multiplexor channel



In this case, transfers from all three discs is more than the MUX channel can handle. Assuming that the system disc is connected to controller A, then it is better to put controllers B&C on software channel 1. This configuration disallows concurrent operations of B&C, but allows A to operate concurrently with either B or C.

EXAMPLE 3 - Two disc controller on the selector channel and three disc controller on the multiplexor channel



In this case, only one disc controller on the multiplex channel may operate if one of the discs on the selector channel is in operation. This may be accomplished by method shown. Note that this is only one solution, and the channels may be used in a different way to accomplish the same task. Only two of the five controllers may operate simultaneously with the configuration shown.

The use of software channels cannot be applied uniformly to all devices. Some devices work well with channels. Other devices should not be used with channels because of system/device timing constraints or device errors. Asynchronous devices need not be restricted since overruns cannot occur. Direct I/O devices must not be used with software channels since they do not invoke the MPE I/O system utilities to allocate and deallocate the channel resource. The following tables denote the clarification of each device that we currently support in MPE.

The following devices are recommended for use on a software channel:

Device Name	Product Number	Transmission Mode
7905 DISC	30129A	S
2888 DISC	30102A	S
7900 DISC	30110A	S
2660 DISC	30103A	S
Card Reader	30106/7A	S

The following devices are not recommended for use on a software channel:

Device Name	Product #	Transmission Mode
Hardware Serial Int.	30360A	S
7970 Mag Tape	30115A	S
Synchronous Single Line Cost	30055A	S
Programmable Controller	30300/1A	A
Line Printer	All	A
Card Punch	30112A	A
Paper Tape Punch	30105A	A
Plotter	30126A	S

The following device may not be used on a software channel:

Device Name	Product #	Transmission Mode
Reader/Punch	30119A	D
Paper Tape Reader	30104A	D

Transmission Mode

S = Synchronous
A = Asynchronous
D = Direct I/O only

INPUT DEVICE DIRECTORY/OUTPUT DEVICE DIRECTORY

IDD/ODD

Word	Bits 0–7	Bits 8–15
0	MAX SIZE	CURSIZE
1	HEADSIZE	ENTRYSIZE
2	SUBENTRY AREA POINTER	
3	DD / NEXT DEVICE FILE ID	
4		FENCE
5		
6		
7		
~~10~~ 8		LDEV #
~~11~~ 9	HEADP	
~~12~~ 10	TAILP	
~~13~~ 11		
⋮	HEAD ENTRY	
	HEAD ENTRY	
	⋮ ETC ⋮	
	1ST SUBENTRY AREA	
	SUBENTRY AREA	
	⋮ ETC ⋮	

Words 8–11: 1ST HEAD ENTRY

MAXSIZE – SIZE (WORDS/128)
CURSIZE – SIZE (WORDS/128)
HEADSIZE – SIZE (4)
ENTRYSIZE – SIZE ($36_8 = 30_{10}$)
DD – 0 = INPUT DEVICE DIRECTORY
1 = OUTPUT DEVICE DIRECTORY
FENCE – SEE JMAT DEFINITION

HEADP/TAILP – HEAD POINTER IMMEDIATELY FOLLOWED BY TAILPOINTER. EACH POINTS TO WORD 0 OF ENTRY. NULL CHAIN: HEAD = 0 TAIL = ADDRESS OF HEAD. CHAIN TERMINATED BY A 0 LINK.

~~NOTE: SEE PAGE ... FOR IDD SUBENTRY FORMAT~~
~~SEE PAGE ... FOR ODD SUBENTRY FORMAT~~ III-4.3

XDD(IDD/ ODD)SUBENTRY FORMAT

BIT RESERVED FOR TAPE CARDS
3-9-76

Bits: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

Word (octal)	Fields	Word (decimal)
0	(bit 0–1 reserved) | ST | OUTPRI | C | DEVICE	0
1	TY | JNUMBER	1
2–5	USER NAME	2–5
6–11	ACCOUNT NAME	6–9
12–15	JOB NAME	10–13
16–21	FILE NAME	14–17
22	[illegible] | DEVICE FILE ID	18
23	TF | DA | /////// | HEADX	19
24	LOGICAL DEVICE | LABEL	20
25	SECTOR ADDRESS	21
26	NUMBER OF EXTENTS | VDEV	22
27	LAST EXTENT SIZE	23
30	[illegible] | [illegible] | RS | F | S | /// | NUMCOPIES	24
31	LINK POINTER	25
32–33	NUMBER OF IMAGES	26–27
34–35	TIME MADE READY	28–29

ST – STATE OF ENTRY
- 0 = ACTIVE
- 1 = READY
- 2 = OPENED

3 = Ignore read 3-9-76

C – DEVICE IS A CLASS INDEX TO DEVICE CLASS TABLE

TY – JOB TYPE
- 1 = SESSION
- 2 = JOB

TF – 1 = THERE EXISTS FORMS

DA – 1 = :DATA CREATED INPUT FILE

HEADX – INDEX OF HEAD ENTRY FOR THIS CHAIN OF SUBENTRIES

~~NA – 1 = NO AUTO PAGE EJECT~~

~~[illegible] PAGE MODE~~

RS – 1 = RESTART IS REQUESTED IF WARM-START WAS NECESSARY

F – 1 = NON-STANDARD FORM CONTROL ON DEVICE

S – 1 = FAILURE DURING ALLOCATION OF NEW EXTENT

← SPOOLFILE EXTENSION
OR LABEL EXTENSION 3-9-76

MPE/30 FILE SYSTEM IMS

Jack MacDonald

3/24/76

TABLE OF CONTENTS

1.0 Introduction

This document will eventually be the IMS for the MPE/30 File System. At present it is simply a notebook used to record information that is not currently available eleswhere.

3.0 Table Formats

This section gives a detailed discussion of the main tables constructed and used by the file system. The location and overall structure of each table is given, in addition to the table format and a discussion of each field in the table.

3.2.4 Access Control Block (ACB)

The ACB is essentially the same as under MPE/20. Some new fields have been added, old fields deleted and others moved to different locations. But the essential features remain intact and the ACB serves the same purpose as it did before.

The most notable change to the ACB is due to the new feature of multi-access. In order to implement multi-access the ACB had to be cleaved into two parts: the Physical ACB (PACB) and the Logical ACB (LACB).

The PACB is a skeleton ACB and holds information that is global to all accessors of the file. The LACB holds information that is local to each accessor of the file.

The unique ACB for each accessor is constructed by overlaying the common PACB with the information contained in the individual LACB. Thus, for a multi-access file, there is exactly one PACB for all accessors and one LACB for each accessor.

The term ACB is temporal and is applied to a PACB only when it has been overlayed with the appropriate LACB. If the file is not multi-access then the ACB is the same as the PACB (an LACB does not exist).

3.2.5 Logical Access Control Block (LACB)

The LACB is unique to the 3000/30. Its function is to support the new feature of multi-access.

Under multi-access each accessor shares a single ACB. However each accessor is permitted to view the shared file in a slightly different manner than the other accessors. For example one accessor may access the file in a read only mode while the other accessors may access the file in a read/write mode. This necessitates that each accessor must have a slightly different ACB.

The LACB is used to hold the fields of the common ACB that are unique to each accessor. When an accessor locks the common ACB, the LACB is used to overlay the ACB with the data that is unique to that accessor; this is done within the intrinsic LOCACB. Similarly, when an accessor unlocks the common ACB, the LACB is updated since some of the fields may have been modified due to the access; this is done within the intrinsic UNLOCACB.

All LACBs have the same overall structure:

```
   0   1   2                                                   15
 -----------------------------------------------------------------
 |   3   |                         16                           |  0
 |---------------------------------------------------------------|
 |                          PACB VECTOR                          |  1
 |---------------------------------------------------------------|
 |                          FILE NUMBER                          |  2
 |---------------------------------------------------------------|
 |                           FOPTIONS                            |  3
 |---------------------------------------------------------------|
 |                           AOPTIONS                            |  4
 |---------------------------------------------------------------|
 |                          STATE FLAGS                          |  5
 |---------------------------------------------------------------|
 |                          CONTROL WORD                         |  6
 |---------------------------------------------------------------|
 |                          MODE FLAGS                           |  7
 |---------------------------------------------------------------|
 |                          ERROR NUMBER                         |  8
 |---------------------------------------------------------------|
 |                 LAST I/O TRANSMISSION LOG                     |  9
 |---------------------------------------------------------------|
 |     FILE NAME - 1ST CHAR.     |     FILE NAME - 2ND CHAR.     | 10
 |---------------------------------------------------------------|
 |     FILE NAME - 3RD CHAR.     |     FILE NAME - 4TH CHAR.     | 11
 |---------------------------------------------------------------|
 |     FILE NAME - 5TH CHAR.     |     FILE NAME - 6TH CHAR.     | 12
 |---------------------------------------------------------------|
```

```
|      FILE NAME - 7TH CHAR.       |       FILE NAME - 8TH CHAR.      | 13
|------------------------------------------------------------------------|
|                   RECORD SIZE (POSITIVE BYTES)                         | 14
|------------------------------------------------------------------------|
|                   BLOCK SIZE (POSITIVE WORDS)                          | 15
--------------------------------------------------------------------------
```

In general the following identifiers are used when referring to a LACB:

```
DEFINE
LACBSIZE        = LACB.(2:14)#,    <<SIZE IN WORDS>>
LACBPACB        = LACB(1)#,        <<PACB VECTOR>>
LACBFNUM        = LACB(2)#,        <<FILE NUMBER>>
LACBFOPTIONS    = LACB(3)#,        <<FOPTIONS>>
LACBAOPTIONS    = LACB(4)#,        <<AOPTIONS>>
LACBSTATE       = LACB(5)#,        <<STATE FLAGS>>
LACBCTL         = LACB(6)#,        <<CONTROL WORD>>
LACBMODE        = LACB(7)#,        <<MODE SETTING>>
LACBERROR       = LACB(8)#,        <<ERROR CODE>>
LACBTLOG        = LACB(9)#,        <<LAST I/O TLOG>>
LACBNAME1       = LACB(10)#,       <<FILE NAME>>
LACBNAME2       = LACB(11)#,       <<FILE NAME>>
LACBNAME3       = LACB(12)#,       <<FILE NAME>>
LACBNAME4       = LACB(13)#,       <<FILE NAME>>
LACBRSIZE       = LACB(14)#,       <<RECORD SIZE - POS. BYTES>>
LACBBSIZE       = LACB(15)#;       <<BLOCK SIZE - POS. WORDS>>
```

The following is an alphabetized list of the above identifiers along with a discussion of their meaning.

3.2.6 [redacted]

The PACB is similar to the ACB under MPE/20. The only notable differences are:

* Some old fields were deleted and some new fields were added. The result is a small increase in the size of the basic ACB.

* The ACBs are no longer linked together. After consultation with concerned parties it was decided that this was a vestigal feature and didn't warrant further support.

* The format of the buffering extension now comes in two flavors. The origional flavor, which has the buffer storage within the block buffer, was retained. The new flavor uses system buffers instead of block buffers.

The overall structure of the PACB is:

```
-------------------------
|                       |
|       BASIC ACB       |      (fixed)
|                       |
|-----------------------|
|                       |
|       BUFFERING       |
|                       |      (variable)
|       EXTENSION       |
|                       |
-------------------------
```

The buffering extension is optional; it is present if and only if the file is accessed with buffering. As mentioned above the buffering extension has two formats: one for ACB buffers and the other for system buffers. As a result there are exactly three possible formats for an ACB:

1. No buffers; the buffering extension is not present.

2. ACB buffers; the buffering extension is present and the buffers are in the block buffers.

3. System buffers; the buffering extension is present and the buffers are not in the block buffers.

The basic ACB (or NOBUF ACB) has the following format:

```
 0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15
```

Table Formats - CBTAB

Fields	Word
2 | COMPLETE ACB SIZE	0
FILE NUMBER |	1
RECORD TRANSFER COUNT	2–3
BLOCK TRANSFER COUNT	4–5
FILE POINTER	6–7
| F | P | L | S | F | T | B | E | C | D | EOF T | EOF M |	8
AOPTIONS	9
ERROR CODE	10
S | B | DEVICE TYPE | LAST LOGICAL I/O STATUS	11
RECORD SIZE IN BYTES	12
BLOCK SIZE IN WORDS	13
FCB VECTOR	14
| B | N | P	15
	16
CARRIAGE CONTROL CODE	17
ACCESS BIT MAP | LOGICAL DEVICE NUMBER	18
VOLUME TABLE INDEX	19
FOPTIONS	20
LAST I/O TRANSMISSION LOG	21
NAME TYPE | FILE DISPOSITION	22
MODE SETTING | BLOCKING FACTOR	23
CURRENT VARIABLE BLOCK NUMBER	24–25

FILE NAME - 1ST CHAR.	FILE NAME - 2ND CHAR.	26
FILE NAME - 3RD CHAR.	FILE NAME - 4TH CHAR.	27
FILE NAME - 5TH CHAR.	FILE NAME - 6TH CHAR.	28
FILE NAME - 7TH CHAR.	FILE NAME - 8TH CHAR.	29
TERMINAL STOP CHARACTER	A | M | B | | SEOFS | EOFS	30
		31
LAST NO-WAIT I/O FILE POINTER		32
		33
NR. INPUT LABC'S	TOTAL NR. LACB'S	34
FMAVT INDEX		35
XDD INDEX		36
S | M |	VIRTUAL LOGICAL DEVICE NR.	37
SPOOLED DEVICE FOPTIONS		38
SPOOLED DEVICE AOPTIONS		39
SPOOLED DEVICE TYPE | SPOOLED DEVICE RECORD SIZE		40
		41
HIGHEST BLOCK NUMBER STARTED		42
		43
| CURRENT BUFFER |	| NR. BUFFERS	44
BUFFER SIZE		45
CURRENT RECORD BYTE POINTER		46
CURRENT RECORD LOGICAL BYTE INDEX		47
CURRENT BUFFER BYTE DISPLACEMENT		48

In general the following identifiers are used when referring to an ACB:

```
DEFINE
ACBSIZE          = ACB.(2:14)#,      <<SIZE IN WORDS>>
ACBFNUM          = ACB(1).(0:8)#,    <<FILE NUMBER>>
ACBRTFRCT        = ACBDBL(1)#,       <<RECORD TRANSFER COUNT>>
ACBBTFRCT        = ACBDBL(2)#,       <<BLOCK TRANSFER COUNT>>
ACBFPTR          = ACBDBL(3)#,       <<RECORD POINTER>>
ACBLSTATE        = ACB(8)#,          <<LOCAL STATE FLAGS>>
ACBAOPTIONS      = ACB(9)#,          <<AOPTIONS>>
ACBERROR         = ACB(10)#,         <<ERROR CODE>>
ACBSYSBUF        = ACB(11).(0:1)#,   <<SYSTEM BUFFER FLAG>>
ACBBREAK         = ACB(11).(1:1)#,   <<BREAK MODE FLAG>>
ACBDTYPE         = ACB(11).(2:6)#,   <<DEVICE TYPE>>
ACBACCCL         = ACB(11).(2:3)#,   <<DEVICE ACCESS CLASS>>
ACBSUBCL         = ACB(11).(5:3)#,   <<DEVICE SUB-CLASS>>
ACBSTATUS        = ACB(11).(8:8)#,   <<LAST I/O STATUS>>
ACBQSTATUS       = ACB(11).(8:5)#,   <<QUALIFYING STATUS PART>>
ACBGSTATUS       = ACB(11).(13:3)#,<<GENERAL STATUS PART>>
ACBRSIZE         = ACB(12)#,         <<RECORD SIZE - BYTES>>
ACBBSIZE         = ACB(13)#,         <<BLOCK SIZE - WORDS>>
ACBFCB           = ACB(14)#,         <<FCB VECTOR>>
ACBGSTATE        = ACB(15)#,         <<GLOBAL STATE FLAGS>>
ACBTEMP1         = ACB(16)#,         <<TEMP CELL>>
ACBCTL           = ACB(17)#,         <<CARRIAGE CONTROL CODE>>
ACBACCESS        = ACB(18).(0:8)#,   <<ACCESS BIT MAP>>
ACBDADDR         = ACB(18).(8:8)#,   <<LOGICAL DEVICE NR.>>
ACBVDADDR        = ACB(19)#,         <<VOLUME TABLE INDEX>>
ACBFOPTIONS      = ACB(20)#,         <<FOPTIONS>>
ACBTLOG          = ACB(21)#,         <<LAST I/O TRANSMISSION LOG>>
ACBDNTYPE        = ACB(22).(0:8)#,   <<NAME TYPE FOR DIRECTORY>>
ACBDISP          = ACB(22).(8:8)#,   <<FILE DISPOSITION>>
ACBMODE          = ACB(23).(0:8)#,   <<MODE FLAGS>>
ACBTAPEERROR     = ACB(23).(4:1)#,   <<REPORT RECOVERED TAPE ERRORS>
ACBINHIBITCRLF   = ACB(23).(5:1)#,   <<INHIBIT TERMINAL CR/LF>>
ACBQUIESCE       = ACB(23).(6:1)#,   <<CRITICAL OUTPUT VERIFICATION>
ACBBLKFACT       = ACB(23).(8:8)#,   <<BLOCKING FACTOR>>
ACBBLK           = ACBDBL(12)#,      <<CURRENT VARIABLE BLOCK NR.>>
ACBNAME          = ACB(26)#,         <<LOCAL FILE NAME>>
ACBSTOPCHAR      = ACB(30).(0:8)#,   <<TERMINAL STOP CHARACTER>>
ACBNOWAITEOF     = ACB(30).(8:1)#,   <<EOF ADVANCED FLAG>>
ACBNOWAITMODE    = ACB(30).(9:1)#,   <<LAST NO-WAIT I/O MODE>>
ACBABORTREAD     = ACB(30).(10:1)#,<<ABORT BROKEN RE-READ FLAG>>
ACBSAVEEOFS      = ACB(30).(12:2)#,<<FOR SAVING ACBEOFS>>
ACBEOFS          = ACB(30).(14:2)#,<<EOF FLAGS>>
ACBTEMP2         = ACB(31)#,         <<TEMP CELL>>
ACBNOWAITFPTR    = ACBDBL(15)#,      <<LAST NO-WAIT FPTR>>

ACBSHCNTS        = ACB(34)#,         <<LACB COUNTS>>
ACBSHCNTIN       = ACB(34).(0:8)#,   <<NR. INPUT LACB'S>>
ACBSHCNT         = ACB(34).(8:8)#,   <<NR. LACB'S>>
ACBFMAVTX        = ACB(35)#,         <<FMAVT INDEX>>
ACBSPXDDX        = ACB(36)#,         <<XDD INDEX>>
ACBSPOOL         = ACB(37).(0:1)#,   <<SPOOLED DEVICE FLAG>>
```

```
ACBSPOOLIO        = ACB(37).(0:2)#, <<SPOOLED IN/OUT>>
ACBSPVDEV         = ACB(37).(8:8)#, <<SPOOLED VIRTUAL DEVICE>>
ACBSPFOPT         = ACB(38)#,       <<SPOOLED DEVICE FOPTIONS>>
ACBSPAOPT         = ACB(39)#,       <<SPOOLED DEVICE AOPTIONS>>
ACBSPTYRC         = ACB(40)#,       <<SPOOLED DEVICE TYPE/RECSIZE>>
ACBSPTYPE         = ACB(40).(0:6)#, <<SPOOLED DEVICE TYPE>>
ACBSPREC          = ACB(40).(6:10)#,<<SPOOLED DEVICE RECSIZE>>
ACBXXXX           = ACB(41)#,       <<TEMP CELL>>

ACBHIBLK          = ACBDBL(21)#,    <<HIGHEST BLOCK STARTED>>
ACBCURRBUF        = ACB(44).(4:4)#, <<CURRENT BUFFER NR.>>
ACBNUMBUFS        = ACB(44).(12:4)#,<<NR. OF BUFFERS LESS 1>>
ACBBUFSIZE        = ACB(45)#,       <<BUFFER SIZE>>
ACBRECPTR         = ACB(46)#,       <<RECORD BYTE POINTER>>
ACBBUFUSED        = ACB(47)#,       <<USED BLOCK BYTE COUNT>>
ACBBUFDISP        = ACB(48)#,       <<BUFFER BYTE DISPLACEMENT>>
ACBBUFPOOL        = ACB(49)#,       <<FIRST BLOCK BUFFER>>

ACBBOT            = ACBGSTATE.(13:1)#, <<BEGINNING OF MAG. TAPE>>
ACBNEWEOF         = ACBGSTATE.(14:1)#, <<NEW EOF NEEDED - M.T. ONLY
ACBPRIV           = ACBGSTATE.(15:1)#, <<PRIVILEGED FILE>>

ACBEOF            = ACBLSTATE.(1:1)#,  <<END OF FILE SENSED>>
ACBLPCTL          = ACBLSTATE.(2:2)#,  <<PAGE AND LINE CONTROL>>
ACBPAGECTL        = ACBLSTATE.(2:1)#,  <<PAGE CONTROL>>
ACBLINECTL        = ACBLSTATE.(3:1)#,  <<LINE CONTROL>>
ACBSTREAM         = ACBLSTATE.(4:1)#,  <<STREAM I/O>>
ACBFKEYS          = ACBLSTATE.(5:1)#,  <<RESTORE FUNCTION KEYS>>
ACBXMITCRLF       = ACBLSTATE.(6:1)#,  <<TRANSMIT CR,LF TO USER>>
ACBTBLOCK         = ACBLSTATE.(7:1)#,  <<DISABLE BLOCK MODE>>
ACBBINARYIO       = ACBLSTATE.(8:1)#,  <<EIGHT BIT TERM. TRANSFERS>
ACBCARRIAGE       = ACBLSTATE.(9:1)#,  <<CARRIAGE CONTROL FLAG>>
ACBDEFBLOCK       = ACBLSTATE.(10:1)#, <<DEFAULT BLOCKING>>
ACBREADCODE       = ACBLSTATE.(11:4)#, <<INPUT EOF CHECK>>
ACBREADTYPE       = ACBLSTATE.(11:2)#, <<INPUT EOF TYPE>>
ACBREADMODE       = ACBLSTATE.(13:2)#; <<INPUT EOF MODE>>
```

The following is an alphabetized list of the above identifiers along with a discussion of their meaning:

ACBABORTREAD This flag is used to abort a broken terminal re-read. The flag is set via the ABORT parameter to FUNBREAK. If the flag is set then the READ PENDING message will be aborted along with the re-read. This feature is needed to handle the BREAK...:ABORT, etc. situation.

ACBACCCL This is the access class part of the device type number. The following are the legal values:

- 0 - direct (e.g. disc)
- 1 - serial input (e.g. card reader)

2 - parallel input/output (e.g. terminal)
3 - serial input/output (e.g. mag tape)
4 - serial output (e.g. line printer)

ACBACCESS — This is the access bit map for the file.

ACBAOPTIONS — This is the AOPTIONS in effect for this file access.

ACBBINARYIO — This bit controls full eight bit transfers on the 2644 page mode terminal. This bit is adjusted by FCONTROL(26) and FCONTROL(27).

ACBBLK — This is the block number of the current variable record format block. Note that this is applicable iff the record format is variable.

ACBBLKFACT — This is the blocking factor for the file. It is the number of records in a block.

ACBBOT — This flag is used to indicate the position of a mag tape. If not set then the tape is not on the load point; if set then the tape is at the load point. Note that this flag is applicable only to mag tape files.

ACBBREAK — This is the break mode flag. It is applicable iff the ACB is the one for $STDIN/$STDLIST. If set it means that the BREAK key has been hit and that the CI should have high priority access to the ACB. The flag will be cleared when a RESUME or ABORT is issued.

ACBBSIZE — This is the block size, in words, of the file.

ACBBTFRCT — This is the total number of blocks transfered to and from the file. The initial value is 0D.

ACBBUFDISP — This is the physical byte displacement, relative to the base of the buffer, corresponding to ACBRECPTR. If ACB buffers are used then the value is the same as ACBBUFUSED. If system buffers are used the value will range from 0 to 255 and is the displacement corresponding to ACBRECPTR from the base of the physical system buffer. It is needed because system buffers are discontiguous; otherwise it could be calculated from ACBRECPTR. Note that this is applicable iff the file access is buffered.

ACBBUFSIZE — This is the size, in words, of the complete block buffer. This includes the block buffer control block, which is 9 words long. If system buffers are used then the value is 9; otherwise ACB buffers are used and the value is the block size plus 9. Note that this is applicable iff the file access is buffered.

ACBBUFUSED — This is the logical byte index, relative the the base of the buffer, corresponding to ACBRECPTR. Note that the value will be the same regardless as to whether ACB or system buffers are used. It is needed because system buffers are discontiguous; otherwise it could be calculated from ACBRECPTR. Note that this is applicable iff the file access is buffered.

ACBCARRIAGE — This bit signifies that the file has carriage control. I think it is redundant; ACBFOPTIONS already has a carriage control bit.

ACBCTL — This is the CONTROL parameter from the last FWRITE. Note that this value is pertainent iff the file was opened with carriage control.

ACBCURRBUF — This is the buffer number (0-relative) containing the most recently referenced record. Note that this is applicable iff the file access is buffered.

ACBDADDR — This is the logical device number of the file. In the case of a disc file this is the logical device number of the first extent.

ACBDEFBLOCK — This bit signifies that the file is to be accessed with default blocking. The bit is initialized from the FOPEN stateword STATE. It does not need to be in the ACB; it is mentioned here only to signify that the bit is effectively used due to the way ACBLSTATE is initialized from STATE.

ACBDISP — This is the file close disposition derived from the FOPEN call. Note that the only way this can be specified is via a file equation. The legal values are the same as those for FCLOSE.

ACBDNTYPE — This is the file reference format type number and is derived from the FOPEN call. The following are the legal values:

0 - full name
1 - account name absent
2 - group and account name absent
3 - null name

This information is needed by FRENAME.

ACBDTYPE — This is the device type number of the file. See ACBSUBCL for a list of legal values.

ACBEOF — This bit is set when EOF has been sensed.

ACBFOFS — This is the EOF sensing mode used for read operations.

ACBERROR — This is the error number for the file. It is used by all intrinsics except FOPEN. When an error is detected the error number is placed in this cell. The error number should not be explicitly cleared; instead it persists until it is overlayed by the next error number.

ACBFCB — This is the FCB vector for the file. Note that this is applicable iff the file is a disc file.

ACBFKEYS — This bit controls the definition of the f1 and f2 function keys on the 2644 page mode terminal. This bit is adjusted by FCONTROL(32) and FCONTROL(33).

ACBFNUM — This is the file number, the range of which is from 1 to 255. This number is used by the file system as an entry index into the AFT so that the ACB may be located.

ACBFOPTIONS — This is the FOPTIONS in effect for this file access.

ACBFPTR — This is the sequential access record pointer; it contains the next sequetial record number. The initial value is 0D. Note that this value is applicable only for the FREAD, FWRITE and FUPDATE intrinsics. However the value is maintained by all data transferring file system intrinsics.

ACBGSTATE — These are miscellaneous state flags. These are "global" in nature in that they are the same for all accessors in a multi-access environment. The constituent bits are described individually.

ACBGSTATUS — This is the general part of the last I/O status for the file. The following are the legal values:

- 0 - pending
- 1 - successful
- 2 - end of file
- 3 - unusual condition
- 4 - irrecoverable error

ACBHIBLK — This is the highest block number for which an anticipatory read has been issued. Note that this is applicable iff the file access is buffered.

ACBINHIBITCRLF — This bit controls the termination of lines written to the terminal. If not set then each line is terminated with a CR and LF; if set then no line termination characters are used. Note that this is valid iff the file is a terminal file. This bit is adjusted by FSETMODE.

ACBLINECTL — This is the line control bit. If not set then each line is post-spaced; if set then each line is pre-spaced. This is used line printers and terminals only. This bit is adjusted by FCONTROL(1) and FWRITE with the appropriate carriage control.

ACBLPCTL	This are the line and page control bits, each of which is described separately.
ACBLSTATE	These are miscellaneous state flags. They are "local" in nature in that they may be different for each accessor in a multi-access environment. Bits (9:6) are initialized from the stateword local variable called STATE in FOPEN; the remaining bits are initialized individually. The constituent bits are described individually.
ACBMODE	These are miscellaneous mode flags. The constituent bits are described individually.
ACBNAME	The is the local file name. The name is eight bytes in length with trailing blanks added.
ACBNEWEOF	This flag is used to indicate that a new tape mark should be written before the tape is rewound or backspaced. If not set then a tape mark is not needed; if set then a tape mark is needed. Note that this flag is applicable only to mag tape files.
ACBNOWAITEOF	This bit is used to save the value of the local eof advanced flag NEWEOF in FNOBUF between the I/O initiation and I/O completion calls. Note that this flag is applicable iff the file is accessed in no-wait I/O mode.
ACBNOWAITFPTR	This cell is used to save the file pointer between no-wait I/O initiation and completion calls. Note that this cell is pertainent iff the file is accessed in no-wait I/O mode.
ACBNOWAITMODE	This cell is used to save the I/O mode between no-wait I/O initiation and completion calls. If the bit is set then the last I/O request was a write; otherwise it was a read. Note that this cell is pertainent iff the file is accessed in no-wait I/O mode.
ACBNUMBUFS	This is the number of buffers, less one, used for the file access. Note that this is applicable iff the file access is buffered.
ACBPAGECTL	This is the page control bit. If not set then a page is assumed to consist of 60 lines (auto page eject); if set then a page is assumed to consist of 66 lines (no auto page eject). This is used primarily for line printers but is also valid for terminals; and these are the only devices for which this is valid. This bit is adjusted by

FCONTROL(1) and FWRITE with the appropriate carriage control.

ACBPRIV	This flag is used to indicate that the file is privileged in that it has a negative file code. If not set then the file is not privileged; if set then the file is privileged and the user must be in privileged mode to open it.
ACBQSTATUS	This is the qualifying part of the last I/O status for the file. Note that the values are unique for each general status part. See I/O System IMS for all legal values.
ACBQUIESCE	This bit controls critical output verification. If not set the buffered output is not guaranteed to be written to the device when control is returned to the user; if set then buffered output is guaranteed to have been written to the device when control is returned to the user. This bit is adjusted by FSETMODE.
ACBREADCODE	This field consists of the input EOF checking type and mode. These fields are described individually.
ACBREADMODE	This field controls the input EOF checking mode.
ACBREADTYPE	This field controls the input EOF checking type.
ACBRECPTR	This is a relative byte pointer to the current record. If system buffers are used then the pointer is relative to the base of the system buffer data segment; otherwise ACB buffers are used and the pointer is relative to the base of the block buffer. Note that this is applicable iff the file access is buffered.
ACBRSIZE	This is the record size, in positive bytes, for the file.
ACBRTFRCT	This is the total number of records transfered to and from the file. The initial value is 0D.
ACBSAVEEOFS	This field is used to save the contents of ACBEOFS during BREAK mode processing.
ACBSIZE	This is the size, in words, of the complete ACB. It includes the buffering extension, if present.
ACBSTATUS	This is the last I/O status for the file. It comes from the I/O status part of the IOCB returned by ATTACHIO. Note that not all ATTACHIO calls update this cell.
ACBSTOPCHAR	This is the record termination character used for terminal reads. This character can be changed via FCONTROL.
ACBSTREAM	This bit signifies inter-block garbage for disc files. If

set then the block size is a multiple of 128 words and therefore there is no garbage data between blocks. This fact is used to improve multi-record I/O by mapping the request into as few ATTACHIOs as possible.

ACBSUBCL — This is the sub-class part of the device type number. Note that the sub-class is unique for each access class. The following are the legal values:

- 0 - direct
  - 0 - moving head disc
  - 1 - fixed head disc
- 1 - serial input
  - 0 - card reader
  - 1 - paper tape reader
- 2 - parallel input/output
  - 0 - terminal
  - 4 - card reader/punch
  - 6 - SSLC
  - 7 - programmable controller
- 3 - serial input/output
  - 0 - mag tape
- 4 - serial output
  - 0 - line printer
  - 1 - card punch
  - 2 - paper tape punch
  - 3 - CALCOMP 500 plotter
  - 4 - CALCOMP 600 plotter
  - 5 - CALCOMP 700 plotter

ACBSYSBUF — This is the system buffer flag. If set it means that the file access is buffered and that system buffers are used instead of ACB buffers.

ACBTAPEERROR — This bit controls the reporting of recovered mag tape errors. If not set the recovered errors are not reported to the user; if set then recovered errors are reported to the user by returning CCL and error number 39. Note that this is valid iff the file is a mag tape file. This bit is adjusted by FSETMODE.

ACBTBLOCK — This bit controls block mode transfers on the 2644 page mode terminal. This bit is adjusted by FCONTROL(28) and FCONTROL(29).

ACBTEMP1 — This cell is unassigned.

ACBTEMP2 — This cell is unassigned.

ACBTLOG — This is the last I/O transmission log for the file. It comes from the I/O transmission log part of the IOCB returned by ATTACHIO. Note that not all ATTACHIO calls update this cell.

ACBVDADDR — This is the volume table index for the file. Note that this value is applicable iff the file is a disc file.

ACBXMITCRLF — This bit controls CR and LF insertion into the user buffer on the 2644 page mode terminal. This bit is adjusted by FCONTROL(30) and FCONTROL(31).

The [redacted]s the following format: is the link between user and the I/O System

```
  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15
 ----------------------------------------------------------------
|                       IOQ ENTRY INDEX                          |  0
|----------------------------------------------------------------|
|                     SYSTEM BUFFER INDEX                        |  1
|----------------------------------------------------------------|
|                        IOCB - STATUS                           |  2
|----------------------------------------------------------------|
|                  IOCB - TRANSMISSION LOG                       |  3
|----------------------------------------------------------------|
|                                                                |  4
|                        BLOCK NUMBER                            |
|                                                                |  5
|----------------------------------------------------------------|
| BLOCK LOGICAL DEVICE NUMBER   |                                |  6
|-------------------------------                                 |
|                     BLOCK SECTOR NUMBER                        |  7
|----------------------------------------------------------------|
| SYSTEM BUFFER DISPLACEMENT|                       | W | M | P  |  8
|----------------------------------------------------------------|
|                                                                |  9
|                                                                |
|                           BUFFER                               |
|                                                                |
|                                                                |
 ----------------------------------------------------------------
```

In general the following identifiers are used when referring to buffering extension part of the ACB:

```
DEFINE
BLKIOQX        = BLK#,              <<IOQ ENTRY INDEX>>
BLKSYSBUFX     = BLK(1)#,           <<SYSTEM BUFFER INDEX>>
BLKIOCB        = BLKDBL(1)#,        <<IOCB>>
BLKLSTAT       = BLK(2)#,           <<IOCB - STATUS>>
BLKTLOG        = BLK(3)#,           <<IOCB - TRANSMISSION LOG>>
BLKBLOCK       = BLKDBL(2)#,        <<BLOCK NUMBER>>
BLKDADDR       = BLKDBL(3)#,        <<BLOCK SECTOR NUMBER>>
```

```
BLKLDEV          = BLK(6).(0:8)#,   <<BLOCK LOGICAL DEVICE NR.>>
BLKSYSBUFDISP    = BLK(8).(0:7)#,   <<SYS. BUF. DISPLACEMENT>>
BLKFLAGS         = BLK(8).(13:3)#,  <<BLOCK I/O FLAGS>>
BLKIOOUT         = BLK(8).(13:1)#,  <<LAST I/O WAS WRITE?>>
BLKDIRTY         = BLK(8).(14:1)#,  <<BUFFER MODIFIED?>>
BLKIOPEND        = BLK(8).(15:1)#,  <<I/O IN PROGRESS?>>
BLKIOCOMP        = BLK(8).(14:2)#,  <<I/O COMPLETE - NOT DIRTY>>
BLKBUFFER        = BLK(9)#;         <<BLOCK BUFFER>>
```

The following is an alphabetized list of the above identifiers along with a discussion of their meaning:

BLKBLOCK — This is the block number of the data contained in the buffer.

BLKBUFFER — This is the block buffer. Note that this is valid iff the buffer is in the ACB; otherwise system buffers are used and the location of the buffer is given by BLKSYSBUFX and BLKSYSBUFDISP.

BLKDADDR — This is logical device number and sector number of the block.

BLKDIRTY — This is the buffer modified flag. It is set if the contents of the buffer is modified and therefore does not reflect the data on the disc. When the block buffer is re-used for another block this flag is checked to see if the block needs to be written to the disc.

BLKFLAGS — These are the miscellaneous flags associated with the block.

BLKIOCB — This is the IOCB returned by the I/O system when the block I/O has completed. On a blocked I/O request this is obtained from the ATTACHIO call; on an unblocked I/O request this is obtained from WAITFORIO.

BLKIOCOMP — This is the buffer modified flag (BLKDIRTY) and the I/O in progress flag (BLKIOPEND).

BLKIOOUT — This is the mode of the I/O operation for the block. It is set if the operation was a write; otherwise it is cleared indication that the operation was a read.

BLKIOPEND — This is the I/O in progress flag. It is set if the I/O is pending; it is cleared when the I/O is complete.

BLKIOQX — This is the IOQ index of the unblocked I/O request for the block.

BLKLDEV — This is the logical device number of the block.

BLKLSTAT	This is the I/O status part of the IOCB. It consists of the PCB number and the error code for the completed I/O request.
BLKSYSBUFDISP	This is the word displacement of the block from the beginning of the system buffer. Generally this is zero since most blocks begin at the start of a system buffer. However when more than one block will fit into a system buffer this cell indicates the blocks displacement.
BLKSYSBUFX	This is the system buffer index of the block buffer. Note that this does not completely specify the location of the block buffer; BLKSYSBUFDISP is needed also. Also this cell is valid iff system buffering is used.
BLKTLOG	This is the transmission log part of the IOCB. It is the number of words/bytes transfered by the the I/O request.

[redacted]

FCB can be 29+64 = 93 words long

The FCB is essentially the same as under MPE/20. The only notable differences are:

* Since extents can be on different logical devices the high byte of an extent descriptor contains a logical device number.

* Since there can be up to 32 extents the extent map may be longer.

The overall structure of the FCB is:

```
 -------------------------
 |                       |
 |       BASIC FCB       |      (fixed)
 |                       |
 |-----------------------|
 |                       |
 |        EXTENT         |
 |                       |      (variable)
 |         MAP           |
 |                       |
 -------------------------
```

The basic FCB has the following format:

```
   0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15
 -----------------------------------------------------------------
 |  1  |               COMPLETE FCB SIZE                         |  0
 |---------------------------------------------------------------|
 |                      NEW FCB VECTOR                           |  1
 |---------------------------------------------------------------|
 |                       FOPTIONS                                |  2
 |---------------------------------------------------------------|
 |                   DEVICE SPECIFICATION                        |  3
 |---------------------------------------------------------------|
 | PREV. LOCK|   DEVICE TYPE    |             |DEVICE SUB-TYPE   |  4
 |---------------------------------------------------------------|
 | NR. OPENS FOR OUTPUT         |    NR. OPENS FOR ANY MODE      |  5
 |---------------------------------------------------------------|
 |                   CREATOR ACB VECTOR                          |  6
 |---------------------------------------------------------------|
 |                       RIN NUMBER                              |  7
 |---------------------------------------------------------------|
 |                    EXCLUSIVE STATUS                           |  8
 |---------------------------------------------------------------|
 |PEND. DISP.|                                                   |  9
```

FILE LIMIT IN BLOCKS		10
		11
RESERVED FOR IMAGE		12
		13
END OF DATA POINTER		14
		15
NR. USER LABELS WRITTEN	NR. USER LABELS AVAILABLE	16
EXTENT SIZE IN SECTORS		17
BLOCKING FACTOR	SECTORS PER BLOCK	18
SECTOR OFFSET TO DATA	| NR. EXTENTS - 1	19
LAST EXTENT SIZE IN SECTORS		20
NR. OPENS INPUT MODE		21
GROUP NAME - 1ST CHAR.	GROUP NAME - 2ND CHAR.	22
GROUP NAME - 3RD CHAR.	GROUP NAME - 4TH CHAR.	23
GROUP NAME - 5TH CHAR.	GROUP NAME - 6TH CHAR.	24
GROUP NAME - 7TH CHAR.	GROUP NAME - 8TH CHAR.	25
ACCOUNT NAME - 1ST CHAR.	ACCOUNT NAME - 2ND CHAR.	26
ACCOUNT NAME - 3RD CHAR.	ACCOUNT NAME - 4TH CHAR.	27
ACCOUNT NAME - 5TH CHAR.	ACCOUNT NAME - 6TH CHAR.	28
ACCOUNT NAME - 7TH CHAR.	ACCOUNT NAME - 8TH CHAR.	29
LOGICAL DEVICE NUMBER		30
FIRST EXTENT SECTOR NUMBER		31
. . .		
LOGICAL DEVICE NUMBER		
LAST EXTENT SECTOR NUMBER		

In general the following identifiers are used when referring to an FCB.

```
DEFINE
FCBSIZE          = FCB.(2:14)#,      <<SIZE IN WORDS>>
FCBNEWFCBV       = FCB(1)#,          <<NEW FCB VECTOR>>
FCBFOPTIONS      = FCB(2)#,          <<FOPTIONS>>
FCBDEVICE        = FCB(3)#,          <<LDEV OR DEVICE CLASS>>
FCBLKST          = FCB(4).(0:2)#,    <<PREVIOUS LOCK STATE>>
FCBDTYPE         = FCB(4).(2:6)#,    <<DEVICE TYPE>>
FCBSUBTYPE       = FCB(4).(12:4)#,   <<DEVICE SUB-TYPE>>
FCBOCNTOUT       = FCB(5).(0:8)#,    <<NR. ACCESSORS - OUTPUT>>
FCBOCNT          = FCB(5).(8:8)#,    <<NR. ACCESSORS>>
FCBACB           = FCB(6)#,          <<CREATOR ACB VECTOR>>
FCBRIN           = FCB(7)#,          <<RIN NUMBER>>
FCBEXCLSTAT      = FCB(8)#,          <<EXCLUSIVE STATUS>>
FCBDISP          = FCB(9).(0:3)#,    <<PENDING DISPOSITION>>
FCBFLIM          = FCBDBL(5)#,       <<FILE LIMIT>>
FCBIMAGE         = FCBDBL(6)#,       <<RESERVED FOR IMAGE>>
FCBEOF           = FCBDBL(7)#,       <<END OF FILE POINTER>>
FCBUSERLBL       = FCB(16)#,         <<USER LABEL INFO>>
FCBLBLEOF        = FCB(16).(0:8)#,   <<NR. LABELS WRITTEN>>
FCBLBL           = FCB(16).(8:8)#,   <<NR. LABELS AVAILABLE>>
FCBEXTSIZE       = FCB(17)#,         <<EXTENT SIZE>>
FCBBLKFACT       = FCB(18).(0:8)#,   <<BLOCKING FACTOR>>
FCBSECTPBLK      = FCB(18).(8:8)#,   <<SECTORS PER BLOCK>>
FCBSECTOFF       = FCB(19).(0:8)#,   <<SECTOR OFFSET TO DATA>>
FCBNUMEXTS       = FCB(19).(11:5)#,<<NR. EXTENTS LESS 1>>
FCBLASTEXTSIZE   = FCB(20)#,         <<LAST EXTENT SIZE>>
FCBOCNTIN        = FCB(21).(8:8)#,   <<NR. ACCESSORS - INPUT>>
FCBGN            = FCB(22)#,         <<GROUP NAME>>
FCBAN            = FCB(26)#,         <<ACCOUNT NAME>>
FCBLABEL         = FCBDBL(15)#,      <<LABEL LDEV AND SECTOR>>
FCBLDEV          = FCB(30).(0:8)#,   <<LABEL LDEV>>
FCBEXTMAP        = FCB(30)#;         <<EXTENT MAP>>
```

The following is an alphabetized list of the above identifiers along with a discussion of their meaning:

FCBACB — This is the vector of the ACB that was created at the same time as the FCB.

FCBAN — This is the account name of the file. It is eight bytes in length with trailing blanks added.

FCBBLKFACT — This is the blocking factor of the file. It is the number of logical records in a physical block. Legal values range from 1 to 255.

FCBDEVICE	This specifies the device on which the file resides. If it is positive then it represents a logical device number; if it is negative then it represents a (negative) device class index.
FCBDISP	This is the pending FCLOSE disposition for the file. Legal values are:
FCBDTYPE	This is the device type number of the first extent of the file. See ACBDTYPE for a list of legal values.
FCBEOF	This is the end-of-file pointer for the file. It is a double word integer representing the number of records in the file. Alternatively it can be viewed as the record number of the next record past EOF.
FCBEXCLSTAT	This is the exclusive status of the file access. If -1 then the file is being accessed exclusively; otherwise it it the number of semi-exclusive accessors.
FCBEXTMAP	This is the extent map of the file.
FCBEXTSIZE	This is the extent size, in sectors, of the file. All extents in the file, with the exception of the last extent, have this extent size. This is a logical value, and legal values range from 1 to 65535 sectors. This restricts the maximum file size to 2097120 sectors.
FCBFLIM	This is the end-of-space pointer for the file. It is a double word integer representing the maximum number of records in the file.
FCBFOPTIONS	This is the FOPTIONS in effect for the file.
FCBGN	This is the group name of the file. It is eight bytes long with trailing blanks added.
FCBIMAGE	This cell is reserved for use by IMAGE. The data is accessed via special privileged mode calls to FLOCK and FUNLOCK.
FCBLABEL	This is the logical device number and sector number of the file label, which is the same as the first extent descriptor.
FCBLASTEXTSIZE	This is the size, in sectors, of the last extent in the file. If the file has one extent then this is the same as FCBEXTSIZE; if the file has more than one extent then this value may be different from FCBEXTSIZE. Note that this is the size of the last physical extent for the file; it is not the size of the last allocated extent.

FCBLBL	This is the number of user labels allocated for the file. Since each label is a sector long, this is also the number of sectors allocated for user labels.
FCBLBLEOF	This is the end-of-data pointer for the user labels. It is analogous to FCBEOF in that it represents the number of labels written.
FCBLDEV	This is the logical device number of the first extent of the file.
FCBLKST	This is the previous lock state of the file and is derived from the file label. Legal values are: 0 - no accessors 1 - read 2 - write 3 - read/write
FCBNEWFCBV	This is the vector of the new FCB for the file. This is used in conjunction with FCBACB to move the FCB to the system (shared FCB) control block table when the second accessor is established.
FCBNUMEXTS	This is the number of extents, less one, allowed for the file. Note that it is not the number of extents allocated.
FCBOCNT	This is the number of accessors for the file. Alternatively it can be viewed as the number of ACBs created for the file.
FCBOCNTIN	This is the number of accessors that have input access to the file.
FCBOCNTOUT	This is the number of accessors that have output access to the file.
FCBRIN	This is the RIN number used to support dynamic locking (i.e. FLOCK and FUNLOCK) for the file. If there is no dynamic locking then this number is zero.
FCBSECTOFF	This is the sector offset from the file label to the first block of the file. Note that this is not necessarily equal to FCBLBL+1 since an integral number of blocks are allocated for the file and user labels.
FCBSECTPBLK	This is the number of sectors in a block for the file.
FCBSIZE	This is the size, in words, of the complete FCB. It includes extent map.
FCBSUBTYPE	This is the device sub-type number of the first extent of

the file.

FCBUSERLBL	This field describes the user labels for the file. It consists of FCBLBL and FCBLBLEOF, each of which is described separately.

3.3 File Label (FLAB)

The file label is essentially the same as under MPE/20. The only notable differences are:

* Since extents can be on different volumes the high byte of an extent descriptor contains a volume table index.
* Since there can be up to 32 extents the extent map may be longer.
* Since the extents are constrained to a single logical device or to members of a single device class the device specification used at creation is recorded in the label.
* The size of the last extent is no longer calculated each time; instead it is recorded in the label.

The file label has the following format:

Bits 0–7	Bits 8–15	Word
FILE NAME - 1ST CHAR.	FILE NAME - 2ND CHAR.	0
FILE NAME - 3RD CHAR.	FILE NAME - 4TH CHAR.	1
FILE NAME - 5TH CHAR.	FILE NAME - 6TH CHAR.	2
FILE NAME - 7TH CHAR.	FILE NAME - 8TH CHAR.	3
GROUP NAME - 1ST CHAR.	GROUP NAME - 2ND CHAR.	4
GROUP NAME - 3RD CHAR.	GROUP NAME - 4TH CHAR.	5
GROUP NAME - 5TH CHAR.	GROUP NAME - 6TH CHAR.	6
GROUP NAME - 7TH CHAR.	GROUP NAME - 8TH CHAR.	7
ACCOUNT NAME - 1ST CHAR.	ACCOUNT NAME - 2ND CHAR.	8
ACCOUNT NAME - 3RD CHAR.	ACCOUNT NAME - 4TH CHAR.	9
ACCOUNT NAME - 5TH CHAR.	ACCOUNT NAME - 6TH CHAR.	10
ACCOUNT NAME - 7TH CHAR.	ACCOUNT NAME - 8TH CHAR.	11
CREATOR NAME - 1ST CHAR.	CREATOR NAME - 2ND CHAR.	12
CREATOR NAME - 3RD CHAR.	CREATOR NAME - 4TH CHAR.	13

```
|   CREATOR NAME - 5TH CHAR.      |   CREATOR NAME - 6TH CHAR.      | 14
|-----------------------------------------------------------------|
|   CREATOR NAME - 7TH CHAR.      |   CREATOR NAME - 8TH CHAR.      | 15
|-----------------------------------------------------------------|
|      LOCKWORD - 1ST CHAR.       |      LOCKWORD - 2ND CHAR.       | 16
|-----------------------------------------------------------------|
|      LOCKWORD - 3RD CHAR.       |      LOCKWORD - 4TH CHAR.       | 17
|-----------------------------------------------------------------|
|      LOCKWORD - 5TH CHAR.       |      LOCKWORD - 6TH CHAR.       | 18
|-----------------------------------------------------------------|
|      LOCKWORD - 7TH CHAR.       |      LOCKWORD - 8TH CHAR.       | 19
|-----------------------------------------------------------------|
|                                                                 | 20
|                         SECURITY MATRIX                         |
|                                                                 | 21
|-----------------------------------------------------------------|
|                                                           | S   | 22
|-----------------------------------------------------------------|
|                           CREATION DATE                         | 23
|-----------------------------------------------------------------|
|                         LAST ACCESS DATE                        | 24
|-----------------------------------------------------------------|
|                      LAST MODIFICATION DATE                     | 25
|-----------------------------------------------------------------|
|                            FILE CODE                            | 26
|-----------------------------------------------------------------|
|                            FCB VECTOR                           | 27
|-----------------------------------------------------------------|
| S | R | L | X |  SUB-TYPE       |      DISC TYPE        |  R/W  | 28
|-----------------------------------------------------------------|
|  NR. USER LABELS WRITTEN        |   NR. USER LABELS AVAILABLE   | 29
|-----------------------------------------------------------------|
|                                                                 | 30
|                      FILE LIMIT IN BLOCKS                       |
|                                                                 | 31
|-----------------------------------------------------------------|
|                                                                 | 32
|-----------------------------------------------------------------|
|                                                                 | 33
|-----------------------------------------------------------------|
|                             CHECKSUM                            | 34
|-----------------------------------------------------------------|
|                           COLD LOAD ID                          | 35
|-----------------------------------------------------------------|
|                             FOPTIONS                            | 36
|-----------------------------------------------------------------|
|                       RECORD SIZE IN BYTES                      | 37
|-----------------------------------------------------------------|
|                       BLOCK SIZE IN WORDS                       | 38
|-----------------------------------------------------------------|
|                   LAST EXTENT SIZE IN SECTORS                   | 40
|-----------------------------------------------------------------|
```

```
|                       EXTENT SIZE IN SECTORS                      |  41
|-------------------------------------------------------------------|
|                                                                   |  42
|                       END OF DATA POINTER                         |
|                                                                   |  43
|-------------------------------------------------------------------|
|      VOLUME TABLE INDEX           |                               |  44
|-----------------------------------                                |
|                    1ST EXTENT SECTOR NUMBER                       |  45
|-------------------------------------------------------------------|
|                                   .                               |
|                                   .                               |
|                                   .                               |
|-------------------------------------------------------------------|
|      VOLUME TABLE INDEX           |                               |
|-----------------------------------                                |
|                    LAST EXTENT SECTOR NUMBER                      |
|-------------------------------------------------------------------|
|                                   .                               |
|                                   .                               |
|                                   .                               |
|-------------------------------------------------------------------|
|   DEVICE NAME - 1ST CHAR.         |    DEVICE NAME - 2ND CHAR.    |124
|-------------------------------------------------------------------|
|   DEVICE NAME - 3RD CHAR.         |    DEVICE NAME - 4TH CHAR.    |125
|-------------------------------------------------------------------|
|   DEVICE NAME - 5TH CHAR.         |    DEVICE NAME - 6TH CHAR.    |126
|-------------------------------------------------------------------|
|   DEVICE NAME - 7TH CHAR.         |    DEVICE NAME - 8TH CHAR.    |127
---------------------------------------------------------------------
```

In general the following identifiers are used when referring to a file label:

```
DEFINE
FLLOCNAME       = FLAB#,              <<LOCAL FILE NAME>>
FLGRPNAME       = FLAB(4)#,           <<GROUP NAME>>
FLACCTNAME      = FLAB(8)#,           <<ACCOUNT NAME>>
FLUSERID        = FLAB(12)#,          <<CREATOR NAME>>
FLLOCKWORD      = FLAB(16)#,          <<LOCKWORD>>
FLSECMX         = FLABDBL(10)#,       <<SECURITY MATRIX>>
FLSECURE        = FLAB(22).(15:1)#,<<FILE SECURE BIT>>
FLCREATE        = FLAB(23)#,          <<CREATION DATE>>
FLLASTACC       = FLAB(24)#,          <<LAST ACCESS DATE>>
FLLASTMOD       = FLAB(25)#,          <<LAST MODIFICATION DATE>>
FLFILECODE      = FLAB(26)#,          <<FILE CODE>>
FLFCBVECT       = FLAB(27)#,          <<FCB VECTOR>>
FLLOCK          = FLAB(28)#,          <<LOCK BITS, ETC.>>
FLSTORE         = FLAB(28).(0:1)#,<<FILE BEING STORED>>
FLRESTORE       = FLAB(28).(1:1)#,<<FILE BEING RESTORED>>
```

```
FLLOAD          = FLAB(28).(2:1)#,<<FILE LOADED>>
FLEXCL          = FLAB(28).(3:1)#,<<EXCLUSIVE ACCESS>>
FLSR            = FLAB(28).(0:2)#,<<S & R BITS>>
FLSRL           = FLAB(28).(0:3)#,<<S, R & L BITS>>
FLSRLX          = FLAB(28).(0:4)#,<<S, R, L & X BITS>>
FLSUBTYPE       = FLAB(28).(4:4)#,<<DEVICE SUB-TYPE>>
FLDTYPE         = FLAB(28).(8:6)#,<<DEVICE TYPE>>
FLSTATUS        = FLAB(28).(14:2)#,<<WRITE/READ STATUS>>
FLUSERLBL       = FLAB(29)#,      <<USER LABEL INFO>>
FLLBLEOF        = FLAB(29).(0:8)#,<<NR. LABELS WRITTEN>>
FLLBL           = FLAB(29).(8:8)#,<<NR. LABELS AVAILABLE>>
FLFLIM          = FLABDBL(15)#,   <<FILE LIMIT>>
FLCHECKSUM      = FLAB(34)#,      <<LABEL CHECK SUM>>
FLCLID          = FLAB(35)#,      <<COLD LOAD ID>>
FLFOPTIONS      = FLAB(36)#,      <<FOPTIONS>>
FLRECSIZE       = FLAB(37)#,      <<RECORD SIZE>>
FLBLKSIZE       = FLAB(38)#,      <<BLOCK SIZE>>
FLSECTOFF       = FLAB(39).(0:8)#,<<SECTOR OFFSET TO DATA>>
FLNUMEXTS       = FLAB(39).(11:5)#,<<NR. EXTENTS LESS 1>>
FLLASTEXTSIZE   = FLAB(40)#,      <<LAST EXTENT SIZE>>
FLEXTSIZE       = FLAB(41)#,      <<EXTENT SIZE>>
FLEOF           = FLABDBL(21)#,   <<END-OF-DATA POINTER>>
FLLABEL         = FLABDBL(22)#,   <<LABEL VTAB AND SECTOR>>
FLVTAB          = FLAB(44).(0:8)#,<<LABEL VTAB INDEX>>
FLEXTMAP        = FLAB(44)#,      <<EXTENT MAP>>
FLDEVNAME       = FLAB(124)#;     <<DEVICE SPECIFICATION NAME>>
```

The following is an alphabetized list of the above identifiers along with a discussion of their meaning:

FLACCTNAME — This is the account name of the file. It is eight bytes in length with trailing blanks added.

FLBLKSIZE — This is the block size, in sectors, of the file.

FLCHECKSUM — This is the check sum of the file label and is used for error detection. Each time the file label is read from disc the check sum is calculated and compared against the value recorded in the file label. Similarly, each time the file label is written to the disc the check sum is calculated and inserted into the file label.

FLCLID — This is the cold load number in effect the last time that the file was accessed. This should always be the current cold load number. If it is not it means that the system crashed while the file was open and that the data in the file label should be "reset" (principally the FCB vector FLFCBVECT).

FLCREATE — This is the creation date of the file. It is in the format defined by the intrinsic CALENDAR.

FLDEVNAME	This is the FOPEN device specification that was used when the file was created. This information is needed when extents are allocated.
FLDTYPE	This is the device type number of the first extent of the file. See ACBDTYPE for a list of legal values.
FLEOF	This is the end-of-file pointer for the file. It is a double word integer representing the number of records in the file. Alternatively it can be viewed as the record number of the next record past EOF.
FLEXCL	This is the exclusive access flag for the file. If set it means that the file has been opened exclusively by the single accessor. If not set then the file is potentially accessible.
FLEXTMAP	This is the extent map of the file. The number of extents is specified by FLNUMEXTS; a 0D extent descriptor indicates that the extent has not been allocated.
FLEXTSIZE	This is the extent size, in sectors, of the file. All extents in the file, with the exception of the last, have this extent size. This is a logical value, and legal values range from 1 to 65535 sectors. This restricts the maximum file size to 2097120 sectors.
FLFCBVECT	This is the vector of the FCB for the file. If 0 it indicates that the file is not being accessed.
FLFILECODE	This is the file code of the file.
FLFLIM	This is the end-of-space pointer for the file. It is a double word integer representing the maximum number of records in the file.
FLFOPTIONS	This is the FOPTIONS of the file.
FLGRPNAME	This is the group name of the file. It is eight bytes long with trailing blanks added.
FLLABEL	This is the volumt table index and sector number of the file label, which is the same as the first extent descriptor.
FLLASTACC	This is the last access date of the file. It is in the format defined by the intrinsic CALENDAR.
FLLASTMOD	This is the last modification date of the file. It is in the format defined by the intrinsic CALENDAR.
FLLASTEXTSIZE	This is the size, in sectors, of the last extent in the

file. If the file has one extent then this is the same as FLEXTSIZE; if the file has more than one extent then this value may be different from FLEXTSIZE. Note that this is the size of the last physical extent for the file; it is not the size of the last allocated extent.

FLLBL This is the number of user labels allocated for the file. Since each label is a sector long, this is also the number of sectors allocated for user labels.

FLLBLEOF This is the end-of-data pointer for the user labels. It is analogous to FLEOF in that it represents the number of labels written.

FLLOAD This is the loaded bit for the file. If set it means that the file is a loaded program file and cannot be modified except by a privileged accessor.

FLLOCK This is the identifier of the word containing the lock bits, each of which are described separately.

FLLOCKWORD This is the lock word of the file. It is eight bytes long with trailing blanks added. If it is all blanks then the file does not have a lockword.

FLLOCNAME This is the local name of the file. It is eight bytes long with trailing blanks added.

FLNUMEXTS This is the number of extents, less one, allowed for the file. Note that it is not the number of extents allocated. Legal values range from 0 to 31, which corresponds to from 1 to 32 extents.

FLRECSIZE This is the record size, in positive bytes, of the file.

FLRESTORE This is the RESTORE bit for the file. If set it means that the file is being RESTOREd and cannot be modified. The bit is cleared when the RESTORE operation has been completed.

FLSECMX This is the security matrix of the file.

FLSECTOFF This is the sector offset from the file label to the first block of the file. Note that this is not necessarily equal to FLLBL+1 since an integral number of blocks are allocated for the file and user labels.

FLSECURE This is the file security enforcement flag for the file. If not set then the file has been RELEASEd and the security matrix FLSECMX should be ignored. If set then secured as specified by the security matrix.

SR — This is the STORE and RESTORE flags for the file, each of which is described separately.

LSRL — This is the STORE, RESTORE and LOADED flags for the file, each of which is described separately.

LSRLX — This is the STORE, RESTORE, LOADED and exclusive flags for the file, each of which is described separately.

LSTATUS — This is the read/write status of the file. Legal values are:

- 0 - no accessors
- 1 - read
- 2 - write
- 3 - read/write

FLSTORE — This is the STORE flag for the file. If set it means that the file is being STOREd and cannot be modified.

FLSUBTYPE — This is the device sub-type number of the first extent of the file.

FLUSERID — This is the creating user name of the file. It is eight bytes long with trailing blanks added.

FLUSERLBL — This field describes the user labels of the file. It consists of FLLBL and FLLBLEOF, each of which is described separately.

FLVTAB — This is the volume table index of the first extent of the file.

3.2 File Control Block Table (CBTAB)

A file control block table can be located in two places: (a) as a sub-part of the PXFILE area; (b) totally contained in a data segment. The purpose of a file control block table is to facilitate the management of (you guessed it) control blocks.

The overall structure of a control block table is:

```
 -------------------------
|                         |
|        OVERHEAD         |       (fixed)
|                         |
|-------------------------|
|                         |
|      VECTOR TABLE       |       (variable)
|                         |
|-------------------------|
|                         |
|                         |
|                         |
|     CONTROL BLOCK       |       (variable)
|         AREA            |
|                         |
|                         |
|                         |
 -------------------------
```

3.2.1 Overhead

The part labeled OVERHEAD contains information that is pertainent to the entire table.

```
 0                                                              15
 ------------------------------------------------------------------
 |                     TABLE SIZE IN WORDS                         |   0
 |----------------------------------------------------------------|
 |                 DST NUMBER CONTAINING TABLE                     |   1
 |----------------------------------------------------------------|
 |                 VECTOR TABLE SIZE IN WORDS                      |   2
 |----------------------------------------------------------------|
 |                          LOCK WORD                              |   3
 |----------------------------------------------------------------|
 |                       IMPEDED QUEUE                             |   4
 ------------------------------------------------------------------
```

In general the following identifiers are used when referring to this part of a control block table:

```
DEFINE
CBTSIZE          = CBTAB#,          <<TABLE SIZE>>
CBTDSTX          = CBTAB(1)#,       <<DST NUMBER>>
CBTVTSIZE        = CBTAB(2)#,       <<VECTOR TABLE SIZE>>
CBTLOCK          = CBTAB(3)#,       <<LOCK WORD>>
CBTQUEUE         = CBTAB(4)#;       <<IMPEDED QUEUE>>
```

The following is an alphabetized list of the above identifiers along with a discussion of their meaning.

CBTDSTX — This is the DST number of the data segment that contains the control block table. If the table is contained in a stack, i.e. in the PXFILE area, then this is the DST number of the stack and not 0.

CBTLOCK — This is the lock word for the table and has the same format as the lock word for a control block in the table. The table is locked, thus insuring exclusive access, whenever (a) a control block is being created or destroyed; (b) a control block is being locked or unlocked. Note that the table is not locked after a control block has been locked.

CBTQUEUE — This is the impeded queue for the table and has the same format as the impeded queue for a control block in the table. There is no second impeded queue because that is is a facility which is used exclusively for BREAK requests against the ACB for $STDIN/$STDLIST.

:BTSIZE — This is the size in words of the table. It is initialized when the table is created and changed when the table is expanded. Note that at present a table is never contracted, even though this is possible.

CBTVTSIZE — This is the size, in words, of the vector table area in the control block table. Note that this is the length of the vector table area and does not reflect the number of entries used or unused.

3.2.2 Vector Table

The part labeled VECTOR TABLE contains information used to locate and lock/unlock control blocks in the control block table.

The overall structure of the vector table is:

```
 -------------------------
|                         |
|         ENTRY 0         |
|                         |
|-------------------------|
|                         |
|            .            |
|            .            |
|            .            |
|                         |
|-------------------------|
|                         |
|         ENTRY N         |
|                         |
 -------------------------
```

where N = (CBTVTSIZE/4)-1. A vector table is restricted to no more than 64 entries; this is due to the addressing limitation inherent in the control block vector format.

The general structure of a vector table entry is:

```
 0                                                       15
 -----------------------------------------------------------
|            CONTROL BLOCK ADDRESS   REL. TO Beginning ACBT |  0
|-----------------------------------------------------------|
|                  CONTROL WORD                             |  1
|-----------------------------------------------------------|
|          HIGH PRIORITY IMPEDED QUEUE  e.g. AF on console  |  2
|-----------------------------------------------------------|
|          LOW PRIORITY IMPEDED QUEUE   e.g. BREAK          |  3
 -----------------------------------------------------------
```

In general the following identifiers are used when referring to this part of a control block table:

```
DEFINE
VTADR              = VT#,              <<CONTROL BLOCK ADDRESS>>
```

```
VTCONTROL        = VT(1)#,          <<CONTROL WORD>>
VTQUEUE          = VT(2)#,          <<HIGH PRIORITY IMPEDED QUEUE>>
VTSAVEDQUEUE     = VT(3)#;          <<LOW PRIORITY IMPEDED QUEUE>>
```

The following is an alphabetized list of the above identifiers along with a discussion of their meaning.

VTADR — This is the table relative address of the control block associated with the vector table entry. It is a word displacement from the beginning of the control block table.

VTCONTROL — This is the control word used to coordinate the access to the control block. It contains a bit which indicates that the control block is being accessed, and therefore "locked", and a byte which contains the PIN of the process which has exclusive access to the control block.

VTQUEUE — This is a byte pair of PINs that are the head and tail of the impeded queue of processes waiting for access to the control block. Processes are impeded and unimpeded by the file system using the normal mechanisms available under MPE.

VTSAVEDQUEUE — This is a byte pair of PINs and has the same format as VTQUEUE. The only time this word is used is when the control block is in BREAK mode, which can only happen to an ACB corresponding to $STDIN/$STDLIST. It is used to save the current VTQUEUE when the control block goes into BREAK mode and to restore VTQUEUE when the control block goes back into non-BREAK mode.

More specifically the last three words of a vector table entry compose a control sub-block that is used to coordinate the access to a general control block in the control block area of the control block table.

The structure of the vector table entry control sub-block is:

LOCK BIT   BREAK BIT   by the same process whose PIN is   control

```
  0   1   2                     7 8                              15
 -------------------------------------------------------------------
 | L | B |      LOCK COUNT       |           LOCK PIN             |  0
 |-----------------------------------------------------------------|
 |    HIGH PRIORITY TAIL PIN     |    HIGH PRIORITY HEAD PIN      |  1
 |-----------------------------------------------------------------|
 |     LOW PRIORITY TAIL PIN     |     LOW PRIORITY HEAD PIN      |  2
 -------------------------------------------------------------------
```

In general the following identifiers are used when referring to this part of a vector table entry:

```
DEFINE
CBLCONTROL        = CBL#,             <<CONTROL WORD>>
CBLLOCK           = CBL.(0:1)#,       <<LOCK BIT>>
CBLBREAK          = CBL.(1:1)#,       <<BREAK BIT>>
CBLCOUNT          = CBL.(2:6)#,       <<LOCK COUNT>>
CBLPIN            = CBL.(8:8)#,       <<PIN HOLDING LOCK>>
CBLQUEUE          = CBL(1)#,          <<HIGH PRIORITY IMPEDED QUEUE>>
CBLTAIL           = CBL(1).(0:8)#,    <<HIGH PRIORITY TAIL PIN>>
CBLHEAD           = CBL(1).(8:8)#,    <<HIGH PRIORITY HEAD PIN>>
CBLSAVEDQUEUE     = CBL(2)#,          <<LOW PRIORITY IMPEDED QUEUE>>
CBLSAVEDTAIL      = CBL(2).(0:8)#,    <<LOW PRIORITY TAIL PIN>>
CBLSAVEDHEAD      = CBL(2).(8:8)#;    <<LOW PRIORITY HEAD PIN>>
```

The following is an alphabetized list of the above identifiers along with a discussion of their meaning.

CBLBREAK	This is the BREAK bit and is used only for the ACB corresponding to $STDIN/$SSDTLIST.
CBLCONTROL	This identifier is used when referring to the first word of the vector table control sub-block.
CBLCOUNT	This is a count of the number of times that the control block is locked by CBLPIN. It is 0 if the control block is not locked and is greater than 0 if the control block is locked.
CBLHEAD	This is the PIN of the process at the head of the high priority impeded queue.
CBLLOCK	This is the lock bit for a control block. If it is 0 then the control block is not locked; if it is 1 then the control block is locked.
CBLPIN	This is the PIN of the process which has locked the control block and has exclusive access to it. If the control block is not locked then this field is 0.
CBLQUEUE	This identifier is used when referring to the second word of the vector table control sub-block.
CBLSAVEDHEAD	This is the PIN of the process at the head of the low priority impeded queue.
CBLSAVEDQUEUE	This identifier is used when referring to the third word of the vector table control sub-block.
CBLSAVEDTAIL	This is the PIN of the process at the tail of the low priority impeded queue.
CBLTAIL	This is the PIN of the process at the tail of the high

priority impeded queue.

3.2.3 Control Block Area

The part labeled CONTROL BLOCK AREA contains the control blocks used by the file system.

All control blocks have the same overall structure:

```
  0   1   2                                            15
 ----------------------------------------------------------
 | TYPE  |                      SIZE                     |  0
 |--------------------------------------------------------|
 |                                                        |  1
 |                                                        |
 |                       DATA                             |
 |                                                        |
 |                                                        |  n
 ----------------------------------------------------------
```

where n = SIZE-1. Since all control blocks have a common structure this facilitates storage management within the control block table.

In general the following identifiers are used when referring to this part of a control block table:

```
DEFINE
CBDESCRIP        = CB#,            <<CONTROL BLOCK DESCRIPTOR>>
CBTYPE           = CB.(0:2)#,      <<CONTROL BLOCK TYPE NR.>>
CBSIZE           = CB.(2:14)#;     <<CONTROL BLOCK SIZE>>
```

The following is an alphabetized list of the above identifiers along with a discussion of their meaning.

CBDESCRIP — This is the first word of a control block; the format is common for all control blocks.

CBSIZE — This is the size (in words) of the control block. The size includes the descriptor word.

CBTYPE — This is the type number of the control block. At present there are only four types of control blocks:

[redacted]

When a control block table is created the initial control block area is completely allocated by a single control block of type garbage. When space is requested for a new control block the control block area is scanned (using a first fit algorithm) for a garbage control block that is as large as the size requested. The space for the new control block is taken from this garbage control block and the space remaining becomes the new garbage control block size.

When space is returned it becomes a new garbage control block. To prevent fragmentation the new garbage control block is combined with either of the two neighboring control blocks if they are of type garbage.

If space is requested and no garbage control block is large enough to contain the new control block then the control block area and control block table is expanded by a sufficient amount.

FMAVT

FMAVTDST = 44
FMAVTSIR = 16

H. Hearn
2-5-75.

SEG SIZE
ENTRY SIZE (=4)
MAX SEG ~~SIZE~~
⋮
Ø
⋮

FREE

1	Ø	G		JIT
LOG DEV				LABEL....
DISK ADDRESS				
PAGE VECTOR				

DISK (+ SPOOLFILE)

1	1	G		JIT
LOG DEV				
Ø				
PAGE VECTOR				

DEVICE

Entry for each multi-access file
currently opened. Searched at FOPEN
time [illegible] LDEV#.

Memory Management

INTRODUCTION AND DESIGN GOALS

CONCEPTS WHICH COALESCED INTO THE MPE30 MEMORY MANAGEMENT DESIGN WERE A FALLOUT OF MPE20 PERFORMANCE EVALUATION AND UNADULTERATED HINDSIGHT. THE FINAL DESIGN WAS SIMULATED TO DETERMINE IT'S FEASIBILITY. A DESCRIPTION OF THE SIMULATOR CAN BE FOUND IN ATTACHMENT 1.

THE REPLACEMENT ALGORITHM IMPLEMENTED FOR MPE20 SELECTED SEGMENTS BASED ON LEAST RECENTLY USED CRITERIA WITHOUT CONSIDERING SPECIFIC PROCESS LOCALITY. THE ALGORITHM IS SATISFACTORY WHEN USERS ARE SHARING A COMMON SUBSYSTEM, BUT CAN CAUSE ERRATIC PROCESS SEGMENT FAULT BEHAVIOR WHEN UNIQUE SUBSYSTEMS ARE COMPETING FOR THE MEMORY RESOURCE. AS A RESULT, PROCESS WORKING SETS HAVE BEEN IMPLEMENTED TO CONTROL REPLACEMENT ON A PROCESS BASIS.

EXTERNAL FRAGMENTATION HAD BEEN A LESS SERIOUS PROBLEM THAN ANTICIPATED, BUT IT WAS A GOAL TO REDUCE THE IMPACT EVEN FURTHER. VARIOUS COMPRESSION ALGORITHMS WERE SIMULATED AND A FREE SPACE CONCATENATION ALGORITHM, I CALL ,LOCAL COMPRESSION WAS IMPLEMENTED.

CONCURRENT EXECUTION OF MEMORY MANAGEMENT FUNCTIONS INDIRECTLY FORCED THE USE OF WAIT RATHER THAN NOWAIT I/O IN PROCESSING MEMORY ALLOCATION REQUESTS. THIS WAS PRIMARILY A DESIGN FAULT IN THE PROCESS MECHANISM WHICH DISALLOWED THE QUEUEING OF ALL MAKEPRESENT REQUESTS THROUGH THE MAP REQUEST QUEUE. THIS HAS BEEN CORRECTED ALLOWING CONCURRENT EXECUTION OF I/O AND CPU PROCESSING OF A MEMORY MANAGEMENT REQUEST.

IN SUMMATION, THE DESIGN GOALS FOR THE MPE30 MEMORY MANAGEMENT FUNCTION WERE:

1. REDUCE THE IMPACT OF EXTERNAL FRAGMENTATION BY APPLYING A LOW OVERHEAD FREE SPACE COMPRESSION ALGORITHM.

2. REDUCE REAL TIME DELAYS IN MAIN MEMORY STORAGE ALLOCATION BY ACHIEVING AS MUCH CONCURRENT I/O WITH CPU PROCESSING AS POSSIBLE.

3. IMPLEMENTATION OF A REPLACEMENT ALGORITHM WHICH IS BASED ON PROCESS LOCALITY.

4. SIMPLFY THE HANDLING OF RACE CONDITIONS BY PROCESSING ALL MEMORY ALLOCATION REQUESTS ON A FIFO BASIS.

MEMORY MANAGEMENT MODULES

THE MEMORY MANAGEMENT FUNCTIONS ARE DISTRIBUTED AMONG THREE SYSTEM SL SEGMENTS, MMCORER, MMDISKR AND DATASEG. THE SEGMENT MMCORER IS THE ONLY ONE OF THE THREE WHICH IS CORE RESIDENT.

MMCORER CONTAINS ALL THE PROCEDURES WHICH ARE ASSOCIATED WITH STORAGE ALLOCATION, DEALLOCATION AND WORKING SET MAINTAINENCE.

MMDISKR CONTAINS PROCEDURES FOR ALLOCATING AND DEALLOCATING CST, CST EXTENSION, DST, PCB AND WSTAB ENTRIES. IN ADDITION, THE PROCEDURES GETSTACK, GETDATASEG, ALTDSEGSIZE, LOCKSEG AND UNLOCKSEG RESIDE IN THIS SEGMENT.

DATASEG CONTAINS PROCEDURES FOR PERFORMING STACK SIZE CHANGES. IN PARTICULAR, DLSIZE, ZSIZE, ALTPXFILESIZE, GETPXSEG AND STACKOVERFLOW. IN ADDITION, DATASEG CONTAINS FUNCTIONS FOR ALLOCATING AND OPERATING ON EXTRA DATA SEGMENTS FOR USERS WITH EXTRA DATA SEGMENT CAPABILTY.

THE MEMORY ALLOCATION MANAGER-MAM

MEMORY MANAGEMENT REQUESTS WHICH SPECIFY THE ALLOCATION OF MAIN MEMORY, THE EXPANSION OF DATA SEGMENTS OR FORCED OVERLAYS ARE PROCESSED BY THE MEMORY ALLOCATION MANAGER, MAM. THIS PROCESS RESIDES IN THE CORE RESIDENT SYSTEM SEGMENT MMCORER. A REQUEST FOR MAM SERVICE IS INITIATED BY LINKING A REQUEST INTO THE MAM REQUEST QUEUE. A REQUEST IS CONSTRUCTED BY CALLING THE PROCEDURE BUILDGENTRY WHICH CALLS LINKSINGLEQ OR LINKDOUBLEQ TO LINK THE REQUEST INTO THE SPECIFIED QUEUE.

PROCEDURES WHICH INITIATE MAM REQUESTS ARE MAKEPRESENT, LOCKSEG,PREP AND EXPANDREG. THE LATER IS CALLED BY THE PROCEDURES INITIATING A STACK OR DATA SEGMENT EXPANSION.

ATTACHMENT 4 SHOWS THE CONTROL FLOW OF REQUESTS INTO THE MAM REQUEST QUEUE. NOTE THAT A REQUEST MAY BE DEFERED TEMPORARILY TO MAINTAIN A STRICT FIFO REQUEST ORDERING. THIS IS COVERED IN MORE DETAIL IN THE SECTION ON ACCESS CONTENTION.

THERE ARE CURRENTLY TWELVE REQUEST TYPES WHICH MAM WILL PROCESS. THE TWELVE ARE:

REQUEST TYPE 0-

INITIATED BY CALLING MAKEPRESENT TO PROCESS A CODE OR DATA SEGMENT ABSENCE TRAP.

REQUEST TYPE 1-

INITIATED BY CALLING MAKEPRESENT FROM THE I/O SYSTEM TO MAKE PRESENT AN ABSENT I/O DRIVER OR I/O BUFFER.

REQUEST TYPE 2-

INITIATED BY CALLING MAKEPRESENT FROM THE PROCEDURE EXCHANGEDB.

REQUEST TYPE 3-

INITIATED BY CALLING EXPANDREG FROM THE PROCEDURE DLSIZE.

REQUEST TYPE 4-

INITIATED BY CALLING EXPANDREG FROM THE PROCEDURE ZSIZE.

REQUEST TYPE 5-

INITIATED BY CALLING EXPANDREQ FROM THE PROCEDURE ALTDSEGSIZE.

REQUEST TYPE 6-

INITIATED BY CALLING EXPANDREG FROM THE PROCEDURE STACKOVERFLOW.

REQUEST TYPE 7-

THE MEMORY ALLOCATION MANAGER-MAM

INITIATED BY CALLING EXPANDREG FROM THE PROCEDURE GETPXSEG.

REQUEST TYPE 8-
INITIATED BY CALLING EXPANDREG FROM THE PROCEDURE ALTPXFILESIZE.

REQUEST TYPES 9-11-
THESE ARE SPECIAL REQUESTS REQUIRED BY THE LOCKSEG MECHANISM. A COMPLETE DEFINITION OF THEIR USE IS DESCRIBED IN THE SECTION ON SEGMENT LOCKING.

1. -QUEUES ACCESSED BY MAM-

1. AREQ-THE MAM REQUEST QUEUE-
THIS TABLE CONTAINS REQUESTS INITIATED BY MAKEPRESENT CALLS.STACK SIZE CHANGES.PROCESS PREPARATIONS.EXTRA DATA SEGMENT EXPANSIONS.PXFIXED OR PXFILE AREA EXPANSIONS, AND FORCED ABSENCES REQUIRED BY THE LOCKSEG MECHANISM.

2. OLSQ-THE OVERLAY SELECTION QUEUE-
SEGMENTS ARE PLACED ON THE OVERLAY SELECTION QUEUE BY CALLING THE PROCEDURES PLACEONOLST OR RELDS. RELDS IS CALLED BY THE PROCEDURE EXCHANGEDB WHEN DB IS TO BE SWITCHED AWAY FROM THE CURRENT DATA SEGMENT TO THE STACK OR ANOTHER DATA SEGMENT. PLACEONOLST IS CALLED IN CONJUNCTION WITH WORKING SET REDUCTION OR WHEN A PROCESS IS DISCARDED. A SEGMENT IS MARKED ABSENT WHEN PLACED ON THE QUEUE BUT THE ALLOCATED MEMORY IS NOT RELEASED UNTIL THE SEGMENT IS ACTUALLY OVERLAYED. IF A SEGMENT ON THE QUEUE IS REFERENCED , THE ENTRY WILL BE REMOVED FROM THE QUEUE AND THE ASSOCIATED SEGMENT MARKED PRESENT.

3. IOCQ-THE MEMORY MANAGEMENT I/O COMPLETION QUEUE-
ENTRIES ARE PLACED IN THIS QUEUE FOLLOWING COMPLETION OF AN I/O OPERATION REQUIRED BY MAM.

4. DREQ-THE DEFERED REQUEST QUEUE-
OCCASSIONALLY, IT IS NECESSARY TO DEFER THE QUEUEING OF A REQUEST INTO THE AREQ TO PRESERVE A FIFO ORDERING. THIS OCCURS WHEN AN INPROGRESS QUEUEING OPERATION TO AREQ IS INTERRUPTED BY A CALL TO MAKEPRESENT BY THE I/O SYSTEM FOR THE SAME SEGMENT. IN THIS CASE, THE I/O SYSTEM CALL TO MAKEPRESENT WILL RESULT IN THE REQUEST BEING LINKED INTO THE DEFERED REQUEST QUEUE. WHEN THE INTERRUPTED PROCESS IS REACTIVATED, IT WILL COMPLETE ITS QUEUEING SEQUENCE AND LINK ALL DEFERED REQUESTS

FOR THE SEGMENT INTO THE MAM REQUEST QUEUE.

5. LOKQ-THE LOCK SEGMENT REQUEST QUEUE-
WHEN A SEGMENT IS IN THE STATE OF BEING LOCKED, OTHER PROCESSES MAY REQUIRE ITS PRESENCE AND INITIATE MAKEPRESENT CALLS. THESE REQUESTS WILL BE LINKED INTO THE LOKQ QUEUE. WHEN THE SEGMENT IS LOCKED, THEY WILL BE LINKED INTO THE MAM REQUEST QUEUE. IF THE DISPATCHER REQUESTS THE PREPARATION OF A PROCESS REQUIRING A SEGMENT WHICH IS IN THE PROCESS OF BEING LOCKED, THE PROCESS WILL BE IMPEDED UNTIL THE LOCK IS COMPLETED.

II. -MAM CODE DIVISIONS-
THE MAM CODING IS SUBDIVIDED INTO SEVEN AREAS:

1. REQUEST DECODING AND VALIDITY CHECKING.

2. FREE SPACE ALLOCATION.

3. THE SELECTION AND OVERLAY OF SEGMENTS AND THE HANDLING OF NO-MEM CONDITIONS.

4. REQUEST COMPLETION.

5. PROCESSING MAMIC COMPLETIONS.

6. A SPECIAL SECTION FOR PROCESSING LOCKSEG REQUESTS.

7. PROCESS PREPARATION.

A.) SEGMENT OVERLAY
OCCASSIONALLY, IT IS NECESSARY TO OVERLAY A SEGMENT IN MAIN MEMORY TO FREE UP SUFFICIENT SPACE FOR A PENDING AREQ REQUEST. WHEN AN OVERLAY IS REQUIRED, THE SEGMENT IS SELECTED FROM THE OVERLAY SELECTION QUEUE. IF THE QUEUE IS EMPTY, THE PROCEDURE DISCARD IS CALLED. THE DISCARD FUNCTION WILL RETURN AN INDEX TO A PROCESS WHICH MAY HAVE ITS STACK, EXCHANGECB SEGMENT AND WORKING SET PLACED ON THE OVERLAY SELECTION QUEUE. IF NO PROCESS CAN BE DISCARDED, THE WORKING SET BELONGING TO THE REQUESTOR PROCESS WILL BE DIMINISHED UNTIL SPACE IS ALLOCATED OR A NO-MEM CONDITION OCCURS.

B.) PROCESS PREPARATION

THE DISPATCHER REQUESTS PROCESS PREPARATION BY INSERTING A PROCESS POINTER IN THE MAM PREP LOCATION %1031. WHEN CALLED, THE PROCEDURE PREP WILL CREATE ONE TO FOUR AREQ REQUESTS. IF PCBI IS A PROCESS POINTER, THEN REQUESTS ARE LINKED IN THE FOLLOWING ORDER:

1. THE PROCESS STACK. ENTRY NUMBER = PCBI(3).(1:10).

2. THE AREQ ENTRY LINKED THROUGH PCBI(PRINX). WHEN AN ABSENCE TRAP OCCURS, MAKEPRESENT IS CALLED AND AN MTAB ENTRY IS CONSTRUCTED WHICH HAS A FORMAT IDENTICAL TO THAT OF A MAM REQUEST. THE INDEX RETURNED FROM THE BUILDQENTRY CALL IS SAVED IN PCBI(PRINX). THE ONLY WAY THIS REQUEST CAN BE SERVICED IS BY THE DISPATCHER REQUESTING THE ASSOCIATED PROCESS TO BE PREPARED. THIS ALLOWS THE DISPATCHER GREATER CONTROL OVER PROCESS SELECTION AND MAIN MEMORY ALLOCATION. WHEN THE PROCESS IS PREP'ED THE ENTRY WILL BE LINKED INTO THE MAM REQUEST QUEUE.

3. THE CODE SEGMENT SPECIFIED IN PCBI(7).(0:8). A STACK SEARCH IS PERFORMED BY THE DISPATCHER WHENEVER A PROCESS STOPS RUNNING. THE CST ENTRY NUMBER IN THE ENVIRONMENT NEAREST THE TOP OF THE STACK IS SAVED IN PCBI(7).(0:8).

4. AN EXCHANGEDB ENTRY IN PCBI(2).(1:10).

THE DISPATCHER WILL BE NOTIFIED OF A PREPARATION COMPLETION WHEN ALL QUEUED REQUESTS GENERATED BY THE PREP HAVE BEEN ALLOCATED SPACE AND MARKED PRESENT.

