SECTION 2
GENERAL OVERVIEW

AOS (Advanced Operating System) is a general purpose, disk based operating system that controls and monitors user program processing on the Eclipse computers. AOS takes responsibility for many program control, input/output, and file access functions.

Using AOS, you can manage and control a wide variety of environments. This system is at once a multi-User system suitable for computational processing, and a real-time system suitable for demanding process control applications. Some of the features that AOS includes:

1) Simultaneous support of time sharing, multi-programming batch and real-time applications.

2) Efficient management of system and User resources, based on measured changes in system environment or performance.

3) A complete hardware/software security system, providing file access protection, process integrity, and system security.

4) A multi-tasking environment

5) A generalized code and data sharing facility

6) Device-independent I/O access procedures that are both powerful and easy to use.

7) Accounting procedures accessed by a descriptive REPORT Utility.

8) Communications between concurrently-existing processes (IPC).

9) Spooled output to character-oriented peripheral devices.

Figure 2.1 illustrates a typical smaller AOS hardware configuration on which the above features may be used.

Troubleshooting an AOS system is much more complex than an RDOS/INFOS system. Under an RDOS system, the Operating System executes in an unmapped mode ("absolute") and each of the hardware User maps was allocated to one of the two grounds - back-ground and fore-ground. Other than the task scheduler and some task call modules which reside within the User's area, a failure was very distinct. Either the failure occurred in a User area or the failure occurred in the

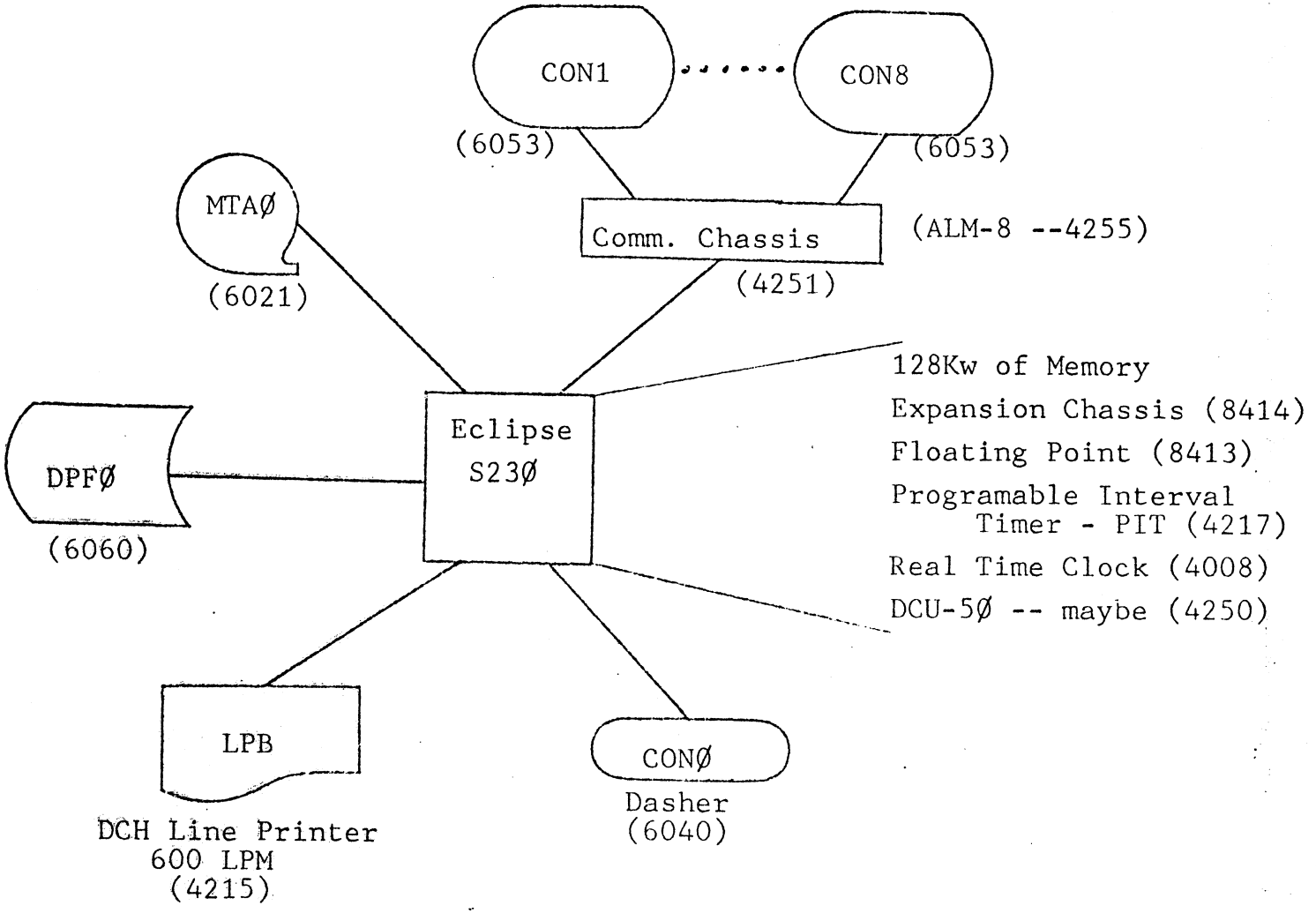


Figure 2.1 A Typical Smaller AOS Hardware Configuration

Operating System. A single variable could be checked to determine which area. INFOS took this concept a step further by allowing the Operating System to run in a mapped mode. Now there were three environments using the two hardware maps - the Operating System and the back-ground/fore-ground. Under both RDOS and INFOS (with the exception of the "window-mapping"), the Operating System executes from a fixed area of core within the lower 32Kw section. This also facilitates troubleshooting an Operating System problem if an intermittent core failure is suspected.

AOS takes the INFOS mapping concepts yet another step further. AOS runs mapped always using the "B" User Map and the "Users" use the "A" User Map. In addition, only the first 16Kw (approx.) executes in a fixed area of core. The second 16Kw is grabbed in 1Kw pages on an "as-needed" basis for tables, overlay areas, buffer areas, etc. Because of this, AOS effectively executes, at one time or another, from most memories in a given system. This causes much more activity on <u>all</u> the memories in a system <u>not</u> just the lower 32Kw. It is therefore not a valid comparison when troubleshooting a system to state "that RDOS/INFOS executes without failure which proves that the hardware is O.K." when AOS is failing. This makes troubleshooting harder because the Operating System can not be relegated to a specific area of core.

As if the random-core usage didn't leave enough "gray" areas when troubleshooting, AOS also has Operating System functions that are not executed within the Operating System space. There are programs that use the "B" User Map, act like user programs yet perform interface features that would have previously been performed by the Operating System itself. These programs are:

1) <u>Peripheral Manager (PMGR)</u>- This program (process) handles the I/O to all <u>byte type</u> devices including the ALM irrespective of whether a DCU-5Ø is involved. If a DCU-5Ø is involved, the PMGR loads the DCU-5Ø memory with the ALM handling code, then interfaces with it on an "as-needed" basis.

2) <u>Ghost</u> - This program is transparent to the User (hence, the name) and handles the following facilities for the User:

a) Debugger
b) all record I/O modules and associated data
c) some miscellaneous system calls

Since the Ghost is used by many User processes, the majority of its code is "shared" among those processes.

The PMGR will exist in all systems, even in a single User environment with only the Master console (device code 1Ø/11) active. The Ghost usually exists in all systems but will be initiated only when it is first needed. In addition to the above two programs, if a system is running many Users, a monitor program is needed called the Executive (EXEC) which also starts a program to handle spooling to a device such as a line printer.

3) Executive (EXEC) - This program (process) controls User access to AOS and basically has four functions:

a) Log-on and Log-off
b) Batch
c) Spooling
d) Mount/Dismount requests to the operator for magnetic tapes and disks.

The EXEC is initialized once the system is executing and is usually initialized by the "UP macro".

4) Line Printer Spooling (XLPT) - This program (process) is initiated by the EXEC and handles spooling to the line printer including header/trailer pages, column/line limits, etc.

5) Plotter Spooling (XPLT) - This program (process) is initiated by the EXEC and handles spooling to the plotter.

6) Paper Tape Punch (XPTP) - Same as XPLT except it is for the paper tape punch.

The above discussion was intended to illustrate that a system failure can not automatically be isolated to a specific component in the system either hardware or software. For example, if suddenly all the EXEC lines running off of a DCU-5Ø/ALM combination quit responding (master console is O.K.), is the problem the DCU5Ø hardware,

the program that the PMGR loaded into the DCU5Ø memory, or ??

The best and most reliable way to troubleshoot an AOS system is to:

a) Understand the functions of each of the software components that make up AOS. You do not need a listing, a functional overview is usually enough.

b) Analyze the symptoms of the problem. Does it happen when a specific function is done; same time of day; high level of system activity; etc.

c) Keep an open mind about where the failure lies in hardware or software. From past history, if you make a snap decision, you will probably be wrong.

d) Make an initial "hypothesis" then try to support it. This could entail running with hooks in the system; running on a console vs batch; running with the EXEC or Comm gear not active; etc.

e) Every time you make a test run re-evaluate the information that was acquired. If the current theory is supported, continue onward; if it is not, re-evaluate the data and develop a new hypothesis.

Using the above, you should be able to define the problem as either hardware or software, then take appropriate action. I have recently seen a problem that was initially classed as a "disk sub-system" problem when only both Unit Ø and Unit #1 were run. During the trouble-shooting of the problem, the hypothesis changed from the disk sub-system to the EXEC/ALM area and finally to a problem with a file on the disk pack. Looking at the initial problem description from the information available and the "failure mode" at that time, a problem with a disk file would never have been suspected.

I Disk Formatter (DFMTR)

The Disk Formatter, DFMTR, is a stand-alone program that defines a "Logical Disk" (LD) and because of this will destroy any previous disk files and structure. LD's form the basis of the AOS disk file structure. An LD is basically a combination of physical disks that are defined to the Disk Formatter as one logical unit. For example, if two 6Ø6Ø 96 Mbyte drives were defined as one logical unit, it would appear logically to AOS as a 192 Mbyte disk unit (disregarding any blocks used for system overhead). Figure 6.14.1 illustrates the structure of a Logical Disk comprised of one 6Ø45 disk. Note that the removable and fixed sections are treated as seperate units by AOS. A couple of thoughts:

a) An LD can be comprised of up to eight physical units.

b) Except on the 6Ø45 type disks, a physical unit should be made equal to a Logical Disk. This way if a drive unit were to break, only that one LD would be affected, not the whole system.

c) Multiple single physical unit LD's increase the flexibility of a system because each can contain only data needed for a particular operation.

d) If an LD must be composed of multiple physical disk units of different transfer rates, make the last physical disk in the LD a high speed device because the "Bit Map" resides on it.

e) If a CDC controller and a 6Ø6Ø/6Ø61 controller, or two 6Ø6Ø/6Ø61 controllers are both to be used on a system, make the highest priority controller (closest to CPU) run the master logical disk. This will usually minimize "data late" activity.

f) In an LD comprised of multiple physical disks, AOS will determine on which physical disk information is stored. If you do not want this to happen, make a physical disk equal to a logical disk.

Once the configuration of the various physical units has been thought out, the DFMTR program can be run.

Once the Disk Formatter has been "booted" from Magnetic Tape and all the questions answered, it will:

a) run any combination of five data patterns on the disk to check for additional bad disk blocks.

b) define a Logical Disk and store this information in the Disk Information Block (DIB). Refer to Section 12, "PARS/FS" for more information on the DIB.

c) build a root directory or a system root directory (:).

d) reserve space for an AOS operating system if a "System Disk".

e) install a boot strap on each physical disk in the LD.

f) build a LD bit map near the front (outer cylinders) of the last physical disk in the LD.

Figure 3.1 illustrates a sample DFMTR dialogue where no bad blocks were found and Figure 3.2 illustrates a sample DFMTR dialogue where two bad blocks were found. These bad block addresses should be copied down as there is no way to get them short of running DFMTR again or reading areas of the disk with DSKED.

Referring to Figure 3.1, we will discuss the first few dialogue questions. These are:

1) "Logical Disk Unique I.D." - This name is used by the system when you initialize an L.D. to determine which physical disks belong to the L.D. It is written in the Disk Information Block of each physical disk in the L.D..

2) "Logical Disk Name" - Used by the system as the name of the Root directory if the L.D. has been "grafted" onto the system. Figure 6.6.2 shows two one-disk L.D.'s where the root directory of the "grafted" L.D. is referenced as the Logical Disk Name.

```
FROM MTØ:2↵
--


AOS DISK FORMATTER REV Ø2.1Ø

LOGICAL DISK UNIQUE I.D. (1 TO 6 CHARS) ? GILGIS↵

LOGICAL DISK NAME (1 TO 31 CHARS) ? GIL2↵

ACCESS CONTRO L LIST
USER NAME O R TEMPLATE (1 TO 15 CHARS) ? OP↵
PRIVILEGES (R, W, A, E, O) ? OWARE↵
USER NAME OR  TEMPLATE (1 TO 15 CHARS) ? ↵

NUMBER OF PHYSICAL DISKS (1TO 8) ? 1↵

-- DISK NUMBER 1
DISK UNIT NAME ? DPD1Ø↵
DEVICE CODE ? ↵
USE OLD BAD BLOCK TABLE ? NO↵
SURFACE ANALYSIS ? Y↵
WHAT PATTERNS (1-ØØØØØØ, 2-177777, 3-Ø52525, 4-125252, 5-155555) ? 5↵
— RUNING PATTRN 155555
-- Ø BAD DISK BLOCKS
SYSTEM DISK ? Y↵

-- LOGICAL DISK CREATED

DONE!
```

Refer to Table 3.1

Figure 3.1

Sample DFMTR Dialogue from the Console

In the majority of cases, the two above names should be made the same, and recorded to prevent the names being used again in the future. If you want all the User Files to reside on a seperate disk pack, the LD can be names "UDD". This LD must be initialized prior to EXEC being started. EXEC will now put all the User files in sub-directories under :UDD. Refer to Figure for a sample directory structure.

3) "User Name or Template"
"Privileges (O,W,A,R,E)" - This pair of questions refers to the privileges and User names that can initialize an L.D. More is discussed in the Operators Guide on Pages 3-4/3-5. For any test packs specify "OP" and "OWARE". The maximum number of pairs is seven and when all have been entered, type a "CR" to the "User Name ...." question.

4) "Number of Physical Disks" - This is the number of Physical Disks in the L.D.. Note that in order to initialize an L.D., all the physical disks in the L.D. must be online and ready at the same time.

The following set of questions will be asked for every physical disk in the L.D. If there are three physical disks, the questions will be asked three times.

5) "Disk Unit Name" - This is the name that AOS refers to the disk. Table 3.1 contains the various disk names vs hardware types.

6) "Device Code" - If this is the same device code as shown in Table 3.1 for the "Disk Unit Name", a "CR" can be typed. Otherwise, enter the device code.

7) "Use Old Bad Block Table" - If disk has never been initialized before, answer "N".

8) "Surface Analysis" - Should be answered "Y" so bad blocks are checked for.

9) "What Patterns ...." - Any combinations of the five patterns can be run. Type the numbers with no spaces between them ending with a "CR".

```
FROM MT0:2
--


AOS DISK FORMATTER REV 1.4

LOGICAL DISK UNIQUE I.D. (1 TO 6 CHARS) ? GILGIS

LOGICAL DISK NAME (1 TO 31 CHARS) ? GIL23

ACCESS CONTROL LIST
USER NAME OR TEMPLATE (1 TO 15 CHARS) ? OP
PRIVILEGES (R, W, A, E, O) ? OWARE
USER NAME OR TEMPLATE (1 TO 15 CHARS) ?

NUMBER OF PHYSICAL DISKS (1 TO 8) ? 1

-- DISK NUMBER 1
DISK UNIT NAME ? DPF0 (Refer to Table 3.1)
DEVICE CODE ?
USE OLD BAD BLOCK TABLE ? N
SURFACE ANALYSIS ?   Y
WHAT PATTERNS (1-000000, 2-177777, 3-052525, 4-125252, 5-155555) ? 5
-- RUNNING PATTERN 155555
-- 2 BAD DISK BLOCKS
      HEAD:000021       SECT:000022       CYL:000240
      HEAD:000021       SECT:000023       CYL:000240
SYSTEM DISK ? Y

-- LOGICAL DISK CREATED

DONE!
```

Figure 3.2 Sample DFMTR Dialogue with "bad" blocks found.

10) "System Disk" - If an AOS operating system is ever to be installed on the disk, answer "Y". If the disk will never be booted, answer "N". Answering "Y" will reserve extra space on the front of the disk as shown on Page 6-1ØØ, approx. 34Ø extra disk blocks.

After the above questions have been asked the required number of times, the message "Logical Disk Created" is typed out.

The time required to run DFMTR will depend on the surface analysis being done and the number of patterns being run. The following table lists the time per pattern for the various disk types:

Disk Type	Time (per pattern)
6Ø6Ø	17 min
6Ø61	34 min
6Ø3Ø	2 min
6Ø45	113 sec
4231	17 min
6Ø63	
6Ø65	

Note that while DFMTR is executing, the "RUBOUT" Key can be used to delete a character; a "^U" will delete a line; and a "^C^A" will restart the DFMTR program over again. Also the following "Page Zero" physical locations are used by DFMTR to store useful information:

Location	Meaning
Ø	Disk Device Code
1	Disk Unit #
3	Disk Error Retry Count
4	Disk Status
2	DFMTR entry point

Note that location 4 can be very useful if DFMTR were to "hang" and could be used to determine whether there was an error on the last disk operation.

Model	Default Device Code or Controller	Unit (Drive)	Name of Removable-Disk Unit	Name of Fixed-Disk Unit
6045 (Dual moving-head units--1 fixed, 1 removable)	33	first pair second pair third pair fourth pair	DPD0 DPD1 DPD2 DPD3	DPD4 DPD5 DPD6 DPD7
	73	first pair second pair third pair fourth pair	DPD10 DPD11 DPD12 DPD13	DPD14 DPD15 DPD16 DPD17
6030 (Moving-head: diskette)	33	first second third fourth	DPD0 DPD1 DPD2 DPD3	
	73	first second third fourth	DPD10 DPD11 DPD12 DPD13	
4231 (Moving-head: top-loading)	33	first second third fourth	DPE0 DPE1 DPE2 DPE3	
	73	first second third fourth	DPE10 DPE11 DPE12 DPE13	
6060 (Moving-head: top-loading pack)	27	first second third fourth	DPF0 DPF1 DPF2 DPF3	
	67	first second third fourth	DPF10 DPF11 DPF12 DPF13	
6063 (Fixed-head disk)	26	first second third fourth		DKB0 DKB1 DKB2 DKB3
	66	first second third fourth		DKB10 DKB11 DKB12 DKB13

Note that the Model 6045 and 6030 (diskette) units share a common controller name. The Model 6045-type disk controller can support both types of units. Remember that the Model 6045 disk units are pairs of units. If you have a pair, and one of them is DPD0 to hold your system disk for loading, the fixed disk unit of the pair will be DPD4. This leaves DPD1, DPD2 and DPD3 to be dialed on diskette units, if any are on this controller.

Table 3.1 Hardware vs. AOS Disk Names

For your reference, the following is the approximate size of the bit map on the last physical disk of an L.D."

Disk Type	#blocks (decimal)*
4231	44.
6Ø45	3/platter
6Ø6Ø	46.
6Ø61	92.

("*" the numbers must be added up for all the physical units in the L.D.)

Refer to Table 3.1a for a list of the DFMTR error messages along with their meanings. If any "Bad Blocks" are found during DFMTR execution, the bits in the disk bit map will be set to "1" and the block addresses will be entered in a Bad Block table which is contained in Physical Disk Block #2. There is no "re-mapping" done to alternate areas such as RDOS does. This Bad Block table will only be used by FIXUP when re-constructing the disk bit map.

Note that there is a bug in DFMTR Rev. 1.ØØ when run on a 6Ø61 (192 Mbyte) which causes it to:

1) Calculate high cylinder # incorrectly.
2) Prints erroneous bad block # when hitting bad block at high cylinder #'s

This problem has been corrected in DFMTR Rev. 1.Ø2 or can be "patched" in the Rev. 1.ØØ version.

Table 3.1a
Disk Formatter Error Messages

Any of the following messages may occur while you are running the Disk Formatter utility.

The error messages below appear on the system console.

Message	Meaning
--DISK ERROR STATUS = xxxxxx	The Disk Formatter has encountered a disk block from which it cannot read or write. It returns to the DISK UNIT NAME? query for this disk. xxxxxx the hardware device status number (octal), which is meaningful to your Data General representative as a diagnostic indicator. *Action:* Retry.
--FATAL DISK ERROR ABORT! STATUS = xxxxxx	The Disk Formatter has encountered an unrecoverable error while attempting to read or write to a disk block. Possible hardware failure. xxxxxx the hardware device status number (octal), which is meaningful to your Data General representative as a diagnostic indicator. *Action:* Press the CONT switch on the front panel and restart the Disk Formatter.
--CAN'T FORMAT DISK	The Disk Formatter cannot format this disk. It may have encountered bad blocks in the area of the disk which it reserves for system data. *Action:* Format another disk.
--TOO MANY BAD DISK BLOCKS	The Disk Formatter has found more than 20 bad blocks on this disk while performing surface analysis. It repeats the DISK UNIT NAME? query for this disk. *Action:* Retry. If the same message recurs during a second surface analysis, format another disk.

II System Installer (INSTL)

The Installer (INSTL) is used to perform two functions, it will:

a) install a disk bootstrap on any disk it processes without destroying any disk files. This can be used to replace the bootstrap if it gets clobbered. Note the DFMTR will also install a bootstrap but will destroy the disk.

b) install an Operating System that you specify into the area of the disk that was reserved by DFMTR when you answered "Y" to "System Disk". If you try to install an operating system on a non-system disk, the error message:

"Disk Not a System Disk"

will be typed on the console.

Figure 3.3 illustrates a sample INSTL dialogue. The "Disk Unit Name" can be gotten from Table 3.1 and uses the same correspondence as described for the DFMTR.

While answering questions, the "RUNBOUT" Key can be used to delete a character; a "^U" will delete a line; and a "^C ^A" will restart the INSTL program over again. Also the following "Page Zero" physical locations are used by INSTL to store useful information:

Location	Meaning
Ø	Disk Device Code
1	Disk Unit #
3	Disk Error Retry Count
4	Disk Status
2	INSTL entry point

Note that location 4 can be very useful if INSTL were to "hang" and could be used to determine whether there was an error on the last disk operation.

Refer to Table 3.1b for a summary of the INSTL error messages along with their meanings.

```
FROM MTØ:3↵
---


AOS INSTALLER REV Ø2.1Ø

DISK UNIT NAME ? DPD1Ø↵        } Refer to Table 3.1
DEVICE CODE ?↵                 }
-- DISK BOO TSTRAP INSTALED
INSTALL A SYSTM ? Y↵
FROM MAG TAPE (M) OR DISKETTE (D) ? M↵
SYSTEM FROM MTAØ, FILE # ? 4↵
-- SYSTEM INSTALLED

DONE!
```

Figure 3.3
Sample INSTL Dialogue on the Console

Table 3.1b
Installer Error Messages

Any of the following messages may appear *on the system console* while you are running the installer.

Message	Meaning
--DISK ERROR, STATUS=xxxxxx	The Installer has encountered a disk block from which it cannot read or write. It repeats the DISK UNIT NAME? query for this disk. *Action:* Retry. If the same message recurs at the same point, see Appendix E.
--DISK FORMAT REVISION NUMBER MISMATCH	The disk has a different revision number from the Installer program (indicating the Disk Formatter used has a different revision number from the Installer). The Installer repeats the DISK UNIT NAME? query. *Action:* Reprocess the disk with an Installer of the same revision number as the disk, or reformat the disk using a Disk Formatter of the same revision number as this Installer.
--MAG TAPE ERROR	The Installer has encountered a magnetic tape block from which it cannot read or write. It repeats the DISK UNIT NAME? query for this disk. *Action:* Retry. If the same message recurs at the same point, see Appendix E.
--FILE DOES NOT EXIST	The entry you typed does not represent a file on the tape mounted on magnetic tape unit 0. The Installer repeats the SYSTEM FROM FILE? query. *Action:* Retry with correct file number.
--READY MAG TAPE UNIT 0, STRIKE ANY KEY	You have typed Y after the INSTALL A SYSTEM? query, but the magnetic tape drive is off line. *Action:* Turn the magnetic tape drive on line and ready it. Make sure a system tape is on unit 0. Then strike any key on the system console.
--SYSTEM DOES NOT FIT	The system program file (file 4 on a system tape, or system diskette #2) is too big for the area reserved for it by the Disk Formatter. This probably means that you loaded the wrong file. The Installer returns to the SYSTEM FROM FILE? query. *Action:* Retry. If problem persists, contact your Data General representative.

III System Booting

There are two types of Boots when using AOS. The first one is used to boot the pack and also to put required files on a brand new pack. The second one is the normal Boot used in starting the system each day. Each of these will be discussed in a following sub-seciton

To boot the disk (hardware wise), you must:

1. Make sure the CPU is powered on and halted.
2. The disk to be Booted is mounted on disk drive Unit # Ø, on-line, and ready.
3. Press "Reset" on the CPU console
4. Put "1ØØØxx" in the console switches, where "xx" is the disk device code. Standard device codes for Data General devices including disks are shown in Table 9.3.
5. Press "Program Load" on the CPU console

The disk has now been booted, and the dialogue in sub-section A or B will transpire via the console on device code 1Ø/11. Your responses will be underlined in the respective figures.

The above procedure will "boot" the operating system on disk drive unit #Ø. In the case of a multiple drive LD, all the disk packs will have the bootstrap installed on them. The pack with the Operating System need not always be on unit #Ø. If it is on a different unit, say #2, step 4 above can be modified so that the bootstrap will be smart enough to get the Operating System from Unit #2. To do this the format that is put in the console switches is changed to:

"1nØØxx"

where: "xx" is the device code (as before)
"n" is the unit # of the disk with the Operating System installed.

In our case with the O.S. on Unit #2, we would put "12ØØxx" in the switches. Note that the hardware program load will always load the bootstrap from Unit #Ø. It is this bootstrap that checks for the unit number by reading the switches.

Once the boot has been completed and all the questions answered, the Operator CLI (OP CLI) be executing on the Master Console. This process is:

a) Swappable - can be swapped to disk if it "Blocks".

b) has one of the higher priorities (PRI=2)

c) has the "Super User" privilege.

If the "Up Macro" is used, a second CLI will be created to run on the Master Console.

Note that AOS requires a RTC for system operation. If the RTC is missing or inoperative, the system will "hang" prior to the CLI message being typed. This is discussed in more detail in Section 4, "Hangs", in the first couple pages.

A. Installation Boot

Figure 3.4 illustrates a sample dialogue for an initial Boot on DPD1Ø. The "installed" Operating System on the "specified" disk is read into core by the bootstrap which is contained in blocks Ø and 1 of the disk. Refer to Figure 6.14.1. After the bootstrap has done its work, the loaded Operating System is entered at the module "SINIT" and its code executed. SINIT will type out the AOS Revision message such as:

"AOS REV Ø1.Ø6"

then ask some questions. The revision of the AOS Operating System is contained in location 412 of AOS address space in a byte format. The high (left) byte will be the major revision and the low (right) byte will be the minor revision. For example, an AOS Rev 1.Ø6 operating system will be:

412:ØØØ4Ø6

which will be in byte format, a "1" and a "6". Note that place within the specific AOS module where the message is output from is annotated on the right of Figure 3.4.

Be aware that during the system initialization process, where all the questions are asked, a lot of disk and core activity is involved. The ":PROC" and the ":PER" directories are deleted if they existed on the disk and then re-created. This is done with the ":PER" directory so that the proper peripheral specifications for the Operating System being "booted" can be entered.

The questions asked are:

a) "DATE?" - self-explanatory

b) "TIME?" - self-explanatory

c) "Override Default Specs?" - The answer will always be "Y" whenever running the "starter" operating system. The reason is that the disk type must be specified in the following question.

d) "Specify the Master Logical Disk" - All the physical units that comprise the master logical disk must be

```
AOS REV 01.00                                    SINIT + 141'

DTE ? 5 15 77↵                                   SINIT + 716'

TIME ? 16 28↵                                    SINIT +742'

OVERRIDE DEFAULT SPECS ? Y↵                      SINIT + 1272'

SPECIFY THE MASTER LOGICAL DISK                  SINIT + 1346'/1351'
? DPD10↵
?↵
GIL2
                                                 MLDUI + 1322'
SYSTEM PATHNAME ?↵                               SINIT + 1565'

SWAP FILE DEFINITION ?↵                          SINIT + 2757'

INITIAL LOAD ? Y↵                                SINIT + 3235'

UNITNAME ? MTA0↵                                 CLIBT + 447'

AOS CLI    REV 00.37      15-MAY-77      16:29:09
)
```

Figure 3.4
Sample Dialogue for an Initial Boot on DPD1Ø

```
AOS REV 01.00

DATE ? 5 15 77↵

TIME ? 15 45↵

OVERRIDE DEFAULT SPECS ? Y↵

SPECIFY THE MASTER LOGICAL DISK
? DPD10↵
?↵
GIL2                                             If "n" is answered,
                                                 these questions are
SYSTEM PATHNAME ?↵                               not asked.

SWAP FILE DEFINITION ?↵

INITIAL LOAD ?↵

AOS CLI    REV 00.37      15-MAY-77      15:45:29
)
```

Figure 3.5
Sample Dialogue for Subsequent Boots on DPD1Ø

specified. Figure 3.4 shows a master LD comprised of only "DPD1Ø". If it were comprised of DPFØ and DPF1, the question would be answered:

```
? DPFØ↵
? DPF1↵
? ↵
```

An error message will be returned if not all the disks that were specified in DFMTR have been specified.

e) "System Pathname?" - type a "CR".

f) "Swap File Definition?" - type a "CR".

g) "Initial Load?" - In our case, the answer will always be "Y".

h) "Unitname?" - The magnetic tape unit that has the AOS System Tape mounted on it. The answer is of the format "MTAX" where "x" is the tape unit #.

At this point, File 1 will be read in from the mag tape. This "Dump" file contains all the programs that reside in the root directory and necessary to execute the system. These programs are:

1) Command Line Interpreter (CLI)
2) Error Message File (ERMES)
3) Peripheral Manager (PMGR)
4) Disk Formatter (DFMTR)
5) Installer (INSTL)
6) Disk Fixer (FIXUP)
7) Ghost program (GHOST)
8) Debugger modules (DEBUG)
9) Tape boot program (TBOOT)

Also the directories ":UTIL", ":HELP", ":PROC" and ":PER" are created. Various entries depending on the Operating System being run are put in ":PER".

After all the above is done, the PMGR process and the CLI process are initiated by CLIBT (CLI boot). The CLI types a message:

"AOS CLI REV1.Ø6 1Ø-JAN-78 Ø9:45:Ø1"

on the master console and then the ")" to indicate that it is ready for a command. The system is now operational.

Since this is the initial load, additional files must be loaded from the magnetic tape. The CLI commands to do this are:

```
)SUPER ON
*)LOAD/V/R  @MTAØ:6
      .
      .
      .
*)
```

for a magnetic tape on Unit #Ø. Once these files are loaded, the Update Tape must also be loaded. Mount the Update Tape on Mag Tape Unit #Ø and type the following commands:

```
*)DIR :
*)LOAD/V/R  @MTAØ:Ø
      .
      .
      .
*)
```

The "RELEASE NOTICE" and the "UPDATE NOTICE" are both contained in the :UTIL directory and should be printed out and read prior to doing anything else. Their filenames will be:

```
RELEASE
UPDATE.Ø1
UPDATE.Ø2
UPDATE.Ø3
etc.
```

The highest numbered update should correspond with the revision number on the Update Tape that was loaded. For example, "UPDATE .Ø6" corresponds to an Update Tape labeled "AOS UPDATE 1.Ø6". The Update Notice will contain the information and com-

mands needed to continue building a system. These may vary from update to update, so the best procedure is to consult the Update Notice.

B. Normal Boot

Figure 3.5 illustrates a sample dialogue for any normal Boot being done with the "Starter" Operating System. This is the one that was "installed" from File 4 of the AOS System Magnetic Tape. The only difference between Figures 3.4 and 3.5 is that the Root directory files needed to be loaded on the first load as shown in Figure 3.4 and on sequential loads this need not be done.

The reason that the answer to "Override Default Specs" is again "Y", is that the "Starter" Operating System has a CDC (DPEØ) defined as the "Master Logical Disk". In most configurations, this will not be true, so the new "Master Logical Disk" must be specified to AOS. Once your system has been setup, an AOSGEN done, and the tailored Operating System installed on the disk, you can answer "N" to the question from then on.

The "installed" operating system is not the only Operating System that can be Booted. You could have a couple Operating Systems in a directory on the disk, say ":SYSGEN". These could be tailored for specific purposes or could be "test" operating systems which have specific "patches" installed. Any one of these operating systems could be Booted by giving the complete "pathname" of where the operating system is on disk in answer to the "System Pathname?" question. Figure 3.6 illustrates the procedure to Boot a "patched" operating system that resides in the ":SYSGEN" directory. Note that the ".SY" extension should always be appended to the operating system name that is given.

Not only Operating Systems can be Booted. An example of the stand-alone utility, PCOPY, being booted from a system disk is shown in Figure 7.15.1, "Sample PCOPY Dialogue for disk 'Dump'". Since the utilities, DFMTR and INSTL are in the root directory of a system disk, they could also be booted. I would not do this though, as if the wrong disk name is specified, the system disk could be wiped out.

The "Swap File Definition" question is used to change the specifications that were entered for the Swap File to AOSGEN. This parameter can either be a complete disk and should be a

```
AOS REV Ø1.ØØ

DATE ? 5 15 77↵

TIME ? 17 9↵

OVERRIDE DEFAULT SPECS ? Y↵

SPECIFY THE MASTER LOGICAL DISK
?↵
GIL1

SYSTEM PATHNAME ? :SYSGEN:AOS22H.SY↵


SYSTEM SHUTDOWN


AOS REV Ø1.ØØ

DATE ? 5 15 77↵

TIME ? 17 9↵

OVERRIDE DEFAULT SPECS ? N↵
GIL1

AOS CLI    REV Ø1.ØØ      15-MAY-77          17:Ø9:12
)
```

Figure 3.6

Sample Procedure to Boot an Operating System
Other than the one Installed on the Disk

fast disk such as the Fixed Head Disk, DKBx; or a section of the Master Logical Disk which is specified in disk blocks. The Swap File should never be made smaller than the default value of 2ØØØ because on an almost full LD, if a larger size is needed at a later date, a sufficient number of contiguous disk blocks may not exist. If this occurs, an error message:

"Swap File Space Exhausted"

will occur on booting the LD.

When a system crash occurs, FIXUP can delete a file if it finds an inconsistancy that it cannot resolve. This file could be a critical one in the Root directory. The next time the disk is booted, an error message such as:

"Override Default Specs? N↵ "
"Fatal Error 25 - File: PMGR.PR"

which is "File Does Not Exist" or the sytem could "hang" after it types out the Logical Disk Name. In either of these cases, FIXUP must be run again since the initialization got far enough to set the "Use bit" on the disk to "1". Now a reload of the file(s) in the Root directory must be done. This reload procedure is identical to the procedure outlined in sub-section A, "Installation Boot". The "Initial Load?" question is answered "Y" and the files in File 1 of the specified Mag Tape will be read in. The difference is that in most cases the files being read in already exist in the Root directory. Whenever a file that already exists is found, a message:

"Error 26 - File :xxx
Delete Old Copy? Y↵"

will be typed out. A "Y" or a "CR" will cuase the old copy on disk to be deleted and the new copy from Mag tape to be written onto the disk. A "N" will cause the old copy not to be deleted and the load will continue to the next file. After answering all the questions, the "CLI..." message should appear or an additional error message. Figure 3.7 illustrates the error

```
AOS REV Ø1.ØØ

DATE ? 5 15 77↵

TIME ? 17 35↵

OVERRIDE DEFAULT SPECS ? Y↵

SPECIFY THE MASTER LOGICAL DISK
? DPD1Ø↵
?↵
GIL2

SYSTEM PATHNAME ?↵

SWAP FILE DEFINITION ?↵

INITIAL LOAD ?↵

FATAL ERROR: 25     FILE:CLI.PR
```

Figure 3.7 Missing File on System Boot

```
AOS REV Ø1.ØØ

DATE ? 5 15 77↵

TIME ? 17 35↵

OVERRIDE DEFAULT SPECS ? Y↵

SPECIFY THE MASTER LOGICAL DISK
? DPD1Ø↵
?↵
GIL2

SYSTEM PATHNAME ?↵

SWAP FILE DEFINITION ?↵

INITIAL LOAD ? Y↵

UNITNAME ? MTAØ↵

ERROR: 26      FILE: PMGR.PR

DELETE OLD COPY ?↵

ERROR: 26      FILE: DEBUG.OL

DELETE OLD COPY ?↵
```

Figure 3.8 Re-load procedure to re-install missing file(s)

message that occurs when a file is missing during a boot. Figure 3.8 illustrates the "error" messages that occur during a re-load procedure to replace the missing file.

IV AOSGEN

After DFMTR and INSTL are run, the "Starter" system is booted, and both File 6 of the AOS System Tape and File Ø of the Update Tape are loaded, a program called AOSGEN can be run to generate a tailored Operating System for the customer's specific configuration. In some cases, a previous AOSGEN had been done and you only want to add or delete a specific device. AOSGEN therefore gives you the option of editing an old dialogue, ie: "SPEC", file. Figure 3.9 illustrates a sample AOSGEN Dialogue. The questions asked can be broken into four categories:

a) Disk Units
b) Other Peripheral Devices
c) Communications Systems
d) System Parameters

After all the questions are answered, AOSGEN asks whether a "System" is to be built. If the answer is "Y", the system build will be started. If the answer is "N", only the dialogue file will be created. After the "System Build is Completed", AOSGEN will ask whether any of the temporary files should be "saved". I would always answer "Y" and save all the temporary files. Note that the ".TMP" files must be renamed as FIXUP will delete any files that begin with a "?" and end with".TMP".

WARNING

There are instances where one of the programs invoked by AOSGEN (these will be discussed later) will "Bomb". When this occurs, the messages:

"SYSTEM BUILD IN PROGRESS"
"SYSTEM BUILD COMPLETED"

will still appear on the console although no Operating System has been created. I would always recommend do a "FILES" CLI command to verify that the three files:

a) operating-system-name
b) operating-system-name.SY

```
OIR :SYSGEN
*) X AOSGEN
DO YOU WANT TO EDIT AN OLD SPEC? (Y,NEWLINE)
NAME OF NEW SYSTEM?  ILLUST

ACCEPTABLE ANSWERS ARE DISPLAYED IN PARENTHESES. PREVIOUS
VALUES ARE DISPLAYED IN BRACKETS. TYPING A NEWLINE WILL
AUTOMATICALLY ENTER THE PREVIOUS VALUE FOR THAT QUESTION.

DO YOU HAVE A FIXED HEAD DISK CONTROLLER ON DEVICE CODE 26? [N] (Y,N)

DO YOU HAVE A FIXED HEAD DISK CONTROLLER ON DEVICE CODE 66? [N] (Y,N)

ENTER NAME OF DISK CONTROLLER ON DEVICE CODE 33
[DPE3] (DPD, DPE, NONE)

ENTER NAME OF DISK CONTROLLER ON DEVICE CODE 73
[NONE] (DPD1, DPE1, NONE)

ENTER NAME OF DISK CONTROLLER ON DEVICE CODE 27
[NONE] (DPF, NONE)  DPF

ENTER NAME OF DISK CONTROLLER ON DEVICE CODE 67
[NONE] (DPF1, NONE)

DO YOU WANT TO SPECIFY OTHER PERIPHERALS? (Y,NEWLINE)  Y

        (WARNING:
        USING A NEWLINE TO ANSWER "HOW MANY" WILL ALSO CAUSE
        DEFAULT VALUES TO BE USED FOR THE FIRST SYSTEM AND
        PREVIOUS CHARACTERISTICS TO BE USED FOR LATER SYSTEMS.)

        HOW MANY MAGTAPE CONTROLLERS?
        [1] (0,1,2)

        HOW MANY MCAS?
        [0] (0,1,2)

        HOW MANY CONSOLES?
        [1] (1,2)  2

        (DEVICE TYPES: 4010A,6040,4010I,6012,6052,6053,OTHER)
        DEVICE TYPE FOR  CONSOLE #0: [6040]

        IS CONSOLE #0 STANDARD?
        [Y] (Y,N)  Y

        (DEVICE TYPES: 4010A,6040,4010I,6012,6052,6053,OTHER)
        DEVICE TYPE FOR  CONSOLE #1: [6040]  6012

        IS CONSOLE #1 STANDARD?
        [Y] (Y,N)

        HOW MANY LINE PRINTERS?
        [0] (0,1,2)  2
```

Figure 3,9 Sample AOSGEN Dialogue

```
IS LINE PRINTER #0 A DATA CHANNEL LINE PRINTER?
[N] (Y,N) N

IS LINE PRINTER #0 STANDARD?
[Y] (Y,N)

IS LINE PRINTER #1 A DATA CHANNEL LINE PRINTER?
[N] (Y,N) Y

HOW MANY CARD READERS?
[0] (0,1,2)

HOW MANY PAPER TAPE PUNCHES?
[0] (0,1,2)

HOW MANY PAPER TAPE READERS?
[0] (0,1,2)

HOW MANY DIGITAL PLOTTERS?
[0] (0,1,2)

DO YOU HAVE AN ASYNCHRONOUS COMMUNICATIONS SYSTEM (ALM)? (Y,NEWLINE) Y

DO YOU WANT TO SPECIFY A NEW ALM CONFIGURATION? (Y,NEWLINE) Y

    DO YOU HAVE A DCU/00 FOR THESE LINES? [N] (Y,N) Y

    ENTER A SET OF LINE NUMBERS HAVING IDENTICAL CHARACTERISTICS.
    SEPARATE WITH SPACES. 0 1 2 3 4 5

    (DEVICE TYPES: 4010A,6040,4010I,6012,6002,6003,OTHER)
            ENTER DEVICE TYPE: 6003
    (ENTER 'STANDARD' FOR ANY STANDARD WORD.)
            ENTER WORD 0: STANDARD
            ENTER WORD 1: STANDARD
            ENTER WORD 2:
                    LINES/PAGE? STANDARD
                    CHARS/LINE? STANDARD
            ENTER LINE INITIALIZATION WORD: STANDARD

    DO YOU HAVE ANY OTHER LINES ON THIS ALM? (Y,NEWLINE)

DO YOU HAVE A SYNCHRONOUS COMMUNICATIONS SYSTEM (SLM)? (Y,NEWLINE) Y

DO YOU WANT TO SPECIFY A NEW SLM CONFIGURATION? (Y,NEWLINE) Y

    ENTER A SET OF LINE NUMBERS HAVING IDENTICAL CHARACTERISTICS.
    SEPARATE WITH SPACES. 16 17

    ARE THESE LINES FULL OR HALF DUPLEX? (HDX,FDX) FDX

    ARE THESE LINES SWITCHED OR DEDICATED? (SWT,DED) DED
```

Figure 3.9 (cont.) Sample AOSGEN Dialogue

```
        ENTER CRC TYPE. (CRC16,CCITT16,HOME) CRC16↵

        DO YOU HAVE ANY OTHER LINES ON THIS SLM? (Y,NEWLINE)↵

DO YOU WANT TO SPECIFY SYSTEM PARAMETERS? (Y,NEWLINE) Y↵

        DEFAULT SWAP FILE SIZE IS 2000(DECIMAL).
        POSSIBLE SWAP DISK UNIT NAMES ARE:

        DPE0  DPE1  DPE2  DPE3
        DPF0  DPF1  DPF2  DPF3

        ENTER SWAP FILE SIZE OR SWAP DISK UNIT NAME.
        [2000] 4000↵

        SELECTING FROM ABOVE, ENTER NAME(S) OF MASTER
        LOGICAL DISK UNIT(S). SEPARATE WITH SPACES.
        [DPE0] DPF0↵

        ENTER REAL-TIME CLOCK FREQUENCY.
        [10] (10,100,1000,50,60)↵

        DO YOU WANT ACCESS CONTROL ENABLED?
        [Y]  (Y,N)↵

DO YOU WANT TO BUILD A SYSTEM? (Y,NEWLINE) Y↵

SYSTEM BUILD IN PROGRESS.
SYSTEM BUILD COMPLETED.
DO YOU WANT TO SAVE ANY TMP FILES? (Y,NEWLINE) Y↵
SAVE ?007.ZINPUT.TMP? (Y,NEWLINE) Y↵
SAVE ?007.ZOUTPUT.TMP? (Y,NEWLINE) Y↵
SAVE ?007.SPEC.TMP? (Y,NEWLINE) Y↵
SAVE ?007.SBC.TMP? (Y,NEWLINE) Y↵

*)
```

Figure 3.9 (cont.) Sample AOSGEN Dialogue

c) operating-system-name.ST

all exist. If the ".SY" does not exist, an error probably occurred somewhere during the "AOSGEN" procedure. I would type out the "?xxx.ZOUTPUT.TMP" and check for error messages. A sample ZOUTPUT file is shown in Figure 3.12. Note that the AOSGEN causes 2 MASMs and 1 BIND to be done in addition to other CLI commands. If the AOSGEN went sucessfully, the next step is to use the "SYSTAPE macro" which is described in Section 8.5 to generate a "back-up" tape. Any file "renaming" can be done as described in the respective Update Notice, the system can be shut down and the new Operating System INSTL'd. Note that AOSGEN does not create an overlay file (.OL), the system overlays are contained at the end of the ".SY" file.

If you have an existing Operating System at say Rev. 1.Ø6 and a new 1.Ø7 Update Tape arrives, as long as you do not wish to change any of the original system specification answers, a shortcut method can be used. The 1.Ø7 Update Tape can be loaded on the disk after "DIRing" into the Root Directory via:

)LOAD/V/R @MTAØ:Ø ↵

The Update Notice can be printed and its procedures followed. At some point within the procedure a new Operating System can be generated via:

)X AOSGEN <operating-system-name> ↵

This will take the old specification (dialogue) file and use it to build a new operating system. After the AOSGEN command is executed, the next message on the console will be "SYSTEM BUILD IN PROGRESS". The same checks, etc, as described in the previous paragraphs should be done once the "build" is complete.

When AOSGEN is invoked, the various questions as illustrated in Figure 3.9 will be asked. The questions are self-explanatory and the "standard" answers can be used. If more information is needed on any one question, Chapter 4, "How to Generate Your Tailored System", of the AOS Load and Generate

```
SEARCH [!SEARCH] :UTIL
SEARCH
XEQ MASM/B=(?004.IRMG1.TMP,?004.IRMG2.TMP)/MEM=20 (IRMG1,IRMG2) &
?004.SPEC.TMP
XEQ BIND/N/B/W/E/MEM=20/P=ILLUST ?004.SBC.TMP/C
DELETE ?004.IRMG1.TMP.OB,?004.IRMG2.TMP.OB,=ILLUST.SY
RENAME ILLUST.<PR SY>
REV ILLUST.SY 1.07

BYE
```

Figure 3.1Ø Sample "?ØØ4.ZINPUT.TMP" File

```
?004.IRMG1.TMP EXTLIB.LB SZERO
STABLE SCHED DEVLIB.LB DUMPR LIB2.LB DCTLIB.LB
PANIC MEMAP SINIT MLDUT CLIBT ?004.IRMG2.TMP
SCALL CLIEN [LIB3.LB]
```

Figure 3.11 Sample "?ØØ4.SBC.TMB" File

User's Manual (Ø93-217) provides an indepth discussion about each question.

After the questions are answered, AOSGEN, creates a file called "?pid.ZINPUT.TMP" and inputs this in a "batch" mode to a copy of the CLI that it started. Figure 3.14 shows the Process Tree that running AOSGEN will create. Figure 3.1Ø illustrates the sample "?ØØ4.ZINPUT.TMP" file. Note that it has CLI commands to set the SEARCH list, to assemble two modules, IRMG1 and IRMG2 using the parameters given in "?ØØ4.SPEC.TMP" (Figure 3.13), and will also BIND the operating system using the command line in the file "?ØØ4.SBC.TMP" (Figure 3.11). After all of this is done, the AOSGEN internal temporary files are deleted, the system is renamed from a ".PR" to a ".SY" and the current revision set. The "BYE" command terminates the CLI which returns control to AOSGEN which informs you on the console that the "SYSTEM BUILD IS COMPLETED", etc. Again note that the "?...TMP" files will be deleted by FIXUP if not renamed.

Note that when answering the AOSGEN questions, it is a bad practice to type a "CR" to a question if the device is to be AOSGEN'd into the system. This can cause additional questions about the specific device not to be asked. Refer to the "Warning" in the middle of the 1st page of Figure 3.9, "Sample AOSGEN Dialogue".

The "Swap File" has a default size of 2ØØØ blocks. It can be a number from 1ØØ to 7777Ø (decimal) or a "disk unit name" if an entire disk is allocated as the Swap Disk. Note that in the latter case, a fast disk such as the Fixed Head Disk (6Ø65 only) should be used. This Swap File which has a filename of "SWAP.SWAP" or the disk itself, is the place where processes transferred (swapped) out of main memory are held. Refer to Figure 6.5.2, "Process Table and Swap File Linkage" for a graphic illustration. If a swap disk is specified it cannot be a disk used as a Logical Disk and DFMTR need not be run on it. The values specified to AOSGEN become the default values which can be over-ridden at system "Boot" time (See Section 3-III).

```
AOS CLI   REV 01.07      31-JAN-78          8:48:15
) SEARCH [!SEARCH] :UTIL
) SEARCH
:PER,:UTIL,:,:UTIL
) XEQ MASM/B=(?004.IRMG1.TMP,?004.IRMG2.TMP)/MEM=20 (IRMG1,IRMG2) &
&)?004.SPEC.TMP
                          .TITL   IRMG1

                          .TITL   IRMG2

) XEQ BIND/N/B/W/E/MEM=20/P=ILLUST ?004.SBC.TMP/C
ILLUST.PR CREATED BY AOS BINDER REV 01.05 ON 1/31/78 AT 8:49:41
SYMBOL UNDEFINED AT END OF PASS 1  DRLMP IN MODULE LIB3.LB
SYMBOL UNDEFINED AT END OF PASS 1  MCACB IN MODULE LIB3.LB
   :       :       :  :  :  :        :    :    :       :
SYMBOL UNDEFINED AT END OF PASS 1  DEBUG IN MODULE LIB3.LB
SYMBOL UNDEFINED AT END OF PASS 1  DEBSW IN MODULE LIB3.LB
SYMBOL UNDEFINED AT END OF PASS 1  IDUMP IN MODULE LIB3.LB
SYMBOL UNDEFINED AT END OF PASS 1  PWREL IN MODULE LIB3.LB
IRMG1
.DPED
   :
   :
SYOV1
SYOV2
SYOV3


000025 WORDS OF ABSOLUTE DATA
ZMAX: 000361
NMAX: 050144


PARTITION   TYPE   START    END   #OF OVERLAY AREAS
000004   NSHR CD  000447 050143  000001


  AREA  START  LENGTH  PARTITION  #OF OVERLAYS
000000 000000  001000     000004   000061



BIND ERROR
) DELETE ?004.IRMG1.TMP.OB,?004.IRMG2.TMP.OB,=ILLUST.SY
) RENAME ILLUST.<PR SY>
) REV ILLUST.SY 1.07
)
) BYE
AOS CLI   TERMINATING   31-JAN-78          8:53:50
```

Figure 3.12

Sample "?ØØ4.ZOUTPUT.TMP" File

```
EXT DPEDC
EXT DPFDC
EXT MTADC
EXT TTIDC
EXT TTODC
EXT TTI1DC
EXT TTO1DC
EXT LPADC
EXT LPB1DC
EXT DCUDC
EXT SLMDC
DCTAB
DCT 33 DPEDC
DCT 27 DPFDC
DCT 22 MTADC
DCT 10 TTIDC
DCT 11 TTODC
DCT 50 TTI1DC
DCT 51 TTO1DC
DCT 17 LPADC
DCT 57 LPB1DC
DCT 64 DCUDC
DCT SLM SLMDC
DCEND:
UNTAB
DISK 33 DPEDC DPE0 0 DPE1 1 DPE2 2 DPE3 3
DISK 27 DPFDC DPF0 0 DPF1 1 DPF2 2 DPF3 3
TAPE 22 MTADC MTA0 MTA1 MTA2 MTA3 MTA4 MTA5 MTA6 MTA7
DCLP 57 LPB1DC LPB1
SLIN SLM SLMDC SLN16 16. FDX DED CRC16
SLIN SLM SLMDC SLN17 17. FDX DED CRC16
UNEND:
PETAB
PER 11 TTODC TTY CON0
PER 10 TTIDC
PER 51 TTO1DC CRT2 CON1
PER 50 TTI1DC
PER 17 LPADC PLINE LPA
PER 64 DCUDC DCS1 5. MUX
LIN 0. CRT3 CON2,,,,24.*400+80.,
LIN 1. CRT3 CON3,,,,24.*400+80.,
LIN 2. CRT3 CON4,,,,24.*400+80.,
LIN 3. CRT3 CON5,,,,24.*400+80.,
LIN 4. CRT3 CON6,,,,24.*400+80.,
LIN 5. CRT3 CON7,,,,24.*400+80.,
PEEND:
SWAP 4000.
MLDU DPF0
FREQ 10.
ENACC
.END
```

Figure 3.13

Sample "?ØØ4.SPEC.TMP" File

The algorithm for calculating a good swapfile/disk size is:

SWAP = n * ((b*4)+1Ø)

where: SWAP = # of 512-byte blocks needed

n = maximum number of processes that will run concurrently

b = maximum number of 1K byte pages of main memory that the largest process included in "n" will need (good figure is 4Ø)

4 = factor changing 1K byte memory pages to 256 word disk blocks

1Ø = disk blocks which AOS uses for each process

If the "Swap File" becomes too full to be useable, the error message:

"Swap File Space Exhausted"

will be generated and means:

a) the Swap file ran out of space

b) the Swap file became so fragmented that a Process could not be swapped to disk

On "Booting" a LD, never over-ride the default specifications and make the Swap file size smaller than the one that already exists. On an almost full pack, this could cause insufficient contiguous disk blocks to be available to make it larger again. The above error message would be typed on "Booting" if the default spec (larger size) was again used.

The parameters that were AOSGEN'd can be typed out by:

)TYPE/L=@LPA :SYSGEN:<operating-system-name>↵

if EXEC is not up or:

)QPRINT :SYSGEN:<operating-system-name>↵

if EXEC is executing. An example of the information provided in this file is shown in Figure 3.15.

Note that there is a hardware limitation on certain hardware configurations. Some of our peripherals will only execute using

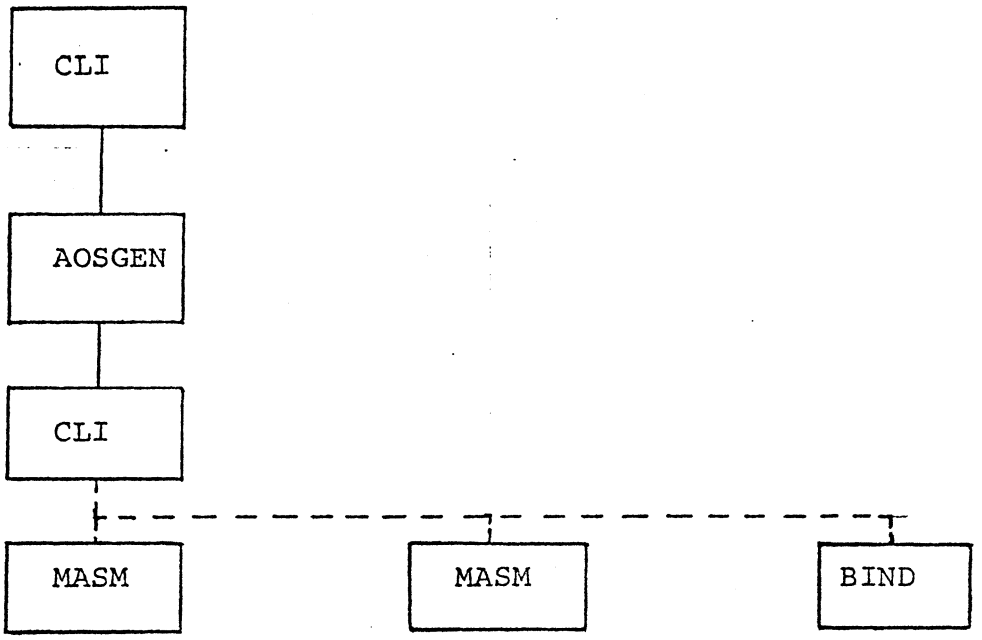


Figure 3.14 Sample AOSGEN Process Tree

the "A" DCH Map on the MMPU1 board. Since theis map only contains 32 slots, only devices can be AOSGEN's whose total slot usage is less than or equal to 32. The slot usage for the devices that can only use the A DCH Map is:

Device	# Slots Used (per controller)
CDC (333Ø)	5
6Ø45	5
MTA	5
DCH LPT	2
DCU-5Ø	17

For example, a CDC, a 6Ø45, a MTA, a DCU-5Ø, and a DCH LPT is an illegal AOSGEN configuration since 34 slots are used. Note that AOSGEN will not flag this configuration as being illegal and the Operating System will do a Panic 12 on trying to Boot it. The reason for the Panic 12, is that the DCU-5Ø is the last device to be assigned map slots. This request for the DCU-5Ø is done by the PMGR and when the DCU-5Ø does not fit, an error (code 15) is returned which causes the PMGR to Panic 12.

While executing AOSGEN, a "^C^A" at any point will cause the process to abort and the CLI prompt will appear on the console.

```
**********************************************************************************

                        ILLUST   SPECIFICATION

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

----DEVICE----------CONTROLLER------DEVICE CODE-------CHARACTERISTICS----------

DISK 0                  DPE             33
DISK 1                  DPE             27

MAGTAPE 0               MTA             22


CONSOLE 0               TTI             10              STANDARD 6040
                        TTO             11
CONSOLE 1               TTI1            50              STANDARD 6012
                        TTO1            51

LINE PRINTER 0          LPA             17              STANDARD
LINE PRINTER 1          LPB1            57              DATA CHANNEL PRINTER
```

Figure 3.15 Sample "Dialogue" File Output

```
********************** ASYNCHRONOUS COMMUNICATIONS SYSTEM **********************

DCU/50: YES

LINES: 0 1 2 3 4 5

DEVICE TYPE: 6053
     WORD 0: STANDARD
     WORD 1: STANDARD
     WORD 2: 24 LINES/PAGE, 80 CHARS/LINE
  LINE INIT: STANDARD


********************** SYNCHRONOUS COMMUNICATIONS SYSTEM ***********************

LINES: 16 17

    DUPLEX: FDX
    SWITCH: DED
  CRC TYPE: CRC16


***************************** SYSTEM PARAMETERS ********************************

SWAPFILE OR SWAP DISK UNIT NAME: 4000

MASTER LOGICAL DISK UNIT NAME(S): DPF0

REAL-TIME CLOCK FREQUENCY: 10

ENABLED ACCESS CONTROL.
```

Figure 3.15 (cont.) Sample "Dialogue" File Output

VI. Disk Fixer (FIXUP)

FIXUP is termed a diagnostic utility and is used to locate and correct errors in the disk file structure. FIXUP should be run periodically to delete temporary files and must be run after:

1) a System Panic (FATAL AOS ERROR)
2) a System Hang or Deadlock
3) a Fatal Hardware Error
4) an Abnormal System Shutdown
5) when submitting an STR.

FIXUP is usually booted from File 5 of the AOS System Magnetic Tape or a magnetic tape created by the SYSTAPE.CLI macro.

A couple words of Warning are in store:

1) If FIXUP finds a problem on the disk that it cannot resolve, it may delete the whole file, a portion of the file, or possibly a Directory. Critical files should be "backed up" on a regular basis.

2) If FIXUP is run, as will be discussed later, and any error messages result, it should be run again, etc, until no error messages are typed. This will guarantee that any problem areas on the disk have been totally corrected.

Table 3.3 at the end of this section will list the various FIXUP error messages and a short discussion of their meaning.

Figure 3.23 illustrates a standard FIXUP dialogue for DPEØ with Verbosity 1 and with output to the line printer (LPA). The questions asked by FIXUP are:

1) "Disk Unit Name?" - This is the name that AOS refers to the disk. Table 3.1 contains the various disk names vs hardware types.

2) "Device Code" - If this is the same device code as shown in Table 3.1 for the "Disk Unit Name", a "CR" can be typed. Otherwise, enter the device code.

```
FROM MTØ:5↵
-----


AOS DISK FIXER, REV Ø1.ØØ

SPECIFY EACH DISK UNIT IN THE LOGICAL DISK

DISK UNIT NAME? DPEØ↵
DEVICE CODE?↵

VERBOSITY (Ø - 2)? 1↵
ERROR LOG FILE? LPA↵
SHOULD I REPORT CLOSING FILES? Y↵ (usally "N")
MAY I FIX IT? Y↵

DONE!
```

Refer to Table 3.1

Figure 3.23
Sample responses for the FIXUP Utility on DPEØ with Verbosity 1 Output to LPA

```
ERROR IN DIRECTORY : -- FILE IS OPEN (FILE PER)
ERROR IN DIRECTORY : -- FILE IS OPEN (FILE PROC)
ERROR IN DIRECTORY : -- FILE IS OPEN (FILE SWAP.SWAP)
ERROR IN DIRECTORY : -- FILE IS OPEN (FILE GHOST.CL)
ERROR IN DIRECTORY :PER -- TRANSIENT FILE (FILE CONSOLE)
ERROR IN DIRECTORY :PER -- TRANSIENT FILE (FILE DPEØ)
ERROR IN DIRECTORY :PER -- TRANSIENT FILE (FILE DPE1)
ERROR IN DIRECTORY :PER -- TRANSIENT FILE (FILE DPE2)
ERROR IN DIRECTORY :PER -- TRANSIENT FILE (FILE DPE3)
ERROR IN DIRECTORY :PER -- TRANSIENT FILE (FILE DPD1Ø)
ERROR IN DIRECTORY :PER -- TRANSIENT FILE (FILE DPD11)
ERROR IN DIRECTORY :PER -- TRANSIENT FILE (FILE DPD12)
ERROR IN DIRECTORY :PER -- TRANSIENT FILE (FILE DPD13)
ERROR IN DIRECTORY :PER -- TRANSIENT FILE (FILE DPD14)
ERROR IN DIRECTORY :PER -- TRANSIENT FILE (FILE DPD15)
ERROR IN DIRECTORY :PER -- TRANSIENT FILE (FILE DPD16)
ERROR IN DIRECTORY :PER -- TRANSIENT FILE (FILE DPD17)
ERROR IN DIRECTORY :PER -- TRANSIENT FILE (FILE DPFØ)
ERROR IN DIRECTORY :PER -- TRANSIENT FILE (FILE DPF1)
ERROR IN DIRECTORY :PER -- TRANSIENT FILE (FILE DPF2)
ERROR IN DIRECTORY :PER -- TRANSIENT FILE (FILE DPF3)
ERROR IN DIRECTORY :PER -- TRANSIENT FILE (FILE MTAØ)
ERROR IN DIRECTORY :PER -- TRANSIENT FILE (FILE MTA1)
ERROR IN DIRECTORY :PER -- TRANSIENT FILE (FILE MTA2)
ERROR IN DIRECTORY :PER -- TRANSIENT FILE (FILE MTA3)
ERROR IN DIRECTORY :PER -- TRANSIENT FILE (FILE MTA4)
```

Figure 3.24
Sample of Verbosity 1 Output from LPA

Note that the above two questions will be asked as many times as there are physical disks in the LD on which FIXUP is being run.

3) "Verbosity (Ø-2)" - This code indicates the amount of information that you wish to know about errors that are found. The codes mean:

   Ø - No report
   1 - Brief report - Refer to Figure 3.24
   2 - Detailed report - Refer to Figure 3.26

   This should always be run with Verbosity = 2.

4) "Error Log File" - This is the AOS mnemonic for the device that you would like any error messages to appear on. The possibilities are:

   a) CONØ (if hard copy)
   b) LPA, LPA1
   c) LPB, LPB1

5) "Should I Report Closing Files?" - My answer to this question is "N" because what I am interested in is actual file structure errors without having to sift through all the file closing messages. Figures 3.23 and 3.25 use "Y" only for illustrative purposes.

6) "May I Fix It" - If the answer is "Y", FIXUP will attempt to resolve the disk even if it causes a portion of a file or a total file to be deleted. If the system has crashed, a bit is set in the Disk Information Block (DIB) which will not allow the pack to be accessed until FIXUP corrects the disk and resets the bit. Presently my opinion is to always say "Y" and to back-up the system frequently enough so a file can be recovered if it is deleted.

When FIXUP finishes it will type "DONE!". As mentioned before, if there are any error messages I would run FIXUP again until none appear during a run.

When "Y" is answered to the question "May I Fix It?", FIXUP

```
FROM MT0:5↵
-----


AOS DISK FIXER, REV 0100

SPECIFY EACH DISK UNIT IN THE LOGICAL DISK

DISK UNIT NAME? DPD10↵   } Refer to Table 3.1
DEVICE CODE?↵            }

VERBOSITY (0 - 2)? 2↵
ERROR LOG FILE? LPA↵
SHOULD I REPORT CLOSING FILE? Y↵  (Usually "N")
MAY I FIX IT? Y↵

DONE!
```

Figure 3.25
Sample Responses for the FIXUP Utility on DPD1Ø
with Verbosity 2 Output to LPA

```
ERROR IN DIRECTORY : -- FILE IS OPEN (FILE PER)
   FILE ADDRESS = 7, INDEX LEVEL = 1, DEVICE DPD10
   LOGICAL ADDRESS = 6, PHYSICAL ADDRESS = 714

ERROR IN DIRECTORY : -- FILE IS OPEN (FILE PROC)
   FILE ADDRESS = 7, INDEX LEVEL = 1, DEVICE DPD10
   LOGICAL ADDRESS = 6, PHYSICAL ADDRESS = 714

ERROR IN DIRECTORY : -- FILE IS OPEN (FILE SWAP.SWAP)
   FILE ADDRESS = 7, INDEX LEVEL = 1, DEVICE DPD10
   LOGICAL ADDRESS = 6, PHYSICAL ADDRESS = 714

ERROR IN DIRECTORY : -- FILE IS OPEN (FILE GHOST.OL)
   FILE ADDRESS = 7, INDEX LEVEL = 1, DEVICE DPD10
   LOGICAL ADDRESS = 6, PHYSICAL ADDRESS = 714

ERROR IN DIRECTORY :PER -- TRANSIENT FILE (FILE CONSOLE)
   FILE ADDRESS = 7, INDEX LEVEL = 1, DEVICE DPD10
   LOGICAL ADDRESS = 12, PHYSICAL ADDRESS = 720

ERROR IN DIRECTORY :PER -- TRANSIENT FILE (FILE DPD0)
   FILE ADDRESS = 7, INDEX LEVEL = 1, DEVICE DPD10
   LOGICAL ADDRESS = 12, PHYSICAL ADDRESS = 720

ERROR IN DIRECTORY :PER -- TRANSIENT FILE (FILE DPD1)
   FILE ADDRESS = 7, INDEX LEVEL = 1, DEVICE DPD10
   LOGICAL ADDRESS = 12, PHYSICAL ADDRESS = 720
```

Figure 3.26

Sample of Verbosity 2 Output from LPA

will perform certain operations. These are:

1) It will build a new disk bit map

2) Multiple allocation of disk blocks

   a) deletes all but the lst use of the blocks
   b) if part of a chain, the whole chain is deleted

3) Closes all files

4) Deletes all entries of a directory except data files, links, and disk directories

5) Deletes all:

   a) IPC files
   b) Unit files
   c) Generic files
   d) Mag Tape files
   e) Mag Tape Volumes
   f) All references to "grafted" LDs
   g) ?xx.xxx.TMP files

6) Verifies directory data blocks and index elements (will delete if necessary)

7) Recomputes:

   a) Count of inferior directories
   b) Current size for each Control Point Directory

The error messages that FIXUP can find are described in Table 3.3. FIXUP will attempt to read a block twice before giving an error.

Figure 6.14.1 illustrates the disk structure for a two physical unit LDU. The logical disk block addresses on DPD1Ø begin with block 7Ø6 physical which is block Ø logical. Note in Figure 3.26, that the difference between the "Physical Address" and the "Logical Address" is 7Ø6.

Table 3.3

FIXUP Console Messages

The following error messages can result from running the FIXUP diagnostic utility. The majority appear only if you typed 1) or 2) in response to VERBOSITY? during the FIXUP dialog. If you typed Y in response to MAY I FIX IT? FIXUP displays a message and tries to correct the error. The remainder are abort messages. These can appear at any time, regardless of your responses.

NOTE: FIXUP error messages are intended to provide diagnostic information to Data General personnel. They are not intended for user correction of disk errors.

Message	Description
CAN'T DELETE ROOT DIRECTORY	*This is an abort message.* Reload FIXUP and run it from the beginning. If trouble persists, call your Data General representative.
DIRECTORY BLOCK TOO HIGH TO BE ADDRESSED	Directory format error. Directory information may be deleted if you typed Y in response to the MAY I FIX IT? query in the FIXUP dialog.
DIRECTORY WITH ELEMENT SIZE NOT = TO 1	Directory format error. Directory information may be deleted if you typed Y in response to the MAY I FIX IT? query in the FIXUP dialog.
DIRECTORIES NESTED TOO DEEPLY	*This is an abort message.* No FIXUP action is possible. Restructure to reduce directory nesting levels.
DISK ERROR READING DIRECTORY BLOCK	Directory format error. Directory information may be deleted if you typed Y in response to the MAY I FIX IT? query in the FIXUP dialog.
FATAL DISK ERROR	*This is an abort message.* No FIXUP action is possible. Reload FIXUP and run it again from the beginning. If trouble persists, call your Data General representative.
FILE IS OPEN	This informational message appears only if you type Y in response to the SHOULD I REPORT CLOSING FILES? query during FIXUP dialog. This message appears for every file that was open at system shutdown. FIXUP closes the file only if you typed Y in response to the MAY I FIX IT? query in the FIXUP dialog.
INCORRECT FORMAT FOR DEVICE INFORMATION BLOCK	*This is an abort message.* Directory format error. No FIXUP action is possible. Call your Data General representative.
INCORRECT FORMAT FOR DIRECTORY BLOCK	Directory format error. Directory information may be deleted if you typed Y in response to the MAY I FIX IT? query in the FIXUP dialog.
INCORRECT LENGTH FOR FIB	Directory format error. Directory information may be deleted if you typed Y in response to the MAY I FIX IT? query in the FIXUP dialog.
INSUFFICIENT MEMORY FOR BIT MAPS	FIXUP does not have enough memory to build new bit maps. Acquire more.
INVALID FAC POINTER IN FIB	Directory format error. Directory information may be deleted if you typed Y in response to the MAY I FIX IT? query in the FIXUP dialog.

Table 3.3

FIXUP Console Messages (Continued)

Message	Description
INVALID FIB POINTER FOR FAC	Directory format error. Directory information may be deleted if you typed Y in response to the MAY I FIX IT? query in the FIXUP dialog.
INVALID FIB POINTER FOR FLB	Directory format error. Directory information may be deleted if you typed Y in response to the MAY I FIX IT? query in the FIXUP dialog.
INVALID FIB POINTER FOR FNB	Directory format error. Directory information may be deleted if you typed Y in response to the MAY I FIX IT? query in the FIXUP dialog.
INVALID FIB POINTER FOR LINK	Directory format error. Directory information may be deleted if you typed Y in response to the MAY I FIX IT? query in the FIXUP dialog.
INVALID INDEX DEPTH SPECIFIED IN FIB	Directory format error. Directory information may be deleted if you typed Y in response to the MAY I FIX IT? query in the FIXUP dialog.
INVALID FIRST LOGICAL ADDRESS	Directory format error. Directory information may be altered if you typed Y in response to the MAY I FIX IT? query in the FIXUP dialog.
INVALID FNB CHAIN FOR FIB	Directory format error. Directory information may be deleted if you typed Y in response to the MAY I FIX IT? query in the FIXUP dialog.
INVALID POINTER IN INDEX BLOCK	File error. File information may be deleted if you typed Y in response to the MAY I FIX IT? query in the FIXUP dialog.
MULTIPLY-ALLOCATED FILE ELEMENT	File error. File information may be deleted if you typed Y in response to the MAY I FIX IT? query in the FIXUP dialog.
MULTIPLY-ALLOCATED INDEX BLOCK	File error. File information may be deleted if you typed Y in response to the MAY I FIX IT? query in the FIXUP dialog.
NON-CONTIGUOUS FILE WITH DOUBLE PRECISION ELEMENT SIZE	Directory format error. Directory information may be deleted if you typed Y in response to the MAY I FIX IT? query in the FIXUP dialog.
TRANSIENT FILE	This informational message appears only if you typed Y in response to the SHOULD I REPORT CLOSING FILES? query in the FIXUP dialog. FIXUP closes and deletes the file only if you typed Y in response to the MAY I FIX IT? query in the FIXUP dialog.

IV Break File Analysis

The analysis that is done at this point is very similar to two other sections contained in this document. These two sections are:

a) Section 5, "System Traps"

b) Section 4, III, (1), "User Process Analysis"

Section 5.2, "Hardware Trap", discusses among other things the format of the "Break File" name, the use of DEDIT to access the "Break File" and also the use of the DISPLAY Utility to type or print out the "Break File" in an "FPRINT" type format.

Now the "Break File" can be analyzed to attempt to isolate the problem. Various possibilities would be a "JMP.", a "JMPØ", or a "tight code loop". We can use the writeup in Section 4, III, (1), "User Process Analysis", to guide us in the proper direction. The first few pages in this section discuss how to gain access to the User Process section of the Core Dump. Since our "Break File" is a mirror image of the Process space, these adjustments are unnecessary. We can follow the procedure beginning with:

"A) Examine the Various Process Pointers"

thru:

"D) Examine the Stack for the Current TCB"

to determine and hopefully find the reason for the Process's problem. I would suggest reading/rereading both section noted in a) and b) above to give yourself a better "feel" when doing the analysis.

DPAC TRANSCRIPTION NOTE PG 4-72

SECTION 4
SYSTEM HANGS

I. Introduction

The broad definition of a System Hang, Deadlock, or Lockout could be "the non-response of the system or a portion of the system to a given command". For practical purposes, these occurances can be subdivided into two categories:

a) The problem only applies to a single console. Specific causes would be:

   1) The console is in Page Mode, the screen has been filled up, and the system is waiting for a "^Q" to continue.

   2) A "^S" was accidentally typed on the console and the system is presently waiting on a "^Q" to continue.

   3) The process executing on the console has done a "JMPØ", "JMP.", or is hung in a tight loop.

   4) The EXEC is having a problem with the specific console. This could appear as not being able to "Log On" once the console is "Log Off".

b) The problem applies to all the consoles including the Master Console or just to the EXEC consoles. Specific causes would be:

   1) The AOS Operating System is constantly in the scheduler (SMON) loop waiting for an event to happen which never does. This could be caused by a resource deadlock or the non-responce of the PMGR with processes tying up core waiting on the response.

   WARNING
   AOS requires a Real Time Clock (RTC). If the RTC is missing or inoperable, the system will hang at "boot" time with a "b)1" case above prior to the initial Op CLI message. This occurs because the CLIBT does a ?DELAY for 1 sec between starting the PMGR and Op CLI. Because there is no RTC, the

delay never expires and the hang prior to the Op CLI being started.

2) The AOS Operating System has performed a "JMPØ" or a "JMP." due to either a software or hardware failure.

3) The EXEC Utility has "crashed" doing a "JMPØ", "JMP.", or has trapped out causing all the EXEC terminals to cease responding.

4) The code within the DCU5Ø has "bombed" or the DCU5Ø hardware has stopped, causing all consoles running off of the DCU5Ø not to respond.

The above are the more frequent cases that have been seen. I suspect that as time goes on, other cases, possibly on a "one-of-a-kind" basis, will be observed.

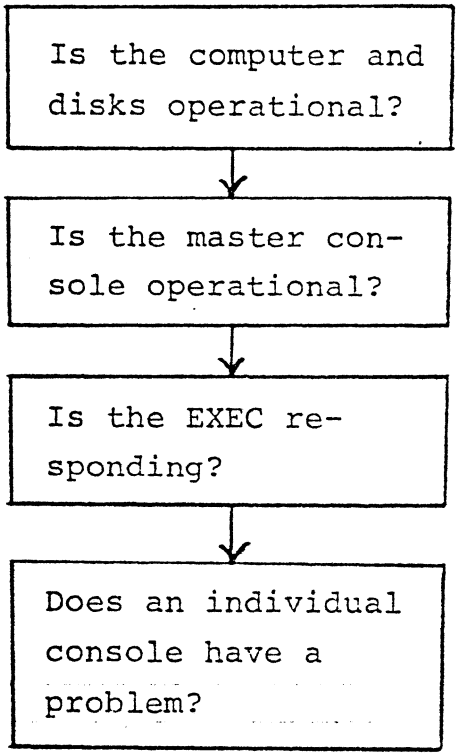
Figure 4.1 illustrates a block diagram of the troubleshooting procedure that will be outlined in this section. The block diagram attempts to isolate the problem to a section of the system so a more finer definition can be done.

Figure 4.2 illustrates the actual troubleshooting flow that can be used to isolate the problem to a major section and then the minimal procedure to correct it. In some cases, the minimal procedure is to stop the system and take a core dump. This is only done in a "worst case" situation when a "Break file" is either not sufficient or cannot be taken. Figure 4.2 should be followed until either a "core dump" or "break file" is taken or until the problem is cured or resolved using the procedures in the flow.

If a "core dump" or "break file" has to be taken, additional sections are provided which discuss areas to examine in order to possibly further narrow down the failure. The sections are:

Figure 4.1

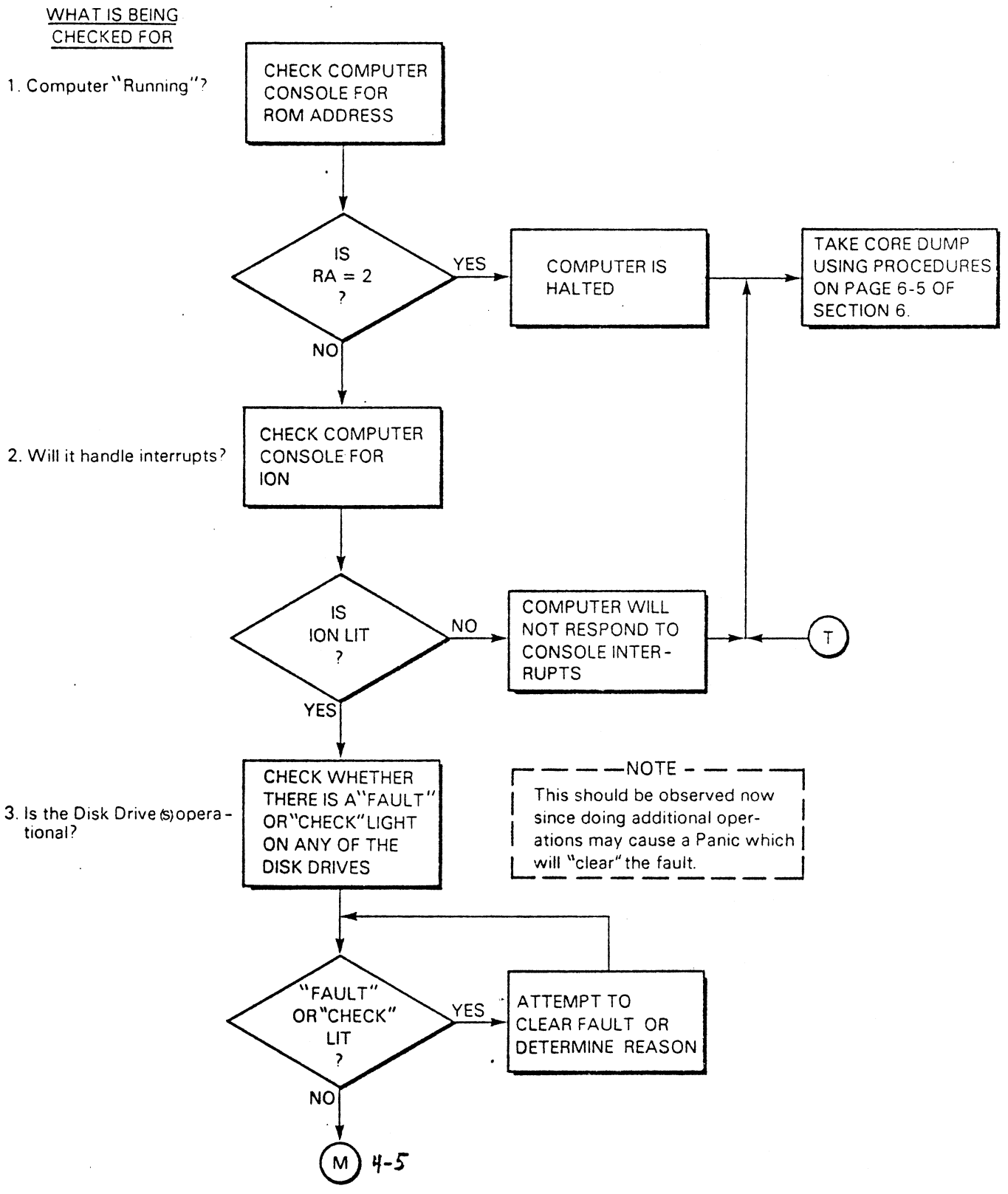
Troubleshooting Functional Block Diagram



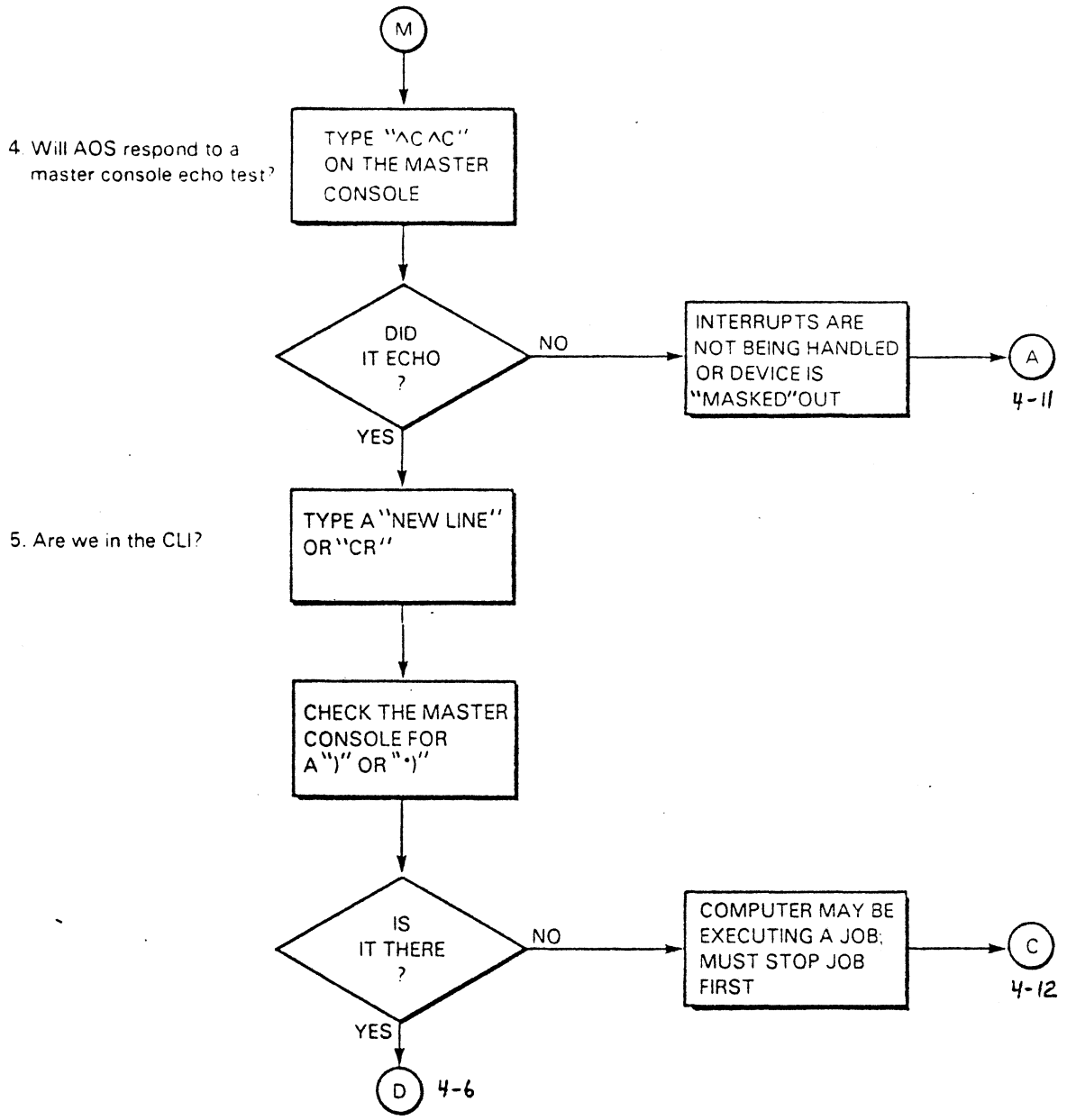
HANG

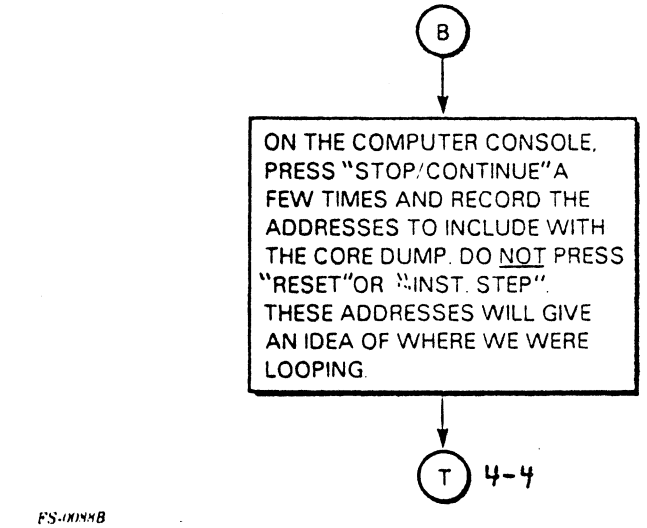
DPACØ: ACØ
DPAC1: AC1
DPAC2: AC2
DPAC3: AC3

Figure 4.2 Trouble shooting flow

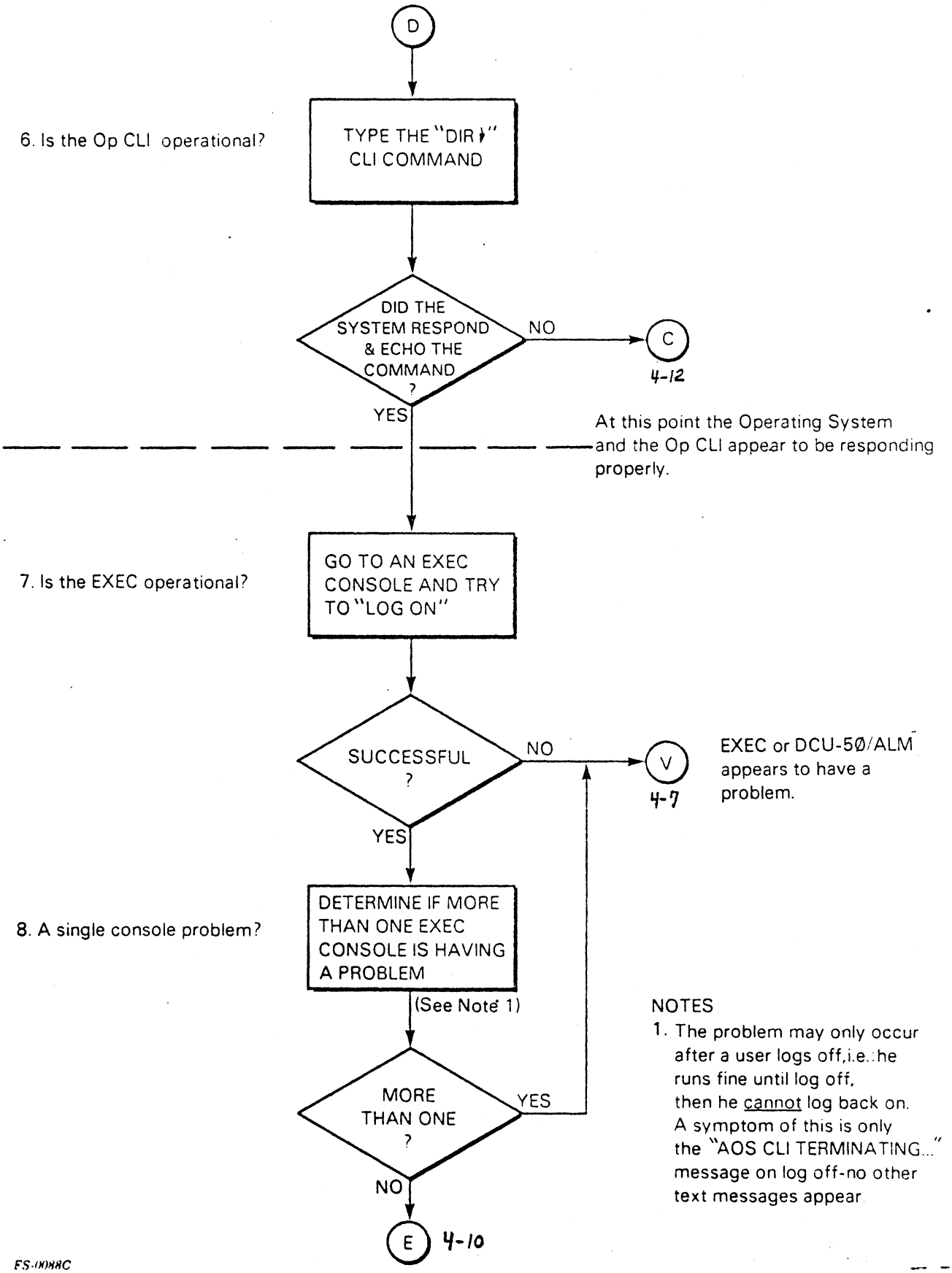


FS-00188A

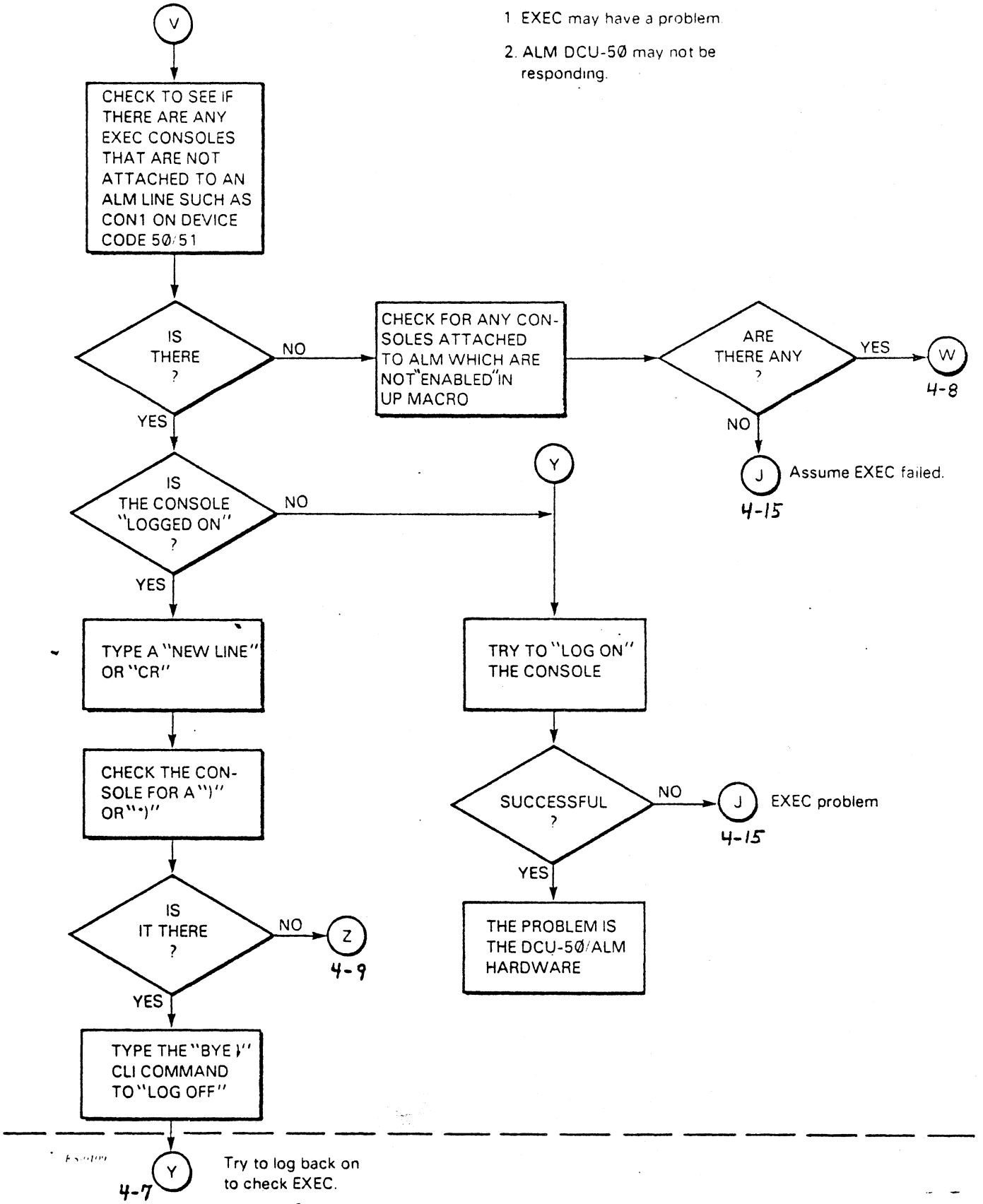


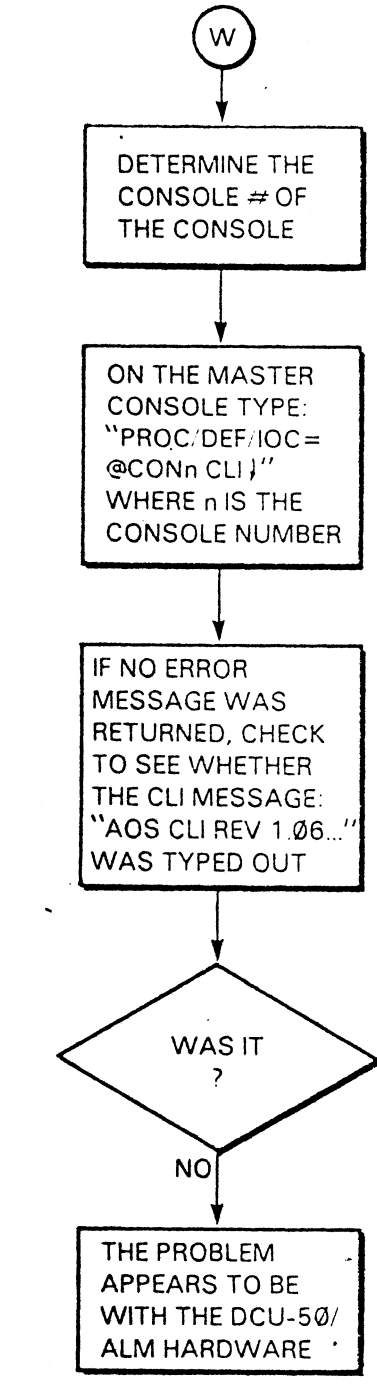


FS-00088



FS-00088C





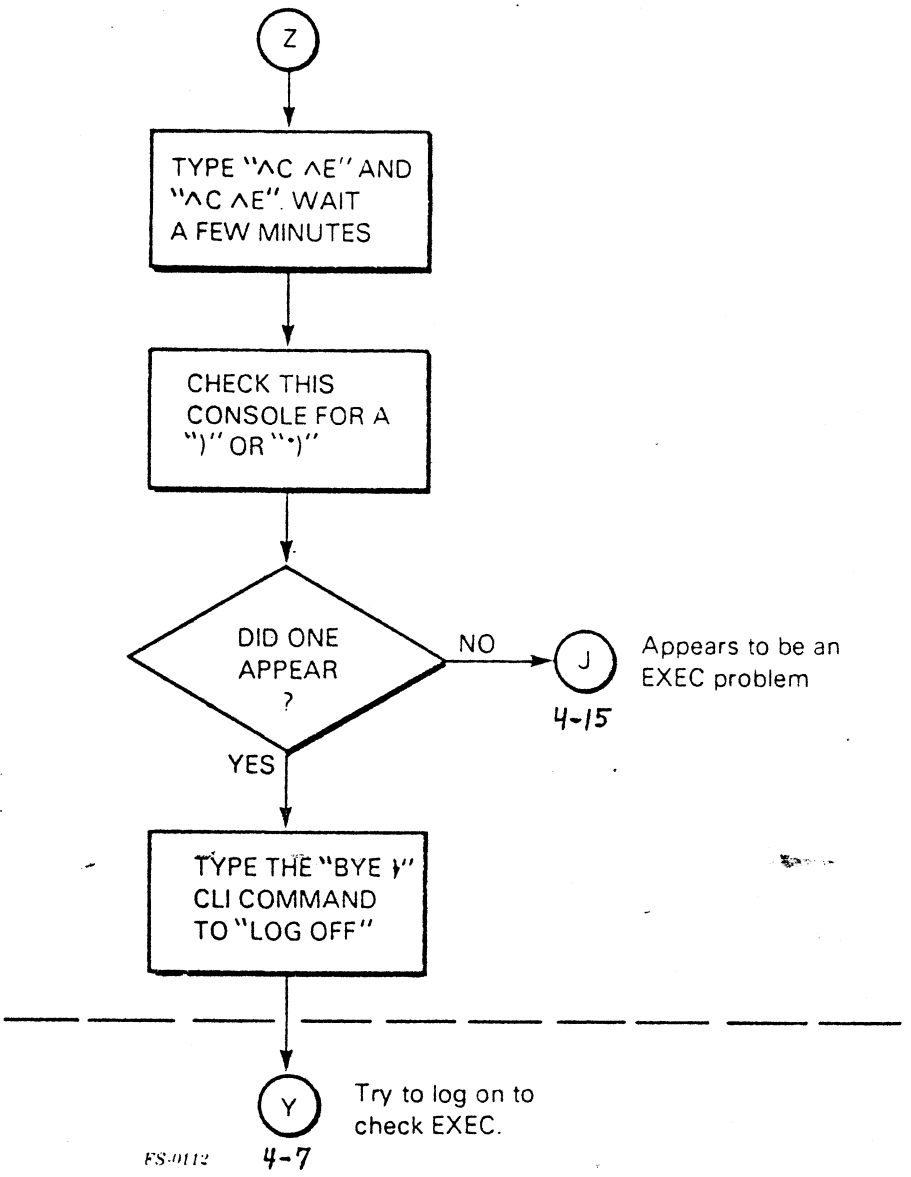
Try to initiate the CLI on a non-EXEC console on an ALM line.

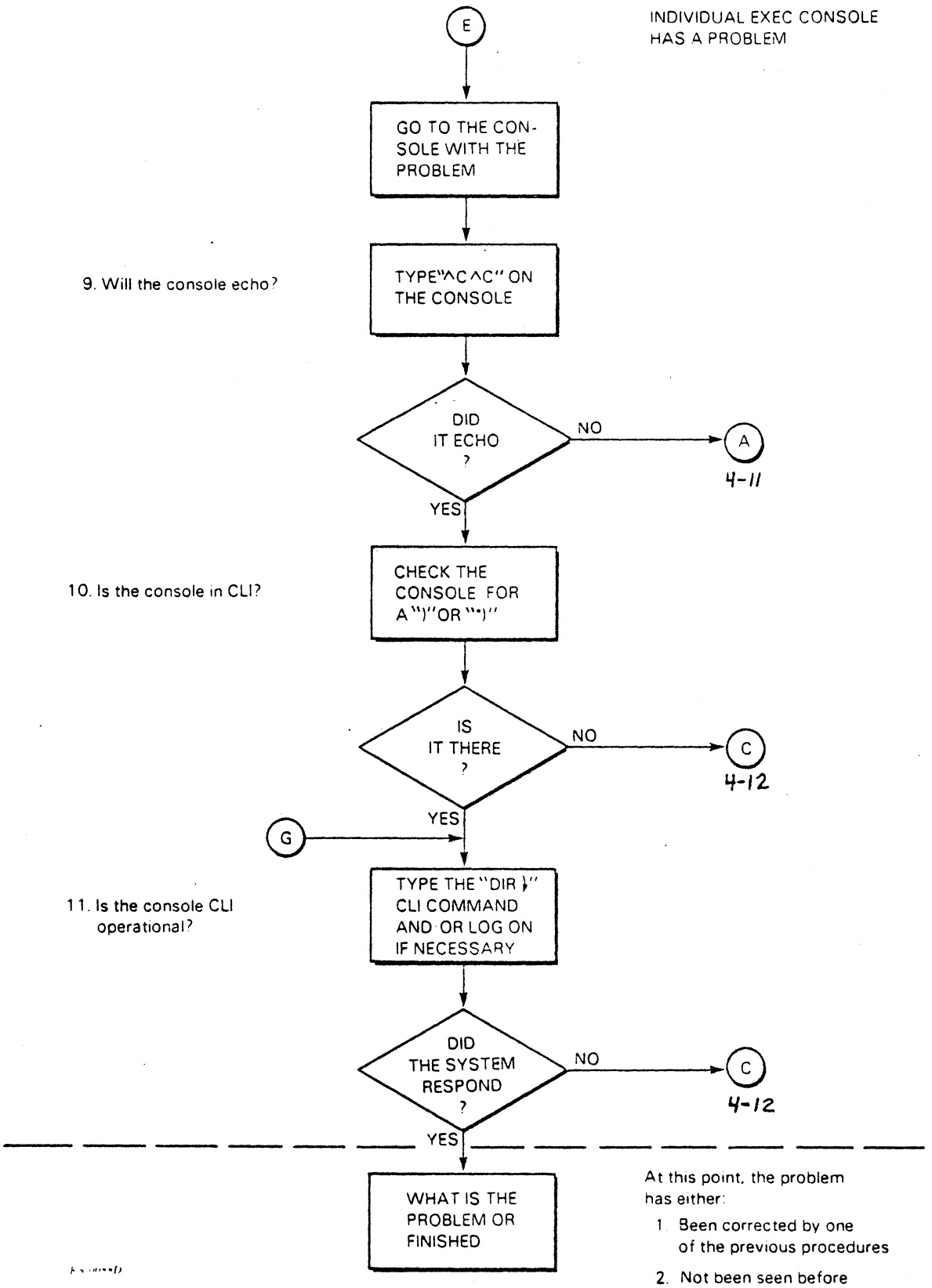
EXEC has died. The ALM line appears to be working o.k.

4-15

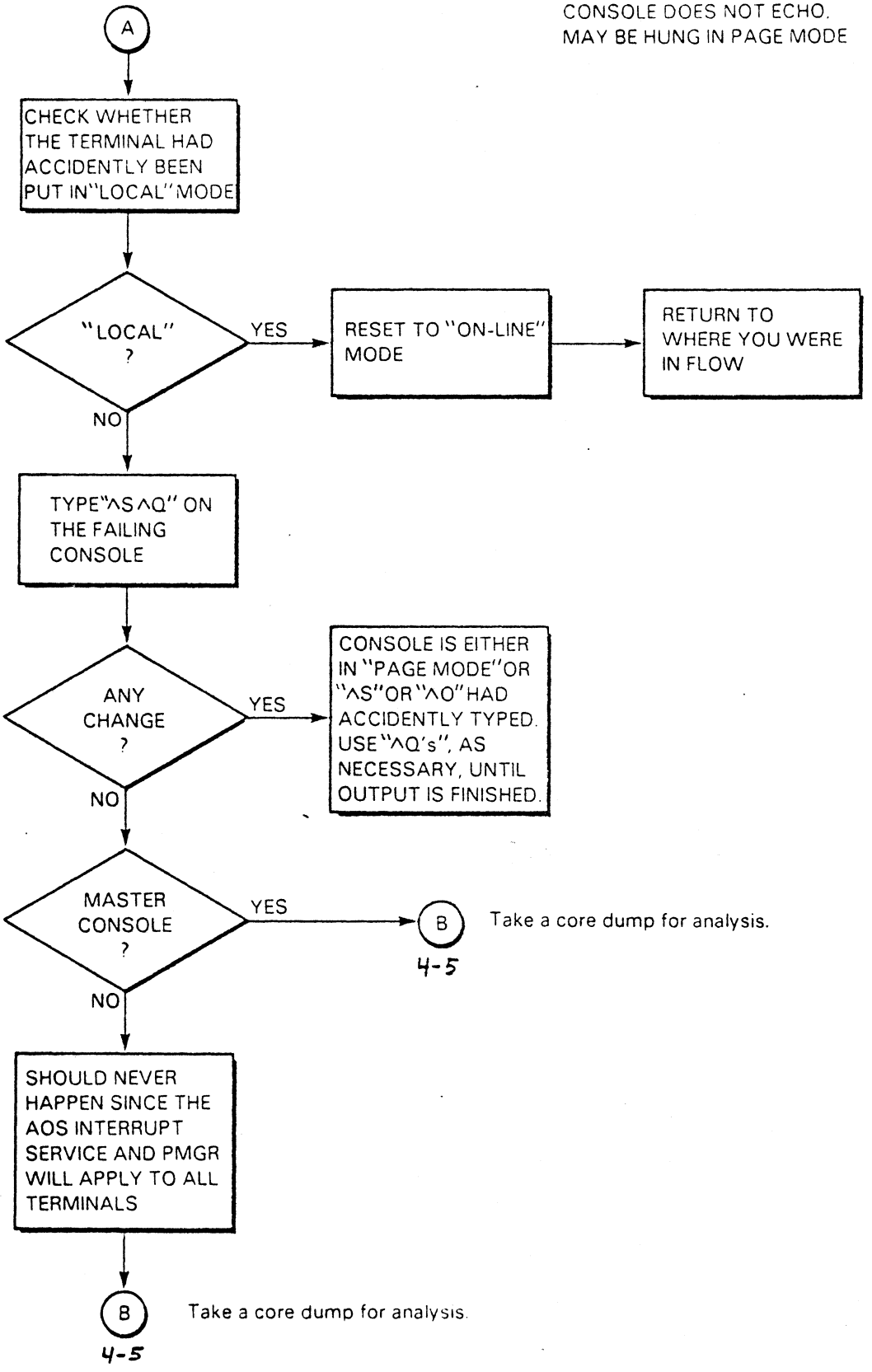
NOTE: make sure the ALM chassis power switch (Local/Remote) was not turned off - middle switch position.

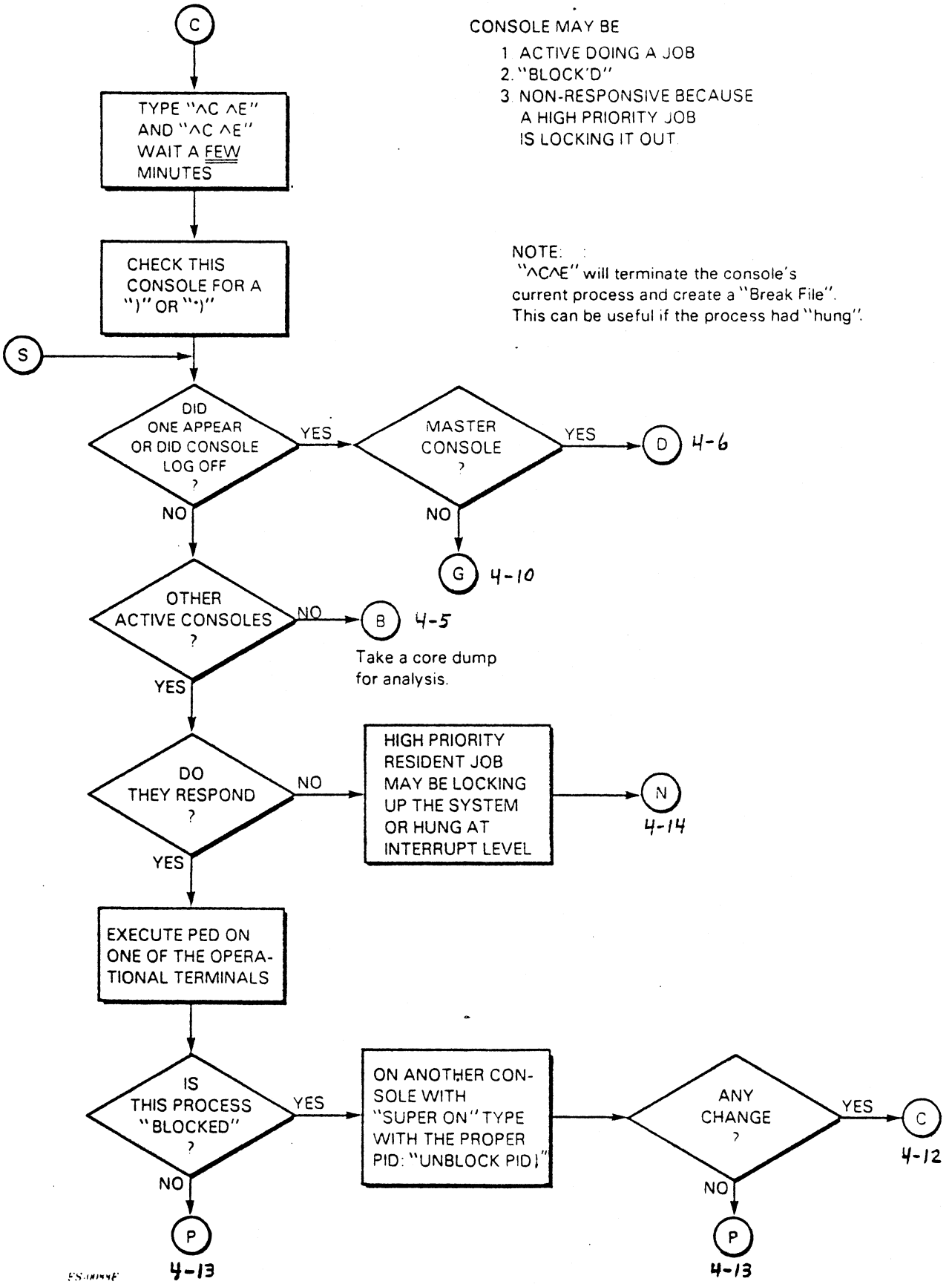
FS-0110





CONSOLE DOES NOT ECHO.
MAY BE HUNG IN PAGE MODE

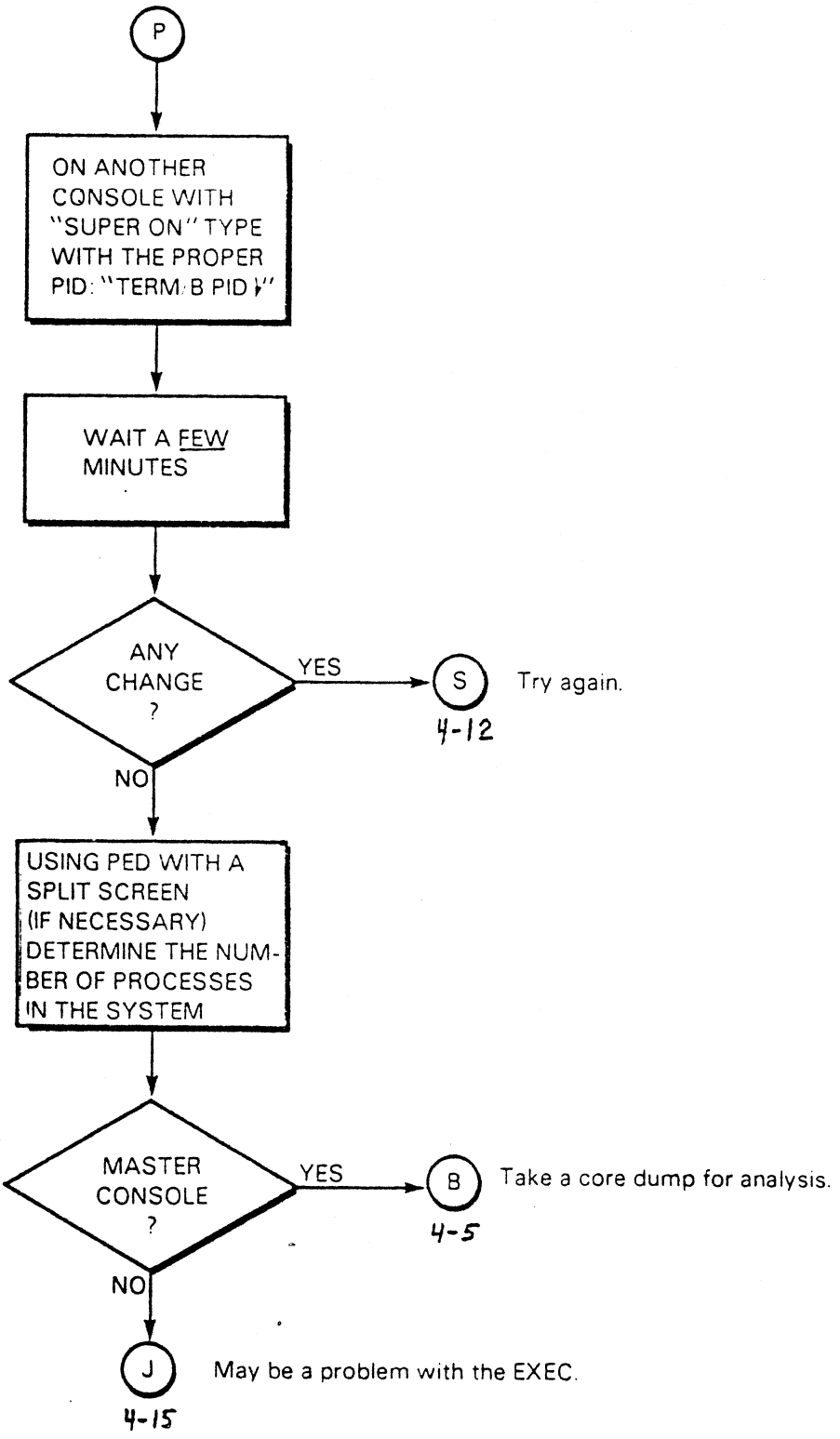




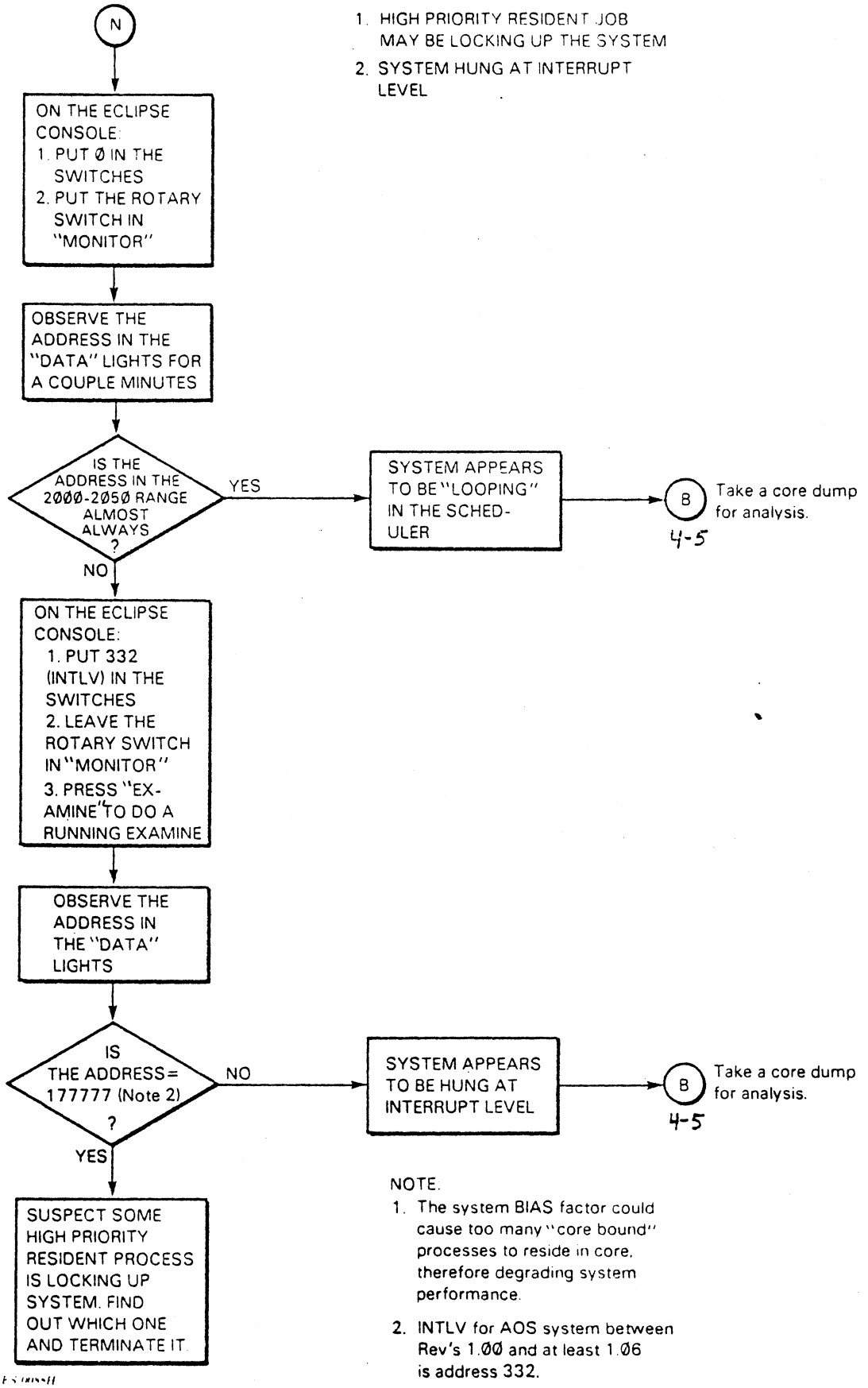
CONSOLE MAY BE

1. ACTIVE DOING A JOB
2. "BLOCK'D"
3. NON-RESPONSIVE BECAUSE A HIGH PRIORITY JOB IS LOCKING IT OUT

NOTE:
"^C^E" will terminate the console's current process and create a "Break File". This can be useful if the process had "hung".



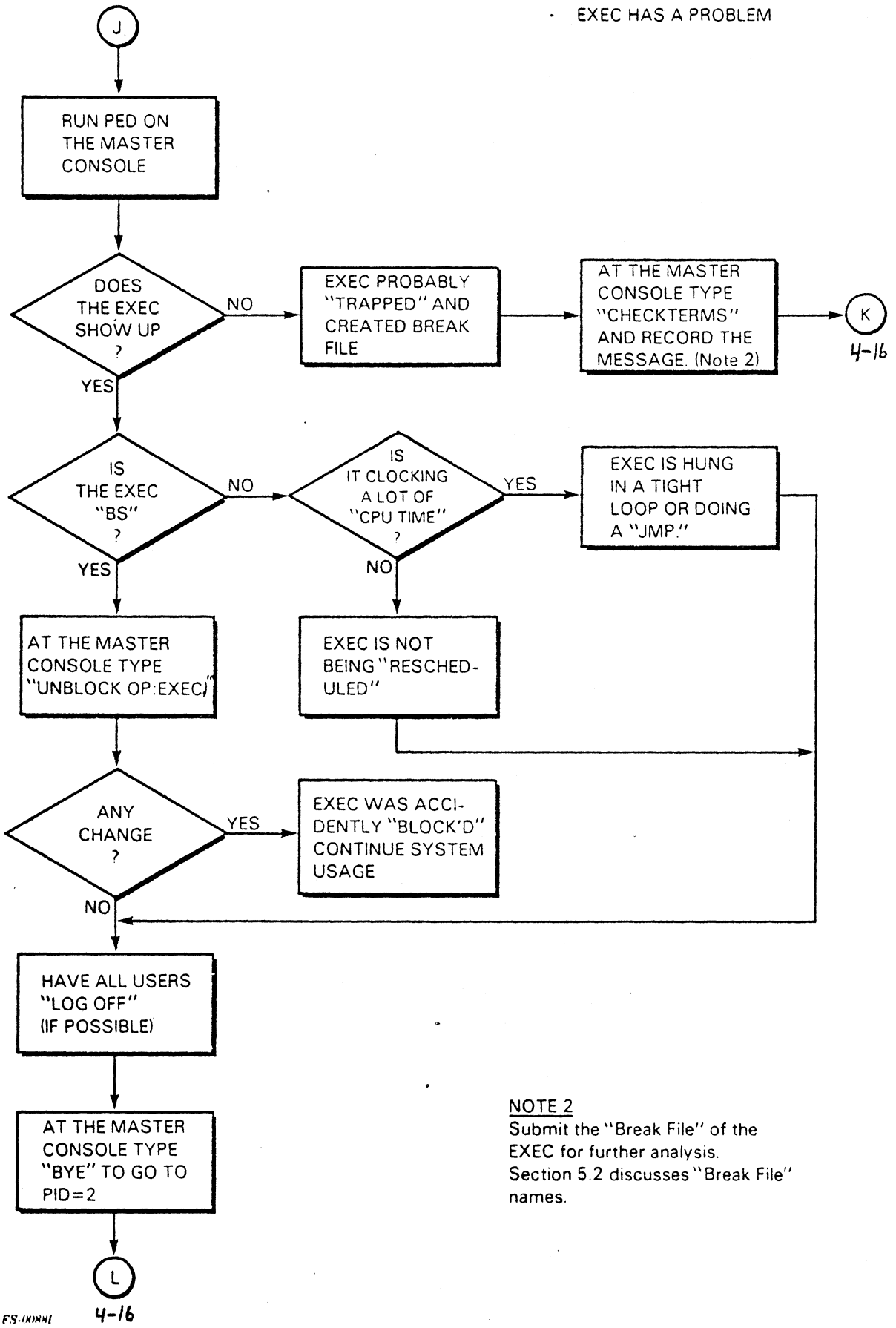
FS-00888G



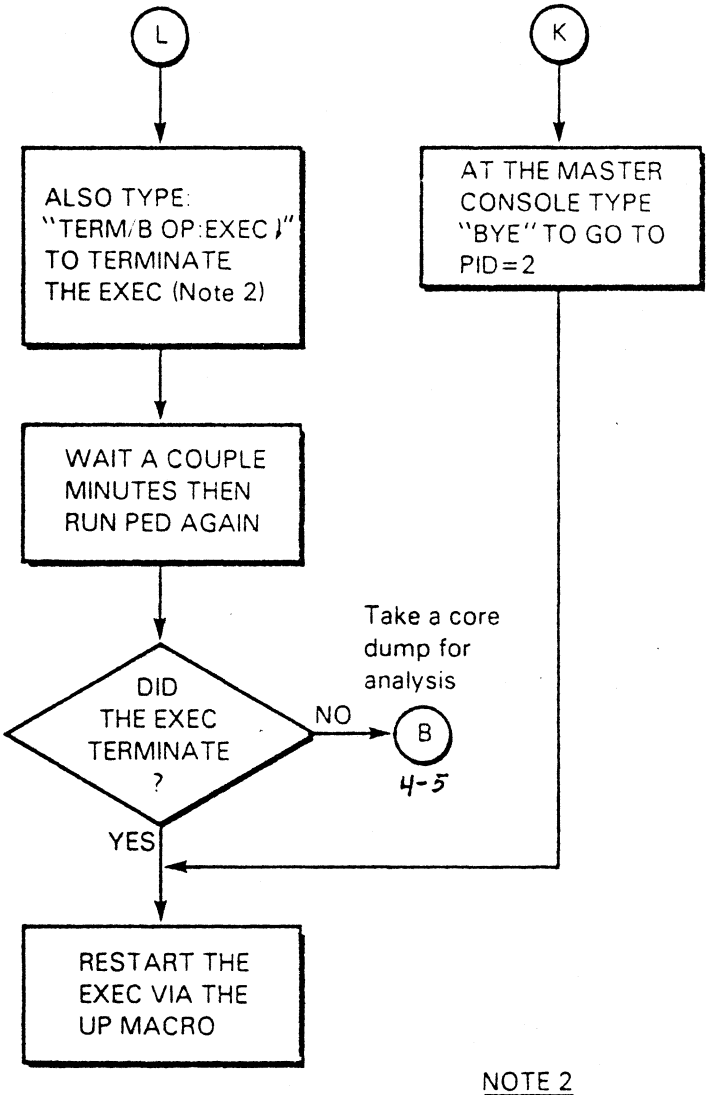
1. HIGH PRIORITY RESIDENT JOB MAY BE LOCKING UP THE SYSTEM
2. SYSTEM HUNG AT INTERRUPT LEVEL

NOTE.

1. The system BIAS factor could cause too many "core bound" processes to reside in core, therefore degrading system performance.
2. INTLV for AOS system between Rev's 1.ØØ and at least 1.Ø6 is address 332.



NOTE 2
Submit the "Break File" of the EXEC for further analysis. Section 5.2 discusses "Break File" names.



NOTE 2
Submit the "Break File" of the EXEC for further analysis. Section 5.2 discusses "Break File" names.

FS-00883

II. AOS Core Dump Analyzer

Having to scan each queue and then check the Status word or Flag words of each Process Table or Control Block can get to be very time consuming, to say the least. Systemo Programming has developed an "AOS Dump Analyzer" which will output certain system status information, scan all the queues and then output information about each Control Block or Process Table on the queue. Figure 4.11 shows the procedure for invoking the analyzer. The CLI generic list file, "LIST", must be set to a filename or line printer name. This can be @LPT or @LPT1 if the EXEC is active or @LPA or @LPA1 if there is no EXEC. If there is a DCH line printer on the system, "LIST" can be set to "DUMPxx.ADA" and then printed later. The analyzer is started via:

)X ADA <core-dump-file-name><symbol-table-file-name>)

As it analyzes each data base, a message as shown in Figure 4.11 will be typed out on the console. When it is finished, the generic "LIST" file should be changed back to what it was prior to the run.

Figures 4.3 thru 4.5 illustrate a small sample of the three sections that will be printed out during an ADA run. These are:

a) Initial Dump Information (Figure 4.3)
b) Core Manager and Control Block Information (Figure 4.4)
c) Process Table information (Figure 4.5)

Each of these sections will be discussed in greater detail in the following paragraphs. If the ADA program is not available, each of the queues that are needed must be scanned using "SYSDMP" and the status and flag words recorded.

There have been a couple cases of ADA bombing with the error message:

```
*ERROR*
END OF FILE
ERROR:FROM PROGRAM
```

This has been caused by ADA looking for values that do not exist causing the "End of File" error. In the majority of cases that I

have seen it, the problem was in a Control Block and was not a problem with the Core Dump. The problem was with ADA because it was looking for values in cases when it should not have been. The way to get around the problem is to remove the C.B. from the ELQUE, ie: link around it. This must be done with DEDIT since SYSDMP does not allow locations to be changed. I will not illustrate a procedure since it can be tricky. If necessary call me and we can discuss the problem over the phone and verify that the problem is with ADA and not the core dump. We can then find a way around it. These problems have been reported and should be corrected in a future revision of ADA.

Figure 4.3 illustrates the initial dump information that is typed out. The system Rev and patch revision are given. The next values will indicate if the dump was taken due to a "Fatal AOS Error", ie: Panic, or whether the system was "hung" when the dump was started. The contents of the AC's at either the time of Panic or the time the dump was started will by typed. The "PC" value will only appear for "Panics" and will be non-zero only for a Panic 7 or 10, a five value Panic caused by a Trap. The currently or lastly executing Process Table or Control Block will be indicated (the contents of "CC"). The "System State" will be:

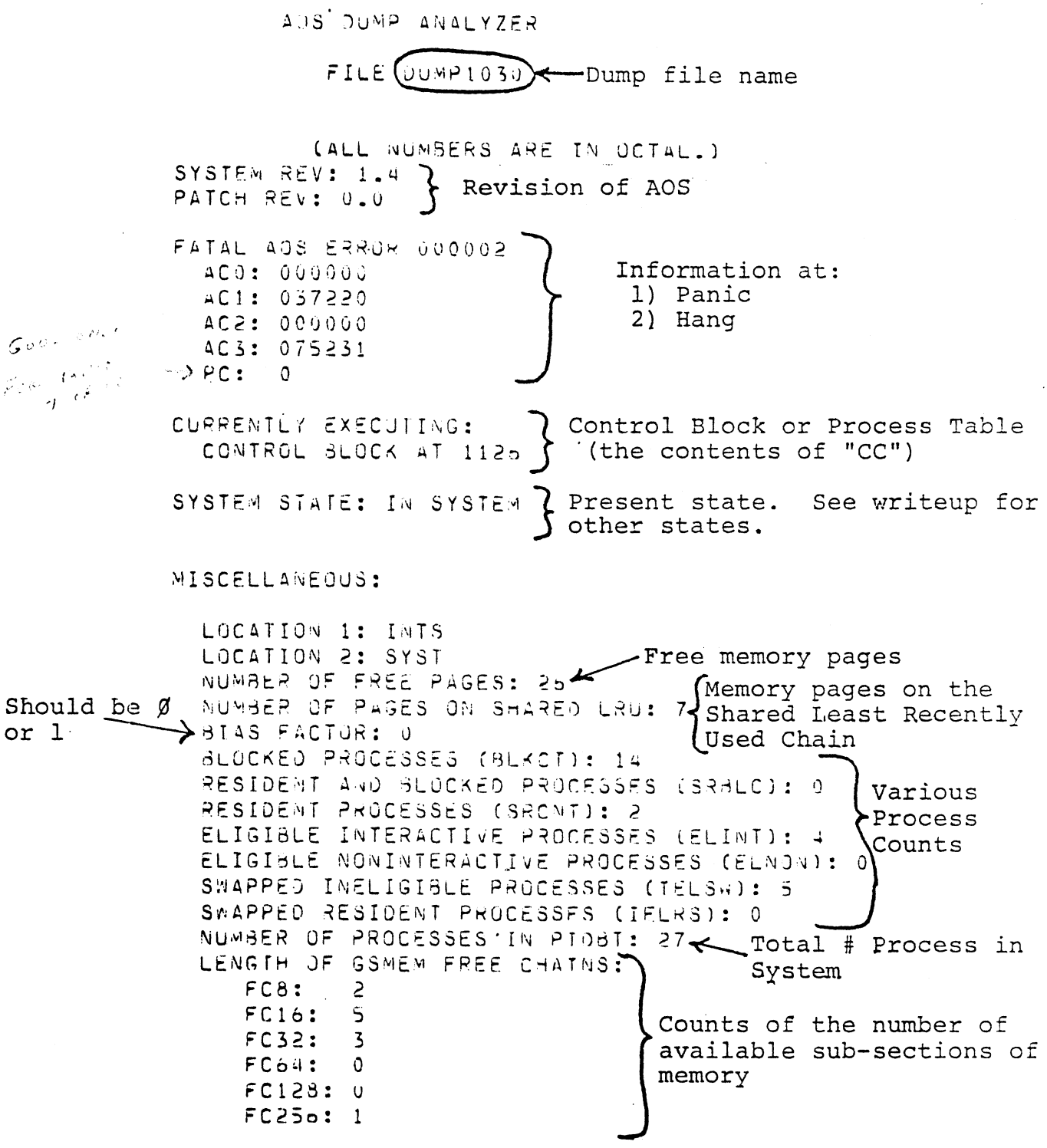
Figure 4.12 case

a) In User
   1) In Primary ①
   2) In Ghost ①
b) In Checksum Loop ②
c) In Interrupt World ⑤
d) In System ② ③
e) Scheduling a User ② ④

If the system was "In Interrupt World" when the core dump was taken, in addition to that message some supplementary values are given. These are:

a) INTLV - "-1" is base level
b) CMSK - This is the contents of physical location 5 and is the Current Interrupt Mask.

Figure 4.3 Initial ADA Dump Information



c) SS+11 - This is a value on the interrupt stack and is the "C and PC" at the time of the first interrupt. This can be a User or an Operating System address.

d) SYSIN - This will indicate whether we were in the Operating System or a User when the first interrupt occurred and will clarify the address space that "SS+11" applies to.

Next are listed the contents of Location 1 and Location 2. Location 2 changes from "SYST" to "UINTR+2Ø" when the dump is taken while we are at interrupt level. Following are some more constants which are self-explanatory. The following should be especially noted:

"BIAS FACTOR" should be Ø or 1
"NUMBER OF PROCESSES IN PIDBT" is the total number of processes in the system.

Note that all the numeric values in this section should be "positive". If I were to find a value such as "177776", I might become a little suspicious. Core is sub-divided into smaller multiples using an octal base, ie: 8 words, 16 words, 32 words. This way, a section of core 128 words long is not tied up because 8 words of segments are needed. The list from FC8 to FC256 is the number of segments of that word multiple available at the time the dump was taken.

Figure 4.4 illustrates the information printed out for the Core Manager and the Control Blocks. First the status word, PSTAT, is given and any set bits are explained. For the Core Manager, the "PC" is now given. This is the "AC3" value in the last stack frame which is due to a "JSR" and stored via a "SAVE" using the value in CSTK of the C.B.. If the Core Manager is idle, the PC will be Ø. Next, a message is printed as to whether any Process Tables are "enqueued" to the Core Manager. Figure 4.6 illustrates the "enqueing" linkage. Process Tables are "enqueued" to the Core Manager when the Processes needs to be swapped In or Out. The overlay, if any, contained in offset 12 (CCRSG) of the Core Man-

Figure 4.4 Core Manager and Control Block Information

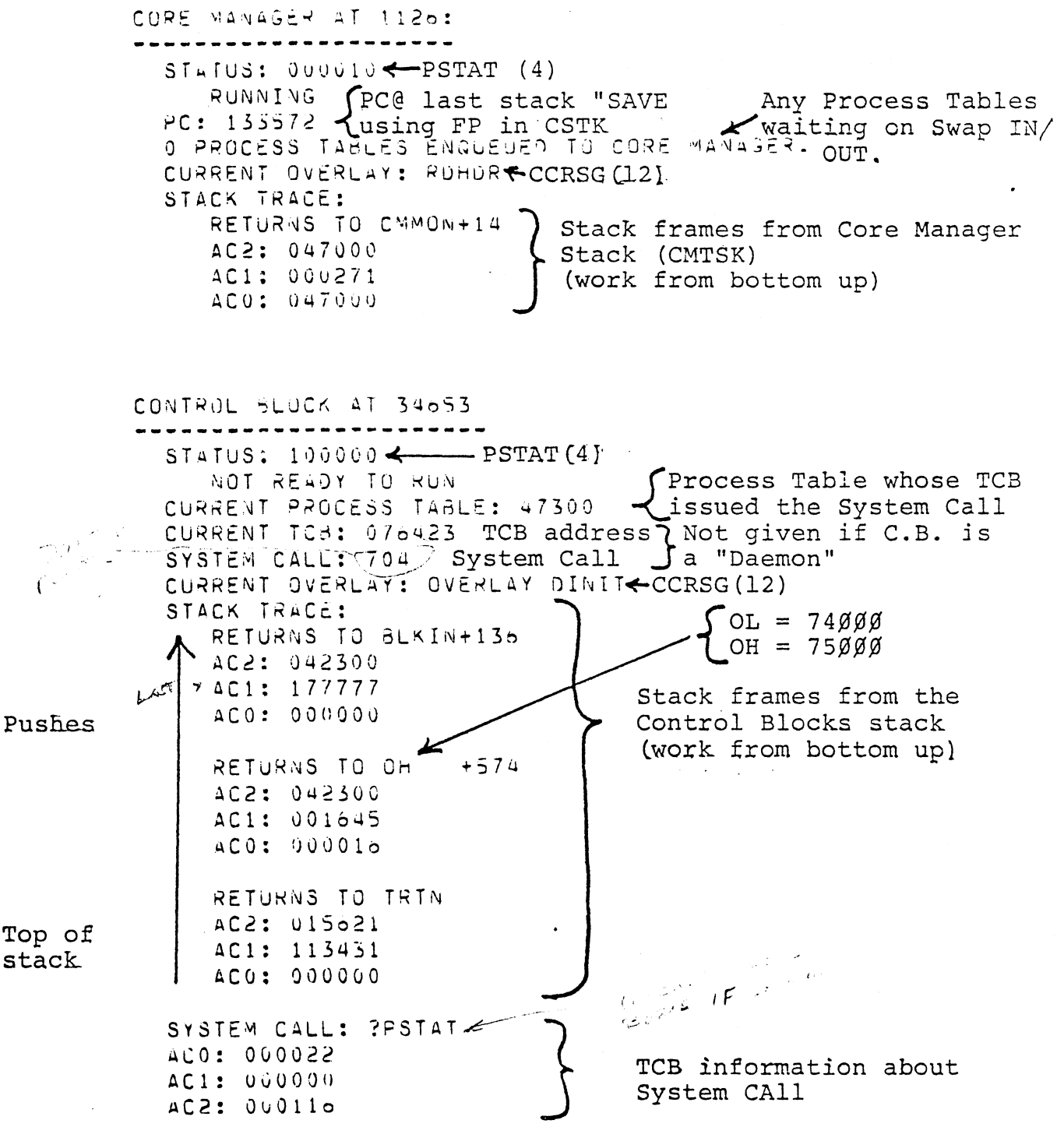


Figure 4.5 Process Table Information

```
**********
* PROCESS TABLES ON ELIGIBLE QUEUE
**********

      .
      .
      .
      .

PROCESS TABLE AT 57000. PID = 1. FATHER AT 1167
-----------------------------------------------
STATUS: 100000 <---------- PSTAT(4)
   NOT READY TO RUN
PFLAG: 004000 <---------- PFLAG(12)
   PROCESS IS ELIGIBLE (IN CORE)
PFLG2: 000012 <---------- PFLG2(13)
   SHARED AREA CHANGED
   SUPERUSER MODE
PFLG3: 004020 <---------- PFLG3(14)
   CCBS WERE REFERENCED IN LAST SLICE
   HAS CREATED AN IPC TYPE ENTRY
EXPONENT: 0 <---Time Slice Exponent, PSLEX(47)
PRIMARY SIZE: 11 {Total # of 1K memory pages in User Process
GHOST SIZE: 0    {- Unshared, PBLKS(22), plus shared, PSHSZ(23)

                 {Total # of 1K memory pages in Ghost - Un-
                 {shared, PGBLS(32), plus Shared, PGSHZ(33)
TCB AT 423
STATUS: 100000
   TASK PENDED
SYSTEM CALL: 177777
PRIORITY: 0

TCB AT 543
STATUS: 100000
   TASK PENDED
SYSTEM CALL: ?ISEND
PRIORITY: 1

TCB AT 473
STATUS: 100000
   TASK PENDED
SYSTEM CALL: ?IREC
PRIORITY: 1

      .
      .
      .

**********
* PROCESS TABLES ON BLOCKED QUEUE
**********
```

Task Control Blocks (TCBs) on the Active TCB chain (See Figure 4.17)

ager Control Block is listed. If the Core Manager is active, the contents of the CMSTK frame by frame are given. The "Returns to" location is the location after the "JSR" to the given subroutine.

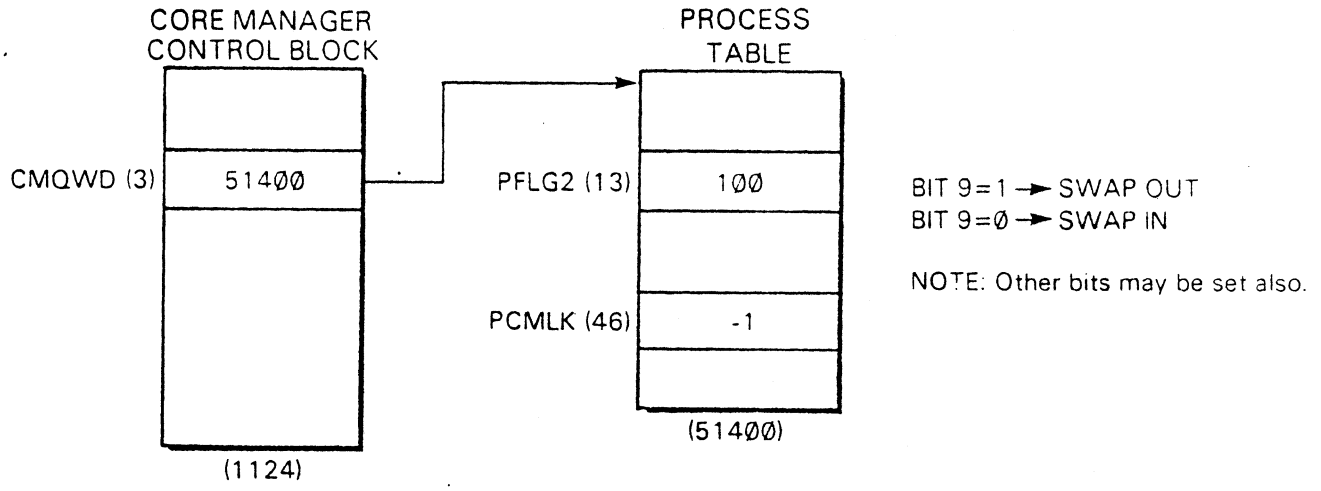
If there are any Control Blocks on the ELQUE, they are now listed. These are also illustrated in Figure 4.4. The status word, PSTAT, is given and any set bits are explained. The "Current Process Table" is the Process that the System Call is being done for. Note that some System Calls will change this location, such as storing the address of the RTPTB there. The "Current TCB" and "System Call" supply the respective information. Neither are given if the Control Block is a "Daemon" because Daemons are not activated directly via the System Call mechanism. The overlay, if any, contained in offset 12 (CCRSG) of the Control Block is listed. A stack trace for the Control Block's stack is given then the "System Call" with the accumulators at the time of the call.

Now the Process Tables attached to the various queues will be listed with information about that respective Process Table. The queues are listed in the following order:

a) Eligible Queue
b) Blocked Queue
c) Ineligible Resident Queue
d) Ineligible Swapped Queue

Also, if a Process is "In Core", ie: eligible, additional information will be listed after the given Process about each Task Control Block (TCB) on the active TCB chain (pointed to by USTAC).

Figure 4.5 illustrates a sample Process Table printout. Within each of the Process Tables' headers, the logical core address of the Process Table is given; it's PID (in octal) - note that PED will list the PID's in decimal; and the logical core address of the Father Process. The highest level Processes, the PMGR and the Op CLI, will have the RTPTB listed as the father since these are the initial Processes created on system initialization. The contents and an explanation of the various bits in the Process



FS-0089

Figure 4.6 Core Manager Swap Linkage

SMFLG BIT	SMFLG WORD	ROUTINE MNEMONIC	MEANING
0	100000	CHNM	Got Memory; Unpend Waiters
1	40000	CMBUF	Grow System Buffer Pool
2	20000	COVGR	Grow System Overlay Pool
3	10000	CBQSC	Scan "Block" Queue
4	4000	CTOUT	Look for Timeout (TQUE)
5	2000	CUERR	Unit Error

FS-0090

Figure 4.7 Core Manager Command Table

Status Word, PSTAT, and the Process Flag words are now given. Additional information about these words can be found in:

Status/Flag word	Table
PSTAT	4.1
PFLAG	4.2
PFLG2	4.3
PFLG3	4.4
PFLG4	4.5

Next the "exponent" is given for the Process. This is the Time Slice Exponent, PSLEX(47), and only applies to Swappable processes. It will vary from 1-6 with 1-5 used for "Interactive" processes and 6 indicating a "Non-Interactive" process. For Resident and Pre-emptible processes, the "exponent" will always be "Ø".

The sizes for both the User Primary and Ghost address spaces are given. These are the total of both the Shared and Unshared areas, and are given in octal. Each Task Control Block (TCB) on the active TCB chain is now listed. Information given is:

a) TCB address
b) TCB status word (?TSTAT) with any bits explained
c) Last or current TCB System Call (?TSYS). Note that this can be values other than the System Call word as defined in the section on "User Process Analysis".
d) TCB Priority (?TIDPR - right byte).

Figure 4.8 illustrates some of the various TCB output formats that can occur. There are a few cases where the "System Call" word that is indicated on the printouts is not correct and can definitely be misleading. Refering to Figure 4.8, the first example has the TCB "Pend" bit (bit Ø) set and the System Call word is not of the form x76xxx, x77xxx, 177777 indicating a ?DELAY, therefore, the System Call word is valid and indicates a ?INTWT. The second example illustrates ?TSYS being used as a ?REC/?XMTW mailbox address which is misinterpreted as a System Call. In the third example, the User TCB had issued a System Call that uses the Ghost. The Ghost is explained in more detail in Section 5 -

Figure 4.8 Sample TCB Printouts

```
#1  TCB AT 447
    STATUS:          100000
      TASK PENDED
    SYSTEM CALL: ?INTWT
    PRIORITY: 0
```

System Call word is valid because ?TSTAT bit Ø = 1 and not of the form x76xxx, x77xxx, or -1.

```
#2  TCB AT 473
    STATUS:          040000
      WAITING FOR OVERLAY AREA OR .XMTW/.REC
    SYSTEM CALL: ?SOPEN
    PRIORITY: 0
```

System Call word is invalid since ?TSTAT bit 1=1. Contents of ?TSYS is probably ?REC/?XMTW mailbox address

```
#3  TCB AT 423
    STATUS:          004000
      IN GHOST CONTEXT
    SYSTEM CALL: ?GPROC
    PRIORITY: 0
```

System Call word is the Ghost System Call The User had originally issued a ?PROC which caused the Ghost to issue the ?GPROC. Info on the User System call is contained in the Ghost TCB Extension.

```
#4  TCB AT 423
    STATUS:          100900
      TASK PENDED
    SYSTEM CALL: 177777
    PRIORITY: 0
```

System Call word has been overwritten. By checking further we can find that the System Call was a ?DELAY and ?TSYS contains the TCB delay chain link.

```
#5  TCB AT 473
    STATUS:          100000
      TASK PENDED
    SYSTEM CALL: ?IREC
    PRIORITY: 1
```

System Call word is valid.

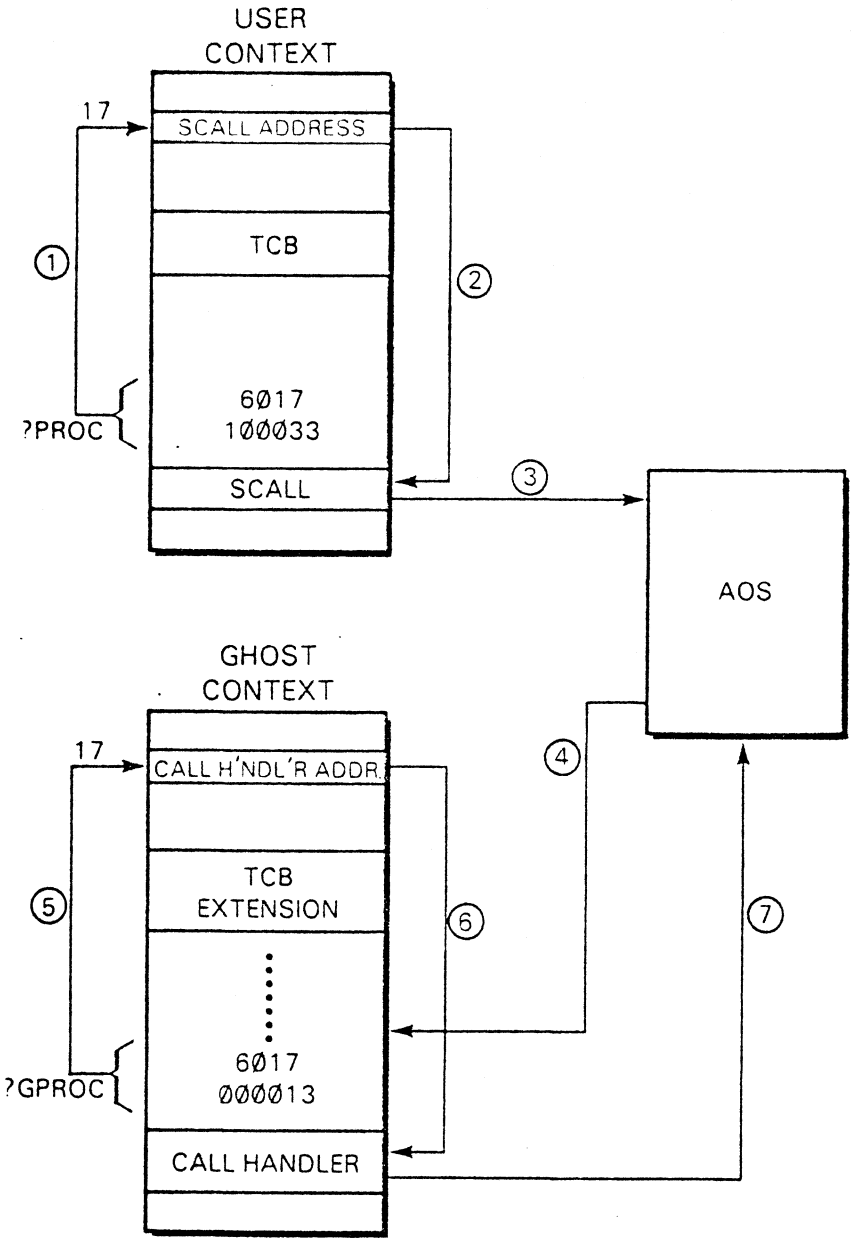
```
#6  TCB AT 543
    STATUS:          000000
    SYSTEM CALL: ?ISEND
    PRIORITY: 1
```

System Call word may or may not be valid. Since bits are no longer set in ?TSTAT, we do not know whether the call was a valid ?ISEND or the mailbox due to a ?REC/?XMTW.

"Traps". Figure 4.9 illustrates the User/Ghost System Call Interface. Effectively, the User issues the ?PROC system call which causes the Ghost to issue the ?GPROC system call. Since the Ghost uses the User TCB, all the parameters for the ?PROC system call are contained in the TCB extension. Because ADA scans the contents of the TCBs and not the extensions, only the ?GPROC system call will appear in the TCB printout. The Key to recognize this is the "In Ghost Context" bit.

Example #4 shows a TCB printout where the System Call word is "177777". This is usually because that task had issued a ?DELAY. Example #5 illustrates another valid System Call TCB where the call is ?IREC. Note that there is a difference between the ?TSYS usage of ?IREC and ?REC!! Example #6 illustrates a TCB printout where the TCB status is "Ø" and the last System Call is indicated as a ?ISEND. This may or may not be valid since at this point we do not know whether the status at the time of "System Call" was a "1ØØØØØ" or "Ø4ØØØØ".

From scanning the Process's TCBs bearing in mind the above exceptional cases, we can get an idea of what the User Process was doing when the "crash" occurred.



FS-0091

Figure 4.9

Simplified User/Ghost System Call Interface

BIT	POSITION	NAME	BIT POINTER	"RENAME"	NOTES	MEANING (*Bit is not used)
Ø	1ØØØØØ	PSRDY	BPSRY (1ØØ)	PSRDB	1	"NOT READY TO RUN"- this can be for various reasons as described in Note 1
1	Ø4ØØØØ	PSETR	BPSEN (1Ø1)	PSBDE PSENB	18	"DON'T ENTER"- indicates a condition where the Process's TCBs are not allowed to be rescheduled, the Process has outstanding System Calls,and the current TCB is not Ready to run. Therefore do not "Block"but do not enter the Process to run the TCB
2	Ø2ØØØØ	PSPND	BPSPN (1Ø2)		*	
3	Ø1ØØØØ	PSEW'	BPSEW (1Ø3)	PSEWB	2	"SCHEDULER ACTION"- this bit is used to indicate to the scheduler that an "action" for this Process is already in operation and not to look further.
4	ØØ4ØØØ	PSBRK	BPSBR (1Ø4)	PSBRB		"OPERATOR INTERRUPT"- will be set if a "^C ^A" (when bit PFNIN=Ø), a "^C ^B" or a "^C ^E" has occurred or if a "hardware" or "Data Base" Trap has occurred
5	ØØ2ØØØ	PSUNP	BPSPU (1Ø5)		*	
6	ØØ1ØØØ	PSBAG'	BPSBG (1Ø6)	PSBGB		"SWAP THIS PROCESS"- will be set if the Process has been setup for a "Swap Out" due to a pre-emption for core.
7	ØØØ4ØØ	PSBLK	BPSBL (1Ø7)	PSBLB		"BLOCKED"- will be set when the Process is "Blocked" and on the Blocked Queue
8	ØØØ2ØØ	PSDP	BPSDP (11Ø)	PSDPB	3,4	"DAEMON REQUEST UP"- will be set if the Process requires a Daemon Control Block initiated
9	ØØØ1ØØ	PSMWT	BPSMW (111)		16	"WAIT FOR MEMORY KEY TO CHANGE"- the Process needs more memory but could not get it. This can be either a Physical Page or AOS Logical Address Space (See bit PFASG)
1Ø	ØØØØ4Ø	PSTSU	BPSTU (112)			"TIME SLICE IS UP"- will be set if the Process's System Call will use more "time slice" than is left in offset PSLCN
11	ØØØØ2Ø	PSPOP	BPSPOP (113)		*	
12	ØØØØ1Ø	PSRUN	BPSRN (114)	PSRNB		"RUNNING"- will be set to indicate that this Process Table or Control Block is presently executing
13	ØØØØØ4	PSSLN			*	
14	ØØØØØ2				*	
15	ØØØØØ1				*	

Table 4.1 PSTAT Status bits (offset +4)
(1ØØ - 114)

Table 4.2 PFLAG Status bits (offset +12)
(24Ø - 257)

BIT	POSITION	NAME	BIT POINTER	"RENAME"	NOTES	MEANING
Ø	1ØØØØØ	PFNIN	BPFNI (24Ø)		5	"NO OPERATOR INTERRUPTS" - if this bit is set "^C ^A" interrupts are ignored
1	Ø4ØØØØ	PFPRE	BPFPR (241)		6	"PRE-EMPTIVE RESIDENT" - the Process type is "Pre-emptive"
2	Ø2ØØØØ	PFDEB	BPFDB (242)			"DEBUG ENTRY" - the Process was entered via the Debugger. Under the CLI this would be "DEBUG xxx". The bit will stay set for the total Process execution.
3	Ø1ØØØØ	PFFIR	BPFIR (243)		4	"FIRST EXECUTION AND LOAD" - will set on the first load of a Process into core or on a chain to prevent the Process Table Extension from being released
4	ØØ4ØØØ	PFELG	BPFEL (244)	PFELB		"PROCESS IS ELIGIBLE (IN CORE)" -
5	ØØ2ØØØ	PFTRP	BPFTR (245)	PFTRB		"TRAP BIT" - will be set if either a "Hardware" or a "Data Base" trap occurs
6	ØØ1ØØØ	PFINT	BPFIN (246)			"CONTROL A FLAG" - will be set on a "^C ^A"
7	ØØØ4ØØ	PFBRB	BPFBR (247)			"BREAK REQUEST" - will be set on a "^C ^E"
8	ØØØ2ØØ	PFTRM	BPFTM (25Ø)			"TERMINATE PROCESS" - will be set if the Process is being terminated, possibly due to its father's demise
9	ØØØ1ØØ	PFMBL	BPFMB (251)		7,8	"PROCESS CAN ONLY BE EXPLICITLY UNBLOCKED" - will be set if the Process is being "blocked" due to a termination (see note 7) or an explicit ?BLKPR system call
1Ø	ØØØØ4Ø	PFUBL	BPFUB (252)			"REQUEST TO UNBLOCK PROCESS" - will indicate to the Core Manager that the reason for the Process' being "Blocked" has been completed such as a delay expiring
11	ØØØØ2Ø	PFRSH	BPFRS (253)	PFRSB	14	"RESCHEDULE FLAG" - will be set to indicate that a TCB has become "Ready". This bit is needed since the "completion" may have occurred after the initial scheduler check
12	ØØØØ1Ø	PFFPR	BPFFP (254)			"PROCESS WAITING FOR FIRST LOAD" - the Process is being Swapped Out while the first load bit is set, ie it never got into core. The Extender will be reloaded from disk at next runtime
13	ØØØØØ4	PFSFT	BPFSF (255)	PSFSB		"SWAP PROCESS WON'T FIT NOW" - will be set if this "Swappable" Process will not fit into core and both ELINT and ELNON are Ø. Indicates not to keep checking until the core situation changes
14	ØØØØØ2	PFSWP	BPFSW (256)	PFSWB	6	"SWAPPING TASK" - the Process type is "Swappable"
15	ØØØØØ1	PFEBL	BPFEB (257)		17	"WAITING FOR SON TERMINATION" - will be set if this Process is blocked due to the start of a Son Process

BIT	POSITION	NAME	BIT POINTER	"RENAME"	NOTES	MEANING (*Bit is not used)
Ø	1ØØØØØ	PFSP	BPFSP (26Ø)	PFSPB	4	"SWAP IN/OUT IN PROGRESS" - refer to PFSWO for the In or Out
1	Ø4ØØØØ	PFSU	BPFSU (261)	PFSUB		"UNPEND SOMEONE WAITING FOR SWAP" - will be set if waiting for another Process to be pre-empted to release core so it can get more memory or if waiting for swap to complete so IPC message can be transferred to core or swap file
2	Ø2ØØØØ	PFCMQ	BPFCM (262)	PFCMB		"PROCESS IS ENQUEUED TO CORE MANAGER" - will be set when the Process is attached to the Core Manager Queue for Swap In Out
3	Ø1ØØØØ	PFDUP	BPFDU (263)			"UNPEND TCB AT HEAD OF DELAY CHAIN" - the TCB needs to be unpended since its time delay expired. Could not be done directly since the Process is "Blocked" or "Not Eligible"
4	ØØ4ØØØ	PFIPL	BPFIL (264)		*	
5	ØØ2ØØØ	PFIP2	BPFI2 (265)		*	
6	ØØ1ØØØ	PFWTO	BPFTO (266)	PFTOB	9	"BLOCK TIME OUT" - refer to Note 9
7	ØØØ4ØØ	PFBLE	BPFBE (267)			"BLOCK ENABLED" - will be set if the Process can be "Blocked" Always Ø for a "Resident" process
8	ØØØ2ØØ	PFWSL	BPFWS (27Ø)			"WAITING ON .SGNL" - the Process is waiting for the PMGR to complete an operation on its behalf - even if part of a termination
9	ØØØ1ØØ	PFSWO	BPFSO (271)	PFSOB		"PROCESS IS BEING SWAPPED OUT" - the Process is being swapped "Out" if set to 1, "In" if Ø and PSFP is set.
1Ø	ØØØØ4Ø	PFBWT	BPFBW (272)	PFBWB		"WAIT FOR TIMEOUT" - the Process has been enqueued to the Block Timeout Queue (TQUE) and is waiting for a timeout to occur or a System Call to complete
11	ØØØØ2Ø	PFATL	BPFTL (273)			"FATAL PROCESS TERMINATION" - the Process's Control Block has been terminated due to an error. The error code is temporarily saved in offset PSL (11) of the Process Table Extension
12	ØØØØ1Ø	PFSUP	BPSUP (274)			"SUPER USER" - Process has the "Super User" privilege
13	ØØØØØ4	PFASG	BPFAS (275)			"MEM WAIT BIT" - Process is waiting for "AOS Address Space"
14	ØØØØØ2	PFSHC	BPFSC (276)	PFSCB	1Ø	"SHARED AREA CHANGED" - will be set if the Process's Shared area has changed
15	ØØØØØ1	PFQSC	BPFQS (277)	PFQSB		"INHIBIT SCAN OF BACKED UP TCB REQUESTS" - will be set if the Process is going to be "Swapped Out" or "Blocked" Because of this, do not start any waiting TCB requests

Table 4.3 PFLG2 Status bits (offset +13)
(26Ø - 277)

BIT	POSITION	NAME	BIT POINTER	RENAME	NOTES	MEANING (*Bit is not used)
Ø	100000	PFTSE	BPFTE (3ØØ)	PFTEB	15	"AT LEAST 1 TIME SLICE ENDED" - will be set for "Swappable" Processes only and indicates that the Process has completed at least one time slice.
1	Ø40000	PFUBD	BPFUD (3Ø1)		17	"UNBLOCK DAD ON RETURN" - will be set if the Process is "Blocked" until the completion of a Son process.
2	Ø20000	PFPTM	BPFFT (3Ø2)		7	"PROCESSING A TERM" - will be set if the Process is being terminated. This bit allows only the first termination to have any effect, all sequential ones are ignored.
3	Ø10000	PFPCN	BPFPC (3Ø3)			"PROCESSING A CHAIN" - the ?CHAIN system call has been issued.
4	ØØ4000	PFCIE	BPFCI (3Ø4)			"HAS CREATED AN IPC TYPE ENTRY" - will be set if the Process has ?CREATE'd an IPC entry in its initial working directory.
5	ØØ2000	PFIWC	BPFIW (3Ø5)			"INTERRUPT WITHOUT USER CONTROL" - will be set if a "^C^B" or a "^C ^E" has occurred indicating that the Process will be terminated.
6	ØØ1ØØØ	PFIRS	BPFIS (3Ø6)	PFIRB		"INTERRUPT WORLD INTERRUPTED TASK" - will be set if an interrupt occurred while the Process's task was executing. A reschedule was done because of the interrupt.
7	ØØØ4ØØ	PFRSL	BPFSL (3Ø7)			"FIRST SLICE HAS NOT OCCURRED"
8	ØØØ2ØØ	PFSTM	BPFST (31Ø)			"SELF-TERMINATION OCCURRED" - the Process has terminated itself via the ?TERM system call.
9	ØØØ1ØØ	PFDIN	BPFDI (311)			"DELAYED OPERATOR INTERRUPT FLAG" - will be set if a "^C ^A" has occurred while the PFNIN bit equaled 1 indicating to ignore "^C ^A".
1Ø	ØØØØ4Ø	PFCFL	BPFCF (312)			"CCBs ARE FAULTED OUT" - will be set if the CCBs have not been loaded into core from the Swap File on disk because they had not been referenced in the last time slice.
11	ØØØØ2Ø	PFCUS	BPFCU (313)		11	"CCBs WERE REFERENCED IN LAST SLICE" - will be set if a System Call is issued that needs a CCB.
12	ØØØØ1Ø	PFCLD	BPFCL (314)			"LOAD IN CCBs (DAEMON BIT)" - the CCBs were not loaded when the Process was Swapped In but are now needed due to a System Call.
13	ØØØØØ4	PFPCH	BPFPH (315)	PFPHB	12	"HOLD ON >1 PARALLEL CALLS" - will be set when a System Call that must be executed by itself is being executed.
14	ØØØØØ2	PFRBI	BPFRI (316)		*	
15	ØØØØØ1	PFIDF	BPFID (317)			"IDEF OR USER POWER FAIL DONE" - will be set if the Process has defined an ?IDEF device.

Table 4.4 PFLG3 Status bits (Offset +14)
(3ØØ - 317)

BIT	POSITION	NAME	BIT POINTER	RENAME	NOTES	MEANING (* Bit is not used)
Ø	100000	PFSLN	BPSLN (320)			will be set if the Process needs a Daemon C.B. to process a Sync Line Transmit or Receive completion.
1	040000	PFDIS	BPFDS (321)			
2	020000					
3	010000					
4	004000					
5	002000					
6	001000					
7	000400					
8	000200					
9	000100					
10	000040					
11	000020					
12	000010					
13	000004					
14	000002					
15	000001					

Table 4.5 PFLG4 Status Bits (Offset +15)
(320 - 321)

Notes

1. The PSRDY bit will be set:

   a) if a Process is idle but cannot be "blocked" as in the case of a Resident Process. This bit is used to indicate not to run the Process.
   b) if a Process is on the blocked timeout queue (TQUE) waiting for a System Call or "timeout" completion.
   c) if a Control Block is pended on an AOS event. Refer to offset "CKEY".
   d) if the Core Manager is idle.

2. The PSEW bit will be set in the following cases:

   a) a Daemon Control Block is processing using bits 4-15 of PSTAT
   b) a ?CHAIN is being executed
   c) an initial Ghost load is being done
   d) a "^C^A" is being processed
   e) waiting for a Control Block to finish

3. Daemon Control Blocks are used to perform an operation for which no User System Call has been directly issued, therefore, no standard Control Block is available. These operations are:

   a) Process terminated
   b) Process Trapped
   c) "^C^B"
   d) "^C^A"
   e) Initial Process Load
   f) Load the Channel Control Blocks (CCB's)
   g) Specific Sync line functions

4. This bit will be set during Process creation. A temporary Process Table Extension has been created in "GSMEM" space so that constant remaps do not have to be done.

5. The PFNIN bit will be set in two cases:

   a) the Process has issued the ?ODIS system call which disables "^C^A".

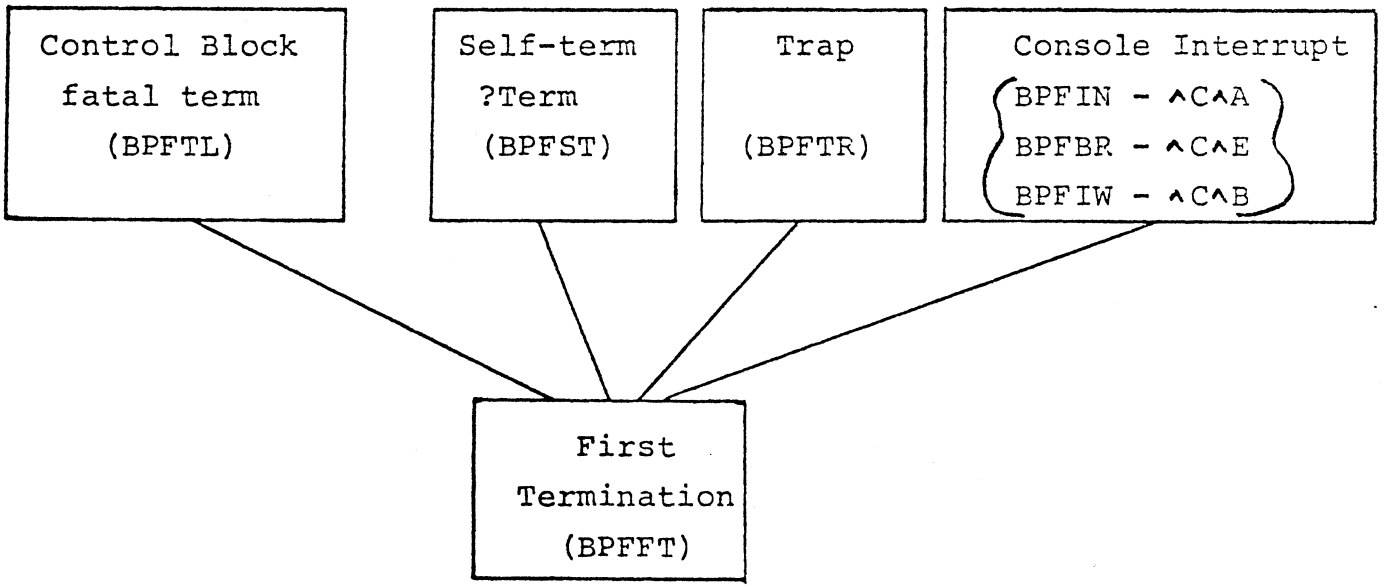
   b) a "^C^A" has occurred and the Process has defined a Console Interrupt Service Task via the ?INTWT system call. Once the first "^C^A" has occurred, this bit is used temporarily to ignore any sequential "^C^A" interrupts.

   If a "^C^A" occurs while this bit is set, it is ignored but the PFINT bit is set. As soon as this bit reverts to "Ø", a previously ignored "^C^A" will be serviced.

   Note also that if no ?INTWT routine is defined by the Process, any "^C^A" interrupts are ignored, although they will echo.

6. If neither PFPRE nor PFSWP is set, the Process type is "Resident".

7. Below is a block diagram of the multiple ways the "First Term" bit can be set:



Even though only the 1st termination request for a Process will be honored, multiple single term request bits can be set since these will be set prior to the BPFFT bit being checked.

8. A "Resident" Process can never be "Blocked".

9. The PFWTO bit will be set if:

   a) the Process which has been put on the "TQUE" has "timed out". It will now be moved to the "Block" queue.
   b) an explicit ?BLKPR system call has been executed against this process. This will cause the Process to be moved directly from the ELQUE to the Block Queue.

10. The Shared Area for a Process can change due to:

    a) the release of a shared slot
    b) the reading of a shared block
    c) the initial Process load
    d) If a block of the Primary Context is remapped to the Ghost Context, such as TCB area, etc. , the Ghost slot is within its Shared Definition space.

    If the Shared Area has not changed since the last disk swap, there is no need to rewrite the information to the Swap File.

11. The following System Calls will reference a CCB:

a)	?RDB	k)	?FSTAT
b)	?WRB	L)	?GNAME
c)	?SPAGE	m)	?GTERM
d)	?GOPEN	n)	?MEMI
e)	?GCLOSE	o)	?SSND
f)	?SOPEN	p)	?SRCV
g)	?GNFN	q)	?SPOL
h)	?GCHAIN	r)	?SEPL
i)	?ABTC	S)	?SDPL
j)	?TABT	t)	?SGES

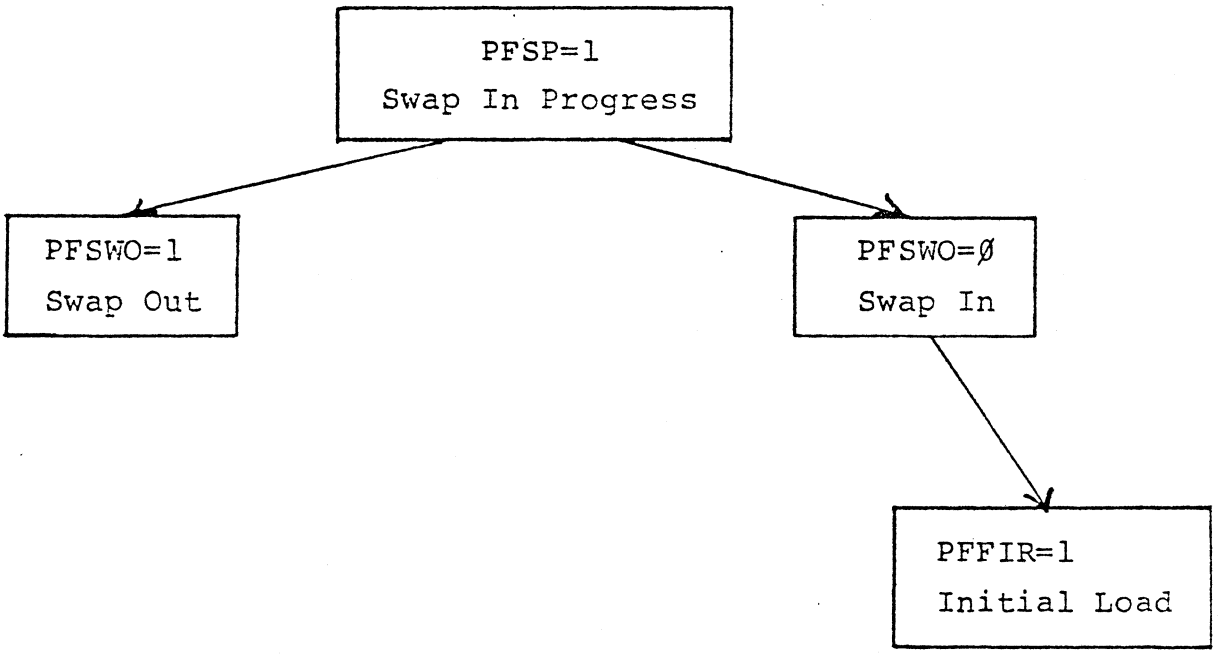
    This bit tells the system on a "Swap In", whether the CCBs were used the last time the Process was in core and whether to read them in now.

12. The following System Calls must be executed by themselves:

    a) ?PROC
    b) ?MEMI
    c) ?CHAIN
    d) ?RPAGE
    e) ?SSHPT
    f) ?SPAGE
    g) ?SOPEN

    Note that any of the above calls will change the state of the Process Environment and therefore muot be executed while no other System Calls are being executed.

13. The general "swapping" flow is as follows



14. The Reschedule Flag (PFRSH) is set to indicate an event completion. Either of the following three cases will cause it to be set:

    a) A Process Table has been "Readied" to run by the Interrupt World due to an event completion ("Significant Interrupt")
    b) A Process Table has been "Readied" to run because of the completion of a Control Block (System Call) or a Daemon.
    c) The PMGR Process Table has this bit set when it is "Readied" because the DCU5Ø needs service.

15. This bit, PFTSE, is used as a "timed out" indicator when a Process must be pre-empted because the core is needed. A "timed out" Process is usually the first one to be pre-empted if the exponent >3 but <6. If 6, the Bias factor must be taken into consideration.

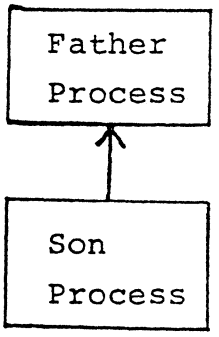
16. Bits PSMWT and PFASG can be used to determine the type of core "wait".

    a) If PSMWT and PFASG, or just PFASG, the AOS dynamic Slots for logical addresses between 34ØØØ and 57777 are all in use, therefore additional physical core cannot be integrated into the system logical address space.

    b) If only PSMWT, the system needs additional physical core and none is presently available. This may result in a pre-empt of another Process.

    The key for the memory wait is saved in offset PMKEY (51) of the respective Process Table. The key is a copy of a core counter, MKEY, that is incremented as core is released. If the scheduler finds MKEY not equal to PMKEY, an attempt is made to get more core.

17. Bits PFUBD and PFEBL are used in a Father/Son linkage blocking scheme. The diagram of this linkage is as follows:



PFEBL Set:
"Waiting for Son termination"

PFUBD Set:
"Unblock DAD on Return"

18. TCB rescheduling can be inhibited by setting specific bits in the User Status Table (UST) flagword, USTFL (offset 7). These bits are:

    a) ?UFDR (bit 2) - Inhibit Rescheduling
    b) ?UFID (bit 4) - Process is in Debugger
    c) ?UFPH (bit 5) - Primary Task Runtime hold on Rescheduling

III. Core Dump Analysis

Using the flow illustrated in Figure 4.2, there are only eight paths that will request that a core dump be taken. The procedures for taking the core dump are described in Section 6 between Pages 6-2 to 6-6. The running of FIXUP after taking the core dump is also described in those pages. The operating system is then re-booted and the core dump loaded as shown in Figure 4.1Ø. (Note that the dump name will vary depending on the month/ day and the symbol table is the actual operating system one). The core dump is now accessed using the "SYSDMP" Utility to determine the value of some specific locations as illustrated in Figure 4.1Ø.

Using the contents of the illustrated variables in addition to the flow illustrated in Figure 4.12, we can determine where in the system/user we were at the time the dump was taken. In our case, since INTLV=-1, we were not servicing an interrupt. Since SYSIN=1, we were in the system somewhere. We also note that INCHK=Ø and the PC at the time of stop was 2Ø56. The contents of CC (365) is 411ØØ, the address of either a Control Block or Process Table. Next we can examine the contents of 411ØØ which is found to be 411ØØ. For a CB, it will be either Ø or 76xxx. For a Process Table, it will be the address of the Process Table. Since it was a Process Table and PSRUN=1 (not shown), we will go to (4).

Under each of the lowest level blocks is a number within a circle. This is a reference as to where within this section additional information can be found and locations to check in evaluating the problem. A list of those areas is:

Number	Subject	Page
(1)	User Process Analysis	4-44
(2)	System Activity Analysis	4-53
(3)	System Call/Daemon Analysis	4-65
(4)	Process Table Analysis	4-74
(5)	Interrupt Analysis	4-76

Figure 4.1Ø Core Dump Load and Initial Analysis

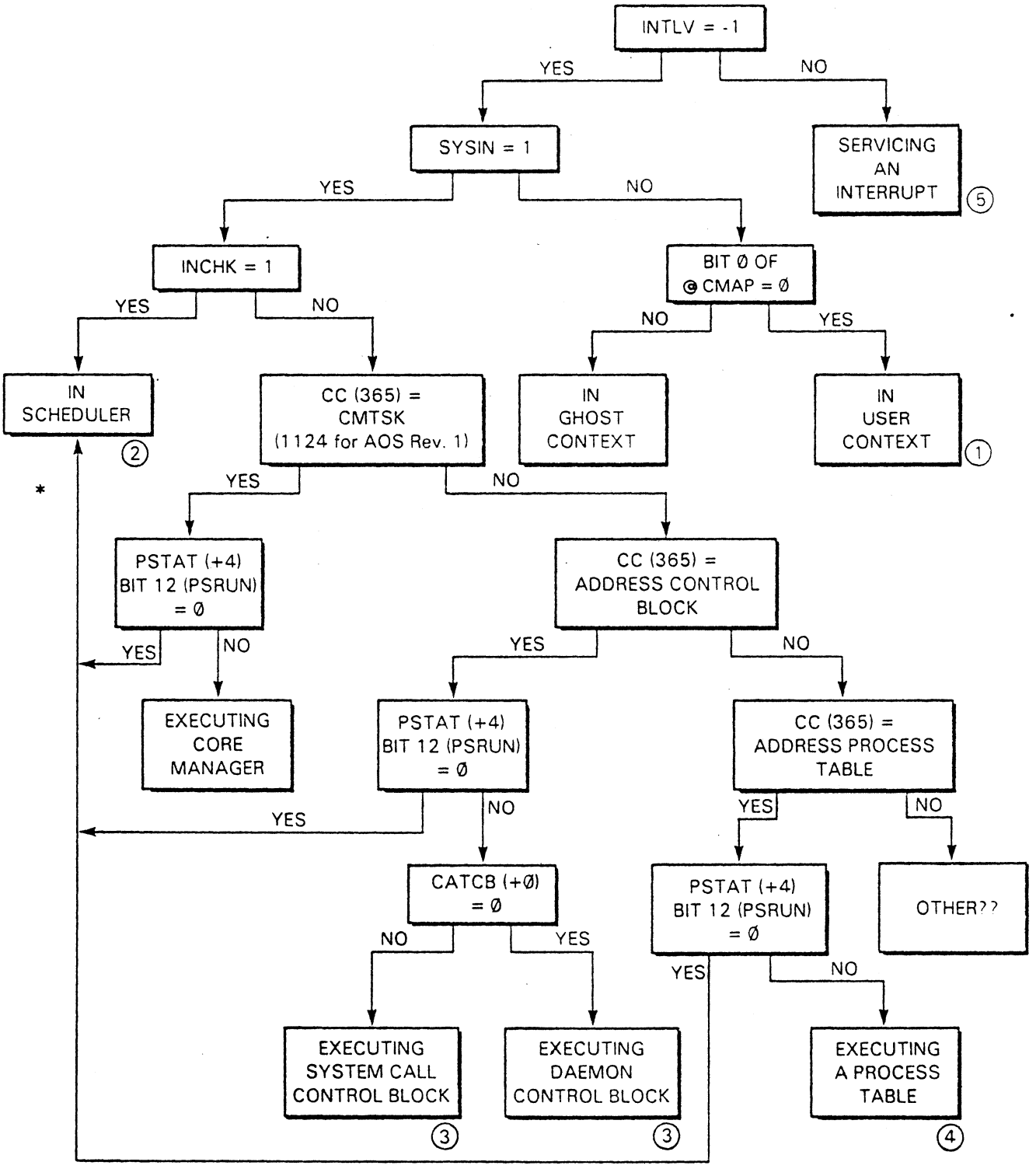
```
)DIR  :UTIL↓
)COPY  DUMP1Ø11  @MTAØ:Ø↓
)X  SYSDMP/S=:SYSGEN:AOS.ST  DUMP1Ø11↓
 AOS SYSTEM DUMP ANALYZER REV 1.1
+INTLV: -1  +↓
+SYSIN: 1  +↓
+INCHK: Ø +↓
+
+Ø:2Ø52+↓  (PC@STOP=>2Ø56)
+
+365: 411ØØ+↓
+
+
+BYE↓

)
```

Figure 4.11 Example AOS Dump Analyzer (ADA) Startup

```
)DIR  :UTIL↓
)LIST  @LPA↓
)X ADA DUMP1Ø11  :SYSGEN:AOS.ST↓
 ADA
 IN COREM
 IN CBLK
 IN CBLK
 IN PTBL
 IN PTBL
 IN PTBL
 IN PTBL
)LIST @LIST↓
)
```

Figure 4.12 Where Was I?



* The contents of CC are residual.

FS-0097

If multiple core dumps need to be analyzed, examining each one to determine the values needed to use Figure 4.12 can be rather time consuming. This can be mechanized by inputting all the commands to "SYSDMP" using a disk file rather than typing them in each time. This will work perfectly since the values that we want to check will be the same each time. Figure 4.13 shows the CLI sequence needed to create the command file, COM. It is just a string of "SYSDMP" commands. Also in Figure 4.13, a macro is created so that "SYSDMP" can be executed each time with the minimal number of arguements being typed. Since I use Console #Ø (master console) when using SYSDMP, I set the "/L=" switch to "@CONØ". It could be set to any console or disk file.

Figure 4.14 illustrates the use of the "DED.CLI" macro. By typing in the macro name, the dump symbol table, and dump file name, the needed values are typed on the CRT. Using these values and the flow in Figure 4.12, we can determine that AOS was "Executing System Call Control Block" when the dump was taken.

Figure 4.13 Command File to get Initial Parameters

```
*) CREATE/I  COM
*))INTLV:
*))SYSIN:
*))@CMAP:
*))INCHK:
*))365:
*))@365:
*))@365+4:
*))366:
*))BYE
*)))
*)
*) CREATE/I  DED.CLI
*))X  SYSDMP/I=COM/L=@CONØ/S=%1%  %2%
*)))
*)
```

Figure 4.14 Example of "DED" Output

```
*)DED  :SYSGEN:AOS1.Ø4.ST  DUMP1111
AOS SYSTEM DUMP ANALYZER REV 1.1
+INTLV:177777+
+SYSIN:ØØØØØ1+
+@CMAP:Ø467ØØ+
+INCHK:ØØØØØØ+
+365:Ø34653+
+@365:Ø76423+
+@365+4:ØØØØ1Ø+
+366:ØØØ613+
+BYE

*)
```

(1) User Process Analysis

This point indicates that we were "hung" in either a User or System Utility type Process. The three usual ways of being "hung" in a User Process are by doing a "JMPØ", a "JMP.", and a "tight code loop". It now has to be determined whether we were in the actual Process or in its Ghost counterpart. If bit Ø=1 of the contents of the address contained in CMAP, we were in the Ghost. The case where we were in the Ghost will be discussed later. Since we know that CC (365) is the address of this Process Table, we can determine the PID of this Process by examining:

(CC) + 42 : pid

Some of the more universal PID assignments are:

PID	Process
1	PMGR (Peripheral Manager)
2	Operator CLI
3	Usually EXEC
4	Usually XLPT
5	Usually CLI

Note that PID's 1 and 2 will always be the same but PID's 3, 4, and 5 might vary depending on how the User has entered the commands in the "UP macro". Once we actually "map" the "User" process under SYSDMP, we can verify the Process type by checking offset USTTC (13) and USTST (21) of the User Status Table. These offsets define the "number to Tasks" and "Shared Area Starting Block #", and will usually vary from Process to Process. For the common System Utilities:

System Utilities	USTTC (413)	USTST (421)
PMGR.PR	6	Ø
EXEC.PR	21	Ø
PED.PR	3	Ø
CON.PR	4Ø	Ø
MPROC.PR	1	Ø
CLI.PR	3	2Ø
XLPT.PR	3	Ø

The actual analysis of the logical address space for a User process can be a real "pain" since its contiguous logical address space can be "mapped" to totally uncontiguous physical memory pages. A technique can be used to very much facilitate this ordeal. A few more locations need to be recorded though:

(CC) + 25: address of primary map
(CC) + 75: physical page of primary map in bits 8-15

The procedure to do this is illustrated in Figure 4.15. The reason for changing from "SYSDMP" to "DEDIT" is that "SYSDMP" does not allow the modification of any locations. If I were to attempt to change a location using "SYSDMP", I would get an error message:

"ILLEGAL COMMAND FOR UTILITY"

I use DEDIT to map the AOS 6ØØØØ slot to the physical page that contains the User Process logical to physical core translation table. When I enter "SYSDMP" the second time, I issue the "MAP" command with the address of that table. I am now "mapped" using the User Process Table Map and all addressed I issue are mapped using that table. At this point, looking through a User Process is as easy as if it were a "Break file".

Since I can now directly access the Process's logical address space, the first value to check would be USTTC. This value in addition to the PID and a knowledge of the system activity, is usually enough to determine whether the Process was a System Utility. If the Process is a System Utility, there is not much more that can be done. If the problem is critical, I would suggest your getting in touch with me as I have listings for most of the System Utilities. We can determine why the Utility was "hung", where it was, and whether the "dump" should be submitted for further analysis. If the situation is non-critical, perhaps the local Systems Engineer can provide additional help and/or submit the dump via the Systems Engineering channels.

At this point, the Process that was hung was a User Program written in either Assembly Language or a "higher level" language

Figure 4.15 "Mapping" a User Logical Address Space

```
)X  SYSDMP/S=:SYSGEN:AOS.ST  DUMP1Ø11
AOS SYSTEM DUMP ANALYZER REV 1.1
+365:  411ØØ+
+
+411ØØ+42:  3+
+
+411ØØ+25:  617ØØ+
+411ØØ+75:  6162+
+
+BYE

)X DEDIT DUMP1Ø11
AOS FILE EDITOR REV 1.1
+473+3Ø:  61777+6Ø162
+
+BYE

)X SYSDMP/S=:SYSGEN:AOS.ST DUMP1Ø11
AOS SYSTEM DUMP ANALYZER REV 1.1
+Ø:2Ø52+
+
+MAP 617ØØ
+
+413: xxx
   :
   :
   :
   :
   :
+BYE
```

All the addresses given are translated using the User logical map. This is now just as easy as examining a "BREAK" file.

such as FORTRAN. From the PID and the number of tasks, the User might be able to identify the Process and provide more insight into the problem. If an analysis of the state of the User Process is necessary, the following is a procedure that can be used. Note that this procedure applies as well to System Utilities because in actuality they are "User Programs". A "load map" needs to be provided by the User for this Process.

A) Examine the various Process Pointers

1) USTCT (414) - indicates the current TCB
2) SP (4Ø) - Current Task Stack Pointer*
3) FP (41) - Current Task Frame Pointer*
4) Location 1 - Indicates whether "rescheduling" should be done. Will be set by the:
   a) System Call interface module (SCALL)
   b) ?DRSCH Task Call

* especially useful if FORTRAN is involved

Note that parameter layouts for the User Status Table (UST) and Task Control Blocks (TCB) can be found at the end of Section 5 "Traps".

B) Examine the current TCB

The key thing that we wish to determine is "what the current TCB was doing". This can probably help to determine the reason for the Process "hang".

If the Process was doing a "JMPØ" or a "JMP.", the values stored in the TCB offsets should be the same as the values that were copied down when the system was "halted" to take the core dump. In this case, either set of values can be used. If the Process was "hung" in a code loop, these sets of values will differ and should be looked at to see what information each can provide.

1) PC at Stop - Examine this value
   a) Was it "Ø"? if so the Process had done a "JMPØ". Overlay areas under FORTRAN 5 are initialized to Ø's. If a User does not check his "ERR" variable, he can

get an error on an overlay load then jump into the overlay area and voila a "JMPØ". This can also be caused in FORTRAN 5 if a User does not provide an "ERR" variable in FORTRAN 5 calls that require it. This can cause a "1" to be randomly deposited in the Process space depending on a stack address. Note that a "1" is the "no error" condition passed back to the User.

b) Examine the contents of this address. Was it a "4ØØ". This indicates that a "JMP." was coded in the User Process. This is usually done in the following sequence:

```
?READ       ;any system call
JMP.        ;error return
:           ;normal return
:
```

Determine what the system call was. Note ACØ at this point will contain the error code. A better way to code the error returns is by defining a variable called "ER" in ZREL as:

```
ER:  @.
```

Then coding the error returns as:

```
JMP @ER
```

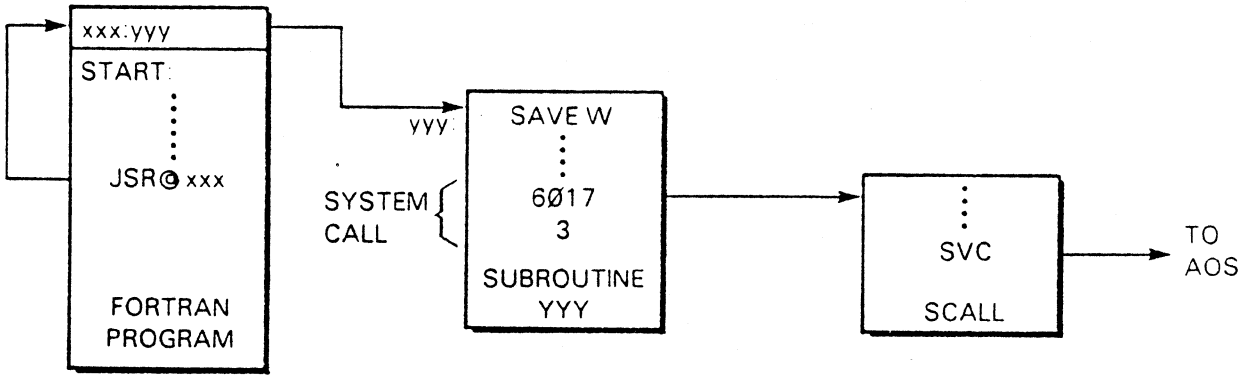
This will cause a "Trap" to occur with a "Defer violation" and the contents of the accumulators. At least the Process doesn't "hang".

c) If it was some other value, equate this value to the load map and then to a listing of the User Process. Determine why the Process was "hung" in the loop.

2) ?TPC - This location will contain the PC at the time of the last significant interrupt. It will probably be "Ø" if the task had been doing a "JMPØ" because of RTC interrupts since the "JMPØ" was first executed.

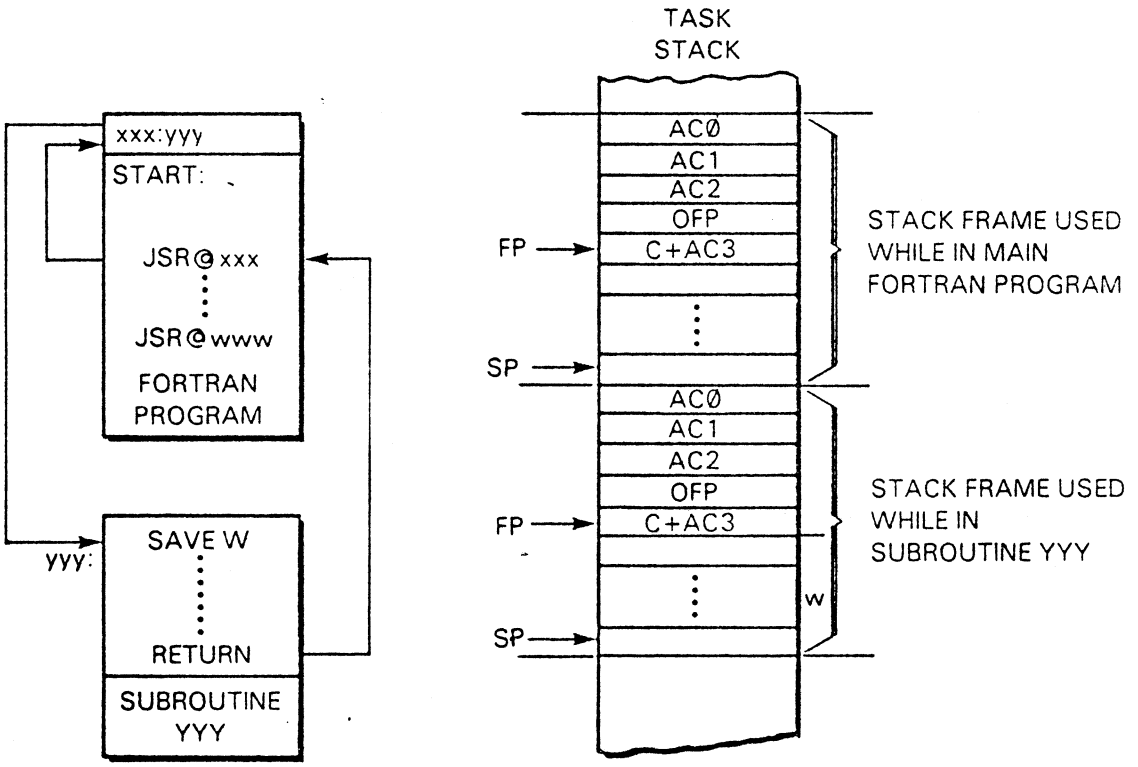
Note that when using FORTRAN, a convention that can be

Figure 4.16 FORTRAN System Call Linkage



FS-00098

Figure 4.17 Stack/Sub-routine Call Linkage



FS-00099

assumed is:

?TPC - address where the JSR to the FORTRAN subroutine took place.

?TAC3 - address where the actual system call took place.

Figure 4.16 illustrates this linkage.

3) ?TAC2 or AC2 at Stop - refer this location to the load map and see if the address is significant.

4) ?TAC3 or AC3 at Stop - determine whether the value is the current frame pointer (FP). This would be the case if we are hung in a sub-routine after a "SAVE" was done. If not FP, refer the location to a load map and see if the address is significant. Many sub-routines return to the main routine via a "JMP 1,3", "JMP Ø,3", etc. This value might be used to "place" the program to an area of code prior to a "JMPØ" being done.

5) ?TSTAT - would expect that none of the bits should be set. If any are set, their meanings should be carefully evaluated.

6) ?TSYS - will contain information about the last User started operation, ie: System Call, Task Call, etc. Can contain:

   a) If System Call - the numeric value of the call. Note that if BØ=1, the System Call will also use the Ghost. A table of these values and their mnemonics is given at the end of Section 5, "Traps".

   b) If a ?DELAY Call - this location is used to hold the TCB link while the TCB is attached on the Process Table delay chain. The value is of the format 76xxx, 77xxx, or -1.

   c) If a ?REC/?XMTW - this location will contain the "Message Key", ie: the core address of the message "pigeon hole".

d) Sync lines - if Sync lines are being used by the TCB, the synchronous line# is stored in this offset during a Sync System Call.

C) Examine the Active TCB Chain

The key thing that we wish to determine is "what each of the TCB's on the active chain was doing". An analysis similar to steps 1-6 for the current TCB should be done for each TCB on the active chain.

D) Examine the Stack for the Current TCB

This is especially helpful if the Process has done a "JMPØ" or has jumped off to code where it should not be.

Refer to Figure 4.17 which illustrates a FORTRAN Main Program which calls a Subroutine YYY. The subroutines usually do a "SAVE" at their beginning and a "RETURN" when they are finished. After the stack is layed out using the "OFP" to point from frame to frame, we can determine the routines or sub-routines that correspond to each frame by equating, the "AC3" value from the respective frame to it's core contents minus one. In the example, if we take "AC3" from the Subroutine YYY frame, subtract one from it, and examine that core location, we will find an instruction of the form "JSR @xxx". Since "xxx" is usually a "global" and a Page zero value by examing the "load map", we can probably determine the subroutine.

If the Process had done a "JMPØ" and the SP and FP were pointing to the Subroutine YYY frame, we can assume that we were in Subroutine YYY when we did the "JMPØ". If we had left the subroutine orderly, the "RETURN" would have "pop'd" a frame.

If the Process had done a "JMPØ" and the SP and FP were pointing to the Main FORTRAN Program frame, we could examine the next lower stack frame to see where the "JSR" was done from. In our case it would have been a "JSR @xxx". This tells us that we have at least passed this point in the main program prior to doing the "JMPØ". By scanning the main

program we can find the next subroutine call as "JSR@www" (in Figure 4.17). We now know that the "JMPØ" was done in the Main Program somewhere between the "JSR@xxx" and "JSR@www" subroutine calls.

Although most failures are not as simple as I make them sound, using the above observations, tools, and techniques, you can easily put yourself in the "ballpark" of where the problem occurred. From there it might be scanning a sub-routine using SYSDMP (or DEDIT if a Breakfile) to check for a word of all "Ø's", a word with only a "1", etc; comparing the AC's at the time of JMPØ for a key constant, finding where in the routine it was loaded and further sub-dividing the problem.

(2) "In Scheduler" Analysis

This point generally indicates that the system is either waiting for an event to complete prior to continuing onward or all the Process Tables are "not ready to run" and never becoming "Ready". We must examine each Process Table and Control Block to try to determine whether it is in the proper state. Process Tables can be placed on one of four queues:

a) Eligible Queue (ELQUE) - will contain "Resident" Processes that have core allocated irrespective of whether they are ready to run; other process types that are ready to run; and other process types that are being "timedout" on the TQUE prior to being put on the "Blocked Queue".

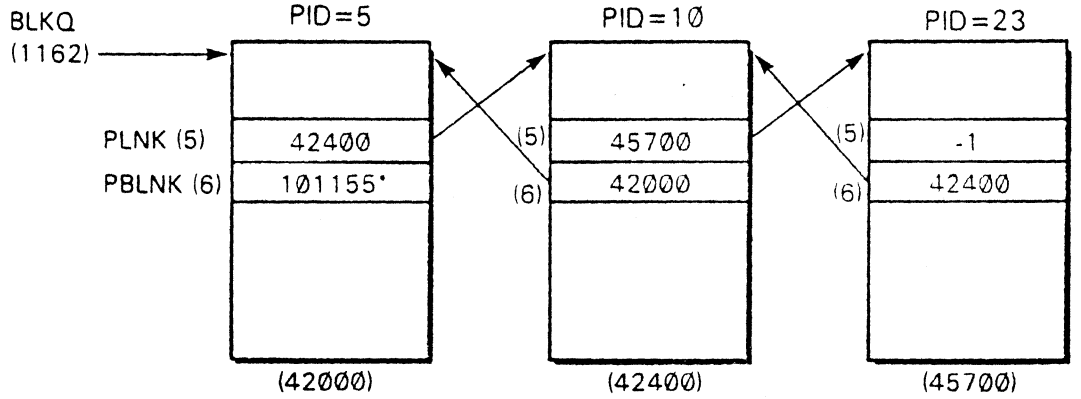
b) Blocked Queue (BLKQ) - will contain any process type except "Resident" which is "Blocked", usually indicating that the process is not ready to run.

c) Ineligible Resident Queue (IERES) - will contain "Resident" (only when the Process is first created) or "pre-emptible" type processes that do not presently possess core.

d) Ineligible Swap Queue (IESWP) - will contain any "swapable" type processes that are "Ready to run" but do not presently possess core.

The variable in parenthesis after the Queue name will be a location in core which will point to the first table on the queue. The Process Tables are linked on their respective queue using offsets PLNK (5) and PBLNK (6). Refer to Figure 4.18. These are the forward and backward links on the chain, respectively. As mentioned above, Process Tables that are linked on the Eligible Queue can also be linked on the "Timeout" Queue (TQUE). This queue is used initially when a "Swappable" or "Pre-emptible" Process is "not ready to run". The process is linked on the TQUE for a period of time prior to being moved to the "Blocked" Queue. This procedure keeps processes that will only be "not ready to run" for a short period of time due to a System Call from being unlinked from the ELQUE and linked on the BLKQ only to have the process immediately become "Ready" again and have to be moved

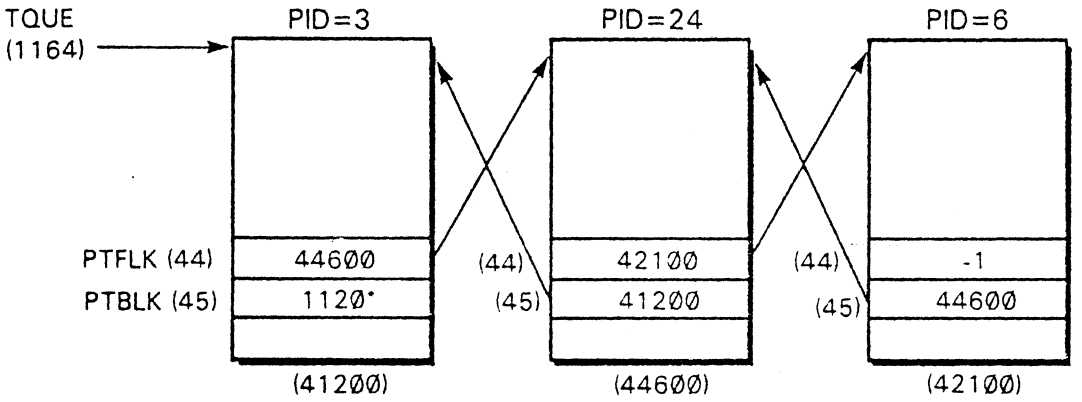
Figure 4.18 BLKQ Linking Example



FS-0100

*Note that 1155+5=1162

Figure 4.19 TQUE Linking Example



*Note that 1120+44=1164

FS-0101

back. Process Tables are linked on the TQUE using offsets PTFLK (44) and PTBLK (45). Refer to Figure 4.19. These are the forward and back links, respectively. Offset PTICK (34) in the Process Table Extension contains the count which is counted down at each RTC interrupt. Since this count is set initially to 2 and with the RTC at 10HZ, the TQUE wait is around 200ms.

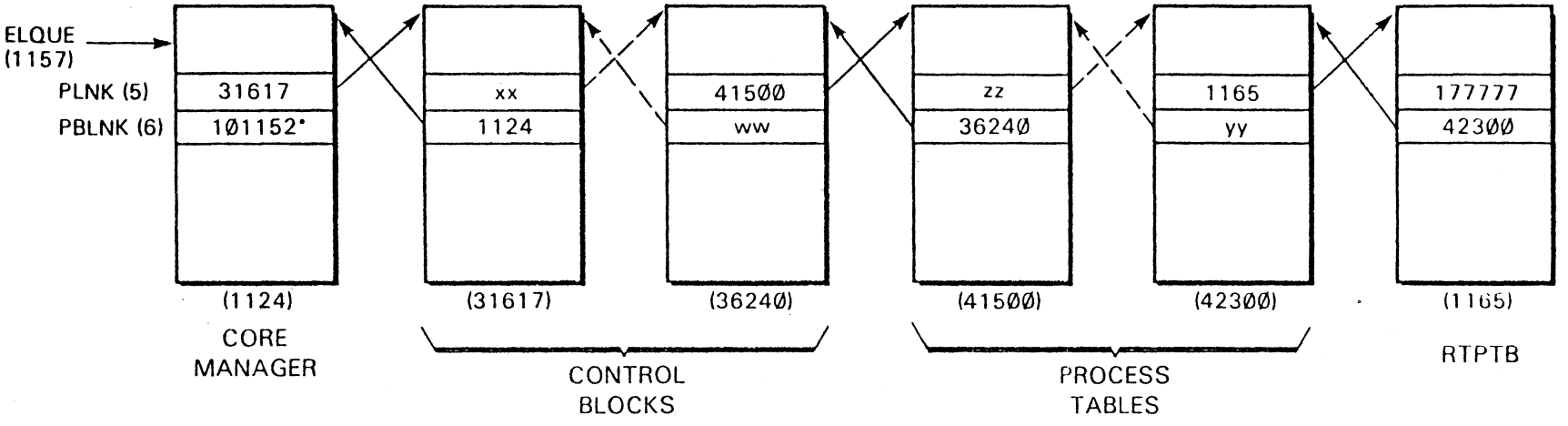
In addition to he Process Table data bases, there are standard Control Blocks and the Core Manager Control Block. The standard Control Blocks will either always be on the ELQUE or not exist at all. The order of these data bases on the ELQUE is shown in Figure 4.2Ø. The Core Manager is <u>always</u> the first data base on the ELQUE. The Core Manager is used, as the name implies, for the management of core, ie: allocation/de-allocation, pre-empting of Processes to regain core, etc. If the Core Manager has nothing to do, he will still be first on the ELQUE but a flag is set as "not ready to run".

Control Blocks will be created on an "as needed" basis and because of this, there may be two or four or none at all on the ELQUE. There is a maximum of 5 Control Blocks at any one time in the system. There are two types of Control Blocks:

a) <u>System Call Control Blocks</u> - these are Control Blocks created in response to a Process's System Call and are used to complete that System Call. The first word of the C.B., offset CATCB (Ø), will contain the TCB address of the TCB that is making the System Call. The format will be x76xxx or x77xxx. Offset CPTAD (15) will usually contain the address of the Process Table where the TCB resides.

b) <u>Daemon Control Blocks</u> - These are Control Blocks created to do a function for which there is no System Call Control Block because no System Call was issued. Their functions are:
   1) Process Terminated
   2) Process Trapped
   3) "^C^B"
   4) "^C^A"
   5) Initial Process load

Figure 4.2Ø Order of the data bases on the ELQUE



The dotted lines indicate that there can be more data bases in the respective grouping than the two shown above.

*Note that 1152+5=1157

FS 0102

6) Load the Channel Control Blocks (CCB's)

7) Specific Sync line functions

The first word of the C.B. for a Daemon, offset CATCB(Ø), is set to "Ø". Note that a couple of System Calls set CATCB to Ø near the end of the call to prevent bit Ø of ?TSTAT from being reset since the Process's core may have already been released.

These are linked on the ELQUE using offsets PLNK (5) and PBLNK (6), the same as for Process Tables. When the function for which the control block was created is finished, it is unlinked from the ELQUE and its core goes into an available pool for future reuse.

When we scan the various queues, there is a status word for each Process Table and Control Block. There are also four flag words for each Process Table. The offsets for these words are:

Word	Mnemonic	Offset	
Status	PSTAT	4	
1st Flag	PFLAG	12	Process Tables Only
2nd Flag	PFLG2	13	
3rd Flag	PFLG3	14	
4th Flag	PFLG4	15	

Each bit in each of the words represents a specific condition about the Process Table or Control Block. This means that for a Process Table, there can be a maximum of about $56_{(1Ø)}$ bits of information.

Since the Operating System is scanning the Control Blocks and Process Tables looking for something to do, the assumption that can be made is that a Process or Processes are in a state so that they are never becoming "Ready". The following checks should be made on the Core Dump:

a) Scan the Eligible Queue (ELQUE)

1) Check the Status of the Core Manager.

A) Is he waiting on some specific event such as an overlay load? (PSTAT)

Table 4.6 Process Table Common ELQUE Bits

WORD	MUST BE THERE	MIGHT BE THERE
PSTAT		PSBAG - Swap this process PSRDY - Not Ready to run PSEW - Scheduler Action PSRUN - Running *$PSBRK - Operator Interrupt *PSDP - Daemon Request Up *PSMWT - Wait for Memory Key to Change
PFLAG	PFELG - Process is Eligible (In Core)	ǂPFPRE - Pre-emptive Resident ǂPFSWP - Swapping Process *PFNIN - No Operator Interrupts *PFDEB - Debug Entry *PFTRP - Trap bit *PFINT - Control A flag (^C^A) *PFBRB - Break Request (^C^E) *PFTRM - Terminate Process *PFMBL - Process can only be explicitly Unblocked
PFLG2		øPFBLE - Block enabled PFSUP - Super User PFSHC - Shared Area Changed *PFDUP - Unpend TCB at head of delay chain *PFWTO - Block timeout *PFWSL - Waiting on .SGNL *PFBWT - Wait for timeout *PFATL - Fatal Process Termination *PFASG - Mem Wait bit

Table 4.6 (Con't) Process Table Common ELQUE Bits

WORD	MUST BE THERE	MIGHT BE THERE
PFLG3		PFTSE - At least 1 time slice ended PFPCN - Processing a chain PFCIE - Has created an IPC type entry PFIRS - Interrupt World interrupted task PFCFL - CCB's are Faulted PFCUS - CCB's were referenced in last slice PFUBD - Unblock DAD on return *PFPTM - Processing a Term *PFIWC - Interrupt without User Control *PFSTM - Self-termination occurred *PFDIN - Delayed Operator Interrupt Flag *PFPCH - Hold on > 1 Parallel Calls *PFIDF - IDEF or User Power Fail Done
PFLG4		*PFSLN - Daemon bit for Sync line

Note:

* indicates that these bits may indicate a failure mode condition - they should be investigated further to determine whether they are normal for the situation or a problem.

ǂ if neither PFPRE nor PFSWP are set, the Process is "Resident"

Ø should never be set for a "Resident" Process

$ Following is a Table of the Process termination conditions. PSBRK will always be set:

Condition	Additional Message
^C^B	Interrupt World without User Control
^C^E	Break Request Interrupt World without User Control
^C^A	Control A Flag
Trap	Trap Bit
?TERM	Self Termination occurred

B) Has his stack been pushed? (CSTK = CSTKC)

2) Check the Status of each Control Block.
   A) Is it waiting on some specific event such as an overlay load? (PSTAT)
   B) Which Process Table is it attached to? (CPTAD)
   C) Is it a Daemon?
   D) What is the TCB address (CATCB)?
   E) What is the System Call?

3) Check the status of each Process Table.
   A) Is it waiting on some specific event such as an overlay load? (PSTAT)
   B) Check the Flag words (PFLAG - PFLG4) for anything unusual. An example of this might be XLPT with the "IDEF" bit set - no way is it done. Table 4.6 lists some of the more common bits. An "*" in the table indicates that these bits might indicate a failure mode and need further examination. Not all bits have been covered and other combinations may occur due to timing vs failure, but this Table is a place to start from.
   C) If the Process Table indicates it, it may be useful to examine each TCB on the active TCB chain to determine if it is "pended" for a legitimate reason.

b) Scan the Blocked Queue (BLKQ)

1) Use the same procedure as outlined for each Process Table in Step 3) above for each Process Table on the Blocked Queue. It is the objective of this to make a determination as to whether the Process should actually be there. Table 4.7 lists some of the more common bits that can be found in PSTAT and PFLAG - PFLG4 for Process Tables on the Blocked Queue.

Note that some Process Tables on the BLKQ may still have core assigned to them ("Eligible"). In this case, the TCB's associated with that Process Table should also be checked as in step 3C above.

Table 4.7 Process Table Common BLKQ Bits

WORD	MUST BE THERE	MIGHT BE THERE
PSTAT	PSBLK - Blocked	PSRDB - Not Ready to run *$PSBRK - Operator Interrupt
PFLAG		ǂPFPRE - Pre-emptive Resident PFELG - Process is Eligible (In Core) PFUBL - Request to Unblock Process PFRSH - Reschedule Flag ǂPFSWP - Swapping Task PFEBL - Waiting for Son Termination *PFNIN - No Operator Interrupts *PFDEB - Debug Entry *PFINT - Control A Flag *PFBRB - Break Request *PFMBL - Process can only be explicitly Unblocked
PFLG2	PFBLE - Block Enabled PFQSC - Inhibit Scan of backed up TCB Requests (only if PFLEG not set)	PFDUP - Unpend TCB at head of Delay Chain PFSP - Swap In/Out in Progress PFCMQ - Process in Enqueued to Core Manager PFSWO - Process is being Swapped Out PFSUP - Super User *PFWSL - Waiting on .SGNL *PFATL - Fatal Process Termination
PFLG3		PFTSE - At least on time slice ended PFUBD - Unblock Dad on return PFCIE - Has created an IPC type entry PFIRS - Interrupt World interrupted task PFPCH - Hold on > 1 parallel calls

(con't)

Table 4.7 (Con't) Process Table Common BLKQ Bits

WORD	MUST BE THERE	MIGHT BE THERE
PFLG3 (con't)		*PFIWC - Interrupt without User Control *PFDIN - Delayed Operator Interrupt Flag *PFIDF - IDEF or User Power Fail Done
PFLG4		(none)

Notes:

$ Refer to same note in Table 4.6

ǂ Either PFPRE or PFSWP must be present

* Refer to same note in Table 4.6

c) Scan the Ineligible Resident Queue (IERES)

1) Same as for Step 3 for ELQUE Process Tables.

d) Scan the Ineligible Swapped Queue (IESWP)

1) Same as for Step 3 for the ELQUE Process Tables.

Using the information from either the above scans or the Core Dump Analyzer (refer to Section 4, II), the objective is to find reasons as to why the various Process Tables are not being run.

If a high priority "Resident" process goes "core bound", the system may appear to be hung and the dump might be taken at the time the system is in the scheduler. In this case, there should be Process Tables further down the ELQUE which are "Ready to run" and would make this case apparent.

(3) System Call/Daemon Analysis

This point would indicate that we were "hung" while executing a Daemon or User System Call Control Block. Note that some System Calls such as the ?CHAIN put a "Ø" in CATCB (offset Ø) to fake a Daemon so that the TCB status word "pend" bit is not reset. This is because the Physical Page containing the TCB may have already been released and re-assigned to another process. From past history, I would expect a Panic to occur rather than a hang if a problem occurred while executing a Control Block.

The procedure for troubleshooting this type failure is relatively simple and will be outlined in the following paragraphs even though I suspect it will never be needed. We already know the Control Block address which was used with the flow in Figure 4.12. There are two key questions to which we are looking for answers:

a) Where are we hung and should we be there?

b) Why are we hung there?

The procedure which follows will attempt to gather any significant initial information in order to answer part "a)". The answer to "b)" will depend upon where we were actually hung. In order to do the following analysis quickly and adequately, the AOS listings are needed. In the future, these should be available in the field. Until then, phone communication or core dump submission can be used to fill the void.

Information that we will gather:

1) Use the "PC" that was recorded at the time of core dump to determine the AOS module and subroutine that we were looping in.

2) Record the values of locations 4Ø (SP) and 41(FP). Use these values to "layout" the last couple of stack frames. Figure 6.1.4 might be useful in this endeavor. AC3 in each stack frame should point to the word after the "JSR" in the "Caller" routine. Refer also to Figure 4.21. By comparing the "PC" at the time of hang with the address of the last subroutine that was "JSR'd" into, we might

be able to determine whether we were hung in the "proper" routine or jumped off into the wrong place in core and ended up hung in a routine that we never intended to enter.

3) Examine location 366(CRSEG) to determine whether the Control Block is using an overlay. As mentioned in other sections, CRSEG can have the following values:

   a) OVTAB + Overlay # - Overlay being accessed is disk based. Figure 6.7.7 illustrates the data base linkages.

   b) 1BØ + RDHDR - Overlay being accessed is a core resident overlay. The overlay # is contained in location RDHDR+3. Figure 6.7.8 provides additional information on these data bases.

   We now have the overlay # and by examining the above data bases we can determine the Physical Page the overlay resides in. Table 4.8 lists the various overlays by name and number and gives the first three words for each overlay. If we suspect that the wrong overlay was loaded into core, we can compare the two sets of values. Note that if SLMDC = 177777, it indicates that the SLM was not AOSGEN'd.

At this point, it is "digging" within the routine in which we were "hung" to attempt to determine the reason. If we feel that the wrong overlay were loaded, this problem should be addressed first. I would suspect a memory or map failure before I would the disk sub-system since both the CDC (4231) and the 6060/61 do position address checking down to the sector level.

A technique that I have found to be quite useful in certain cases is the "dump search". This technique applies equally as well to a Panic as to a hang. In many cases, while examining a core dump, a value is found in a data base which is totally wrong and meaningless. I have found that many times this value can be the "key" in solving the problem with the data base. I have also

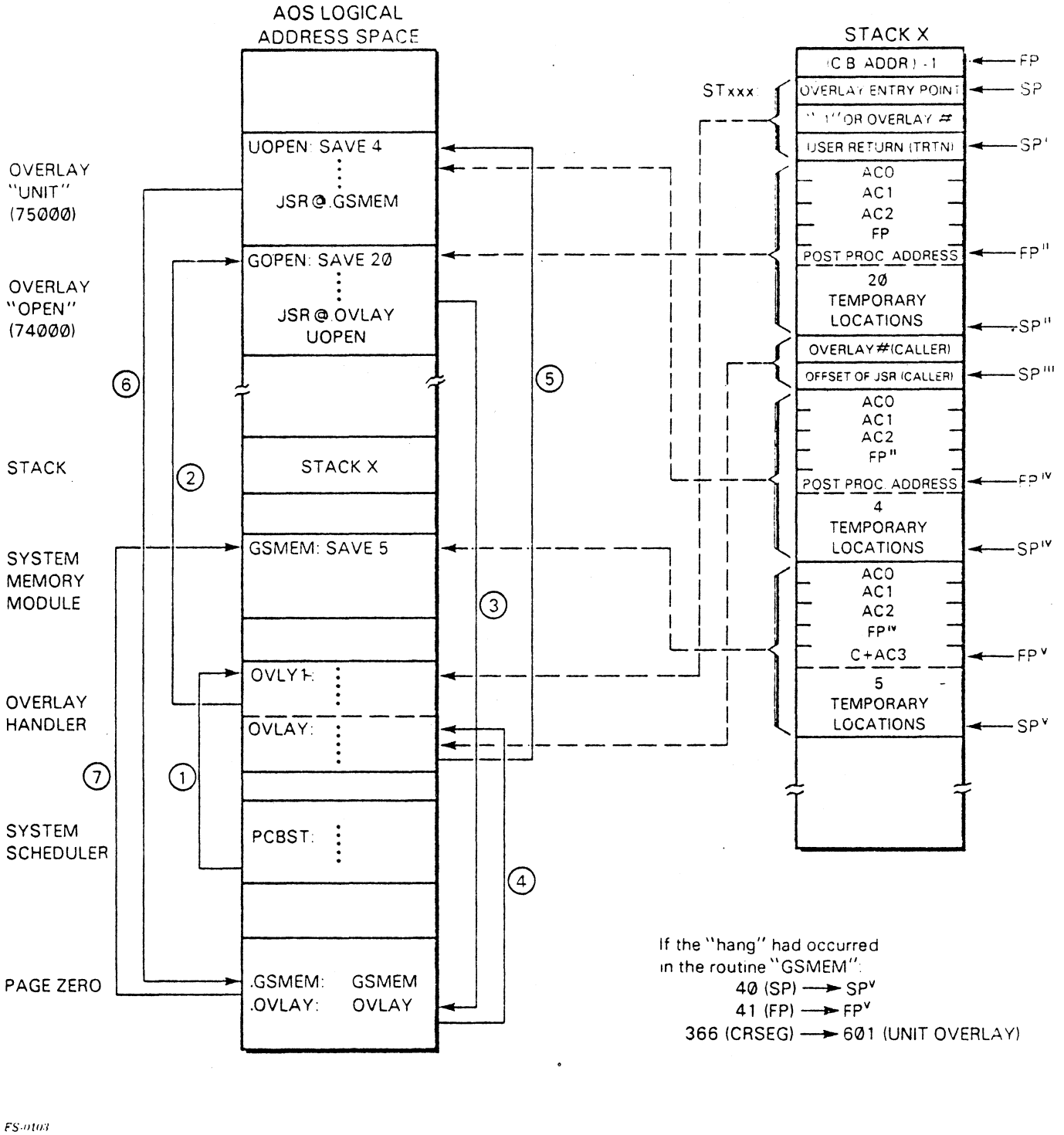
found that by knowing where else in a core dump the value resides, it might be determined which routine was responsible for misplacing the value. Both DEDIT and SYSDMP as described in Section 7.5 and 7.6, respectively, "System Utilities", will search the core dump and the procedures are shown in the illustrations within that section. Be aware of the following tradeoffs between the two utilities:

a) DEDIT
   1) Searches a "file" in 32Kw increments.
   2) Addresses are relative to the beginning of each 32Kw block
   3) Will search a complete core dump fairly quickly
   4) Any "hits" will have to be converted manually to a Physical Page/Offset for use in SYSDMP. Table 4.9 on the following page will facilitate the conversion process.

b) SYSDMP
   1) Searches a "file" using logical addresses within a logical to physical translation table or a 1K page.
   2) Addresses are always logical relative to the translation table or 1K page.
   3) The searching is very slow because of translations and can take on the order of 5-10 min. just for the SMAP area
   4) Will stop searching with the 1st "illegal address" so that map slots above that address are never checked.
   5) Under AOS, the Logical to Physical for window map slots are not included in SMAP, therefore, these must be checked seperately.
   6) Although slow, this will allow any logical system space between Ø and the first unused slot with a maximum of 57777 to be checked for a specific value. If a "hit" occurs, you know that it resides within that address range.

Note that even when indicated, this procedure will not always lead you in the proper direction but I have had a couple of "tough nuts" which this procedure has helped to crack.