C.) PROCESSING A NO-MEM CONDITION

A NO-MEM OCCURS WHEN THERE IS NO FREE AREA OF SUFFICIENT SIZE TO SATISFY THE REQUEST AND ALL ATTEMPTS TO FREE UP ADDITIONAL SPACE HAVE FAILED. NO-MEM'S ARE HANDLED IN TWO WAYS:

1) IF THE REQUEST IS FOR A STACK EXPANSION, THE MAM ERROR FLAG IN PCBI(2) IS SET. SINCE THE STACK OF THE CALLER WAS INITIALLY OVERLAYED, WE CAN BE SURE OF ALLOCATING AT LEAST THE INITIAL AMOUNT OF MEMORY. THE AREA IS REALLOCATED, AND CONTROL IS RETURNED TO THE CALLER. EXPANDREQ WILL RESET THE MAM ERROR FLAG, AND IF IT WAS ON, A CCG CONDITION CODE WILL BE RETURNED TO THE FUNCTION CALLING EXPANDREG. AT THIS POINT, THE

ORIGINATING FUNCTION WILL DETERMINE WHAT ACTION TO TAKE. THE NORMAL ACTION IS TO PASS ON A CONDITION CODE INDICATING FAILURE. IF THE ORIGINATOR WAS STACKOVERFLOW, A SUDDENDEATH IS INVOKED.

2) IN ALL OTHER CASES, MAM'DONE IS CALLED WITH AN ERROR CODE OF THREE. A COUNT OF SUCCESSIVE NO-MEM FAILURES IS MAINTAINED IN THE PROCESS PCB. ANY SUCCESSFUL ALLOCATION WILL RESULT IN THE FIELD BEING ZEROED. IF THE COUNT REACHES FIVE, A SUDDENDEATH IS INVOKED-AT LEAST FOR NOW.

THE NO-MEM QUEUE, NOMQ, HAS BEEN SET ASIDE TO CATALOGUE THE PROCESSES WHICH FAIL ON A NO-MEM CONDITION. EVENTUALLY, AN ALGORITHM WILL BE DESIGNED TO SEARCH THIS QUEUE AND RETRY THE ALLOCATION AT A MORE APPROPRIATE TIME.

D.) REQUEST DECODING AND VALIDITY CHECKING

A MAM REQUEST WILL BE DECODED TO INITIALIZE THE FOLLOWING Q-RELATIVE PARAMETERS:

- LGTH - THE VALUE OF THE SECOND OPTIONAL PARAMETER.
- CLABEL - A CST-RELATIVE INDEX OR CLABEL.
- DONE - THE VALUE OF THE FLAGS PARAMETER. THIS PARAMETER CONTAINS THE PIN,REQUEST TYPE RT AND STATE FIELDS U,F AND C.
- INC - THE VALUE OF THE FIRST OPTIONAL PARAMETER.
- INX - CST-RELATIVE INDEX TO CST OR DST ENTRY.
- PINX - PCB-RELATIVE INDEX TO REQUESTORS PROCESS.
- PROC - SYSDB-RELATIVE POINTER TO REQUESTORS PROCESS.
- WS - SYSDB-RELATIVE POINTER TO WORKING SET.
- REF - THE REFERENCE COUNTER VALUE TO BE ASSIGNED THE REQUEST.
- RT' - A REMAPPING OF THE CURRENT REQUEST TYPE.
- RSIZE - THE SIZE OF THE AREA REQUIRED.

IF THE SEGMENT IS PRESENT OR ON THE OVERLAY SELECTION QUEUE, THE REQUEST IS QUICKLY COMPLETED. MULTIPLE REQUESTS FOR THE SAME SEGMENT MAY BE PENDING AT ANY TIME. THE REQUEST SERVICED FIRST MAKES THE SEGMENT PRESENT.

A SEGMENT SIZE OF ZERO INDICATES A REQUEST FOR AN INVALID ENTRY. IT HAS EITHER BEEN CLOBBERED, IS NOT ASSIGNED, OR IS SIMPLY OUT OF ANY REASONABLE RANGE. IF THE REQUEST SPECIFIED A DATA SEGMENT, THE SYSTEM IS KILLED BY CALLING THE PROCEDURE SUDDENDEATH. IF THE REQUEST IS FOR AN INVALID CODE SEGMENT ENTRY, AN ERROR CODE OF 2 IS PASSED TO THE PROCEDURE MAM'DONE INDICATING THE FAILURE. WHEN THE REQUESTORS PROCESS IS RUN AGAIN, AN ABSENCE TRAP WILL OCCUR. MAKEPRESENT WILL DETERMINE THE FORM OF THE INVALID ENTRY AND ABORT THE PROCESS WITH THE APPROPRIATE MESSAGE. IT NORMALLY INDICATES THAT THE USER CLOBBERED HIS OWN STACK PRIOR TO STOPPING. WHEN THE DISPATCHER PERFORMED A STACK SEARCH TO LOCATE THE CST ENTRY NUMBER IN THE TOP MOST ENVIRONMENT, HE PICKED UP A GARBAGE VALUE. UNFORTUNATELY, THIS VALUE IS PLACED INTO THE HIGH ORDER BYTE OF PCBI(7). WHEN THE PROCESS WAS SUBSEQUENTLY PREPARED, A REQUEST FOR THE INVALID ENTRY WAS QUEUED.

IF THE REQUEST IS FOR A SEGMENT LOCK, SPECIAL ACTION MUST BE TAKEN. SEE THE SECTION ON SEGMENT LOCKING FOR FURTHER INFORMATION.

IF THE REQUEST IS FOR A STACK OR EXTRA DATA SEGMENT EXPANSION, AN OVERLAY IS INITIATED TO UPDATE THE VDS IMAGE. THE CURRENT SPACE WILL BE LINKED INTO THE FREE SPACE LIST PRIOR TO ALLOCATING A LARGER AREA.

IF THE SEGMENT IS IN THE PROCESS OF BEING LOCKED BY ANOTHER PROCESS, THE PROCESS ASSOCIATED WITH THE CURRENT REQUEST IS IMPEDED. THE IMPEDENCE WILL BE REMOVED WHEN THE SEGMENT IS LOCKED.

THE LAST PROCESSING DONE IN THIS SECTION IS TO UPDATE THE REFERENCE COUNTERS OF THE ASSIGNED WORKING SET AND PERFORM ANY INDICATED WORKING SET REDUCTIONS.

E.) FREE SPACE ALLOCATION

THE INITIAL LINK LIST SEARCH FOR FREE SPACE WILL SCAN THE ENTIRE FREE LIST. IF THIS SEARCH FAILS, SUBSEQUENT SEARCHES WILL ONLY EXAMINE THE HEAD FREE ENTRY SINCE AREAS RETURNED TO THE FREE LIST ARE LINKED TO THE LIST HEAD. THEREFORE, IF THE HEAD ENTRY ISN'T LARGE ENOUGH, NEITHER WILL ANY OTHER FREE AREA.

IF SPACE IS FOUND, ANY REMAINDER EXCEEDING 135 WORDS WILL BE LINKED BACK INTO THE FREE LIST. THE ALLOCATED AREA IS THEN

RESERVED IN MEMORY BY SETTING THE SIGN BIT OF THE FIRST AND LAST WORD OF THE AREA. THE HEAD AND TAIL LINKS ARE ALSO INITIALIZED AT THIS TIME.

WORKING SET DESCRIPTION

THE MOST FUNDAMENTAL CHANGE IN THE MEMORY MANAGEMENT IMPLEMENTATION WAS THE INTRODUCTION OF WORKING SETS. THE REPLACEMENT ALGORITHM USED IN CONJUNCTION WITH THE WORKING SETS IS BASED ON A PAPER BY WESLEY W. CHU AND HOLGER OPDERBECK TITLED "THE PAGE FAULT FREQUENCY REPLACEMENT ALGORITHM". SEE APPENDIX 2.

THE WORKING SET WS(T,TAU) AT A GIVEN TIME T IS THE SET OF DISTINCT SEGMENTS REFERENCED IN THE PROCESS (OR VIRTUAL) TIME INTERVAL (T-TAU+1,T). WHEN I REFER TO PROCESS LOCALITY IN THE TEXT, I GENERALLY IMPLY CODE SEGMENT LOCALITY. A STACK WILL NEVER BE REPRESENTED IN A WORKING SET, THOUGH EXTRA DATA SEGMENTS MAY.

ONE MAJOR DESIGN PROBLEM WAS THE HANDLING OF SHARED SEGMENTS. IT WAS ESSENTIAL THAT THE IMPLEMENTATION SATISFY THE FOLLOWING CRITERIA:

1. CPU OVERHEAD MUST BE AS LOW AS POSSIBLE.
2. THE SIZE AND NUMBER OF TABLES IN CORE RESIDENT MEMORY SHOULD BE KEPT TO A MINIMUM.
3. THE RESULTS SHOULD BE PREDICTABLE AND REPEATABLE.

THE SCHEME ADOPTED ALLOWS PROCESSES ACCESSING THE SAME PROGRAM TO SHARE A COMMON WORKING SET. THE WORKING SET OF A PROCESS IS ACCESSED VIA A POINTER IN IT'S PCB. FOR EXAMPLE, ALL SESSIONS USING THE TEXT EDITOR WILL HAVE THE SAME WORKING SET POINTER. PROCESS AND PROGRAM CODE LOCALITY WILL BE THE SAME IF ONLY ONE USER IS RUNNING THE PROGRAM. IF WE HAVE SUFFICIENT MEMORY, THE PROCESS LOCALITIES WILL NOT CONFLICT WITH ONE ANOTHER SINCE ALL ACCESSED PROGRAM SEGMENTS WILL REMAIN IN MAIN MEMORY. THE ONLY ADDITIONAL OVERHEAD WILL BE CAUSED BY THE SHUFFLING OF SEGMENTS IN AND OUT OF THE WORKING SET. WHEN THE EFFECTIVE MEMORY SIZE IS REDUCED, DUE TO A HEAVY USER LOAD OR BECAUSE PHYSICAL MEMORY IS SMALL, WE MUST EXAMINE THE EFFECT ON UNIQUE PROCESS LOCALITIES CAUSED BY THEIR SHARING OF THE WORKING SET. IF WE DEFINE "A" TO BE THE INTERSECTION OF THE SEGMENTS SHARED IN COMMON BY THE PROCESSES RUNNING PROGRAM P, WE MUST EXAMINE "B", THE COMPLIMENT OF "A", TO DETERMINE THE EFFECT OF THE ALGORITHM ON SYSTEM PERFORMANCE. THIS IS THE SET OF SEGMENTS THAT WILL NORMALLY BE SELECTED FROM THE WORKING SET WHEN IT MUST BE REDUCED IN SIZE, SINCE THEY WILL ,IN GENERAL, BE REFERENCED LESS FREQUENTLY THAN SEGMENTS IN "A". ONE THING IS CLEAR,FOR A SEGMENT TO ARRIVE ON THE OVERLAY SELECTION QUEUE IT WAS THE LEAST RECENTLY OR HEAVILY USED SEGMENT IN THE SET. WE MAY ASSUME

THAT SOME OF THE SEGMENTS IN "B" ARE IN THE SET. THESE ARE THE SEGMENTS WHICH WERE REFERENCED BY THE LAST PROCESS RUNNING THE PROGRAM. THEORITICALLY, AS MORE USERS ACCESS A PROGRAM, THE SPACE ALLOCATED TO IT IS INCREASED TO ACCOMODATE THE NEW USERS. SIMULATION RESULTS INDICATED THAT THE ALGORITHM WOULD PERFORM IN THIS FASHION.

A RATHER MINOR CHANGE TO CREATE CAN BE MADE TO FORCE EACH PROCESS TO BE ASSIGNED A UNIQUE WORKING SET. THIS APPROACH IGNORES THE SHARING PROBLEM COMPLETELY. UNPREDICTABILTY IN UPDATING REFERENCE COUNTERS COULD CAUSE ERRATIC REPLACEMENT BEHAVIOR,HOWEVER.

THE EFFECTIVENESS OF ANY REPLACEMENT ALGORITHM IS DEPENDENT ON THE AMOUNT OF MAIN MEMORY AVAILABLE AND THE NUMBER OF CONCURRENTLY RUNNING PROGRAMS. OVERLOADING THE SYSTEM WILL RESULT IN A THRASHING CONDITION REGARDLESS OF THE REPLACEMENT ALGORITHM USED. THIS PROBLEM CAN ONLY BE RESOLVED BY NOT SCHEDULING JOBS OR SESSIONS WHEN THE OVERLAY RATE IS HIGH.

THE PROBLEM OF SCHEDULING AND HIGH OVERLAY RATE DETECTION IS STILL A GREY AREA IN THE MPE30 DESIGN. I FEEL THAT THIS IS A MAJOR PROBLEM WHICH REQUIRES A COMPLETE OR ,AT LEAST, PARTIAL SOLUTION.

EVERY CST AND DST ENTRY WHICH REFERENCES A PRESENT SEGMENT CONTAINS A USE REFERENCE COUNTER. WHEN THE WORKING SET OF A PROCESS MUST BE REDUCED IN SIZE, THE REFERENCE COUNTER IS USED TO SELECT THE SEGMENT(S) THAT HAVE BEEN LEAST RECENTLY USED OR LEAST HEAVILY ACCESSED WHEN ALL ENTRIES HAVE BEEN RECENTLY ACCESSED.

THE OVERLAY SELECTION QUEUE DOUBLES AS THE WORKING SET OVERFLOW QUEUE. A WORKING SET ENTRY IS FIXED IN SIZE AND MAY NOT BE LARGE ENOUGH TO DEFINE THE PROCESS LOCALITY. WHEN THE SET IS FULL, THE LEAST RECENTLY USED SEGMENT IN THE SET WILL BE DELETED AND THE NEW SEGMENT WILL BE ADDED TO THE SET. THE DELETED SEGMENT WILL BE PLACED ON THE OVERLAY SELECTION QUEUE. THE PROCEDURE ADDTOWS IS CALLED TO ADD AN ENTRY TO THE WORKING SET.

SEGMENTS MAY BE PLACED ON THE QUEUE FOR OTHER REASONS. THE REPLACEMENT ALGORITHM DICTATES WHEN SET REDUCTIONS MUST BE PERFORMED. THE PROCEDURE DELETENRE IS CALLED TO DELETE ALL NONREFERENCED SEGMENTS FROM A WORKING SET. THE PROCEDURE TAKEFROMWS IS CALLED TO DELETE THE LEAST RECENTLY USED SEGMENT FROM A SET. FINALLY, THE PROCEDURE CLEARWS IS CALLED BY THE UNLOAD FUNCTION TO DELETE ALL OVERLAYABLE WORKING SET ELEMENTS PRIOR TO RELEASING THE WORKING SET. ALL OF THE ABOVE PROCEDURES CALL THE PROCEDURE PLACEONOLST TO PLACE SEGMENTS ON

THE OVERLAY SELECTION QUEUE. ATTACHMENT 4 SHOWS THE RELEVANT TABLE AND QUEUE LINKAGES. A WORKING SET CONTAINS THE FOLLOWING INFORMATION:

1. A COUNT SPECIFYING THE MAXIMUM NUMBER OF SEGMENT LOCATORS ALLOWED IN THE WORKING SET. THE CURRENT VALUE OF THIS COUNT IS THE WSTAB ENTRY SIZE-4. A WSTAB ENTRY IS CURRENTLY 24 WORDS IN LENGTH. LATER, THE WSTAB ENTRY SIZE WILL BE CONFIGUREABLE.

2. A COUNT SPECIFYING THE NUMBER OF SEGMENT LOCATORS CURRENTLY IN THE WORKING SET. THIS VALUE IS <= TO THE MAXIMUM SET SIZE.

3. A VIRTUAL TIME TAU, IN MILLISECONDS. WHICH IS USED TO CONTROL THE ADDITION OR DELETION OF SEGMENT LOCATORS. WHEN A SEGMENT FAULT OCCURS, TAU IS COMPARED TO THE VIRTUAL PROCESS TIME IN PCBI(%15). THIS VALUE IS THE ELLAPSED PROCESS TIME SINCE THE LAST SEGMENT FAULT. IF THE TIME IS > TAU, ALL NONREFERENCED SEGMENTS WILL HAVE THEIR CORRESPONDING LOCATORS DELETED FROM THE WORKING SET PRIOR TO THE ADDITION OF THE NEW SEGMENT LOCATOR.

4. A VIRTUAL TIME ALPHA, IN MILLISECONDS. WHICH SPECIFIES HOW MUCH VIRTUAL TIME HAS ELLAPSED SINCE THE REFERENCE COUNTERS ACCESSED THROUGH THE SEGMENT LOCATORS WERE LAST UPDATED. THIS VALUE IS THE SUMMATION OF THE VIRTUAL TIMES OF ALL PROCESSES ACCESSING THE SET. THE REFERENCE COUNTERS WILL BE UPDATED ON THE FIRST SEGMENT FAULT OR ONQLST CALL WHERE ALPHA >= TAU. THIS IMPLIES THAT REFERENCE COUNTER UPDATES WILL GENERALLY OCCUR ABOUT TAU MILLISECONDS APART. OCCASSIONALLY, A SEGMENT MAY BE A MEMBER OF TWO OR MORE WORKING SETS. IN THIS CASE, THE REFERENCE COUNTER FOR THE SEGMENT MAY BE UPDATED MORE FREQUENTLY THAN EVERY TAU MILLISECONDS. THIS WILL RARELY OCCUR SINCE SEGMENT LOCATORS REFERENCING AN ABSENT SEGMENT WILL BE DELETED FROM A WORKING SET WHEN THE PROCEDURES DELETENRE,CLEARWS AND TAKEFROMWS ARE CALLED. NOTE THAT AN ENTRY IS ONLY ADDED TO A WORKING SET WHEN A SEGMENT FAULT OCCURS.

5. THE REMAINING WORDS IN THE WORKING SET ENTRY CONTAIN DST OR CST ENTRY LOCATORS. THE LOCATOR FOR A DST OR CST ENTRY IN THE RANGE CST(1,%277) IS A CST-RELATIVE INDEX TO THE SPECIFIED ENTRY. A LOCATOR FOR AN ENTRY IN THE CST EXTENSION, CST(%301,%377), CONTAINS TWO FIELDS WHICH ARE USED TO DETERMINE THE LOCATION IN THE EXTENSION AREA. THE ALGORITHM

FOR CALCULATING THE DST-RELATIVE INDEX INTO THE CST EXTENSION IS:

```
DSTINDEX := CSTBLK(LOCATOR.(3:7))+4*LOCATOR.(10:6).
```

THE ONLY I/O PERFORMED BY THE MEMORY MANAGER IS THE READING OF DATA FROM A DISK OR THE WRITING OF A DATA SEGMENT TO THE SYSTEM DISK. ONE OF THE DESIGN GOALS WAS TO PERFORM MEMORY MANAGEMENT I/O IN NOWAIT MODE TO TAKE ADVANTAGE OF THE REAL I/O-CPU CONCURRENCY.

INITIATING A MEMORY MANAGEMENT READ OR WRITE-

THE PROCEDURE MAMIO IS CALLED TO INITIATE A MEMORY MANAGEMENT I/O REQUEST. THE REQUEST IS CONSTRUCTED BY ALLOCATING AN MTAB ENTRY AND LINKING IT TO THE LOGICAL DEVICE SPECIFIED BY THE CALLER. THERE ARE TWO WORDS RESERVED IN EACH DISK DEVICE INFORMATION TABLE, BIT(8) & BIT(9), WHICH HOLD THE HEAD AND TAIL INDICES OF THE REQUESTS LINKED TO THAT DEVICE. ADDITIONAL INFORMATION FOR EXECUTING THE I/O IS LOCATED IN THE EIGHT WORD LINK ATTACHED TO THE TARGET AREA IN MAIN MEMORY.

WRITE REQUESTS ARE OPTIMIZED IN THE SENSE THAT IF MAMIO IS CALLED TO INITIATE A WRITE ON A SEGMENT AND IT HAPPENS THAT A WRITE REQUEST FOR IT IS ALREADY IN THE QUEUE, THEN ONE OF THE FOLLOWING EVENTS WILL TAKE PLACE:

1. IF THE QUEUED REQUEST HAS NOT BEEN STARTED, THE NEW REQUEST IS THROWN AWAY SINCE THE I/O WILL EVENTUALLY BE PERFORMED ANYWAY.

2. IF THE QUEUED REQUEST IS BEING EXECUTED, IT WILL BE RESTARTED WHEN IT COMPLETES. AGAIN, THE NEW REQUEST MAY BE THROWN AWAY SINCE THE I/O WILL EVENTUALLY BE PERFORMED.

THIS "FEATURE" ELIMINATES REDUNDANT I/O REQUESTS AND MINIMIZES THE NUMBER OF MTAB ENTRIES REQUIRED BY THE MAMIO MECHANISM.

WHILE ON THE SUBJECT OF WRITES, I SHOULD MENTION A CONCEPT I CALL AN "ANTICIPATORY" WRITE. ONE OF THE GOALS WAS TO MINIMIZE THE PHYSICAL TIME REQUIRED TO MAKE PRESENT AN ABSENT SEGMENT. MOST LONG DELAYS ARE CAUSED BY WAITS GENERATED WHEN OVERLAYING DATA SEGMENTS TO FREE UP SPACE IN MAIN MEMORY. AN OVERLAY WRITE INITIATED FOR A DATA SEGMENT WHEN IT IS PLACED ON THE OVERLAY SELECTION QUEUE IS CALLED AN "ANTICIPATORY" WRITE. WHAT IS ANTICIPATED, OF COURSE, IS THE OVERLAY OF THE SEGMENT. A SWITCH LOCATED IN %1052 CAN BE SET TO ENABLE THE COMPLETE ANTICIPATORY WRITE FEATURE. CURRENTLY , ANTICIPATORY WRITES ARE AUTOMATIC IN PLACEONQUEST ONLY. HOPEFULLY, WHEN A SEGMENT IS SELECTED FOR OVERLAY BY MAM, THE I/O WILL HAVE BEEN

COMPLETED AND THE SPACE CAN BE USED IMMEDIATELY. MEASUREMENTS TO DATE INDICATE THAT ANTICIPATORY WRITES HAVE A NEGATIVE EFFECT ON SYSTEM PERFORMANCE WHEN THE DENSITY OF EXCHANGEDB'S IS HIGH. EVEN THOUGH IT TAKES LESS THAN TWO MILLISECONDS TO INITIATE AN ANTICIPATORY WRITE, THE ADDED CPU OVERHEAD MORE THAN CANCELS OUT ANY BENIFITS. WHEN THE EXCHANGEDB RATE IS LOW, PERFORMANCE IS IMPROVED.

THE I/O SYSTEM WILL ABORT ANY WRITE I/O FOR A PRESENT SEGMENT UNDER THE ASSUMPTION THAT IT HAS BEEN TAKEN OFF THE OLSQ AND MADE ACTIVE AGAIN.

A READ REQUEST ALWAYS TAKES PRECEDENCE OVER A WRITE REQUEST, AND WILL BE LINKED IMMEDIATELY BEHIND THE FIRST ELEMENT IN THE QUEUE IF THE QUEUE IS NONEMPTY. A READ REQUEST WILL NOT BE INITIATED BY MAM INTO THE READ TARGET AREA IN MAIN MEMORY UNTIL THE AREA IS FREE OF ALL ONGOING I/O, I.E. ALL REQUIRED WRITES HAVE BEEN COMPLETED.

COMPLETION OF A MEMORY MANAGEMENT I/O REQUEST-

WHEN THE I/O SYSTEM HAS COMPLETED A MEMORY MANAGEMENT I/O REQUEST, IT CALLS THE PROCEDURE MAMIODONE. IF THE COMPLETION IS FOR A WRITE MARKED FOR RESTART, THE REQUEST WILL BE RELINKED TO THE END OF THE QUEUE. IF THE COMPLETION IS REQUIRED BY MAM TO SATISFY THE CURRENT MAM REQUEST, THE MTAB ENTRY USED IN THE I/O WILL BE LINKED INTO THE MAM I/O COMPLETION QUEUE ,IOCQ. MAM WILL THEN BE NOTIFIED OF THE COMPLETION. IF THE COMPLETION IS NOT REQUIRED BY A MAM REQUEST, THE MTAB ENTRY USED IN THE REQUEST IS RETURNED TO THE MTAB FREE LIST.

MAMIO IS CALLED FROM A NUMBER OF ROUTINES. SEE A CROSS REFERENCE FOR THE OCCURRENCES.

DATA SEGMENT EXPANSIONS/CONTRACTIONS

A STACK EXPANSION REQUEST IS DEFINED TO BE A MAM REQUEST WHICH SPECIFIES EXPANSION OF ONE OF THE FOLLOWING STACK AREAS:

1. (DL-DB)-INITIATED BY CALLING DLSIZE
2. (DB- Z)-INITIATED BY CALLING ZSIZE OR A STACK OVERFLOW
3. (C- B)-INITIATED BY CALLING GETPXSEG
4. (D- C)-INITIATED BY CALLING ALTPXFILESIZE

SEE THE PCBX GENERAL FORMAT FOR THE DEFINITION OF B,C & D.

THE PHYSICAL EXPANSION OF THE STACK IS INITIATED BY CALLING EXPANDREG. THE REQUEST IS QUEUED INTO THE MAM REQUEST QUEUE WHEN THE SEGMENT OVERLAY IS COMPLETED, MAM WILL INITIATE A SEGMENT READ INTO A LARGER AREA IN MAIN MEMORY. THE TARGET ADDRESS WILL BE OFFSET FROM THE AREA BASE BY THE VALUE OF THE EXPANSION SIZE. FOR EXAMPLE, IF THE ADDRESS OF THE ALLOCATED AREA IS A AND THE SIZE OF THE EXPANSION WAS N WORDS, THE STARTING ADDRESS FOR THE READ WOULD BE A+N. WHEN THE READ IS COMPLETED, ALL OR A PORTION OF THE PCBX WILL BE MOVED FROM THE ADDRESS A+N TO THE ADDRESS A. THE NUMBER OF WORDS MOVED IS DEPENDENT ON THE SOURCE OF THE REQUEST. IF THE EXPANSION IS FOR THE AREA (DL-DB), THE ENTIRE PCBX IS MOVED. IF THE EXPANSION IS FOR THE PXFIXED AREA, THE PCBX THROUGH THE PXFIXED AREA IS MOVED. IF THE EXPANSION IS FOR THE PXFILE AREA, THE PCBX THROUGH THE PXFILE AREA IS MOVED. IN THE CASE OF A (DB-Z) EXPANSION, NO MOVE IS REQUIRED SINCE THE EXPANSION IS ON THE HIGH END OF THE STACK.

STACK OR DATA SEGMENT REDUCTIONS ARE PERFORMED IN THE FUNCTIONS DLSIZE,ZSIZE,ALTPXFILESIZE OR ALTDSEGSIZE BY MOVING THE PCBX OF THE STACK IN MAIN MEMORY AND THEN RETURNING TO THE FREE LIST THE UNUSED SPACE. NO PCBX MOVEMENT IS PERFORMED ON A DATA SEGMENT. REDUCTIONS OF LESS THAN 128 WORDS ARE TREATED AS A NOP.

SOFTWARE LINKAGE OF MAIN MEMORY

MAIN MEMORY IS SUBDIVIDED INTO TWO CONTIGUOUS AREAS, A CORE RESIDENT AREA AND A LINKED MEMORY AREA. THE CORE RESIDENT AREA OCCUPIES THE LOWER PART OF MEMORY BANK ZERO. THE REMAINING MEMORY IS REFERED TO AS LINKED MEMORY. FREE AREAS ARE LINKED TOGETHER TO FORM THE "FREE LIST". A TWO WORD EXTENDED ADDRESS STORED IN %1042 AND %1043 POINTS TO THE FREE LIST HEAD. THE FREE LINK COUNT IS STORED IN LOCATION %1044. FREE AREAS ARE DOUBLY LINKED, SPANNING ALL LINKED MEMORY. MASKS ON EACH BANK BOUNDRY ARE USED TO BLOCK THE OVERFLOW OF AN AREA INTO AN ADJACENT BANK. THE MASK IS FOUR WORDS IN LENGTH. THE FORMAT OF THE MASK IS: [%100000,-4,0,%100001]. THIS IS A CONCATENATION OF THE NORMAL EIGHT WORD HEADER AND ONE WORD TRAILER. THE SEGMENT APPEARS TO THE SYSTEM AS A RESERVED AREA -4 WORDS IN LENGTH.

SPACE ALLOCATED TO CODE AND DATA SEGMENTS IS TAKEN FROM THE AVAILABLE FREE AREAS. THE ASSIGNED AREAS ARE NOT LINKED TOGETHER AND MUST BE ACCESSED THROUGH THEIR CORRESPONDING DST OR CST DESCRIPTORS.

-ASSIGNED AREA STATES-

AN ASSIGNED AREA IS IN ONE OF TWO STATES; OVERLAYABLE OR NONOVERLAYABLE. ONLY SEGMENTS WHICH ARE OVERLAYABLE CAN BE PLACED ON THE OVERLAY SELECTION LIST. OVERLAYABILITY IS A FUNCTION OF THE FOLLOWING SUBSTATES:

-CORE RESIDENCY-
A SEGMENT IS IN EITHER LINKED OR NONLINKED MEMORY. IF THE SEGMENT IS IN NONLINKED MEMORY, IT IS SAID TO BE CORE RESIDENT. CORE RESIDENT SEGMENTS ARE NOT OVERLAYABLE.

-LOCK-
A SEGMENT MAY BE "LOCKED" IN MAIN MEMORY BY CALLING THE PRIVILEGED PROCEDURE LOCKSEG. A LOCKED SEGMENT WILL BE ALLOCATED MAIN MEMORY AT OR NEAR A MAIN MEMORY BANK BOUNDRY. LOCKED SEGMENTS ARE NOT OVERLAYABLE. A LOCKED SEGMENT WHICH IS NOT FROZEN MAY BE MOVED WITHIN THE BANK IT IS LOCKED TO MAXIMIZE THE USE OF MEMORY.

-EXCHANGEDB COUNT-
OF SPECIAL INTEREST IS THE HANDLING OF DATA SEGMENTS WHICH ARE MANIPULATED BY CALLING THE FUNCTION EXCHANGEDB. THE COUNT OF THE NUMBER OF EXCHANGEDB'S TO A DATA SEGMENT IS MAINTAINED BY THE MEMORY MANAGER. A DATA SEGMENT WITH A NONZERO EXCHANGEDB COUNT IS GENERALLY NONOVERLAYABLE, HOWEVER, CERTAIN MEMORY MANAGEMENT FUNCTIONS MAY REQUIRE SUCH AN OVERLAY. IF THIS OCCURS, THE PROCESSES ACCESSING THE SEGMENT WILL BE PLACED IN ABSTATE, AND MUST BE PREP'ED AGAIN BEFORE RUNNING.

-FREEZE-
THE STRONGEST HOLD A USER CAN APPLY TO A SEGMENT IN LINKED MEMORY IS TO FREEZE IT. A FROZEN SEGMENT IS NONOVERLAYABLE AND MAY NOT BE MOVED FROM ITS FROZEN POSITION. TWO FORMS OF FREEZES ARE AVAILABLE. A NORMAL FREEZE IN ENVOKED BY CALLING THE FUNCTION FREEZE. THE I/O SYSTEM MUST FREEZE A SEGMENT BY CALLING THE FUNCTION IOFREEZE.

-SUMARIZATION OF OVERLAYABILITY AND MOVEABILITY-

A SEGMENT IS OVERLAYABLE IF IT IS NOT CORE RESIDENT,HAS A ZERO EXCHANGEDB COUNT,IS NOT LOCKED AND IS NOT FROZEN. CONVERSELY, A SEGMENT IS NONOVERLAYABLE IF IT IS CORE RESIDENT OR FROZEN OR LOCKED OR HAS A NONZERO EXCHANGEDB COUNTER.

A SEGMENT IS MOVEABLE WITHIN LINKED MEMORY IF IT IS NOT

FROZEN. CONVERSELY, A SEGMENT IS NOT MOVEABLE IN LINKED MEMORY IF IT IS FROZEN.

PROCESSING A LOCK REQUEST

THE SEGMENT LOCKING FACILITY PROVIDES A WAY TO PLACE A SEGMENT AT OR NEAR A MAIN MEMORY BANK BOUNDRY. SINCE A LOCKED SEGMENT IS NONOVERLAYABLE, SEVERE LINKED MEMORY FRAGMENTATION COULD OCCUR IF THE SEGMENT WAS NOT LOCATED NEAR A BOUNDRY.

A LOCK REQUEST IS INITIATED BY CALLING THE PROCEDURE LOCKSEG. THE CALLER MAY SPECIFY LOCKING ON AN UPPER OR LOWER BOUNDRY. THE UPPER BOUNDRY IS THE DEFAULT ASSIGNMENT. THE CALLER MAY ALSO SPECIFY THAT THE SEGMENT BE LOCKED IN BANK ZERO ONLY. IF BANK ZERO IS NOT SPECIFIED, THE PROCEDURE BESTBANK WILL RETURN THE NUMBER OF THE SELECTED BANK. THE BESTBANK SELECTION CRITERIA IS QUITE SIMPLE. A COUNT OF THE NUMBER OF LOCKED SEGMENTS AND LOCKED SPACE SIZE IS MAINTAINED FOR EACH BANK IN THE BANKTAB TABLE. VALUES ARE KEPT FOR BOTH THE UPPER AND LOWER BOUNDRYS. THE SMALLEST DOUBLET FOR THE SPECIFIED BOUNDRY IS CONSIDERED THE BEST.

THE PROCEDURE LSEARCH IS CALLED BY LOCKSEG TO LOCATE THE AREA AFTER THE BANK AND BOUNDRY HAVE BEEN DETERMINED. LSEARCH WILL SCAN THE BANK IN THE DIRECTION SPECIFIED IN THE BOUNDRY SELECTION TO LOCATE AN AREA. THE SEARCH WILL BYPASS ANY LOCKED OR FROZEN AREAS. A ZERO IS RETURNED IF THE SEARCH FAILS.

A SYSBUF ENTRY IS ALLOCATED AS A TEMPORARY BUFFER TO SAVE THE INFORMATION REQUIRED BY THE LOCK REQUEST. THE SYSBUF RELATIVE INDEX ,LINX, IS OFFSET SO THAT IT POINTS TO THE SIXTH WORD OF THE BUFFER. THE BUFFER WILL CONTAIN THE FOLLOWING INFORMATION SET UP BY LSEARCH:

SYSBUF(LINX-5)=BANK NUMBER

SYSBUF(LINX-4)=THE ADDRESS IN THE BANK WHERE THE SEGMENT WILL BE LOCATED.

SYSBUF(LINX-3)=THE SIZE OF THE AREA ALLOCATED TO THE LOCKED SEGMENT.

SYSBUF(LINX-2)=THE ADDRESS IN THE BANK OF AN AREA TO BE RETURNED TO THE FREE SPACE LIST. NORMALLY, THE AREA TO BE CLEARED IS LARGER THAN THE AREA NEEDED TO SATISFY THE LOCK REQUEST. THE DIFFERENCE, IF ANY, IS RELINKED INTO THE FREE SPACE LIST.

SYSBUF(LINX-1)=THE SIZE OF THE AREA TO BE RETURNED TO THE FREE LIST.

SYSBUF(LINX-0)=THE NUMBER OF SEGMENTS WHICH MUST BE CLEARED FROM THE AREA. THIS NUMBER IS DESIGNATED BY N.

SYSBUF(LINX+1)=NOT USED.

THE NEXT 2*N WORDS OF THE TABLE CONTAIN DOUBLE WORD ENTRIES SPECIFYING WHAT THE SEGMENT IS AND HOW IT SHOULD BE CLEARED. EACH TWO WORD ENTRY HAS THE FOLLOWING FORMAT , WHERE [I=(1,N):K_LINX+2*I]:

SYSBUF(K),=0 IF AREA IS LINKED INTO FREE LIST.
>0 IF ASSIGNED AREA, CELL CONTAINS A CLABEL.

SYSBUF(K+1)=ADDRESS OF AREA IN THE BANK SPECIFIED IN SYSBUF(LINX-5).

IF AN AREA OF SUFFICIENT SIZE IS LOCATED, LOCKSEG WILL CALL THE PROCEDURE MOVEMOUT TO CLEAR IT OUT. MOVEMOUT WILL USE THE INFORMATION ENTERED IN THE SYSBUF ENTRY. NOTE THAT A LARGE PORTION OF THE WORK REQUIRED IN LOCKING A SEGMENT IS PERFORMED IN NON RESIDENT SEGMENTS. THE FINAL CLEARING MUST BE PERFORMED IN A CORE RESIDENT ROUTINE SINCE THE SEGMENT CONTAINING LOCKSEG MAY ITSELF BE IN THE AREA TO BE CLEARED. MOVEMOUT WILL PERFORM THE FOLLOWING OPERATIONS:

A) IF AN AREA IS FREE, IT WILL BE DELINKED FROM THE FREE LIST.

B) IF AN AREA IS ASSIGNED, THEN ONE OR MORE OF THE FOLLOWING STEPS WILL BE PERFORMED:

1) THE SPACE IS RESERVED IN MAIN MEMORY BY SETTING THE SIGN BIT OF THE FIRST AND LAST WORD OF THE AREA. THIS PLACES THE AREA IN A NONMOVEABLE, NONOVERLAYABLE STATE.

2) IF THE AREA HAS A NONZERO I/O FREEZE COUNTER, THE LOCK REQUEST BIT IN THE ASSOCIATED DESCRIPTOR WILL BE SET TO ONE. WHEN THE I/O FREEZE COUNTER GOES TO ZERO, MAM WILL BE NOTIFIED OF IT'S AVAILABILTY. THIS IS DONE IN IOUNFREEZE BY LINKING A REQUEST INTO THE MAM REQUEST QUEUE.

3) IF THE AREA IS NOT I/O FROZEN, THE ABSENCE BIT WILL BE SET IN THE ASSOCIATED DESCRIPTOR. IF THE SEGMENT IS SUBSEQUENTLY ACCESSED, IT WILL BE ALLOCATED SPACE IN ANOTHER PART OF

MEMORY.

IF THE AREA IS IN USE BY A DATA SEGMENT, AND IT IS NOT THE STACK I AM CURRENTLY RUNNING ON, AN OVERLAYIO WILL BE FIRED UP TO CLEAR THE AREA. ALL PROCESSES REFERENCING THIS SEGMENT AS A STACK OR EXTRA DATA SEGMENT WILL HAVE THEIR STATE SET TO ABSTATE.

IF IT HAPPENS TO BE THE STACK I AM RUNNING ON, A MAM REQUEST OF TYPE=10 WILL BE LINKED INTO THE MAM REQUEST QUEUE. MAM WILL SUBSEQUENTLY OVERLAY THE AREA.

4) WHEN ALL N AREAS HAVE BEEN PROCESSED, A MAM REQUEST OF TYPE 9 WILL BE QUEUED IF THE SEGMENT TO BE LOCKED IS NOT IN THE AREA BEING CLEARED, OR A REQUEST OF TYPE 10 WILL BE QUEUED IF THE SEGMENT TO BE LOCKED IS IN THE AREA. MAM IS THEN WOKEN TO PROCESS THE REMAINDER OF THE REQUEST.

FOUR LOCATIONS IN SYSGLOB ARE RESERVED FOR PROCESSING A LOCK REQUEST. THEY ARE:

%1271=LOKINX...CLABEL OF SEGMENT BEING LOCKED.

%1272=LOKPINX..PCB RELATIVE INDEX OF PROCESS REQUESTING THE LOCK.

%1273=LOKWCNT..THE NUMBER OF OVERLAY I/O COMPLETIONS REQUIRED BEFORE THE AREA CAN BE USED.

%1274=LOKLINX..A SYSBUF RELATIVE INDEX TO THE INFORMATION BUFFER.

FINALLY, MAM TAKES OVER TO COMPLETE THE REQUEST PROCESSING. MAM WILL PERFORM THE FOLLOWING ACTIONS:

1) IF RT=9 AND THE REQUEST SPECIFIES A DATA SEGMENT, IT WILL BE OVERLAYED, THE DISK ADDRESS RESTORED IN THE DESCRIPTOR, AND THE SEGMENT MARKED ABSENT.

2) IF RT=10, A CHECK IS MADE TO SEE IF LOKWCNT HAS GONE TO ZERO. THIS INDICATES THAT ALL OVERLAYS ARE COMPLETED AND THE AREA IS NOW FREE TO USE. MAM WILL BUILD A TYPE 11 REQUEST,

RETURN ANY UNUSED PORTION OF THE AREA TO THE FREE LIST, AND LINK ANY REQUESTS QUEUED INTO THE LOCK REQUEST QUEUE INTO THE MAM REQUEST QUEUE.

3) IF RT=11, THE SIZE AND ADDRESS OF THE AREA IS TAKEN FROM THE SYSBUF ENTRY. THE FREE SPACE SEARCH IS BYPASSED SINCE THE SPACE HAS BEEN ALREADY ALLOCATED. THE REMAINING PROCESSING IS IDENTICAL TO THAT FOR A REQUEST OF TYPE ZERO.

THE LOCAL COMPRESSION ALGORITHM

THE MEMORY COMPACTION SCHEME IMPLEMENTED IS A HEURISTIC DESIGN I CALL LOCAL COMPRESSION. THE IDEA IS TO ANTICIPATE FRAGMENTATION AND ATTEMPT TO CONCATENATE DISJOINT FREE AREAS WHENEVER SPACE IS BEING RETURNED TO THE FREE SPACE LIST. LET ME GIVE THE FOLLOWING CASE AS AN EXAMPLE:

SUPPOSE WE HAVE A LIST OF SEQUENTIAL AREAS IN MAIN MEMORY (F1,A1,A2,A3,F2), WHERE F1 AND F2 ARE FREE AREAS, AND A1,A2, AND A3 ARE ASSIGNED AREAS. SUPPOSE THAT THE AREA A2 IS RETURNED TO THE FREE SPACE LIST, BECOMING A FREE AREA CALLED F0. WE NOW HAVE A LIST THAT LOOKS LIKE (F1,A1,F0,A3,F2).

THE LOCAL COMPRESSION ALGORITHM WILL ATTEMPT TO CONCATENATE F1,F2 AND F0 INTO ONE LARGER FREE AREA. IDEALLY, WE SHOULD ARRIVE AT ONE OF THE FOLLOWING SEQUENCES: (F',A1,A3) OR (A1,A3,F'), WHERE F' IS THE NEW FREE AREA.

THE CRITERIA FOR APPLYING THE ALGORITHM IS AS FOLLOWS:

1. NO MEMORY MANAGEMENT I/O MUST BE GOING ON WHEN THE ATTEMPT IS MADE TO COMPACT THE AREAS.

2. SEGMENT A1 OR A3 MUST BE MOVEABLE. THEY DON'T HAVE TO BE OVERLAYABLE, JUST MOVEABLE. NOTE THAT THE COMPRESSION MAY SUCCEED ON ONLY ONE END; THAT'S GOOD ENOUGH.

3. THE SIZE OF A1 OR A3 MUST NOT EXCEED 4K WORDS. THIS IS TO MINIMIZE THE MOVE TIME REQUIRED. THE CURRENT VALUE OF 4K IS DEFINED IN THE EQUATED VARIABLE BESTSIZE IN THE SEGMENT MMCORER. WHETHER THIS IS THE BEST VALUE IS CERTAINLY DEBATABLE.

4. LOCAL COMPRESSION MUST BE ENABLED BY SETTING THE SYSGLOB LOCATION %1053 TO A TRUE STATE. THE FLAG IN CURRENTLY ENABLED WHEN THE SYSTEM IS LOADED.

THE PROCEDURES EXPABOVE,EXPBELOW AND EXPCOM ARE THE ONLY FUNCTIONS USED BY THE ALGORITHM.

HANDLING ACCESS CONTENTION

THE PDISABLE AND PENABLE OPERATORS ARE USED IN MOST MEMORY MANAGEMENT PROCEDURES TO PROTECT THE EXECUTION OF CRITICAL CODE SECTIONS. I VIEW THEM AS SUPER P & V FUNCTIONS. THEY DO NOT , UNFORTUNATELY, PROVIDE PROTECTION WHEN AN I/O SYSTEM FUNCTION CALLS THE PROCEDURE MAKEPRESENT. IN EFFECT, THE I/O SYSTEM DISREGARDS THE PDISABLED STATE. OBVIOUSLY, THE PROBLEM COULD BE SOLVED IF I USED A LONG DISABLE INTERRUPT SEQUENCE. THIS SOLUTION WAS REJECTED SINCE THE DELAY WOULD HAVE EXCEEDED ONE MILLISECOND. TO PROVIDE THE REQUIRED PROTECTION, SEGMENT ACCESS IS CONTROLLED BY "BUSY" AND "REQUEST" SEMAPHORES. THE SEMAPHORES ARE LOCATED IN THE BIT FIELD (9:2) OF THE THIRD WORD OF A CST OR DST ENTRY. BIT 9 IS THE "REQUEST" SEMAPHORE, AND BIT 10 IS THE "BUSY" SEMAPHORE.

WHEN A STACK SIZE CHANGE IS REQUESTED BY CALLING DLSIZE, ZSIZE,ALTPXFILESIZE, ETC. , THE PROCEDURE SEGSTATE IS CALLED TO DETERMINE THE ACCESSABILITY OF THE STACK. IF THE STACK IS OVERLAYABLE, THE "BUSY" BIT IS SET IN THE ASSOCIATED DST ENTRY. THIS IS DONE , OF COURSE, WHILE INTERRUPTS ARE DISABLED. IF AN I/O SYSTEM FUNCTION CALLS MAKEPRESENT WHILE THE SEGMENT IS "BUSY", IT'S UNBLOCKED REQUEST WILL BE QUEUED INTO THE DEFERED REQUEST QUEUE RATHER THAN THE MAM REQUEST QUEUE. IF THIS OCCURS, THE "REQUEST" SEMAPHORE IS SET IN THE DESCRIPTOR. EVENTUALLY, THE PROCEDURE PERFORMING THE STACK CHANGE WILL CALL THE PROCEDURE CHEKDEFERALS. THIS PROCEDURE WILL RESET THE "BUSY" BIT AND IF ON, LINK ALL DEFERED REQUESTS FOR THE SEGMENT INTO THE MAM REQUEST QUEUE. THEY ARE DELINKED FROM THE DEFERED REQUEST QUEUE, OF COURSE.

NOTE THAT THE SEMAPHORES SHARE POSITIONS WITH THE FREEZE AND I/O FREEZE BITS WHEN THE SEGMENT IS PRESENT. A STACK CHANGE CAN NOT BE PROCESSED IF THE STACK IS IN A FROZEN STATE. IF A STACK CHANGE IS ALLOWED, THE DST ENTRY IS MARKED ABSENT AND THE REFERENCE COUNTER IS ZEROED EVEN THOUGH I AM RUNNING ON THE STACK. AS LONG AS I REMAIN PDISABLED, THIS CAUSES NO PROBLEMS, AND FORCES THE I/O SYSTEM TO CALL THE MAKEPRESENT PROCEDURE.

MEMORY MANAGEMENT TABLES

MEMORY LINK FORMAT-ASSIGNED-

```
  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15
-----------------------------------------------------------------
* R*  *BL*               FORM                                  *   0
-----------------------------------------------------------------
*             SIZE OF AREA ALLOCATED                           *   1
-----------------------------------------------------------------
*          LDEV                *          * S*    HODA         *   2
-----------------------------------------------------------------
*               LOW ORDER DISK ADDRESS                         *   3
-----------------------------------------------------------------
*          RESERVED            *      LOCK COUNTER             *   4
-----------------------------------------------------------------
*          RESERVED            *      FREEZE COUNTER           *   5
-----------------------------------------------------------------
*          RESERVED            *      I/O FREEZE COUNTER       *   6
-----------------------------------------------------------------
*                       RESERVED                               *   7
-----------------------------------------------------------------
```

DEFINITION OF TERMS USED

```
1 . LDEV               LOGICAL DEVICE NUMBER OF DISK
2 . HODA               HIGH ORDER DISK ADDRESS
3 . LODA               LOW ORDER DISK ADDRESS
4 . S                  SYSTEM SEGMENT IF = 1
5 . R              =0  AREA IS ASSIGNED
                   =1  AREA IS RESERVED FOR A LOCK OR
                       MAM REQUEST.
6 . BL             =0  THEN FORM IS A DST-RELATIVE INDEX
                   =1  THEN FORM IS A BLOCK LABEL HAVING
                       THE FOLLOWING FORMAT:
                       (3:7)=CSTBLK INDEX
                       (10:6)=EN-%300.WHEN EN IS THE CST
                       INDEX.
```

MEMORY MANAGEMENT TABLES

MEMORY LINK FORMAT-UNASSIGNED-

```
  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
-------------------------------------------------
* R*                     0                      *  0
-------------------------------------------------
*            SIZE OF UNASSIGNED AREA            *  1
-------------------------------------------------
*      LDEV'           *     * U* S*   HODA'    *  2
-------------------------------------------------
*            LOW ORDER DISK ADDRESS-LAST ASSG   *  3
-------------------------------------------------
*                                *     LBN      *  4
-------------------------------------------------
*                     LADR                      *  5
-------------------------------------------------
*                                *     NBN      *  6
-------------------------------------------------
*                     NADR                      *  7
-------------------------------------------------
```

MEMORY LINK FORMAT-BACK LINK-

```
  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
-------------------------------------------------
*UL*AU*              (AREA SIZE+8)/4            *
-------------------------------------------------
```

used to find out if Previous chunk is For A

DEFINITION OF TERMS USED

```
1.  LBN              BANK NUMBER OF LAST FREE AREA
2 . LADR             BANK ADDRESS OF LAST FREE AREA
3.  NBN              BANK NUMBER OF NEXT FREE AREA
4 . NADR             BANK ADDRESS OF NEXT FREE AREA
5 . HODA             HIGH ORDER DISK ADDRESS
6 . LDEV             LOGICAL DEVICE NUMBER OF DISK
7 . R                =0 AREA IS LINKED INTO FREE LIST
                     =1 AREA IS RESERVED FOR A LOCK OR
                        MAM REQUEST.
8 . LDEV'            LDEV WHEN LAST ASSIGNED
9 . HODA'            HODA WHEN LAST ASSIGNED
10. U                REQUIRED BY LOCKSEG CALL
11. AU               =0 AREA IS FREE
```

=1 AREA IS ASSIGNED

12. UL =0 AREA IS AS SPECIFIED BY AU
=1 AREA IS RESERVED FOR A LOCK OR
MAM REQUEST.

TABLE STRUCTURES DST AND CST

```
DST(0) -----------------------------------------------
       *   NUMBER OF ENTRIES IN DST                  *
       -----------------------------------------------
       *   ENTRY SIZE                                *
       -----------------------------------------------
       *   NUMBER OF UNASSIGNED ENTRIES              *
       -----------------------------------------------
       *   INDEX OF FIRST UNASSIGNED ENTRY           *
       -----------------------------------------------
       *                                             *
       *                                             *
       *   TABLE                                     *
       *                                             *
       *                                             *
CST(0) -----------------------------------------------
       *   NUMBER OF ENTRIES IN CST                  *  :=DFC
       -----------------------------------------------
       *   ENTRY SIZE                                *
       -----------------------------------------------
       *   NUMBER OF UNASSIGNED SL ENTRIES           *
       -----------------------------------------------
       *   INDEX OF FIRST UNASSIGNED SL ENTRY        *
       -----------------------------------------------
       *                                             *
       *   PERMANENT, PRE-ASSIGNED SYSTEM            *
       *              ENTRIES                        *
       *                                             *
       -----------------------------------------------
       *                                             *
       *   DYNAMICALLY ASSIGNED SL ENTRIES           *
       *                                             *
       -----------------------------------------------
       *                     0                       *
       -----------------------------------------------
       *                     0                       *
       -----------------------------------------------
       *   NUMBER OF UNASSIGNED CST BLOCK ENTRIES    *
       -----------------------------------------------
       *   INDEX TO UNASSIGNED BLOCK ENTRY           *
       -----------------------------------------------
       *                                             *  :=DFS
       *                                             *
       *   CST EXTENSION BLOCK                       *
       *                                             *
```

*   *

DEFINITION OF TERMS USED AND A COMMENT

1 . DFC — DISPLACEMENT FROM BASE OF DST TO BASE OF CST. DFC=(@CST-@DST).

2 . DFS — DISPLACEMENT FROM BASE OF DST TO FIRST BLOCK OF ASSIGNABLE PROGRAM ENTRIES. THE CST ENTRY NUMBER FOR THE START OF EACH BLOCK IS %300.

COMMENT — THE RELATIONSHIP OF THE DST AND CST IN MEMORY MUST BE AS SHOWN. THE DST WILL IMMEDIATELY PRECEED THE CST IN MEMORY. IMMEDIATELY FOLLOWING THE CST IN MAIN MEMORY IS THE CST EXTENSION.

THE VALUE OF DFC AND DFS ARE STORED WHEN THE SYSTEM IS LOADED INTO THE SYSGLOB LOCATIONS %32 AND %33 RESPECTIVELY.

TABLE FORMAT-CSTBLK-

```
CSTBLK(0) ----------------------------------------
        0 *   NUMBER OF ENTRIES IN TABLE         *
          ----------------------------------------
        1 *   UNASSIGNED ENTRY = -1              *
          ----------------------------------------
        2 *   ASSIGNED ENTRY > 0                 *
          ----------------------------------------
          *                                      *
          *   REMAINING CSTBLK TABLE ENTRIES     *
          *                                      *
          ----------------------------------------
```

COMMENTS-
THE TABLE IS INITIALIZED TO MINUS ONE IN EACH ENTRY. WHEN SELECTED, THE ENTRY IS REPLACED BY A DST-RELATIVE INDEX INTO THE CST EXTENSION BLOCK.

THE FOLLOWING FORMAT IS TYPICAL OF A CST EXTENSION BLOCK ENTRY.

ENTRY FORMAT-CST EXTENSION BLOCK-

```
CSTBLK(CIX)------->    ------------------------------
                  0    * M=# OF CST'S IN BLOCK      *
                       ------------------------------
                  1    * VALIDITY=%125252           *
                       ------------------------------
                  2    * # OF USERS SHARING BLOCK   *
                       ------------------------------
                  3    *            0               *
                       ------------------------------
%301       ------->    * HAS CST ENTRY FORMAT       *
                       ------------------------------
%302       ------->    * HAS CST ENTRY FORMAT       *
                       ------------------------------
                                    .
                                    .
                       ------------------------------
%300+M     ------->    * HAS CST ENTRY FORMAT       *
                       ------------------------------
```

COMMENT   THE VALUE OF CIX IS ESTABLISHED WHEN A CST EXTENSION BLOCK IS ALLOCATED. THIS INDEX INTO THE ARRAY CSTBLK IS MAINTAINED IN THE PCB OF EACH PROCESS SHARING THE BLOCK.

ENTRY FORMAT-DATA SEGMENT TABLE-

1. PRESENT

```
  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15
-----------------------------------------------------------------
* U*CL* R*               LENGTH/4                          *   0
-----------------------------------------------------------------
*               REFERENCE SHIFT COUNTER                    *   1
-----------------------------------------------------------------
*CR*LR*LU*   EXDBC    * C* L* F* I* S*      BN             *   2
-----------------------------------------------------------------
*                    MEMORY ADDRESS                        *   3
-----------------------------------------------------------------
```

2. ABSENT

```
  0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15
-----------------------------------------------------------------
* 1*CL* R*               LENGTH/4                          *   0
-----------------------------------------------------------------
*LK*                  LINK                                 *   1
-----------------------------------------------------------------
*          LDEV(=SYSDISK)*   *MI*IM* S*    HODA            *   2
-----------------------------------------------------------------
*             LOW ORDER DISK ADDRESS                       *   3
-----------------------------------------------------------------
```

DEFINITION OF TERMS USED

```
1 .  R                     SEGMENT REFERENCE BIT
2 .  IM=I                  "BUSY" SEMAPHORE
3 .  EXDBC                 EXCHANGE DB COUNTER
4 .  F                     FREEZE BIT
5 .  C                     CORE RESIDENT BIT
6 .  L                     LOCK BIT
7 .  I                     I/O FREEZE BIT
8 .  S                     SYSTEM SEGMENT BIT
9 .  BN                    BANK NUMBER
10.  HODA                  HIGH ORDER DISK ADDRESS
11.  CR                    CHANGE REQUEST BIT
12.  MI=F                  "REQUEST" SEMAPHORE
13.  LDEV                  LOGICAL DEVICE NO. OF SYS DISK
14.  CL                 =0 SEGMENT IS DIRTY
                        =1 SEGMENT IS CLEAN
15.  (LK,LINK)=(0,  0) THE SEGMENT IS ABSENT.
```

=(0,> 0) THE SEGMENT IS LINKED INTO THE OVERLAY SELECTION QUEUE.

=(1,>=0) THE SEGMENT IS BEING LOCKED. ANY INTERVENING REQUESTS WILL BE LINKED INTO THE LOCK REQUEST QUEUE.

16. LR =1 SEGMENT IS REQUIRED TO SATISFY A LOCK REQUEST. AREA WILL BE OVERLAYED.

17. LU USED IN CONJUNCTION WITH L. ZERO IMPLIES LOWER BOUND. ONE IMPLIES UPPER BOUND.

ENTRY FORMAT-CODE SEGMENT TABLE-

1. PRESENT

```
  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
-------------------------------------------------
* 0* M* R* T*          LENGTH/4                 *  0
-------------------------------------------------
*             REFERENCE SHIFT COUNTER           *  1
-------------------------------------------------
*CR*LR*LU*          * C* L* F* I* S*    BN      *  2
-------------------------------------------------
*                MEMORY ADDRESS                 *  3
-------------------------------------------------
```

2. ABSENT

```
  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
-------------------------------------------------
* 1* M* R* T*          LENGTH/4                 *  0
-------------------------------------------------
*LK*                 LINK                       *  1
-------------------------------------------------
*        LDEV           *   *MI*IM* S*   HODA   *  2
-------------------------------------------------
*          LOW ORDER DISK ADDRESS               *  3
-------------------------------------------------
```

DEFINITION OF TERMS USED

```
1 .  R                   SEGMENT REFERENCE BIT
2 .  M                   SEGMENT MODE BIT
3 .  T                   SEGMENT TRACE BIT
4 .  LDEV                LOGICAL DEVICE NUMBER
5 .  F                   FREEZE BIT
6 .  C                   CORE RESIDENT BIT
7 .  L                   LOCK BIT
8 .  I                   I/O FREEZE BIT
9 .  S                   SYSTEM SEGMENT BIT
10.  BN                  BANK NUMBER
11.  HODA                HIGH ORDER DISK ADDRESS
12.  CR                  CHANGE REQUEST BIT
13.  IM=I                "BUSY" SEMAPHORE
14.  MI=F                "REQUEST" SEMAPHORE
15.  (LK,LINK)=(0,  0) THE SEGMENT IS ABSENT.
              =(0,> 0) THE SEGMENT IS LINKED INTO THE
```

```
                         OVERLAY SELECTION QUEUE.
                 =(1,>=0) THE SEGMENT IS BEING LOCKED. ANY
                         INTERVENING REQUESTS WILL BE LINKED
                         INTO THE LOCK REQUEST QUEUE.
16.   LR              =1 SEGMENT IS REQUIRED TO SATISFY A LOCK
                       REQUEST. AREA WILL BE OVERLAYED.
17.   LU              USED IN CONJUNCTION WITH L. ZERO IMPLIES
                      LOWER BOUND, ONE IMPLIES UPPER BOUND.
```

TABLE STRUCTURES-PCB-

```
PCB(0) -------------------------------------------------
       *    NUMBER OF ENTRIES IN PCB TABLE             *
       -------------------------------------------------
       *    ENTRY SIZE                                 *
       -------------------------------------------------
       *    NUMBER OF UNASSIGNED ENTRIES               *
       -------------------------------------------------
       *    INDEX TO FIRST UNASSIGNED ENTRY            *
       -------------------------------------------------
       *                                               *
       *    REMAINING PCB TABLE ENTRIES                *
       *                                               *
       -------------------------------------------------
```

TABLE STRUCTURES-WSTAB-

```
WSTAB(0) -----------------------------------------
         *   NUMBER OF ENTRIES IN WSTAB TABLE    *
         -----------------------------------------
         *   ENTRY SIZE                          *
         -----------------------------------------
         *   NUMBER OF UNASSIGNED ENTRIES        *
         -----------------------------------------
         *   INDEX TO FIRST UNASSIGNED ENTRY     *
         -----------------------------------------
         *                                       *
         *   REMAINING WSTAB ENTRIES             *
         *                                       *
         -----------------------------------------
```

ENTRY FORMAT-WSTAB TABLE-

```
               0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
              ------------------------------------------------
          0.  *     RESERVED FOR LATER USE                   *
              ------------------------------------------------
          1   *     VIRTUAL TIME SINCE REF CTR UPDATE        *
              ------------------------------------------------
          2   *     CRITICAL-INTER-SEG-ABSENCE-TIME          *
              ------------------------------------------------
WSX--     3   *        MAXNE          *          NE          *
              ------------------------------------------------
          4   *     *BL*        FORM                         *
              ------------------------------------------------
                                  .
                                  .
              ------------------------------------------------
       NE+3   *     *BL*        FORM                         *
              ------------------------------------------------
```

DEFINITION OF TERMS USED

```
1 . WSX                 WORKING SET INDEX IN PCB POINTS HERE
2 . MAXNE               MAXIMUM NUMBER OF ENTRIES ALLOWED
                        IN PROCESS WORKING SET
3 . NE                  CURRENT NUMBER OF ENTRIES IN WORKING
                        SET
4 . BL               =0 FORM IS A DST-RELATIVE INDEX TO THE SEGMENT
                        ENTRY.
                     =1 FORM IS A BLOCK LABEL HAVING THE FOLLOWING
```

FORMAT:

(3: 7) CST BLOCK INDEX
(10: 6) EN-%300, WHERE EN IS A CST ENTRY NUMBER > %

TABLE FORMAT-MTAB-

```
MTAB....-------------------------------------------------------
     0 *    INDEX TO NEXT FREE-INITIALLY=15                   *
        -------------------------------------------------------
     1 *    INDEX TO HEAD OF MAM REQUEST QUEUE                *
        -------------------------------------------------------
     2 *    INDEX TO TAIL OF MAM REQUEST QUEUE                *
        -------------------------------------------------------
     3 *    INDEX TO HEAD OF OVERLAY SELECTION QUEUE          *
        -------------------------------------------------------
     4 *    INDEX TO TAIL OF OVERLAY SELECTION QUEUE          *
        -------------------------------------------------------
     5 *    INDEX TO HEAD OF MAM I/O COMPLETION QUEUE         *
        -------------------------------------------------------
     6 *    INDEX TO TAIL OF MAM I/O COMPLETION QUEUE         *
        -------------------------------------------------------
     7 *    INDEX TO HEAD OF LOCK REQUEST QUEUE               *
        -------------------------------------------------------
     8 *    INDEX TO TAIL OF LOCK REQUEST QUEUE               *
        -------------------------------------------------------
     9 *    INDEX TO HEAD OF DEFERED REQUEST QUEUE            *
        -------------------------------------------------------
    10 *    INDEX TO TAIL OF DEFERED REQUEST QUEUE            *
        -------------------------------------------------------
    11 *    RESERVED                                          *
        -------------------------------------------------------
    12 *    RESERVED                                          *
        -------------------------------------------------------
    13 *    RESERVED                                          *
        -------------------------------------------------------
    14 *    RESERVED                                          *
        -------------------------------------------------------
    15 *    FIRST FREE MTAB ENTRY                             *
        -------------------------------------------------------
```

COMMENTS-
THE TABLE WILL CONTAIN (M) F WORD ENTRIES WHERE M IS A CONFIGURATION CONSTANT. THE FIRST THREE ENTRIES ARE RESERVED FOR USE AS SPECIFIED IN THE ABOVE TABLE DESCRIPTION

```
ENTRY FORMAT-(AREQ) MAM REQUEST QUEUE- \
            -(DREQ) DEFERED REQUEST QUEUE-

        0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
      -------------------------------------------------
0     *            INDEX TO NEXT ENTRY                *
      -------------------------------------------------
1     *            OPTIONAL-2                         *
      -------------------------------------------------
2     *     *BL*   CLABEL FORM                        *
      -------------------------------------------------
3     *            PIN         * P*    RT     * U* F* C*
      -------------------------------------------------
4     *            OPTIONAL-1                         *
      -------------------------------------------------

1 . OPTIONAL-2    THIS PARAMETER IS REQUIRED IN THOSE
                  OPERATIONS WHERE THE PCBX MUST BE
                  RELOCATED FOLLOWING THE PULL OPERATION.
                  IN THESE CASES, IT'S VALUE REPRESENTS
                  A MOVE COUNT.
2 . C             WAKE PROCESS BIT-OR "DONE" BIT
3 . PIN           PROCESS IDENTITY NUMBER
4 . RT            REQUEST INITIATED TO PROCESS:
              = 0 ABSENCE TRAP
              = 1 I/O SYSTEM REQUEST
              = 2 EXCHANGEDE REQUEST
              = 3 CL EXPANSION REQUEST
              = 4 Z EXPANSION REQUEST
              = 5 EXTRA DATA SEGMENT EXPANSION REQUEST
              = 6 STACK OVERFLOW EXPANSION REQUEST
              = 7 PCBX FREE SPACE EXPANSION REQUEST
              = 8 PXFILE SPACE EXPANSION REQUEST
              = 9 LOCK REQUEST-FORCED OVERLAY OF SEGMENT PARKED
                               IN AREA REQUIRED BY LOCK.
              =10 LOCK REQUEST-COMMUNICATION TO MAM WHEN AREA
                               FOR LOCK WAS INITIALLY FREE.
              =11 LOCK REQUEST-LOCK AREA IS FREE, ALLOCATE THE
                               AREA TO THE LOCK REQUEST.
              >11 TO BE ASSIGNED
5 . U             UNBLOCKED REQUEST BIT
6 . F             FREEZE SEGMENT BIT
7 . P          =1 PROCESS PREPARATION REQUEST. WHEN ON, WORKING
                  SET TESTS AND ADJUSTMENTS WILL BE IGNORED.
8 . BL         =0 CLABEL FORM IS A DST-RELATIVE INDEX TO SEG EN
               =1 CLABEL FORM IS A BLOCK LABEL WITH THE FOLLOWI
```

FIELD DEFINITIONS:

```
( 3: 7)  CSTBLK INDEX
(10: 6)  EN-%300, WHERE EN IS A CST ENTRY NUMBER > %300
```

9 . THE FOLLOWING TABLE DEFINES THE MEANING OF THE OPTIONAL-1 AND OPTIONAL-2 PARAMETER FOR ALL REQUESTS. THEY GENERALLY SPECIFY A MOVE COUNT AND OFFSET FROM SEGMENT BASE WHERE THE PULL OPERATION WILL START. WHEN A MOVE COUNT IS SPECIFIED, A MOVE OF THE STACK FROM THE BASE READ ADDRESS TO THE SEGMENT BASE ADDRESS WILL BE PERFORMED WHEN THE PULL OPERATION IS COMPLETED. THIS COMPLETES THE ADJUSTMENT OF THE PCBX (OR A PART OF IT) AS SPECIFIED BY THE EXPANSIO

RT	OPTIONAL-1	OPTIONAL-2
0	0	0
1	POINTER TO IOC OR DIT	0
2	0	0
3	SIZE OF CL EXPANSION	LENGTH OF PCBX
4	SIZE OF Z EXPANSION	0
5	SIZE OF EDS EXPANSION	0
6	SIZE OF Z EXPANSION	0
7	128 (EXPANSION SIZE)	THE LENGTH (C-A) IN PCBX
8	SIZE OF PXFILE EXPANSION	LENGTH OF PCBX-4
9	0	0/1
10	0	0/1
11	0	0

ENTRY FORMAT-(ICCG) MAM I/O COMPLETION QUEUE-

```
        0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
       -------------------------------------------------
0      *              INDEX TO NEXT ENTRY              *
       -------------------------------------------------
1      *LF*DW*        SEGMENT INDEX (DST-RELATIVE)     *
       -------------------------------------------------
2      *RW*           WORD COUNT                       *
       -------------------------------------------------
3      *   DISPLACEMENT FROM LINK HEAD TO DATA  *  BN  *
       -------------------------------------------------
4      *              MEMORY ADDRESS                   *
       -------------------------------------------------
```

DEFINITION OF TERMS USED

1 . RW — =0 READ
=1 WRITE

2 . BN — BANK NUMBER

3 . THE MEMORY ADDRESS SPECIFIED, IS THE LOCATION OF THE SEGMENT LINK. THE DISK ADDRESS FOR THE SEGMENT IS IN THE 3RD AND 4TH WORDS OF THE LINK-SEE ASSIGNED LINK FORMAT FOR FURTHER DESCRIPTION. THE STARTING ADDRESS FOR THE I/O TRANSFER IS THE MEMORY ADDRESS SPECIFIED PLUS THE VALUE OF THE DISPLACEMENT FIELD.

4 . LF — =1 THE I/O WAS INITIATED TO CLEAR THE AREA FOR A PENDING LOCK REQUEST.

5 . DW — =1 INDICATES THE WRITE WAS INITIATED BY A CALL TO WRITEDSEG. WORD(1).(2:14) WILL CONTAIN A PCB RELATIVE INDEX OF THE REQUESTING PROCESS. THE PROCESS WILL BE IMPEDED UNTIL THE WRITE IS COMPLETED.

ENTRY FORMAT-(LCKQ) LOCK/UNLOCK REQUEST QUEUE-

```
       0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
      ------------------------------------------------
0     *           INDEX TO NEXT ENTRY                *
      ------------------------------------------------
1     *           0                                  *
      ------------------------------------------------
2     * 1*        PINX                               *
      ------------------------------------------------
3     *           0                                  *
      ------------------------------------------------
4     *           0                                  *
      ------------------------------------------------
```

DEFINITION OF TERMS USED

1 . PINX   A PCB RELATIVE PROCESS INDEX.

NOTE*   THIS FORM OF REQUEST IS ENTERED WHEN A PROCESS PREP WAS ABORTED BECAUSE A SEGMENT TO BE PREPED WAS IN THE PROCESS OF BEING LOCKED. THE PROCESS INDEXED BY PINX WILL BE IMPEDED UNTIL THE LOCK IS COMPLETED.

ALL OTHER REQUESTS LINKED IN THE LOKQ LIST HAVE A FORM IDENTICAL TO THE ENTRY FORMAT FOR AREQ ELEMENTS. WHEN THE LOCK IS COMPLETED, THE ENTRIES WILL BE LINKED INTO THE AREQ QUEUE.

ENTRY FORMAT-(CLSQ) OVERLAY SELECTION QUEUE-

```
       0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
     -------------------------------------------------
0    *              INDEX TO NEXT ENTRY              *
     -------------------------------------------------
1    *              INDEX TO LAST ENTRY              *
     -------------------------------------------------
2    * T*  *BL*     FORM                             *
     -------------------------------------------------
3    *              SYSDB-RELATIVE POINTER TO W-SET  *
     -------------------------------------------------
4    *              LAST VALUE OF REFERENCE COUNTER  *
     -------------------------------------------------
```

DEFINITION OF TERMS USED

```
1 . T          =0 SEG SELECTED FROM WORKING SET
               =1 SEG SELECTED FROM PCB. IMPLIES
                  STACK OR XCS.
2 . BL         =0 FORM IS A CST-RELATIVE INDEX TO SEG ENTRY.
               =1 FORM IS A BLOCK LABEL WITH THE FOLLOWING
                  FORMAT:
                  (3:7) CSTBLK INDEX
                  (10:6) EN-%300. WHERE EN IS THE CST ENTRY
                  NUMBER.
```

TABLE STRUCTURE-VDSMAP-

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
VDSMAP(0)	0	0	0	0	0	1	1	1	1	1	1	0	0	0	0	1
	1	1	1	1	0	0	0	0	0	0	0	0	1	1	1	1
								.								
								.								
	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0
								.								
								.								

THE FOLLOWING SYSTEM DB-RELATIVE VALUES ARE REQUIRED IN THE ALLOCATION AND DEALLOCATION OF VIRTUAL DISK SPACE-VDS.

VDSMAP	POINTER TO BIT MAP-THE VDSMAP TABLE-
VDSTART	SECTOR ADDRESS OF THE FIRST ASSIGNABLE AREA ON THE SYSTEM DISK USEABLE FOR VDS.
SCANEND	THE UPPER BOUND OF THE BITMAP VDSMAP. THE CLOSED SET [0,SCANEND] REPRESENT THE USEABLE BITS OF THE MAP.
SCANWALL	THE BIT POSITION WHERE THE NEXT BITMAP SEARCH WILL START. THIS VALUE IS SET TO THE POSITION IMMEDIATEL FOLLOWING THE LAST BIT ASSIGNED ON THE LAST VDS ALLOCATION. THE POSITION WILL WRAP ARROUND TO ZERO WHEN THE UPPER BOUND,SCANEND,IS REACHED.
VDSPAGE	THE SIZE,IN WORDS,OF A VDS PAGE. EACH BIT IN THE BITMAP,VDSMAP,REPRESENTS ONE PAGE. A PAGE MUST BE AN INTEGRAL MULTIPLE OF 128 WORDS. THE MINIMUM SIZE IS 256 WORDS.
VDSL	THE VDS PAGE ALLOCATION TABLE. ONE WORD IS REQUIRED FOR EACH DST ENTRY CONFIGURED.

*NOTE EACH BIT IN THE BIT MAP REPRESENTS ONE PAGE. IF A BIT POSITION IS ON(=1),THE PAGE IS FREE. IF THE BIT POSITION IS OFF(=0),THE PAGE IS ASSIGNED.

TABLE STRUCTURE-VDSL-

```
                 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
                 ------------------------------------------------
VDSL(0)          * D* Q* R* S*              NP                  *
                 ------------------------------------------------
                 * D* Q* R* S*              NP                  *
                 ------------------------------------------------
                 *                    .                         *
                 *                    .                         *
                 ------------------------------------------------
```

DEFINITION OF TERMS USED

1 . D =1 SEGMENT IS REQUIRED BY MAM. INDICATES WCNT IN MAM WILL BE DECREMENTED WHEN REQUEST IS COMPLETED.

2 . Q =1 AN I/O ENTRY HAS ALREADY BEEN QUEUED FOR A WRITE REQUEST REFERENCING THIS DATA SEGMENT THIS IS VALID FOR A WRITE REQUEST ONLY.

3 . R =1 RESTART THIS REQUEST WHEN THE CURRENT I/O I COMPLETED.

4 . S =1 SEGMENT IS A STACK
=0 SEGMENT IS AN EDS

5 . NP THE NUMBER OF VDS "PAGES" ASSIGNED THIS ENTRY

NOTE* THE ENTRIES CONTAIN A ZERO IS UNASSIGNED

TABLE STRUCTURE-BANKTAB-

```
            ------------------------------
BANKTAB(0)  * FIRST                      *
            ------------------------------
         1  * LAST                       *
            ------------------------------
         2  * #LOCKS (LB)-BANK 0         *
            ------------------------------
         3  * SUM LOCK SPACE (LB)        *
            ------------------------------
         4  * #LOCKS (UB)-BANK 0         *
            ------------------------------
         5  * SUM LOCK SPACE (UB)        *
            ------------------------------
```

A SIX WORD ENTRY IS SET UP FOR EACH MEMORY BANK CONFIGURED ON THE SYSTEM. THE MAXIMUM LENGTH OF THE BANKTAB TABLE IS 24 WORDS. WORDS 2 -5 OF EACH ENTRY ARE INITIALIZED TO ZERO. LINKED MEMORY IN EACH BANK IS BOUNDED BY THE FOUR WORD MARKER (%100000,-4,0,%100001). IN BANK ZERO,THE LOWER BOUND MARKER IMMEDIATELY FOLLOWS CORE RESIDENT MEMORY. THESE MARKERS ARE USED TO GUARANTEE THAT BANK OVERFLOWS DO NOT OCCUR. FIRST AND LAST CONTAIN THE ADDRESS OF THE MARKER ON THE LOWER AND UPPER BANK BOUNDRYS RESPECTIVELY.

NOTE* A POINTER TO BANKTAB(0) IS LOCATED IN SYSGLOB(%50).
THE NUMBER OF CONFIGURED MEMORY BANKS IS CONTAINED
IN THE PARAMETER ,NBANKS, LOCATED IN SYSGLOB(%47).
THE VALUE OF NBANKS MUST BE IN THE RANGE [0,3].
THE ACTUAL NUMBER OF BANKS IS NBANKS+1.

DEFINITION OF MEMORY MANAGEMENT FUNCTIONS

DEFINITION OF MEMORY MANAGEMENT FUNCTIONS

IN SEVERAL OF THE FOLLOWING DECLARATIONS, A COMMON DEFINITION IS REQUIRED FOR A PASSED PARAMETER. TO SHORTEN THE TEXT I HAVE INTRODUCED THE FOLLOWING NOTATION TO DESCRIBE A FORWARD DEFINITION. THE LIST OF COMMON DEFINITIONS WILL FOLLOW THIS DISCUSSION. THE NOTATION USED IS THE FOLLOWING. THE DEFINITION WILL BE GIVEN IN THE FORWARD TEXT. A REFERENCE TO THE DEFINITION LATER ON WILL BE MADE BY PRECEEDING THE NAME WITH AN ASTERISK (*).

PINX	PCB-RELATIVE INDEX OF THE PROCESS TO WHICH THE SEGMENT BELONGS. IF THE SEGMENT BELONGS TO THE CALLERS PROCESS THE VALUE MAY BE ZERO.
EN	CST OR DST ENTRY NUMBER.
LDEV	THE LOGICAL DEVICE NUMBER OF THE DISK WHERE THE CODE OR DATA SEGMENT RESIDES. DATA SEGMENTS ALWAYS RESIDE ON THE SYSTEM DISK, LOGICAL DEVICE NUMBER 1.
DISKADR	THE HIGH AND LOW ORDER DISK ADDRESS OF THE SPECIFIED SEGMENT.

CLABEL — TWO FORMS OF CLABEL EXIST:

1. (2:1)=0
   (3:13) CST-RELATIVE INDEX TO SL OR DST ENTRY

2. (2:1)=1
   (3: 7) CSTBLK INDEX.
   (10:6) EN-%300, WHERE EN IS THE ENTRY NUMBER OF A SEGMENT IN THE CST EXTENSION BLO

*** DECLARATION ***

```
INTEGER PROCEDURE MAKEPRESENT(EN,TEST,TYPE,PIOQ);
   VALUE        EN,TEST,TYPE,PIOQ;
   INTEGER      EN,     TYPE,PIOQ;
   LOGICAL         TEST;
   OPTION       UNCALLABLE,PRIVILEGED,EXTERNAL;
```

*** FUNCTION ***

THIS PROCEDURE IS CALLED TO MAKE PRESENT AN ABSENT CODE OR DATA SEGMENT. MAKEPRESENT CAN BE CALLED FOR A PRESENT SEGMENT AS WELL TO SET THE REQUESTED CONDITIONS. THE CALLER MAY SPECIFY THAT THE CALL IS UNBLOCKED,I.E.-CONTROL IS RETURNED TO THE CALLER AFTER CONSTRUCTING A MAM REQUEST FOR AN ABSENT SEGMENT. THE UNBLOCKED FORM OF CALL MAY BE MADE BY THE I/O SYSTEM ONLY AT THIS TIME. THE NORMAL FORM IS TO BE BLOCKED IF THE SEGMENT IS ABSENT UNTIL THE REQUEST IS COMPLETED BY MAKING PRESENT THE SEGMENT.

*** PARAMETER DEFINITION ***

```
EN                 *EN

TEST.( 0: 8)       IF THE REQUESTED SEGMENT BELONGS TO THE
                   CALLERS PROCESS,THE VALUE MAY BE ZERO.
                   IF THE SEGMENT BELONGS TO A PROCESS OTHER
                   THAN THE CALLERS, THE PIN OF THAT PROCESS
                   MUST BE SUPPLIED.

     ( 8: 2)       RESERVED. MUST BE ZERO.

     (10: 3)=0     CALLED FROM ABSENCE TRAP ROUTINE.
            =1     CALLED FROM I/O SYSTEM ROUTINE.
            =2     CALLED FROM EXCHANGEDB.
            >2     ILLEGAL REQUEST. RESERVED FOR MAM.

     (13: 1)=0     THE REQUEST IS BLOCKED.
            =1     THE REQUEST IS UNBLOCKED.

     (14: 1)=0     NO FREEZE REQUESTED.
            =1     IF I/O SYSTEM CALL, THEN INCREMENT THE I/O
                   FREEZE COUNTER.
                   IF NOT AN I/O SYSTEM CALL, THEN INCREMENT
                   THE FREEZE COUNTER.
```

(15: 1)=0	REQUEST IS FOR A CODE SEGMENT.
=1	REQUEST IS FOR A DATA SEGMENT.
TYPE	THE VALUE WILL BE ZERO IN ALL CASES EXCEPT THE FOLLOWING: IF THE REQUEST WAS INITIATED BY A CALL TO A CODE SEGMENT, TYPE WILL CONTAIN A PROGRAM LABEL SPECIFING THE CST AND RELATED STT NUMBER.
PIOQ	ZERO IN ALL CASES EXCEPT REQUESTS FROM THE I/O SYSTEM. IF AN I/O SYSTEM REQUEST, THEN PIOQ WILL CONTAIN A SYSTEM DB-RELATIVE POINTER TO AN IOQ ELEMENT IF THE REQUEST IS FOR A DATA SEGMENT,OTHERWISE,A SYSTEM DB-RELATIVE POINTER TO A DIT.
MAKEPRESENT	THE VALUE WILL BE ZERO EXCEPT FOR THE FOLLOWING CASE: IF THE REQUEST WAS INITIATED BY A PCAL TO AN ABSENT OR PRESENT CODE SEGMENT AND TYPE CONTAINS A NON-ZERO LABEL, THEN THE VALUE OF THE PB-RELATIVE ADDRESS SPECIFIED BY THE ORDERED PAIR (STT,CST) WILL BE RETURNED.

*** CONDITION CODES ***

CC=CCE	REQUEST GRANTED. SEGMENT WAS ALREADY PRESENT.
=CCG	REQUEST GRANTED. SEGMENT IS BEING MADE PRESENT. THIS CODE IS APPLICABLE FOR UNBLOCKE REQUESTS ONLY.
=CCL	REQUEST FAILED. NO-MEM,I/O ERROR, OR OTHER FATAL EVENT OCCURED.

*** ACTION TAKEN ON REQUEST COMPLETION IF ABSENT ***

REQUESTS INITIATED BY A SEGMENT TRAP OR EXCHANGEDB CALL ARE COMPLETED BY CALLING THE SCHEDULING PROCEDURE MAM'DONE. REQUESTS ORIGINATING FROM THE I/O SYSTEM ARE COMPLETED AS FOLLOWS:

1. REQUEST TO MAKE A CODE SEGMENT PRESENT.
   THE DRIVER FROZEN BIT IN WORD ZERO OF THE DLT IS SET, AND AWAKEIO IS CALLED.

2. REQUEST TO MAKE A DATA SEGMENT PRESENT.

THE DATA FROZEN BIT IN THE IOQ ELEMENT IS SET, AND AWAKEIO IS CALLED.

```
*** DECLARATION ***

PROCEDURE LOCKSEG(EN,TEST,PINX):
   VALUE       EN,TEST,PINX;
   INTEGER     EN,      PINX;
   LOGICAL        TEST;
   OPTION      UNCALLABLE,PRIVILEGED,EXTERNAL;

*** FUNCTION ***

   THIS FUNCTION IS CALLED TO LOCK A SEGMENT IN MAIN MEMORY AT
OR NEAR A BANK BOUNDRY.

*** PARAMETER DEFINITION ***

EN               *EN

TEST.( 0:13)     RESERVED. MUST BE ZERO.

      (13: 1)=0  SEGMENT WILL BE LOCKED IN THE MOST CON-
                 VIENENT MEMORY BANK.
             =1  SEGMENT WILL BE LOCKED IN BANK 0.

      (14: 1)=0  ALLOCATE SPACE FOR SEGMENT AT OR NEAR UPPER
                 BANK BOUNDRY. THIS IS THE DEFAULT CASE.
             =1  ALLOCATE SPACE FOR SEGMENT AT OR NEAR LOWER
                 BANK BOUNDRY.

      (15: 1)=0  IF A CODE SEGMENT.
             =1  IF A DATA SEGMENT.

PINX             *PINX

*** CONDITION CODES ***

CC=CCE           REQUEST GRANTED OR SEGMENT WAS CORE
                 RESIDENT.
  =CCL           REQUEST DENIED. INVALID ENTRY SPECIFICATION
                 OR NO SPACE AVALABLE.
  =CCG           REQUEST FAILED. SEGMENT IS FROZEN OR AN I/O
                 ERROR OCCURED.
```

*** DECLARATION ***

```
PROCEDURE UNLOCKSEG(EN,TEST,PINX);
   VALUE          EN,TEST,PINX;
   INTEGER        EN,     PINX;
   LOGICAL           TEST;
   OPTION         UNCALLABLE,PRIVILEGED,EXTERNAL;
```

*** FUNCTION ***

THIS FUNCTION IS CALLED TO UNLOCK A CODE OR DATA SEGMENT LOCKED IN MAIN MEMORY.

*** PARAMETER DEFINITION ***

EN	*EN
TEST.(0:15)	RESERVED. MUST BE ZERO.
(15: 1)=0	UNLOCK CODE SEGMENT.
=1	UNLOCK DATA SEGMENT.
PINX	*PINX

*** CONDITION CODES ***

CC=CCE	REQUEST GRANTED AND SEGMENT IS NOW UNLOCKED OR SEGMENT WAS CORE RESIDENT.
=CCG	REQUEST GRANTED, SEGMENT WAS NOT UNLOCKED SINCE LOCK COUNTER DID NOT GO TO ZERO.
=CCL	REQUEST DENIED, INVALID ENTRY SPECIFICATION

*** DECLARATION ***

```
PROCEDURE FREEZE(EN,TEST,PINX);
   VALUE       EN,TEST,PINX;
   INTEGER     EN,      PINX;
   LOGICAL        TEST;
   OPTION      UNCALLABLE,PRIVILEGED,EXTERNAL;
```

*** FUNCTION ***

FREEZE A CODE OR DATA SEGMENT IN MAIN MEMORY. THE REQUEST WILL BE GRANTED IF AND ONLY IF THE SEGMENT IS PRESENT.

*** PARAMETER DEFINITION ***

```
EN                 *EN

TEST        =0     REQUEST IS FOR A CODE SEGMENT.
            =1     REQUEST IS FOR A DATA SEGMENT.

PINX               *PINX
```

*** CONDITION CODES ***

```
CC=CCE             REQUEST GRANTED.
  =CCL OR CCG      REQUEST FAILS. SEGMENT IS NOT PRESENT.
```

*** COMMENTS ***

A SECONDARY ENTRY POINT EXISTS FOR THIS PROCEDURE. THE ENTRY NAME IS IOFREEZE. WHEN THE I/O SYSTEM WANTS TO FREEZE A SEGMENT, IT MUST CALL THE FUNCTION IOFREEZE. A SEPARATE COUNTER FOR I/O FREEZES IS MAINTAINED.