As mentioned previously, Figure 4.21 illustrats the "routine" entries versus stack linkage that might be found if a hang occur-



FS-0103

Figure 4.21 Simplified Overlay Call and "SAVE" Stack Relationships

Table 4.8

(Current as of AOS 1.Ø6)

Overlay # w/SLM	Overlay # w/o SLM	Title	Word Ø	Word 1	Word 2
Ø	Ø	SSOV1	2	1Ø247Ø	52Ø
1	1	UNIT	2	1Ø247Ø	615
2	2	CMOV1	16371Ø	Ø	1Ø247Ø
3	3	SCMOD	1Ø247Ø	654	1Ø247Ø
4	4	IOOV	16371Ø	11	514Ø7
5	5	SWAPI	16371Ø	37	514Ø1
6	6	SWAPO	16371Ø	37	514Ø1
7	7	SWAPM	1Ø247Ø	37Ø	32234
1Ø	-	SYNLP	35Ø5Ø	13311Ø	314ØØ
11	-	SYNCH	17711Ø	35Ø46	21Ø4Ø
12	1Ø	RESLV	2Ø	1Ø2ØØØ	41Ø
13	11	RESL2	5	1Ø247Ø	414
14	12	OPEN	1	2ØØ31	414Ø1
15	13	CLOSE	16	1Ø247Ø	447
16	14	DE	15	126ØØØ	4541Ø
17	15	CREAT	22	31771	15324Ø
2Ø	16	DRLSE	24	1Ø2ØØØ	4142Ø
21	17	BOOT	36	1Ø247Ø	7
22	2Ø	PROC	32Ø	1Ø77	3Ø
23	21	PROC2	3Ø	34Ø41	21412
24	22	PROC3	3Ø	3Ø365	31Ø16
25	23	PRCNG	24	1Ø247Ø	3Ø6
26	24	SSOV3	Ø	1Ø247Ø	46Ø
27	25	SSOV4	3	1264ØØ	445
3Ø	26	SSOV5	Ø	10247Ø	613
31	27	DINIT	45	122Ø7Ø	535
32	3Ø	XINIT	45	1Ø247Ø	6Ø2
33	31	NAMES	12	1Ø247Ø	34
34	32	CHAIN	1Ø247Ø	346	21
35	33	SSOV2	Ø	1Ø247Ø	571
36	34	SSOV6	1Ø	1Ø247Ø	215
37	35	DIRST	1	32365	21ØØ6

Table 4.8 (Con't)

Overlay # w/SLM	Overlay # w/o SLM	Title	Word Ø	Word 1	Word 2
4Ø	36	ACLST	14	1Ø247Ø	42
41	37	PNAME	2	1Ø247Ø	631
42	4Ø	SYSER	5Ø	1Ø24ØØ	414Ø3
43	41	NAME2	5	1Ø247Ø	32
44	42	SSOV7	7	1Ø247Ø	744
45	43	SSOV8	11	1Ø247Ø	4Ø3
46	44	SSOV9	11	1Ø247Ø	516
47	45	INFOS	5	1Ø247Ø	7
5Ø	46	IREC	15	6137	12344Ø
51	47	ISEND	25	17777Ø	1
52	5Ø	ISEN2	6265	57	25
53	51	CLNUP	3Ø	1264ØØ	1ØØØ15
54	52	SOV11	3	1Ø247Ø	564
55	53	SOV1Ø	4	1Ø247Ø	7
56	54	SYOV1	1	1Ø247Ø	416
57	55	SYOV2	15	1Ø24ØØ	1Ø152Ø
6Ø	56	SYOV3	7	172Ø7Ø	(address)

red while a Control Block was executing. If the "hang" had occurred in the "GSMEM" subroutine, the values we would find would be:

a) PC = GSMEM + n
b) 4Ø (SP) = SP^V
c) 41 (FP) = FP^V
d) 366 (CRSEG) = 6Ø1 (unit overlay)

The stack at this point would appear as illustrated for "STACK X". The following is a simplified discussion of the System Call flow, once the TCB has been attached to the Process Table Extension (PSWD), which would set up the illustrated stack blocks. Much more happens than I will discuss. My purpose is to make you familiar with the overall flow.

The "action" begins in the System Scheduler where a TCB that has a System Call outstanding, in this case the ?GOPEN, has been found on the PSWD TCB chain (refer to Figure 5.3, if necessary):

1. The System Scheduler determines which TCB with an outstanding System Call is to be "run". A Control Block is allocated and linked on the ELQUE. A stack is also acquired. It is also determined whether the System Call needs an overlay. In this case it does and the "Overlay Handler" is entered.

2. The Overlay Handler will push on the stack the:
   1 Overlay Entry Point - in this case "1Ø5135" for GOPEN
   2) a "-1" since this is the initial concurrent overlay being used by this Control Block
   3) the return address upon completion which is "TRTN"

   In addition the overlay is loaded, if necessary, and a SAVE block is pushed on the stack with the number of "temporaries" needed by the overlay. GOPEN uses $2Ø_8$ temporaries.

3. The overlay, OPEN, will be entered at the entry point, GOPEN, and its code will execute. It may use subroutines which will do "SAVE's" and "RETURN's" but at the point it needs to call another overlay, UNIT, no additional frames have been PUSH'd on the stack.

4. Because the overlay "UNIT" is needed, a "JSR" through a Page

Table 4.9 DEDIT/SYSDMP Address Conversion Table
(all numbers are octal)

1K Segment	#H value yielding physical page								Logical Address Range
	Ø	1	2	3	4	5	6	7	
Ø	Ø	4Ø	1ØØ	14Ø	2ØØ	24Ø	3ØØ	34Ø	Ø - 1777
1	1	41	1Ø1	141	2Ø1	241	3Ø1	341	2ØØØ - 3777
2	2	42	1Ø2	142	2Ø2	242	3Ø2	342	4ØØØ - 5777
3	3	43	1Ø3	143	2Ø3	243	3Ø3	343	6ØØØ - 7777
4	4	44	1Ø4	144	2Ø4	244	3Ø4	344	1ØØØØ - 11777
5	5	45	1Ø5	145	2Ø5	245	3Ø5	345	12ØØØ - 13777
6	6	46	1Ø6	146	2Ø6	246	3Ø6	346	14ØØØ - 15777
7	7	47	1Ø7	147	2Ø7	247	3Ø7	347	16ØØØ - 17777
1Ø	1Ø	5Ø	11Ø	15Ø	21Ø	25Ø	31Ø	35Ø	2ØØØØ - 21777
11	11	51	111	151	211	251	311	351	22ØØØ - 23777
12	12	52	112	152	212	252	312	352	24ØØØ - 25777
13	13	53	113	153	213	253	313	353	26ØØØ - 27777
14	14	54	114	154	214	254	314	354	3ØØØØ - 31777
15	15	55	115	155	215	255	415	355	32ØØØ - 33777
16	16	56	116	156	216	256	316	356	34ØØØ - 35777
17	17	57	117	157	217	257	317	357	36ØØØ - 37777
2Ø	2Ø	6Ø	12Ø	16Ø	22Ø	26Ø	32Ø	36Ø	4ØØØØ - 41777
21	21	61	121	161	221	261	321	361	42ØØØ - 43777
22	22	62	122	162	222	262	322	362	44ØØØ - 45777
23	23	63	123	163	223	263	323	363	46ØØØ - 47777
24	24	64	124	164	224	264	324	364	5ØØØØ - 51777
25	25	65	125	165	225	265	325	365	52ØØØ - 53777
26	26	66	126	166	226	266	326	366	54ØØØ - 55777
27	27	67	127	167	227	267	327	367	56ØØØ - 57777
3Ø	3Ø	7Ø	13Ø	17Ø	23Ø	27Ø	33Ø	37Ø	6ØØØØ - 61777
31	31	71	131	171	231	271	331	371	62ØØØ - 63777
32	32	72	132	172	232	272	332	372	64ØØØ - 65777
33	33	73	133	173	233	273	333	373	66ØØØ - 67777
34	34	74	134	174	234	274	334	374	7ØØØØ - 71777
35	35	75	135	175	235	275	335	375	72ØØØ - 73777
36	36	76	136	176	236	276	336	376	74ØØØ - 75777
37	37	77	137	177	237	277	337	377	76ØØØ - 77777

Zero variable, .OVLAY, to the Overlay Handler will be done. The Overlay Handler will now push on the stack:

1) Overlay # of the overlay we had just "JSR'd" out of.
2) Offset of the "JSR" within the overlay. If AC3 when we "JSR'd" pointed to 74735, 736 will be stored. This bypasses the non-executable overlay entry point, UOPEN.

These values are needed since the overlay is released and may not be in the same slot section when we are ready to re-enter it. The overlay is loaded, if necessary, and a SAVE block is pushed on the stack with the number of "temporaries" needed by the overlay. UOPEN uses 4_8 temporaries.

5. The overlay, UNIT, will be entered at the entry point, UOPEN, and its code will execute. In this case, a core resident subroutine is needed, GSMEM.

6. A "JSR" is done through a Page Zero variable, .GSMEM, into the subroutine.

7. The subroutine GSMEM will do a "SAVE 5" which pushes a Return Block and 5 temporaries on the stack. If we were to "hang" in GSMEM, that is the way the stack would look at this point.

The above was just a general flow. If you can understand the general linkages in the nested direction, it is easy to work back up when a problem occurs at a "nested" level.

(4) Process Table Analysis

This point would indicate that we were "hung" in the scheduler, or a Scheduler called subrcutine, while setting up but prior to executing:

a) the User Process

b) a Control Block for an outstanding System Call for that User Process

Figure 4.22 illustrats a block diagram of this sequence or flow with correlation to the "Numbers" used in Figure 4.12. Note that this is an intermediate step between looping in the Scheduler waiting for something to do and executing a Control Block for a process or the Process itself.

From past history, I have never seen a hang while "scheduling a Process". Because of this and the fact that the total Scheduler code is only approximately 165Ø words in length, I will not discuss this type problem any more. If we find in the future, that we are "hanging" in this area, I will enlarge this section. Note that even though the Scheduler is approx. 165Ø words in length, only a subset of those words are used during "scheduling a Process".

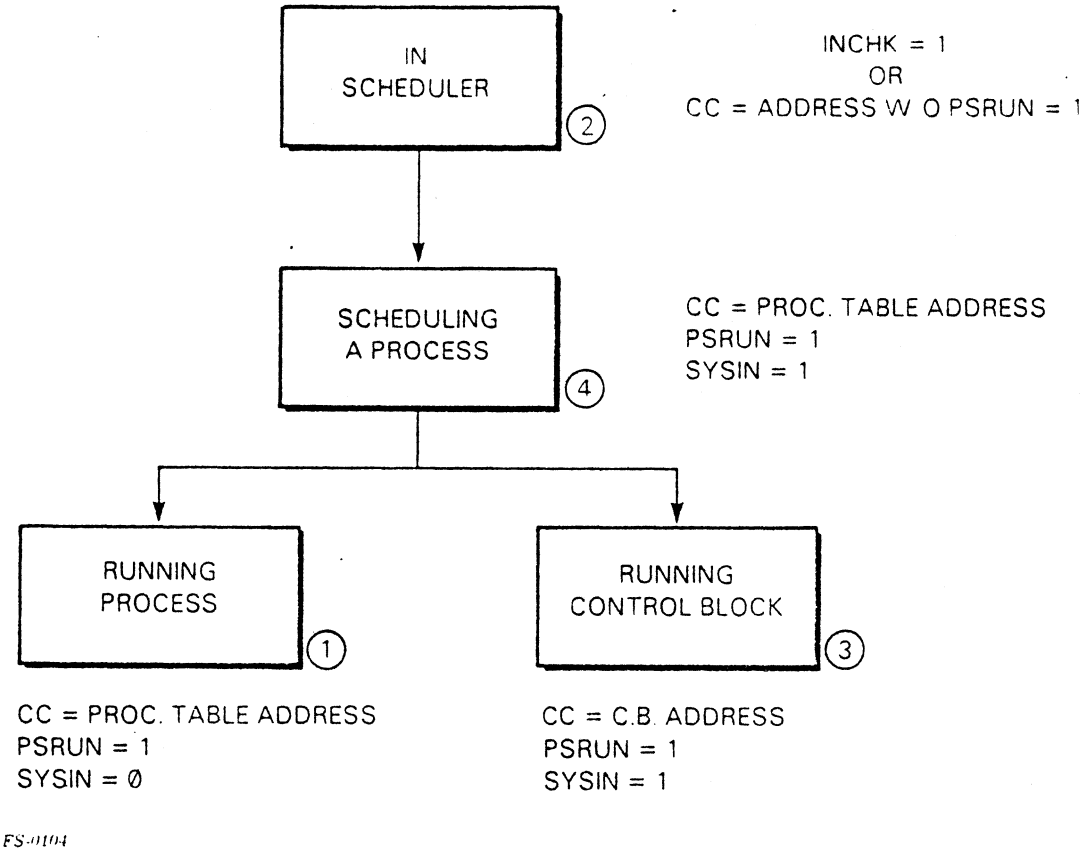


Figure 4.22 "Scheduling a Process" Block Diagram

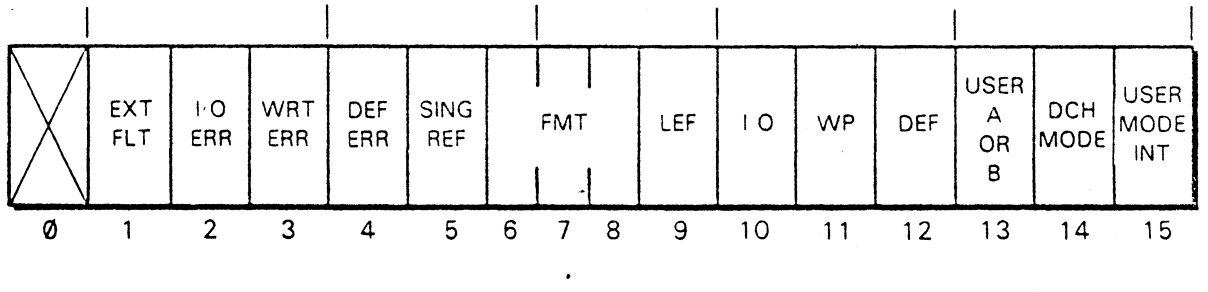


Figure 4.23 C33Ø Map Status (DIA) Breakdown

(5) Interrupt Analysis

This point would indicate that the core dump was taken while the system was servicing an interrupt. It does not immediately follow that the system was "hung" at interrupt level because there is a second case. The two cases are:

1) The system was actually "hung" at interrupt level.
2) The system was "hung" (or not responding) somewhere else but we caught the system servicing an interrupt when the dump was taken.

The following analysis will attempt to differentiate between the two cases and then determine the next course of action.

We automatically know that INTLV ≠ -1 since this came out of our flow using Figure 4.12, "Where Was I?". The initial common locations will be listed below in the order they should be checked. At some point, you will have to branch out further on your own, depending on the circumstances, since I will never be able to cover all the wierd cases that can occur. To begin:

a) INTLV

We know that this value is not -1, but we should also know how many levels of nesting deep we are. This location is "ISZ'd" for each level deep, ie: INTLV = 3 is four nested interrupts.

b) SYSIN

The contents of this location will tell us where we were, System or User, prior to the 1st interrupt. A "Ø" indicates User, a "1" indicates System.

c) Location 5 (CMSK)

The contents of this location will be the current interrupt mask. It is put here by the Vector instruction. At the 1st level, it will be the actual device mask. At nested levels, it will be the "OR" of the old contents of location 5 and the actual device mask. Table 6.7.1 lists the various device masks used by AOS.

d) Location Ø

The contents of this location will be the PC at the time of last interrupt. If there are nested levels of interrupts, this will be the PC of the deepest nested level.

e) Location 4

The contents of this location will be the address of the 1st word of the Interrupt Stack, SS. This location is moved to location 4Ø by the Vector instruction when a "stack change" is indicated.

f) Location 4Ø (SP)

The contents of this location should point into the Interrupt Stack at the last location that has been used. The layout of the Interrupt Stack is shown in Figure 6.1.5 and will differ slightly depending on whether the interrupt was caused by a System or User Defined (?IDEF) device.

g) Location 41 (FP)

The contents of this location should point to the last "Return Block" word within a Stack Frame. Figures 6.1.4 and 4.21 illustrate the relationship of the SP and FP when using "SAVE" instructions. The FP will usually appear as "Ø" until the 1st "SAVE" is done but the Programmer Reference states the contents as "unpredictable" after the Vector instruction with stack change is executed.

At this point, the stack should be "layed out" using Figure 6.1.5 with interrupt levels and SAVE blocks being marked off. Use the "OFP" within the "SAVE" blocks to mark off the "SAVE" stack frames. This is important because individual words may be "PUSH'd" on the stack between the "SAVE" frames. Once this is done, other values on the stack can be examined for any relevant values. Specific values will be discussed below:

h) C+PC in the Interrupt Return Block

This location will indicate where within the User Process or Operating System we were prior to the interrupt. Since Loc Ø can be over-written by nested interrupts, this is a good value to keep in mind.

i) Old Mask

This will be the System Interrupt mask that was in effect prior to this interrupt happening. This can be useful in determining the devices within nested interrupt levels. Again Table 6.7.1 can be used to determine the device. Note that this can be an "OR" of multiple devices at nested interrupt levels. Refer to the Eclipse Programmer's Reference Manual (Ø15-24) on the Vector Instruction (Page 4-7) for additional information, if needed.

j) Old Location 2

When no interrupts are being serviced, location 2 points to "SYST" which handles System Calls from User Space via the SVC, SCL instructions. At interrupt level location 2 is set to the address of "DISMIS" so that at the end of an interrupt service the dismiss routine can be entered quickly and easily via an "SVC" instruction. The old value is stored on the interrupt stack so that it can be restored by the dismiss code.

k) Old Map DIA

This is the map DIA which is taken immediately after the interrupt occurred. It is used to properly restore the User/System state after the interrupt is completed. Figure 4.23 illustrates the breakdown of this word for a C33Ø (MMPU1) which might provide useful information.

l) CUR6Ø - CUR74

These are the current logical to physical translations for their respective slots and are stored so they can be properly restored at the end of the interrupt. Figure 6.7.1. "AOS Logical Address Space Layout", graphically illustrates the usage of those slots.

m) Old CRSEG

The contents of CRSEG, which contains information about overlay usage, is "PUSH'd" on the stack so that it can later be restored. The possible contents of CRSEG are defined on Page 6-64 in the section dealing with AOS Panic 1Ø's.

The main reason for "laying out" the Interrupt Stack is to determine where we were when the system was stopped to take the core dump and whether we should legitimately be there. The information copied down by the operator when the system was stopped should also be taken into account. Since each case is different, for one reason or another, and it is hard to generalize past this point, following are a couple examples which use the previously mentioned locations. These should help you to think through a couple of actual core dumps and formulate your own procedures to troubleshoot this type problem.

Example #1

Figure 4.24 illustrates the SYSDMP output from the analysis of a core dump which was "hung" at Interrupt Level. Initially, we can make the assumption that the hang at interrupt level is valid until we can prove differently. The customer description of the problem was "slow die situation, consoles would not restart after log-off, EXEC no longer processed messages".

We noted that INTLV ≠ -1 during our analysis using Figure 4.12. I would also note the value of SYSIN, since this will tell us where we were prior to the interrupt. In our case, we were at the 1st level of Interrupt and were in the User prior to the interrupt. By checking the Current Mask which is contained in Phys location 5 and using Table 6.7.1, we can determine that the interrupt was from the Programable Interval Timer (PIT). Location Ø should be the PC at the time of interrupt which is "Ø" (keep that in mind). Examining location 4, we find that the Interrupt Stack, SS, begins at 156Ø. The current Stack Pointer (SP) and Frame Pointer (FP) are 16Ø2 and Ø, respectively. An FP of "Ø" is O.K. if no "SAVE's" have been done since it was "initialized" to "Ø".

Now we can "layout" the interrupt stack using the diagrams in Figure 6.1.5. We can note the old stack values. The "Return Block" is next. Note that the "C+PC" at the time of interrupt is "Ø" (agrees with the contents of loc Ø). The SP is pointing to location SS+22. The next frame contains a "SAVE" block but

```
*) X SYSDMP/S=ADL1.03.S1.ST 1018771100.S1.CD
AOS SYSTEM DUMP ANALYZER REV 1.1
+ INTLV:000000 +          lst level of Interrupt
+ SYSIN:000000 +          In User at time of Interrupt
+ 5:001000 +              Current Interrupt Mask == PIT
+
+ 0:000000 +              PC at time of Interrupt
+ 4:001560 +              Beginning of Intr Stack (SS)
+
+ 40:001602 +             SP
+ 41:000000 +             FP
+ 42:001776 +             SL
+ 43:013450 +             SO
+
+ 1560:000000 +
SS+1  :031645 +           SP
SS+2  :031645 +           FP
SS+3  :032011 +           SL
SS+4  :013450 +           SO
SS+5  :000000 +           ACØ
SS+6  :000417 +           AC1
SS+7  :000000 +           AC2
SS+10 :014124 +           AC3
SS+11 :000000 +           C + PC
SS+12 :000000 +           Old Mask
SS+13 :010501 +           Old Loc 2
SS+14 :000173 +           Old Map DIA
SS+15 :060205 +           CUR6Ø
SS+16 :062114 +           CUR62
SS+17 :164172 +           CUR64
SS+20 :066102 +           CUR66
SS+21 :174213 +           CUR74
SS+22 :000000 +           Old CRSEG
SS+23 :076062 +           ACØ
SS+24 :000000 +           AC1
SS+25 :000761 +           AC2
SS+26 :000000 +           OFP
SS+27 :114241 +           AC3
SS+30 :060205 +
SS+31 :002205 +
+
+ 14274:012222 +
IPIT+1 :103510 +
+
+ MODE N
+
+ 14274:ISZ @.RESC 0  +
IPIT+1 :SUC 0 0  +
+
+ MODE F
+
+ 365:041200 +             Current Process Table Address
+ 41200+42:000003 +       Process ID (PID)
+ 41200+22:000025 +       # Primary Blocks
+
+ BYE
```

Values Copied by Operator

```
ACØ = 661Ø2  CUR66
AC1 = Ø1Ø24  ERCDC
AC2 = Ø1Ø15  PITDC
AC3 =164172  CUR64
 PC = 14275
User, ION = 1
Carry = Ø
```

Note "JMP Ø"

SP

SAVE Ø

JSR @.MLB
@ IRTC+513

Figure 4.2Ø Example #1 Interrupt Level "Hang"

this was done from RTC code at IRTC+513. Since this interrupt was from the PIT, I would suspect that that "SAVE" block was residual, and it is.

The values copied down by the operator when the dump was taken are shown in the upper right hand corner of Figure 4.24. Examining the PC of 14275, we find a "1Ø351Ø" which using Mode N is an SVC Ø, Ø. Note that the interrupt service routine for the PIT only consists of the two instructions shown in the figure. The SVC automatically directs code execution to the DISMIS routine, the PC found in Loc Ø would not correspond to the PC found in the lst level Interrupt "Retun Block" (excluding nested interrupts) and that the PC in location Ø would probably be close to the PC when the Operator stopped the systemv The PC in location Ø probably would have been updated since we would have taken another interrupt, especially with a mask of "1ØØØ", and should point into the area where we are "hung".

The conclusions that can be drawn are:

1) There was no hang while at Interrupt level.
2) A "User" Process had done a "JMPØ" which may have cauoed the customers symptoms.

The question is to try to determine the "User" Process which had a PC of "Ø". Location 365 (CC) should contain the Process Table address of the last process that was running when the interrupt occurred. I can make the assumption that it was a Process Table since "SYSIN = Ø" (would be "1" for a C.B.). Offset 42 of the Process Table is the PID which in our case is "3". This is usually the EXEC and would explain why the symptoms were "slow die as consoles were logged off" - the EXEC had done a "JMPØ".

You would now have to examine the EXEC as a User Process (discussed in sub-section 1) to determine why the EXEC did the JMPØ.

Example #2

Figure 4.25 illustrates the SYSDMP output from the analysis of a core dump which was "hung" at interrupt level. Initially, we can make the assumption that the hang at interrupt level is valid until we can prove differently. The customer description

```
*) X SYSDMP/S=ADL1.03.S1.ST 1004771630.S1.CD
.AOS SYSTEM DUMP ANALYZER REV 1.1
+ INTLV:000000 +          1st level of Interrupt
+ SYSIN:000001 +          In System at time of Interrupt
+ 5:000717 +              Current Interrupt Mask == MHD
+
+ 0:004151 +              PC at time of Interrupt
+ 4:001560 +              Beginning of Intr Stack (SS)
+
+ 40:001610 +             SP
+ 41:001610 +             FP
+ 42:001776 +             SL
+ 43:013450 +             SO
+
+ 1560:000000 +
+ 1561:031276 +           SP
SS+2 :031276 +           FP
SS+3 :031616 +           SL
SS+4 :013450 +           SO
SS+5 :177461 +           ACØ
SS+6 :020313 +           AC1
SS+7 :000257 +           AC2
SS+10 :000306 +          AC3
SS+11 :102054 +          C + PC
SS+12 :000000 +          Old Mask
SS+13 :010501 +          Old Loc 2
SS+14 :000437 +          Old Map DIA
SS+15 :060165 +          CUR6Ø
SS+16 :062047 +          CUR62
SS+17 :064174 +          CUR64
SS+20 :066024 +          CUR66
SS+21 :174101 +          CUR74
SS+22 :000000 +          Old CRSEG
SS+23 :042100 +          MHD DIA status
SS+24 :000000 +          ACØ
SS+25 :002100 +          AC1
SS+26 :032241 +          AC2
SS+27 :000000 +          OFP
SS+30 :103765 +          AC3          <- SP, FP
SS+31 :000005 +          ACØ
SS+32 :002100 +          AC1
SS+33 :032241 +          AC2
SS+34 :041140 +          AC3
SS+35 :104151 +          C + PC
SS+36 :000717 +          Old Mask
SS+37 :014440 +          Old Loc 2
SS+40 :000037 +          Old Map DIA
SS+41 :060165 +          CUR6Ø
SS+42 :062047 +          CUR62
SS+43 :064174 +          CUR64
SS+44 :066024 +          CUR66
SS+45 :174101 +          CUR74
SS+46 :000000 +          Old CRSEG
SS+47 :032313 +
+
+ 4151:000400 +          Location @last Interrupt
DPAST+20 :000000 +               ("JMP .")
+
+ BYE
```

Values Copied by Operator

ACØ = ØØØØ5
AC1 = Ø21ØØ
AC2 = 32241
AC3 = 4114Ø
PC = Ø4151
User, ION = 1
Carry = not copied

1st level

2nd level

Figure 4.25 Example #2 "Interrupt Level "Hang"

of the problem was "consoles would not echo."

We noted the INTLV ≠ -1 during our analysis using Figure 4.12. I would also note the value of SYSIN, since this will tell us where we were prior to the interrupt. In our case, we were at the 1st level of interrupt and we in the System (actually the Scheduler) prior to the interrupt. By checking the Current Mask which is contained in Physical location 5 and using Table 6.7.1 for a Pre-Rev 1.Ø5 Operating System, we can determine that the interrupt was from the Moving Head Disk (MHD). Location Ø is the PC at the time of interrupt which is "4151". Note that this does not agree with the 1st level "Return Block" PC and is actually the 2nd level "Return Block" PC. Examining location 4, we find that the Interrupt Stack, SS, begins at 156Ø. The current Stack Pointer (SP) and Frame Pointer (FP) are 161Ø and 161Ø respectively.

Now we can "layout" the interrupt stack using the diagrams in Figure 6.1.5. We can note the old stack values. The "Return Block" is next. Note that the "C+PC" at the time of interrupt is "1Ø2Ø54". Since SMON1 begins at 2ØØ1 and INCHK = 1, we were in the System Scheduler Checksum loop prior to the interrupt. After the "old CRSEG" location, the Moving Head Disk Interrupt Service routine "PUSH'd" the disk DIA on the stack. Using Table 6.2.1 and since the disk is a CDC (333Ø), we note that there are no errors and that the interrupt was due to a "SEEK DONE" on Unit Ø.

The values copied down by the operator when the dump was taken are shown in the upper right hand corner of Figure 4.25. We can note the coincidentally:

1) The PC at the point that the operator stopped the system to take a dump was "4151".
2) Location Ø which is the PC at the time of the last interrupt was "4151".

At this point, I examined location 4151 and found the value to be a "JMP." (4ØØ). The value 4151 equates to "DPAST+2Ø" which is in the Moving Head Disk Startup routine.

The conclusions that can be drawn are:

1) We were actually "hung" at Interrupt Level.
2) The cause of the hang was a "JMP." at DPAST+2Ø in the Moving Head Disk Startup routine.

In reality, there was a "race" condition, when multiple disk drives on the same controller were in use, with the disk "time out handler" which wuld cause an "illegal" disk state to be entered and the "JMP.". Both the problems of being able to enter an illegal disk state and the "JMP." being in the code have been fixed in Update 1.Ø6.

SECTION 5

SYSTEM TRAPS

A "trap message" is the operating system response to a non-fatal hardware failure such as a map violation or a non-fatal software failure such as an invalid Task Control Block (TCB) address. A non-fatal error may cause the <u>process</u> to abort, but the operating system will still be functioning. A fatal error will cause the <u>operating system</u> to abort, ie: Panic. Note that the "non-fatal software failure" may have been caused by a hardware failure which went undetected by any existing hardware check circuitry.

The "trap messages" will be broken down into the following categories for easier discussion:

The last category, section 5.5, will document a couple of the more obscure cases which will probably never occur but should be listed for completeness. All "Traps" will have the following general format:

```
*ABORT*
Group-reason, Specific-reason
Cx, PCxxx, ACØx, AC1x, AC2x, AC3x
ERROR:  FROM PROGRAM
```

The "Group-reason" and "Specific-reason" are filled in with certain text strings. The values of these text strings in addition to whether a break file was created will determine the sub-section of this chapter to use. Refer to Table 5.1 which outlines the possibilities.

"GROUP-REASON"	"SPECIFIC-REASON"	BREAKFILE CREATED ?	SECTION
USER TRAP	(Normally None)	NO	5.1
USER TRAP	WRITE VALIDITY I/O ACCESS DEFER	YES	5.2
USER TRAP	DATA BASE	YES	5.3
GHOST TRAP	(Any)	YES	5.4
USER TRAP (Obscure - Use Section 5.2 First)	(None) - C300 VALIDITY - C330	YES	5.5

FS-0075

Table 5.1 "Trap" Differentiating Table

5.1 "?RETURN" System Call Trap

The "?RETURN" System Call is documented in the AOS Programmers Manual (Ø93-12Ø) on Page 2-18 for a Revision 1 manual. It is used to "Terminate a Process and Return a Message" to the father process.

This system call can be used both for a "good" return and for an "error" return. The contents of the accumulators being passed back indicate the type of return. If a "good" return is being taken and AC2 is not set properly, the system will attempt to interpret AC2 as best that it can using the following format:

?RETURN FLAGS	TERM FIELD	PROCESS ID
Ø 4	5 7	8 15

Term Field (octal value)	Meaning
Ø	No message
1	"User Trap"
2	"Console Interrupt"
3	"Superior Process Terminated"
4	"Terminated by AOS - Unknown Message Code"
5	(5)
6	(6)
7	(7)

Note that "Field Values" of 5, 6, or 7 only return their numeric equivalents. If a combination of bits in AC2 contain a Term Field octal value of "1", the father process will think that a son process had "trapped" when in fact, no trap had occurred. If the father process happens to be the CLI, a "Trap" message will be printed on the console such as:

```
*ABORT*
USER TRAP
CØ, PCØ, ACØ 1, AC1 Ø, AC2 Ø, AC3 Ø
ERROR:  FROM PROGRAM
X,RETST
```

even though no actual trap occurred.

The contents of the message that is typed on the console will vary depending on the random values of other pointers. Normally the message will be close to the above. A key is that no break file is created because the "trap" actually did not occur.

5.2 Hardware Trap

A "hardware trap message" is the software response to a hardware map violation. On the Eclipse series machines, a hardware map violation can be caused by either:

1. Validity Violation - an attempt to access outside of the user's given logical address space (sometimes referred to as "context").
2. Write Protect Violation - an attempt to write into an area of core that is write protected. Core is write protected on a page basis (2K octal words).
3. Defer Violation - The program has attempted to do nested "indirects" more than sixteen levels deep.
4. I/O Violation - an attempt has been made to issue a programmed I/O instruction to a device code which is protected by the map.

Note that "Processes" which run under AOS will execute with LEF mode on. The implication of this is that if a process inadvertently issues an I/O instruction such as a "halt" (63Ø77) without first turning LEF mode off, the "Halt" will be interpreted by the map as an LEF instruction. A trap will only occur if the effective address calculated is outside of the logical address space, and then as a "validity" violation not as an "I/O" violation.

When a map violation is detected, the hardware disables the map mode and executes a "JMP@3". The software then moves the PC and the accumulators which were stored on the stack at the time of trap into the appropriate storage places. The ultimate software action will depend on "who" was executing when the trap occurred.

The following possibilities exist:

Process Executing	Resulting Action
1. The Operating System running mapped.	1. Panic 7
2. The Peripheral Manager handling byte device I/O.	2. Panic 12
3. A non-recoverable AOS Process such as the Operator CLI.	3. Panic 1Ø
4. A User Process.	4. Error message typed on the respective console.

The various "Panics" are discussed in detail in Section 6 which discussed Panics. If a User Process was in control when a map violation occurs, a message similar to the following will by typed on the respective console:

```
    *ABORT*
    USER TRAP, VALIDITY
    CØ, PC 453, ACØ 155555, AC1 144444, AC2 66666, AC3 77777
    ERROR:  FROM PROGRAM
    XEQ, TEST15
    *)
```

or:

```
    *ABORT*
    USER TRAP, I/O ACCESS
    CØ, PC 456, ACØ Ø, AC1 44444, AC2 55555, AC3 66666
    ERROR:  FROM PROGRAM
    XEQ, TEST16
    *)
```

Note that in the above "Trap", LEF mode had been turned off prior to the trap occurring!! In addition to the above message, a file is created on the disc called a "break save" file which contains a core image of the logical User Space at the time the map violation occurred. (This is the counterpart to the file created under RDOS called "BREAK.SV" OR "FBREAK.SV"). This file will have a file name similar to:

```
    ?Ø15.17_33_19.BRK
```

or:

```
    ?ØØ4.Ø9_54_42.BRK
```

The file name has the following format with each element having a particular significance

```
    ?PID.HOUR_MINUTE_SECOND.BRK
```

where PID is the Process Identification number. The file can either be accessed dynamically using a system utility called "DEDIT" with a sequence such as:

```
)XEQ DEDIT ?ØØ4.Ø9_54_42.BRK
AOS FILE EDITOR REV 1.Ø
+34426:  1Ø2111 + ↓
+34427:  14ØØØØ + ↓
    .
    .
    .
    .
+BYE↓
)
```

or it can be accessed by printing the file on the line printer in a format similar to the RDOS "FPRINT" using the following CLI command:

```
)XEQ DISPLAY/L=@device  ?ØØ4.Ø9_54_42.BRK
```

Where device is:

LPA - standard Line Printer, primary device code

LPA1- standard Line Printer, secondary device code

If the system has a Data Channel line printer, LPB or LPB1, either the EXEC can be initiated via the UP macro and the LPT/LPT1 used or the "/L" switch on the DISPLAY command can be changed to "/L=filename" and then the file printed after the DISPLAY is finished via:

```
 )SUPER ON
*)PRTYPE/R 2   (assumes the console is OP CLI)
*)TYPE/L=@device filename
*)PRTYPE/S 2
*)SUPER OFF
```

A process that accesses the Data Channel line printer must be "Resident" or the following error will occur:

"ILLEGAL PROCESS TYPE, FILE:LPB"

The DISPLAY process cannot be made Resident via the EXECUTE (X) CLI command, therefore the above method is needed.

The printout can now be examined while the system is being used to run other AOS jobs or diagnostics.

We have looked at the trap mechanism from a hardware map standpoint, now let's examine it from a possible cause standpoint. To adequately troubleshoot a "trap" we must take into account the following four possibilities:

1. a hardware failure occurred and the address in the "PC" is one greater than the instruction actually causing the trap.
2. a hardware failure occurred but did not immediately cause a trap, therefore the "PC" points to the instruction or the instruction+1 address that caused the trap. It does not point to the area where the actual hardware failure occurred.
3. a software problem caused the trap and the "PC" points to the area where the actual "mistake" occurred.
4. a software problem caused the trap but the system code execution ping-pong'd around prior to sensing a trap. The "PC" points to the area where the actual trap occurred but not to the area where the initial "mistake" occurred.

There are no real rules that can be given to differentiate between cases 1 and 2, and cases 3 and 4. You should assume that your problem is either a case 1 or case 3 until you either run out of possibilities or find evidence pointing to case 2 or 4. My only objective in mentioning the other cases is to make you aware that they can exist and should always be kept in the back of your mind but do not jump on them as an explanation too quickly, at least not prior to exhausting other possibilities.

Be aware also that the above cases become more complex on an interleaved machine than on a non-interleaved machine. On a noninterleaved machine there is a margin for error since each module is 8K, or 16K words, or 32K words. On an interleaved machine, since instructions are put in sequential modules up to 8way interleaving, you almost have to know the instruction that failed. If you are off by one instruction either way, you are in a different module. If it appears to be a hardware problem, troubleshoot it on an un-interleaved machine.

5.3 Software Trap

A "software trap message" is the software response to an illegal address usually in a Task Control Block (TCB). A brief summary of the reasons for the software checks failing are:

1. Delay Chain - When a task within a process issues a "?DELAY" system call, which delays the task for x number of milliseconds, the task's TCB is placed on a delay chain which is linked from the Process Table Extension. Refer to the figure 5.1. The Process Table resides in System Space and the TCB's are accessed using the window mapping facility with addresses 76ØØØ-77777.

If we are:

a) putting a TCB on the delay chain, the number of existing TCB's can not be greater than $32_{1Ø}$ and the TCB address must be between 76423 and 77623.

b) removing a TCB from the delay chain because it has timed out, the delay chain must be re-linked around the TCB which was removed. The TCB link address must be of the format x76xxx, x77xxx, or 177777.

c) searching the delay chain to remove a TCB because it has been "aborted", all the TCB addresses linked on the chain must be between 76423 and 77623 with the exception of the last link which is a "177777".

If any one of the above conditions is not true, an error in the form of a trap is caused.

2. System Call - The task which issued the system call has its TCB address (ie: 423,612, etc) passed to the "Call Processor". Within the Call Processor this address is verified by adding 76ØØØ to the TCB address and checking for either Carry or bit Ø being set to a "1". If either is set, the TCB address is invalid. The TCB address is used by the system to determine the type of system call and the accumulator values when the call was made.

3. Ghost - As mentioned before, either the User or Ghost can issue system calls. A check is made to see whether the system call was issued by the User or Ghost. If it wa the User, a second check is made to verify that the "In Ghost Context Bit" in the TCB is reset (bit4=Ø of ?TSTAT). If it is not, an error (ie: trap) is caused.

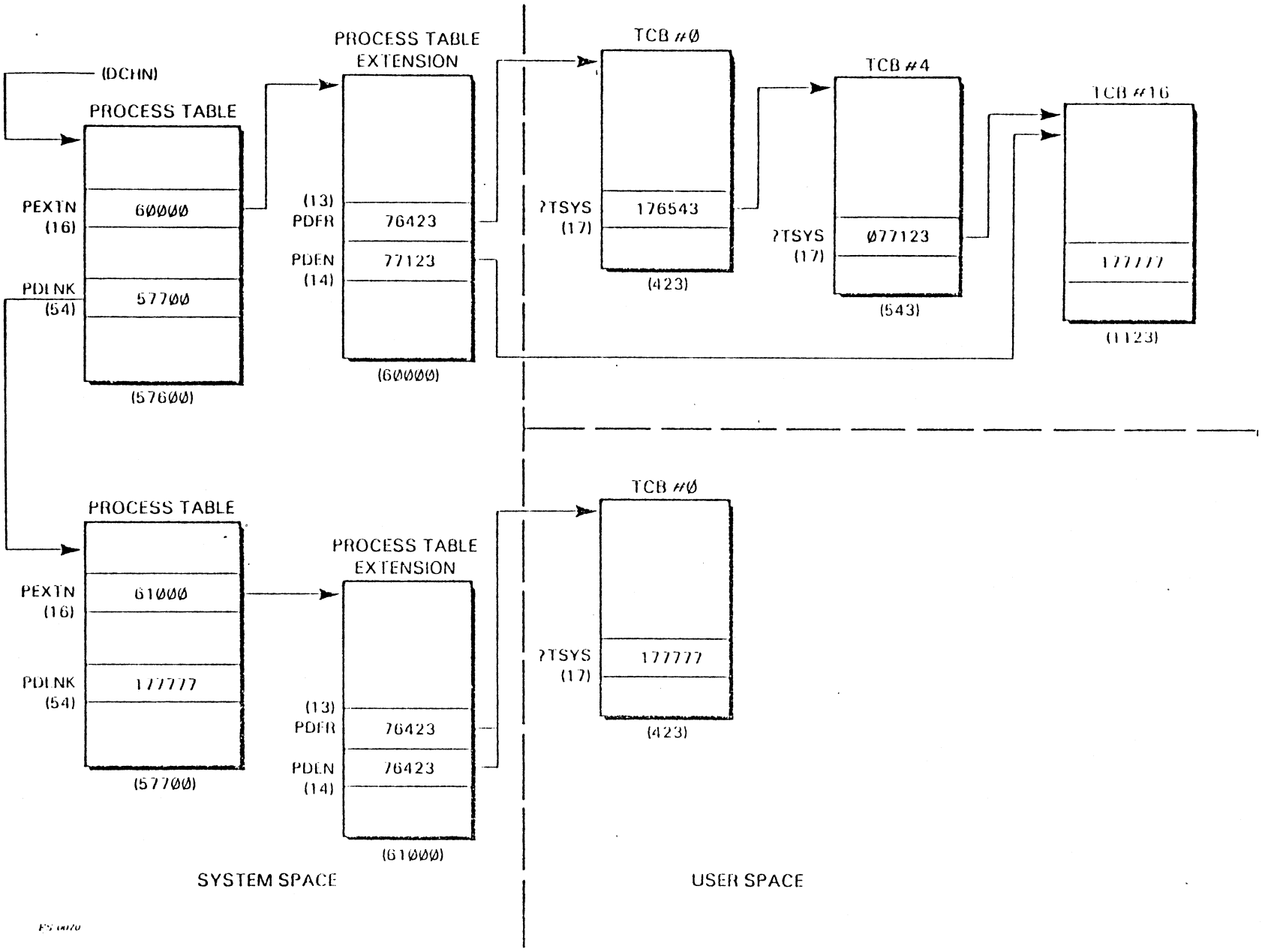


Figure 5.1 - Example of Task Control Block (TCB) Delay Chain Linkage

4. TCB Scheduling - User Tasks (TCB's) are executed by priority level and on a "round-robin" basis within a priority level. In order for a task to be executed, various searches and their respective checks are made (refer to figure 5.2).

a) Search the active TCB chain pointed to by USTAC to find the highest priority task ready to execute. During this search two checks are made:
   1) the number of TCB's on the active chain is $\leq 45_8$
   2) that the TCB link (?TLNK), if it is not -1, is between 423 and 1777.

b) Reorder TCB?. A check is made whether the Ready TCB that was found by the search needs to be re-ordered within a priority level. If so, the same checks as in (a) above are made.

c) Floating Point save area needs to be mapped. Any task that uses the Floating Point Unit must reserve an 18 word area in which to save the Floating Point state when that task or process is rescheduled. This area is reserved via the ?IFPU task call. When the task again gets control, the Floating Point Unit must be reloaded if offset ?TFPS≠Ø. If an error occurs on the reload because the address in ?TFPS is invalid, a software "trap" will occur.

d) Extended Variable save area needs to be remapped. This technique is explained in the AOS Programmer's Manual (Ø93-12Ø) on Page D-4 for a Rev 1 version. Basically this allows each Task to utilize a "common" area of the 1st memory page (Ø-1777) for its own specific information. This area is restored from the task's storage area when the task gets control and copied into that storage area when the task loses control. The offset ?TELN contains the address of the task's storage area. The UST offset, USTES, contains the address of the "common" area in the 1st memory page and USTEZ contains the number of words in this "common" area. If offset ?TELN≠Ø for the current TCB, this area is being used. During reload, two checks are made:

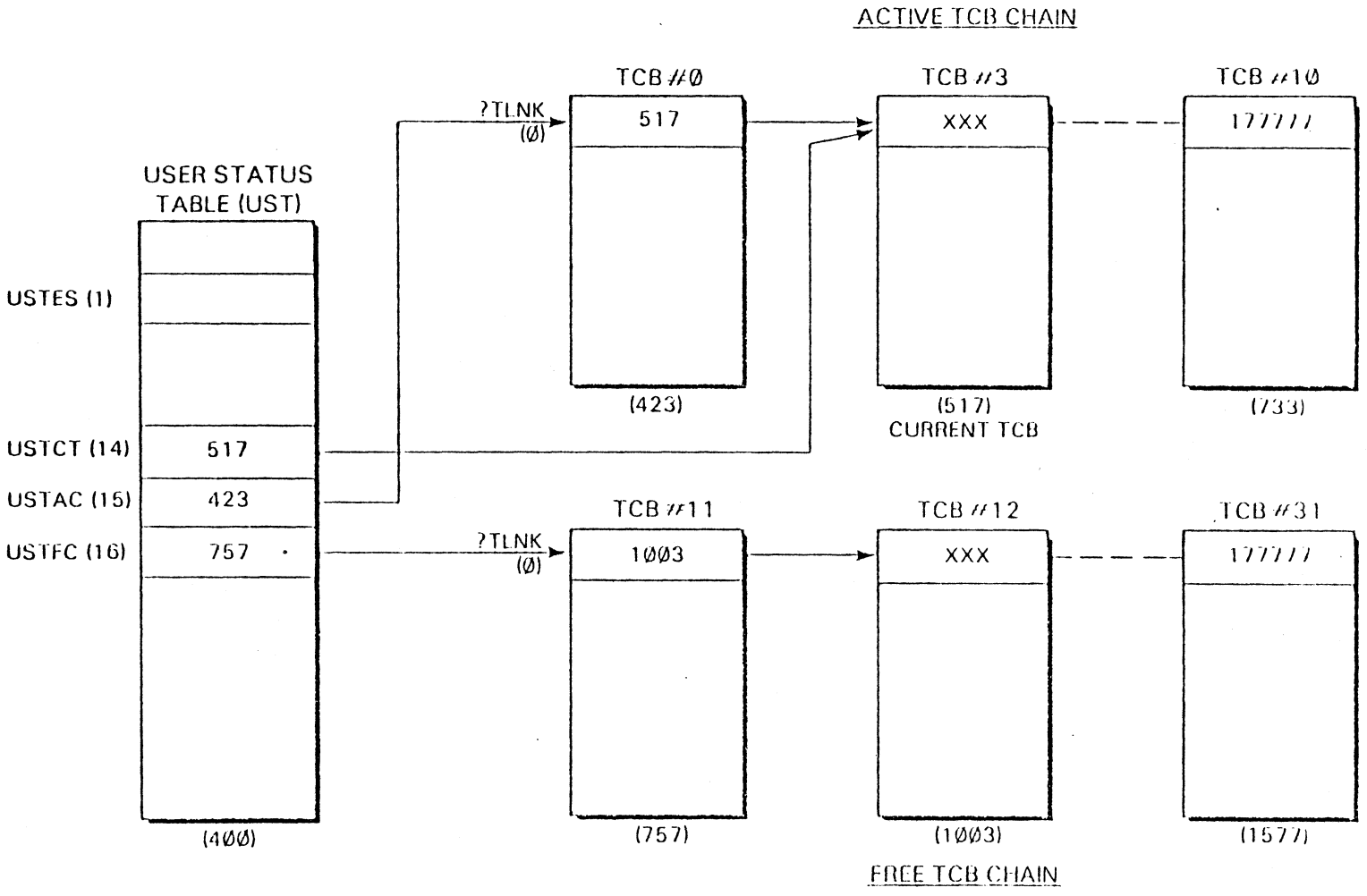


Figure 5.2 - User Task Control Block (TCB) Linkage Example

1) that the address in ?TELN is valid and exists in logical address space.

2) that the address in USTES is between 0 and 1777.

Either of the above will cause a "software" trap to result.

e) Execute the current TCB. Prior to executing, re-verify that USTCT (414) still contains a valid TCB address. It must be between 423 and 1777.

Any of the above errors will cause a trap to occur.

5. Process Table - TCB's can be linked onto the Process Table extension if the system call they have issued cannot be immediately executed. Checks are made when putting them on and taking them off. (Refer to figure 5.3).

a) Linking on - When a TCB is being linked on, two checks are made:

1) the number of TCB's presently attached must be less than 45_8.

2) the TCB address must be between 76422 and 77754.

b) Linking off - when a TCB is being removed, the TCB link addresses contained in offset ?TSLK(21) for each linked TCB are checked to be between 76423 and 77754.

If either of the above occurs, an error (ie: trap) is indicated.

6. Synchronous I/O - If the Synchronous Communications gear, SLM, is being used, the User TCB is checked in two places:

a) that the "Sync I/O in Prog" bit, ?TSYN, is set in the TCB status word, ?TSTAT.

b) that the synchronous line # that is stored in offset ?TSYS of the TCB is the same as the current line number.

Both of these checks are done during the processing of a synchronous line completion. If either fails, a "software" trap will occur.

The most obvious software cause for this type of failure would be if a User were to accidently write into the area where the UST and/or TCB's reside. One wrong word in the proper place would cause this failure since the above areas are not write-protected. Be aware that a failure in any of the above checks need not be software in nature but could very easily be caused by accessing the wrong location or by a bit pick or drop.

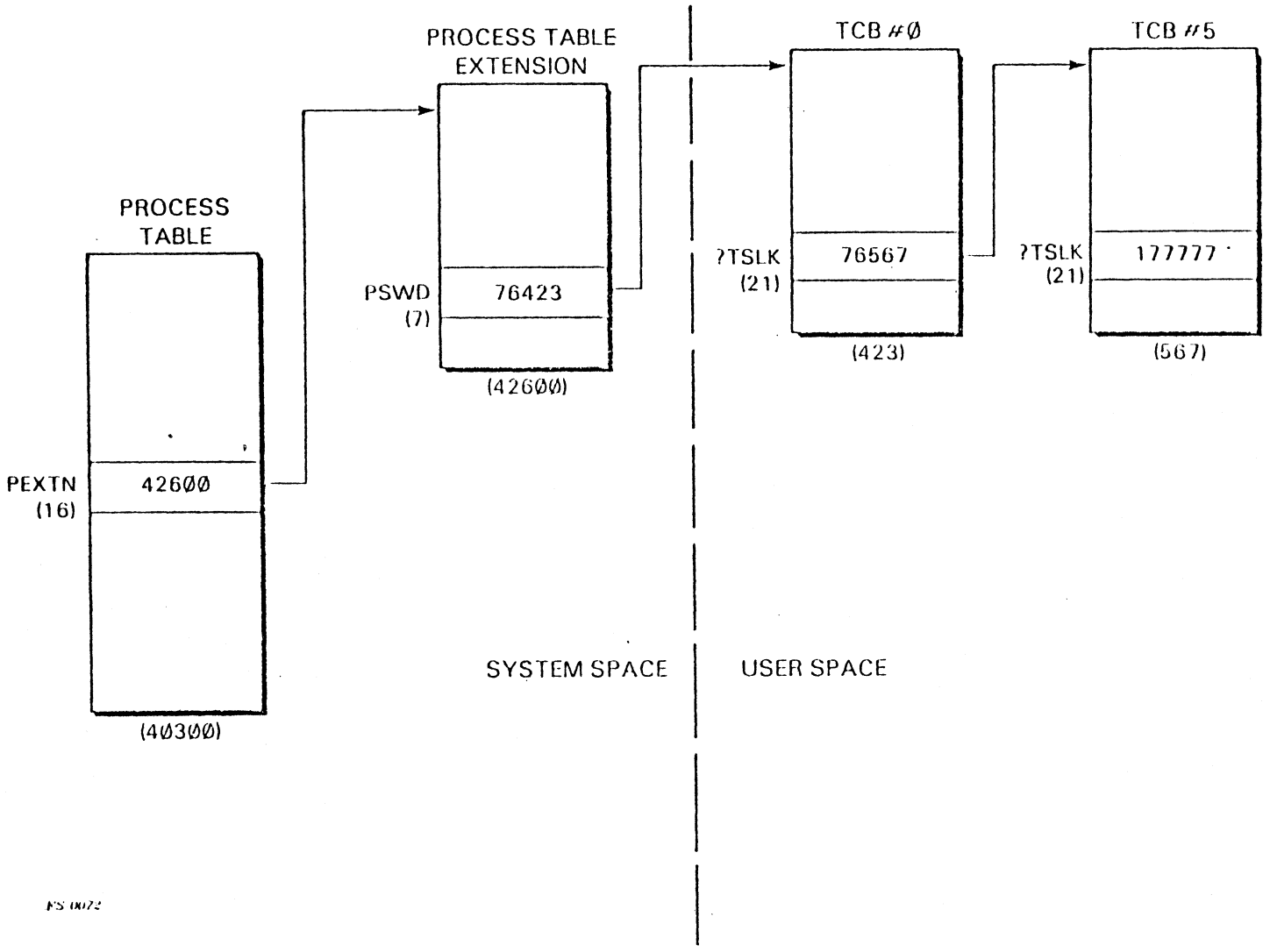


Figure 5.3 - Process Table / Waiting TCB Linkage

In the above instances, the trap bit is set for the respective process. When the process again gets control, an error similar to the following is typed out:

```
*ABORT*
USER TRAP, USER DATA BASE ERROR
CØ, PC 15776, ACØ Ø, AC1 15, AC2 11361Ø, AC3 447
ERROR: FROM PROGRAM
PROC/DEF/RES/IOC/BLOCK, CON
*)
```

Note that the Carry, PC and accumulators in the above typeout are meaningless because no information was stored in those respective locations when the failure occurred.

A "Break" file is created, the same as for the "hardware trap" case. Using the procedures outlined in the "hardware trap" section, the breakfile can be "DEDIT'd" or "DISPLAY'd" to verify that the contents of the TCB's seem reasonable. The key locations to check are as follows:

a) ?TSTAT (offset 1) - verify whether bit 4=1. Finding bit 4=1 is not in itself conclusive of a software failure. The rest of the bits should be examined for validity and it's contents should be noted for future reference.

b) ?TSYS (offset 17) - verify whether the address in ?TSYS is of the format x76xxx, x77xxx, or 177777; and if so is between 76423 and 77623 for a TCB doing a ?DELAY system call. To determine whether a TCB is involved in a ?DELAY system call, two checks must be true:

   1) bit Ø must be equal to 1 in ?TSTAT (ie: the TCB suspended).

   2) Using bit 1-15 of ?TPC as an address, the code which is pointed to must be of the following format:

```
          ØØ6Ø17
          ØØØØ15
          XXXXXX
   ?TPC ——> XXXXXX
```

   If 1) and 2) are both true, the TCB is involved in a ?DELAY system call.

c) ?TLNK (offset Ø) - verify that this link value is valid. This can be done by tracing through the active chain which is pointed to by USTAC (offset 15). The last TCB on the chain will contain a "177777" in this location. Note that there cannot be more than 45_8 TCB's on the active chain.

e) ?TFPS (offset 15) - verify that the value contained in ?TFPS if not Ø can be properly mapped. Does the address fall within the valid logical address space of the User Program? Are any of the User Tasks actually using the Floating Point Unit and if so, is this one of them?

f) ?TELN (offset 14) - verify that the value contained in ?TELN if not Ø can be properly mapped. Does the address fall within the valid logical address space of the User Program? Is the User using the Extended Variable Save Area feature?

There are three locations within the User Status Table (UST) that can also be checked for validity. They are:

g) USTES (offset 1) - verify that the value found is between Ø and 1777.

h) USTCT (offset 14) - verify that the value found is between 423 and 1777.

i) USTAC (offset 15) - same as for USTCT.

The above are the checks that can be made with a minimum of effort.

If checks of the above areas do not yield any clues as to the problem and the failure is occurring often enough, a test operating system can be built which will "Panic" or "Halt" when any of the errors outlined in 1 through 6 occurs. A core dump can then be taken and analyzed as to the cause for the failure. The implimentation of the "hooks" in the test operating system is beyond the scope of this package. If you feel that you are at that point of investigation, request additional help from National Technical Support.

The contents, along with a short meaning for each offset in the User Status Table (UST) and Task Control Block (TCB), can be found toward the end of the User Parameter (PARU) file on a system disk, illustrated in Appendix E of the AOS Programmer's Manual (093-120), and on the next couple of pages.

TABLE 5.2 SUMMARY TABLE

Task Control Blocks (TCB) on the Active Chain

Mnemonic	Offset	Valid Values
?TLNK	Ø	423-1777 or 177777
?TSTAT	1	Note whether bit 4=1
?TELN	14	Note if not =Ø
?TFPS	15	Note if not =Ø
?TSYS	17	if ?DELAY call must be of format X76XXX, X77XXX or 177777. If so must be between 76423 and 77623.
?TSLK	21	Ø or 76423-77754

User Status Table (UST)

Mnemonic	Offset	Valid Values
USTES	1	Ø-1777
USTCT	14	423-1777
USTAC	15	423-1777

TABLE 5.3

```
        ; USER STATUS TABLE (UST) TEMPLATE

000400 .DUSR    UST=     400      ; START OF USER STATUS AREA

000000 .DUSR    USTEZ=   0        ; EXTENDED VARIABLE WD CNT
000001 .DUSR    USTES=   1        ; EXTENDED VARIABLE PAGE 0 START
000002 .DUSR    USTSS=   2        ; SYMBOLS START
000003 .DUSR    USTSE=   3        ; SYMBOLS END
000004 .DUSR    USTS1=   4        ; SYSTEM WORD
000005 .DUSR    USTS2=   5        ; SYSTEM WORD
000006 .DUSR    USTDA=   6        ; DEB ADDR OR -1
000007 .DUSR    USTFL=   7        ; FLAG WORD (SEE DEFINITION BELOW)
000010 .DUSR    USTSL=   10       ; SHARED LIBRARY LIST POINTER
000011 .DUSR    USTIT=   11       ; INTERRUPT ADDRESS
000012 .DUSR    USTRV=   12       ; REVISION OF PROGRAM
000013 .DUSR    USTTC=   13       ; NUMBER OF TASKS (1 TO 32.)
000014 .DUSR    USTCT=   14       ; CURRENTLY ACTIVE TCB
000015 .DUSR    USTAC=   15       ; START OF ACTIVE TCB CHAIN
000016 .DUSR    USTFC=   16       ; START OF FREE TCB CHAIN
000017 .DUSR    USTBL=   17       ; # IMPURE BLKS
000020 .DUSR    USTOD=   20       ; OVLY DIRECTORY ADDR.
000021 .DUSR    USTST=   21       ; SHARED STARTING BLK #
000022 .DUSR    USTSZ=   22       ; SHARED SIZE IN BLKS

000022 .DUSR    USTEN=   USTSZ    ; LAST ENTRY


        ; REDEFINITIONS FOR SYSTEM USAGE
000006 .DUSR    USTGB=   USTDA    ;GHOST BKPT ENTRY ADDRESS
000002 .DUSR    USTGC=   USTSS    ;GHOST CLEAN UP ADDRESS FOR TERMS
000012 .DUSR    USTGD=   USTRV    ;GHOST DISPATCHER ADDR


        ; UST FLAGS (in USTFL)

100000 .DUSR   ?UFIN=    1B0      ;GHOST IS INITIALIZED AND CAN B ENTERED
040000 .DUSR   ?UFIP=    1B1      ;GHOST INIT IS IN PROGRESS- MUUST WAIT F
020000 .DUSR   ?UFDR=    1B2      ;INHIBIT RESCH
010000 .DUSR   ?UFDB=    1B3      ;PROCESS IS BEING DEBUGGED
004000 .DUSR   ?UFID=    1B4      ;PROCESS IS IN DEBUGGER
002000 .DUSR   ?UFPH=    1B5      ;PRIMARY TASK RUNTIME HOLD ON RESCHEDULI
001000 .DUSR   ?UFHP=    1B6      ;HOLD PRIMARY TASKS - RUN GHOST ONLY
```

TABLE 5.4

```
        ; TASK CONTROL BLOCK (TCB) TEMPLATE
        ; ?TSP THROUGH ?TCUD MUST BE KEPT IN ORDER AND CONTIGUOUS
        ; ?TSP THROUGH ?TPC ARE ORDERED FOR RESTORE INSTRUCTION

000000 .DUSR    ?TLNK=   0          ;LINK (NOTE- MUST BE OFFSET 0 !!!)
000001 .DUSR    ?TSTAT=  1          ;STATUS BITS
000002 .DUSR    ?TSP=    2          ;STACK POINTER
000003 .DUSR    ?TFP=    3          ;FRAME POINTER
000004 .DUSR    ?TSL=    4          ;STACK LIMIT
000005 .DUSR    ?TSO=    5          ;FAULT HANDLER
000006 .DUSR    ?TAC0=   6          ;AC0
000007 .DUSR    ?TAC1=   7          ;AC1
000010 .DUSR    ?TAC2=   10         ;AC2
000011 .DUSR    ?TAC3=   11         ;AC3
000012 .DUSR    ?TPC=    12         ;PC AND CARRY
000013 .DUSR    ?TUSP=   13         ;USP
000014 .DUSR    ?TELN=   14         ;TCB EXTENTION ADDR
000015 .DUSR    ?TFPS=   15         ;FPU SAVE AREA POINTER
000016 .DUSR    ?TCUD=   16         ;CURRENT DESCRIPTOR
000017 .DUSR    ?TSYS=   17         ;SYSTEM CALL WORD
000020 .DUSR    ?TIDPR=  20         ;ID (LEFT BYTE) PRIORITY (RIGHT BYTE)
000021 .DUSR    ?TSLK=   21         ;SYSTM CALL LINK
000022 .DUSR    ?TKAD=   22         ;KILL POST PROCESSING ROUTINE
000023 .DUSR    ?TGEX=   23         ;GHOST EXTENSION ADDRESS

000024 .DUSR    ?TLN=    ?TGEX-?TLNK+1    ;LENGTH OF TCB
000015 .DUSR    ?TGXL=   ?TSYS-?TSP+1     ;LENGTH OF TCB EXTENSION

        ; TASK STATUS BITS (in ?TSTAT)

100000 .DUSR    ?TSPN=   1B0        ; TASK PENDED
040000 .DUSR    ?TSSG=   1B1        ; WAITING FOR OVERLAY AREA OR .XMTW/.REC
020000 .DUSR    ?TSSP=   1B2        ; SUSPENDED
010000 .DUSR    ?TSRC=   1B3        ; WAITING FOR TRCON
004000 .DUSR    ?TSIG=   1B4        ; IN GHOST CONTEXT
002000 .DUSR    ?TSIW=   1B5        ; WAITING FOR GHOST INIT TO COMPLETE
001000 .DUSR    ?TSGS=   1B6        ; GHOST WAITING FOR SYNCH.
000400 .DUSR    ?TSAB=   1B7        ; PENDED AWAITING ?GABORT
000200 .DUSR    ?TSUF=   1B8        ; PEND BIT AVAILABLE DIRECTLY TO USERS
000100 .DUSR    ?TSYN=   1B9        ; SYNCHRONOUS I/O COMPLETION BIT
```

5.4 Ghost Trap

When I gave the examples for the trap messages, the second line of each began with the text "User Trap, ...". In actuality, there are two possibilities - "User Trap" or "Ghost Trap". The User space and Ghost space are two seperate contexts with a maximum of 32KW which interact with each other but do not use any of the other's space with the exception of the 1st User page containing the UST and TCBs. The User space is obvious, it contains the actual User written modules, task modules, etc. The Ghost space contains a program which is transparent to the User (ie: ghost) and handles the following facilities for the User:

1. Debugger
2. all record I/O modules (ie: byte devices) and associated data.
3. some miscellaneous system calls.

There are interactions between the two programs. System calls will be made from both User space and Ghost space by the respective program. The Operating System uses the B program map and the User/ Ghost share the A program map. When doing a "DISPLAY" of a break file where a User was interacting with the Ghost, the Ghost space will be contained between $100000-177777_8$.

Figure 5.4 shows the linkages between a Process Table and the User and Ghost areas. Also shown are the general contents of each area. An example of the interactions between the User and Ghost would be:

1. The User issues a ?READ system call to read a 40 byte record from a disk file.
2. The Operating System recognizes the ?READ as a system call involving the Ghost. The Ghost is enabled and control is passed to it along with information about the system call.
3. The Ghost analyzed the system call and it issues an ?RDB system call to read the entire disk block containing the appropriate record into its buffer area.
4. After all the data from the ?RDB system call has been moved into the Ghost buffer area via the data channel, the Ghost

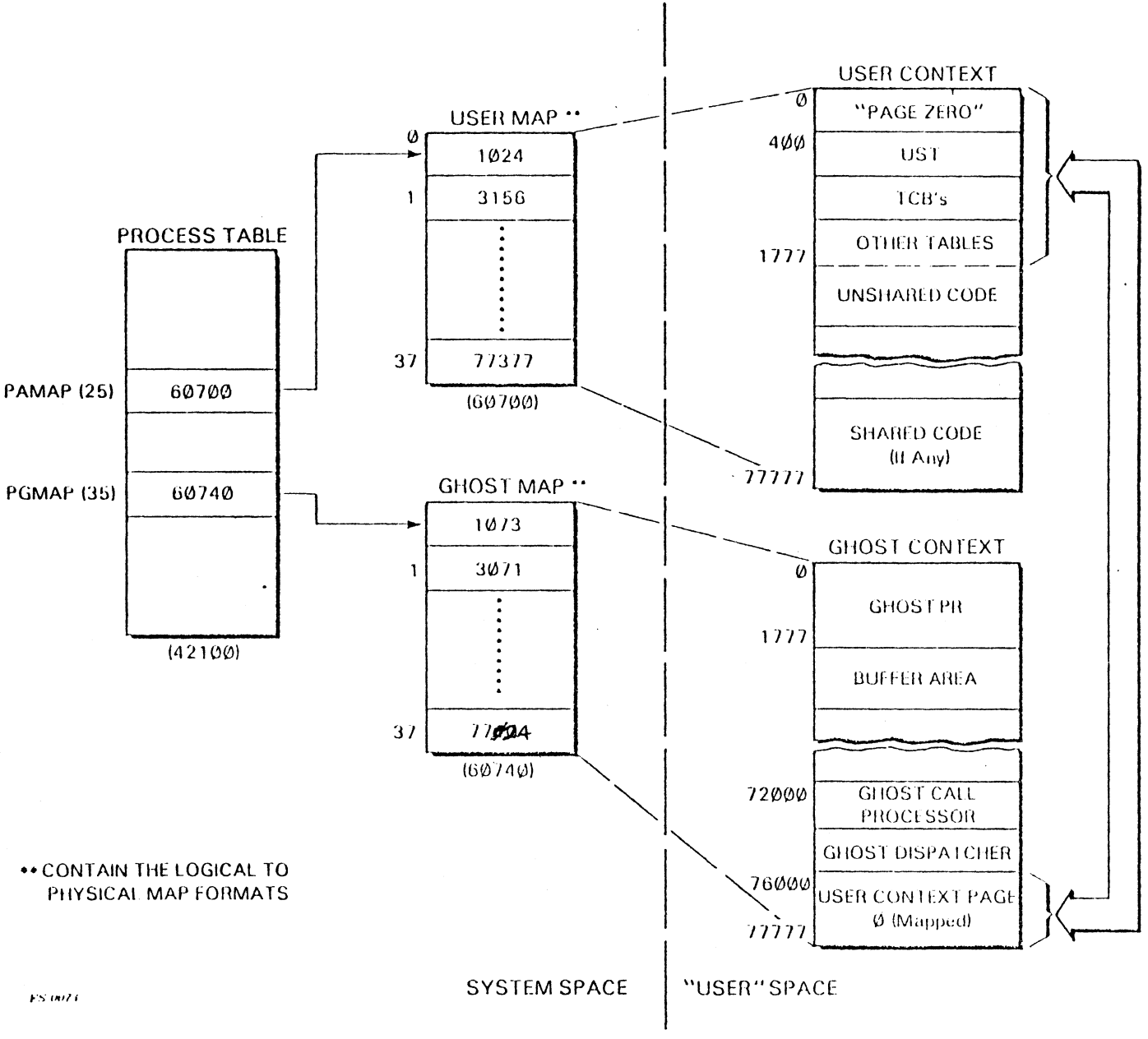


Figure 5.4 Linkages between the Ghost and User Areas and the Process Table

issues a ?MBFG system call. This call moves the bytes associated with the record requested by the User from the Ghost buffer into the User buffer.

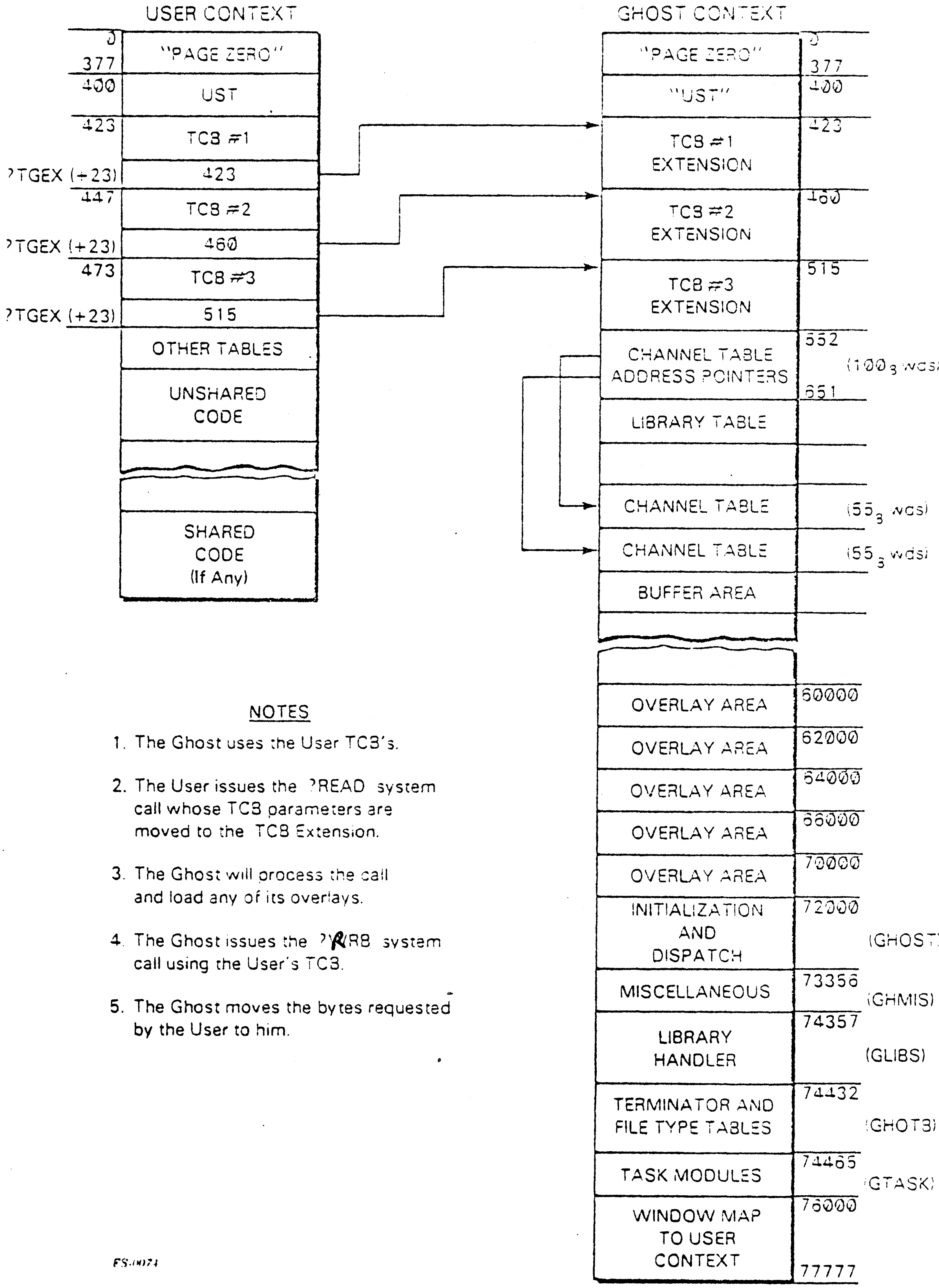
5. When Ghost is finished, control is returned to the User at the ?READ normal return.

As can be noted by the above example, the User has no idea that his system call was partly processed by the Ghost. The Ghost, as was mentioned before, is transparent to the User.

Now to examine both the Ghost and the ?READ System Call in a little more detail. Figure 5.5 shows the User/Ghost TCB Linkages. When the User issues the ?READ system call, control is passed to the Ghost who will eventually issue the ?RDB system call. Since both the Ghost and the User use the same TCB, the information stored in the TCB must be moved prior to the ?RDB system call taking place to avoid it being over-written. The Ghost moves this information into a TCB Extension in the Ghost context. There is one Extension for each possible User TCB. If a User defines 12 TCBs, there will be 12 TCB Extensions. The contents of the TCB extension is shown in Table 5.7. After the information has been moved into the extension, the Ghost can use the User TCB to issue the ?RDB system call. After the system call has completed, the Ghost moves the bytes from his buffer to the User buffer via the ?MBFG system call. The User TCB is now restored from the respective Extension, and control is returned to the User at the ?READ normal return.

While the Ghost is handling the processing of the system call, a bit in the TCB status word, ?TSIG, indicates that the processing that is taking place is in the Ghost context not the User context. Offset 23, ?TGEX, of the TCB will be the address of the TCB Extension in the Ghost context.

The Ghost keeps track of "channels" that have been "Open'd" by the User as well as channels that the Ghost itself has opened. Each channel is defined by a Channel Descriptor Table whose offsets are shown in Table 5.8. There is a Channel Address Table of $1\emptyset\emptyset_8$ words in length, one for each possible channel, whose contents point to the Channel Descriptor Table (CDT) for that channel. These links are shown in Figure 5.5.



NOTES

1. The Ghost uses the User TCB's.
2. The User issues the ?READ system call whose TCB parameters are moved to the TCB Extension.
3. The Ghost will process the call and load any of its overlays.
4. The Ghost issues the ?WRB system call using the User's TCB.
5. The Ghost moves the bytes requested by the User to him.

Figure 5.5 User/Ghost Task Control Block (TCB) Linkage

The Ghost uses both a Program File (.PR) and an Overlay File (.OL). The Program File is 1000 wds in length and only contains the Page Zero information (GZERO) and the Ghost Bootstrap (GBOOT) program. The Overlay File contains the 6 overlays, GOV0-GOV5, plus the other modules which are read into the Shared Area between 72000 and 75777. The entry points into the GOVx overlays are shown in Table 5.5.

Certain error messages can occur either while the Ghost is being "initialized" or during its execution. These are documented in Table 5.6 along with a description of the message.

TABLE 5.5

GHOST OVERLAY USAGE

Ghost Overlay	Overlay #	Entry Points
GDICT	Ø	--
GOVØ	1	?OPEN
GOV1	2	?READ, ?WRITE, ?SPOS, ?RTN
GOV2	3	?PIO, ?TERM, ?CHAIN, ?ISHR, ?DBLD, ?SEND, ?ENQUE, ?OVOPN, ?ISHG
GOV3	4	?GTME, ?RETURN, ?ABORT, ?PRCC
GOV4	5	?LOPEN, ?GPOS, ?EXEC ?CTOD, ?CDAY, ?SEOF ?CEO, ?SIO
GOV5	6	?CLOSE, ?EMSG, ?ASSIGN, ?DEASSIGN, ?GCHAR ?SCHAR

TABLE 5.6

SPECIAL GHOST ERROR MESSAGES

1. "ERROR: code AT: address DURING GHOST INITIALIZATION"

   (GBOOT) - Used to handle System Call error returns during the Ghost initialization. At the time of failure:

   code = System Error code

   address = AC3 contents from the "JSR" in the System Call error return.

   System Calls monitored by this method are ?SOPEN, ?SSHPT, and ?SPAGE.

2. "GINIT MEMORY FAILURE"

   (GHOST) - An error occurred on a "?MEMI" system call. The error code is not saved. The text message is sent via the IPC with the "GTERM" system call.

3. "Ghost Trap" - "Defer" Violation

   (GOV2) - DBSHR+5 - location after the ?.RETURN call.

   (GOV3) - RET3+3 and RET3+4 - locations after the ?.TERM call

   (GHMIS) - FSHAR+13 - in releasing a shared overlay area, the page being released is marked as "not in use".

   (GHMIS) - GSENQ+12 - while enqueing an area of core to a "free area" chain (each chain is by area size), we find the area already on the chain.

   (GHMIS) - FMEM+6 and FMEM+21 - both checks deal with illegal values during the release of memory that the Ghost was using.

   (GTASK) - GOBCK+6 - when a virtual address is saved on the Ghost stack, bit Ø = 1. In this case, what is supposed to be a virtual address "POP'd" from the stack, and bit Ø is found to be "Ø".

4. "Unknown Message Code xxxxxx"

   (GOV5) - ERROR - Error messages are grouped in an error message FILE. The actual error code message is accessed via a "Group#/ERR#" type format. While using

this format, it is found that the:

1) Group # is improper (not Ø=#=177)
2) The error code is not defined within the Group or is out of range.
3) a byte pointer to the text message is invalid (=Ø).

Compare the code given with the appropriate values in the error message file.

5. "STACK OVERFLOW IN GHOST"

(GHMIS) - STILT - A stack overflow has occurred due to Loc 4Ø = Loc 42. Each Ghost stack is initialized to a specific value depending on the number of temporaries needed.

TABLE 5.7

```
        ; DEFINE THE GHOST EXTENSION

000000 .DUSR GXSP=      0               ; STACK POINTER
000001 .DUSR GXFP=      GXSP+1          ; FRAME POINTER
000002 .DUSR GXSL=      GXFP+1          ; STACK LIMIT
000003 .DUSR GXSO=      GXSL+1          ; FAULT HANDLER
000004 .DUSR GXAC0=     GXSO+1          ; AC0
000005 .DUSR GXAC1=     GXAC0+1         ; AC1
000006 .DUSR GXAC2=     GXAC1+1         ; AC2
000007 .DUSR GXAC3=     GXAC2+1         ; AC3
000010 .DUSR GXPC=      GXAC3+1         ; PC AND CARRY
000011 .DUSR GXUSP=     GXPC+1          ; USP
000012 .DUSR GXELN=     GXUSP+1         ; TCB EXTENSION ADDRESS
000013 .DUSR GXFPSV=    GXELN+1         ; FPU SAVE AREA POINTER
000014 .DUSR GXCUD=     GXFPSV+1        ; CURRENT DESCRIPTOR
000015 .DUSR GXSYS=     GXCUD+1         ; SYSTEM CALL WORD
000015 .DUSR GXTND=     GXSYS           ; END OF TCB COPY AREA
000016 .DUSR GXTPN=     GXTND+1         ; TASK IDENTIFIER NUMBER

        ;THE FOLLOWING FIVE LOCATIONS ARE USED AS A TEMPORARY
        ;STACK FRAME WHEN THE TASK FIRST ENTERS THE GHOST, MATCH IT, JAC

000017 .DUSR GXCDT=     GXTPN+1         ; SHARED LIBRARY CHANNEL
000020 .DUSR GXIOP=     GXCDT+1         ; I/O PACKET ADDRESS
000021 .DUSR GXTM1=     GXIOP+1         ; TEMPORARY
000022 .DUSR GXTM2=     GXTM1+1         ; TEMPORARY
000023 .DUSR GXTM3=     GXTM2+1
000024 .DUSR GXTM4=     GXTM3+1
000025 .DUSR GXDSC=     GXTM4+1         ;ADDR OF CURRENT OVERLAY DESC.

000016 .DUSR GXSFA=     GXTPN           ;ADDRESS OF TEMPORARY STACK FRAM
000025 .DUSR GXSUL=     GXDSC           ;TEMPORARY STACK FRAME LIMIT

000026 .DUSR GXGSA=     GXDSC+1         ; GHOST STACK ADDRESS
000027 .DUSR GXGSZ=     GXGSA+1         ; GHOST STACK SIZE
000030 .DUSR GXGSP=     GXGSZ+1         ; GHOST STACK POINTER
000031 .DUSR GXGFP=     GXGSP+1         ; GHOST FRAME POINTER
000032 .DUSR GXGSL=     GXGFP+1         ; GHOST STACK LIMIT
000033 .DUSR GXGSO=     GXGSL+1         ; GHOST STACK FAULT ADDRESS
000034 .DUSR GXFLG=     GXGSO+1         ; GHOST FLAG WORD

000034 .DUSR GXEND=     GXFLG           ; END OF EXTENDER
000035 .DUSR GXLEN=     GXEND+1         ; LENGTH OF GHOST EXTENSION
```

TABLE 5.3

```
        ; DEFINE CHANNEL DEFINITION TABLE
000000  .DUSR CDLOK=    0               ; SYNC LOCK WORD
000001  .DUSR CDTCB=    CDLOK+1         ; ADDRESS OF OPENER'S TCB(USER +
000002  .DUSR CDFST=    CDTCB+1         ; I/O STATUS WORD (SEE BELOW)
000003  .DUSR CDOST=    CDFST+1         ; OPEN STATUS
000004  .DUSR CDORL=    CDOST+1         ; RECORD LENGTH (OPEN)
000005  .DUSR CDTTA=    CDORL+1         ; TERMINATOR TABLE ADDRESS
000006  .DUSR CDUBF=    CDTTA+1         ; USER'S DATA AREA ADDRESS
000007  .DUSR CDTM1=    CDUBF+1         ;TEMPORARIES
000010  .DUSR CDTM2=    CDTM1+1

000011  .DUSR CDPKT=    CDTM2+1         ; READ/WRITE PACKET
000011  .DUSR CDCHN=    CDPKT+?ICH      ;CHANNEL NUMBER
000012  .DUSR CDSIS=    CDPKT+?ISTI     ;STATUS WORD (IN)
000013  .DUSR CDSOS=    CDPKT+?ISTO     ;STATUS WORD (OUT)
000014  .DUSR CDBAD=    CDPKT+?IBAD     ;BYTE ADDRESS FOR DATA
000015  .DUSR CDRES=    CDPKT+?IRES     ;RESERVED
000016  .DUSR CDRCL=    CDPKT+?IRCL     ;RECORD LENGTH
000017  .DUSR CDRLR=    CDPKT+?IRLR     ;RECORD LENGTH (RETURNED)
000020  .DUSR CDRNH=    CDPKT+?IRNH     ;RECORD # (HIGH)
000021  .DUSR CDRNL=    CDPKT+?IRNL     ;RECORD # (LOW)
000022  .DUSR CDFNP=    CDPKT+?IFNP     ;FILE NAME BYTE POINTER
000023  .DUSR CDMRS=    CDPKT+?IMRS     ;MAG TAPE RECORD SIZE
000024  .DUSR CDDEL=    CDPKT+?IDEL     ;DELIMITER TABLE ADDRESS

        ;DEFINE ?GOPEN RETURN PACKET AREA

000025  .DUSR CDGOP=    CDPKT+?IBLT             ;?GOPEN RETURN  PACKET
000025  .DUSR CDGFL=    CDGOP+?OPFL     ; FLAGS WORD(ON ENTRY ONLY)
000025  .DUSR CDPCH=    CDGOP+?OPCH     ;CHANNEL
000026  .DUSR CDPTY=    CDGOP+?OPTY     ;FORMAT/FILE TYPE
000027  .DUSR CDPFC=    CDGOP+?OPFC     ;FILE CONTROL PARAMETERS
000027  .DUSR CDPPH=    CDGOP+?OPPH     ;CONTROL PORT (HIGH)
000030  .DUSR CDPPL=    CDGOP+?OPPL     ;CONTROL PORT (LOW)
000030  .DUSR CDPEH=    CDGOP+?OPEH     ;FILE EOF(HIGH)
000031  .DUSR CDPEL=    CDGOP+?OPEL     ;FILE EOF(LOW)

000032  .DUSR CDLBF=    CDGOP+?OPLT     ; CHANNEL BUFFER LENGTH
000033  .DUSR CDBSP=    CDLBF+1         ; BYTE POINTER TO BUFFER
000034  .DUSR CDBCP=    CDBSP+1         ; CURRENT BUFFER BYTE POINTER
000035  .DUSR CDBCE=    CDBCP+1         ;CURRENT END OF DATA
000036  .DUSR CDBEP=    CDBCE+1         ; BUFFER END POINTER
000037  .DUSR CDBRB=    CDBEP+1         ; RECORD BUFFER ADDRESS
```

TABLE 5.3 (Con't)

```
; DEFINE BLOCK I/O SUBSET OF CHANNEL DEFINITION TABLE

000040 .DUSR CDBIO=     CDBRB+1          ; BLOCK I/O TABLE
000040 .DUSR CDSTI=     CDBIO+?PSTI      ; RECORD COUNT (RIGHT),STATUS IN
000041 .DUSR CDSTO=     CDBIO+?PSTO      ; STATUS WORD (OUT)
000042 .DUSR CDUAD=     CDBIO+?PCAD      ; CORE ADDRESS
000043 .DUSR CDPRS=     CDBIO+?PRES      ; RESERVED WORD
000044 .DUSR CDDBH=     CDBIO+?PRNH      ; RECORD NUMBER (HIGH)
000045 .DUSR CDDBL=     CDBIO+?PRNL      ; RECORD NUMBER (LOW)
000046 .DUSR CDBLN=     CDBIO+?PRCL      ; MAX RECORD LENGTH
000047 .DUSR CDBC=      CDBIO+?PBLT      ; TOTAL BYTES MOVED (RETURNED)


; DEFINE BUFFER STATUS BIT ADDRESSES

000040 .DUSR BCDMD=     CDFST*SCBPW+0.   ; BUFFER HAS BEEN MODIFIED
000041 .DUSR BCDPD=     CDFST*SCBPW+1.   ; FILE IS A PERIPHERAL DEVICE
000042 .DUSR BCDEC=     CDFST*SCBPW+2.   ; ECHO
000043 .DUSR BCDES=     CDFST*SCBPW+3.   ; END OF ADDRESS SPACE ENCOUNTER
000044 .DUSR BCDVL=     CDFST*SCBPW+4.   ; VAR LENGTH KNOWN
000045 .DUSR BCDSI=     CDFST*SCBPW+5.   ; SERIAL I/O DEVICE
000046 .DUSR BCDMC=     CDFST*SCBPW+6.   ; BUFFER EMPTY
000047 .DUSR BCDMT=     CDFST*SCBPW+7.   ; MAG TAPE FILE
000050 .DUSR BCDSP=     CDFST*SCBPW+8.   ; SPOOLER FILE
000051 .DUSR BCDMF=     CDFST*SCBPW+9.   ; MAGIC FILE INDICATOR
000052 .DUSR BCDWA=     CDFST*SCBPW+10.  ; WRITE ACCESS ALLOWED TO FILE
000053 .DUSR BCDPI=     CDFST*SCBPW+11.  ; PRIORITY REQUEST ALLOWED.
000054 .DUSR BCDEQ=     CDFST*SCBPW+12.  ; EXEC SPOOL QUEUE
000055 .DUSR BCDIP=     CDFST*SCBPW+13.  ; IPC FILE
000057 .DUSR BCDER=     CDFST*SCBPW+15.  ; ERROR OCCURRED

; DEFINE USER STATUS BIT ADDRESSES

000245 .DUSR BCDXO=     CDUIS*SCBPW+?IOEX         ; FORCE EXCLUSIVE IO
000241 .DUSR BCDCF=     CDUIS*SCBPW+?ICFB         ; CHANGE RECORD FORMAT
000242 .DUSR BCDST=     CDUIS*SCBPW+?IPTB         ; RECORD POSITIONING
000244 .DUSR BCDFO=     CDUIS*SCBPW+?IFOB         ; FORCE OUTPUT BIT
000251 .DUSR BCDCR=     CDUIS*SCBPW+?OF1B         ; CREATE FILE BIT ADDRES
000252 .DUSR BCDCE=     CDUIS*SCBPW+?OF2B         ; CORRECT ERROR BIT ADDR
000243 .DUSR BCDIB=     CDUIS*SCBPW+?IBIB         ; BINARY IO MODE
000250 .DUSR BCDAP=     CDUIS*SCBPW+?APBT         ; APPEND TO FILE
000247 .DUSR BCDPR=     CDUIS*SCBPW+?PDLM         ; WANT PRIORITY I/O TO H
000246 .DUSR BCDNW=     CDUIS*SCBPW+?IIPS         ; NO WAIT ON IPC OPENS
000073 .DUSR BCDIN=     CDOST*SCBPW+?OPIB         ; INPUT FILE
000074 .DUSR BCDOT=     CDOST*SCBPW+?OPOB         ; OUTPUT FILE

000060 .DUSR BCDFG=     CDOST*SCBPW+0             ;CREATE FLAG USED ONLY F
```

TABLE 5.3 (Con't)

```
        ; DEFINE IPC SUBSET OF CHANNEL DEFINITION TABLE

000040  .DUSR CDIPC=    CDBIU           ; IPC TABLE
000040  .DUSR CDSFL=    CDIPC+?ISFL     ; MESSAGE FLAGS
000041  .DUSR CDFLG=    CDIPC+?IUFL     ; MESSAGE FLAGS
000042  .DUSR CDHIM=    CDIPC+?IDPH     ; PORT # OF FELLA I'M TALKIN' TO
000044  .DUSR CDME=     CDIPC+?IDPN     ; MY PORT #
000045  .DUSR CDLTH=    CDIPC+?ILTH     ; LENGTH OF MESSAGE OR BUFFER (I
000046  .DUSR CDPTR=    CDIPC+?IPTR     ; POINTER TO MESSAGE/BUFFER
000047  .DUSR CDRLT=    CDPTR+1         ; ?IS.R RECEIVE LENGTH
000050  .DUSR CDRPT=    CDRLT+1         ; ?IS.R RECEIVE BUF PTR

000051  .DUSR CDATA=    CDRPT+1         ; IPC DATA AREA
000051  .DUSR CDRDP=    CDATA           ; READ PORT NUMBER
000053  .DUSR CDWRP=    CDRDP+2         ; WRITE PORT NUMBER

000054  .DUSR CDRWE=    CDWRP+1         ;DATA AREA END

000004  .DUSR CDTLN=    CDRWE-CDATA+1   ; LENGTH OF DATA AREA

000054  .DUSR CDIND=    CDRWE           ; END OF IPC SUBSET

        ; SUBSET NEEDED FOR SPOOLER SUPPORT

000027  .DUSR CDSFN=    CDPFC           ; PTR TO TRUE FILENAME
000027  .DUSR CDEXC=    CDPFC           ; PTR TO EXEC PACKET
```

5.5 Special Case Traps

Following are a couple of "Special" case Traps. They will probably never occur but are documented here for completeness.

I. ?SMSK System Call

The ?SMSK System Call is used to "modify the current interrupt mask" and is documented in the AOS Programmer's Manual (Ø93-12Ø) on Page 8-6 for a Rev 1 version.

This System Call can only be issued from an interrupt service routine therefore the system must be at some interrupt level. If the ?SMSK call is issued at other than interrupt level, a "Trap" occurs.

Presently, the "Trap" feature does not appear to exist and AOS 1.Ø3 has eliminated the system call altogether.

II. False "SYC" Instruction

Various System Calls can be issued by the User which enter the operating system via the "SYC" instruction, complete their function in a few lines of code, and return to the User. Examples of this type would be the ?LEFD (Disable LEF mode) and ?LEFE (Enable LEF mode).

When the "SYC" is done to enter the system a "Return" block should be pushed on the stack pointed to by Physical locations 4Ø - 43. Prior to returning to the User, this stack is checked to verify that a "Return" block had been pushed. If the SP is equal to the Stack Base, no "Return" block was pushed and a "JMP@3" will occur. The "JMP" will cause the system to think a "Trap" occurred and the standard "User Trap..." message will be printed out similar to the following (C33Ø):

```
*ABORT*
USER TRAP, VALIDITY
CØ, PCØ, ACØ 72xxx, AC1 74xxx, AC3 xx
ERROR:  FROM PROGRAM
```

The key is that the "Carry" and "PC" will usually be zero. If a program jumps to the wrong place in core, the PC, C, and AC's can be random values.

TABLE 5.9

```
;
; SYSTEM CALLS
;

?SCL2   ?CREATE 0               ;CREATE FILE
?SCL1   ?DELETE 1               ;DELETE FILE
?SCL1   ?RENAME 2               ;RENAME A FILE
?SCL1   ?MEM    3               ;RETURN MEMORY LIMITS
?SCL1   ?GCHN   4               ;CHANGE THE CURRENT PROGRAM
?SCL1   ?PSTAT  5               ;PROCESS STATUS
?SCL1   ?DPMGR  6               ;DEFINE PERIPHERAL MANAGER

;***** WARNING:  CODE IN IOOV.SR ASSUMES ?RDB = 7 *****
?SCL2   ?RDB    7               ;READ A PHYSICAL BLOCK
; NOTE - ?RDB RESERVES 10

?SCL2   ?WPB    11              ;WRITE A PHYSICAL BLOCK
; NOTE - ?WPB RESERVES 12
?SCL2   ?GPRNC  13              ;CREATE A PROCESS
?SCL1   ?MEMI   14              ;ALLOCATE MEMORY
?SCL1   ?DELAY  15              ;SUSPEND FOR A TIME PERIOD
?SCL1   ?INTWT  16              ;WAIT FOR A CONSOLE INTERRUPT
?SCL1   ?MBTG   17              ;MOVE BYTES TO GHOST
?SCL1   ?MBFG   20              ;MOVE BYTES FROM GHOST
?SCL1   ?MBTU   21              ;MOVE BYTES TO PROCESS
; NOTE -  ?MBTU RESERVES 22
?SCL1   ?MBFU   23              ;MOVE BYTES FROM PROCESS
; NOTE -  ?MBFU RESERVES 24
?SCL2   ?ISEND  25              ;SEND AN IPC MESSAGE

;***** WARNING:  CODE IN SCHED.SR ASSUMES ?IREC = 26 *****
?SCL2   ?IREC   26              ;RECEIVE AN IPC MESSAGE

?SCL1   ?ILKUP  27              ;LOOK UP AN IPC NAME
?SCL2   ?RUNTM  30              ;GET RUNTIME STATISTICS
?SCL1   ?TABT   31              ;ABORT A TASK CALL
?SCL1   ?XLATE  32              ;TRANSLATE TO A MAPPED ADDR
?SCL1   ?CTYPE  33              ;CHANGE PROCESS TYPE
?SCL1   ?LINF   34              ;USER CALL TO USE INFOS
?SCL1   ?INFI   35              ;DECLARE THIS AS THE INFOS PROC
?SCL1   ?GTOD   36              ;GET TIME OF DAY
?SCL1   ?STOD   37              ;SET TIME OF DAY
?SCL1   ?SDAY   40              ;SET DATE
?SCL1   ?GDAY   41              ;GET DATE
?SCL1   ?IDEF   42              ;DEFINE A USER DEVICE
?SCL1   ?IRMV   43              ;REMOVE A USER DEFINED DEVICE
?SCL1   ?SSHPT  44              ;SET SHARED PARTITION
?SCL1   ?RPAGE  45              ;RELEASE A SHARED PAGE
?SCL1   ?DDIS   46              ;DISABLE CONSOLE INTERRUPTS
?SCL1   ?DEBL   47              ;ENABLE CONSOLE INTERRUPTS
?SCL1   ?DEBL   50              ;ENABLE DEVICE ACCESS
?SCL1   ?UDIS   51              ;DISABLE DEVICE ACCESS
?SCL1   ?STMAP  52              ;SET DCH MAP FOR USER DEVICE
?SCL1   ?ABTC   54              ;ABORT A SYSTEM CALL
?SCL1   ?GTERM  55              ;TERMINATE A PROCESS
?SCL2   ?GOPEN  56              ;OPEN (FOR GHOST USE)
?SCL2   ?GCLOSE 57              ;CLOSE (FOR GHOST USE)
?SCL1   ?SCLOSE 57              ;CLOSE A SHARED FILE
; NOTE - ?SCLOSE MUST HAVE SAME CODE AS ?GCLOSE
```

Table 5.9 (Con't)

```
;***** WARNING:  CODE IN IOQV,SP ASSUMES ?SPAGE = 61 *****
?SCL2   ?SPAGE  61                      ;INCLUDE A SHARED PAGE
; NOTE - ?SPAGE RESERVES 61

;***** WARNING:  CODE IN SCALL,SP ASSUMES ?IGHOST = 62 *****
?SCL1   ?IGHOST 62                      ;INITIALIZE THE GHOST

?SCL2   ?SOPEN  63                      ;SHARED OPEN OF A FILE
?SCL1   ?GPORT  64                      ;GET A PORT'S OWNER
?SCL1   ?TPORT  65                      ;TRANSLATE A PORT NUMBER
?SCL1   ?BLKPR  66                      ;BLOCK A PROCESS
?SCL1   ?UBLPR  67                      ;UNBLOCK A PROCESS
?SCL1   ?PRIPR  70                      ;CHANGE A PROCESS PRIORITY
?SCL1   ?SGNL   71                      ;SIGNAL THE SYSTEM
?SCL1   ?GUNM   72                      ;GET A PROCESS'S USER NAME
?SCL1   ?GSHPT  73                      ;GET SHARED PARTITION VALUES
?SCL1   ?GHRZ   74                      ;GET CLOCK FREQ
?SCL1   ?DIR    75                      ;CHANGE WORKING DIRECTORY
?SCL2   ?INIT   76                      ;INITIALIZE AN LDU OR MTV
?SCL2   ?FSTAT  77                      ;GET FILE STATUS
?SCL1   ?RLSE   100                     ;RELEASE AN LDU OR MTV
?SCL1   ?RELEASE 100                    ;RELEASE AN LDU OR MTV
?SCL1   ?SLIST  101                     ;SET SEARCH LIST
?SCL1   ?GLIST  102                     ;GET SEARCH LIST
?SCL1   ?SYLOG  103                     ;MANIPULATE SYSTEM LOG
?SCL1   ?GBIAS  104                     ;GET BIAS FACTOR
?SCL1   ?SBIAS  105                     ;SET BIAS FACTOR
?SCL1   ?IHIST  106                     ;INIT HISTOGRAM
?SCL1   ?KHIST  107                     ;KILL HISTOGRAM
?SCL1   ?GNAME  111                     ;GET FULL PATHNAME
?SCL1   ?GCPN   112                     ;GET CONSOLE PORT NUMBER
?SCL1   ?SUSER  113                     ;CHANGE SUPERUSER STATUS
?SCL1   ?SACL   114                     ;SET A FILE'S ACL
?SCL1   ?GACL   115                     ;GET A FILE'S ACL
?SCL1   ?PNAME  116                     ;PROCESS NAME <-> PID
?SCL1   ?SUPRV  117                     ;SET UNIVERSAL PRIVILEGES
?SCL1   ?FLUSH  120                     ;FLUSH A SHARED PAGE TO DISK
?SCL1   ?GTACP  122                     ;GET ACP'S FOR A FILE
?SCL1   ?GLINK  124                     ;GET CONTENTS OF A LINK ENTRY
?SCL1   ?GPRNM  125                     ;GET A PROCESS' PROGRAM NAME
?SCL1   ?LOGEV  126                     ;LOG AN EVENT IN SYSTEM LOG
?SCL1   ?DADID  127                     ;GET A PROCESS' FATHER'S PID
?SCL1   ?CPMAX  130                     ;SET CONTROL POINT DIR MAX SIZE
?SCL2   ?GNFN   131                     ;GET NEXT FILE NAME FROM DIR
?SCL1   ?SPY    132                     ;MAP PERFORMANCE DATA TO USER
?SCL1   ?RDUDA  133                     ;READ USER DATA AREA
?SCL1   ?WRUDA  134                     ;WRITE USER DATA AREA
?SCL1   ?CRUDA  135                     ;CREATE USER DATA AREA
?SCL1   ?ASCFL  136                     ;ASSOCIATE A FILE
?SCL1   ?DSCFL  137                     ;DISASSOCIATE A FILE
?SCL1   ?GNAFN  140                     ;GET NEXT ASSOCIATED FILENAME
?SCL1   ?SATR   141                     ;SET FILE ATTRIBUTES
?SCL2   ?IPC.R  142                     ;IPC SEND/RECEIVE
?SCL2   ?BRKFL  143                     ;BREAK FILE
?SCL1   ?SESL   144                     ;ENABLE A SYNCHRONOUS LINE
?SCL1   ?SDSL   145                     ;DISABLE A SYNCHRONOUS LINE
?SCL1   ?SSND   146                     ;START A SEND ON A SYNC LINE
?SCL1   ?SRCV   147                     ;START A RECV ON A SYNC LINE
?SCL1   ?SPOL   150                     ;DEFINE A POLL/SELECT LIST FOR A
```

```
?SCL1    ?SEPL    151                 ;MULTI-CPOP SYNC LINE
?SCL1    ?SDPL    152                 ;ENABLE A M.D. TERM FOR POLLING
?SCL1    ?SGES    153                 ;DISABLE A M.D. TERM FROM POLLIN
                                      ;GET SYNC LINE ERROR STATISTICS
```

```
; DEFINE GHOST CALLS

100000 .DUSR   ?.OPEN= @LBGHO+400+0        ;OPEN
100001 .DUSR   ?.CLOSE= @LBGHO+400+1       ;CLOSE
100002 .DUSR   ?.READ= @LBGHO+400+2        ;READ
100003 .DUSR   ?.WRITE=@LBGHO+400+3        ;WRITE
100004 .DUSR   ?.TERM= @LBGHO+400+4.       ;TERMINATE GHOST CALLER
100005 .DUSR   ?.CHAIN=@LBGHO+400+5        ;CHAIN TO ANOTHER PROCESS
100006 .DUSR   ?.GPOS= @LBGHO+400+6        ;GET CURRENT FILE POSITION
100011 .DUSR   ?.GTMES=@LBGHO+400+9.       ;GET CLI MESSAGE
100012 .DUSR   ?.RETURN=@LBGHO+400+10.     ;RETURN TO CLI WITH MESSAGE
100013 .DUSR   ?.EMSG=  @LBGHO+400+11.     ;ERROR MESSAGE PROCESSOR
100014 .DUSR   ?.GCHR= @LBGHO+400+12.      ;GET CHARACTERISTICS
100015 .DUSR   ?.SCHR= @LBGHO+400+13.      ;SET CHARACTERISTICS
100016 .DUSR   ?.ASSIGN=@LBGHO+400+14.     ;ASSIGN DEVICE
100017 .DUSR   ?.DEASSI=@LBGHO+400+15.     ;RELEASE DEVICE
100020 .DUSR   ?.SEND= @LBGHO+400+16.      ;SEND MESSAGE
100021 .DUSR   ?.ENQUE=@LBGHO+400+17.      ;ENQUEUE REQUEST
100025 .DUSR   ?.ABORT=@LBGHO+400+21.      ;ABORT REQUEST
100026 .DUSR   ?.SPOS= @LBGHO+400+22.      ;SET FILE POSITION
100024 .DUSR   ?.GTEST=@LBGHO+400+20.      ;TEST ENTRY FOR DISPATCHER
100027 .DUSR   ?.OVOPN=@LBGHO+400+23.      ;SHARED  OVERLAY  OPEN
100030 .DUSR   ?.SEOF= @LBGHO+400+24.      ;INTERNAL SET EOF CALL
100032 .DUSR   ?.OPR=  @LBGHO+400+26.      ;COMMUNICATE WITH OPER
100033 .DUSR   ?.PROC= @LBGHO+400+27.      ;PROC CALL
100034 .DUSR   ?.LOPEN=@LBGHO+400+28.      ;?LOPEN CALL
100035 .DUSR   ?.EXEC= @LBGHO+400+29.      ;?EXEC CALL
100036 .DUSR   ?.CTOD= @LBGHO+400+30.      ;?CTOD CALL
100037 .DUSR   ?.CDAY= @LBGHO+400+31.      ;?CDAY CALL
100040 .DUSR   ?.CEO=  @LBGHO+400+32.      ;?CEO CALL(INTERNAL)
```

TABLE 5.10

```
;
;USER RUNTIME CALLS
;

?TCL2    TASK     ;DEFINE A TASK OR TASKS
?TCL1    INTSK    ;INITIATE STASK MANAGER
?TCL1    SUS      ;SUSPEND CALLER
?TCL1    PRI      ;CHANGE CALLER'S PRIORITY
?TCL1    KILL     ;KILL CALLER
?TCL1    KILAD    ;DEFINE KILL PROCESSING ADDRESS
?TCL1    PRRDY    ;READY BY PRIORITY
?TCL1    PRSUS    ;SUSPEND BY PRIORITY
?TCL1    PRKIL    ;KILL BY PRIORITY
?TCL1    IOSTA    ;STATUS BY I.D.
?TCL1    IORDY    ;READY BY I.D.
?TCL1    IOSUS    ;SUSPEND BY I.D.
?TCL1    IOKIL    ;KILL BY I.D.
?TCL1    IOPRI    ;CHANGE PRIORITY BY I.D.
?TCL1    IOGOTO   ;FORCE A TASK TO A NEW P.C. BY I.D.
?TCL1    GMSK     ;GET SYSTEM MASK
?TCL1    LEFE     ;ENABLE LEF MODE
?TCL1    LEFD     ;DISABLE LEF MODE
?TCL1    LEFS     ;SAMPLE LEF MODE
?TCL1    XMT      ;XMIT A TASK MESSAGE
?TCL1    XMTW     ;XMIT A TASK MESSAGE AND WAIT FOR ITS RECEIPT
?TCL1    REC      ;RECEIVE A TASK MESSAGE
?TCL1    ERSCH    ;ENABEL TASK RESCHEDULING
?TCL1    DRSCH    ;DISABLE TASK RESCHEDULING
?TCL1    TRCON    ;READ FROM CONTROLLING CONSOLE
?TCL1    IOPCOM   ;INITIATE OPCOM
?TCL2    DQTSK    ;DEQUEUE A QUEUED TASK
?TCL1    OVLOD    ;VANILLA LOAD AN OVERLAY
?TCL1    OVEX     ;EXIT FROM AND RELEASE VANILLA OVERLAY
?TCL1    OVREL    ;RELEASE A VANILLA OVERLAY
?TCL1    OVKILL   ;RELEASE VANILLA OVERLAY AND ?KILL CALLING TASK
?TCL1    IXMT     ;SEND MESSAGE FROM INTERRUPT ROUTINE TO TASK
?TCL1    IMSG     ;RECEIVE MESSAGE FROM INTERRUPT ROUTINE
?TCL1    IXIT     ;EXIT FROM USER I/O INTERRUPT SERVICE ROUTINE
?TCL1    IFPU     ;INITIALIZE THE FLOATING POINT UNIT
```

REV.

SECTION 6
SYSTEM PANICS

A catastrophic failure in either one of the system data bases or in a critical hardware unit will cause a "Fatal System Error", commonly called a "Panic". Table 6-Ø-1 lists the various type Panics by "octal code #" along with a brief description and a page reference where more indepth information can be found.

I. General Overview

System Panics are placed through both the resident core sections of the AOS operating system and the system overlays. They are there to detect a critical failure at the earliest moment and to give information with which the cause of the failure can be determined. The Assembly language format for a "Panic" is:

```
JSR @.PNIC
Code
```

This is basically a "JSR" through a Page Zero variable to a routine called "PANIC". This routine stores away the accumulators at the time of the failure for printing out later. It also initially does an "INTDS" and later an "IORST". The "Fatal AOS Error ...." message is typed out, as shown in the next section, and the routine finally "JMP's" to another routine called "DUMPR" so that a core dump can be taken.

The "Fatal AOS Error ...." message will be one of two types. These are:

1) Panic code and ACØ thru AC3

2) Panic code, ACØ thru AC3, and C+PC

Type 1) will be used for all Panics except Panic code 7 and Panic code 1Ø. These are special cases since in actuality, they are hardware map violations, ie: "Traps", of the AOS Operating System or the initial CLI, respectively. These messages contain the contents of the "Return Block" at the time of the "trap" rather than the contents of the accumulators at the time of "Panic".

PANIC DEFINITION TABLE

OCTAL Code #	REVISION/PAGE # 1.0	BRIEF DEFINITION
1	6-7	System or Intr Stack fault
2	6-16	Master Disk data Error
3	X	Master Disk timeout
4	6-33	Unclearable, undefined interrupt
5	6-36	Logical error in Swap File Map content
6	6-41	Fatal file system data base error
7	6-50	AOS trapped
10	6-71	AOS Process trapped (OP-CLI)
11	6-80	Database checksum was bad
12	6-83	PMGR trapped
13	6-96	Multiple bit ERCC error
14	6-99	Internal consistency error
15	X	Power Failure

"X" indicates that the "Panic" code does not presently exist

"Table" Reference Table 6-Ø-3

LOC. DATA
REG PANIC CODE
REG+1 AC0
REG+2 AC1
REG+3 AC2
REG+4 AC3
REG+5 FC + CARRY
IF APPLICABLE

"Figure" Reference Table 6-Ø-2

"Figure" Reference Table 6-Ø-2

Because of the limited number of Panic codes that are used and the common error paths within codes, many of the software type error codes can have multiple causes. This is where AC3 becomes useful. Since a "JSR" is done in order to get to the "Panic" routine, AC3 will point to the area in core, and consequently the routine where the failure was caught. The only exception to this rule are a Panic code 7 and a Panic code 1Ø, since AC3 will contain the value at the time of "Trap".

It was mentioned before that a Panic can occur from either core resident code or from an overlay. Since AOS executes in a mapped mode, it uses a window type feature to access overlay code. Overlays are loaded into core beginning at $1K_8$ boundaries and referenced using logical addresses beginning at 74ØØØ or 75ØØØ. For example, word Ø of overlay #n would be referenced as 74ØØØ and word 35 of overlay #m would be referenced as 75Ø35. What this means is that in the cases mentioned in the previous paragraph where AC3 was meaningful, if AC3 contains a number of the format 74XXX or 75XXX, the "JSR" to the "PANIC" routine occurred from an overlay. If not, the "JSR" occurred from core resident code.

Within the indepth Panic analysis, AC3, when meaningful, will be used as a "Key" to seperate the different error paths which use the same Panic code. A format such as "AC3=INTS+" will be used. This means that if the numeric value contained in AC3 is referenced via a Symbol Table using SYSDMP, the symbol closest to and less than the actual numeric value would be "INTS". In actuality, the value would be "INTS+37" but since the offset may vary from revision to revision, I will not give it except when the "JSR" will come from an overlay. In this case, the offset can be used to determine which overlay the "JSR" occurred from. The module name in which the "JSR" resides will be listed on the right within parenthesis such as "(SGSUB)".

II. Panic Message/Core Dump Procedures

When a Panic occurs on a system, a message in one of two formats will be typed on the master console. These two formats are:

FATAL AOS ERROR = code ACØ AC1 AC2 AC3

or

FATAL AOS ERROR = code ACØ AC1 AC2 AC3 C+PC

Record the above information in a log or on a sheet of paper. After either of the above messages are typed out, a second set will be typed which will allow a core dump to be taken. The format is:

AOS CORE DUMP
LOAD TAPE FOR DUMPING ON MTAØ
STRIKE ANY KEY WHEN READY

At this point, a Mag Tape is mounted on Unit Ø of the primary Mag Tape Controller, device code 22. Note that the tape must have a "write ring" in it. The tape drive is loaded and put on-line. A key on the console is now pressed and the core dump will start. The core dump is written into file Ø of the Mag Tape with each record being 2Ø48 bytes. Note that each record corresponds to a "page" of memory and is written in sequential fashion beginning with Page Ø continuing thru Page 377. At the completion of the core dump, an EOF is written on the tape, it is rewound, and the message typed:

AOS CORE DUMP COMPLETED

The program will now halt. If the core dump fails, a message such as:

FATAL ERROR
AOS CORE DUMP FAILED

The problem which caused the "fatal error" should be corrected. If the Mag Tape has not moved off BOT, the problem may be as simple as:

1) No write ring in the tape
2) The tape unit is not on-line
3) The tape unit is not Unit Ø of device code 22

If the Mag Tape has moved off BOT, possible problems could be:

4) Bad tape
5) Two units selected as Unit Ø
6) A problem in the drive itself

Note that if the Core Dump Routine halts due to an error status from the Mag Tape Controller because of either a bad tape or unit problem, at the time of halt:

AC2 = Mag Tape Status (DIA)

and the Mag Tape Status should also be displayed in the "data lights" on the Eclipse console, if not in monitor.

Correct the problem and/or switch tape drives or Mag Tapes. Then press continue on the CPU console. The first message, "AOS Core Dump ....", will be repeated and you may continue.

III. "Fixup" the Disk

After the core dump is complete, the system utility, Fixup, must be run on the disk prior to attempting a "re-boot". The procedures for running Fixup are explained in Section 3.

Fixup should be run with Verbosity=2, Report Closing Files=N, and the Error Log directed to LPA, LPB, or CONØ (if a Dasher Printer). Be aware that this log needs to be kept since FIXUP can delete disk files or disk directories if it finds a file pointer that it cannot resolve. If a file is found missing at a later date, the FIXUP printout can be checked for an error message against that file. A complete description of the FIXUP error messages is contained in the Operators Guide (Ø93-194) in Appendix D around Page D-9 for a Rev. 2 manual and Section 3, "Fixup".

IV. General Core Dump Analysis

The disk can now be re-booted and the core dump can be loaded on the disk for analysis. This can be done using the following CLI commands:

)DIR :UTIL

)Copy DUMPmmdd @MTAØ:Ø

where mm = the current month, ie: May =Ø5

dd = the current day, ie: Ø3

This dating procedure allows for more than one dump to be saved on a disk and will keep the confusion of which dump belongs to which information to a minimum. Remember to record the above file name, DUMPmmdd, on the sheet of paper where the "Fatal

AOS Error ....." information was recorded.

The dump can now be analyzed using the SYSDMP or DEDIT system utilities which are described in section 7. Indepth analysis of each panic type can be used at this point by referencing through Table 6-Ø-1.

V. Core Dump Submission

In some instances, there is only so much information that can be gleamed from a core dump in the field. When additional information is needed, a core dump will be requested by National Tech. Support. Along with the core dump in file Ø of the Mag Tape, the following information is needed:

a) "Panic" type core dumps
   1) The Operating System Symbol Table (XX.ST) and the Operating System (XX.SY) dumped into file 1 of the Mag Tape.
   2) The output of the FIXUP program using Verbosity = 2
   3) The Panic message contents.
   4) Repeatability of the problem
   5) Activity of the system at the time of Panic
   6) Revision of AOS
   7) Hardware configuration

b) "Other" types of core dumps
   1) Same
   2) Same
   3) Any error messages to any console
   4) Same
   5) Same
   6) Same
   7) Same
   8) If the system stopped itself or was manually stopped, include also:
      a) PC when stopped
      b) ACØ thru AC3 contents
      c) Map, Ion, C light conditions

Note: If you are stopping the system because it is "hung" press only HALT. Do not press RESET until after recording the above values.

9) Whether "CRTL-C" is echoed on the master console.

Note that a core dump can be taken at any point by stopping the system, putting 14 in the switches, press "Reset" and "Start". The "AOS Core Dump ..." message will be displayed on the CRT. Refer to note in Step 8 above.

PANIC CODE 1 - STACK OVERFLOW

General Information:

On an Eclipse type machine, the stack overflow is triggered by the hardware. The following locations are used by the Eclipse and AOS:

Location 4Ø - Stack Pointer (SP)
41 - Frame Pointer (FP or CSP)
42 - Stack Limit (CSL)
43 - Stack Fault Address (CSO)

If the next "PUSH", "SAVE", or nested interrupt causes the Stack Pointer to exceed the stack limit value, a stack overflow type Panic will occur.

The System Stacks which are used with nested subroutines are a different area of core from the Interrupt Stack which is pointed to by the variable "SS".

When dealing with interrupts specific locations are used by the Eclipse and AOS to store the interrupt stack pointers. These are:

Location 4 - Interrupt Stack Pointer (ISP)
6 - Interrupt Stack Limit (ISL)
7 - Interrupt Stack Fault Address (ISO)

On the first level of interrupt, these values are moved to locations 4Ø, 42 and 43 respectively. On each successive level of interrupt, the contents of locations 4Ø and 41 are increased. If the contents of location 4Ø exceeds the contents of location 42, an interrupt stack overflow occurs.

Specifics:

AC3 = INTS+ (INTS) Either an interrupt or a system stack overflow has occurred. To determine which, either of the following checks may be made:

1) Check to see if the interrupt level counter, (INTLV) = -1. If it is -1, we are not at interrupt level.

At the time of Panic:
AC2 - contains the old contents of AC3.

AC3 = SSOVF (SGSUB) This routine used to be used to handle system stack overflows but it appears that it is not used anymore. The "INTS" routine handles both types. At the time of Panic:

AC2 = CSP

Discussion:

When discussing stack overflows, there are three general cases that most failures will fall into:

a) There was a valid stack overflow because of an excessive number of "PUSH's" due to legitimate nested subroutines or interrupts. This can only be determined by tracing through the various frames on the respective stack.

b) There was a valid stack overflow but the cause was due to a software or a hardware failure such as an interrupt with device code Ø from an ALM because of a broken priority chain.

c) The stack overflow is invalid and is caused by a failure such as a hardware memory failure and "JMPing" into the error handler without getting an actual hardware fault.

Cases a) and b) can be differentiated from case c) by comparing the SP with the CSL. If SP is close to CSL, say ±1Ø, the overflow is probably valid but we have to check for the cause. This can be done by examining the stack to see if the values appear valid, ie: "Return" or "Interrupt" blocks.

Case b) can be narrowed down a little further by checking the contents of a variable called PCNT. This location is incremented everytime an interrupt from device code Ø occurs. If this counter overflows, a Panic Code 4 will occur but it takes 65K "INC's to do it. More than likely, a Panic Code 1 will occur much before. The contents of this counter may give an indication whether interrupts from device code Ø are a problem.

Example:

Figure 6-1-1 shows a sample Panic Code 1 Typeout and the procedure to load the core dump from Mag Tape onto the disk. It assumes that you are familiar enough with the DEDIT commands to follow the procedure. Basically the "train of thought" behind the analysis was:

a) Determine whether it was a system or interrupt stack overflow? Well, INTLV = 1, the 2nd level of nesting.

b) Was the overflow valid? Well, SP = 1765 and CSL = 1Ø1757, therefore since SP - CSL (disregard bit Ø), the overflow is valid.

c) Was the cause of the overflow valid? Looking at the stack frames, the 1st level appears valid, the 2nd level looks a little strange (not necessarily bad), but there appears to be additional "PUSH's" from there to the end of the stack (not shown) which are very similar. The "PC" for each "Return Block", x1762Ø, is shown with the mneumonics for the code. I would be very suspicious at this point of a Case b) problem.

d) Although not shown in the example the contents of PCNT are 367.

With all these facts in mind, I would start looking for a device that could give me interrupts with a device code of Ø. In actuality, the failure was caused by a broken INTP/DCHP chain from the expansion chassis of an Eclipse to a Comm chassis with an ALM-8 connecting to the Extended I/O Bus. Be aware of the characteristics of this type failure.

Indepth AOS Information:

In the more general previous discussions, it was mentioned that the AOS system stacks are different from the interrupt stack. The various types of stacks that are used by AOS are:

a) Interrupt Stack(SS)

b) User Process Stack (STK1)

c) System Stacks which are pointed to by the PSØØØ Table

d) Core Manager Stack (CMSTK)

```
FATAL AOS ERROR - 1 061562 063531 075623 016605
                    AC0     AC1     AC2     AC3
AOS CORE DUMP
LOAD TAPE FOR DUMPING ON MTA0
STRIKE ANY KEY WHEN READY

AOS CORE DUMP COMPLETED
```

Sample Panic Code 1 Typeout

Figure 6-1-1

```
) DIR :UTIL
) COPY DUMP0414.6 0MTA0:0
) X DEDIT DUMP0414.6
AOS FILE EDITOR REV 1.0
+ STAB
FILENAME? :SYSGEN:AOS21.ST
+ 0:017620 +  PC last Interrupt
+ 5:177777 +     CMSK  (PFL mask)
+ SYSIN:000000 +
+ INTLV:000001 +  2nd level
+
+ 40:001765 +     SP
+ 41:000000 +     FP
+ 42:101757 +     SL
+ 43:016603 +     SO
+
+ 4:001560 +      ISP
+ 7:016603 +      ISO
+ 6:001757 +      ISL
+
+ SS=1560
+ SS:000000 +
SS+1 :001571 + 40
SS+2 :000000 + 41   Old Stack
SS+3 :001757 + 42     Values
SS+4 :016603 + 43
SS+5 :000000 + AC0
SS+6 :020037 + AC1
SS+7 :001602 + AC2
SS+10 :075623 + AC3
SS+11 :117620 + C+PC
SS+12 :000000 + Old mask
SS+13 :013642 + Old Loc 2
SS+14 :020037 + Old Map DIA
SS+15 :061562 + CUR60
SS+16 :063531 + CUR62
SS+17 :065741 + CUR64
SS+20 :067562 + CUR66
SS+21 :075623 + CUR74
SS+22 :000000 +
SS+23 :067562 +
SS+24 :001024 +
SS+25 :001006 +
SS+26 :065741 +
SS+27 :116661 +     2nd INTR (ALM)
SS+30 :000004 +
SS+31 :020037 +
SS+32 :001620 +     INTR
SS+33 :075623 +     PUSH
SS+34 :017622 +
SS+35 :000004 +
SS+36 :020037 +     INTR
SS+37 :001625 +     PUSH
SS+40 :075623 +
SS+41 :117620 +
SS+42 :000004 +
SS+43 :020037 +
SS+44 :001632 +     INTR
SS+45 :075623 +     PUSH
SS+46 :017620 +
SS+47 :000004 +
SS+50 :020037 +
SS+51 :001637 +     INTR
SS+52 :075623 +     PUSH
SS+53 :117620 +
SS+54 :000004 +
+   (etc)
+ 17615:103210 +     POP 0,0
+ 17616:062077 +     MSKO 0
+ 17617:040005 +     STA 0,cmsk
→+ 17620:020332 +    LDA 0,INTLV
+ 176213100014 +     COM# 0,0,SZR
+ 17622:000514 +     JMP .+114
+
+ BYE
)
```

1st INTR (RTC

Sample DEDIT Dump Analysis for Interrupt Stack Overflow
Figure 6-1-2

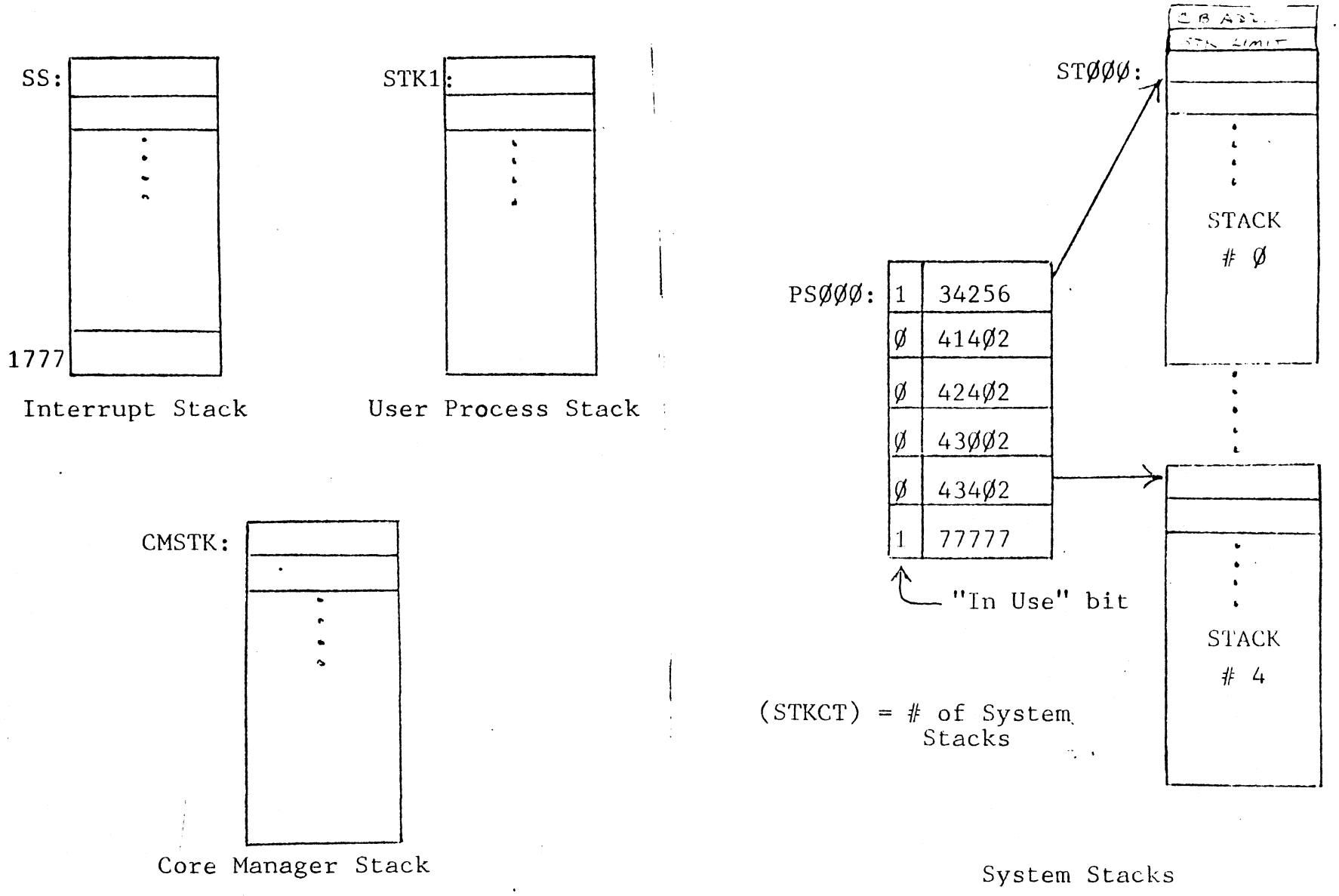


Figure 6.1.3 Various Types of Stacks Under AOS

These are illustrated in Figure 6.1.3 on the following page. A short discussion of each type of stack will follow:

a) Interrupt Stack - This stack is pointed to by the contents of the variable SS and physical location 4. This stack will use any remaining space between SS and the top of the 1st memory page (1777).

b) User Process Stack - This stack is shared among all the Processes. It is used by System Calls from the User and AOS that will not "pend" once the stack is assigned. A System Call that needs to access a non-resident overlay, could not use this stack because it would have to wait for the overlay to be loaded.

c) System Stacks - The number of these stacks is fixed at 5 and is not a "AOSGEN" parameter. These stacks can be used by System Calls, etc, that may be "pended" waiting for I/O to complete. The last stack, STACK #4, is reserved for a special purpose. It can only be used to process the ?ISEND system call. This avoids a system deadlock where Processes may be waiting on a ?IREC, but the ?ISEND can never be issued because no system stack is available. The linkage and format is similar to RDOS.

d) Core Manager Stack - The Core Manager, who controls the allocation and de-allocation of pages of core to Users and the Operating System, uses this stack for his nested subroutine calls.

The current stack that is being used or the last stack that was used is pointed to by the contents of physical locations 4Ø-43.

The layout of the System Stacks will be very different from the layout of the interrupt stack. Figure 6.1.4 shows the layout of a Sample Stack assuming the nested sub-routines only do "SAVE's" and "RETURN's". Low addresses are used to make it easier for you to follow the relationship between the Stack Pointer (SP) and the Frame Pointer (FP). By understanding this relationship, you can "layout" a system stack sub-routine by sub-routine and actually follow the progression of the path through the code.

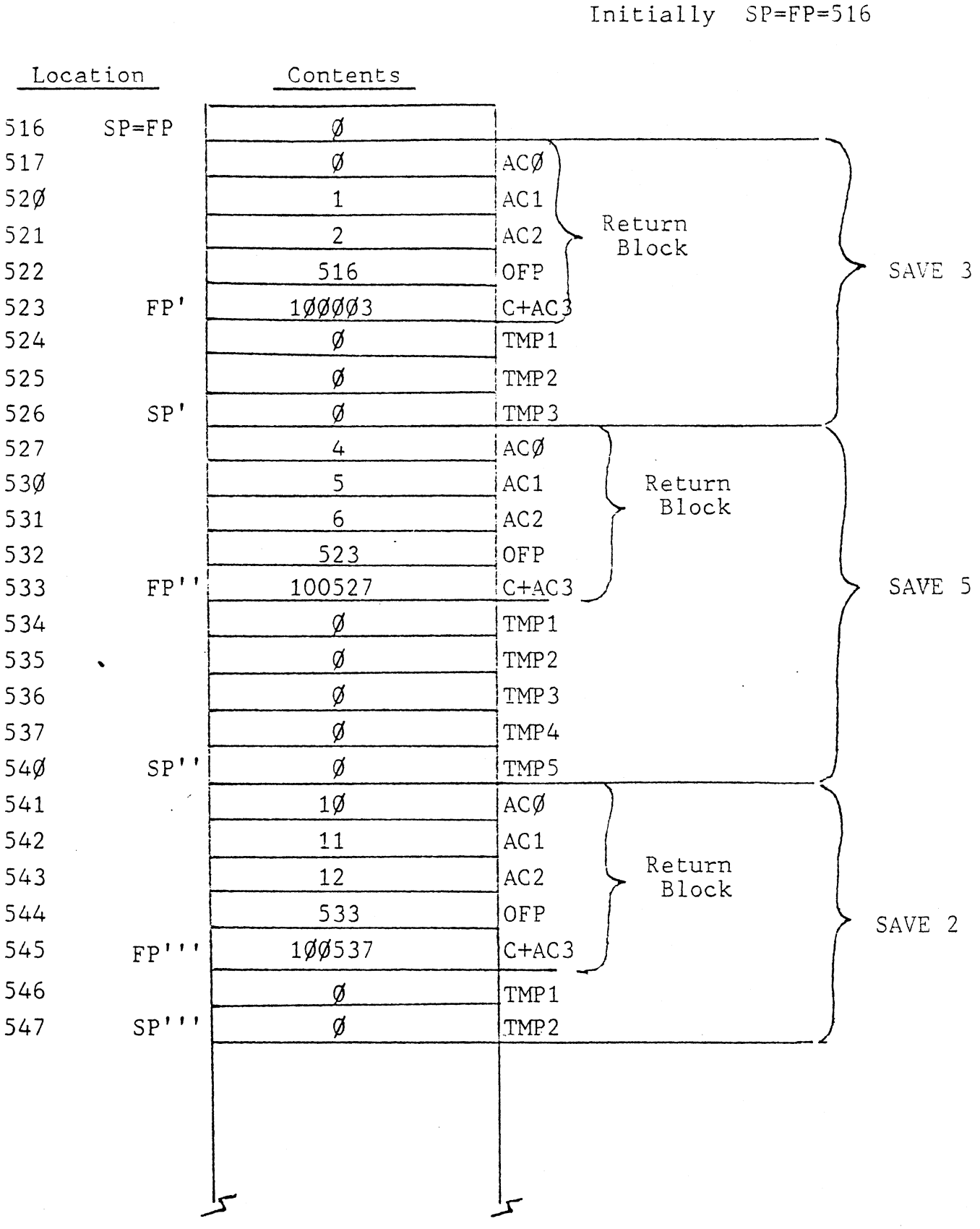


Figure 6.1.4 Sample Stack Layout for FP vs SP Relationships

The layout of the Interrupt Stack has basically four different types of layouts. These are:

1) Interrupt occurred from a system device
2) Interrupt occurred from a User ?IDEF'd device
3) A "SCL" call was executed
4) A "SAVE" was done by a subroutine while at interrupt level.

Types 1--3 are illustrated in Figure 6.1.5. Type 4 is the same as the "SAVE"/"RETURN" sequence illustrated for the System Stacks and will not be shown again.

Be aware that any combination of the above types may be mixed on the same stack due to nested interrupts and service routines.

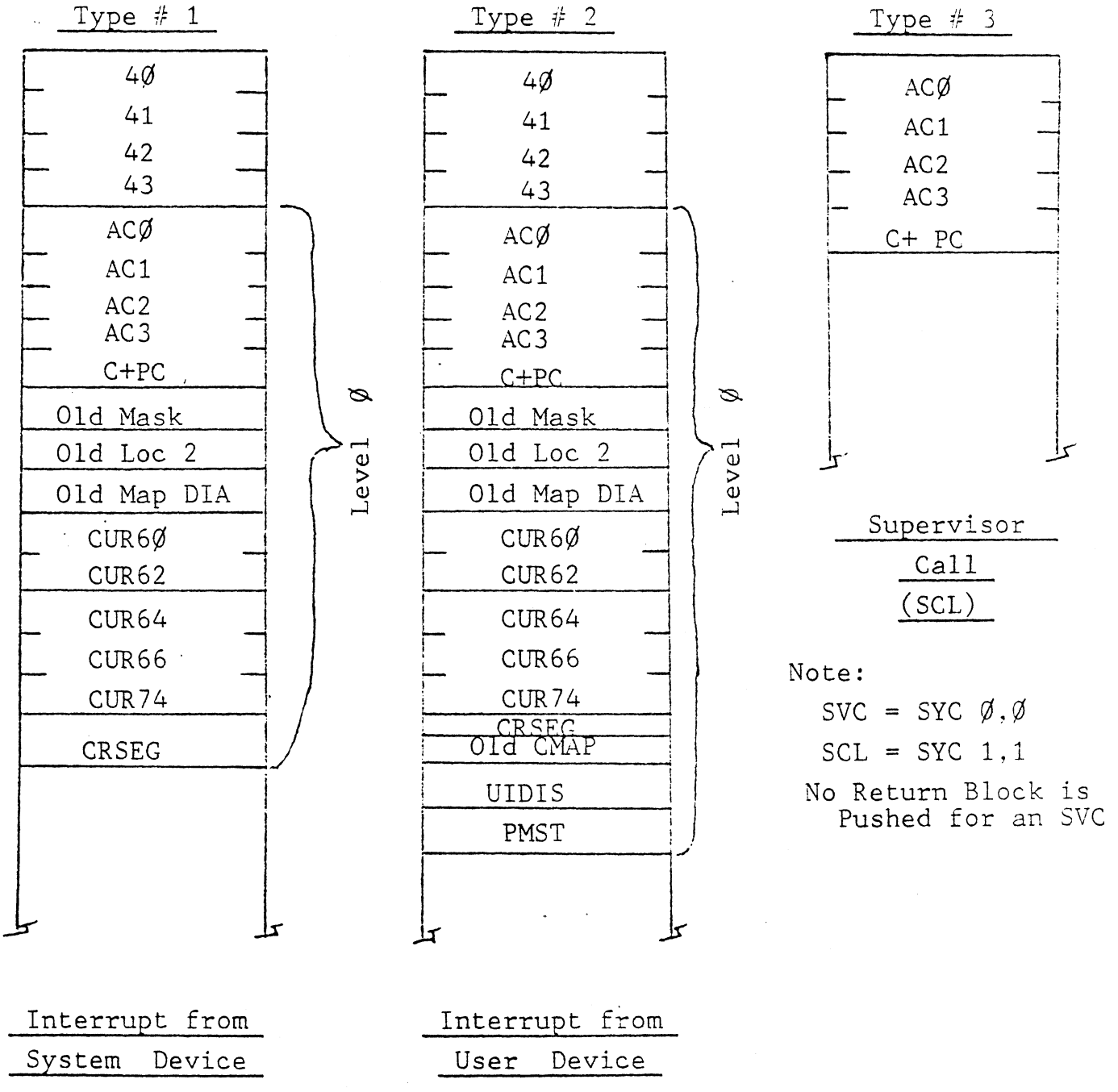


Figure 6.1.5 Interrupt Stack Layout Examples

PANIC CODE 2 - MASTER DISK DATA ERROR OR TIMEOUT

General Information:

a) Data Error - This Panic is caused by a non-recoverable master device disk error, ie: checkword, address, data late, etc., that has been retried $7_{(8)}$ times without success up to Rev. 1.Ø3; $15_{(8)}$ from Rev. 1.Ø4 on.

b) Timeout - This Panic is caused by the non-response, ie: interrupt, of the master disk within the $3_{(8)}$ second time-out period.

Specifics:

AC3 = OVDEQ+ (SOVLY) A disk operation had been started, and a "JSR" to a subroutine BWAIT was done to "Wait on Buffer I/O" completion. A fatal error occurred on the operation, which caused the Panic. At the time of Panic:

ACØ = "unpend" Address
‡ AC1 = UDB Address
AC2 = Buffer Header Address
(365)+16 = Error code* (365 = CC)
(AC1)+26 = Disk Status**
(AC1)+27 = System Status Word
(where bits 13-15 = # retries)

‡ Note: Use the given UDB address in AC1 in Sample Analysis #1 to find needed information.

The following three error "JSR's" will be identical in nature.

AC3 = 741Ø3,751Ø3 (SWAPI overlay) - SWIN+1Ø3
AC3 = 74Ø46,75Ø46 (SWAPO overlay) - SWOUT+46
AC3 = 74231,75231 (SWAPM overlay) - SWIAB+231

The above overlays are used to read in and swap out the shared portion of a swapped out process, and the CCB's and unshared portion. On one of the above disk operations, a fatal error occurred. At the time of Panic:

AC0 = 0
AC1 = (will vary)
AC2 = Buffer Header Address
or 0
(365)+16 = Error code* (365 = CC)
(OVUDB)+26 = Disk Status**
(OVUDB)+27 = System Status Word
(where bits 13-15 = # retries)

* Some common error codes are defined below with a complete list at the end of this chapter.

Error Code	Meaning
75	File Read Error
76	Device Timeout
121	Physical Unit Failure
123	Physical Unit Off-line

**Note that for a non-6066 (Zebra) type disk drive, the Disk Status will always be the DIA. For a Zebra, the Disk Status will either be the DIA or DIB depending on which one contains the relevant error bit. For a Fixed Head Disk, the DIC.

Discussion:

One of the problems in troubleshooting this type of problem is that AOS can start the next disk operation, if there is one waiting, prior to "Panicing". This will cause pointers to point to the wrong software table and makes getting the needed information more difficult. In the "Example" section, I will present a couple of methods to get the needed information. None of the methods, except in very specific circumstances, are "foolproof". If you are having problems, give me a phone call, and I will help you to get the needed information.

Some additional "tips" are following:

1. If the problem is very intermittent and cannot be found by diagnostics, run FIXUP to clean up the disk, and have the customer run using the SYSLOG facility to log any soft errors. FIXUP should be run with verbosity=2, Report Closing Files=N, and The Error Log directed to LPA, LPB or CON0 (if a Dasher Printer). Be aware that this

READ STATUS (DIA)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R/W DN	SEEK DONE				DKT	VS		UNS	RDY	SE	EOC	ADR	CE	DL	ERR

4234, 6030, 6045

4231 (3330)

READ STATUS (DIA)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
DP DONE	COMMAND DONE SEEK 0	SEEK 1	SEEK 2	SEEK 3	DUAL PROC	SECTOR ERROR	HEAD ERROR	ADR ERROR	DISC READY	SEEK ERROR	END ERROR	UN-SAFE	CHECK ERROR	DATA LATE	ERROR

READ DRIVE STATUS (DIB)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
INV ST	RES	TSP	RDY	BSY	OFF	WR DIS		ILL ADR	ILL CMD	DC FLT	UNS	POS FLT	CLK FLT	WR FLT	DRV FLT

READ DATA TRANSFER STATUS (DIA)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
CNT FUL	R/W DN	SEEK		DONE		PAR	SEC ADD	ECC	BAD SEC	CYL ADD	SEC SRF	VFY	R/W TIM	DAT LAT	R/W FLT

6060

MAGNETIC TAPE

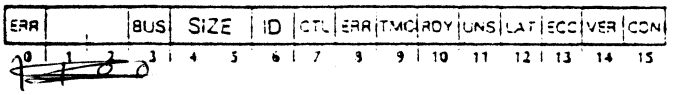
READ STATUS (DIA)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
ERROR	DATA LATE	RE-WIND-ING	ILL-EGAL	HIGH DEN-SITY	PARITY ERROR	EOT	EOF	BOT	9 TRACK	BAD TAPE	SEND CLOCK	FIRST CHAR.	WRITE LOCK	ODD CHAR	UNIT READY

FS-0016

TABLE 6.2.1 Disk and Mag Tape Status Word Breakdowns

Status (Word offset 3) or DIC



Fixed Head Disk

(6Ø63/6Ø65)

Bits	Name	Meaning when 1
0	Error	Bits 7,8,9,11,12,13, or 14 are 1 or bit 10 is 0.
1-2	—	Reserved for future use.
3	Bus Enable	In dual processor systems, the controller has access to the drive units. In single processor systems, always set to 1.
4-5	Size	Specifies the capacity of the selected drive as follows: 00 No drive 01 1Mbyte 10 2Mbyte 11 Reserved for future use
6	Idle Done	An idle function has been completed.
7-8	Ctrl Error	The controller has detected an error along data paths internal to the controller.
9	Sector Timeout	An extra sector pulse was detected or a sector pulse was missed.
10	Ready	The disc is ready for a data transfer.
11	Unsafe	A disc unit failure such as Power Not OK, or Speed Not OK. Disc Unsafe causes the Done flag to be set to 1 and requests an interrupt.
12	Data Late	The data channel did not respond in time to service the disc.
13	ECC	The ECC system detected an error.
14	Verify	The data verify system detected an error.
15	Command Done	A read, write, or data verify function has been completed.

TABLE 6.2.1 (Con't) Disk and Mag Tape
Status Word Breakdowns

log needs to be kept since FIXUP can delete disk files or disk directories if it finds a file pointer that it cannot resolve. If a file is found missing at a later date, the FIXUP printout can be checked for an error message against that file. A complete description of the FIXUP error messages is contained in the Operators Guide (Ø93-194) in Appendix D around Page D-9 for a Rev. 2 manual and Section 3 of this manual, "FIXUP".

2. If a disk drive goes "FAULT" (CDC), any error status is returned to the system and stored in UDB+26. It appears that the system issues a RECAL to the drive before it "Panics" because the drive "FAULT" indication is lost. "UNSAFE" will be set in the disk status.
3. Unfortunately if a "hard" disk error occurs, such as a checkword error, by the time the system "Panics", the contents of the temporary DOA and DOC storage areas have usually been overwritten so we do not know where the error is on the disk. The disks respective Reliability can be run in "Read-only" mode to find the bad spot but be very careful when you do this as a mistake will wipe the disk out. Verify 1st that the disk has been properly backed up. Do not do it if you are not sure what to do!!
4. To determine whether a disk error or timeout occurred, check the disk status word particularly for "Ready" and the Error Code. As a rule of thumb,

Code	Possibilities
75	Checkword error
121	Unsafe, timeout

These are the errors vs codes that I have seen so far. The list will probably grow as time goes on.

Examples:

There are two areas of core from which a Panic 2 can occur. These are the "Resident" section of code and an overlay area. Each of these requires a different troubleshooting technique and will be discussed seperately below.

I. Resident Code (AC3=OVDEQ+)

Sample Analysis #1:

Figure 6.2.1 shows a Panic code 2 typeout. Since AC3=16575, the "JSR" was from core resident code at OVDEQ+. Referring to the procedure at the beginning of the Panic Code 2 section, we find that the contents of the accumulators at the time of Panic were:

AC0 = 34104 (unpend address)
AC1 = 37200 (UDB address)
AC2 = 34353 (Buffer Header Address)

This type failure printout is the most straight forward to analyze of the group. AC1, the UDB address, is the key. Refer to Figure 6.2.2 which shows the analysis of this type of Panic. I first check SYSIN and INTLV. I next used the contents of CC to find the Error Code to be 121 - "Physical Unit Failure". Since the address of the UDB is given in AC1, I can check UDB+26 to find the "Disk Status" which is 2000. Note that the CDC DIA status is missing the "Ready" bit which is probably the reason for the panic. The system status word, at offset 27, indicated 0 retries and leads me to believe the drive "FAULT'd" either prior to the command, or the 1st time the command was issued. For your reference, this failure was caused by turning the master disk "off-line" and then issuing the CLI "DUMP" command.

II. Overlay Code (AC3=74xxx, 75xxx)

Figure 6.2.3 show a Panic code 2 typeout. Examining the typeout, we note that AC3=75231. Since AC3 is of the format 74xxx or 75xxx, the "JSR @.PNIC" occurred in an overlay. Comparing the address with the available possibilities, we find that the "JSR" was from the SWAPM overlay. Note also that both AC0 and AC1 equal 0.

Sample Analysis #2 (**Valid for Single Unit Disk System)

This procedure is valid only when the system overlay file resides on the same disk as the one which caused the panic. This will be true in single disk systems or multiple disk

```
FATAL AOS ERROR - 2 034134 037200 034333 016576
                    AC0    AC1    AC2    AC3
AOS CORE DUMP
LOAD TAPE FOR DUMPING ON MTA0
STRIKE ANY KEY WHEN READY

AOS CORE DUMP COMPLETED
```

Figure 6.2.1

Sample #1 Panic Code 2 Typeout

```
) DIR :UTIL
) COPY DUMP0514 @MTA0:0
) X SYSDMP DUMP0514
AOS SYSTEM DUMP ANALYZER REV 1.0
+ STAB
FILENAME? :SYSGEN:AOS22.ST
+ INTLV:177 777  +
+ SYSIN:000001 +
+
+ 365:034735 +
+ 34735+16:000121 +
+

+
+ 37200+26:002000 +
+ 37200+27:140000 +
+
+ BYE

)
```

Figure 6.2.2

Sample #I SYSDMP Analysis to find the Disk Error Status

systems where the Master Logical Disk comprises only one physical unit (or platter as in the case of a Diablo 44 or 6Ø45). Refer to Figure 6.2.4. I routinely check INTLV to see if we were at interrupt level and SYSIN to see if we were in the system. Following the analysis discussed for an overlay failure, I determine the contents of OVUDB to be 344ØØ. At offset 26, the disk status was 1Ø211l which for a 333Ø indicates an "UNSAFE". At offset 27, the system status word was 14ØØØØ which indicates Ø retries and leads me to believe the drive "FAULT'd" either prior to the command or the 1st time the command was issued. Examining the contents of CC (which is always location 365) and adding 16 to it we find the error code to be 121 - Physical Unit Failure.

Sample Analysis #3

Refer to Figure 6.2.5. Again I check SYSIN and INTLV. I used a slightly different method to find the UDB address. Since my problem was with a CDC (333Ø) which is on device code 33, I examine the system Interrupt Vector Table to find the Device Control Table (DCT). Note that ITBL is offset 1, therefore to check the DCT for device code 33, I examine ITBL+32. Offset ~~12~~ 2 of the DCT contains the address of the current UDB. (Remember that the DCT is on a controller basis). Since I now have the address of the UDB, I check offset 26 for the disk status of 1Ø21Ø5 which for a 333Ø is a Checkword Error. At offset 27, the system status word indicates that 7 retries were done. Using the contents of CC, I find the Error Code to be 75 - File Read Error. This technique will usually work. The exceptions are if there are disk operations waiting to be executed. AOS will attempt to start the next operation which could use a different physical disk and therefore a different UDB. By the time the panic occurs, the pointer in the DCT now points to the new UDB rather than the one that caused the panic.

If neither the procedures in Sample Analysises #1 or #2 provide the correct failing "Disk Status", the alternative remaining is to refer to the "Indepth AOS Information" in the following discussion and using the LCB/UDB pointers in Figure

```
FATAL AOS ERROR - 2 000000 034440 000000 075231
                        AC0     AC1     AC2     AC3
AOS CORE DUMP
LOAD TAPE FOR DUMPING ON MTA0
STRIKE ANY KEY WHEN READY

AOS CORE DUMP COMPLETED
```

Figure 6.2.3
Sample Panic Code 2 Typeout

```
) X SYSDMP DUMP0413
AOS SYSTEM DUMP ANALYZER REV 1.0
+ STAB
FILENAME? :SYSGEN:AOS20.ST
+ INTLV:177 77 7 +
+ SYSIN:000001 +
+ OVUDB:034400 +
+
+ 34400+26:102111 +
+ 34400+27:140000 +
+
+ 365:001123 +
+ 1123+16:000121 +
+
+ BYE

)
```

Figure 6.2.4
Sample #2 SYSDMP Analysis to find
the Disk Error Status

```
) X SYSDMP DUMP0412
AOS SYSTEM DUMP ANALYZER REV 1.0
+ STAB
FILENAME? :SYSGEN:AOS11.ST
+ INTLV:177 77 7 +
+ SYSIN:00 001 +
+ ITBL+32:132266 +
+
+ 32266+12:034400 +
+
+ 34400+26:102105 +
+ 34400+27:140007 +
+
+ 365:001123 +
+ 1123+16:000075 +
+
+ BYE

)
```

Figure 6.2.5
Sample #3 SYSDMP Analysis to find
the Disk Error Status

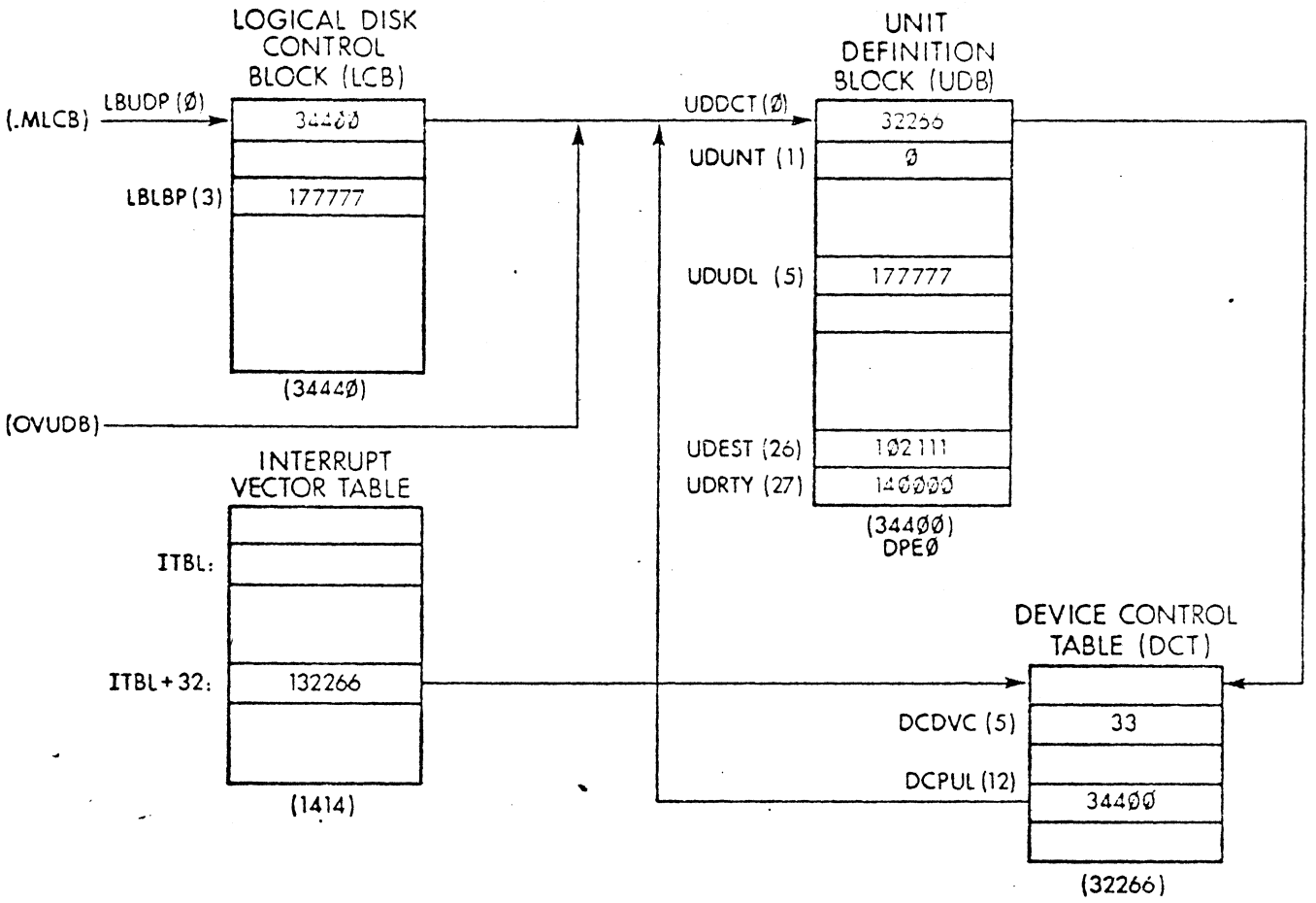
6.2.6 and Figure 6.6.2B, check offset 26 of each UDB to determine the one with the bad "Disk Status". Note that offset 1 of that UDB will indicate the "Unit #" of the drive. Note that on a 6Ø45 or a Diable 44, the removable packs are referenced as Units Ø-3, and the fixed surfaces as Units 4 - 7. Offset Ø of the UDB contains the address of the DCT. The contents of the DCT+5 is that controller's device code. Now we know the "Unit #" and the "Device Code", both of which uniquely define a hardware disk unit.

Indepth AOS Information:

The reason that two types of analyses were shown in Sample Analysis #2 and #3 is that there are cases when the method used in Analysis #2 will not point you to the proper UDB. In order for you to understand when Analysis #2 can be used, let us discuss the disk structure under AOS.

AOS defines a unit of disk storage as a Logical Disk (LD). Each Logical Disk is specified to the System Disk Formatter Utility, DFMTR, and may be comprised of multiple "spindles" for a 4231 or a 6Ø6Ø type disk or multiple disk surfaces for a 4234 or a 6Ø45 type disk. For example, a Logical Disk could be composed of Unit Ø and Unit 1 of a 6Ø6Ø Disk Subsystem for 192 Mbyte of storage or could be composed of Unit Ø (removable), Unit Ø (fixed), and Unit 1 (fixed) of a 6Ø45 Disk Subsystem for 15 Mbyte of storage. Each physical unit is defined to AOS by a Unit Definition Block (UDB), each disk controller is defined to AOS by a Device Control Table (DCT) and each Logical Disk (LD) is defined to AOS by a Logical Unit Control Block (LCB).

Let's examine a simple example. Figure 6.2.6 shows a system where there is only one Logical Disk initialized which is composed of only one 4231 disk unit, Unit Ø. The variable .MLCB will point to the first LCB on the chain. Since there is only one Logical Disk in our system, therefore, only one LCB, the link offset to the next LCB (LBLBP) will be a 177777. Offset Ø of the LCB points to the first UDB. Offset 5 of the UDB is the forward link with the next UDB. Again since there is only one



NOTE

LCBP WILL AT TIMES POINT TO THE CURRENT LCB OR WILL BE A 177777.

FS-0020

Figure 6.2.6 AOS Disk Table Linkage in a Simple one Unit System

disk unit, therefore, only one UDB, these links are 177777. For a device code 33 disk, offset ITBL+32 in the Interrupt Vector Table points to the DCT. Offset 12^2 of the DCT points to the UDB for the disk unit that contains the system overlays. In this case since there is only one disk unit both the current UDB in the DCT and OVUDB will point to the same UDB, therefore, either analysis method #2 or method #3 may be used.

If our Logical Disk now composed two CDC disk units, Unit Ø and Unit 1, OVUDB would always point to the disk unit with the system overlays, Unit Ø. If a fatal disk error occurred on Unit 1, OVUDB would be pointing to the wrong UDB but analysis method #3 will give us the correct UDB. Note also that if at the time of failure on Unit #1, there had been an outstanding request for Unit #Ø, the DCT UDB offset would probably be pointing to the Unit #Ø UDB. In this case both analysis methods #2 and #3 will give us the wrong UDB. At this point, we must check offset 26 of all the UDB's for the failing "Disk Status".

The conclusions that can be drawn from the above two examples are:

1) If your Logical Disk consists of only one physical unit, either analysis method #2 or method #3 will give you the correct UDB. Remember that a 4234 or a 6Ø45 is composed of two physical units as far as AOS is concerned.
2) If your Logical Disk consists of more than one physical unit, analysis method #3 will most times give the correct unit. Analysis method #2 may or may not give the correct unit depending on which unit the error occurred.

Table 6.2.2

```
;SYSTEM ERROR CODES  (1-277)

.DUSR ERICA=    1       ;ILLEGAL SYSTEM COMMAND
.DUSR ERFNO=    2       ;CHANNEL NOT OPEN
.DUSR ERCPR=    3       ;CHANNEL ALREADY OPEN
.DUSR ERSAL=    4       ;SHARED I/O REQ NOT MAP SLOT ALIGNED
.DUSR ERMEM=    5       ;INSUFFICIENT MEMORY AVAILABLE
.DUSR ERADR=    6       ;ILLEGAL STARTING ADDRESS
.DUSR EROVN=    7       ;ILLEGAL OVERLAY NUMBER
.DUSR ERTIM=    10      ;ILLEGAL TIME ARGUMENT
.DUSR ERNOT=    11      ;NO TASK CONTROL BLOCK AVAILABLE
.DUSR ERXMT=    12      ;SIGNAL TO ADDRESS ALREADY IN USE
.DUSR ERQTS=    13      ;ERROR IN QTASK REQUEST
.DUSR ERTID=    14      ;TASK I.D. ERROR
.DUSR ERDCH=    15      ;DATA CHANNEL MAP FULL
.DUSR ERMPR=    16      ;SYSTEM CALL PARAMETER ADDRESS ERROR
.DUSR ERABT=    17      ;TASK NOT FOUND FOR ABORT
.DUSR ERIRB=    20      ;INSUFFICIENT ROOM IN BUFFER
.DUSR ERSPC=    21      ;FILE SPACE EXHAUSTED
.DUSR ERSFT=    22      ;USER STACK FAULT
.DUSR ERDDE=    23      ;DIRECTORY DOES NOT EXIST
.DUSR ERIFC=    24      ;ILLEGAL FILENAME CHARACTER
.DUSR ERFDE=    25      ;FILE DOES NOT EXIST
.DUSR ERNAE=    26      ;FILE NAME ALREADY EXISTS
.DUSR ERNAD=    27      ;NON-DIRECTORY ARGUMENT IN PATHNAME
.DUSR EREOF=    30      ;END OF FILE
.DUSR ERDID=    31      ;DIRECTORY DELETE ERROR
.DUSR ERWAD=    32      ;WRITE ACCESS DENIED
.DUSR ERRAD=    33      ;READ ACCESS DENIED
.DUSR ERAWD=    34      ;APPEND AND/OR WRITE ACCESS DENIED
.DUSR ERNMC=    35      ;NO CHANNELS AVAILABLE
.DUSR ERSRL=    36      ;RELEASE OF NON-ACTIVE SHARED SLOT
.DUSR ERPRP=    37      ;ILLEGAL PRIORITY
.DUSR ERPMX=    40      ;ILLEGAL MAX SIZE ON PROCESS CREATE
.DUSR ERPTY=    41      ;ILLEGAL PROCESS TYPE
.DUSR ERCON=    42      ;CONSOLE DEVICE SPECIFICATION ERROR
.DUSR ERNSW=    43      ;SWAP FILE SPACE EXHAUSTED
.DUSR ERIDS=    44      ;DEVICE ALREADY IN SYSTEM
.DUSR ERDNM=    45      ;ILLEGAL DEVICE CODE
.DUSR ERSHP=    46      ;ERROR ON SHARED PARTITION SET
.DUSR ERRMP=    47      ;ERROR ON REMAP CALL
.DUSR ERGSG=    50      ;ILLEGAL GHOST GATE CALL
.DUSR ERPRN=    51      ;NUMBER OF PROCESSES EXCEEDS 64.
.DUSR ERMEF=    52      ;IPC MESSAGE EXCEEDS BUFFER LENGTH
.DUSR ERIVP=    53      ;INVALID PORT NUMBER
.DUSR ERNMS=    54      ;NO MATCHING SEND
.DUSR ERNOR=    55      ;NO OUTSTANDING RECEIVE
.DUSR ERIOP=    56      ;ILLEGAL ORIGIN PORT
.DUSR ERIOP=    57      ;ILLEGAL DESTINATION PORT
.DUSR ERSEN=    60      ;INVALID SHARED LIBRARY REFERENCE
.DUSR ERIPL=    61      ;ILLEGAL RECORD LENGTH SPECIFIED(=0)
.DUSR ERARC=    62      ;ATTEMPT TO RELEASE CONSOLE DEVICE
```

```
000063  .DUSR ERDAI=    63      ;DEVICE ALREADY IN USE
000064  .DUSR ERARU=    64      ;ATTEMPT TO RELEASE UNASSIGNED DEVICE
000065  .DUSR ERACU=    65      ;ATTEMPT TO CLOSE UNOPEN CHANNEL/DEVICE
000066  .DUSR ERIIC=    66      ;I/O TERMINATED BY CLOSE
000067  .DUSR ERLTL=    67      ;LINE TOO LONG
000070  .DUSR ERPAR=    70      ;PARITY ERROR
000071  .DUSR ERRXC=    71      ;RESIDENT PROC TRIED TO PUSH (.EXEC)
000072  .DUSR ERNDR=    72      ;NOT A DIRECTORY
000073  .DUSR ERNSA=    73      ;SHARED I/O REQUEST NOT TO SHARED AREA
000074  .DUSR ERSNM=    74      ;ATTEMPT TO CREATE > MAX # SPMS
000075  .DUSR ERFIL=    75      ;FILE READ ERROR
000076  .DUSR ERDTO=    76      ;DEVICE TIMEOUT
000077  .DUSR ERIOT=    77      ;WRONG TYPE I/O FOR OPEN TYPE
000100  .DUSR ERFTL=    100     ;FILENAME TOO LONG
000101  .DUSR ERBOF=    101     ;POSITIONING BEFORE BEGINNING OF FILE
000102  .DUSR ERPRV=    102     ;CALLER NOT PRIVILEGED FOR THIS ACTION
000103  .DUSR ERSIM=    103     ;SIMULTANEOUS REQUESTS ON SAME CHANNEL
000104  .DUSR ERIFT=    104     ;ILLEGAL FILE TYPE
000105  .DUSR ERNRD=    105     ;INSUFFICIENT ROOM IN DIRECTORY
000106  .DUSR ERILO=    106     ;ILLEGAL OPEN
000107  .DUSR ERPRH=    107     ;ATTEMPT TO ACCESS PROC NOT IN HIERARCHY
000110  .DUSR ERBLR=    110     ;ATTEMPT TO BLOCK UNBLOCKABLE PROC
000111  .DUSR ERPRE=    111     ;INVALID SYSTEM CALL PARAMETER
000112  .DUSR ERGES=    112     ;ATTEMPT TO START MULTIPLE GHOSTS
000113  .DUSR ERCIU=    113     ;CHANNEL IN USE
000114  .DUSR ERICS=    114     ;INSUFFICIENT CONTIGUOUS DISK BLOCKS
000115  .DUSR ERSTO=    115     ;STACK OVERFLOW
000116  .DUSR ERIBM=    116     ;INCONSISTENT BIT MAP DATA
000117  .DUSR ERBSZ=    117     ;ILLEGAL BLOCK SIZE FOR DEVICE
000120  .DUSR ERXMZ=    120     ;ATTEMPT TO XMT ILLEGAL MESSAGE
000121  .DUSR ERPUF=    121     ;PHYSICAL UNIT FAILURE
000122  .DUSR ERPWL=    122     ;PHYSICAL WRITE LOCK
000123  .DUSR ERPOL=    123     ;PHYSICAL UNIT OFFLINE
000124  .DUSR ERIOO=    124     ;ILLEGAL OPEN OPTION FOR FILE TYPE
000125  .DUSR ERNDV=    125     ;TOO MANY OR TOO FEW DEVICE NAMES
000126  .DUSR ERMIS=    126     ;DISK AND FILE SYS REV #'S DON'T MATCH
000127  .DUSR ERIUD=    127     ;INCONSISTENT DIR DATA
000130  .DUSR ERILD=    130     ;INCONSISTENT LD
000131  .DUSR ERICU=    131     ;INCOMPLETE LD
000132  .DUSR ERIUT=    132     ;ILLEGAL DEVICE NAME TYPE
000133  .DUSR ERPDF=    133     ;ERROR IN PROCESS UST DEFINITION
000134  .DUSR ERVIU=    134     ;LD IN USE, CANNOT RELEASE
000135  .DUSR ERSRE=    135     ;SEARCH LIST RESOLUTION ERROR
000136  .DUSR ERCGF=    136     ;CAN'T GET IPC DATA FROM FATHER
000137  .DUSR ERILB=    137     ;ILLEGAL LIBRARY NUMBER GIVEN
000140  .DUSR ERRFM=    140     ;ILLEGAL RECORD FORMAT
000141  .DUSR ERARG=    141     ;TOO MANY OR TOO FEW ARGUMENTS TO ERGR
000142  .DUSR ERIGM=    142     ;ILLEGAL SYSMES PARAMETERS
000143  .DUSR ERICL=    143     ;ILLEGAL CLI MESSAGE
000144  .DUSR ERMRD=    144     ;MESSAGE RECEIVE DISABLED
000145  .DUSR ERNAC=    145     ;NOT A CONSOLE DEVICE
000146  .DUSR ERMIL=    146     ;ATTEMPT TO EXCEED MAX INDEX LEVEL
000147  .DUSR ERICN=    147     ;ILLEGAL CHANNEL #
```

```
.DUSR ERNRR=    150     ;NO RECEIVER WAITING
.DUSR ERSRR=    151     ;SHORT RECEIVE REQUEST
.DUSR ERTIN=    152     ;TRANSMITTER INOPERATIVE
.DUSR ERILN=    153     ;ILLEGAL USER NAME
.DUSR ERILN=    154     ;ILLEGAL LINK #
.DUSR ERDPE=    155     ;DISK POSITIONING ERROR
.DUSR ERTXT=    156     ;MSG TEXT LONGER THAN SPEC'D.
.DUSR ERSTP=    157     ;SHORT TRANSMISSION
.DUSR ERHIS=    160     ;ERROR ON HISTOGRAM INIT/DELETE
.DUSR ERIRV=    161     ;ILLEGAL RETRY VALUE
.DUSR ERASS=    162     ;ASSIGN ERROR - ALREADY YOUR DEVICE
.DUSR ERPET=    163     ;MAG TAPE REQ PAST LOGICAL END OF TAPE
.DUSR ERSTS=    164     ;STACK TOO SMALL (?TASK)
.DUSR ERTMT=    165     ;TOO MANY TASKS REQUESTED (?TASK)
.DUSR ERSOC=    166     ;SPOOLER OPEN RETRY COUNT EXCEEDED
.DUSR ERACL=    167     ;ILLEGAL ACL
;       CODE 170 UNUSED
.DUSR ERIMP=    171     ;IPC FILE NOT OPENED BY ANOTHER PROC
;       CODE 172 UNUSED
.DUSR ERPNM=    173     ;ILLEGAL PROCESS NAME
.DUSR ERPNU=    174     ;PROCESS NAME ALREADY IN USE
.DUSR ERDCT=    175     ;DISCONNECT ERROR (MODEM CONTROLLED)
.DUSR ERIPR=    176     ;NONBLOCKING PROC REQUEST ERROR
.DUSR ERSNI=    177     ;SYSTEM NOT INSTALLED
.DUSR ERLVL=    200     ;MAX DIRECTORY TREE DEPTH EXCEEDED
.DUSR ERROO=    201     ;RELEASING OUT-OF-USE OVERLAY
.DUSR ERROL=    202     ;RESOURCE DEADLOCK
.DUSR ERFO1=    203     ;FILE IS OPEN, CAN'T EXCLUSIVE OPEN
.DUSR ERFO2=    204     ;FILE IS EXCLUSIVE OPENED, CAN'T OPEN
.DUSR ERIPD=    205     ;INIT PRIVILEGE DENIED
.DUSR ERMIM=    206     ;MULTIPLE ?IMSG CALLS TO SAME DCT
.DUSR ERLNK=    207     ;ILLEGAL LINK
.DUSR ERIDF=    210     ;ILLEGAL DUMP FORMAT
.DUSR ERXNA=    211     ;EXEC NOT AVAILABLE (MOUNT, ETC.)
.DUSR ERXUF=    212     ;EXEC REQUEST FUNCTION UNKNOWN
.DUSR ERFSO=    213     ;ONLY EXEC'S SONS CAN DO THAT
.DUSR ERPBO=    214     ;REFUSED BY OPERATOR
.DUSR ERMNT=    215     ;VOLUME NOT MOUNTED
.DUSR ERISV=    216     ;ILLEGAL SWITCH VALUE (>65K DECIMAL)

;THE NEXT FOUR ERROR CODES MUST BE CONTIGUOUSLY NUMBERED

.DUSR ERIFN=    217     ;INPUT FILE  DOES NOT EXIST
.DUSR EROFN=    220     ;OUTPUT FILE  DOES NOT EXIST
.DUSR ERLFN=    221     ;LIST FILE  DOES NOT EXIST
.DUSR ERDFN=    222     ;DATA FILE DOES NOT EXIST

.DUSR ERGFE=    223     ;RECURSIVE GENERIC FILE OPEN FAILURE
;       CODE 224 UNUSED
.DUSR ERNUD=    225     ;USER DATA AREA DOES NOT EXIST
.DUSR ERDVC=    226     ;ILLEGAL DEVICE TYPE FROM AOSGEN
.DUSR ERRST=    227     ;AOS RESTART OF SYSTEM CALL
.DUSR ERFUP=    230     ;PROBABLY FATAL HARDWARE RUNTIME ERROR
.DUSR ERNNC=    231     ;CHAIN DOES NOT EXIST
```

```
000232  .DUSR ERECC=    232     ;END OF CHAIN
000233  .DUSR ERUAE=    233     ;USER DATA AREA ALREADY EXISTS
000234  .DUSR ERCAE=    234     ;FILE ALREADY ON CHAIN
000235  .DUSR ERIAS=    235     ;ILLEGAL FILE ASSOCIATION
        ;       CODE 236 UNUSED
000237  .DUSR ERCPD=    237     ;CONTROL POINT DIRECTORY MAX SIZE EXCEED
000240  .DUSR ERNSO=    240     ;SYS OR ROOT DISK NOT PART OF MASTER LD
000241  .DUSR ERUSY=    241     ;UNIVERSAL SYSTEM, YOU CAN'T DO THAT
000242  .DUSR ERREAD=   242     ;EXECUTE ACCESS DENIED
000243  .DUSR ERFIX=    243     ;CAN'T INIT LD, RUN FIXUP ON IT
000244  .DUSR ERFAD=    244     ;FILE ACCESS DENIED
000245  .DUSR ERDAD=    245     ;DIRECTORY ACCESS DENIED
000246  .DUSR ERIAD=    246     ;ATTEMPT TO DEFINE > 1 SPECIAL PROC
000247  .DUSR ERIND=    247     ;NO SPECIAL PROCESS IS DEFINED
000250  .DUSR ERPRO=    250     ;ATTEMPT TO ISSUE MCA REQUEST WITH
                                ;DIRECT I/O IN PROGRESS
000251  .DUSR ERDIO=    251     ;ATTEMPT TO ISSUE MCA DIRECT I/O WITH
                                ;OUTSTANDING REQUESTS
000252  .DUSR ERLTK=    252     ;LAST TASK WAS KILLED
000253  .DUSR ERLRF=    253     ;RESOURCE LOAD OR RELEASE FAILURE
000254  .DUSR ERNAL=    254     ;ZERO LENGTH FILENAME SPECIFIED


        ;SYNCHRONOUS LINE ERROR CODES
000255  .DUSR ESBOF=    ERNAL+1 ;BUFFER OVERFLOW
000256  .DUSR ESNAK=    ESBOF+1 ;TRANSMISSION FAILURE (MAX COUNT)
000257  .DUSR ESTOF=    ESNAK+1 ;TRANSMISSION FAILURE (TIMEOUTS)
000260  .DUSR ESDIS=    ESTOF+1 ;DISCONNECT OCCURRED ON SWITCHED LINE
000261  .DUSR ESEOT=    ESDIS+1 ;EOT CHARACTER RECEIVED
000262  .DUSR ESOTH=    ESEOT+1 ;POSSIBLE LOST DATA ON HASP LINE
000263  .DUSR ESDCU=    ESOTH+1 ;DCU INOPERATIVE (CAN'T BE INITIALIZED)
000264  .DUSR ESCON=    ESDCU+1 ;CONVERSATIONAL REPLY RECEIVED
000265  .DUSR ESEPL=    ESCON+1 ;END OF POLLING LIST REACHED
000266  .DUSR ESIRT=    ESEPL+1 ;ILLEGAL RELATIVE TERMINAL NUMBER
000267  .DUSR ESRVI=    ESIRT+1 ;RVI RESPONSE RECEIVED
000270  .DUSR ESILN=    ESRVI+1 ;ILLEGAL LINE NUMBER
000271  .DUSR ESPLS=    ESILN+1 ;NOT ENOUGH SPACE FOR POLL LISTS
000272  .DUSR ESCTN=    ESPLS+1 ;CONTENTION SITUATION WHILE BIDDING
000273  .DUSR ESSEQ=    ESCTN+1 ;OUT-OF-SEQUENCE GEN ENTRY DURING SINIT
000274  .DUSR ESNSL=    ESSEQ+1 ;ATTEMPT TO ENABLE NON-SYNC LINE
000275  .DUSR ESIMM=    ESNSL+1 ;NOT ENOUGH MEMORY FOR POLL/SELECT LIST
000276  .DUSR ESEPE=    ESIMM+1 ;LINE ALREADY ENABLED ON ?SENL CALL
000277  .DUSR ESNSD=    ESEPE+1 ;LINE ALREADY DISABLED ON ?SDBL CALL
000300  .DUSR ESLNA=    ESNSD+1 ;I/O REQUEST FOR DISABLED LINE
000301  .DUSR ESLIS=    ESLNA+1 ;LINE IN SESSION ON ?SSND INITIAL CALL
000302  .DUSR ESSCS=    ESLIS+1 ;?SSND CONTINUE WITHOUT LINE IN SESSION
000303  .DUSR ESBCT=    ESSCS+1 ;SEND BYTE COUNT EXCEEDS SYSTEM BUFFER
000304  .DUSR ESRNK=    ESBCT+1 ;BID ERROR (TOO MANY NAKS)
000305  .DUSR ESNAB=    ESRNK+1 ;NAK RECEIVED (HASP LINE ONLY)
000306  .DUSR ESBPE=    ESNAB+1 ;USER BUFFER BYTE POINTER INVALID
000307  .DUSR ESRRT=    ESBPE+1 ;RETRY COUNT EXCEEDED
000310  .DUSR ESETX=    ESRRT+1 ; 'ETX' CODE RECEIVED
```

```
.DUSR ESISE=     ESETY+1 ;INPUT STATUS ERROR (FORMAT)
.DUSR ESFCT=     ESISE+1 ;FAILURE TO CONNECT
.DUSR ESUNI=     ESFCT+1 ;UNINTERRUPTABLE RESPONSE RECEIVED
.DUSR ESENO=     ESUNI+1 ;ENQ RECEIVED AFTER TIME-OUT
.DUSR ESCRC=     ESENO+1 ;CRC CHECK
.DUSR ESIPP=     ESCRC+1 ;INITIALIZATION PARAMETER ERROR
.DUSR ESTOF=     ESIPP+1 ;TRANSMITTER FAILURE ERROR

.DUSR ERITF=     ESTOF+1 ;INCOMPATIBLE LPR TAR FORMAT
.DUSR ERPRM=     ERITF+1 ;CANNOT DELETE PERMANENT FILE
.DUSR ERSCA=     ERPRM+1 ;SYSTEM CALL ABORT
```

PANIC CODE 3

**** Does not exist at this time ****

PANIC CODE 4 - UNCLEARABLE, UNDEFINED INTERRUPT

General Information:

The Panic Code 4 can be caused from two different places but both deal with interrupts. It is not always a hardware problem but can be a User procedural type problem. A more detailed analysis will be done under "Discussion".

Specifics:

AC3 = IUD+

(INTS) the two cases will be discussed seperately.

Case #1: When AOS gets an interrupt, it uses the Vector instruction to access the Interrupt Vector Table which is a list of Device Control Tables (DCT) by device code. If a specific device code has not been defined either at AOSGEN time or via the ?IDEF system call, the slot in the Vector Table points to an "Undefined Interrupt" routine, IUD. Within this routine an NIOC is issued to the device code and a SKPDN is also issued to see whether the interrupt was cleared. This sequence, NIOC/SKPDN is done a maximum of $2000_{(10)}$ times. If the interrupt has not cleared after the 2000th time, a counter goes to zero, and the Panic occurs. At the time of Panic:

AC1 = 0

AC2 = device code

Note: That INTLV has already been reset to its previous level.

Case #2: If AOS gets an interrupt with a device code of "0", it assumes

a "Power Fail" and vectors into the Power Fail Handler (PFL). Within the PFL routine, a "SKPDZ CPU" instruction is executed to check for a valid Power Fail. If no valid Power Fail exists, a counter (PCNT) is incremented (initially Ø), and the dismiss routine (DISI2) is entered. There is no way to attempt to clear an interrupt from device code Ø. If the interrupt is still there or occurs frequently enough, the above sequence is executed until the counter overflows (65K times). When that occurs, a Panic happens. At the time of Panic:

ACØ =
AC1 =
AC2 = PFLDC (Address of Power Fail DCT)
PCNT = Ø

Discussion:

1. It was mentioned previously about a software procedural problem which will cause this Panic. The basic sequence is that after certain Comm devices are initialized, if an interrupt from that device occurs, it cannot be clear'd by an NIOC thereby the system will Panic. The procedural problem that usually causes this, is when a User ?IDEF's the device, initializes it, then ↑C↑A (aborts) the program without deinitializing the device. The next interrupt from the device will cause the Panic. Devices, other than a User Widget, that can cause this problem and may be on an AOS system are:
    1. ALM
    2. SLM
    3. Digital I/O (56Ø2)

4. External Interrupt Module (4Ø67)
5. DCU/5Ø

2. I suspect that you will never see Case #2 on an AOS system. The reason being that the Interrupt Stack will probably overflow and cause a Panic Code 1 before PCNT will be incremented 65K times. The only exception to this would be if the system is getting very intermittent interrupts with a device code of Ø that cause PCNT to be incremented but not enough interrupts at any one time to cause the Interrupt Stack to overflow.

Examples:

None

PANIC CODE 5 - LOGICAL ERROR IN SWAP FILE MAP CONTENT

General Information:

An AOS system will allow more Users or Processes to be executing than will fit into core at any one time. The code that belongs to a given Process along with other information about that Process is "swapped" out to disk when the core it is using is needed by a higher priority Process. The "Swap File" size on disk can be defined at AOSGEN time and changed at system "Boot" time. Additionally, a complete disk unit can be dedicated as a swap device. When this is done, the disk unit cannot be used for any other purpose and should be a "fast" device. A bit map is maintained indicating which blocks of the "Swap File" are in use.

Specifics:

AC3 = 74336,75336 (SSOV3 overlay) - CSWP+32 There are two cases which will cause this Panic, both dealing with a discrepancy between the "bit map" and other "Swap File" pointers.

Case #1 - A free area within the "Swap File" has been found and an attempt is made to allocate it. An "SZBO" (skip on Zero bit and set to 1) instruction is executed and the skip does not occur. This indicates that even though the area is "free", the "bit map" indicates it as being in use. At the time of Panic:

AC0 = Negate of required size
AC1 = field address
AC2 = word/bit address } *

* Note that the above accumulators use the format of the 32 bit addressing of bits feature.

Case #2 - Blocks within the "Swap File" are being released for later use by another Process. An "SNB" (skip on non-Zero bit) instruction is executed and the skip does not occur. This indicates that the area to be released is not marked as in use in the "Swap File" bit map. At the time of Panic:

AC0 = -# of bits for the swap area
AC1 = word/bit address*
AC2 = buffer address containing the address of the Map Area start (BQADR)

Discussion:

There is not very much that I can give for troubleshooting tips as the problem could be disk or mainframe. The software analysis is much deeper than the scope of this document. I would suggest running EMORT L and doing a CRUNALL under DTOS. The core dump should also be submitted for further analysis.

Two thoughts can be kept in mind:

1. The Swap File bit map as shown in Figure 6.5.2 has to be read from disk into core if it is not presently there. If due to a hardware failure, the block is read into the wrong place in memory, the software will access wrong data and will Panic.
2. The Bit map, once in core, is accessed using Eclipse bit instructions, ie: SZBO and SNB. The usage of these instructions is defined in the Eclipse Line Programmers Reference Manual (015-24) on Page 2-8 and 3-14. If a bit is picked during the use of either instruction, the wrong area of the bit map could be accessed and the Panic will occur.

The above are two possible causes for the failure. There are

probably other causes which have yet to come to light. Be aware also that this is an obscure Panic which rarely occurs.

Example:

The following page shows an example of a Panic Code 5 Type-out, Figure 6.5.1. Note that AC3 is 75336 which points to the SSOV3 overlay. AC2 does contain a buffer address, therefore this is a Case #2 type failure.

Indepth AOS Information:

Figure 6.5.2 shows the layout of the System Swap File, "SWAP.SWAP". The first block of the Swap File contains a bit map for the remainder of the file where each bit represents a group of 4 blocks. If a bit is set to a "1" the group is "In Use", if "Ø" it is available for use. The reason a "bit map" is used for disk block allocation is that the number of disk blocks used by a Process will grow or shrink as the Process acquires more or releases memory respectively. In the case of a growing Process, the slot in the disk file where it had been previously swapped might not now be large enough. The slot may not be able to be made larger because other Processes are using the disk space on both sides of the slot. A new slot now has to be found which is large enough for the newly grown Process to fit. Bits are set in the bit map for the area used by the new slot and reset for the area being released by the old slot. A word in the Process Table, PSWBN, will point to the relative disk block being used by that Process Table. Note that each Page of core will use up 4 disk blocks, the reason for the grouping by four.

The Swap file size is defined at AOSGEN time and can be overridden at Boot-up time. You should never make the Swap File smaller than its original value since if disk space is used next to the new swap file, the original swap file size may not fit anymore.

System error code #43 is defined as "Swap File Space Exhausted". This can occur by either the Swap File actually running out of space or becoming so fragmented that a "new slot" as discussed above cannot be found.

```
FATAL AOS ERROR - 5 177774 000002 034160 075336
                          AC0    AC1     AC2     AC3
AOS CORE DUMP
LOAD TAPE FOR DUMPING ON MTA0
STRIKE ANY KEY WHEN READY

AOS CORE DUMP COMPLETED
```

Figure 6.5.1
Sample Panic Code 5 Typeout

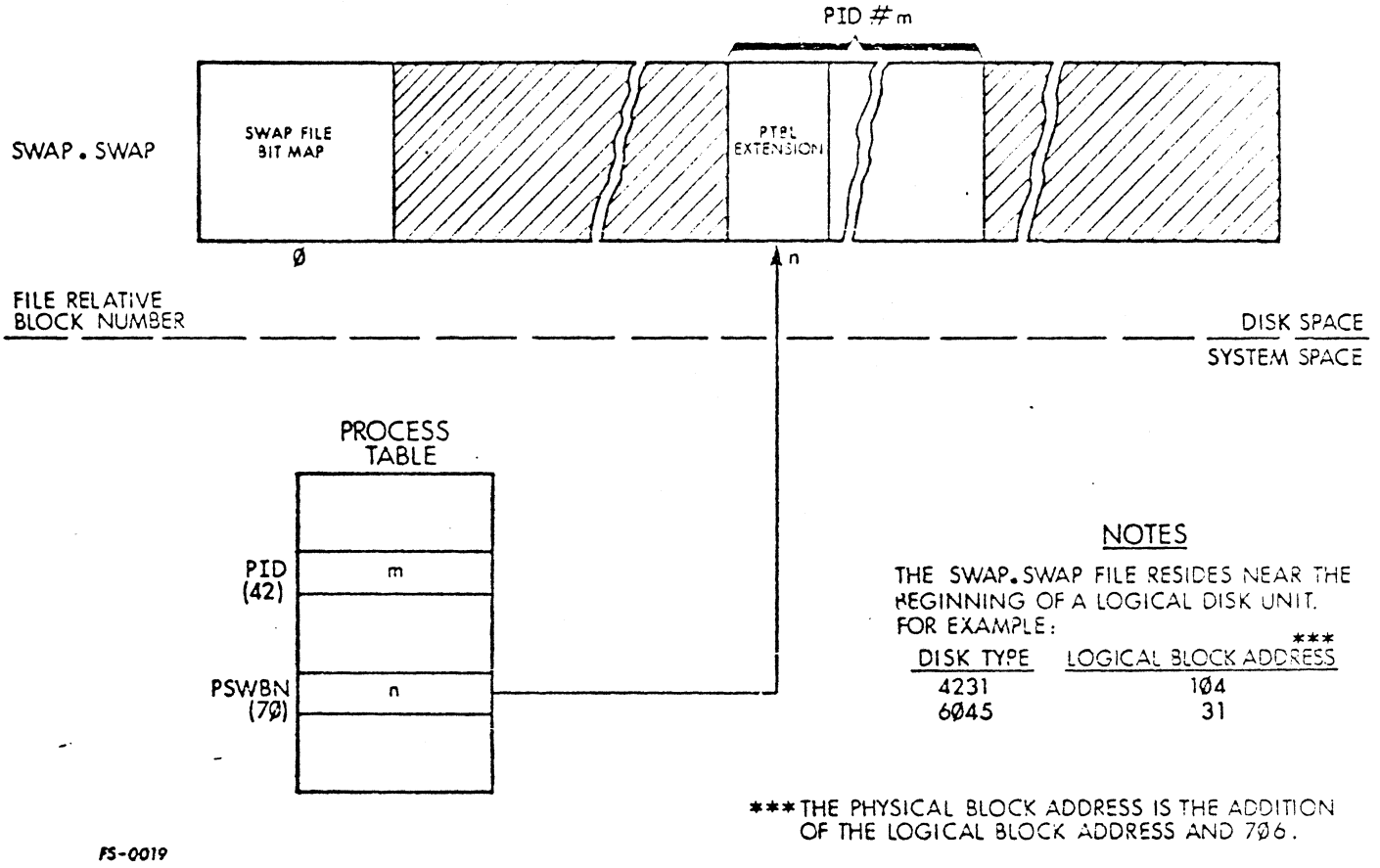


FIGURE 6.5.2 Process Table and Swap File Linkage

PANIC CODE 6 - FATAL SYSTEM FILE DATA BASE ERROR

General Information

A Panic Code 6 will occur because of invalid information being found in either a Channel Control Block (CCB), which is similar to a "UFT" under RDOS, or Logical Disk Control Block (LCB) which contains information about each Logical Disk (LD) that is initialized.

Specifics:

AC3 = 74447,75447 (DRLSE) IRLSE+446 Additional Logical Disks (LD) can be added to the master tree or removed from the master tree via System Calls or CLI commands. In this case, an attempt is made to release an LD because of a "?RELEASE" System Call or a "RELEASE" CLI command being issued. There are two cases that can occur.

Case #1: The Parent CCB is found to have a Use Count (CBUSC) of "1" instead of at least "2". At the time of Panic:

AC2 = address of Parent CCB

Case #2: This routine is used to remove the Logical Disk Unit (LDU) from the Logical Disk Control Block (LCB) chain. The chain is scanned LCB by LCB to find the proper LCB to remove. The end of the chain has been encountered without finding the right LCB. At the time of Panic:

AC0 - LCB pointer

AC2 - link LCB address

AC3 = 74525,75525

(XINIT) EINIT+524 The following two cases can occur during the initialization of a Logical Disk Unit (LDU). This can be done via the ?INIT Assembly Language System Call or the "INIT" CLI command. There are two cases.

Case #1: An error occurred during a call to "VCREATE" which creates the disk volume name. The rest is the same type error as DRLSE case #1.

Case #2: Same as DRLSE Case #2.

AC3 = 74523,75523

(SYOV1) SYDBL+522 These routines are used to disable a Synchronous line via the "?SDBL" System CAll. The address of the Parent CCB is loaded and checked to verify that the "Use Count" is non-zero. If the "Use Count" is zero, an error will occur. At the time of Panic:

AC0 = 0
AC2 = (CCB+6)
(365) - Cell Address (365 = CC)
((365)+2) = Channel Control Block Address

AC3 = 74215,75215

(Close) GCLOS+47 On closing a User Channel, the parent CCB offset in the current CCB is found to be 0 (should never happen)

AC0 = 0
AC1 = 100 (CBPER mask)
AC2 = 0 (CBPCB)

AC3 = 74424,75424

(Close) ECLOS+73

Case #1: On "Killing" an FCB, the CCB count in the Parent CCB is

decremented to zero (CBUSC)
AC2 = Parent CCB Address

Case #2 On "Killing" a system CCB, the CCB count in the Parent CCB is decremented to zero.
AC2 = Parent CCB Address

Discussion:

The above problems are either software in nature or a mis-addressing problem on the part of the hardware. Submit a core dump.

Examples:

None

Indepth AOS Information:

A. LDU Structure

When a system is boot'd, there is only one Logical Disk Unit (LDU) therefore only one Logical Disk Control Block (LCB). Note that the LDU may consist of more than one physical unit. Additional LDU's can be "grafted" (added) to the system via the CLI command "INIT" or removed from the system via the CLI command "RELEASE".

Figure 6.6.1 A and B shows an example of the directory structure of a single disk LDU named "GIL1". There is only one LCB and one Unit Descriptor Block (UDB) for the single CDC disk DPEØ. After "DIR'ing" into the directory UTIL, I issue the following CLI command:

```
) INIT  DPD1Ø  DPD14
  GIL2
)
```

I had to give two physical unit descriptions to the "INIT" command because my second LDU was composed of two units - Unit Ø (removable) and Unit Ø (fixed) on device code 73.

Figure 6.6.2 A and B shows the directory structure and the LCB/UDB linkage after the "INIT" had been done. The 2nd LDU's root is "grafted" to whatever directory I was in when the "INIT"

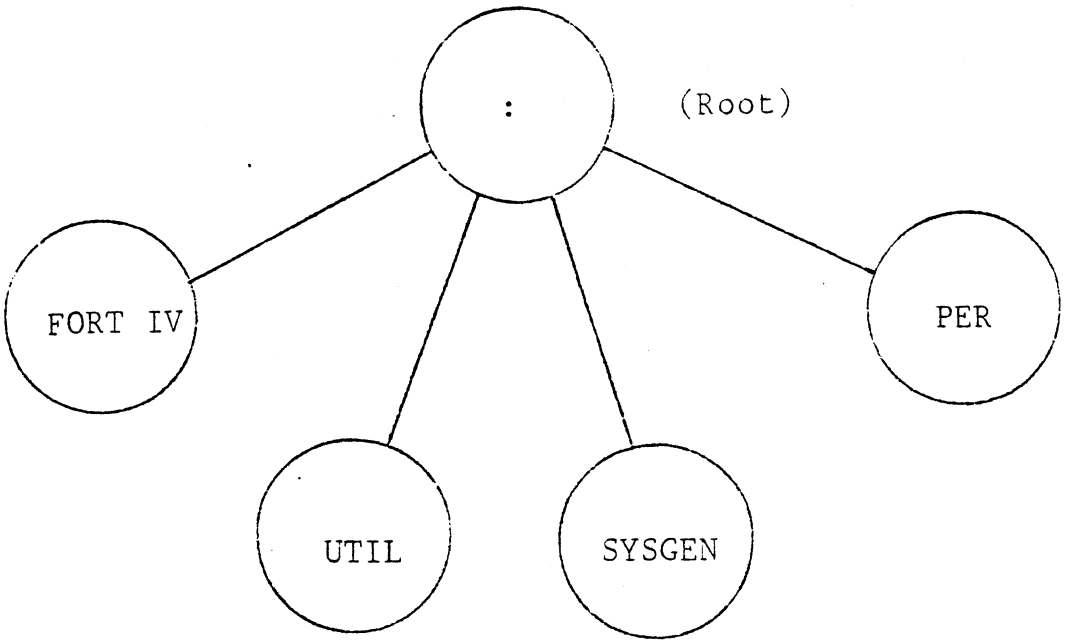


Figure 6.6.1 A Sample Single LDU Directory Structure

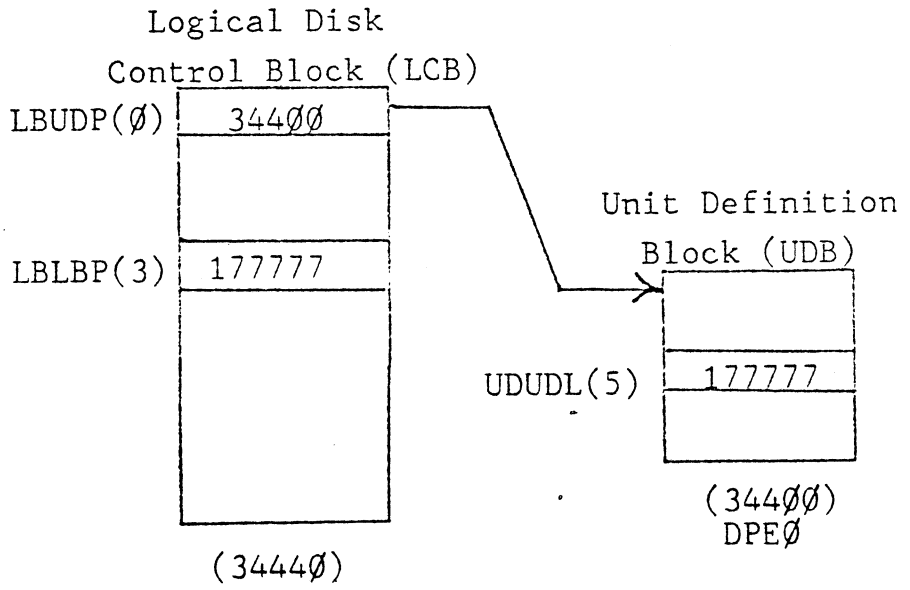


Figure 6.6.1 B Sample Single LDU/Unit Linkage

Figure 6.6.2 A Sample Dual LDU Directory Structure

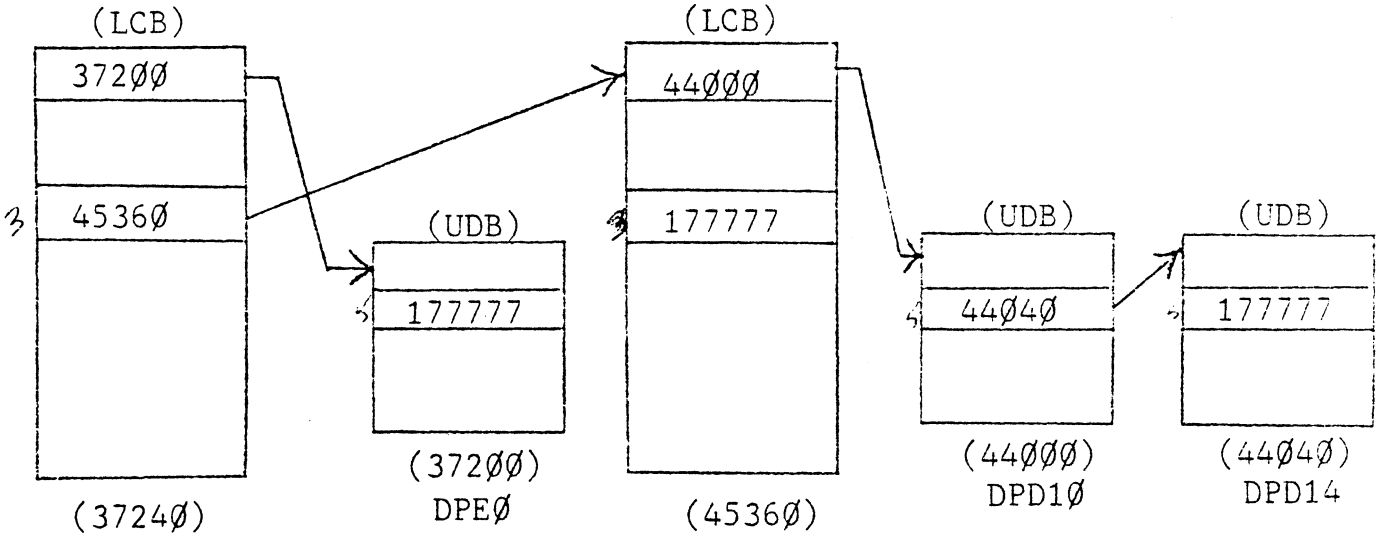


Figure 6.6.2 B Sample Dual LDU/Unit Linkage

command was issued. In this case, I had been in UTIL. The root (:) of the 2nd LDU is now addressed by the LDU "Logical Disk Name" which is GIL2. If I were in the 1st LDU's root, :, and wanted to go into the 2nd LDU's root, I would issue:

```
) DIR  :UTIL:GIL2 ↵
)
```

This would put me in GIL2, the 2nd LDU's root.

Figure 6.6.2 B shows the linkage that now exists for the addition of the 2nd LCB because of the"INIT". Attached to the 2nd LCB are the two UDB's for DPD1Ø and DPD14 which comprise the 2nd LDU.

B. CCB Linkage

A Channel Control Block (CCB) basically has two purposes under AOS:

1. When a User "Opens" a file, information must be maintained about that file on a "per User" basis. There is one CCB for every file or device a User "opens". If two Users open the same file, there are two CCBs - one for each User. These are referred to as "User CCBs".
2. A CCB is also maintained for every intermediate directory that is between the root directory and an "Open'd" User File. There is only one CCB for an "In Use" directory independent of the number of Users whose file pathnames "use" that directory. These are referred to as "System CCBs".

Figure 6.6.3 illustrates a sample AOS disk structure consisting of four directories, including the root, and one program file, TEST.

Figure 6.6.4 shows the corresponding CCB linkage for Figure 6.6.3. There is one CCB for each directory. The root directory (:) uses a special CCB, RTCCB, since it will always be "In use" anytime that the Logical Disk Unit is active. Each directory CCB has a Use count of 2 since there are two Users executing the program file, TEST. The Use Count is maintained in offset 15, CBUSC, when the CCB is used for a directory. No Use Count is

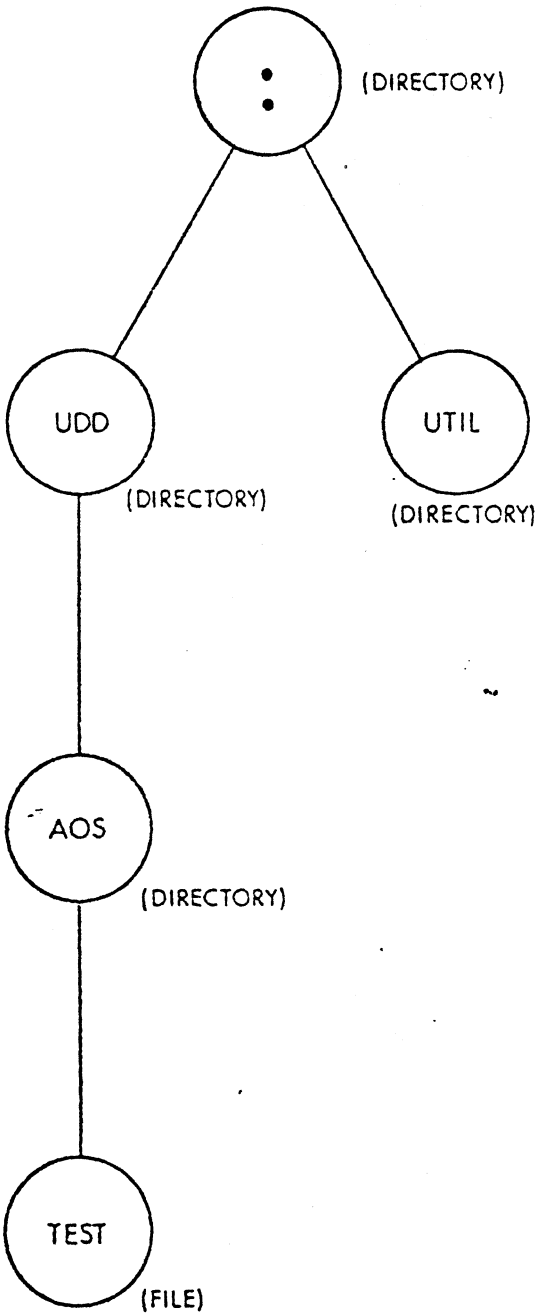


Figure 6.6.3 Sample AOS Directory/File Structure

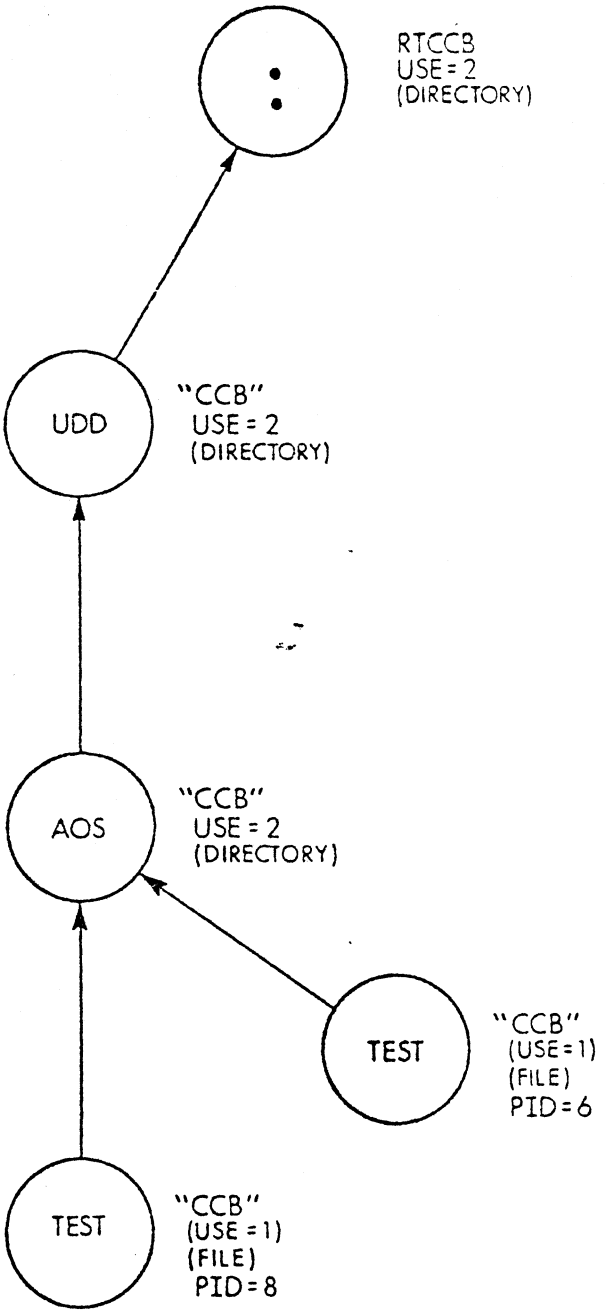


Figure 6.6.4 Sample CCB Linkage for AOS Directory/ File Structure

maintained if the CCB is used for a file since only one User can have access to that CCB. There is one CCB for each "Open" of the program file, TEST, and it will contain the address of the respective User's Process Table.

CCB's are linked only to their parent CCB as indicated by the upward arrows in Figure 6.6.4. Note that there is no CCB shown for the directory UTIL because no files are in use within UTIL nor is it being currently used in any "pathname". Because of this, it's Use Count is Ø and it was removed from the CCB tree.

Note that there is presently no way to determine the "filename" that belongs to a respective CCB using only "in core" information. The CCB does link to a File Control Block (FCB) which contains more information about the specific file or directory (Refer to Figure 6.14.4 and 6.14.5) but there still is not enough information left in core to determine the "filename". The information contained in the disk file structure is needed to find a specific "filename".

Also because a seperate CCB is maintained for each User who "Opens" a specific file, the pointers to that file are intact even if one of those Users (with the proper ACL) deletes the file. The remaining Users will continue using the file as if nothing had happened. When the last of those Users "Closes" the file, it will be lost to all forever. Because none of the other Users are aware that the file had been deleted, make sure, via ACL, that only the proper User has the ability to delete it.

PANIC CODE 7 - THE AOS OPERATING SYSTEM TRAPPED

General Information:

The AOS operating system runs with the map enabled and therefore will trap on a map violation the same as a User Process. AOS uses the full 32K logical address space, some of which is used for executable code, other sections for overlay windows, still other sections are used as "scratch" to construct needed temporary data bases. Because of the window mapping that the system uses, AOS can actually be the owner of more than 32K of physical core. Figure 6.7.1 shows a layout of the AOS logical address space. The format of the Panic message will give 6 values as shown in the example. These values are the Panic code, ACØ thru AC3, and the C+PC.

Specifics:

AC3 = Not meaningful

(PANIC) A memory violation has occurred while we were either:

1. Executing code in the Operating System, ie: SYSIN ≠ Ø.
2. Executing code while servicing an interrupt, ie: INTLV≠177777, which is done partly with the Map turned on.

Either of the above is considered a fatal error and at the time of Panic:

ACØ
AC1
AC2
AC3
PC

} The values they contained at the time of trap

Discussion:

Basically, the same discussion as for the Panic code 1Ø and Panic code 12 holds true for this Panic also. The information that is needed to evaluate the trap is:

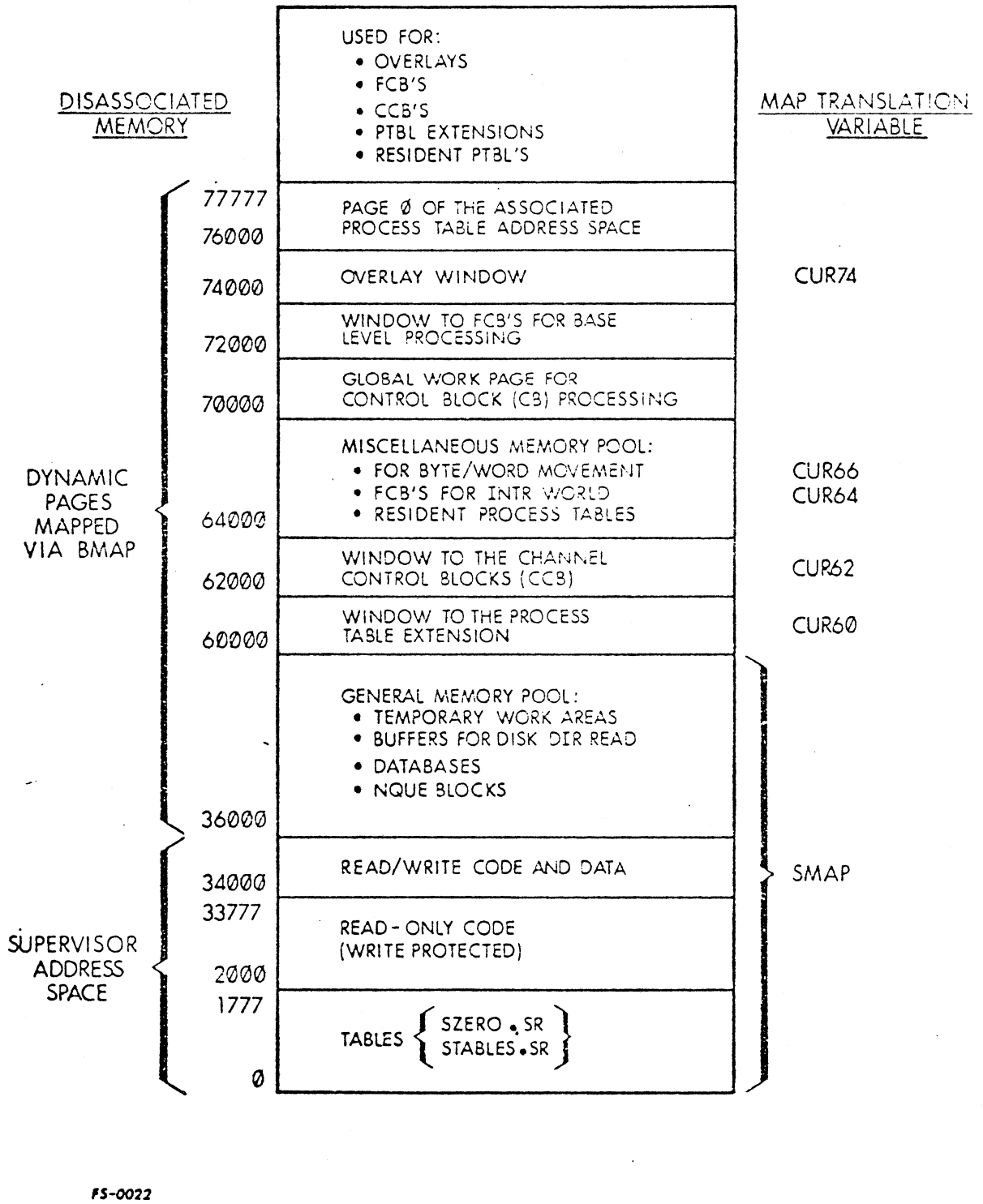


Figure 6.7.1 AOS Logical Address Space Layout

a) PC at the time of trap

b) AC's at the time of trap

c) Type of Map Violation

d) Physical core being used when the trap occurred.

The values for a) and b) above are contained in the Panic message. The value for c) is lost but can usually be assumed to be a "Validity" or a "Write-Protect" violation. I would strongly suggest reading the discussion under "Traps" in section 5 of this document as much of the information and cases cited are very relevent.

First we can evaluate the PC to determine where the trap occurred. I would expect the PC to be in one of two ranges:

a) 2ØØØ to 43777

b) 74ØØØ to 75777

If the PC falls with range a) the trap occurred within core resident code. If the PC falls with range b), the trap occurred within an overlay that is window mapped between 74ØØØ and 75777.

We can also examine the values for SYSIN and INTLV. Both need to be looked at as we can be "in the system" when an interrupt occurs. If INTLV≠ 17777, this would indicate that we were in interrupt code when the trap occurs. The "re-start" of block devices is done differently under AOS than under RDOS. Under RDOS, all the parameter setup is done at "base level" (non-interrupt level). Under AOS, the parameter setup for the next transfer is done at interrupt level. This means that many routines, nested or otherwise, can be executed while at interrupt level therefore a more prevalent chance of a "Trap" occurring.

To determine the physical core being used at the time of "Trap", refer to Figure 6.7.1. The AOS core translation table begins at the value "SMAP" and continues for 37 more locations. This map is the direct equivalent of the MPTx area under RDOS. "SMAP" will usually contain the core translation for logical addresses Ø-57777. The Logical to Physical Translation for current values between 6ØØØØ and 75777 are contained in "Current Page" variables. For example, "CUR6Ø" will contain the current logical to physical translation for the page beginning at 6ØØØØ. In order to examine a Process Table Extension or a User Map, the Physical

Page pointed to by PVEXT (offset 75 of the respective Process Table) is needed. A given memory page can only contain information on a maximum of four Processes, therefore, "CUR6Ø" will only be accurate for a maximum of four processes. PVEXT will always be accurate for the given process and should be used to determine the physical page for that process's 6ØØØØ slot.

In many cases, because of the amount of core whose addresses are valid when running in the AOS operating system mode, a problem may not be found until quite a ways "down the road" from the actual cause. This was referred to as the "Ping-Pong" effect in the discussion of "System Traps", Section 5. The most we may hope to accomplish, in many cases, is to limit the amount of possible physical "suspect" core. If a User has a 256Kw machine, we might be able to limit the problem to within 24Kw which definitely narrows the amount of core to be "swapped".

Examples:

Note that on all these types of Panics, the "Typeout" will contain 6 values as shown in the Sample Panic Code 7 typeout, Figure 6.7.2. If the problem appears to be complicated, there is minimal that you can do on-site and the dump should be submitted to National Tech Support for further analysis.

Sample Analysis #1:

The Panic message for the sample #1 Panic typeout yields the following information:

ACØ = 1Ø24ØØ
AC1 = 41ØØØ
AC2 = 41ØØ2
AC3 = 41ØØ3
C+PC = 41ØØ4

Since I now know the AC values and the PC at the time of trap, I can now use DEDIT and SYSDMP to get additional information. I find that:

SYSIN = 1
INTLV = Ø

which tells me that I was servicing an interrupt at the time of "trap".

```
FATAL AOS ERROR - 7 102400 041000 041000 041000 041004
                        AC0    AC1    AC2    AC3    C+PC
AOS CORE DUMP
LOAD TAPE FOR DUMPING ON MTA0
STRIKE ANY KEY WHEN READY

AOS CORE DUMP COMPLETED
```

Figure 6.7.2 Sample Panic Code 7 Typeout

	MASK	HARDWARE MASK BIT	DEVICE
	3	15	TTO, TTO1
	3	14	TTI, TTI1
	7	13	PTP
	17	12	MCA
	17	12	PLT
	10	12	LPT
(517)*	717	7	MHD
(777)*	377	8	SLM, ALM
	1737	11	PTR
	1777	10	CDR
	7 7777	4	DCU5Ø
	4	13	RTC
	1ØØØ	6	Interval Timer
	177777	--	ERCC
	177777	--	Power Fail
	1777	10	MTA

* changed to this value in AOS Rev. 1.Ø5

TABLE 6.7.1 INTERRUPT MASKS

I next found out the core being used by the AOS operating system using SMAP. The logical addresses being used were:

Logical Addresses	Physical 1K Pages
Ø-43777	Ø-21
52ØØØ-57777	174-176

The contents of the current window pages was:

Current Window	Physical 1K Pages
CUR6Ø	162
CUR62	141
CUR64	152
CUR66	Ø
CUR74	136

I now know the memory being used.

Since I was at interrupt level, I can examine location 5 (CMSK) to find the current interrupt mask. Since CMSK is a "3" and by comparing this to the table on the following page, I find that the "TTY" caused the last interrupt, actually the only interrupt since INTLV = Ø.

I also made a layout of the Interrupt Stack using the formats given under the Panic Code 1 discussion. Since listings are not available in the field, the analysis of the various "SAVE's" is not reasonable to expect and is illustrated here only for your reference.

At this point, I would probably be tempted to "Swap" either one of the lower 8K modules, Mod Ø or Mod 1, or the 1st 16K module, Mod Ø, and exercise the system to see whether the failure occurred again. Interleaving must be taken into account if it exists in the system. In parallel to this, I would expect the core dump to be submitted as outlined at the beginning of this chapter.

Sample Analysis #2:

The Panic message for the sample #2 Panic typeout yields the following information:

ACØ = Ø
AC1 = Ø
AC2 = 1757

```
*) DIR :UTIL↵
*) COPY DUMP0420 @MTA0:0↵
*) X DEDIT DUMP0420↵
AOS FILE EDITOR REV 1.0
+ STAB↵
FILENAME? :SYSGEN:AOS21.ST↵
+ INTLV:000000 +↵
+ SYSIN:000001 +↵
+ SMAP:001400 +<LF>
SMAP+1 :003601 +<LF>
SMAP+2 :005602 +
SMAP+3 :007603 +
SMAP+4 :011604 +
SMAP+5 :013605 +
SMAP+6 :015606 +
SMAP+7 :017607 +
SMAP+10 :021610 +
SMAP+11 :023611 +
SMAP+12 :025612 +
SMAP+13 :027613 +
SMAP+14 :031614 +
SMAP+15 :033615 +
SMAP+16 :035416 +
SMAP+17 :037417 +
SMAP+20 :041420 +
SMAP+21 :043421 +
SMAP+22 :045777 +
SMAP+23 :047777 +
SMAP+24 :051777 +
SMAP+25 :053576 +
SMAP+26 :055575 +
SMAP+27 :057574 +
SMAP+30 :061777 +
SMAP+31 :063777 +
SMAP+32 :065777 +
SMAP+33 :067777 +
SMAP+34 :071777 +
SMP+35 :073777 +
SMAP+36 :07577 7 +
SMAP+37 :077 777 +↵
+ BYE↵

*)
```

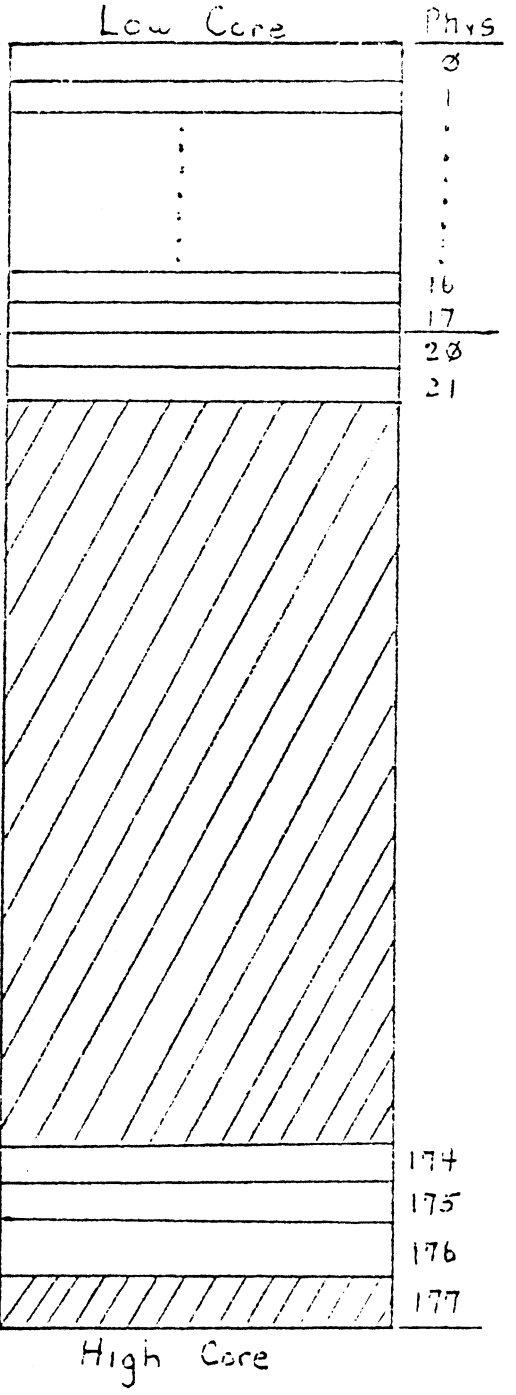


Figure 6.7.3

Sample DEDIT Dump Analysis and System Core Allocation

```
) X SYSDMP DUMPØ42Ø
AOS SYSTEM DUMP ANALYZER REV 1.Ø
+ STAB
FILENAME? :SYSGEN:AOS21.ST
+ 5:ØØØØØ3 + CMSK
+ 4:ØØ156Ø + ISP
+ 6:ØØ1757 + ISL
+ 7:Ø166Ø3 + ISO
+ 4Ø:ØØ1625 + SP
+ 41:ØØ162Ø + FP
+ 42:ØØ1757 + CSL
+ 43:Ø166Ø3 + CSO
+ 156Ø:ØØØØØØ +
SS+1 :Ø347Ø4 +    4Ø
SS+2 :Ø42443 +    41
SS+3 :Ø35Ø5Ø +    42
SS+4 :Ø166Ø3 +    43
SS+5 :Ø324Ø2 +    ACØ
SS+6 :Ø16Ø27 +    AC1
SS+7 :ØØØ141 +    AC2
SS+1Ø:ØØØ3Ø6 +    AC3
SS+11:1Ø2Ø36 +    C+PC
SS+12:ØØØØØØ +    Old Mask
SS+13:Ø13642 +    Old Loc 2
SS+14:Ø2ØØ37 +    Old Map DIA
SS+15:Ø61562 +    CUR6Ø
SS+16:Ø63541 +    CUR62
SS+17:Ø65573 +    CUR64
SS+2Ø:Ø67541 +    CUR66
SS+21:Ø75736 +    CUR74
SS+22:ØØØØØØ +
SS+23:Ø35624 +    TTI1DC    (PSH 2,2)
SS+24:ØØØØØØ +    ACØ
SS+25:ØØØØØØ +    AC1
SS+26:Ø65ØØ6 +    AC2
SS+27:ØØØØØØ +    OFP
SS+3Ø:127171 +    C+AC3              SAVE 3
SS+31:127165 +    TMP1
SS+32:Ø63541 +    TMP2             (TISER+46)
SS+33:ØØØØØØ +    TMP 3
SS+34:ØØØØØØ +    ACØ
SS+35:Ø3Ø5ØØ +    AC1
SS+36:Ø65ØØ6 +    AC2                SAVE Ø
SS+37:ØØ161Ø +    OFP
SS+4Ø:Ø27361 +    C+AC3              (BLD+Ø)
SS+41:1Ø24ØØ +    ACØ
SS+42:Ø41ØØØ +    AC1
SS+43:Ø41ØØ2 +    AC2              "TRAP" RTN Block
SS+44:Ø41ØØ3 +    AC3
SS+45:Ø41ØØ4 +    C+PC
SS+46:ØØ1622 +
```

Figure 6.7.4 Additional SYSDMP Dump Analysis of the Interrupt Stack

```
FATAL AOS ERROR - 7 000030 000000 001757 034316 174055
                        AC0    AC1    AC2    AC3    C+PC
AOS CORE DUMP
LOAD TAPE FOR DUMPING ON MTA0
STRIKE ANY KEY WHEN READY

AOS CORE DUMP COMPLETED
```

Figure 6.7.5

Sample #2 Panic Code 7 Typeout

```
) X SYSDMP DUMP0414
AOS SYSTEM DUMP ANALYZER REV 1.0
+ STAB
FILENAME? :SYSGEN:AOS20.ST
+ SYSIN:000001 +
+ INTLV:1777 77 +
+
+ CUR74:075662 +
+
+ 366:000642 +
+ OVTAB=600
+ 642-600=42
+
+ MAP 62
+
+ 54:041027 +
+ 55:006175 +
+
+ MODE N
+
+ 54:STA 0 27 2  +
+ 55:JSR @.UNMA 0  +
+
+ 1757+27=2006
+
+ MAP SMAP
+
+ BYE

)
```

Figure 6.7.6

Sample SYSDMP Dump Analysis of Failure within an Overlay

AC3 = 34Ø16

C+PC = 174Ø55

First looking at the PC, I can tell from the previous discussion that the error occurred in an overlay. We will discuss a little later about finding out which overlay.