*** DECLARATION ***

```
PROCEDURE UNFREEZE(EN,TEST,PINX);
   VALUE          EN,TEST,PINX;
   INTEGER        EN,     PINX;
   LOGICAL           TEST;
   OPTION         UNCALLABLE,PRIVILEGED,EXTERNAL;
```

*** FUNCTION ***

UNFREEZE A CODE OR DATA SEGMENT WHICH IS FROZEN IN MAIN MEMORY. THE FREEZE COUNTER OF THE SEGMENT IS DECREMENTED BY ONE. THE SEGMENT IS UNFROZEN WHEN THE COUNTER GOES TO ZERO.

*** PARAMETER DEFINITION ***

EN		*EN
TEST	=0	REQUEST IS FOR A CODE SEGMENT.
	=1	REQUEST IS FOR A DATA SEGMENT.
PINX		*PINX

*** CONDITION CODES ***

CC=CCE	REQUEST GRANTED. SEGMENT IS UNFROZEN.
=CCG	REQUEST GRANTED BUT SEGMENT WAS NOT UNFROZEN SINCE FREEZE COUNTER WAS NOT DECREMENTED TO ZERO.
=CCL	REQUEST FAILED. SEGMENT NOT PRESENT.

*** COMMENTS ***

A SECONDARY ENTRY POINT NAMED IOUNFREEZE EXITS FOR THIS PROCEDURE. THE I/O SYSTEM MUST CALL THIS PROCEDURE TO DECREMENT THE I/O FREEZE COUNTER.

DEFINITION OF MEMORY MANAGEMENT FUNCTIONS

*** DECLARATION ***

```
INTEGER PROCEDURE ALCSTBLOCK(NUM);
   VALUE         NUM;
   INTEGER       NUM;
   OPTION        UNCALLABLE,PRIVILEGED,EXTERNAL;
```

*** FUNCTION ***

ALLOCATE A BLOCK OF NUM CONTIGUOUS ENTRIES FROM THE CST EXTENSION BLOCK.

*** PARAMETER DEFINITION ***

NUM ALLOCATE THIS NUMBER OF CONTIGUOUS ENTRIES.

ALCSTBLOCK A CSTBLK TABLE INDEX IS RETURNED IF THE CALL IS SUCCESSFUL. THE CSTBLK ENTRY WILL CONTAIN A DST-RELATIVE INDEX TO THE ASSIGNED CST EXTENSION BLOCK. THE INDEX RETURNED TO THE CALLER MUST BE STORED IN THE PCB OF THE PROCESS BEING ALLOCATED THE BLOCK.

*** CONDITION CODES ***

CC=CCE REQUEST GRANTED.
  =CCL REQUEST FAILED. INSUFFICIENT NUMBER OF ENTRIES LEFT TO SATISFY REQUEST.

*** COMMENTS ***

SOMEHOW OR OTHER, THE INDEX RETURNED BY ALCSTBLOCK MUST BE STORED IN THE PCB OF THE PROCESS FOR WHOM THE CST BLOCK ENTRIES ARE BEING ALLOCATED.

*** DECLARATION ***

```
PROCEDURE DEALCSTBLOCK(EIX);
   VALUE        EIX;
   INTEGER      EIX;
   OPTION       UNCALLABLE,PRIVILEGED,EXTERNAL;
```

*** FUNCTION ***

DEALLOCATE THE BLOCK OF CST ENTRIES IN THE CST EXTENSION BLOCK INDEXED BY CSTBLK(EIX).

*** PARAMETER DEFINITION ***

EIX — SEE DESCRIPTION OF ALCSTBLOCK FOR DEFINITION.

*** CONDITION CODES ***

CC=CCE — REQUEST GRANTED.
=CCL — REQUEST FAILED. INDEX.CSTBX, DOES NOT REFER TO A VALID CST BLOCK IN THE CST EXTENSION.

*** COMMENTS ***

MAIN MEMORY FOR PRESENT SEGMENTS REFERENCED IN THE CST BLOCK WILL BE RELEASED.

*** DECLARATION ***

```
PROCEDURE PUTCSTBLOCK(EIX,LSEGNUM,MASK,LDEV,DISKADR);
   VALUE       EIX,LSEGNUM,MASK,LDEV,DISKADR;
   INTEGER     EIX,LSEGNUM,MASK,LDEV;
   DOUBLE                           DISKADR;
   OPTION      UNCALLABLE,PRIVILEGED,EXTERNAL;
```

*** FUNCTION ***

THIS PROCEDURE IS CALLED TO INITIALIZE THE INDIVIDUAL ENTRIES OF A JUST ALLOCATED CST BLOCK.

*** PARAMETER DEFINITION ***

EIX		INDEX RETURNED TO CALLER OF ALCSTBLOCK. SEE PROCEDURE DESCRIPTION FOR DEFINITION.
LSEGNUM		THE LOGICAL SEGMENT NUMBER. ZERO IS THE FIRST LOGICAL SEGMENT NUMBER.
MASK	(0: 1)	WILL BE SET TO ONE TO INDICATE ABSENCE
	(1: 1)	SEGMENT MODE BIT
	(2: 1)	REFERENCE BIT. SHOULD BE ZERO
	(3: 1)	SEGMENT TRACE BIT
	(4:12)	(SEGMENT SIZE)/4
LDEV		*LDEV
DADR		*DISKADR

*** CONDITION CODES ***

CC=CCE	REQUEST GRANTED.
=CCG	LSEGNUM < 0 OR LSEGNUM > MAX ASSIGNED ENTRY.

*** DECLARATION ***

```
PROCEDURE PLACECNOLST(CLABEL,WSX,CODE);
   VALUE         CLABEL,WSX,CODE;
   INTEGER       CLABEL,WSX,CODE;
   OPTION        UNCALLABLE,PRIVILEGED,EXTERNAL;
```

*** FUNCTION ***

PLACE A CODE OR DATA SEGMENT ON THE MEMORY MANAGEMENT OVERLAY SELECTION QUEUE. THE SEGMENT SPECIFIED BY INDEX IS MARKED ABSENT IF THE REQUEST IS GRANTED.

*** PARAMETER DEFINITION DEFINITION ***

CLABEL		*CLABEL
WSX		SYSTEM DB-RELATIVE POINTER TO THE CALLERS WORKING SET. WILL BE ZERO IF PINX OF CALLER WAS ZERO.
CODE	=0	ENTRY HAS BEEN SELECTED FROM A WORKING SET.
	=1	STACK.
	=2	EXTRA DATA SEGMENT.

*** DECLARATION ***

```
PROCEDURE RELDS(EN,PINX,XDSFLAG);
   VALUE       EN,PINX,XDSFLAG;
   INTEGER     EN,PINX;
   LOGICAL             XDSFLAG;
   OPTIN       UNCALLABLE,PRIVILEGED.EXTERNAL;
```

*** FUNCTION ***

THIS PROCEDURE IS CALLED TO PLACE A DATA SEGMENT ON THE OVERLAY SELECTION QUEUE. IF THE XDSFLAG IS TRUE, THE EXCHANGEDB COUNTER WILL BE DECREMENTED BY ONE.

*** PARAMETER DESCRIPTION ***

EN	*EN
PINX	PCB-RELATIVE INDEX OF PROCESS WHOM ENTRY EN BELONGS TO.
XDSFLAG=0	EXCHANGEDB COUNTER IS NOT DECREMENTED.
=1	THE EXCHANGEDB COUNTER IS DECREMENTED BY ONE IF IT IS NONZERO.

*** DECLARATION ***

```
PROCEDURE PUTCST(EN,MASK,LDEV,DISKADR);
   VALUE       EN,MASK,LDEV,DISKADR;
   INTEGER     EN,MASK,LDEV;
   DOUBLE      DISKADR;
   OPTION      UNCALLABLE,PRIVILEGED,EXTERNAL;
```

*** FUNCTION ***

THIS PROCEDURE IS CALLED TO INITIALIZE A CST ENTRY WHICH WAS ALLOCATED BY CALLING THE FUNCTION GETENTRY OR GETENTRYS. THIS PROCEDURE CAN NOT BE USED TO INITIALIZE A CST EXTENSION BLOCK ENTRY; USE THE PROCEDURE PUTCSTBLOCK FOR THAT PURPOSE. AN ERROR WILL BE RETURNED IF THE ENTRY HAS ALREADY BEEN INITIALIZED.

*** PARAMETER DEFINITION ***

```
EN            *EN

LDEV          *LDEV

MASK   ( 0: 1)    WILL BE SET TO ONE TO INDICATE ABSENCE
       ( 1: 1)    SEGMENT MODE BIT
       ( 2: 1)    REFERENCE BIT, SHOULD BE ZERO
       ( 3: 1)    SEGMENT TRACE BIT
       ( 4:12)    (SEGMENT SIZE)/4

DISKADR       *DISKADR
```

*** DECLARATION ***

```
INTEGER PROCEDURE GETDATASEG(MEMSIZE,VDSIZE);
   VALUE          MEMSIZE,VDSIZE;
   INTEGER        MEMSIZE,VDSIZE;
   OPTION         UNCALLABLE,PRIVILEGED,EXTERNAL;
```

*** FUNCTION ***

THIS FUNCTION IS CALLED TO CREATE AN EXTRA DATA SEGMENT. A DST ENTRY AND VDS IS ALLOCATED. THE ENTRY IS INITIALIZED TO THESE VALUES. IF THE REQUEST IS GRANTED, THE ENTRY NUMBER IS RETURNED TO THE CALLER. THIS PROCEDURE HAS AN ENTRY POINT CALLED GETSTACK-SEE COMMENT BELOW.

*** PARAMETER DEFINITION ***

MEMSIZE	THIS IS THE INITIAL SIZE THAT WILL BE ALLOCATED IN MAIN MEMORY WHEN THE SEGMENT IS MADE PRESENT. THIS WILL ALSO BE THE SIZE OF THE VDS ALLOCATED IF THE VDSIZE PARAMETER IS LESS THAN OR EQUAL TO MEMSIZE. IT SHOULD BE NOTED THAT MEMSIZE CAN BE LESS THAN VDSIZE. THIS ALLOWS THE ALLOCATION OF A LARGER VDS AREA FOR FUTURE SEGMENT EXPANSION IF ALTDSEGSIZE CALLS ARE ANTICIPATED.
VDSIZE	IF THIS VALUE IS GREATER THAN MEMSIZE, THIS AMOUNT OF VDS WILL BE ALLOCATED. IF LESS THAN MEMSIZE, MEMSIZE WORTH OF VDS WILL BE ALLOCATED.
GETDATASEG	IF THE REQUEST IS GRANTED, THE DST ENTRY NUMBER WILL BE RETURNED. IF THE REQUEST FAILS, A ZERO WILL BE RETURNED.

*** CONDITION CODES ***

CC=CCE	REQUEST GRANTED.
=CCG	REQUEST DENIED. NO FREE DST ENTRY IS AVAILABLE.
=CCL	REQUEST DENIED. INSUFFICIENT VDS AVAILABLE.

*** COMMENT ***

THIS PROCEDURE HAS A SECONDARY ENTRY POINT NAMED GETSTACK. THE VALUE OF VDSIZE IS DIREGARDED FOR A STACK ALLOCATION. THE INITIAL AMOUNT OF VDS ALLOCATED IS COMPUTED AS FOLLOWS:

```
INITIAL VDS := MEMSIZE+F1+F2
  WHERE
  F1 =1152      REQUIRED FOR SYSTEM CLEANUP IF AN ABORT DUE
                TO STACKOVERFLOW OCCURS.
  F2 = 384      REQUIRED FOR POSSIBLE PXFIXED EXPANSION

NOTE*    THE VALUES OF F1 AND F2 IMPLY THAT MAXDATA FOR
         A STACK MUST NOT EXCEED 31232 WORDS.
```

*** DECLARATION ***

```
PROCEDURE LINKFA(ADR,N);
   VALUE      ADR,N;
   INTEGER        N;
   DOUBLE     ADR;
   OPTION     UNCALLABLE,PRIVILEGED,EXTERNAL;
```

*** FUNCTION ***

LINK THE MEMORY AREA WITH THE EXTENDED STARTING ADDRESS ADR AND SIZE N WORDS INTO THE LINKED MEMORY FREE SPACE LIST. IF THE ADJACENT AREAS ARE FREE THEY ARE COMBINED TO FORM A LARGER FREE AREA.

*** PARAMETER DEFINITION ***

ADR   THE EXTENDED MEMORY ADDRESSOF THE AREA HEAD.

N   THE SIZE OF THE AREA TO BE LINKED.

*** DECLARATION ***

```
PROCEDURE DELINKFA(ADR);
   VALUE        ADR;
   DOUBLE       ADR;
   OPTION       UNCALLABLE,PRIVILEGED,EXTERNAL;
```

*** FUNCTION ***

THIS FUNCTION IS CALLED TO UNLINK A FREE AREA WITH EXTENDED STARTING ADDRESS ADR FROM THE LINKED MEMORY FREE SPACE LIST.

*** PARAMETER DEFINITION ***

ADR THE EXTENDED MEMORY ADDRESS OF THE AREA HEAD.

*** DECLARATION ***

```
PROCEDURE EXHVDS(INX,SIZE,PXFIX);
   VALUE        INX,SIZE,PXFIX;
   INTEGER      INX,SIZE;
   INTEGER      POINTER  PXFIX;
   OPTION       UNCALLABLE,PRIVILEGED,EXTERNAL;
```

*** FUNCTION ***

THIS FUNCTION IS CALLED TO ALLOCATE A LARGER VDS AREA FOR THE STACK OF THE CALLER. A > CONDITION WILL BE RETURNED IF THE REQUEST FAILS, OTHERWISE AN = CONDITION IS RETURNED. IF THE REQUEST IS GRANTED, THE SEGMENT WILL BE WRITTEN TO THE LARGER AREA WHEN NEXT OVERLAYED.

*** PARAMETER DEFINITION ***

INX	CST-RELATIVE INDEX TO THE DATA SEGMENT ENTRY.
SIZE	THE NEW VDS SIZE REQUESTED.
PXFIX	A DB-RELATIVE POINTER TO THE BASE OF THE PXFIXED AREA IN THE STACK PCBX.

*** CONDITION CODES ***

CC=CCE	REQUEST GRANTED.
=CCG	REQUEST DENIED. INSUFFICIENT VDS AVAILABLE.

DEFINITION OF MEMORY MANAGEMENT FUNCTIONS

*** DECLARATION ***

```
DOUBLE PROCEDURE GETVDSPACE(EN,N);
   VALUE         EN,N;
   INTEGER       EN,N;
   OPTION        UNCALLABLE,PRIVILEGED,EXTERNAL;
```

*** FUNCTION ***

ALLOCATE VDS FOR THE DST WITH ENTRY NUMBER EN. IF N WORDS OF SPACE ARE ALLOCATED, THE STARTING SECTOR ADDRESS IS RETURNED TO THE CALLER. A ZERO IS RETURNED IF THE REQUEST FAILS.

*** PARAMETER DEFINITION ***

EN	*EN
N	THE SIZE OF THE AREA TO BE ALLOCATED.
GETVDSPACE	A DOUBLE WORD DISK ADDRESS IS RETURNED TO THE CALLER IF THE REQUEST WAS GRANTED. ONLY THE LOW ORDER 20 BITS ARE SIGNIFICANT. A ZERO IS RETURNED IF THE REQUEST FAILS.

*** DECLARATION ***

```
PROCEDURE RELVDSPACE(EN);
   VALUE         EN;
   INTEGER       EN;
   OPTION        UNCALLABLE,PRIVILEGED,EXTERNAL;
```

*** FUNCTION ***

DEALLOCATE VDS ASSIGNED TO THE DST WITH ENTRY NUMBER EN.

*** PARAMETER DEFINITION ***

EN   *EN

*** DECLARATION ***

```
PROCEDURE TAKEFROMWS(WS);
   VALUE          WS;
   INTEGER POINTER WS;
   OPTION         UNCALLABLE,PRIVILEGED.EXTERNAL;
```

*** FUNCTION ***

SELECT FROM THE WORKING SET WS AN ENTRY REFERENCING A PRESENT SEGMENT WITH THE SMALLEST REFERENCE COUNTER VALUE. IF MORE THAN ONE OF THE ENTRIES ARE TIED FOR THE SMALLEST, THE FIRST ENCOUNTERED IN THE SET IS SELECTED. IF AN ENTRY IS FOUND, IT WILL BE PLACED ON THE OVERLAY SELECTION QUEUE, OLSQ.

*** PARAMETER DEFINITION ***

WS	SYSTEM DB-RELATIVE POINTER TO THE BASE OF THE WORKING SET.

*** CONDITION CODES ***

CC=CCE	NO ENTRY WAS SELECTED, HENCE NO ENTRY WAS PLACED ON THE OVERLAY SELECTION LIST.
=CCG	ENTRY WAS SELECTED FROM SET WS AND PLACED ON THE OVERLAY SELECTION LIST.

*** COMMENT ***

IF A CLABEL HAS BEEN SELECTED, THE ENTRY IS DELETED FROM THE WORKING SET. IF THE SELECTION WAS UNSUCCESSFUL IT IMPLIES THAT THE SET WAS EMPTY, OR ALL OF THE SET ENTRIES REFERENCE ABSENT OR NONOVERLAYABLE SEGMENTS.

*** DECLARATION ***

```
INTEGER PROCEDURE SCANVOSMAP(NP);
   VALUE        NP;
   INTEGER      NP;
   OPTION       UNCALLABLE,PRIVILEGED,EXTERNAL;
```

*** FUNCTION ***

THIS FUNCTION IS CALLED TO PERFORM A BITMAP SEARCH OF THE VOS BITMAP FOR A STRING OF NP CONTIGUOUS BITS IN AN ON POSITION. IF THE SEARCH IS SUCCESSFUL, THE STARTING BIT POSITION IS RETURNED TO THE CALLER. IF THE SEARCH FAILS, A ZERO IS RETURNED.

*** PARAMETER DEFINITION ***

NP	THE NUMBER OF VOS PAGES REQUIRED. EACH BIT IN THE TABLE REPRESENTS A PAGE.
SCANVOSMAP	IF THE SEARCH WAS SUCCESSFUL, THE STSRTING BIT POSITION IS RETURNED. IF THE SEARCH FAILS, A ZERO IS RETURNED.

*** DECLARATION ***

```
PROCEDURE CHANGEBITS(B,N,FLG);
   VALUE          B,N,FLG;
   INTEGER        B,N;
   LOGICAL            FLG;
   OPTION         UNCALLABLE,PRIVILEGED,EXTERNAL;
```

*** FUNCTION ***

THIS FUNCTION IS CALLED TO TURN ON OR OFF N CONTIGUOUS BITS IN THE VDS BITMAP, VDSMAP. THE STARTING BIT POSITION IS B AND THE BITS WILL BE TURNED ON IF FLG IS TRUE, OTHERWISE, THEY WILL BE TURNED OFF.

*** PARAMETER DEFINITION ***

B	THE STARTING BIT POSITION.
N	THE NUMBER OF BITS TO TURN ON OR OFF.
FLG	IF TRUE, THE BITS WILL BE SET, OTHERWISE THE BITS WILL BE RESET.

*** DECLARATION ***

```
INTEGER PROCEDURE SEGSTATE(INX,P);
   VALUE        INX,P;
   INTEGER      INX;
   INTEGER POINTER P;
   OPTION       UNCALLABLE,PRIVILEGED,EXTERNAL;
```

*** FUNCTION ***

THIS FUNCTION IS CALLED TO CHECK THE CORE RESIDENT,LOCKED, FREEZE OR IOFREEZE SETTING FOR A DATA SEGMENT REQUIRING A STACK SIZE CHANGE ENVOKED BY A CALL TO DLSIZE, ZSIZE OR SOME OTHER ROUTINE.

*** PARAMETER DEFINITION ***

INX	A DST-RELATIVE SEGMENT INDEX
P	A POINTER TO THE BASE OF THE STACK PCBX.
SEGSTATE	IF THE CONDITION CODE IS CCE THEN THE REFERENCE COUNTER VALUE OF THE ENTRY IS RETURNED.

*** CONDITION CODES ***

CC=CCE	THE SEGMENT IS NOT FROZEN,CORE RESIDENT, LOCKED OR IN I/O FREEZE MODE.
=CCL	THE SEGMENT IS CORE RESIDENT,LOCKED OR FROZEN IN MAIN MEMORY.
=CCG	YOUR PROCESS WAS IMPEDED UNTIL AN I/O FREEZE WAS REMOVED. OK TO PROCEED NOW.

*** DECLARATION ***

```
PROCEDURE EXPANDREQ(INX,INC,CODE.MVCNT):
   VALUE          INX,INC,CODE,MVCNT;
   INTEGER        INX,INC,CODE,MVCNT;
   OPTION         UNCALLABLE,PRIVILEGED,EXTERNAL:
```

*** FUNCTION ***

THIS FUNCTION IS CALLED BY ROUTINES THAT ENVOKE A STACK OR EDS EXPANSION. IT BUILDS A MAM REQUEST AND LINKS IT INTO THE MAM REQUEST QUEUE, AREQ. MAM WILL OVERLAY THE SEGMENT SPECIFIED BY INX AND RECALL IT. IF A STACK EXPANSION, MAM WILL MODIFY THE PCBX TO ADJUST ITS POSITION RELATIVE TO THE BASE OF THE ALLOCATED MAIN MEMORY.

*** PARAMETER DEFINITION ***

INX	THE DST-RELATIVE INDEX OF THE STACK OR EDS.
INC	AN OPTIONAL VALUE USED BY MAM WHEN THE SEGMENT IS RECALLED.
CODE	A CODE WHICH SPECIFIES THE TYPE OF REQUEST TO MAM.
MVCNT	WHEN A RELOCATION OF THE PCBX IS REQUIRED FOLLOWING A STACK EXPANSION, THIS VALUE REPRESENTS THE LENGTH OF THE MOVE.

*** DECLARATION ***

```
PROCEDURE MAMIC(LADR,INC,INX,LDEV,COUNT,RW);
   VALUE        LADR,INC,INX,LDEV,COUNT,RW;
   INTEGER      INC,INX,LDEV,COUNT;
   DOUBLE       LADR;
   LOGICAL                   RW;
   OPTION       UNCALLABLE,PRIVILEGED,EXTERNAL;
```

*** FUNCTION ***

THIS FUNCTION IS CALLED BY THE PROCESS MAM TO BUILD AN I/O REQUEST FOR MEMORY MANAGEMENT PURPOSES. THE ROUTINE WILL SELECT A FREE ENTRY FROM THE MTAB TABLE, AND AFTER FILLING THE ENTRY WITH INFO, LINK IT TO THE TAIL OF THE REQUEST LIST FOR THAT LOGICAL DEVICE.

*** PARAMETER DEFINITION ***

LADR	THE EXTENDED ADDRESS OF THE SEGMENT LINK HEAD.
LDEV	*LDEV
COUNT	THE NUMBER OF WORDS TO TRANSFER.
INX	A DST-RELATIVE INDEX TO THE CODE OR DATA SEGMENT BEING WORKED ON.
INC	THE DISPLACEMENT FROM THE SEGMENT BASE WHERE THE I/O WILL START. THIS VALUE WILL ALWAYS BE ZERO FOR WRITES.
RW	READ/WRITE FLAG. IF A READ THEN THE VALUE WILL BE ZERO. IF A WRITE THE VALUE WILL BE %100000.

*** DECLARATION ***

```
PROCEDURE MAMIODONE(DIT,ERRORX);
   VALUE          DIT,ERRORX;
   INTEGER POINTER DIT;
   INTEGER             ERRORX;
   OPTION         UNCALLABLE,PRIVILEGED,EXTERNAL;
```

*** FUNCTION ***

THIS FUNCTION IS CALLED BY THE I/O SYSTEM TO COMPLETE A MEMORY MANAGEMENT I/O REQUEST INITIATED BY A CALL TO THE PROCEDURE MAMIC. THE HEAD REQUEST IS DELINKED FROM THE LOGICAL DEVICE SPECIFIED BY DIT, AND THE ERROR CONDITION IS SAVED IN THE ENTRY. THE ENTRY IS NOW LINKED INTO THE MAM I/O COMPLETION QUEUE IOCQ AND MAM IS WOKEN UP. WHEN MAM RUNS, IT WILL PROCESS THE REQUEST. THE I/O SYSTEM WILL EXAMINE THE LIST POINTER IN THE DIT TO SEE IF ANOTHER REQUEST IS QUEUED.

*** PARAMETER DEFINITION ***

DIT	THIS IS A POINTER TO THE DEVICE INFORMATION TABLE WHERE THE I/O WAS COMPLETED.
ERRORX	THIS IS THE I/O ERROR CODE. A ZERO INDICATES THAT THE I/O WAS COMPLETED SUCCESSFULLY WITH NO ERRORS. A NONZERO VALUE INDICATES AN ERROR OCCURED.

*** DECLARATION ***

```
DOUBLE PROCEDURE ALTSTKSIZE(INX,INC,SZ,PCBX);
   VALUE           INX,INC,SZ,PCBX;
   INTEGER POINTER PCBX;
   INTEGER                INX,INC,SZ;
   OPTION          UNCALLABLE,PRIVILEGED,EXTERNAL;
```

*** FUNCTION ***

THIS FUNCTION IS CALLED BY THE PROCEDURES DLSIZE AND ALTPXFLILESIZE TO CONTRACT THE STACK OF THE CALLER.

*** PARAMETER DEFINITION ***

PCBX	A DB-RELATIVE POINTER TO THE BASE OF THE PCBX.
INX	THE DST-RELATIVE INDEX OF THE STACK.
INC	A NEGATIVE VALUE INDICATING THE SIZE OF THE STACK REDUCTION.
SZ	THE CURRENT SIZE OF THE STACK SEGMENT.
ALTSTKSIZE	THE ADDRESS OF THE AREA TO BE LINKED INTO THE FREE SPACE LIST.

*** DECLARATION ***

```
INTEGER PROCEDURE ALTPXFILESIZE(SIZECHANGE);
   VALUE          SIZECHANGE;
   INTEGER        SIZECHANGE;
   OPTION         UNCALLABLE,PRIVILEGED,EXTERNAL;
```

*** FUNCTION ***

THIS FUNCTION IS CALLED BY THE FILE SYSTEM TO EXPAND OR CONTRACT THE SIZE OF THE PXFILE AREA. SIZECHANGE IS THE CHANGE IN SIZE OF THE PXFILE AREA. THE AREA IS EXPANDED OR CONTRACTED AT THE HIGH END,I.E-HIGHEST INDEX. IF SIZECHANGE IS NEGATIVE THE AREA WILL BE CONTRACTED. IF SIZECHANGE IS POSITIVE, BUT NOT ZERO, THE AREA WILL BE EXPANDED. NO CHANGE WILL BE MADE IF SIZECHANGE IS ZERO. THE PHYSICAL INCREMENTS WILL BE ROUNDED UP TO AN INTEGRAL MULTIPLE OF 128 WORDS. FOR EXAMPLE, A REQUEST TO INCREASE THE SIZE BY 100 WORDS WOULD RESULT IN AN INCREMENT OF 128 WORDS. A REQUEST TO DECREASE THE AREA BY 100 WORDS WOULD LEAVE THE AREA SIZE UNCHANGED. A REQUEST TO DECREASE THE AREA SIZE BY 150 WORDS, ON THE OTHER HAND, WOULD REDUCE THE AREA BY 128 WORDS. THE ACTUAL SIZE OF THE PXFILE AREA IS CONTAINED IN PXFILE(0).

*** PARAMETER DEFINITION ***

SIZECHANGE	THE SIZE BY WHICH THE AREA WILL BE CHANGE
ALTPXFILESIZE	=0 REQUEST GRANTED. =1 REQUEST PARTIALLY GRANTED. THE AREA WAS INCREASED BY THE DIFFERENCE BETWEEN MAXDATA AND THE CURRENT STACK SIZE. =2 REQUEST DENIED. STACK IS FROZEN,CORE RESIDENT OR LOCKED. =3 REQUEST DENIED. A VDS EXPANSION WAS REQUIRED TO SATISFY REQUEST AND IT FAILED.

*** DECLARATION ***

```
PROCEDURE BLKLABEL;
   OPTION      UNCALLABLE,PRIVILEGED,EXTERNAL;
```

*** FUNCTION ***

THIS FUNCTION IS CALLED TO CONVERT A CST BLOCK LABEL INTO A DST-RELATIVE INDEX. THE LABEL HAS THE FORM OF A CLABEL. IT ASSUMES THAT THE CLABEL RESIDES A Q-4. THE INDEX WILL BE RETURNED TO Q-4.

*** DECLARATION ***

```
DOUBLE PROCEDURE ENTRYINDEX(EN,PINX,SEGFLG);
   VALUE      EN,PINX,SEGFLG;
   INTEGER    EN,PINX;
   LOGICAL    SEGFLG;
   OPTION     UNCALLABLE,PRIVILEGED,EXTERNAL;
```

*** FUNCTION ***

THIS FUNCTION IS CALLED TO CONVERT A CODE OR DATA SEGMENT ENTRY NUMBER INTO A DST-RELATIVE INDEX.

*** PARAMETER DEFINITION ***

EN	*EN
PINX	*PINX
SEGFLG	=1 EN IS A DST ENTRY NUMBER =0 EN IS A CST ENTRY NUMBER
ENTRYINDEX	THE CALCULATED DST-RELATIVE INDEX IS RETURNED IN THE FIRST WORD OF THE DOUBLET. THE SECOND WORD WILL CONTAIN A CLABEL IF REQUIRED.

DEFINITION OF MEMORY MANAGEMENT FUNCTIONS

*** DECLARATION ***

```
PROCEDURE EXPBELOW / EXPABOVE;
   OPTION      UNCALLABLE,PRIVILEGED,EXTERNAL;
```

*** FUNCTION ***

THE PROCEDURES ARE CALLED BY THE FUNCTION LINKFA. ALL OF THE PARAMETERS ARE REFERENCED THROUGH Q- ADDRESSING. THE FUNCTION OF THE PROCEDURES IS TO CONCATENATE FREE AREAS ABOVE OR BELOW A TARGET FREE , WHERE THE FREE AREAS ARE SEPARATED BY A MOVEABLE ASSIGNED SEGMENT. THESE CONCATENATIONS REQUIRE THE MOVEMENT OF THE ASSIGNED AREAS. THE MOVES WILL NOT TAKE PLACE IF A MAMIO IS IN OPERATION IN THE RANGE OF THE MOVE. IF THE MOVES ARE ALLOWED, THEY WILL BE DONE BY THE ROUTINE EXPCOM.

DEFINITION OF MEMORY MANAGEMENT FUNCTIONS

*** DECLARATION ***

```
PROCEDURE EXPCOM(FLG);
   VALUE       FLG;
   LOGICAL     FLG;
   OPTION      UNCALLABLE,PRIVILEGED,EXTERNAL;
```

*** FUNCTION ***

THIS FUNCTION IS CALLED BY THE PROCEDURES EXPABOVE OR EXPBELOW TO PERFORM THE PHYSICAL CONCATENATION OF FREE AREAS SEPARATED BY A MOVEABLE ASSIGNED AREA THAT ISN'T TOO DAMM BIG. THE CONCATENATION WILL NOT BE PERFORMED IF A MAM I/O IS IN OPERATION IN THE RANGE OF THE MOVE.

*** PARAMETER DEFINITION ***

FLG =0 A LOW TO HIGH ADDRESS MEMORY MOVE IS PERFORMED.
=1 A HIGH TO LOW ADDRESS MEMORY MOVE IS PERFORMED.

*** DECLARATION ***

```
INTEGER PROCEDURE CSTCONV(EN,PINX);
   VALUE        EN,PINX;
   INTEGER      EN,PINX;
   OPTION       UNCALLABLE,PRIVILEGED,EXTERNAL;
```

*** FUNCTION ***

THIS FUNCTION IS CALLED TO CALCULATE THE DST-RELATIVE INDEX OF A CST WITH ENTRY NUMBER EN.

*** PARAMETER DEFINITION ***

EN	*EN
PINX	*PINX
CSTCONV	A DST-RELATIVE INDEX TO THE CST ENTRY.

*** DECLARATION ***

```
LOGICAL PROCEDURE ONOLST(CLABEL,WS);
   VALUE       CLABEL,WS;
   INTEGER     CLABEL;
   INTEGER     POINTER WS;
   OPTION      UNCALLABLE,PRIVILEGED,EXTERNAL;
```

*** FUNCTION ***

THIS FUNCTION IS CALLED TO DETERMINE IF THE SEGMENT SPECIFIED IS IN RESIDENCE ON THE OVERLAY SELECTION LIST OLSQ. IF IN RESIDENCE, IT WILL BE DELETED FROM THE OLSQ QUEUE AND MARKED PRESENT.

*** PARAMETER DEFINITION ***

CLABEL	*CLABEL
WS	A SYSTEM DB-RELATIVE POINTER TO A WORKING SET. MAY BE ZERO IF THE PINX OF THE CALLER WAS ZERO.
ONOLST	=0 SEGMENT IS NOT ON OLSQ. =1 SEGMENT WAS ON OLSQ AND HAS BEEN MARKED PRESENT.

*** DECLARATION ***

```
PROCEDURE CHANGESTATE(EN);
   VALUE      EN;
   INTEGER    EN;
   OPTION     UNCALLABLE,PRIVILEGED,EXTERNAL;
```

*** FUNCTION ***

THIS FUNCTION IS CALLED TO SET ALL PROCESS' REFERENCING THE DATA SEGMENT WITH ENTRY NUMBER EN TO THE ABS STATE.

*** PARAMETER DEFINITION ***

EN   *EN

DEFINITION OF MEMORY MANAGEMENT FUNCTIONS

*** DECLARATION ***

```
PROCEDURE PREF;
   OPTION      UNCALLABLE,PRIVILGED,INTERNAL;
```

*** FUNCTION ***

THIS FUNCTION IS CALLED BY MAM TO BUILD REQUESTS FOR PREPARING A PROCESS TO RUN. IT WILL BUILD UP TO FOUR AREQ REQUESTS.

*** DECLARATION ***

```
PROCEDURE LINKSINGLEQ / LINKDOUBLEQ(HEAD,INDEX);
   VALUE      HEAD,INDEX;
   INTEGER    HEAD,INDEX;
   OPTION     UNCALLABLE,PRIVILEGED,EXTERNAL;
```

*** FUNCTION ***

THESE FUNCTIONS ARE CALLED TO LINK AN MTAB ENTRY INTO ONE OF THE QUEUES AREQ,DREQ,OLSQ,LOKQ OR IOCQ. IF SINGLE LINKAGE IS SPECIFIED, BY CALLING LINKSINGLEQ, THE ENTRY WILL BE LINKED IN THE FORWARD DIRECTION ONLY. IF LINKDOUBLEQ IS CALLED, THE ENTRY WILL BE LINKED FORWARD AND BACKWARD. CURRENTLY, THE ONLY DOUBLY LINKED CALL IS FOR LINKING INTO THE OVERLAY SELECTION QUEUE, OLSQ.

*** PARAMETER DEFINITION ***

HEAD	= 1 LINK INTO AREQ
	= 3 LINK INTO OLSQ
	= 5 LINK INTO IOCQ
	= 7 LINK INTO LOKQ
	= 9 LINK INTO DREQ
INDEX	MTAB ENTRY INDEX

DEFINITION OF MEMORY MANAGEMENT FUNCTIONS

*** DECLARATION ***

```
PROCEDURE UNLINKSINGLEQ / UNLINKDOUBLEQ(HEAD,INDEX);
   VALUE       HEAD,INDEX;
   INTEGER     HEAD,INDEX;
   OPTION      UNCALLABLE,PRIVILEGED,EXTERNAL;
```

*** FUNCTION ***

THESE FUNCTIONS ARE CALLED TO UNLINK AN MTAB ENTRY FROM A DOUBLY OR SINGLELY LINKED QUEUE.

*** PARAMETER DEFINITION ***

```
HEAD              = 1 UNLINK FROM AREQ
                  = 3 UNLINK FROM CLSQ
                  = 5 UNLINK FROM IOCQ
                  = 7 UNLINK FROM LOKQ
                  = 9 UNLINK FROM DREQ

INDEX                 MTAB ENTRY INDEX
```

*** DECLARATION ***

```
INTEGER PROCEDURE BUILDQENTRY(HEAD,INDEX,TEST,OPT1,OPT2);
   VALUE       HEAD,INDEX,TEST,OPT1,OPT2;
   INTEGER     HEAD,INDEX,TEST,OPT1,OPT2;
   OPTION      UNCALLABLE,PRIVILEGED,EXTERNAL;
```

*** FUNCTION ***

THIS FUNCTION IS CALLED TO ALLOCATE A FREE ENTRY FROM THE MTAB TABLE AND LINK IT ,IF SPECIFIED, INTO ONE OF THE QUEUES: AREQ,OLSQ,DREQ,LCKQ OR IOCQ. THE ENTRY WILL BE INITIALIZED WITH THE VALUES SPECIFIED IN THE CALLING SEQUENCE.

*** PARAMETER DEFINITION ***

HEAD	= 0 DON'T LINK = 1 LINK INTO AREQ = 3 LINK INTO OLSQ (DOUBLE LINK) = 5 LINK INTO IOCQ = 7 LINK INTO LCKQ = 9 LINK INTO DREQ
INDEX	SEE TABLE DESCRIPTION FOR MEANING
TEST	"
OPT1	"
OPT2	"
BUILDQENTRY	THE MTAB ENTRY INDEX ALLOCATED IS RETURNED.

*** DECLARATION ***

```
PROCEDURE RTNMTABENTRY(INDEX);
   VALUE       INDEX;
   INTEGER     INDEX;
   OPTION      UNCALLABLE,PRIVILEGED,EXTERNAL;
```

*** FUNCTION ***

THIS FUNCTION IS CALLED TO RETURN AN MTAB ENTRY BACK TO THE MTAB FREE LIST.

*** PARAMETER DEFINITION ***

INDEX AN MTAB ENTRY INDEX.

*** DECLARATION ***

```
PROCEDURE CHECKDEFERALS(CLABEL);
   VALUE       CLABEL;
   INTEGER     CLABEL;
   OPTION      UNCALLABLE,PRIVILEGED,EXTERNAL;
```

*** FUNCTION ***

THIS FUNCTION IS CALLED TO CHECK FOR DEFERED REQUESTS FOR THE SEGMENT SPECIFIED BY CLABEL. IF THERE ARE DEFERED REQUESTS FOR THE SEGMENT, THEY WILL BE UNLINKED FROM THE DREQ QUEUE AND LINKED INTO THE QUEUE AREG.

*** PARAMETER DEFINITION ***

CLABEL   *CLABEL

*** DECLARATION ***

```
PROCEDURE CHEKCTRS(INX,RT,FZREGF);
   VALUE        INX,RT,FZREQF;
   INTEGER      INX,RT;
   LOGICAL      FZREGF;
   OPTION       UNCALLABLE,PRIVILEGEL,EXTERNAL;
```

*** FUNCTION ***

THIS FUNCTION IS CALLED TO CHECK FOR A REQUIRED INCREMENT OF THE FREEZE OR EXCHANGEDB COUNTERS FOR THE SEGMENT WITH A DST-RELATIVE INDEX OF INX.

*** PARAMETER DEFINITION ***

```
INX                   DST-RELATIVE ENTRY INDEX

RT              = 2   INCREMENT EXCHANGEDB COUNTER
                <>2   NO ACTION

FZREGF.(14:1)=0       NO ACTION
             =1       RT = 1 THEN INCREMENT I/O FREEZE
                      COUNTER OTHERWISE INCREMENT FREEZE
                      COUNTER.
```

*** DECLARATION ***

```
PROCEDURE MOVEMOUT(LINX,CLABEL,PINX,BANK,STKINX,ULFLG);
   VALUE      LINX,CLABEL,PINX,BANK,STKINX,ULFLG;
   INTEGER    LINX,CLABEL,PINX,BANK,STKINX,ULFLG;
   OPTION     UNCALLABLE,PRIVILEGED,EXTERNAL;
```

*** FUNCTION ***

THIS FUNCTION IS CALLED BY LOCKSEG TO INITIATE THE OVERLAY OF SEGMENTS IN AN AREA OF MAIN MEMORY WHICH IS REQUIRED BY A SEGMENT LOCK REQUEST.

*** PARAMETER DEFINITION ***

LINX — THE INDEX OF A SYSSBUF ENTRY WHICH HAS BEEN ALLOCATED FOR THE PROCESSING OF THE LOCKSEG REQUEST.THE ENTRY IS ALLOCATED IN LOCKSEG AND WILL BE RETURNED BY MAM WHEN THE REQUEST IS COMPLETED.

CLABEL — *CLABEL

PINX — *PINX

BANK — THE BANK NUMBER OF THE MEMORY AREA REQUIRED FOR THE SEGMENT LOCK REQUEST.

STKINX
```
( 0: 1) = 0 SEGMENT BEING LOCKED IS ABSENT
        = 1 SEGMENT BEING LOCKED IS PRESENT
( 1:15) DST RELATIVE INDEX OF THE STACK OF THE
        REQUESTOR.
```

ULFLG
```
= 0 AREA IS ON LOWER BANK BOUNDRY
= 1 AREA IS ON UPPER BANK BOUNDRY.
```

*** DECLARATION ***

```
INTEGER PROCEDURE ALTDSEGSIZE(EN,SIZE);
   VALUE       EN,SIZE;
   INTEGER     EN,SIZE;
   OPTION      UNCALLABLE,PRIVILEGED,EXTERNAL;
```

*** FUNCTION ***

THIS FUNCTION IS CALLED TO CHANGE THE SIZE OF A DATA SEGMENT SPECIFIED BY THE DST ENTRY NUMBER EN.

*** PARAMETER DEFINITION ***

EN	DST ENTRY NUMBER
SIZE	THE AMOUNT TO INCREMENT OR DECREMENT THE CURRENT SEGMENT SIZE.
ALTDSEGSIZE	THE SIZE OF THE SEGMENT FOLLOWING THE CHANGE.

*** CONDITION CODES ***

CC=CCE	THE REQUEST WAS SUCCESSFUL.
=CCL	THE REQUEST FAILED. THE SEGMENT SPECIFIED BY EN IS CORE RESIDENT, LOCKED OR FROZEN.
=CCG	THE REQUEST FAILED. THE NEW SEGMENT SIZE IS <= 0 OR THE NEW SIZE EXCEEDS THE AMOUNT OF VDS ALLOCATED THE SEGMENT.

*** DECLARATION ***

```
PROCEDURE RELDATASEG / RELCODESEG(EN);
   VALUE       EN;
   INTEGER     EN;
   OPTION      UNCALLABLE,PRIVILEGED,EXTERNAL;
```

*** FUNCTION ***

THESE FUNCTIONS ARE CALLED TO RETURN THE RESOURCES ALLOCATED A DATA SEGMENT OR A CODE SEGMENT WITH AN ENTRY INDEX < %300.IF RELDATASEG IS CALLED, THE VDS, MAIN MEMORY, AND DST ENTRY WILL BE RETURNED. IF A CODE SEGMENT ,THE MAIN MEMORY AND CST ENTRY WILL BE RETURNED.

*** PARAMETER DEFINITION ***

EN   *EN

*** DECLARATION ***

```
INTEGER PROCEDURE BESTBANK(ULFLG);
   VALUE       ULFLG;
   LOGICAL     ULFLG;
   OPTION      UNCALLABLE,PRIVILEGED,EXTERNAL;
```

*** FUNCTION ***

THE FUNCTION IS CALLED BY THE LOCKSEG PROCEDURE TO SELECT THE BEST BANK IN WHICH TO LOCK A SEGMENT.

*** PARAMETER DEFINITION ***

ULFLG	= 0 SELECT AREA FROM LOWER BANK BOUNDRY = 1 SELECT AREA FROM UPPER BANK BOUNDRY
BESTBANK	THE BANK NUMBER OF THE BEST BANK IS RETURNED

DEFINITION OF MEMORY MANAGEMENT FUNCTIONS

*** DECLARATION ***

```
LOGICAL PROCEDURE LSEARCH(LSIZE,LINX,BASE,BANK,ULFLG);
   VALUE       LSIZE,LINX,BASE,BANK,ULFLG;
   INTEGER     LSIZE,LINX,BASE,BANK;
   LOGICAL     ULFLG;
   OPTION      UNCALLABLE,PRIVILEGED,EXTERNAL;
```

*** FUNCTION ***

THIS FUNCTION IS CALLED BY LOCKSEG TO SELECT FROM A MEMORY BANK AN AREA OF APPROPRIATE SIZE FOR THE LOCK REQUEST.

*** PARAMETER DEFINITION ***

LSIZE		THE SIZE OF THE AREA REQUIRED.
LINX		THE INDEX OF A SYSSBUF ALLOCATED TO PROCESS THE LOCK REQUEST.
BASE		AN INDEX WHICH IS USED TO START THE SEARCH ON THE UPPER OR LOWER BANK BOUNDRY.
BANK		THE BANK NUMBER OF THE MEMORY BANK TO SEARCH.
ULFLG	= 0	ALLOCATE AREA ON LOWER BANK BOUNDRY.
	= 1	ALLOCATE AREA ON UPPER BANK BOUNDRY.
LSEARCH	= 0	SEARCH FAILED. NO AREA OF SUFFICIENT SIZE CAN BE FOUND ON BOUNDRY SPECIFIED.
	= 1	REQUEST WAS SUCCESSFUL.

C1LC Bell

~~cc: Dick Stuart~~

HEWLETT hp PACKARD

GENERAL SYSTEMS · 5303 Stevens Creek Blvd., Santa Clara, California 95050, Telephone 408 249-7020

FROM: Russ Blake DATE: February 17, 1976

TO: SUBJECT: Scheduling on MPE Series II

This note updates three sections of the MPE ERS, namely Scheduling Processes, the QUANTUM command, and the SHOWQ command.

In addition to process scheduling, MPE II incorporates priority driven selection of spooled jobs for execution, as well as priority controlled output of spooled listings. These features permit users to specify the relative importance of jobs and listings. Overall control of spooling and job selection is given to the operator, through console commands to monitor and manage the system. These facilities are unchanged from MPE Version C, and details can be found in the MPEC Console Operator's Manual (p.5-1 Job Evolution & Control; p.6-1 Output Spooling; p.6-9 Outpri) and also the MPEC Reference Manual (p.3-13 input priority; p.3-14, 5-17 output priority).

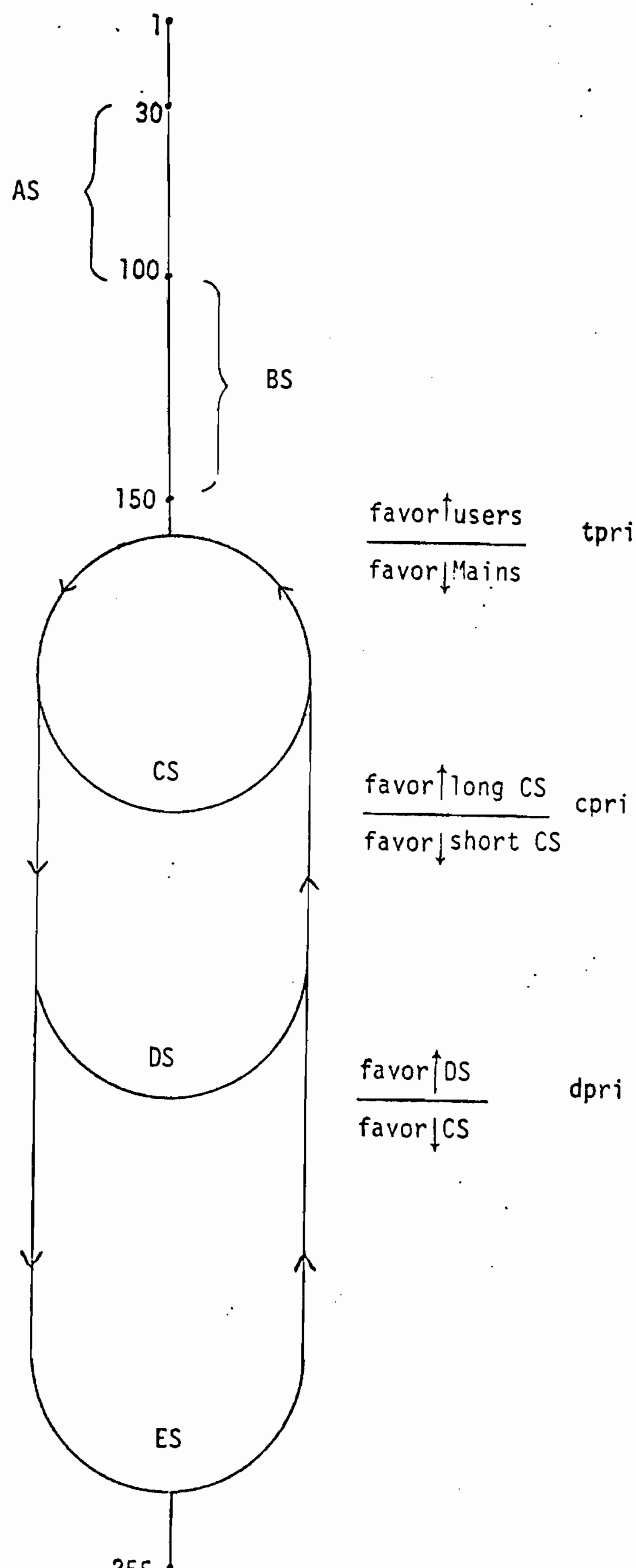
SCHEDULING PROCESSES

All processes sharing the system access its resources in an orderly manner, under the direction of MPE Series II. The system places runnable processes in the master scheduling queue in order of their priority. The CPU is given to the highest priority process in the master queue which is able to use it.

On the master queue, a small priority number (as low as 1) corresponds to a high priority, whereas successively lower priorities are given by successively larger priority numbers (up to 255). These priority numbers are known only to extra-capability (System Supervisor or Privilege Mode) users. Standard users see the master queue as a set of logical areas called "subqueues". To each subqueue there corresponds a priority class (AS, BS, CS, DS, or ES), a scheduling type (linear or circular), and a range of priority numbers normally given to processes in that subqueue.

Linearly scheduled processes (AS and BS) have fixed priority, and access the CPU on an as-needed basis, highest priority first. The priority of a circularly scheduled process is dynamically adjusted by an amount depending on priority class (i.e. CS, DS, or ES). The priority will tend to fall as the CPU is consumed, but rise while waiting for the CPU. This insures optimum equipment utilization and fair sharing of the system.

HP3000 Series II Master Queue Structure



In detail the subqueues are (see diagram)

AS is a linear subqueue containing processes of very high priority. Its priority range is 30-99 and it is presently used only by MPE.

Scheduling Type: Linear
Priority Range: 30-99

BS is a linear subqueue containing processes of very high priority. It is accessible to users having MAXPRI = BS. Normally, priority for a BS process is 100. However, by specifying a rank > 0 in the CREATE or GET PRIORITY intrinsics, the process may be set in the master queue at min (100 + rank, 149).

Scheduling Type: Linear
Priority Range: 100-149

CS is a circular subqueue generally devoted to interactive sessions. A CS process which uses its quantum of CPU will be lowered in priority, but not below the C Subqueue Priority Limit, called cpri (which may be set in SYSDUMP or the QUANTUM Command).

Scheduling Type: Circular
Priority Range: 150-cpri

DS is a circular subqueue generally devoted to batch jobs. A DS process which uses its quantum of CPU will be lowered in priority more rapidly than a CS Process, but not below the D Subqueue Priority Limit, or dpri (which may be set in SYSDUMP or QUANTUM Command).

Scheduling Type: Circular
Priority Limits: 150-dpri

ES is a circular subqueue generally used for so-called "idle" processes. When an ES process consumes a quantum of CPU, its priority is set to 250. Such a process will have a minimal impact on the performance of porcesses in the other subqueues.

Scheduling Type: CIrcular
Priority Limits: 150-250

QUANTUM

QUANTUM <quantum>, <tpri>, <cpri>, <dpri>

The QUANTUM Command is used to change the system's scheduling parameters during system operation. These parameters have been selected to provide maximum control of process scheduling: the command is restricted to users with System Supervisor Capability. Due to the possibly dramatic effects on performance, the command should be used with caution. The current values of the parameters can be displayed with the SHOWQ Command.

The scheduling parameters may also be changed during system configuration (see SYSDUMP). The configured parameters are always overridden immediately when QUANTUM is issued.

<quantum> Quantum

The quantum is specified in milliseconds of CPU consumed. During system operation, circularly scheduled processes which receive CPU service are periodically lowered in priority. Quantum specifies the period: after consuming Quantum milliseconds of CPU, the priority of circularly scheduled process is lowered, an amount depending on priority class. It may then continue to use the CPU. However, if such a process must wait for the CPU, its priority is automatically incremented. A quantum of about 1/2 second (400-600 milliseconds) will usually produce smooth operation. If the quantum is too short, excessive swapping may result from too frequent process switching. If the quantum is too long, response may become erratic.

<tpri> Terminal Priority

When a circularly scheduled user process resumes execution after reading terminal input, it is scheduled at this priority. The tpri should be set at or near 150. To favor the response to system commands, set the tpri a few points deeper than 150 (at, say, 152 or 153). Setting tpri to a much greater value will have unpredictable results, which depend on system demand and configuration.

<cpri> C Subqueue Priority Limit

A CS process which expires its quantum will be lowered in priority, but not below cpri. With cpri close to tpri, long CS transactions are favored over short ones. With cpri far from tpri, responses to short transactions are favored somewhat over those to longer transactions. However, setting cpri less than tpri or greater than dpri will have unpredictable results.

<dpri> D Subqueue Priority Limit

A DS process which expires its quantum will be lowered in priority, but not below dpri. With dpri close to cpri, DS processes receive increased thruput, possibly at the expense of CS process reponse. As the distance between cpri and dpri is increased, the impact of a DS process on CS processes is decreased, CS process reponse may improve, and DS process thruput may suffer somewhat.

By managing these parameters, system operation may be balanced as is most appropriate.

SHOWQ

The SHOWQ Command is provided to give the System Supervisor insight into the dynamic operation of the system. SHOWQ is used to display the queues of processes within MPE. If executed at the system console, the queues are displayed directly on the console. If executed from job or session, the command requires the user to have System Supervisor Capability and will print on $STDLIST.

The display is divided into three major columns. In the right-hand column are listed, in order of high-to-low priority, those processes which currently or will imminently require the CPU in order to continue. The CPU will automatically be given to the first process in the right-hand list which is able to use it.

In the center column are listed, in high-to-low priority order, those processes which are willing to yield their main memory resources to other processes. Memory is taken from lower priority processes first. A process is placed in the center column when it will not require the CPU for a relatively long time.

In the left-hand column are listed, in numerical order, processes which have no main memory resources. These processes are waiting on even longer-term events, and will again contend for the CPU (and return to the right) when those events occur.

We stress that the migration of a process from one column to the next is entirely automatic and is in no way the concern of the programs themselves.

The folling information is printed for each process:

$$\left\{\begin{matrix}L\\C\\D\\E\end{matrix}\right\}\begin{bmatrix}M\\U\end{bmatrix}\ \langle\text{pin}\rangle\ \left[\ \#\left\{\begin{matrix}J\\S\end{matrix}\right\}\ \langle\text{jobnum}\rangle\ \right]$$

Q: Scheduling subqueue

L	A linearly scheduled process on the AS, BS or Master Queue
C	A circularly scheduled process on the CS queue
D	A circularly scheduled process on the DS queue
E	A circularly scheduled process on the ES queue
blank	Process in Creation (queue unknown)

PIN: Process Identification Number

M	This is a job/session Main process, running the Command Interpreter for some job/session
U	This is a User process, running a system or user program for the job or session

(Processes which are neither Main nor user processes are system processes)

<pin> Process identification Number for this process

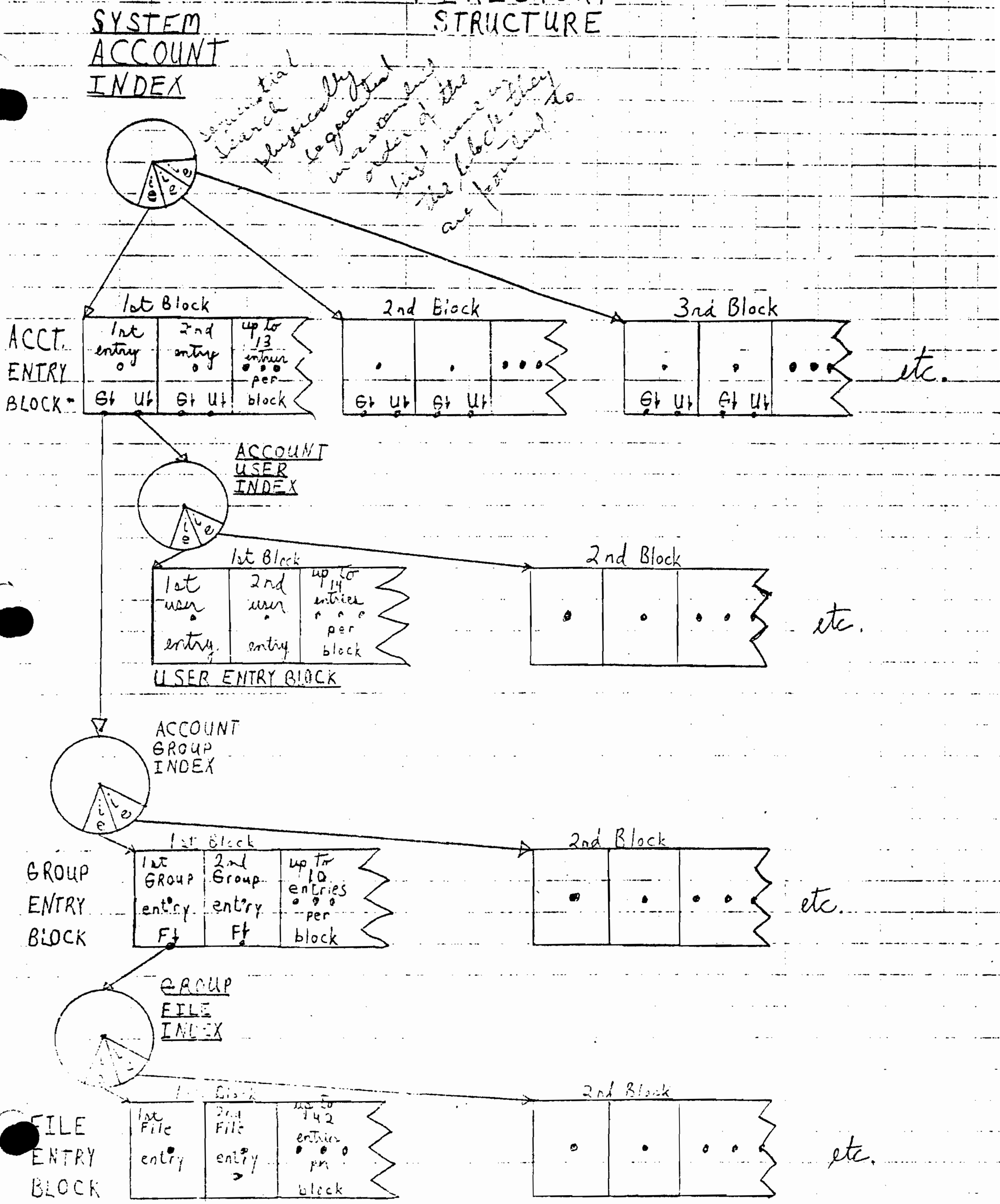
JOBNUM: Job Number

(for User processes, job information is included:)

J	Process is executing in a batch job
S	Process being executed from a session
<jobnum>	Job or Session Number

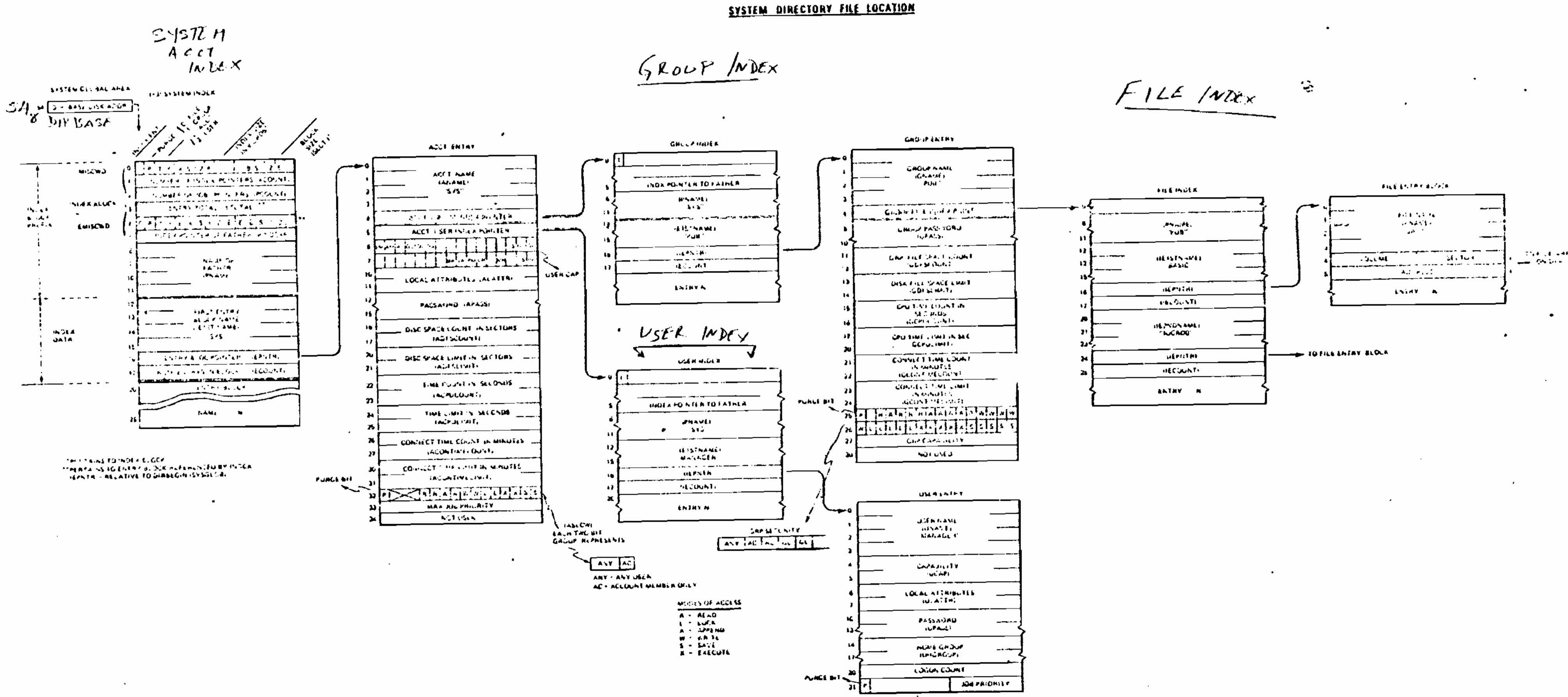
After displaying the process queues, SHOWQ prints the number of processes and the scheduling parameters currently in effect. The scheduling parameter may be established during system configuration (SYSDUMP), or with the QUANTUM command. For a detailed explanation of these parameters, see the QUANTUM command description.

DIRECTORY STRUCTURE



ie ≡ index entry

SYSTEM DIRECTORY FILE LOCATION



~~see~~ COMMUNICATOR

Size of Directory

Sectors =

$$6 + A\left(1_{\text{group index}} + 1_{\text{user index}} + 2_{\substack{\text{group entry}\\\text{block}}} + 2_{\substack{\text{user entry}\\\text{block}}}\right)$$

$$+ G\left(2_{\text{file index}} + 2_{\text{file entry}}\right)$$

$$+ \frac{18\text{ words}}{128\text{ words}}\left(U - 14_{\substack{\text{entries in}\\\text{initial block}}}\right) + \frac{25\text{ words}}{128\text{ words}}\left(G - 10_{\substack{\text{entries in}\\\text{initial block}}}\right)$$

$$+ \frac{6\text{ words}}{128\text{ words}}\left(F - 42_{\substack{\text{entries in}\\\text{initial block}}}\right)$$

or

2401 = 6 + 60 + 2000 + 75 + 10 + 250

$$\text{Sectors} = 6 + 6*A + 4*G + .15U + .2G + .05F$$

where 6 = Fudge Factor

10 = A = Total number of accounts in system

500 G = Total number of groups in system

500 U = Total number of users in system

5000 F = Total number of files in system

BLOCK	# sectors/block	# entries/block	size of entry (words)	Max. # entries/limb	Usable Maxima
SYS INDEX*	3	62	6		
ACCT ENTRY	3	13	29	806	650
GRP INDEX*	1	19	6		
GRP ENTRY	2	10	25	190	155
FILE INDEX*	2	41 ← 40	6		
FILE ENTRY	2	42	6	1680	1385
USER INDEX*	1	19	6		
USER ENTRY	2	14	18	266	200

2 entries in The ~~SYS~~ INDEX are used up - for HEAD & tail pointers

Max # entries in index = (((#sectors in index)*128)-10)/6

128 = #words/sector

10 = Prefix size in index

6 = index entry size

*note only one block in an index

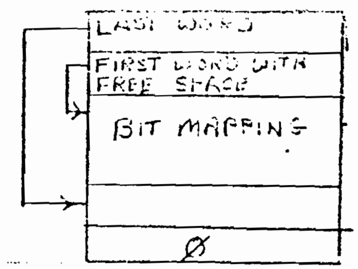
SECTOR %000034

START OF BIT MAP

000:	000041	000006	000000	000000	017777	177474	177777	177777
010:	177777	177777	177777	177777	177777	177777	177777	177777
020:	177777	177777	177777	177777	177777	177777	177777	177777
030:	177777	177777	177777	177777	177777	177777	177777	177777
040:	177777	177777	000000	051505	020050	030051	020124	047440
050:	041511	047443	031466	020111	046114	042507	040514	020122
060:	042523	050117	047123	042440	052117	020103	044517	021510
070:	031467	020123	045443	031476	020127	[illegible]	[illegible]	020122
100:	042043	032060	020104	040524	040440	053517	051104	020040
110:	020125	047111	020111	051440	021443	031066	020125	020123
120:	044117	052514	042040	041105	020040	052122	052503	052111
130:	021443	032061	020102	052506	043105	051040	041510	042503
140:	045523	052515	020040	020117	053105	051106	020103	054514
150:	020040	020040	020104	040524	020050	047522	042123	024510
160:	042057	051440	043105	051122	042504	020050	044075	020111
170:	051475	054120	024443	032062	020125	047104	042506	044516

DEFINES AVAILABLE/USED SECTORS IN DIRECTORY.

EACH BIT IN A WORD CORRESPONDS TO A SECTOR
1 = AVAILABLE
Ø = USED



SECTOR %000037

SYSTEM INDEX

000:	120503	000001	000000	000002	023503	000000	020040	020040
010:	020040	020040	051531	051440	020040	020040	000010	000002
020:	000002	000002	000002	000002	000002	000002	000002	000002
030:	000002	000002	000002	000002	000002	000002	000002	000002
040:	000002	000002	000002	000002	000002	000002	000002	000002
050:	000002	000002	000002	000002	000002	000002	000002	000002
060:	000002	000002	000002	000002	000002	000002	000002	000002
070:	000002	000002	000002	000002	000002	000002	000002	000002
100:	000002	000002	000002	000002	000002	000002	000002	000002
110:	000002	000002	000002	000002	000002	000002	000002	000002
120:	000002	000002	000002	000002	000002	000002	000002	000002
130:	000002	000002	000002	000002	000002	000002	000002	000002
140:	000002	000002	000002	000002	000002	000002	000002	000002
150:	000002	000002	000002	000002	000002	000002	000002	000002
160:	000002	000002	000002	000002	000002	000002	000002	000002
170:	000002	000002	000002	000002	000002	000002	000002	000002

ACCT ENTRY BLOCK = %10 + (DIRBASE) = %44

"NAME" OF ENTRY BLOCK

SECTOR %000044

S ? S

000:	051531	051440	020040	020040	000007	000006	177003	001773
010:	000000	000000	020040	020040	020040	020040	000000	031240
020:	077777	177777	000000	002114	077777	177777	000000	001465
030:	077777	177777	002531	000226	000000	053101	047120	020040
040:	020040	$\boxed{000030}$	$\boxed{000027}$	071003	000700	000000	000000	020040
050:	020040	020040	020040	000000	000400	077777	177777	000000
060:	000005	077777	177777	000000	000014	077777	177777	$\boxed{002525}$
070:	000226	000000	000000	000000	000000	000000	[illegible]	000000
100:	000000	[illegible]	000000	000000	000000	000000	000000	000000
110:	000000	000000	000000	000000	000000	000000	000000	000000
120:	000000	000000	000000	000000	000000	000000	000000	000000
130:	000000	000000	000000	000000	000000	000000	000000	000000
140:	000000	000000	000000	000000	000000	000000	000000	000000
150:	000000	000000	000000	000000	000000	000000	000000	000000
160:	000000	000000	000000	000000	000000	000000	000000	000000
170:	000000	000000	000000	000000	000000	000000	000000	000000

ASECW

ACCOUT ENTRY BLOCK

FOR ACCOUNT VANP

AGIPNTR = ______

AUIPNTR = ______

SECTOR %000064

000:	110601	000001	000000	000001	013102	000003	053101	047120
010:	020040	020040	050125	041040	020040	020040	$\boxed{000033}$	000001
020:	000001	000001	000001	000001	000001	000001	000001	000001
030:	000001	000001	000001	000001	000001	000001	000001	000001
040:	000001	000001	000001	000001	000001	000001	000001	000001
050:	000001	000001	000001	000001	000001	000001	000001	000001
060:	000001	000001	000001	000001	000001	000001	000001	000001
070:	000001	000001	000001	000001	000001	000001	000001	000001
100:	000001	000001	000001	000001	000001	000001	000001	000001
110:	000001	000001	000001	000001	000001	000001	000001	000001
120:	000001	000001	000001	000001	000001	000001	000001	000001
130:	000001	000001	000001	000001	000001	000001	000001	000001
140:	000001	000001	000001	000001	000001	000001	000001	000001
150:	000001	000001	000001	000001	000001	000001	000001	000001
160:	000001	000001	000001	000001	000001	000001	000001	000001
170:	000001	000001	000001	000001	000001	000001	000001	000001

GROUP INDEX BLOCK

```
SECTOR %000067

000:  050125  041040  020040  020040  000031  020040  020040  020040
010:  020040  000000  000000  077777  177777  000000  000000  077777
020:  177777  000000  000000  077777  177777  020143  015006  000700
030:  000000  000000  000000  000000  000000  000000  000000  000000
040:  000000  000000  000000  000000  000000  000000  000000  000000
050:  000000  000000  000000  000000  000000  000000  000000  000000
060:  000000  000000  000000  000000  000000  000000  000000  000000
070:  000000  000000  000000  000000  000000  000000  000000  000000
100:  000000  000000  000000  000000  000000  000000  000000  000000
110:  000000  000000  000000  000000  000000  000000  000000  000000
120:  000000  000000  000000  000000  000000  000000  000000  000000
130:  000000  000000  000000  000000  000000  000000  000000  000000
140:  000000  000000  000000  000000  000000  000000  000000  000000
150:  000000  000000  000000  000000  000000  000000  000000  000000
160:  000000  000000  000000  000000  000000  000000  000000  000000
170:  000000  000000  000000  000000  000000  000000  000000  000000
```

GROUP ENTRY BLOCK

```
SECTOR %000065

000:  100602  000001  000000  000002  000602  000030  050125  041040
010:  020040  020040  043111  046105  040440  020040  000037  000002
020:  000002  000002  000002  000002  000002  000002  000002  000002
030:  000002  000002  000002  000002  000002  000002  000002  000002
040:  000002  000002  000002  000002  000002  000002  000002  000002
050:  000002  000002  000002  000002  000002  000002  000002  000002
060:  000002  000002  000002  000002  000002  000002  000002  000002
070:  000002  000002  000002  000002  000002  000002  000002  000002
100:  000002  000002  000002  000002  000002  000002  000002  000002
110:  000002  000002  000002  000002  000002  000002  000002  000002
120:  000002  000002  000002  000002  000002  000002  000002  000002
130:  000002  000002  000002  000002  000002  000002  000002  000002
140:  000002  000002  000002  000002  000002  000002  000002  000002
150:  000002  000002  000002  000002  000002  000002  000002  000002
160:  000002  000002  000002  000002  000002  000002  000002  000002
170:  000002  000002  000002  000002  000002  000002  000002  000002
```

FILE INDEX BLOCK

SECTOR %000067

	B A	S E	G R	T				
000:	041101	051505	043522	050040	(000041	020040	020040	020040
010:	020040	(000000	000000)	077777	177777	000000	000000	077777
020:	177777	000000	000000	077777	177777	002041	004102	000600
030:	000000	050125	041040	020040	020040	000031	020040	020040
040:	020040	020040	000000	000400	077777	177777	000000	000005
050:	077777	177777	000000	000014	077777	177777	020143	015006
060:	000700	000000	000000	000000	000000	000000	000000	000000
070:	000000	000000	000000	000000	000000	000000	000000	000000
100:	000000	000000	000000	000000	000000	000000	000000	000000
110:	000000	000000	000000	000000	000000	000000	000000	000000
120:	000000	000000	000000	000000	000000	000000	000000	000000
130:	000000	000000	000000	000000	000000	000000	000000	000000
140:	000000	000000	000000	000000	000000	000000	000000	000000
150:	000000	000000	000000	000000	000000	000000	000000	000000
160:	000000	000000	000000	000000	000000	000000	000000	000000
170:	000000	000000	000000	000000	000000	000000	000000	000000

GROUP ENTRY BLOCK

(AFTER A SECOND USER WAS BEEN CREATED) UNDER ACCOUNT VANP)

SECTOR %000063

USER INDEX

000:	130601	000001	000000	000001	032302	000003	053101	047120
010:	020040	020040	042117	047040	020040	020040	000035	000001
020:	000001	000001	000001	000001	000001	000001	000001	000001
030:	000001	000001	000001	000001	000001	000001	000001	000001
040:	000001	000001	000001	000001	000001	000001	000001	000001
050:	000001	000001	000001	000001	000001	000001	000001	000001
060:	000001	000001	000001	000001	000001	000001	000001	000001
070:	000001	000001	000001	000001	000001	000001	000001	000001
100:	000001	000001	000001	000001	000001	000001	000001	000001
110:	000001	000001	000001	000001	000001	000001	000001	000001
120:	000001	000001	000001	000001	000001	000001	000001	000001
130:	000001	000001	000001	000001	000001	000001	000001	000001
140:	000001	000001	000001	000001	000001	000001	000001	000001
150:	000001	000001	000001	000001	000001	000001	000001	000001
160:	000001	000001	000001	000001	000001	000001	000001	000001
170:	000001	000001	000001	000001	000001	000001	000001	000001

SECTOR %000071

USER ENTRY

000:	042117	047040	020040	020040	071003	000700	000000	000000
010:	020040	020040	020040	020040	050125	041040	020040	020040
020:	000000	000226	000000	000000	000000	000000	000000	000000
030:	000000	000000	000000	000000	000000	000000	000000	000000
040:	000000	000000	000000	000000	000000	000000	000000	000000
050:	000000	000000	000000	000000	000000	000000	000000	000000
060:	000000	000000	000000	000000	000000	000000	000000	000000
070:	000000	000000	000000	000000	000000	000000	000000	000000
100:	000000	000000	000000	000000	000000	000000	000000	000000
110:	000000	000000	000000	000000	000000	000000	000000	000000
120:	000000	000000	000000	000000	000000	000000	000000	000000
130:	000000	000000	000000	000000	000000	000000	000000	000000
140:	000000	000000	000000	000000	000000	000000	000000	000000
150:	000000	000000	000000	000000	000000	000000	000000	000000
160:	000000	000000	000000	000000	000000	000000	000000	000000
170:	000000	000000	000000	000000	000000	000000	000000	000000

IMS

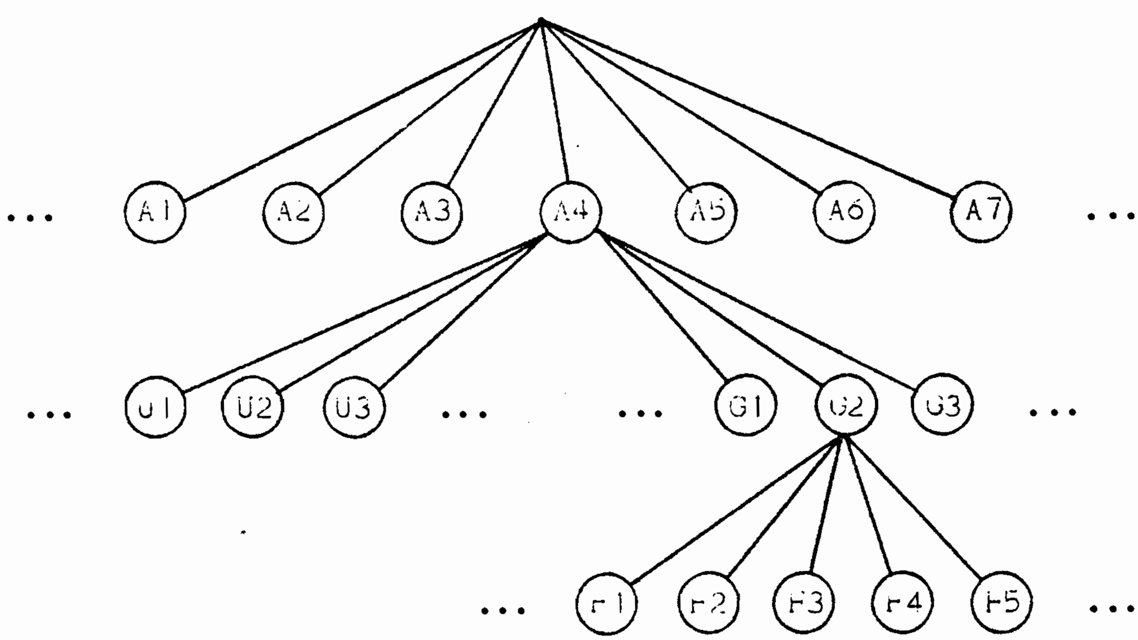
2. DIRECTORY

2.1 INTRODUCTION

2.1.1 Requirements

The logical requirements for a file directory and organizational directory were evolved concurrently. Accounts were to be the major partition; belonging to each account would be a set of users and a set of groups; and every file belonged to a group. The user possessed no files directly, but rather he accessed files of groups. The connection of logon groups and users served to give the user a local file domain, and also provided an entity to which resources were accumulated and limited. The final structure, and some rationale for it are described in the ERS.

The directory can be thought of as consisting of two trees, one overlayed upon the other. One tree is the account/user tree; the other is the account/group/file tree. This causes the maintenance of the MPE directory to be somewhat non-standard from usual tree manipulations (e.g. tree scanning). In some maintenance routines it is necessary to distinguish which subtree under an account is the desired target.



GROUP ENTRY (25 WORDS)

0	GNAME	0	GROUP NAME
1		1	
2		2	
3		3	
4	GFIPNTR	4	GROUP FILE INDEX POINTER
5	GPASS	5	
6		6	PASSWORD
7		7	
10		8	
11	GDFSCOUNT	9	DISC FILE SPACE COUNT (SECTORS)
12		10	
13	GDFSLIMIT	11	DISC FILE SPACE LIMIT (SECTORS)
14		12	
15	GCPUCOUNT	13	CPU TIME COUNT (SECONDS)
16		14	
17	GCPULIMIT	15	CPU TIME LIMIT (SECONDS)
20		16	
21	GCONTIMECOUNT	17	CONNECT TIME COUNT (MINUTES)
22		18	
23	GCONTIMELIMIT	19	CONNECT TIME LIMIT (MINUTES)
PURGE FLAG 24		20	
25	GSEC	21	GROUP SECURITY (SEE BELOW)
26		22	
27	GCAPABILITY	23	GROUP CAPABILITY
30		24	

25	P		R ANY	R AC	R AL	R GU	R GL	A ANY	A AC	A AL	A GU	A GL	W ANY	W AC	W AL	W GU	21
26	W GL	L ANY	L AC	L AL	L GU	L GL	X ANY	X AC	X AL	X GU	X GL	S ANY	S AC	S AL	S GU	S GL	22

Figure 2.1-B

ACCOUNT ENTRY (29 WORDS)

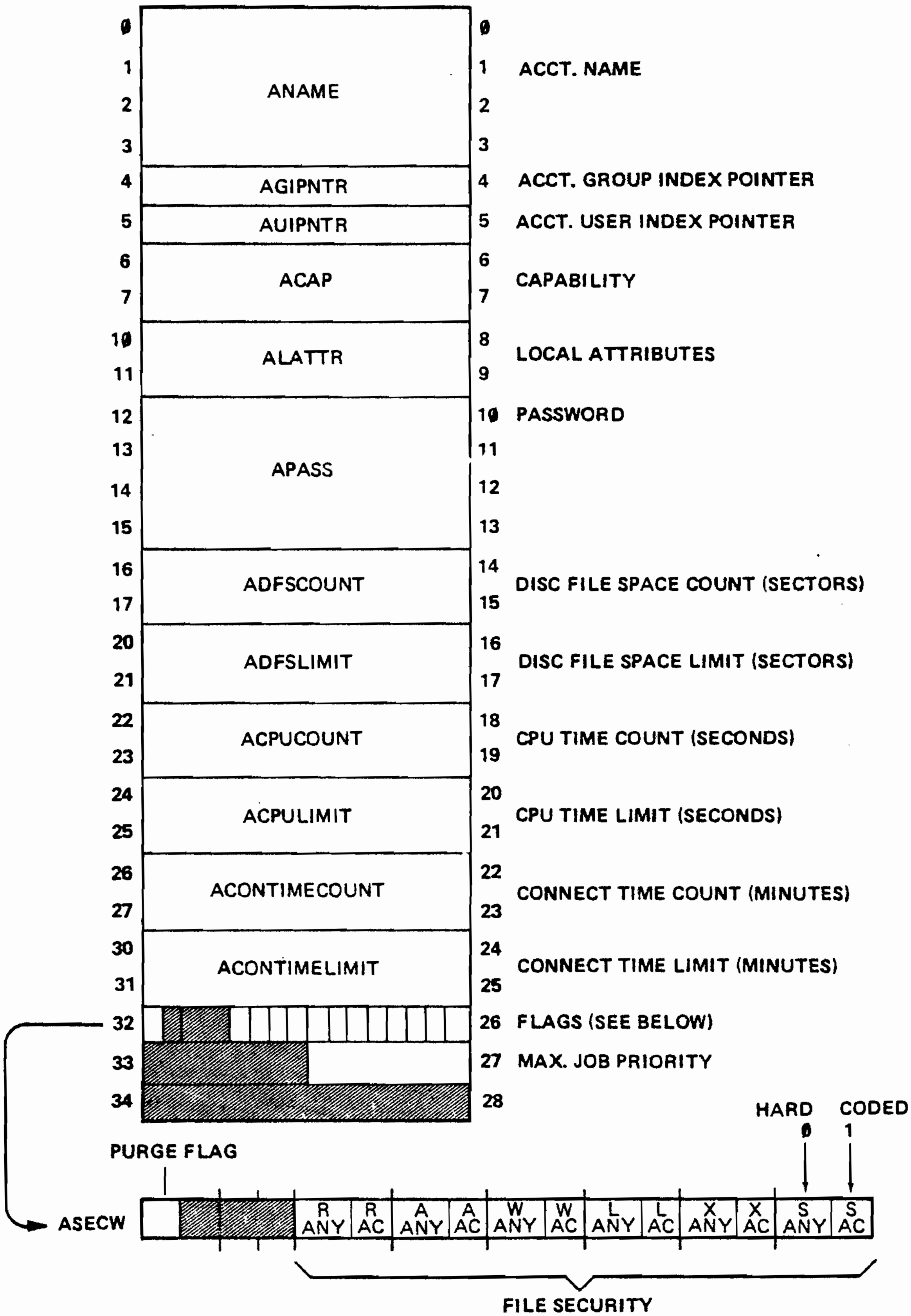


Figure 2.1-A

USER ENTRY (19 WORDS)

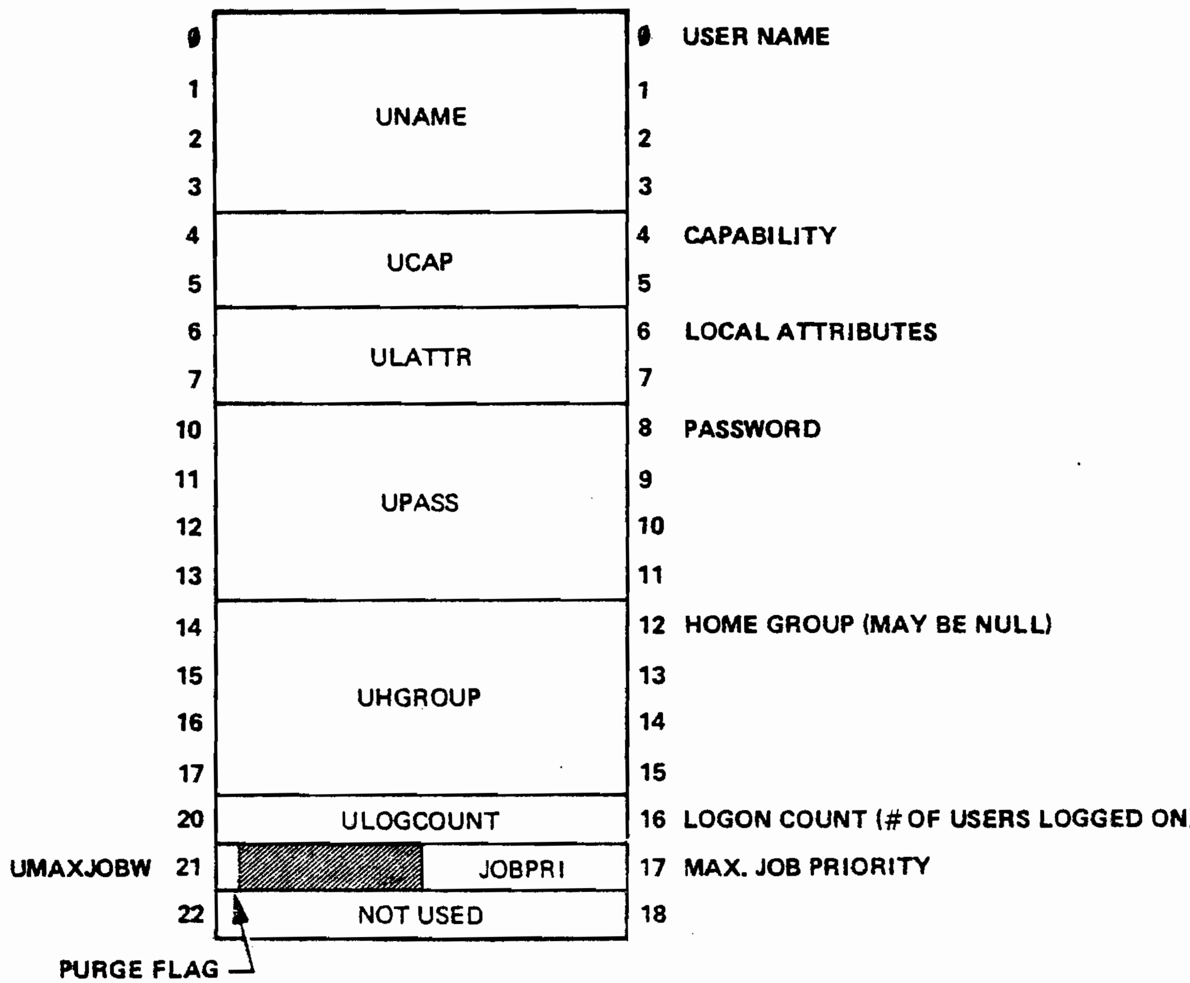
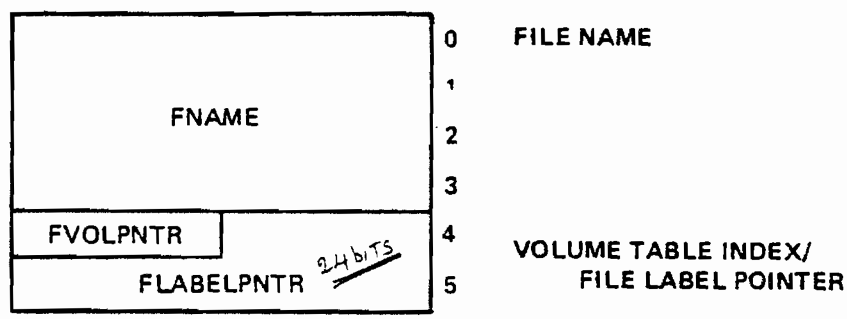


Figure 2.1-D

FILE ENTRY (FILE POINTER) (6 WORDS)



FLABELPNTR is sector address (on FVOLPNTR) of the label

Figure 2.1-C

2.1.2 Directory Information

The directory serves essentially two independent purposes:

1. File partitioning and security; and
2. MPE access, capability restrictions, and resource accumulation.

Consequently, each directory node has information pertaining to both of these requirements. This information is diagrammed in Figures 2.1-A,B,C,D. Some notes on the contents of the directory entries follows:

NAMES- All names in the directory (as in MPE, in general), are represented internally by four contiguous words containing the name in ASCII, left-justified and padded with blanks.