I now use DEDIT to examine the core dump. I first check:

SYSIN = 1

INTLV = -1

which tells me I was executing system code at the time of failure.

Since overlays are always executed via logical locations 74ØØØ-75777, I can examine CUR74 which is 75662 and tells me the Physical Page being used is "62". To find out which overlay, I can:

(366) = 642

OVTAB = 6ØØ

642-6ØØ = 42

Now to explain the above. The location 366 is called "CRSEG" and points to an entry in an Overlay Table. Each entry in this table, tells the state of one of the system overlays of which there are $61_{(8)}$. The beginning of the table is pointed to by the address of OVTAB. By subtracting (CRSEG) - OVTAB, we can find out the overlay number. In this case it is 42, which by comparing to the list of overlays at the end of this chapter, we can find to be "SYSER" which handles errors. (Note that I had the SLM Aosgen'd in the system).

By "MAPing" Page 62 under SYSDMP, we can find the contents of the possible PC locations to be:

54:41027 = STA Ø, 27, 2

55:6175 = JMP @175, Ø

Since AC2=1757, the location to be stored into is "2ØØ6". Remember that core beginning at location 2ØØØ is "Write-Protected". Since we tried to store at location 2ØØ6, naturally we got a "Write-Protect" violation.

Again since we do not have the listings available, I would suggest 1st "swapping" the module containing Physical Page #62, Mod 6 (8K) or Mod 3 (16K). Note that his module may vary on an

interleaved machine depending on the level of interleaving. Although we haven't "solved" the problem using core dump analysis, we know the overlay that we failed in, the location in the overlay, and the physical core containing the overlay. If the problem still occurs, I would "swap" the core being used by "SMAP".

Note that again in parallel with the above "swapping", I would suggest a core dump be submitted to National Tech Support as outlined in the beginning of this chapter.

Indepth AOS Information:

The following sections will discuss in a little more depth the topics that you may encounter with this Panic.

I. AOS Core Usage

The "Disassociated Memory" at the top of Figure 6.7.1 is core pages that belong to the AOS operating system and contain AOS operating system information but are currently not mapped into any logical page slot. Logical addresses Ø-35777 are referred to as the "Supervisor Address Space", the "nucleus", or the "kernel". This area, once loaded, will be mapped to itself, ie: Phys. Page 1Ø = Log Page 1Ø, and will never be dynamically changed other than on a word storage basis. The contents of Logical Addresses 36ØØØ-77777 are dynamically changed on an as needed basis.

II. Overlays

There are in reality three types of overlays:

Type	Overlay #	Usage
1	Ø-1	System Initialization
2	2-11	Resident Overlays (Once in memory, they are locked in) overlays #1Ø and 11 are used for SLM. If never used, they are never loaded.
3	12-6Ø	Disk Based - Read

into memory and deleted via LRU (least recently used) algorithm.

The "System Initialization" overlays are used only once for system initialization, then not used again. The "Resident Overlays" and the "Disk Based" overlays are the two types that we will be mainly interested in. Each type uses its own data bases and will be discussed seperately below. Note that the "Disk Based" overlay #'s will differ depending on whether SLM gear has been sysgen'd in the current operating system. This difference is indicated in Table 6.7.2 which lists the entry points for each overlay.

1. Disk Based Overlays

The Disk Based overlays are orgainized according to the structure shown in Figure 6.7.7. Each overlay had an entry in an Overlay Table, the first entry of which is indicated by OVTAB. The entries in this table can have any one of the following values:

Value	Meaning
Ø	Overlay not currently in core
1	Overlay being read into core
address	Indicated the Overlay description block which describes the overlay.

Note that offsets in OVTAB will still be included for the "Resident" overlays but these offsets will always contain a "Ø". All non-resident overlays in core are described by an "Overlay Descriptor Block" which is attached to a chain of these blocks. Each descriptor has link words, the physical block number that the overlay resides in, the overlay number, and other information. The beginning of the chain is pointed to by the contents of OVMCH and the end of the chain by the contents of OVEND.

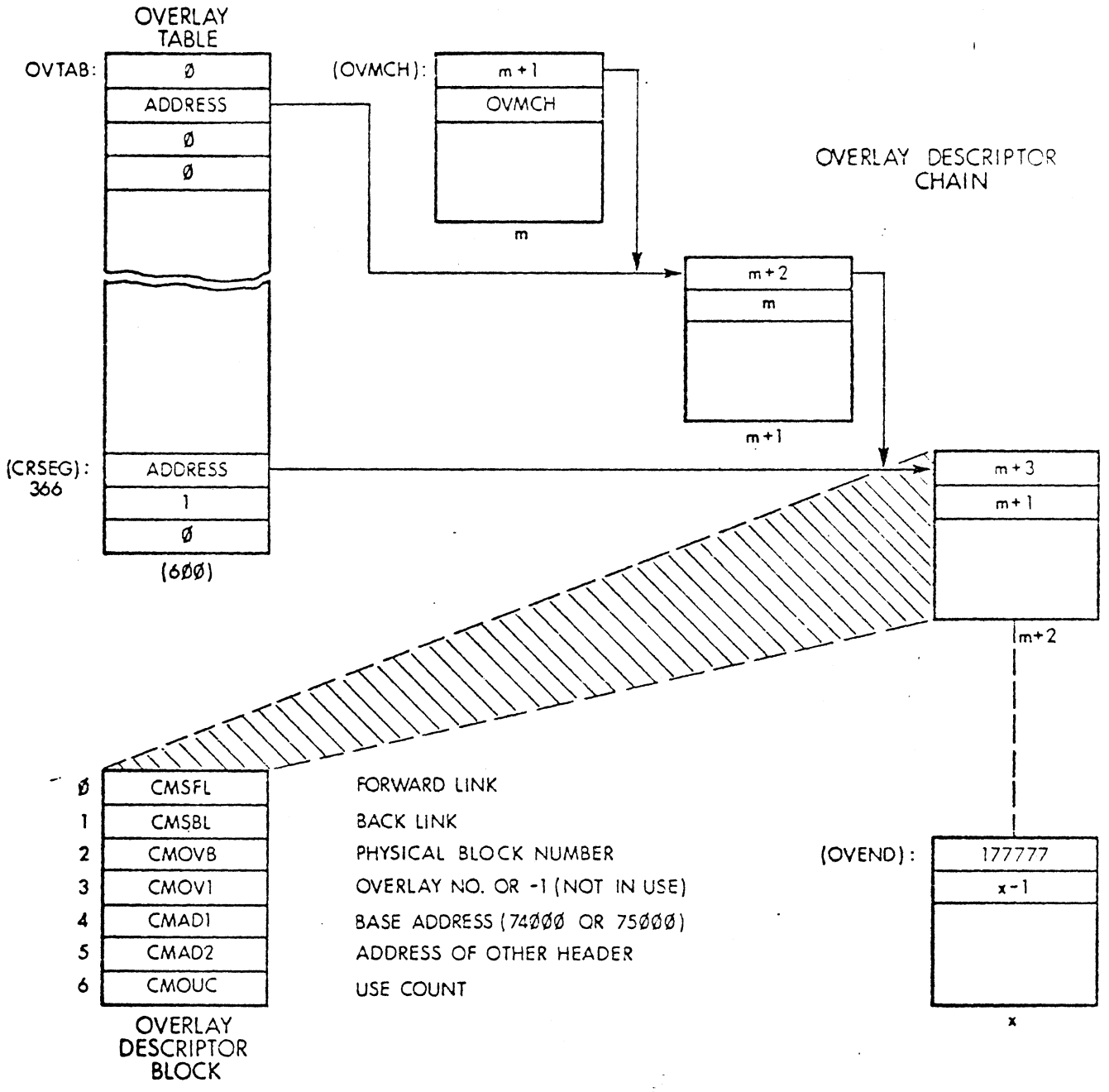


Figure 6.7.7
Disk Based Overlay Chain Data Base Linkages

2. Resident Overlays

The Resident overlays are organized according to the structure shown in Figure 6.7.8. When a resident overlay is being executed, CRSEG will point to the Resident Overlay Descriptor Block and bit Ø of CRSEG will be a "1". This descriptor block is common to all resident overlays and there is only one of these in the system. Offset 3 indicates the "overlay #" and offset 4 indicates the "base address". The Physical Page containing the overlay is in the respective offset of the Resident Overlay Core Table and the contents of this offset are also copied into "CUR74". For example (refer to Figure 6.7.8), if the operating system were executing overlay #4, the data bases would look as:

CRSEG: 1ØØ747
CMOV1: 4
CMAD1: 74ØØØ
CUR74: 174232

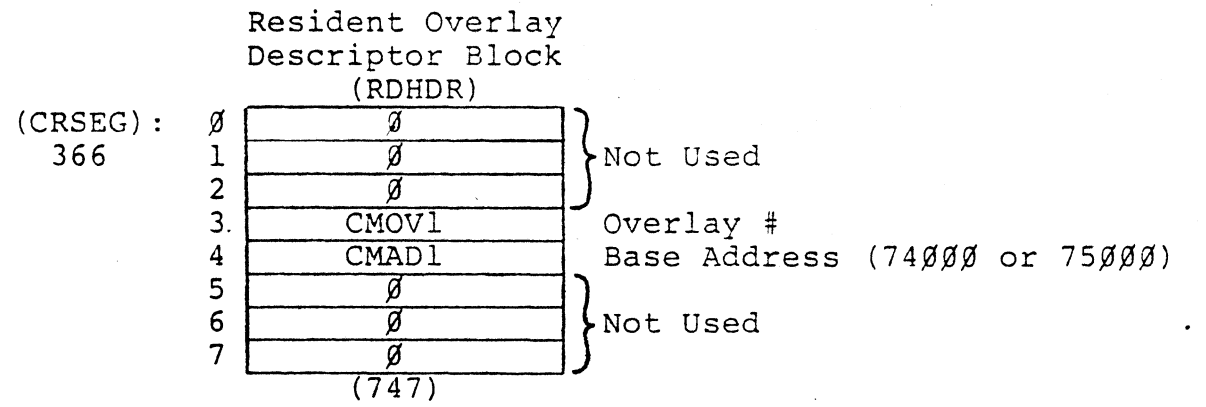
Note that CUR74 is the same as offset 1 of ROVTB which is the slot used for overlays #4 and 5.

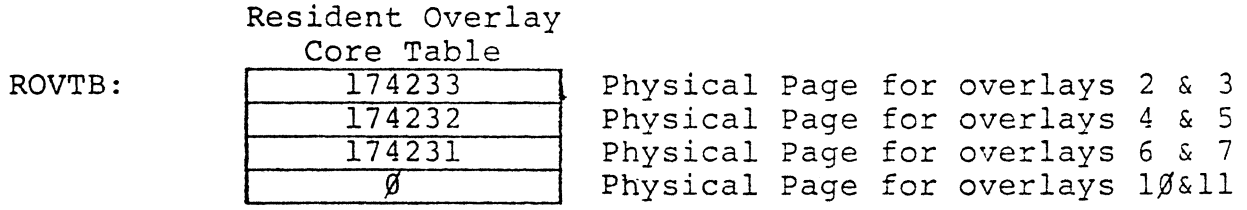
Table 6.7.2 is a table of the overlays listing the octal overlay number, the module name, and the entry points into the overlay. Note that Logical Page $36_{(8)}$ (74ØØØ-75777) is used as a window to execute overlay code. The actual physical page mapped to will change many times during system operation but overlays will always be addressed using a logical base address of 74ØØØ or 75ØØØ.

The current overlay being executed is pointed to by CRSEG (location 366) whose contents can have any of the following formats:

a) Ø - no overlay is currently being executed
b) 1BØ+RDHDR - a "Resident" overlay is being executed and is pointed to by offset 3 of RDHDR.
c) Overlay#+OVTAB - a "Disk Based" overlay is being executed whose # is (CRSEG-OVTAB)

AOS optimized overlay core usage by putting two overlays in each core page ($2ØØØ_8$ locations). Therefore, each over-





Note that if the SLM is not "AOSGEN'd", as in the above illustration, overlay #1Ø and 11 now become disk based overlays and a "Ø" is included in that table slot.

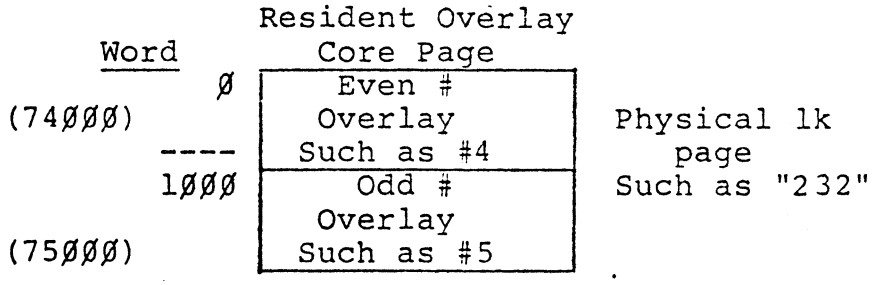


Figure 6.7.8 Resident Overlay Data Bases

lay can be a maximum of 1000_8 words and one overlay begins at word 0 in a page, the other at word 1000. Since overlays are executed via logical locations 74000-75777, one overlay will begin at 74000, the other at 75000 .

The example on the following page, Figure 6.7.9 and 6.7.10, is the same as Example #2 but the Overlay Descriptor Blocks are shown for the failing overlay, SYSER.

Note also that an overlay virtual address can be broken down into the following format:

bit 0 = 1

bits 1-7 = overlay #

bits 8-15 = entry point offset within the overlay.

For example, the overlay entry point "SDOWN" is given on a load map as "122663" and yields overlay #45, entry point offset #263 (SSOV8). (Note w/SLM Aosgen'd)

```
FATAL ACS ERROR - 7 000000 000000 001757 034016 174055
                    AC0    AC1    AC2    AC3    CrPC

ACS CORE DUMP
LOAD TAPE FOR DUMPING ON MTA0
STRIKE ANY KEY WHEN READY

ACS CORE DUMP COMPLETED
```

Figure 6.7.9 Sample #2 Panic Code 7 Typeout

```
*) DIR :UTIL↵
*) COPY DUMP0414 @MTA0:0↵
*) X DEDIT DUMP0414↵
ACS FILE EDITOR REV 1.0
+ STAB↵
FILENAME? :SYSGEN:ACS20.ST↵
+ SYSIN:000001 +↵
+ INTLV:177 777 +↵
+ SMAP+35:073777 +<LF>
SMA+36 :075777  +<LF>
SMAP+37 :077777 +↵
+ OVMCH:036330 +↵
+
+ 36330:042160 +<LF>
CBASE+1317 :000760 +
CBASE+1320 :000116 +
CBASE+1321 :000046 +
CBASE+1322 :075000 +
CBASE+1323 :036320 +
CBASE+1324 :000001 +↵
+
+ 36340:177 777  +<LF>
CBASE+1327 :036350 +
CBASE+1330 :000062 +
CBASE+1331 :000042 +
CBASE+1332 :074000 +
CBASE+1333 :036350 +
CBASE+1334 :000001 +↵
+
+ BYE↵
```

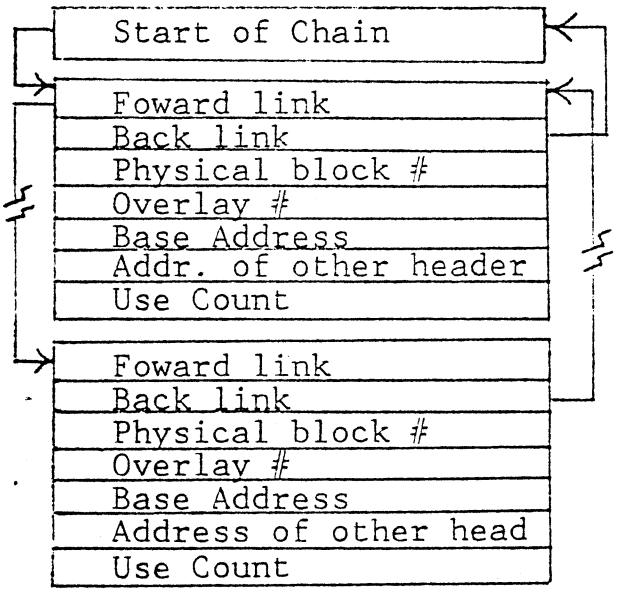


Figure 6.7.10

Sample DEDIT Dump Analysis of Failure within an Overlay

TABLE 6.7.2 (Revised per AOS 1.Ø7)

(W/O SLM SYSGEN'D) OVERLAY #	(W/SLM SYSGEN'D) OVERLAY #	TITLE	ENTRY POINTS
Ø	Ø	SSOV1	MEM, IGDAY, GTIME, STIME, IGTIM, ISTOD, GDAY, SDAY, SSHPT, GSHPT, INREL
1	1	UNIT	UOPEN, UCLOSE, GTUDB, RLUDB, GDUDB
2	2	CMOV1	BFGRW, FREBF, FREOV, IELSC, BSCAN
3	3	SCMOD	RPAGE, TIMEQ, SIGNAL, MBTG, MBFG RMBTU, RMBFU, SCMOD
4	4	IOOV	IOCAL
5	5	SWAPI	SWIN
6	6	SWAPO	SWOUT
7	7	SWAPM	SWIAB, CCBIO
-	1Ø	SYNLP	SYIPR, SYCPR, SLMRE, SYTPR, SYNPR
-	11	SYNCH	SYINT, SYTTS, RCVIT, SYTCH, SYNRC
1Ø	12	RESLV	RESLV, WRSLV, DRSLV, LKRTN, RSTACK, DBLK, NAMSP, CPAGE, SLPTR
11	13	RESL2	LOOKUP, FFCB, LINK, ESTAC, PESTAC, UESTAC, TMATCH
12	14	OPEN	GOPEN, EOPEN, SOPEN, IOPEN, XIOPEN, XEOPEN, GCCB
13	15	CLOSE	GCLOSE, ECLOSE, RESET, KFCB, KCCB, KCCBE
14	16	DE	DELETE, IDELETE, VDELETE, CLDEL
15	17	CREAT	BCREATE, CREATE, ICREATE, VCREATE
16	20	DRLSE	DRLSE, IRLSE
17	21	BOOT	BOOT, IBOOT
2Ø	22	PROC	PROC
21	23	PROC2	PROC2

(W/O SLM SYSGEN'D) OVERLAY #	(W/SLM SYSGEN'D) OVERLAY #	TITLE	(Revised per AOS Rev. 1.Ø7) ENTRY POINTS
22	24	PROC3	PRCER
23	25	PRCNG	RET, FPTRM, PRNCI, PROIN, TERM
24	26	SSOV3	WSWP, DSWP, CSWP, SPY , IPRLD,XLATE
25	27	SSOV4	BRKFL, BLKPR, CNGTY, GRUNT, IGRNT, GUNM, PRIPR, IHIST, PRSTAT, TRMPR, UBLPR
26	30	SSOV5	GPRNM, DADID, GLIST, SLIST, RELSL GNCP, ILKUP, SLKUP, GPORT, TPORT SBIAS, GBIAS, HISTI, KHIST
27	31	DINIT	DINIT
3Ø	32	XINIT	XINIT, EINIT
31	33	NAMES	ALIAS, VALIAS, RENAME, GNAME, IGNAME, GTACP
32	34	CHAIN	GCHAIN
33	35	SSOV2	BLK, UNBLK, INTAD, SSO, GHRZ, OEBL, ODIS, STMAP, IDEF, IRMV, DDIS, DEBL, SUSER
34	36	SSOV6	IGHOS, TABT
35	37	DIRST	DIR, RDIR, CPMAX, FSTAT
36	40	ACLST	SACL, ISACL, JSACL, KSACL, VSACL, GACL,
37	41	PNAME	NMTID, PNAME, IPNME, SPNAM
4Ø	42	SYSER	UNERR
41	43	NAME2	PURGE, VPURGE, CLINK, GLINK,GALIAS, MPTHN
42	44	SSOV7	DIPCF, LOGEV, LOGGR, ILOGE, ILOGG
43	45	SSOV8	DPMGR, SDOWN, IMBTU, MBTU, MBFU
44	46	SSOV9	MEMI, FLUSH, CFAULT

(Revised per AOS Rev. 1.Ø7)

(W/O SLM SYSGEN'D) OVERLAY#	(W/SLM) SYSGEN'D OVERLAY #	TITLE	ENTRY POINTS
45	47	INFOS	RDUDA, WRUDA, CRUDA
46	50	IREC	IREC, IS.R
47	51	ISEND	ISEND, SIPC, IIPC
5Ø	52	ISEN2	ISEN2
51	53	CLNUP	CLNUP
52	54	SOV11	BRKLTH, UBRK, IBRK, LINF, INFI, PRI TYPEC, SATR
53	55	SOV1Ø	JELLO, GEIB, GNFN
54	56	SYOV1	SYEBL, SYDBL
55	57	SYOV2	SYSND, SYRCV , SYGES
56	6Ø	SYOV3	SYNCP , SYPOL, SYEPL, SYDPL

PANIC CODE 1Ø - OPERATOR CLI TRAPPED

General Information:

The Operator CLI is the Process which executes under AOS at PID=2. It is this Process which is used by the operator to initialize the EXEC and other processes on any additional terminals. The format of the Panic message gives 6 values as shown in the example Figure 6.1Ø.1. These are the Panic Code, ACØ thru AC3, and the PC.

Specifics:

AC3 = Not meaningful — (PANIC) A trap has occurred in the Operator CLI process. This is considered a fatal error since the Operator controls the process environment.

ACØ, AC1, AC2, AC3, PC — The values they contained at the time of trap.

Note that the "Abnormal system shutdown" message will always precede the Panic Code 1Ø message.

Discussion:

Basically the Operator CLI is a User Process and can be troubleshot as such. Figure 6.1Ø.2 on the following page depicts the CLI logical address space. The CLI has one page, Page Ø, of unshared space. The rest of the CLI is shared among all the processes running the CLI, is write-protected, and extends from logical addresses 4ØØØØ thru 77777.

The procedure for troubleshooting a CLI Trap is similar to the sequence used to troubleshoot the Peripheral Manager Trap (Panic Code 12). First we have to find the address of the CLI's Process Table. There is a table in AOS that is ordered by PID#

```
ABNORMAL SYSTEM SHUTDOWN

FATAL AOS ERROR - 10 000104 001760 063476 063225 077574
                        AC0    AC1    AC2    AC3    C+PC
AOS CORE DUMP
LOAD TAPE FOR DUMPING ON MTA0
STRIKE ANY KEY WHEN READY

AOS CORE DUMP COMPLETED
```

Figure 6.10.1 Sample of a Panic Code 10 Typeout

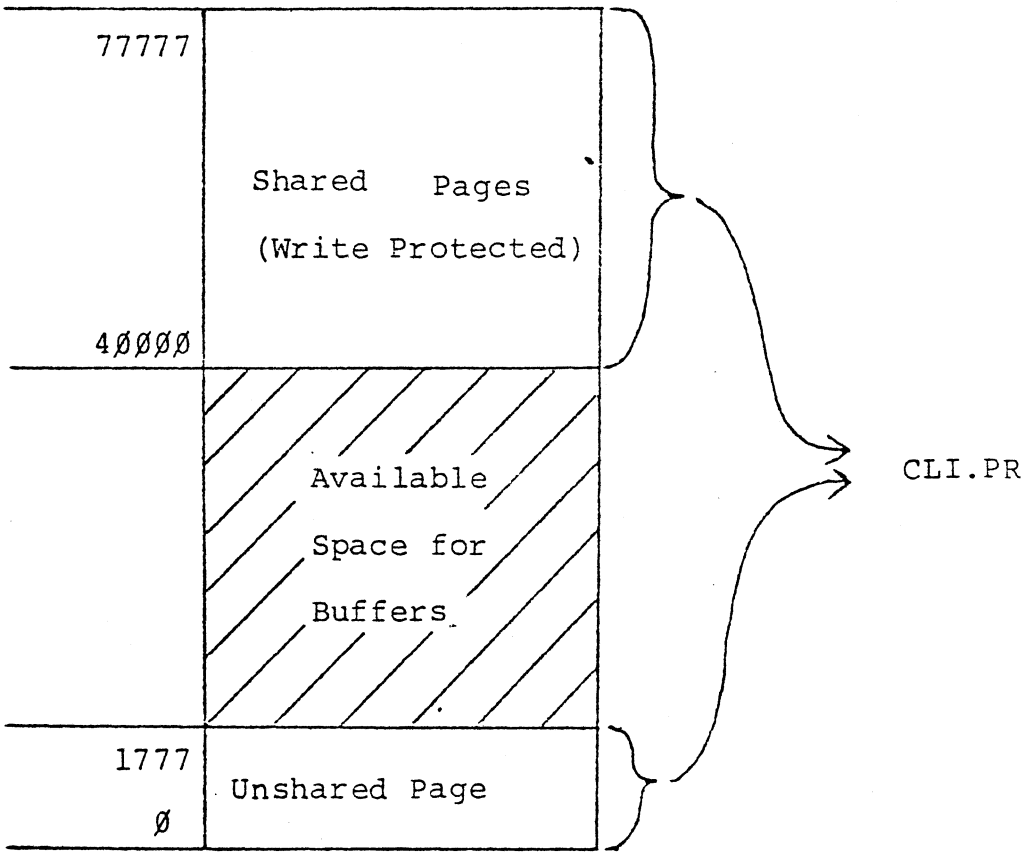


Figure 6.10.2 Example of CLI Logical Core Utilization

and points to each PID's respective Process Table. It is shown in Figure 6.1Ø.3. The entry for PID 2 will point to the Process Table for the Operator CLI. In attempting to evaluate the "Trap", there is certain information that we would like to know:

a) PC at the time of Trap

b) AC's at the time of Trap

c) Type of Map Violation

d) Physical Core being used when the trap occurred

The values for a) and b) are contained in the Panic message. The values for c) and d) have to be gotten from the Process Table Extension and the Process Table, respectively. These procedures are written quite explicitly under the Panic 12 "Discussion" and I will not repeat them here.

A present problem with this Panic is that all the system resources for the Operator CLI are released prior to the Panic. This causes the Physical Page values for the "Sharable" Core Pages to be released, therefore removed from the User Map and replaced with "377" (C3ØØ) for the physical. What this all means is that we do <u>not</u> know what physical core was being used by the Operator CLI.

Since that area of the CLI code is being shared among all processes executing the CLI, if we can find another PID or Process that was using CLI, we can find the physical pages being used. If the User or Operator can tell you what PID was also running the CLI, you can use the PID Table to find the Physical Pages being used. If no other process was running the CLI, that is as far as you can go. Submit the Dump as outlined in the "General Panic Discussion" to National Tech. Support.

<u>Example</u>:

(Note that the Discussion and Examples under <u>Panic Code 12 should be read prior to this</u>).

The sample Panic Code 1Ø, Figure 6.1Ø.1, typeout will provide us with the following information:

ACØ = 1Ø4

AC1 = 176Ø

AC2 = 63476

AC3 = 63225

C+PC = 77574

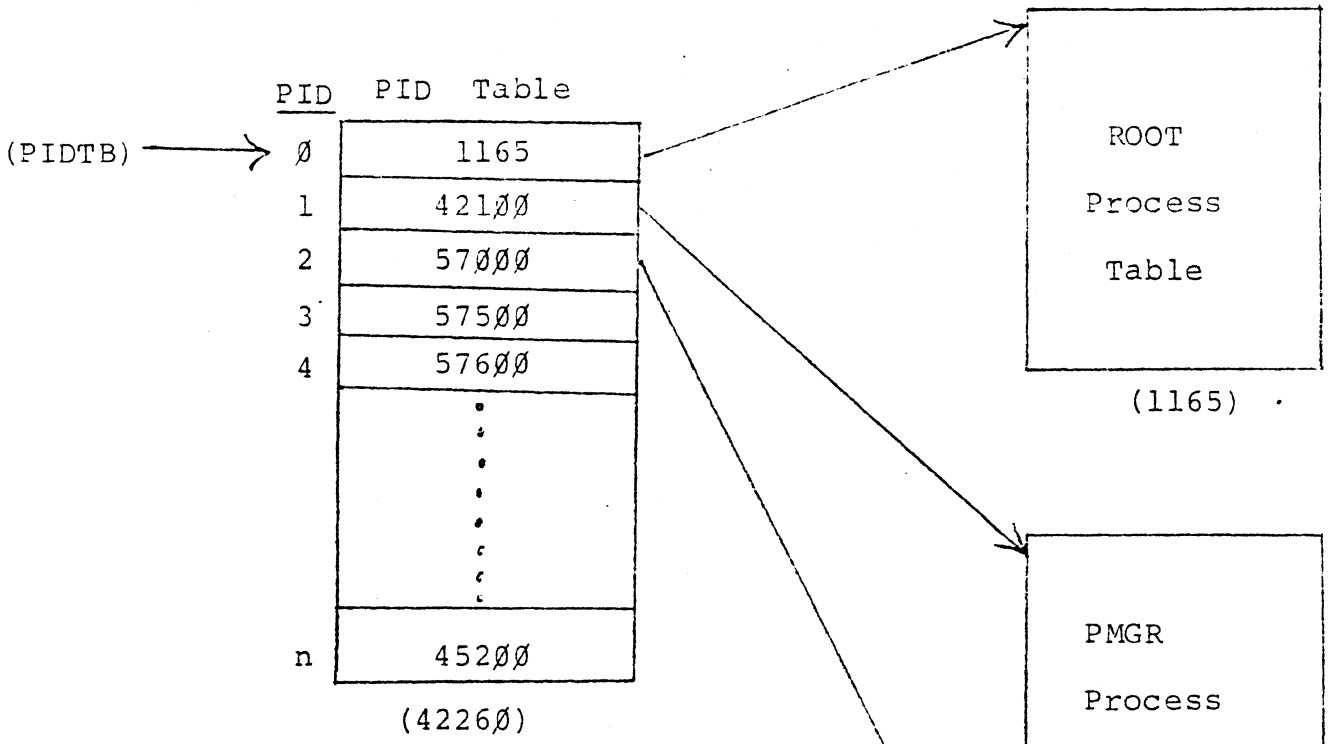
Using the SYSDMP utility, we can analyze the core dump to find the type of trap and the physical memory. The value of (PIDTB)+2 is 57ØØØ which is the Process Table address for PID = 2 or the Operator CLI. The PFLAG word is 6222 with bit 5 indicating that a Trap occurred. Offset 16 points to the Process Table Extension and Offset 25 points to the User Map. Since both pointers are between 6ØØØØ-61777 we can use the contents of offset 75, PVEXT, to find the physical page that is mapped to that slot and then issue the "MAP 162" command. Remembering that only address bits 6-15 are now used, we examine offset 66 of the Process Table Extension which gives the Map DIC and is "2Ø142". Using the Table under Panic 12, we find a "Write Protect Violation" has occurred. We can examine the User Map which begins at location 17ØØ and find that only Page Ø is mapped to physical 142. All the other pages are "Validity" protected. This agrees with the section under "Discussion".

From a discussion with the customer, we find the following system activity:

PID	Process
1	PMGR
2	OP-CLI (CON Ø)
3	CLI (CON 2)
4	MASM (CON 2)

Since PID=3 was also executing the CLI and had not trapped, it's User Map should contain the physical pages for the CLI. To again examine the PID Table, we must remap to the System Map using the command "MAP SMAP". Note that we had been only examining Physical Page 162. Checking the PID #3 offset we find the Process Table Extension at 6Ø4ØØ and the User Map Table at 6Ø7ØØ.

The User Map Table shows that the "Sharable" portion of the CLI, logical addresses 4ØØØØ-77777, are using contiguous Physical Pages 76 thru 115. For 8K modules, this would be modules



(PIDLN) = length of the PID Table

PIDBT = A Table of $20_{(8)}$ words in length where each bit references a PID #. If the bit is a "1", the PID # is "In Use"; if a "Ø", the PID # is available.

For example:

Word/Bit Position	PID #(10)
Word Ø, Bit Ø	Ø
Word Ø, Bit 1	1
Word Ø, Bit 5	5
Word 1, Bit Ø	16

Figure 6.1Ø.3 Relationship Between PID and Process Table(s)

```
) X SYSDMP DUMPØ513
AOS SYSTEM DUMP ANALYZER REV 1.Ø
+ STAB
FILENAME?  :SYSGEN:AOS22.ST
+ SYSIN:ØØØØØ1 +
+ INTLV:177777 +
+
+ PIDTB:Ø4226Ø +          PID Table Address
+ 4226Ø+2:Ø57ØØØ +        PID = 2
+
+ 57ØØØ+12:ØØ6222 +       PFLAG
+ 57ØØØ+16:Ø614ØØ +       Proc Tbl Extension Addr
+ 57ØØØ+25:Ø617ØØ +       User Map Addr
+ 57ØØØ+75:ØØ6162 +       Process's Map Slot 60K
+ CUR6Ø:Ø61562 +          Log/Phys for 6ØØØØ-61777
+
+ MAP 162                 Map it !!!!!
+
+ 1466:Ø2Ø142 +           Map DIC
+
+ 17ØØ:ØØ1142 +           Page Ø
+ 17Ø1:ØØ3377 +           Page 1
+ 17Ø2:ØØ5377 +           Page 2
+ 17Ø3:ØØ7377 +           Page 3
+   :     :                 :  :        PID = 2
+ 172Ø:Ø41377 +           Page 2Ø
+ 1721:Ø43377 +           Page 21
+   :     :                 :  :
+ 1737:Ø77377 +           Page 37
+ MAP SMAP
+ 4226Ø+3:Ø575ØØ +        PID = 3
+
+ 575ØØ+12:ØØ4ØØ2 +       PFLAG
+ 575ØØ+16:Ø6Ø4ØØ +       Proc Tbl Extension Addr
+ 575ØØ+25:Ø6Ø7ØØ +       User Map Addr
+ 575ØØ+75:ØØ2162 +       Process's Map Slot 6ØK
```

Figure 6.1Ø.4 Panic Code 1Ø Analysis Example

```
+ MAP 162
+
+ 7ØØ:ØØ1117 +          Page Ø
+ 7Ø1:ØØ3377 +          Page 1
+ 7Ø2:ØØ5377 +          Page 2
+   :    :              :   :
+ 717:Ø37377 +          Page 17
+ 72Ø:Ø41315 +          Page 2Ø
+ 721:Ø43314 +          Page 21
+ 722:Ø45313 +          Page 22
+ 723:Ø47312 +           :   :
+ 724:Ø51311 +           :   :
+ 725:Ø5331Ø +           :   :
+ 726:Ø553Ø7 +           :   :
+ 727:Ø573Ø6 +           :   :
+ 73Ø:Ø613Ø5 +           :   :          PID = 3
+ 731:Ø633Ø4 +           :   :
+ 732:Ø653Ø3 +           :   :
+ 733:Ø673Ø2 +           :   :
+ 734:Ø713Ø1 +           :   :
+ 735:Ø733ØØ +           :   :
+ 736:Ø75277 +           :   :
+ 737:Ø77276 +          Page 37
+
```

Figure 6.1Ø.4 Panic Code 1Ø Analysis Example (Continued)

7, 1Ø, and 11; for 16K modules, this would be modules 3 and 4. (Note that these exclude the Page Ø page).

We now know the physical core pages being used by the CLI. We can also try to determine the failure mode. Using the DEDIT utility and examining the PC locations, we note that location 77573=454Ø4=STA 1,4,3. (Refer to Figure 6.1Ø.5) Since AC3= 63225 and remember the logical addresses from 4ØØØØ-77777 are Write-Protected, we know the cause of the Trap. Since it appears that AC3 was loaded wrong or we jumped to the wrong section of code, that is as far as we can go. In some of these cases a picked or dropped bit can be noted. At this point, you would need additional software support to possibly examine a CLI listing or you can change one or more memory modules, then try to have the failure re-occur.

```
)
) X DEDIT CLI.PR
AOS FILE EDITOR REV 1.Ø
+
+ 77573:Ø45404 +
+ 77574:Ø24040 +
+
+ MODE N
+
+ 77573:STA 1 4 3  +
+ 77574:LDA 1 40 Ø  +
+
+ BYE

)
```

Figure 6.1Ø.5

DEDIT Analysis of CLI.PR showing 77573 = STA 1,4,3

PANIC CODE 11 - AOS DATA BASE CHECKSUM ERROR

AC3 = SMOND+

(SCHED) When the Operating System is idle, the "Scheduler" calculates a checksum on the finite area between ZCKST and up to but not including ZCKEN. Also included are locations 2Ø-37. Each word is "XOR'd" into accumulator Ø. If the checksum calculated in ACØ does not agree with a known good value stored in CKSUM, A Panic will occur. At the time of Panic:

ACØ - Calculated Checksum
AC1 - Contents of CKSUM

Discussion:

The checksum is calculated between ZCKST (5Ø) and ZCKEN-1 (3Ø5)* also including locations 2Ø-37. The problem with this method is that it tells you there is a problem after all the locations have been added together. The actual location that caused the error is unknown. Also because an "XOR" is being done, you can tell the bit(s) that failed by comparing ACØ and AC1 but you can not tell whether the bits were picked or dropped.

A procedure to determine the actual failing location is as follows:

1. Take a core dump of the actual failure
2. Run FIXUP on the disk(s)
3. Re-boot the same Operating System
4. Try to do the exact same operation as when you previously "Panic'd"
5. Stop the CPU and take a 2nd core dump on a different Mag Tape.
6. Again run FIXUP on the disk(s)

* note that this value was changed to 3Ø3 for AOS 1.Ø6 on

7. Re-boot the system and load both core dumps onto the disk via CLI commands such as:

) COPY DUMPØ515.2 @MTAØ:Ø

) COPY GOOD @MTA1:Ø

8. Use the System Utility, FILCOM, to compare the GOOD and bad core dumps such as:

)X FILCOM/L=@LPA GOOD DUMPØ515.2

9. After the Utility has printed up to location 5ØØ, you can type "↑C↑B" on the console to abort the Utility.

10. Compare the good vs bad for locations 2Ø-37 and 5Ø-3Ø5* and try to determine the failing locations.

Since we know which bits were effected from a comparison of ACØ and AC1 at the time of Panic, the above procedure may give us locations which do not agree for which we can check for those bit differences.

Example:

The following page shows an example, Figure 6.11.1, of a Panic Code 11 Typeout. From a comparison of ACØ and AC1, we can determine that bit 15 was either picked or dropped. I took a core dump and ran FIXUP on the disk. I re-booted, got the system to the same place I had been, and pressed "STOP". I took a second core dump and again ran FIXUP. I re-booted and loaded both core dumps onto the disk - the bad one called "DUMPØ515.2" and the controlled core dump called "GOOD". I then ran the FILCOM Utility on both dumps and examined the output as shown in Figure 6.11.2. The only difference was location 31 where the good was 4 and the bad was 5. Therefore, I had picked bit 15 in physical location 31. My first action would be to change whatever core module contained location 31. Note that this could vary because of the interleaving in the system.

*Note: changed in AOS 1.Ø6 on to 3Ø3

```
FATAL AOS ERROR - 11 036343 036342 000040 002073
                        AC0     AC1     AC2     AC3
AOS CORE DUMP
LOAD TAPE FOR DUMPING ON NTAC
STRIKE ANY KEY WHEN READY

AOS CORE DUMP COMPLETED
```

Figure 6.11.1 Sample Panic Code 11 Typeout

```
                GOOD     DUMPC915.2

000000          024722   015622
000010          000131   000104
000031          000014   000005
000041          032032   037043
000305          036440   045004
000307          076423   076447
000311          075415   075447
000313          000044   000143
000314          032433   076423
000321          000041   000001
000322          057564   075533
000326          000430   001156
000327          000000   000761
000330          000037   020037
000331          100100   000001
000334          135771   000213
000335          000163   000156
000337          076423   076447
000342          173531   176015
000343          000010   000003
000346          000001   000001
000351          163564   163777
000354          075755   075735
000355          132553   177343
000365          036440   046000
000367          021325   024435
000436          000131   000000
000447          036440   046001
000451          110110   150004
```

Only Locations
5Ø - 3Ø5 *
2Ø - 37
Used by Checksum
Routine

* changed to 3Ø3
for AOS rev 1.Ø6
on

Figure 6.11.2

Sample FILCOM Output of the Panic Core Dump
with a "Controlled" Core Dump

PANIC CODE 12 - PERIPHERAL MANAGER TRAPPED (PMGR)

General Information:

The Peripheral Manager (PMGR) is a Process which executes under AOS at PID = 1. It handles the I/O to all byte type devices including the ALM irrespective of whether a DCU5Ø is involved. I/O to all block type devices is handled by the AOS Operating System.

Specifics:

AC3 = 74133, 75133

(PRCNG) Term + 61 A trap has occurred in the PMGR process. This is a fatal situation and causes a Panic.

At the time of Panic:

AC1 = 1 (PID)

AC2 = Process Table Address

AC3 = OSER+
UBLK+
MXMD+

(AOS 1.Ø6 on)

(PERDR) This Panic can occur from seven different places for one of two reasons. These are:

a) Input or Output Done is being attempted to be set twice.
b) Input Done is being attempted to be set on a PIB that is not on the SUNB queue.

Both of these problems are complex including many of the PMGR data bases. The analysis of this case will not be covered within this document because of the depth of understanding needed. If this case occurs, submit a core dump in accordance with the procedures at the front of this section.

Discussion:

Since the Peripheral Manager executes very similarly to a User Program, we could troubleshoot the Panic as if it were a User Trap. In attempting to evaluate a "Trap", there is certain information that we would like to know:

a) PC at the time of Trap
b) AC's at the time of Trap
c) Type of map violation
d) Physical core being used when the trap occurred.

We should first verify that AC1 in the Panic message is actually a "1". This should always be a "1" if the PMGR caused the trap.

Refer to Figure 6.12.2. AC2 in the Panic message is defined as a Process Table Address. The Process Table is similar in usage to the"Program Table" under RDOS. Offset 12 (PFLAG) in the Process Table is a "Flag word" where bit 5 indicates whether a trap occurred. Offset 16 (PEXTN) points to a Process Table Extension which contains the "MAP DIC", the accumulators, and PC at the time of trap. The following offsets are used:

(Con't on Page 6-86)

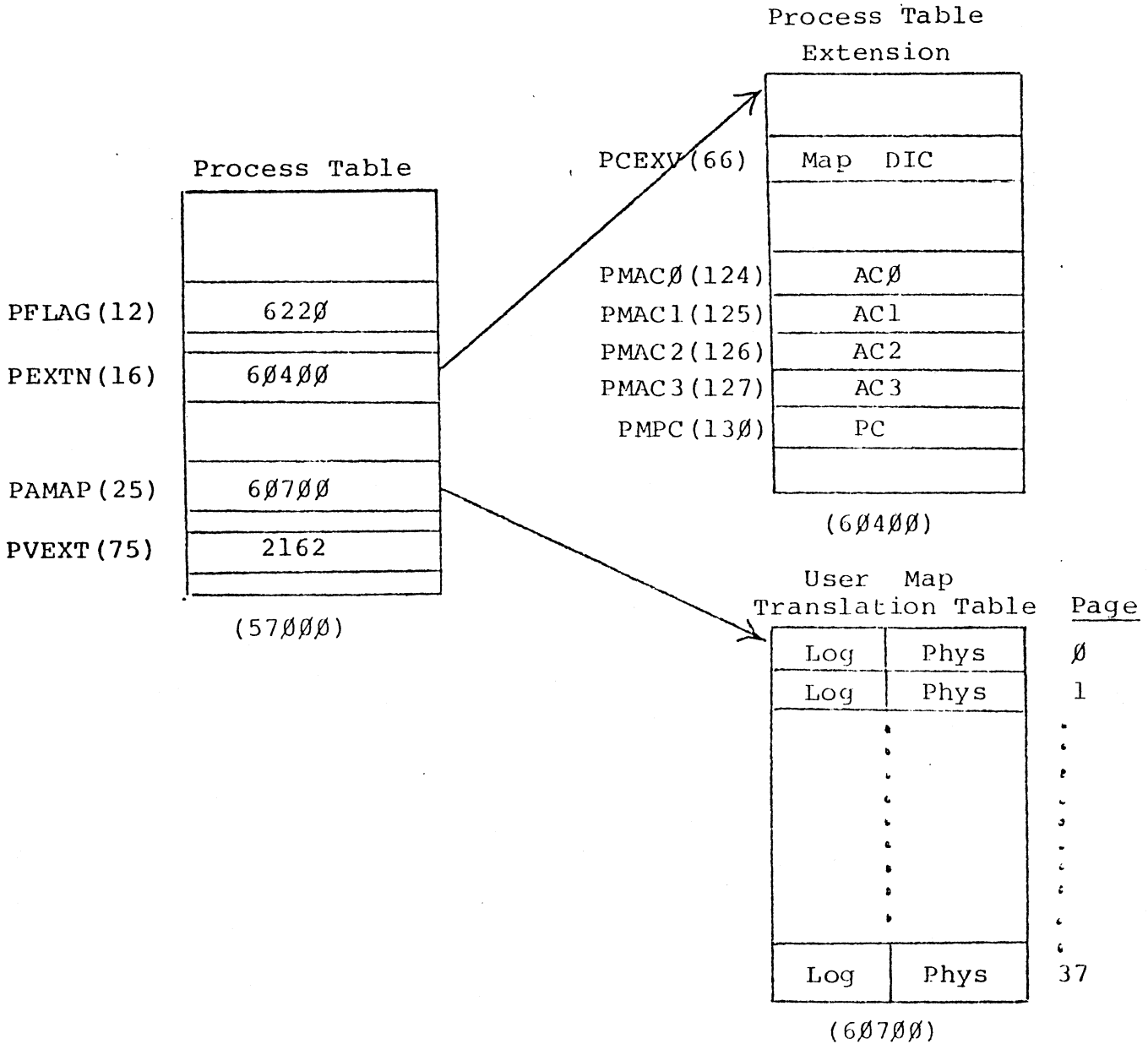


Figure 6.12.1 Relationship Between Process Table and needed Trap Parameters

```
FATAL ACS ERROR - 12 1:2401 000001 057000 074133
                        AC0    AC1     AC2     AC3
ACS CORE DUMP
LOAD TAPE FOR DUMPING ON MTA0
STRIKE ANY KEY WHEN READY

ACS CORE DUMP COMPLETED
```

Figure 6.12.2 Sample Panic Code 12 Typeout

```
) DIR :
) SUPER ON
*) X DEDIT PMGR.PR
ACS FILE EDITOR REV 1.0
+
+ 55:011164 +          Real Contents of
+ 143:100143 +             Page 0 Loc's
+
+ BYE

*)
```

Figure 6.12.3

DEDIT of PMGR.PR to find Proper Values for Indirect Words

Offset	Meaning
66	Map DIC
124	AC0
125	AC1
126	AC2
127	AC3
130	PC

By examining the Map DIC, we can determine whether the trap was caused by a Write, Validity, I/O, or Defer Protect Violation. The bit masks for these positions are:

C300	Violation
20000	Write Protect
10000	Validity Protect
4000	Indirect Protect
2000	I/O Protect
20	Data Base Error

Note that on a C330 type machine, the DIC error bits are converted to their C300 bit positions prior to storing in Offset 66. Consequently, Offset 66 will always appear to have a C300 type Map Status in it irrespective of the hardware.

If the DIC location contains a "20", a "Data Base" Trap has occurred. This is further explained in Section 5 which discusses Traps. Note that no other bits should be set for this trap.

If the DIC location contains a "4000" which indicates a "Defer" Violation, there is a good chance that the Panic was caused intentionally by the PMGR. The contents of the Trap PC can be correlated with the "Trap PC" addresses in Table 6.12.1 to determine the reason for failure. In some cases the contents of AC0 and AC1, offsets 124 and 125 respectively, may be needed to further define the problem. It is important to note whether the Panic code 12 occurred prior to the Op CLI (PID=2) typing the initial message:

"AOS CLI REV 1.03 11-SEP-77 14:51:08"

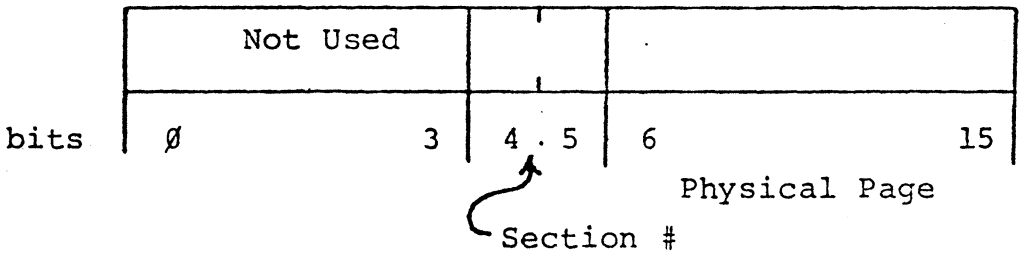
If it did, there may be a problem in the way the booted system was AOSGEN'd. The Panic description from Table 6.12.1 can be used in addition to the AOSGEN dialogue to determine the reason

for failure. Note that the values given in Table 6.12.1 for the "Trap PC" will be exact for a Rev 1.Ø3 PMGR. In future or previous revisions, the values may vary slightly. Table 6.12.2 is also provided which lists the AOSGEN Byte Device Types.

Since we got to this point, the DIC indicated a Write Protect, a Validity Protect or an I/O Protect violation. We have now answered questions a) through c) which were posed earlier.

To determine the physical core being used by the Process, we can examine offset 25 (PAMAP). This points to a Table where the actual map words are contained to do the translation of Logical to Physical for Logical Pages Ø thru 37. (A User context can still only be 32K wds.). We can now use the PC to determine which logical page the error occurred in. From the Table, we can determine the physical page and thereby the physical location of trap.

Since AOS uses the same 6ØØØØ slot to look at the Process Table Extension and User Map Translation Table for all processes, it needs a pointer to indicate which physical page contains those two tables for a given process. Offset 75, PVEXT, is used to contain this information. AOS divides a memory page into 4 sections. Each section contains the two tables for one process, therefore a given memory page contains the tables for four processes. Offset 75 contains the physical page and the section for the respective process in the following format:



where the section can be defined as:

Section #	Words within 1K Page
ØØ	Ø-377
Ø1	4ØØ-777
1Ø	1ØØØ-1377
11	14ØØ-1777

The "Section #" is not needed as far as we are concerned since Offsets 16 and 25 contain full addresses which implicitly contain the respective offset within the 1K page in address bits 6-15. The above was only for your reference in case bits 4 and 5 are found set.

Note that "CUR6Ø" will not always point to the proper physical page for the 6ØØØØ slot since it can only point to four Processes at any given time and there can be many more active in the system. Always use PVEXT to determine the 6ØØØØ's slot respective physical page.

At this point, we can attempt to determine the reason for the trap but since we do not have listings for the PMGR program, we can swap out the physical memory that the PMGR resides in module by module and can attempt to duplicate the problem.

Additionally, any of the following can be done.

1) Run EMORT L for a minimum of 12 hours to try and find any memory problems
2) Do a CRUNALL* which will run the list of programs in addition to DCH activity.
3) If the problem only occurs when Comm gear is running, try exercising the system under AOS without using any of the Comm. lines. If the system has both consoles, 1Ø/11 and 5Ø/51, use those to run programs.
4) If the error only occurs with the Comm. gear running and the Comm. chassis is cabled from the extended I/O bus connector, verify whether ECO#4167A is installed which cures a "Y" termination problem on the INTP/DCHP signals.

I have assumed from the previous software discussion that the problem is a memory failure. In fact, it may be a Map or CPU which is why I suggested the CRUNALL under 2) above.

* "CRUNALL" will only exercise a disk on Device Code 33. Do a "RUNALL" for all others including a Zebra (6Ø6Ø,6Ø61).

Example:

The following figure, Figure 6.12.2, shows an example of a

Panic Code 12 type-out. Figure 6.12.4 on the next page is a sample SYSDMP Analysis of the sample Panic.

First, verify that AC1 = 1. AC2 gives us the address of the Process Table which is 57ØØØ. We can then check the Process Table for the addresses of the Process Table Extension and the User Program Map. Since these are in the 6ØØØØ logical address range, we must check offset 75 to determine the Logical to Physical translation for that page. After mapping the page and remembering to use only bits 6-15 for addresses within this 1K page, we found:

```
Map DIC = 1ØØ24
    ACØ = 1
    AC1 = 613
    AC2 = 177777
    AC3 = 423
     PC = 32Ø43
```

The "Map DIC" using the table in the "Discussion" section tells us that we had a "Validity Protect" violation. Note that the PMGR will intentionally cause a Panic 12, as discussed before, by doing an "infinite Defer" when it finds information in its data bases that cannot be resolved. This is accomplished within the PMGR code via:

```
PANIC : @.
```

If the Map error type indicated by the DIC is a "Defer" error, refer to the prior discussion and Tables 6.12.1 and 6.12.2 to attempt to resolve it. If the cause does not appear to be readily apparent, I will request a core dump to be submitted. Let's next examine the physical slots used by the PMGR.

Using the PAMAP offset, we found and typed out the User program space beginning with location 7ØØ (bits 6-15!!). The PMGR occupies two areas of core:

1) Physical Page 24
2) Physical Pages 14Ø - 156

Note that Physical Pages 14Ø - 156 are all contained in Module 14 and 15 for 8K modules or Module 6 for 16K modules.

```
) DIR :UTIL
) COPY DUMP0523.3 @MTA1:0
) X SYSDMP DUMP0523.3
AOS SYSTEM DUMP ANALYZER REV 1.0
+ STAB
FILENAME? :SYSGEN:AOS22.ST
+
+ 57000+12:006223 +  (PFLAG) 1B5=trap
+ 57000+16:060400 +  PTABLE Extension
+ 57000+25:060700 +  User Program Map
+ 57000+75:002162 +  Phys Page for Logical 60000

+ MAP 162
+
+ 400+66:010024 +    Map DIC
+ 400+124:000001 +   AC0
+ 400+125:000613 +   AC1
+ 400+126:17777 7 +  AC2
+ 400+127:000423 +   AC3
+ 400+130:032043 +   PC
+
+ 700:001024 +       Page 0
+ 701:003156 +       Page 1
+ 702:005155 +       Page 2
+ 703:007154 +       Page 3
+ 704:011153 +
+ 705:013152 +
+ 706:015151 +
+ 707:017150 +
+ 710:021147 +
+ 711:023146 +
+ 712:025145 +
+ 713:027144 +
+ 714:031143 +
+ 715:033142 +
+ 716:035141 +
+ 717:037140 +
+ 720:041377  +      Page 20
+ 721:043377  +      Page 21
+
+ 737:077377 +       Page 37
+
+ MAP 24
+ 414:000423 +       Current TCB
+
+ MAP 142
+ 43:006055 +        PC @ Trap
+ 44:002143 +        PC+1 @ Trap
+
+ MAP 24
+ 55:051164 +        Page 0 Indirect Loc
+ 143:100143 +       "      "        "
+
+ BYE
```

Return Block at time of TRAP

User Space Map
Logical to Physical translation

Figure 6.12.4 Sample SYSDMP Analysis for a PMGR Trap

Wanting to try to determine the reason for the failure, I again look at the PC which is 32Ø43. I find this to be contained in Logical Page #15 which translates to Physical Page #142. After mapping physical page 142, the instruction in the PC location at the time of trap is a "6Ø55" which is a "JSR @ 55, Ø" and the contents of the PC+1 location is a "2143" which is a "JMP@143,Ø". The contents of location 143 is a "1ØØ143" which appears to be a "JMP @." and if it were ever executed, it would cause a "Defer" trap and not the "Validity" trap that we had. We can now assume that the "JSR" instruction caused the trap. Looking again at the Logical vs Physical Map, we can note that logical space only goes up to logical address 37777. Therefore, I would suspect the address of "51164".

By doing a DEDIT, as shown, of the PMGR.PR, we find that the real contents of location 55 are 11164 not 51164 which is a pick of bit 1 in module #2 for 8K's or module #1 for 16K's. Note that interleaving would also change the module number!!

Table 6.12.1 PMGR "DEFER" Trap
Error Table

Trap PC (approx Address) PMGR Rev 1.Ø3	Trap PC (approx Address) 1.Ø5	Module/ Offset (approx)		Meaning
1176	1170	PMGR/RMEM+1Ø		Attempting to release a memory page below base of memory pool
12Ø3	1175	PMGR/RMEM+15		Attempting to release a memory page already free in bit map
1344	133Ø	PMGR/RIPCH+2		(Same as RMEM+1Ø)
1351	1335	PMGR/RIPCH+7		(Same as RMEM+15)
1364	1347	PMGR/IPCER+6	*	IPC ?ISEND error
1371	1354	PMGR/IPCER+13		Counter of IPC ?ISEND errors overflowed (65k)
14Ø1	1364	PMGR/IPCER+23	*	?DELAY error
1432	1415	PMGR/TIMER+4	*	?MEMI error
1443	1426	PMGR/TIMER+15	*	?DELAY error
1735	1716	CONCO/DRPIR+3		Counter of IPC ?IREC errors overflowed (65k)
227Ø	2245	CONCO/ASAME+41	*	?XLATE error
232Ø	2275	CONCO/ASAME+71	*	?XLATE error
2415	2372	CONCO/OSTRC+41	¢	?SGNL error
2534	25Ø7	CONCO/PIBRE+4		Trying to release a PIB not on the owner's PIBLK chain
3Ø17	277Ø	CONCO/PIBRE+267		PID of caller=Ø or 1
3Ø24	2775	CONCO/PIBRE+274	*	?GHRZ error
3Ø33	3ØØ4	CONCO/PIBRE+3Ø3	*	?DELAY error
3Ø42	3Ø13	CONCO/REUB		(Not used)
3173	3141	CONCO/RKILL+13Ø		Phoney IPC header-No task to kill
34Ø1	3346	CONCO/RKILL+336		Owner's PID does not match
35Ø2	3447	CONCO/RKILL+437		State Save Area Address=Ø
3523	347Ø	CONCO/RKILL+46Ø	¢	?SGNL error
3777	3744	CONCO/RKILL+734		Timer task (?DELAY) at Priority =Ø cannot be found
4114	4Ø61	CONCO/RKILL+1Ø51	*	?DELAY error
6Ø54	5757	IOCOM/READ+1Ø64	+	Magic character problem
6Ø7Ø	5773	IOCOM/READ+11ØØ	+	(Same as READ+1Ø64)
6341	6244	IOCOM/LMOV+21	+	Inconsistancy with Magic character
65Ø2	64Ø3	IOCOM/BLCMN+16	+	(Same as RKILL+734)
745Ø	734Ø	IOCOM/ICHR+1Ø4		Error on device start
1Ø415	1Ø312	SCEDT/SCIN+234	+	Magic Character inconsistancy on "Erase Line" command
1Ø431	1Ø326	SCEDT/SCIN+25Ø	+	Magic Character inconsistancy on "Move Right 1 column" command
11172	11Ø76	TASK/?URTB		Dummy entry points to fool linker -?URTB, ?XLD, ?XSV, ??IME, ??DEL
11232	--	TASK/WAKE+33		Unlinking TCB which cannot be found on active TCB chain
11247	--	TASK/WAKE+5Ø		Last TCB was killed-should never happen
--	11113	TASK/SCRTN+7		65K "Disable Task Rescheduling" has been done
--	11123	TASK/SCRTN+17		Enable Task Resched w/o Disable having been done

PMGR	Trap PC (approx Address) Rev 1.Ø3	Trap PC (approx Address) 1.Ø5	Module/ Offset (approx)		Meaning
	~~11374~~	~~11274~~	TASK/WAKE+175 (RUN+16)%		Next location after hardware "RSTR" instruction-should never occur
	11521	11376	TASK/LTCBS+21 (KRTN+45)%		Counter of .IDKILL errors overflowed (65k)
	11536	114Ø4	START/EROST+1		Attempting to start a non-output device for output-ACØ=error code AC1=device code
	11737	116Ø5	START/ERRCL+1		AOSGEN'd illegal device type-ACØ=error code, AC1=device code
	12247	12125	CARD/CREAD+1Ø3		Error in Card Reader packed binary routine. AC1>3. (Should never occur because of a prior AC masking)
	12574	12452	DCUC/DOUST+15	$	"Host to DCU5Ø" Queue wrapped around (PMGR rev 1.Ø6, see Pg 6-94A)
	31227	311Ø5	INIT/START+14	*	?SUSER error
	31234	31112	INIT/START+21	*	?PNAME error
	31236	31114	INIT/START+23		My PID=2; should always be "1"
	3125Ø	--	INIT/START+35		No data in PERTB (AOSGEN spec.)
	31277	31155	INIT/START+64		No ALM lines specified (PERTB)
	3135Ø	31255	INIT/START+135		Line Table data over-ran in PERTB
	314Ø3	3131Ø	INIT/START+17Ø		Non-Mux device data over-ran in PERTB
	31417	31324	INIT/START+2Ø4	@	Illegal device from AOSEGEN-ACØ= error code, AC1=device type
	31424	31331	INIT/START+211	#	Trying to assign greater than $255_{(10)}$ Ports.
	31444	31351	INIT/START+231		Too many PIBs.
	31455	31362	INIT/START+242		PIBs do not fit within 1k
	31517	31424	INIT/START+3Ø4		Tables being built would overwrite the Init code
	31555	31462	INIT/START+342		Invalid Buffer Address in ACØ-bit Ø="1"
	31762	31667	INIT/BUILD+111	*	?DPMGR error
	32ØØØ	317Ø5	INIT/BUILD+127		Device is neither Mux nor Non-Mux
	32Ø23	3173Ø	INIT/BUILD+152		Present line# > Max Line#
	321Ø1	32ØØ6	INIT/BUILD+23Ø		More data than should be during PIB build
	32155	32Ø62	INIT/BUILD+3Ø4	*	?SUSER error
	32166	32Ø73	INIT/BUILD+315		.TASK error-ACØ=error code
	3224Ø	32145	INIT/INITM+51	*	?XLATE error
	32265	32172	INIT/INITM+76	@	Illegal device type from AOSGEN-AC1 > $47_{(8)}$ for MUX
	32361	32266	INIT/INITM+172	@	Illegal device type from AOSGEN-AC1 > $47_{(8)}$ for Non-MUX

% Offset changed in PMGR Rev 1.Ø5

PMGR	Trap PC (approx Address) Rev 1.Ø3	Trap PC (approx Address) 1.Ø5	Module/ Offset (approx)	Meaning
	~~32544~~	~~32451~~	INIT/INITM+355	* ?DPMGR error
	32563	3246Ø	INIT/INITM+374	* ?DPMGR error
	32574	325Ø1	INIT/INITM+4Ø5	* ?CREATE error
	37Ø66	36773	DCUI/DLOAD+7	Not enough DCU-5Ø Map slots were mapped (Should never happen)

NOTES:

* Error occurred on a System Call-ACØ=error code.

¢ Error occurred on a ?SGNL System Call-ACØ=error code. This system call is used by the PMGR to send an "IPC" type message to the AOS root code. AOS can "suspend" the Process waiting for the PMGR to send it this message. (See PFWSL in PFLG2)

+ Some types of console displays require a two character sequence for cursor movement. For example, an "Erase Line" command might be an "Escape" character followed by a "K" character. The first character, in this case the "Escape", is referred to as the Magic character. A Table in the PMGR indicates whether this additional character is needed.

PMGR IPC Ports are assigned on a per device basis, with the number of ports needed depending on the device.

For example:

Device	# Ports	Port Types
Console	3	Read, Write, Control
Line Printer	2	Write, Control
Card Reader	2	Read, Control

The total number of PMGR IPC ports must be less or equal to $255_{(10)}$ for all AOSGEN'd devices whether active or not.

@ Refer to Table 6.12.2 for the AOSGEN Byte Device Types. Also, check the AOSGEN "Spec" file, ?PID.SPEC.TMP to try and find the invalid value.

$ This type failure can occur during the UP macro if the presently running AOS operating system has been AOSGEN'd for a DCU-5Ø w/ALM's, but the DCU5Ø controller had been removed from the chassis. This tends to confuse the PMGR during line initialization.

```
DCUK:    0          ;CALLER TO HERE
DCUB7:   0          ;=0=RUNNING, =-1=STOPPED
DCUDN:   0          ;=0=NOT DONE, =-1=DONE
DCUPC:   0          ;PC OF STOPPED DCU AFTER RESET (IN AC1)
DCUD1:   0          ;DIAGNOSTIC DATA BEFORE RESET (IN AC0)
DCUD2:   0          ;DIAGNOSTIC DATA AFTER RESET (IN AC2)

; DIAGNOSTIC DATA STATUS BITS :
;
; 100000=           SETCARRY
; 040000=           (NOT) SETSKIP
; 020000=           PULSE
; 010000=           (NOT) DEFER
; 004000=           CPUCLK
; 002000=           SCWRTIF
; 001000=           DP2
; 000400=           DP1
; 000200=           1RADR0
; 000100=           1RADR1
; 000040=           1RADR2
; 000020=           2WPADR0
; 000010=           2WPADR1
; 000004=           1WADR0
; 000002=           1WADR1
; 000001=           CUMADR2
```

AOS Rev 1.Ø6

Log Page	Offset
5	1Ø3Ø
5	1Ø31
5	1Ø32
5	1Ø33
5	1Ø34
5	1Ø35

Notes:

1. "PC" in Extension at the time of Panic == DOUST+52
2. These checks were added to determine the state of the DCU hardware. Information is being put on the queue for the DCU to take but it is not being taken. Cause --- DCU hardware died; Code in the DCU is not functioning properly; etc.
3. A routine is being added into AOS Rev 1.Ø8 to dump the code in the DCU on a Panic onto the end of the Core Dump Mag Tape so it can be checked later.

"Host to DCU Queue Wrap-around" Status indicators
(AOS Rev 1.Ø6 onward)

Table 6.12.2 AOSGEN Byte Device Types

Device Type Octal (Decimal.)	Mnemonic	Description
1 (1.)	TTY	4Ø1ØA Teletype or 6Ø4Ø Dasher
2 (2.)	CRT1	4Ø1ØI Infoton
3 (3.)	CRT2	6Ø12
4 (4.)	CRT3	6Ø52
5 (5.)	CRT4	"Other" CRT
6 (6.)	PSEU	(not used)
24 (2Ø.)	RTAPE	4Ø11 Paper Tape Reader
25 (21.)	RCARD	4Ø16 Card Reader
26 (22.)	PTAPE	4Ø12 Paper Tape Punch
27 (23.)	PLINE	4Ø34 Line Printer (non-DCH)
3Ø (24.)	PLOTR	4Ø17 Digital Plotter
36 (3Ø.)	DCS1	Mux with DCU-5Ø
37 (31.)	DCSØ	Mux without DCU-5Ø

PANIC CODE 13 - MULTI-BIT ERCC ERROR

General Information:

The older 21-bit 8Kw memories and the new 32Kw MOS memories have a feature called Error Checking and Correction (ERCC). There is a 5 bit "check field" constructed each time a word is written. The "check field" is reconstructed and compared with the original field each time the memory word is read. This "check field" allows the hardware to correct single bit errors and to interrupt on non-correctable multi-bit errors.

Specifics:

AC3 = IERCC + 26

(INTS) A multi-bit ERCC error has occurred. Note that these errors are non-correctable. At the time of Panic:

For a C3ØØ or S2ØØ:

ACØ = low order 16 bits of the failing 17 bit address

AC1 = bits Ø-4 = fault code
bits 5-14 = all Ø's
bit 15 = high order bit of the 17 bit physical address

For a C33Ø or S23Ø:

ACØ = low order 16 bits of the failing 18 bit address

AC1 = bits Ø-4 = fault code
bits 5-13 = all Ø's
bits 14+15 = high order bit of the 18 bit physical address

For a S13Ø:

ACØ = Bits Ø-3 = complement of bits 1-4 of the 17 bit physical addr in error

bits 4-11 = reserved
bits 12-15 = complement of the low order 4 bits of the 17 bit physical address in error.

AC1 = bits Ø-4 = fault code
bits 5-14 = all Ø's
bit 15 = high order bit of the 17 bit physical addr.

Be aware that because of the changes in the DIA/DIB output on an S13Ø, a failure can only be troubleshot to a 4K section of core. Note that bits 12-15 of ACØ are used to determine the failing memory on an interleaved machine.

Since the 32K MOS memories use 4Kx1 RAMs (DGØ59Ø), by knowing the bit that failed and the 4K section, the failing chip can be replaced. If a "Multi-bit" error occurs though, the problem of which RAM within 4K gets more complicated.

Note that the following fault codes are defined as multi-bit ERCC codes:

Code (ACØ)	Meaning
Ø5ØØØX	All 21 bits are "1"
124ØØX	All 21 bits are "Ø"
Ø3ØØØX	Multiple bit error
Ø74ØØX	Multiple bit error
134ØØX	Multiple bit error
144ØØX	Multiple bit error
154ØØX	Multiple bit error
164ØØX	Multiple bit error
17ØØØX	Multiple bit error
174ØØX	Multiple bit error

Discussion:

1. Remember that the recoverable ERCC errors are logged in the SYSLOG disk file. This file can be examined using the Report Utility as outlined in Section 7. The information that

is logged on soft errors is:

a) ERCC code

b) Physical Address

Note that the SYSLOG must be started by the User in order for errors to be logged.

2. If greater than $256_{(10)}$ recoverable ERCC errors occur since AOS was initially boot'd, AOS will output an error message of the form:

"Single bit Error, Physical Address
XXXXXX, Code nnn"

and will continue operating in Mode 2 - "Enable checking and correction but do not interrupt on memory error".

3. By using the Physical Address in the Panic and the contents of the SYSLOG file, the failing memory can be replaced.

4. The counter that sensed the $256_{(10)}$ soft ERCC errors is called "NERCC" and for a Rev. 1.Ø AOS system resides in physical location 1Ø33. This location begins at $4ØØ_{(8)}$ and is decremented to Ø. It can be monitored from the console when the system is idle.

Examples:

None

Indepth AOS Information:

None

PANIC CODE 14 - INTERNAL DATA CONSISTANCY ERROR

General Information:

This Panic can occur from at least $25_{(10)}$ different places. It covers failures in just about all the different Data Bases (a kind of "catch-all"). The failures that will cause this Panic are as stated, internal inconsistancies and are not easy to understand much less troubleshoot. At this point, the scope of these problems is above the depth of this document so presently this Panic Code will remain undocumented with the exception of the one case which follows. If any of the other cases occur, submit a dump in accordance with the procedures outlined at the beginning of this chapter.

Specifics: (For only this one case)

AC3 = NQBHR+ (BUFIO) Before a disk request is started, a check is made to determine whether the logical disk block address being requested to be read or written is greater than the maximum number on the given disk. If it is, this panic will result. At the time of panic:

ACØ = Max address (low or high word)
AC1 = requested address (low or high word)
AC2 = buffer address

Each disk type has its own set of maximum addresses which are given in the following table:

Maximum Address

Disk Type	High Word	System Disk Low Word	Non-System Disk Low Word
6Ø3Ø	Ø	6Ø1	1141
4234/6Ø45	Ø	22171	22531
4231	2	135720	136260
6Ø6Ø	2	155121	155461
6Ø61	5	124761	125321

The above values are for a single physical device Logical Disk Unit (LDU). In an LDU composed of multiple physical units, the addresses above will be for the 1st physical unit, with the others being an approximate multiple of the above values. For example, an LDU is composed of DPD1Ø, a system disk, and DPD14, a non-system disk, both of which are 6Ø45 type drives. The UDB addresses will be as follows:

Unit #	High Word	Low Word	Range
DPD1Ø	Ø	22171	Ø-22171
DPD14	Ø	45265	22172-45265

Note that 22171 + 22531 = 44722 which is approximately 45265. The additional 343 blocks are gained because certain tables need not be on the front end of the 2nd physical disk. Figure 6.14.1 depicts the relationship of the physical blocks on these two units. Note that there is more system information contained on the front end of DPD1Ø than on DPD14.

Discussion:

When a Panic 14 occurs and has been defined as the above case, a couple of questions need to be asked.

1. When was the last time FIXUP was run on the Logical Disk.
2. Has the Logical Disk been run on any other CPU system? This is especially relevant in a dual CPU configuration.

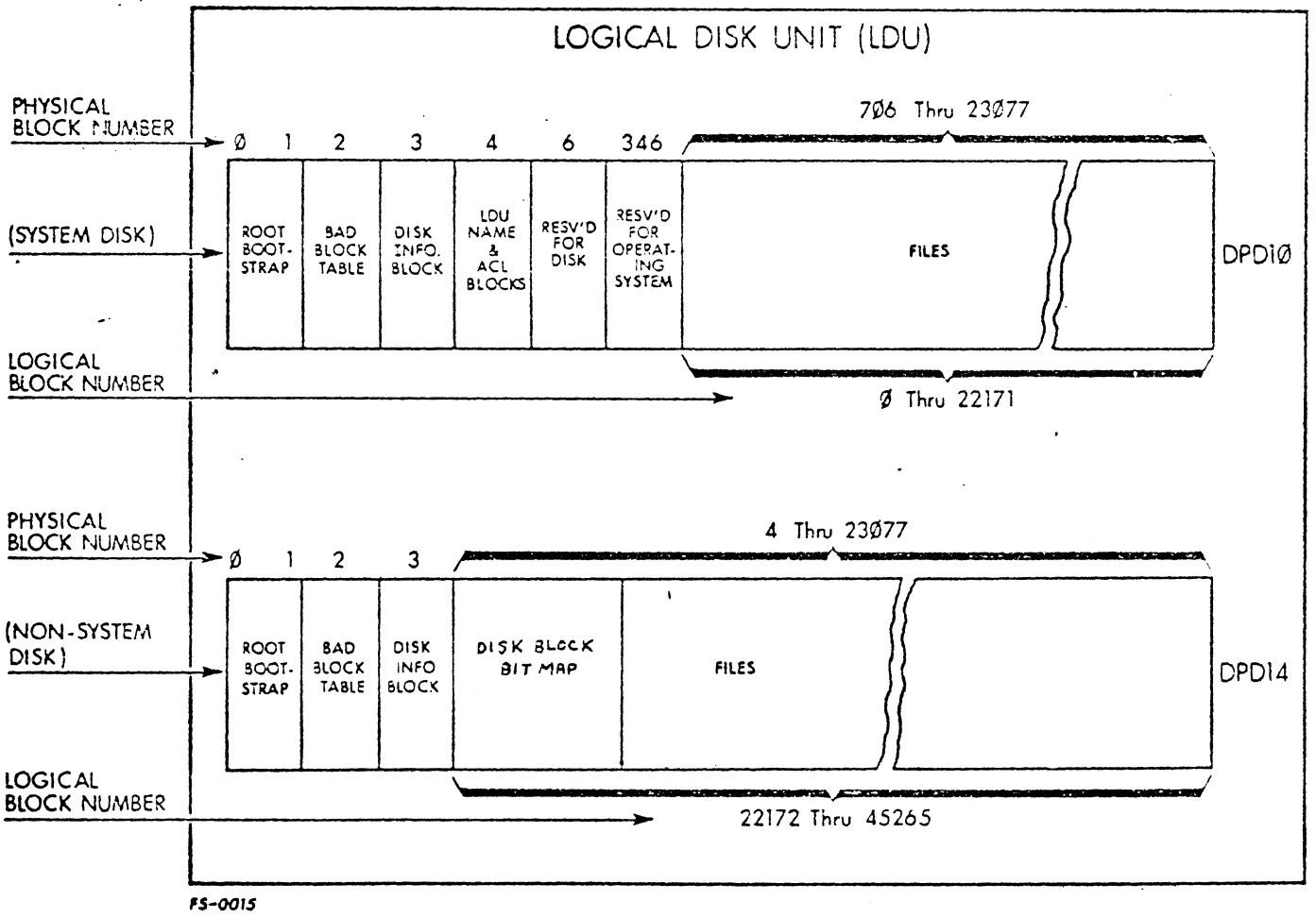


Figure 6.14.1 Two Physical Unit LDU Disk Structure

From a Field Engineering standpoint another question is whether the Panic 14 that just occurred was caused by:

a) A memory/CPU/Map failure which caused a system memory pointer to be clobbered or a disk block to be loaded in the wrong place therefore causing the panic.

b) A type a) failure which occurred days or weeks ago but because that particular file or directory had never been accessed again until now, never showed up. This is particularly true in a dual configuration where Logical Disk Units are swapped between machines.

A procedure that may help determine whether the "pointer" error was written bad on the disk or was caused by a mainframe problem is as follows:

1) Ask the customer/operator the questions outlined in 1 and 2 above.

2) Make sure a core dump was taken as outlined at the beginning of this chapter.

3) Run FIXUP on the failing Logical Disk Unit with the following specified:
   a) "Report Closing Files" - N
   b) "Error Log" - Line Printer (LPA, LPB)
   c) "Verbosity" - 2

4) After FIXUP is completed, check the Line Printer for any error messages. If there were none, FIXUP did not find any file links that it could not resolve and therefore the Panic 14 problem was probably due to a mainframe problem.

   If there were error messages, check for messages other than the following:

   "Can't Delete Root Directory"
   "Directories nested too deeply"
   "Fatal Disk Error"
   "Insufficient Memory for Bit Maps"

   Any other error message would indicate a file/directory problem on the disk.

Note: FIXUP will delete a file or directory if it finds an error that it cannot resolve.

5) If the problem was due to a mainframe problem, I would suggest running EMORT L with disk overnight, a pass of "CRUNALL" under DTOS, and running "CONTEST" for an hour or so.

6) If FIXUP did find an error on the disk, I would suggest rebuilding the pack (if possible). Prior to this, running DFMTR with all 5 surface checking patterns to check for any marginal blocks that may have appeared since the last time it was run. Note that some patterns will find errors others will not, therefore it is best to run all 5. Have the customer rebuild the pack, then let him run with Error logging (SYSLOG/START) to note any retries. EMORT L should also be run on the mainframe overnight.

7) The procedure in 6) is designed to start from a known test position:
   a) disk pack(s) checked for marginal spots via DFMTR.
   b) disk pack(s) also were rebuilt so there should be no "residual" bad information on the packs.
   c) EMORT L run on the mainframe to check for possible flakey problems.

**Note: If modules had been replaced since the last time FIXUP was run or if the disk pack(s) are run on more than one system, the above procedure may have to be modified as to CPU to exercise, etc.

Examples:

Figure 6.14.2 illustrates a sample Panic code 14 typeout. AC3 when compared using the SYSDMP utility is NQBHR+35 and is therefore the discussed case. ACØ is 22171 which is the "System Disk" low word for a 4234/6Ø45. The bad address is 42211. FIXUP was run on the Logical Disk Unit and an error message:

"Error in File :sysgen -- INVALID POINTER IN INDEX BLOCK" was typed on the logging device. This would indicate that there was a problem in the SYSGEN directory on the disk. Note that

```
FATAL AOS ERROR - 14  Ø22171   Ø42211   Ø33720   Ø27553
                        ACØ      AC1      AC2      AC3

AOS CORE DUMP
LOAD TAPE FOR DUMPING ON MTAØ
STRIKE ANY KEY WHEN READY

AOS CORE DUMP COMPLETED
```

Figure 6.14.2 Sample Panic 14 Typeout #1

```
FATAL AOS ERROR - 14  Ø22171   177777   Ø34Ø64   Ø3Ø174
                        ACØ      AC1      AC2      AC3

AOS CORE DUMP
LOAD TAPE FOR DUMPING ON MTAØ
STRIKE ANY KEY WHEN READY

AOS CORE DUMP COMPLETED
```

Figure 6.14.3 Sample Panic 14 Typeout #2

because of this error, FIXUP deleted the whole SYSGEN directory which had to be reloaded from Mag Tape. Because of this customers should periodically backup there disk packs. The procedures in step 6 should be done.

Figure 6.14.3 illustrates a second Panic 14 typeout. AC3 again points to NQBHR+35. Note that the difference in the numeric value for AC3 in the two figures is due to AOSGEN answer differences. Note that AC0 =22171 which is the same as above. The bad address is 177777. FIXUP was run on the Logical Disk Unit with no error messages. From this, I would assume a mainframe problem and proceed with Step 5 as described above.

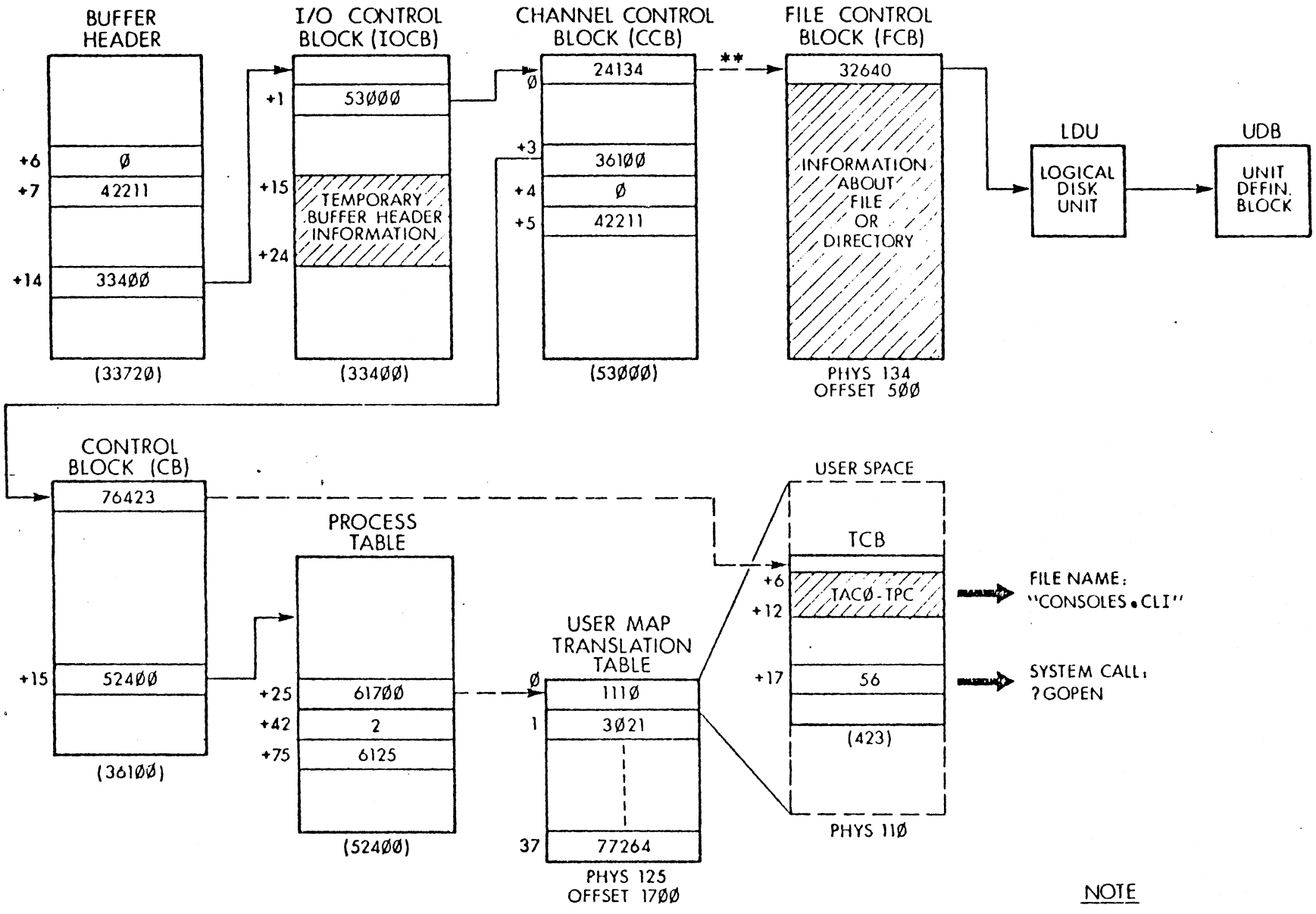
Indepth AOS Information:

I am not going to discuss all the AOS file structures at this point but will illustrate the linkages that are used so that a Panic can be traced, if necessary. This section may be updated at a later point if more information is needed for troubleshooting.

Figure 6.14.2 showed the 1st Panic 14 message. This Panic occurred while the UP macro was being executed. AC2 of the Panic message pointed to a buffer header. The file linkages for this failure are shown in Figure 6.14.4. The User, the Op CLI (PID=2), had issued a "?GOPEN" system call with a byte pointer to the filename "CONSOLES.CLI". Remember that the UP macro has the following command line:

"CONTROL @EXEC ENABLE @CON([CONSOLES])"

The Op CLI was probably searching for the macro "CONSOLES.CLI" prior to using "CONSOLES". The various offsets are defined in PARS and I will not go through them here. The bad address, 42211, appeared in the Buffer Header, IOCB, and CCB. It did not appear in the FCB. Because of various information in the FCB such as File Type, Hash Frame Size, Last byte, and Data Element Size, I could determine that the FCB belonged to the SYSGEN Directory definition. Since the first logical address appeared O.K., the error must have been at a lower level of index. I found the buffer area in core where the 1st level of index was read into. The code was:



NOTE

** ADDRESS NOT DIRECTLY TRANSLATABLE:

① BITS 6-15 = PHYS PAGE

② BITS Ø-5 = OFFSET WITHIN PAGE. HAVE TO BE RIGHT SHIFTED 5 BIT POSITIONS.

Figure 6.14.4 File Linkage For System Request

FS-0013

Offset	Contents
Ø	4766Ø
1	42211 ←
2	1523Ø6
3	1Ø6767
4	131754
5	14665

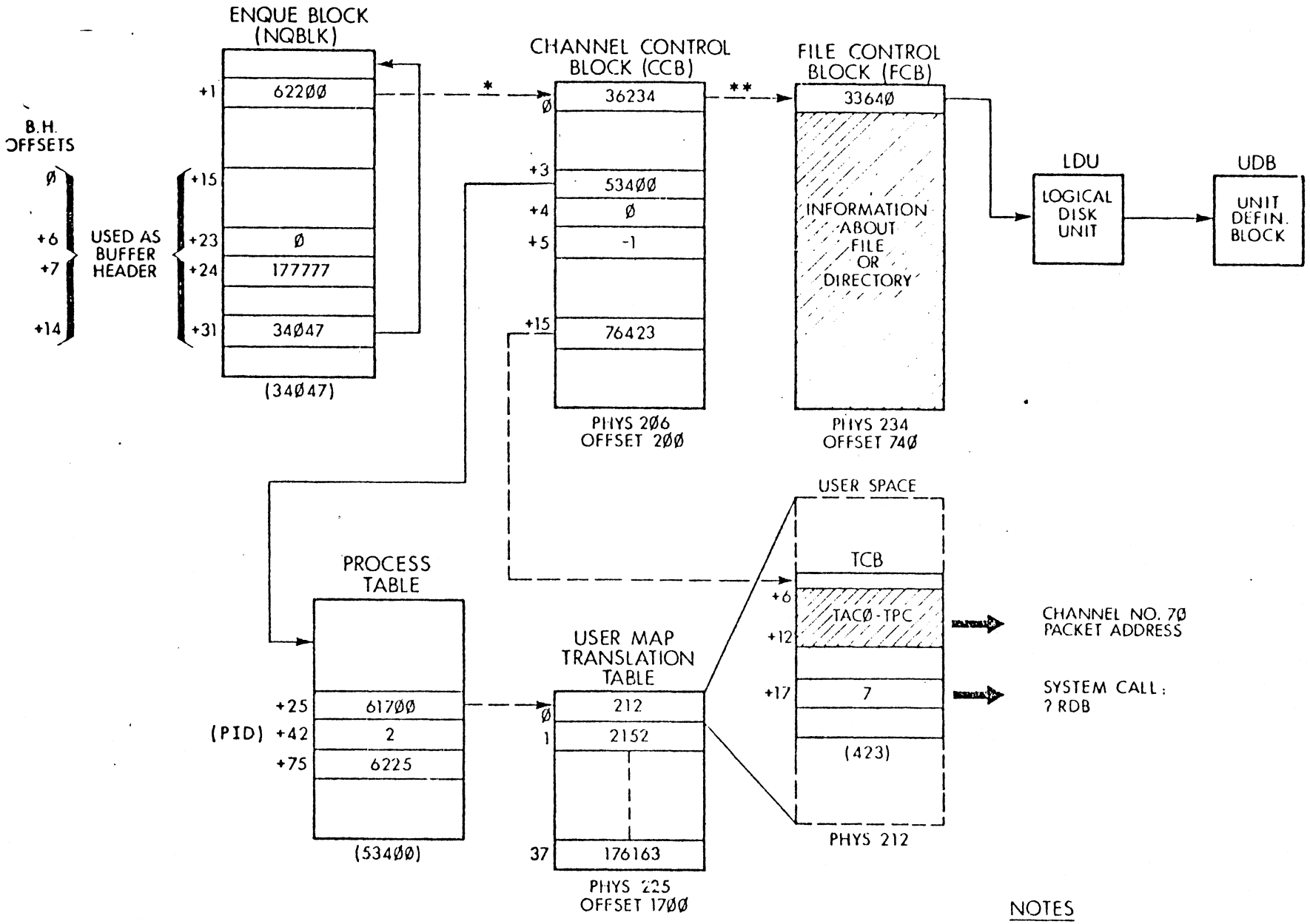
Since this definitely is not an index block nor is it ASCII, I would make the assumption that the last time that the SYSGEN index block was updated, the block was written from the wrong area of core. This could be a memory or map problem and may have happened a while ago if FIXUP had not been run since the occurance. Remember that this is further verified by the fact the FIXUP found an

"Invalid pointer in Index block"

in the SYSGEN Directory. The conclusion could be determined because:

1. A core dump and .ST file were available
2. It had been noted exactly when the error occurred (UP macro) and a copy of the macro submitted.
3. The output of FIXUP as described previously had been submitted.

Figure 6.14.3 shows the 2nd Panic 14 message. This Panic occurred while a CLI dump command was being executed. The file linkages for this failure are shown in Figure 6.14.5. The User, again the Op CLI (PID=2), had issued a "?RDB" system call. The linkages are different than in Figure 6.14.4 because the error occurred on attempting to read a block from disk into a User area, therefore no buffer header is necessary and an area of an NQBLK is used to hold the relevant information. This is similar to the RDOS usage of a cell for a "Fake buffer header". Again the bad values appeared in the NQBLK, the CCB, but not in the FCB. By checking the block that had been used as the first level index, the following was found:



NOTES

* PAGE HAS TO BE MAPPED USING "CUR62" FOR CORRECT PHYSICAL.

** ADDRESS NOT DIRECTLY TRANSLATABLE:
① BITS 6-15 = PHYS PAGE
② BITS Ø-5 = OFFSET WITHIN PAGE. HAVE TO BE RIGHT SHIFTED 5 BIT POSITIONS.

FS-0014

Figure 6.14.5 User/Ghost Request File Linkage

Offset	Contents
Ø	2ØØØ4
1	76
2	Ø
3	-1 ←
4	Ø

I would suspect that since FIXUP gave no errors for file linkages, that the proper index block had been read into the wrong area of core. The problem now would be to find where in core and determine the reason for the failure.

PANIC CODE 15 - POWER FAILURE

**** Does not exist at this time ****

4. Display

This utility is used to print a disk file in both octal and ASCII. Each line contains the octal and ASCII representation of eight 16-bit words from the file. Figure 7.4.1 illustrates an example of line printer output from the DISPLAY utility. If the ASCII character is unprintable, DISPLAY writes it as <nnn>, where "nnn" is the ASCII code for the character. This applies to the following characters: form feed, tab, new line, and carriage return. If a byte contains a quantity which does not have an ASCII equivalent, a blank is printed.

The file is DISPLAY'd in a row format beginning with the first word in the disk file which is referenced as word Ø of row Ø or simply word Ø. The rows increment by a factor of 10_8. There are 10_8 words in each row. For example, the row which begins with the number 256Ø will contain words 256Ø thru 2567 of the disk file. If any row contains words, all of which contain "Ø", the row will not be printed. To indicate that a row(s) is skipped, a series of "**" is printed on the left hand side.

If the file being displayed is a ".PR" file or a "Break" file, the word address given corresponds to the actual logical core address of the word. Additionally, in the case of a "Break" file both the User Context and the Ghost Context will be printed out. The User Context will be contained between addresses Ø and 77777, and the Ghost Context will be contained between addresses 1ØØØØØ and 177777.

If the file being displayed is another type of file, the word address given corresponds to the words offset from the beginning of the file. This can apply to "source" files, data bases, etc.

The format for the DISPLAY utility command is:

)XEQ DISPLAY[switch] pathname ↵

where the only switch applicable is:

/L	Write the output to @LIST instead of @OUTPUT (usually the console)

```
 —Word #(increments of 10₈)
              Word #(units position)
          ┌  Ø      1      2      3      4      5      6      7  ┐
002560| 077566 102470 077617 132470 000471 106470 011433 024417 v 8   8 9 8   )
002570| 132470 000463 106470 011426 110110 142710 166070 000056  8 3 8   H   8 .
002600| 147010 151400 143010 000750 000000 000000 000000 006620              <15
002610| 000060 000071 005426 041517 047123 047514 042440 037440  0 9  CONSOLE ?
002620| 000000 000000 000000 000000 000000 005626 005640 005664
002630| 005462 041101 042040 051503 051111 050124 005000 005556  2BAD SCRIPT<12>
002640| 177777 000000 000040 013622 000000 013324 000040 177777
002650| 000000 000000 000001 000000 040032 000000 005664 000000          @
002660| 177777 000000 000000 000000 005664 000000 177777 020040
002670| 020040 020040 020040 020040 020040 020040 020040 020040
002700| 020040 020040 020040 020040 020040 020040 020040 020040
002710| 020040 020012 000000 030060 030060 030060 054130 000000    <12>  000000XX
002720| 051503 051111 050124 020106 044514 042440 047101 046505 SCRIPT FILE NAME
002730| 020077 020000 020123 041522 044520 052056 050122 005000  ?   SCRIPT.PR<12
*********
003100| 000000 000000 000000 000000 000000 040032 000000 000000            @
003110| 000000 177777 000000 000000 000000 006240 000000 177777            <14>
003120| 040114 050124 000000 006250 047125 046502 042522 020117 @LPT  <14> NUMBER
003130| 043040 046511 047125 052105 051440 037440 000000 006300 F MINUTES ?   <14
003140| 044517 020126 040514 052505 020077 020000 006316 051503 IO VALUE ?  <14>
003150| 051111 050124 020116 052515 041105 051040 036440 020040 RIPT NUMBER =
003160| 020040 020040 005000 006336 000000 000000 000000 000000     <12> <14>
*********
003220| 000000 000000 000000 000000 006452 042506 043111 041511          <15>*EFFI
003230| 042516 041531 005000 006470 051105 051520 047516 051505 ENCY<12> <15>8RES
003240| 020040 020040 047125 046502 042522 020117 043040 052111     NUMBER OF TI
003250| 046505 051412 000000 005556 005556 005574 006536 046520 MES<12>    n n |<1
```

Figure 7.4.1 Example of Line Printer output from the DISPLAY utility.

/L=name Write the output to file name instead of @OUTPUT. This could be a disk file such as "Junk" or the line printer such as "@LPT".

An example of using DISPLAY is:

```
)X  DISPLAY/L=@LPT  CON.PR ↵
```

This will produce an Octal and ASCII listing of CON.PR on the line printer. Note that because this is a ".PR" file, the addresses on the listing will correspond to the actual logical core addresses of the code.

5. Disk File Editor (DEDIT)

This utility is used to examine or modify locations in AOS disk files. You can edit any kind of file on disk using DEDIT. DEDIT can be used to "patch" a ".PR" file to avoid having to reassemble and rebind the source program, or if the source program is not available.

DEDIT uses a subset of the DEBUG commands. The commands that apply to DEDIT and are also common to the DEBUG and SYSDMP utilities are listed in Table 7.5.1. DEDIT has two files on the disk that are needed - "DEDIT.PR" and "DEBUG.OL".

The format for the DEDIT utility command is:

)X DEDIT[switches] pathname ↵

where "pathname" is either a file in he working directory (where you are now), or a pathname indicating the location of the file outside of the working directory. The "switches" that can be appended to the DEDIT command are:

/I=pathname	DEDIT commands will come from the pathname specified. This lets you build a file of DEDIT commands and apply it with a single CLI command. This is the concept used by the "PATCH" macro discussed in Section 8.3, "PATCH.CLI macro". An example of how to use this switch to search multiple core dumps is shown in Figures 4.13 and 4.14 and discussed in Section 4-III, "Core Dump Analysis".
/S=pathname	Allows the symbol table specified by "pathname" to be included. This would have the same effect as if the switch were omitted and a "STAB" command issued in DEDIT.
/L=pathname	Save all the DEDIT dialogue (commands) in a log file identified by "pathname". This would have the same effect as if the switch were omitted and a "LOG" command issued in DEDIT.

Table 7.5.1 DEBUG/DEDIT/SYSDMP Commands

Command	Meaning
LOG ↵ FILENAME? log - filename ↵	Save the console dialog in a disk file(also "/L=" switch on CLI command)
CLOSE ↵	Close the dialogue log file.
STAB ↵ FILENAME? symbol-table-name ↵	Append a Symbol Table (also "/s" switch on CLI command)
SYMON ↵	Causes a search to be made for a new symbol each time "CR/LF" is pressed.
SYMOF ↵ SYMOFF ↵	No search is done each time "CR/LF" is pressed. Previous symbol name plus offset in given.
NOSYM low; high ↵	Used to eliminate intermediate symbols. Will not display additional symbols between the range given; will appear as "low +..."
BYE ↵	Terminate the session.
SHARE ↵ FILENAME? library-name ↵	DEBUG/Edit a shared library.
[ADDRESS]:	Display the contents of a location.
(SHIFT N)	Display the contents of the previous location.
(CR/LF)	Display the contents of the next location.
DISP start-address; end-address ;[increment]; [condition] ↵	Display a range of location contents between the two given limits. Also: ;[increment] is the value added to the address each time a location is examined. ;[condition] is a conditional expression, such as "@.%EQ%4ØØ" which must be true for the location to be displayed.
MES error-code ↵	This will cause the error message to be typed out for the given "error-code". The "error-code" can be in an accumulator and the command given as: MES #Ø ↵

Table 7.5.1 DEBUG/DEDIT/SYSDMP Command (cont.)

Command	Meaning
DSTR byte-address [;length] ↵	Display core locations in ASCII beginning with the byte-pointer contained in "byte-address". The display will cease when: a) the null is encountered b) the "length" in characters is reached. c) 131 characters have been typed out.
?M	Display current "Display " modes (Refer to tables 7.5.2 through 7.5.4)
mode-character (ESC)	Display last item with a diffrent "Display" mode (Refer to tables 7.5.2 through 7.5.4)
[address;] value ↵ or (CR/LF)	Modify the contents the last location accessed or "address" to the value given.
MODE mode-character ↵	Change the "Display" and "Address" modes. (Refer to Tables 7.5.2 through 7.5.4.
expression [mode-character]=	Compute "expression" and display result. (Refer to Tables 7.5.2 through 7.5.4.
LLIST	(Refer to manual).
#H (Dedit only) SET #H;n	Used to indicate which 32Kw segment the logical addresses being given apply to. #H — 32Kw section Ø — 1st 32Kw 1 — 2nd 32Kw 2 — 3rd 32Kw etc. — etc. Note that if DEDIT is being used on a "Break file", the #H symbolism can be interpreted as follows: #H — Meaning Ø — User Context 1 — Ghost Context

Table 7.5.2 Format Submodes

Character	Displays data as
* F	A 16-bit numeric constant
H	An 8-bit numeric constant
A	A pair of ASCII characters
S	A symbol plus offset
N	An instruction
P	Single-precision floating point data
Q	Double-precision floating point data

Table 7.5.3 Radix Submodes

Character	Action
B	Binary format
* O	Octal
D	Decimal with trailing period(.)
X	Hexadecimal

Table 7.5.4 Sign Submodes

Character	Display Data
* VS	Without regard to sign
SI	Interpret sign bit. Display a minus sign prior to number if it is negative.

(Note the "*" indicates the default modes.)

Once DEDIT is running and has successfully loaded the file you specified for editing, it displays the DEDIT text message and then the prompt, "+", which indicates that it is ready to accept commands. Figure 7.5.1 shows a sample DEDIT dialogue session which will be explained in more detail later.

Note that typing errors can be corrected in one of the following ways:

a) if only a couple of characters need to be corrected, type "RUBOUT" the required number of times.

b) if a complete command line needs to be deleted, type "^U".

When you are finished using the DEDIT utility, you can type "BYE" to leave it and return to your previous (father) process. A complete discussion of DEDIT can be found in the Debugger and Disk File Editor User's Manual (Ø93-195).

When using DEDIT to "patch" a file, it is invoked as discussed above and will display the DEDIT text message and then the prompt, "+". If the patches are to be entered using alphabetic symbols, either the "/S=" switch on the DEDIT command must have been used or the DEDIT "STAB" command must now be given. If the symbol-table does not reside within the current working directory (where you were when you started DEDIT), a full pathname must be given. To examine a location, the logical address is typed followed by a colon (:) such as:

+START+4:ØØØ3Ø1+↵

To modify the location, the "new" value is typed immediately after the second plus sign. To change the logical location "START+4" from "3Ø1" to "4Ø2", we would type:

+START+4:ØØØ3Ø1+4Ø2↵

The "under-lined" values are ones that you must type. More examples are given in Section 6, "Panics" in the figures.

Aside from "patching", examining, or modifying a disk file, DEDIT can be very useful in certain types of core dump or "Break file" analysis. In some instances, while troubleshooting a prob-

```
*) X DEDIT DUMP0501
AOS FILE EDITOR REV 1.0
+
+ #H=0                                          Search of
+                                               1st 32Kw
+ DISP 0;77777;;@.%EQ%107561
+
+ SET #H;1
+                                               2nd 32Kw
+ DISP 0;77777;;@.%EQ%107561
+
+ SET #H;2
+                                               3rd 32Kw
+ DISP 0;77777;;@.%EQ%107561
+
+ SET #H;3
+                                               4th 32Kw
+ DISP 0;77777;;@.%EQ%107561
END OF FILE
+
+ BYE

*)
```

Figure 7.5.1 Sample of a DEDIT search of 128Kw for the octal pattern "107561"

lem, "strange" values are found within the core image or an accumulator after the "crash". These values are usually totally meaningless when trying to form an analysis. I have found, in some cases, that by knowing where within the core image other copies of this value reside, we might be able to understand why it is where it shouldn't have been. In order to determine where the other copies of the values are, the core image must be searched (or "DISPlayed") for the value.

Because a logical address space is limited to 32Kw by hardware design, DEDIT will only search 32Kw at a time. If we have a core dump of 128Kw, four searches will be needed to be done; for 256Kw, eight searches. DEDIT determines which 32Kw section it is presently accessing by a variable referred to as "#H". If #H=Ø, the 1st 32Kw is being accessed; #H=1, the second 32Kw, etc. The variable can be examined by typing the variable and then an "=" sign as shown in Figure 7.5.1. It can be modified using the DEDIT "SET" command again as shown in Figure 7.5.1. Once addresses within each 32Kw section have been determined, they can be converted to a "Physical Page/offset" format if they need to be input to SYSDMP using Table 4.9, "DEDIT/SYSDMP Address Conversion Table". The reason for using DEDIT to do the search rather than SYSDMP is two-fold:

a) DEDIT is extremely faster

b) SYSDMP uses logical addresses, therefore, each physical page must be put in a log/phy translation table or done manually. This is a real "pain"!

Possibly in the future, DEDIT can be modified to search a complete file rather than only 32Kw at a time.

Before we discuss the display commands shown in Figure 7.5.1, let's discuss a little about how values can be compared. DEDIT will accept the following logical operators in addition to nested parenthesis:

a) %EQ% equal to
b) %LT% less than
c) %LE% less than or equal to
d) %GE% greater than or equal to
e) %GX% greater than
f) %NE% not equal

It will also accept the Boolean Arithmetic Operators:

g) %AND% Logical and
h) %OR% Inclusive or
i) %XOR% Exclusive or

and also:

j) period (.) which indicates to use the current address in the calculation.

k) indirect (@) which indicates to use the contents of the given address rather than the address itself.

Now a couple of examples to make this a little clearer:

1) @. means to use the contents of the current address.

2) @.%EQ%-1 means to compare the contents of the current location to a "-1" and determine if they are equal.

3) (@.%AND%377)%EQ%15 means to take the contents of the current location, AND it with 377 and compare the result to "15" and determine if they are equal. In this case, I only wish to examine the low byte and check if for a CR.

I would always use nested parenthesis to avoid having to worry about how the "operators" are evaluated. Remember that parenthesis are evaluated (ie: commands executed) from the innermost set outward. A good "rule of thumb" is to work from left to right with the checks causing all the "left-hand" parenthesis

to be on the far left of the expression. For example:

```
(((@.%AND%377)%XOR%76)%EQ%3)
  will be ex-
  ecuted first.
```

For addition and more indepth information on the use of "operators", refer to Chapter 3 of the Debugger and Disk File Editor User's Manual.

The command used by DEDIT to search is the display command, DISP. The format of this command is:

+DISP start-address; end-address; [increment]; [expression]

where: "start-address" is the starting logical address within the 32Kw, in our case, it will be "Ø"

"end-address" is the ending logical address within the 32Kw, in our case, it will be "77777"

"increment" is an optional expression which defaults to "1" and indicates the step between addresses checked. If it were 4, every fourth word would be checked. Similar to a FORTRAN "DO" loop value. In our case, no value will be used.

"expression" is a combination of logical, Boolean operators, etc, as discussed before. If the expression result is "True", the address and contents are printed. If the expression result is "False", nothing is printed and the "next" location is checked. This will vary depending on what we are checking for.

Figure 7.5.1 is a sample of a DEDIT search of 128Kw for an octal pattern of "1Ø7561". Note that the "End of File" message occurred within the last 32Kw so the system was somewhere between 96Kw and 128Kw. No values were found which matched the requested value.

Figure 7.5.2 is a sample of a DEDIT search of 32Kw for the octal pattern of "x774ØØ" where "x" is a "don't care". Since we do not care whether bit Ø is a Ø or a 1, we mask it off using the AND instruction. This will cause bit Ø always to be a "Ø" so we compare with the value "Ø774ØØ.

Table 7.5.2 lists the various Format Submodes; Table 7.5.3 lists the various Radix Submodes; and Table 7.5.4 lists the various Sign Submodes. The Modes can be changed using the "MODE" command or the "mode-character (ESC)" as listed in Table 7.5.1. For example to examine a location, I type the logical address and a colon (:) such as:

+3:Ø425Ø1+A$ EA +

where the contents of location 3 is "425Ø1". To see if the location previously addressed contains any ASCII characters, I can type "A$", where "$" is an Escape, and find that it contains the ASCII characters "E" and "A", high and low bytes respectively.

The previous discussions were not intended as a tutorial but as a refresher. The Debugger and Disk File Editor User's Manual should be read in detail as this Utility along with SYSDMP will be one of the most used ones.

```
*) X DEDIT DUMP0501↵
AOS FILE EDITOR REV 1.0
+ ↵
+ DISP 0;77777;;(@.%AND%77777)%EQ%77400↵

75 :077400
112 :177400
1603 :077400
6402 :177400
10500 :077400
14055 :077400
50065 :177400
50502 :177400
+↵
+ BYE↵

*)
```

Figure 7.5.2 Sample of a DEDIT search of 32 Kw for the octal pattern "x774ØØ" where x is a "don't care"

6. System Dump Analyzer (SYSDMP)

This utility is used to examine and analyze locations in an AOS core dump. The automated analyzer, ADA, should be run praor to using SYSDMP. ADA procedures are discussed in Section 4-II, "AOS Core Dump Analyzer". If ADA is not available or "bombs" with an "End of File", SYSDMP can be used to access the file. In most cases, SYSDMP will have to be used anyway as ADA will not provide all the information needed to completely analyze a core dump.

WARNING

Because of the many similarities between DEDIT and SYSDMP, only the differences will be discussed in this section. It will be assumed that Section 7.5, "DEDIT", has been read and understood prior to reading this section. Basically, SYSDMP commands are a subset of the DEBUG commands with a couple special ones added. Other than the special commands, DEDIT and SYSDMP use identical commands.

The commands which apply to SYSDMP, in addition to DEDIT and DEBUG, are listed in Table 7.5.1. The special commands which apply to SYSDMP only are listed in Table 7.6.1.

The format for the SYSDMP utility command is:

)X SYSDMP/S=sym-table {switches} pathname ↵

where "pathname' is either a file in the working directory (where you are now), or a pathname indicating the location of the file outside of the working directory. The "/S=" switch must be given to SYSDMP because the "symbol-table" for the operating system which was running when the crash occurred is needed, not optional. The "sym-table" filename has the same restrictions as mentioned above for "pathname". All the other "switches" are the same as for DEDIT. Figure 7.6.2 illustrates what will happen using SYSDMP if no symbol table is given. Note that the "STAB" command can also be used to define a symbol table, but the "/S" is easier.

The commands used to access "addresses" within the file are

Table 7.6.1 SYSDMP only Commands

Command	Meaning
MAP logical-address ↵ MAP physical-memory-page ↵ MAP SMAP ↵	This command will set the logical to physical address translation for future addresses to: a) the "logical address" is the address of the 40_8 word logical to physical address table that has to be used. b) the "physical memory page" is the only page for which addresses are to be translated. The page value is between 0 and a maximum of 377. Only bits 6-15 of a future logical address are used. c) same as a) above. SMAP is a symbol that corresponds to the AOS logical to physical translation table.
?MAP ↵	Display the contents of the current "MAP" setting.

the same, but the "addresses" have a different meaning to the utility. DEDIT applies its addresses relative to the first word of the file. If I gave an address of 4Ø371 to DEDIT, it would show me the 4Ø372nd word in the file. No memory or core address significance is applied by DEDIT. Because of this, DEDIT can be used to access any type of file, even an ASCII text file. It just so happens that when we are talking about ".PR" and "Break" files, both of which are core images, the word offsets within the file are the same as the logical core addresses.

SYSDMP, on the other hand, applies it's addresses strictly as logical core addresses. In order to use a logical address, the utility must have some idea of which physical page the logical address applies to. Every process that runs under AOS and the operating system itself has a logical to physical address translation table that is $4Ø_8$ words in length. When SYSDMP is initially invoked, it uses the AOS Operating System logical to physical translation table, called SMAP. Because SYSDMP needs to know where SMAP resides in core, the symbol-table for the Operating System which was running when the system "crashed" is required.

It should be noted that not all the slots in SMAP are mapped to physical pages. The number of unused ones will depend on the Operating System activity when the "crash" occurred. Because of this, if a "DISP" is tried on all the Operating System space, when the "DISP" encounters the first unused slot, an "ILLEGAL ADDRESS" message will be displayed and the search will stop. If there were valid pages on the other side of the "un-used" slot, they would never be searched using the standard method. Figure 7.6.1 illustrates doing a search using both DEDIT and SYSDMP. Because SYSDMP has to do constant logical to physical translations, it is much (!!!) slower on a "DISP" than DEDIT. Also not all the Operating System logical to physical page translations are kept in SMAP, some other variables are used. Refer to Figure 6.7.1, "AOS Logical Address Space Layout".

Because the initial mapping under SYSDMP is set to SMAP, there are commands to change the current mapping ("MAP") and

```
*) X DEDIT DUMPØ5Ø3↵
AOS FILE EDITOR REV 1.Ø
+ STAB↵
FILENAME? :SYSGEN:AOS21.ST↵
+
+ DISP Ø;77776;;@.%EQ%1Ø7561↵

CLIBT+1355 :1Ø7561
OH+2776 :1Ø7561
+
+ BYE↵

*) X SYSDMP DUMPØ5Ø3↵
AOS SYSTEM DUMP ANALYZER REV 1.Ø
+ STAB↵
FILENAME? :SYSGEN:AOS21.ST↵
+
+ DISP Ø;77776;;@.%EQ%1Ø7561↵
 ILLEGAL ADDRESS
+
+ BYE↵

*)
```

DEDIT

SYSDMP

Figure 7.6.1 Sample of a DEDIT vs a SYSDMP search for the octal pattern "1Ø7561"

```
*) X SYSDMP DUMPØ5Ø3↵
AOS SYSTEM DUMP ANALYZER REV 1.Ø
+
+ 6ØØØØ: UNKNOWN OR ILLEGAL SMAP VALUE
+ 4ØØØØ: UNKNOWN OR ILLEGAL SMAP VALUE
+ Ø: UNKNOWN OR ILLEGAL SMAP VALUE
+
+ BYE↵

*)
```

Figure 7.6.2 Sample of an attempt to use SYSDMP without defining a Symbol Table

```
*) X SYSDMP DUMP0503↵
AOS SYSTEM DUMP ANALYZER REV 1.0
+
+ STAB↵
FILENAME? :SYSGEN:AOS21.ST↵
+
+ MAP SMAP↵
+ ?MAP↵
000473
+ SMAP=473
+
+ 40000:052000 +↵
+ 40002:175152 +↵
+
+ 60000: ILLEGAL ADDRESS
+
+ MAP 20↵
+ ?MAP↵
000020
+
+ 0:052000 +↵
+ 2:175152 +↵
+
+ 3000:044120 +↵
+ 1000:044120 +↵
+ 5000:044120 +↵
+
+
+ MAP SMAP↵
+
+ 41000:044120 +↵
+ 43000:042000 +↵
+
+ BYE↵

*)
```

Examining locations Using SMAP

Using Physical Page 20

Using SMAP

Figure 7.6.3
Sample of SYSDMP location analysis by using the AOS System Map (SMAP) and by using a Physical 1K page

to examine where the current mapping is set ("?MAP"). These two commands are discussed in Table 7.6.1. Note that if a "physical-memory-page " is used as an arguement, SYSDMP only looks at bits 6-15 of the logical address given. These bits will generate a logical address range from Ø thru 1777 which is a physical memory page. Figure 7.6.3 shows a sample of SYSDMP location analysis using the AOS System Map (SMAP) and using a physical $1K_{1Ø}$ page. While accessing Physical Page 2Ø in Figure 7.6.3, the addresses 3ØØØ, 1ØØØ, and 5ØØØ all yield the same value since in all the cases bits 6-15 are "1ØØØ".

Remember that SYSDMP only uses the one symbol table. When mapped to SMAP, a logical address of 14ØØ would be equated to a symbol of "LTBL". If I were mapped to Physical Page 24Ø and gave an address of 14ØØ, I still would get the symbol of "LTBL". SYSDMP uses a logical address to compare for the proper symbol. In both the above cases, the logical address is 14ØØ but in one case the symbol equated by SYSDMP is meaningless. Whenever using SYSDMP, always issue the "SYMON" command to cause symbols to change when doing "CR/LF"s.

A problem that can be encountered when using SYSDMP is finally determining that the problem is within some User Process and then wanting to examine logical addresses within that process to determine the process's problem. Without modifying the core dump, each logical address that has to be accessed must be translated to it's slot#, the physical page determined, that physical page mapped, then the logical address used to check the location. This can be much work if many locations must be examined in the Process's Logical Address Space. A procedure to modify the 6ØØØØ slot is SMAP so that the logical to physical translation table for that process can be mapped is discussed in Section 4-III-(1), "User Process Analysis" and illustrated in Figure 4.15, "'Mapping' a User Logical Address Space".

12. RDOSBIND

The RDOS Binder, RDOSBIND, is a program that runs on an AOS system but produces an RDOS Save File (.SV). It requires as input - object files and libraries in the AOS format. Because of this, the standard AOS Macro Assembler and Library File Editor can be used to prepare input.

The format for the RDOSBIND utility command is:

)X RDOSBIND {global-switches} arguments {argument-switches} ↵

where the global switches are:

/B	Produce a listing of the symbol list with symbols ordered both alphabetically and numerically.
/D	Causes the RDOS user debugger to be loaded and enters Global Symbols in the Save File.
/E	Output the load map to @OUTPUT, even if a listing file (/L) has been specified.
/G=n	Designate that "n" number of channels are required.
/H	List all numbers in Hexadecimal to @OUTPUT and @LIST.
/K=n	Allocate "n" TCB's for multi-task use regardless of how many are specified in a .TSK statement.
/L	Produce a listing file and direct it to @LIST
/L=name	Produce the listing file to a file called name.
/N	Do not search the system library, ASYSOB.LB. If this switch is omitted, ASYSOB.LB must be "converted" using the CONVERT Utility (refer to Section 7.9, "Convert")
/O	Suppress error flags whenever bind-overwrites occur. A bind-overwrite occurs when one module places code in one or more locations and a succeeding module overwrites these locations.

/P=name — Create save file called "name.SV". If you omit this switch, the save file will be given the name of the first module in the RDOSBIND command line.

/Z — Produce an RTOS/SOS compatible save file

The arguement switches are:

name/C — Allows the specification of a "command file" which is required when defining overlays using square brackets.

/O — Suppresses load overwrite messages for this module only. (Refer to Global "/O")

/V — Designates a virtual node (used as in RLDR with the same restrictions)

n/Z — Set the current ZREL base to "n". This counter cannot go below its current value. Initial value is $5\emptyset_8$.

Note that ".COMM" task will not work as it did in RDOS. Use the .TSK pseudo-op or the /K function switch to specify the number of TCB's required. Use the /G switch to specify the number of channels.

Also as with the AOS Binder, you must use a command file in order to specify any nodes and overlays.

The RDOSBIND consists of program file, RDOSBIND.PR, and overlay file, RDOSBIND.OL. It requires a program file, MAPER.PR, to produce symbol lists. This MAPER.PR is identical to the one supplied with the AOS Binder.

An example of an RDOSBIND command line is:

```
)X RDOSBIND/N/B/L=@LPT/P=WORKS  TEST  RDOS.LB )
)
```

The "load map" from the RDOSBIND program is shown in Figure 7.12.1. Note that the resulting save file is called "WORKS.SV" because of the "/P" switch. The RDOS ".RB's" which were extracted from the RDOS SYS.LB, were "CONVERT'd", then put into a li-

```
WORKS.SV CREATED BY AOS - RDOS BINDER REV 01.01 ON 1/19/78 AT 10:34:18
TEST
TMIN
NSAC3


000002 WORDS OF ABSOLUTE DATA
ZMAX: 000051
NMAX: 000554


PARTITION    TYPE    START     END   # OF NODES
000004     NSHR CD   000446  000553   000000

.SAC0       020016
.SAC1       024016
.SAC2       030016
.SAC3       034016
NMAX        000554
START       000446
TMIN        000461
USTAD       000400
ZMAX        000051


ZMAX        000051
USTAD       000400
START       000446
TMIN        000461
NMAX        000554
.SAC0       020016
.SAC1       024016
.SAC2       030016
.SAC3       034016
```

Figure 7.12.1 Sample RDOSBIND "load map"

brary using the AOS LFE utility called "RDOS.LB". Since the library was called RDOS.LB and not ASYSOB.LB, the "/N" switch was needed.

SECTION 8
SYSTEM MACROS

A "macro" basically is a technique whereby you can use one CLI command to execute a file containing one or more CLI commands. Anybody can write CLI Macros. They are described in the CLI Manual (093-122) in Chapter 5.

"System Macros" are macros that are written by Data General and supplied on the standard Operating System magnetic tape or diskette. These "System Macros" are:

File Name		Figure	Page Described
a) Operating System			
1)	UP.CLI	8.1	8-5
2)	DOWN.CLI	8.2	8-10
3)	PATCH.CLI	8.3	8-10
4)	CONTEST.CLI	8.4	8-10
5)	SYSTAPE.CLI	8.5	8-24
b) Fortran IV			
6)	FORT4.CLI	8.8	8-35
c) Fortran 5			
7)	F5.CLI	8.6	8-24
8)	F5LD.CLI	8.7	8-24
d) Basic			
9)	BIND.CLI	8.9	8-35
10)	BINDSP.CLI	8.10	8-35

Note that all System Macros end in the suffix ".CLI". This ending is an indicator to the CLI that the command is a macro. Along with the command, other arguments can be passed from your command line to the macro and replace "dummy" arguments. If my command line was:

```
*)SYSTAPE @MTA0 AOS.SY
```

"SYSTAPE" is the name of the macrofile; "@MTA0" is dummy argument #1 which is indicated within the macro file by %1%; "AOS.SY" is

dummy argument #2 which is indicated within the macro file by %2%.

Referring to Figure 8.5 which illustrates the SYSTAPE macro, we can note various lines which contain the dummy arguments %1% and %2%. If I had issued the above command line, the various lines of the macro would change when they were executed:

REW %1% ⟹ REW @MTAØ

COPY %1%:Ø TBOOT ⟹ COPY @MTAØ:Ø TBOOT

and so forth. The actual commands in the disk file would not change.

Macros can get very complex and are able to check arguments to see if they are "Equal"; can "branch" on specific conditions; etc. For example, refer to Figure 8.1 which illustrates the UP macro. The first line of the macro is:

[!EQ,1,2]

This checks if the number "1" is equal to the number "2". If they are equal, the macro continues executing sequential commands. If they are not equal, the macro will "branch" (follow the arrow in Figure 8.1) to a group of commands which use the CLI "Write" command to put the following message on the CRT screen:

"Non-executable sample ...." etc.

and then it finishes. The second line in the macro uses another "!EQ" to check whether any arguments or switches were appended to the command line and also whether the command line was issued by the "Root CLI" (OP CLI=PID 2). If either is not true, the macro will "branch" (again follow the arrow) and type out the appropriate message.

The above was just to give you an idea of the macro concepts of "checking and branching" and "argument passing using dummy arguments". Much more detail is contained in Chapter 5 of the CLI manual dealing with macros.

Note that the command "!EQ" is referred to as a "Pseudo-macro". These are listed along with an explanation of their functions in Chapter 6 of the CLI manual beginning around Page 6-61 and in Section 1Ø.5, "Useful CLI Commands", of this document.

1. UP.CLI macro

The UP macro is illustrated in figure 8.1. It is used to bring up the AOS Operating System and includes initialization commands for the CLI and the EXEC. When it is first loaded into the directory :UTIL during your system initialization, it is in a non-executable format (because of the first !EQ) and should be edited using, SPEED or LINEDIT to tailor it to the needs of the customer's installation.

Some considerations for tailoring the UP macro:

a) "CHAR/HARDCOPY" command assumes the master console is a hardcopy device such as a Dasher Printer. If it is another type device refer to the options under the "Characteristics" CLI command. Other options are 6Ø12, 4Ø1ØI, 6Ø5x, or CRT4.

b) "...@LPA" is set up for the default line printer - primary device code, non-DCH. The other options are:

@LPA1 - secondary device code, non-DCH
@LPB - primary device code, DCH
@LPB1 - secondary device code, DCH

Note that this must conform with the questions that were or will be answered in the AOSGEN dialogue.

c) "SYSLOG/START" is a command that does not presently exist in the UP macro. It is needed if hardware errors are to be logged to a disk file. This command does not have to be in the UP macro, but has to be executed prior to the "Execute CLI" command in the UP macro since it must be issued from PID=2. If the User has a large disk, CDC or 6Ø6Ø/61, the command should be put in the UP macro and the log file periodically printed out if it grows too large. This precludes the operator forgetting to issue the command and hardware errors are then not logged.

d) "[CONSOLES]" is an additional macro that is used by the UP macro. It contains the numeric portion of the con-

soles that you want the EXEC to enable for log-on. For example, a "6" would cause the EXEC to try to enable "@CON6". Note that the CONSOLES file must not contain a "new-line" terminator character.

e) "[!EQ,1,2]" Since the initial macro is provided as a sample, the two numeric arguments are not equal, therefore the macro can never be sucessfully executed. They must be edited to be equal after any other needed changes are made.

f) Other useful information which may impact the editing of the UP macro is contained in a section of the Release Notice called "How to Prime the EXEC capability" under "Additional Programs, Features, and Functions". The Release Notice is contained in disk file ":UTIL:RELEASE".

Note that after the UP macro has been edited, it should be moved along with the CONSOLES file to the Root directory, such as:

*)MOVE : UP.CLI, CONSOLES ↵

This must be done with Super User "On" and prevents a future re-loading of the :UTIL directory from over-writing your edited versions.

Note in Figure 8.1 that lines 9 thru 12 all begin with:

"CONTROL @EXEC ...."

This is the mechanism through which a User can communicate with the EXEC. "Control" is a CLI command and has the following format:

"CONTROL <IPC-name> <arguments>"

The "@EXEC" is an Inter Process Communications (IPC) port. The "@" indicates that this port is in the :PER directory. Since the EXEC is a process (EXEC.PR), this is how your console process, CLI, will communicate with it. The "IPC" is explained further in Chapter 4 of the AOS Programmer's Manual (Ø93-12Ø). The next argument following the "IPC-name" is the command for the EXEC such as "ENABLE", "START", or "CONTINUE". Further arguments will

vary depending on the specific command. Note also that EXEC commands are not accepted from a "logged-on" console. If they are issued, an error message:

```
"FROM PID 3: (EXEC) VALID ONLY FROM OPERATOR,....
 FROM PID 3: (EXEC) 15:ØØ:29"
```

is typed on the sending console.

Now we should discuss exactly what the UP macro does. Each command line in Figure 8.1 is numbered and will be discussed below.

Line 1 - this is the check that must be edited in the non-exectable sample macro. If the numeric values are not equal, the message pointed to by the arrow will be typed out.

Line 2 - this line checks that no arguments or switches were appended to the macro, and that the macro is executed from PID=2 (Operator CLI). If either fails, the message pointed to by the arrow will be typed out.

Line 3 - adds the ":UTIL" directory to the search list.

Line 4 - the "environment" level is now pushed. This allows it to be changed temporarily and later POP'd to its previous state.

Line 5 - turn Super User "ON". This is needed so that "ACL's" can be changed and the EXEC started in the :PER directory. (Note that this is an "environment" change).

Line 6 - change the access for the Magnetic Tape Units so that all Users ("+") can access them.

Line 7 - create the EXEC process with all the privileges of the OP CLI ("default") in with a working directory of :PER ("@") and call it EXEC.

Line 8 - pause for 2 seconds while the EXEC is created and initialized.

Line 9 - tell the EXEC to try to enable any console whose number is in the CONSOLES file.

```
    (!EQ,1,2)
(!EQ,(((!NEQ,$02,((!PID)))ABORT(!END)%/%%-%),())
 SEARCH :PER :UTIL :
 PUSH
 SUPER ON
 ACL $MTA(0,1) + MARE
 PROCESS/DEFAULT/DIRECTORY=@/NAME=EXEC EXEC
 PAUSE 2.000
 CONTROL $EXEC ENABLE $CON((CONSOLES))
 CONTROL $EXEC ((CONTINUE 1) VERBOSE)
 CONTROL $EXEC START (BATCH_<OUTPUT LIST> LPT) $LPA
 CONTROL $EXEC CONTINUE $LPA
 CHAR/HARDCOPY
 POP
 PROMPT TIME CHECKTERMS
 EXECUTE CLI
(!ELSE)
 WRITE *ABORT*
 WRITE %0\4 IS VALID WHEN INVOKED BY THE ROOT CLI
 WRITE REQUIRED ARGUMENTS: NONE
 WRITE OPTIONAL SWITCHES: NONE
-(!END)
(!ELSE)
WRITE NON-EXECUTABLE SAMPLE...YOU CAN EXECUTE UP.CLI BY MAKING
WRITE THE ARGUMENTS IN THE COMMAND ON THE FIRST LINE EQUAL AND
WRITE MOVING THIS MACRO TO THE ROOT DIRECTORY.

(!END)
```

Figure 8.1 UP macro

Line 10 - tell the EXEC to "continue" Batch Stream 1 and that all the details about any Batch Stream job should be told to the operator console ("Verbose").

Line 11 - tell the EXEC to link the Batch-Output, the Batch-List, and the LPT Queues to the physical output device, @LPA.

Line 12 - tell the EXEC that the physical device, @LPA, can "continue".

Line 13 - set the characteristics of the master console to "Hard copy".

Line 14 - "POP" the environment to where it was prior to the "PUSH".

Line 15 - indicates to the CLI that before issuing a prompt, ")" or "*)", type the time and type any process termination messages.

Line 16 - this will effectively "push" a level of CLI. This is done so that if the CLI is accidentally "aborted", the system will not crash but control will be returned to the previous level CLI. Also prevents the DOWN macro from executing since the PID is now no longer "2".

The rest of the macro contains the error messages and the [!ELSE] /[!END] for their respective [!EQ...] .

Figure 8.1a illustrates the responses that the EXEC sends to the master console when the UP macro is executed. The line numbers in Figure 8.1a correspond with the specific command line numbers in Figure 8.1.

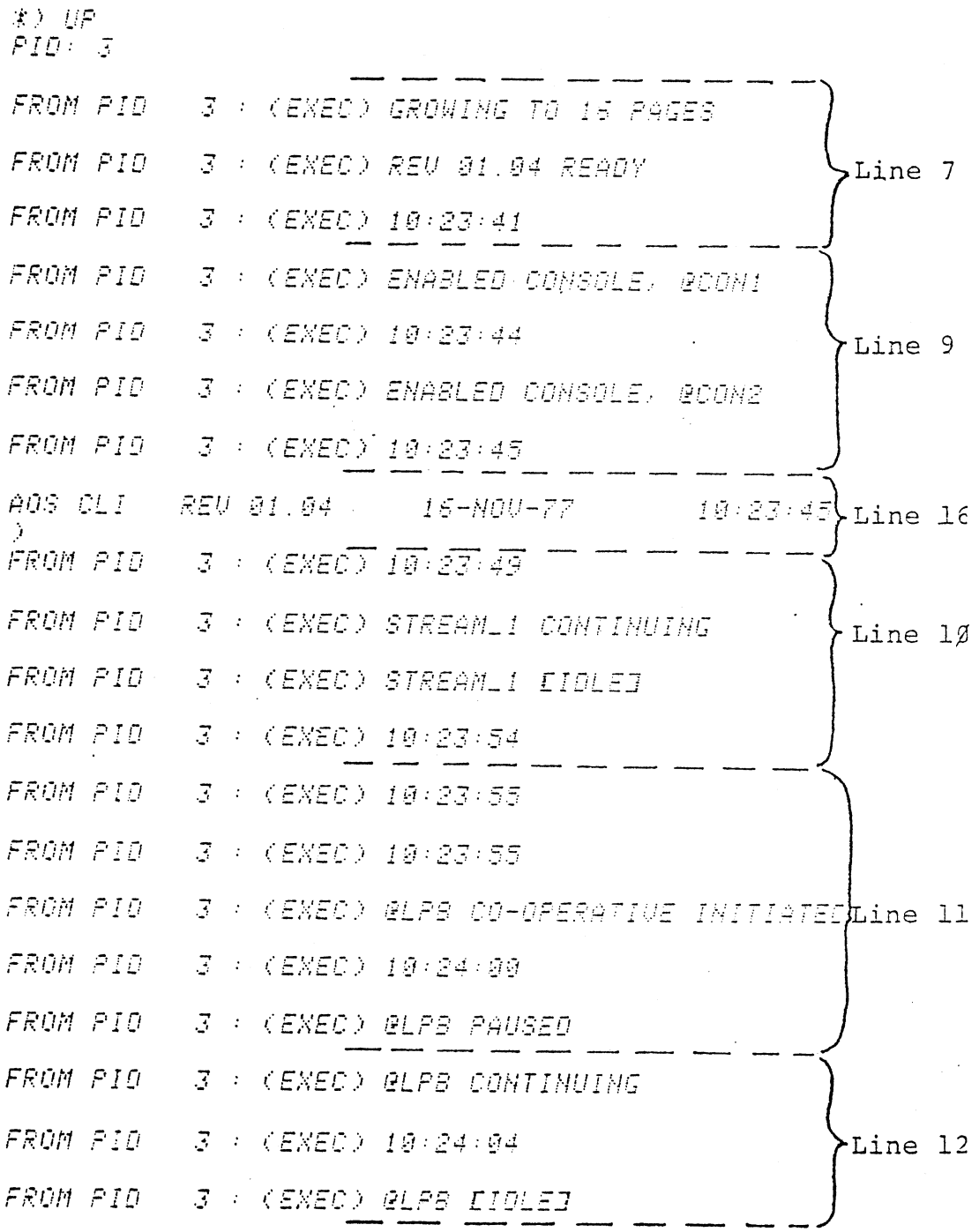


Figure 8.1a Sample EXEC Responses

2. DOWN.CLI macro

The DOWN macro is illustrated in figure 8.2. It is used to shutdown the EXEC and to stop the system logging function. When it is first loaded into the directory :UTIL during your system initialization, it is in a non-executable format (because of the first !EQ) and should be edited using SPEED or LINEDIT to tailor it to the needs of the customer's installation.

There are only a couple of considerations for tailoring the DOWN macro:

a) The command "SYSLOG/STOP" is not really necessary since the logging function stops automatically at system shutdown.

b) If a report were wanted of the daily system activity from the SYSLOG using the REPORT utility, it could be done in this macro.

c) If directories were to be backed up on Mag Tape, this could be done via this macro.

d) "[!EQ,1,2]" Since the initial macro is provided as a sample, the two numeric arguments are not equal, therefore the macro can never be sucessfully executed. They must be edited, as illustrated in Figure 8.2, to be equal after any other needed changes are made.

Much of the content of this macro which would be "edited" will be very installation dependent.

Note that after the DOWN macro has been edited, it should be moved to the Root directory such as,

```
*)MOVE : DOWN.CLI ↵
```

This must be done with Super.User "On" and prevents a future re-loading of the :UTIL directory from over-writing your edited versions.

Now we should discuss exactly what the Down macro does. Each command line in Figure 8.2 is numbered and will be discussed below:

Line 1 - this is the check that must be edited in the non-executable sample macro. If the numeric values are <u>not</u> equal, the message pointed to by the arrow will be typed out.

Line 2 - this line checks that <u>no</u> arguments or switches were appended to the macro, and that the macro is executed from PID=2 (Operator CLI). If either fails, the message pointed to by the arrow will be typed out. Note that since the UP macro "pushes" a level of CLI, a BYE must be typed on the master console prior to the DOWN.

Line 3 - causes the EXEC to be terminated. Along with this any co-operative processes created by the EXEC such as XLPT or STACKER are also terminated.

Line 4 - turns off the system disk logging feature. This applies also to <u>hardware</u> error logging.

Line 5 - pause for 2 seconds to allow for the terminations to complete.

Line 6 - CHECKTERMS is a CLI command which will check for the termination of any son process and output the appropriate message.

The rest of the macro contains the error messages and the [!ELSE]/[!END] for their respective [!EQ...].

```
 1  [!EQ,2,2]
 2  [!EQ,[[!NEQ,002,[!PID]]!ABORT[!ENDI%/%%-%).C]]
 3   TERMINATE/2=ERROR OP:EXEC
 4   SYSLOG/STOP
 5   PAUSE 2.000
 6   CHECKTERMS
 7  [!ELSE]
 8   WRITE *ABORT*
 9   WRITE %0\% IS VALID ONLY WHEN INVOKED BY THE ROOT CLI
10   WRITE REQUIRED ARGUMENTS: NONE
11   WRITE OPTIONAL SWITCHES: NONE
12  [!END]
13 [!ELSE]
14 WRITE NON-EXECUTABLE SAMPLE...YOU CAN EXECUTE DOWN.CLI BY MAKING
15 WRITE THE ARGUMENTS IN THE COMMAND ON THE FIRST LINE EQUAL AND
16 WRITE MOVING THIS MACRO TO THE ROOT DIRECTORY.

17 [!END]
```

Figure 8.2 DOWN macro

```
X DEDIT/I=%2%/S=%3% %1%
```

Figure 8.3 PATCH macro

```
PROC/DEF/REB/IOC/BLOCK CON
```

Figure 8.4 CONTEST macro

3. PATCH.CLI macro

The Patch macro is illustrated in Figure 8.3. It was originally intended to be used to "patch" the AOS operating system or a System Utility file. This method is presently not used since a monthly Update Magnetic Tape containing module replacements is provided. Since it might be used in the future, it will be discussed briefly. The macro requires the following three arguments:

a) name of the Program File to be patched

b) name of the file containing the patch commands. These are actually commands to the DEDIT utility.

c) name of the Symbol Table file for the program to be patched.

An example of the use of this macro is:

```
)PATCH  PROG.PR  PATCH.Ø1  PROG.ST ↵
```

The patches will be made to the program file on disk. Note that if this method is ever used to patch an AOS Operating System, you must either "re-install" it on the disk; or constantly override the default specifications and specify the system path-name each time the disk is booted. The patched AOS Operating System should be re-installed on the disk using the SYSTAPE macro to create a new backup tape and then invoking INSTL to prevent any future confusion with regard to which Operating System is being run.

4. CONTEST.CLI macro

A. General Overview

The CONTEST macro is illustrated in Figure 8.4. It is used by Field Engineering personnel to test the hardware via a Confidence Package. This is done by initiating additional processes to check out the system. The actual procedures for system checkout are defined in Section 11, "Field Engineering AOS Checkout Procedure".

The CONTEST macro initiates a resident process called "CON.PR" which is effectively a monitor. CON.PR asks the following questions:

"Script File Name?"
"Number of Minutes?"

There are two versions of "Script" files, one for a 128KW system and another for a 256KW system. The only difference is the number of processes that are initially started. There are two types of sub-processes. One does I/O to the disk unit, the other exercises memory. The number of each is broken down as follows:

Script	I/O Processes	Memory Processes
SCRIPT.128	2	8
SCRIPT.256	4	16

Figure 8.4a illustrates the SCRIPT.128 command file and Figure 8.4b illustrates the SCRIPT.256 command file.

The "number of minutes" can either be answered with a value, such as 144Ø (24 hours), or with a value of "Ø". If a value of "Ø" is specified, CONTEST will run until it is manually stopped via a "^C^A". If a value other than Ø is used, the "^C^A" will have no effect since the ?INTWT System Call is never issued ("^C^B" will always terminate CONTEST). Note also that if a value of Ø is used, no "End of Minute" messages will be typed out. This is very useful when combined with the modification described in the following paragraph.

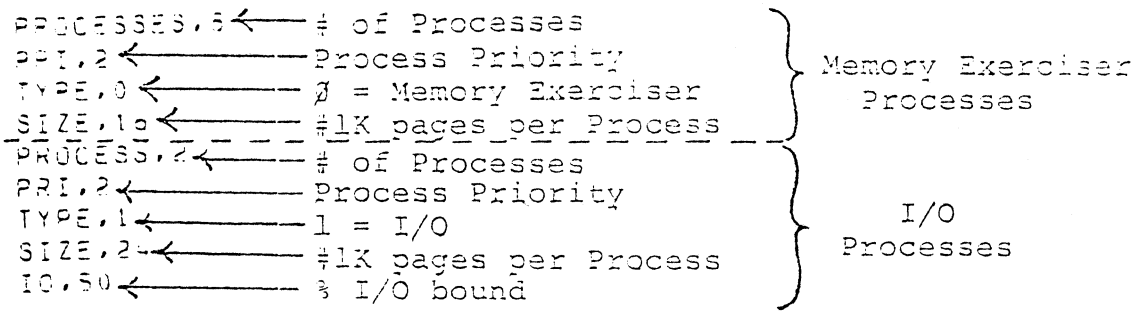


Figure 8.4a SCRIPT.128 command file

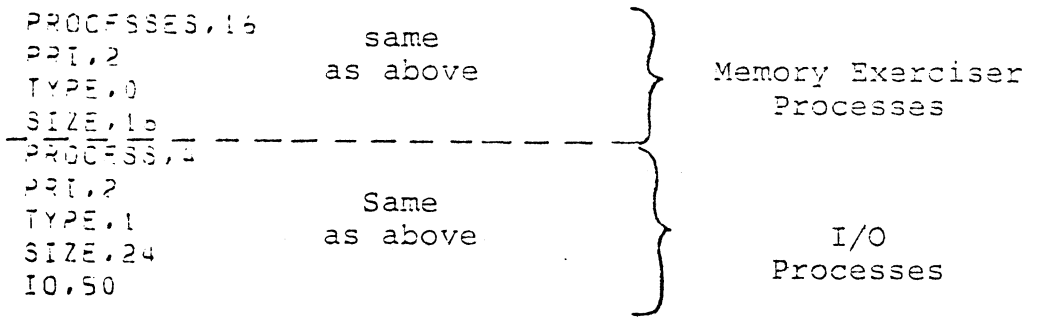


Figure 8.4b SCRIPT.256 command file

CONTEST will normally "halt" when it finds a memory error after typing out the message described in section C-1 which follows. In many cases, it might be useful to accumulate a few failures during an overnight run so they can be compared. The module, MPRCC.PR, can be patched to continue executing if it finds a memory failure. The patch is (for a Rev 1.03 MPROC version):

location	Old	new
16ØØ	63Ø77	4Ø1
16Ø1	4ØØ	4Ø1

Now MPROC will keep running. A problem is that since CONTEST will not log errors to a lineprinter or disk file, the "End of Minute" message would push the errors off a CRT screen by the time morning came around. Well, run the "patched" CONTEST with "number of minutes" equal to Ø.

B. Data Pattern

The pattern that is written in core could be called a "sequential-random" in that each new word depends on the last word and does not frequently repeat. The hardware "MUL" instruction is used to generate the pattern using AC1 and discarding ACØ of the result. This pattern sequence is used to fill the entire buffer area then checked using the same method. Note that all locations with the same right-most address digit will have a content with a consistant right-most digit. For example:

Location	Contents
1ØØØ2	1Ø5377
1ØØØ3	162424
.	.
.	.
1ØØ12	62727
1ØØ13	165514
.	.
.	.
1ØØ22	72657
1ØØ23	175204

All addresses that end in the digit "2" have contents that end in the digit "7". All addresses that end in the digit "3" have contents that end in the digit "4", etc. A couple of general rules can be made from this:

1. If the right-most digit of both the good and bad data agree, there is a good chance that either a data bit was picked (which is obvious) or an address bit was dropped or picked. If it is an address bit, predominate possibilities would be a logical address bit between 1 and 12 or a physical address bit between PA6 and PA12. Physical address bits KPAØ - PA5 could also cause the failure but are less likely since addressing the wrong "page" should in the majority of the cases cause the right-most digit not to agree.

2. If the right-most digit of the good and the bad data does not agree, I would suspect that the data was read from or written to the wrong "page". Possibilities would be KPAØ - PA5 or the MMPU1.

The above are general rules and cases will most likely occur which will break them but not as frequently. Try to accumulate two or three failures and then compare the failure modes to make a first guess.

C. Failure Modes

1) Sub-Process Detects a bit drop

If a failure occurs, during the checkout of memory, the following information is typed on the CRT:

a) Error Page - physical memory page error occurred in
b) Offset - failing word within the memory page (Ø-1777)
c) Good - good data
d) Bad - bad data

There are situations where the "Error Page" may not be the true failing page. This is especially true in a system where CONTEST is being run in addition to many other Processes. The reason for this is that the sub-

processes that check core are not resident processes but swapable processes. If they are swapped to disk between the time the error occurs and the time the error is detected, the "Error Page" will be wrong because the swap has taken place.

Since they are not always wrong, if you can accumulate a number of failures many of which point to the same physical page, that would be the first memory board that I would swap. In a 256KW system, that technique may help you to at least make "educated guesses" at the failing module.

This problem is presently being addressed with Systems Programming and hopefully a future revision will cure this problem.

2) "Fatal Memory Error . . . "

If any of the sub-processes either "Trap" or "no longer respond" (such as going into a tight loop), the monitor program will abort and type the following message on the screen:

"Fatal Memory Error - Process Died"

then return to CLI prompt level. No information is printed on the CRT or put into any log file. If this happens frequently, it might be easier to swap memories one at a time and run CONTEST to see if a failure occurs. If sufficient good spare memories are available, the swap block could be 32Kw. This would allow rotating a 256Kw system totally in 8 executions. This is not a valid long term solution and is presently being addressed.

If the problem is very intermittent, the above swap technique is totally inadequate. In this case, the AOS Operating System can be "patched" to halt when a Trap occurs allowing a core dump to be taken. The core dump can then be analyzed to determine the core that was in use at the time of failure. Prior to patching an Operating System, a copy should be made and the copy patched.

This way if a mistake is made, there is still a good version on the disk pack.

The following procedure can be used to determine the core being used by the failing process.

a) "Copy" the Operating System to be patched:

```
)COPY  TEST.SY   <operating-system-name.SY>↵
)COPY  TEST.ST   <operating-system-name.ST>↵
```

b) Patch the Operating System to "halt" if a trap occurs. The following patch is for AOS Rev 1.Ø5 but should be close for the other revisions:

```
)X DEDIT/S=TEST.ST  TEST.SY↵
AOS FILE EDITOR REV 1.1
+MAGIC+47: Ø24Ø32+↵
+MAGIC+5Ø: 13371Ø+↵
+MAGIC+51: 1Ø2Ø7Ø + 63Ø77↵
+BYE↵
)
```

Note that the first two locations are for reference only. This will help if the offsets change slightly in a future revision. If the EXEC is not up, make sure the "Search list" includes ":UTIL" or a "File not found" can occur when invoking DEDIT.

c) Boot up the patched Operating System. Below is an example from my system.

```
AOS REV Ø1.Ø5
DATE? 11 19 77↵
TIME? 15 6↵
OVERRIDE DEFAULT SPECS?  Y↵
SPECIFY THE MASTER LOGICAL DISK
?DPFØ↵
?↵
GIL2Ø
SYSTEM PATHNAME? :SYSGEN:TEST.SY↵
SYSTEM SHUTDOWN
```

AOS REV Ø1.Ø5

etc.

d) Run CONTEST until a failure occurs and the Operating System halts. Note that no message will appear on the CRT since the halt takes place prior to any message handling routine.

e) Take a core dump by pressing RESET; putting 14 in the switches; pressing START on the computer console. The core dump message will be asked. Refer to Section 6, Pages 6-2 thru 6-4 for additional core dump procedures.

f) Run FIXUP on the disk

g) Load the core dump onto the disk. Refer to Figure 8.4c. The "Discussion" for a Panic 12 on pages 6-83 thru 6-88 should be reviewed for the below notes to be more meaningful.

   1) location 365 is the address of the Process Table that trapped which will be referred to as "PTA".
   2) PTA+16 is the address of the Process Table Extension, "PTE".
   3) PTA+25 is the address of the Process Table "Logical to Physical" Translation table for the core being used by the Process, "PTM".
   4) PTA+75 is the Physical Page that contains the Extension and the Map. Only bits 3-15 are meaningful.
   5) We now "Map" the Physical Page from PTA+75.
   6) PTE+66 is the Map DIC. Note that the C33Ø DIC has been converted to a C3ØØ DIC. Refer to Table and discussion on Page 6-86.
   7) PTE+124 thru PTE+13Ø contain the AC's and the "C+PC" at the time of "trap".
   8) PTM+Ø thru PTM+37 will contain the actual Map "Logical to Physical" translation for each Map slot.

Figure 3.4c "CONTEST" core dump analysis

```
)DIR :UTIL ↵
)COPY CONDUMP  3MTAØ:Ø ↵
)X  SYSDMP/S=:SYSGEN:TEST.ST  CONDUMP ↵
 AOS SYSTEM DUMP ANALYZER REV 1.1
+365:Ø442ØØ+ ↵        Process Table that trapped
+
+442ØØ+16:614ØØ+ ↵          Address of Process Table Extension
+442ØØ+25:617ØØ+ ↵          Address of Process Table Map
+442ØØ+75: 6247+ ↵          Extension and Map Physical Page
+
+MAP 247 ↵                  Map that Page!
+
+614ØØ+66: Ø1ØØØØ+ ↵        Map DIC bits for Trap
+
+614ØØ+124: Ø+ ↵            ACØ
+614ØØ+125: Ø+ ↵            AC1
+614ØØ+126: 7Ø7+ ↵          AC2
+614ØØ+127: Ø+ ↵            AC3
+614ØØ+130: 457+ ↵          C+PC
+
+617ØØ: 176+ ↵              Page Ø
+617ØØ+1: 2175+ ↵           Page 1
+617ØØ+2: 1Ø5777+ ↵         Page 2
+617ØØ+3: 1Ø7777+ ↵         Page 3
    .       .                .
    .       .                .
    .       .                .
    .       .                .
    .       .                .
    .       .                .
    .       .                .
+617ØØ+36: 175777+ ↵        Page 36
+617ØØ+37: 177777+ ↵        Page 37
+
+BYE ↵
)
```

We now know the reason for the trap (type violation), the AC's and the PC when the trap occurred, and the physical core being used by the process.

h) Using the above information, determine why the Process trapped and/or which memory to swap first.

In the example in Figure 8.4c only two memory slots are being used with the physical pages being 176 and 175. Therefore I would swap module 3 since my C33Ø has 32Kw MOS modules and is not interleaved. It is always easier to troubleshoot this type problem on an uninterleaved machine.

Note that MPROC.PR, the program that exercises core and does disk I/O, is only 2Ø74 words in length. The rest is used as "scratch" area. If the "trap" occurs prior to the process getting this "scratch" area from the Operating System via the ?MEMI System Call, only 2 pages will appear in the "Logical to Physical" translation table (as was the case in Figure 8.4c). Since only the code in the first 2Ø74 words is ever executed, if a trap occurs and the "PC" is much larger, such as 34Ø32, I would suspect a failure in the first two logical pages which caused the program to "jump off" prior to the trap occurring. This would be a "case #2" condition as explained in Section 5 dealing with Traps around Page 5-7.

The ideal situation is to have two or three core dumps to analyze. The memory pages being used can be compared between all the dumps to see if they reside in the same module. If they do, then swap the module. If they do not, then I would either suspect multiple core modules or a problem with the map, MMPU or MMPU1. I would put a lower suspicion factor on the CPU boards.

If you are having problems with CONTEST and other levels of support are not available, contact me at National Tech Support and I will be glad to help.

3) "Illegal Destination Port"

The error has the same causes as in 2) above. The only reason the message is different, is because of certain AOS system timing. If events happen one way, message 2) will result; if they happen a different way, message 3) will result. The cause is the same, a sub-process "Trapped". Troubleshoot using the procedures in 2) above.

4) AOS Panic

Another failure would be for AOS to "Panic". Since we are trying to exercise the system as a whole, there is a possibility of our causing a memory failure, etc, in an AOS area as well as our own test area. These "Panic" failures should be troubleshot the same way as if a User were operating the system. Refer to Section 6 which discusses "System Panics" and the procedures to troubleshoot them.

I would always suggest running the basic diagnostics first such as a "CRUNALL' under DTOS. There are some failures that they will pick up that EMORT will not. Next, I would run EMORT L for a minimum of 24 hours with a disk

Again I will gladly give additional assistance with any CONTEST failure.

5. SYSTAPE.CLI macro

The SYSTAPE macro is illustrated in Figure 8.5. It is used to generate a "backup" tape of the Operating System and all system Utilities after an AOSGEN is done so the System disk can be restored in case of a system crash. User files and directories are not backed up via this macro. A separate procedure would have to be used. No editing is needed for this macro and it resides in the :SYSGEN directory. The macro requires only two arguments:

a) The Mag Tape Unit on which your tape is mounted.
b) The filename of the Operating System including the .SY extension.

A sample command line to invoke this macro is:

```
)SYSTAPE  @MTAØ  AOS.SY ↵
```

Note that this must be done after DIR'ing to the :SYSGEN directory.

Now we should discuss exactly what the SYSTAPE macro does. Each command line in Figure 8.5 is numbered and will be discussed below:

Line 1 - this line checks that there are only two arguments in the command line as required and that no switches are specified. If either is not true, the message pointed to by the arrow is typed out.

Line 2 - the Mag Tape Unit specified as argument #1 is rewound.

Line 3 - the "environment" level is now pushed. This allows it to be changed temporarily and later POP'd to its previous state.

Line 4,5 - the environment is changed by turning "Super On" and setting CLASS 2 to ERROR.

Line 6 - changing the current directory to the Root (:).

Line 7 - write TBOOT to File Ø

Line 8 - dump certain needed programs into File 1.

Line 9 - write the disk formatter (software), DFMTR, to File 2.

Line 10 - write the disk Operating System Installer, INSTL, to File 3.

Line 11 - change the current directory to the previous one, :SYSGEN.

Line 12 - write the Operating System specified in the macro command line to File 4. If no ".SY" extension was specified, append one prior to executing the "COPY" CLI command.

Line 13 - back to the Root (:) directory.

Line 14 - write the disk fixup program, FIXUP, to File 5.

Line 15 - dump the contents of the :UTIL, :HELP, and :SYSGEN directories into File 6.

Line 16 - "POP" the environment

Line 17 - rewind the Magnetic Tape being used.

The rest of the macro contains the error messages and the [!ELSE] / [!END] for their [!EQ..].

```
 1  {!EQ,{{!EQ,%1%,!NO_1{!END}{!EQ,%2%,!NO_2{!END}%3-%%/%}},{!
 2   REM %1%
 3   PUSH
 4   CLASS2 ERROR
 5   SUPER ON
 6   DIR :
 7   COPY %1%:6 TBOOT
 8   DUMP/V/2=WARNING %1%:1 GHOST.PR GHOST.OL CLI.PR DEBUG.OL ERMES &
       DEBUG.ST PMGR.PR FIXJP DFMTR INSTL TBOOT
 9   COPY %1%:2 DFMTR
10   COPY %1%:3 INSTL
11   DIR/P
12   COPY %1%:4 %2%{!NE,,{!FT,%2%.SY{!}.SY{!END}
13   DIR :
14   COPY %1%:5 FIXJP
15   DUMP/V/2=WARNING %1%:6 + <UTIL, HELP, SYSGEN> :+
16   POP
17   REM %1%
18 {!ELSE}
19   WRITE *ABORT*
20   WRITE REQUIRED ARGUMENTS: 1 - UNIT NAME
21   WRITE,,,,,,,,,,,,,,,,,,,,,,2 - SYSTEM NAME
22   WRITE OPTIONAL SWITCHES: NONE
23 {!END}
```

Figure 8.5 SYSTAPE macro

```
X FORTRAN/%0/4 %-%
```

Figure 8.6 FORTRAN 5 F5 macro

```
X BIND/%0/% %-% F5ISA.LB F5MATH.LB F5IO.LB F5ENV.LB
```

Figure 8.7 FORTRAN 5 F5LD macro

SECTION 9

MISCELLANEOUS SYSTEM PROBLEMS

This section will discuss the remainder of the possible system problems, most of which produce error messages. The procedures to follow will be specific on a message by message basis. The messages discussed are:

In many of the above cases, an intermittent, possibly recoverable, error will occur <u>prior</u> to the system finally crashing. The customer should be made aware that just because only "Soft" or "Single bit" errors are occurring, they should be reported. A pattern of soft errors will many times occur <u>prior</u> to a total (hard) failure. By fixing the cause of the soft errors through either preventive maintenance or diagnostic troubleshooting, a future system "Panic" may be avoided.

I Abnormal System Shutdown

Figure 9.1 illustrates the cases that can occur when shutting down the Operating System along with the system variables which differentiate between these cases. The "Abnormal ...." message can occur in two cases:

a) the Operator CLI (PID = 2) trapped and this message is typed prior to the Panic code 1Ø message.

b) during a System Shutdown, the Root CCB (RTCCB) is found with a "Use Count" greater than 1. Since this count is ISZ'd/DSZ'd, respectively, unless a problem occurs, the count should be "1" after all the DSZ's are done.

When either of the above cases occur, the status bit on the disk which indicates a "crash" will be set and FIXUP must be run prior to re-booting the Logical Disk(s). If this is not done, a "System Error code 243" will occur on booting and is defined as "cannot Init LD, run FIXUP on it".

Note that in case b) above, the final "Use Count" in the Root CCB will be faked to "1" during shutdown. The procedure to analyze the core dump to determine the reason for the non-one use count is complex. If this problem consistantly occurs, I would suggest submitting a Core Dump in accordance with the procedures outlined on Pages 6-2 thru 6-5 of Section 6, "System Panics".

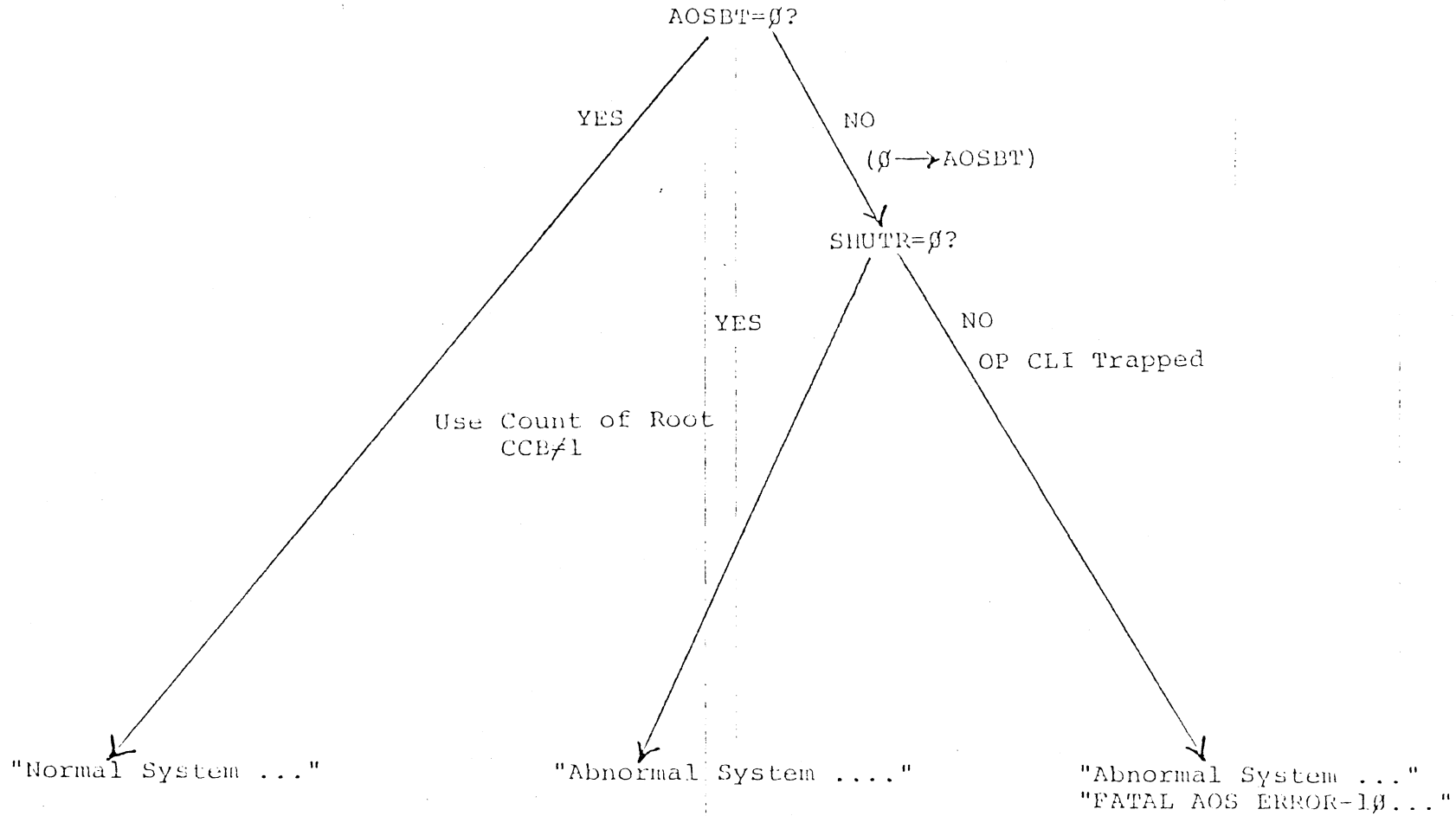


Figure 9.1 System Shutdown Types

II Memory Failures

On Eclipse machines with Error Checking and Correction (ERCC) memories, when an error occurs, an AOS subroutine in the Interrupt Service module (INTS) will read the "fault" code and thereby determine whether an ERCC error is a single or multi-bit error. There is a 5 bit "check field" constructed by the hardware each time a word is written. The "check field" is re-constructed and compared with the original field each time the memory word is read. This "check field" allows the hardware to correct single bit errors and "flag" multi-bit errors. If it is a multi-bit error, a Panic 13 will result which is discussed in Section 6, "System Panics". If it is a single bit error and SYSLOG has been started, the error will be logged to the disk. This log can be typed out using the REPORT utility with the "/ME" switch. The date and time of each error is given along with the "Physical Address" and ERCC "failure code". More on using each one of these values will be discussed further on. The REPORT utility is discussed in Section 7, "Utilities".

It should be noted that if the System Logging function is active, not all errors will be recorded although all single bit errors will be corrected by the hardware. All errors will not be logged if errors occur immediately one after the other. The reason for this is the overhead in recording the errors - the disk file must be updated, an overlay is needed (SYSER), etc. Note also that after 400_8 errors occur, a message as shown in Figure 9.5 containing information about the last error will be typed on the Operator Console and the ERCC will be put in Mode 2 - "Enable checking and correction, do not interrupt on memory error." This might pose a problem if a multi-bit error were to happen at this point although the problem is obvious since SYSLOG is full of error printouts (if logging were enabled).

The hardware ERCC option uses two different formats, one for the S130 and one for the S2xx/C3xx machines. Presently, at least through AOS Rev 1.06, the system attempts to handle both formats in the same way, causing the ERCC "Physical Ad-

dresses" for an S13Ø to be wrong. The existing formats will be discussed below with the S13Ø problem being corrected in a future revision.

a) S13Ø Eclipse

The accumulator breakdown is listed in the section describing Panic 13. The differences are that full physical addresses are not given. Since AOS treats the accumulators as a full physical address, the resulting AOS physical address must be rebroken down, then examined for the proper contents. For example, the first entry in the SYSLOG in Figure 9.3 lists the Physical Address as "355ØØØ5". The rightmost 16 bits would be the DIA and any remaining bits would be the DIB. Now given:

```
11  1Ø | 1  1Ø1  ØØØ  ØØØ  ØØØ  1Ø1  => 355ØØØ5
DIB    |           DIA
```

Therefore, the actual accumulators would be:

DIA = 15ØØØ5
DIB = 16 (excluding Fault code)

Using the formats under Panic 13, we can find:

Low order bits = 12
Page = 2
HOB's = 1

From this we can find the full page address as "22" via:

```
ØØ  Ø1 | Ø  Ø1Ø  => 22
 HOB's | Page
```

Remember that both the original DIA and DIB values have to be complemented. We can now reference Table 9.1 to convert the "ERCC Page" to the physical memory pages. We find that:

ERCC Page 22 = Physical Memory Pages 11Ø-113

Note that each "ERCC Page" is 4 memory pages. Since the system had uninterleaved 32kw SC Memories, board #2 (starting with board Ø) would be swapped. If the system were interleaved, the LOBs would be used to determine the actual memory. In our

Table 9.1 ERCC/Memory Page Conversion

ERCC "Page"	Beginning Memory Page
Ø	Ø
1	4
2	1Ø
3	14
4	2Ø
5	24
6	3Ø
7	34
1Ø	4Ø
11	44
12	5Ø
13	54
14	6Ø
15	64
16	7Ø
17	74
2Ø	1ØØ
21	1Ø4
22	11Ø
23	114
24	12Ø
25	124
26	13Ø
27	134
3Ø	14Ø
31	144
32	15Ø
33	154
34	16Ø
35	164
36	17Ø
37	174

case, we determined the memory using a Physical of 11Ø as follows:

```
ØØ1  Ø | Ø1  ØØØ  => 11Ø
board  | 32 1K pages
 #2    |    per board
```

The reason for the ERCC change on an S13Ø is that the 32Kw MOS memories use 4Kx1 RAM's (DGØ59Ø) and by knowing the bit that failed and the 4K section, the failing chip can be replaced.

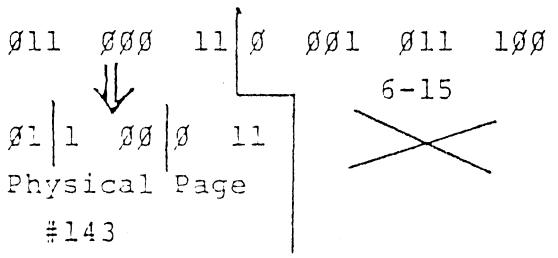
The fault code given is the binary equivilent of the octal code. In the first entry of Figure 9.3, the code is "34" and is determined as:

```
11 | 1ØØ
3  | 4
```

The code of 34 is defined as "Data Bit 15" by referring to Table 9.2.

b) S2xx/C3xx Eclipse

The accumulator breakdown is listed in the section describing Panic 13. The addresses given in the DIA/DIB is the actual physical address. The only difference between the S2ØØ and S23Ø, and the C3ØØ and C33Ø is that an extra physical address bit is used in the DIB. The SYSLOG formats will be similar to those shown in Figure 9.3. The address will be the actual physical address. This can be broken down to the physical memory page by disregarding bits 6-15 initially and checking the high order bits. For example if the SYSLOG address was 3Ø6134, we would:



If the Eclipse were an uninterleaved machine with 32Kw SC Memories, the module that would be swapped would be board #3 (starting with board #Ø). This was determined as:

Table 9.2 Single Bit ERCC Codes

CODE	FAILURE DESCRIPTION
Ø	No error
1	Check bit 4
2	Check bit 3
3	Data bit Ø
4	Check bit 2
5	Data bit 1
7	Data bit 3
1Ø	Check bit 1
11	Data bit 4
13	Data bit 6
14	Data bit 7
15	Data bit 8
16	Data bit 9
2Ø	Check bit Ø
21	Data bit 11
22	Data bit 12
23	Data bit 13
24	Data bit 14
26	Data bit 2
3Ø	Data bit 1Ø
32	Data bit 5
34	Data bit 15

ØØ1 1	ØØ Ø11 ⇒ 143
board	32 1K pages
#3	per board

If the system were interleaved, bits 13-15 would have to be taken into account to determine the actual memory.

The fault code given is the binary equivilent of the octal code. In the first entry of Figure 9.3, the code is "34" and is determined as:

11	1ØØ
3	4

The code of 34 is defined as "Data bit 15" by referring to Table 9.2.

```
15-DEC-77        16:50:39        REPORT REV 01.04

                        DEVICE ERROR DUMP

                    DEVICE  UNIT  STATUS RETRIES  TYPE
                     CODE
 2-SEP-77  9:15:52  *** SYSLOG STARTED ***
 2-SEP-77 10:01:52  *** SYSLOG STOPPED ***
 2-SEP-77 17:53:20  *** SYSLOG STARTED ***
 3-SEP-77 10:52:19  *** SYSLOG STARTED ***
 3-SEP-77 20:37:55  *** SYSLOG STOPPED ***
23-NOV-77 15:33:48  *** SYSLOG STARTED ***
23-NOV-77 20:14:55    27       0     10101      1      SOFT
23-NOV-77 20:32:03    27       0     10101      1      SOFT
23-NOV-77 22:37:00    27       0     60041      1      SOFT
23-NOV-77 23:46:09    27       0     10101      1      SOFT
24-NOV-77  0:20:02    27       0     60041      1      SOFT
24-NOV-77  3:20:55    27       0     10101      1      SOFT
24-NOV-77  3:52:16    27       0     60041      1      SOFT
24-NOV-77  4:09:30    27       0     60041      1      SOFT
25-NOV-77 15:17:42  *** SYSLOG STARTED ***
25-NOV-77 17:16:21    27       0     60041      1      SOFT
25-NOV-77 17:39:51    27       0     60021      1      SOFT
25-NOV-77 20:10:58    27       0     10101      1      SOFT
26-NOV-77  1:07:18    27       0     60041      1      SOFT
26-NOV-77  2:41:29    27       0     60041      1      SOFT
26-NOV-77  5:50:35    27       0     60041      1      SOFT
26-NOV-77  7:55:50    27       0     10101      1      SOFT
26-NOV-77 14:28:07  *** SYSLOG STOPPED ***
```

Figure 9.2 SYSLOG "Device Errors"

```
15-DEC-77        16:46:39        REPORT REV 01.04

                        MEMORY ERROR DUMP

                    ADDRESS  CODE
15-DEC-77 15:21:08  *** SYSLOG STARTED ***
15-DEC-77 15:21:08  3550005  11100
15-DEC-77 15:21:08  3420013  11100
15-DEC-77 15:21:08  3560003  11100
15-DEC-77 15:21:08  3420015  11100
15-DEC-77 15:21:13  3500005  11100
15-DEC-77 15:21:13  3500011  11100
15-DEC-77 15:21:13  3500010  11100
15-DEC-77 15:21:13  3500007  11100
15-DEC-77 15:21:13  3500006  11100
15-DEC-77 15:21:13  3500005  11100
15-DEC-77 15:21:13  3500004  11100
15-DEC-77 15:21:13  3500003  11100
15-DEC-77 15:21:13  3500002  11100
15-DEC-77 15:21:13  3500001  11100
15-DEC-77 15:21:43  *** SYSLOG STOPPED ***
**********************************
```

Figure 9.3 S13Ø SYSLOG "Memory Errors" (ERCC)

```
FROM SYSTEM:
HARD ERROR, DEVICE 22 0, STATUS 106101, RETRIES 15
```

```
FROM SYSTEM:
SOFT ERROR, DEVICE 22 0, STATUS 106101, RETRIES 01
```

Figure 9.4 Hard and Soft "Device" Errors

```
FROM SYSTEM:
SINGLE BIT ERROR, PHYSICAL ADDRESS 3730015, CODE 34
```

```
FROM SYSTEM:
SINGLE BIT ERROR, PHYSICAL ADDRESS 3420006, CODE 32
```

Figure 9.5 S13Ø Single Bit "Memory" Errors (ERCC)

III I/O Device Errors

If an error occurs on a device such as a disk unit or magnetic tape unit, a message is typed on the console and the error is logged in SYSLOG (if it had been "START'd"). The only exception would be if a "hard" disk error were to occur on the master logical disk while accessing a "system file". Then a "Panic 2" would occur which is discussed in Section 6, "System Panics". Note that any errors that result in a "Panic 2" will not be logged in SYSLOG irrespective of whether it had been "START'd".

Figure 9.4 illustrates both a Hard and a Soft device error message that would appear on the master console (device code 1Ø/11). A "Soft" error is one which has been retried and been successful without the retry count going to "Ø". The retry count is 7 for AOS revisions 1.Ø3 and below; and 15 for AOS revisions 1.Ø4 and greater. This count is "DSZ'd" on each error and if the count goes to "Ø", the error is defined as a "Hard" error. A "Hard" error will also be flagged if the error is a type that cannot be retried and the number of retries will be Ø. In addition to typing whether the error is Hard or Soft, the device code and unit number for the failing device, the error status, and the number of retries will be given. The standard device codes versus device names for all Data General devices are given in Table 9.3. The "error status" given will be the DIA for magnetic tape and all disks except the 6Ø6Ø (Zebra) and the 6Ø63, Fixed Head Disk. For the 6Ø6Ø, it will be either the DIA or DIB depending on which one the error occurred in. For the 6Ø63, it will be the DIC. The breakdown of the status words for disks and magnetic tape is given in Table 6.2.1 on Page 6-18.

Figure 9.2 illustrates a SYSLOG printout using the REPORT utility. The usage of the REPORT utility to generate this report is discussed in Section 7, "Utilities". The values given are the same as discussed above for the console message with the addition of the date and time of error.

Table 9.3 Device Codes vs. Device Names

Octal Device Code	Mnemonic	Device Name	Octal Device Code	Mnemonic	Device Name
00	----	Unused	34	MUX	Communications system controller (multiplexor) (ALM)
01	WCS	Writable control store	46	MCAT1	Second multiprocessor communications transmitter
02	ERCC	Error checking and correction	47	MCAR1	Second multiprocessor communications receiver
03	MAP	Memory allocation and protection	50	TTI1	Second TTY input
06	MCAT	Multiprocessor communications adapter transmitter	51	TTO1	Second TTY output
07	MCAR	Multiprocessor communications adapter receiver	52	PTR1	Second paper tape reader
10	TTI	TTY input	53	PTP1	Second paper tape punch
11	TTO	TTY output	54	RTC1	Second real-time clock
12	PTR	Paper tape reader	55	PLT1	Second incremental plotter
13	PTP	Paper tape punch	56	CDR1	Second card reader
14	RTC	Real-time clock	57	LPT1	Second line printer
15	PLT	Incremental plotter	62	MTA1	Second magnetic tape controller
16	CDR	Card reader	64	DCU	Data Control Unit
17	LPT	Line printer	66	DKB1	Second Model 6Ø63 Fixed Head Disk Controller
22	MTA	First magnetic tape controller	67		Second Model 6Ø6Ø disk controller
26	DKB	First Model 6Ø63 Fixed Head Disk Controller	73	DKP1	Second disk controller (except Model 6060)
27		First Model 6Ø6Ø disk controller	77	CPU	Central processor and console functions
33	DKP	First disk controller (except Model 6Ø6Ø)			

IV Power Fail

AOS presently does not support automatic system recovery after a Power Failure. Even though this is true, the computer front console should have the key in the "Lock" position to avoid the console switches accidentally being pressed. On machines with MOS memories, I would suggest keeping the key in "On" position not "Lock" because of the unpredictable results on power recovery. The following paragraphs will discuss what happens on a Power Failure and the visable results when power returns.

When the Eclipse detects a low voltage, an interrupt will be generated which will indicate to the Vector instruction a device code of "Ø". A mask of "-1" will be issued to all the system I/O devices which prevents them from interrupting. Because of a bug in all AOS revisions up to and including at least AOS Rev. 1.07, a "Panic code 1" will now occur. The reason is that interrupts are inadvertently turned back on prior to the power down routine. This will cause continuous nested interrupts until the interrupt stack is exhausted which causes the Panic. Physical location Ø will contain the address of PFL when the Panic occurs. If the "Power Fail" occurs because of "spikes" on the power line, the Panic code 1 message will be typed on the console. If the "Power Fail" occurs because of a full power outage, there may be insufficient time depending on the baud rate of the master console to type out the Panic code 1 message prior to completely loosing power. The master console will not have any message on it. Physical location Ø will have the address of PFL in it. When power returns and if the system is in "Lock", this address will be executed and unpredictable results will occur. Following are some values that can be checked. Some may be changed because of the unpredictable results on power up.

Location	Contents	Sample Value
Ø	PFL	15415
5	PFL Mask	177777
4Ø	Stack Pointer (SP)	2Ø16
42	Stack Limit (SL)	1Ø2ØØ1
up to Rev 1.Ø5: 332 1.06: 33Ø	Interrupt level (INTLV)	11

When the above problem is fixed, or if it is patched, the Operating System will hang at a "JMP." at approximately location PFL+13 waiting for power to cease. Note that it has already checked for a real power failure via a "SKPDZ CPU" instruction and has also stored a "4ØØ" or "JMP." in physical location Ø. When power returns and if the key is in "Lock", the restart hardware will execute the contents of physical location Ø. Since this contains a "4ØØ", the system will "hang" there. The above recovery symptoms and the hang at location Ø, are for machines with core only. On machines with MOS memories which are volitile, the content of physical location Ø on power being restored is unpredictable as is the final result.

Since a power failure is initially interpreted as an interrupt from device code Ø, if for some reason an interrupt with a device code of Ø were to occur which was not a Power Failure, a problem could occur. This is the reason for the check using the "SKPDZ CPU" instruction which will "skip" for a non-power failure. In all AOS revisions up to and including Rev 1.Ø5, an interrupt from device code Ø could cause the system to "hang" at Interrupt level. This problem has been cured in AOS Rev 1.Ø6 which throws away the interrupt. In all revisions, an interrupt with device code Ø will cause a counter, PCNT, to be incremented. If the counter increments 65K times, thereby causing it to overflow, a Panic 4 will occur.

SECTION 14
HARDWARE NOTES

This section will discuss problems and tips that relate to hardware devices. In many cases, failures that appear to be hardware are actually software. If you understand what AOS expects from the specific device, many times you can determine whether the problem is with the device or with AOS. Also included will be short comments denoting pitfalls that can be encountered with the specific device.

1. Programmable Interval Timer (PIT)

AOS requires a Programmable Interval Timer (PIT) for efficient operation and expects it to be jumpered in a specific configuration. The PIT is only used by AOS for "time slicing" - giving each User a section of CPU time. The actual algorithm used in the software is complex. If the clock rate is not jumpered properly, AOS performance can be two to four times worse than normal when executing Fortran compiles, etc., because of the higher interrupt activity. The PIT is available as a 4Ø68 (the DIO module) or a 4217 (the DCH LPT module). In either case, it should be jumpered for 1ØKHZ. The jumper configuration is:

	CLOCK JUMPERS	
MODULE	IN	OUT
4Ø68	W57	W56, W58, W55, W54
4217	W5	W4, W7, W6

AOS also expects the device code to be 43 and the mask bit to be "Bit 6".

Since there have been problems with the Data Channel Line Printer PIT being jumpered to the wrong frequency, this jumpering should be verified on each customer board and noted on the label on the module edge that it has been checked.

Other Notes

1. Any system checkout should include the running of the PIT diagnostic. At least one case has occurred where a PIT was causing system failures.

I Real Time Clock (RTC)

AOS requires a Real Time Clock (RTC) for operation. Note that this is different from RDOS in which the RTC was optional. This fact should be kept in mind on system installations and when the 4Ø1Ⱡ or 4Ø75 controller is swapped.

If there is no RTC in a system, AOS will not even boot up. The last message asked will be:

"Override Default Specs?"

then the system will "hang" in the Scheduler. This is discussed in more software detail in Section 4, "System Hangs". Basically, AOS does a delay for 2 seconds between initiating the Peripheral Manager (PMGR) and the Operator CLI (Op CLI). The timer which counts down the delay is triggered by the RTC. Since the delay never expires, the system hangs.

AOS expects the RTC to be device code 14 and using mask bit 13. The RTC frequency can be set during the AOSGEN and should usually be set for 1ØHZ to keep system overhead to a minimum.

III Data Channel Line Printer (LPB, LPB1)

AOS will use either the Data Channel Line Printer or a standard line printer. The type of line printer at a specific device code is defined at AOSGEN time. The line printers are accessed as:

	MNEMONIC	
Device Code	Under EXEC	Non-EXEC
17	LPT	LPB
57	LPT1	LPB1

The LPB uses two volatile memories. The "TAB" memory is contained on the controller and controls the tab positioning on a given line. If the CPU is shut off the TAB memory is cleared. The "VFU" memory is contained in the Line Printer and controls the line positioning on a given page. If the line printer is powered off, the VFU memory is cleared. The line printer should never be powered on or off while AOS is running as this can cause the system to crash or hang. The problem is with the line printer and is further discussed in FAB S1Ø14, Software.

The present release of AOS does not use the electronic VFU option, therefore the "Tape" Led will always be lit. Even with the "Tape" Led lit, the DCH line printer should work O.K. because a hardware default exists for an 11 inch form with three line skip over. This is determined by the J9 chip in the line printer. Under the AOS Operating System, the TAB memory default is every 8th column and is loaded at device "Open" time. There was a problem with TAB loading under AOS which was fixed in AOS Rev 1.Ø6. The symptoms were possible "garbage" being initially printed and "format" errors on printing. Other notes:

1. Since the electronic VFU is not loaded, if a <22> (channel command) is issued by a program, a "FORMAT" error will occur on the line printer if the Spooler is not being used. This could occur if a User were to print a "binary file".

2. Non-printable ASCII characters such as <ØØ> and <23> are removed by the spooler, therefcre no <4Ø> (space) will be printed by the LPT.

3. Remember ECO #6140A and 6141 which protect the DCH "load" codes. These "load" codes must be preceded by a "DOA" to be effective. This protects the inadvertant "turn-on" of the VFU or TAB memories.

4. If the User loads the electronic VFU via the ?WRB system call (Appendix C of the System Programmer Manual, 093-120-01), he must reload the VFU when he is finished with a default specification since AOS never uses it. See the note on Page C-5 about the ?ENOV bit for that system call.

5. If the LPB is accessed without spooling such as:

)TYPE/L=@LPB XXX

   The CLI process must be resident or an error such as:

   "ILLEGAL PROCESS TYPE, FILE:LPB"

   will be generated. Since DCH activity is done from a Process area, the Process must be made un-swappable or resident.

6. If you attempt to print a file which contains both upper and lower case on an upper case only line printer, spaces will appear for the lower case alphabetics. A patch has to be made to XLPT.PR to convert lower case to upper case alphabetics. Once the patch is installed, the conversion is always done. If there are both types of line printers on a system, there must be a patched and unpatched XLPT.PR and they are "RENAME'd" back and forth as the line printers are started.

7. Note that each Data Channel Line Printer uses 2 DCH slots in a data channel Map.

8. It is possible to disable perforation skip-over by removing W5 on the DAVFU and installing W4 to allow paper out switch to function.

IV Multi-Processor Comm. Adapter (MCA)

AOS will support the MCA - both the 4038 and the 42Ø6 within the hardware guidelines. It had been initially stated in various places that AOS would support only the 42Ø6 because the 4Ø38 will run only on "A" Data Channel Map. The 4Ø38 will run on both "A" and "B" Data Channel Maps for a 23Ø or 33Ø. Either MCA would be assigned by AOS to "B" Map and will use 1Ø data channel slots - 5 for transmit and 5 for receive.

Other Notes

1. Processes that use the MCA must be of the "Resident" type when attempting to use the MCA.

V. Communications Chassis

The Comm. Chassis under AOS is used to support the Asynchronous Line Multiplexors (ALM) and Synchronous Line Multiplexors (SLM). Both ALM's and SLM's can be mixed in a Comm. Chassis if the Comm. Chassis is not driven by a DCU5Ø and if there is an empty slot between the modules to break the board priority chain. Under Rev 1.xx AOS, only ALM modules are supported being driven by a DCU-50. There are a couple of problems that apply specifically to the Comm. Chassis:

1. If system failures are occurring when the Comm gear is initially used, verify:

   a) If the ALM was a field installation, that ECO #4167A has been installed on the CPU backplane. This cures a "Y" termination problem on the INTP/DCHP signals.

   b) That all the "pico-fuses" on the Comm Chassis are good.

   Note that a) only applies to Comm. Chassis' that are not driven by a DCU-5Ø.

2. This second problem will not cause a system to crash but will degrade AOS's performance. When the UP Macro is executed, a file called "CONSOLES" is accessed to determine the console numbers the EXEC attempts to enable. By default, the system tries to enable CON1 thru CON17 because the numbers 1 thru 17 are contained in the "CONSOLES" file. If the respective ALM line had been AOSGEN'd and is included in the "CONSOLES" file, the EXEC will attempt to send the LOGON message to the line. If there is a test plug on that line, this message is returned on another line. Because of this sequence, AOS is kept busy servicing interrupts and returning "invalid" message text strings. Without a DCU5Ø, the system throughput is decreased significantly since all the overhead falls on AOS. This decrease is not apparent unless the given programs have been run on a system w/o the problem - the system just seems slow.

The solution to this problem can be done in one of two ways:

a. Remove the test plugs from the unused lines on the back of the ALM chassis.

b. Have the customer modify the "CONSOLES" file so that only "CONx" with a device connected to them are enabled.

Either way will cure the problem, it depends on how frequently the "test plugs" are used for diagnostics.

3. If the priority chain between the expansion chassis and the Comm Chassis is broken and EXEC is brought up, a Panic code 1 will occur. Note that this can happen because a board has been removed without jumpering around the slot.

A. Synchronous Line Multiplexor (SLM)

The SLM runs under AOS as device code 44 but is not supported running off of a DCU-5Ø. The hardware CRC (4266) is required for AOS SLM support. The SLM uses mask bit 8 and an AOS mask of "ØØØ377" up through AOS Rev. 1.Ø4 and "ØØØ777" from AOS Rev. 1.Ø5 on.

Other Notes

1. Patches are needed when using Sync gear under AOS Rev. 1.Ø6 to prevent system crashes.
2. When the SLM is operated with a modem, the clock jumpers W5 for line Ø and W6 for line 1 must be removed. Diagnostics require these jumpers so they could accidentally be left in. The symptoms are not obvious and has caused much time tracking a non-existant problem.

B. Asynchronous Line Multiplexor (ALM)

The ALM runs under AOS as device code 34 and is supported with or without a DCU-5Ø. Note that the DCU-5Ø is device code 64. The ALM uses mask bit 8 and an AOS mask of "ØØØ377". The DCU5Ø uses mask bit 4 and an AOS mask of "Ø77777".

Other Notes

1. The Rev. 1.xx AOSGEN only allows ALM lines Ø-31 to be specified. If more are needed, a patch can be made.
2. The DCU5Ø uses 17 data channel slots in a data channel map.
3. On systems without a DCU5Ø, the device code of the ALM is 34. There can be a contention problem between a Cassette I/O board (4Ø75) and the ALM even if the cassette option is not on the 4Ø75 board because the BUSY/DONE logic for device code 34 is still there. TIB S1Ø27, Communications, discusses this problem in more detail along with corrective action.

VI Low Cost Display (LCD)

AOS supports the LCD in any console position that has been AOSGEN'd for a 6Ø62/53. If the console had been AOSGEN'd for another type device, the following CLI command can be issued:

)CHAR/6Ø5x ↵

Note that the "x" above is actually an "X" and not a fill-in. The big difference between the 6Ø12 and the 6Ø5x is cursor positioning. If there is a problem with cursor positioning, especially under the utility PED, issue:

)CHAR ↵

and the type console ASOGEN'd for that line will by typed on the 1st line. Make sure it is a "6Ø5x".

There is a hardware problem with the Dasher Display (Model 6053 ONLY). This is documented in FAB S1Ø11, Display/Terminal which includes a hardware fix. The problem is that when the "Break" Key is depressed, a function code of "Ø36-223" is sent out prior to the "Break" character appearing on the line. The ASCII representation of the "223" is a "CNTRL S".

All revisions of AOS, and revisions of RDOS from Rev. 6.1 on, use the "CNTRL S" character to inhibit display of data on your console. No data is lost and a "CNTRL Q" will resume output. Once the "CNTRL S" is typed, all input from the console and output to the console is "Spooled" and for AOS, any commands typed will be executed just not echo'd. When the "CNTRL Q" is typed, all the "Spooled" data will appear on the console.

Under AOS or RDOS, this gives the appearance of a "Console Hang". Nothing typed on the console will be echo'd. If you encounter this situation, first type a "CNTRL Q" to see if this is the problem.

Other Notes

1. The LCD will check parity on both input to the display and output from the display. For AOS, the "parity switch" on the display should be in the unmarked middle position of

the three position switch, "No Parity". In either of the two other positions (EVEN, ODD), Parity errors will appear on the screen as AOS echoes characters. A parity error is signified on the display as:

6Ø52 - an ASCII underscore, < 137 >, will appear instead of the character. Do not confuse with the underscore attribute of the 6Ø53!!

6Ø53 - A "block" character, < 177 >, will appear instead of the character.

VII 6Ø6Ø (Zebra) Disk

AOS supports both the single density (6Ø6Ø) and double-density (6Ø61) disk drives. They are accessed under ACS as DPFØ-3 for device code 27 and DPF1Ø-13 for device code 67. The disk subsystem uses two CPU slots, requires an Adapter, and use mask bit 7. The AOS mask was "ØØØ717" up to AOS Rev 1.Ø4 and "ØØØ517" from AOS Rev 1.Ø5 onward.

Other Notes

1. Be aware that although he 6Ø6Ø and the 4231 (CDC) use the same physical disk pack - 3336 type pack, once they are formatted, they are not interchangeable between models, ie: a 6Ø6Ø pack cannot be read or written on a 4231 drive; and a 4231 pack cannot be read or written on a 6Ø6Ø drive. The symptom would probably be constant "address" errors.
2. Currently, "CRUNALL" under DTOS which is in the recommended AOS hardware test sequence does not support the 6Ø6Ø, 6Ø61. Use "RUNALL" until it is fixed.
3. The 6Ø6Ø/6Ø61 uses 9 data channel slots in B data channel map.

VIII Card Reader (CDR)

AOS will always support the buffered card reader controller (43Ø6) and with limited support of the old card reader controller (4Ø16). Both card readers use device code 16 or 56, and mask bit 1Ø. The AOS mask used is "ØØ1777". The 43Ø6 is a fully buffered 8Ø column card reader controller and supports the Documation card readers (4Ø16C-4Ø16J).

Because the old card reader controller (4Ø16) is not 8Ø column buffered, there can be a problem with losing characters when used with a 1ØØØ CPM reader under AOS. The problem is that AOS is unable to respond to interrupts in required time. Since the response time is a direct function of system activity, a card reader may work one time and not others. Systems Engineering has stated that the "4Ø16 should not be configured with AOS systems."

Other Notes

1. Note that the 43Ø6 does not support the D/A converter (4Ø37) or the Scope Control (4Ø53) options.
2. The EOF card has all holes punched in column 1.
3. Hopper empty does not generate physical end of file. Cards may be added to continue operation.
4. The M6ØØL Documation Card Reader, under normal operation, uses column 81 for an all dark check to verify the operation of the light sensing photo cells during a Read operation. If column 81 is used for punched data, the all dark check must be disabled. Refer to TIB S1ØØ1, Card Equipment for the disabling procedure.

IX Fixed Head Disk (DKBx)

AOS supports both the 1MB (6Ø63) and the 2MB (6Ø64) units but only the 2MB disk can be used as a swap device. They are accessed as DKBØ-3 for device code 26 and DKB1Ø-13 for device code 66. The disk sub-system uses two CPU slots and mask bit 9. The AOS mask is "ØØØ517" and 7 data channel Map B slots are used. The AOS Fixed Head Disk support began in AOS Rev. 1.Ø7.

X. Miscellaneous

1. Centronix line printers should be ACSGEN'd with the ?MNAS characteristic.

2. The 3Ø CPS Dasher can use cable ØØ5-7428. There is a wiring error on old cables. If there is a green wire (CTS) going to Pin 7 of the black Amp connector, it should be removed and taped up at the computer end. ECO 61Ø6 deletes this wire from the cable. If this wire is not removed, it can cause noise on the "INTP" signal.

3. The 6Ø CPS Dasher required "CTS" for proper operation. The "CTS" signal is a busy indication asserted when the FIFO is full, which occurs on a carriage return in 6ØØ baud operation, or a loss of "Ready", caused by a paper fault. Verify this signal is properly connected or "lost characters" can occur at high speed operation.

4. On a system with a 1ØMB disk (Diablo 44 or DG/Disk) without ECO #5888 installed, if the CPU is powered off while the disk is "Ready" (not cycled down), there is the potential to "glitch" the disk pack. This can later show up under AOS as a Panic code 2 because of a "checkword" error, etc., on the disk pack. This ECO should be installed and will cure the problem by deselecting the disk when the CPU drops +5VDC.

5. If a system with a 4231 (CDC) disk is experiencing Panic code 2's (disk error or timeout) with the "Seek Error" status bit set but a retry count of "Ø", it is possible that a timeout occurred during the seek which generated no attention to the controller. Normally a Seek Error is a retryable condition. Further information on this problem is contained in TIB S1Ø58, Moving Head Disk.

6. If your system is experiencing "flakey" failures such as Fault's on a CDC (333Ø), losing line printer interrupts, etc., make sure that the Pico fuse that supplies +5v to the External I/O terminator has not blown. If an ALM cable is

attached to the Extended I/O Bus, the terminator is attached to the Comm. Chassis and the Pico fuse on the Comm. Chassis supplies the voltage to the terminator. Refer to the tips in TIB S1Ø26, Processors, when changing the Pico fuse.

7. If having trouble reading Magnetic Tapes on all units of a multiple tape drive system, verify that the last tape drive in the daisy chain has the power applied. This drive has the terminator on it and with no voltage to the drive, there is no termination.

8. If intermittent "crashes" are occurring and the system has 16Kw Split Sense core modules, verify that each module has ECO #6371 installed. This prevents an intermittent dropping of bit Ø.

9. The 16ØØ bpi Magnetic Tape (4196) is supported by AOS with either W5 (4Kw max) or W6 (65Kw max) in. AOS will only support a maximum transfer of $4Kw_{1Ø}$ since only 5 DCH Map slots are assigned. A 4Kw transfer is also the maximum size for the 8ØØ bpi (62Ø2Ø) tape drive.

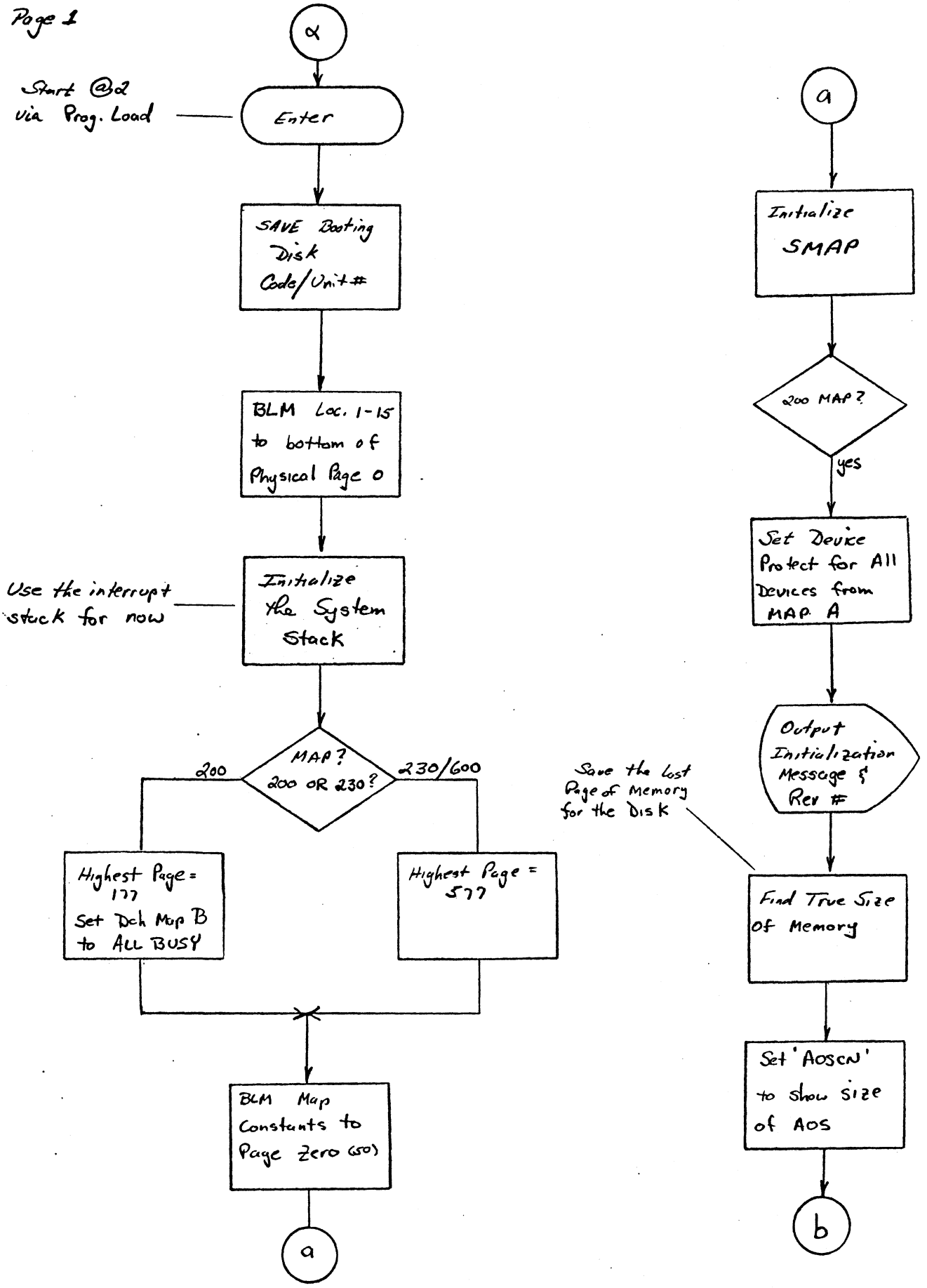
SECTION 14

HARDWARE NOTES

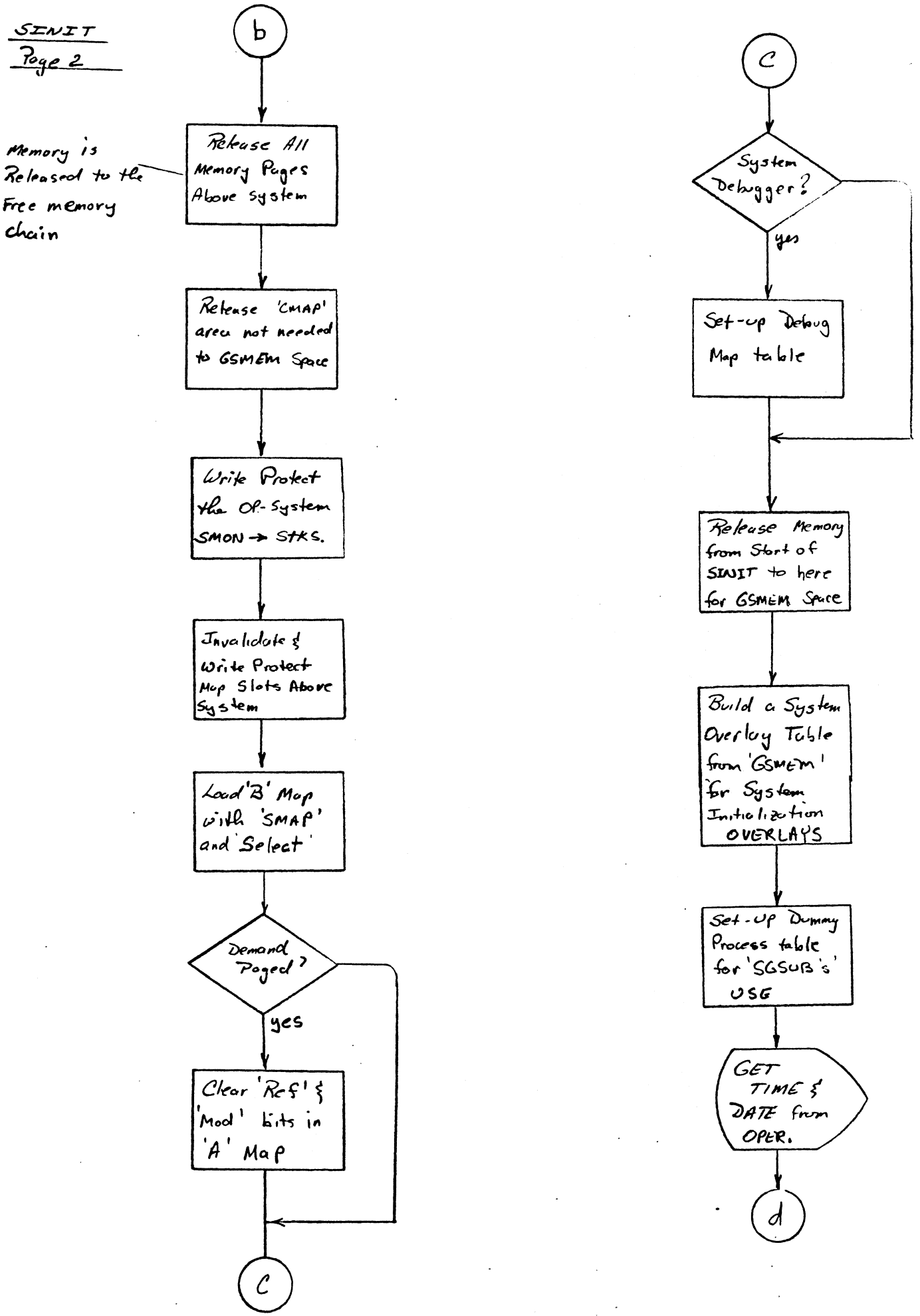
This section will discuss problems and tips that relate to hardware devices. In many cases, failures that appear to be hardware are actually software. If you understand what AOS expects from the specific device, many times you can determine whether the problem is with the device or with AOS. Also included will be short comments denoting pitfalls that can be encountered with the specific device.

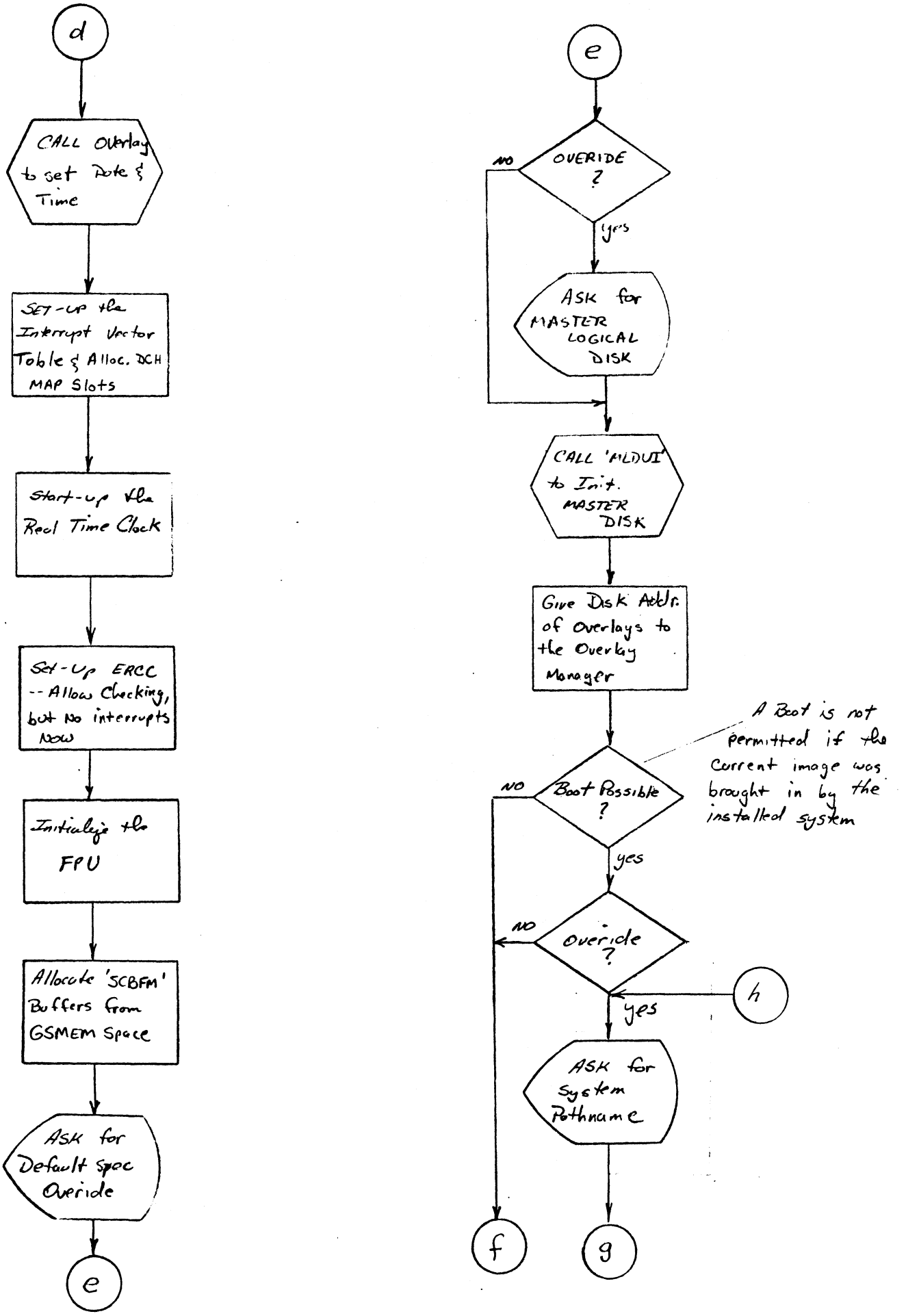
SINIT

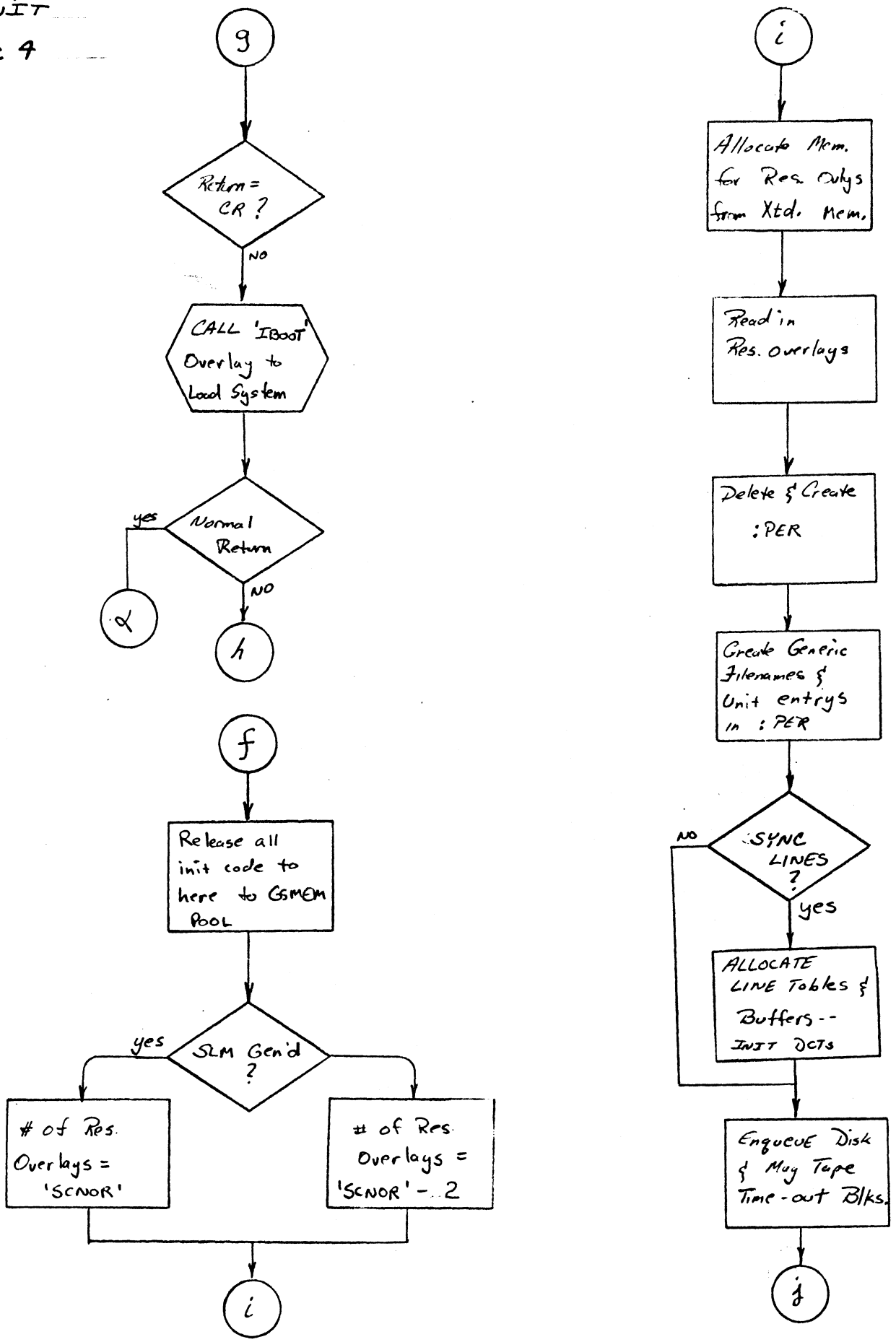
Page 1

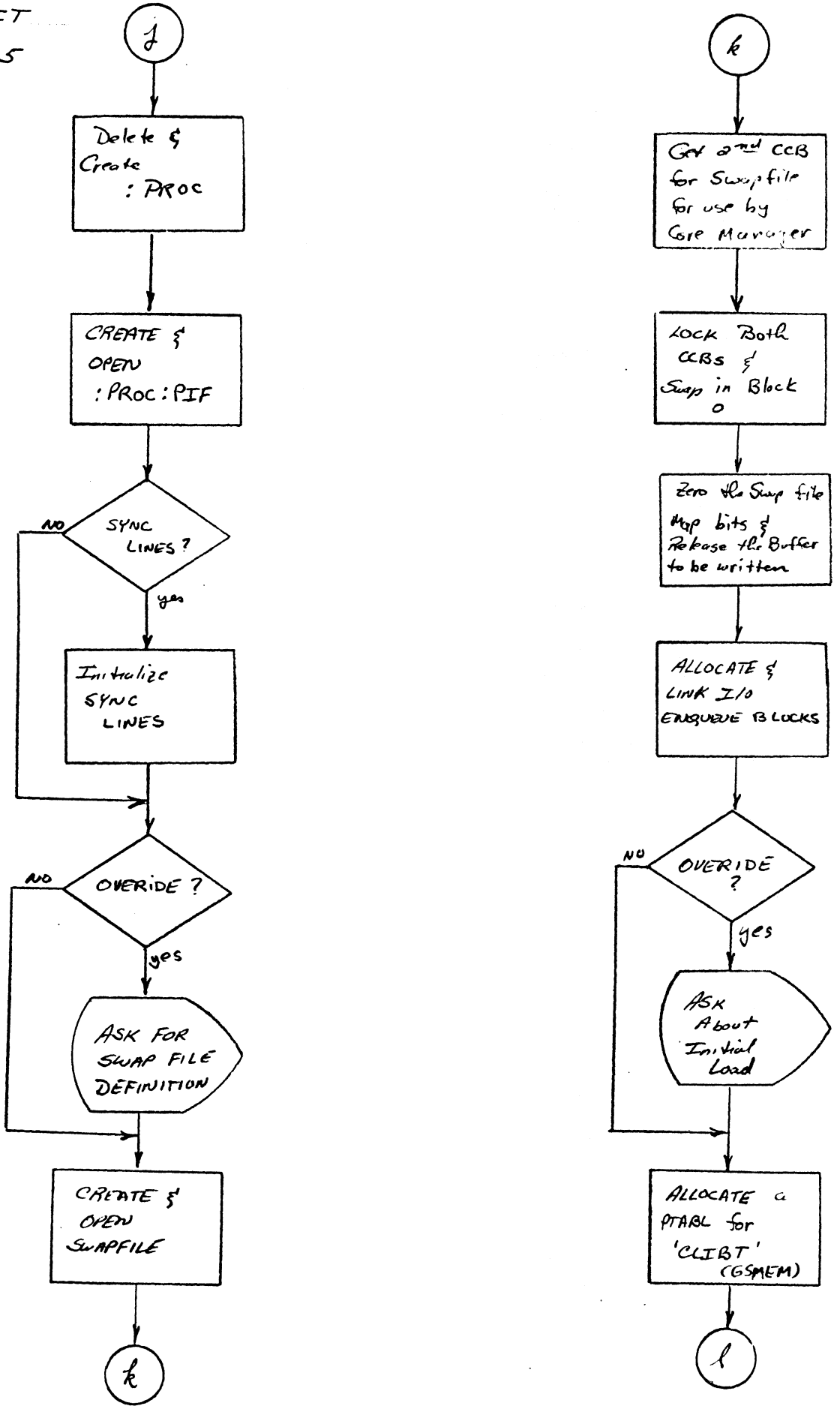

α
Start @2 via Prog. Load
Enter
SAVE Booting Disk Code/Unit#
BLM Loc. 1-15 to bottom of Physical Page 0
Use the interrupt stack for now
Initialize the System Stack
MAP? 200 OR 230?
200
230/600
Highest Page = 177 Set Dch Map B to ALL BUSY
Highest Page = 577
BLM Map Constants to Page Zero (50)
a
a
Initialize SMAP
200 MAP?
yes
Set Device Protect for All Devices from MAP A
Output Initialization Message & Rev #
Save the last Page of Memory for the Disk
Find True Size of Memory
Set 'AOSCN' to show size of AOS
b

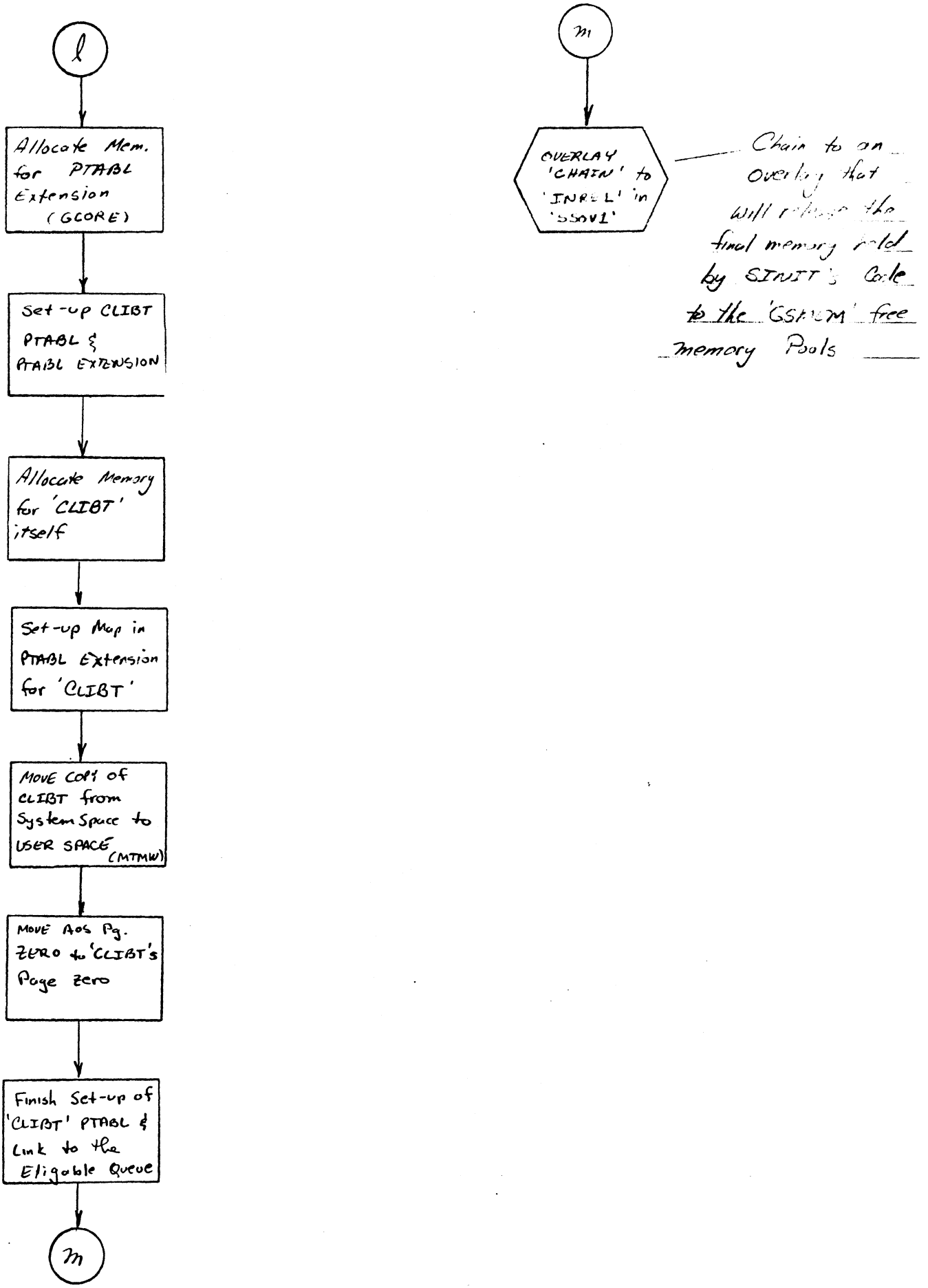
SINIT
Page 2


b
Release All Memory Pages Above system
Memory is Released to the Free memory chain
Release 'CMAP' area not needed to GSMEM Space
Write Protect the OP-System SMON → STKS.
Invalidate & Write Protect Map Slots Above System
Load 'B' Map with 'SMAP' and 'Select'
Demand Paged?
yes
Clear 'Ref' & 'Mod' bits in 'A' Map
C
C
System Debugger?
yes
Set-up Debug Map table
Release Memory from Start of SINIT to here for GSMEM Space
Build a System Overlay Table from 'GSMEM' for System Initialization OVERLAYS
Set-up Dummy Process table for 'SGSUB's' USG
GET TIME & DATE from OPER.
d


d
CALL Overlay to set Date & Time
SET-UP the Interrupt Vector Table & Alloc. DCH MAP Slots
Start-up the Real Time Clock
Set-Up ERCC -- Allow Checking, but no interrupts Now
Initialize the FPU
Allocate 'SCBFM' Buffers from GSMEM Space
ASK for Default Spec Overide
e
e
OVERIDE ?
no
yes
ASK for MASTER LOGICAL DISK
CALL 'MLDUI' to Init. MASTER DISK
Give Disk Addr. of Overlays to the Overlay Manager
Boot Possible ?
no
yes
A Boot is not permitted if the current image was brought in by the installed system
overide ?
no
yes
h
ASK for System Pathname
f
g

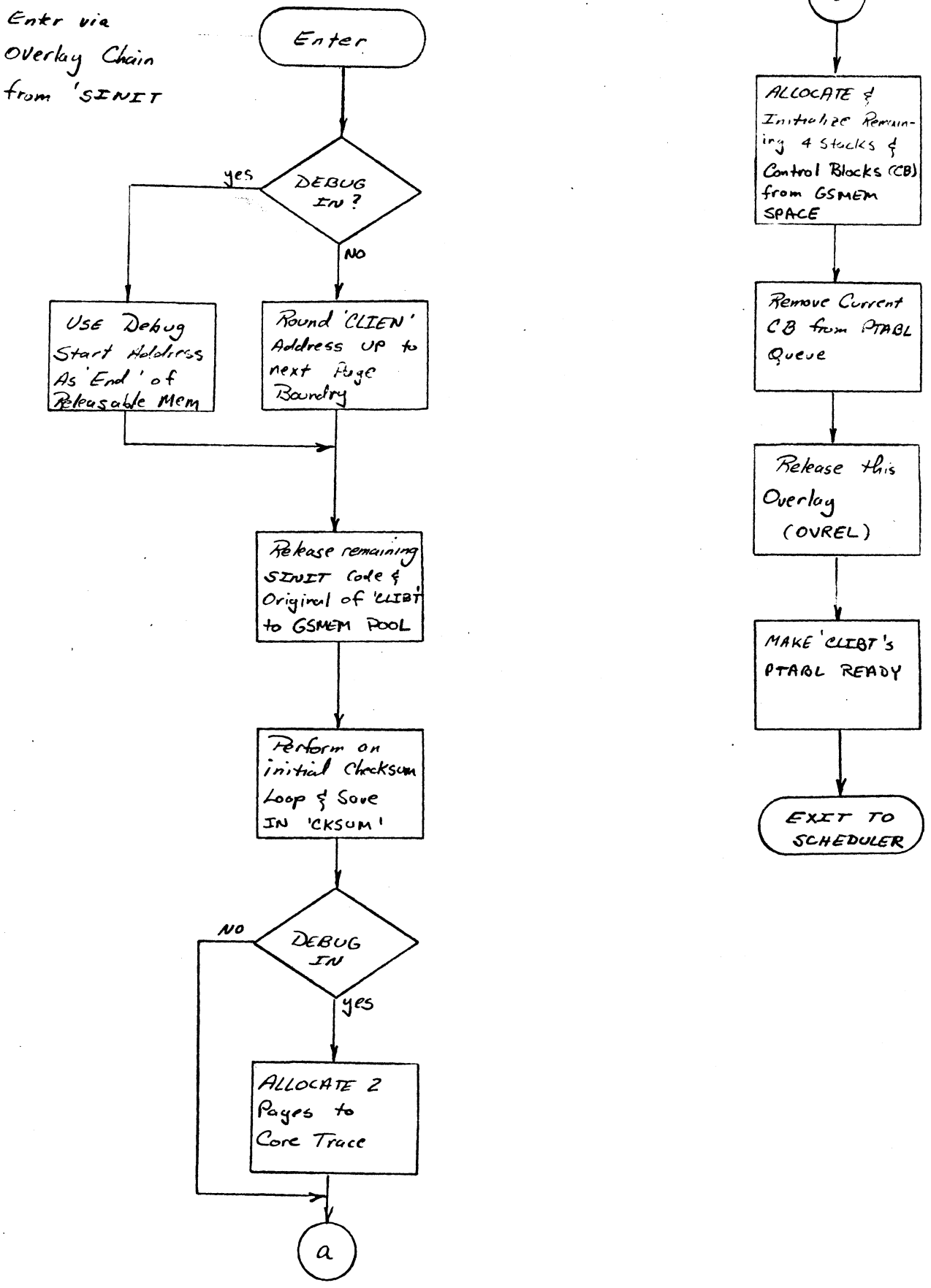

g
Return = CR ?
NO
CALL 'IBOOT' Overlay to Load System
yes
Normal Return
NO
d
h
f
Release all init code to here to GSMEM POOL
yes
SLM Gen'd ?
# of Res. Overlays = 'SCNOR'
# of Res. Overlays = 'SCNOR' - 2
i
i
Allocate Mem. for Res. Ovlys from Xtd. Mem.
Read in Res. Overlays
Delete & Create :PER
Create Generic Filenames & Unit entrys in :PER
NO
SYNC LINES ?
yes
ALLOCATE LINE Tables & Buffers -- INIT DCTs
Enqueue Disk & Mag Tape Time-out Blks.
j