INDEX POINTERS- These are one-word positive integers that are directory-base relative addresses of subtrees. Thus account entries have two INDEX POINTERS, one for the group subtree and one for the user subtree. Logically, these pointers can be thought of as representing subtrees and the actual interpretation of them will be deferred.

PASSWORDS- These are used for establishing MPE logon access only. The user and account passwords are always required, if non-null; and the group password is required when accessing a non-home group. This is fully described in the MPE ERS. A null password is represented by all blanks.

CAPABILITIES and LOCAL ATTRIBUTES- These have meanings as described in the ERS. The bits of account and user CAPABILITIES correspond one-for-one to the bits returned by the WHO intrinsic when capability is returned; see Figure 2.1-E. Groups' CAPABILITIES are only the so-called "capability class attributes", which is the second word of the full two-word capability. The following bits are either vestigial (due to abandoned capabilities) or reserved for a future version of MPE.

word 0: bits 6, 13.
word 1: bits 6, 10, 11, 13.

The contents of these bits should be ignored.

COUNTS and LIMITS- These are double-word, positive integers, each representing the indicated usage count or limit. The maintenance of these variables is described later. Where an "unlimited" quantity is required it is represented by %17777777777, which is essentially unlimited in most applications, permitting no special test to be necessary for the unlimited case.

FILE SECURITIES- These words contain bit representations of file security "permission matrices". The bit correspondence is diagrammed. File security is described in Subsection 4.5.

MAXIMUM PRIORITY- This is the numerical priority value which restricts the priority of any process of any job running under the account/group.

HOME GROUP- This is used as described in the ERS.

LOGON COUNT- This is the number of people currently logged-on under the user (and account). Its primary use is for determining whether a user entry can be removed.

PURGEFLAGS- These flags are set whenever a :PURGE... has been attempted on the entry but was unable to complete successfully due to one of the conditions described in the ERS. Currently, it is never examined.

FILE POINTER- In the directory this double-word consists of a VTAB index which effectively defines the disc device which contains the file (left byte, 1st word) and the disc relative-sector address of the file label.

2.1.3 Design Constraints

There were several problem areas that had to be resolved that had a very influential effect on the final design of the MPE directory. Some of these were problems which had to be addressed in every file directory organization; others were due to MPE requirements- both external design requirements and internal implementation restrictions. Following is a description of some of the more notable problems that had to be addressed.

The first consideration was that the directory- due to its potential size- had to be a disc-based data structure. The (software-managed) data segmentation facilities could be used only for directory buffering requirements, and an independent facility had to be implemented solely for directory space management. The fact that the directory had to reside on disc also confined the alternatives for the data structure. For example, any orgnaization which required chains spanning sectors was rejected because each access along such a chain could involve a separate disc access.

Another interesting consideration was the necessity to easily generate alphabetical listings. Some directory-type data structures (such as hashed tables) do not contain the information in any semblance of an alphabetical order. This would necessitate either the maintenance of a separate access structure incorporated into the basic structure (such as an alphabetical chain in a hashing scheme); or an "offline" sorting phase when listings were requested. Often, the work required to maintain a separate alphabetical access provision was in fact the same work that another scheme would require in its entirety.

Of course, the logical requirements were the most important design criterion to respect, of which the hierarchial structure was the

USER ATTRIBUTES / CAPABILITY

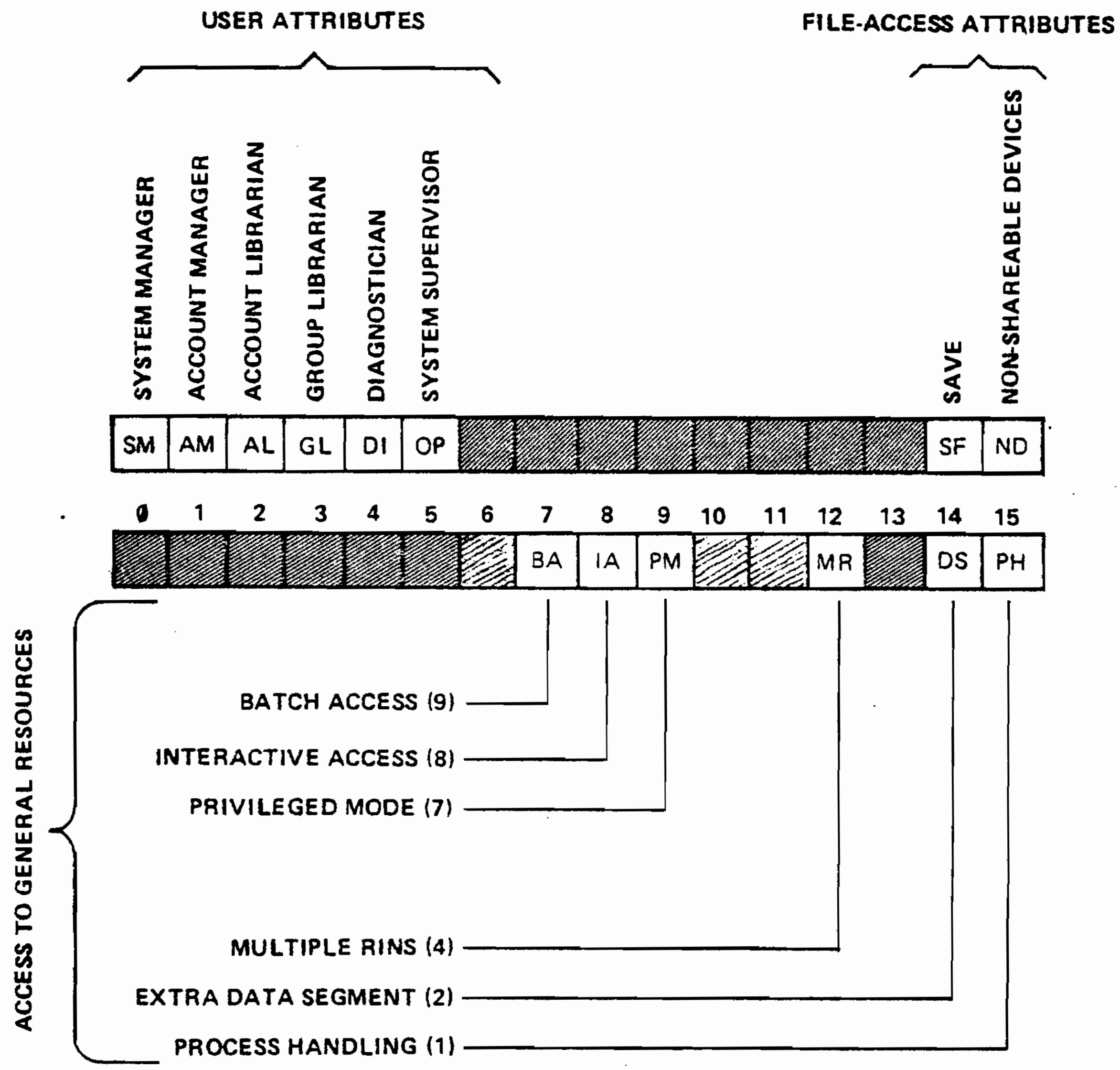


Figure 2.1-E

most significant factor to be considered. Given the hierarchial organization, it was desirable, of course, to be able for the user to make relatively fast accesses to his local (logon group) file domain, and to his account file domain. Essentially it was determined that a means for "pointing" directly to a group/file level, and to an account/group level was desirable.

The means for achieving this tied in closely to another typical file directory problem: In performing a listing operation it is desirable to keep a "pointer" to the entry being listed, so that the next entry can be easily retrieved. The essential problem of keeping such a pointer is that - while depending on the pointer - some directory operation can occur asynchronously that might shift some directory entries (e.g. purge, insert), thus making the "pointer" invalid unless some provision is made explicitly for this case.

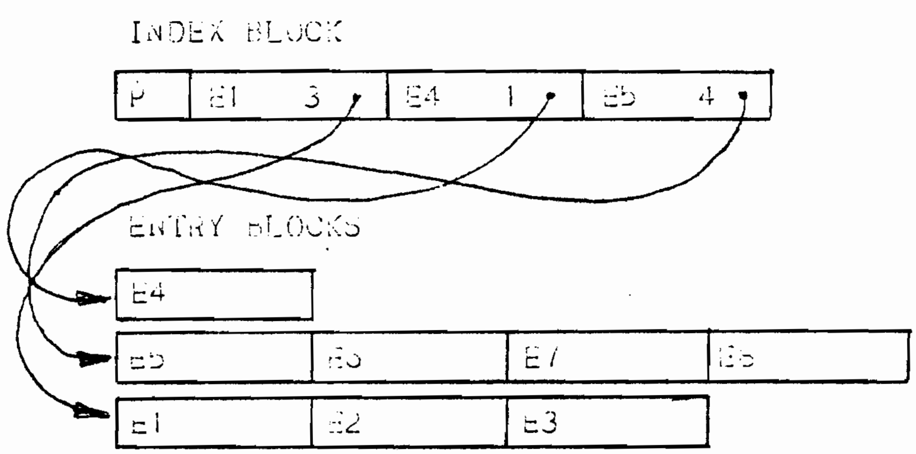
Additionally, there is the requirement that the directory structure on the disc maintain its integrity across abrupt system failures (crashes, non =SHUTDOWN shutdowns, etc.). For this reason, directory vulnerablity is minimized, with return from directory routines signifying a completely recoverable directory structure on disc.

The preceding discussion is intended to give the reader a background of the design requirements, and thus a better understanding of the ultimate structure employed.

2.2 OVERVIEW

2.2.1 Data Structure

The MPE directory is a hierarchial data structure in the form of two trees, each making use of a common first level (account). (See Figure 2.2-A.) Descending down the structure, it can be described as consisting of a system root and its subtrees of account nodes; each account node consists of the account entry (information) and subtrees of user nodes and subtrees of group nodes; the user nodes are "leafs"; the group nodes consist of the group entry and subtrees of file entries; each file entry is a pointer to the file label and is a leaf as far as the directory is concerned. (In the listings, the term "level" is sometimes used in place of the more correct term "subtree"; it is used as meaning a particular subtree at a level. Level (when used according to its proper definition) is always considered to be relative to the system root.) Each subtree is implemented by an indexed sequential structure in which every block of nodes (entries) is pointed to by an index, the set of indices for a subtree being contained in one block. Pictorially, the structure for each subtree looks like this:



The actual information that the directory must keep is contained in entries. Entries are distributed among a set of entry blocks for each subtree. The entries are maintained alphabetically within each block. Every block is pointed to by an index, which contains the name of the first entry of "its" block and a pointer to the block. The indices are kept alphabetically in an index block, of which there is one for each subtree. The entry blocks are thus logically arranged alphabetically (according to the indices) even though the actual blocks are in fact not contiguous. [Every index block is preceded by a prefix containing necessary access information; it is described below.] Note that a logical subtree is represented by an index block.

As might be inferred, space is allocated/deallocated in blocks, each consisting of an integral number of contiguous sectors. Blocks that are logically contiguous, however, need not be physically contiguous. The total space available for the directory consists of one contiguous area on the system disc in which all space is managed by the directory routines by means of a simple space bitmap. This bitmap is the first three sectors of the directory disc area.

A detailed diagram of the entire organization can be seen in Figure 2.2-B,C.

Note that any entry or the directory can be retrieved by either performing a global search starting from the root node, or by performing a shorter search on a subtree, assuming there exists a means of "pointing" to the desired subtree. Because every subtree is equivalent to an index block, pointers to subtrees are pointers to index blocks.

The provisions necessary for listing and quick access- i.e. maintenance of pointers- has been alluded to above. The pointers used for the MPE directory consist of pointers to the appropriate index block. Thus fast access for local, group files is achieved by keeping a pointer to the user's logon group/file index block; pointers for the purposes of listing are pointers to the relevant index block and the name being listed. The only problem in using "pointers" is that it must be guaranteed that the item which is being pointed to cannot be moved (including deleted). Therefore, the following mechanism is employed: Only pointers to index blocks can be considered safe; this is achieved by means of incrementing an "index pointer count" associated with each index block whenever that block is being pointed to. As long as the count is non-zero, the index block cannot be moved, or deleted. Note that pointers to actual indices, or to entries, or to entry blocks cannot be considered secure because (without a similar facility) indices, entries, and entry blocks can be moved or deleted when the directory is not locked. Although an index block may be "frozen", its indices are not and may be moved and deleted.

2.2.2 Definitions

DIRBASE is the sector address on the system disc that is the beginning of the disc directory area.

A POINTER is a one-word, positive, DIRBASE-relative sector displacement. DIRBASE +POINTER yields sector address.

LOGICAL ORGANIZATION OF DIRECTORY

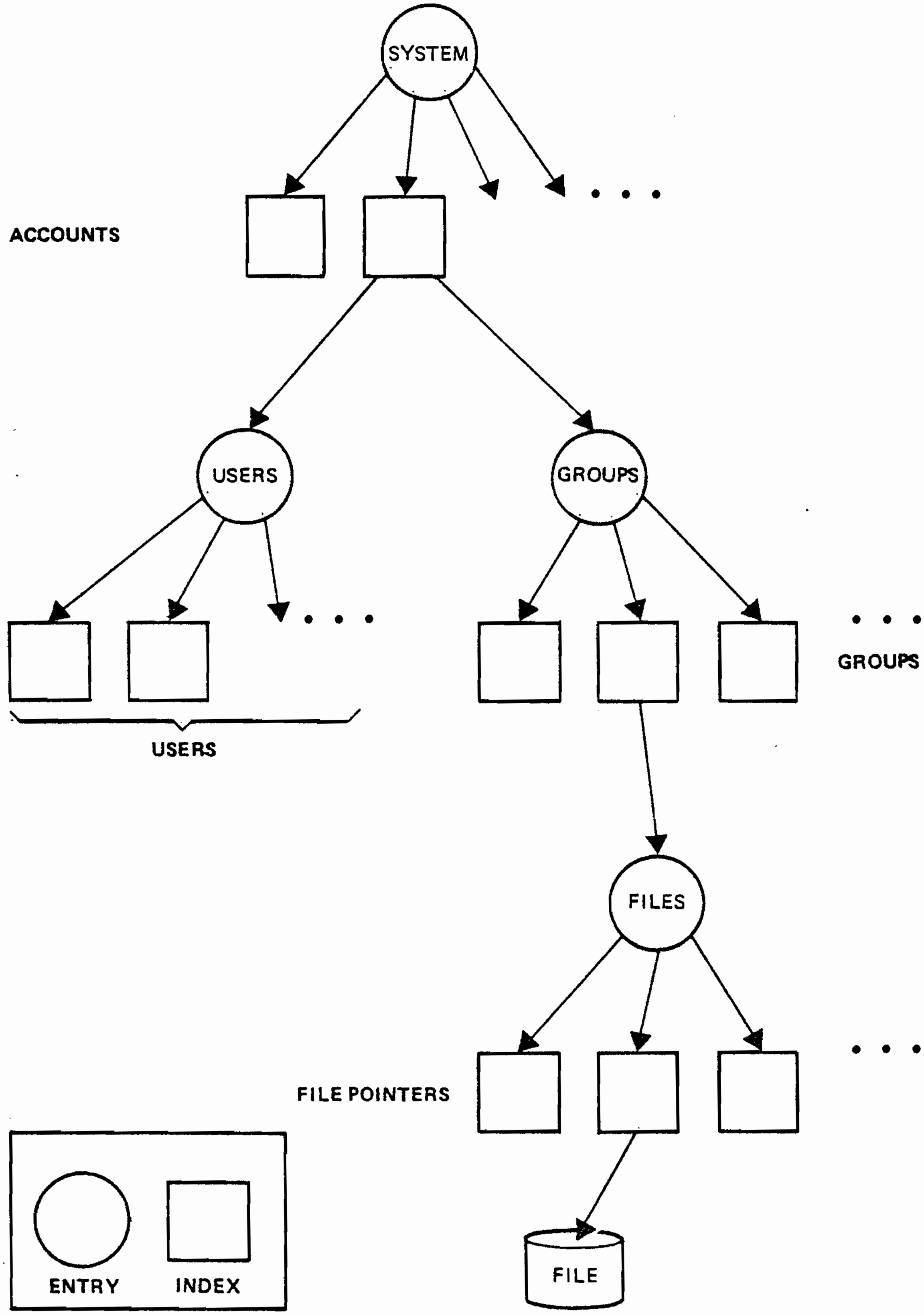


Figure 2.2-A

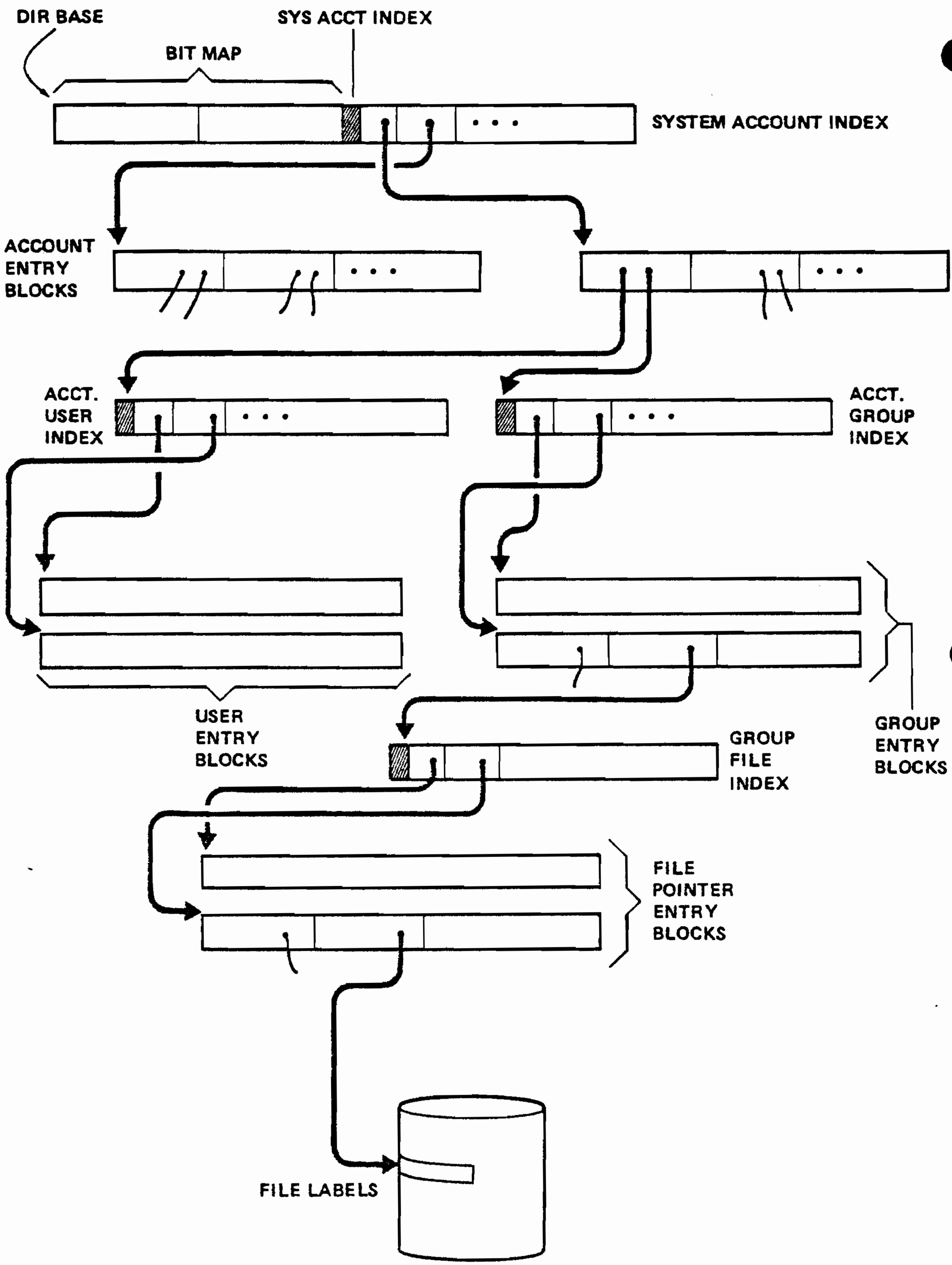

DIR BASE
SYS ACCT INDEX
BIT MAP
SYSTEM ACCOUNT INDEX
ACCOUNT
ENTRY
BLOCKS
ACCT.
USER
INDEX
ACCT.
GROUP
INDEX
USER
ENTRY
BLOCKS
GROUP
FILE
INDEX
GROUP
ENTRY
BLOCKS
FILE
POINTER
ENTRY
BLOCKS
FILE LABELS

Figure 2.2-C

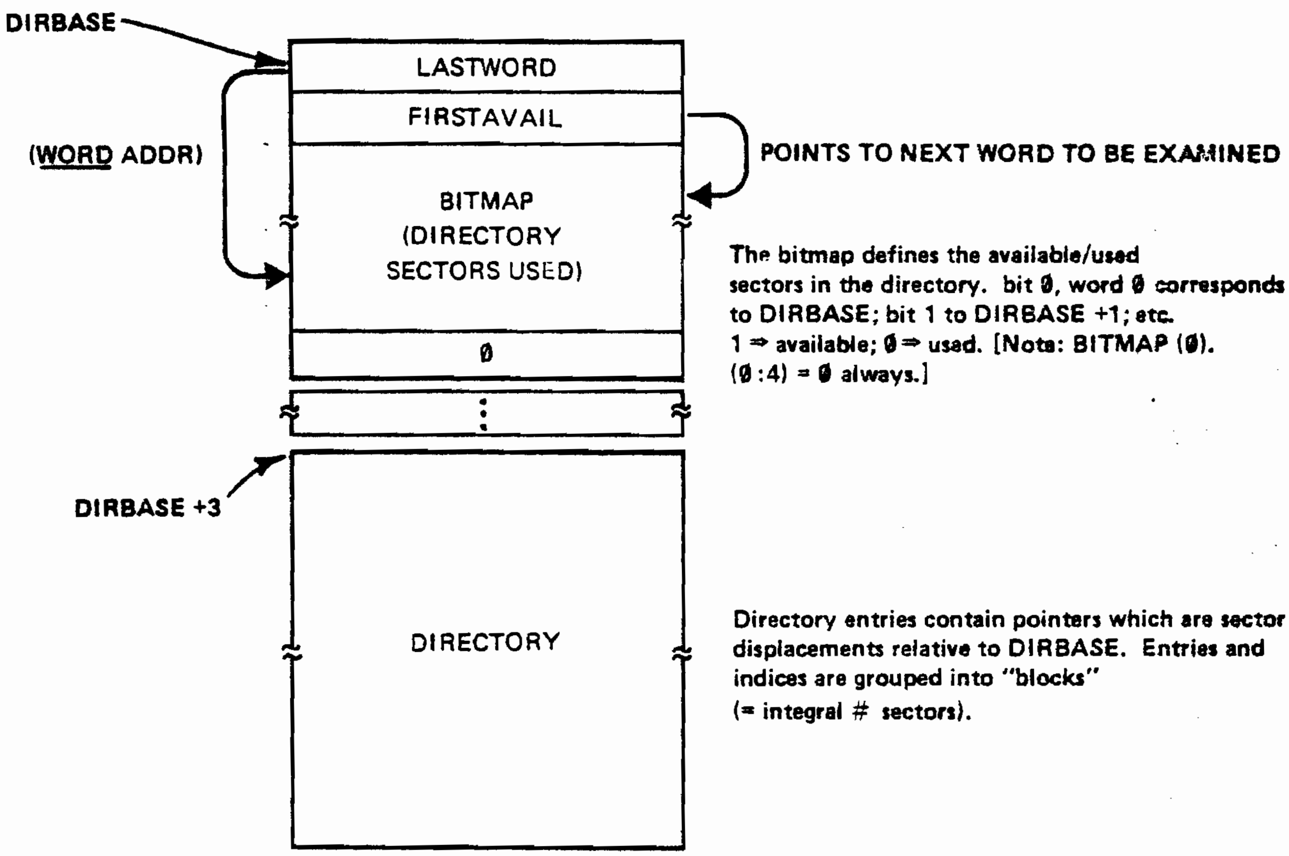
DIRECTORY

DIRECTORY ON DISC CONSISTS OF A CONTIGUOUS AREA

SYSGLOB cells:

DIRIOADDR ←——— [DISKLDEV: SYSGLOB + 62_{10}]

DIRBASE ←——— absolute disk addr of base [SYSGLOB + 44_{10}]



The bitmap defines the available/used sectors in the directory. bit Ø, word Ø corresponds to DIRBASE; bit 1 to DIRBASE +1; etc. 1 ⇒ available; Ø ⇒ used. [Note: BITMAP (Ø). (Ø:4) = Ø always.]

Directory entries contain pointers which are sector displacements relative to DIRBASE. Entries and indices are grouped into "blocks" (= integral # sectors).

The capacities for accounts/groups/users/files are dependent on their block sizes, described in the directory data segment.

* SYSSAIBSIZE	System acct index block size (sectors)
SYSAUIBSIZE	Acct. user index block size (sectors)
SYSAGIBSIZE	Acct. group index block size (sectors)
SYSGFIBSIZE	Group file index block size (sectors)
* SYSAEBSIZE	Acct. entry block size (sectors)
SYSUEBSIZE	User entry block size (sectors)
SYSGEBSIZE	Group entry block size (sectors)
SYSFEBSIZE	File entry block size (sectors)
SYSMAXBSIZE	Maximum of above. (used to initialize DDS.)

*These values are used once for the creation of the (root) system, account index or new systems. This root index is always at address DIRBASE +3.

Figure 2.2-B

address are arranged to coincide with these initial three sectors. Directory space management is entirely accomplished in this data segment.

The format of this area is shown in Figure 2.2-D. A bit set to 1 implies that the corresponding page is available. The size of the directory disc area is implicitly defined by the number of sectors available as indicated by the bits on between (and including) the third word and LASTWORD. FIRSTAVAIL is a cycling pointer to the first word that is to be interrogated for availability: space allocation takes place cyclically to minimize fragmentation.

2.2.3.2 DIRECTORY DATA SEGMENT

The directory data segment (DDS) is shown in detail in Figure 2.2-E. This data segment is used as the i/o buffer for the directory, and all directory management is effected in it. It can be divided into the following areas:

A PAGE is the smallest quantum of allocatable space, i.e. a sector. PAGE when used in a term such as "PAGE number" refers to the DIRBASE-relative sector displacement of a location.

A BLOCK is a contiguous group of PAGES; manipulated as one linear area - and is the basic unit of I/O transfer. ENTRIES and INDICES are grouped into BLOCKS. Directory blocks can be of different sizes.

SYSACCTINDEX is the fixed POINTER of the directory root; i.e. a POINTER to the system account INDEX BLOCK.

ENTRIES are the actual information-containing constituents of the directory. The essential purpose of the directory is to mantain ENTRIES; all other directory components exist in order to make this maintenance possible. There are four kinds of ENTRIES, each of a different size: ACCOUNT ENTRIES, GROUP ENTRIES, USER ENTRIES, FILE ENTRIES. An ENTRY may contain a POINTER to subtrees. The MPE directory FILE ENTRY should not be confused with file labels: directory FILE ENTRIES contain disc addresses of file labels. ENTRIES are grouped into ENTRY BLOCKS.

An INDEX is a 6-word item associated with every ENTRY BLOCK. It contains the name of the first ENTRY in the BLOCK, a count of the number of ENTRIES in the BLOCK and a POINTER to the ENTRY BLOCK. INDICES are grouped into INDEX BLOCKS.

ELEMENT is the generic term for an item which is either an INDEX or an ENTRY.

The four types of subtrees of the MPE directory are (INDEX BLOCKS) known as SYSTEM ACCOUNT INDEX (directory root), ACCOUNT USER INDEX, ACCOUNT GROUP INDEX, and GROUP FILE INDEX. These terms refer to a specific INDEX BLOCK unless used in a descriptive context. Every ACOUNT ENTRY contains a POINTER to its ACCOUNT USER INDEX BLOCK and a POINTER to its ACCOUNT GROUP INDEX BLOCK, etc.

The INDEX BLOCK PREFIX (or simply PREFIX) is the first part of every INDEX BLOCK and contains information necessary to access the INDICES, and the ENTRIES of the ENTRY BLOCKS to which the INDICES point.

2.2.3 Basic Algorithms

2.2.3.1 SPACE MANAGEMENT

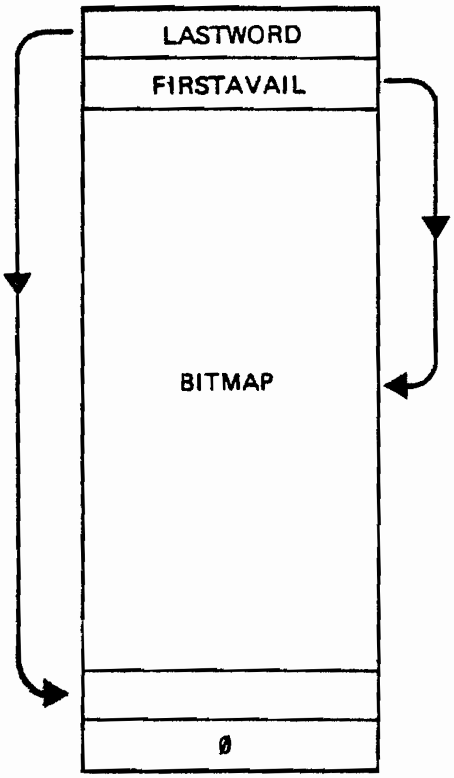
The first three sectors of the directory disc area contain a space bitmap for the entire area. [The system account index always exists and is the first block created for a null directory. Therefore SYSACCTINDEX is always 3.] The DIRECTORY SPACE DATA SEGMENT is a data segment whose length and disc swapping area

DIRECTORY SPACE DATA SEGMENT (DSDS)

8-1-73

$DST = 21_{10}$

$SIR = 8_{10}$

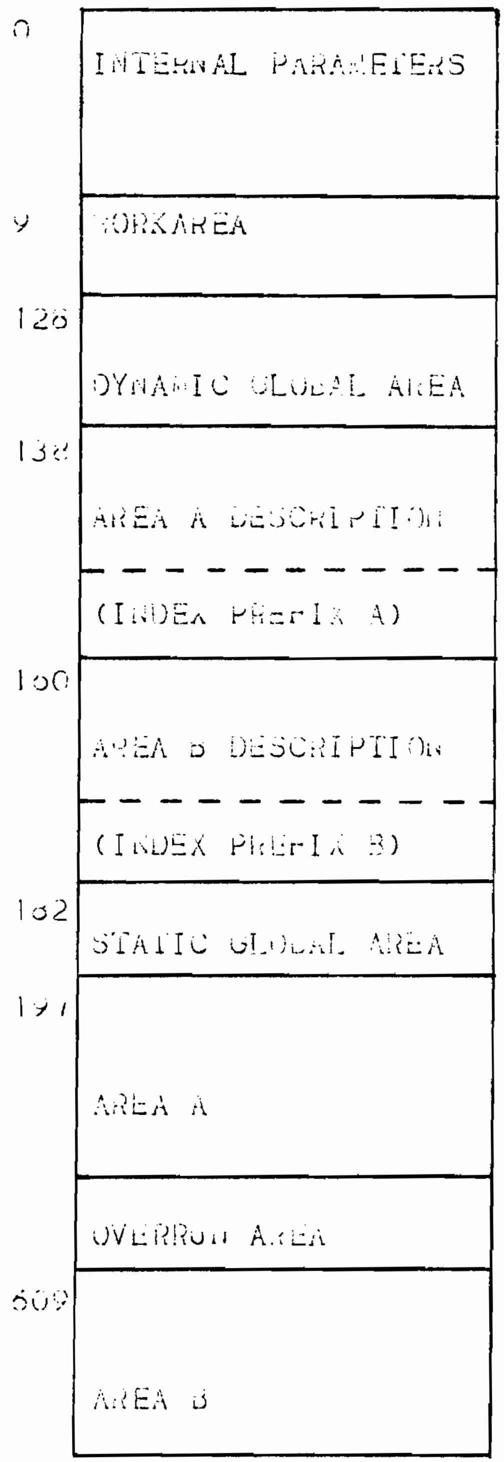


DEFINES LAST WORD OF BIT MAP

DEFINES NEXT WORD TO BE EXAMINED

THE BITMAP DEFINES THE AVAILABLE/USED SECTORS IN THE DIRECTORY. BIT Ø WORD Ø CORRESPONDS TO DIRBASE; BIT 1 WORD Ø TO DIRBASE + 1 ETC. 1 ⇒ AVAILABLE Ø ⇒ USED. [NOTE: BITMAP (Ø) · (Ø : 4) = Ø ALWAYS.]

Figure 2.2-D



AREA A and AREA B are used as the directory i/o buffers and are large enough to accomodate the largest directory block. Two areas are required primarily because of the entry insertion algorithm which splits a block into two blocks at times. This algorithm is described below. AREA A is used solely for entry blocks and AREA B is used primarily for index blocks (in addition to entry block splits). The OVERRUN AREA is as large as the largest entry and also is provided for entry insertion.

AREA A DESCRIPTION and AREA B DESCRIPTION are (directly-addressable) variables that contain various attributes and characteristics of their respective areas. (E.g. number of elements; dirty flag; page number; etc.) For index blocks, this information is obtained from the index block prefix; for entry blocks this information is obtained from the entry block's index and the index' block prefix. These areas are set whenever a block is read into the DDS, and always reflect the correct attributes. The last part of these DESCRIPTION AREAS have space for index block prefixes (when the block they contain is an index block; this area is consequently never used for the AREA A DESCRIPTION because AREA A currently never contains an index block.).

The STATIC GLOBAL AREA contains configuration parameters and is never changed after cold load.

The DYNAMIC GLOBAL AREA contains information pertaining to the last call of a directory uncallable intrinsic. As will be seen later, there is considerable commonality of parameters and necessary setup among these uncallable intrinsics. This area contains some of this information that is needed by some of the internal directory procedures.

The first 128 words of the DDS are used for two purposes. The beginning area is used for submission and return of entry information to/from internal directory routines. The entire area is also available as a work area for the "recipient" procedures of DIRECSCAN (see below).

2.2.3.3 PRIMITIVES FOR BASIC MANIPULATIONS

In light of the data structure described above, the algorithms to manipulate the directory follow directly. This section considers the cases of finding, purging and inserting entries into the first level of a subtree. The descriptions are overviews of the operations of internal directory primitives. The next sections describe extensions necessary for the hierarchial structure and special file system considerations.

Given a pointer to a subtree and a first level name to find, the first step is to read the designated index block (into area B) and find the index whose name is the greatest among those which are equal to or less than the target name. This index points to the entry block that must contain the desired entry; this block is read (into area A) and the entry is searched for.

DIRECTORY DATA SEGMENT

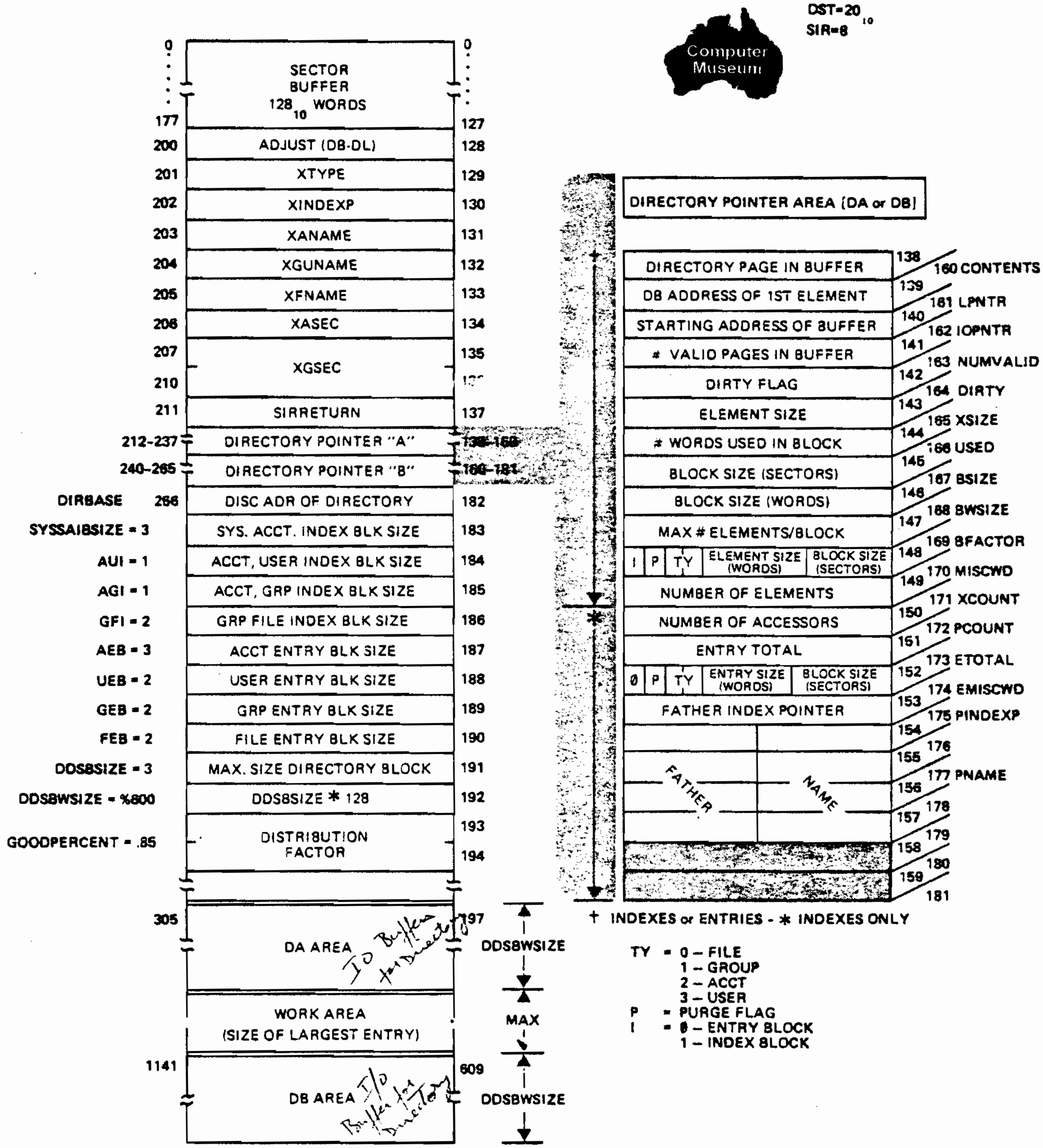


Figure 2.2-E

or logon account files; i.e. when a fully-qualified reference is not supplied.] For the implementation of this concept, a common initializing routine is used [DIRSTARTOFF]. It analyzes the caller's specification, performs any necessary directory accesses and returns (to the directory uncallable intrinsic) the final entry name and associated index block pointer for the target. Once the target is so distinguished, the pertinent directory uncallable intrinsic then completes the operation by performing the necessary juggling to effect the requested insert, purge or find. The following overviews describe the operations implemented after the target has been "marked" as just explained.

The find operation simply returns the desired node, using the appropriate primitive.

Insertion uses the insert primitive to insert the entry at the designated level. But depending on the level, additional provisions for subtrees must be made. Specifically, a group file index must be created when inserting groups; an account user index and an account group index must be created when inserting accounts. Simple entry insertions are made for files and users.

Purging is complicated by the requirement that when an entry is purged, its subtrees (if any) must also be purged. When purging user or file entries, only the entry is deleted and no further actions need be taken. However, when purging accounts or groups, their respective subtrees must first be purged. To do this, the designated tree is scanned in endorder [endorder is the traversal of a tree by visiting all subtrees before visiting the root, where a visit of a tree means traversal of it in endorder]. The target root (i.e. the account or group entry to be removed) is deleted only if all its subtrees have been successfully removed.

2.2.3.5 ACCOUNTING AND FILE SECURITY CONSIDERATIONS

The following considerations have been made for permanent file space accounting and file security (both functionally defined in the ERS). These special actions are invoked through special directory uncallable intrinsics [DIRECFINDFILE, DIRECPURGEFILE and DIRECINSERTFILE], that are available to MPE callers manipulating file entries and requiring these functions. These routines are to be distinguished from the basic complement of directory uncallable intrinsics described above, which operate on directory entries without regard to accounting and security. [There is one exception to the last comment: purging a group adjusts the account's file space accordingly.] Note that these special considerations are made only when dealing with files.

The special provisions for finding a file consist of returning additional information regarding file security at the account and group level. Otherwise it is identical to a general directory find of a file entry.

In order to purge a first level entry from a designated subtree the first step is to find the entry, as described above. Once it has been located, the entry is removed from the entry block by moving the succeeding entries of the entry block so that the space occupied by the target entry is overlayed. Several variables (e.g. counts) are updated in the index block (and the DDS description areas). If the entry was the only element in the entry block, the block is deallocated and its index is removed. Index blocks are never deallocated by any of the primitive facilities because they are considered to be a necessary component of their parent entries. (Index block deallocation is done by higher level routines.)

The insertion primitive is the most complicated. Again, the first step is to locate the entry block into which the new entry should be inserted (if the target name is less than the first index name, the first entry block is considered.) If it can fit into that entry block, it is inserted (counts are adjusted, etc.) and the operation is complete. If no entry blocks currently exist, one is allocated, an index is created for it in the index block, and the entry is inserted into the new entry block. If an entry won't fit into an entry block then a re-adjustment is required: The logically neighboring block with the lowest number of entries is chosen, and the total space utilization of the two blocks is checked against a "good percentage" (85%). If utilization is less than this percentage, the entries of the two blocks (including the new one) are equally distributed between the two existent entry blocks; if the space utilization criterion is not satisfied, the neighboring block is no longer considered, and the entries (of the old entry block plus the new entry) are equally distributed between the existent block and a newly-allocated one.

2.2.3.4 TREE-STRUCTURE CONSIDERATIONS FOR BASIC MANIPULATIONS

Once the algorithms for the primitive manipulations against one level have been established, it is necessary to extend them to cover the entire tree structure. For MPE, the important constituent of a directory node is the information it contains; for directory manipulations, the important consideration is that a directory node may have subtrees (depending on the level of the node- The MPE directory is further distinguished from a generalized tree structure in that subtrees permissable for a given level are pre-defined).

The following descriptions apply to a basic class of directory uncallable intrinsics which primarily operate directly on the directory data structure, ignoring entry-dependent considerations described below. [These uncallable intrinsics are DIRECFIND, DIRECINSERT and DIRECPURGE.]

Both global references and subtree-type references are allowed for these functions. The latter type of specification permits a search to begin at a designated subtree directly, thus avoiding several accesses. [This form is primarily used when accessing logon group

2.2.4 Limitations

It may be seen from the description of the insertion algorithm that once the index block is full, that subtree can accomodate no more entry blocks. Thus the following situation exists when an entry cannot be inserted: the entry block that should accomodate the entry is full; a distribution with a neighboring block fails the goodpercentage test; and no more blocks can be allocated because there is no more room for indices. But not all the entry blocks are full at this point, so that we are getting partial utilization of all the available space. The space utilization in these cases will be discussed later.

As mentioned previously, entry blocks are not all of the same size. The block size depends on the block type. In this implementation, the following values were chosen:

	VARIABLE NAME	BLOCK SIZE	BLOCK FACTOR
SYSTEM ACCOUNT INDEX	SYSSAIBSIZE	3	62
ACCOUNT USER INDEX	SYSAUIBSIZE	1	19
ACCOUNT GROUP INDEX	SYSAGIBSIZE	1	19
GROUP FILE INDEX	SYSGFIBSIZE	2	41
ACCOUNT ENTRY	SYSAEBSIZE	3	13
USER ENTRY	SUSUEBSIZE	2	13
GROUP ENTRY	SYSGEBSIZE	2	10
FILE ENTRY	SUSFEBSIZE	2	42
(MAXIMUM BLOCK SIZE	SYSMAXBSIZE	3)	

Obviously, a tradeoff existed in the establishment of these values: potential available space vs. potential wasted space. These values were set in consideration of:

1. The need to accomodate more of certain elements (e.g. files).
2. The expected variance in number of elements at a given level.
3. The space utilization percentages for certain index/entry block sizes [the average space utilization and space utilization when full are dependent on the index block and entry block factors].
4. The lower the level the more critical is the value, because there are more blocks of that type (e.g. the system account index could've been much larger than 3 at small cost in space, since there's only one).

The block sizes are relevant primarily in determining block factors:

BLOCKFACTOR = FLOOR[((BLOCKSIZE*128)[-PREFIX])/ELEMENTSIZE]

The theoretical entry limit at a level is the (index block factor) * (entry block factor). But as was noted above, an insertion

When inserting or purging files under these provisions, however, the file space counts of the account entry and group entry must be adjusted. Consequently, such accesses are always global because the group and account entries must be visited anyway to update the counts. In addition, insertion insures that the user has BUILD access to the group. (When purging, the caller has supposedly determined that the user has WRITE access.)

2.2.3.6 SCANNING AND UPDATING

Certain commands are designed to operate on a set of directory entries [:LISTF, :LISTGROUP, :LISTUSER, :LISTACCT, :STORE, :RESTORE, :REPORT, :RESETACCT]. The concept of set is defined in the ERS and a consistent syntax is employed: the final directory level is implied by the command; lower levels precede higher levels, separated by periods; omitted trailing specifications imply "logon ..."; and leading "@"s mean "all ..." at the corresponding level.

There is one directory uncallable intrinsic that is used for all these functions [DIRECSCAN]. As arguements, it takes a "recipient procedure" and a parameter array; it scans the designated subtree in preorder, visiting each node [preorder is the traversal of a tree by visiting its root node before traversing its subtrees (in preorder)]. This routine visits a node by calling the "recipient procedure", passing it the visited entry and the parameter array, which has no relevance to the directory routines.

The recipient procedure can- each time it is invoked- simply examine the contents of the directory node [:LISTF, :LISTGROUP, :LISTUSER, :LISTACCT, :REPORT, :STORE], or actually modify it obeying certain restrictions described in a later section [:RESETACCT, :RESTORE]. Because the entry can be changed, this routine can be used as an update procedure [:RESTORE]. The directory can be released by the recipient procedure if necessary (such would be the case when return to the directory routine might be after a relatively long period of time (e.g. listing)). This directory routine uses the pointer reference count mechanism described earlier to ensure that the current index block is not removed, when the directory is unlocked.

For later discussion, it will be helpful to note at this time that the specification of a subtree of targets requires the specification of a root and the leaf level; the root specification depends on whether this is a global, logon account or logon group specification.

2.3 DIRECTORY UNCALLABLE INTRINSICS

2.3.1 Common Conventions

2.3.1.1 ENVIRONMENT

All directory uncallable intrinsics must be called from the stack. If called with any of the following SIRs locked, the caller must respect this nested SIR order:

FILESIR (37)   DIRECTORY (8)   LPDT (9)   DISC SPACE (12)

The directory routines always acquire the directory SIR, but don't directly invoke any procedures which acquire any other SIRs, except, of course, the "recipient procedure" of DIRECSCAN.

The following intrinsics are always invoked by the directory uncallable intrinsics:

GETSIR/RELSIR	- to acquire and release the directory
EXCHANGEDB	- to shift to the DDS and the directory space data segment, if necessary
ATTIO	- if an i/o is required. i/o will not be required only if the necessary directory block(s) is already in the DDS.

The directory routines access SYSGLOB +62 (system disc address) and SYSGLOB +44 (DIRBASE).

2.3.1.2 RETURNS

All directory uncallable intrinsics, except DIRECLOGON/DIRECLOGOFF, are double procedures and use the following returns [shown are (S-0), (S-1) and "cc"; these will be referenced in the various intrinsic descriptions to show the possible returns from the particular intrinsic]:

failure will occur before the entry blocks are entirely full. Simulation has shown that the entry blocks will be between 79 - 82% full when the first failure occurs at a level. Thus, the practical limit of number of entries at a level is about 80% theoretical limit. The ERS shows these practical limits for entries per level, based on 80.5%. Note that this is the expected utilization at the first failure: it may be possible to still add entries which will be accepted if they fall within partially full entry blocks, or distribution is possible.

Simulation has also shown that, before failure, the entry blocks are 76 - 80% full on the average. So to calculate the space used during normal operation, the effective entry block factor to use should be about 78% of the actual block factor. The formula for the space used by the directory in the ERS, uses these practical block factors in the ceiling expressions, which are the number of entry blocks used by the indicated entries. The formula is a simplification of:

```
space bitmap                                         3
+ index/entries for accounts                         +3+3*a/10.15
+ (#accounts) * (index/entries for groups)           +a*(1+2*g/7.8)
+ (#accounts) * (index/entries for users)            +a*(1+2*u/10.15)
+ (#groups) * (index/entries for files)              +a*g*(2+2*f/32.75)
```

These initial parameters are common to all these intrinsics and serve to define the target by specifying:

1. What level is the desired target (file, group, account, user);

2. If this is a shortened search, where to start. This occurs in the case of finding an entry in the logon group or logon account. The JIT contains index block directory pointers to the logon account group index block (subtree of groups in logon account) and logon group file index block (subtree of files in logon group). One of these pointers can be passed to the directory routines to bypass one or two levels of the search.

The parameters serve these purposes as follows:

TYPE. (11:2) = ENDLEVEL = the final target level:

- Ø file
- 1 group
- 2 account
- 3 user

TYPE. (13:3) = STARTLEVEL = specification of starting subtree for search:

- Ø GLOBAL - Full global search: ANAME [GUNAME [FNAME]] used (depending on level). INDEX ignored.
- 1 LOGON ACCOUNT - Group or file in logon account. INDEX is directory pointer for the appropriate account group index block (cf. JIT). GUNAME [FNAME] used as required.
- 2 LOGON GROUP - File in logon group. INDEX is directory pointer for the appropriate group file index block (cf. JIT). FNAME used.

INDEX may contain a directory index block pointer, as defined by TYPE. (13:3).

All name parameters are 4-word arrays, right padded with blanks, containing:

ANAME	account.
GUNAME	group or user name, depending on TYPE.(11:2)=3.
FNAME	file name.

Each is used as dictated by TYPE.(13:3).

cc	(S-Ø)	(S-1)	Meaning
CCE	Ø	Ø	Successful return.
CCL	Ø	Ø	I/O error. (Currently never returned. The routines SUDDENDEATH (4) upon detecting any i/o error.)
CCG	1	Ø	Duplicate name on insertion.
	2	n	Non-existent name at some point in search. (S-1) indicates the level of the non-existent node: Ø file 1 group 2 account 3 user
	3	n	User does not have SAVE access to <n>: 1 group 2 account
	4	n	No room. Insertion failed because subtree cannot accommodate any more entry blocks. n% of total entry space actually in use.
	5	Ø	No room. More than 65K entries. (Currently never returned because of block sizes).
	6	n	No room. System directory has no room to accommodate <n> contiguous blocks.
	7	Ø	Entry cannot be purged because it (or some constituent of its subtree) is in use. See ERS for in-use conditions.
	8	n	Permanent file space limit would be exceeded for <n>: 1 group 2 account

Every routine can return CCE, CCL and certain CCG failures. Those CCG failures possible are noted in the descriptions. For any CCG failure, the directory is left in the state when called - no part of the operation has been effected.

2.3.2 General Directory Entry Manipulation

2.3.2.1 CALLING CONVENTIONS

All of the following uncallable intrinsics use a call of the following form (appended by additional intrinsic-relevant parameters):

```
DOUBLE PROCEDURE directory (TYPE, INDEX, ANAME, GUNAME, FNAME, ...);
    VALUE TYPE, INDEX, ...;
    INTEGER TYPE, INDEX, ...;
    ARRAY ANAME, GUNAME, FNAME, ...;
        :
    OPTION EXTERNAL;
```

2.3.2.4 PURGING

The following uncallable intrinsic attempts to purge the designated entry and all of its subtrees from the directory:

```
DOUBLE PROCEDURE DIRECPURGE (TYPE, INDEX, ANAME, GUNAME, FNAME);
    VALUE TYPE, INDEX;
    INTEGER TYPE, INDEX;
    ARRAY ANAME, GUNAME, FNAME;
    OPTION EXTERNAL;
```

All parameters are described above.

Purging proceeds as follows:

FILES - The directory entry is removed.

GROUPS - All files of the designated group are FDELETEd and removed from the directory; and the group entry is removed. [In performing a DIRECPURGE of a group, the following filespace accounting consideration is made: the Number of sectors released via FDELETE are accumulated and subtracted from (the group entry, which remains if any of its files are in use, and) the account entry. This is the only content-dependent action performed by any of the uncallable intrinsics in this section.]

USERS - The entry is removed.

ACCOUNTS-All users and groups of the designated account are purged (as above) and the account entry is removed; unless all groups and users are not removed.

As noted in the ERS, certain conditions will preclude the removal of an entry. No father node will be removed unless all its sons have been removed.
This intrinxic obtains the FILESIR (before the directory SIR), when purging account, group or file entries.
The possible returns are:

CCE
CCL
CCG 2,7

2.3.2.5 SCANNING/UPDATING

For background information about the following routine, refer to the discussion under "DIRECTORY-OVERVIEW". The following routine traverses a designated subtree in preorder, visiting (by invoking a parameter-procedure) each entry encountered

2.3.2.2 INSERTION

The following uncallable intrinsic inserts an entry into any level of the directory:

```
DOUBLE PROCEDURE DIRECINSERT (TYPE, INDEX, ANAME, GUNAME, FNAME, NTRY);
    VALUE TYPE, INDEX;
    INTEGER TYPE, INDEX;
    ARRAY ANAME, GUNAME, FNAME, NTRY;
    OPTION EXTERNAL;
```

TYPE, INDEX, ANAME, GUNAME, FNAME are described above.

NTRY is words (4:n) of the appropriate entry as shown in Figures 2.1-A,B,C,D, [n is the last word.] Space for index pointers (as required) must be provided, but their contents are supplied by DIRECINSERT: account user index blocks and account group index blocks are established for new accounts; and group file index blocks are established for new groups.

The possible returns are:

CCE
CCL
CCG 1,2,4,5,6

2.3.2.3 FINDING

The following uncallable intrinsic locates and returns any directory entry:

```
DOUBLE PROCEDURE DIRECFIND (TYPE, INDEX, ANAME, GUNAME, FNAME, NTRY);
    VALUE TYPE, INDEX;
    INTEGER TYPE, INDEX;
    ARRAY ANAME, GUNAME, FNAME, NTRY;
    OPTION EXTERNAL;
```

TYPE, INDEX, ANAME, GUNAME, FNAME are described above.

NTRY must be large enough to accommodate the entire entry, which will be returned on successful completion.

The possible returns are:

CCE
CCL
CCG 2

```
in RECIP:
      .
      .
      .
   INTEGER ARRAY
        ARRZØ (*) = Q+Ø,
        ARRQ1 (*) = Q+1;
   INTEGER
        DELTAQ = Q+Ø;
      .
      .
      .
   PARMSDISPL := PARMSDISPL -DELTAQ;
      .
      .
      .
   TOS := ARRQØ (PARMSDISPL);
   ROS := ARRQØ
   TOS := ARRQ1 (PARMSDISPL);
      .
      .
      .
```

The directory can be released* (unless the DDS is changed) by RECIP, using SIRS as follows:

```
   TOS := SIRS;
   RELSIR (*,*);
```

RECIP can simply examine NTRY (e.g. to list it), can alter NTRY, or alter any part of the DDS*, provided of course, that the contents of the DDS remain consistent. [Refer to the DDS description in "DIRECTORY-OVERVIEW" for definitions of the references that follow.]

RECIP can make use of WORKAREA, for its own needs.

RECIP can change AREA A and/or AREA A DESCRIPTION*, but must set*/ DB+142 ["DADIRTY"] to TRUE if either is changed. [NTRY will always be in AREA A when RECIP is called.]

RECIP can change AREA B and/or AREA B DESCRIPTION*, but must set DB+164 ["DBDIRTY"] to TRUE if either is changed. [The index block for the subtree containing NTRY will always be in AREA B when RECIP is called.]

RECIP can invoke any directory routines, from the stack. [The caller must respect all SIR orders, however.]

RECIP can, in general, avail itself of any information in the DDS; e.g. DYNAMIC GLOBAL AREA and STATIC GLOBAL AREA.

---

*see, however, restrictions on "bracketted" visits, below.

```
DOUBLE PROCEDURE DIRECSCAN (TYPE, INDEX, ANAME, GUNAME, FNAME, RECIP, PARMS);
    VALUE TYPE, INDEX;
    INTEGER TYPE, INDEX;
    ARRAY ANAME, GUNAME, FNAME, PARMS;
    INTEGER PROCEDURE RECIP;
    OPTION EXTERNAL;
```

The first five parameters are used to define the subtree desired as will be described later. The last two are used for the "visit" of a node.

For each target entry hit in the designated subtree, the following procedure-provided by the caller- will be invoked:

```
INTEGER PROCEDURE RECIP (NTRY, LEVEL, PARMSDISPL, SIRS);
    VALUE LEVEL, PARMSDISPL, SIRS;
    ARRAY NTRY;
    INTEGER LEVEL, PARMSDISPL;
    DOUBLE SIRS;
```

The procedure is passed to DIRECSCAN via the RECIP parameter. It is invoked with DB at the directory data segment (DST #20) and the directory SIR locked. NTRY is a pointer to the current directory entry being visited (note that this is a pointer into the DDS); and LEVEL indicates the kind of entry:

Ø file
1 group
2 account
3 user

PARMS - as passed to DIRECSCAN - is an array of information relevant only to the caller's application and will be passed to RECIP by DIRECSCAN. It can contain parameters required by RECIP as well as space for "OWN variables" of RECIP; none of its contents is pertinent to DIRECSCAN. Because PARMS is a stack array, it is "passed" to RECIP as a negative, DIRECSCAN, Q-relative displacement, PARMSDISPL. RECIP must subtract delta-Q of its stack marker in order to access the first element of PARMS from the DDS. For example:

<allflag> = TYPE.(10:1) meaning "all of <endlevel>"

<tolevel> = TYPE.(7:3) (leaflevel)

0 Files

1 Groups

2 Accounts

3 Users

In addition, <hitflag> (TYPE.(6:1)) indicates that every node encountered (i.e. those hit in finding the root) should also be visited. This seemingly complex notation can be summarized by the table on the following page:

[RECIP is not as unrestricted as indicated above for certain, "bracketted" visits of particular DIRECSCAN invokations, specified below.] RECIP cannot release the directory (using SIRS) if it alters the DDS; but should release it if performing actions possibly causing an indefinite wait (e.g. non-disc i/o; PUTMSG; etc.) RECIP returns an indication to DIRECSCAN of how to proceed with the traversal, and also an indication of whether the directory was unlocked or not:

RECIP.(15:1) = SIR action
- Ø directory SIR was released
- 1 directory SIR was not reLeased

RECIP.(13:1) = scan continuation
- Ø continue traversal
- 1 skip this entry's subtrees (i.e. visit the entry's "brother" next)
- 2 stop traversal (i.e. return to DIRECSCAN caller).

RECIP should not re-acquire the directory SIR once it is released. RECIP should release all resources it has acquired before returning to DIRECSCAN; DB must be at the DDS on return.

DIRECSCAN Subtree Definition Parameters

In order to specify the subtree desired, it is necessary to supply:

a. Domain - global, logon group, logon account.
b. Subtree root - e.g. an account; all groups of an account etc.
c. Leaf level

TYPE is extended to supply this information. [note that in the previous uncallable intrinsics only domain and (one) node had to be supplied.] TYPE specifies the following request to DIRECSCAN.

"VISIT <tolevel> OF [<allflag>]/endlevel> STARTING AT <startlevel>"

where

<startlevel> = TYPE.(13:2)
- 0 System root.
- 1 Account group index (supplied)
- 2 Group file index (supplied)

<endlevel> = TYPE.(11:2)
- 0 Files
- 1 Groups
- 2 Accounts
- 3 Users

The preceding table shows, in brackets, the additional entries visited when TYPE.(6:1) is set. RECIP can only examine or alter NTRY for these visits (setting DADIRTY if altered) and must leave the DDS otherwise in tact. An extra GETSIR is performed before calling RECIP for these entries so that an attempt to RELSIC will be ineffective.

The table is simply a summary of the effects of TYPE extended to include <allflag>, <tolevel> and <hitflag>. It is useful when the desired subtree is pre-determined. However, the formulation of TYPE by sub-fields may be necessary if the subtree is not known beforehand (e.g. user specification). The reader is referred to the command interpreter procedure PRODUCEPARMS for a routine which analyzes a subtree specification and forms the appropriate DIRECSCAN parameters.

The possible returns from DIRECSCAN are:

CCE
CCL
CCG 2

CCE will be returned if the tree is found; this includes a null tree (never invoking RECIP) or a traversal stopped by RECIP.

2.3.3 Special File Entry Manipulations

The following uncallable intrinsics have been provided to account for special considerations for:

a. permanent file space accounting
b. file security

All these intrinsics apply to files (actually directory file entries), and three of them always use global domains. For these last three, the TYPE and INDEX - which are Ø, Ø implicitly - have been replaced with NUMSECTS, a variable used for file space accounting.

To minimize the possibility of the directory containing a file entry pointing to garbage, the file should be fully created on disc before DIRECINSERTFILEing and removed from disc after DIRECPURGEFILEing.

LEAF	SUBTREE	TYPE	INDEX	AN	GUN	FN	ENTRIES VISITED [if TYPE.(6:1)]
FILES	F.G.A	%000	*	A	G	F	[AG](F)
	F.G	%001	agi	*	G	F	[G](F)
	F	%002	gfi	*	*	F	[ ](F)
	@.G.A	%040	*	A	G	*	[AG](fff...)
	@.G	%041	agi	*	G	*	[G](fff...)
	@	%042	gfi	*	*	*	[ ](fff...)
	@.@.A	%050	*	A	*	*	[A](g(fff...)g(fff...)...)
	@.@	%051	agi	*	*	*	[ ](g(fff...)g(fff...)...)
	@.@.@	%060	*	*	*	*	[ ](a(g(fff...)g(fff...)...)a(g(fff...)g(fff...)...)...)...
GROUPS	G.A	%110	*	A	G	*	[A](G)
	G	%111	agi	*	G	*	[ ](G)
	@.A	%150	*	A	*	*	[A](ggg...)
	@	%151	agi	*	*	*	[ ](a(ggg...)
	@.@	%160	*	*	*	*	[ ](a(ggg...)a(ggg...)...)
USERS	U.A	%330	*	A	U	*	[A](U)
	U +	%330	*	A	U	*	[A](U)
	@.A	%370	*	A	*	*	[A](uuu...)
	@ +	%370	*	A	*	*	[A](uuu...)
	@.@	%360	*	*	*	*	[ ](a(uuu...)a(uuu...)...)
ACCOUNTS	A	%220	*	A	*	*	[ ](A)
	@	%260	*	*	*	*	[ ](aaa...)

Notes - * means parameter ignored.

+ because the account user index for logon account is not available, this case is handled like global request.

[ ] indicates entries visited if TYPE.(6:1) set; see following discussion for restrictions.

( ) shows subtree of entries traversed; capitalized entries are particular ones.

The first five parameters have the same meaning and usage as described in DIRECFIND.

FRETURN will contain the following on a successful return:

```
0 | FILE          |
1 | POINTER       |  }  2
2 | GROUP         |        }  1
3 | SECURITY      |              }  Ø          TYPE.(13:3)
4 | ACCT SECURITY |
```

The numbers on the right show the amount returned depending on TYPE.(13:3), the search domain. In other words, each account/group encountered will be used for FRETURN. These securities are to be used along with the file security to determine the user's access to the file, using ACCCHECK. [If FRETURN (2:4) is initialized to the logon group and logon account securities (c.f. JIT) then the true group/account securities for this access will be found in FRETURN (2:4) on return from DIRECFINDFILE.]

This intrinsic makes no other provisions for security or filespace accounting.

The possible returns are:

CCE

CCL

CCG 2

2.3.3.3 PURGING

The following intrinsic is used to remove a file entry from the directory and decrease the account/group filespace counts by the sectors used for the file:

```
DOUBLE PROCEDURE DIRECPURGEFILE (NUMSECTS, AN, GN, FN);
    VALUE NUMSECTS;
    DOUBLE NUMSECTS;
    ARRAY AN, GN, FN;
    OPTION EXTERNAL;
```

2.3.3.1 INSERTION

The following uncallable intrinsic is used to insert a file entry into the directory:

```
DOUBLE PROCEDURE DIRECINSERTFILE (NUMSECTS, AN, GN, FN, FADDR);
     VALUE NUMSECTS, FADDR;
     DOUBLE NUMSECTS, FADDR;
     ARRAY AN, GN, FN;
     OPTION EXTERNAL;
```

NUMSECTS is a positive double integer containing the number of sectors that are allocated and are to be accounted for.

AN, GUN, FN are the account name, group name and file name for the insertion. Note that they must be supplied (even for logon group/account).

FADDR is a double containing the VTAB index in (Ø:8) of the first word, and the sector address in the remainder of FADDR.

The routine increments the account and group file space counts by NUMSECTS; checks the counts against the limits; ensures that the user has SAVE access to the group; and (if no errors exist) inserts FADDR in the directory.

Possible returns are:

CCE

CCL

CCG 1,2,3,4,5,6,8

2.3.3.2 FINDING

The following uncallable intrinsic returns a file address and file securities:

```
DOUBLE PROCEDURE DIRECFINDFILE (TYPE, INDEX, AN, GN, FN, FRETURN);
     VALUE TYPE, INDEX;
     INTEGER TYPE, INDEX;
     ARRAY AN, GN, FN, FRETURN;
     OPTION EXTERNAL;
```

2.3.4 Logon and Logoff

It is necessary to perform some directory manipulations for logon and logoff that would be rather cumbersome and inefficient using only the above uncallable intrinsics. For this reason, the following intrinsics were implemented. It was noted earlier that in order to prevent purging of users, accounts and groups that were logged-on (providing fast-access group and account file domains) certain counts were maintained. The following two intrinsics adjust these counts for the designated user/account/group, and perform other actions necessary for logon and logoff.

The following intrinsic

1. Finds and returns to pre-defined locations:
   a. the user entry to stack DB+30
   b. the account entry to stack DB+50
   c. the group entry to stack DB+80
2. Increments (to prevent purging):
   a. the user entry logon count
   b. the account group index block pointer count
   c. the group file index block pointer count

```
INTEGER PROCEDURE DIRECLOGON (Z, JMATENTRY, D1, D2);
     VALUE Z, D1, D2;
     INTEGER Z;
     ARRAY JMATENTRY;
     DOUBLE D1, D2;
     OPTION EXTERNAL:
```

Z must be Ø.
JMATENTRY is a stack image of the pertinent JMAT entry.
D1 and D2 are ignored.

Returns:

(Condition code unchanged)
DIRECLOGON=

- Ø fully successful (all counts incremented)
- 1 group does not exist (only user entry logon count incremented)
- 2 user (and possibly group) does not exist (no counts incremented)
- 3 account does not exist (no counts incremented)

NUMSECTS is negative double integer containing the number of sectors that will be released.

AN, GN and FN are the account name, group name and user name all of which must be supplied and existent.

To use this intrinsic, it is necessary to have DIRECFINDFILEd the file at some time. NUMSECTS, derived from the file label, is passed to DIRECSCAN in order to decrease the account and group filespace counts. No provision for file security is needed, since the caller has supposedly already determined that the user has WRITE access to the file.

The possible returns are:

CCE
CCL

[No CCG return is applicable because AN, GN, FN must exist since the file exists, and the caller has accounted for file security and in-use.]

2.3.3.4 EXTENT ADDITION AND REMOVAL

The following intrinsic is used to adjust the account and group filespace counts, intended for extent addition and removal:

```
DOUBLE PROCEDURE DIRECADJUST (NUMSECTS, AN, GN);
    VALUE NUMSECTS;
    DOUBLE NUMSECTS;
    ARRAY AN, GN;
    OPTION EXTERNAL:
```

NUMSECTS is the positive/negative sector adjustment to be applied against the account and group;

AN and GN are the account name and group name.

This routine simply adjusts and checks the filespace counts of the indicated account and group. Again, AN and GN are assumed to be existent since they are derived from some existent file.

The possible returns are:

CCE
CCL
CCG 8

2.4 IMPLEMENTATION

The following discussion concerns the actual implementation of the directory routines. Overviews of the basic algorithms used have already been given, and much of the actual details will therefore be self-evident from the listings directly. This discussion, then, serves as a guide to the listings, and the listings should be used in conjunction with the following descriptions.

2.4.1 Naming Conventions

Being uncallable intrinsics (i.e. procedures), the directory routines allocate no global storage. They do, however, make use of global storage available to them in the DIRECTORY DATA SEGMENT (DDS), and the DIRECTORY SPACE DATA SEGMENT. These two data segments are first initialized by the configurator, and the information is subsequently maintained by the directory routines. The information of these data segments is assumed to be consistent, and no special checks for garbage data is made (in the data segments and implicitly in the directory as a whole).

The first seven pages of the listing are definitions of entry and index content displacements and fields; some system global information; definitions of the DDS and DIRECTORY SPACE DATA SEGMENT (DIRSPACE) contents; flags and flag fields for the directory routines; and miscellaneous stack declarations. The descriptions that follow will present the naming conventions employed. The next section will describe the meanings of some of the variables.

Word displacements into elements are denoted by prefixing the content identifier by a letter representing the element type: A, G, F, U and I (index). Element sizes are EQUATEd to identifiers named "xSIZE". Field definitions are suffixed by "...F", such as xPURGEFLAGF. The content displacements of index block prefixes are prefixed by "PRE...".

Each level of the directory is represented by a unique number that is used consistently:

0 Files.
1 Groups.
2 Accounts.
3 Users.

Refer to the diagram given earlier in this Section of the DDS and to Figure 2.2-2, for definitions of the areas and variables described below. Remember, the DDS contains two pairs of "twin" areas: AREA A / AREA A DESCRIPTION and AREA B / AREA B DESCRIPTION. The two description areas contain the exact same kind of information, positioned in the same relative order. The displacements from the beginning of the description areas for both

For logoff, the next intrinsic performs the following actions as directed by the caller's indication of which entries were found on a logon (attempt).

1. Finds the account entry, user entry, and group entry
2. Decrements
   a. the user entry logon count
   b. the account group index block pointer count
   c. the group file index block pointer count
3. Updates and checks the account and group connect and cpu times.

```
LOGICAL PROCEDURE DIRECLOGOFF (MASK, JMATENTRY, CONTIME, CPUTIME);
     VALUE MASK, CONTIME, CPUTIME;
     INTEGER MASK;
     ARRAY UMATENTRY;
     DOUBLE CONTIME, CPUTIME;
     OPTION EXTERNAL;
```

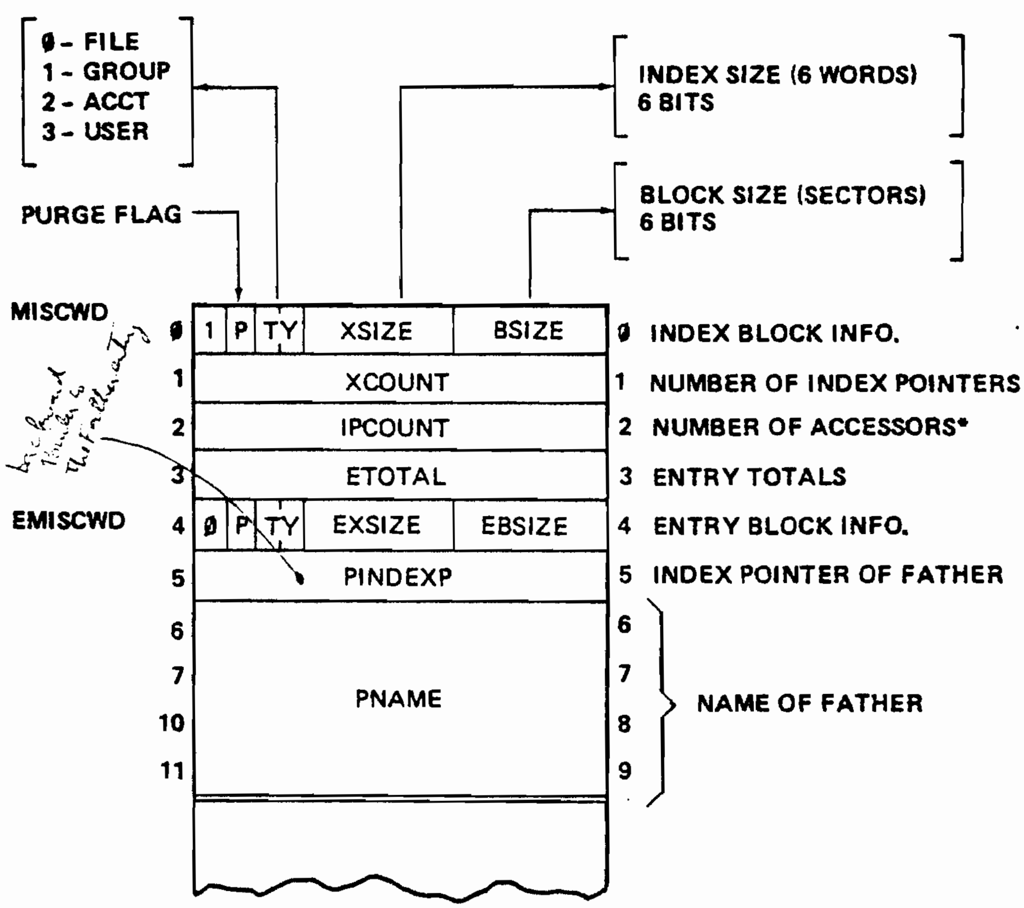
MASK is the integer returned by DIRECLOGON.
JMATENTRY is a stack image of the pertinent JMAT entry.
CONTIME and CPUTIME are (if MASK=Ø) the connect time and cpu time.

Returns:

If MASK <> Ø then Ø
If MASK =Ø then
   .(15:1) Account connect time exceeded
   .(14:1) Account CPU time exceeded
   .(13:1) Group connect time exceeded
   .(12:1) Group CPU time exceeded

[Note that the time counts are always incremented, and the "time exceeded" indications are just warnings.]

INDEX BLOCK PREFIX (1Ø WORDS)



* THE COUNT IS INCREMENTED BY EACH ACCESS THAT USES AND RELIES UPON A POINTER TO THE INDEX BLOCK, IE, IT IS GUARANTEED NOT TO BE PURGED WHILE THE COUNT IS ≠ Ø

Figure 2.4-A

areas are defined by a common set of EQUATEs. The names were chosen to represent the contents of the corresponding word. Both description areas are (not coincidentally) entirely directly addressable. These directly-addressable variables are defined for each area in terms of these displacements. AREA A DESCRIPTION variables are preceded by "DA...", and AREA B DESCRIPTION variables are preceded by "DB...". Those directory routines which are generalized to operate on either area use displacements based off BASE, a variable setup to point to the pertinent area; where possible, of course, the directly- addressable variable is used (this usually is the case when the target is an index (must be in B) or an entry known to be in A (entries appear in B only when splitting blocks on insertion)). When necessary, AREA A is represented by 0 and AREA B is represented by 1.

[When possible, each content displacement is defined by reference to the preceding displacement, so that variable insertions/deletions would be easier.]

Procedures have been named according to a loosely-defined convention regarding their "internalness". Specifically, the most basic procedures are named "D..."; the next higher level procedures are named "DIR..."; and only uncallable intrinsics are named "DIREC...".

2.4.2 Data Structure

An overview of the basic data structure used has already been given. The following description augments the listings and gives some of the implementation details regarding the data structure. Only those variables which are not already self-evident will be described.

2.4.2.1 DIRECTORY BLOCKS

The structure of the directory on the disc has been described in a previous subsection, where it was stated that the size of directory blocks is not constant. Otherwise, the standard block concept was used, in which each block simply is the physical unit of transfer and contains an integral number of entries (c.f. records). A minor exception to this is index blocks which contain a prefix.

The prefix is very important. A diagram of an index block prefix appears in Figure 2.4-A. It contains two words which describe the characteristics of this index block, and all of its entry blocks [MISCWD and EMISCWD]: level; element size; and block size. The high order bit of these words indicates whether this applies to an index block (1) or entry block (0). The only other piece of information that is necessary to fully characterize the block is the number of entries that it contains: for index blocks, this is in the prefix; for entry blocks this is in the entry block's index.

INDEX ENTRY (6 WORDS)

Field	Word	Description
IE1STNAME	Ø	1st NAME OF ENTRY BLOCK
	1	
	2	
	3	
IEPNTR	4	POINTER TO ENTRY BLOCK
IECOUNT	5	NUMBER OF ENTRIES IN a BLOCK

Figure 2.4-B

Note that the "misc" word and element count fully characterize a block. [Note also that not only can blocks at different levels be of different size, but they can be different even on the same level (the only restriction is that all entry blocks of an index block be of the same size).]

Additional information appearing in the index prefix is the pointer count, indicating the number of accessors relying on a pointer to the index block; a count of the total number of entries; and the index pointer and name of the "father" of this index block. It will be shown later that an index pointer and name are sufficient to locate any entry in the directory.

The indices immediately follow the index block prefix in the index block, and are diagrammed in figure 2.4-B.

2.4.2.2 DIRECTORY SPACE DATA SEGMENT

The directory space data segment (DIRSPACE) is shown in Figure 2.2-D. The swapping disc address of this data segment is adjusted to be coincident with the beginning of the directory disc area: it is swapped directly to this area which is referenced during cold load for initialization. In other words, DIRSPACE is as much an integral part of the directory as is the actual entry part.

Directory space is monitored using the bitmap in DIRSPACE which lies between word 2 and LASTWORD (inclusively). FIRSTWORD is a cycling pointer that points to the first word to examine for availability. The word pointed to by LASTWORD must be followed by a word of zeroes. The bitmap shows allocation of the entire directory disc area: word(0).(0:1) represents the first sector (0); word(1).(1:1) represents sector 17; etc. Because DIRSPACE is the first three sectors of the directory disc area; and the system root index is the first allocated and must obviously be at least one; word(0).(0:4) = 0, always.

2.4.2.3 DIRECTORY DATA SEGMENT (DDS)

A diagram of the DDS is shown in Figure 2.2-E. Its basic purpose is to serve as a buffer and working area for the directory management routines. A breakdown into its major areas and their use has been described above. Following is a more detailed, implementation-oriented description of the contents of this data segment.

The first 128 words are used- in addition to a miscellaneous workarea- as the area in which stack arrays are moved, for passing to and from internal directory routines. These arrays include names and/or whole entries.

The next area- "Dynamic Global Area"- is set up on every call to a directory uncallable intrinsic. It contains information pertinent

SIZE, resetting it whenever it comes across an allocated page (0). If it finds PPSIZE contiguous pages, it updates FIRSTAVAIL in DIRSPACE (to point to the word containing the newly-allocated pages); allocates them, using DIRXXXLLOCATE; and returns to the caller of DIRALLOCATE. A return to the procedure body implies a failure to find the pages between (and including) LOWLIM and UPLIM.

The procedure body of DIRALLOCATE uses FIND to examine, first, the bitmap following FIRSTAVAIL; and, if that fails, the portion before FIRSTAVAIL.

DIRDEALLOCATE simply calls DIRXXXLLOCATE to set the appropriate bits back to 1. The PPSIZE pages starting at PNTR must already have been allocated.

2.4.4 Primitive Utilities

DIRWRITE writes out the blocks contained in WHICH [WHICH defines the buffer area in question: 0=A, 1=B]. Because the block is written out, the area can be considered "clean", and the dirty flag is always reset. If the block is an index block, then information in the description area relating to the prefix updates the prefix in the buffer area before the block is written. The routine SYSABORTs if any error is detected in the write (the IOCB error code can be found in the X register).

DIRREAD reads at least one block's worth of pages into WHICH, starting at PNTR in the directory disc area. It returns immediately if the pertinent buffer already contains the requested block. If the buffer is "dirty", it calls DIRWRITE to update the disc. [The long sequence of code in comments was an attempt to achieve a very minor (if any) performance increase by not issuing a disc read if all the pages of the desired block are imbedded anywhere in the DDS.] The disc read is for the maximum number of pages that the DDS can hold. Again, if an i/o error is detected the routine SYSABORTs, with the IOCB error code in the X register. Once the appropriate block is read into the DDS, the corresponding description area must be set up. For index blocks, this information is obtained from the index block prefix. Therefore:

For index blocks, EXCOUNT and EEMISCWD are ignored.

The characteristics of a particular entry block are defined by IECOUNT of the entry block's index, and EMISCWD of that index' index block prefix. Therefore:

For entry blocks, EXCOUNT is the number of entries in the entry block (derived from the entry block's index); and EEMISCWD is the word characterizing the entry block, as defined by EMISCWD of the index block prefix.

So, from the last two parameters (entry blocks) or from the index to the particular call and useful or necessary information that can be used by internal routines or to correctly complete the request. This includes (DB-DL) for array moves; an exact copy of the first five parameters to the uncallable intrinsic (with INDEXP adjusted to be the final target index); security matrices that are initialized and updated as each account/group is encountered; and the return value from the initial GETSIR (DIRSIR).

As was described earlier, the DDS contains two buffers, A and B; and their corresponding "description" areas. The description areas fully characterize the contents of the buffers. The meanings of the information contained is evident from the descriptions in Figure 2.2-E. These description areas are used for both index blocks and entry blocks; index blocks, however, make use of an "extended" description area to contain information derived from the index block prefix. [When an entry block is read into the DDS. the index block prefix "miscwd", EMISCWD, is used to set the MISCWD in the appropriate description area for the entry block.]

Following the two description areas are the "static global area" containing invariant global information and the two buffers (separated by an "overrun area").

2.4.3 Space Management

DWRITEBITMAP simply updates the disc copy of the directory space bitmap, by writing out the current DIRSPACE bitmap onto the directory disc area. This is done whenever the bitmap is changed, due to an allocation/deallocation.

DIRXXXLLOCATE is a utility procedure used by directory allocation/ deallocation

which simply changes PPSIZE pages starting at PNTRIN to SETTO; i.e. changes the bits in the bitmap corresponding to those pages to 0 (allocate) or 1 (deallocate). It HALTs if it detects an allocation of an allocated page, or a deallocation of a deallocated page. It writes out the bitmap before returning.

DIRALLOCATE attempts to allocate PPSIZE contiguous pages, and (if successful) returns the directory pointer of the first page. The cc returns are:

CCE- Successful allocation. pointer returned.
CCL- Can't find PPSIZE contiguous pages. 0 returned.
CCG- PPSIZE is too large for the DDS as configured. 0 returned.

The subroutine FIND searches between the words of the bitmap delimited by LOWLIM and UPLIM inclusive. It does this by examining two words at a time, and therefore may examine UPLIM+1 (this is why LASTWORD must be followed by a zero). In examining the bits, it keeps track of the number of acceptable (i.e. contiguous) pages in

2.4.5 Management Primitives

2.4.5.1 INSERTION

DIRINSERT is the basic routine that inserts entries into a given subtree. The subtree is specified by INDEXPOINTER, which is the pointer to the index block representing the subtree. The entire entry must be located at DDS DB+0 onward. The possible returns from DIRINSERT are a subset of the returns prossible from the uncallable intrinsics:

OD- Successful.
Other- See CCG returns 0,1,4,5,6.

Subroutine ZINSERT is the routine that does the actual "spread and insert" of a specified index or entry. It puts the entire element defined by ELEMENT into the location PNTR (if it is non-0) in area WHICH, moving all the elements following PNTR one element-width to make room for the new one. If PNTR is 0, ZINSERT will determine PNTR. Note that the caller of ZINSERT has ensured that the element will fit and found the place for it at PNTR (if PNTR is not 0). If PNTR is 0, ZINSERT uses DIRSCAN to find the point where ELEMENT should be inserted (by looking for exact/next). ZINSERT returns the pointer where ELEMENT was inserted (supplied or determined by ZINSERT).

Subroutine ZNEWENTRYBLOCK allocates an entry block and makes it defined for the subtree by creating the proper index for it. The name of the first entry of the block (used for the index, also) is at NAME; INDEXPLACE (if non-0) is the pointer in area B where the new index should be put (INDEXPLACE can be 0, in which case the proper place for the new index, according to NAME, is automatically determined). If successful, the routine returns the pointer for the new entry block. If some failure occurs, the subroutine returns directly to DIRINSERT's caller with the proper error indication.

Subroutine ZSET merely initializes the following variables in anticipation of a "distribution" (on entry, ZT is the total number of entries (including the new one) involved in the distribution):

ZTOTAL := ZT * entrysize.
ZH1 and ZH2 are a division of zt into halves.
(if ZT is odd, ZH2 will be the greater one)
ZHALF1 and ZHALF2 are the word equivalents of ZH1 and ZH2.