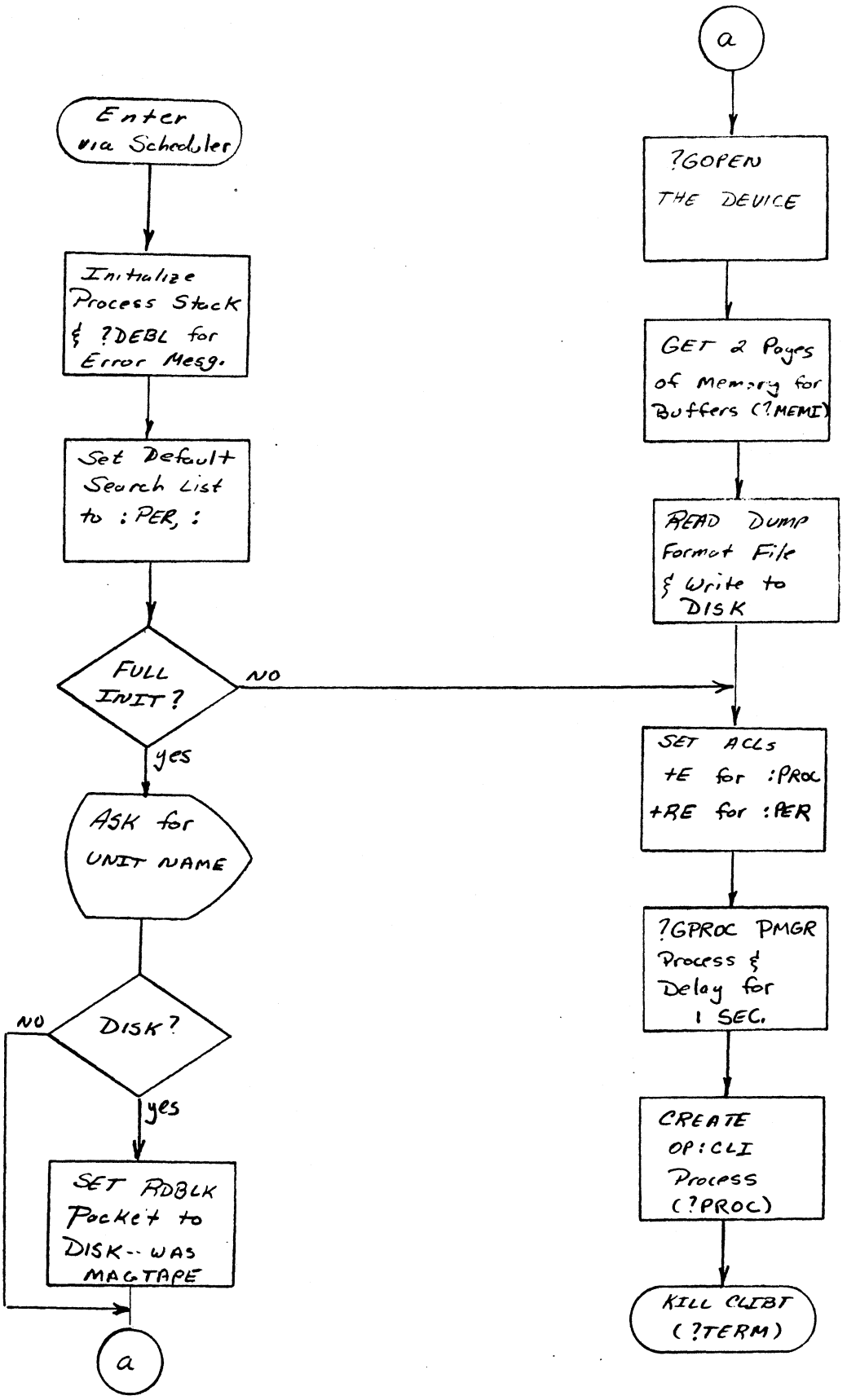

j
Delete &
Create
: PROC
CREATE &
OPEN
: PROC : PIF
NO
SYNC
LINES ?
yes
Initialize
SYNC
LINES
NO
OVERIDE ?
yes
ASK FOR
SWAP FILE
DEFINITION
CREATE &
OPEN
SWAPFILE
k
k
Get 2nd CCB
for Swap file
for use by
Core Manager
LOCK Both
CCBs &
Swap in Block
0
Zero the Swap file
Map bits &
Release the Buffer
to be written
ALLOCATE &
LINK I/O
ENQUEUE BLOCKS
NO
OVERIDE
?
yes
ASK
About
Initial
Load
ALLOCATE a
PTABL for
'CLIBT'
(GSMEM)
l


l
Allocate Mem. for PTABL Extension (GCORE)
Set-up CLIBT PTABL & PTABL EXTENSION
Allocate Memory for 'CLIBT' itself
Set-up Map in PTABL Extension for 'CLIBT'
MOVE COPY of CLIBT from System Space to USER SPACE (MTMW)
MOVE Aos Pg. ZERO to 'CLIBT's Page Zero
Finish Set-up of 'CLIBT' PTABL & Link to the Eligable Queue
m
m
OVERLAY 'CHAIN' to 'INREL' in 'SSMV1'
Chain to an Overlay that will release the final memory held by SINIT's Code to the 'GSTORM' free memory Pools

INREL

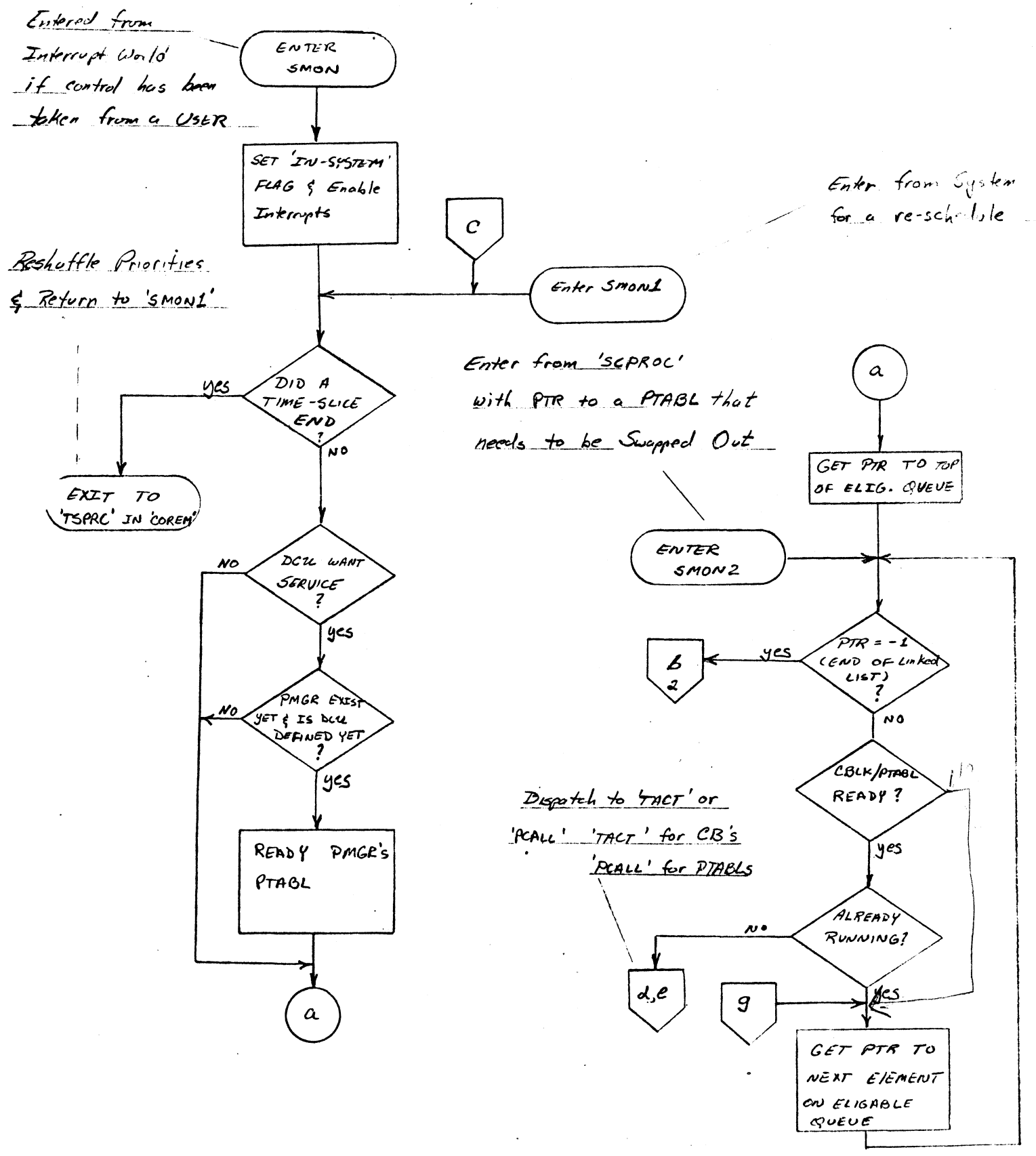
Page 1

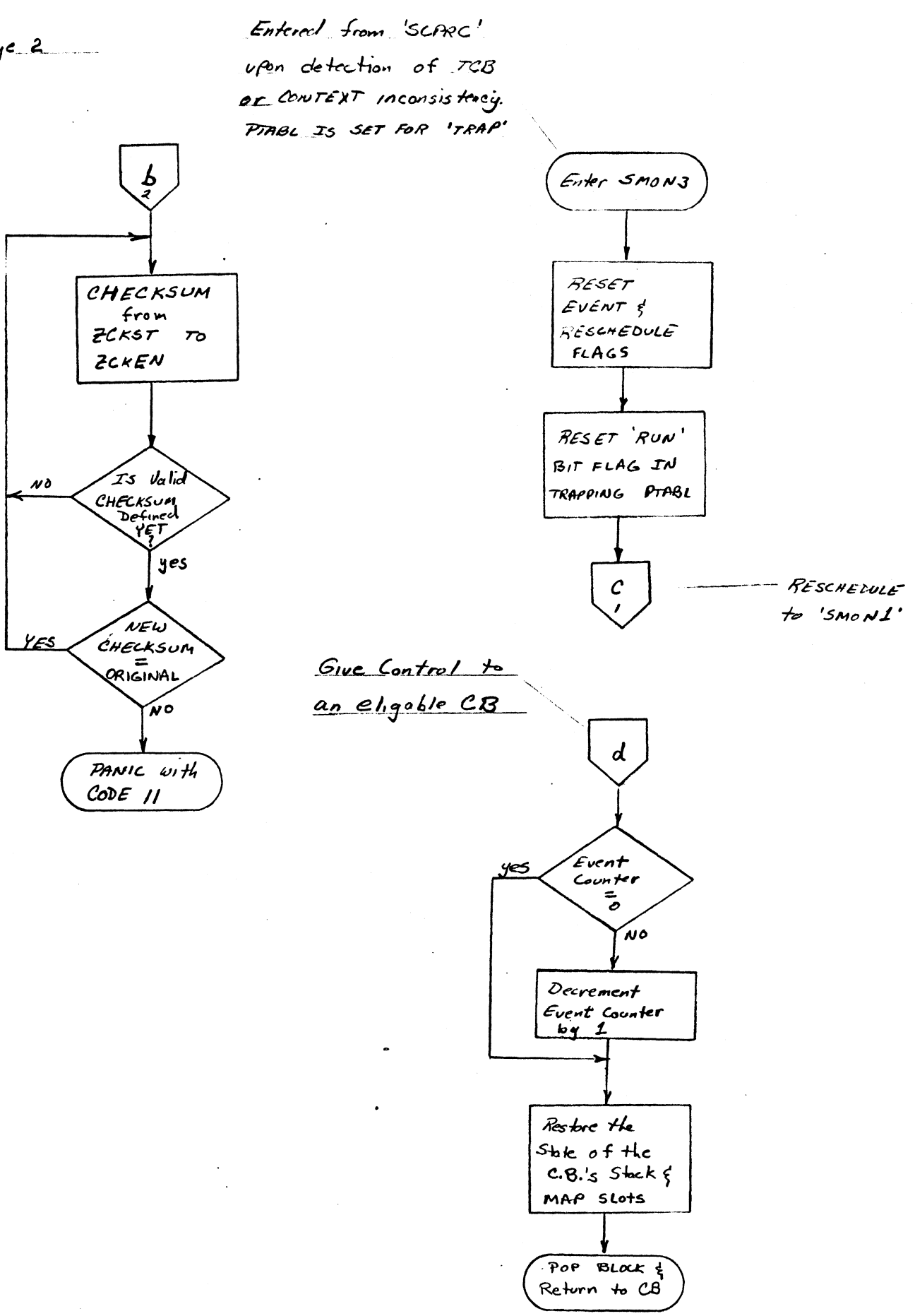

Entr via Overlay Chain from 'SINIT
Enter
DEBUG IN ?
yes
NO
USE Debug Start Address As 'End' of Releasable Mem
Round 'CLIEN' Address UP to next Page Boundry
Release remaining SINIT Code & Original of 'CLIBT' to GSMEM POOL
Perform on initial Checksum Loop & Save IN 'CKSUM'
DEBUG IN
NO
yes
ALLOCATE 2 Pages to Core Trace
a
a
ALLOCATE & Initialize Remaining 4 Stacks & Control Blocks (CB) from GSMEM SPACE
Remove Current CB from PTABL Queue
Release this Overlay (OVREL)
MAKE 'CLIBT's PTABL READY
EXIT TO SCHEDULER


Enter via Scheduler
Initialize Process Stack & ?DEBL for Error Mesg.
Set Default Search List to :PER,:
FULL INIT?
NO
yes
ASK for UNIT NAME
DISK?
NO
yes
SET RDBLK Packet to DISK -- WAS MAGTAPE
a
a
?GOPEN THE DEVICE
GET 2 Pages of Memory for Buffers (?MEMI)
READ Dump Format File & Write to DISK
SET ACLs +E for :PROG +RE for :PER
?GPROC PMGR Process & Delay for 1 SEC.
CREATE OP:CLI Process (?PROC)
KILL CLIBT (?TERM)

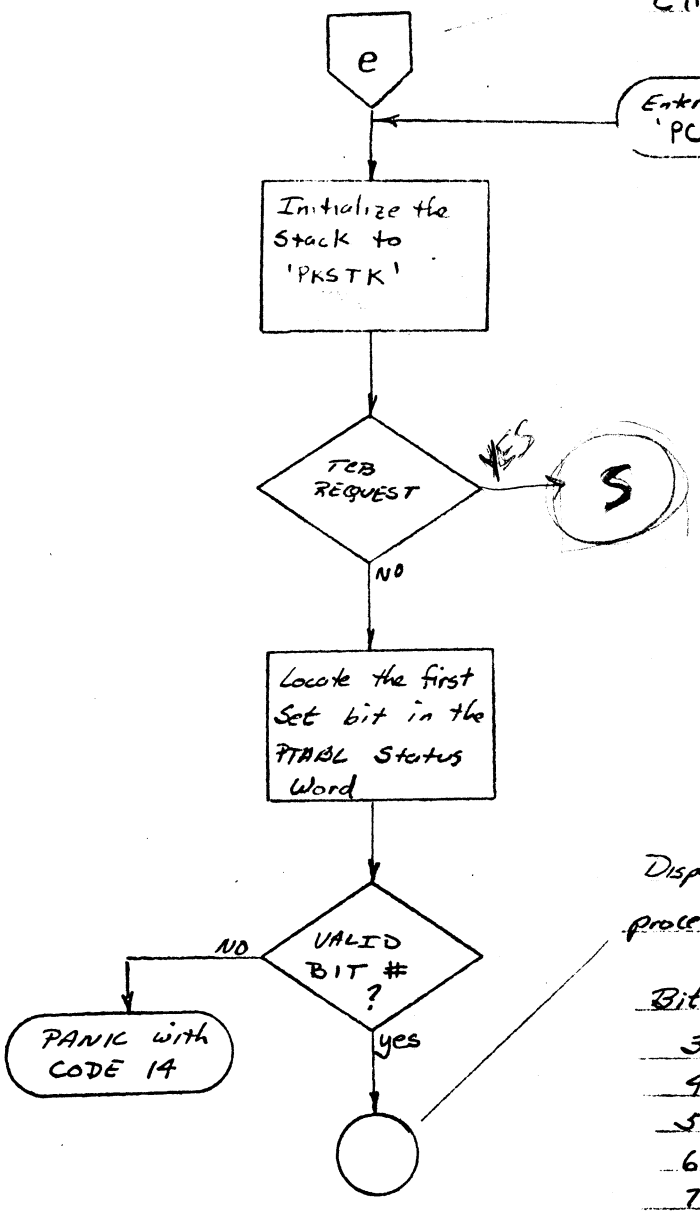
SCHED

- Processor Scheduler -



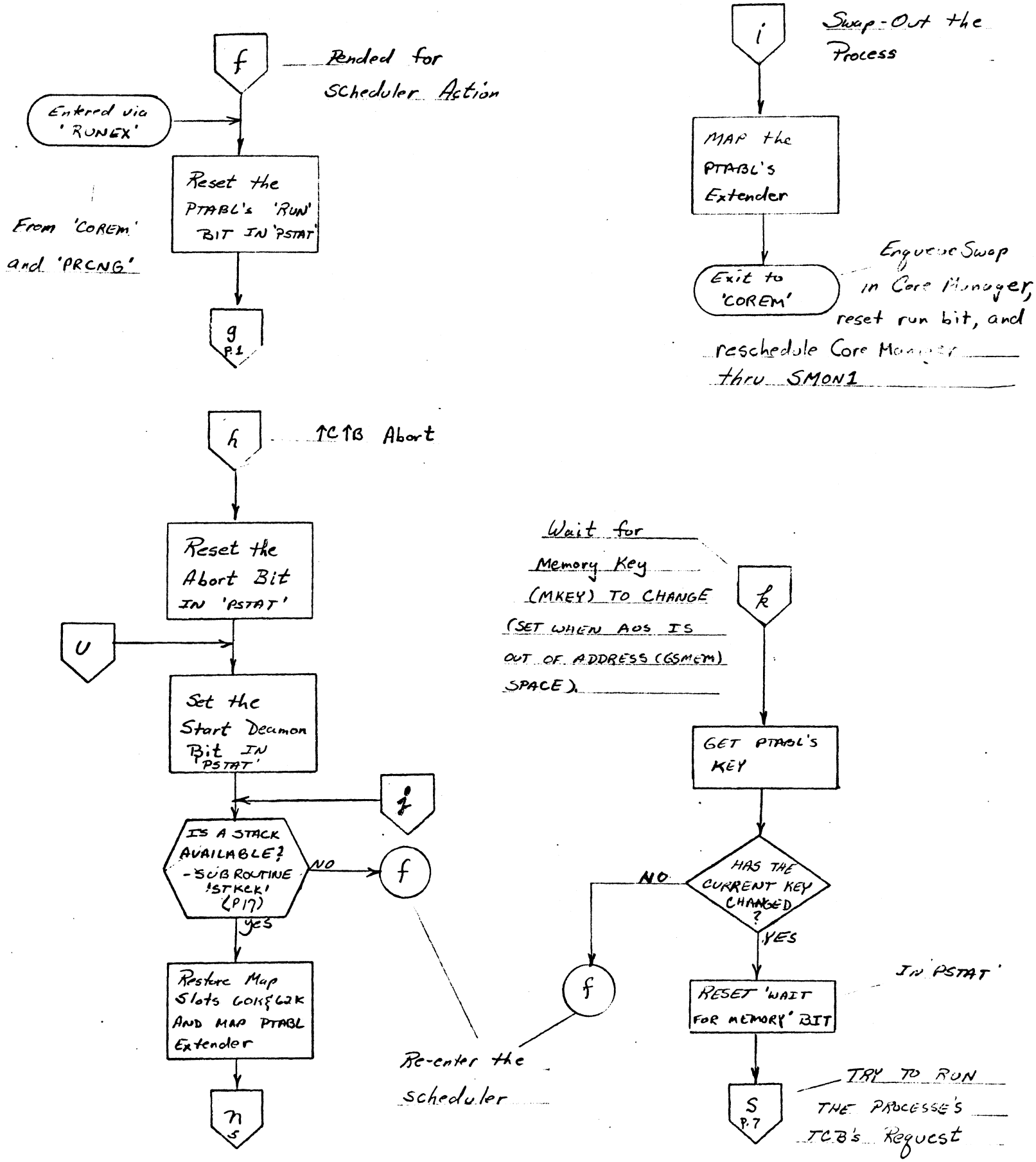

Entered from 'SCARC' upon detection of TCB or CONTEXT inconsistency. PTABL IS SET FOR 'TRAP'
b 2
CHECKSUM from ZCKST TO ZCKEN
Is Valid CHECKSUM Defined YET?
NO
yes
NEW CHECKSUM = ORIGINAL
YES
NO
PANIC with CODE 11
Enter SMON3
RESET EVENT & RESCHEDULE FLAGS
RESET 'RUN' BIT FLAG IN TRAPPING PTABL
C 1
RESCHEDULE to 'SMON1'
Give Control to an eligable CB
d
Event Counter = 0
yes
NO
Decrement Event Counter by 1
Restore the State of the C.B.'s Stack & MAP SLOTS
POP BLOCK & Return to CB

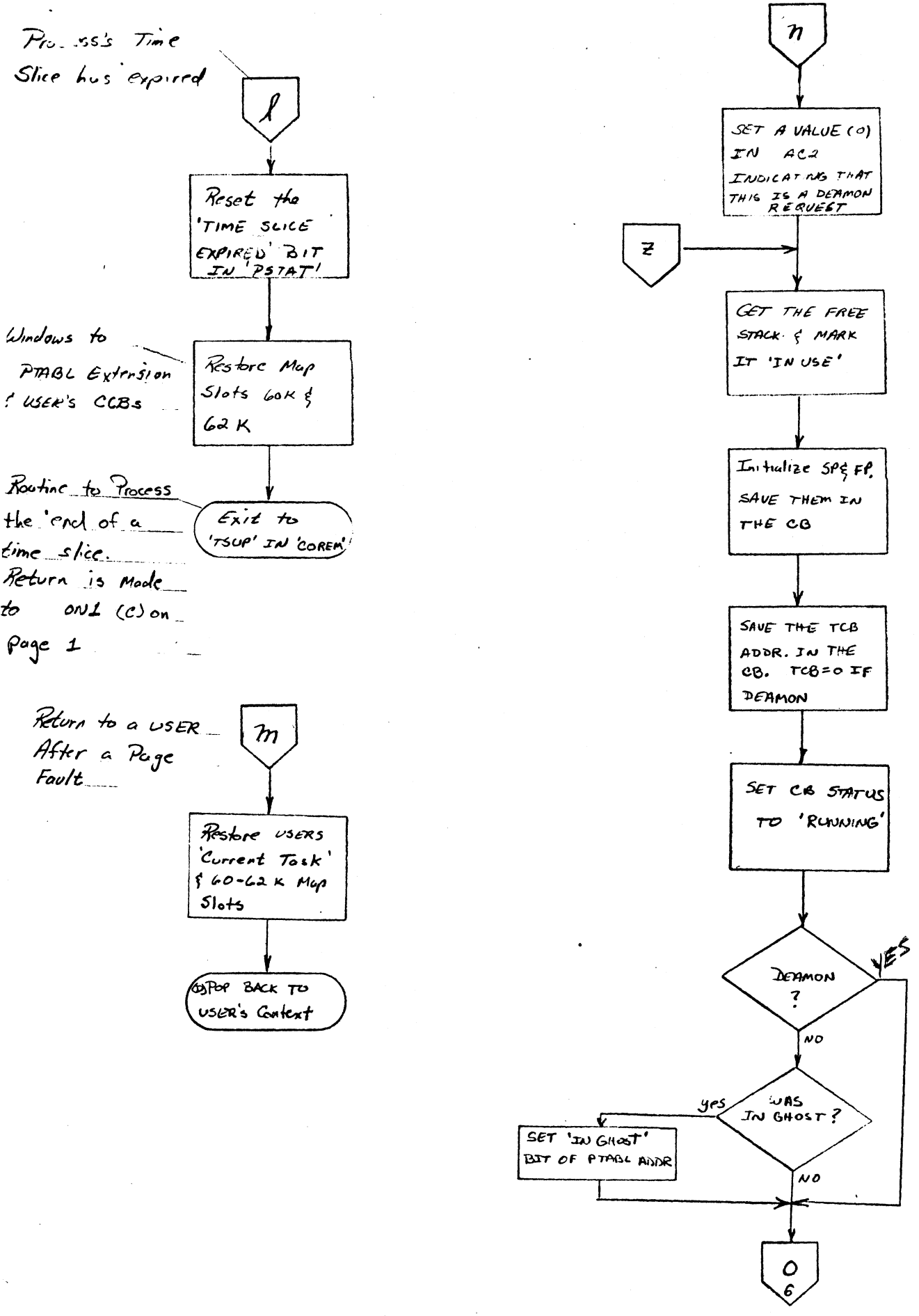
Give control to an eligable PTABL

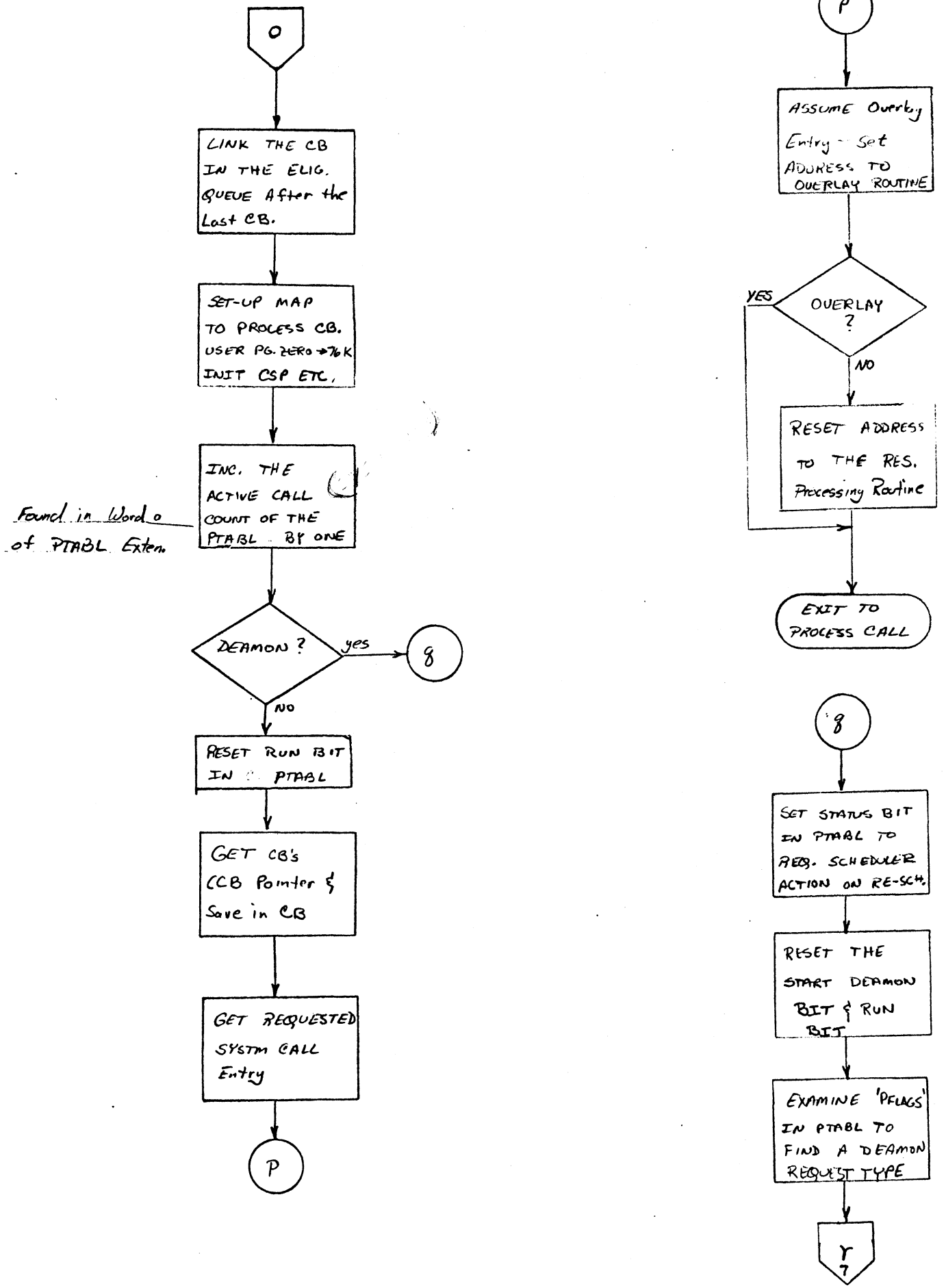


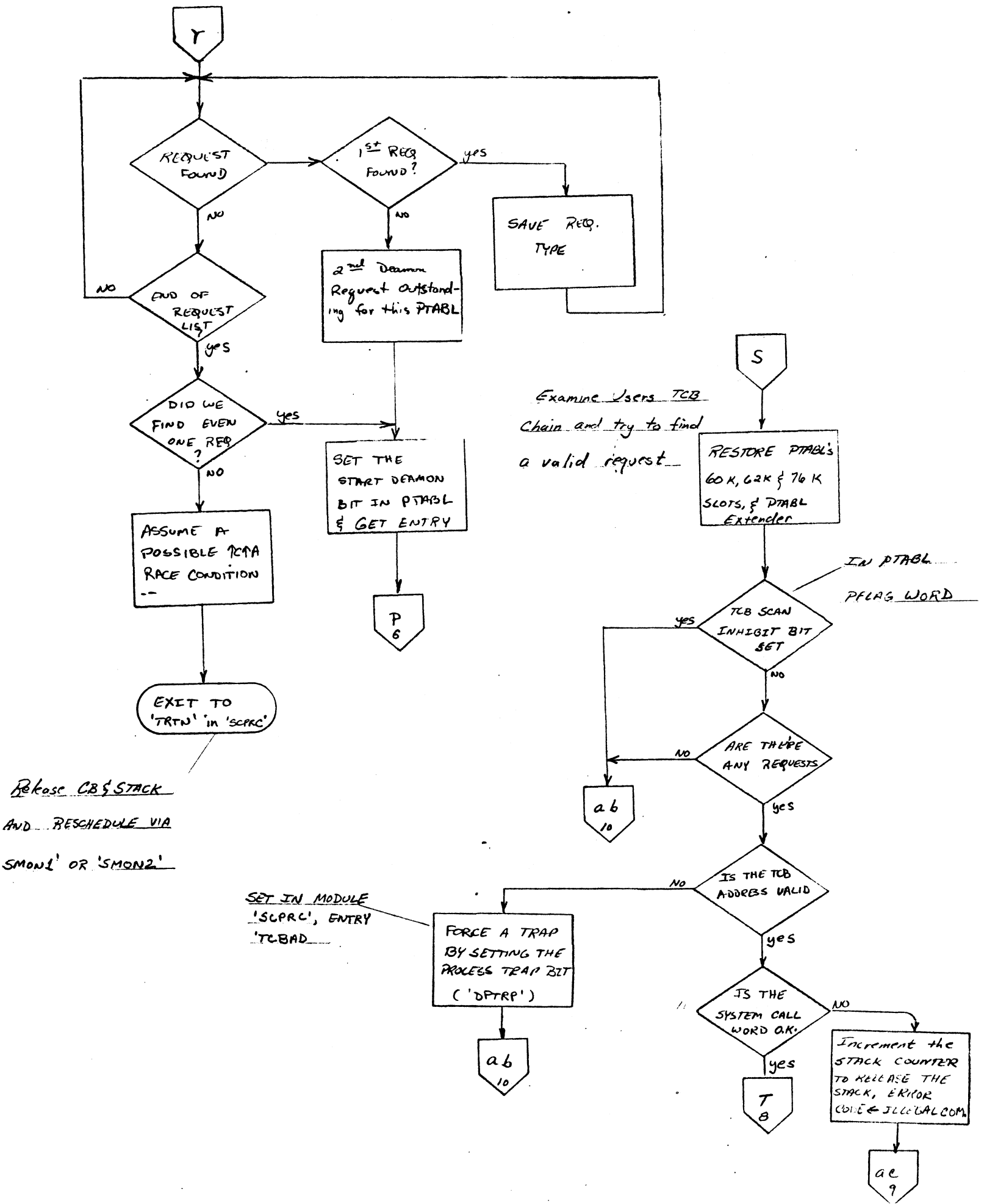
Dispatch to Status processing routine

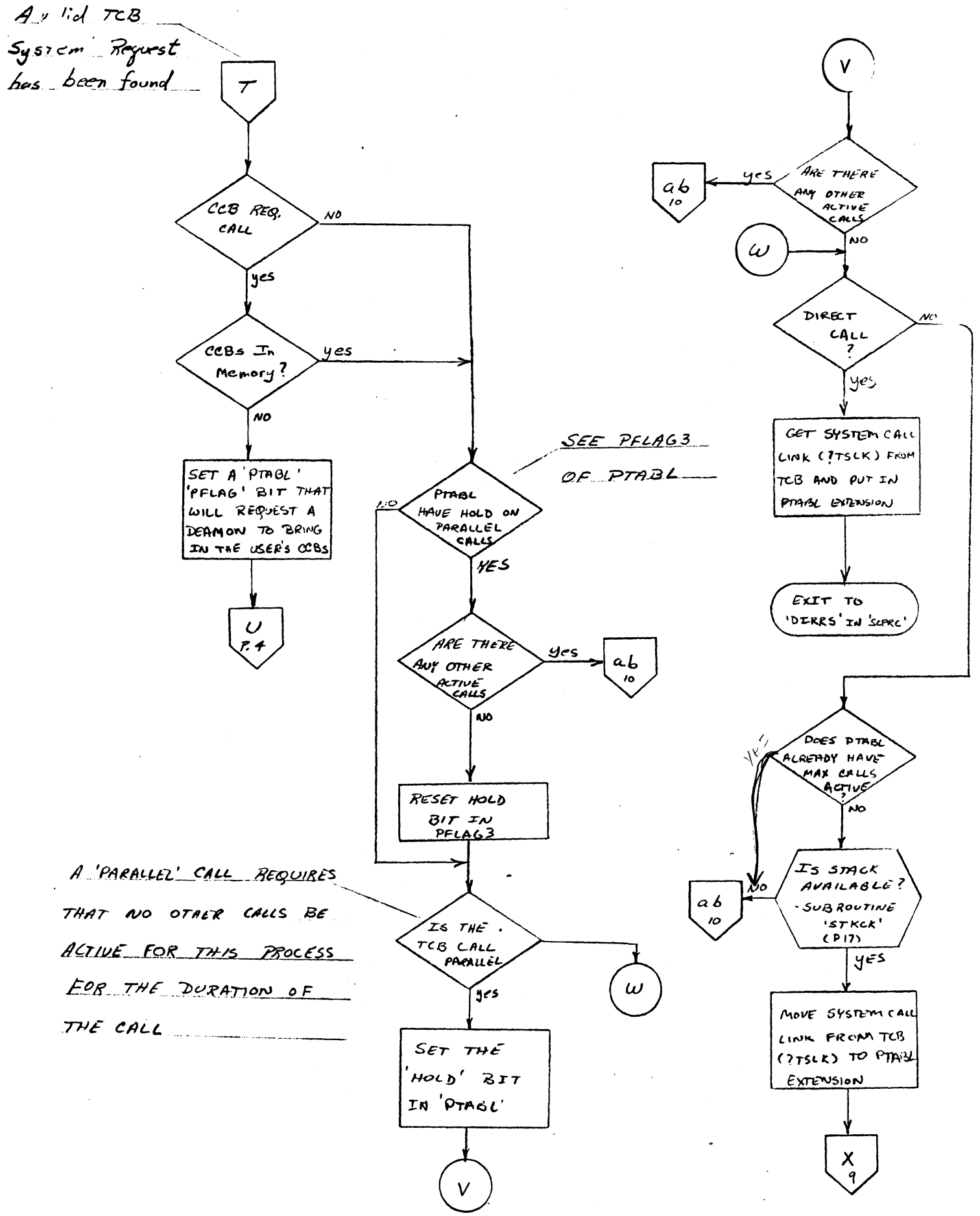
Bit Pos.	Entry
3	f
4	h
5	f
6	i
7	f
8	j
9	k
10	l
11	m (Demand Paged Systems Only

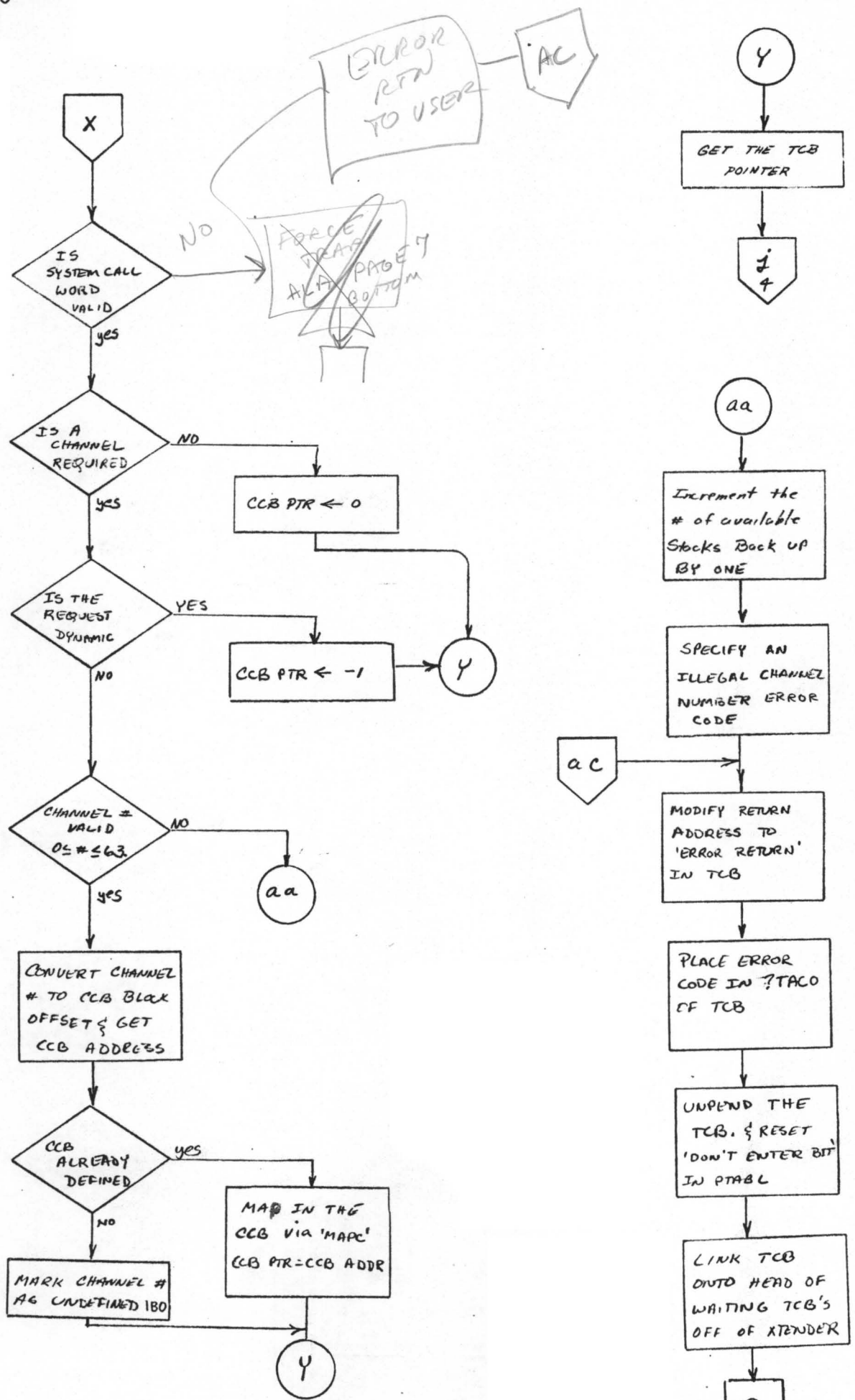

f
Needed for Scheduler Action
Entered via 'RUNEX'
From 'COREM' and 'PRCNG'
Reset the PTABL's 'RUN' BIT IN 'PSTAT'
g P.1
i
Swap-Out the Process
MAP the PTABL's Extender
Exit to 'COREM'
Enqueue Swap in Core Manager, reset run bit, and reschedule Core Manager thru SMON1
h
TCTB Abort
Reset the Abort Bit IN 'PSTAT'
U
Set the Start Deamon Bit IN 'PSTAT'
j
IS A STACK AVAILABLE? - SUBROUTINE 'STKCK' (P17)
NO
yes
f
Restore Map Slots GOKEQK AND MAP PTABL Extender
n S
Re-enter the scheduler
Wait for Memory Key (MKEY) TO CHANGE (SET WHEN AOS IS OUT OF ADDRESS (GSMEM) SPACE).
k
GET PTABL'S KEY
HAS THE CURRENT KEY CHANGED ?
NO
YES
f
RESET 'WAIT FOR MEMORY' BIT IN 'PSTAT'
S P.7
TRY TO RUN THE PROCESSE'S TCB's Request

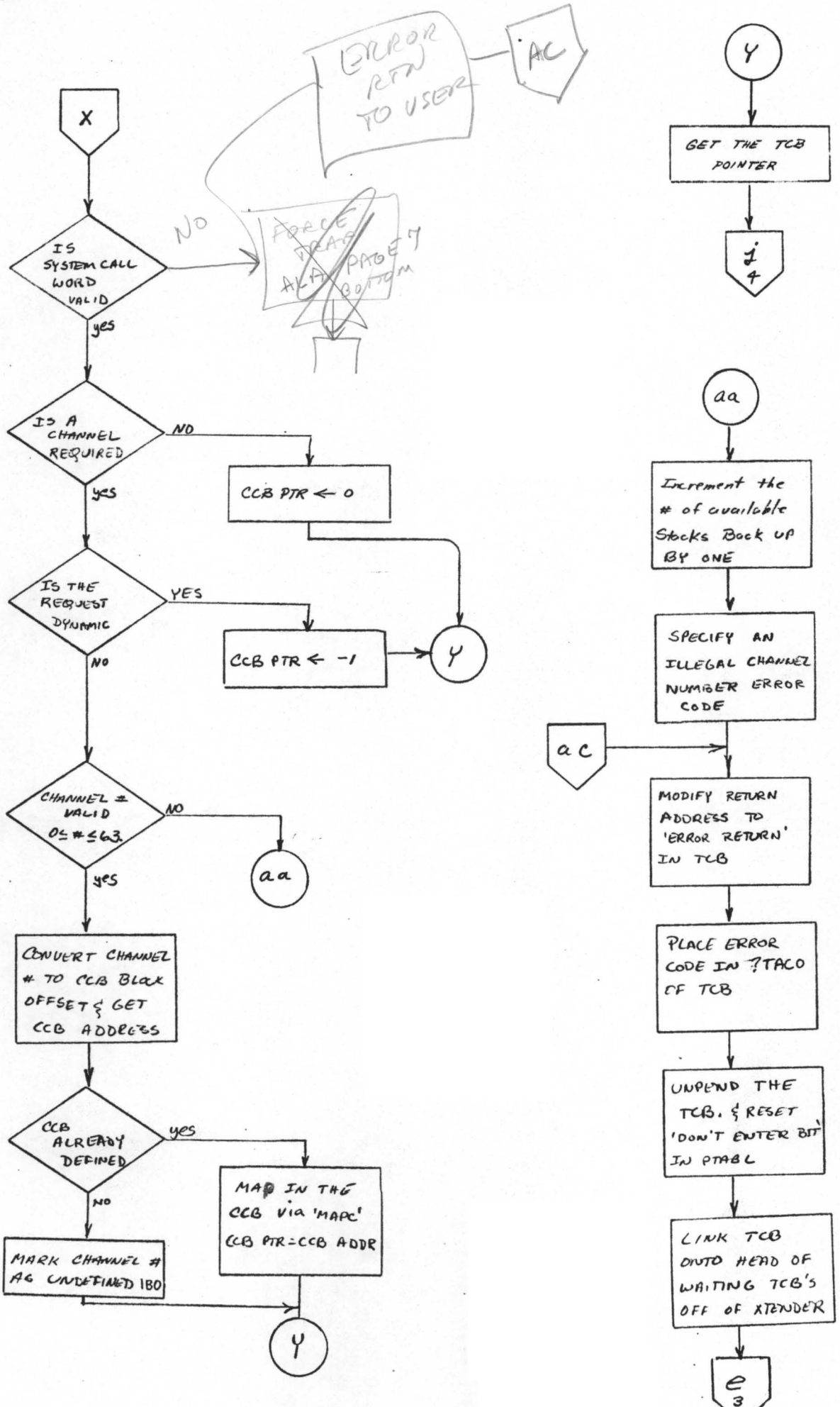

Process's Time Slice has expired
l
Reset the 'TIME SLICE EXPIRED' BIT IN 'PSTAT'
Windows to PTABL Extension & USER's CCBs
Restore Map Slots 60K & 62K
Routine to Process the 'end of a time slice. Return is Mode to ON1 (C) on Page 1
Exit to 'TSUP' IN 'COREM'
Return to a USER After a Page Fault
m
Restore USERS 'Current Task' & 60-62K Map Slots
POP BACK TO USER's Context
n
SET A VALUE (0) IN AC2 INDICATING THAT THIS IS A DEAMON REQUEST
z
GET THE FREE STACK & MARK IT 'IN USE'
Initialize SP & FP. SAVE THEM IN THE CB
SAVE THE TCB ADDR. IN THE CB. TCB=0 IF DEAMON
SET CB STATUS TO 'RUNNING'
DEAMON ?
YES
NO
WAS IN GHOST ?
yes
SET 'IN GHOST' BIT OF PTABL ADDR
NO
O 6

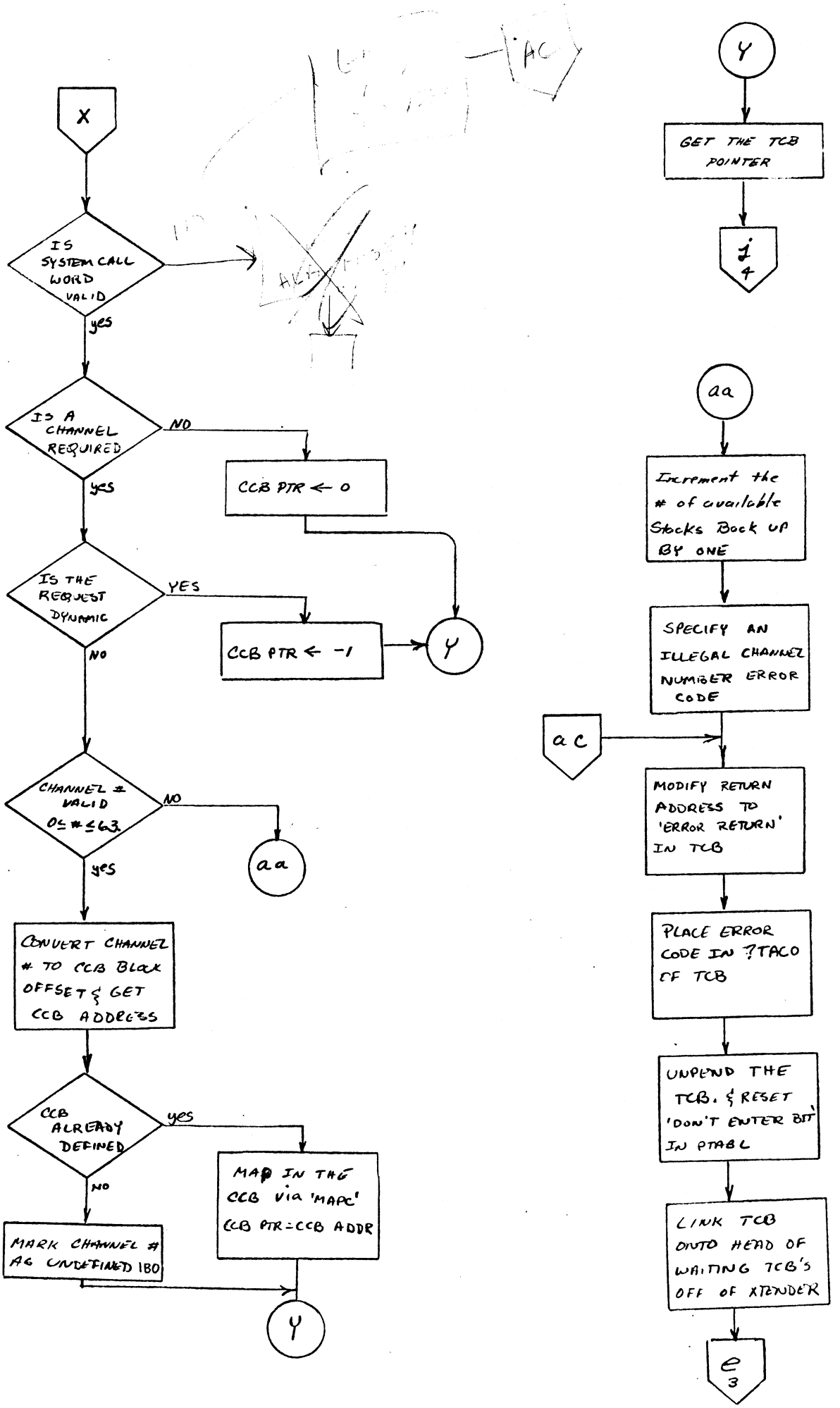

O
LINK THE CB IN THE ELIG. QUEUE After the Last CB.
SET-UP MAP TO PROCESS CB. USER PG. ZERO → 76K INIT CSP ETC,
INC. THE ACTIVE CALL COUNT OF THE PTABL BY ONE
Found in Word 0 of PTABL Extem.
DEAMON ?
yes
g
NO
RESET RUN BIT IN PTABL
GET CB's CCB Pointer & Save in CB
GET REQUESTED SYSTM CALL Entry
P
P
ASSUME Overlay Entry - Set ADDRESS TO OVERLAY ROUTINE
YES
OVERLAY ?
NO
RESET ADDRESS TO THE RES. Processing Routine
EXIT TO PROCESS CALL
g
SET STATUS BIT IN PTABL TO REQ. SCHEDULER ACTION ON RE-SCH.
RESET THE START DEAMON BIT & RUN BIT
EXAMINE 'FLAGS' IN PTABL TO FIND A DEAMON REQUEST TYPE
Y
7

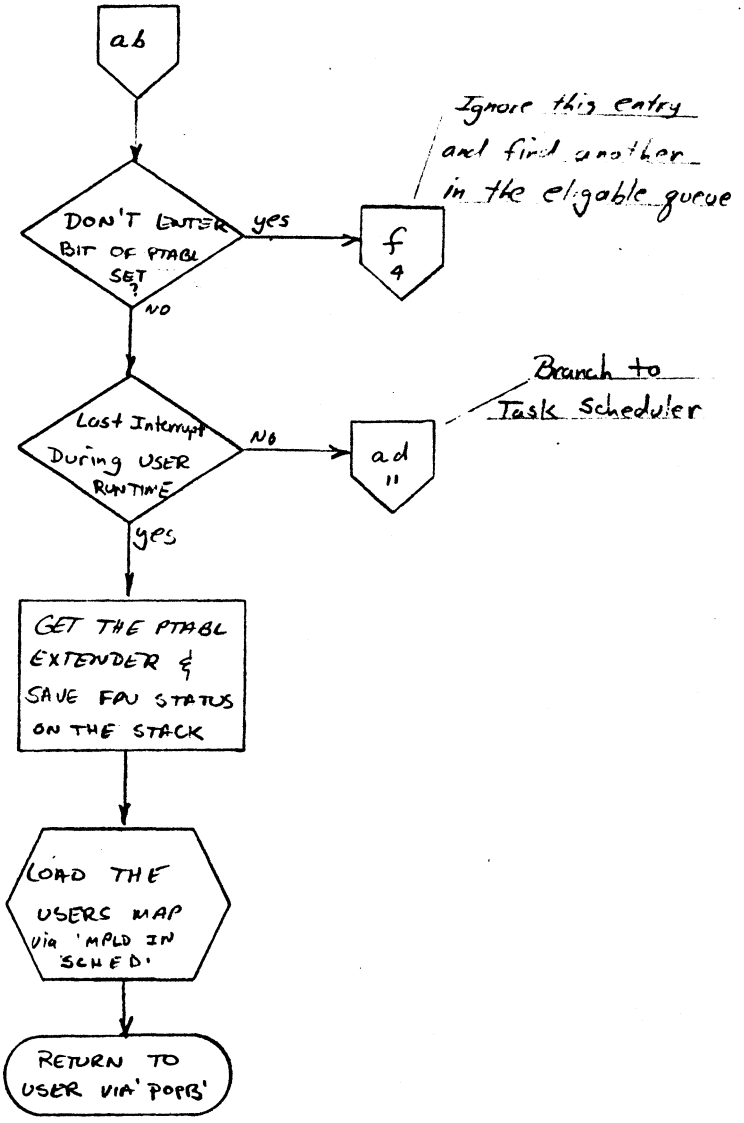

Y
REQUEST FOUND
NO
1ST REQ FOUND?
yes
NO
SAVE REQ. TYPE
2nd Deamon Request outstanding for this PTABL
NO
END OF REQUEST LIST?
yes
DID WE FIND EVEN ONE REQ?
yes
NO
SET THE START DEAMON BIT IN PTABL & GET ENTRY
P 6
ASSUME A POSSIBLE TCMA RACE CONDITION --
EXIT TO 'TRTN' in 'SCPRC'
Release CB & STACK AND RESCHEDULE VIA SMON1' OR 'SMON2'
S
Examine Users TCB Chain and try to find a valid request
RESTORE PTABL's 60K, 62K & 76K SLOTS, & PTABL Extender
IN PTABL PFLAG WORD
TCB SCAN INHIBIT BIT SET
yes
NO
ARE THERE ANY REQUESTS
NO
yes
a b 10
IS THE TCB ADDRESS VALID
NO
yes
SET IN MODULE 'SCPRC', ENTRY 'TCBAD'
FORCE A TRAP BY SETTING THE PROCESS TRAP BIT ('DPTRP')
a b 10
IS THE SYSTEM CALL WORD O.K.
NO
yes
T 8
Increment the STACK COUNTER TO RELEASE THE STACK, ERROR CODE ← ILLEGALCOM
a c 9


A valid TCB System Request has been found
T
CCB REQ. CALL
NO
yes
CCBs In Memory?
yes
NO
SET A 'PTABL' 'PFLAG' BIT THAT WILL REQUEST A DEAMON TO BRING IN THE USER'S CCBs
U P.4
SEE PFLAG3 OF PTABL
PTABL HAVE HOLD ON PARALLEL CALLS
NO
YES
ARE THERE ANY OTHER ACTIVE CALLS
yes
ab 10
NO
RESET HOLD BIT IN PFLAG3
A 'PARALLEL' CALL REQUIRES THAT NO OTHER CALLS BE ACTIVE FOR THIS PROCESS FOR THE DURATION OF THE CALL
IS THE TCB CALL PARALLEL
yes
W
SET THE 'HOLD' BIT IN 'PTABL'
V
V
ARE THERE ANY OTHER ACTIVE CALLS
yes
ab 10
W
NO
DIRECT CALL ?
NO
yes
GET SYSTEM CALL LINK (?TSLK) FROM TCB AND PUT IN PTABL EXTENSION
EXIT TO 'DIRRS' IN 'SCPRC'
DOES PTABL ALREADY HAVE MAX CALLS ACTIVE ?
NO
YES
ab 10
NO
IS STACK AVAILABLE? - SUBROUTINE 'STKCK' (P17)
yes
MOVE SYSTEM CALL LINK FROM TCB (?TSLK) TO PTABL EXTENSION
X 9

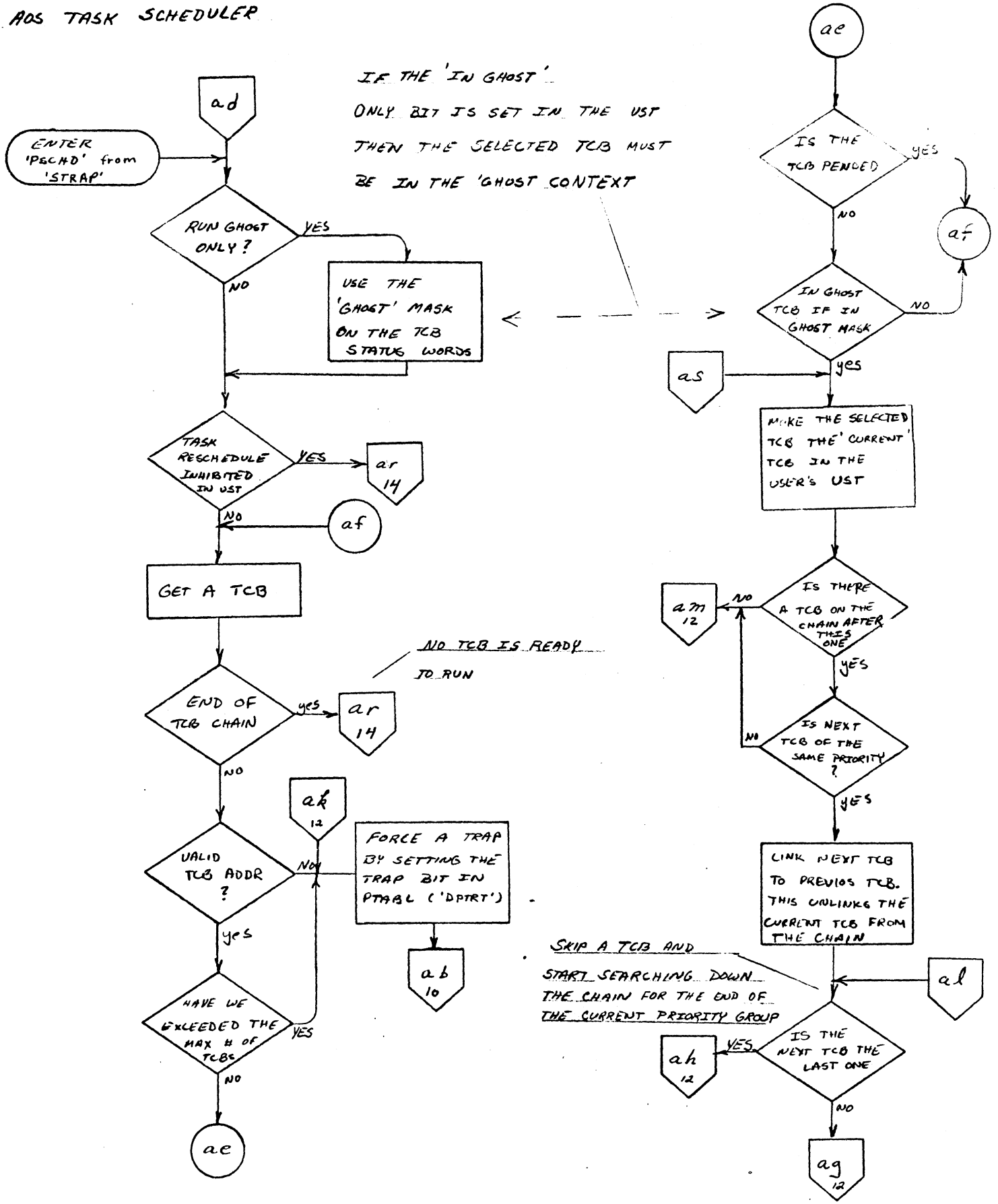

X
IS SYSTEM CALL WORD VALID
NO
ERROR RTN TO USER
AC
yes
IS A CHANNEL REQUIRED
NO
CCB PTR ← 0
yes
IS THE REQUEST DYNAMIC
YES
CCB PTR ← -1
Y
NO
CHANNEL # VALID 0≤#≤63
NO
aa
yes
CONVERT CHANNEL # TO CCB BLOCK OFFSET & GET CCB ADDRESS
CCB ALREADY DEFINED
yes
MAP IN THE CCB via 'MAPC' CCB PTR=CCB ADDR
NO
MARK CHANNEL # AS UNDEFINED IBO
Y
Y
GET THE TCB POINTER
j 4
aa
Increment the # of available Stacks Back UP BY ONE
SPECIFY AN ILLEGAL CHANNEL NUMBER ERROR CODE
ac
MODIFY RETURN ADDRESS TO 'ERROR RETURN' IN TCB
PLACE ERROR CODE IN ?TACO OF TCB
UNPEND THE TCB. & RESET 'DON'T ENTER BIT' IN PTABL
LINK TCB ONTO HEAD OF WAITING TCB'S OFF OF XTENDER

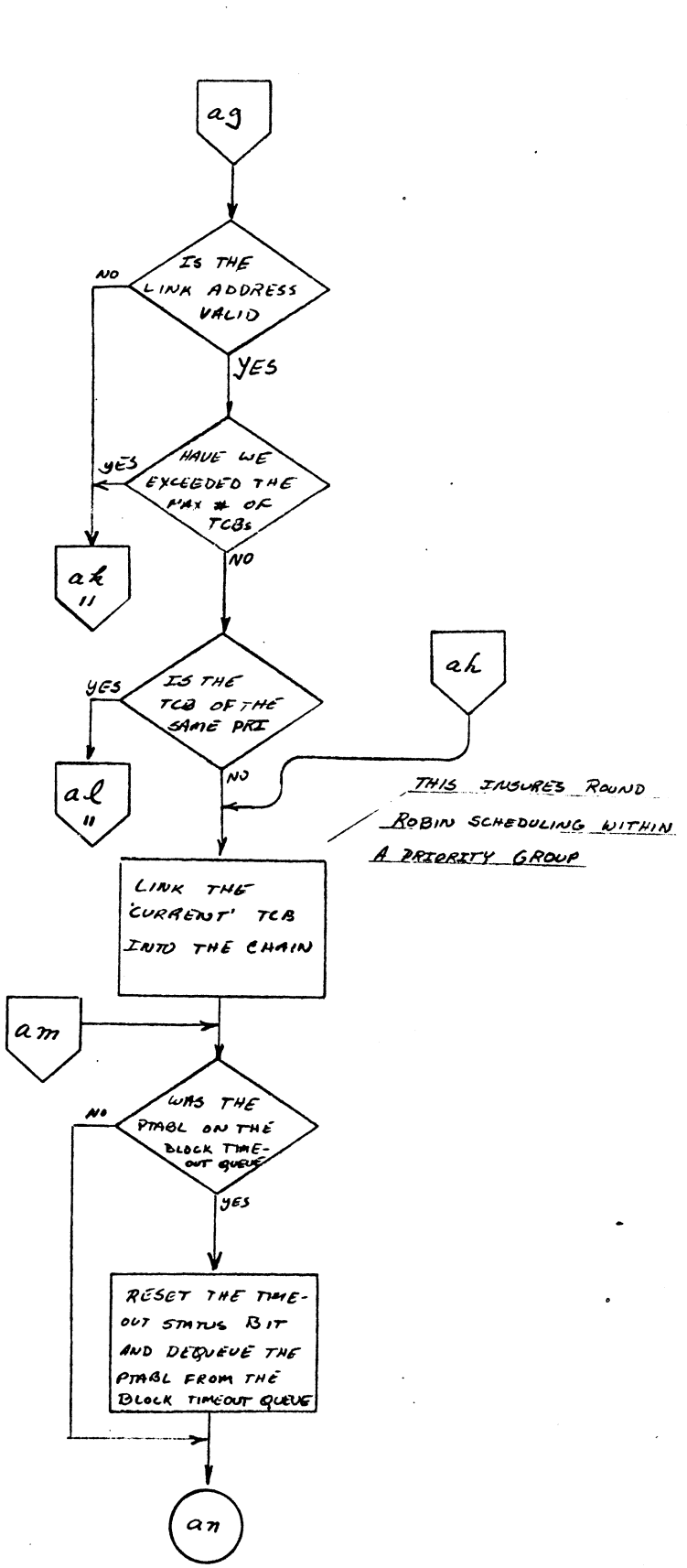

X
IS SYSTEM CALL WORD VALID
NO
ERROR RTN TO USER
AC
FORCE TRAP ALA PAGE 7 BOTTOM
yes
IS A CHANNEL REQUIRED
NO
CCB PTR ← 0
yes
IS THE REQUEST DYNAMIC
YES
CCB PTR ← -1
Y
NO
CHANNEL # VALID 0≤#≤63.
NO
aa
yes
CONVERT CHANNEL # TO CCB BLOCK OFFSET & GET CCB ADDRESS
CCB ALREADY DEFINED
yes
MAP IN THE CCB via 'MAPC' CCB PTR=CCB ADDR
NO
MARK CHANNEL # AS UNDEFINED IBO
Y
Y
GET THE TCB POINTER
j 4
aa
Increment the # of available Stacks Back UP BY ONE
SPECIFY AN ILLEGAL CHANNEL NUMBER ERROR CODE
ac
MODIFY RETURN ADDRESS TO 'ERROR RETURN' IN TCB
PLACE ERROR CODE IN ?TACO OF TCB
UNPEND THE TCB. & RESET 'DON'T ENTER BIT' IN PTABL
LINK TCB ONTO HEAD OF WAITING TCB'S OFF OF XTENDER
e 3

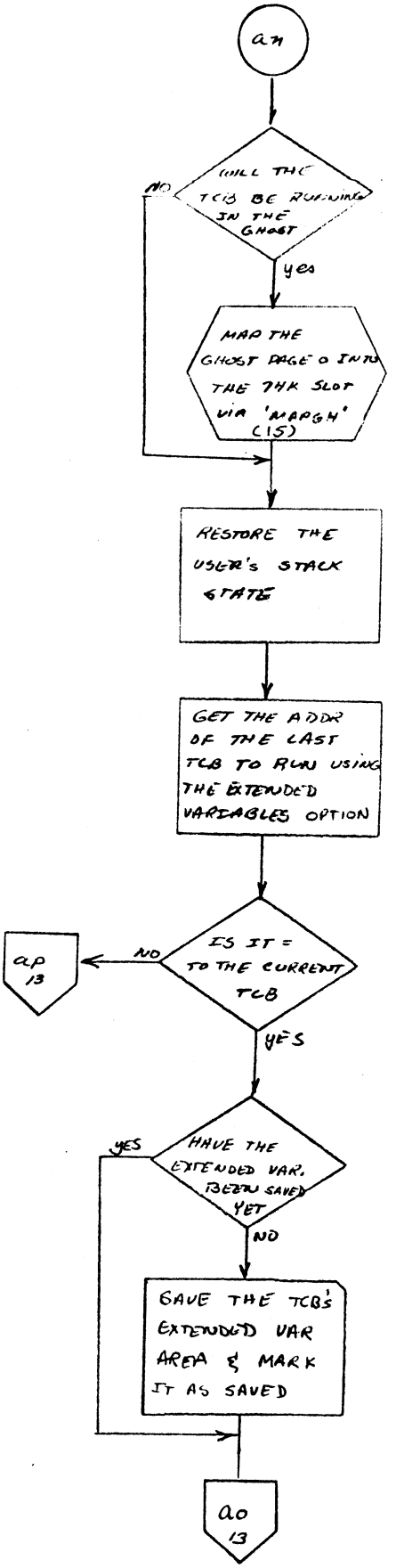

X
IS SYSTEM CALL WORD VALID
yes
IS A CHANNEL REQUIRED
NO
CCB PTR ← 0
yes
IS THE REQUEST DYNAMIC
YES
CCB PTR ← -1
Y
NO
CHANNEL # VALID 0≤#≤63
NO
aa
yes
CONVERT CHANNEL # TO CCB BLOCK OFFSET & GET CCB ADDRESS
CCB ALREADY DEFINED
yes
MAP IN THE CCB via 'MAPC' CCB PTR=CCB ADDR
NO
MARK CHANNEL # AS UNDEFINED IBO
Y
Y
GET THE TCB POINTER
j 4
aa
Increment the # of available Stacks Back UP BY ONE
SPECIFY AN ILLEGAL CHANNEL NUMBER ERROR CODE
ac
MODIFY RETURN ADDRESS TO 'ERROR RETURN' IN TCB
PLACE ERROR CODE IN ?TACO OF TCB
UNPEND THE TCB. & RESET 'DON'T ENTER BIT' IN PTABL
LINK TCB ONTO HEAD OF WAITING TCB'S OFF OF XTENDER
e 3

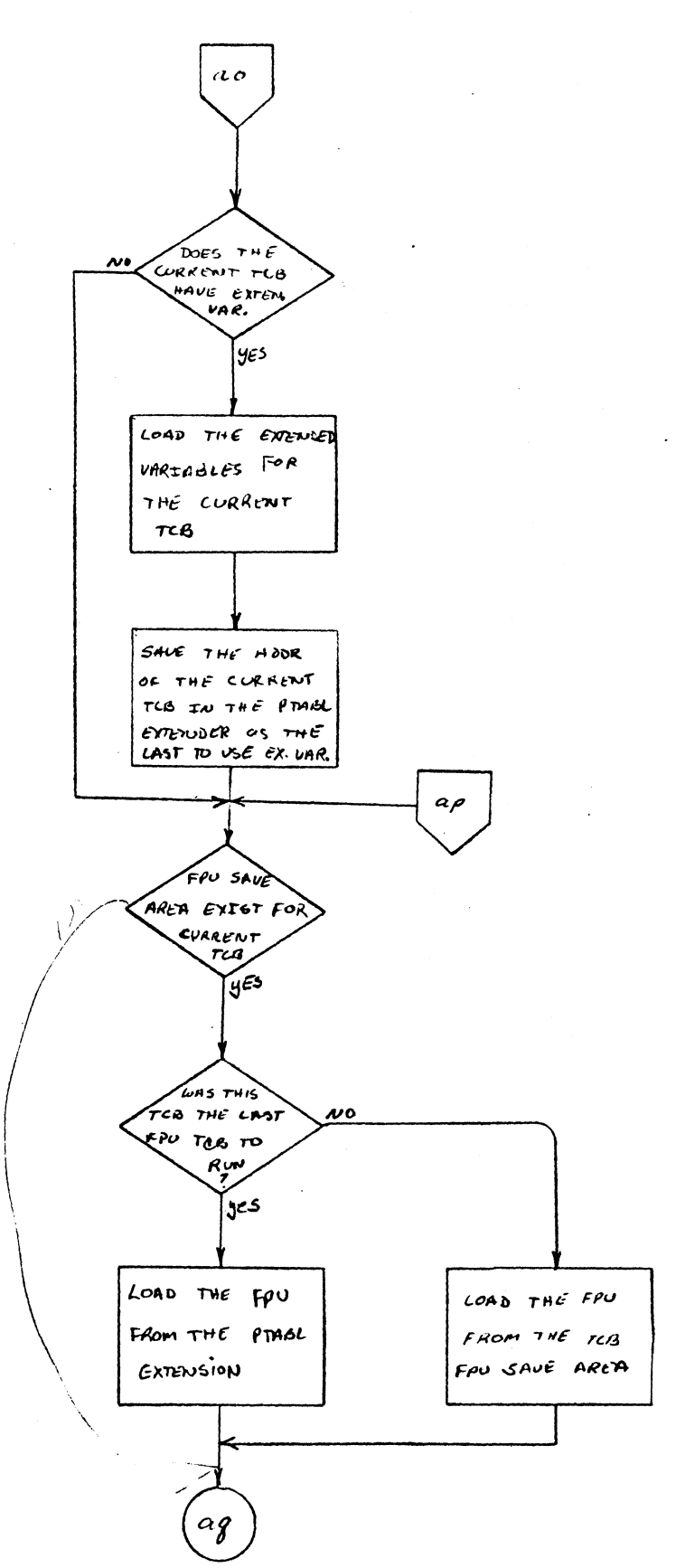

ab
DON'T ENTER BIT OF PTABL SET ?
yes
f
4
Ignore this entry and find another in the eligable queue
NO
Last Interrupt During USER RUNTIME
No
ad
11
Branch to Task Scheduler
yes
GET THE PTABL EXTENDER & SAVE FAU STATUS ON THE STACK
LOAD THE USERS MAP via 'MPLD IN SCHED'
RETURN TO USER VIA 'POPB'

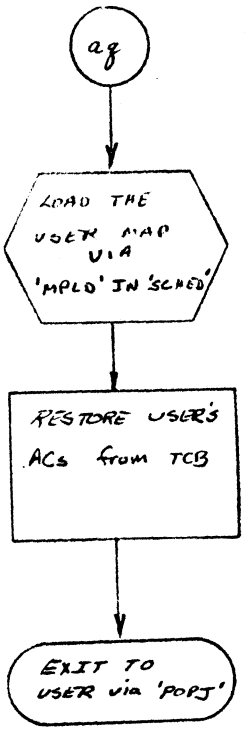
AOS TASK SCHEDULER

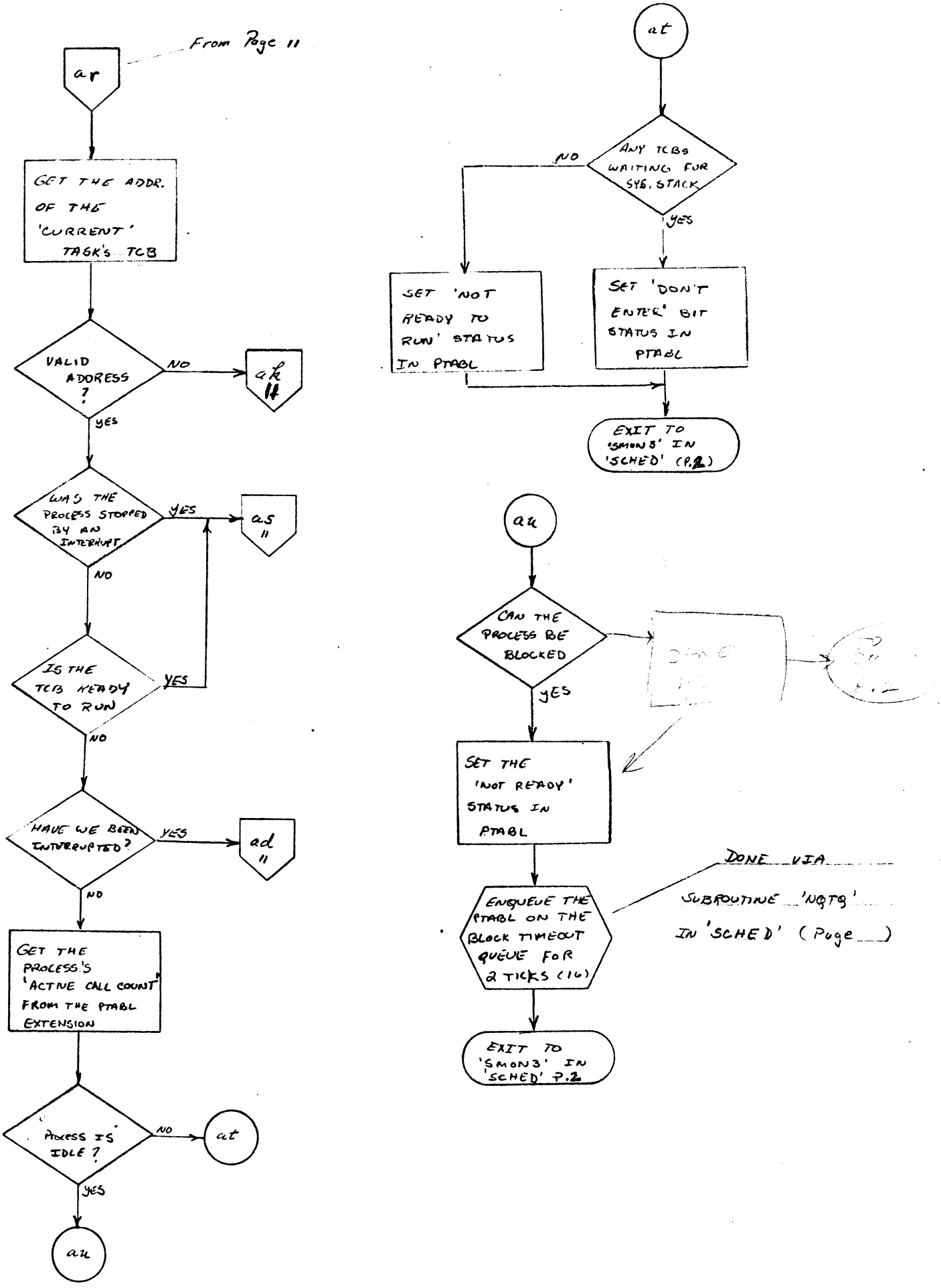



ag
IS THE LINK ADDRESS VALID
NO
YES
HAVE WE EXCEEDED THE MAX # OF TCBs
YES
NO
ak 11
IS THE TCB OF THE SAME PRI
YES
NO
al 11
ah
THIS INSURES ROUND ROBIN SCHEDULING WITHIN A PRIORITY GROUP
LINK THE 'CURRENT' TCB INTO THE CHAIN
am
WAS THE PTABL ON THE BLOCK TIME-OUT QUEUE
NO
YES
RESET THE TIME-OUT STATUS BIT AND DEQUEUE THE PTABL FROM THE BLOCK TIMEOUT QUEUE
an

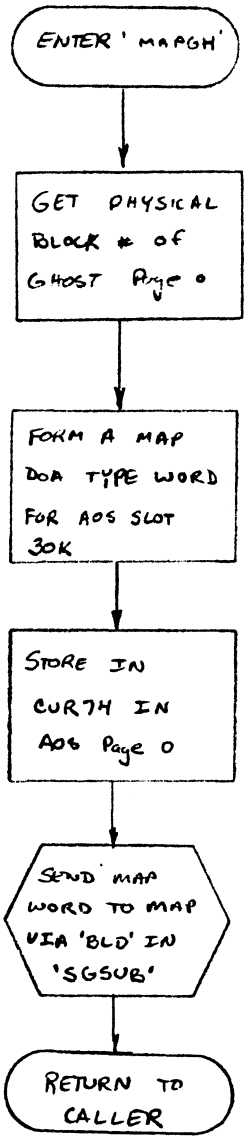

an
WILL THE TCB BE RUNNING IN THE GHOST
NO
YES
MAP THE GHOST PAGE 0 INTO THE 7HK SLOT VIA 'MAP&H' (15)
RESTORE THE USER's STACK STATE
GET THE ADDR OF THE LAST TCB TO RUN USING THE EXTENDED VARIABLES OPTION
IS IT = TO THE CURRENT TCB
NO
ap 13
YES
HAVE THE EXTENDED VAR. BEEN SAVED YET
YES
NO
SAVE THE TCB's EXTENDED VAR AREA & MARK IT AS SAVED
ao 13

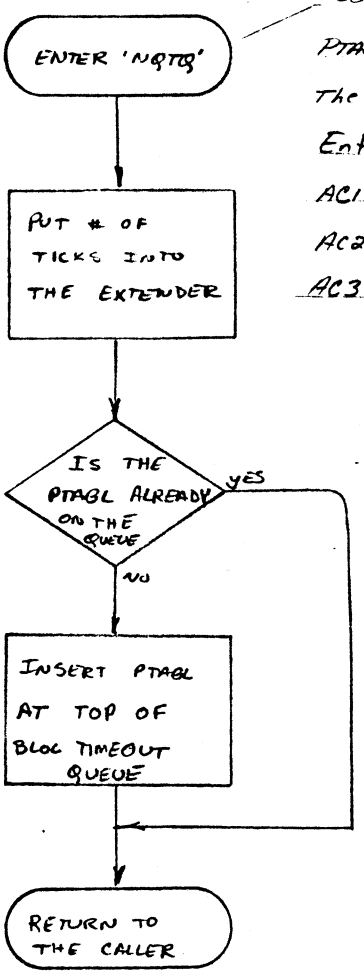

clo
DOES THE CURRENT TCB HAVE EXTEN VAR.
NO
YES
LOAD THE EXTENDED VARIABLES FOR THE CURRENT TCB
SAVE THE ADDR OF THE CURRENT TCB IN THE PTABL EXTENDER OF THE LAST TO USE EX. VAR.
ap
FPU SAVE AREA EXIST FOR CURRENT TCB
YES
WAS THIS TCB THE LAST FPU TCB TO RUN ?
NO
YES
LOAD THE FPU FROM THE PTABL EXTENSION
LOAD THE FPU FROM THE TCB FPU SAVE AREA
ag


ag
LOAD THE USER MAP VIA 'MPLD' IN 'SCHED'
RESTORE USER'S ACs from TCB
EXIT TO USER via 'PORJ'


From Page 11
ar
GET THE ADDR. OF THE 'CURRENT' TASK'S TCB
VALID ADDRESS ?
NO
ar 11
YES
WAS THE PROCESS STOPPED BY AN INTERRUPT
YES
as 11
NO
IS THE TCB READY TO RUN
YES
NO
HAVE WE BEEN INTERRUPTED?
YES
ad 11
NO
GET THE PROCESS'S 'ACTIVE CALL COUNT' FROM THE PTABL EXTENSION
PROCESS IS IDLE ?
NO
at
YES
au
at
ANY TCBs WAITING FOR SYS. STACK
NO
YES
SET 'NOT READY TO RUN' STATUS IN PTABL
SET 'DON'T ENTER' BIT STATUS IN PTABL
EXIT TO 'SMON3' IN 'SCHED' (P.2)
au
CAN THE PROCESS BE BLOCKED
YES
SET THE 'NOT READY' STATUS IN PTABL
ENQUEUE THE PTABL ON THE BLOCK TIMEOUT QUEUE FOR 2 TICKS (1G)
DONE VIA SUBROUTINE 'NQTQ' IN 'SCHED' (Page ___)
EXIT TO 'SMON3' IN 'SCHED' P.2

Subroutine to Map a GHOST Context's Page 0 to AOS Context's Page 30,0





Subroutine to Enqueue a PTABL to the head of the Block Timeout Queue

Enter with:

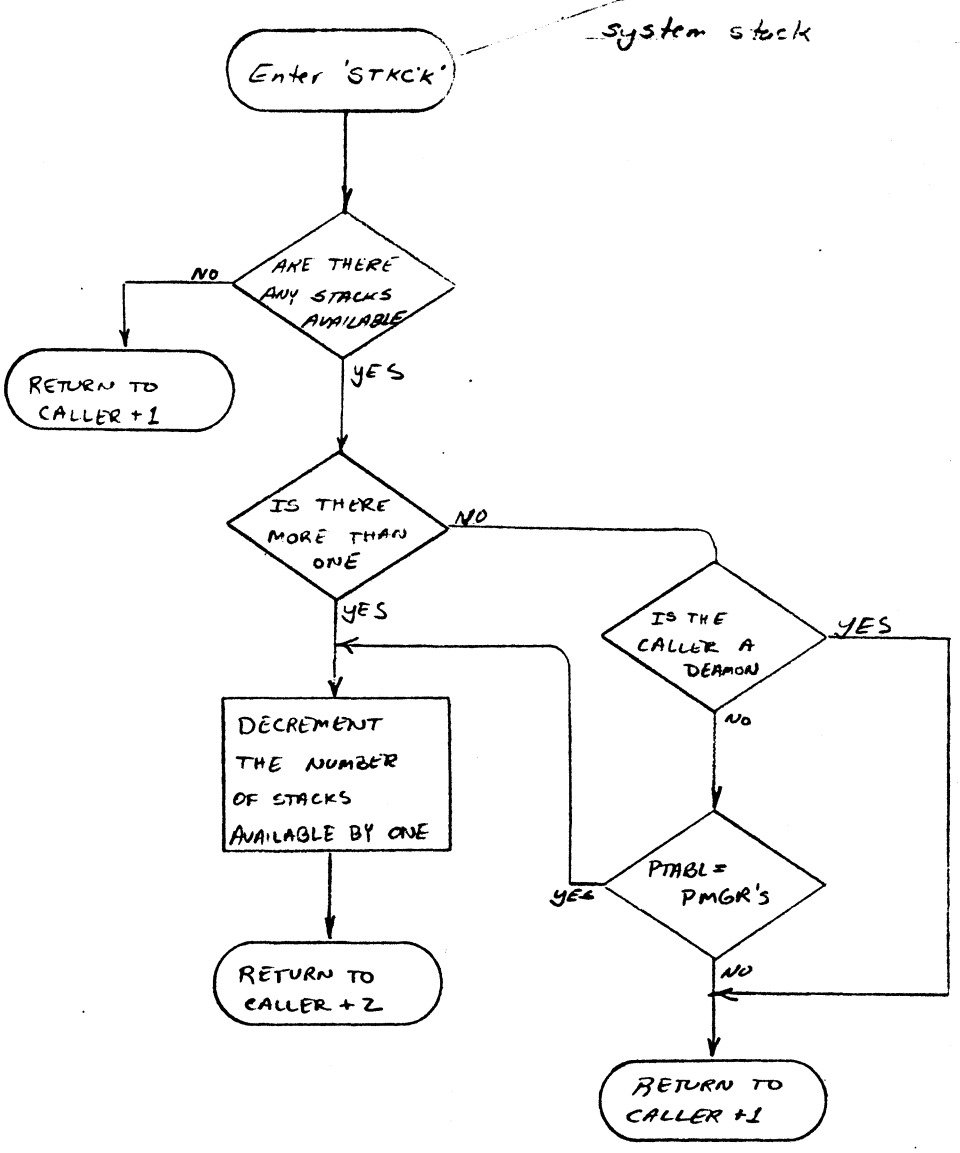
AC1 = # of Ticks

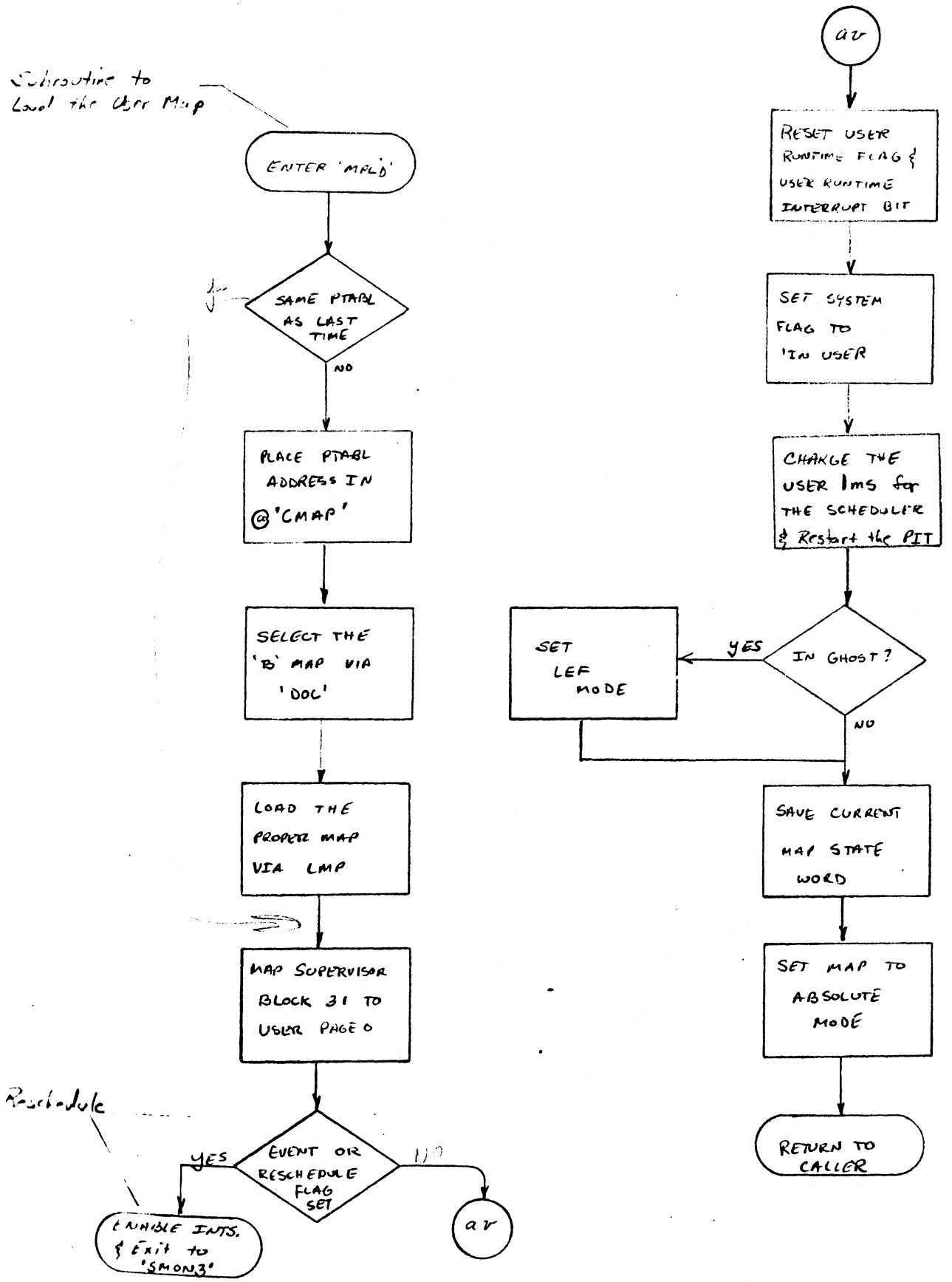
AC2 = Addr of PTABL Extender

AC3 = Addr of PTABL

Note: This routine Does Not Exist in Rev. 2

Subroutine to allocate a system stack

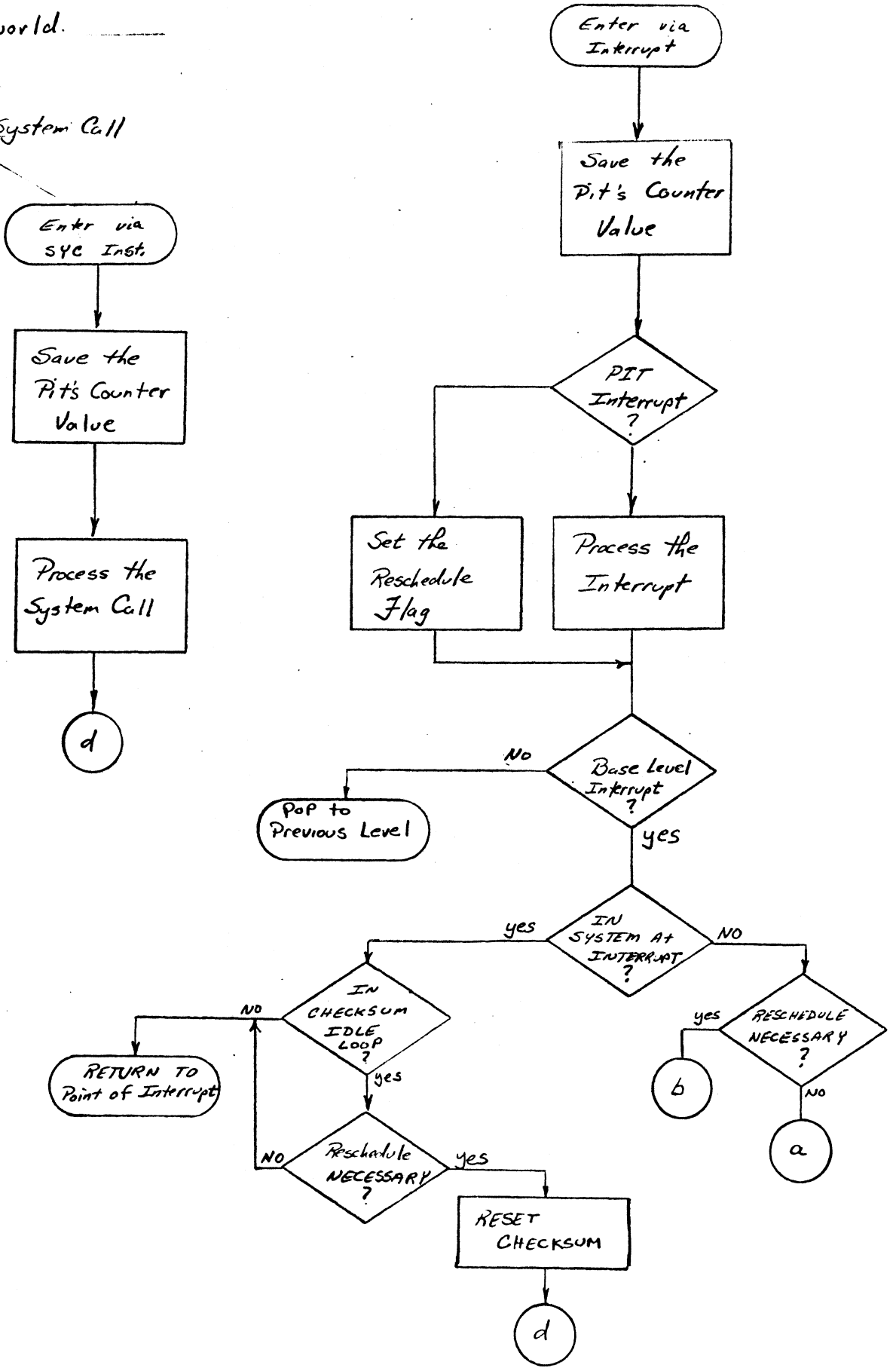


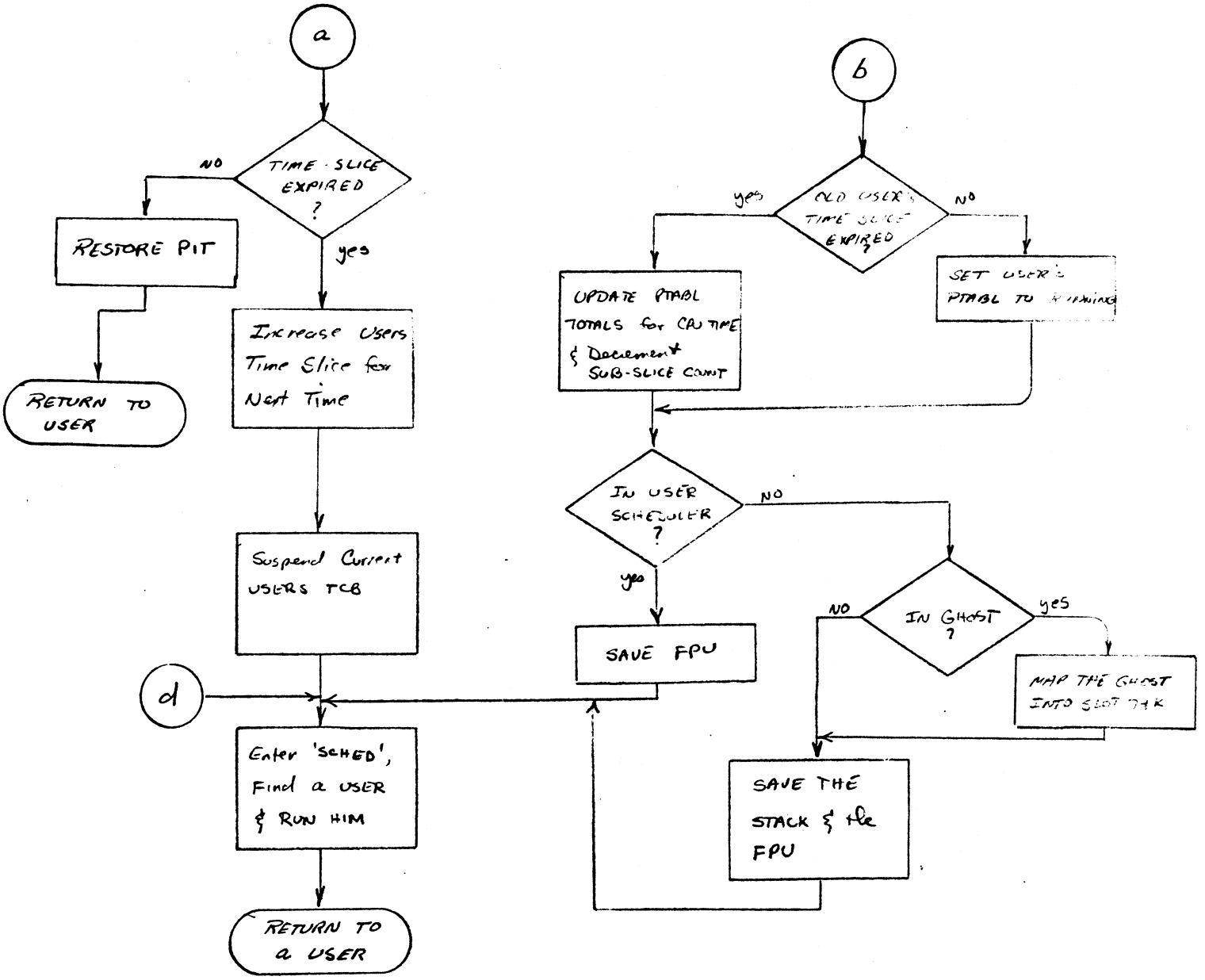

Subroutine to Load the User Map
ENTER 'MAPLD'
SAME PTABL AS LAST TIME
NO
PLACE PTABL ADDRESS IN @'CMAP'
SELECT THE 'B' MAP VIA 'DOC'
LOAD THE PROPER MAP VIA LMP
MAP SUPERVISOR BLOCK 31 TO USER PAGE 0
Reschedule
YES
EVENT OR RESCHEDULE FLAG SET
NO
ENABLE INTS. & EXIT to 'SMON3'
GV
GV
RESET USER RUNTIME FLAG & USER RUNTIME INTERRUPT BIT
SET SYSTEM FLAG TO 'IN USER
CHANGE THE USER 1ms for THE SCHEDULER & Restart the PIT
IN GHOST?
YES
SET LEF MODE
NO
SAVE CURRENT MAP STATE WORD
SET MAP TO ABSOLUTE MODE
RETURN TO CALLER

Overview of _____

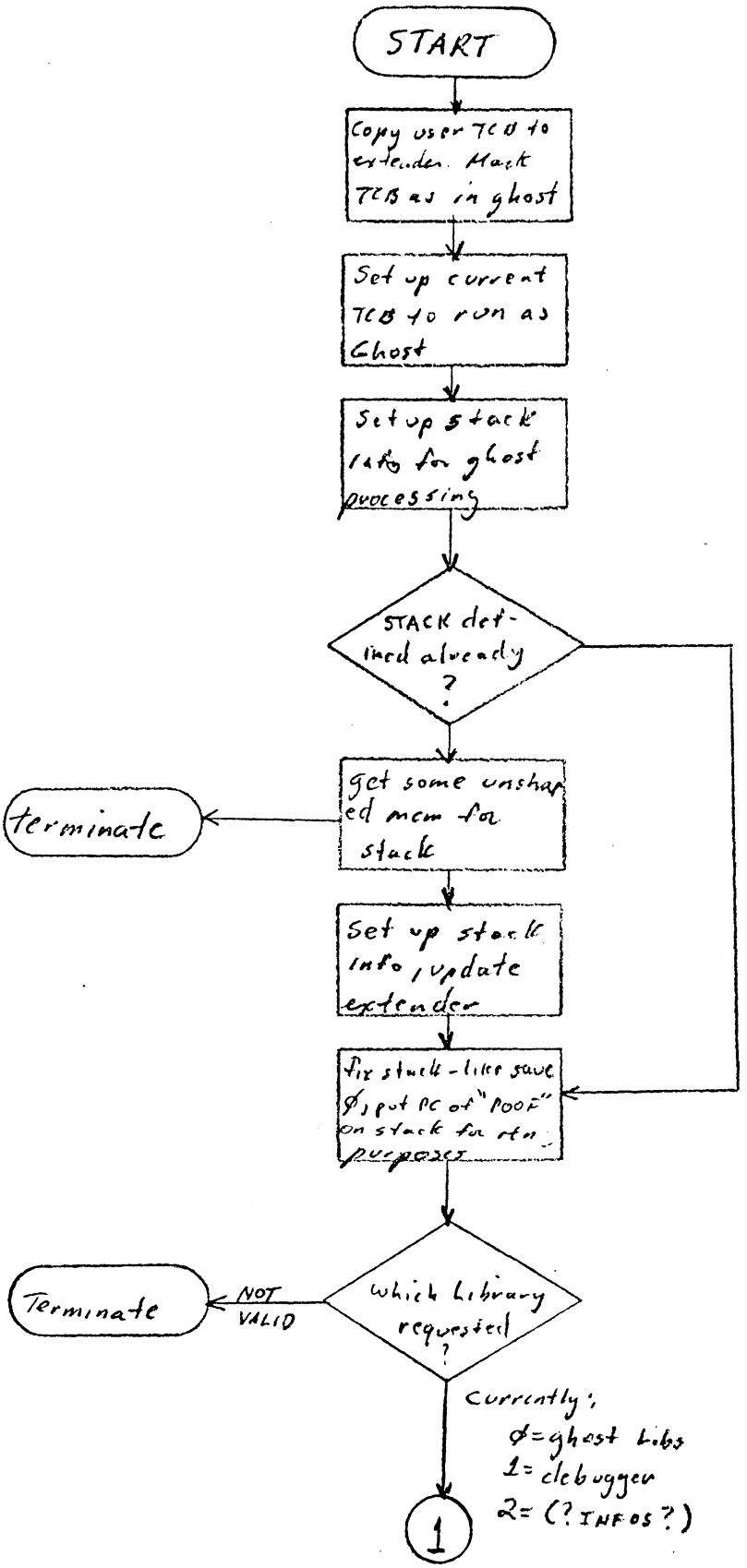
· Re-schedule operation from the interrupt world.