Subroutine ZDISTRIBUTE uses the variables set up by ZSET to divide the overflowing entry block in area A, to equal entry blocks in area A and area B. It writes these new entry blocks on the disc.

The main procedure body follows the algorithm described earlier for insertion. It first reads in the designated index block (,makes a check that there are fewer than 65K entries in the subtree (!)) and block prefix, the entire description area is initialized.

DIRNEWINDEX sets up description area B to be a null index block with the specified attributes:

IBSIZE is the index block size.
ILEVEL is the level.
EBSIZE is the entry block size.
ESIZE is the entry size.

The caller must ensure that area B is clear and can accept the initialization; and must also have moved the father name and index pointer into DBPNAME and DBPINDEXP. Once area B is initialized, it is written out onto the area DIRNEWINDEX has obtained using DIRALLOCATE. The returns are:

CCE- okay.
CCG- IBSIZE or EBSIZE is too large for DDS as configured.
CCL- can't allocate IBSIZE contiguous pages.

DIRSCAN is the general routine that searches area A or area B for a given element:

ENTRYNAME (a misnomer) is the DDS address of the element name.
TYPE'WHICH.(13:2) is the type of request:
  0 Search for exact element match.
  1 If no exact match, return next element.
  2 If no exact match, return preceding element.
TYPE'WHICH.(15:1) is the area to search:
  0 A.
  1 B.
[All combinations of TYPE'WHICH are symbolically defined, globally].

A DDS pointer is returned:

CCG- Exact element returned.
CCL- Next [preceding] element returned.
CCE- No exact or next [preceding] element found. Pointer to "pseudo element," beyond last [before first] returned.

Currently, a simple linear search is made for the element exactly matching, or the "lowest" element "greater" than the target, ENTRYNAME. That will be the exact hit, or the place where the target, logically, falls. [Some improvement might be obtained by simply changing the implementation of this routine to a binary search.]

2.4.5.2 FINDING

DIRFIND will locate an entry of a specified subtree. The subtree is represented by the index block pointer, INDEXPOINTER; and the entry name is expected to be found in DDS DB+0 thru DB+3. If not found, 0D is returned; otherwise,

(S-0) = Address of entry (always in area A).
(S-1) = Address of entry's entry block index (always in area B).

The algorithm consists simply of reading the index block; finding the containing entry block (DIRSCAN exact/preceding); reading the entry block; and locating the entry.

2.4.5.3 DELETING

DIRREMOVE simply removes ELEMENT from area <WHICH>. It sets DxDIRTY; decrements DxXCOUNT; and adjusts DxUSED. It removes the element by performing a move of the following entries over the target. If this is an entry block that is thereby depleted, its space is deallocated. [Remember that while entry blocks are deallocated when empty, index blocks remain as long as their parent exists.]

2.4.6 Uncallable Intrinsics

The routines described above are general procedures that operate on the data structure of the directory, ignoring the contents of the entries that they manipulate. In fact, they are generally ignorant even of the hierarchial structure, and operate only on one level of a subtree. The following procedures effect the extensions necessary for the hierarchial structure and other content-dependent considerations (i.e. permanent file space and security). Obviously, these routines make use of the "one-level" procedures described above.

Generally, the implementations of the actual uncallable intrinsics follow directly from the descriptions already given. Therefore, only comments and unusual operations will be described for the uncallable intrinsics.

2.4.6.1 TWO UTILITY PROCEDURES

DIRSTARTOFF is used by almost every uncallable intrinsic. A great part of the parameter specification of the different uncallable intrinsics is identical (TYPE, INDEX, ANAME, GUNAME, FNAME, ...). This is because these parameters essentially serve to define one target, which is to be inserted, found or removed. Thus, the setup for the operation is the same for the different instrinsics, with deviations occuring once the target is established. The specification of this target can be non-trivial (admitting to any tries to locate the entry block that is to contain the target entry by using DIRSCAN on the index block, looking for an exact/preceding hit. If there is no such block, it will consider the first entry block for the insertion; and if no entry blocks exist at all, it will create a new one (and index) using ZNEWENTRYBLOCK (plus setup of area A) and insert the target entry in the new entry block in the normal fashion.

Once an entry block has been determined, a test is made to see if the entry will fit into the entry block. If it will, the entry block is read in, the entry is inserted (using ZINSERT), some variables are updated to reflect the new entry, the name in the index is changed (in case the new entry is the first of the block), and the entry block and index block are written out.

If the entry doesn't fit into the entry block, then either a distribution with an existing (logical) neighbor block, or distribution with a newly-allocated entry block must occur. It will be distributed with a neighbor if one exists and if the total number of entries of the two blocks (plus the new one) is less than GOODPERCENT of the total number of entries that can be accomodated. [If there is only one neighbor, it is chosen; if there are two, the one with the lower number of entries is chosen.] Note that- at this point- no entry block has actually been read as all calculations were made using the entry counts of the indexes.

If a distribution of two existing blocks is attempted, the lower one is read into area A, and the higher one is read into the DDS at the address immediately following the last entry of the lower block. This results in one, overflowing entry block in A, into which the new entry is to be inserted. Area B may be destroyed by this operation (if it isn't, it will be by the following distribution). After kluging some area A description variables so that the entire (overflowing) block is considered, the new entry is inserted with ZINSERT. Assuming no duplicate entry exists, the whole mess is now distributed into entry blocks in both area A and area B by ZDISTRIBUTE, which writes out the two entry blocks; the indexes are corrected and the index block is written out.

When distribution with a new block is required, the existing entry block is read into area A; space for a new entry block, and a new index, is established by ZNEWENTRYBLOCK; the index is updated and the index block written out. A null entry block representing the newly-allocated one is set up in area B. The existing entry block is "overflowed" by a ZINSERT of the new entry, but it is distributed between the old and new block by ZDISTRIBUTE (writing out both entry blocks). The index block has already been updated and written out.

backward, until the root is reached.

2.4.6.2 INSERTION

DIRECINSERT performs a DIRSTARTOFF and- once index blocks are allocated for groups and users (accounts) or files (groups)- does a DIRINSERT of the entry. If any error is detected, any index blocks allocated must be deallocated.

DIRECINSERTFILE calls DIRSTARTOFF (with the options of incrementing filespace counts and checking save access) and then performs a simple DIRINSERT of tne directory file pointer entry, as supplied by the caller.

2.4.6.3 FINDING

DIRECFIND performs a simple DIRFIND, after the target is positioned by a standard call to DIRSTARTOFF.

DIRECFINDFILE performs the same actions that DIRECFIND does, except that the file pointer and file security (possibly) are returned (instead of the entire entry). Both group and account, just group, or no file securities are returned depending on "startlevel" as described in the previous definition.

2.4.6.4 PURGING

As described earlier, the algorithm for purging consists of scanning the target subtree in endorder, deleting a node only if its entire subtree is deleted. The routines that perform this use the following conventions: each returns a double representing the total number of file sectors removed (as returned by FDELETE), with CARRY set if the entire entry/subtree has been removed. The purge procedures are divided into: a procedure which removes a subtree; four procedures designed to remove entries; and the actual uncallable intrinsic procedure. As might be expected, the routines for the "higher" levels make use of the "lower" level routines, possibly causing several layers of recursion: the uncallable intrinsic calls an entry purge procedure, which may call the purge subtree perocedure, which in turn will call an entry purge procedure, which may again call the purge subtree procedure, and so on.

DIRPURGESCAN is the "purge subtree" procedure. The index block for the target subtree is located in area B on entry, and the procedure is passed the appropriate "entry purge" procedure in PURGER. It simply scans B's index block, reading in each entry block and attempting to purge each entry using PURGER. If an entry block becomes depleted, the index is removed (the entry block has been deallocated by DIRREMOVE called from PURGER); otherwise DIRPURGESCAN updates the directory by writing out each non-empty "starting point" and any target level). The essential purpose of DIRSTARTOFF, then, is to thread through the directory hierarchy, performing some useful functions as it goes, and winding up with the final target defined by one name and one index defining the subtree.

DIRSTARTOFF analyzes the specification part (first 5 parameters) for directory uncallable intrinsics, and performs any directory accesses necessary to leave:

XINDEXP to the index block pointer of the target entry; and
DB+0 thru DB+3 to the target name.

In addition, the following DDS variables are initialized:

ADJUST to the stack (DB-DL) [for moves];
XTYPE to the TYPE parameter;
XANAME, XGNAME and XFNAME to the corresspondinq stack
addresses;
XASEC and XGSEC to the security matrices of the account/group
as (if) encountered: -1,-1D indicate not encountered; and
SIRRETURN to the inital GETSIR (DIRSIR), performed
by DIRSTARTOFF.

Input parameters:

PARR is stack address of intrinsic's parameters;
NUMSECTS, if specified, is added to account and group filespace
counts.
RECIP and PARMS, if specified, causes each entry actually
encountered, to be visited by RECIP (passing in PARMS),
like DIRECSCAN; and
if only PARMS specified, then S access to group is verified.

This procedure returns 0D, if successful; or a double word identical to the CCG returns for the uncallable intrinsics described earlier.

Implementation follows directly from the description given above. Basically, it consists of an initialization portion, followed by a switch transferring control to the code necessary to handle the "start" level of TYPE.(13:3). Each level automatically drops to the lower level, until the "end" level is hit. At each level, the following processing is performed:

1. Actually find the relevant entry;
2. [Check for save access;]
3. [Bump the filespace count;]
4. [Visit the entry;]
5. And set XINDEXP to the next "subtree".

DIRRESET is used by routines when they determine that the filespace counts of the [group [and account]] of the current access need to be adjusted, by NUMSECTS. Using DBPINDEXP and DBPNAME, it traces

GETSIRRESULT), which is to perform the actual visit.

DIRSCANTREE performs a normal scan by reading in the appropriate index block, reading in each entry block, and "hitting" each entry. But because RECIP can release the directory, the DDS and the directory may have changed. For example, perhaps the directory was released by RECIP and some entry close to the current point of traversal is removed. In other words, nothing about the directory can be guaranteed. For this reason, DIRSCANTREE uses the pointer reference count mechanism to ensure that at least the current index block will remain, and finds the "next" entry by saving the "next" name (in the stack) and performing the find using only the index block pointer (known to be existent) and a "candidate" for the next name (which may or may not exist). Like DIRDOENTRY, DIRSCANTREE adjusts PARMS so that it always contains the correct Q-relative displacement for RECIP. The index is always restored after returning from DIRDOENTRY, and before leaving DIRSCANTREE the pointer reference count is decremented. Note that the use of DIRSCANTREE and DIRDOENTRY in this way results in a recursion similar to that effected by the purge routines, but purging is a traversal in endorder.

DIRDOENTRY is responsible for performing a proper visit of the entry using RECIP. Entry ELEMENT of level, LEAFLEVEL, is visited by passing PARMS (adjusted) to RECIP, as described earlier. DIRDOENTRY saves information it (or subsequent routines) need; performs the call; possibly re-locks the directory SIR and restores some information; and either invokes DIRSCANTREE for ELEMENT's subtree or simply returns, depending on whether LEAFLEVEL has yet been reached or not. [The reason DIRDOENTRY is separate from DIRSCANTREE is that either of them may be called from DIRECSCAN:]

DIRECSCAN calls DIRSTARTOFF in a standard fashion, or requesting visits, depending on "hitflag", TYPE.(6:1). When DIRSTARTOFF visits entries, it is depending on DDS addresses to continue its operation: it does not perform the save/restore that the normal scan procedures do, and that is why RECIP is restricted in the actions it can perform when operating on entries visited by DIRSTARTOFF. DIRECSCAN then calls DIRSCANTREE or DIRDOENTRY depending on whether a "pure" subtree is being requested or an entry (and its subtrees). ["@" is a specification for a subtree, as opposed to an explicit entry specification; see the DIRECSCAN chart accompanying the uncallable intrinsic definition above.] If returning from either routine with an unrecorded change in the DDS, the relevant block is written out.

2.4.6.6 MISCELLANEOUS

DIRECADJUST performs the requested adjustment of the filespace counts by simply calling DIRSTARTOFF with the "adjust option". Note that the specification request to DIRSTARTOFF is for "all files", so that no access is attempted for a file.

entry block. Before returning, area A is restored to what it had on entry. The (possibly empty) index block is left in area B.

The following four entry purge routines have the following in common: they require the entry to be purged to be in area A and pointed to by NTRY, and if area B must be destroyed by the purge operation for the entry, it must be restored by the "entry purge" routine before exiting. [The double result and CARRY conventions are still obeyed.] Each entry is removed by DIRREMOVE ((which will set DADIRTY or deallocate the entry block when depleted); or if it cannot be removed because it is in use, DADIRTY is set if the entry is at all changed (e.g. setting a (currently unused) "purge" flag).

DDELFILE determines the disc LDN from the VTAB and calls FDELETE.

DDELUSER, the simplest, is straightforward.

DDELGROUP saves the pointer for B, invokes DIRPURGESCAN on the group entry's group file index block, and removes the entry or flags it depending on the result from DIRPURGESCAN.

DDELACCT saves B's CONTENTS, invokes DIRPURGESCAN for the account user index, invokes DIRPURGESCAN for the account group index, and either removes the account entry or flags it depending on both returns from DIRPURGESCAN.

DIRECPURGE and DIRECPURGEFILE differ only slightly and are implemented by the same porcedure. DIRECPURGE performs a standard DIRSTARTOFF for positioning; whereas DIRECPURGEFILE performs a DIRSTARTOFF with adjustment. In both cases, the entry is located and removed: by calling the appropriate "entry purge" routine for groups, users and accounts; or by simply DIRREMOVEing file entries (FDELETE is invoked only when purging groups (and accounts); it is assumed that the file is FOPENed when purge is required and the caller is responsible for deallocating the file's disc space after return from DIRECPURGE[FILE]). If the entry is fully purged, the index is updated, or removed, and the index block is written out. If the entry was not fully removed, an indication is returned to the caller, and - if it was a DIRECPURGEFILE - the filespace counts are returned to the values they were before DIRSTARTOFF adjusted them. The account's filespace count is adjusted for the files removed by a DIRECPURGE of a group.

2.4.6.5 SCANNING

In order for DIRECSCAN to traverse the specified subtree in "preorder" as described in the preceding definition, it makes use of two procedures: one that scans a subtree, and one that actually performs the visit.

DIRSCANTREE traverses the tree specified by INDEX to the level, LEAFLEVEL, in preorder. For each entry encountered in the traversal, DIRSCANTREE invokes DIRDOENTRY (with RECIP, PARMS and

DIRECLOGON/OFF, quite straighforwardly, performs the actions defined earlier. Because it does not require any positioning, DIRSTARTOFF is not invoked, and DIRECLOGON/OFF does any necessary initializing actions itself.

2.4.7 Improvements and Extensions

Retaining the basic directory structure and algorithms just described, it may be profitable to incorporate some of the following extensions. In most cases (with the possible exception of the binary search), the performance increase or additional capability must be considered a "frill" at the time of this writing: either the performance increase is marginal (if existent), or the case occurs too infrequently to justify alterations at this time.

1. Propagate an i/o error detected in DWRITEBITMAP, DIRWRITE or DIRREAD back to the caller.
2. Change DIRSCAN so that it uses a more efficient search than a linear one (e.g. binary).
3. For insertion, if two existent blocks fail the "GOODPERCENT" test but there is room for one more entry and no new entry block can be allocated, insert and distribute with existent blocks.
4. For insertion, if target entry would be the last one of a full block, try to insert it in the next block.
5. For insertion, develop some scheme of distributing the entire subtree before failing because of no room.
6. Change high level routines so that an index block is not allocated until it is needed.
7. Return actual cause of the in-use failure from DIRECPURGE.
8. Change DIRSTARTOFF so that the special entry visits due to "hitflag", do not restrict RECIP actions.

REV B.

DIRECTORY

SEGMENT

(DIRC)

(32000A.00.Q2

6/18/73

D. R. McCLELLAN

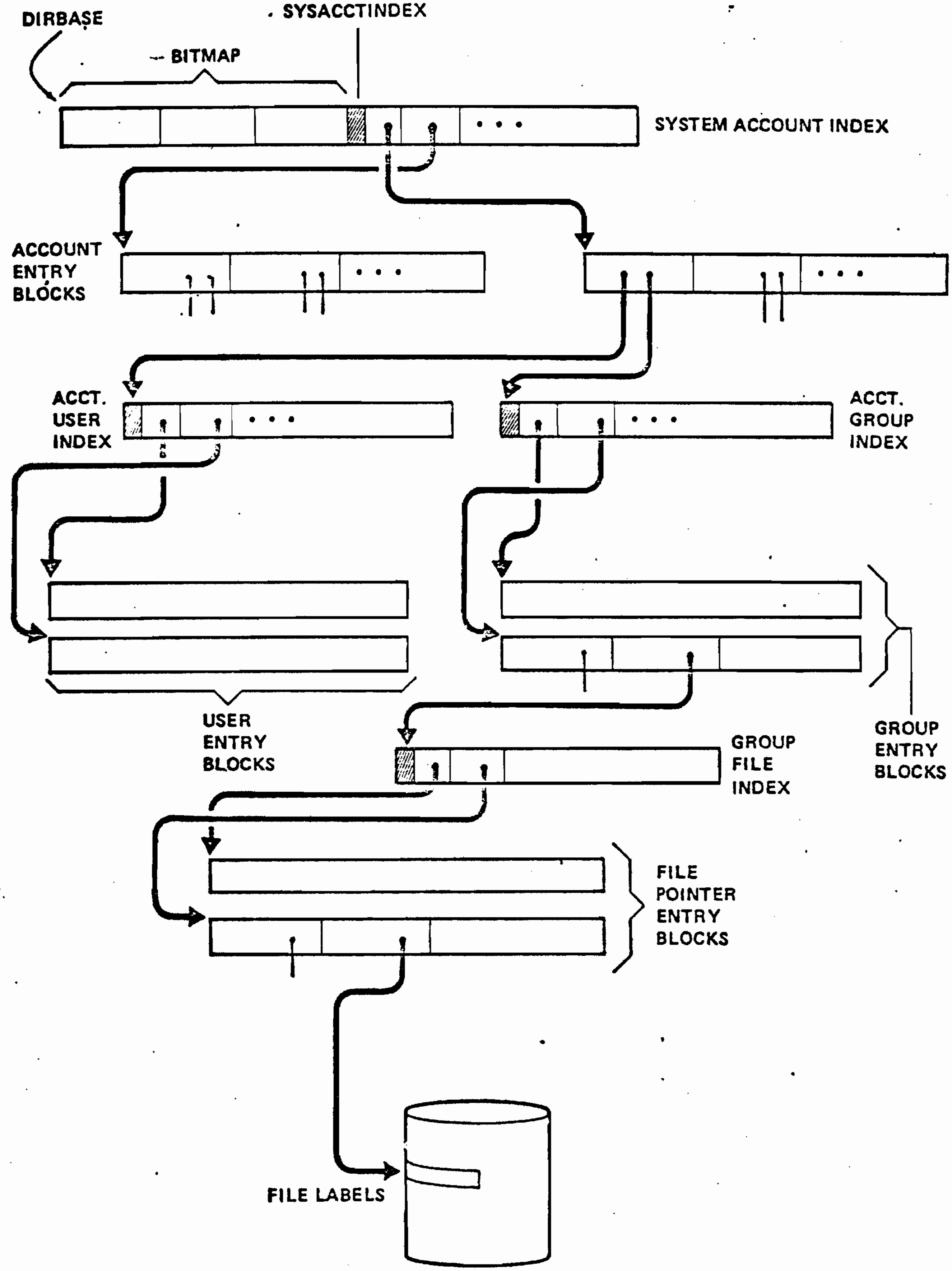
: 2

FLOWCHART PAGE

30,31
17B
22,23
21
22
22
3
3
26
20
21B
33
18,19
32
28,29
28,29
7
24
24
27
15
11,12,13,14
6
23
5
15

VREC,

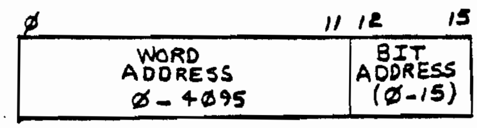
EXT.	PROCEDURE	TYPE	FLOWCHART PAGE
	DIRRESET	PROC, INTERNAL	34
	DIRSCAN	PROC, INTERNAL	8
	DIRSCANTREE	PROC, INTERNAL	25
	DIRSET	PROC, INTERNAL	1
	DIRSTARTOFF	PROC, INTERNAL	16,17
	DIRXXXLOCATE	PROC, INTERNAL	1
	DWRITE	PROC, INTERNAL	4
	DWRITEBITMAP	PROC, INTERNAL	1
	FIND	SUBR (OF DIRALLOCATE)	2
	VISIT	SUBR (OF DIRSTARTOFF)	17B
	ZDISTRIBUTE	SUBR (OF DIRINSERT)	10
	ZINSERT	SUBR (OF DIRINSERT)	9
	ZNEWENTRYBLOCK	SUBR (OF DIRINSERT)	9
	ZSET	SUBR (OF DIRINSERT)	10


DIRBASE
SYSACCTINDEX
BITMAP
• • •
SYSTEM ACCOUNT INDEX
ACCOUNT ENTRY BLOCKS
ACCT. USER INDEX
ACCT. GROUP INDEX
USER ENTRY BLOCKS
GROUP FILE INDEX
GROUP ENTRY BLOCKS
FILE POINTER ENTRY BLOCKS
FILE LABELS

LEGEND

DIRC (MPE 1.2)
USEFUL INFORMATION

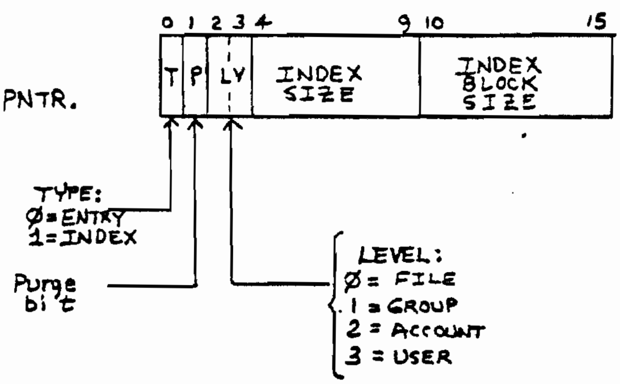
DIRECTORY BITMAP ADDRESSING:



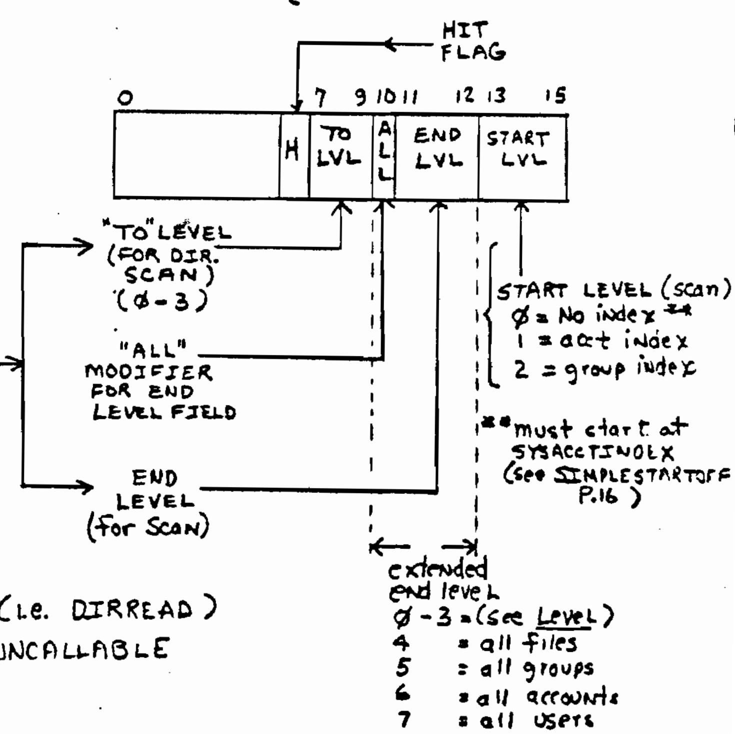
* DIRECTORY SPACE DATA SEGMENT DST #: %25 (21)
* DIRECTORY DATA SEGMENT (DDS). DST #: %24 (2Ø)
DIRECTORY SIR (DIRSIR) : %1Ø (8)

Same 30

MISCWD FORMAT
(WORD %12 OF DA/DB DIRECTORY PNTR. IN DDS)



TYPE PARAMETER (See P.16) FORMAT



PROCEDURES:

STARTING WITH 'DIR' - INTERNAL (i.e. DIRREAD)
STARTING WITH 'DIREC' - EXTERNAL, UNCALLABLE
ALL OTHERS - INTERNAL

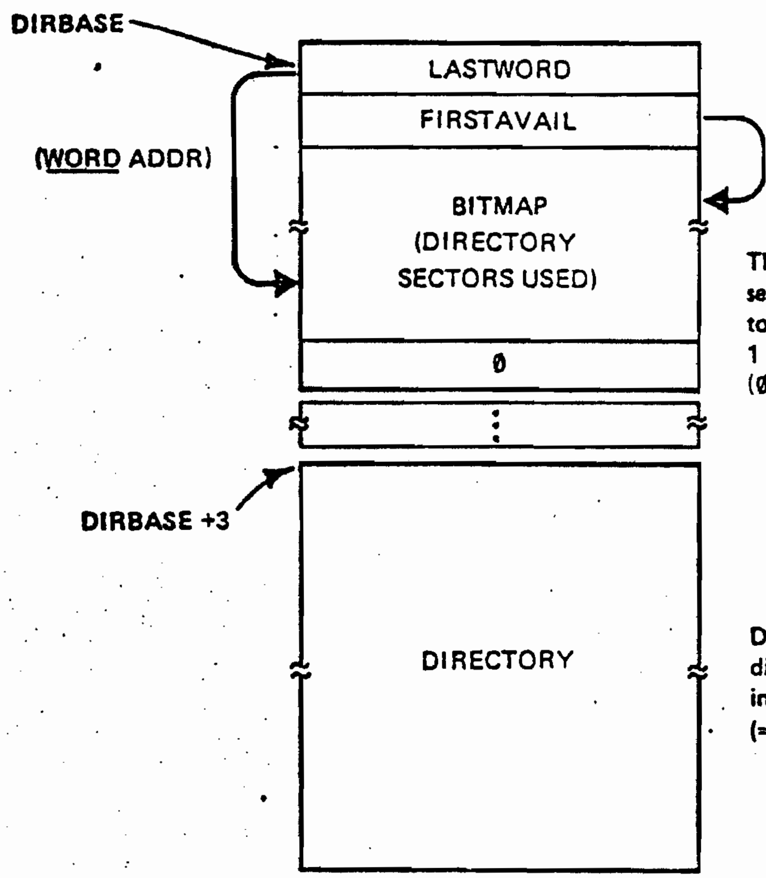
* SEE HANDOUT FOR LAYOUT

DIRECTORY ON DISC CONSISTS OF A CONTIGUOUS AREA

SYSGLOB cells:

DIRIOADDR ⟵ [DISKLDEV: SYSGLOB + 62_{10}]

DIRBASE ⟵ absolute disk addr of base [SYSGLOB + 44_{10}]



POINTS TO NEXT WORD TO BE EXAMINED

The bitmap defines the available/used sectors in the directory. bit Ø, word Ø corresponds to DIRBASE; bit 1 to DIRBASE +1; etc. 1 ⇒ available; Ø ⇒ used. [Note: BITMAP (Ø).(Ø:4) = Ø always.]

Directory entries contain pointers which are sector displacements relative to DIRBASE. Entries and indices are grouped into "blocks" (= integral # sectors).

The capacities for accounts/groups/users/files are dependent on their block sizes, described in the directory data segment.

• SYSSAIBSIZE	System acct index block size (sectors)
SYSAUIBSIZE	Acct. user index block size (sectors)
SYSAGIBSIZE	Acct. group index block size (sectors)
SYSGFIBSIZE	Group file index block size (sectors)
• SYSAEBSIZE	Acct. entry block size (sectors)
SYSUEBSIZE	User entry block size (sectors)
SYSGEBSIZE	Group entry block size (sectors)
SYSFEBSIZE	File entry block size (sectors)
SYSMAXBSIZE	Maximum of above. (used to initialize DDS.)

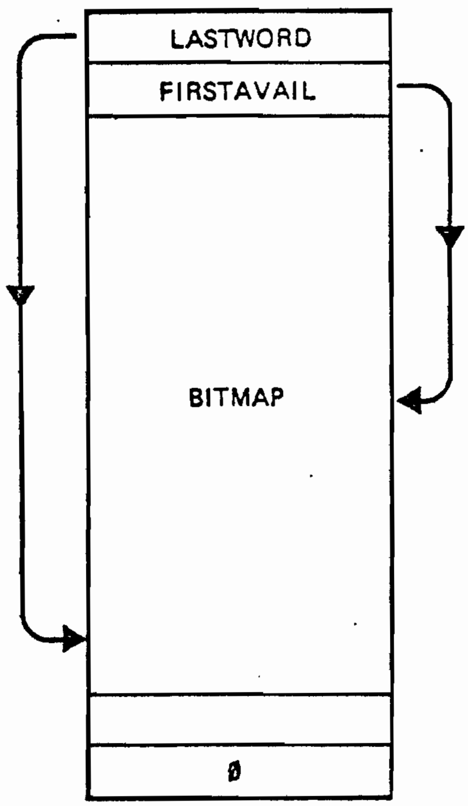
*These values are used once for the creation of the (root) system, account index or new systems. This root index is always at address DIRBASE +3.

DIRECTORY SPACE DATA SEGMENT (DSDS)

12-1-73

DST = 21_{10}

SIR = 8_{10}

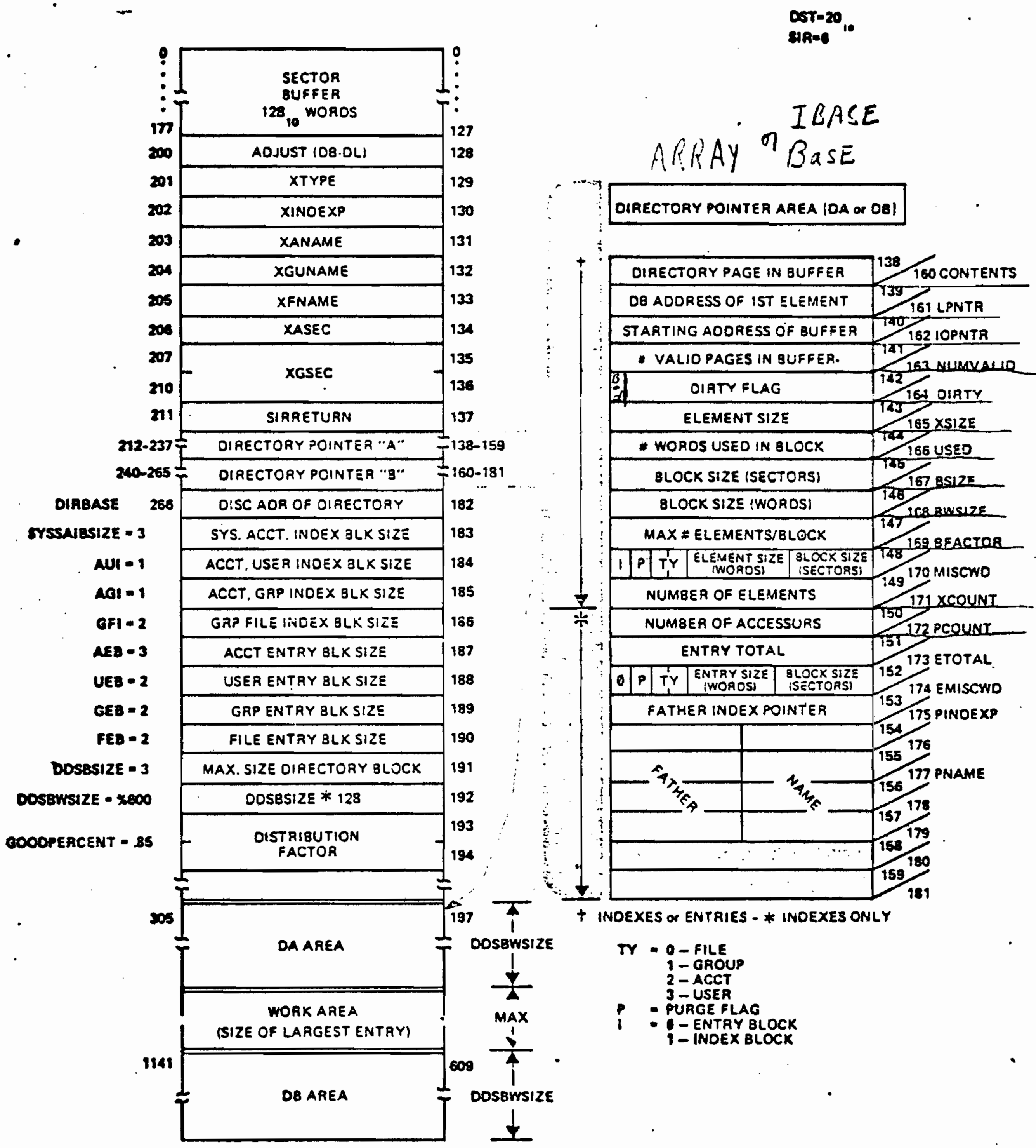


DEFINES LAST WORD OF BIT MAP

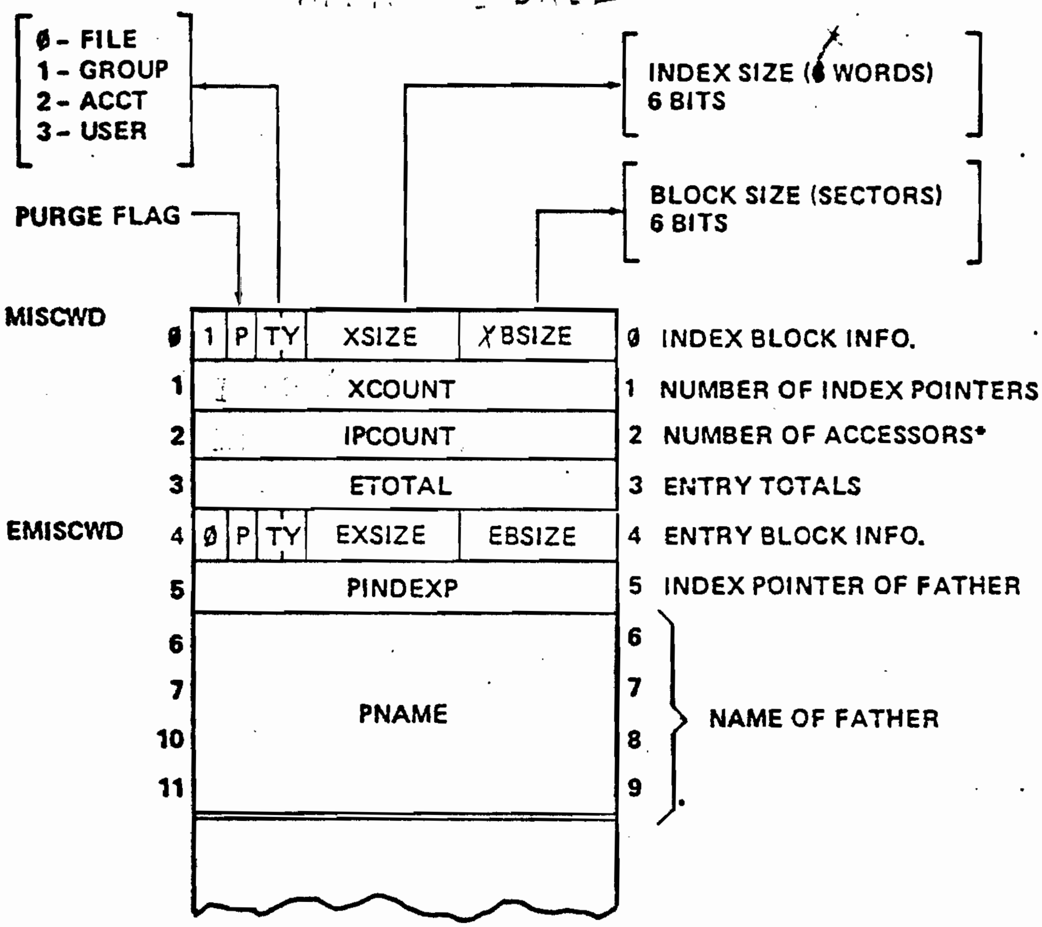
DEFINES NEXT WORD TO BE EXAMINED

THE BITMAP DEFINES THE AVAILABLE/USED SECTORS IN THE DIRECTORY. BIT Ø WORD Ø CORRESPONDS TO DIRBASE; BIT 1 WORD Ø TO DIRBASE + 1 ETC. 1 ⇒ AVAILABLE Ø ⇒ USED. [NOTE: BITMAP (Ø) · (Ø : 4) = Ø ALWAYS.]

DIRECTORY DATA SEGMENT



INDEX BLOCK PREFIX (10 WORDS)



* THE COUNT IS INCREMENTED BY EACH ACCESS THAT USES AND RELIES UPON A POINTER TO THE INDEX BLOCK, IE, IT IS GUARANTEED NOT TO BE PURGED WHILE THE COUNT IS ≠ Ø

DIRECTORY DEFINITIONS

- **PAGE** – SMALLEST ALLOCATABLE RECORD ("PHYS. RECD") – CURRENTLY 1 SECTOR.
- **BLOCK** – INTEGRAL # OF PP. CONTAINS CONTIGUOUS INDICES OR ENTRIES.
- **INDEX** – POINTER TO ENTRY BLOCK, CONTAINING NAM OF 1st ENTRY.
- **ENTRY** – INFORMATION – CONTAINING "OBJECT" MAY CONTAIN POINTER TO AN INDEX BLOCK.
- **POINTER** – 15-BIT POSITIVE RELATIVE PAGE NUMBER (RELATIVE TO DIRECTORY BASE).
- **DDS** – DIRECTORY DATA SEGMENT.
- **ELEMENT** – A GENERIC NAME FOR INDEX OR ENTRY.

INDEX ENTRY (6 WORDS)

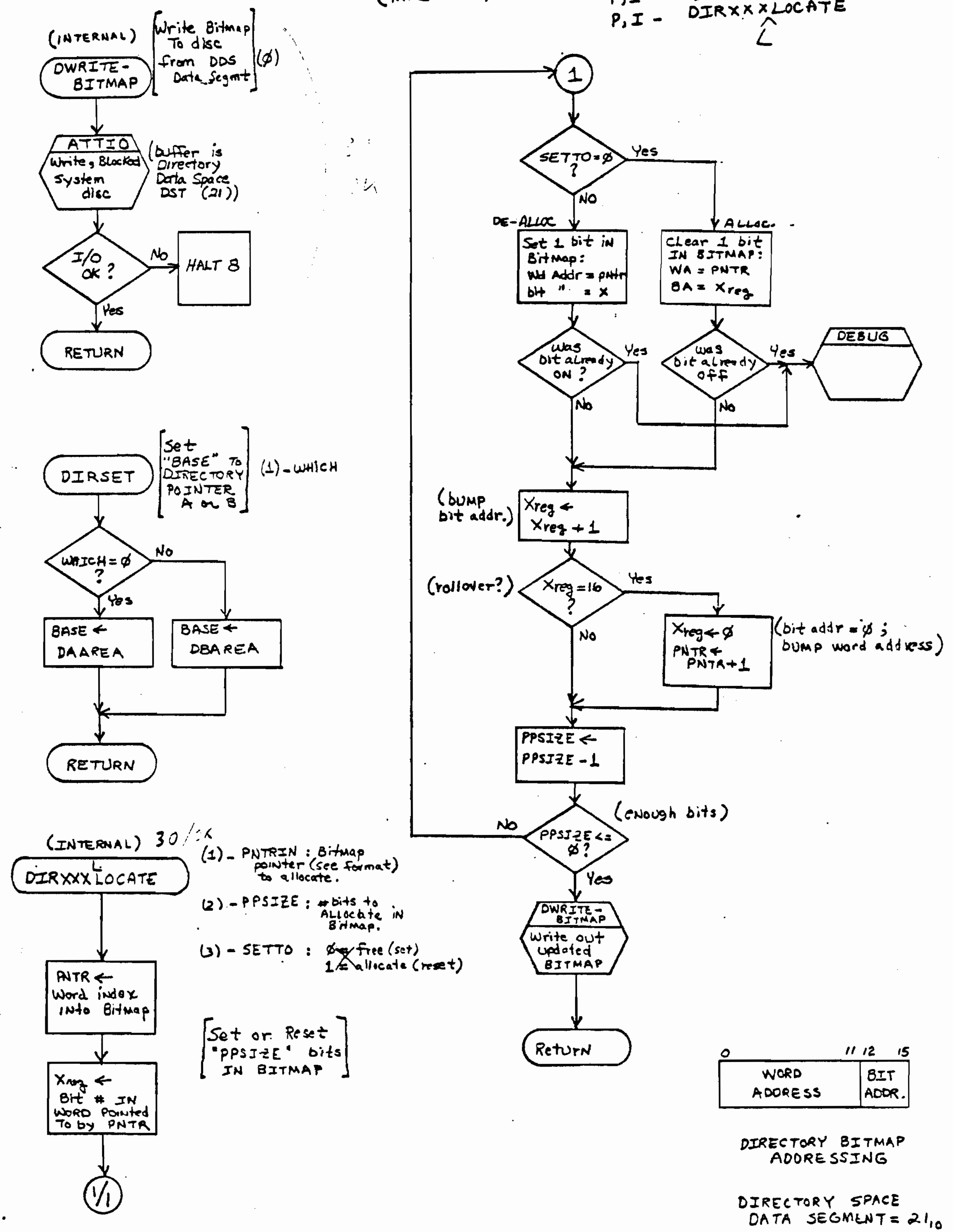
IIPNTR - IPNTR

Field	Word	Description
IE1STNAME	0	1st NAME OF ENTRY BLOCK
	1	
	2	
	3	
IEPNTR	4	POINTER TO ENTRY BLOCK
IECOUNT	5	NUMBER OF ENTRIES IN e BLOCK
1S1 LE		

DIRC
(MPE 1.2)

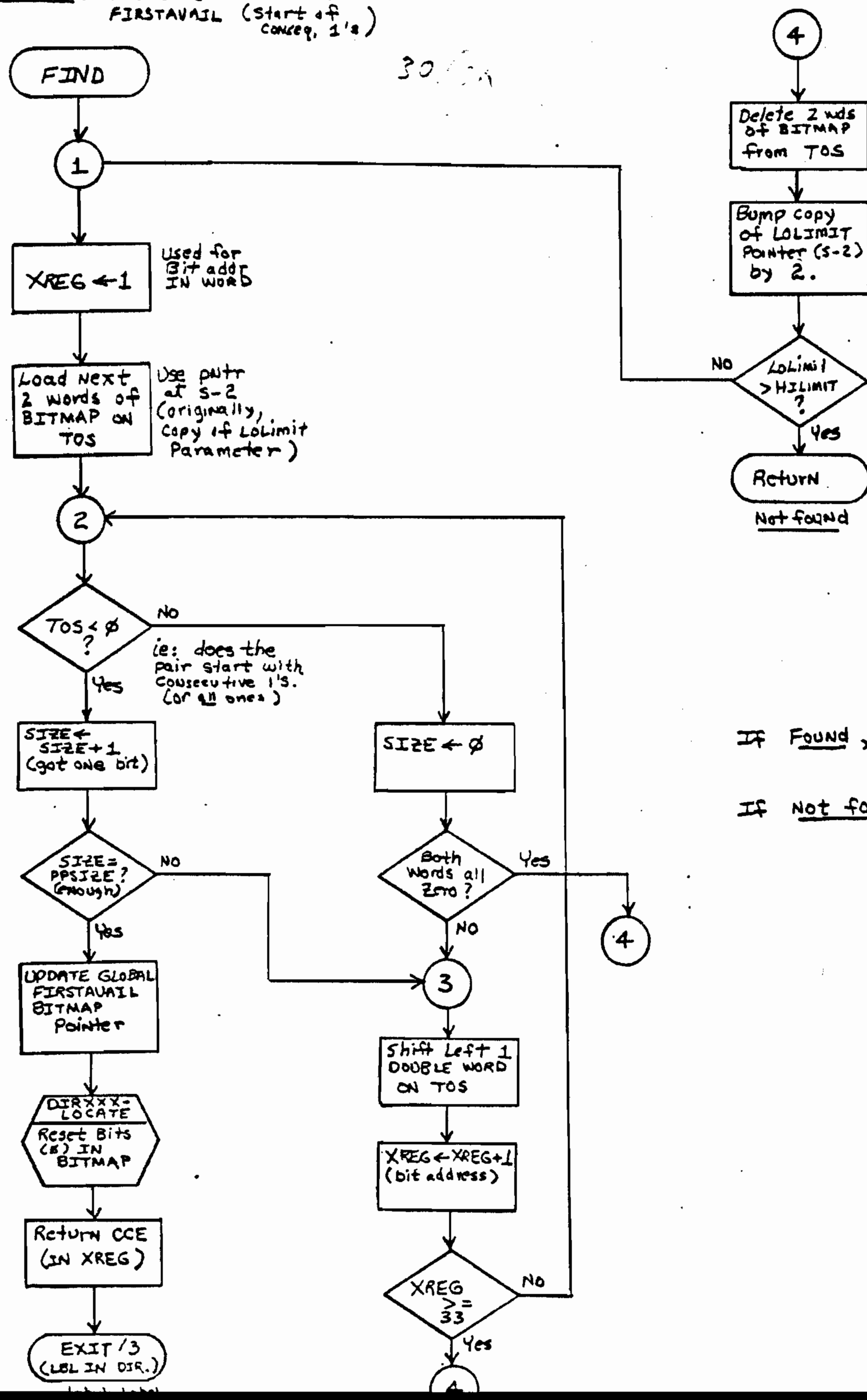
P,I - DWRITEBITMAP
P,I - DIRSET
P,I - DIRXXXLOCATE

1



SUBROUTINE FIND(LOLIM, HILIM)
CALLED BY DIRALLOCATE
FINDS CONTIGUOUS 1'S
IN BITMAP.

GLOBALS: PPSIZE (# bits Needed)
FIRSTAVAIL (Start of Conseq. 1's)



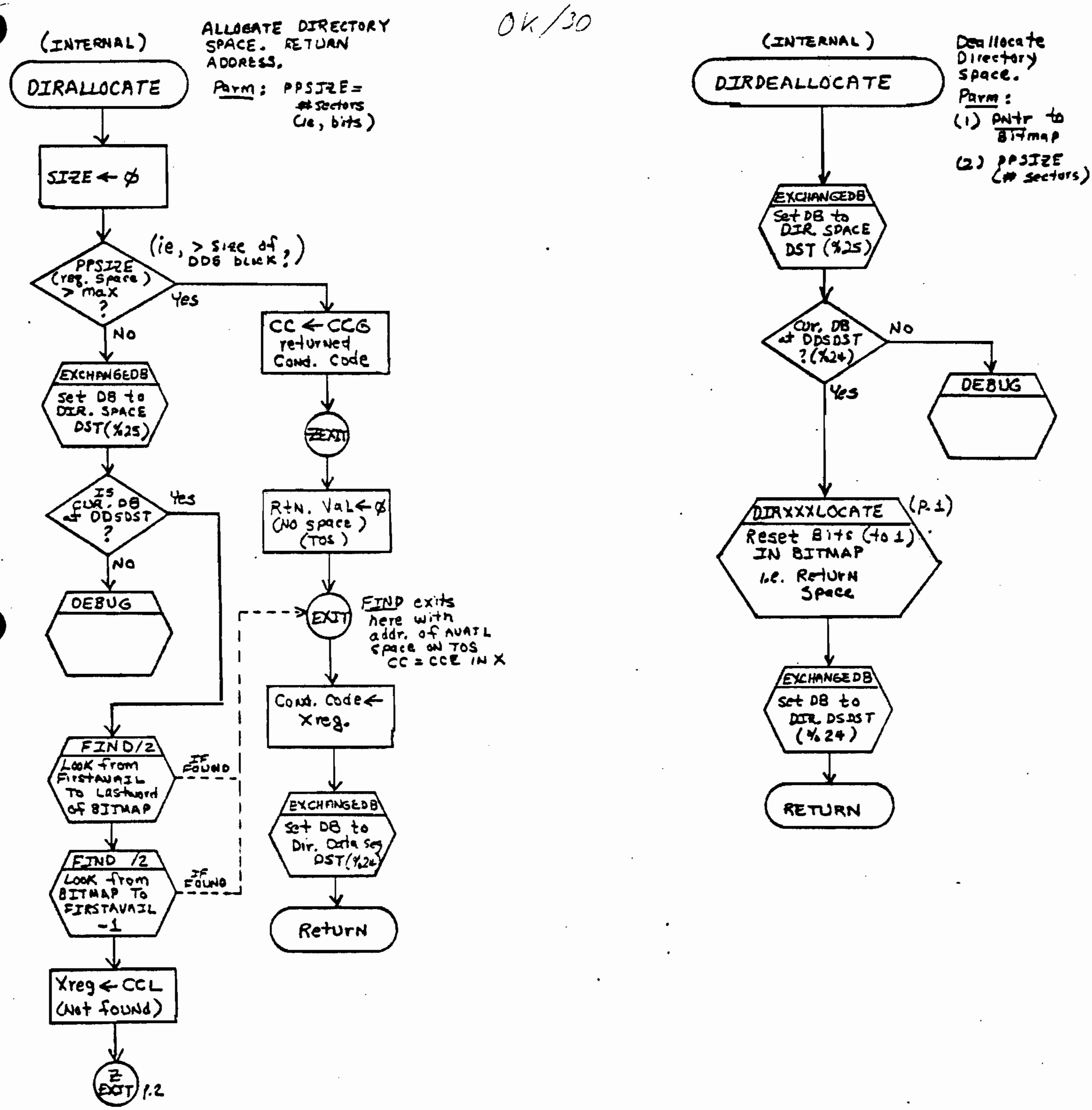
IF Found, exits to Label EXIT IN DIRALLOCATE. (X=CCE)

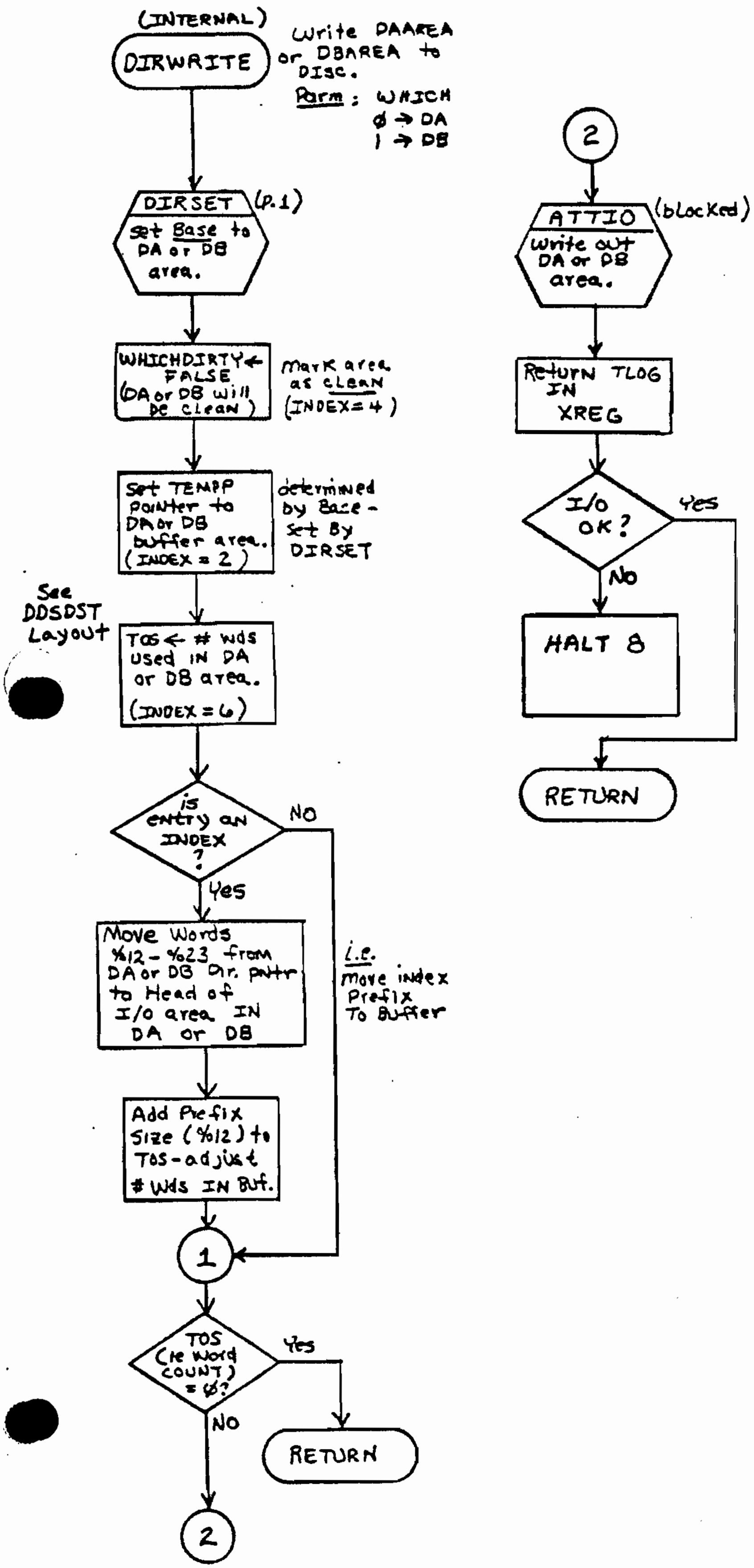
IF Not found, Returns IN Line

DIRC
(MPE 1.2)

P.I - DIRALLOCATE
P.I - DIRDEALLOCATE

OK/30

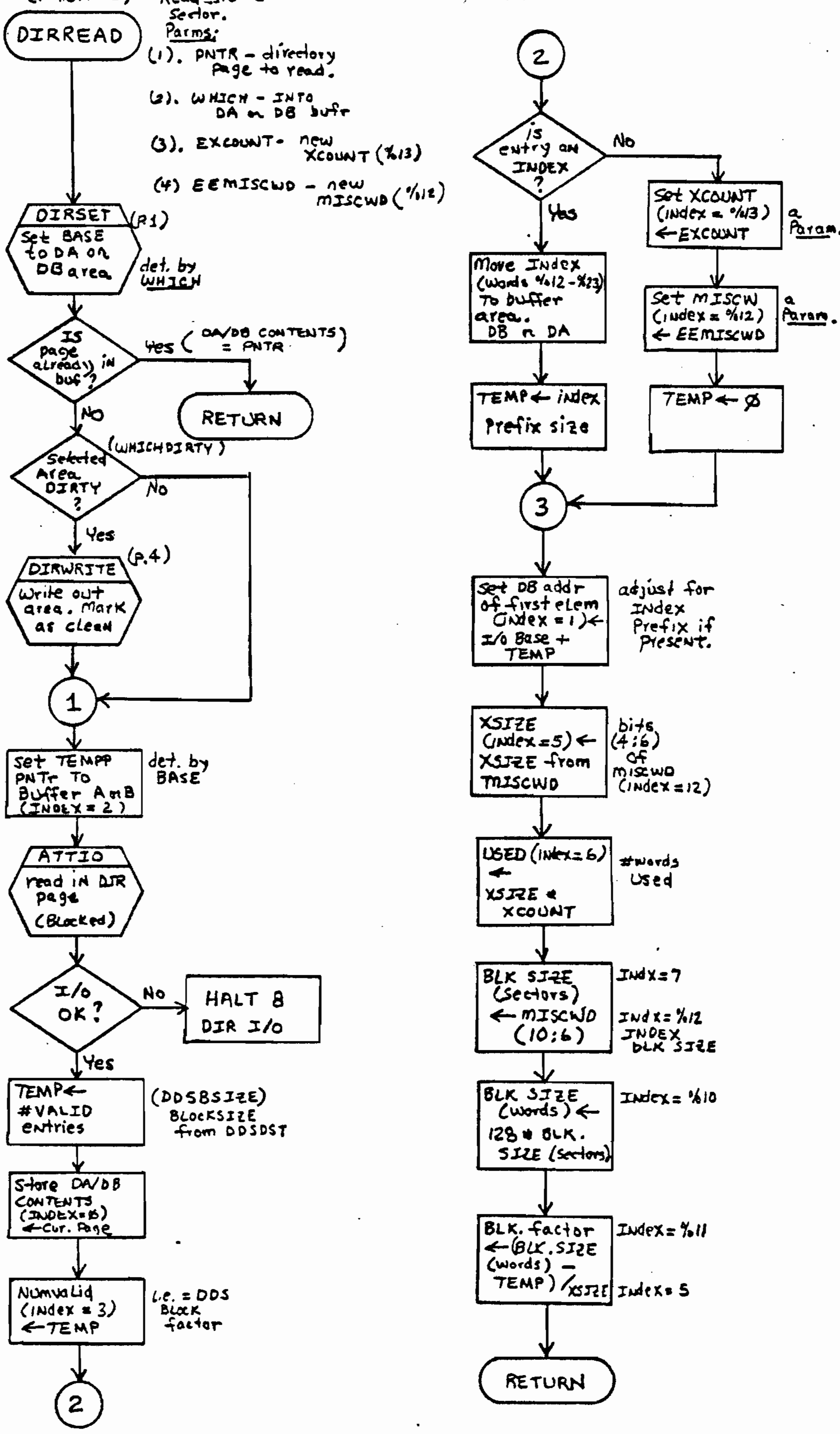



(INTERNAL)
DIRWRITE
Write DAAREA or DBAREA to DISC.
Parm: WHICH
ø → DA
1 → DB
DIRSET (p.1)
set Base to DA or DB area.
WHICHDIRTY ← FALSE (DA or DB will be clean)
Mark area as clean (INDEX = 4)
Set TEMPP pointer to DA or DB buffer area. (INDEX = 2)
determined by Base - set by DIRSET
See DDSDST Layout
TOS ← # wds used in DA or DB area. (INDEX = 6)
is entry an INDEX?
NO
Yes
Move Words %12 - %23 from DA or DB Dir. pntr to Head of I/O area in DA or DB
i.e. move index Prefix To Buffer
Add Prefix Size (%12) to TOS - adjust # Wds in Buf.
1
TOS (ie word count) = ø?
Yes
RETURN
NO
2
2
ATTIO (blocked)
write out DA or DB area.
Return TLOG in XREG
I/O OK?
Yes
No
HALT 8
RETURN

DIRC
(MPE 1.2)

P, I - DIRREAD

30/0K

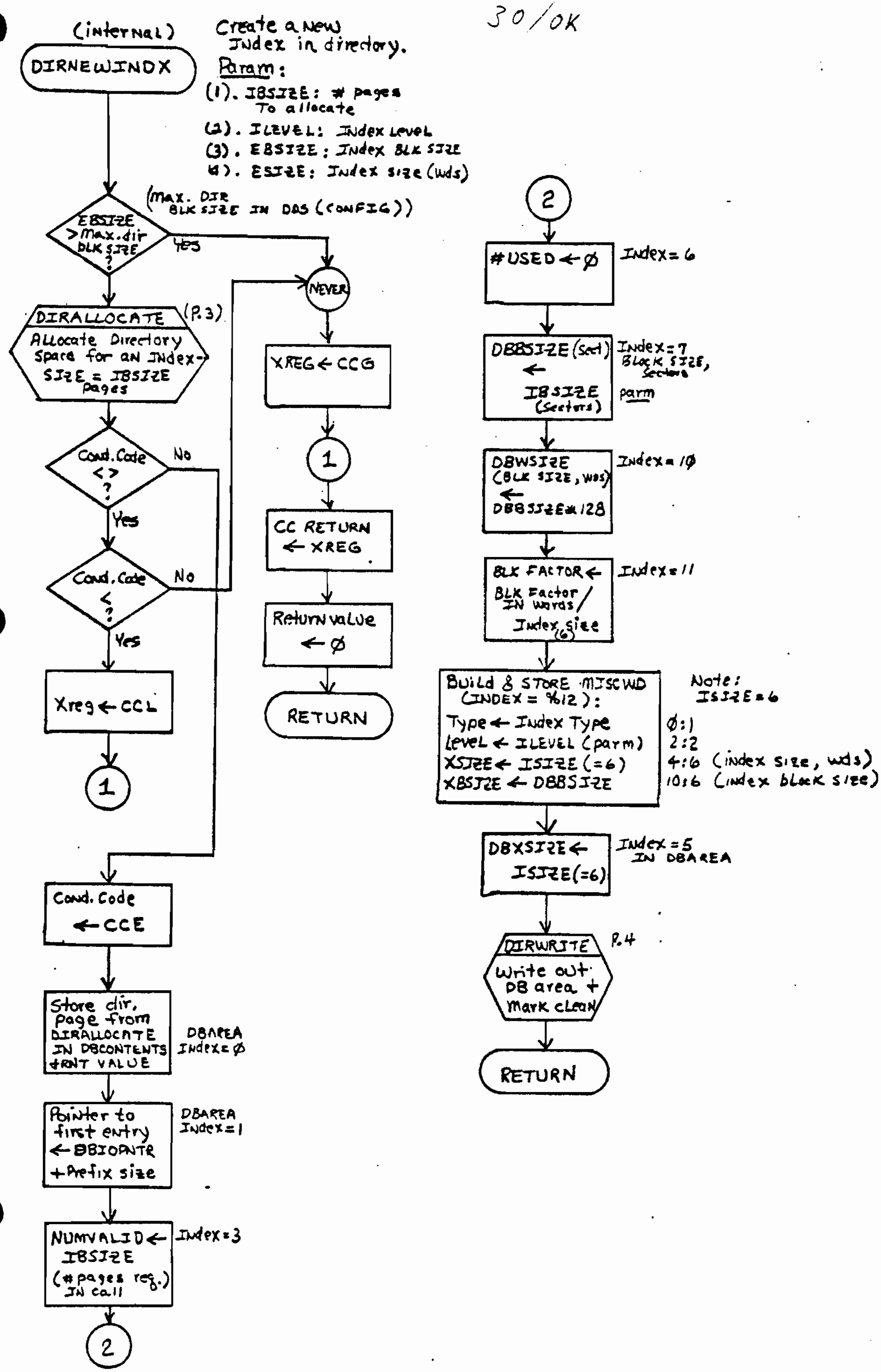


Caller must move father index pointer (PINDEXP - index 17) & Program Name into PNAME (index 20-23).

DIRC
(MPE 1.2)

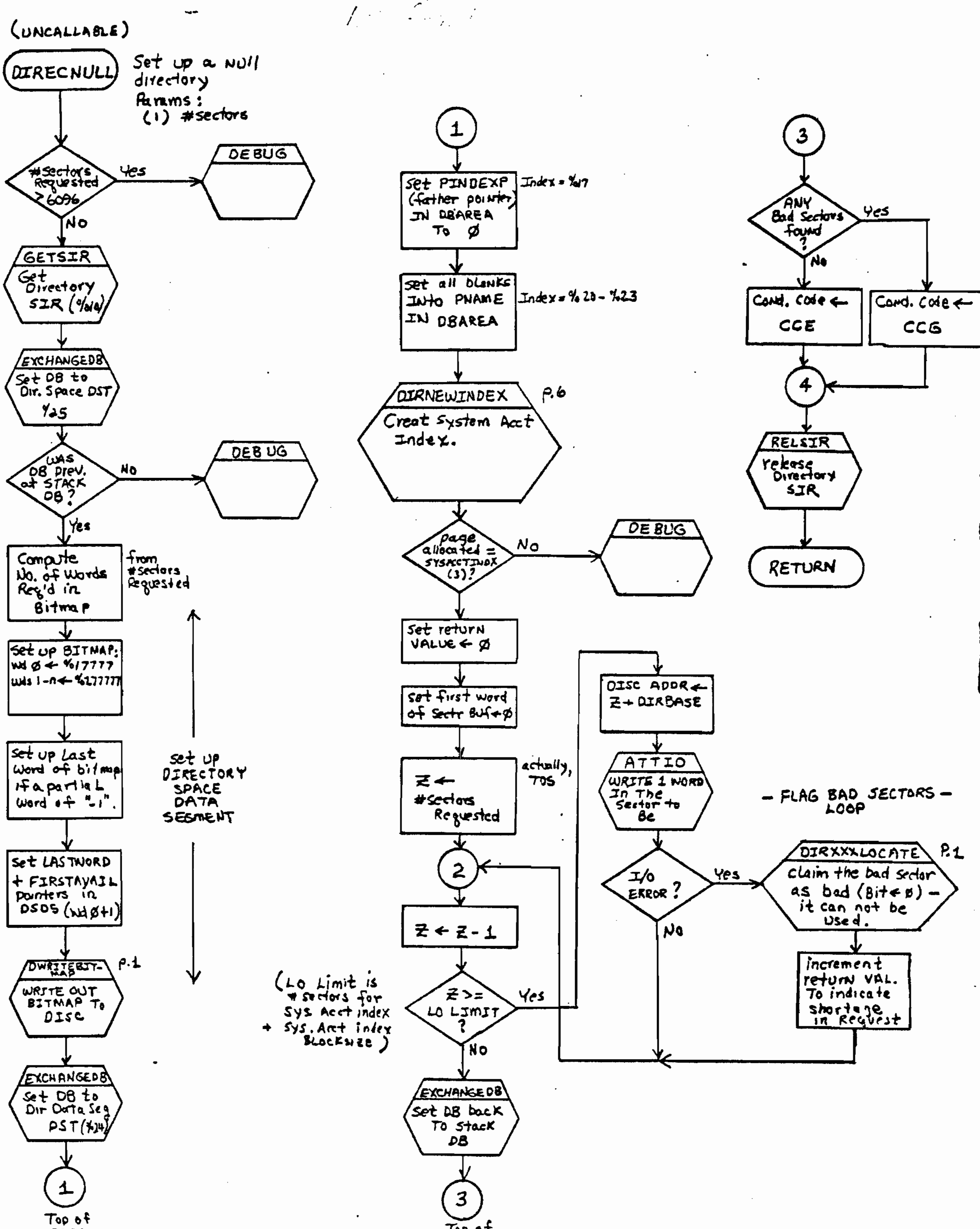
P, I - DIRNEWINDX

30/OK



DIRC
(MPE 1.2)

P, U – DIRECNULL

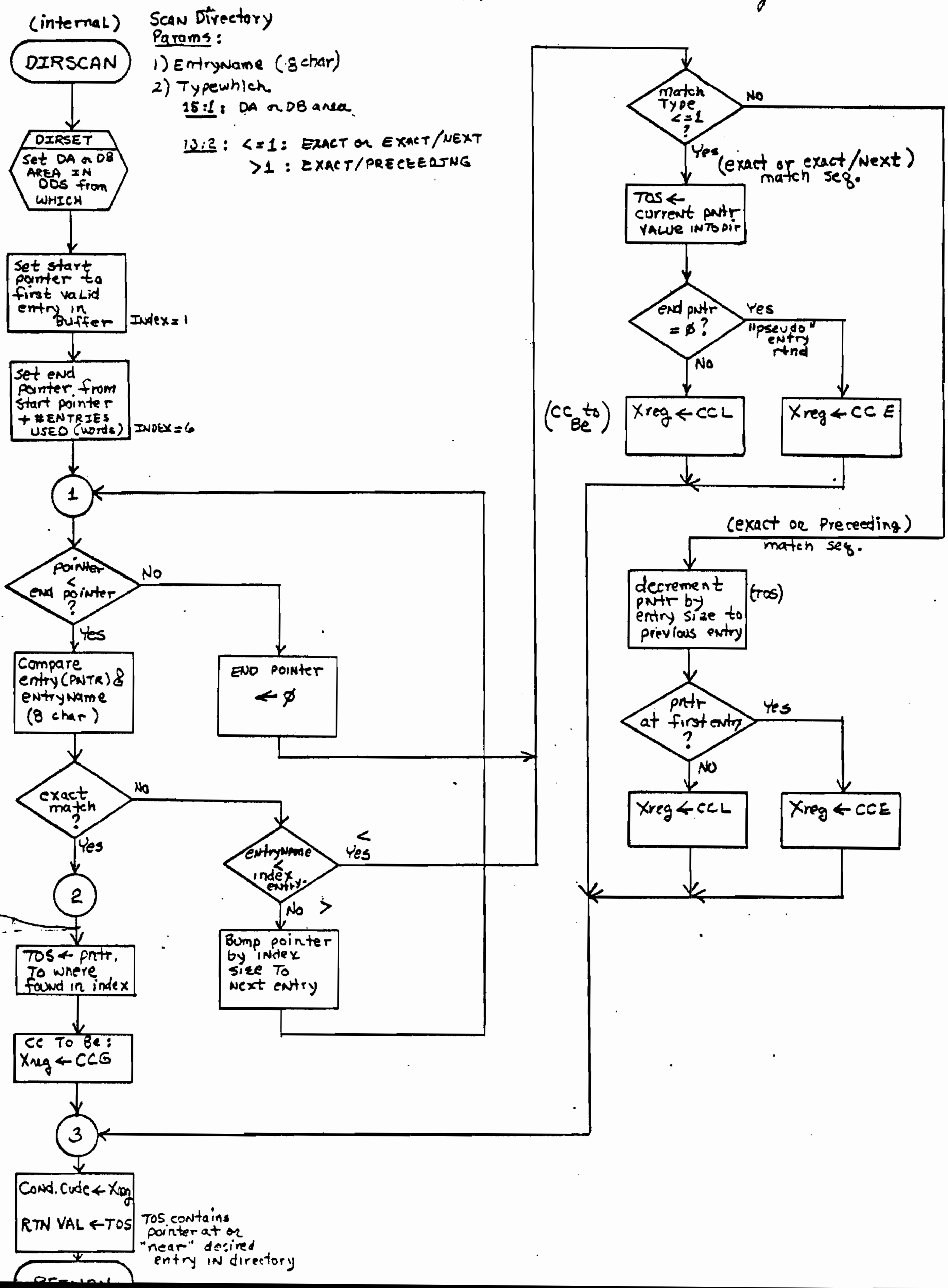


DIRC
(MPE 1.2)

P, I – DIRSCAN

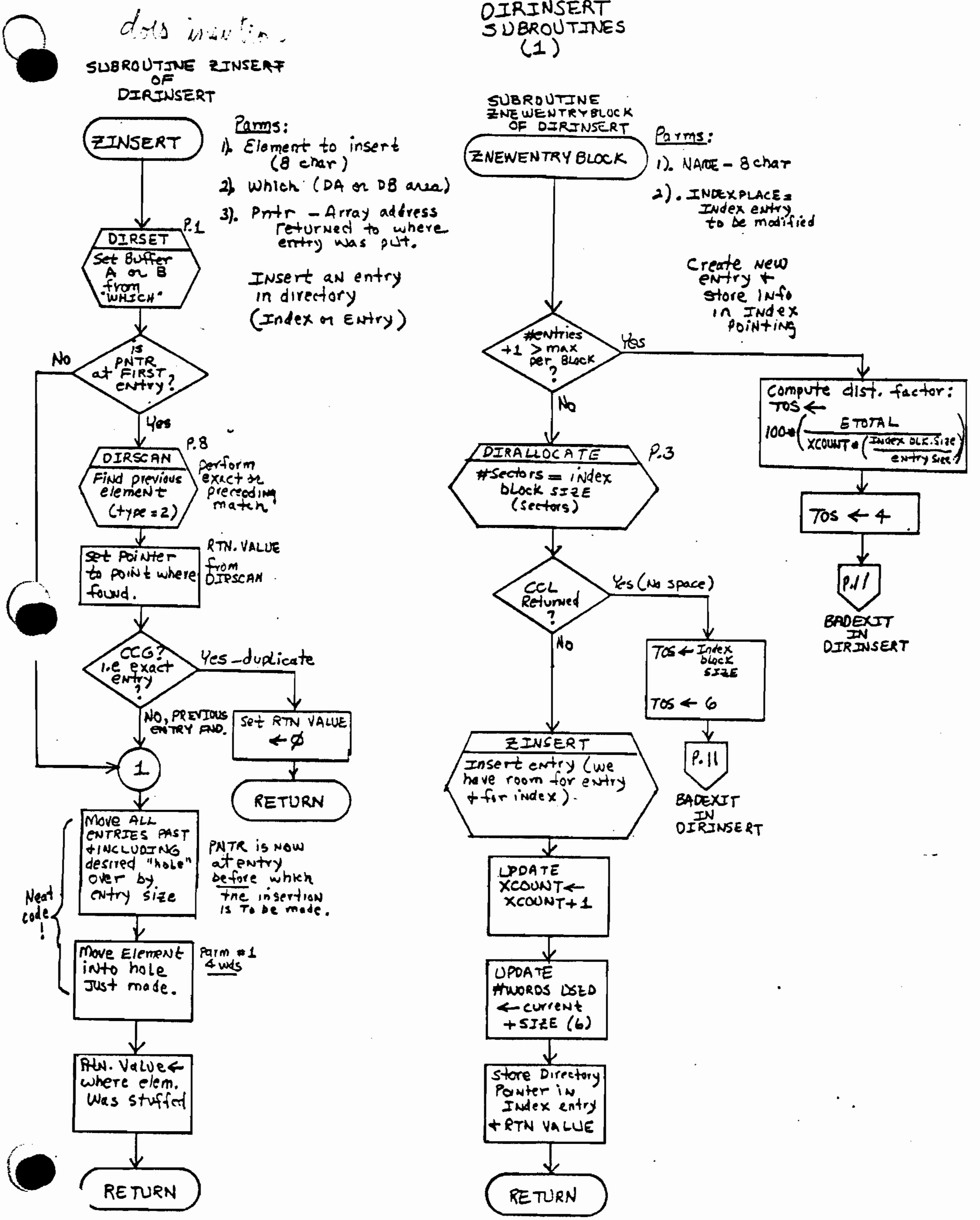
30/0x

... entry ... index



DIRC
(MPE 1.2)
DIRINSERT
SUBROUTINES
(1)

S - ZINSERT
S - ZNEWENTRYBLOCK

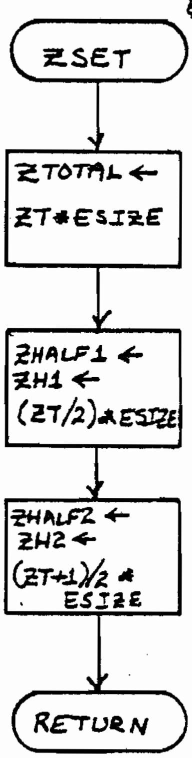


DIRC (MPE 1.2)

DIRINSERT SUBROUTINES (2)

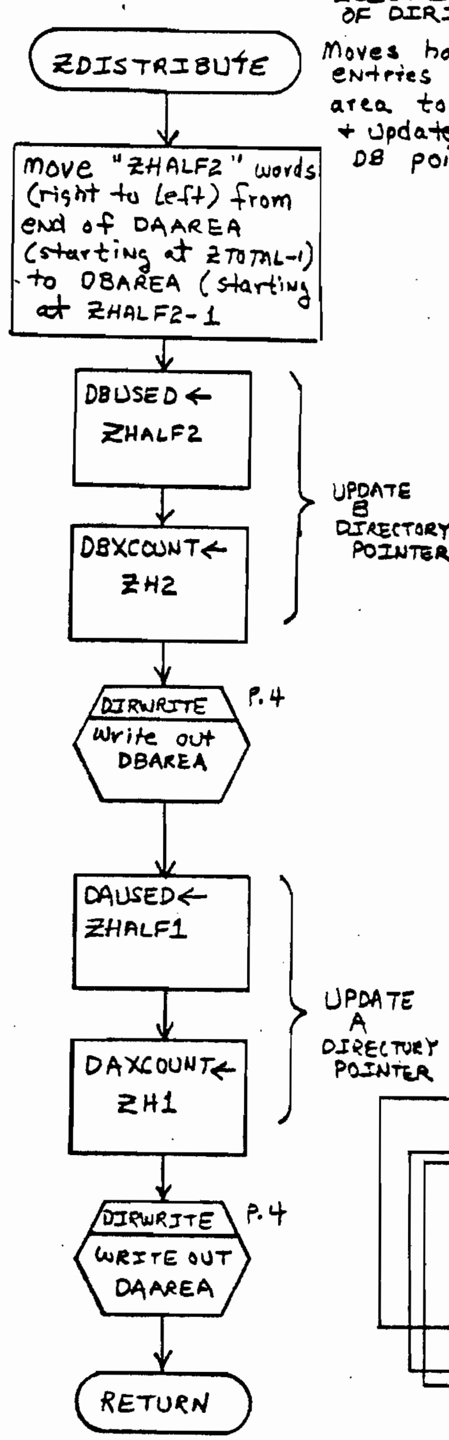
SUBROUTINE ZSET — OF DIRINSERT

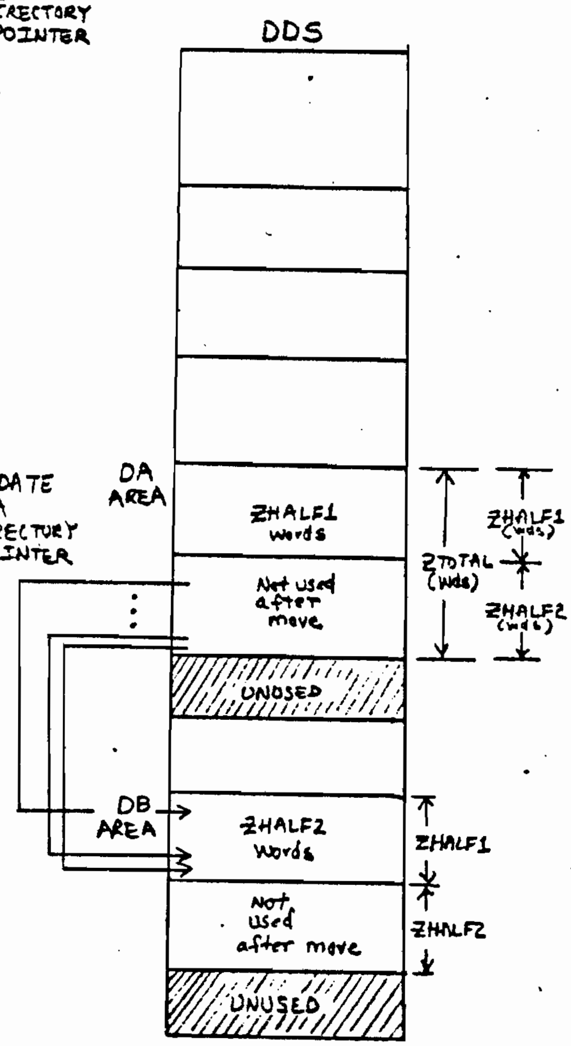
Sets globals ZTOTAL, ZHALF1, ZHALF2, ZH1 + ZH2 from ESIZE & ZT



SUBROUTINE ZDISTRIBUTE OF DIRINSERT

Moves half of entries in DA area to DB area + updates DA + DB pointers.



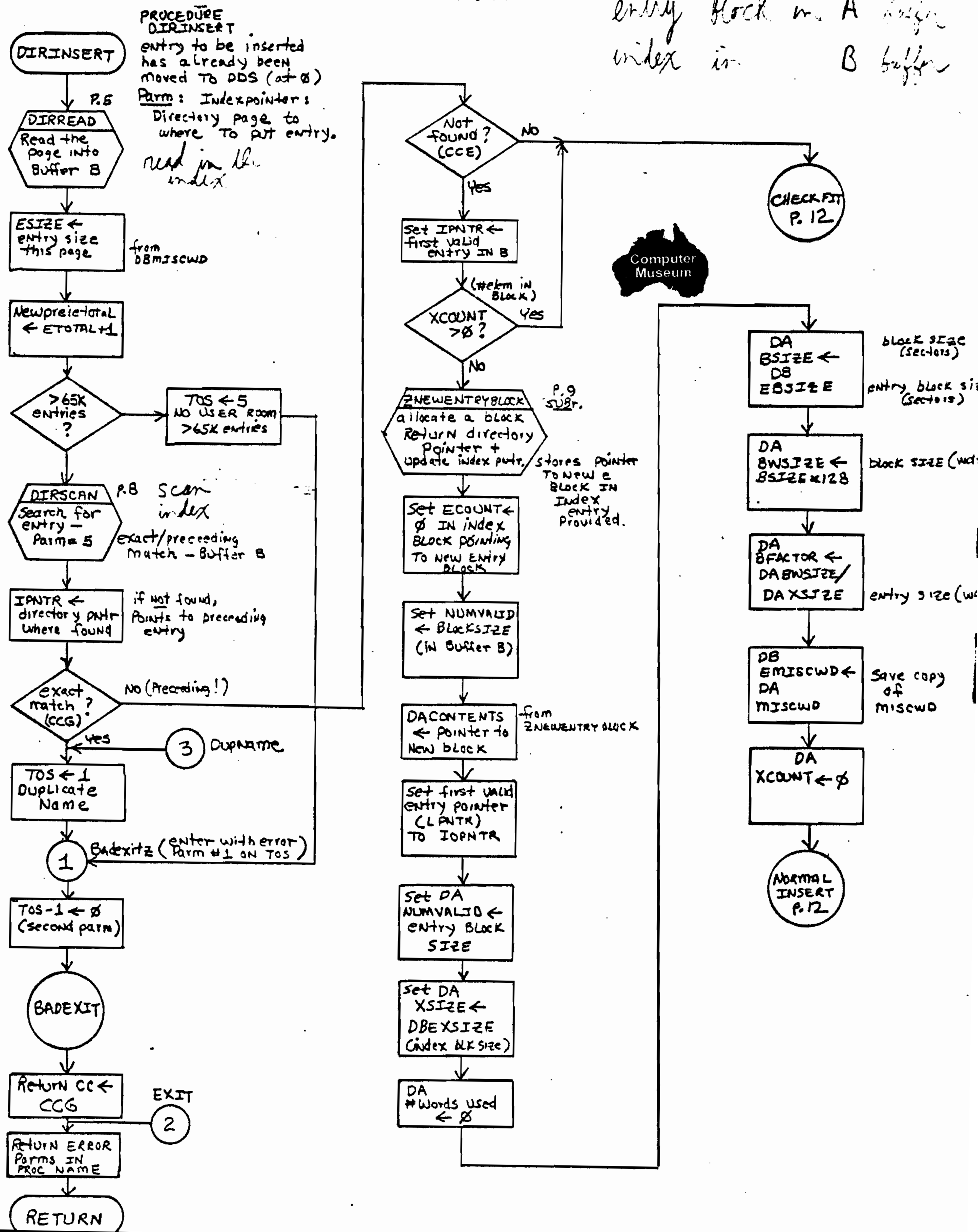


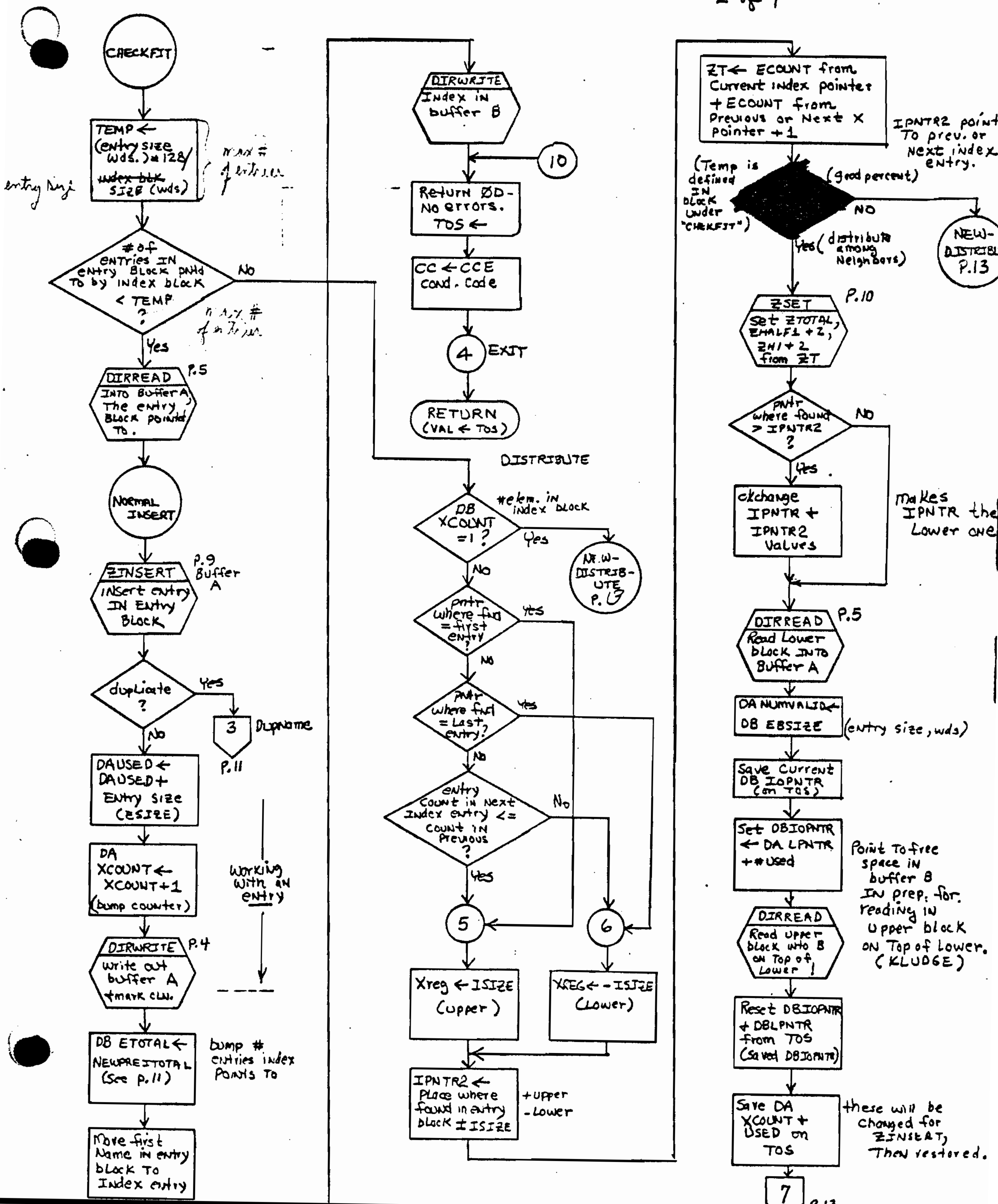
DIRC
(MPE 1.2)

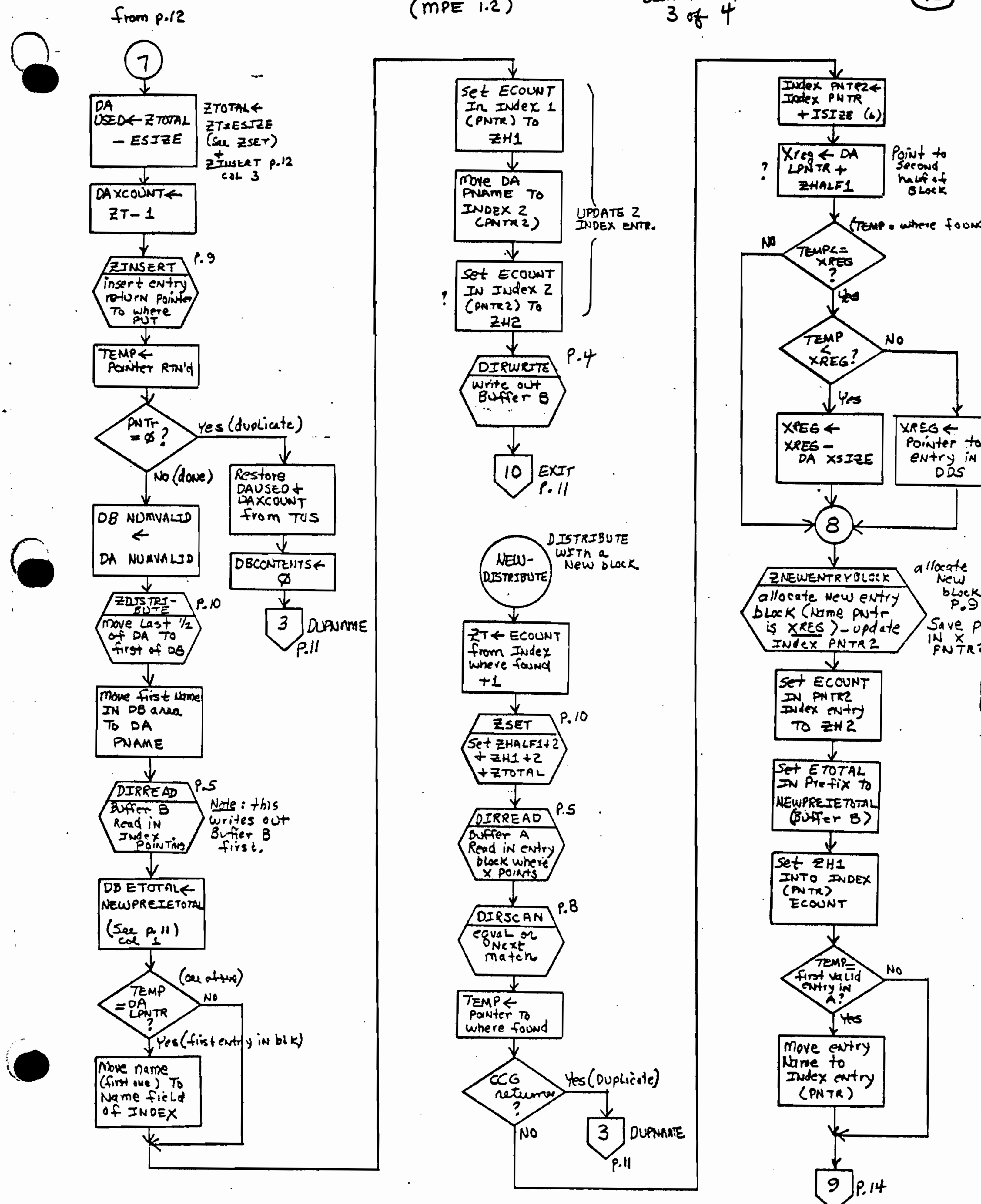
30/3K

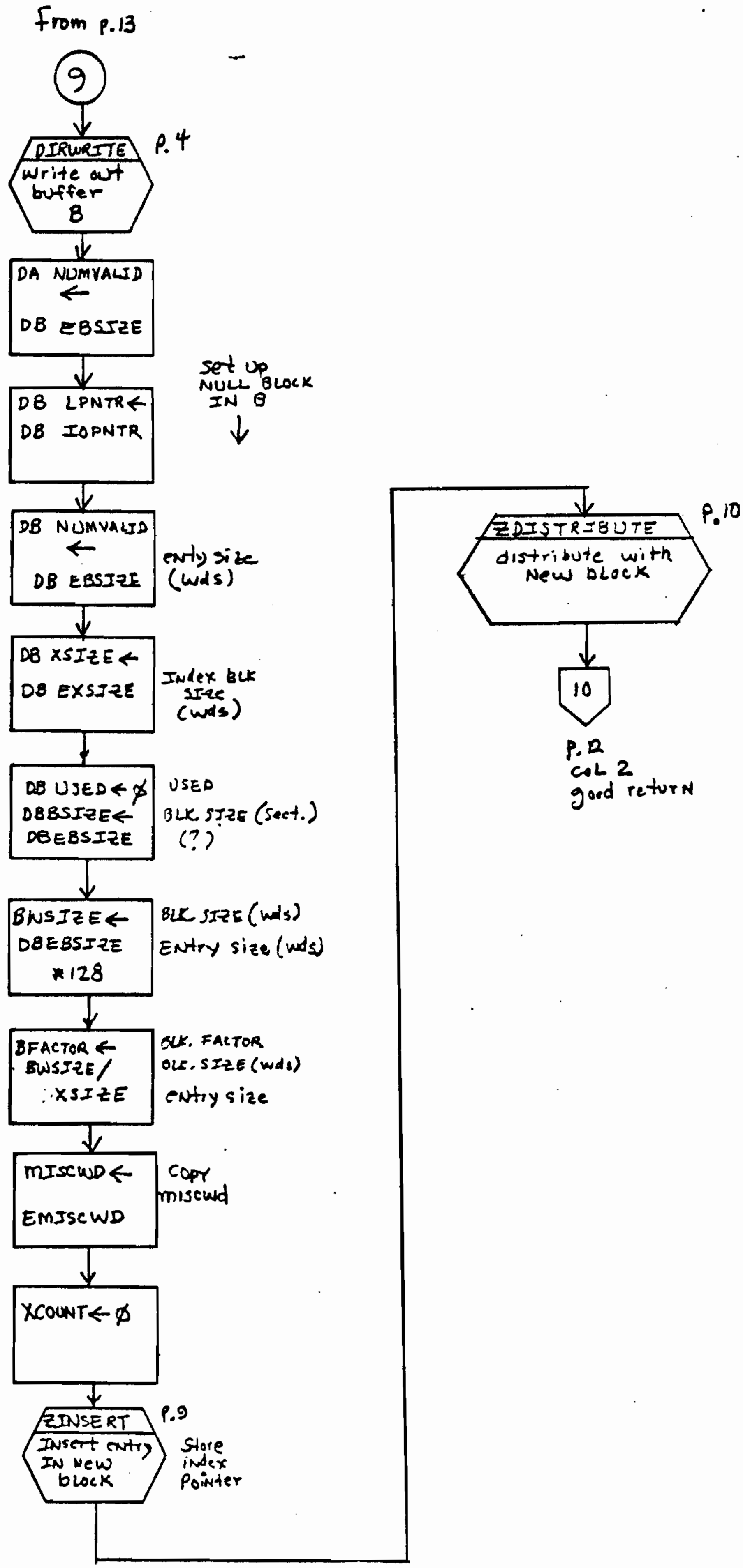
P,I DIRINSERT
1 of 4

entry block in A buffer
index in B buffer




CHECKFIT
TEMP ← (entry size wds.) * 128 / ~~index blk~~ SIZE (wds)
entry size
max # of entries
# of entries in entry block pntd to by index block < TEMP ?
No
max # of entries
Yes
DIRREAD
P.5
INTO Buffer A The entry block pointed to.
NORMAL INSERT
ZINSERT
P.9
Buffer A
INSERT entry IN Entry Block
duplicate ?
Yes
3
Dupname
P.11
No
DAUSED ← DAUSED + Entry size (ESIZE)
DA XCOUNT ← XCOUNT + 1 (bump counter)
Working with an entry
DIRWRITE
P.4
write out buffer A + mark CLN.
DB ETOTAL ← NEWPREITOTAL (See p.11)
bump # entries index Points To
Move first Name in entry block To Index entry
DIRWRITE
Index in buffer B
10
Return ØD - No errors. TOS ←
CC ← CCE cond. code
4
EXIT
RETURN (VAL ← TOS)
DISTRIBUTE
DB XCOUNT = 1 ?
#elem. in index block
Yes
NEW-DISTRIBUTE P.13
No
pntr where fnd = first entry ?
Yes
No
pntr where fnd = Last entry?
Yes
No
entry Count in Next Index entry <= count in Previous ?
No
Yes
5
6
Xreg ← ISIZE (upper)
XREG ← - ISIZE (Lower)
IPNTR2 ← Place where found in entry block ± ISIZE
+ upper
- Lower
ZT ← ECOUNT from Current index pointer + ECOUNT from Previous or Next X Pointer + 1
IPNTR2 points To prev. or Next index entry.
(Temp is defined IN block under "CHECKFIT")
(good percent)
NO
NEW-DISTRIBUTE P.13
Yes (distribute among Neighbors)
ZSET
P.10
Set ZTOTAL, ZHALF1 + 2, ZH1 + 2 from ZT
pntr where found > IPNTR2 ?
NO
Yes
exchange IPNTR + IPNTR2 Values
Makes IPNTR the Lower one
DIRREAD
P.5
Read Lower block INTO Buffer A
DA NUMVALID ← DB EBSIZE
(entry size, wds)
Save Current DB IOPNTR (on TOS)
Set DBIOPNTR ← DA LPNTR + # Used
Point To free space in buffer B IN prep. for reading in upper block ON Top of Lower. (KLUDGE)
DIRREAD
Read upper block into B on Top of Lower!
Reset DBIOPNTR + DBLPNTR from TOS (Saved DBIOPNTR)
Save DA XCOUNT + USED on TOS
these will be changed for ZINSEAT, Then restored.
7
P.13

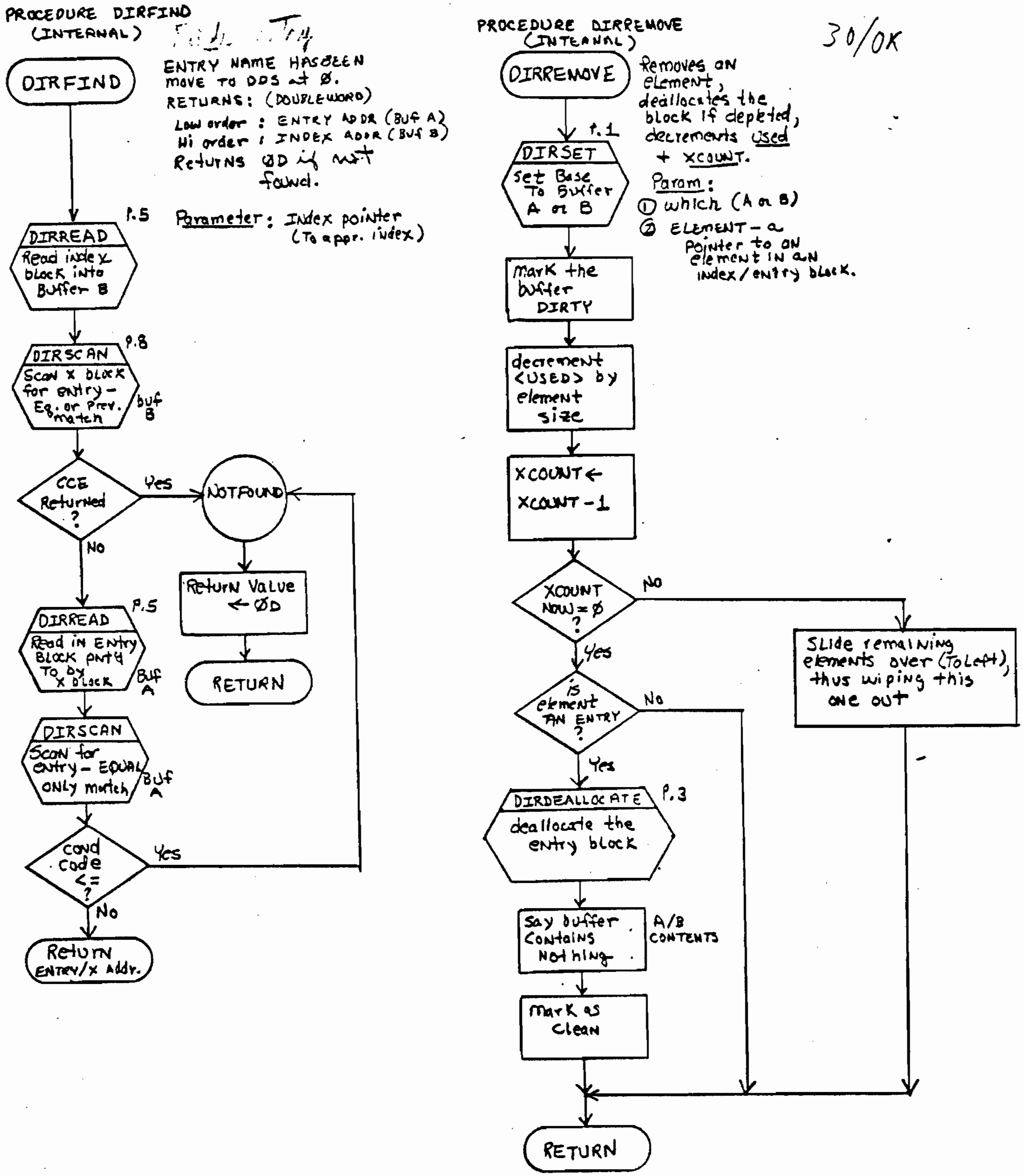

from p.12
7
DA USED ← ZTOTAL − ESIZE
ZTOTAL ← ZT*ESIZE (See ZSET) + ZINSERT p.12 col 3
DA XCOUNT ← ZT − 1
ZINSERT
insert entry return pointer to where put
P.9
TEMP ← Pointer RTN'd
PNTr = Ø ?
Yes (duplicate)
No (done)
Restore DAUSED + DAXCOUNT from TOS
DBCONTENTS ← Ø
3
DUPNAME
P.11
DB NUMVALID ← DA NUMVALID
ZDISTRIBUTE
move last ½ of DA to first of DB
P.10
Move first name in DB area to DA PNAME
DIRREAD
Buffer B Read in Index Pointing
P.5
Note: this writes out Buffer B first.
DB ETOTAL ← NEWPREIETOTAL (See p.11 col 1)
TEMP = DA LPNTR ?
(see above)
NO
Yes (first entry in blk)
Move name (first one) to Name field of INDEX
Set ECOUNT In Index 1 (PNTR) To ZH1
Move DA PNAME To INDEX 2 (PNTR2)
Set ECOUNT IN Index 2 (PNTR2) To ZH2
UPDATE 2 INDEX ENTR.
DIRWRITE
Write out Buffer B
P.4
10
EXIT P.11
NEW-DISTRIBUTE
DISTRIBUTE WITH a New block
ZT ← ECOUNT from Index where found +1
ZSET
Set ZHALF1+2 + ZH1+2 + ZTOTAL
P.10
DIRREAD
Buffer A Read in entry block where X points
P.5
DIRSCAN
equal or next match
P.8
TEMP ← Pointer to where found
CCG returned ?
Yes (Duplicate)
NO
3
DUPNAME
P.11
Index PNTR2 ← Index PNTR + ISIZE (6)
Xreg ← DA LPNTR + ZHALF1
Point to second half of block
(TEMP = where found
TEMP <= XREG ?
NO
Yes
TEMP < XREG?
NO
Yes
XREG ← XREG − DA XSIZE
XREG ← Pointer to entry in DDS
8
ZNEWENTRYBLOCK
allocate new entry block (Name pntr is XREG) - update Index PNTR2
allocate New block P.9
Save P IN X PNTR
Set ECOUNT IN PNTR2 Index entry To ZH2
Set ETOTAL IN Prefix To NEWPREIETOTAL (Buffer B)
Set ZH1 INTO INDEX (PNTR) ECOUNT
TEMP = first valid entry in A?
No
Yes
Move entry Name to Index entry (PNTR)
9
P.14


From p.13
9
DIRWRITE
write out buffer B
p.4
DA NUMVALID ← DB EBSIZE
set up NULL BLOCK IN B
DB LPNTR ← DB IOPNTR
DB NUMVALID ← DB EBSIZE
entry size (wds)
DB XSIZE ← DB EXSIZE
Index BLK size (wds)
DB USED ← ∅
DBBSIZE ← DBEBSIZE
USED
BLK SIZE (Sect.)
(?)
BNSIZE ← DBEBSIZE *128
BLK SIZE (wds)
Entry size (wds)
BFACTOR ← BNSIZE / XSIZE
BLK. FACTOR
BLK. SIZE (wds)
entry size
MISCWD ← EMISCWD
Copy miscwd
XCOUNT ← ∅
ZINSERT
Insert entry IN New block
P.9
Store index Pointer
ZDISTRIBUTE
distribute with New block
P.10
10
P.12
col 2
good return

DIRC (MPE 1.2)

DIRREMOVE (P,I)
DIRFIND (P,I)

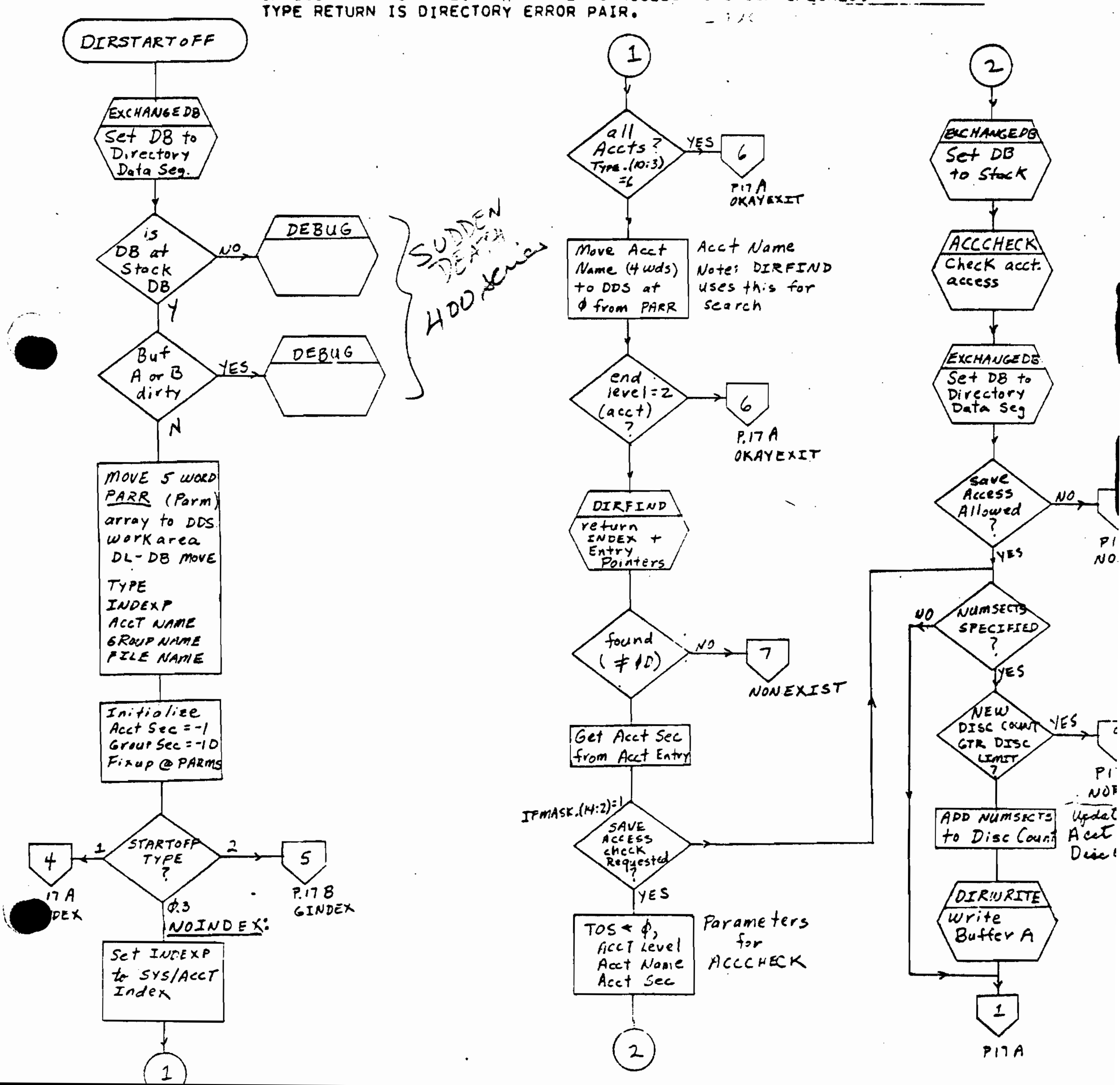
30/OK

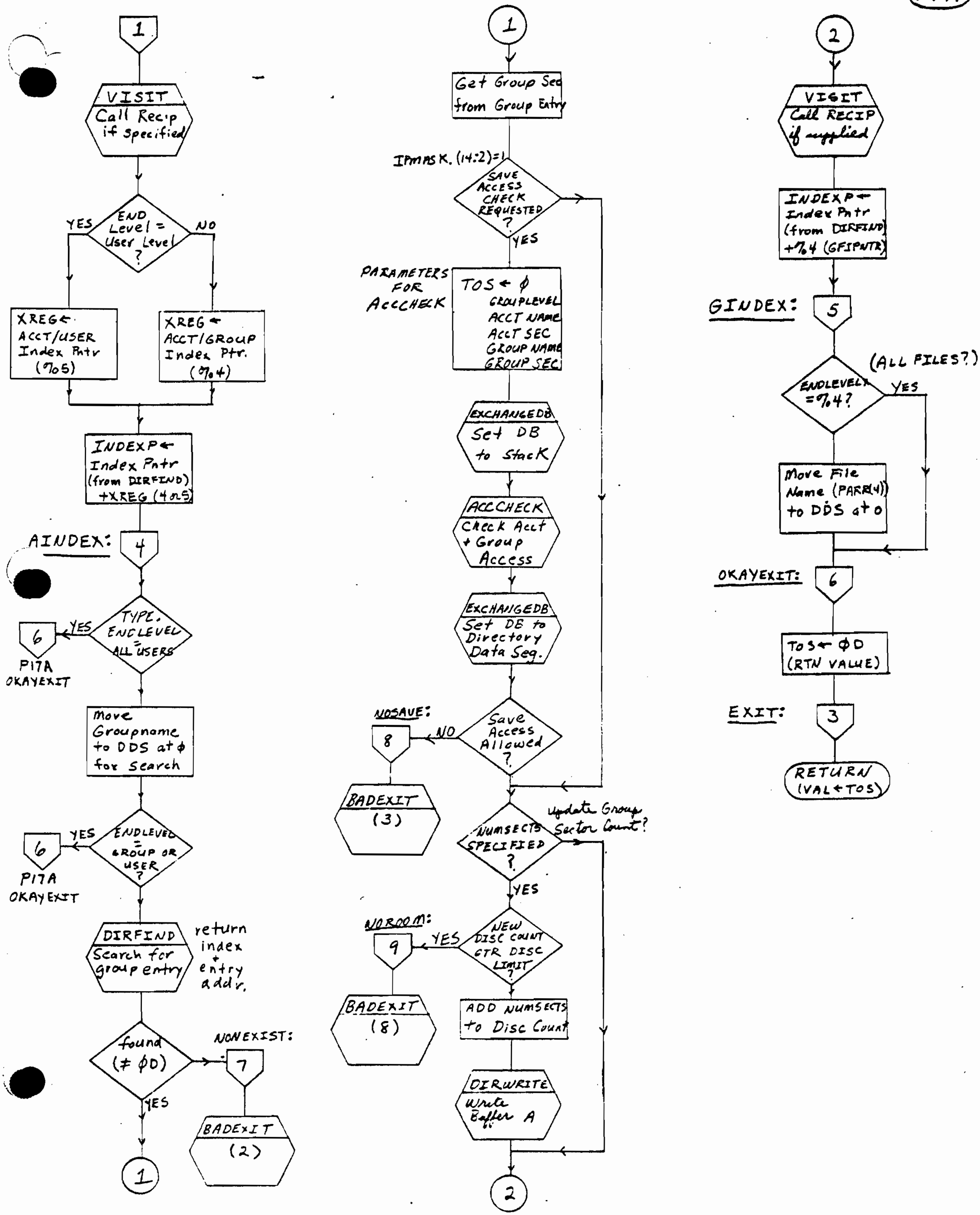


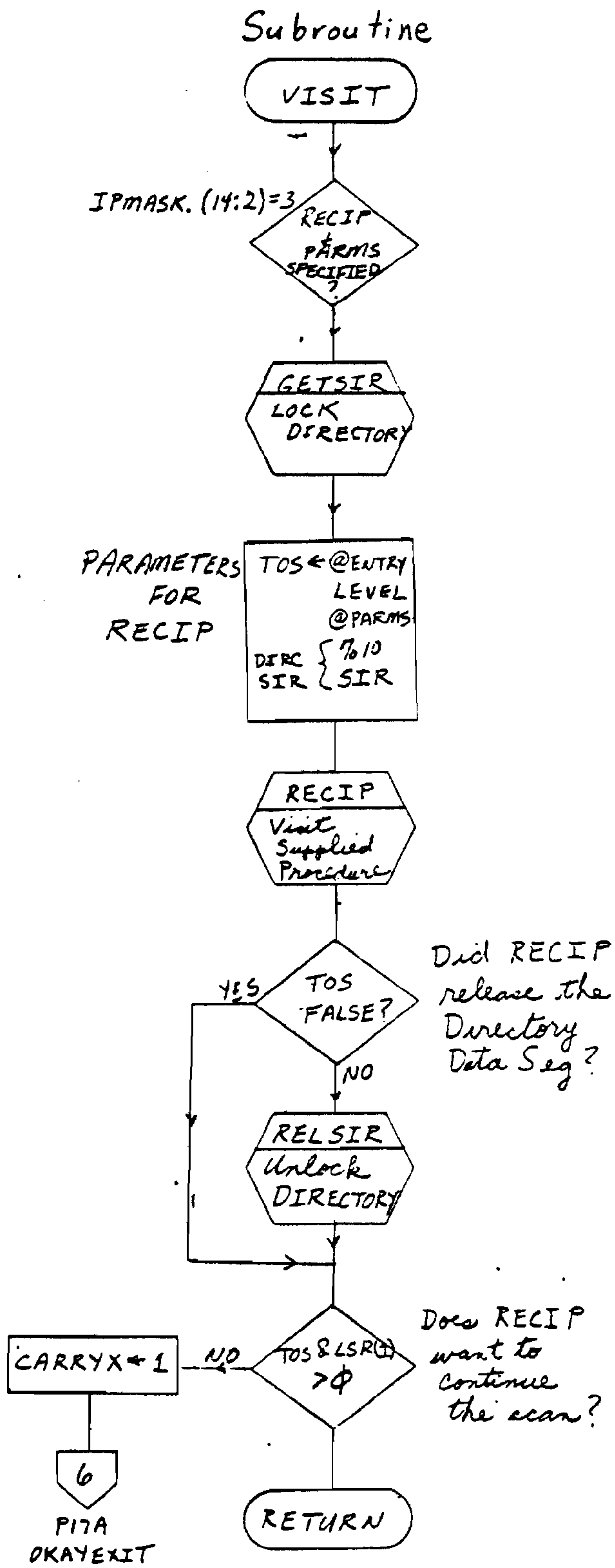
30/1K FORMERLY SIMPLESTARTOFF

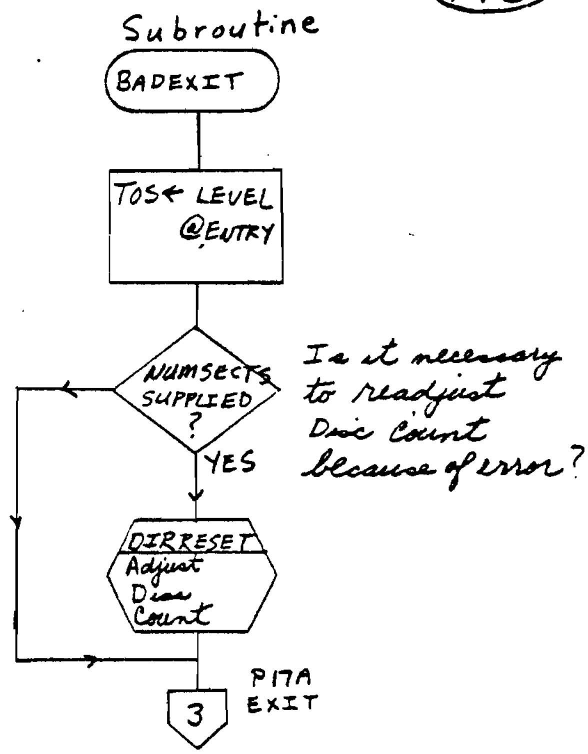
16

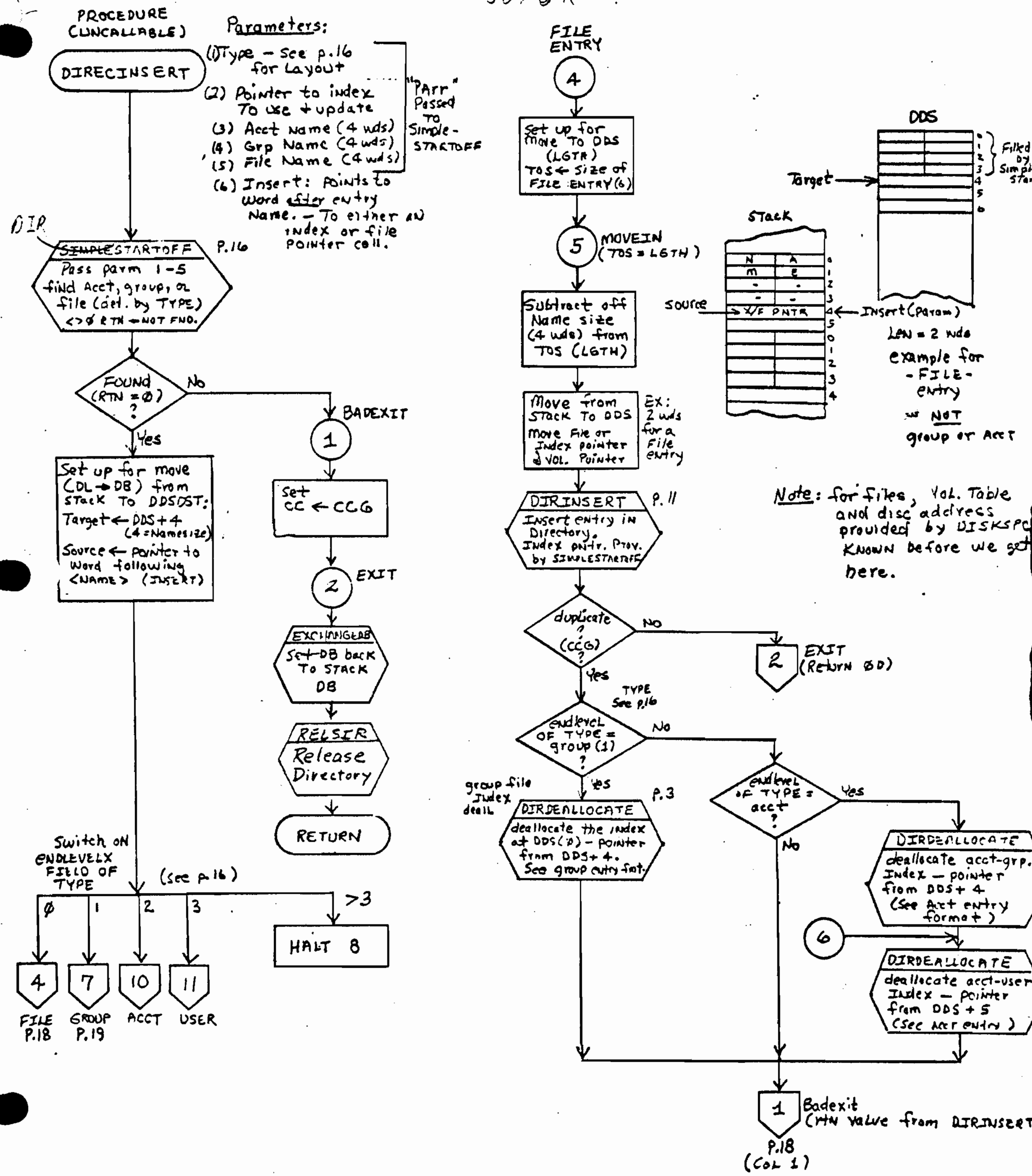
```
DOUBLE PROCEDURE DIRSTARTOFF (PARR, NUMSECTS, RECIP, PARMS);
   VALUE NUMSECTS, PARMS;
   ARRAY PARR;                           <<DB ADDR OF SPEC PART>>
   DOUBLE NUMSECTS;                      <<TO ADJUST ACCT/GROUP>>
   INTEGER PROCEDURE RECIP;              <<FOR VISIT OF @ HIT>>
   INTEGER PARMS;                        <<FOR VISIT OF @ HIT>>
   OPTION VARIABLE, PRIVILEGED, INTERNAL;
<<
   ANALYZES THE SPECIFICATION PART FOR DIRECTORY ROUTINES, AND
   GOES DOWN TREE UNTIL JUST BEFORE HIT OF TARGET, LEAVING:
      ADJUST, XTYPE, XINDEXP, XANAME, XGUNAME, XFNAME, XASEC AND XGSEC;
      DB THRU DB+3 TO FINAL NAME.
   IF <NUMSECTS> SPECIFIED, THEN IT'S ADDED TO ACCT AND GROUP.
   IF <RECIP> AND <PARMS> SPECIFIED, THEN @ ENTRY HIT IS VISITED.
      CARRY SET ON RETURN => RECIP SAID STOP OR DON'T SCAN MY TREE.
   IF JUST <PARMS> SPECIFIED, THEN S ACCESS TO GROUP CHECKED.
   TYPE RETURN IS DIRECTORY ERROR PAIR.
```

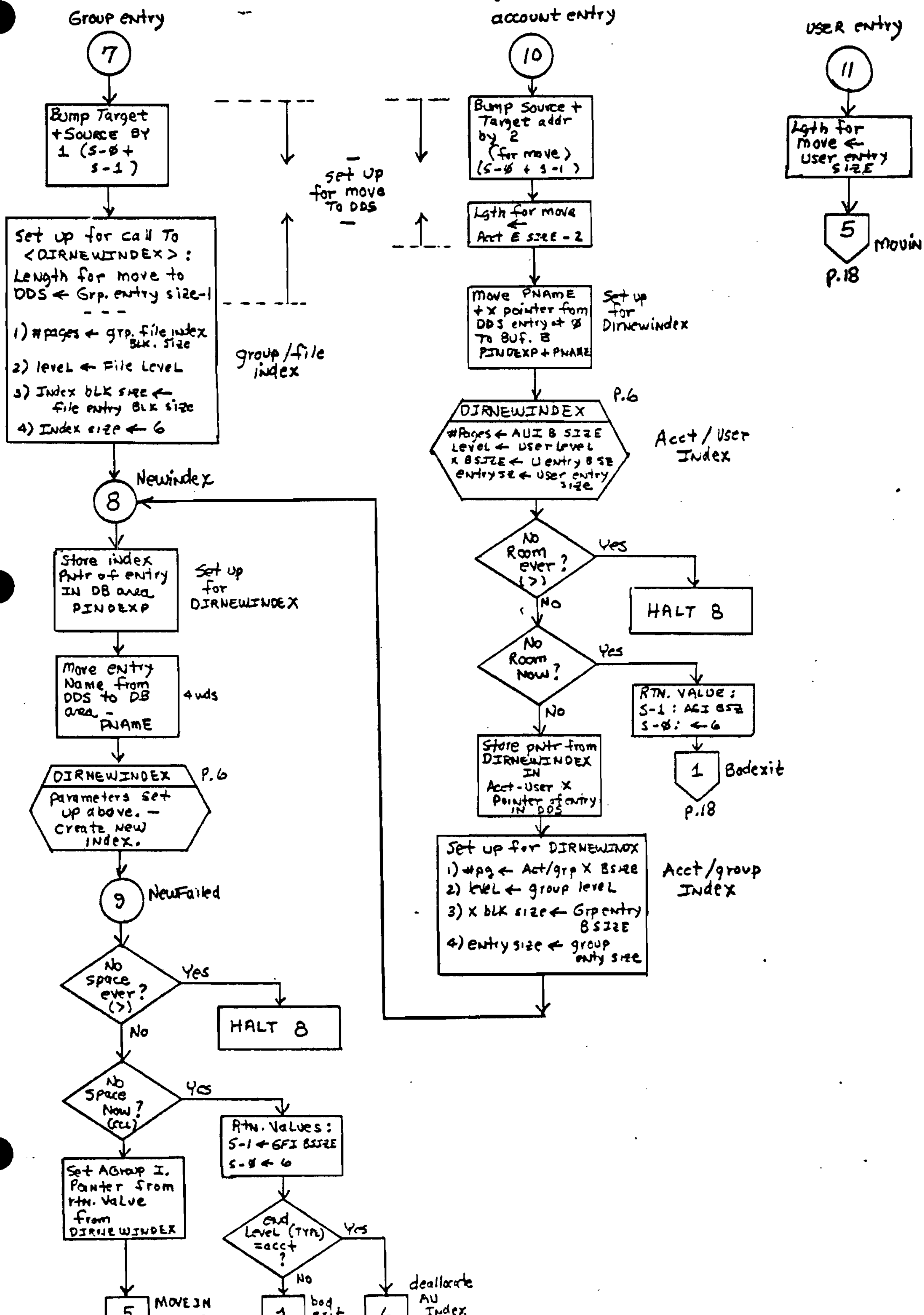



1
VISIT
Call Recip if specified
END Level = User Level ?
YES
NO
XREG ← ACCT/USER Index Pntr (%5)
XREG ← ACCT/GROUP Index Ptr. (%4)
INDEXP ← Index Pntr (from DIRFIND) +XREG (4 or 5)
AINDEX:
4
TYPE. ENDLEVEL = ALL USERS
YES
6
P17A OKAYEXIT
Move Groupname to DDS at ϕ for Search
ENDLEVEL = GROUP OR USER ?
YES
6
P17A OKAYEXIT
DIRFIND
Search for group entry
return index + entry addr.
found (≠ ϕD)
NONEXIST:
7
BADEXIT (2)
YES
1
1
Get Group Sec from Group Entry
IPMASK.(14:2)=1
SAVE ACCESS CHECK REQUESTED ?
YES
PARAMETERS FOR AccCHECK
TOS ← ϕ
GROUPLEVEL
ACCT NAME
ACCT SEC
GROUP NAME
GROUP SEC
EXCHANGEDB
Set DB to Stack
ACCCHECK
Check Acct + Group Access
EXCHANGEDB
Set DB to Directory Data Seg.
Save Access Allowed ?
NO
NOSAVE:
8
BADEXIT (3)
NUMSECTS SPECIFIED ?
Update Group Sector Count?
YES
NEW DISC COUNT GTR DISC LIMIT ?
YES
NOROOM:
9
BADEXIT (8)
ADD NUMSECTS to Disc Count
DIRWRITE
Write Buffer A
2
2
VISIT
Call RECIP if supplied
INDEXP ← Index Pntr (from DIRFIND) +%4 (GFIPNTR)
GINDEX:
5
(ALL FILES?)
ENDLEVEL = %4?
YES
Move File Name (PARM(4)) to DDS at 0
OKAYEXIT:
6
TOS ← ϕD (RTN VALUE)
EXIT:
3
RETURN (VAL ← TOS)


Subroutine
VISIT
IPMASK.(14:2)=3
RECIP PARMS SPECIFIED ?
GETSIR
LOCK DIRECTORY
PARAMETERS FOR RECIP
TOS ← @ENTRY
LEVEL
@PARMS
DIRC { %10
SIR { SIR
RECIP
Visit Supplied Procedure
YES
TOS FALSE?
NO
Did RECIP release the Directory Data Seg?
RELSIR
Unlock DIRECTORY
CARRYX ← 1
NO
TOS & LSR(1) > Φ
Does RECIP want to continue the scan?
6
P17A
OKAYEXIT
RETURN

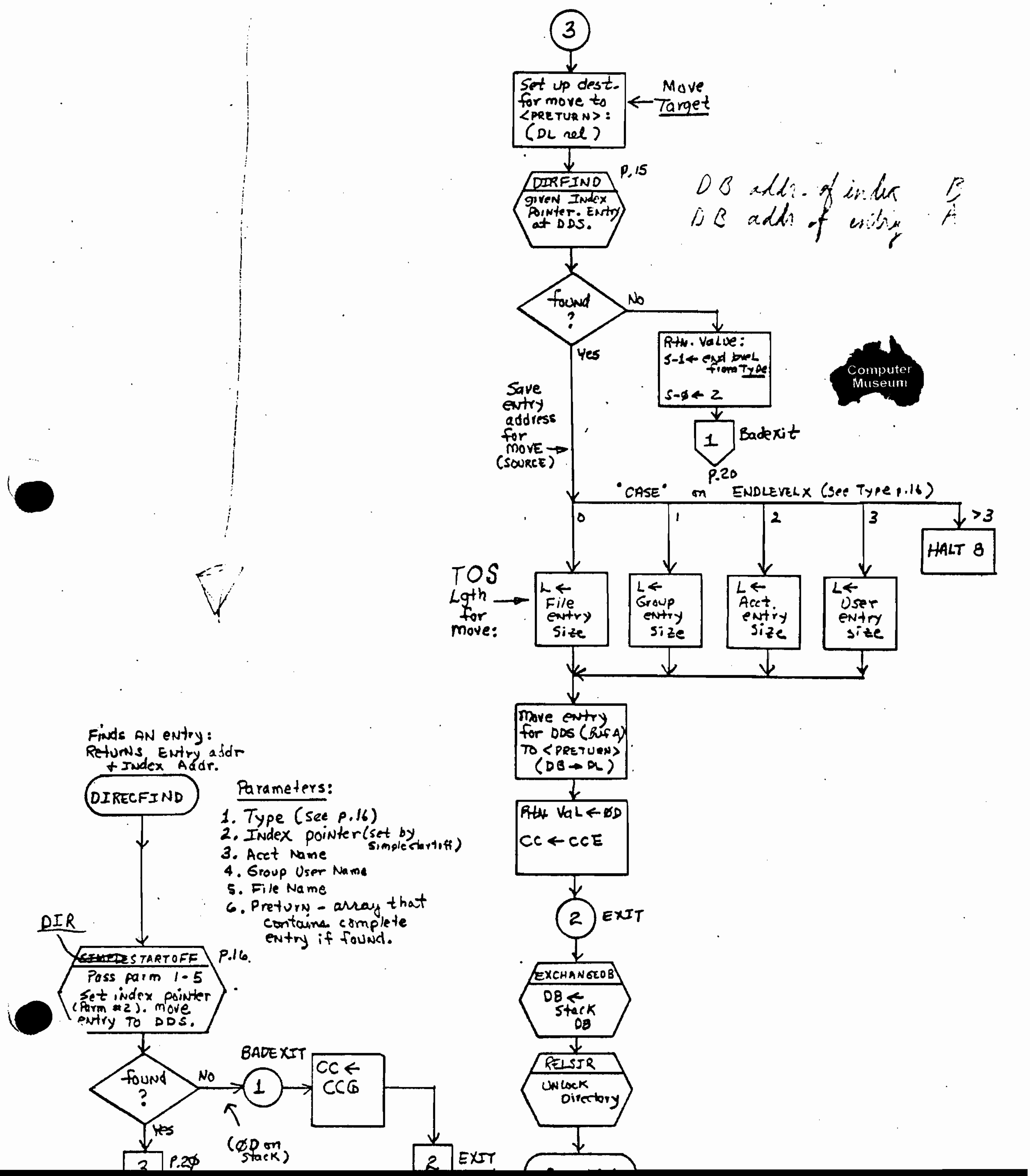

Subroutine
BADEXIT
TOS ← LEVEL
@ENTRY
NUMSECTS SUPPLIED ?
Is it necessary to readjust Disc Count because of error?
YES
DIRRESET
Adjust Disc Count
3
P17A
EXIT


PROCEDURE (UNCALLABLE)
DIRECINSERT
Parameters:
(1) Type – see p.16 for Layout
(2) Pointer to index To use + update
(3) Acct Name (4 wds)
(4) Grp Name (4 wds)
(5) File Name (4 wds)
"PARR" Passed To Simple-STARTOFF
(6) Insert: points to word after entry Name. – To either an index or file pointer cell.
DIR
SIMPLESTARTOFF P.16
Pass parm 1–5 find Acct, group, or file (det. by TYPE)
<>Ø RTN ← NOT FND.
FOUND (RTN = Ø) ?
No
BADEXIT
1
Set CC ← CCG
2
EXIT
EXCHANGEDB
Set DB back To STACK DB
RELSIR
Release Directory
RETURN
Yes
Set up for move (DL → DB) from STACK To DDSDST:
Target ← DDS+4 (4 = Namesize)
Source ← pointer to word following <NAME> (INSERT)
Switch ON ENDLEVELX FIELD OF TYPE
(see p.16)
Ø 1 2 3 >3
HALT 8
4 7 10 11
FILE P.18
GROUP P.19
ACCT
USER
FILE ENTRY
4
Set up for move To DDS (LGTR)
TOS ← size of FILE ENTRY (6)
5
MOVEIN (TOS = LGTH)
Subtract off Name size (4 wds) from TOS (LGTH)
Move from STACK To DDS
move File or Index pointer & VOL. Pointer
Ex: 2 wds for a File entry
DIRINSERT P.11
Insert entry in Directory, Index pntr. Prov. by SIMPLESTARTOFF
duplicate ? (CCG)
NO
2
EXIT (Return ØD)
Yes
TYPE See p.16
endlevel OF TYPE = group (1) ?
No
Yes
group file Index deall
DIRDEALLOCATE P.3
deallocate the index at DDS(Ø) – pointer from DDS+4. See group entry fmt.
endlevel of TYPE = acct ?
Yes
No
DIRDEALLOCATE
deallocate acct-grp. Index – pointer from DDS+4 (See Acct entry format)
6
DIRDEALLOCATE
deallocate acct-user Index – pointer from DDS+5 (See Acct entry)
1
Badexit (rtn value from DIRINSERT)
P.18 (Col 1)
DDS
Filled by Simple Start
Target
0 1 2 3 4 5 6
STACK
N A
m e
X/F PNTR
source
Insert (Param)
0 1 2 3 4 5 0 1 2 3 4
LEN = 2 wds
example for -FILE- entry
NOT group or Acct
Note: for files, Vol. Table and disc address provided by DISKSPC KNOWN before we get here.

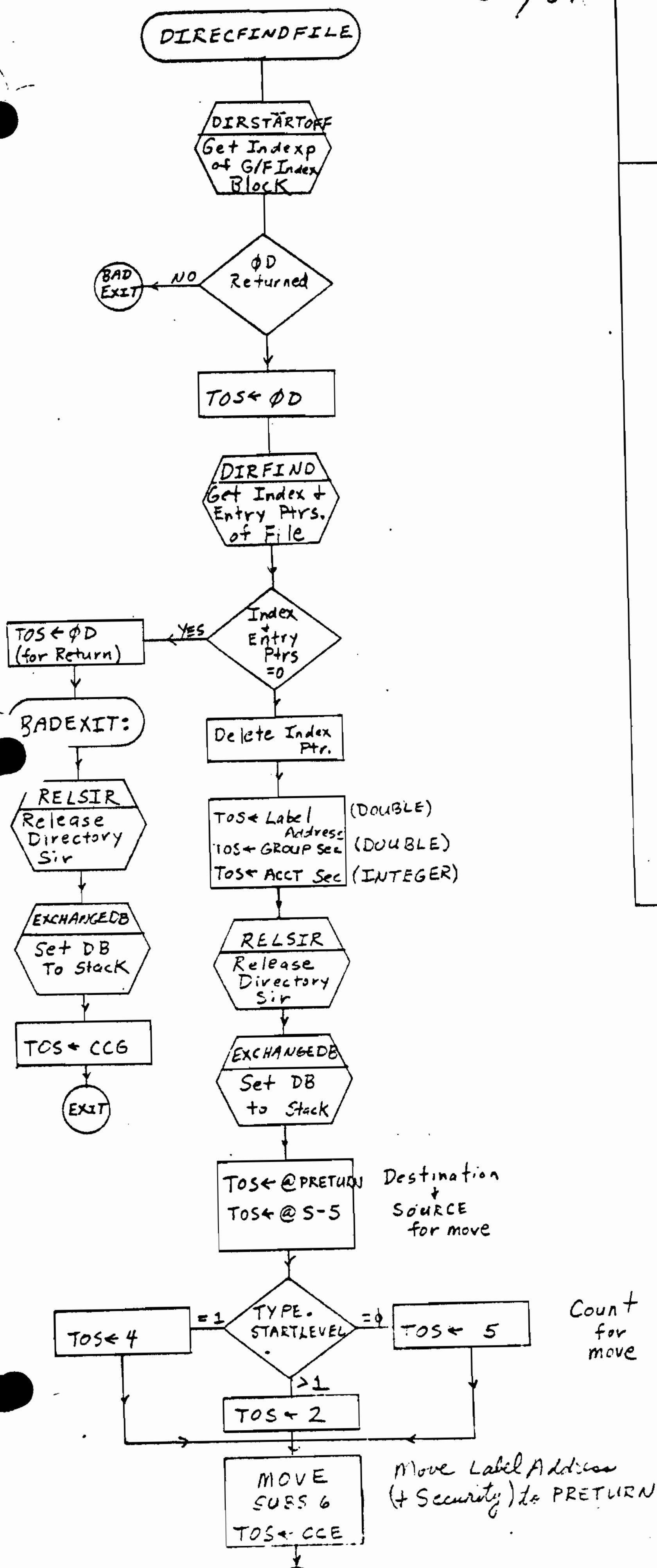

Group entry
7
Bump Target +Source BY 1 (S-Ø + S-1)
Set up for call To <DIRNEWINDEX>:
Length for move to DDS ← Grp. entry size-1
1) #pages ← grp. file index BLK. Size
2) level ← File Level
3) Index blk size ← file entry BLK size
4) Index size ← 6
set up for move To DDS
group/file index
Newindex
8
Store index Pntr of entry IN DB area PINDEXP
Set up for DIRNEWINDEX
Move entry Name from DDS to DB area - PNAME
4 wds
DIRNEWINDEX
Parameters set up above. - Create New Index.
P.6
9
NewFailed
No space ever? (>)
Yes
HALT 8
No
No Space Now? (CCL)
Yes
Rtn. Values:
S-1 ← GFI BSIZE
S-Ø ← 6
Set AGroup I. Pointer from rtn. value from DIRNEWINDEX
5
MOVE IN
end Level (TYPE) =acct ?
Yes
No
1
bad exit
6
deallocate AU Index
account entry
10
Bump Source + Target addr by 2 (for move) (S-Ø + S-1)
Lgth for move ← Acct E size - 2
Move PNAME + X pointer from DDS entry at Ø To BUF. B PINDEXP + PNAME
Set up for Dirnewindex
DIRNEWINDEX
P.6
#Pages ← AUI B SIZE
Level ← user level
X BSIZE ← U entry B SZE
entry sz ← user entry size
Acct / User Index
No Room ever? (>)
Yes
HALT 8
No
No Room Now?
Yes
RTN. VALUE:
S-1 : AGI BSZ
S-Ø : ← 6
1
Badexit
P.18
No
Store pntr from DIRNEWINDEX IN Acct-User X Pointer of entry IN DDS
Set up for DIRNEWINDX
1) #pg ← Act/grp X Bsize
2) level ← group level
3) X blk size ← Grp entry BSIZE
4) entry size ← group entry size
Acct/group Index
USER entry
11
Lgth for move ← USER entry SIZE
5
Movin
P.18

DIRC
(MPE 1.2)
OK/30

DIRECFIND (P,U)



30/OK



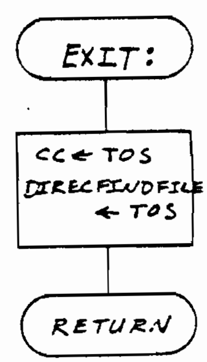
DOUBLE PROCEDURE DIRECFINDFILE (TY…
INDEXP, ANAME, GNAME, FNAME, PRETURN…
VALUE TYPE, INDEXP;
LOGICAL TYPE, INDEXP;
ARRAY ANAME, GNAME, FNAME, PRETURN…

1. TYPE - defined in the legend
2. INDEXP - index pointer
(depends on start level in TYPE…

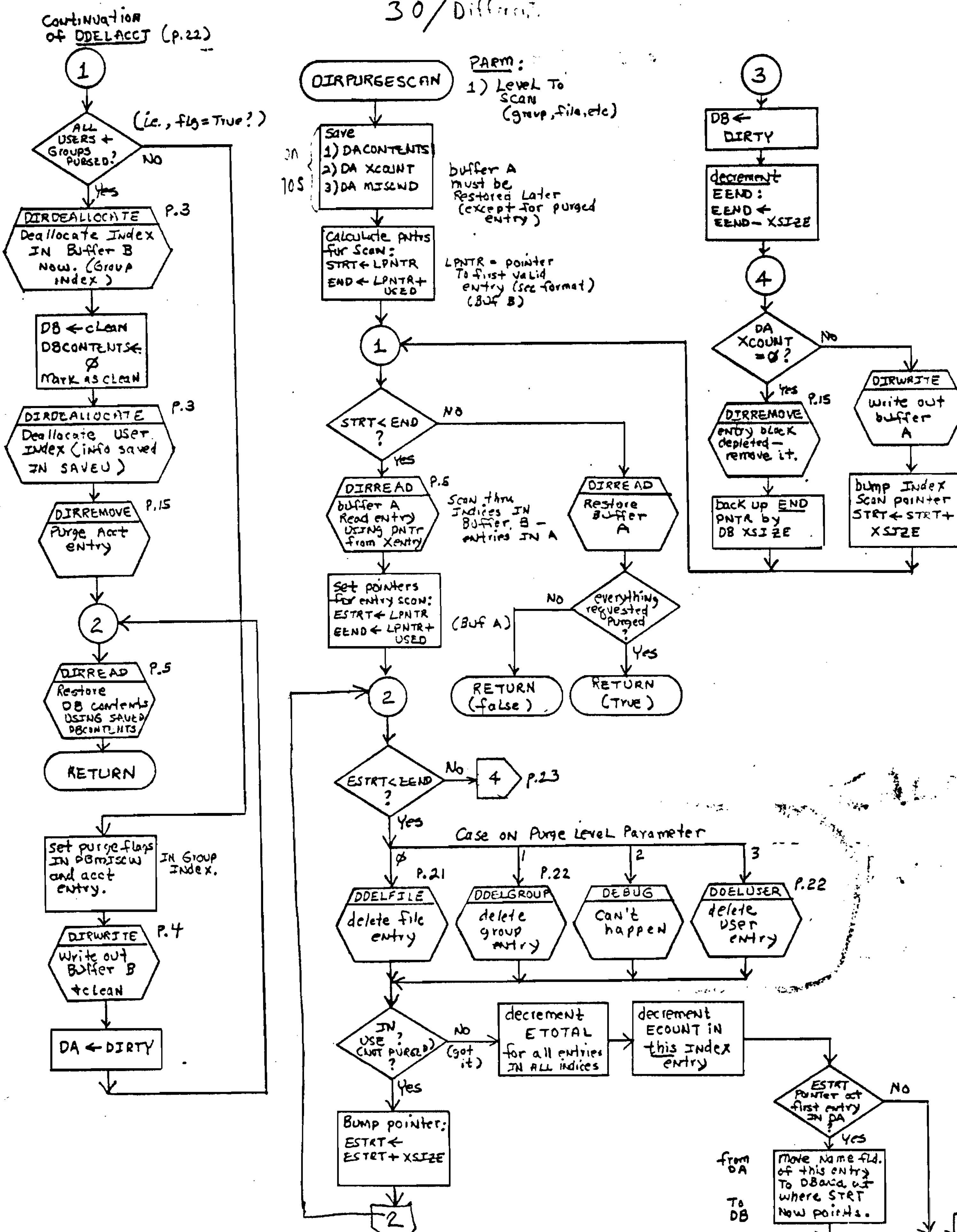
Start Level	INDEXP
Φ	not used
1	ACCT/GROUP Index Poi…
2	GROUP/FILE Index Poi…

3. ANAME - account name
4. GNAME - group name
5. FNAME - file name.
6. PRETURN - Returned information
(depends on start level in T…

Start level	PRETURN
Φ	File Ptr, Group Sec, Acct…
1	File Ptr, Group Sec
2	File Ptr



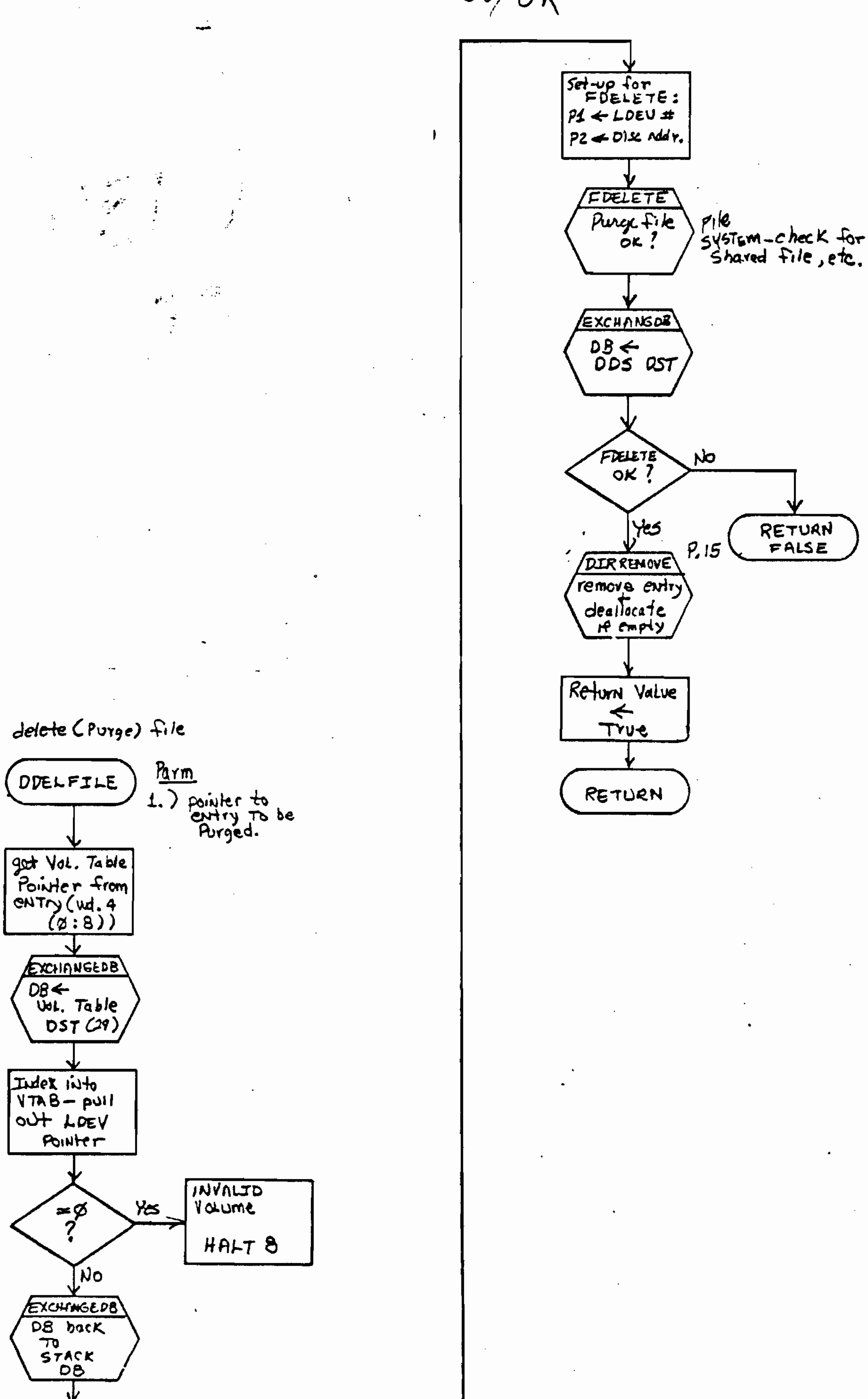
30/Diff...

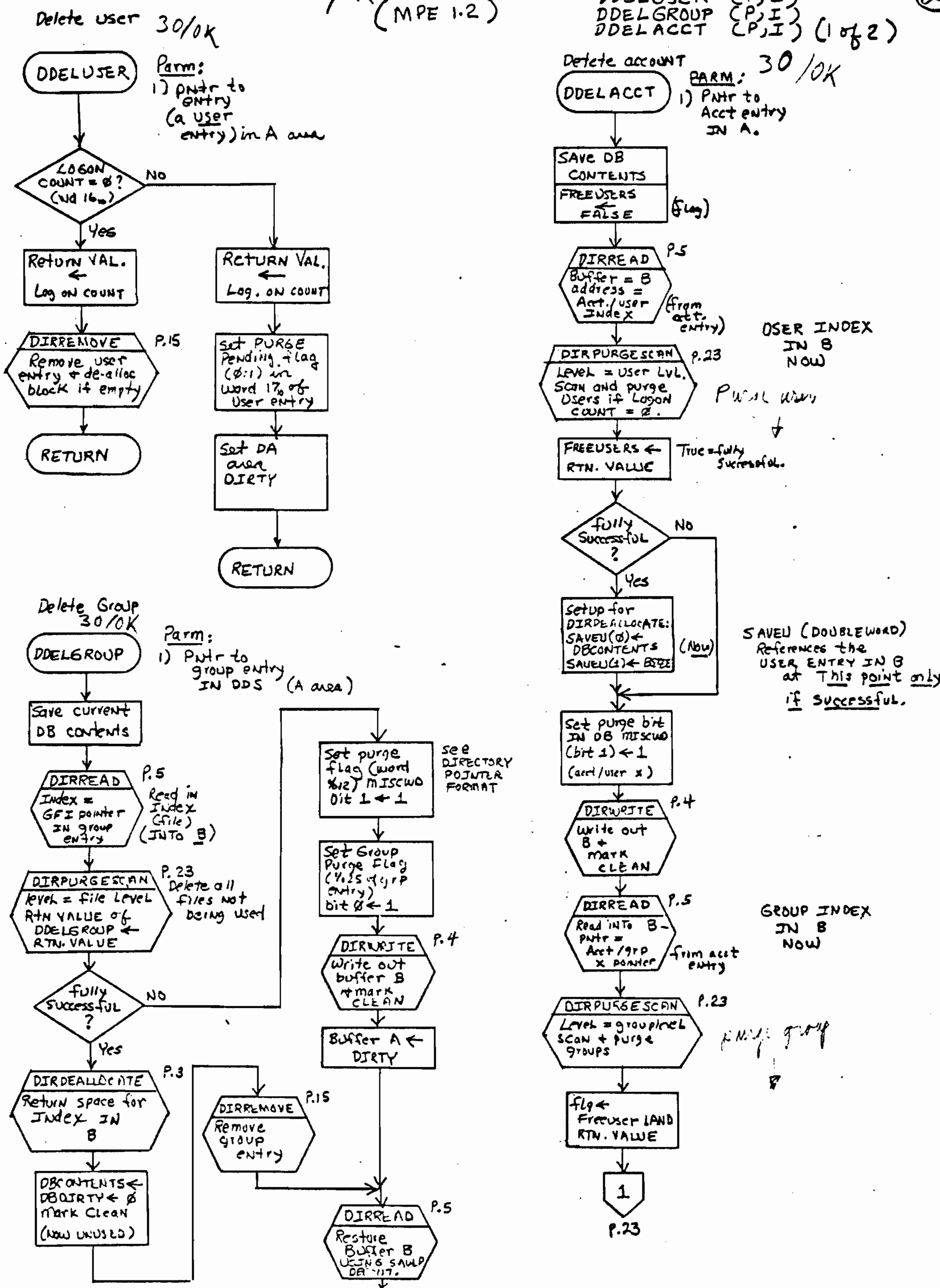

Continuation of DDELACCT (p.22)
1
ALL USERS + Groups PURGED?
(i.e., flg=True?)
No
Yes
DIRDEALLOCATE
P.3
Deallocate Index IN Buffer B Now. (Group index)
DB ← cLean
DBCONTENTS ← Ø
Mark as clean
DIRDEALLOCATE
P.3
Deallocate User Index (info saved IN SAVEU)
DIRREMOVE
P.15
Purge Acct entry
2
DIRREAD
P.5
Restore DB contents USING SAVED DBCONTENTS
RETURN
set purge flags IN DBMISCW and acct entry.
IN Group Index.
DIRWRITE
P.4
Write out Buffer B + clean
DA ← DIRTY
DIRPURGESCAN
PARM:
1) Level To Scan (group, file, etc)
Save
1) DACONTENTS
2) DA XCOUNT
3) DA MISCWD
ON
TOS
buffer A must be Restored Later (except for purged entry)
Calculate pntrs for Scan:
STRT ← LPNTR
END ← LPNTR + USED
LPNTR = pointer To first valid entry (see format) (Buf B)
1
STRT < END ?
NO
Yes
DIRREAD
P.5
buffer A Read entry USING pntr from Xentry
Scan thru Indices IN Buffer B - entries IN A
DIRREAD
Restore Buffer A
everything requested Purged?
No
Yes
(Buf A)
RETURN (false)
RETURN (True)
Set pointers for entry scan:
ESTRT ← LPNTR
EEND ← LPNTR + USED
2
ESTRT < EEND ?
No
4
p.23
Yes
Case ON Purge Level Parameter
Ø
1
2
3
P.21
P.22
P.22
DDELFILE
delete file entry
DDELGROUP
delete group entry
DEBUG
Can't happen
DDELUSER
delete user entry
IN USE? (NOT PURGED)
No
(got it)
Yes
decrement ETOTAL for all entries IN ALL indices
decrement ECOUNT IN this Index entry
Bump pointer:
ESTRT ← ESTRT + XSIZE
2
3
DB ← DIRTY
decrement EEND:
EEND ← EEND - XSIZE
4
DA XCOUNT = Ø?
No
Yes
P.15
DIRREMOVE
entry block depleted - remove it.
DIRWRITE
write out buffer A
back up END PNTR by DB XSIZE
bump Index Scan pointer
STRT ← STRT + XSIZE
ESTRT Pointer at first entry IN DA
No
Yes
from DA
To DB
Move Name fld. of this entry To DBarea at where STRT Now points.

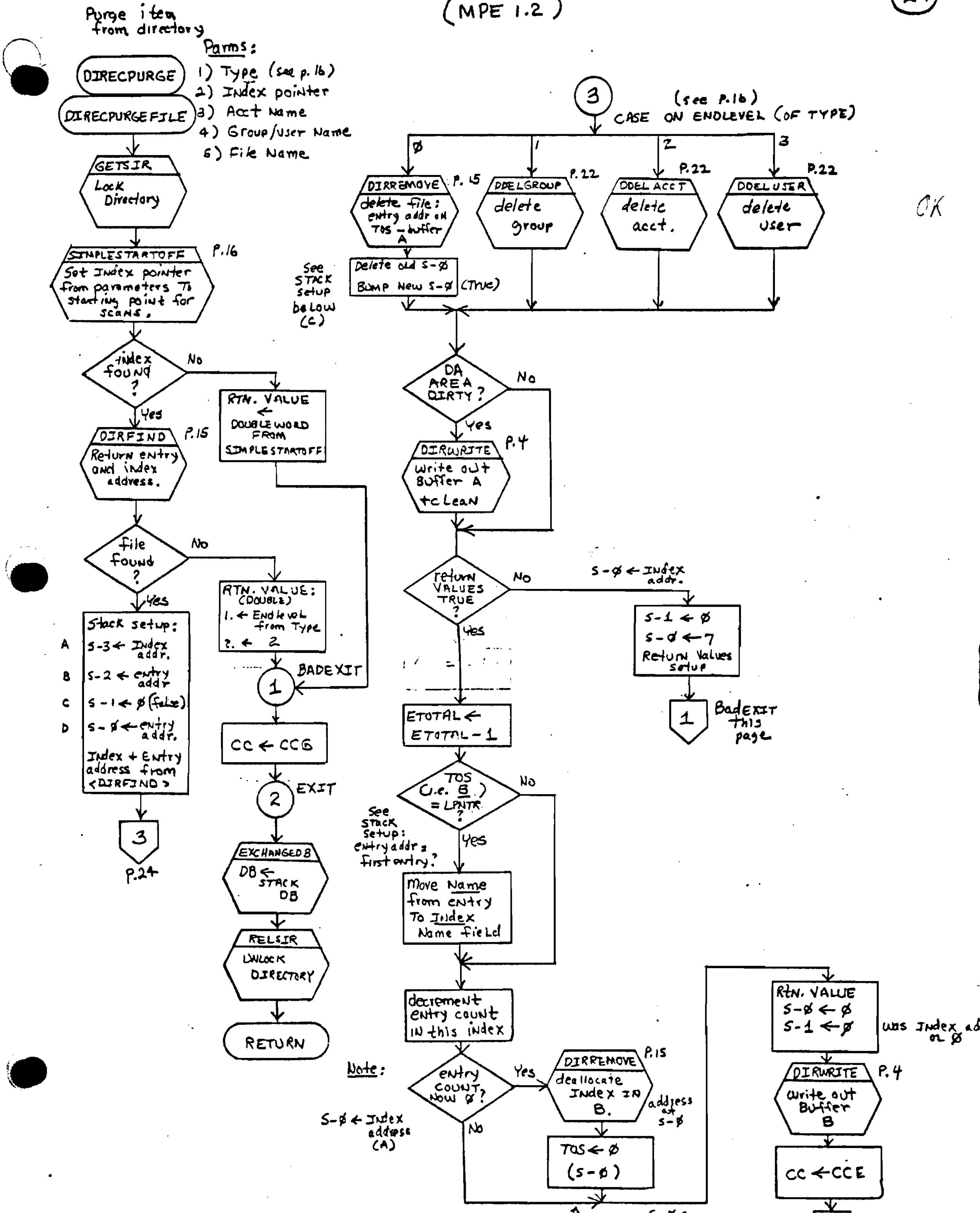
DIRC
(MPE 1.2)

30, OK

DDELFILE(P,I)



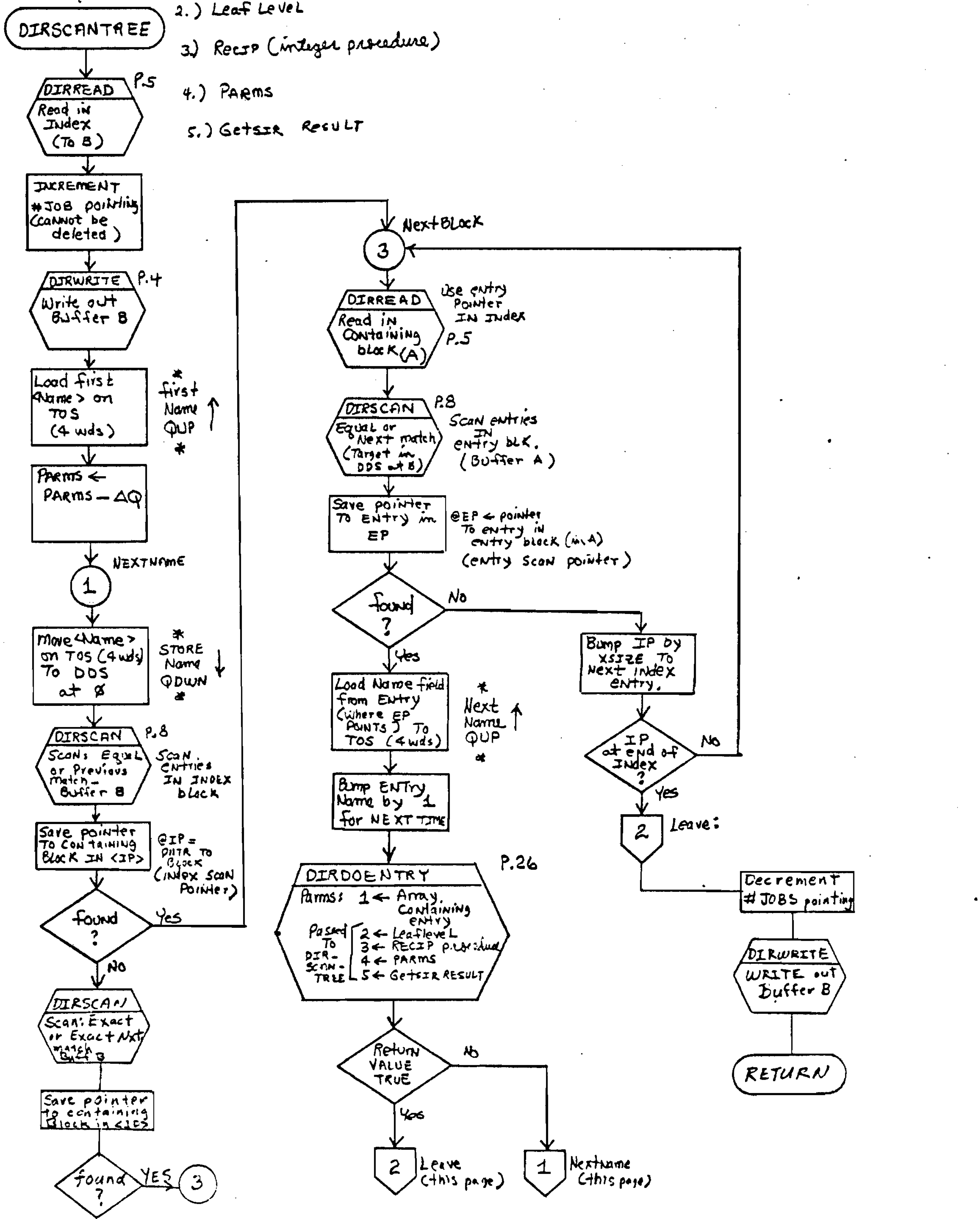

Delete user 30/0K
DDELUSER
Parm:
1) pntr to entry (a user entry) in A area
LOGON COUNT = Ø ? (wd 16₁₀)
NO
Yes
Return VAL. ← Log on count
DIRREMOVE
P.15
Remove user entry & de-alloc block if empty
RETURN
Return Val. ← Log. on count
Set PURGE Pending flag (Ø:1) in word 17₈ of User entry
Set DA area DIRTY
RETURN
Delete Group 30/0K
DDELGROUP
Parm:
1) Pntr to group entry IN DDS (A area)
Save current DB contents
DIRREAD
P.5
Index = GFI pointer IN group entry
Read in Index (file) (INTO B)
DIRPURGESCAN
P.23
Level = file Level
RTN VALUE of DDELGROUP ← RTN. VALUE
Delete all files not being used
fully Successful ?
NO
Yes
DIRDEALLOCATE
P.3
Return space for Index IN B
DBCONTENTS ←
DBDIRTY ← Ø
Mark Clean
(NOW UNUSED)
Set purge flag (word %12) MISCWD bit 1 ← 1
see DIRECTORY POINTER FORMAT
Set Group Purge Flag (%25 of grp entry) bit Ø ← 1
DIRWRITE
P.4
Write out buffer B & mark CLEAN
Buffer A ← DIRTY
DIRREMOVE
P.15
Remove group entry
DIRREAD
P.5
Restore Buffer B using SAVED DA
Delete account
30/0K
DDELACCT
PARM:
1) Pntr to Acct entry IN A.
Save DB CONTENTS
FREEUSERS ← FALSE
(flag)
DIRREAD
P.5
Buffer = B
address = Acct./user Index
(from act. entry)
USER INDEX IN B NOW
DIRPURGESCAN
P.23
Level = User Lvl.
Scan and purge Users if Logon COUNT = Ø.
Purge users
FREEUSERS ← RTN. VALUE
True = fully Successful.
fully Successful ?
No
Yes
Setup for DIRDEALLOCATE:
SAVEU(Ø) ← DBCONTENTS
SAVEU(1) ← BSIZE
(Now)
SAVEU (DOUBLEWORD) References the USER ENTRY IN B at This point only if successful.
Set purge bit IN DB MISCWD (bit 1) ← 1
(acct/user x)
DIRWRITE
P.4
Write out B & mark CLEAN
DIRREAD
P.5
Read into B - Pntr = Acct/grp x pointer
from acct entry
GROUP INDEX IN B NOW
DIRPURGESCAN
P.23
Level = group level
Scan & purge groups
Empty group
Flg ← Freeuser LAND RTN. VALUE
1
P.23


Purge item from directory
Parms:
1) Type (see p. 16)
2) Index pointer
3) Acct Name
4) Group/user Name
5) File Name
DIRECPURGE
DIRECPURGEFILE
GETSIR
Lock Directory
SIMPLESTARTOFF
P. 16
Set Index pointer from parameters To starting point for scans.
index found ?
No
Yes
RTN. VALUE ← DOUBLE WORD FROM SIMPLESTARTOFF
DIRFIND
P. 15
Return entry and index address.
file found ?
No
Yes
RTN. VALUE: (DOUBLE)
1. ← Endlevel from Type
2. ← 2
BADEXIT
1
Stack setup:
A S-3 ← Index addr.
B S-2 ← entry addr
C S-1 ← ø (false)
D S-ø ← entry addr.
Index + Entry address from <DIRFIND>
3
P. 24
CC ← CCG
2
EXIT
EXCHANGEDB
DB ← STACK DB
RELSIR
UNLOCK DIRECTORY
RETURN
3
(see P. 16)
CASE ON ENDLEVEL (OF TYPE)
ø
1
2
3
DIRREMOVE
P. 15
delete file: entry addr on TOS - buffer A
DDELGROUP
P. 22
delete group
DDEL ACCT
P. 22
delete acct.
DDELUSER
P. 22
delete user
OK
See STACK setup below (C)
Delete old S-ø
Bump New S-ø (True)
DA AREA DIRTY ?
No
Yes
DIRWRITE
P. 4
write out Buffer A + clean
return VALUES TRUE ?
No
Yes
S-ø ← Index addr.
S-1 ← ø
S-ø ← 7
Return Values setup
1
Badexit this page
ETOTAL ← ETOTAL - 1
TOS (i.e. B) = LPNTR ?
No
Yes
See STACK setup: entry addr = first entry?
Move Name from entry To Index Name field
decrement entry count in this index
Note:
S-ø ← Index address (A)
entry count now ø?
Yes
No
DIRREMOVE
P. 15
deallocate Index in B.
address at S-ø
TOS ← ø
(S-ø)
RTN. VALUE
S-ø ← ø
S-1 ← ø
was Index ad or ø
DIRWRITE
P. 4
write out Buffer B
CC ← CCE
S-ø ←

OK/30

Params:

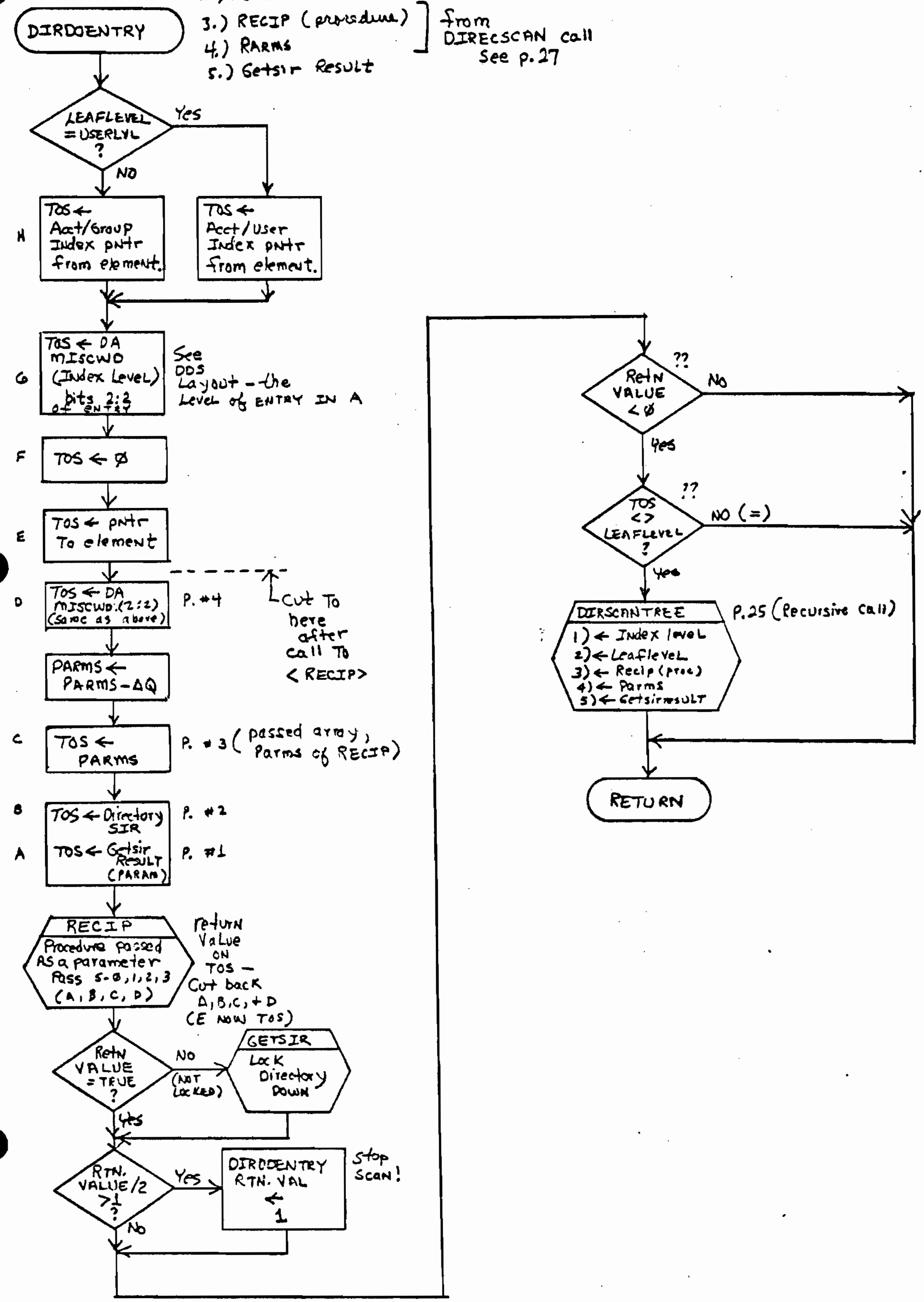
1.) Index (where to start)
2.) Leaf Level
3) Recip (integer procedure)
4.) Parms
5.) Getsir Result



30/OK

Parms:
1.) Array containing element
2.) Leaflevel
3.) RECIP (procedure)
4.) PARMS
5.) Getsir Result

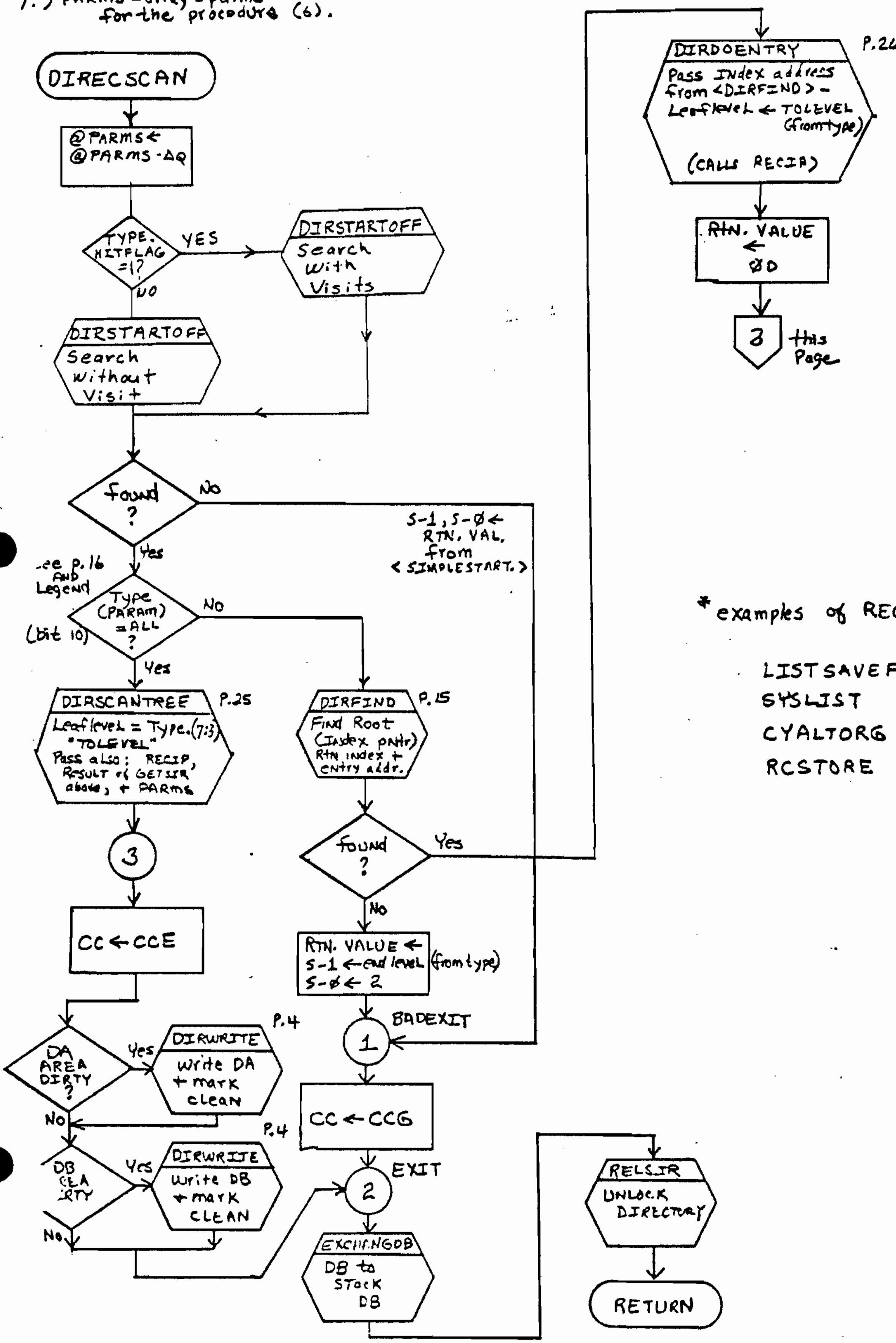
from DIRECSCAN call See p.27



Scan Directory
PARMS:
1.) Type (See legend)
2.) Index pointer
3.) Acct Name - 4A2
Group User Name - 4A2
5.) File Name - 4A2
* 6.) RECIP (procedure) - called for each entry
7.) PARMS - array - parms for the procedure (6).

DIRC (MPE 1.2)

DIRECSCAN (P,U)



* examples of RECIP procedure

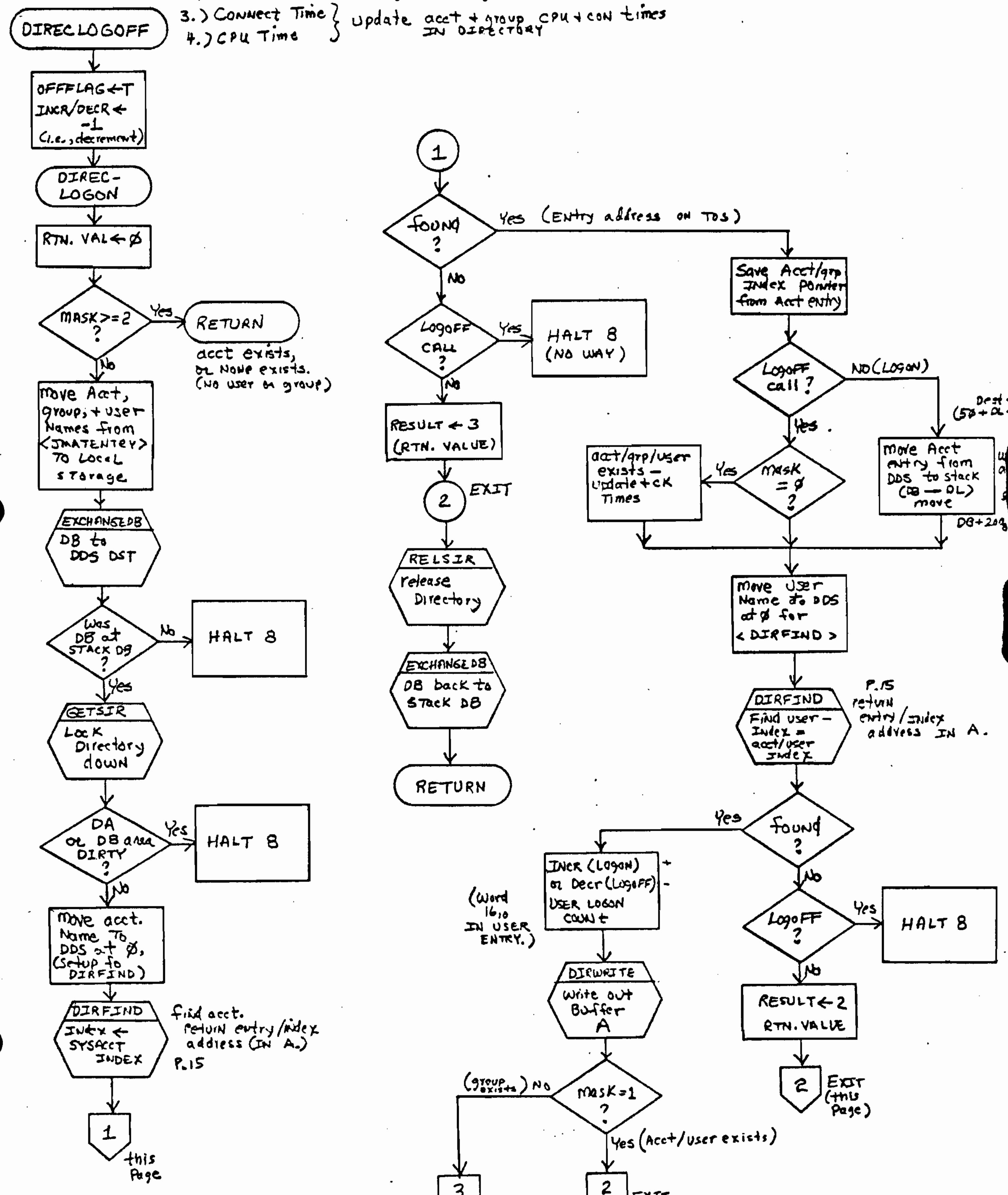
LISTSAVEFILES
SYSLIST } COMMAND' INTER
CYALTORG
RCSTORE (STORRSTR)

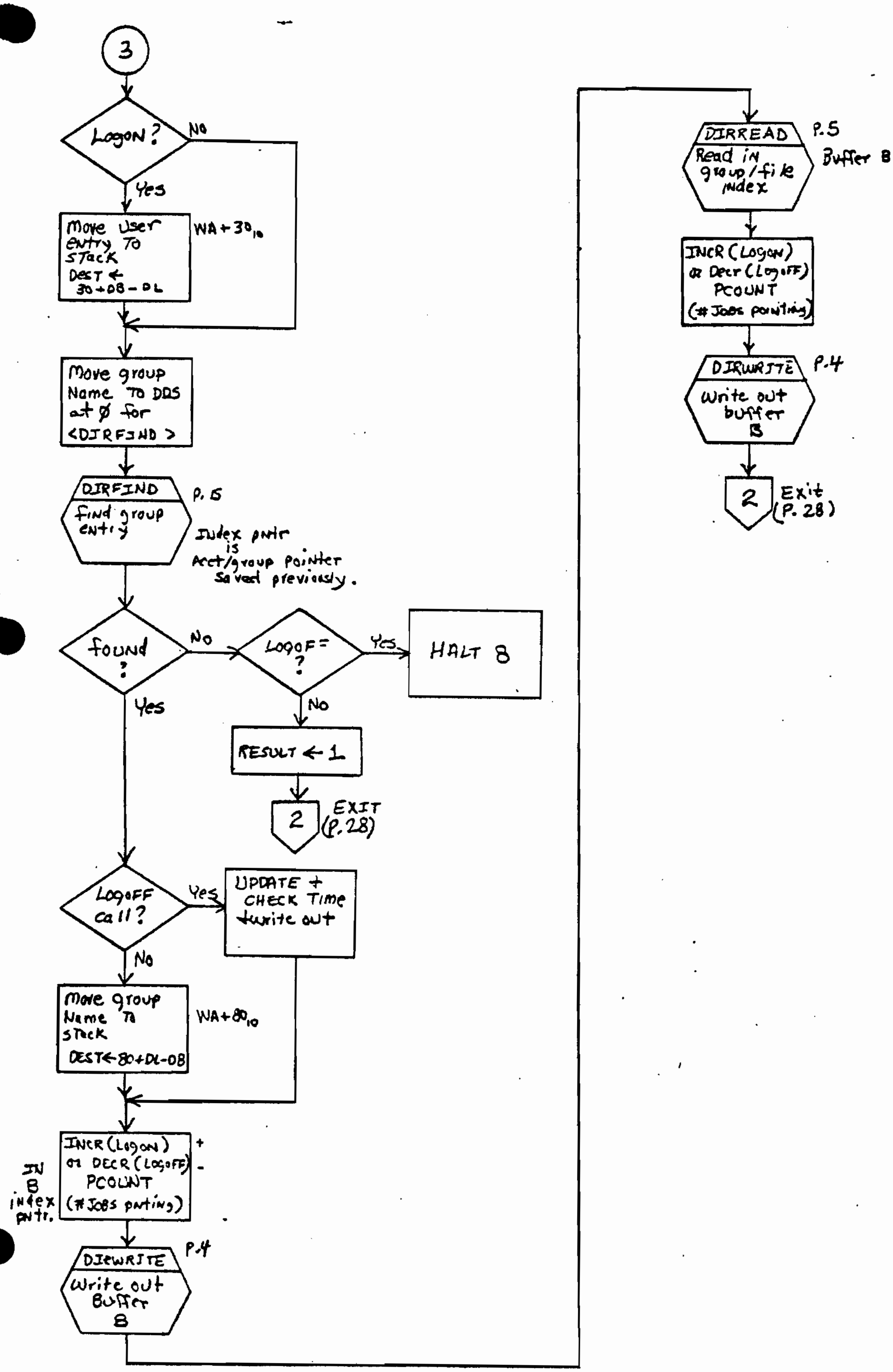
DIRC
(MPE 1.2)

DIRECLOGON (P,U)
DIRECLOGOFF (P,U)
(P. 1 of 2)

Parms:

1.) Mask (Ø for Logon) - determines if acct, group, or user exists.
2.) JMATENTRY (used to get Acct, group, user Names)
3.) Connect Time } update acct + group CPU + con times IN DIRECTORY
4.) CPU Time }




3
LogoN?
No
Yes
Move User entry To STack
DesT ← 30+08−DL
WA+30_{10}
Move group Name To DDS at Ø for <DIRFIND>
DIRFIND
Find group entry
P. 5
Index pntr is Acct/group pointer saved previously.
found ?
No
LOGOF= ?
Yes
HALT 8
No
RESULT ← 1
2
EXIT (P.28)
Yes
LogoFF call?
Yes
UPDATE + CHECK Time +write out
No
Move group Name To stack
DEST←80+DL−DB
WA+80_{10}
INCR (Logon) or DECR (LogoFF) PCOUNT (#Jobs pnting)
+
-
IN B index pntr.
DIRWRITE
Write out Buffer 8
P.4
DIRREAD
Read in group/file index
P.5
Buffer 8
INCR (Logon) or Decr (LogoFF) PCOUNT (# Jobs pointing)
DIRWRITE
Write out buffer 8
P.4
2
Exit (P. 28)

30/0K

(30A)

```
LOGICAL PROCEDURE ACCCHECK (LEVEL, ACCTNAME, ACCTSEC, GROUPNAME, GROUPSEC,
                            CREATOR, FILESEC, USERINFO)
VALUE LEVEL, ACCTSEC, GROUPSEC, FILESEC;
BYTE ARRAY ACCTNAME, GROUPNAME, CREATOR, USERINFO;
INTEGER LEVEL;
LOGICAL ACCTSEC;
DOUBLE GROUPSEC, FILESEC;
OPTION VARIABLE, PRIVILEGED, UNCALLABLE;
```

---

1. LEVEL = level of checking desired
   = 0 - FILE
     1 - GROUP
     2 - ACCT
2. ACCTNAME = Account name to be checked
3. ACCTSEC = Account security mask to be checked
4. GROUPNAME = Group name to be checked (Optional if level = 2)
5. GROUPSEC = Group security mask to be checked (Optional if level = 2)
6. CREATOR = File Creator's name (Optional if level = 1 or 2)
7. FILESEC = File security mask to be checked (Optional if level = 1 or 2)
8. USERINFO (0:7) = User's Account Name
   (8:15) - User's Home Group Name (If level - 1)
   (16:23) = User's Logon Group Name (If level - 1)
   (24:31) = User's Name (If level = 0)

   If USERINFO is omitted the JIT is accessed to get the information

   NOTE: USERINFO = JIT1 (8:23)

.ACC CHECK is a logical procedure and returns 6 bits

	R	A	W	L	X	S

1 - access O.K., 0 = access not allowed ACC CHECK is initialized to %77.

.The User attributes are used to build a user accessor mask which consists of (ANY, AC, AL, GU, GL, CR).

.The user accessor mask is compared with a security mask and if at least one user accessor is elegible, the test passes (i.e. AND the two masks, if the result is not zero, the access test passes; if the result is zero, the test fails).

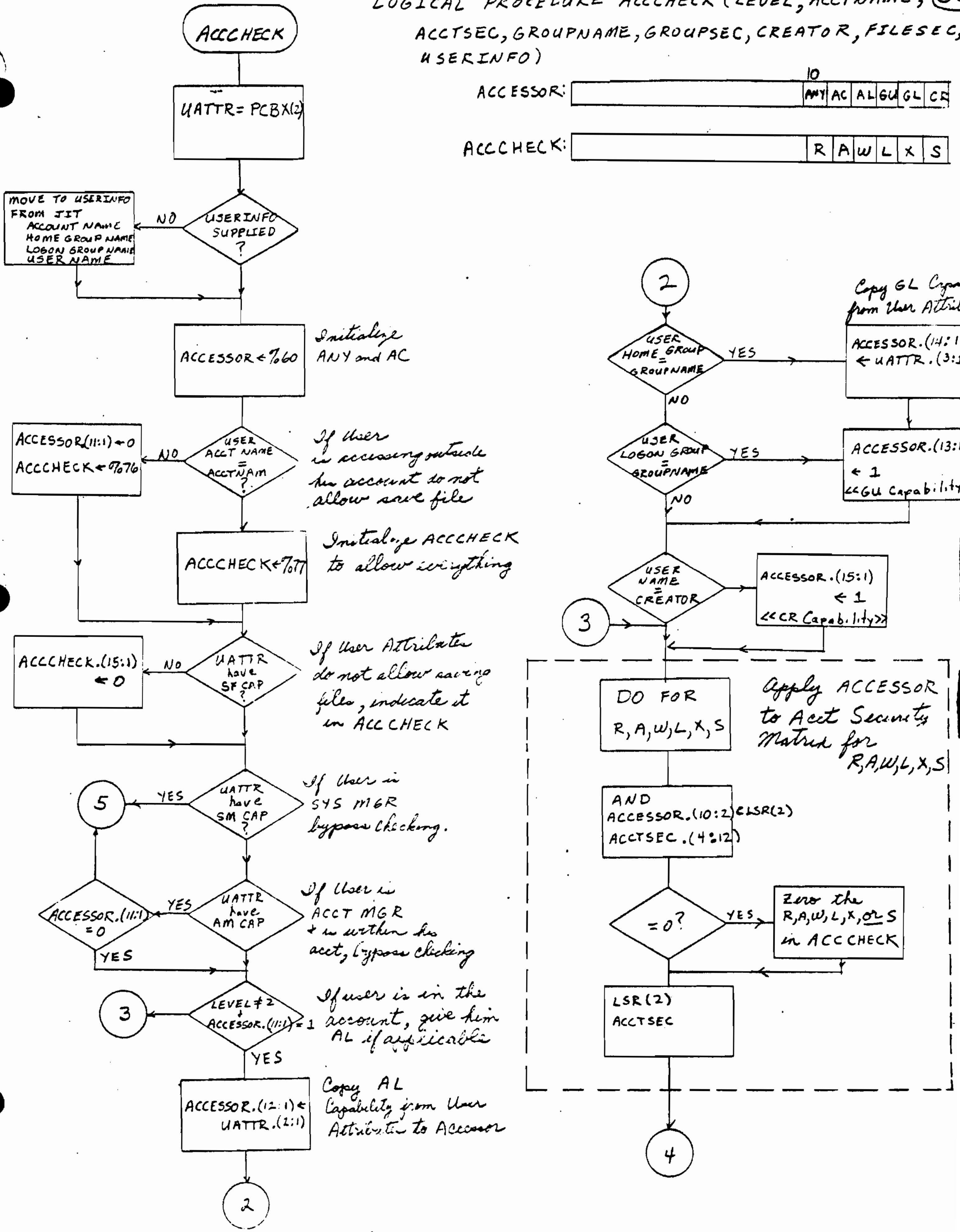
.The mask comparison is made for each type of access (R,A,W,L,X,~~R~~S) at the account level. Whenever a test fails, the corresponding bit in ACCCHECK is set to Ø.

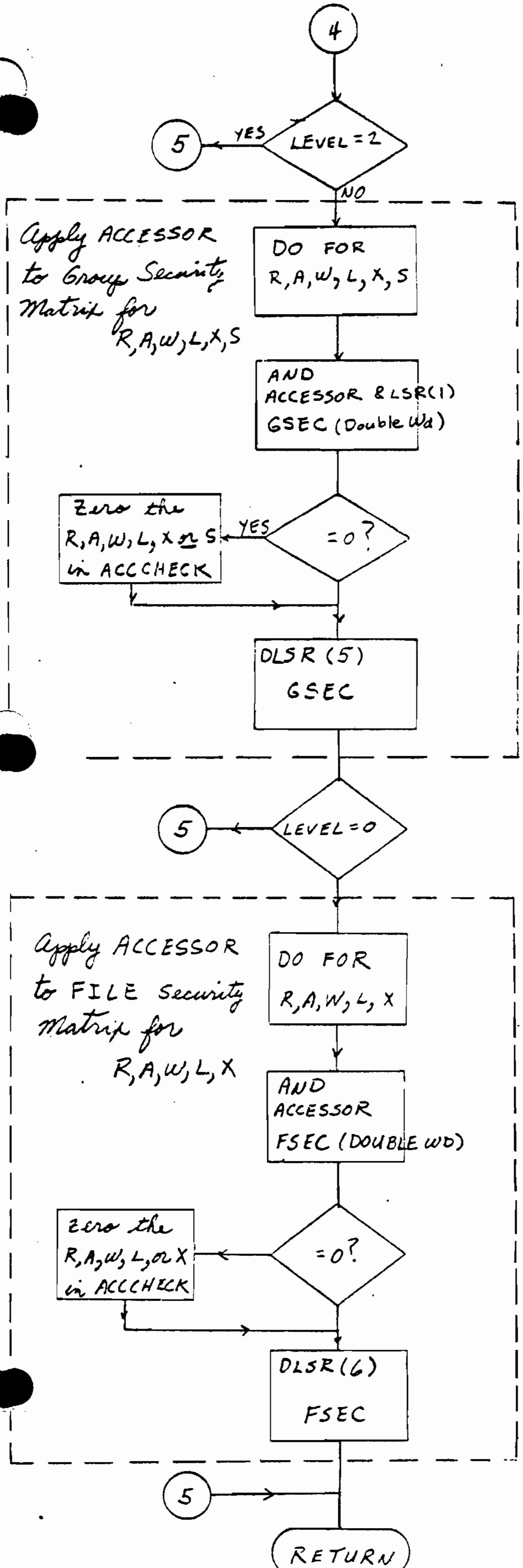
.The comparisons are repeated at the group and file levels depending on LEVEL.

.The final result is ACCCHECK (R,A,W,L,X,S) with a 1 meaning the access is o.k.

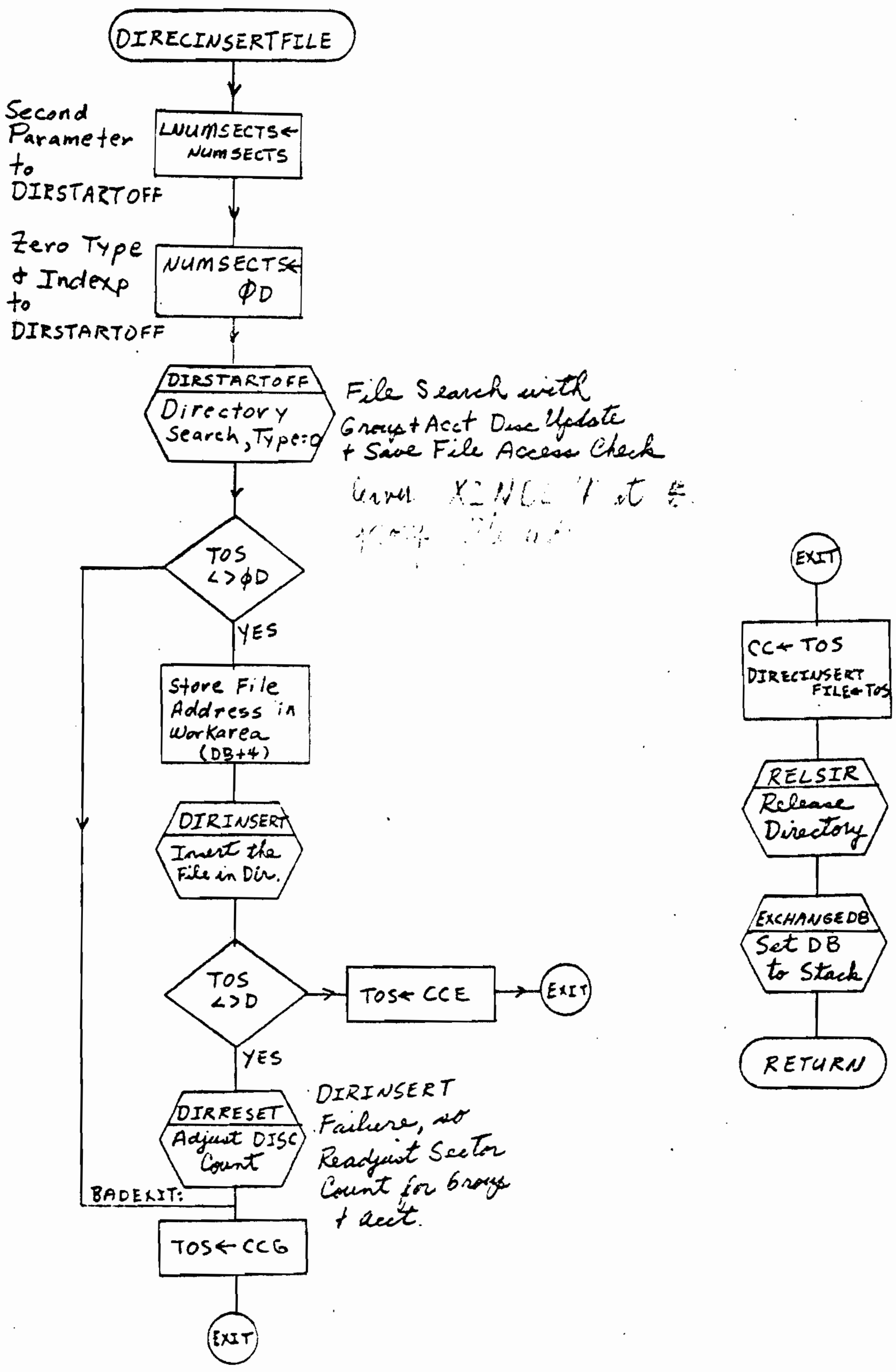
.System manager and Account Manager accessing within the account automatically return a %76 or %77 in ACCHECK.

LOGICAL PROCEDURE ACCCHECK(LEVEL, ACCTNAME, ACCTSEC, GROUPNAME, GROUPSEC, CREATOR, FILESEC, USERINFO)

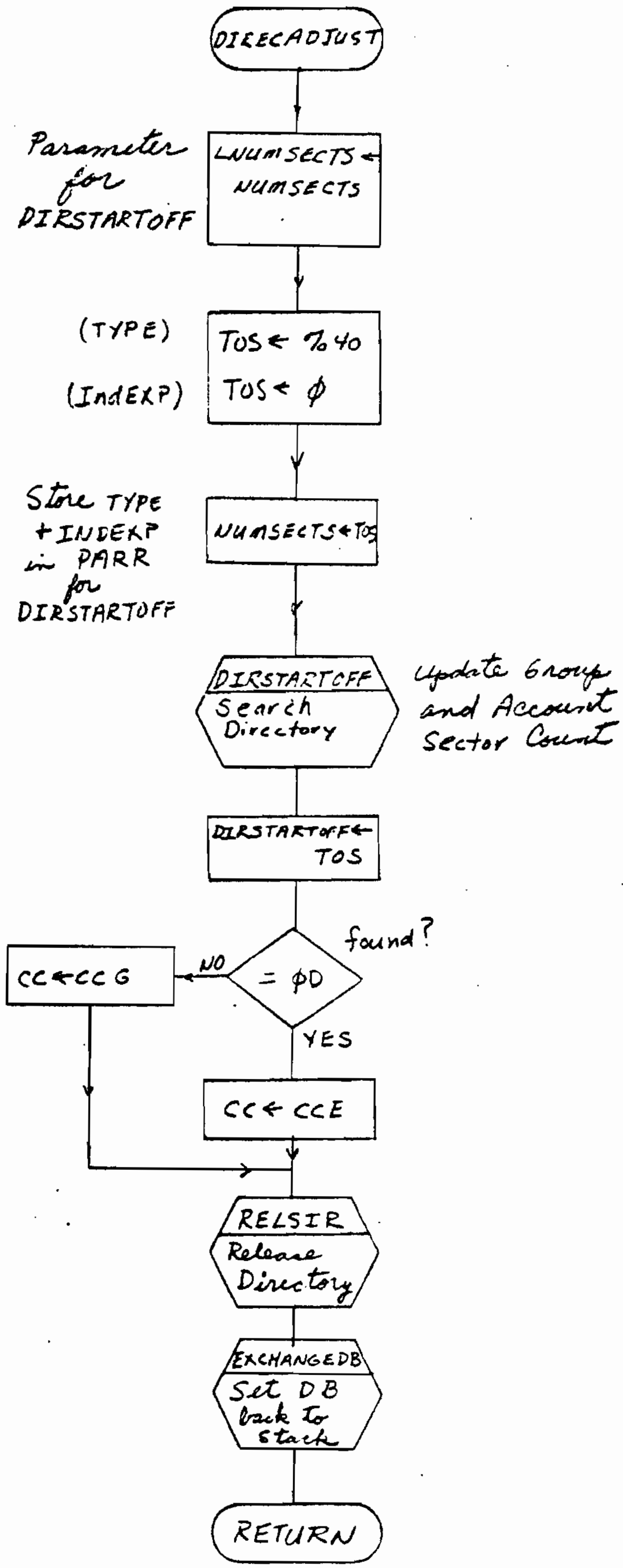



4
5
YES
LEVEL = 2
NO
Apply ACCESSOR to Group Security Matrix for R,A,W,L,X,S
DO FOR R,A,W,L,X,S
AND ACCESSOR & LSR(1) GSEC (Double Wd)
Zero the R,A,W,L,X or S in ACCCHECK
YES
=0?
DLSR (5) GSEC
5
LEVEL=0
Apply ACCESSOR to FILE Security Matrix for R,A,W,L,X
DO FOR R,A,W,L,X
AND ACCESSOR FSEC (DOUBLE WD)
zero the R,A,W,L,or X in ACCCHECK
=0?
DLSR(6) FSEC
5
RETURN

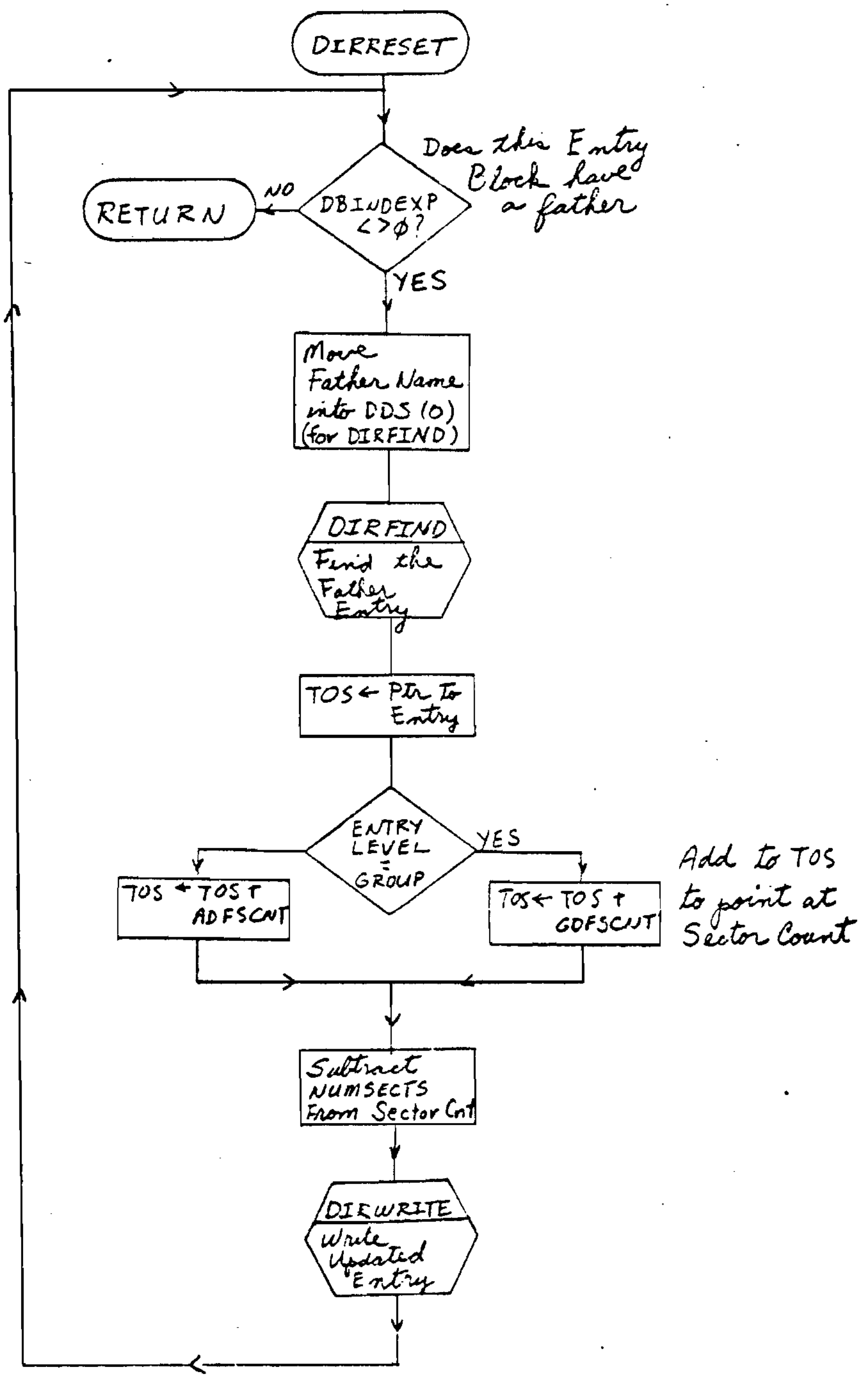
```
DOUBLE PROCEDURE DIRECINSERTFILE (NUMSECTS, ANAME, GNAME,
      FNAME, FADDR);
   VALUE NUMSECTS, FADDR;
   DOUBLE NUMSECTS, FADDR;
   ARRAY ANAME, GNAME, FNAME;
   OPTION PRIVILEGED, UNCALLABLE;
<<
   INSERTS FILE ENTRY UNDER ACCT AND GROUP.
   INCREMENTS ACCT AND GROUP SPACE COUNTS BY <NUMSECTS>.
   CHECKS THAT USER HAS SAVE ACCESS TO GROUP.
   (ALWAYS GLOBAL ACCESS).
```



```
DOUBLE PROCEDURE DIRECADJUST (NUMSECTS, ANAME, GNAME);
   VALUE NUMSECTS;
   DOUBLE NUMSECTS;
   ARRAY ANAME, GNAME;
   OPTION PRIVILEGED, UNCALLABLE;
<< ADJUSTS THA ACCT AND GROUP SPACE COUNTS BY NUMSECTS >>
```



```
PROCEDURE DIRRESET (NUMSECTS);                                         << ... >>
   VALUE NUMSECTS;
   DOUBLE NUMSECTS;
   OPTION INTERNAL, PRIVILEGED;
<< CALLED TO SUBTRACT <NUMSECTS> FROM FATHER (AND GRANDFATHER) WHEN
   ERROR DETECTED AFTER THEY ARE BUMPED.  ASSUMES B CONTAINS CURRENT
   INDEX (THUS POINTER TO FATHER)
```

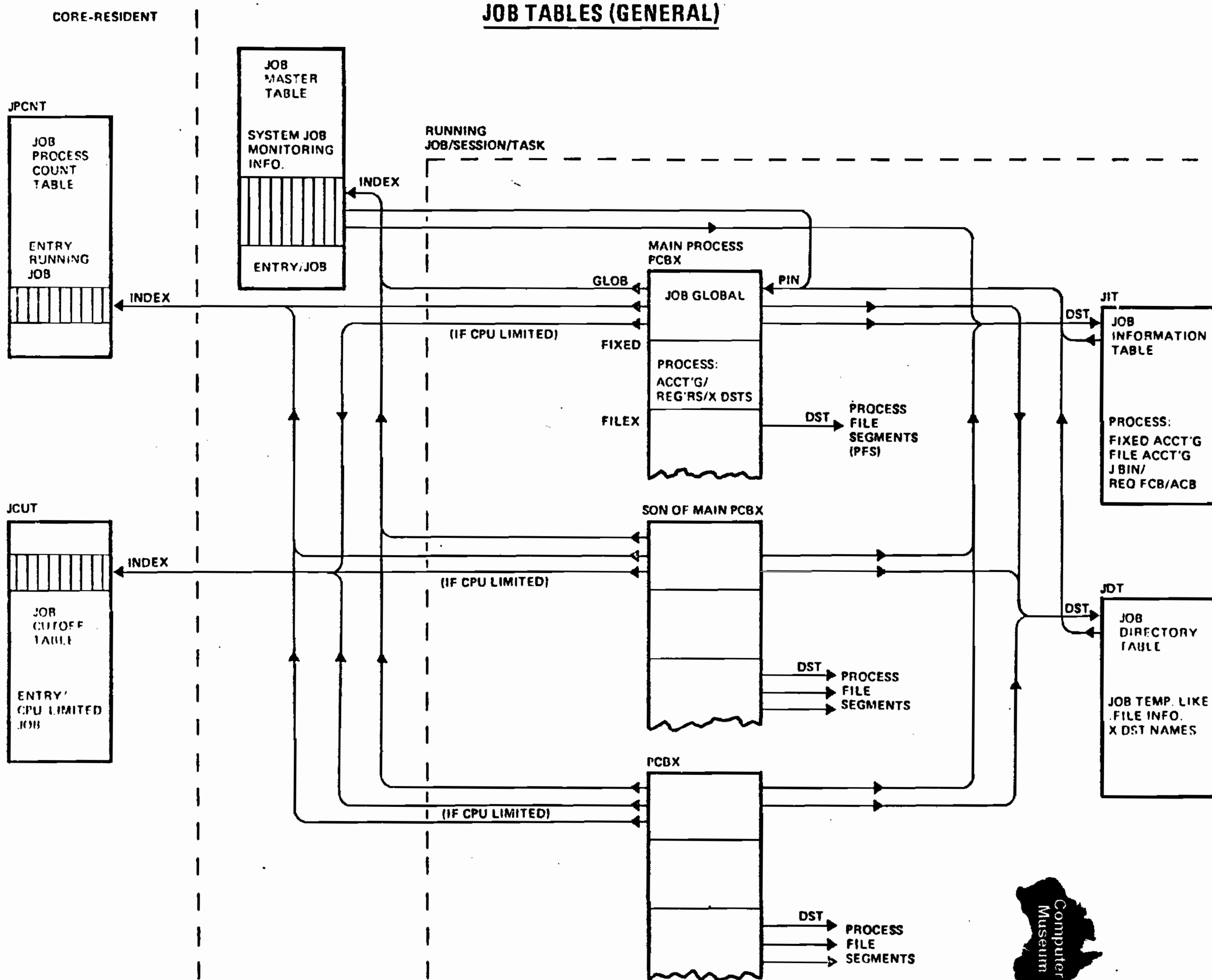


BINDER ORGANIZATION

MAJOR TABS	MINOR TAB DIVISIONS
CODE	CST XCSTE ~~LST~~ LOADER
DATA	DST DSTALLOCATION
DIRECTORY	DDS DSDS SYSTEM VDD – IDD – ODD
FILE	ACB FCB FILE LABEL FILE SEGMENT FMEAST USL PROGRAM SL RL
LOW FIXED CORE	LOW FIXED CORE
GENERAL LAYOUTS	MAIN MEMORY DISC BANK TABLE MEMORY LINKS WSTAB MTAB MRQ IOCS LDKQ DCSQ
INITIAL	DRIVER TABLE ~~CTAB~~ ~~CTABO~~ COMP DATA DISC LABEL DEFECTIVE TRACKS TABLE DISC COLD LOAD INFORMATION INITIAL PROGRAM CST MAP
INPUT/OUTPUT	DRT LPDT IDT DCT ~~IOQH~~ ILT DIT IOQE DLT ~~IOCB~~ TBUF SBUF
JOB TABLES	JMAT JPCNT JCUT JIT JDT
MISC	FLUT BREAKPOINT VTAB DISC FREE SPACE

PROCESS	PCB PCBX P TO P COMM ICS GLOBAL STACK
RESOURCE	SIR ~~CSIR~~ RIN
REQUEST QUEUES	UCOP MAKE PRESENT TRL
SYSTEM DB	SYSTEM DB
SYSTEM MESSAGES	MESSAGE CATALOG RIT MESSAGE STORAGE AREAS
DEVICE FILES	SPOOL TABLES

JOB TABLES (GENERAL)



(41) 10-1-74

JMAT – JOB MASTER TABLE STRUCTURE

DST = 25_{10}

SIR = 15_{10}

Bits: ∅ 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

Word (octal)	Contents	Word (decimal)	Description
∅	MAXSIZE / CURSIZE	∅	MAX JMAT SIZE (WORDS/128) CURRENT JMAT SIZE (WORDS/128)
1	(unused) / ENTRY SIZE	1	JMAT ENTRY SIZE (26)
2	ENTRY POINTER	2	DB POINTER TO FIRST ENTRY (26)
3	SCHEDULING HEAD POINTER	3	DB POINTER TO WORD ∅ OF HEAD ENTRY IN SCHEDULING QUEUE
4	∅1 SCHEDULING TAIL POINTER	4	DB POINTER TO WORD ∅ OF TAIL ENTRY IN SCHEDULING QUEUE
5	TY / SCOUNTER	5	NEXT ASSIGNABLE SESSION #, TY=1
6	TY / JCOUNTER	6	NEXT ASSIGNABLE BATCH #, TY=2
7	(unused) / JOB FENCE (bits 12 13 14 15)	7	DEFER JOB/SESSIONS WHOSE INPRI <= THIS VALUE
10	SLIMIT	8	MAXIMUM NUMBER SESSIONS
11	SNUM	9	CURRENT NUMBER SESSIONS
12	JLIMIT	10	MAXIMUM NUMBER BATCH JOBS
13	JNUM	11	CURRENT NUMBER BATCH JOBS
14–31	WORKAREA (14 WDS)	12–25	

Words 8–11: CURRENTLY EXECUTING

Words ∅–25: ZEROTH ENTRY

ENTRY 1

SCHEDULING QUEUE
- WAITING SESSIONS
  - FIFO WITHIN HIPRI/INPUT PRIORITY
- ERROR JOBS
  - FIFO
- WAITING JOBS
  - FIFO WITHIN HIPRI/INPUT PRIORITY

LAST ENTRY

(4)

JMAT – JOB MASTER TABLE LEGEND

ENTRY TYPES [(ENTRY WORD Ø - (Ø : 6)]

Ø	UNUSED ENTRY
1	JOB INTRODUCED*
%40	JOB WAITING
%60	JOB BEING INITIALIZED (CI)
2	JOB EXECUTING
3	MP TERMINATED OR TERMINATING

DEFINITIONS

Ø D – JIN/JLIST DUPLICATIVE
I – JIN/JLIST INTERACTIVE
G – GROUP
A – ACCOUNT 1 => PASSWORD NOT VALIDATED
U – USER
C – ORIGJLIST IS CLASS INDEX
Q – Ø => JOB NOT QUIET MESSAGES
1 => JOB IS QUIET
INPRI – INPUT PRIORITY OF JOB

1 TY – TYPE
1 = SESSION
2 = JOB
JNUMBER – JOB NUMBER

18 JIN – CURRENT JOB INPUT DEVICE
JLIST – CURRENT JOB LIST DEVICE

19 STARTTIME – STARTING TIME STAMP (CHRONOS)

22 XPRI – EXECUTING PRIORITY REQUESTED
MAIN PIN – MAIN PIN

23 TIMELIMIT – JOB CPU TIME LIMIT IN SECONDS
Ø => OMITTED, DEFAULT
1 => UNLIMITED

24 SPOOLED – ORIGJIN IS SPOOLED
RESTART – RESTART IS REQUESTED
P – XPRI REQUESTED WAS NOT GRANTED MAX PERMISSIBLE USED
OUTPRI – OUTPUT SPOOL PRIORITY
NUMCOPIES – NUMBER OF COPIES TO BE OUTPUT
F – NON STANDARD FORMS CONTROL ON DEVICE

25 SCHEDLINK – DB POINTER TO WORD Ø OF NEXT ENTRY ON SCHEDULING QUEUE;
Ø => LAST

25 ORIGJIN – ORIGINAL JOB INPUT DEVICE
ORIGJLIST – REQUESTED JOB LIST DEVICE

(4)

10-1-74

JMAT – JOB MASTER TABLE ENTRY

(State %1 – Introduced)

Bits: Ø, 1, 2, 3, 4, 5, 6, 7, 8, 9, 1Ø, 11, 12, 13, 14, 15

Word (octal)	Contents	Word (decimal)
Ø	%1 (bits Ø–5) | D | I | G | A | U | C | INPRI	Ø
1	TY | JNUMBER	1
2, 3, 4, 5	USER NAME	2, 3, 4, 5
6, 7, 1Ø, 11	ACCOUNT NAME	6, 7, 8, 9
12, 13, 14, 15	JOB NAME	1Ø, 11, 12, 13
16, 17, 20, 21	GROUP LOG-ON NAME	14, 15, 16, 17
22	J IN | J LIST	18
23, 24, 25	STARTING TIME STAMP (CHRONOS)	19, 20, 21
26	(unused) | XPRI	22
27	JOB TIME LIMIT (CPU - SEC)	23
30	S | R | P | (unused) | 5 OUTPRI 8 | 9 NUMCOPIES 15	24

C = DEVICE CLASS-SPECIFIED

U = USER
A = ACCT
G = GROUP

Ø ⇒ PASSWORD VALIDATED

1 ⇒ NO PASS WORD SUPPLIED, IE, C.I. MUST CHECK FOR NO PASS REQ'D OR PROMPT USER

I = INTERACTIVE

D = DUPLICATIVE

JIN – CURRENT JOB INPUT DEVICE

JLIST – CURRENT JOB LIST DEVICE

EXECUTING PRIORITY REQUESTED

Word 23: [Ø IF OMITTED, -1 IF UNLIMITED]

SPOOLED – ORIGJIN IS SPOOLED

RESTART – REQUESTED

P – XPRI REQUESTED NOT GRANTED, MAX ALLOWED USED

OUTPRI – OUTPUT SPOOL PRIORITY

NUMCOPIES – NUMBER OF COPIES OUTPUT

JMAT –JOB MASTER TABLE ENTRY

(State %40 – Job Waiting)

Bits: ∅ 1 2 3 4 5 6 7 8 9 1∅ 11 12 13 14 15

Word	Contents	Word
∅	%40 (bits ∅–5) | D | I | G | A | U | C | INPRI (bits 12–15)	∅
1	TY | JNUMBER	1
2–5	USER NAME	2–5
6–11	ACCOUNT NAME	6–9
12–15	JOB NAME	1∅–13
16–21	GROUP LOG-ON NAME	14–17
22	JIN | J LIST	18
23–25	STARTING TIME STAMP (CHRONOS)	19–21
26	(unused) | XPRI	22
27	JOB TIME LIMIT (CPU-SEC)	23
30	S | R | P | (unused) | 5 OUTPRI 8 | 9 NUMCOPIES 15	24
31	SCHEDLINK	25

JOB TIME LIMIT: ∅ IF OMITTED; -1 IF UNLIMITED

C = DEVICE CLASS SPECIFIED

U = USER
A = ACCT
G = GROUP

∅ ⇒ PASSWORD VALIDATED

1 ⇒ NO PASSWORD SUPPLIED, I.E., C.I.MUST CHECK FOR NO PASS REQ'D OR PROMPT USER.

I = INTERACTIVE

D = DUPLICITIVE

SCHEDLINK – DB POINTER TO WORD ∅ OF NEXT ENTRY ON SCHEDULING QUEUE. ∅ => LAST

(41)

10-1-74

JMAT – JOB MASTER TABLE ENTRY

(State % 60- Job Being Initialized) (By C. I.)

Word	Bits 0–15	Word
0	% 60 (bits 0–5) | D | I | G | A | U | C | INPRI (bits 12–15)	0
1	TY | JNUMBER	1
2, 3, 4, 5	USER NAME	2, 3, 4, 5
6, 7, 10, 11	ACCOUNT NAME	6, 7, 8, 9
12, 13, 14, 15	JOB NAME	10, 11, 12, 13
16, 17, 20, 21	GROUP LOG-ON NAME	14, 15, 16, 17
22	JIN | JLIST	18
23, 24, 25	STARTING TIME STAMP (CHRONOS)	19, 20, 21
26	MAIN PIN |	22
27	JOB TIME LIMIT (CPU-SEC)	23
30	S | R | | F | 3 4 (shaded) | 5 OUTPRI 8 | 9 NUMCOPIES 15	24
31	ORIGJIN 7 | 8 ORIGJLIST	25

C = DEVICE CLASS SPECIFIED

U = USER, A = ACCT, G = GROUP: Ø ⇒ PASSWORD VALIDATED; 1 ⇒ NO PASS WORD SUPPLIED, IE, C.I. MUST CHECK FOR NO PASS REQ'D OR PROMPT USER.

I = INTERACTIVE

D = DUPLICATIVE

Word 23: [Ø IF OMITTED; -1 IF UNLIMITED]

F = NON STANDARD FORMS CONTROL ON DEVICE

ORIGJIN – ORIGINAL JIN DEVICE

ORIGJLIST – REQUESTED JLIST DEVICE

JMAT – JOB MASTER TABLE ENTRY

(State %2 - Job Running)

(State %3 – Job Terminating)

Bits: Ø 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

Word (octal)	Bits / Fields	Word (decimal)
Ø	%2 (bits 0–5) | D (6) | I (7) | Q (8) | shaded (9–10) | C (11) | INPRI (12–15)	Ø
1	TY | JNUMBER	1
2–5	USER NAME	2–5
6, 7, 10, 11	ACCOUNT NAME	6–9
12–15	JOB NAME	10–13
16, 17, 20, 21	GROUP LOG-ON NAME	14–17
22	JIN | JLIST	18
23–25	STARTING TIME STAMP (CHRONOS)	19–21
26	MAIN PIN | XPRI	22
27	JOB TIME LIMIT (CPU-SEC)	23
30	S | R | P | shaded | 5 – 8 | 9 NUMCOPIES 15	24
31	ORIGJIN (0–7) | 8 ORIGJLIST	25

QUIET BIT
1 ⇒ DON'T PERMIT =TELL OR :TELL TO THIS JOB.

I = INTERACTIVE
D = DUPLICATIVE
C = ORIGJLIST IS CLASS INDEX

JMAT – JOB MASTER TABLE ENTRY

(State % 50 - Error)

Word	Bits 0–5	Bit 6	Bit 7	Bits 8–15	Word
0	% 50	J/H	C	J LIST	0
1	PARM	ERRNUM			1
2					2
3					3
4	1ST 48 CHARACTERS OF COMMAND IMAGE				4
5					5
6					6
7					7
10					8
11					9
12					10
13					11
14					12
15					13
16					14
17					15
20					16
21					17
22					18
23					19
24					20
25					21
26					22
27					23
30					24
31					25

J/H 0 ⇒ JOB
1 ⇒ HELLO

FOR "ERR N, P"

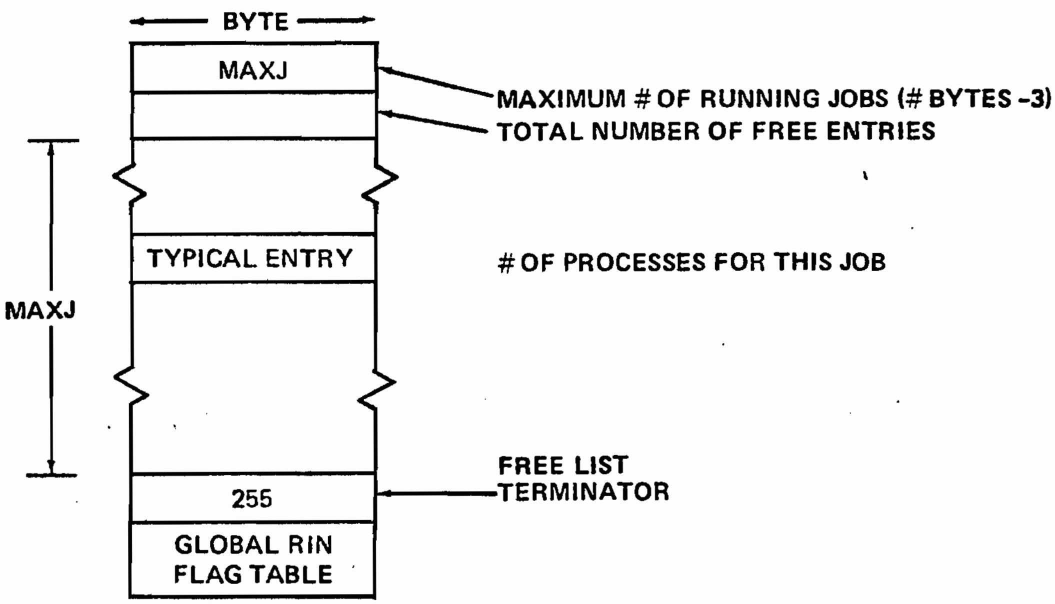
JLIST IS CLASS INDEX

JPCNT – JOB PROCESS COUNT TABLE

(1 Entry/Running Job)

CORE RESIDENT

SYSGLOB BASE = DB +13
DST = 24_{10}
SIR = 13_{10}



A JPCNT ENTRY MUST BE ALLOCATED BEFORE THE MAIN PROCESS CAN BE PROCREATED.

THE JOB SIR (PXGJSIR) = SOME BASE + JPCNT INDEX.

NOTE: THIS TABLE IS COMPLETELY BYTE ORIENTED WITH EACH ENTRY MADE CONSISTING OF ONE BYTE. ENTRIES ARE TAKEN FROM AVAILABLE POOL ON A "FIRST FOUND" BASIS. 254 (376 OCTAL) IN A BYTE DENOTES A FREE ENTRY. 255(377 OCTAL) DENOTES THE LAST FREE ENTRY.

GLOBAL RIN FLAG TABLE

THIS TABLE IS A BIT TABLE WHICH IMMEDIATELY FOLLOWS THE "FREE LIST TERMINATOR" BYTE. IT IS INITIALIZED TO Ø AND IS INDEXED BY JPCNT INDEX. FOR EACH JOB.

JCUT – JOB CUTOFF TABLE

1 Entry/CPU - Limited Job

CORE RESIDENT

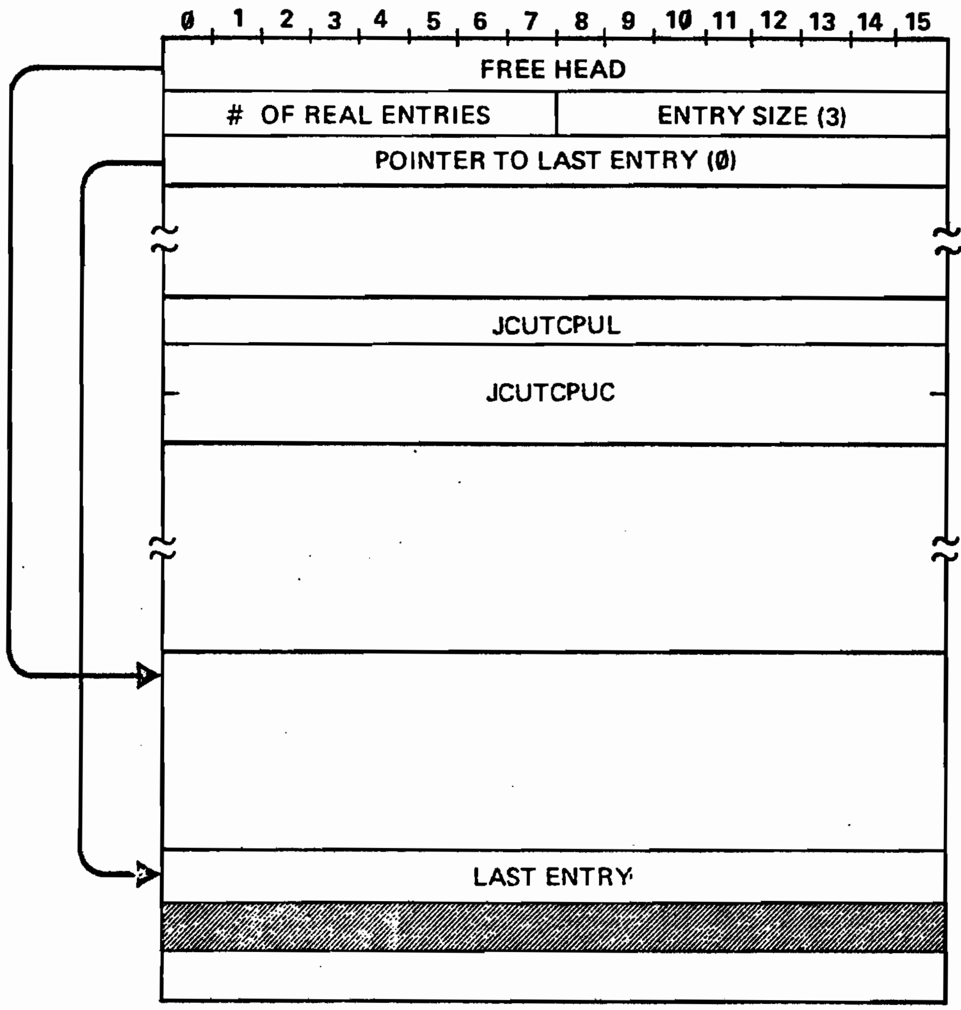
DST = 36_{10}

SIR = 14_{10}

DEFAULT CPU TIME LIMIT IN

SYSGLOB +52_{10}

(-1 ⇒ UNLIMITED)



TYPICAL ENTRY

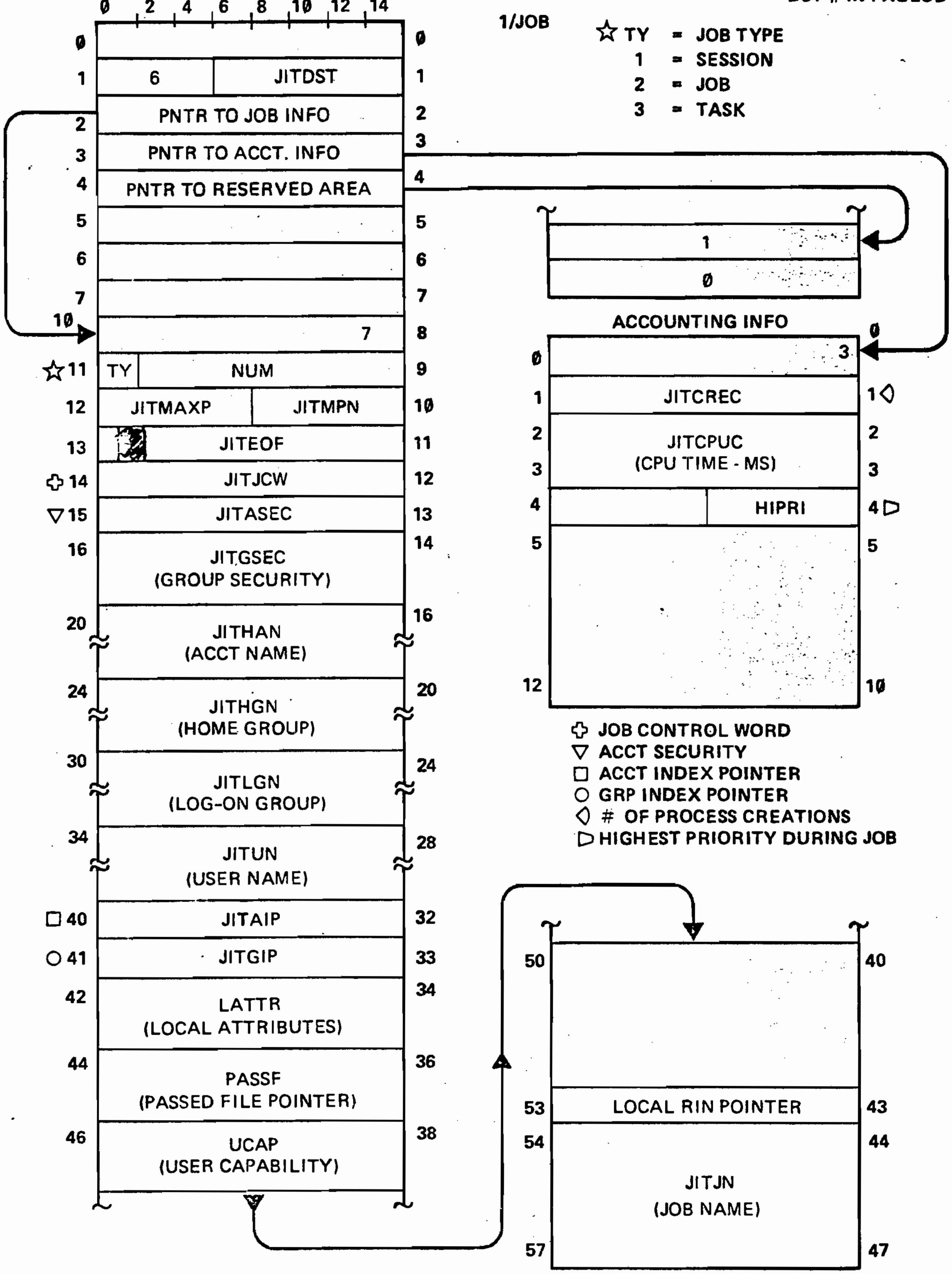
TIME LIMIT (SECONDS)

TIME COUNT (MSEC)

10-1-74

JIT – JOB INFORMATION TABLE

DST # IN PXGLOB

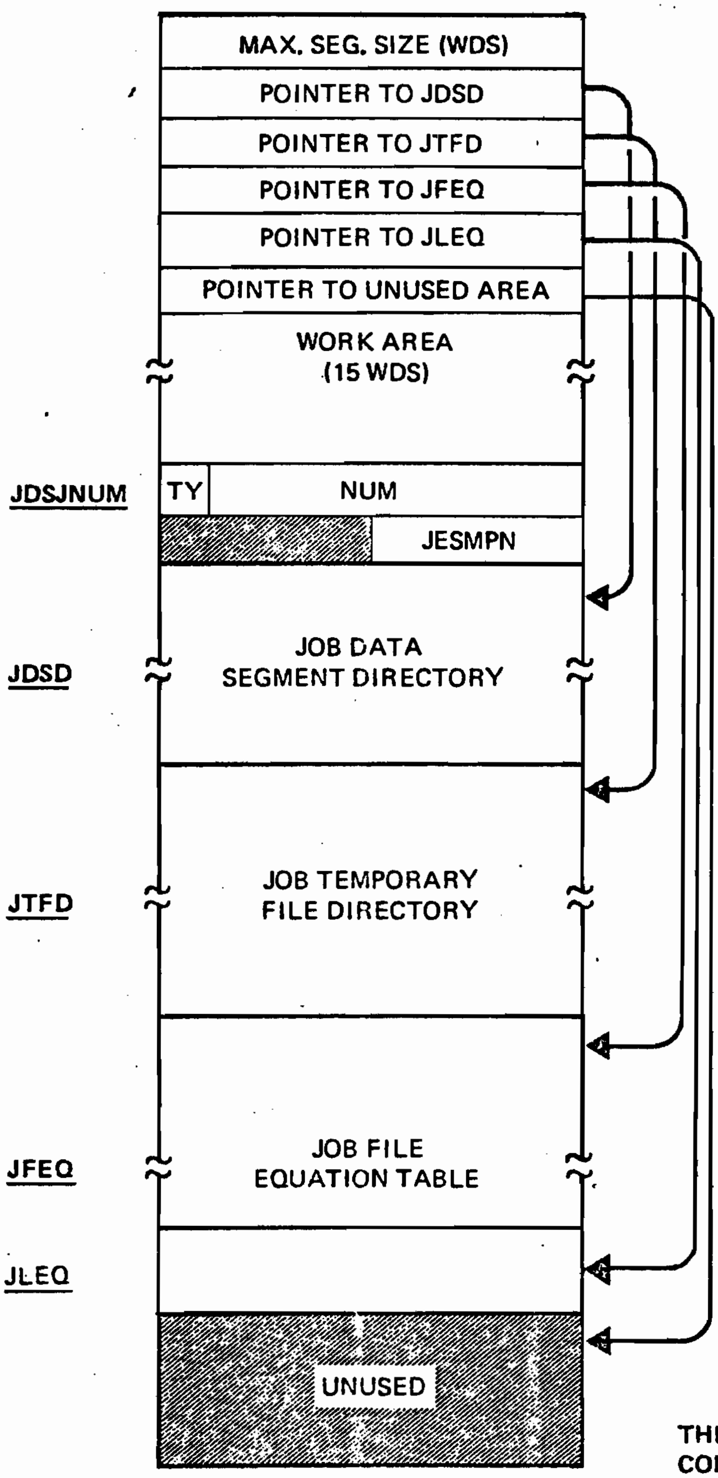


(TS)

10/1/74

JDT – JOB DIRECTORY TABLE

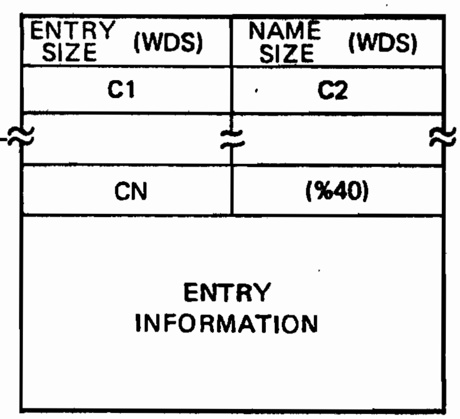
1 ENTRY PER JOB
DST # IN PXGLOB



JOB NUMBER

MAIN PROCESS NUMBER

JDSD/
JTFD/JFEQ ENTRY FORMAT



THE NAME IS A CONCATENATION OF UP TO 3 SUBNAMES. BIT Ø OF THE 1ST CHARACTER OF EACH SUBNAME IS 1.

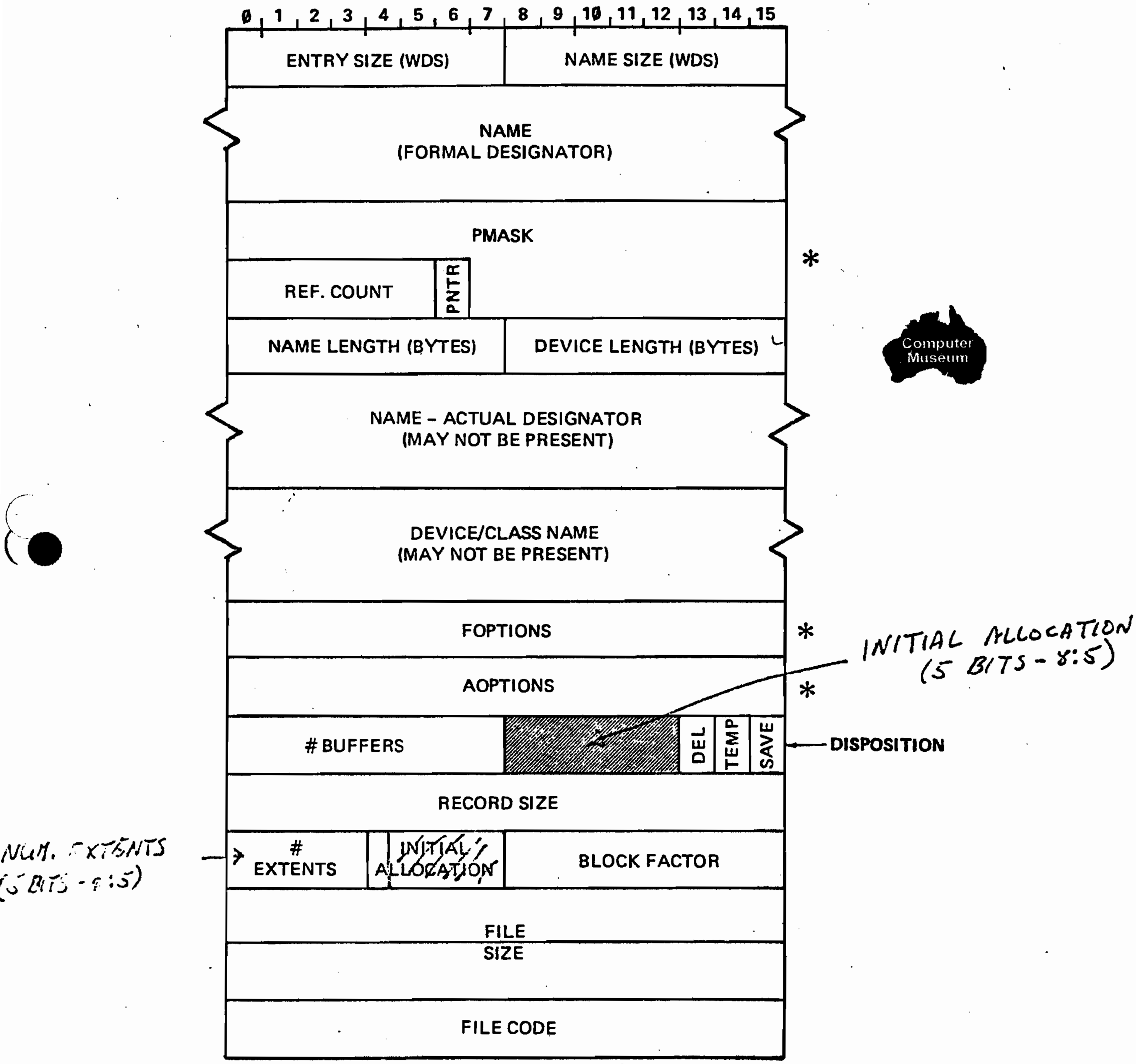
JOB DATA SEGMENT DIRECTORY ENTRY – (IN - JDT)

Ø 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15	
ENTRY SIZE (WORDS)	NAME SIZE (WORDS)	Ø
"NAME"		1
EXTRA DATA SEGMENT DST INDEX		2
# OF PROCESSES ACCESSING		3

JOB TEMPORARY FILE ENTRY – (IN JDT)

Ø 1 2 3 4 5 6 7	8 9 10 11 12 13 14 15
ENTRY SIZE (WDS)	NAME SIZE (WDS)
NAME – ACTUAL FILE DESIGNATOR	
VOLUME POINTER	
FILE LABEL POINTER	

FILE EQUATION TABLE ENTRY – (IN JDT)



*SEE NEXT PAGE

AOPTIONS, FOPTIONS, AND PMASK WORD BREAKDOWN

OPTION WORD 2 (AOPTIONS)

Bit	Value	Field
0	0	
1	0	
2	0	
3	0	
4	0	
5	0	
6	0	
7		INHIBIT BUFF.
8–9		EXCLUSIVE
10	0	
11		MULTI-RECORD
12–15		ACCESS TYPE

OPTION WORD 1 (FOPTIONS)

Bit	Value	Field
0	0	
1	0	
2	0	
3	0	
4	0	
5	0	
6		NO LABEL
7		CARRIAGE CONTROL
8–9		RECORD FORMAT
10–12		DEFAULT DESIGNATION
13		ASCII/BINARY
14–15		DOMAIN

PMASK

Row	Left field	Left value	Right field
0	REF COUNT		BLOCKFACTOR
1	REF COUNT		RECSIZE
2	REF COUNT		DISPOSITION
3	REF COUNT		NUMBUFFERS
4	REF COUNT		INHIBIT BUFFERING
5	REF COUNT		EXCLUSIVE
6	POINTER ENTRY		MULTI-RECORD
7	WAIT, NOWAIT	0	ACCESS TYPE
8	MULTI	0	NO LABEL
9	ACCESS = 2	0	CARRIAGE CONTROL
10	NUMBER	0	RECORD FORMAT
11	OUTPRI. = 2	0	DEFAULT DESIGNATION
12	FILECODE		ASCII/BINARY
13	FILESIZE		DOMAIN
14	NUMEXTS		DEVICE
15	INIT ALLOC		NAME

6/9/75
L.B.

1 ⇒ INFO PRESENT

0 ⇒ INFO ABSENT

APPENDIX C
Summary of Intrinsic Calls

All intrinsic calls available to the user are summarized below, listed alphabetically. For each intrinsic, the complete declaration head appears; the intrinsic call format is distinguished from the remainder of the head by a box. The function of the intrinsic is described. For those intrinsics that are type procedures, the procedure *type* is noted. Optional parameters are bold face in the intrinsic head and are also noted separately. All condition codes that can be returned by the intrinsic are listed. The intrinsic error-code number is also presented. (This is the number that appears in the abort-error message generated when an error is encountered in the corresponding intrinsic.) The functional categories represented by the error number set are

Error Number Range			Functional Category
1 – 29			File Management
30 – 39			Resource Management
40 – 49			System Timer (Clock)
50 – 59			Traps
60 – 79			Utility Routines
80 – 99			Program Management
100 – 119			Process Control
120 – 129			Scheduling
130 – 149	160 – 179	DS	Data Segments
180 – 199			Input/Output Utilities
200 – 209			Special Utilities

Where intrinsics have special attributes, those attributes are noted as follows:

- Uncallable intrinsics (those that can only be invoked by the user in privileged mode) are indicated by an asterisk (*).
- Intrinsics that can be called in privileged mode even though the user does not have the appropriate capability are denoted by a plus sign (+).
- Intrinsics that can be called by a process when the DB register is not pointing to that process' stack are denoted by a dollar sign ($).

The MPE/3000 Optional Capability (if any) required to invoke the intrinsic, and the page in the main text where the intrinsic is discussed, are also noted.

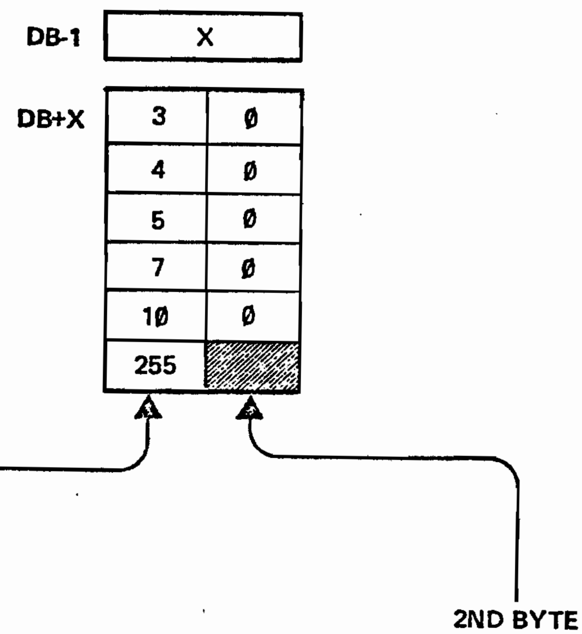
```
00000 2    12   0   INVALID ASCII DIGIT
00000 2    13   0   INVALID DECIMAL DIGIT
00000 2    14   0   INVALID WORD COUNT
00000 2    15   0   INVALID DECIMAL OPERAND LENGTH
00000 2    16   0   DECIMAL DIVIDE|BY|ZERO
00000 2    17   0   STT UNCALLABLE
00000 2    18   0   PROCESS QUIT
00000 2    19   0   PROGRAM QUIT
00000 2    20   0   STACK OVERFLOW
00000 2    21   0   PROGRAM KILLED
00000 2    22   0   INVALID STACK MARKER
00000 2    23   0   ADDRESS VIOLATION
00000 2    24   0   BOUNDS VIOLATION
00000 2    25   0   NON-RESPONDING MODULE
00000 2    26   0   DATA PARITY
00000 2    27   0   MEMORY PARITY
00000 2    28   0   SYSTEM PARITY
00000 2    29   0   STACK UNDERFLOW
00000 2    30   0   CST VIOLATION
00000 2    31   0   STT VIOLATION
00000 2
00000 2  TABLE 3: INTRINSIC
00000 2
00000 2     1   5   FOPEN
00000 2     2   5   FREAD
00000 2     3   5   FWRITE
00000 2     4   5   FUPDATE
00000 2     5   5   FSPACE
00000 2     6   5   FPOINT
00000 2     7   5   FREADDIR
00000 2     8   5   FCLOSE
00000 2
00000 2    10   5   FCHECK
00000 2    11   5   FGETINFO
00000 2    12   5   FREADSEEK
00000 2    13   5   FCONTROL
00000 2    14   5   FSETMODE
00000 2    15   5   FLOCK
00000 2    16   5   FUNLOCK
00000 2    17   5   FRENAME
00000 2    18   5   FRELATE
00000 2    19   5   FREADLABEL
00000 2    20   5   FWRITELABEL
00000 2    21   0   PRINTFILEINFO
00000 2    22   0   IOWAIT
00000 2
00000 2    30   0   GETLOCRIN
00000 2    31   0   FREELOCRIN
00000 2    32   0   LOCKLOCRIN
00000 2    33   0   UNLOCKLOCRIN
00000 2    34  11   LOCKGLORIN
00000 2    35   0   UNLOCKGLORIN
00000 2    40   0   TIMER
00000 2    41   0   CHRONOS
00000 2    42   0   PROCTIME
00000 2    43   0   CALENDAR
00000 2    44   0   CLOCK
```

01063000	00000	2	45	0	PAUSE
01064000	00000	2	50	0	XARITRAP
01065000	00000	2	51	0	ARITRAP
01066000	00000	2	52	0	XLIBTRAP
01067000	00000	2	53	0	XSYSTRAP
01068000	00000	2	54	0	XCONTRAP
01069000	00000	2	55	0	RESETCONTROL
01070000	00000	2	56	0	CAUSEBREAK
01071000	00000	2	60	0	TERMINATE
01072000	00000	2	62	0	BINARY
01073000	00000	2	61	0	CTRANSLATE
01074000	00000	2	63	0	ASCII
01075000	00000	2	64	0	READ,READX
01076000	00000	2	65	0	PRINT
01077000	00000	2	66	0	PRINTOP
01078000	00000	2	67	0	PRINTOPREPLY
01079000	00000	2	68	0	COMMAND
01080000	00000	2	69	0	WHO
01081000	00000	2	70	0	SEARCH
01082000	00000	2	71	10	MYCOMMAND
01083000	00000	2	72	0	SETJCW
01084000	00000	2	73	0	GETJCW
01085000	00000	2	74	0	DBINARY
01086000	00000	2	75	0	DASCII
01087000	00000	2	76	0	QUIT
01088000	00000	2	77	0	STACKDUMP
01089000	00000	2	78	0	SETDUMP
01090000	00000	2	79	0	RESETDUMP
01091000	00000	2	80	6	LOADPROC
01092000	00000	2	81	6	UNLOADPROC
01093000	00000	2	82	0	INITUSLF
01094000	00000	2	83	0	ADJUSTSLF
01095000	00000	2	84	0	EXPANDUSLF
01096000	00000	2	99	0	DEBUG
01097000	00000	2	100	7	CREATE
01098000	00000	2	102	0	KILL
01099000	00000	2	103	9	SUSPEND
01100000	00000	2	104	6	ACTIVATE
01101000	00000	2	105	0	GETORIGIN
01102000	00000	2	106	0	MAIL
01103000	00000	2	107	0	SENDMAIL
01104000	00000	2	108	0	RECEIVEMAIL
01105000	00000	2	109	0	FATHER
01106000	00000	2	110	0	GETPROCINFO
01107000	00000	2	112	0	GETPROCID
01108000	00000	2	120	7	GETPRIORITY
01109000	00000	2	130	0	GETDSEG
01110000	00000	2	131	0	FREEDSEG
01111000	00000	2	132	0	DMOVIN
01112000	00000	2	133	0	DMOVOUT
01113000	00000	2	134	0	ALTDSEG
01114000	00000	2	135	0	DLSIZE
01115000	00000	2	136	0	ZSIZE
01116000	00000	2	139	0	SWITCHDB
01117000	00000	2	[illegible]	0	PTAPE
01118000	00000	2	200	0	[illegible]
01119000	00000	2	201	0	[illegible]

FORTRAN LOGICAL UNIT TABLE (FLUT)

THE SEGMENTER WILL BE RESPONSIBLE FOR THE PREPARATION AND INITIALIZATION OF A FORTRAN LOGICAL UNIT TABLE. THIS WILL BE DONE WHEN A PROGRAM IS PREPARED AND WHEN THAT PROGRAM CONTAINS AT LEAST ONE PROGRAM UNIT THAT REFERENCES A LOGICAL UNIT. THE LOCATION OF THE FLUT WILL BE IN THE SECONDARY DB AREA AND THIS LOCATION WILL BE CONTAINED IN DB-1.

THE FLUT WILL BE FORMATTED AS PER THE FOLLOWING EXAMPLE:



LIST OF THE LOGICAL UNIT NUMBERS REFERRED TO IN THIS FORTRAN-PRODUCED PROGRAM. (255 TERMINATES).

THE MPE FILE NUMBER (AS RETURNED BY FOPEN) USED IN ACCESSING THE FILE. Ø IF FILE NOT OPEN. FILLED IN BY FORMATTER AS EACH L.U. IS INITIALLY REFERENCED.

BREAKPOINT TABLE

THIS TABLE IS ALLOCATED PERMANENT STORAGE IN NON-LINKED MEMORY AND IS USED BY SYSTEMDEBUG IN RELATION TO "BREAKPOINT" PROCESSING. EACH ENTRY IS 4 WORDS LONG. ENTRIES ARE ARRANGED IN SEQUENTIAL ORDER WITH THE FIRST ENTRY LOCATED STARTING WITH FIRST WORD OF THE TABLE (I.E. THERE IS NO SPECIAL "ZEROTH ENTRY" OR "LINKED LIST OF FREE ENTRIES" AS IN MOST OTHER MPE TABLES).

ENTRY FORMAT

Word	Bit 0	Bits 1–15
0	S/D	PHYSICAL PCB #
1	PHYSICAL CST #	
2	P-PB	
3	SAVED INSTRUCTION	

blk label format

See MTAB for explanation

S/D = 0 IF BREAKPOINT SET FROM SYSTEMDEBUG
= 1 IF BREAKPOINT SET FROM DEBUG

VOLUME TABLE

SIR #22 = 26_8
DST #29 = 35_8

ZERO ENTRY

WORD	Bits 0–7	Bits 8–15	
0	# OF ENTRIES (NOT COUNTING ZERO)	ENTRY SIZE = 16_8	0
1	COLD LOAD ID		1
⋮			⋮
15			13

TYPICAL ENTRY

INDEXED BY VOLUME #

WORD	Bits 0–7	Bits 8–15		
0	VOLUME NAME		0	
1			1	
2			2	
3			3	
4	GROUP NAME		4	ZEROES FOR SYSTEM VOLUMES
5			5	
6			6	
7			7	
10	ACCOUNT NAME		8	
11			9	
12			10	
13			11	
14	LOGICAL DEVICE # (= 0 IF NOT MOUNTED)		12	
15	DIRECTORY ADDRESS (= 0 FOR SYSTEM VOLUMES)		13	

DISK FREE SPACE TABLE GENERAL DIRECTION

30_{10}

THERE IS ONE DISC-FREE-SPACE TABLE FOR EACH DISC IN THE SYSTEM. THE TABLE BEGINS AT SECTOR # 20_{10} AND ITS SIZE IS SPECIFIED WITHIN THE TABLE ITSELF. THE TABLE IS READ ONE PAGE AT A TIME, AS NEEDED, INTO AN EXTRA DATA SEGMENT (DST # 17_{10}). A PAGE CONSISTS OF ONE OR MORE SECTORS AND IS CURRENTLY EQUAL TO 2 SECTORS. THE TABLE IS KEPT PACKED AT ALL TIMES AND DISC ADDRESSES ARE IN ASCENDING ORDER. ANY TWO ENTRIES REPRESENT TWO DISJOINT AREAS ON THE DISC.

FORMAT OF PAGE # 1

HEADER
ENTRY # 1
ENTRY # 2
⋮
ENTRY # 62_{10}

FORMAT OF SUCCEEDING PAGES

ENTRY # η
ENTRY # $(\eta + 1)$
⋮
ENTRY # $(\eta + 63)$

HEADER FORMAT (8 WORDS)

WORD INDEX		DOUBLEWORD INDEX	
Ø	# ENTRIES IN TABLE	Ø	
1	TABLE SIZE (SECTORS)		
2, 3	LOWER BOUND FOR FREE SPACE (SECTOR #)	1	MIN
4, 5	UPPER BOUND FOR FREE SPACE (SECTOR #)	2	MAX
6, 7		3	

MIN & MAX ARE DISC ADDRESSES INDICATING THE MINIMUM AND MAXIMUM DISC ADDRESSES ACCOUNTABLE BY DISCSPC (I.E. NO SPACE WHICH GOES OUTSIDE THESE BOUNDS MAY BE REQUESTED OR RETURNED).

DISK FREE SPACE (Con't)

ENTRY FORMAT (4 WORDS/ENTRY)

WORD INDEX		DOUBLEWORD INDEX
ø	SECTOR NUMBER WHERE SPACE STARTS	ø
1		
2	# SECTORS AVAILABLE	1
3		

1/8/74

PCB table, entries Ø, 1, and 2

		Name	Initial
Ø	# real entries		
	entry SIZE		16
	# free entries		
	free list head		
	Quantum	Quantum	Ø
5	TPRI	Tpri	Sysdump
	CPRI	Cpri	
	DPRI	Dpri	
	MAM PCB WORD INDEX	Mampcb	
	POPT		
10			
	Ø	default wsp must be Ø	
15			

ENTRY 1, READY LIST, RL

		Name	Initial
16	1 0 Ø HEAD LINK	RLH	%10ØØ65
	Ø	dummy status	Ø
		LQ set	%2ØØØØ
	ØØØ TAIL LINK	RLT	%1ØØØ8Ø

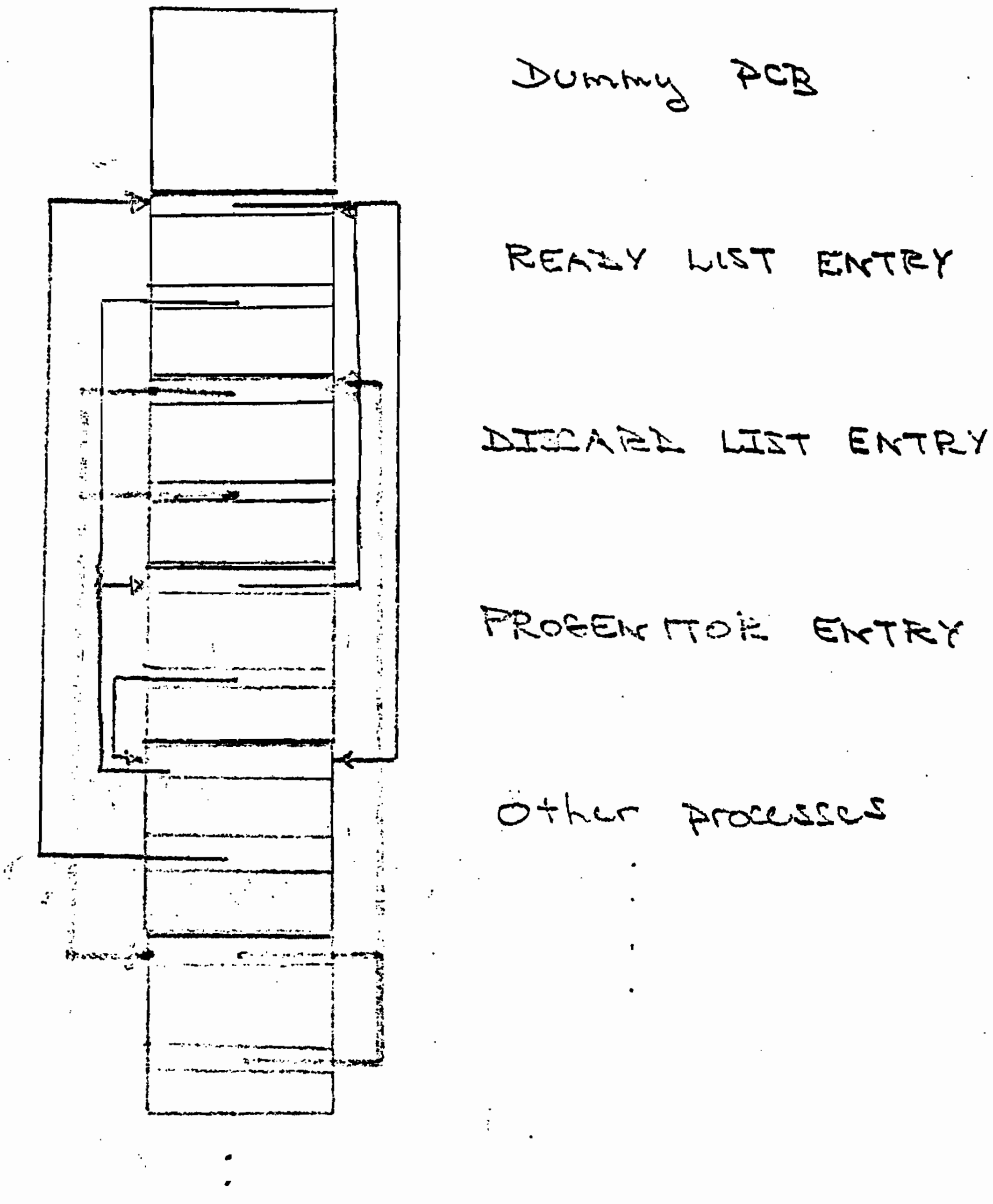
12/19/73

ENTRY 2, DISCARD LIST, DL

32	000	HEAD LINK	DLH	32
	000	TAIL LINK	DLT	32

Entry 3 will be the PROGENITOR's PCB. It will be linked into the RL (links to 16) by INITIAL.

General layout of PCB table showing READY and DISCARD lists.



- - DISCARD list

— READY LIST

Both lists are circular, doubly linked, with the pointers PCBB relative and pointing to the first word of an entry.

12/19/73

PROCESS CONTROL BLOCK - TABLE ENTRY

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
PCBØØ	LT	Ø	Ø	FQP												
PCBØ1	SW	ODP										MSTATE				
PCBØ2	ADB	XDS												MEF	OA	
PCBØ3	W W_S	STK										SC	PFC			OVR
PCBØ4	M	RG	RL	MA	BIO	IO	UCP	JNK	TIM	INT	SON	FA	ACT	IMP	SIR	CR
PCBØ5	FPTR								SPTR							
PCBØ6	BPTR								PRI							
PCBØ7	PCST								BPTLINK							
PCBØ8	Ø	E	LQ	C	D	NL	DEAD	FAC	IQPTR							
PCBØ9	LIV	BMS		PPC		ST_{OV}	PTYPE				HK	SK	ST	HB	CY	BK
PCB1Ø	PSIM			BQP												
PCB11	TLQ															
PCB12	PBX															
PCB13	LAT															
PCB14	WSP															
PCB15	PRINX															

PCB? = table name, system table 2

PCBSIZE = entry size, 16

PCBB = table base

PROCESS CONTROL BLOCK ENTRY

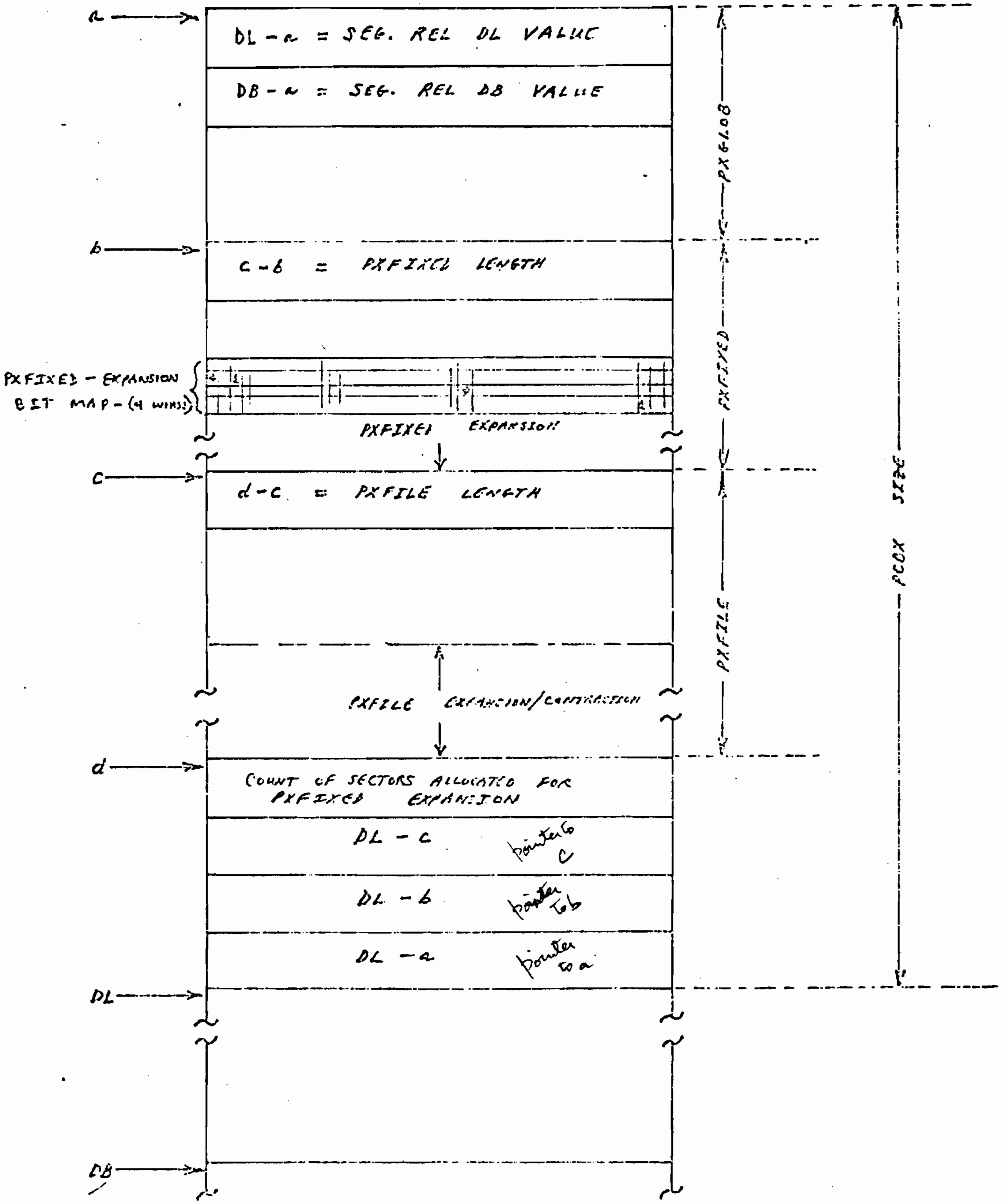
```
PCB     =  TABLE NAME,  SYSTEM TABLE 2

PCBSIZE =  ENTRY SIZE,   16

PCBB    =  TABLE BASE


CERTAIN FIELDS BELONG TO CLASSES:
    OXDS, XDS, STK ARE DST, DATA SEGMENT INDICES.
    FQP, BQP ARE PLINK, PROCESS LINKS.
    M - FA ARE COLLECTIVLY KNOWN AS WAIT.
    M - ACT ARE KNOWN AS WAITA.
    IMP - CR ARE KNOWN AS ICS.
    HK - BK ARE KNOWN AS PSIF, PSEUDO INTERRUPT FIELDS.



PCB00 .(0:1)     LT       LIST TYPE FOR WHICH LIST PCB IS LINKED ONTO.
                          0:   DISCARD OR NO LIST
                          1:   READY
PCB00 .(3:13)    FQP      FORWARD QUEUE POINTER, PCBB RELATIVE.
                          < > 0: ON A LIST.
                           =   0: NOT ON A LIST.

PCB01 .(0:1)     SW       SET IF PROCESS IS ON A SHORT WAIT.
PCB01 .(1:10)    ODP      POINTER TO MAM OLD DISCARD LIST.
PCB01 .(11:5)    MSTATE   MEMORY RESOURCE STATE.
                          0 :  CORE RESIDENT
                          1 :  OUT OF MAIN MEMORY
                          2 :  IN CORE
                          3 :  ABSENT SEGMENT.

PCB02 .(0:1)     ADB      SET IF DB IS POINTING TO AN ABSOLUT ADDRESS.
PCB02 .(1:10)    XDS      DST INDEX FOR EXTRA DATA SEGMENT.
                          0   IF NONE.
PCB02 .(11:2)                 UNUSED.
PCB02 .(13:1)             SET ON MAM ERRORS FOR MAKEPRESENT.
PCB02 .(14:2)    OA       ACTIVITATION ORIGIN FOR THE PROCESS.
                          0   OTHER REASON
                          1   FATHER
                          2   SON

PCB03 .(0:1)     WWS      WAKE UP WAITING SWITCH.
                          1   IF AWAKE IS MISSING.
PCB03 .(1:10)    STK      DST INDEX FOR STACK
PCB03 .(11:1)    SC       SET IF EXECUTING SYSTEM CODE
PCB03 .(12:3)    PFC      COUNTER FOR NUMBER OR SUCCESSIVE PREPARATION
                          FAILURES.
PCB03 .(15:1)    OVR      OVER-RUN BIT FOR PSEUDO INTERRUPT PROCESSING.

PCB04 .(0:1)     M        MOURNING WAIT
PCB04 .(1:1)     RG       GLOBAL RIN WAIT
PCB04 .(2:1)     RL       LOCAL RIN WAIT
PCB04 .(3:1)     MA       MAIL WAIT
PCB04 .(4:1)     BIO      BLOCKED I/O WAIT
```

```
PCB04 .(5:1)      IO        I/O WAIT
PCB04 .(6:1)      UCP       UCOP WAIT
PCB04 .(7:1)      JNK       JUNK WAIT
PCB04 .(8:1)      TIM       TIMER WAIT
PCB04 .(9:1)      INT       INTERRUPT WAIT
PCB04 .(10:1)     SON       SON WAIT
PCB04 .(11:1)     FA        FATHER WAIT
PCB04 .(12:1)     ACT       SET IF PROCESS IS ACTIVE
PCB04 .(13:1)     IMP       SET IF PROCESS IS IMPEDED
PCB04 .(14:1)     SIR       SET IF PROCESS HAS SIR
PCB04 .(15:1)     CR        SET IF PROCESS IS CRITICAL

PCB05 .(0:8)      FPTR      FATHER POINTER
PCB05 .(8:8)      SPTR      SON POINTER

PCB06 .(0:8)      BPTR      BROTHER POINTER
PCB06 .(8:8)      PRI       PROCESS PRIORITY

PCB07 .(0:8)      PCST      REQUESTED CODE SEGMENT
PCB07 .(8:8)      BPTLINK   BREAKPOINT LIST FOR PROCESS

PCB08 .(0:1)                ZERO
PCB08 .(1:1)      E         SET IF PROCESS IS ON ES QUEUE SCHEDULED
PCB08 .(2:1)      LQ        SET IF PROCESS IS SET IN A LINEAR MANNER
PCB08 .(3:1)      C         SET IF PROCESS IS ON CS QUEUE SCHEDULED
PCB08 .(4:1)      D         SET IF PROCESS IS ON DS QUEUE SCHEDULED
PCB08 .(5:1)      NL        SET IF PROCESS IS CORE RESIDENT
PCB08 .(6:1)      DEAD      SET DURING EXPIRATION
PCB08 .(7:1)      FAC       IF SET, THE FATHER IS TO BE ACTIVATED ON
                            PROCESS TERMINATION
PCB08 .(8:8)      IQPTR     IMPEDE QUEUE POINTER

PCB09 .(0:1)      LIV       SET IF PROCESS IS ALIVE
PCB09 .(1:2)      BMS       BLOCK MAIL, VALID IF MA SET.
                            0     SEND TO FATHER
                            1     RECEIVE FROM FATHER
                            2     SEND TO SON
                            3     RECEIVE FROM SON
PCB09 .(3:2)      PPC       PROCESS TO PROCESS COMMUNICATIONS SET
                            WITH RESPECT TO SON.
                            0     NULL
                            1     SON TO FATHER
                            2     FATHER TO SON
                            3     BLOCKED
PCB09 .(5:1)      STOV      STACK OVERFLOW BIT
PCB09 .(6:3)      PTYPE     PROCESS TYPE
                            0     USER
                            1     USER, SON OF MAIN
                            2     USER, MAIN
                            3     USER, MAIN, TASK
                            4     SYSTEM
                            5     NOT USED
                            6     SYSTEM, UCOP
                            7     NOT USED
PCB09 .(9:1)                      UNUSED
PCB09 .(10:1)     HK        HARD KILL PSEUDO INTERRUPT
PCB09 .(11:1)     SK        SOFT KILL PSEUDO INTERRUPT
PCB09 .(12:1)     ST        STOP PSEUDO INTERRUPT
PCB09 .(13:1)     HB        HYBERNATE PSEUDO INTERRUPT
PCB09 .(14:1)     CY        CONTROL Y PSEUDO INTERRUPT
```

```
PCB09 .(15:1)    BK       BREAK PSEUDO INTERRUPT

PCB10 .(0:3)     PSIM     PSEUDO INTERRUPT MODE
                          1     HARD KILL
                          2     SOFT KILL
                          3     STOP
                          4     HYBERNATE
                          5     ESCAPE  (CONTROL Y)
                          6     BREAK
                          7     NORMAL
PCB10 .(3:13)    BQP      BACKWARDS QUEUE POINTER

PCB11 .(0:16)    TLQ      TIME LEFT IN QUANTUM, VALID ONLY IF LQ = 0

PCB12 .(0:16)    PBX      PROCESS BLOCK POINTER. INDEX TO SHARABLE CST
                          POINTER INTO CSTBT.

PCB13 .(0:16)    LAT      TIME SINCE LAST ABSENCE

PCB14 .(0:16)    WSP      WORKING SET POINTER

PCB15 .(0:16)    PRINX    INDEX TO MAM PRESENCE REQUEST IF <> 0
```

PCBX GENERAL FORMAT

Revised 2/5/79



PXGLOB FORMAT

<table>
<tr><th>Ø</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>1Ø</th><th>11</th><th>12</th><th>13</th><th>14</th><th>15</th><th></th></tr>
<tr><td colspan="16">DL–a = SEG. REL DL VALUE</td><td>Ø</td></tr>
<tr><td colspan="16">DB–a = SEG. REL DB VALUE</td><td>1</td></tr>
<tr><td colspan="16">USER ATTRIBUTES</td><td>2</td></tr>
<tr><td colspan="8">JMAT INDEX</td><td colspan="8">ACTUAL JOB INPUT LDN</td><td>3</td></tr>
<tr><td colspan="8">JPCNT INDEX (Rel Byte Addr)</td><td colspan="8">ACTUAL JOB OUTPUT LDN</td><td>4</td></tr>
<tr><td colspan="6"></td><td colspan="10">JDT DST INDEX</td><td>5</td></tr>
<tr><td colspan="2"></td><td colspan="2">TY</td><td>D</td><td>I</td><td colspan="10">JIT DST INDEX</td><td>6</td></tr>
<tr><td colspan="8">JCUT INDEX</td><td colspan="8"></td><td>7</td></tr>
</table>

I = JOB IN/LIST INTERACTIVE

D = JOB IN/LIST DUPLICATIVE

TY = JOB TYPE

Ø = UNDEFINED
1 = SESSION
2 = JOB
3 = TASK

PXFIXED FORMAT

0			c-b PXFIXED SIZE	0
1			RELATIVE S (S-DB)	1
2			RELATIVE Z (Z-DB)	2
3			INITIAL Q (Q-DB)	3
4			RELATIVE DL (DB-DL)	4
5			GENERAL RESOURCE CAPABILITY (FROM PROG FILE)	5
6			ABORT X	6
7			EXTRA DATA SEGMENT COUNT (# of [illegible])	7
10	P	S	EXTRA DATA SEGMENT DST INDEX	8
11	P	S	EXTRA DATA SEGMENT DST INDEX	9
12	P	S	EXTRA DATA SEGMENT DST INDEX	10
13	P	S	EXTRA DATA SEGMENT DST INDEX	11
14	ABORT Y		INITIAL CST INDEX	12
15			MAXIMUM STACK SIZE (MAXDATA LIMIT)	13
16			ARITHMETIC TRAP MASK	14
17			ARITHMETIC TRAP PLABEL	15
20			LIBRARY TRAP PLABEL	16
21			SYSTEM TRAP PLABEL	17
22			CONTROL Y PLABEL	18
23	JOB TYPE		JOB #	19
24			ACTUAL SIZE OF VIRTUAL SPACE ALLOCATED TO STACK	20
25			USER ABORT PLABEL	21
26			LOAD PROCEDURE I.D.	22
27			CURRENT MAX STACK SIZE (# WORDS IN VIR STORAGE)	23
30			PROCESS CPU TIME	24
31			(MSEC)	25
32			MAXIMUM DATA SEG SIZE USED (IN SECTORS)	26
33			TOTAL VIRTUAL STORAGE USED (IN SECTORS)	27
34			CURRENT EXTRA DATA SEGMENT SPACE	28

NOTES: P = 1 IF OPENED BY PRIV USER
S = 1 IF DATA SEG IS SHARABLE

3.1 File System Section of PCBX (PXFILE)

The PXFILE area is a sub-section of the PCBX. It is a contiguous, expandable and contractable block of storage that is managed by the file system primarily for its own use. Other subsystems, namely CS and DS, also make use of the PXFILE section. In doing so they must conform to the conventions of the file system.

The overall structure of the PXFILE area is:

```
-------------------------
|                       |
|       OVERHEAD        |     (fixed)
|                       |
|-----------------------|
|                       |
|   CONTROL BLOCK       |     (variable)
|       TABLE           |
|                       |
|-----------------------|
|                       |
|      AVAILABLE        |     (variable)
|                       |
|-----------------------|
|                       |
|  AVAILABLE FILE       |     (variable)
|      TABLE            |
|                       |
-------------------------
```

3.1.1 Overhead

The part labeled OVERHEAD contains information that is pertainent to the entire table.

0 1 7	8 15	Word
PXFILE SIZE IN WORDS		0
LAST DOPEN ERROR NUMBER	LAST COPEN ERROR NUMBER	1
[redacted]		2
RESERVED FOR DS		3
LAST KOPEN ERROR NUMBER	[redacted]	4
[redacted]		5
CS TRACE FILE INFO		6
LAST RESPONDING NO-WAIT I/O AFT ENTRY NUMBER		7
1st, USER (NOBUF) CONTROL BLOCK TABLE DST NUMBER		8
2nd, USER (NOBUF) CONTROL BLOCK TABLE DST NUMBER		9
3rd, USER (NOBUF) CONTROL BLOCK TABLE DST NUMBER		10
4th, USER (NOBUF) CONTROL BLOCK TABLE DST NUMBER		11
5th, USER (NOBUF) CONTROL BLOCK TABLE DST NUMBER		12
6th, USER (NOBUF) CONTROL BLOCK TABLE DST NUMBER		13
7th, USER (NOBUF) CONTROL BLOCK TABLE DST NUMBER		14
8th, USER (NOBUF) CONTROL BLOCK TABLE DST NUMBER		15

Each DST can contain multiple CBs

In general the following identifiers are used when referring to this part of the PXFILE area:

```
DEFINE
PXFSIZE        = PXFILE#,          <<PXFILE SIZE>>
PXFDOPEN       = PXFILE(1).(0:8)#,<<LAST DOPEN ERROR CODE>>
PXFCOPEN       = PXFILE(1).(8:8)#,<<LAST COPEN ERROR CODE>>
```

[redacted]

```
PXFDSINFO      = PXFILE(3)#,      <<RESERVED FOR DS>>
PXFKOPEN       = PXFILE(4).(0:8)#,<<LAST KOPEN ERROR CODE>>
PXFFOPEN       = PXFILE(4).(8:8)#,<<LAST FOPEN ERROR CODE>>
PXFAFTSIZE     = PXFILE(5)#,      <<AFT SIZE IN WORDS>>
PXFCTRINFO     = PXFILE(6)#,      <<CS TRACE FILE INFO>>
PXFLEFTOFF     = PXFILE(7)#,      <<LAST RESPONDING AFT NR.>>
PXFCBT1        = PXFILE(8)#,      <<1ST USER CBT DST NR.>>
PXFCBT2        = PXFILE(9)#,      <<2ND USER CBT DST NR.>>
PXFCBT3        = PXFILE(10)#,     <<3RD USER CBT DST NR.>>
PXFCBT4        = PXFILE(11)#,     <<4TH USER CBT DST NR.>>
PXFCBT5        = PXFILE(12)#,     <<5TH USER CBT DST NR.>>
PXFCBT6        = PXFILE(13)#,     <<6TH USER CBT DST NR.>>
PXFCBT7        = PXFILE(14)#,     <<7TH USER CBT DST NR.>>
PXFCBT8        = PXFILE(15)#;     <<8TH USER CBT DST NR.>>
```

The following is an alphabetized list of the above identifiers along with a discussion of their meaning.

PXFAFTSIZE — This is the size (in words) of the Available File Table. Note that the size is in words and not in terms of number of entries. The reason for this is that it simplifies the calculation for the size of the available block.

PXFCBT1-8 — These are the DST numbers of the user (NOBUF) control block tables. A DST number of 0 indicates that no data segment is allocated. Note that a DST number is representable with ten bits; a full word is used to simplify the code.

PXFCOPEN — This contains the last COPEN error number. It is not used by the file system; it is included here for completeness only.

PXFCTRINFO — This contains information pertainent to the CS trace file. It is not used by the file system; it is included here for completeness only.

PXFDOPEN — This contains the last DOPEN error number. It is not used by the file system; it is included here for completeness only.

PXFDSINFO — This cell is reserved for DS. It is not used by the file system; it is included here for completeness only.

PXFFOPEN — This contains the last FOPEN error number. If it is zero then the last FOPEN completed successfully; if it is non-zero then the last FOPEN completed unsuccessfully and the number represents the file system error number. Note that only eight bits are needed to hold the error number; a full word is used to simplify the code.

PXFKOPEN — This contains the last "KOPEN" error number. Since KSAM is imbedded in the file system, an FOPEN failure on a KSAM file can be caused by a failure to open either the key file or the data file. This error number is used in conjunction with PXFFOPEN to determine which file caused the KSAM open failure. Note that this error number is not used by the file system; it is included here for completeness only.

PXFLEFTOFF — This is the AFT entry number of the last file/line that completed a no-wait I/O; if zero then no no-wait I/O has been completed. This cell is maintained solely by and for the IOWAIT intrinsic.

PXFNOCB — This bit is used to signify that no control blocks are to be created in the PXFILE control block table. This bit is set by the NOCB parameter to the CREATE intrinsic or the :RUN command. The reason for this feature is to permit the 3000/20 user to have as much stack space as possible; otherwise the MPE/30 file system will take away several hundred words of stack for the PXFILE control block table.

PXFSIZE — This is the size (in words) of the complete PXFILE area. It is the sum of the overhead block, the control block table, the available file table and the available block.

3.1.2 Control Block Table (PXFCBT)

The part labeled CONTROL BLOCK TABLE contains a file control block table. This is a new feature with MPE/30; it is not present under MPE/20.

The format of the control block table is the same as any other file control block table. The only difference is that addressing is slightly more complicated since the table does not begin at DB+0. As a result all pointers within the table are table relative; the starting address of the table must be added to a pointer to generate a final DB-relative address. This addressing convention is consistently applied to all file control block tables.

When the control block table is expanded, space is taken from the AVAILABLE area. If no space is available then the PXFILE area is expanded and the acquired space is added to the AVAILABLE area.

The interested reader is referred to section 3.2 for a more detailed description of file control block tables.

0 ... 15	
TABLE SIZE IN WORDS	16
DST NUMBER CONTAINING TABLE	17
VECTOR TABLE SIZE IN WORDS	18
LOCK WORD	19
IMPEDED QUEUE	20
VECTOR TABLE	21
CONTROL BLOCK AREA	

In general the following identifiers are used when referring to this part of the PXFILE area:

DEFINE

```
PXFCBTAB        = PXFILE(16)#,    <<CONTROL BLOCK TABLE>>
PXFCBTSIZE      = PXFILE(16)#,    <<TABLE SIZE IN WORDS>>
PXFDSTX         = PXFILE(17)#,    <<TABLE DST NUMBER>>
PXFVTSIZE       = PXFILE(18)#,    <<VECTOR TABLE SIZE IN WORDS>>
PXFLOCK         = PXFILE(19)#,    <<TABLE LOCK WORD>>
PXFQUEUE        = PXFILE(20)#,    <<TABLE IMPEDED QUEUE>>
PXFVT           = PXFILE(21)#;    <<VECTOR TABLE>>
```

The following is an alphabetized list of the above identifiers along with a discussion of their meaning.

PXFCBTAB	This is the first word of the control block table. In general this is used only when referring to the entire control block table.
PXFCBTSIZE	This is the size in words of the control block table. In general this is used only when calculating the size of the available block.
PXFDSTX	This is the DST number of the data segment that contains the control block table. This is the same as the DST number of the stack. Note that the convention of referring to the DST number of the stack as zero is not used. The reason for this is that the file system may refer to a PXFILE control block table in another stack. This would result in an ambiguity since that PXFILE control block table would also have a DST number of zero.
PXFLOCK	This is the lock word for the table and has the same format as the lock word for a control block in the table.
PXFQUEUE	This is the impeded queue for the table and has the same format as the impeded queue for a control block in the table.
PXFVT	This is the first word of the vector table. It is used when referring to the vector table in general.
PXFVTSIZE	This is the size, in words, of the vector table. Note that this is the length of the table and does not reflect the number of entries used or unused.

3.1.3 Available Block

The part labeled AVAILABLE BLOCK is used to provide space when the Control Block Table or the Available File Table is expanded. These two tables grow towards each other, and when more space is needed it is simply taken from the Available Block.

When the Available Block is exhausted, the PXFILE area is expanded, the AFT is relocated and the new space is added to the Available Block.

Note that currently the PXFILE area is only expanded; it is never contracted.

3.1.4 [redacted]

The part labeled AVAILABLE FILE TABLE contains information used by the file system (or CS, DS, etc.) to grossly characterize the file access and, most importantly, to give the location of the control blocks.

The overall structure of the AFT is:

```
--------------------------
|                        |
|        ENTRY N         |
|                        |
|------------------------|
|                        |
|           .            |
|           .            |
|           .            |
|                        |
|------------------------|
|                        |
|        ENTRY 1         |
|                        |
--------------------------
```


where N = PXFAFTSIZE/4.

The AFT is as long as specified by PXFAFTSIZE. Unused entries are all zero's. When the table is full it is expanded by taking space from the AVAILABLE block.

The AFT is negatively indexed by file number: the entry at DL-8 corresponds to file number 1, the entry at DL-12 corresponds to file number 2, etc.

The structure of an AFT entry is:

```
   0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15
-----------------------------------------------------------------
|  ENTRY TYPE     | N |                                         |  0
|---------------------------------------------------------------|
|                     PHYSICAL ACB VECTOR                       |  1
|---------------------------------------------------------------|
|                     LOGICAL ACB VECTOR                        |  2
|---------------------------------------------------------------|
|                     NO-WAIT I/O IOQX                          |  3
-----------------------------------------------------------------
```

Note that the entry format is dependent on the entry type. The one shown above is the one used by the file system.

In general the following identifiers are used when referring to an AFT entry:

```
DEFINE
AFTTYPE          = AFT.(0:4)#,        <<ENTRY TYPE>>
AFTNULL          = AFT.(4:1)#,        <<$NULL FILE>>
AFTPACBV         = AFT(1)#,           <<PACB VECTOR>>
AFTLACBV         = AFT(2)#,           <<LACB VECTOR>>
AFTIOQX          = AFT(3)#;           <<NO-WAIT I/O IOQX>>
```

The following is an alphabetized list of the above identifiers along with a discussion of their meaning.

AFTIOQX — This is the IOQ index of the pending no-wait I/O (if any). Note that this is applicable iff the file was opened with the NOWAIT option specified. Also, CS and DS have the same capability and used this cell in a consistent manner. The reason for this is that the IOWAIT intrinsic services the file system as well as CS and DS, and is the principal user of this cell. If the cell is zero then there is no I/O pending; otherwise the cell contains the IOQ index corresponding to the pending I/O.

AFTLACBV — This is the vector of the Logical ACB (LACB) (if any). Note that this is applicable iff the file was opened with the multi-access option specified.

AFTNULL — This bit signifies that the file is $NULL and that there are no control blocks.

AFTPACBV — This is the vector of the Physical ACB (PACB). Note that a PACB exists for all files except $NULL.

AFTTYPE — This is the AFT entry type number. At present the following entry types are defined:

0 - file system
1 - remote file
2 - DS
3 - DS
4 - CS
5 - CS

PCBX FOR CORE RESIDENT SYSTEM PROCESS STACKS
(MICRO-MINI PCBX)

Ø	DL – a (Seq Rel DL Value)				Ø	PXGLOB
1	DB – a (Seq Rel DB Value)				1	PXGLOB
2	USER ATTRIBUTES (always –1)				2	PXGLOB
3	Ø		INPUT DEV LDEV #		3	PXGLOB
4	Ø		OUTPUT DEV LDEV #		4	PXGLOB
5	Ø				5	PXGLOB
6	Ø	D	I	Ø	6	PXGLOB
7	Ø				7	PXGLOB
1Ø	PXFIXED SIZE (c – b)				8	PXFIXED
11	RELATIVE S (S – DB)				9	PXFIXED
12	RELATIVE Z (Z – DB)				1Ø	PXFIXED
13	INITIAL Q (Q – DB)				11	PXFIXED
14	RELATIVE DL (DB – DL)				12	PXFIXED
15	GENERAL RESOURCE CAPABILITY (–1)				13	PXFIXED
16	ABORT X				14	PXFIXED
17	PCBX FREE SPACE POINTER (–1)				15	PXFIXED
2Ø	DL – b				16	PXFIXED
21	DL – a				17	PXFIXED

Notes:
1. There is no PXFILE area.
2. The PXFIXED area is much smaller than a normal PCBX.
3. The PCBX FREE SPACE POINTER is –1 indicating that the PCBX *cannot be expanded.*

PROCESS TO PROCESS COMMUNICATION TABLE

THIS TABLE IS USED AS THE COMMUNICATION LINK BY WHICH FATHER AND SON PROCESSES COMMUNICATE WITH ONE ANOTHER VIA THE MAILBOX SCHEME. THIS TABLE CONTAINS TWO WORDS PER ENTRY AND IS INDEXED BY PCB # (ENTRY INDEX Ø IS MEANINGLESS). EACH TWO WORD ENTRY OF INDEX N ESSENTIALLY RELATES WHERE, AS WELL AS HOW MUCH, MAIL MAY BE FOUND FOR A PROCESS N WITH RESPECT TO COMMUNICATIONS BETWEEN N AND HIS *FATHER* PROCESS.

ENTRY FORMAT

WORD Ø	WORD COUNT
WORD 1	MAIL WORD OR DST #

WHERE WORD Ø = THE # OF MAIL WORDS TO BE TRANSFERRED.

WORD 1 = THE ONLY WORD OF MAIL ITSELF IF WORD Ø = 1

OTHERWISE

IT CONTAINS THE DST # OF THE EXTRA DATA SEGMENT WHERE "WORD COUNT" WORDS OF MAIL EXIST.

NOTE: SAY PROCESS N IS THE SON OF PROCESS M. THEN THE PROCESS TO PROCESS TABLE INDEX WHICH WILL BE USED FOR MAILBOX COMMUNICATION BETWEEN SON N AND FATHER M WILL BE THAT OF THE SON (I.E. N).

```
QI
-18   DISAP               PSEB,PSDB COUNTER.
-17   XXXXX               UNUSED.
-16   SDST                PROCESS STACK DST.
-15   PSTA                PSEUDO-INTERRUPT STATUS.
-14   PADDR               PSEUDO-INTERRUPT ADDRESS.
-13   TRACE FLAG          FLAG SET NON-ZERO IN IXIT FROM ICS.
-12   PFAIL               POINTER TO POWERFAIL PCB.
-11   JCUT                ABSOLUTE JCUT ADDRESS.
-10   XP                  POINTER TO EXECUTING PROCESS PCB.
 -9   PCBX                ABSOLUTE STACK ADDRESS.
 -8   Z                   STACK DB RELATIVE Z <<REL TO QI-4>>.
 -7   DL                  STACK RELATIVE DL <<REL TO QI-4>>.
 -6   S                   STACK DB RELATIVE S <<REL TO QI-4>>.
 -5   S BANK              <<NOT NECESSARILY EQUAL TO RETURN DB BANK>>.
 -4   STDB                ABL STACK DB <<NOT NECESSARILY EQUAL TO
 -3   X                   !                  RETURN DB>>.
 -2   P                   ! DISPATCH
 -1   STATUS              ! STACK
  0   DQ (ALWAYS 0)       ! MARKER.
  1   DB BANK             DISPATCHER RETURN DB BANK.
  2   DB                  DISPATCHER RETURN DB.
  3   PARAM               DEVNO IF EXT INT,ELSE PARAM.
```

ICS GLOBAL WITH CELLS INITIALIZED FOR PROGENITOR.

STDB - ABSOLUTE ADDRESS OF THE CURRENTLY RUNNING PROCESS STACK.
SBANK - BANK ADDRESS FOR PROCESS STACK.
S - STACK DB RELATIVE S.
DL - STACK DB RELATIVE DL.
Z - STACK DB RELATIVE Z.
PCBX - ABSOLUTE STACK ADDRESS.
XP - PCB TABLE RELATIVE POINTER TO WORD ZERO OF THE RUNNING PROCESS PCB.

INITIAL SETS THE FOLLOWING AS DESCRIBED.

CPCB - ABSOLUTE 4 IS AN ABSOLUTE VERSION OF XP. IF CPCB IS ZERO, THEN THE ABOVE CELLS ARE INVALID. THIS WILL NEVER BE THE CASE IN A PROCESS. CPCB SHOULD ALSO BE SET BY INITIAL.
SDST - DST NUMBER FOR THE RUNNING PROCESS STACK.
JCUT - THE BANK ZERO ABSOLUTE ADDRESS OF THE JCUT TABLE.
PADDR - PB RELATIVE ADDRESS FOR THE PROCEDURE PSEUDOINT.
PSTA - STATUS VALUE FOR PSEUDOINT,%140000+CST#.
DISAP - PSDB COUNTER,INITIALLY ZERO.

SUB-SYSTEM RESERVED DL AREA

	REMAINING DL AREA	
DB-12	RESERVED FOR SORT/MERGE	DB-10
DB-11	RESERVED FOR TRACE	DB-9
DB-10	EXTERNAL PLABEL OF OUTER BLOCK	DB-8
DB-7	RESERVED FOR TRACE	DB-7
DB-6	DB ADDRESS OF STLT	DB-6
DB-5	RESERVED FOR COBOL	DB-5
DB-4	RESERVED FOR COBOL	DB-4
DB-3	RESERVED FOR COBOL	DB-3
DB-2	RESERVED FOR FORMATTER	DB-2
DB-1	DB ADDRESS OF FLUT	DB-1
	DB AREA	

See EXT

DST #53 11/16/73 JFB

SIR # ALLOCATION

DECIMAL SIR #	OCTAL SIR #	SIR NAME
1	1	~~FREE MEMORY LINK~~ LOAD PROCESS SIR
2	2	~~MEMORY OVERLAY~~ LOCK SEGMENT SIR
3	3	~~DST ENTRIES~~ IDD
4	4	~~PCB ENTRIES~~ ODD
5	5	PROCESS TREE STRUCTURE
6	6	SCHEDULING QUEUE
7	7	CST ENTRIES
8	10	SYSTEM DIRECTORY
9	11	LPDT
10	12	LDT
11	13	STORAGE IN OVERLAY AREA
12	14	DISC FREE SPACE TABLE
13	15	JPCNT
14	16	JCUT
15	17	JMAT
16	20	(FREE)
17	21	LOADER SEGMENT TABLE
18	22	VDD
19	23	~~(FREE)~~ SPOOL
20	24	MESSAGE CATALOGUE
21	25	RIT
22	26	VOLUME TABLE
23	27	MESSAGE QUEUE DST # 33_{10}
24	30	MESSAGE QUEUE DST # 34_{10}
25	31	~~(FREE)~~ CS ALLOC[illegible]
26	32	LOGGING BUFFER
27-34	33-42	(FREE)
35	43	SUB-QUEUE MAPPING TABLE
36	44	(FREE)
37	45	FILE INTEGRITY
38	46	RIN
39	47	(FREE)
40	50	1st JOB
41	51	2nd JOB
⋮	⋮	⋮

* NOT USED

MULTIPLE SIR ALLOCATION

THE FOUR CONVENTIONAL CHAINS USED BY MPE FOR SIR ALLOCATION AND DEALLOCATIO

LOWER → LOGICAL RANK → HIGHER

1. LDT(10) → LPDT(9) → VDD(18)
2. JMATSIR(15) → LPDT(9) → JPCNT(13)
3. FILESIR(37) → DIRECTORY(8) → LPDT(9) → DISC FREE SPACE TABLE(12)
4. FILESIR(37) → RINTABLE(38)

MULTIPLE SIR ALLOCATION REQUIRES CARE TO AVOID PROCESS DEADLOCK SITUATIONS. TH
RULE THAT SHOULD BE FOLLOWED WHEN WORKING WITH THE ABOVE SIRS IS AS FOLLOWS:

NEVER ATTEMPT A GETSIR OF LOWER RANK THEN THE SIR CURRENTLY HELD (IF ANY).

FOR EXAMPLE: SUPPOSE TWO PROCESSES, A AND B, REQUIRED THE SIRS FOR THE LDT AND LPDT. DEADLOCK WOULD RESULT IF DONE AS BELOW DUE TO PROCESS A NOT FOLLOWING THE CONVENTION ORDER.

INCORRECT ORDER		CORRECT ORDER	
PROCESS "A"		PROCESS "B"	
GETSIR(9)	[LPDT]	GETSIR(10)	[LDT]
GETSIR(10)		GETSIR(9)	

DEADLOCK!

SIR TABLE INFORMATION

11/16/73 JFB

THE SYSTEM INTERNAL RESOURCE TABLE IS LOCATED IN NON-LINKED MEMORY (RESIDENT TABLE). THE SIR TABLE IS USED TO PROTECT CRITICAL SYSTEM ELEMENTS AGAINST ACCESS BY MORE THAN ONE PROCESS, I.E., IT PROVIDES A "LOCK OUT" MECHANISM. EACH CRITICAL SYSTEM RESOURCE (USUALLY A TABLE) IS ASSIGNED A SPECIFIC SIR NUMBER. PROCEDURES ARE PROVIDED WITHIN MPE TO LOCK (GETSIR) AND UNLOCK (RELSIR) THE SIR. PROCESSES ATTEMPTING TO OBTAIN A SIR THAT IS NOT AVAILABLE ARE IMPEDED BY THE SYSTEM. THE SIR TABLE ENTRIES FORM THE HEAD OF A LINKED LIST IN THIS CASE. IF MORE THAN ONE PROCESS BECOMES IMPEDED WORD 8 OF THE PCB ENTRY IS USED TO ADD THE "NEW" PROCESS TO THE GROWING LIST. THE METHOD OF DIS-IMPEDING THE PROCESS DEPENDS ON THE SIR TYPE.

<u>SHORT SIRS AND LONG SIRS?</u>

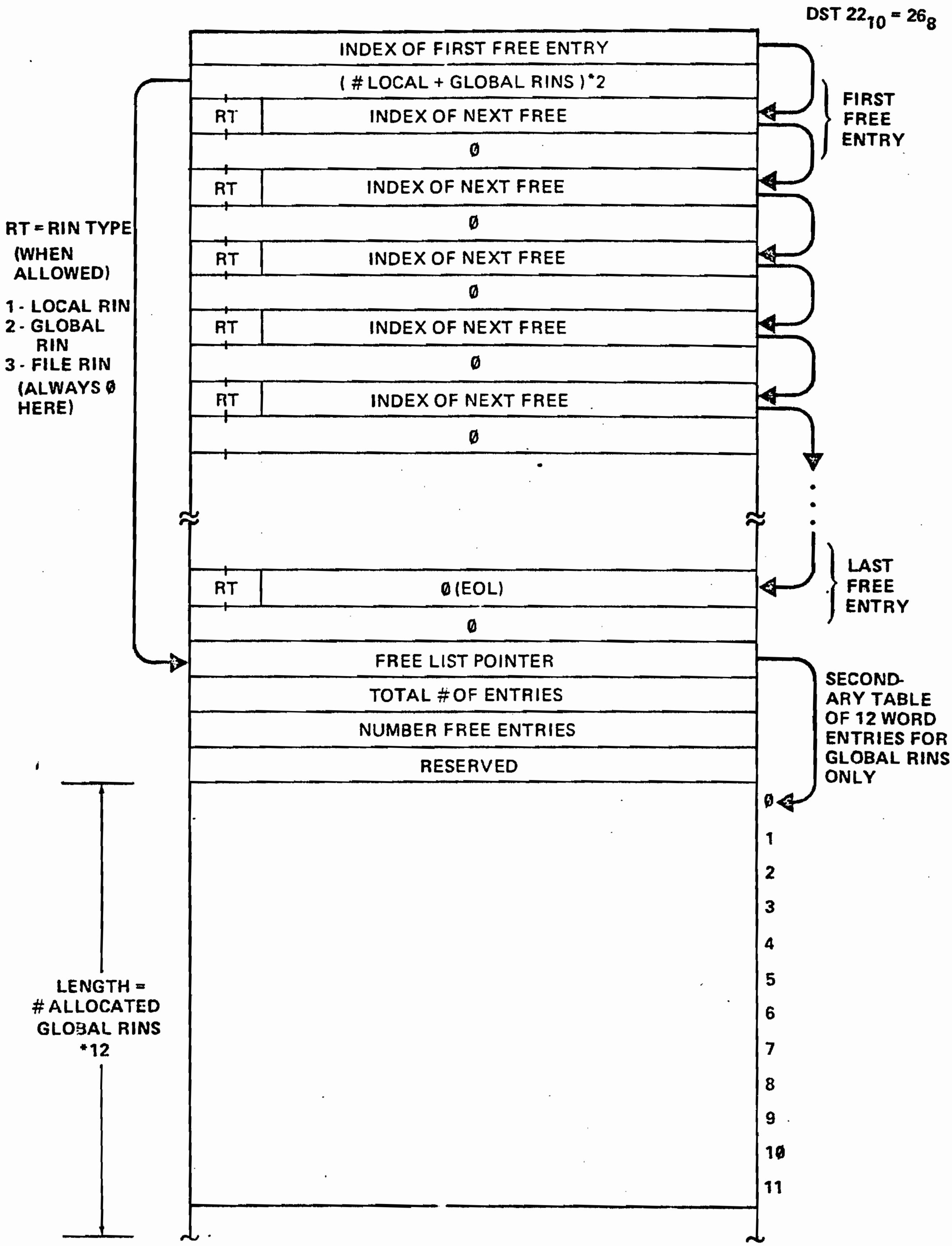
A SHORT SIR IS DEFINED AS A SIR THAT CAN USUALLY BE OBTAINED AND RELEASED WITHIN ONE SUBQUEUE TIME QUANTUM. A LONG SIR THEREFORE IS A SIR THAT CANNOT SATISFY THE SHORT SIR REQUIREMENTS. THE DIFFERENCE IN THE SYSTEM HANDLING OF A SHORT OR LONG SIR IS APPARENT ONLY WHEN ONE OR MORE PROCESSES BECOME IMPEDED ON BEHALF OF A SIR.

<u>SHORT SIR:</u> A SHORT SIR IS SENSITIVE TO THE PRIORITY OF PROCESSES IN THE IMPEDED LIST. AS PROCESSES REQUESTING THE SIR BECOME IMPEDED, THE SYSTEM COMPARES PRIORITIES, AND ALWAYS RECORDS THE PRIORITY NUMBER OF THE HIGHEST PRIORITY PROCESS IN THE SIR TABLE. WHEN A PROCESS RELEASES THE SIR, ALL PROCESSES IN THE LIST ARE DIS-IMPEDED. IF THE CURRENT HOLDER OF THE SIR IS ALSO THE HIGHEST PRIORITY PROCESS IN THE LIST, A NORMAL RETURN TO THE CALLING PROCESS IS MADE AND EXECUTION CONTINUES. IF THE CURRENT HOLDER OF THE SIR IS NOT THE HIGHEST PRIORITY PROCESS IN THE LIST, A TRANSFER TO THE DISPATCHER IS MADE, AND THE HIGHEST PRIORITY PROCESS WILL BE DISPATCHED AND PRESUMABLY OBTAIN THE NOW FREE SIR.

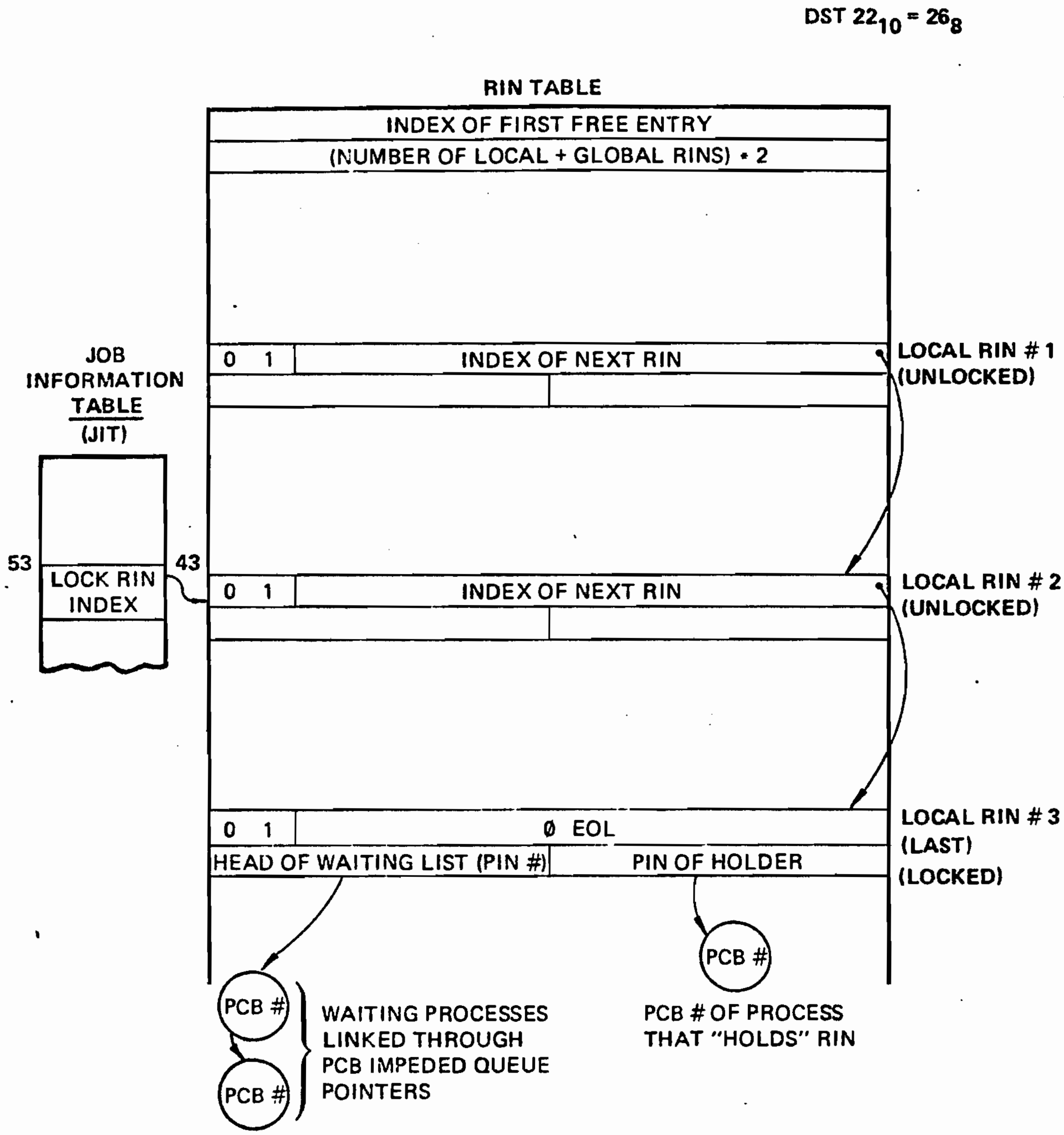
<u>LONG SIR:</u> A ~~LONG~~ SIR DOES NOT RESPECT PROCESS PRIORITY AND OPERATES IN A FIFO MANNER. AS PROCESSES BECOME IMPEDED ON BEHALF OF A ~~LONG~~ SIR THE NEW ENTRIES ARE ENTERED AT THE TAIL OF THE IMPEDED LIST. WHEN THE CURRENT HOLDER OF THE SIR RELEASES IT, ONLY THE FIRST PROCESS IN THE LIST (POINTED AT BY THE HEAD POINTER) IS DIS-IMPEDED. THE LINKED LIST HEAD AND ALL POINTERS ARE THEN UPDATED AND THE NEWLY DIS-IMPEDED PROCESS WILL ~~PRESUMABLY~~ OBTAIN THE SIR.

8-1-73

RIN TABLE GENERAL LAYOUT (Initialized State)



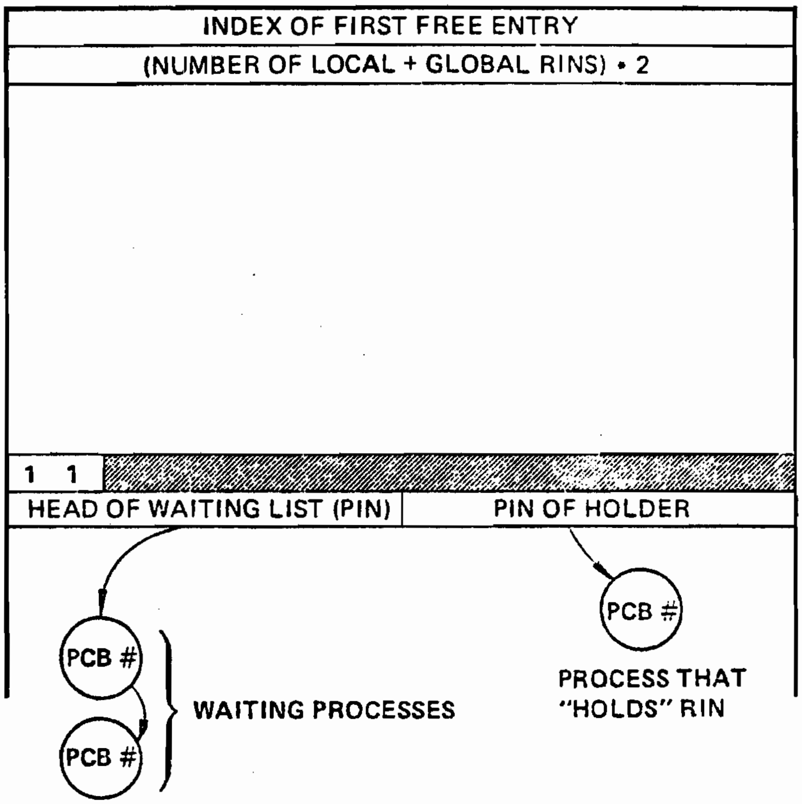
ALLOCATION AND LOCKING OF LOCAL RINS



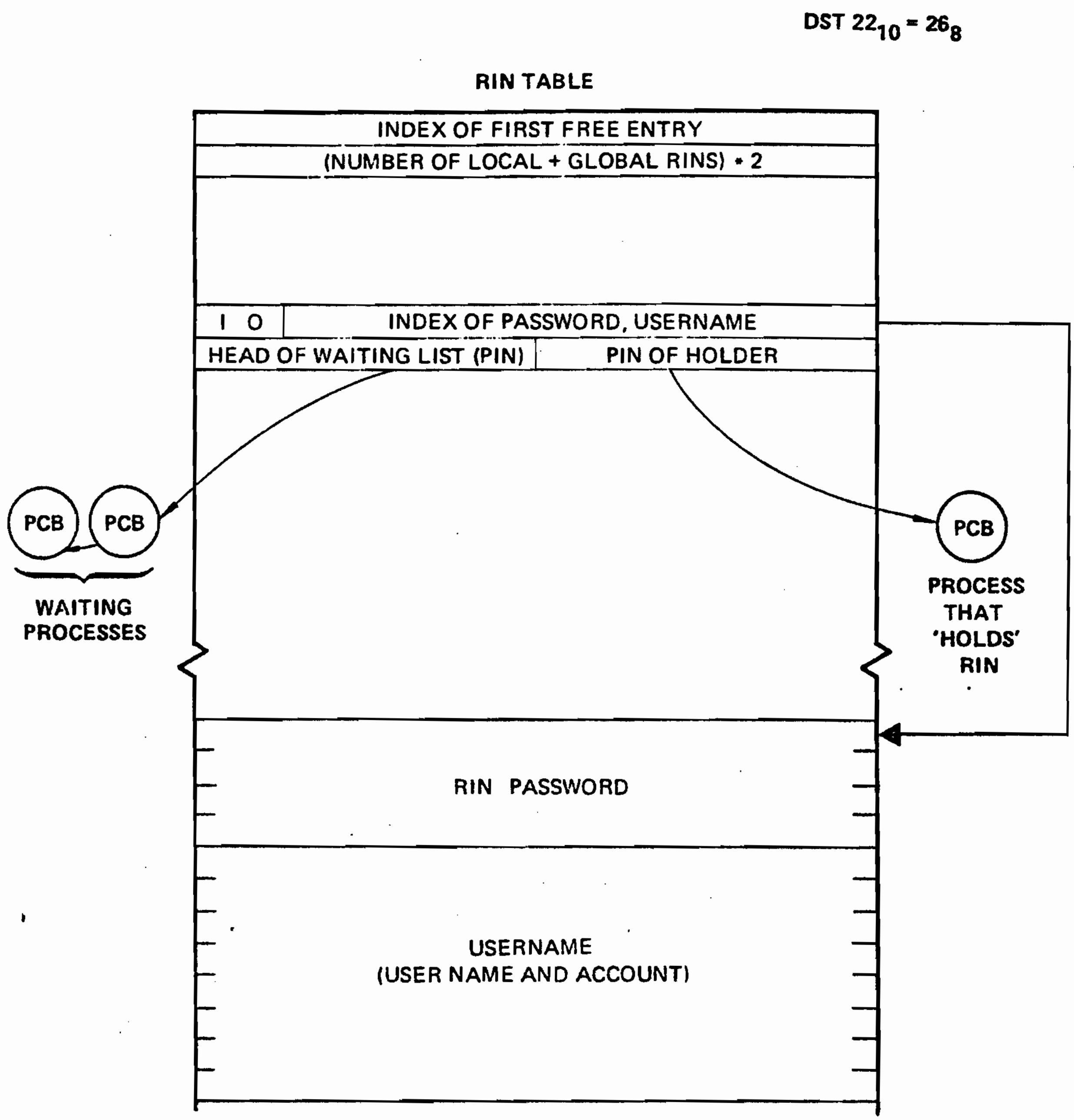
ALLOCATION AND LOCKING OF FILE RINS

DST $22_{10} = 26_8$

RIN TABLE



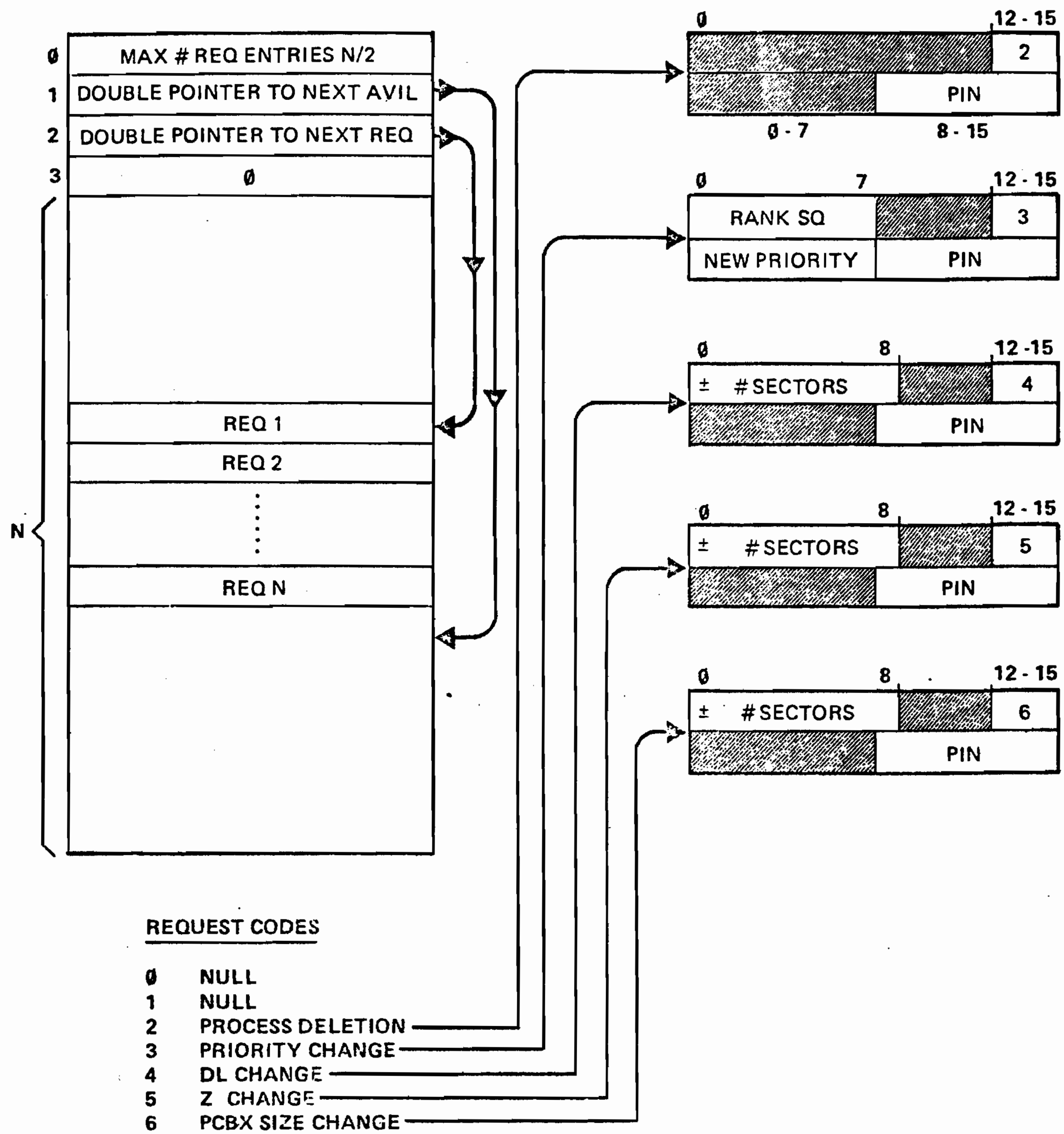
ALLOCATION AND LOCKING OF GLOBAL RINS



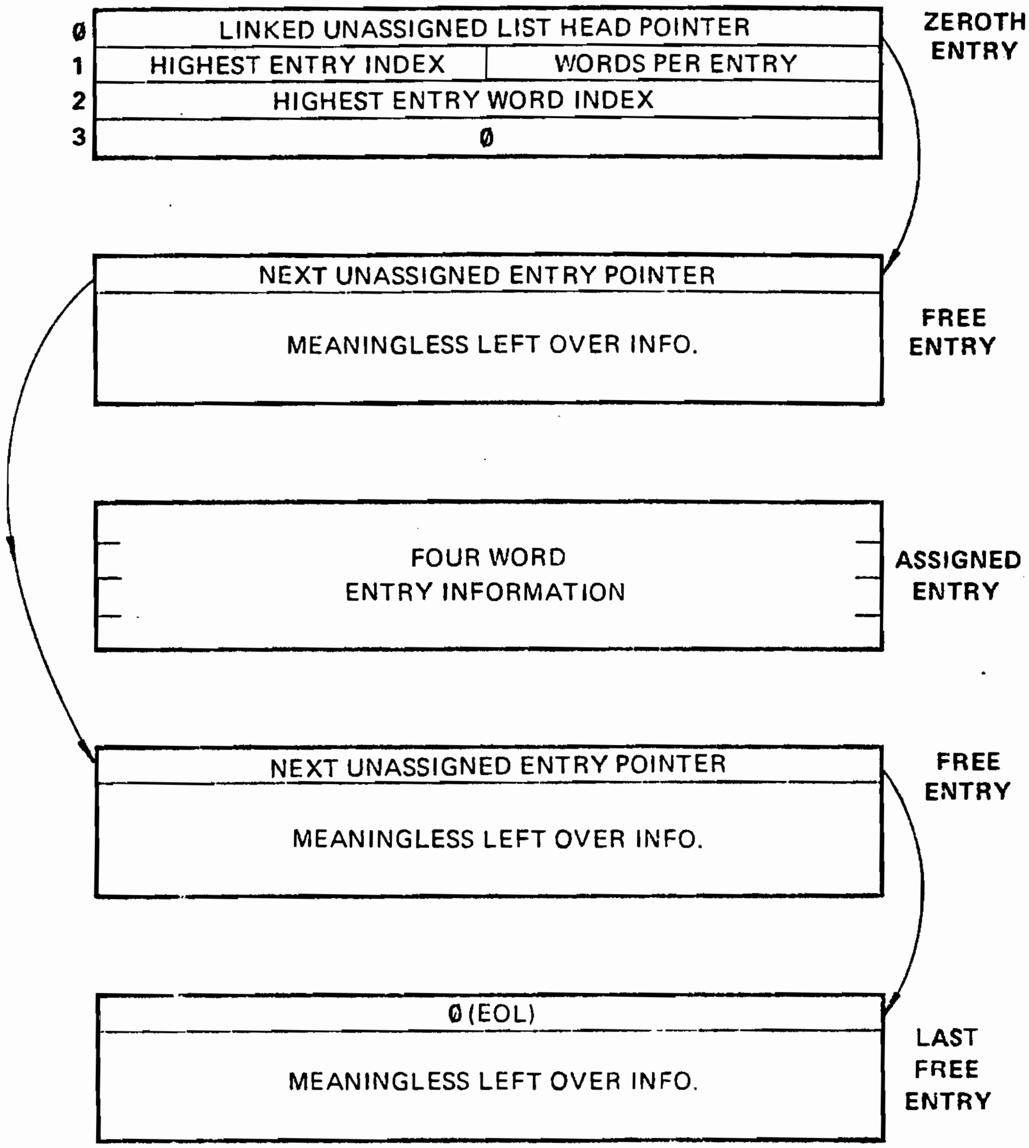
8-1-73

UCOP REQUEST QUEUE (DST #9)

ENTRY FORMATS



MAKEPRESENT REQUEST QUEUE GENERAL LAYOUT



THIS TABLE IS LOCATED IN A SPECIALLY ALLOCATED AREA OF NON-LINKED MEMORY AND IS USED TO CONTAIN REQUESTS TO MEMORY MANAGEMENT FOR MAKING CODE OR DATA SEGMENTS PRESENT IN LINKED MEMORY. REQUESTS MAY BE ENTERED IN THIS TABLE ON BEHALF OF EITHER THE DISPATCHER OR THE I/O SYSTEM. FREE ENTRIES ARE IN A LINKED FREE LIST SIMILAR TO OTHER MPE TABLES.

MAKEPRESENT REQUEST QUEUE TABLE ENTRY DEFINITION

	Ø	1	2	3	4	5	6	7	8	9	1Ø	11	12	13	14	15
$MPTAB_Ø$	LL								NL							

LL = INDEX OF PRECEEDING ENTRY
NL = INDEX OF FOLLOWING ENTRY

	Ø	1	2	3	4	5	6	7	8	9	1Ø	11	12	13	14	15
$MPTAB_1$	F	H		D	E	MPM										

F = 1 FREEZE SEGMENT IN MAIN MEMORY
H = 1 HOLD SEGMENT IN MAIN MEMORY
D = 1 REQUEST FOR DATA SEGMENT
E = 1 EXTRA DATA SEGMENT IF D = 1.
MPM = DST INDEX IF REQUEST ORIGINATES IN THE DISPATCHER.
IOQ ENTRY INDEX IF REQUEST IS GENERATED FOR AN I/O REQUIREMENT.
CST INDEX IF I = Ø (IN $MPTAB_2$) AND CD = 1 (IN $MPTAB_3$).

MAKEPRESENT REQUEST QUEUE TABLE ENTRY DEFINITION (Con't)

MPTAB$_2$

Ø	1	2	3	4	5	6	7	8	9	1Ø	11	12	13	14	15
P			B	X	I	PR	W	PIN							

P	=		LINKING PRECEDENCE CODE
B	=	1	ALLOCATE STORAGE ON LINKED MEMORY BOUNDARY
X	=	1	IGNORE THIS REQUEST – SUSPENDED
I	=	Ø	DISPATCHER REQUEST
	=	1	I/O SYSTEM REQUEST
PR	=	1	PARTIAL READ OK
W	=	1	WAKE PROCESS OR NOTIFY DISPATCHER WHEN REQUEST IS SATISFIED
PIN	=		PROCESS NUMBER OF REQUESTOR

MPTAB$_3$

Ø	1	2	3	4	5	6	7	8	9		11	12	13	14	15
L							CD	SL							
IOQH															

IOQ
HEAD

L	=	1	DISPATCHER REQUEST OF LOW PRIORITY
CD	=	1	CODE SEGMENT REQUEST
SL	=		SET INDEX OF A FAMILY OF REQUESTS FROM THE DISPATCHER
IOQH	=		IF THE ORIGIN OF THE REQUEST IS THE I/O SYSTEM THE ENTIRE WORD IS USED TO CONTAIN THE ADDRESS OF THE PARTICULAR IOQ HEAD

SYSTEM CLOCK

The system clock will be a clock/TTY card in DRT 3. It will interrupt every 100 ms, with the CR being automaticly cleared. The interrupt handler will be the procedure CLOCK. On entry, DB should be pointing to the timer request list. Besides timeout requests, the clock will also control time slicing and memory manager sampling.

TIME REQUEST LIST <TRL>

Entry	Word	Contents	
ENT Ø	Ø	FREE LIST PTR	
	1	NR. ENTRIES | ENTRY SIZE (4)	
	2	TRACE WORD	
	3	Ø	
ENT 1	4	QUANTUM / 100 ms	QTIME
	5	time of day	DTIME
	6		
	7	YEAR | Julian DAY	
ENT 2	8	PTR to most active request	HEAD
	9	Ø	
	10	Ø	dummy time
	11	Ø	
ENT 3	12	A | CODE | PTR to next	assignable entries
	13	REQ	
		Time to service after request in front	

A: Ø if inactive request
1 if active request

SYSTEM CLOCK

THE SYSTEM CLOCK WILL BE A CLOCK/TTY CARD IN DRT #3. IT WILL INTERRUPT EVERY 100 MS, WITH THE CR BEING AUTOMATICLY CLEARED. THE INTERRUPT HANDLER WILL BE THE PROCEDURE CLOCK.
ON ENTRY DB SHOULD BE POINTING TO THE TIMER REQUEST LIST. BESIDES TIMEOUT REQUEST, THE CLOCK WILL ALSO CONTROL TIMESLICING AND MEMORY MANAGER SAMPLING.

TIME REQUEST LIST (T R L)

A	0	IF INACTIVE REQUEST
	1	IF ACTIVE REQUEST

CODE + REQ INDICATE THE TYPE OF REQUEST

CODE	REQ	TYPE
0	DITP	HANGUP
1	DITP	CARRIER FAILURE
2	DITP	202 TURNAROUND
3	DITP	READ
4	DITP	LOGON
5	PCBB INDEX TO PROCESS	DELAY
6	DITP	LP NOT READY
7	DITP	2640

THE LIST OF PENDING REQUESTS IS KEPT ORDERED BY TIME WITH LATER ENTRIES AT THE TAIL.

ALSO IF CODE.(12:1) THEN

```
BEGIN
DITP.(8:1) := 1;
CODE.(12:1):= 0;
END;
```

Main mem

SYS GLOBAL

1000

OCTAL	CONTENTS	NAME	
0			
1	CST BASE - SYS BASE	SYSCST	NOTE: THESE
2	DST BASE - SYS BASE	SYSDST	ARE CELL NAMES.
3	PCB BASE - SYS BASE	SYSPCB	TABLE NAMES DO
4	MTAB BASE - SYS BASE	SYSMTAB	NOT HAVE THE
5	IOQ BASE - SYS BASE	SYSIOQ	"SYS" PREFIX.
6	SBUF BASE - SYS BASE	SYSSBUF	
7	WSTAB BASE - SYS BASE	SYSWSTAB	
10	LPDT BASE - SYS BASE	SYSLPDT	
11	STOPS BASE - SYS BASE	SYSSTOPS	
12	TRL BASE - SYS BASE	SYSTRL	
13	JCUT BASE - SYS BASE	SYSJCUT	
14	SIR BASE - SYS BASE	SYSSIR	
15	JPCNT BASE - SYS BASE	SYSJPCNT	
16	TBUF BASE - SYS BASE	SYSTBUF	
17	MONBUF BASE - SYS BASE	SMONBUF	(MONITOR BUF)
20			! RESERVED
21			! FOR MEMORY
22			! RESIDENT
23			! TABLES.
24			
25	TAU - WORKING SET PARAM - IN MILLISECONDS	WSTAU	
26	HIGH ORDER VDS DISC ADDRESS	VDSTART1	
27	LOW ORDER VDS DISC ADDRESS	VDSTART2	\ MANAGEMENT
30	CURRENT CST BLOCK INDEX	CSTBX	\
31	PREPARATION POINTER	MAMP	\
32	DISPLACEMENT TO CODE =@CST(0)-@DST(0)	DFC	\
33	DISPLACEMENT TO SHARABLE=@CST(LAST)-@DST(0)	DFS	\
34	VDS BIT MAP POINTER	VDSMAP	\
35	ABS ADDRESS (SYSDIT(8))	SYSDIT8	\
36	NUMBER OF LAST USABLE BIT IN VDS MAP	SCANEND	\
37	STARTING POS IN VDS FOR NEXT SEARCH	SCANWALL	\
40	VDS PAGE SIZE	VDSPAGE	\
41	VDS PAGE COUNT TABLE	VDSL	\
42	EXTENDED MAIN MEMORY ADDRESS -	FREEHEAD1	\
43	OF FIRST FREE LINK +6	FREEHEAD2	\
44	NUMBER OF FREE LINKS	NFREE	\
45	TEMP SPACE POINTER FOR LOCKSEG. INIT TO 0.	LOCKTANK	\
46	USED IN CONJUNCTION WITH LOCKTANK	LOCKTNKCT	\
47	NUMBER OF MEMORY BANKS. CONFIGURED TO -1.	NBANKS	\
50	MEMORY BANK SPECIFICATION TABLE	BANKTAB	\
51	POINTER TO CSTBLK TABLE	CSTBLK	\
52	ANTICIPATORY WRITE ENABLE FLAG		\
53	ALLOW LOCAL COMPRESSION FLAG		\
54			\
[redacted]	[redacted]	BUSY	: RESERVED
[redacted]	[redacted]	HEAD	: FOR I/O
[redacted]	[redacted]	TAIL	: SYSTEM.
60	NUMBER OF SIO PROGRAMS EXECUTING	SIOCOUNT	:
61	PARITY ERROR FLAG	PARITY	:
62	I/O MESSAGE QUEUE HEAD INDEX	IOMSGQX	:
63	I/O LOG QUEUE HEAD INDEX	IOLOGQX	:
64	NUMBER OF TERMINALS READING	RDCOUNT	:
65	NUMBER OF TERMINALS WRITING	WRTCOUNT	:

```
 66 DSETB                                              DSETB    :
 67 LAST TIMER                                         LTIME1   :
 70 LAST TIMER                                         LTIME2   :
 71 HIGHEST DRT NUMBER                                 HSYSDRT  :
 72 POWERFAIL                                          PWRFAIL  :
 73 SYSTEM UP FLAG                                     SYSUP    :
 74 SYSTEM CONSOLE LOGICAL DEVICE NUMBER               CONSLDEV :
 75 COLD LOAD COUNT                                    CLOADID  & RESERVED
 76 SHARED FCB DST NUMBER                              SHFCBDST & FOR FILE
 77 MONITORING FLAGS                                   MONITOR  & SYSTEM.
100 MAX NUMBER OF SPOOL KILLOSECTORS                   MAXSSECT1&
101 MAX NUMBER OF SPOOL KILLOSECTORS                   MAXSSECT2&
102 CURRENT NUMBER OF SPOOL KILLOSECTORS               NUMSSECT1&
103 CURRENT NUMBER OF SPOOL KILLOSECTORS               NUMSSECT2&
104 NUMBER SECTORS/SPOOLFUL EXTENT                     EXTSSECT &
105 MAX CODE SEGMENT SIZE
106 MAX NUMBER OF CODE SEGMENTS PER PROCESS
107 MAX STACK SIZE (MAX DATA)
110 DEFAULT STACK SIZE
111 MAX EXTRA DATA SEGMENT SIZE
112 MAX NUMBER OF EXTRA DATA SEGMENTS PER PROC
113 C. I. WORKING SET POINTER                          CIWSP
114 UPDATE LEVEL                                       UPDATEL
115 FIX LEVEL                                          FIXL
116 VERSION LEVEL                                      VERSION
117 DEFAULT CPU TIME LIMIT
120 NUMBER OF SECONDS TO LOGON
121 JOBSYNCH BITS   (13:3)
122 EXTERNAL PLABEL OF INITIATE
123 INTERNAL LABEL OF INITIATE
124      MESSAGE CATALOG -
125      DISC ADDRESS
126 SL.PUB.SYS LDEV   /SL.PUB.SYS
127      DISC ADDRESS
130      DIRECTORY
131      DISC ADDRESS
132 SPOOLINDEX                                                  # RESERVED
133                                                             # FOR CS
134                                                             #
135 CSIOWAIT PLABEL                                             #
136                                                             #
137 CSVERSION                                                   #
140 CCLOSE PLABEL                                               #
141 LOGICAL PROCESS TABLE        (PROGEN)         0
142 LOGICAL PROCESS TABLE        (MAM)            1
143 LOGICAL PROCESS TABLE        (UCOP)           2
144 LOGICAL PROCESS TABLE        (PFAIL)          3
145 LOGICAL PROCESS TABLE        (DEVREC)         4
146 LOGICAL PROCESS TABLE        (PRMSG)          5
147 LOGICAL PROCESS TABLE        (STMSG)          6
150 LOGICAL PROCESS TABLE        (LOG)            7
151 LOGICAL PROCESS TABLE        (LOAD)           8
152 LOGICAL PROCESS TABLE        (IOMESSPROC)     9
153 LOGICAL PROCESS TABLE        (SYSIOPROC)      10
154 LOGICAL PROCESS TABLE        (RESERVED)       11
155 EXTERNAL PLABEL OF "TERMINATE"
```

EXAMPLE PAGE LISTING OF MESSAGE CATALOGUE

Catalogue # | Reply Flag | Reply Type | Max String Len | I/O Type | Prefix # | Format # | I/O Control parameter | Message Length | # Parms | Parameter Description

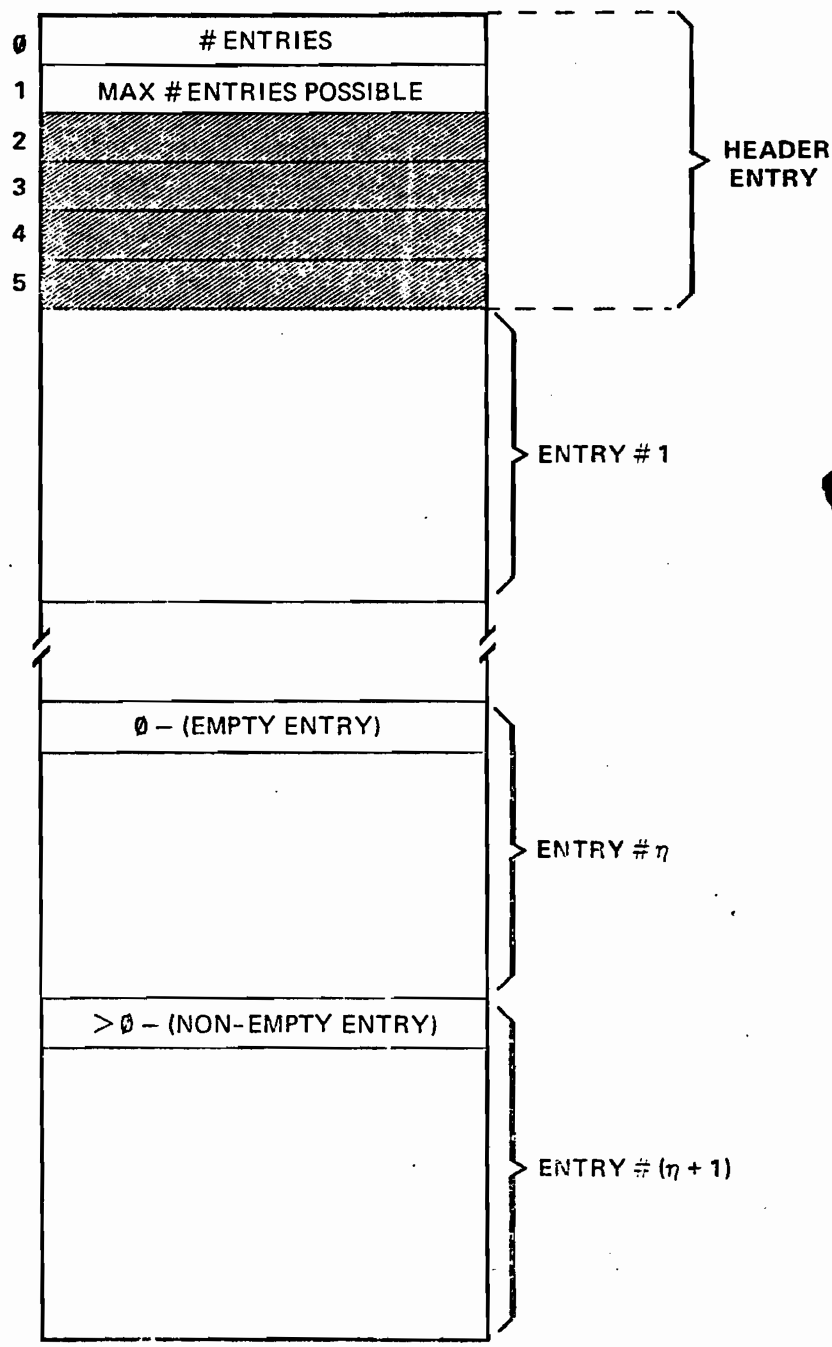
```
CAT# PNTR RFLAG RTYPE MAXSL IOTYPE PR# FMT# IOCONTR MSGL #PAR T P  T P  T P  T P
26   163  0     0     0     0      0   1    %000000 25   1    1 24
LOGON TIMED-OUT ON LDEV#

CAT# PNTR RFLAG RTYPE MAXSL IOTYPE PR# FMT# IOCONTR MSGL #PAR T P  T P  T P  T P
27   182  0     0     0     1      0   1    %000200 29   0
VIRTUAL DEVICE DIRECTORY FULL

CAT# PNTR RFLAG RTYPE MAXSL IOTYPE PR# FMT# IOCONTR MSGL #PAR T P  T P  T P  T P
28   198  0     0     0     1      2   1    %000020 22   1    2 21
JOB MASTER TABLE FULL

CAT# PNTR RFLAG RTYPE MAXSL IOTYPE PR# FMT# IOCONTR MSGL #PAR T P  T P  T P  T P
29   204  0     0     0     1      2   3    %000200 1    1    2 0


CAT# PNTR RFLAG RTYPE MAXSL IOTYPE PR# FMT# IOCONTR MSGL #PAR T P  T P  T P  T P
30   211  0     0     0     1      4   0    %000000 2    2    2 0  2 1
/

CAT# PNTR RFLAG RTYPE MAXSL IOTYPE PR# FMT# IOCONTR MSGL #PAR T P  T P  T P  T P
31   234  0     0     0     0      6   0    %000000 38   0
COMMAND IGNORED - NOT ALLOWED IN BREAK

CAT# PNTR RFLAG RTYPE MAXSL IOTYPE PR# FMT# IOCONTR MSGL #PAR T P  T P  T P  T P
32   249  0     0     0     0      6   0    %000000 21   0
ONLY ALLOWED IN BREAK

CAT# PNTR RFLAG RTYPE MAXSL IOTYPE PR# FMT# IOCONTR MSGL #PAR T P  T P  T P  T P
33   17   0     0     0     0      6   0    %000000 25   0
PREMATURE JOB TERMINATION

CAT# PNTR RFLAG RTYPE MAXSL IOTYPE PR# FMT# IOCONTR MSGL #PAR T P  T P  T P  T P
34   25   0     0     0     0      6   0    %000000 7    0
IGNORED

CAT# PNTR RFLAG RTYPE MAXSL IOTYPE PR# FMT# IOCONTR MSGL #PAR T P  T P  T P  T P
35   36   0     0     0     0      6   0    %000000 14   0
END OF SESSION

CAT# PNTR RFLAG RTYPE MAXSL IOTYPE PR# FMT# IOCONTR MSGL #PAR T P  T P  T P  T P
36   45   0     0     0     0      6   0    %000000 10   0
END OF JOB

CAT# PNTR RFLAG RTYPE MAXSL IOTYPE PR# FMT# IOCONTR MSGL #PAR T P  T P  T P  T P
37   57   0     0     0     0      6   0    %000000 16   0
INVALID RESPONSE

CAT# PNTR RFLAG RTYPE MAXSL IOTYPE PR# FMT# IOCONTR MSGL #PAR T P  T P  T P  T P
38   68   0     0     0     0      6   0    %000000 14   0
END OF PROGRAM

CAT# PNTR RFLAG RTYPE MAXSL IOTYPE PR# FMT# IOCONTR MSGL #PAR T P  T P  T P  T P
39   84   0     0     0     0      6   0    %000000 23   0
FILE EQUATION NOT FOUND

CAT# PNTR RFLAG RTYPE MAXSL IOTYPE PR# FMT# IOCONTR MSGL #PAR T P  T P  T P  T P
40   95   0     0     0     0      6   3    %000200 14   0
FILE NOT FOUND
```

REPLY INFORMATION TABLE

THE REPLY INFORMATION TABLE IS AN EXTRA DATA SEGMENT (DST $\# 23_{10}$). ENTRIES ARE OF FIXED LENGTH AND NO PROVISION IS MADE FOR OVERFLOW OTHER THAN AN ERROR RETURN. EVERY MESSAGE REQUIRING A REPLY SENT VIA PROCEDURE PUTMSG WILL HAVE AN ENTRY PUT IN THE RIT. ENTRIES ARE REMOVED BY THE PROGENITOR WHEN THE REPLY IS RECEIVED.

TABLE FORMAT:

(6 WORDS PER ENTRY)



REPLY INFORMATION TABLE ENTRY FORMAT

Ø	PROCESS NUMBER (PIN)	
1	DST # (FOR REPLY)	
2	BUFFER ADDRESS (DST RELATIVE)	
3	REPLY TYPE EXPECTED	
4	TIME STAMP	FROM TIMER INTRINSIC
5		

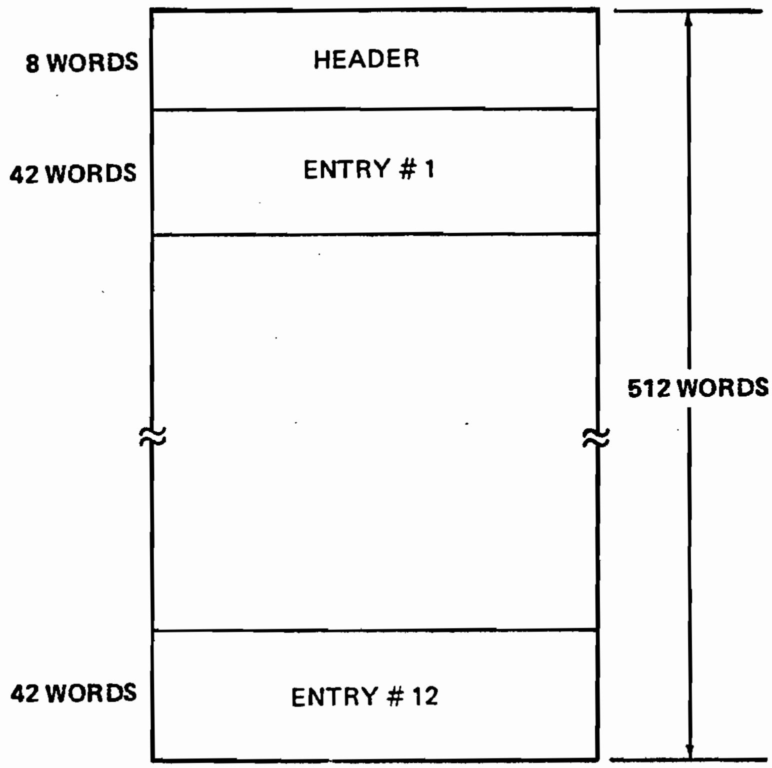
NOTE: PROCESS NUMBER = Ø MEANS ENTRY IS EMPTY

REPLY TYPE = Ø FOR NUMBER (NUM)
= 1 FOR YES OR NO (Y/N)
= 2 FOR STRING (SXX)
= 3 FOR YES, NO, OR NUMBER (YN#)

MESSAGE STORAGE AREAS

THERE ARE TWO MESSAGE STORAGE AREAS (IDENTICAL IN LENGTH AND FORMAT). ONE IS FOR "STANDARD" MESSAGES (DST # 34_{10}) AND THE OTHER IS FOR "PRE-EMPTIVE" MESSAGES (DST#33_{10}). ENTRIES ARE OF FIXED LENGTH (42 WORDS). EACH TABLE IS 512_{10} WORDS LONG AND CAN THUS ACCOMMODATE 12 ENTRIES PLUS A HEADER WHICH IS 8 WORDS LONG.

GENERAL TABLE FORMAT



HEADER FORMAT (MESSAGE STORAGE AREA)

Ø	# ENTRIES TABLE CAN HOLD (CURRENTLY = 12)
1	
2	1st PROCESS IN WAITING LIST (Ø IF NONE)
3	
4	ADR. OF 1ST POSSIBLE ENTRY
5	ADR. OF LAST POSSIBLE ENTRY
6	ADDRESS OF 1ST ENTRY IN QUEUE
7	ADDRESS OF LAST ENTRY IN QUEUE

WORD Ø – CAPACITY OF THIS TABLE IN ENTRIES.

WORD 1 – NOT USED

WORD 2 – PIN OF 1st PROCESS IN WAITING LIST. IF PUTMSG IS CALLED AND NO ENTRIES ARE AVAILABLE IN THE STORAGE AREA, THE CALLING PROCESS IS IMPEDED AND ADDED TO A WAITING LIST (LINKED VIA IMPEDED QUEUE POINTER IN PCB ENTRY). WHEN AN ENTRY BECOMES AVAILABLE, ALL WAITING PROCESSES ARE UN-IMPEDED AND ALLOWED TO COMPETE FOR THE AVAILABLE ENTRY.

WORD 3 – NOT USED.

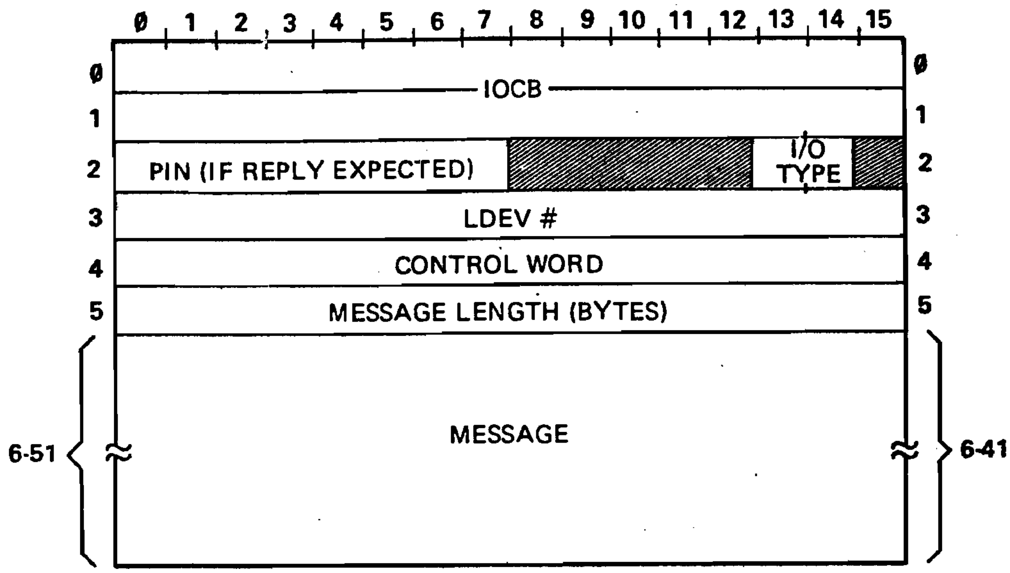
WORD 4 – SEGMENT RELATIVE ADDRESS OF 1st POSSIBLE ENTRY (CURRENTLY 8).

WORD 5 – SEGMENT RELATIVE ADDRESS OF LAST POSSIBLE ENTRY (CURRENTLY 47Ø).

WORD 6 – ADDRESS OF FIRST ENTRY IN QUEUE

WORD 7 – ADDRESS OF LAST ENTRY IN QUEUE

ENTRY FORMAT (MESSAGE STORAGE AREA)



WORDS Ø,1	– IOCB. WORD Ø IS USED TO INDICATE WHETHER OR NOT THE ENTRY IS AVAILABLE. THIS WORD IS SET TO Ø BY PUTMSG AND NON-ZERO BY ATTIO. WORD 1 IS USED AS A POINTER TO THE NEXT ENTRY IN THE QUEUE (Ø IF NONE). THIS WORD IS SET BY PUTMSG.
WORD 2 . (Ø:8)	– PROCESS NUMBER IF MESSAGE REQUIRES A REPLY (Ø OTHERWISE).
. (13:2)	– I/O TYPE.
WORD 3	– LOGICAL DEVICE NUMBER WHERE MESSAGE IS TO BE SENT.
WORD 4	– CONTROL WORD. THIS VALUE IS PUT IN QPAR1 WHEN ATTIO IS CALLED.
WORD 5	– MESSAGE LENGTH IN BYTES.
WORDS 6-51	– MESSAGE IN ITS FINAL FORM.