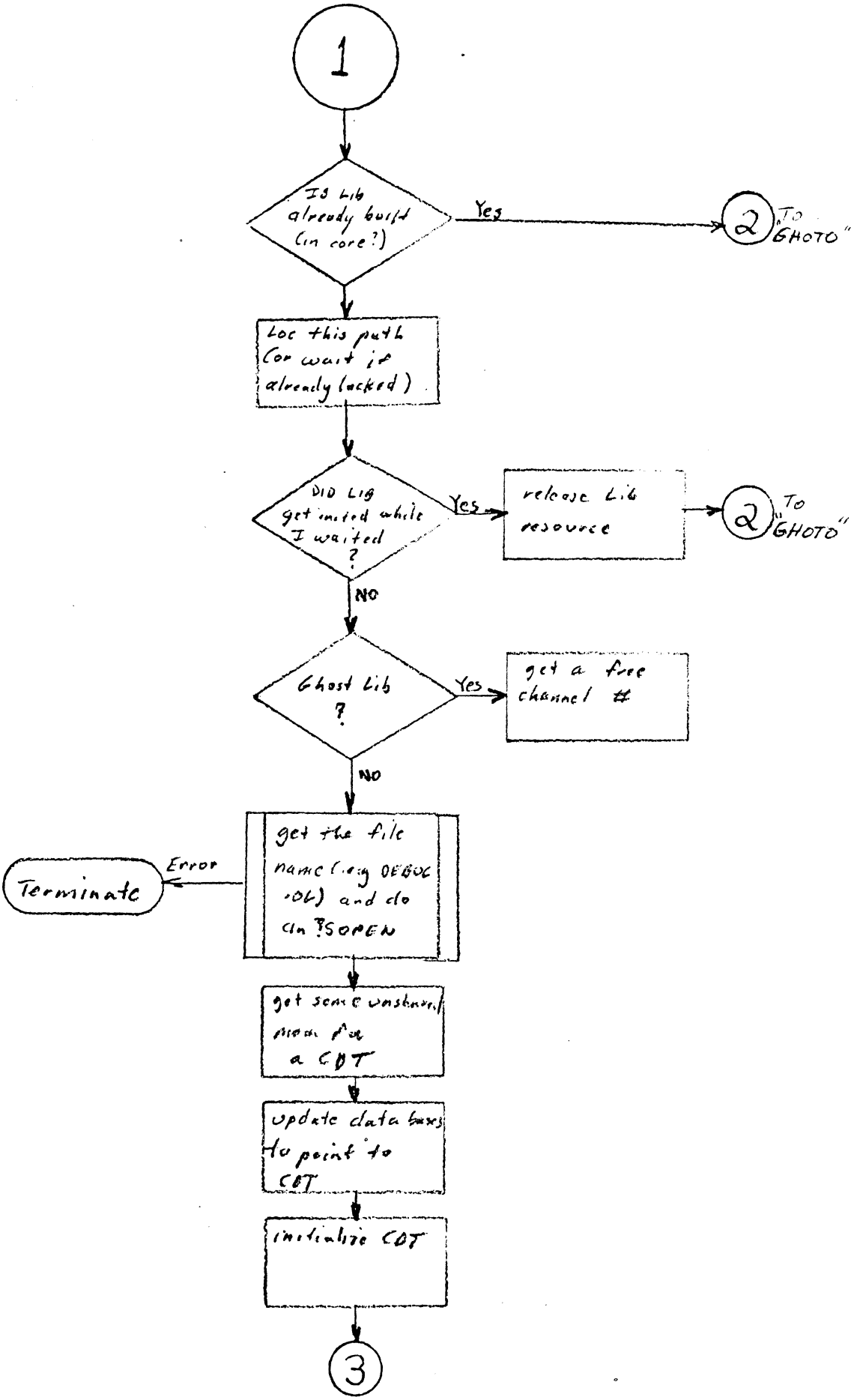
System entered via a Trap/System Call

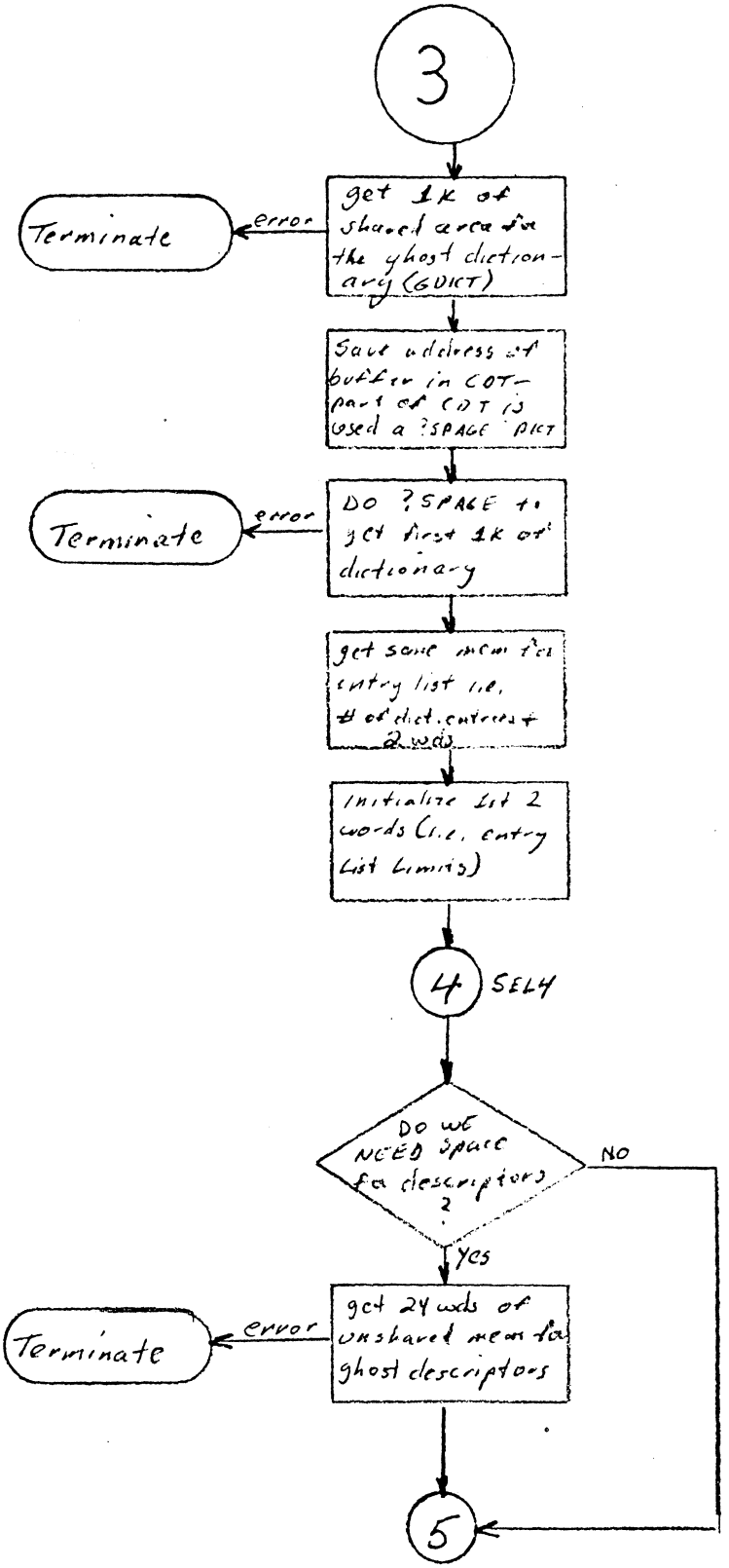


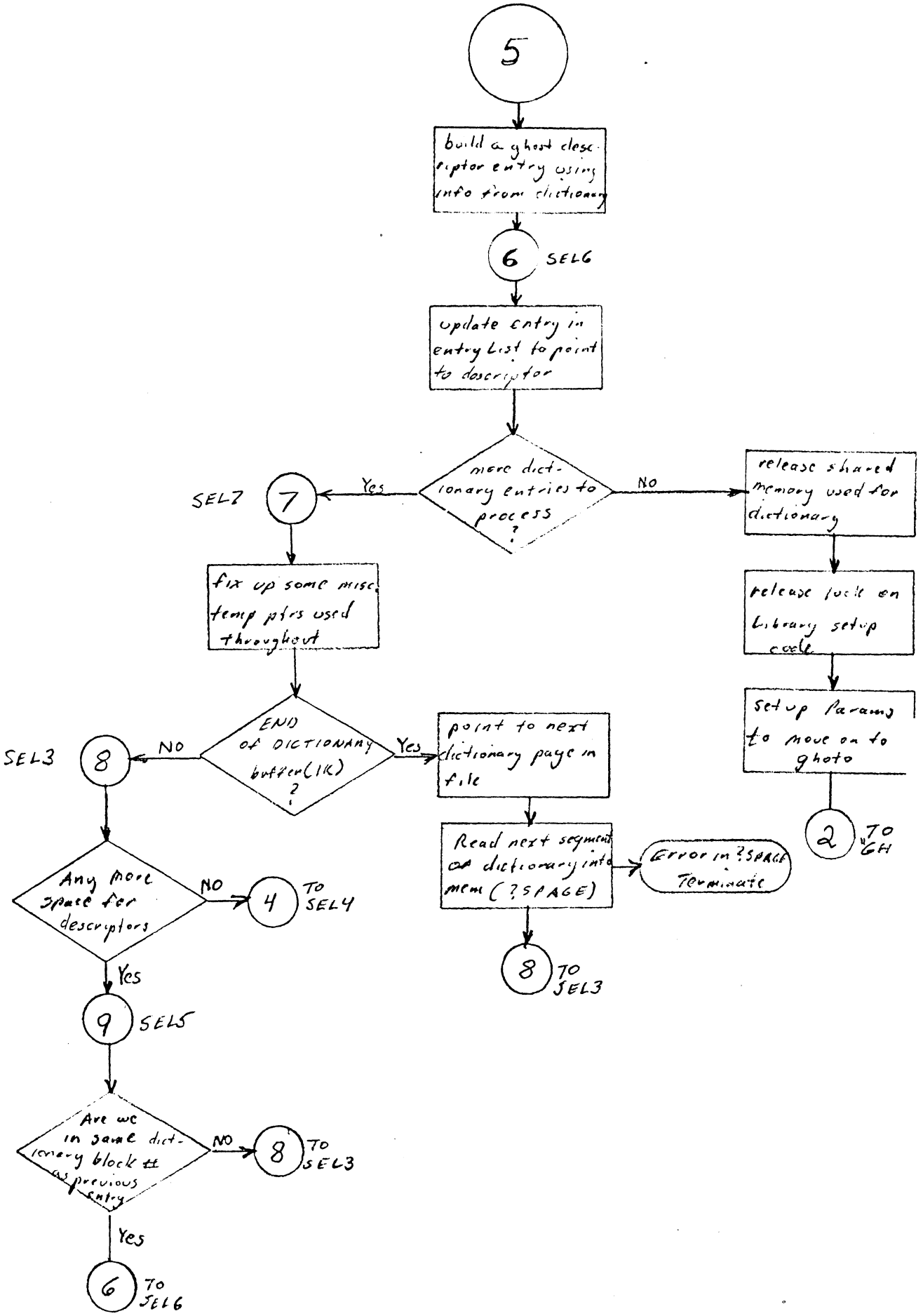

a
TIME-SLICE EXPIRED ?
NO
yes
RESTORE PIT
RETURN TO USER
Increase Users Time Slice for Next Time
Suspend Current USERS TCB
d
Enter 'SCHED', Find a USER & RUN HIM
RETURN TO a USER
b
OLD USER'S TIME SLICE EXPIRED ?
yes
NO
UPDATE PTABL TOTALS for CPU TIME & Decrement SUB-SLICE CNT
SET USER'S PTABL TO RUNNING
IN USER SCHEDULER ?
NO
yes
SAVE FPU
IN GHOST ?
NO
yes
MAP THE GHOST INTO SLOT 74K
SAVE THE STACK & the FPU

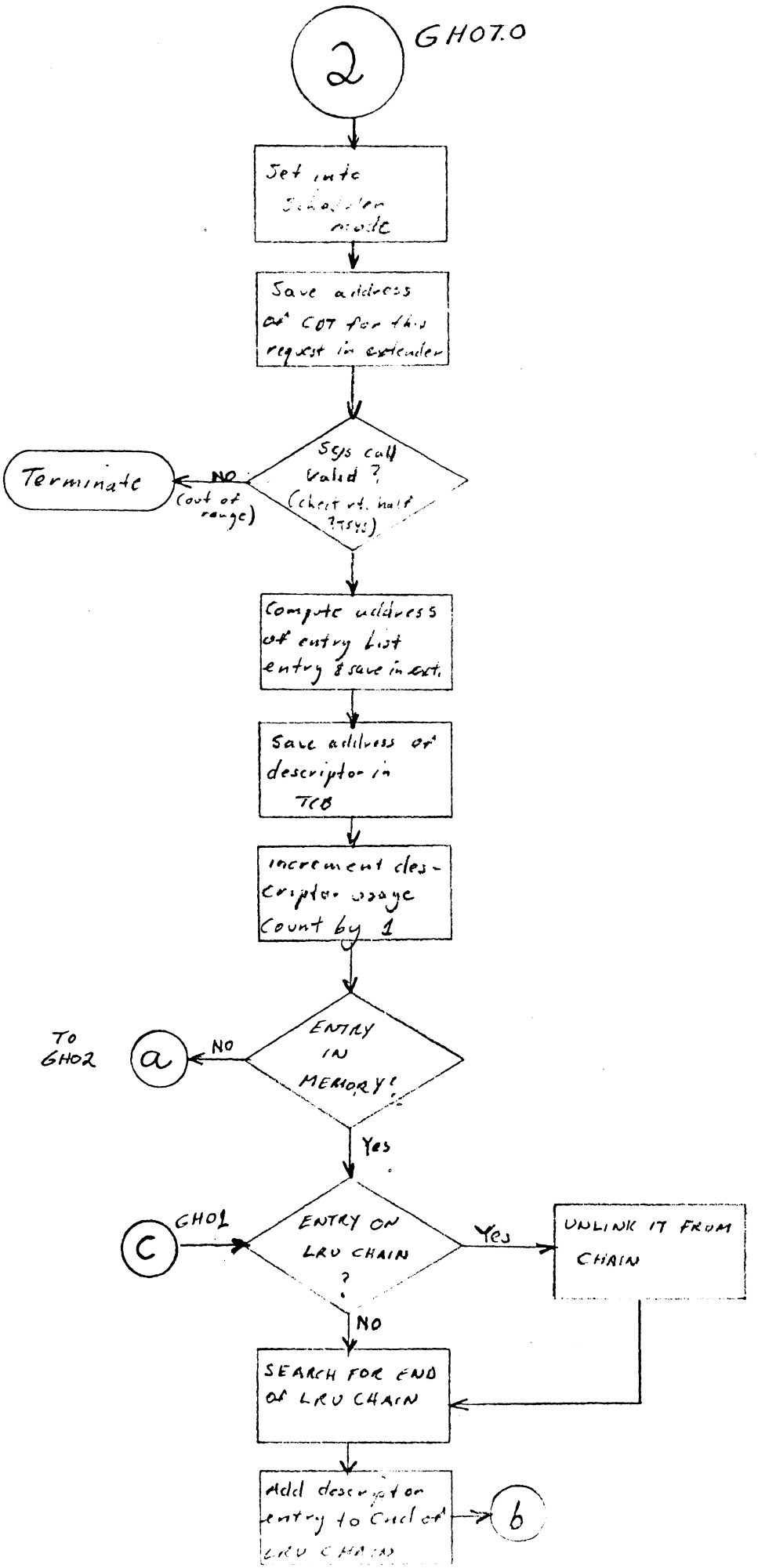
GHOST: ghost dispatcher

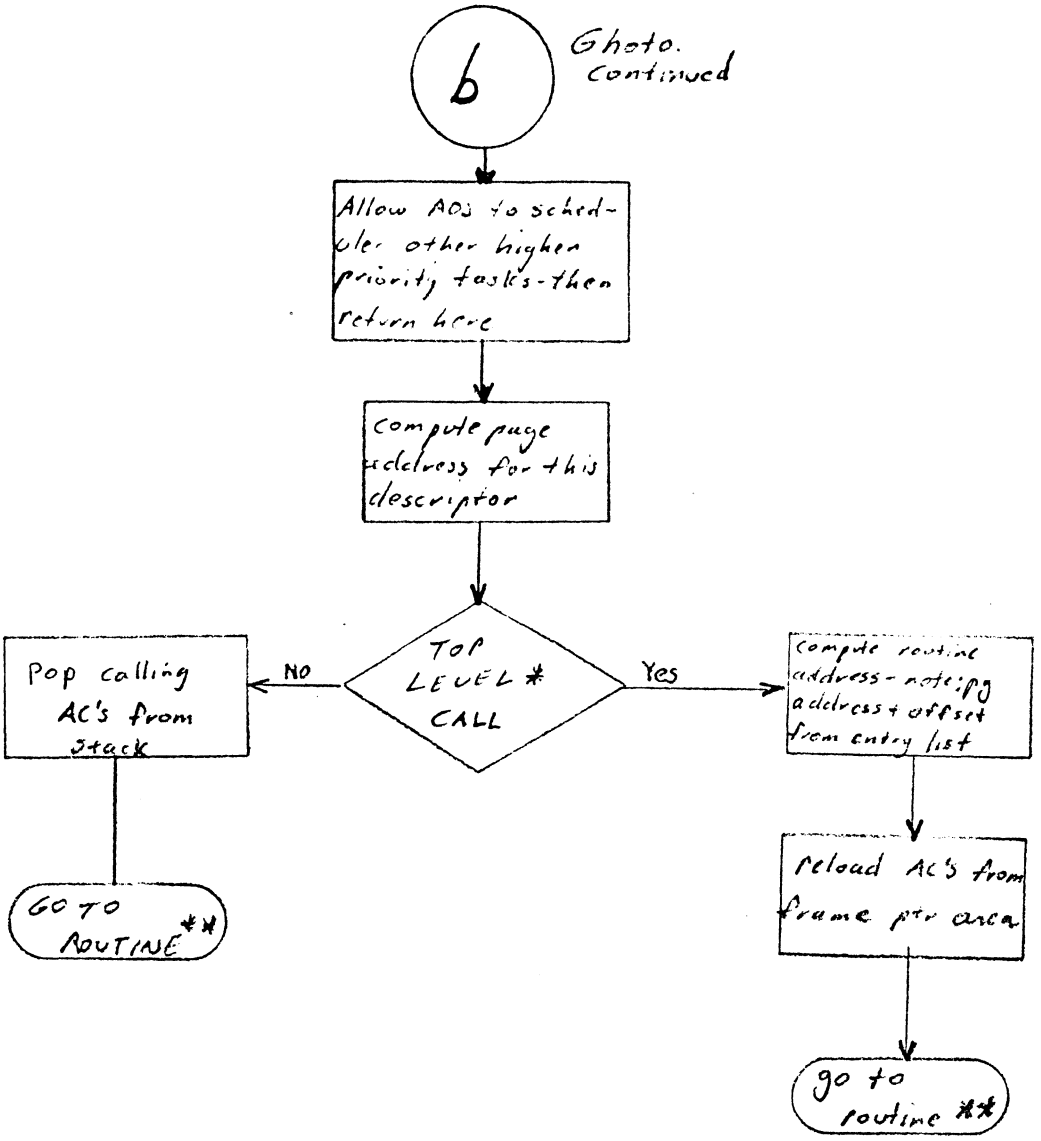



1
Is Lib already build (in core?)
Yes
2 To "GHOTO"
Loc this path (or wait if already locked)
DID Lib get invited while I waited ?
Yes
release Lib resource
2 To "GHOTO"
NO
Ghost Lib ?
Yes
get a free channel #
NO
get the file name (e.g. DEBUG .OL) and do an FSOPEN
Error
Terminate
get some unshared mem for a CDT
update data bases to point to CDT
initialize CDT
3


3
get 1K of shared area for the ghost dictionary (GDICT)
error
Terminate
Save address of buffer in CDT - part of CDT is used a ?SPACE DICT
DO ?SPACE to get first 1K of dictionary
error
Terminate
get some memory for entry list i.e. # of dict. entries + 2 wds
Initialize 1st 2 words (i.e. entry List Limits)
4
SEL4
DO WE NEED SPACE for descriptors ?
NO
Yes
get 24 wds of unshared memory for ghost descriptors
error
Terminate
5

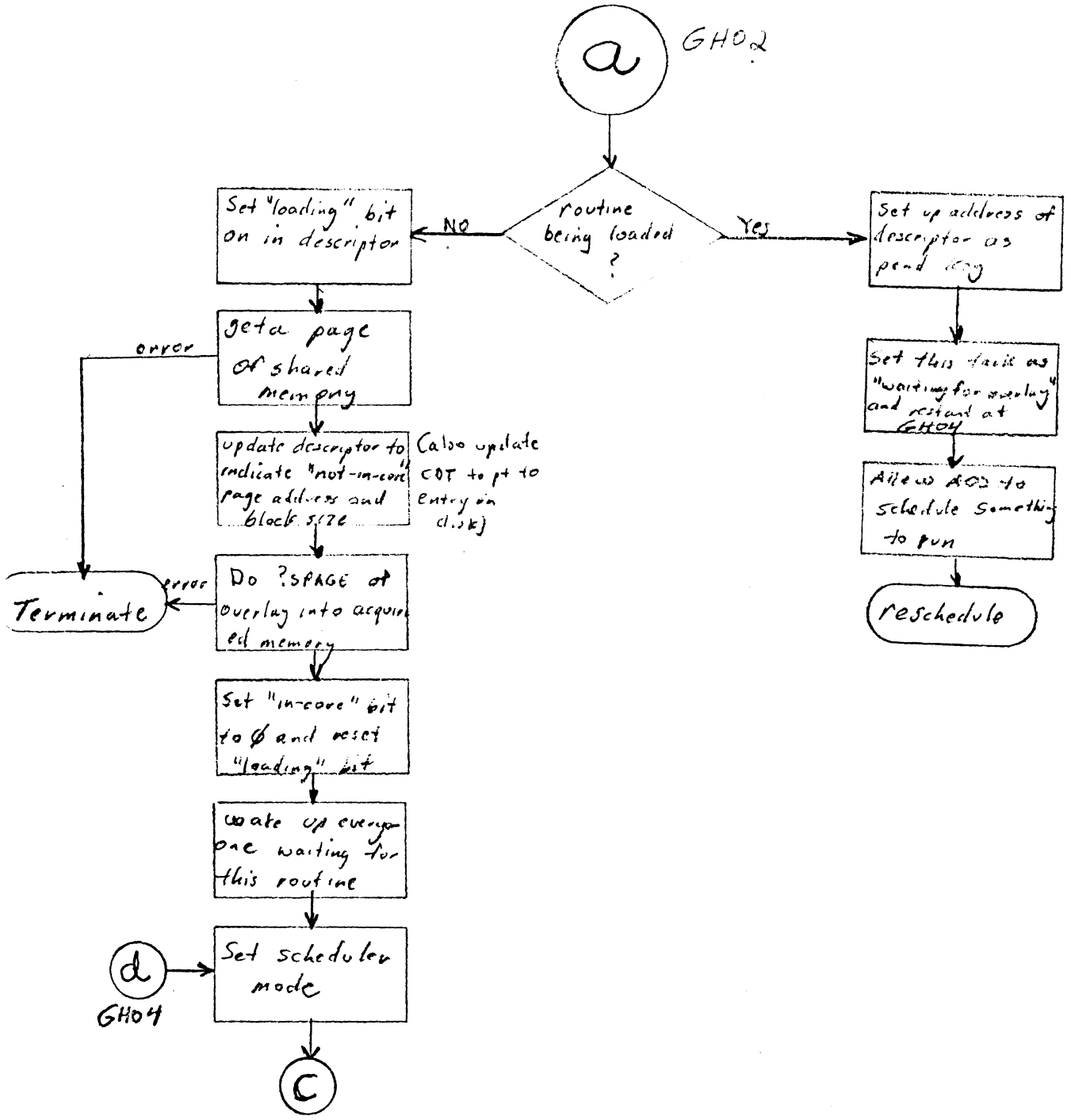

5
build a ghost descriptor entry using info from dictionary
6 SEL6
update entry in entryList to point to descriptor
more dictionary entries to process ?
Yes
No
SEL7 7
release shared memory used for dictionary
fix up some misc temp ptrs used throughout
release lock on Library setup code
END of DICTIONARY buffer(1K) ?
NO
Yes
SEL3 8
point to next dictionary page in file
set up params to move on to ghoto
2 TO GH
Read next segment of dictionary into mem (?SPACE)
Error in ?SPACE Terminate
Any more space for descriptors
NO
4 TO SEL4
8 TO SEL3
Yes
9 SEL5
Are we in same dictionary block as previous entry
NO
8 TO SEL3
Yes
6 TO SEL6

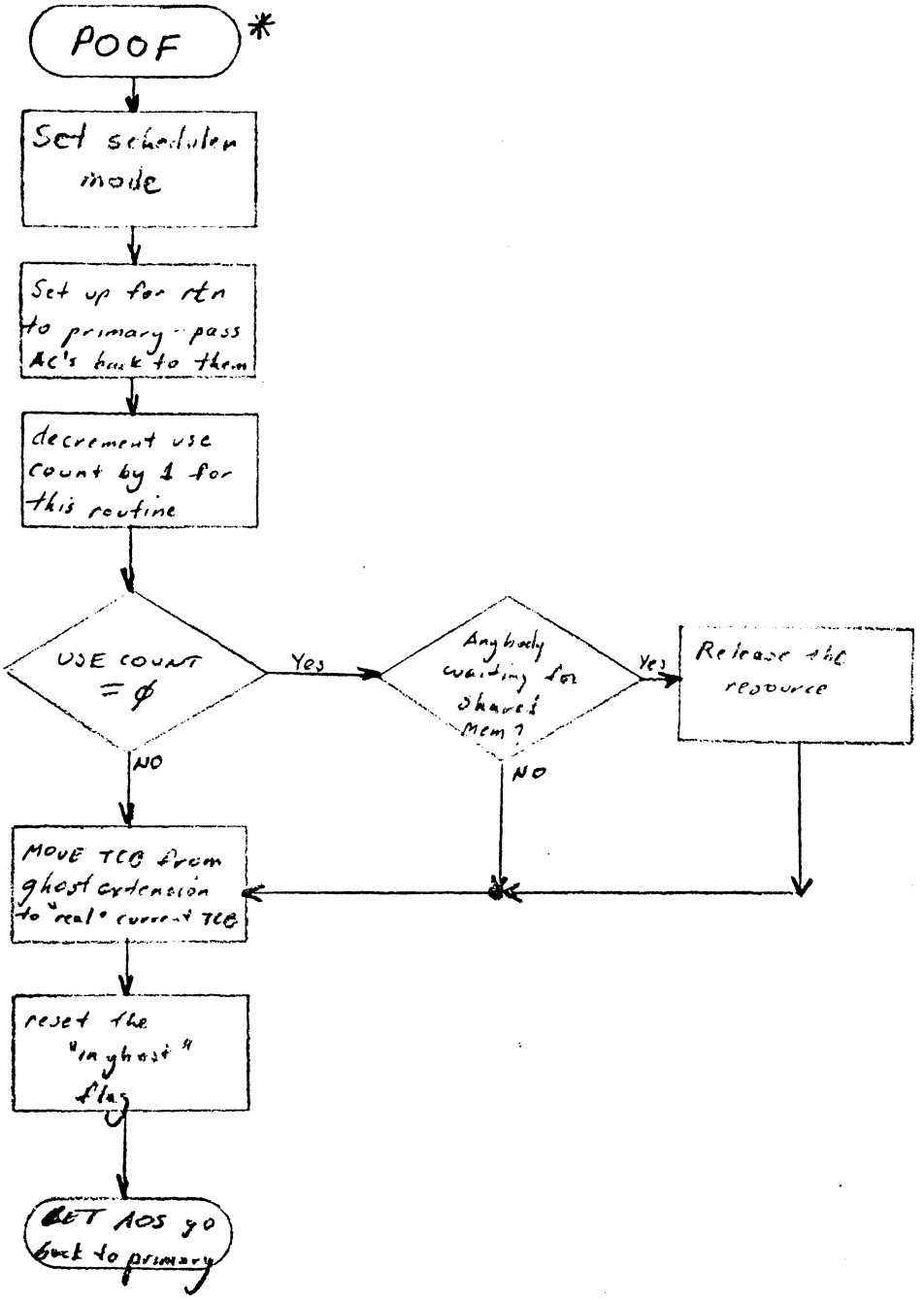

2
GHOT.0
Set into Scheduler mode
Save address of CDT for this request in extender
Sys call valid ? (check vt. half ?TSYS)
NO (out of range)
Terminate
Compute address of entry List entry & save in ext.
Save address of descriptor in TCB
increment descriptor usage count by 1
ENTRY IN MEMORY ?
NO
a
TO GHO2
Yes
ENTRY ON LRU CHAIN ?
GHO1
C
Yes
UNLINK IT FROM CHAIN
NO
SEARCH FOR END OF LRU CHAIN
Add descriptor entry to end of LRU CHAIN
b



* A top level call is made by the first routine to be entered on a ghost call
If that routine does a "call" or "chain" then subsequent calls are not top level calls

** A RTN from a routine will either go to "POOF" (if from top level) or back to caller (if recursive call and not a chain)

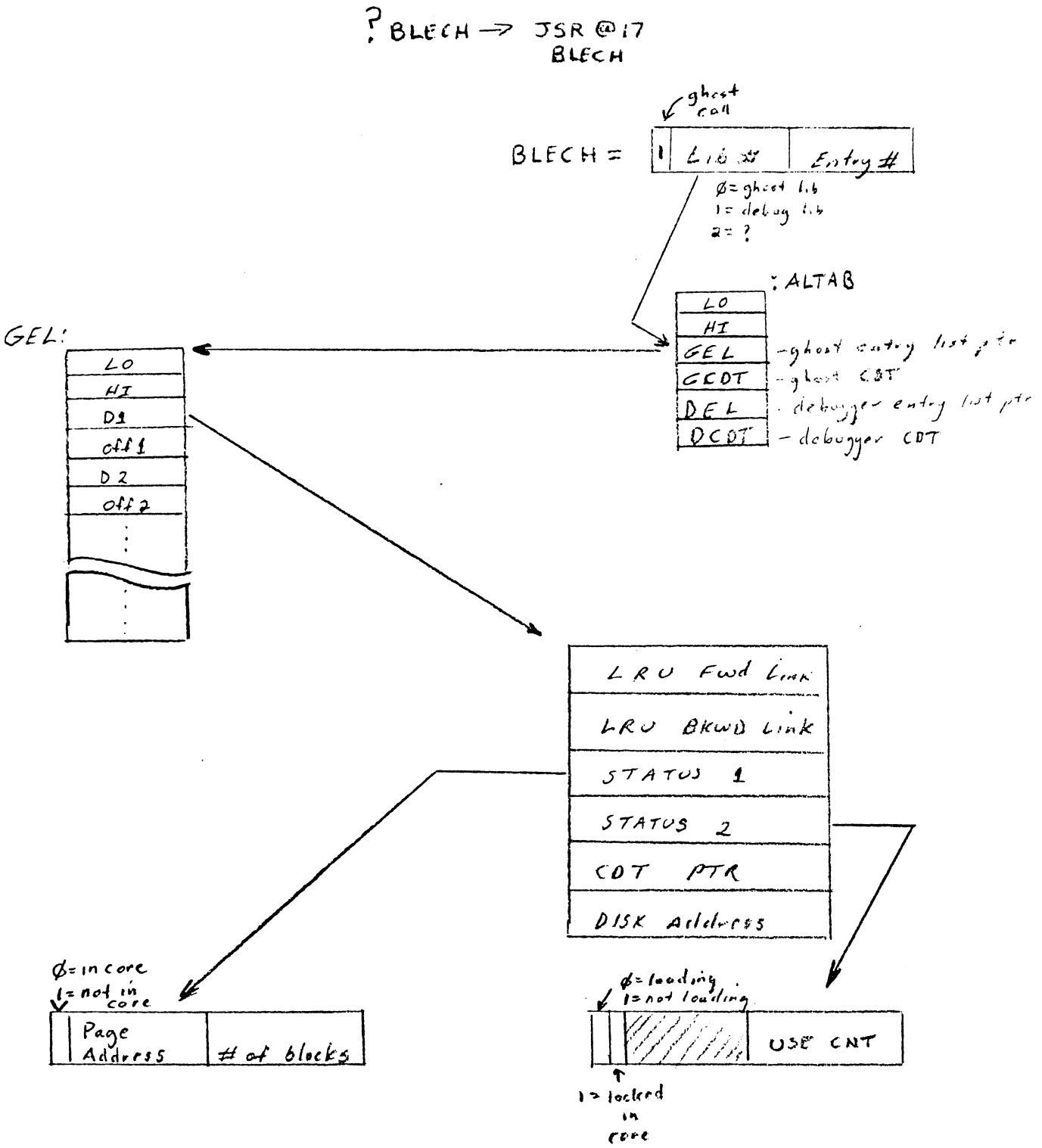

a
GHO2
routine being loaded ?
NO
Yes
Set "loading" bit on in descriptor
Set up address of descriptor as pend arg
get a page of shared memory
error
Set this task as "waiting for overlay" and restart at GHO4
update descriptor to indicate "not-in-core" page address and block size
Calso update COT to pt to entry on disk
Allow AQ2 to schedule something to run
Terminate
error
Do ?SPAGE of overlay into acquired memory
reschedule
Set "in-core" bit to Ø and reset "loading" bit
wake up everyone waiting for this routine
d
GHO4
Set scheduler mode
C



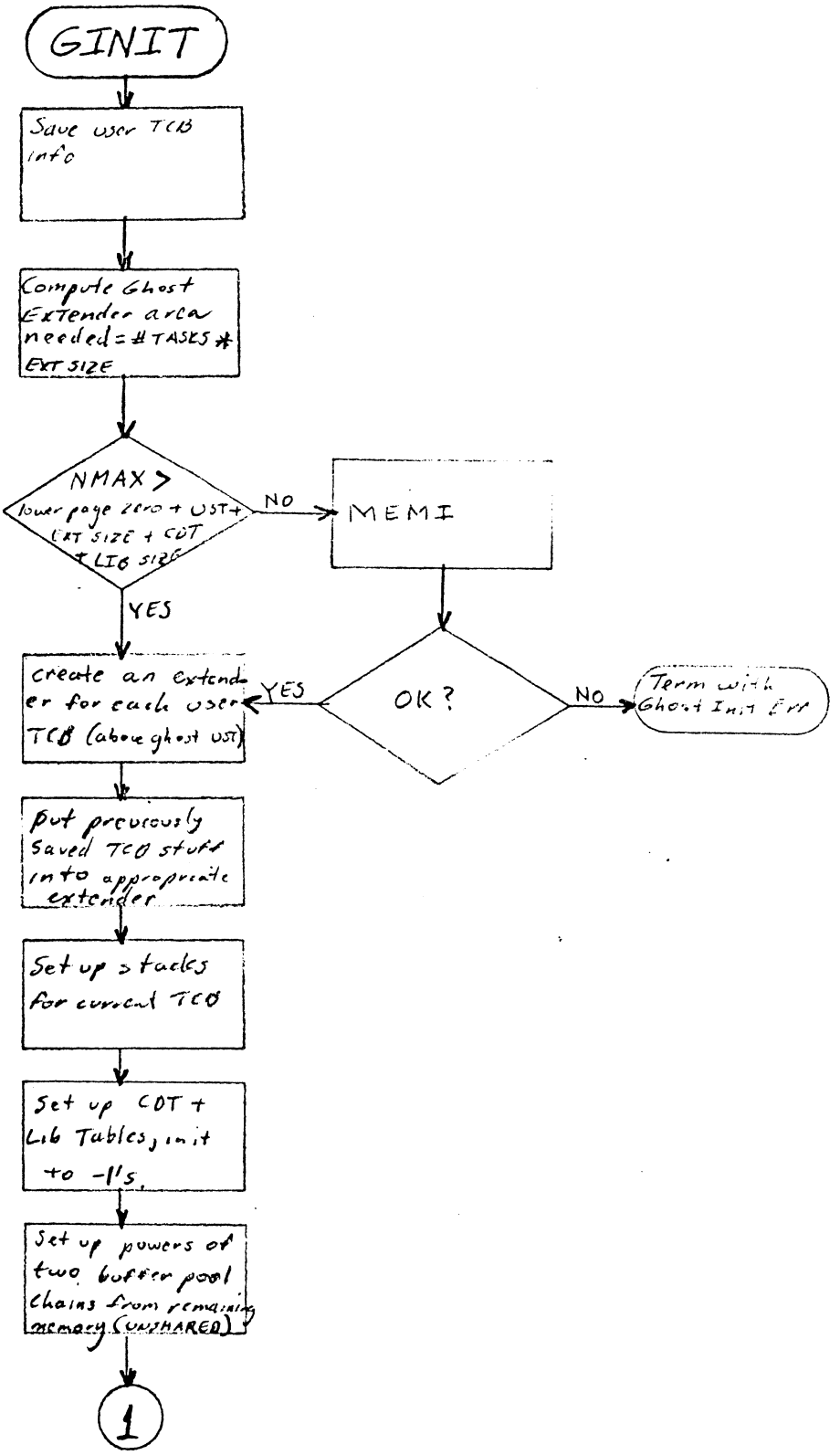
* always come here when ghost is to return to primary

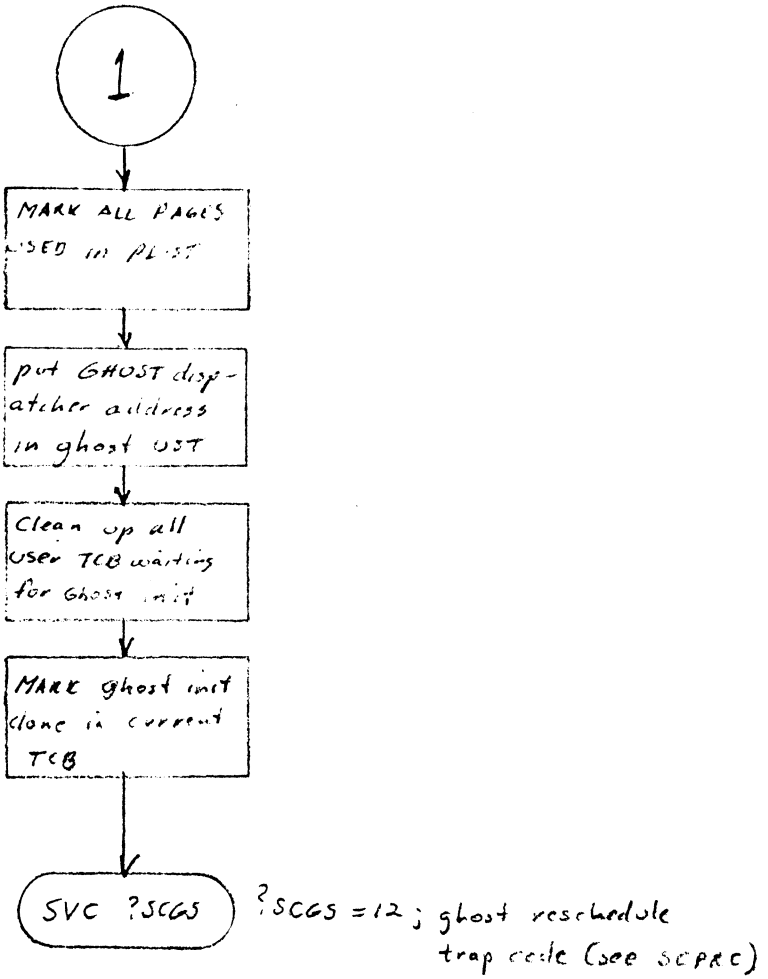
Appendix B

GHOST DATA BASES

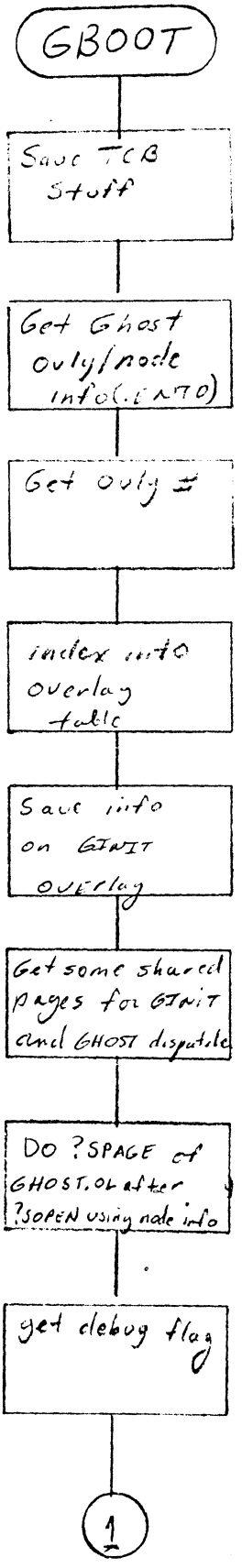


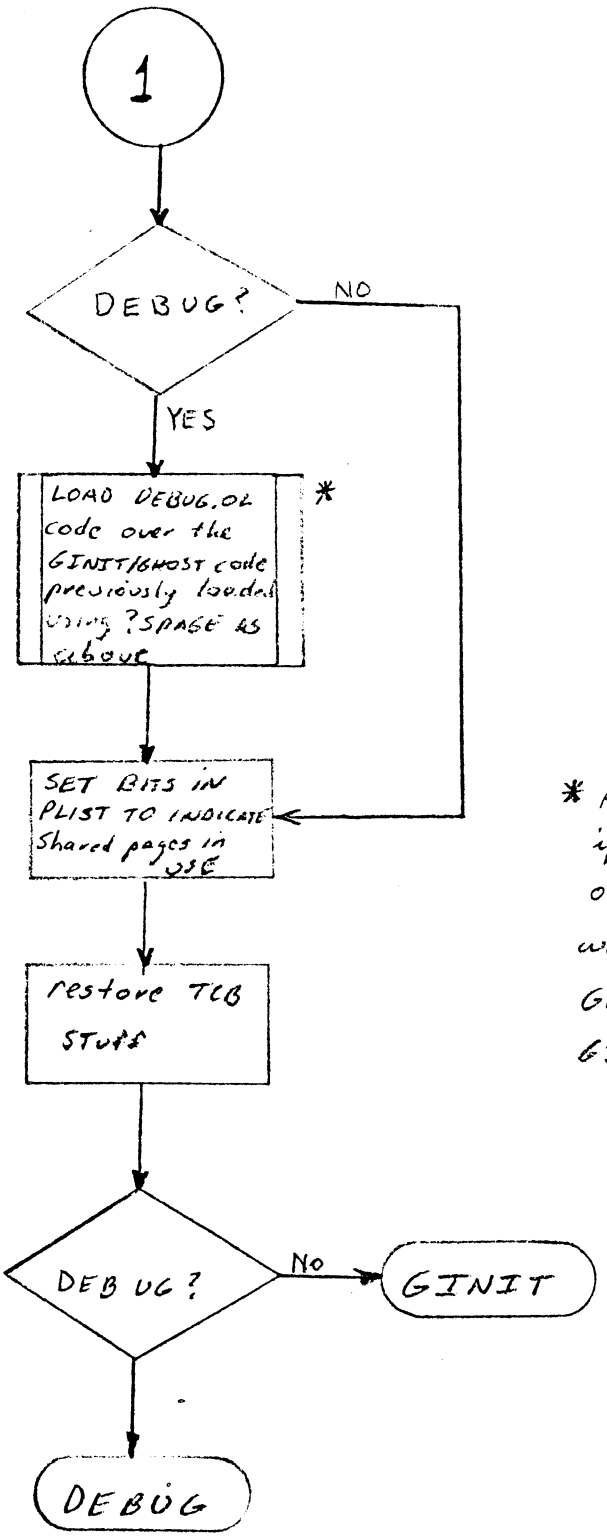
GINIT - second phase of ghost initialization entered from GHOST -
sets up the ghost (secondary) context environment




1
MARK ALL PAGES USED IN PLIST
put GHOST dispatcher address in ghost UST
Clean up all user TCB waiting for Ghost init
MARK ghost init clone in current TCB
SVC ?SCGS
?SCGS = 12 ; ghost reschedule trap code (see SCPRC)

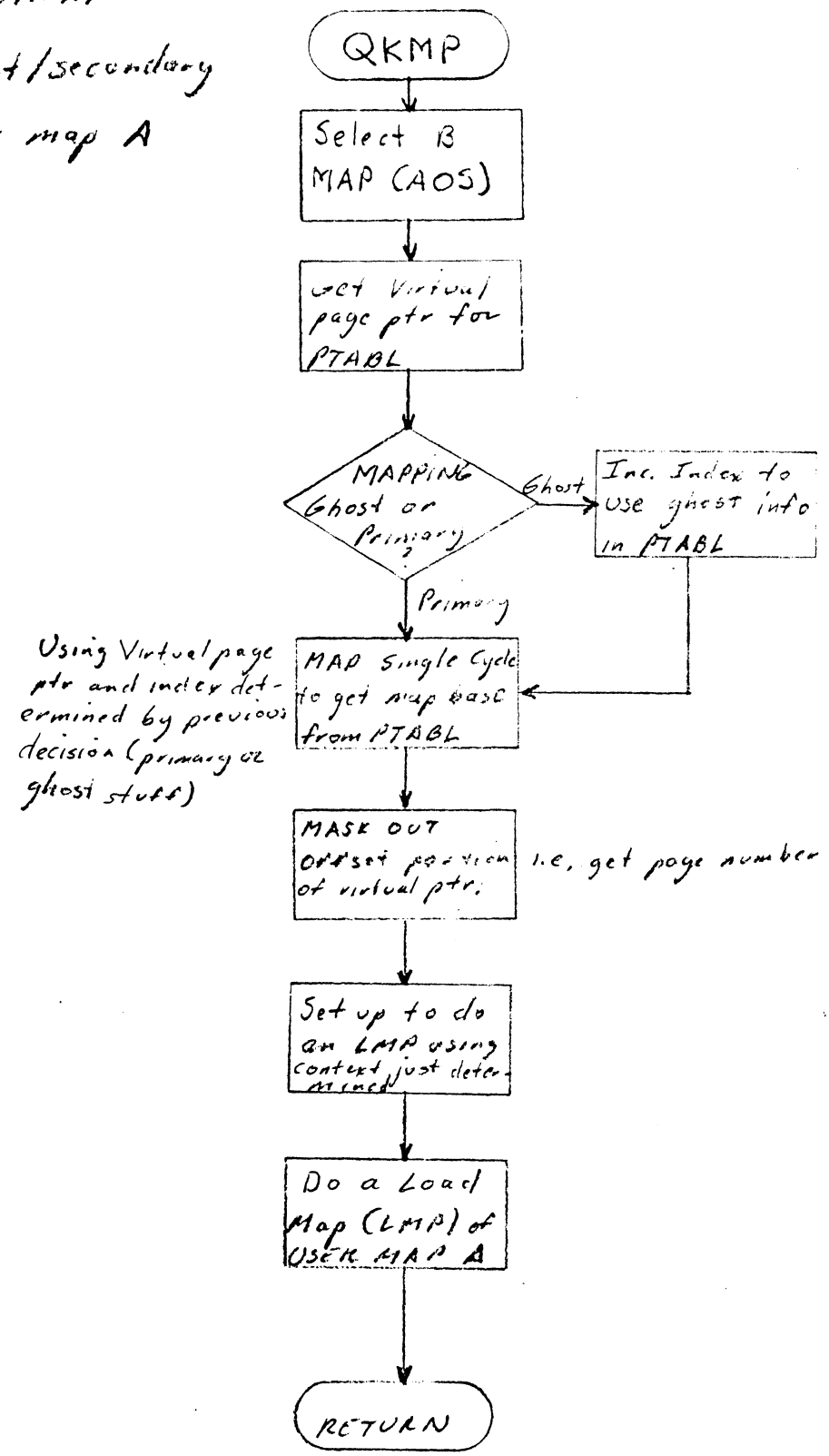
GHOST BOOTSTRAP - loaded by system into page zero of ghost context (from GHOST.PR) and entered by system to perform GHOST initialization



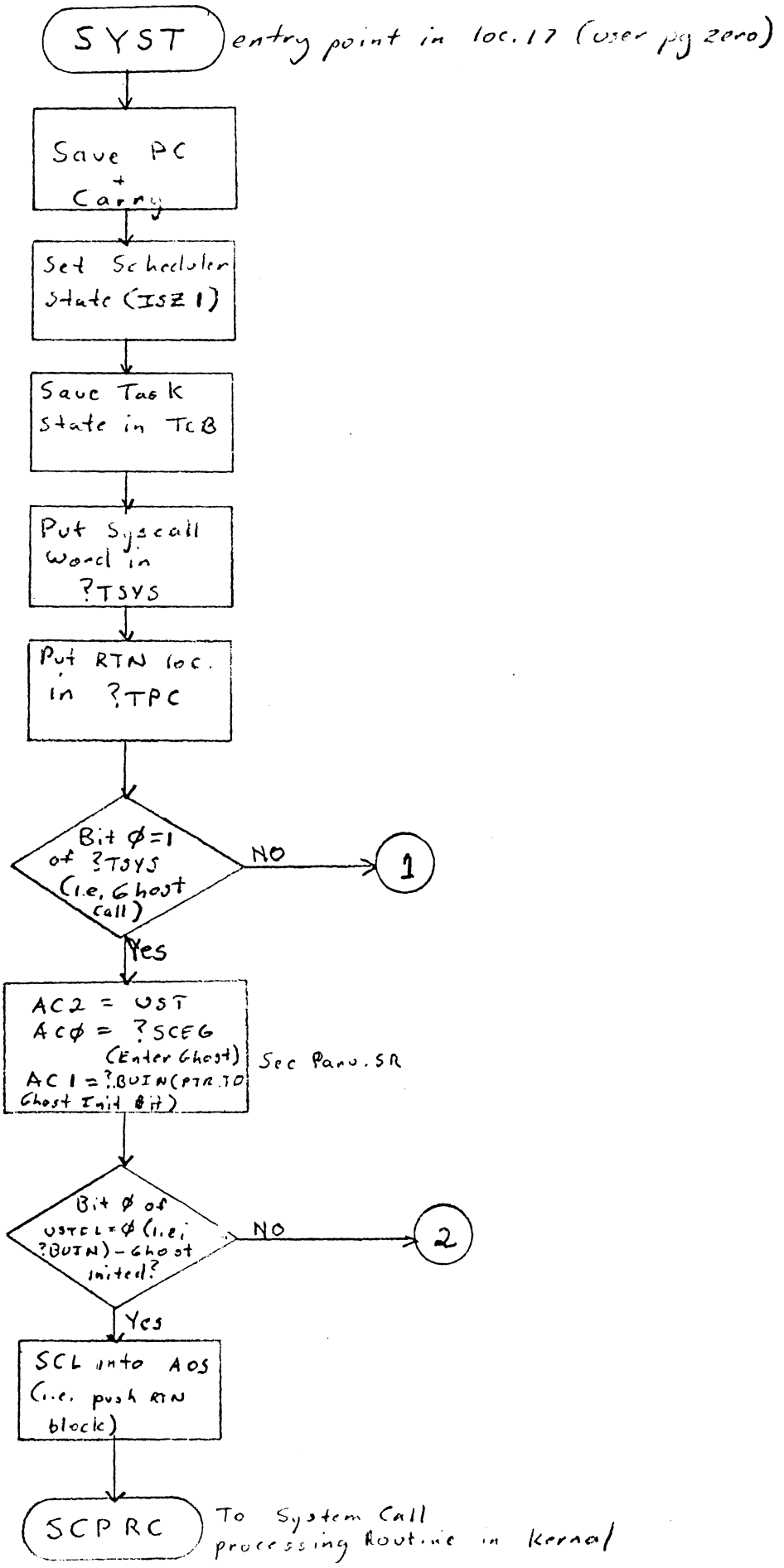


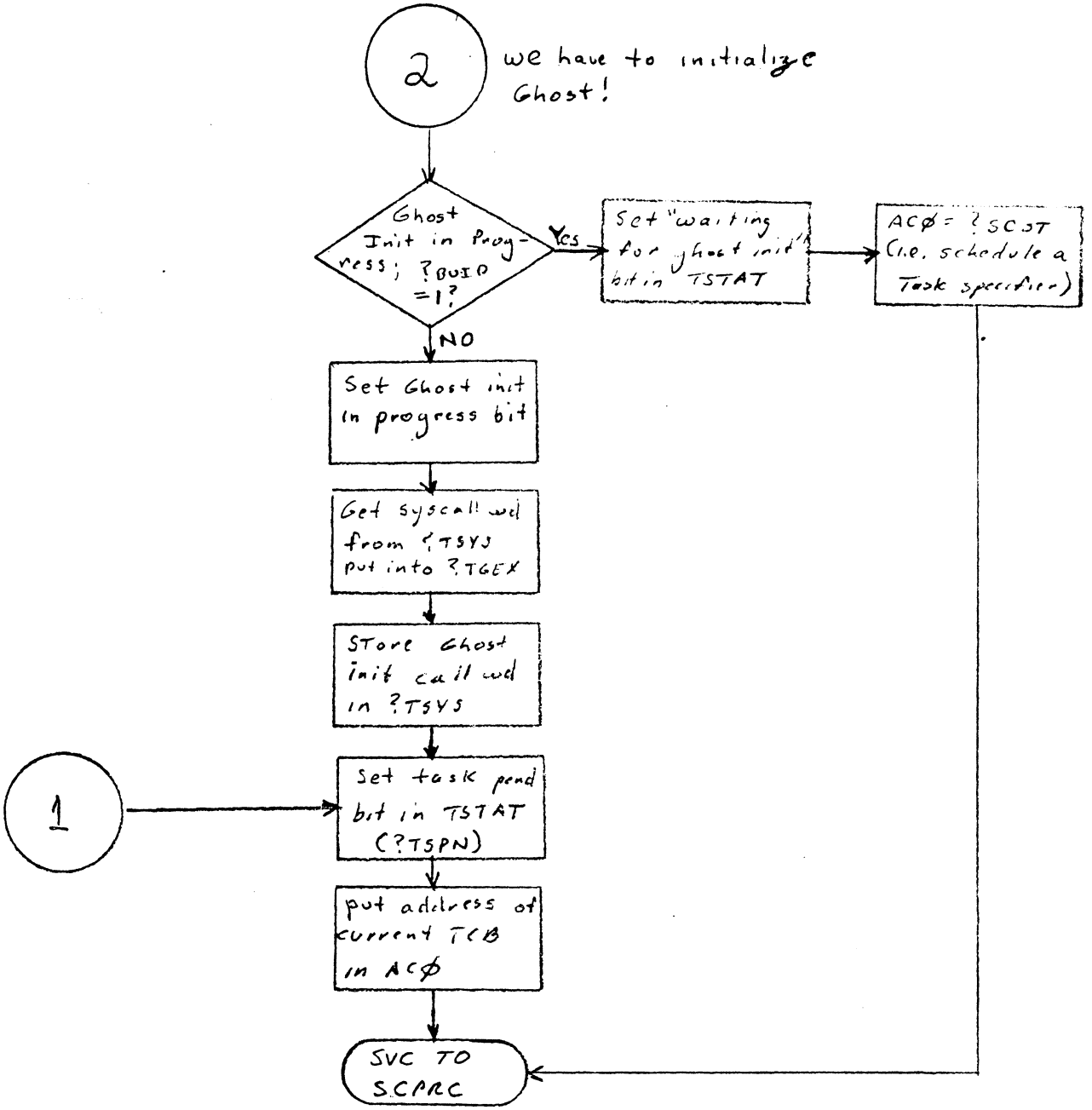
* Also does a DEBL if an error received on the DEBL call, GBOOT will then bring up the Ghost in non-debug or GINIT code

This routine is called by EGST of STRAN to set up the ghost/secondary co. ct in user map A



SCALL: Bound into user address space; entered via JSR @17




2
we have to initialize Ghost!
Ghost Init in Progress; ?BUID =1?
Yes
Set "waiting for ghost init" bit in TSTAT
AC0 = ?SCST (i.e. schedule a Task specifier)
NO
Set Ghost init in progress bit
Get syscall wd from ?TSYS put into ?TGEX
Store Ghost init call wd in ?TSYS
1
Set task pend bit in TSTAT (?TSPN)
put address of current TCB in AC0
SVC TO SCPRC

VIRTUAL PROCESS TABLE DESIGN SPECIFICATION

I. INTRODUCTION

THIS DOCUMENT DESCRIBES IN DETAIL THE DESIGN OF VIRTUAL PROCESS TABLES IN AOS. A VIRTUAL PROCESS TABLE IS A PROCESS TABLE THAT MUST BE MAPPED IN ORDER TO BE REFERENCED, I.E. DOES NOT RESIDE PERMANENTLY IN AOS'S LOGICAL ADDRESS SPACE. THE IMPLEMENTATION PRESENTED HERE ALSO ADDS A SECOND LEVEL QUEUE TO THE AOS SCHEDULER. THIS ALLOWS A CLEANER SEPARATION OF THE ISSUES OF "ELIGIBILITY" (I.E. ALLOCATING MAIN MEMORY TO A PROCESS) AND "RUNNABILITY" (I.E. ALLOCATING THE CPU TO A PROCESS). THUS IT HELPS SIMPLIFY MANAGEMENT OF THE TWO MOST IMPORTANT SYSTEM RESOURCES.

THE DESIGN ALSO INTRODUCES THE CONCEPT OF A "VIRTUAL PROCESSOR" TO AOS. A VIRTUAL PROCESSOR IS AN ABSTRACT REPRESENTATION OF A PHYSICAL CENTRAL PROCESSING UNIT; IT IS IN THIS SENSE ANALOGOUS TO A PROCESS, WHICH IS AN ABSTRACTION OF A USER'S PROGRAM.

COMBINING THE VIRTUAL PROCESSOR IDEA WITH THE SECOND LEVEL SCHEDULER QUEUE ALLOWS SOLUTION OF TWO PROBLEMS:

1. THE AOS LOGICAL ADDRESS SPACE PROBLEM

2. THE PERFORMANCE PROBLEM EXHIBITED IN AOS SYSTEMS WITH LARGE AMOUNTS OF PHYSICAL MEMORY

THIS DESIGN WILL PROVIDE THE FUNCTIONALITY DESCRIBED IN THE DOCUMENT "VIRTUAL PROCESS TABLE FUNCTIONAL SPECIFICATION". ONE NOTION INTRODUCED IN THE FUNCTIONAL SPECIFICATION HAS BEEN DROPPED, HOWEVER -- THE IDEA OF SWAPPED PROCESS TABLES. SWAPPING PROCESS TABLES TO DISK ENTAILS TOO MANY INEFFICIENCIES AND COMPLICATIONS TO BE FEASIBLE. THE CHIEF DIFFICULTY IS REFERENCING PROCESS TABLES WHEN SWAPPED. ANY CODE PATH THAT REFERENCES THE PROCESS TABLE OF A PROCESS OTHER THAN THE CALLER POTENTIALLY MAY REFERENCE (AND MODIFY) A SWAPPED PROCESS TABLE. HANDLING QUEUES OF PROCESS TABLES IS ESPECIALLY DIFFICULT AND TIME CONSUMING IF THE PROCESS TABLES ON THE QUEUE CAN BE ON DISK. THUS THE DECISION WAS MADE NOT TO INCLUDE THIS ASPECT IN THE FINAL DESIGN. THIS GREATLY SIMPLIFIES THE IMPLEMENTATION, WILL IMPROVE PERFORMANCE, AND COSTS NOTHING FUNCTIONALLY.

THIS DOCUMENT SPECIFIES CHANGES TO THE SCHEDULER AND THE CORE MANAGER, THEIR ASSOCIATED QUEUES AND DATA BASES. WHILE IT INDICATES AND IMPLIES CHANGES NECESSARY IN OTHER PARTS OF THE SYSTEM (PROCESSING SYSTEM CALLS) IT DOES NOT SPELL OUT THE DETAILS OF THESE CHANGES.

II. GOALS

THE GOAL OF THIS EFFORT IS PRIMARILY TO SOLVE THE DIFFICULTIES AOS HAS WITH THE LIMITED LOGICAL ADDRESS SPACE OF THE ECLIPSE. THIS MANIFESTS ITSELF WITH A FATAL AOS ERROR 14 WHEN TRYING TO RUN A LARGE NUMBER OF PROCESSES. THE DIFFICULTY IS THAT THE LOGICAL ADDRESS SPACE AVAILABLE IS INSUFFICIENT FOR THE NUMBER OF PROCESS TABLES NEEDED. THE RESULT IS THAT AOS CANNOT REALLY SUPPORT 64 PROCESSES AS ADVERTISED.

THE MOST IMPORTANT GOAL OF THIS DESIGN IS TO ACHIEVE AN IMPLEMENTATION OF AOS THAT WILL FULLY SUPPORT 64 PROCESSES IN A WORKING ENVIRONMENT (64 PROCESSES CAN BE SUPPORTED CURRENTLY, BUT USUALLY ONLY UNDER SPECIAL CONDITIONS). A HIGHLY DESIRABLE EXTENSION OF THIS GOAL IS TO INCREASE THE NUMBER OF PROCESSES THAT CAN BE SUPPORTED TO 128, OR EVEN POSSIBLY 256. THERE IS A POTENTIAL TRADE-OFF HERE WITH PERFORMANCE, BUT IDEALLY TWICE AS MANY ACTIVE PROCESSES WOULD STILL BE ABLE TO RUN, THOUGH PERHAPS WITH ONLY HALF THE RESPONSE TIME.

THE SECOND PRIMARY GOAL OF THIS PROJECT IS RELIEVING THE PERFORMANCE BOTTLENECK AOS ENCOUNTERS WHEN LARGE NUMBERS OF PROCESSES ARE ACTIVE ON LARGE MEMORY SYSTEMS. THIS PROBLEM IS DUE TO THE FACT THAT AOS TRIES TO FIT AS MANY PROCESSES INTO MAIN MEMORY AS POSSIBLE, WITH THE RESULT THAT MANY PROCESSES ARE ON THE SCHEDULER'S ELIGIBLE QUEUE. THIS GREATLY INCREASES SCHEDULER OVERHEAD AS THE SYSTEM SPENDS MUCH OF ITS TIME DECIDING WHAT TO DO. RESPONSE AND PERFORMANCE SHOULD GRADUALLY DEGRADE AS LOAD INCREASES, RATHER THAN DROPPING OFF SUDDENLY WHEN A CERTAIN THRESHOLD IS REACHED.

III. INTERNAL STRUCTURE

THE MAJOR INTERNAL CHANGE TO AOS IS THAT PROCESS TABLES WILL NO LONGER BE ALLOCATED IN GSMEM SPACE. INSTEAD, PROCESS TABLES WILL BE ALLOCATED IN MAPPED MEMORY, HENCE THE NAME VIRTUAL PROCESS TABLE.

IN ORDER TO BE RUN BY THE SCHEDULER, A PROCESS WILL HAVE TO BE BOUND TO A VIRTUAL PROCESSOR. CONCEPTUALLY THE SCHEDULER MERELY CHOOSES A VIRTUAL PROCESSOR TO BE RUN ON THE PHYSICAL PROCESSOR. PRAGMATICALLY, A VIRTUAL PROCESSOR CAN BE THOUGHT OF AS A RESIDENT PROCESS TABLE. THUS VIRTUAL PROCESS TABLES WILL BE BOUND TO VIRTUAL PROCESSORS BY ALLOCATING A RESIDENT PROCESS TABLE TO THE PROCESS. BY CONTROLLING THE NUMBER OF VIRTUAL PROCESSORS (RESIDENT PROCESS TABLES) CONTENTION FOR THE PROCESSOR CAN BE REDUCED.

A POOL OF VIRTUAL PROCESSORS WILL BE ALLOCATED DURING SYSTEM INITIALIZATION. THE NUMBER OF VIRTUAL PROCESSORS CAN BE A FUNCTION OF MANY VARIABLES, BOTH DYNAMIC AND STATIC, E.G. MEMORY SIZE, NUMBER AND TYPE OF PROCESSES ON THE SYSTEM, ETC. NOTE THIS POOL CAN BE GROWN AND SHRUNK DYNAMICALLY TO MEET CHANGES IN THE PROCESS ENVIRONMENT. INITIALLY, A SIMPLE SCHEME THAT BASES THE NUMBER OF VIRTUAL PROCESSORS ON THE MEMORY SIZE OF THE SYSTEM AND DOES NOT DYNAMICALLY ALTER THIS NUMBER WILL BE IMPLEMENTED. THIS IS PRIMARILY FOR EASE OF INITIAL IMPLEMETATION, AND BECAUSE OF THE DESIGN PRINCIPLE "KISS"- KEEP IT SIMPLE, STUPID. ONCE SOME EXPERIENCE WITH THE PERFORMANCE OF AOS WITH VIRTUAL PROCESS TABLES IS GAINED, EVALUATION OF THIS ALGORITHM AND EXPERIMENTS WITH OTHER METHODS INCLUDING THOSE THAT DYNAMICALLY VARY THE NUMBER OF VIRTUAL PROCESSORS SHOULD BE DONE.

THE SCHEDULER WILL DEAL ONLY WITH VIRTUAL PROCESSORS, I.E. PROCESS TABLES THAT HAVE BEEN BOUND TO RESIDENT PROCESS TABLES. ALL OTHER PROCESS QUEUES, HOWEVER, WILL BE QUEUES OF VIRTUAL PROCESS TABLES. A NEW QUEUE, THE VIRTUAL ELIGIBLE QUEUE (VELQUE) WILL BE CREATED. IT WILL CONSIST OF THOSE PROCESSES THAT HAVE BEEN ALLOCATED MAIN MEMORY (I.E. ARE "ELIGIBLE" BY THE CURRENT AOS DEFINITION) BUT HAVE NOT BEEN BOUND TO A VIRTUAL PROCESSOR. THE VIRTUAL ELIGIBLE QUEUE WILL ACT AS THE ELIGIBLE QUEUE DOES NOW WITH RESPECT TO SWAPPING OPERATIONS; THAT IS THE CORE MANAGER WILL SELECT PROCESSES FROM IT TO BE SWAPPED OUT, AND WILL PUT PROCESSES ON VELQUE FROM THE INELIGIBLE QUEUES WHEN THEY HAVE BEEN SWAPPED IN.

MOVEMENT FROM THE VIRTUAL ELIGIBLE QUEUE TO THE ELIGIBLE QUEUE WILL BE CONTROLLED BY THE SCHEDULER. ESSENTIALLY, WHEN THE SCHEDULER PUTS A PROCESS ON VELQUE, IT MEANS AOS HAS THE MEMORY RESOURCE TO RUN THE PROCESS BUT NOT THE PROCESSOR RESOURCE. THERE ARE MANY INSTANCES CURRENTY WHEN A PROCESS IS ON THE ELIGIBLE QUEUE AND THUS BEING LOOKED AT BY THE SCHEDULER, BUT CANNOT BE RUN. ALL SUCH INSTANCES ARE CASES WHERE THE PROCESS COULD EASILY BE PUT ON THE VIRTUAL ELIGIBLE QUEUE. NOTE HOWEVER, THAT FOR EFFICIENCY REASONS WE WILL NEVER PUT A PROCESS ON VELQUE UNLESS THERE IS A PROCESS ON VELQUE THAT CAN BE MOVED TO ELQUE. THIS SIMPLY MEANS THAT IF THERE ARE FEWER ACTIVE PROCESSES THAN VIRTUAL PROCESSORS, THOSE PROCESSES WILL REMAIN ON ELQUE EVEN IF THEY COULD BE MADE VIRTUAL.

PROCESSES WILL BE MOVED BETWEEN VELQUE AND ELQUE AT THE END OF A FULL TIME SLICE. IN DECIDING WHICH PROCESS TO REMOVE FROM ELQUE, THE DECIDING FACTOR WILL BE THE AMOUNT OF CPU TIME THE PROCESS HAS ACCUMULATED SINCE BEING PLACED ON ELQUE. TO ACCOMPLISH THIS A COUNT OF THE NUMBER OF SUB-SLICES USED WILL BE KEPT IN THE PROCESS TABLE, AND ZEROED EACH TIME A PROCESS IS BOUND TO A VIRTUAL PROCESSOR. THIS METHOD ENSURES THAT EACH PROCESS THAT WANTS TO RUN WILL GET A TURN. IF PRIORITY WERE USED, THE ALGORITHM WOULD HAVE TO BE VERY CAREFUL THAT THE SAME PROCESSES WERE NOT REPEATEDLY PUT ON ELQUE AND THEN TAKEN OFF AGAIN WITHOUT ACCUMULATING ANY EXECUTION TIME.

THE MAJOR IMPLICATIONS OF THESE CHANGES ARE THAT CODE PATHS THAT REFERENCE PROCESSES BY USING THE PROCESS TABLE ADDRESS MUST BE CHANGED, BECAUSE THE PROCESS TABLE ADDRESS WILL NO LONGER REMAIN CONSTANT SINCE THE PROCESS CAN BE BOUND TO DIFFERENT VIRTUAL PROCESSORS AT DIFFERENT TIMES. INSTEAD, ALL REFERENCES TO PROCESSES (SUCH AS THE POINTER TO THE FATHER PROCESS) MUST BE CHANGED TO BE DONE USING THE PROCESS IDENTIFIER (PID). ALL QUEUES OF PROCESSES SUCH AS THE DEALY CHAIN, THE HISTOGRAM CHAIN, ETC. AS WELL AS THE VARIOUS SCHEDULER QUEUES MUST BE MADE VIRTUAL (WITH THE EXCEPTION OF COURSE, OF ELQUE). THE FOLLOWING SECTIONS DETAIL THE CHANGES NECESSARY.

ALL NUMBERS USED IN THE REMAINDER OF THIS SPECIFICATION ARE OCTAL, UNLESS FOLLOWED BY A DECIMAL POINT WHICH INDICATES THE NUMBER IS DECIMAL.

NEW DATA BASES

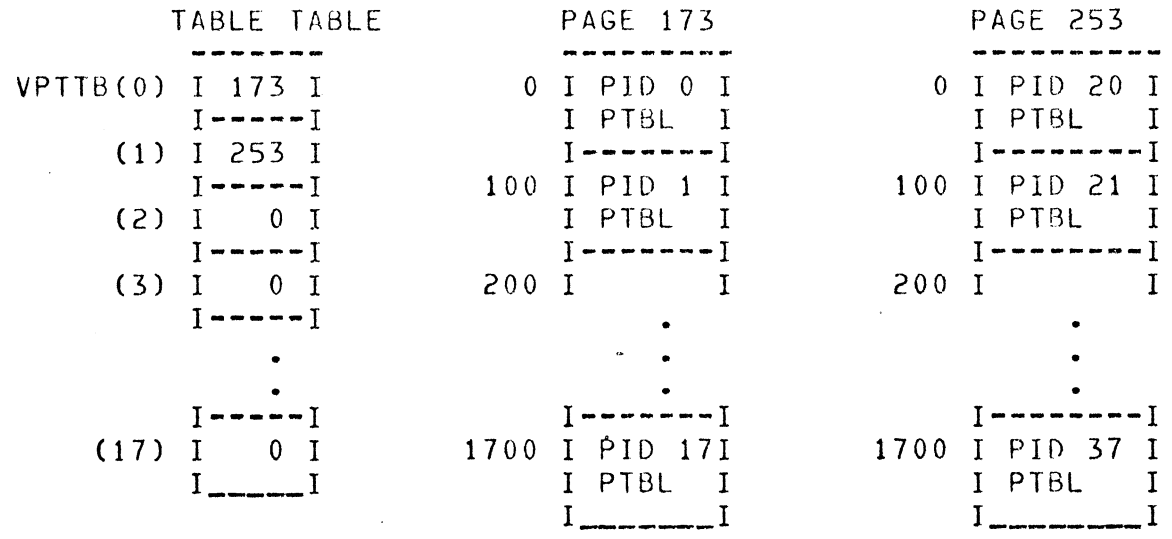
1. VIRTUAL PROCESS TABLE STRUCTURE

VPTTB - THE VIRTUAL PROCESS TABLE TABLE

VPTTB IS A DATA BASE DEFINED IN STABLE THAT RECORDS THE MAPPED MEMORY USED TO HOLD THE VIRTUAL PROCESS TABLES. IT WILL CONTAIN ONE ENTRY FOR EACH PHYSICAL PAGE OF MEMORY ALLOCATED TO HOLD VIRTUAL PROCESS TABLES. NOTE THAT SINCE THE RESIDENT PORTION OF THE PROCESS TABLE IS LESS THAN 64. WORDS (100 OCTAL), 16. VIRTUAL PROCESS TABLES WILL FIT IN A 1 K PAGE OF MEMORY (64. WORDS PER PROCESS TABLE * 16. PROCESS TABLES = 1024. WORDS). SINCE AOS LIMITS THE NUMBER OF PIDS TO 256. (RANGING FROM 0 TO 255.), THE MAXIMUM LENGTH OF VPPTB IS 16. WORDS (ASSUMING WE CHOOSE TO IMPLEMENT 256. PROCESSES).

THUS THE STRUCTURE OF VPPTB AND THE VIRTUAL PROCESS TABLES IS AS FOLLOWS:

VPTTB(I) = PHYSICAL PAGE OF MEMORY ALLOCATED TO HOLD THE PROCESS TABLES FOR PROCESSES WITH PIDS IN RANGE 16. * I <= PID < 16. * (I+1)



NOTE: LET N = INTEGER QUOTIENT OF PID DIVIDED BY 16.
LET M = REMAINDER,
I.E. PID = 16. * N + M
THEN:

VPTTB (N) CONTAINS PHYSICAL MEMORY PAGE HOLDING VIRTUAL PROCESS TABLE FOR THE PROCESS

100 * PID = BEGINNING OFFSET OF THE PROCESS TABLE WITHIN THAT PAGE

PAGES WILL BE ALLOCATED TO HOLD VIRTUAL PROCESS TABLES ONLY AS NEEDED. PROC2 CURRENTLY EXPANDS PIDTB AND PIDBT WHEN NECESSARY; THE SAME CODE CAN ALLOCATE ANOTHER PAGE TO VIRTUAL PROCESS TABLES AND FILL IN VPTTB ACCORDINGLY. SINCE INITIALLY PIDLN = 16., WHICH IS USED TO DETERMINE THE SIZE OF PIDBT AND PIDTB, ONE PAGE WILL BE ALLOCATED AND ONE ENTRY IN VPTTB FILLED IN. THIS WILL BE DONE BY SINIT. NOTE THIS MEANS VPTTB (I) = 0 IF 16. * I > VALUE OF PIDLN

CHANGES TO PIDTB

PIDTB IS THE PROCESS-IDENTIFIER-TO-PROCESS-TABLE-ADDRESS CONVERSION TABLE. CURRENTLY, GIVEN A PID:

PIDTB (PID) = 0 IF THERE IS NO CURRENTLY EXISTING PROCESS WITH THAT PROCESS IDENTIFIER

OTHERWISE,

PIDTB (PID) = ADDRESS OF PROCESS TABLE OF THAT PROCESS

WITH VIRTUAL PROCESS TABLES, THIS MUST BE MODIFIED, SINCE ANY PROCESS NOT BOUND TO A VIRTUAL PROCESSOR WILL NOT HAVE A RESIDENT PROCESS TABLE ADDRESS. HENCE PIDTB WILL BE RE-DEFINED AS FOLLOWS:

PIDTB (PID) = 0 IF NO PROCESS EXISTS WITH THAT PID

PIDTB (PID) = 100000 IF THE PROCESS WITH THAT PID EXISTS BUT IS NOT BOUND TO A VIRTUAL PROCESSOR (I.E. HAS A VIRTUAL PROCESS TABLE ONLY)

PIDTB (PID) = VIRTUAL PROCESSOR ADDRESS IF THAT PID EXISTS AND HAS BEEN BOUND TO A VIRTUAL PROCESSOR (I.E. HAS BEEN ALLOCATED A RESIDENT PROCESS TABLE).

THUS, IF PIDTB (PID) = 100000, THE PROCESS TABLE CAN BE ACCESSED BY USING VPTTB AS DESCRIBED ABOVE TO FIND THE CORRECT PHYSICAL PAGE AND MAPPING IT.

THE VIRTUAL PROCESSOR POOL

RATHER THAN MAKE A CALL TO ALLOCATE MEMORY FOR A VIRTUAL PROCESSOR EACH TIME THE SCHEDULER WANTS TO BIND A PROCESS TO A VIRTUAL PROCESSOR, A POOL OF PRE-ALLOCATED VIRTUAL PROCESSORS WILL BE MAINTAINED. THEN ALLOCATING A VIRTUAL PROCESSOR SIMPLY BECOMES PICKING A FREE ONE FROM THIS POOL. SINIT WILL CALL GSMEM DURING INITIALIZATION AND SET UP A CHAIN OF VIRTUAL PROCESSORS (I.E. OF RESIDENT PROCESS TABLES).

THREE NEW ENTRIES WILL BE DEFINED IN STABLE TO IMPLEMENT THE VIRTUAL PROCESSOR POOL:

VPCNT -- VIRTUAL PROCESSOR COUNT

THIS IS THE NUMBER OF VIRTUAL PROCESSORS IN THE SYSTEM. (BOTH FREE AND IN USE). NOTE THIS IS A POTENTIALLY CRITICAL TUNING PARAMETER. THE INITIAL IMPLEMENTATION WILL MAKE THE VALUE OF VPCNT A SIMPLE LINEAR FUNCTION OF MEMORY SIZE. CLEARY PERFORMANCE MEASUREMENT AND EVAULATION IS NECESSARY ON THIS VALUE. THIS PARAMETER COULD EASILY BE DYNAMICALLY ALTERED BY THE SYSTEM BASED ON LOAD CHARACTERISTICS OR OTHER ENVIRONMENTAL CONDITIONS.

VPFCN -- VIRTUAL PROCESSOR FREE COUNT

THIS IS THE NUMBER OF FREE (UNBOUND) VIRTUAL PROCESSORS, I.E. A COUNT OF THE NUMBER OF ENTRIES ON VPCHN. STRICTLY SPEAKING, IT IS NOT NECESSARY, BUT THIS NUMBER MAY BE USEFUL AS INPUT TO POLICY ALGORITHMS, AND CERTAINLY WILL BE USEFUL FOR PERFORMANCE MONITORING.

VPCHN -- THE VIRTUAL PROCESSOR CHAIN

THIS IS A POINTER TO THE FIRST VIRTUAL PROCESSOR IN THE CHAIN OF FREE VIRTUAL PROCESSORS. THIS CHAIN IS A SINGLY LINKED LIST, WITH OFFSET PLNK IN EACH POINTING TO THE NEXT VIRTUAL PROCESSOR. A NULL POINTER (-1) IN VPCHN DENOTES THE CHAIN IS EMPTY, AND IS ALSO USED TO TERMINATE THE CHAIN.

NOTE THAT AS A CONSISTENCY CHECK, VPCHN = -1 IF AND ONLY IF VPFCN = 0.

VELQUE - THE VIRTUAL ELIGIBLE QUEUE

VELQUE WILL BE A LOCATION DEFINED IN STABLE CONTAINING THE PID OF THE FIRST PROCESS ON THE VIRTUAL ELIGIBLE QUEUE. ALL PROCESSES ON VELQUE ARE "ELIGIBLE" AS TRADITIONALLY DEFINED BY AOS; THAT IS THEIR PROCESS IMAGES ARE CORE RESIDENT. (MORE SIMPLY PUT, BIT PFELG = 1). THEY ARE ALL CANDIDATES TO BE BOUND TO A VIRTUAL PROCESSOR AND PLACED ON ELQUE, BUT CONVERSELY THEY ARE ALSO CANDIDATES FOR SWAPPING (UNLESS THEIR PROCESS TYPE IS RESIDENT - THIS DESIGN DOES ALLOW RESIDENT TYPE PROCESSES TO BE UNBOUND FROM VIRTUAL PROCESSORS).

AS IS ELQUE, VELQUE WILL BE PRIORITIZED BY PNQF. LIKE THE OTHER SCHEDULER PROCESS QUEUES, VELQUE WILL BE DOUBLY LINKED, USING OFFSETS PLNK AND PBLNK, WHICH WILL CONTAIN THE PID OF THE NEXT AND THE PREVIOUS PROCESSES ON THE QUEUE, RESPECTIVELY. THE FORWARD LINK WILL TERMINATE WITH -1, THE BACK LINK WITH 1B0 + ADDRESS OF VELQUE.

THUS IMPLEMENTING THE VIRTUAL ELIGIBLE QUEUE WILL REQUIRE TWO ENTRIES IN STABLE:

VELQUE - PID OF FIRST PROCESS ON VIRTUAL ELIGIBLE QUEUE (-1 IF QUEUE EMPTY)

VELCN - VIRTUAL ELIGIBLE QUEUE COUNT CONTAINS THE NUMBER OF PROCESSES ON VELQUE. THIS IS MAINTAINED PRIMARILY FOR PERFORMANCE MONITORING, THOUGH IT COULD BE USED AS A MEASURE OF SYSTEM LOAD.

SWPVP - THE SWAPPING VIRTUAL PROCESSOR

SWPVP IS A VIRTUAL PROCESSOR THAT THE CORE MANAGER CAN USE TO BIND THE VIRTUAL PROCESS TABLE OF A PROCESS BEING SWAPPED IN OR OUT. ONE IMPLICATION OF VIRTUAL PROCESS TABLES IS THAT ASSIGNING A RESIDENT PROCESS TABLE TO A PROCESS IS INDEPENDENT OF ALLOCATING/DEALLOCATING MEMORY TO THE PROCESS AND SWAPPING IT IN/OUT. THIS MEANS THE CORE MANAGER MUST DEAL WITH VIRTUAL PROCESS TABLES. A DIFFICULTY ARISES FROM THE MANNER IN WHICH SWAP I/O IS PERFORMED THAT MAKES IT NECESSARY TO HAVE A RESIDENT PROCESS TABLE FOR A PROCESS BEING SWAPPED. THE FOLLOWING EXPLANATION SHOULD MAKE THIS CLEAR.

FOR EFFICIENCY REASONS, SWAP I/O IS DONE USING THE SYSTEM PRIMITIVE NQBHR -- ENQUEUE BUFFER HEADER. THIS ALLOWS THE ENTIRE PROCESS IMAGE TO BE READ/WRITTEN IN A SINGLE DISK REQUEST. HOWEVER, ONE OF THE OFFSETS IN THE BUFFER HEADER THAT IS ENQUEUED, BQMAP, MUST CONTAIN EITHER THE PHYSICAL PAGE NUMBER THE I/O IS TO, OR THE PROCESS TABLE ADDRESS OF THE PROCESS THE REQUEST IS FOR. IN THE CASE OF SWAPS, BQMAP MUST BE A PROCESS TABLE ADDRESS, SO THAT THE DATA CHANNEL MAPPING CODE (SWAMP) CAN ACCESS THE PROCESS MAP AND PERFORM THE READ/WRITE TO/FROM THE CORRECT MEMORY LOCATIONS. BQMAP IS NOT A PROBLEM FOR OTHER TYPES OF I/O, BECAUSE IN ALL CASES EXCEPT SWAP I/O THE PROCESS WILL HAVE TO BE BOUND TO A VIRTUAL PROCESSOR, AND THUS A PROCESS TABLE ADDRESS CAN BE USED.

SWPVP WILL THUS BE A DUMMY PROCESS TABLE THAT CAN BE USED TO FULFILL THE REQUIREMENTS OF BQMAP. THE CORE MANAGER WILL BIND THE SWAPPING PROCESS TO SWPVP BEFORE CALLING SWAPI OR SWAPO TO SWAP THE PROCESS, AND WILL UNBIND THE PROCESS AFTER THE SWAP IS COMPLETE. SWPVP IS DEFINED IN STKS, AS IS THE CORE MANAGER STACK CMSTK.

CHANGES TO EXISTING QUEUES

THE DESCRIPTION OF VELQUE NOTED THAT THE VIRTUAL ELIGIBLE QUEUE WILL USE PIDS AS FORWARD AND BACKWARD LINKS. ALL OTHER PROCESS TABLE QUEUES IN AOS WITH THE EXCEPTION OF THE ELIGIBLE QUEUE (WHICH REMAINS UNCHANGED) MUST BE CHANGED IN A SIMILAR MANNER. PROCESS TABLE ADDRESSES CANNOT BE USED SINCE THEY WILL CHANGE EACH TIME THE PROCESS IS BOUND TO A DIFFERENT VIRTUAL PROCESSOR. THE QUEUES AFFECTED, AND A SUMMARY OF THE IMPLICATIONS OF THE CHANGE, ARE LISTED BELOW.

IESWP - THE INELIGIBLE SWAPPED QUEUE

QUEUE OF VIRTUAL PROCESS TABLES OF SWAPPED OUT PROCESSES. NOTE BLOCKED, SWAPPED PROCESSES WILL BE ON BLKQ, NOT IESWP.

IERES - THE INELIGIBLE RESIDENT QUEUE

THIS QUEUE LINKS THE VIRTUAL PROCESS TABLES OF ANY RESIDENT OR PRE-EMPTIBLE TYPE PROCESSES THAT HAVE NOT BEEN ALLOCATED MEMORY. RESIDENT PROCESSES CAN ONLY BE ON THIS QUEUE WHEN BEING CREATED, OR IF THEY HAVE JUST HAD THEIR TYPE CHANGED TO RESIDENT. PRE-EMPTIBLE PROCESSES THAT ARE BLOCKED WILL BE ON BLKQ, NOT IERES.

BLKQ - THE BLOCKED QUEUE

ALL PROCESSES THAT HAVE BEEN BLOCKED WILL BE ON THIS QUEUE. NOTE THIS WILL BE A QUEUE OF VIRTUAL PROCESS TABLES, WHICH IMPLIES THE ACT OF BLOCKING A PROCESS SHOULD UNBIND THE PROCESS FROM ITS VIRTUAL PROCESSOR IF IT IS BOUND TO ONE. IT MAY NOT ALWAYS BE POSSIBLE TO UNBIND THE PROCESS (E.G. IF A SYSTEM CALL IS IN PROGRESS), SO THE SCHEDULER MUST BE CHANGED TO TRY AND UNBIND PROCESSES IT FINDS ON ELQUE THAT ARE BLOCKED, AND MOVE THEM TO BLKQ. AS NOW, PROCESSES ON BLKQ MAY STILL HAVE MEMORY ALLOCATED TO THEM.
(NOTE BLEND, WHICH INDICATES THE LAST PROCESS ON BLKQ, MUST ALSO BE CHANGED TO A PID RATHER THAN A PROCESS TABLE ADDRESS.)

DCHN - THE DELAY CHAIN

ALL PROCESSES THAT HAVE OUTSTANDING ?DELAY CALLS ARE LINKED ON THIS CHAIN. NOTE THE LINKS WILL STILL BE PIDS, EVEN THOUGH SOME PROCESSES ON DCHN WILL BE BOUND TO VIRTUAL PROCESSORS AND OTHERS WILL NOT. A SEPARATE DOCUMENT DETAILS THE DESIGN AND IMPLEMENTATION OF THE DELAY FUNCTION.

HISLS - HISTOGRAM QUEUE

HISLS IS A QUEUE OF PROCESSES WITH ACTIVE HISTOGRAM REQUESTS. UNLIKE THE SCHEDULER QUEUES MENTIONED SO FAR, WHICH ARE DOUBLY LINKED VIA PLNK AND PBLNK, THIS QUEUE IS SINGLY LINKED USING OFFSET PHLNK IN THE PROCESS TABLE.

CMQWD - CORE MANAGER'S REQUEST QUEUE

CMQWD IS A CONTROL BLOCK OFFSET THAT ONLY HAS MEANING FOR THE CORE MANAGER CONTROL BLOCK, CMTSK. IT IS THE BEGINNING OF THE CHAIN OF PROCESSES THAT ARE ENQUEUED TO THE CORE MANAGER FOR SWAPPING, EITHER TO BE SWAPPED IN OR SWAPPED OUT. THUS THIS OFFSET WILL NOW CONTAIN A PID RATHER THAN A PROCESS TABLE ADDRESS. NOTE THAT THIS QUEUE IS LINKED THROUGH OFFSET PCMLK IN THE PROCESS TABLE AND IS A SINGLY LINKED LIST.

PROCESS TABLE CHANGES

THE VIRTUAL PROCESS TABLE DESIGN REQUIRES THE MEANING OF THREE OFFSETS IN THE PROCESS TABLE BE CHANGED, THE ADDITION OF TWO NEW STATUS BITS, AND THE ADDITION OF A NEW OFFSET.

1. PDAD, PSON, AND PSONL

   THESE THREE OFFSETS LINK PROCESS TABLES ACCORDING TO THE PROCESS HIERARCHY - PDAD LINKS A PROCESS TO ITS FATHER PROCESS, PSONP TO ONE OF ITS SONS, THE REST OF THE SONS ARE LINKED TO THE SON POINTED TO BY PSONP THROUGH PSONL. THESE THREE CHAINS WILL BE MODIFIED SO THAT PIDS WILL BE USED AS LINKS RATHER THAN PROCESS TABLE ADDRESSES.

2. PFNVT

   THIS NEW STATUS BIT WILL BE DEFINED IN FLAG WORD PFLG4, AND MEANS "DO NOT MAKE THIS PROCESS VIRTUAL". THUS ONLY PROCESSES BOUND TO VIRTUAL PROCESSORS WILL HAVE THIS BIT SET. IF SET, THE PROCESS CAN NOT BE UNBOUND UNTIL THE BIT IS CLEARED. THIS IS NECESSARY, FOR EXAMPLE, SO THAT A PROCESS WITH A SYSTEM CALL IN PROGRESS DOES NOT GET UNBOUND, AS THE PROCESS TABLE ADDRESS IN THE CONTROL BLOCK PROCESSING THE CALL (OFFSET CPTAB) WOULD NO LONGER BE VALID. (NOTE THIS PARTICULAR CASE COULD BE HANDLED BY CHECKING PSQCT, THE PROCESS TABLE OFFSET WITH THE NUMBER OF OUTSTANDING SYSTEM CALLS, BUT IT IS BETTER TO CONSOLIDATE THE SPECIAL CHECKS INTO ONE BIT). NOTE PFNVT MUST BE SET WHENEVER PSQCT IS INCREMENTED, AND MUST BE CLEARED WHENEVER PSQCT IS DECREMENTED TO ZERO. THIS AFFECTS SCPRC AND THE VARIOUS I/O POST PROCESSORS.

   CONDITIONS UNDER WHICH PFNVT IS SET:

   - A SYSTEM CALL IS BEING PROCESSED

   - THE PROCESS IS THE PMGR (THIS IS FOR EFFICIENCY)

3. PFBVP

   THIS NEW STATUS BIT IS ALSO DEFINED IN FLAG WORD PFLG4, AND IF SET INDICATES THE PROCESS IS BOUND TO A VIRTUAL PROCESSOR. THIS IS MOSTLY FOR IMPLEMENTATION EASE, AS THERE ARE CASES WHEN IT IS NECESSARY TO REMOVE A PROCESS FROM A QUEUE, AND THE STATE OF THIS BIT DETERMINES WHETHER THE PROCESS IS ON THE ELIGIBLE QUEUE OR ONE OF THE VIRTUAL QUEUES. THUS PFBVP IS SET IF AND ONLY IF THE PROCESS IS BOUND TO A VIRTUAL PROCESSOR AND ON THE ELIGIBLE QUEUE.

4. PSSEL

THIS NEW OFFSET CONTAINS THE NUMBER OF SUB-SLICES THE PROCESS HAS USED SINCE IT WAS LAST PUT ON THE ELIGIBLE QUEUE, I.E. BOUND TO A VIRTUAL PROCESSOR. THIS IS USED IN DETERMINING WHEN A PROCESS SHOULD BE UNBOUND. PSSEL IS ZEROED EACH TIME THE PROCESS IS BOUND, AND INCREMENTED EACH TIME A SUB-SLICE EXPIRES.

OTHER DATA BASE CHANGES

THE QUEUES PREVIOUSLY DESCRIBED ARE NOT THE ONLY DATA BASES THAT PREVIOUSLY USED PROCESS TABLES ADDRESSES. THIS SECTION LISTS OTHER INTERNAL DATA BASES THAT MUST BE MODIFIED BY REPLACING THE USAGE OF PROCESS TABLE ADDRESSES AS A MEANS OF SPECIFYING A PROCESS WITH THE PID. EXCEPT FOR CHANNEL CONTROL BLOCKS (CCB'S), THE IMPACT OF THESE CHANGES IS SLIGHT.

INFID - INFOS PROCESS IDENTIFIER

THIS STABLE LOCATION CONTAINS THE PROCESS TABLE ADDRESS OF THE INFOS PROCESS, OR -1 IF THERE IS NONE DEFINED. THIS MUST BE CHANGED TO BE A PID.

MODULES AFFECTED: STABLE, SCMOD, CLNUP, SOV11

USER DEVICE TABLE

THIS IS A TABLE DEFINED FOR EACH USER DEVICE THAT HAS BEEN DEFINED VIA AN ?IDEF CALL. OFFSET UIPTB IS DEFINED AS THE PROCESS TABLE ADDRESS OF THE PROCESS MAKING THE ?IDEF CALL. THIS MUST BE CHANGED TO A PID. (THE ALTERNATIVE IS TO KEEP IT AS A PROCESS TABLE ADDRESS, BUT FORCE PROCESSES DOING ?IDEF'S TO ALWAYS BE BOUND TO A VIRTUAL PROCESSOR. THERE IS NO ADVANTAGE TO THIS, EVEN THOUGH THE ?IDEFING PROCESS MUST BE A RESIDENT TYPE PROCESS).

MODULES AFFECTED: SSOV2, INTS

CHANNEL CONTROL BLOCKS

THE CHANNEL CONTROL BLOCK, OR CCB, ALLOCATED EACH TIME A USER PROCESS OPENS A CHANNEL, CONTAINS THE PROCESS TABLE ADDRESS OF THE USER PROCESS IN OFFSET CCPTA. THIS MUST BE CHANGED TO BE A PID. THE IMPLICATIONS OF THIS CHANGE ARE PERVASIVE, AS THIS OFFEST IS REFERENCED IN MANY PLACES AS INDICATED BELOW. IF THIS CHANGES CANNOT BE MADE EASILY AND EFFICIENTLY IN ALL THESE PLACES, IT MAY BE NECESSARY TO RETAIN CBPTA AS A PROCESS TABLE ADDRESS. THIS WOULD FORCE THE UPDATING OF THIS OFFSET IN EVERY OPENED CCB EVERYTIME THE PROCESS WAS BOUND TO A VIRTUAL PROCESSOR, A RATHER EXPENSIVE TASK. MORE INVESTIGATION IS NEEDED TO INSURE PIDS WILL SUFFICE.

MODULES AFFECTED:

BOOT	CLOSE	DE	DIRST
DSKIO	IOOV	IREC	ISEN2
LPBIO	MCAIO	MTAIO	OPEN
SGSUB	SRDB	SSOV2	SSOV6
SSOV7	SOV10	SOV11	SYOV1
SYOV2	UNIT		

SYSTEM PRIMITIVES

HAVING DEFINED THE NEW DATA BASES AND THE MODIFICATIONS TO EXISTING DATA BASES, IT IS CLEAR THAT SOME SOME NEW BASE LEVEL ROUTINES MUST BE DEFINED TO ACT ON THESE DATA BASES. ALREADY EXISTING ROUTINES MAY NEED TO BE MODIFIED OR DELETED. EACH OF THESE CATEGORIES IS LISTED BELOW. FOR THE NEW PRIMITIVES, A FULL DESCRIPTION IS GIVEN USING THE DOCUMENTATION TEMPLATE CURRENTLY EMPLOYED TO DOCUMENT SYSTEM ROUTINES. FOR PRIMITIVES BEING MODIFIED, ONLY THE CHANGES ARE GIVEN AS THESE ROUTINES ARE ALREADY DOCUMENTED.

NEW PRIMTIVE ROUTINES

GPTBL - GET PROCESS TABLE
BVIRP - BIND VIRTUAL PROCESSOR
UVIRP - UNBIND VIRTUAL PROCESSOR
MKEL - SELECT A PROCESS TO BE PUT ON VELQUE
MKINEL- SECECT A PROCESS FOR REMOVAL FROM VELQUE
VPENQ - VIRTUAL PROCESS ENQUEUE ROUTINE
VPDEQ - VIRTUAL PROCESS DEQUEUE ROUTINE
BLNCE - BALANCE ROUTINE

MODIFIED PRIMITIVE ROUTINES

FITER - ENQUEUE TO AN INELIGIBLE QUEUE
CMENQ - ENQUEUE TO THE CORE MANAGER'S QUEUE
CTBLK - BLOCK A PROCESS
IELSC - SCAN INELIGIBLE QUEUE
BSCAN - SCAN BLOCKED QUEUE
RUBLK - UNBLOCK AN ELIGIBLE PROCESS
DQBLQ - DEQUEUE FROM BLKQ
PRPR - SELECT A PROCESS FOR PREEMPTION
TSPRC - PROCESS END OF TIME SLICE
TSUP - PROCESS END OF SUB-SLICE
PRBAG - MARK A PROCESS FOR PREEMPTION

PRIMITIVES NO LONGER NEEDED

HPENQ - ENQUEUE TO HEAD OF PRIORITY GROUP

```
;:::DOC_START

;*****************************************************************
;                       GPTBL
;*****************************************************************

;FUNCTIONAL DESCRIPTION:        GET PROCESS TABLE
;
;       GPTBL IS THE PRIMARY ROUTINE FOR ACCESSING A PROCESS TABLE.
;       GIVEN A PID, IT RETURNS AN ADDRESS THAT MAY BE USED TO
;       REFERENCE THE CORRESPONDING PROCESS TABLE.

;AC0 IN:        PID OF PROCESS WHOSE PROCESS TABLE DESIRED
;AC1 IN:        NOT USED
;AC2 IN:            "
;AC3 IN:            "

;CALLING CONVENTIONS:   JSR@    .GPTBL
;                       <RETURN>

;ASSUMPTIONS:   NONE

;CAUTIONS:      MAY CHANGE CONTENTS OF 62000 MAP SLOT

;ERROR RETURN CONDITIONS:       NONE, NO ERROR RETURN
;                               PANICS IF PID IS INVALID

;AC0 OUT:       UNCHANGED
;AC1 OUT:           "
;  2 OUT:       ADDRESS OF PROCESS TABLE (EITHER A 62000 MAPPED
;               ADDRESS OR A RESIDENT PROCESS TABLE ADDRESS)
;AC3 OUT:       FRAME POINTER

;GLOBAL DATA BASES ACCESSED:
;       PIDTB - PID TO PROCESS TABLE CONVERSION TABLE
;       PIDLN - MAXIMUM LEGAL PID VALUE
;      *CUR62 - CURRENT CONTENTS OF 62000 MAP SLOT
;       VPTTB - VIRTUAL PROCESS TABLE TABLE
;       INTLV - INTERRUPT LEVEL INDICATOR
;       CC - CURRENT CONTROL BLOCK
;      *CONTROL BLOCK: OFFSET CWN62 - CONTENTS OF 62000 SLOT

;ANY DATA CHANGED:
;       ITEMS MARKED WITH A * MAY BE CHANGED

;ROUTINES CALLED:
;       .PNIC           .BLD

;FILENAME:      MAPER.SR

;MISCELLANEOUS:
;       GPTBL OPERATES AS FOLLOWS, WHERE N = PID PASSED IN AC0.
;               IF N > PIDLN OR PIDTB (N) = 0
;                       THEN PANIC
;               IF B0 OF PIDTB(N) = 0,
;                       THEN RETURN PIDTB (N)
;                       ELSE RETURN 62000+100*(N MOD 16.)
;       NOTE IN THE LAST CASE, THE PROCESS TABLE IS VIRTUAL AND
;       MUST BE MAPPED INTO THE 62000 SLOT, UPDATING CUR62.
;       THE PAGE TO MAP IS DETERMINED BY LOOKING AT THE
;       APPROPRIATE ENTRY IN VPTTB, NAMELY VPTTB(N/16.).
;       IF THE CALL IS NOT FROM THE INTERRUPT WORLD, AND A
```

```
;       BLOCK MUST BE UPDATED SO THE PROPER CONTEXT IS RESTORED
;       WHENEVER THE CONTROL BLOCK RUNS.

;DOCUMENTED BY: ANDY HUBER

;DOCUMENTATION DATE:     FEBRUARY 4, 1978

;:::DOC_END
```

```
;;;;DOC_START

;*****************************************************************
                        BVIRP
;*****************************************************************

;FUNCTIONAL DESCRIPTION:         BIND VIRTUAL PROCESSOR
;
;       GIVEN A PID AND A VIRTUAL PROCESSOR, BIND THE PROCESS
;       INDICATED BY THE PID TO THE VIRTUAL PROCESSOR

;AC0 IN:         PID
;AC1 IN:         NOT USED
;AC2 IN:         VIRTUAL PROCESSOR ADDRESS
;AC3 IN:         NOT USED

;CALLING CONVENTIONS:    JSR     BVIRP
;                        <RETURN>

;ASSUMPTIONS:    PID IS VALID

;CAUTIONS:       USES 64000 AND 66000 SLOTS
;                DISABLES, THEN RE-ENABLES INTERRUPTS

;ERROR RETURN CONDITIONS:        NONE

;AC0 OUT:        UNCHANGED
;AC1 OUT:            "
;AC2 OUT:            "
;AC3 OUT:        FRAME POINTER

;GLOBAL DATA BASES ACCESSED:
;       PROCESS TABLE OFFSETS:
;               *PFLG4 - FLAG WORD 4, BIT *PFBVP
;               *PSSEL - # OF SUB-SLICES RUN SINCE ON ELQUE
;       PROCESS TABLE EXTENDER OFFSETS:
;               *PRES - ADDR. OF RESIDENT PART OF PROCESS TABLE
;      *PIDTB - PID TO PROCESS TABLE CONVERSION TABLE
;      *CUR64 - CURRENT CONTENTS OF 64000 SLOT
;      *VIRTUAL PROCESSOR
;       VPTTB - VIRTUAL PROCESS TABLE TABLE

;ANY DATA CHANGED:       ITEMS MARKED WITH * ABOVE ARE CHANGED

;ROUTINES CALLED:
;       .BLD            .MP66

;FILENAME:       SCHED.SR

;MISCELLANEOUS:
;       BVIRP COPIES THE VIRTUAL PROCESS TABLE OF THE SPECIFIED
;       PROCESS TO THE VIRTUAL PROCESSOR INDICATED IN AC2.  IT
;       UPDATES PIDTB TO INDICATE THE PROCESS IS NOW BOUND. THE
;       64000 SLOT IS USED TO MAP THE VIRTUAL PROCESS TABLE.
;       PROCESS TABLE OFFSET PSSEL, THE NUMBER OF SUB-SLICES THE
;       PROCESS HAS RUN SINCE BEING PUT ON ELQUE, IS ZEROED.
;       PROCESS TABLE EXTENDER OFFSET PRES, THE ADDRESS OF
;       THE RESIDENT PORTION OF THE PROCESS TABLE, IS SET TO
;       THE RESIDENT PROCESS TABLE ADDRESS PASSED IN AC2.
;       NOTE THE ORDER OF OPERATIONS IS IMPORTANT SO THAT REF-
;       ERENCES TO THE PROCESS TABLE BY THE INTERRUPT WORLD
;       FIND AND REFERENCE THE CORRECT PROCESS TABLE.  THIS ORDER
```

```
;                   1. FIND AND MAP THE VIRTUAL PROCESS TABLE
;                   2. DISABLE INTERRUPTS
;                   3. COPY VIRTUAL PROCESS TABLE TO VIRTUAL
                       PROCESSOR
;                   4. CHANGE PIDTB TO INDICATE THE PROCESS IS
;                      NOW BOUND TO A VIRTUAL PROCESSOR AND TURN
;                      ON "BOUND TO VIRTUAL PROCESSOR" BIT, PFBVP
;                   5. ENABLE INTERRUPTS

;DOCUMENTED BY: ANDY HUBER

;DOCUMENTATION DATE:    FEBRUARY 5, 1978

;:::DOC_END
```

```
;:::DOC_START

;*****************************************************************
;                       UVIRP
; *****************************************************************

;FUNCTIONAL DESCRIPTION:        UNBIND VIRTUAL PROCESSOR
;
;       UVIRP UNBINDS A SPECIFIED PROCESS FROM ITS VIRTUAL
;       PROCESSOR

;AC0 IN:         PID OF PROCESS TO BE UNBOUND
;AC1 IN:         NOT USED
;AC2 IN:             "
;AC3 IN:             "

;CALLING CONVENTIONS:   JSR     UVIRP
;                       <RETURN>

;ASSUMPTIONS:   THE PROCESS IS BOUND TO A VIRTUAL PROCESSOR

;CAUTIONS:      USES THE 64000 MAP SLOT
;               DISABLES AND THEN RE-ENABLES INTERRUPTS

;ERROR RETURN CONDITIONS:       NONE

;AC0 OUT:        UNCHANGED
;AC1 OUT:            "
;AC2 OUT:            "
;  3 OUT:        FRAME POINTER

;GLOBAL DATA BASES ACCESSED:
;       *CUR64 - CURRENT CONTENTS OF 64000 SLOT
;       *VPTTB - VIRTUAL PROCESS TABLE TABLE
;       *VIRTUAL PROCESS TABLE OF SPECIFIED PROCESS
;       *PROCESS TABLE OFFSETS:
;               *PFLG4 - FLAG WORD 4, BIT *PFBVP
;       *PIDTB - PID TO PROCESS TABLE CONVERSION TABLE

;ANY DATA CHANGED:      ITEMS MARKED WITH A * ABOVE

;ROUTINES CALLED:
;       .BLD

;FILENAME:      SCHED.SR

;MISCELLANEOUS:
;       UVIRP IS THE INVERSE OF BVIRP, SEVERING THE CONNECTION
;       BETWEEN A PROCESS AND A VIRTUAL PROCESSOR. IT COPIES
;       THE PROCESS TABLE INFO FROM THE VIRTUAL PROCESSOR TO
;       THE VIRTUAL PROCESS TABLE OF THE SPECIFIED PROCESS,
;       AND UPDATES PIDTB TO SHOW THE PROCESS IS NOW UNBOUND.
;       LIKE BVIRP, THE ORDER OF OPERATIONS IS IMPORTANT SO
;       THAT INTERRUPT WORLD REFERENCES TO THE PROCESS TABLE
;       ARE MADE TO THE MOST RECENT COPY OF THE PROCESS TABLE.
;       THUS THE ORDERING OF OPERATIONS MUST BE:
;               1. DETERMINE AND MAP THE VIRTUAL PROCESS
;                  TABLE ADDRESS
;               2. DISABLE INTERRUPTS
;               3. COPY PROCESS TABLE FROM VIRTUAL PROCESSOR
;                  TO THE VIRTUAL PROCESS TABLE
;               4. UPDATE VPTTB TO SHOW PROCESS IS UNBOUND
```

```
;                   BIT, PFBVP
;                5. ENABLE INTERRUPTS

;  CUMENTED BY: ANDY HUBER

;DOCUMENTATION DATE:     FEBRUARY 5, 1978

;:::DOC_END
```

```
;:::DOC_START

;*****************************************************************
                        MKEL
;*****************************************************************

;FUNCTIONAL DESCRIPTION:         MAKE A PROCESS ELIGIBLE
;
;       THIS ROUTINE SELECTS A PROCESS FROM VELQUE TO BE
;       MOVED TO ELQUE, I.E. TO BE BOUND TO VIRTUAL PROCESSOR

;AC0 IN:         NOT USED
;AC1 IN:             "
;AC2 IN:             "
;AC3 IN:             "

;CALLING CONVENTIONS:    JSR     MKEL
;                        <ERROR RETURN>
;                        <GOOD RETURN>

;ASSUMPTIONS:    NONE

;CAUTIONS:       USES THE 62000 MAP SLOT

;ERROR RETURN CONDITIONS:        NO PROCESS CAN BE FOUND

;AC0 OUT:        UNCHANGED
;AC1 OUT:            "
;AC2 OUT:        PID OF SELECTED PROCESS (IF GOOD RETURN)
;AC3 OUT:        FRAME POINTER

;GLOBAL DATA BASES ACCESSED:
;       PROCESS TABLE OFFSETS:
;               PSTAT - STATUS WORD BITS PSRDY, PSEW
;               PLNK  - FORWARD LINK WORD
;               PID   - PROCESS IDENTIFIER
;      *CUR62 - CURRENT CONTENTS OF 62000 SLOT
;       VELQUE - VIRTUAL ELIGIBLE QUEUE

;ANY DATA CHANGED:       ITEMS MARKED WITH A * ABOVE

;ROUTINES CALLED:
;       MPPID           GPTBL

;FILENAME:       SCHED.SR

;MISCELLANEOUS:
;       MKEL IMPLEMENTS THE POLICY FOR CHOOSING A PROCESS TO
;       BE MADE RUNNABLE BY SELECTING A PROCESS TO BE BOUND
;       FROM THE VIRTUAL ELIGIBLE QUEUE.  IT RETURNS THE FIRST
;       PROCESS ON VELQUE THAT IS READY TO RUN, I.E. COULD
;       ACTUALLY RUN IF ON ELQUE.  IF NO SUCH PROCESS EXISTS,
;       I.E. ALL PROCESSES ON VELQUE HAVE SOME STATUS BIT SET
;       THAT WOULD CAUSE THE SCHEDULER NOT TO RUN THEM, THE
;       FIRST PROCESS ON VELQUE IS RETURNED.  THUS THE ERROR
;       RETURN IS TAKEN CURRENTLY ONLY IF VELQUE IS EMPTY.

;DOCUMENTED BY: ANDY HUBER

;DOCUMENTATION DATE:     FEBRUARY 5, 1976

;:::DOC_END
```

```
;:::DOC_START

;*****************************************************************
                        MKINEL
;*****************************************************************

;FUNCTIONAL DESCRIPTION:         SELECT A PROCESS FOR REMOVAL FROM ELQUE
;
;       THIS ROUTINE CHOOSES A PROCESS FROM THE ELIGIBLE
;       QUEUE TO BE UNBOUND FROM ITS VIRTUAL PROCESSOR
;       AND HENCE MOVED TO VELQUE

;AC0 IN:         NOT USED
;AC1 IN:              "
;AC2 IN:              "
;AC3 IN:              "

;CALLING CONVENTIONS:    JSR     MKINEL
;                        <ERROR RETURN>
;                        <GOOD RETURN>

;ASSUMPTIONS:    NONE

;CAUTIONS:       NONE

;ERROR RETURN CONDITIONS:        NO PROCESS CAN BE UNBOUND

;AC0 OUT:        UNCHANGED
;AC1 OUT:             "
;AC2 OUT:        VIRTUAL PROCESSOR TO BE UNBOUND
;AC3 OUT:        FRAME POINTER

;GLOBAL DATA BASES ACCESSED:
;       ELQUE - THE ELIGIBLE QUEUE
;       PROCESS TABLES ON ELQUE, OFFSETS:
;               PSTAT - STATUS WORD BITS PSRDY, PSEW
;               PLNK  - FORWARD LINK WORD
;               PSSEL - # OF SUB-SLICES RUN SINCE ON ELQUE
;               PNQF  - PRIORITY ENQUEUE FACTOR

;ANY DATA CHANGED:       NONE

;ROUTINES CALLED:        NONE

;FILENAME:       SCHED.SR

;MISCELLANEOUS:
;       MKINEL IMPLEMENTS THE POLICY FOR DECIDING WHEN TO
;       MAKE A PROCESS NOT RUNNABLE BY UNBINDING IT FROM
;       ITS VIRTUAL PROCESSOR AND REMOVING IT FROM ELQUE.
;       IT SELECTS FOR REMOVAL THE FIRST PROCESS ON ELQUE
;       THAT CAN BE MADE VIRTUAL, I.E. DOES NOT HAVE THE
;       "DO NOT MAKE VIRTUAL" BIT SET (PFNVT), AND IS NOT
;       READY TO RUN BECAUSE SOME STATUS BIT IS SET THAT
;       WOULD CAUSE THE SCHEDULER TO SKIP OVER THE PROCESS.
;       IF NO SUCH PROCESS CAN BE FOUND, THEN THE PROCESS
;       THAT CAN BE MADE VIRTUAL (PFNVT=0) WHOSE # OF SUB-
;       SLICES USED SINCE LAST MADE ELIGIBLE (PSSEL) IS GREATEST
;       IS CHOSEN.  IF NO PROCESS CAN BE MADE VIRTUAL CURRENTLY,
;       THE ERROR RETURN IS TAKEN.  THIS COULD HAPPEN, FOR
;       EXAMPLE, IF ALL PROCESSES ON ELQUE HAVE SYSTEM
```

```
;       BLOCK FOR THE CALL HAS THE VIRTUAL PROCESSOR ADDRESS
;       (RESIDENT PROCESS TABLE ADDRESS) IN OFFSET CPTAD.

;' 'CUMENTED BY: ANDY HUBER

;DOCUMENTATION DATE:    FEBRUARY 5, 1978

;:::DOC_END
```

```
;:::DOC_START

;**************************************************************
;                          VPENQ
;**************************************************************

;FUNCTIONAL DESCRIPTION:        ENQUE A PROCESS TO A VIRTUAL QUEUE
;
;       THIS ROUTINE IS IDENTICAL TO PENQ EXCEPT IS ASSUMES
;       THAT THE PROCESS TABLE TO BE ENQUEUED IS VIRTUAL AND
;       THE QUEUE TO BE ENQUEUED TO IS A VIRTUAL QUEUE.

;AC0 IN:         NOT USED
;AC1 IN:         QUEUE ADDRESS OF A VIRTUAL QUEUE
;                (VELQUE, IERES, OR IESWP)
;AC2 IN:         PID OF PROCESS TO BE ENQUEUED
;AC3 IN:         NOT USED

;CALLING CONVENTIONS:    JSR     VPENQ
                         <RETURN>

;ASSUMPTIONS:    NONE

;CAUTIONS:       USES MAP SLOT 62000
;                NOT TO BE USED TO ENQUEUE TO BLKQ

;ERROR RETURN CONDITIONS:        NONE

;AC0 OUT         UNCHANGED
;AC1 OUT:            "
;AC2 OUT:            "
;AC3 OUT:        FRAME POINTER

;GLOBAL DATA BASES ACCESSED:
;       *QUEUE TO BE ENQUEUED TO
;       *CUR62 - CURRENT CONTENTS OF 62000 SLOT
;       *PROCESS TABLE OFFSETS:
;                *PLNK - FORWARD QUEUE LINK
;                *PBLNK - BACKWARD QUEUE LINK
;                 PNQF - PRIORITY ENQUE FACTOR

;ANY DATA CHANGED:       ITEMS MARKED WITH A * ABOVE

;ROUTINES CALLED:
;                GPTBL           MPPID

;FILENAME:       COREM.SR

;MISCELLANEOUS:
;       THE VIRTUAL PROCESS TABLE OF THE SPECIFIED PROCESS
;       IS LINKED INTO THE QUEUE BY PRIORITY, USING PNQF
;       AS THE DETERMINING PRIORITY.  IF SEVERAL PROCESSES
;       HAVE THE SAME PNQF, THE PROCESS IS ENQUEUED AS THE
;       LAST OF THE GROUP OF PROCESSES WITH IDENTICAL PNQFS.

;DOCUMENTED BY: ANDY HUBER

;DOCUMENTATION DATE:     FEBRUARY 5, 1978

;:::DOC_END
```

```
;:::DOC_START

;*****************************************************************
                        VPDEQ
;*****************************************************************

;FUNCTIONAL DESCRIPTION:         DEQUEUE A PROCESS FROM A VIRTUAL QUEUE
;
;       THIS ROUTINE IS IDENTICAL TO PDEQ EXCEPT IT ASSUMES
;       THAT THE PROCESS TABLE TO BE DEQUEUED IS VIRTUAL AND
;       THE QUEUE TO BE DEQUEUED FROM IS A VIRTUAL QUEUE.

;AC0 IN:         NOT USED
;AC1 IN:             "
;AC2 IN:         PID OF PROCESS TO BE DEQUEUED
;AC3 IN:         NOT USED

;CALLING CONVENTIONS:    JSR     VPDEQ
;                        <RETURN>

;ASSUMPTIONS:    NONE

;CAUTIONS:       USES MAP SLOT 62000
;                NOT TO BE USED TO DEQUEUE FROM BLKQ

;ERROR RETURN CONDITIONS:        NONE

;AC0 OUT         UNCHANGED
;AC1 OUT:            "
   2 OUT:           "
;AC3 OUT:        FRAME POINTER

;GLOBAL DATA BASES ACCESSED:
;       *QUEUE TO BE DEQUEUED FROM
;       *CUR62 - CURRENT CONTENTS OF 62000 SLOT
;       *PROCESS TABLE OFFSETS:
;                *PLNK - FORWARD QUEUE LINK
;                *PBLNK - BACKWARD QUEUE LINK

;ANY DATA CHANGED:       ITEMS MARKED WITH A * ABOVE

;ROUTINES CALLED:
;       .BLD

;FILENAME:       COREM.SR

;MISCELLANEOUS: NONE

;DOCUMENTED BY: ANDY HUBER

;DOCUMENTATION DATE:     FEBRUARY 5, 1978

;:::DOC_END
```

```
;:::DOC_START

;*****************************************************************
;                          BLNCE
;  *****************************************************************

;FUNCTIONAL DESCRIPTION:          DETERMINE IF SYSTEM LOAD BALANCED
;
;       BALANCE IS CALLED TO DETERMINE IF THE SYSTEM LOAD IS
;       BALANCED ENOUGH THAT ANOTHER PROCESS SHOULD BE MADE
;       ELIGIBLE.  IT IS A POLICY ONLY ROUTINE.  SEE MISCEL-
;       LANEOUS FOR A DESCRIPTION OF THE ALGORITHM USED.

;AC0 IN:         NOT USED
;AC1 IN:              "
;AC2 IN:              "
;AC3 IN:              "

;CALLING CONVENTIONS:    JSR     BLNCE
;                        <ERROR RETURN>
;                        <GOOD RETURN>

;ASSUMPTIONS:    NONE

;CAUTIONS:       NONE

;ERROR RETURN CONDITIONS:         SYSTEM LOAD IS SUCH THAT NO MORE
;                                 PROCESSES SHOULD BE MADE ELIGIBLE

;  0 OUT:        UNCHANGED
;AC1 OUT:             "
;AC2 OUT:             "
;AC3 OUT:        FRAME POINTER

;GLOBAL DATA BASES ACCESSED:      NONE

;ANY DATA CHANGED:       NONE

;ROUTINES CALLED:        NONE

;FILENAME:       COREM.SR

;MISCELLANEOUS:
;       BLNCE DETERMINES IF THE SYSTEM SHOULD MAKE ANOTHER
;       PROCESS ELIGIBLE.  IT IS ENTIRELY A POLICY ALGORITHM.
;       CURRENTLY BLNCE ALWAYS TAKES THE GOOD RETURN, SINCE
;       INITIALLY WE WILL RUN AS MANY PROCESSES AS WILL FIT
;       (AS DOES REV. 1 AOS) UNTIL SUITABLE PERFORMANCE
;       EXPERIMENTS CAN BE RUN TO DETERMINE APPROPRIATE
;       ALGORITHMS.

;DOCUMENTED BY: ANDY HUBER

;DOCUMENTATION DATE:     FEBRUARY 10, 1978

;:  DOC_END
```

MODIFIED PRIMITIVES

THIS SECTION SUMMARIZES THE MODIFICATIONS THAT MUST BE MADE TO EXISTING PRIMITIVES. IN MOST CASES, THE PRIMITIVE MERELY MUST BE MADE AWARE THAT IT IS NOW DEALING WITH VIRTUAL PROCESS TABLES AND/OR VIRTUAL QUEUES. THE MODULE EACH PRIMITIVE IS DEFINED IN IS INCLUDED IN PARENTHESIS.

FITER - ENQUEUE PROCESS TO INELIGIBLE QUEUE (COREM)

INPUT: AC2 = PID, NOT PROCESS TABLE AS NOW

THE FUNCTION OF FITER IS UNCHANGED, BUT SINCE THE INELIGIBLE QUEUES ARE NOW VIRTUAL, IT MUST CALL VPENQ TO ENQUEUE THE PROCESS TO IESWP OR IERES.

CMENQ - ENQUE A PROCESS TO CORE MANAGER'S QUEUE (COREM)

INPUT: AC2 = PID, NOT PROCESS TABLE AS NOW

CMENQ'S FUNCTION IS UNCHANGED, BUT THE QUEUE, LINKED THROUGH OFFSET PCMLK, AND WHOSE BEGINNING IS DEFINED BY OFFSET CMQWD IN CMTSK, IS NOW VIRTUAL. SINCE PROCESSES ARE ENQUEUED AT THE END OF THIS QUEUE, IT MAY BE DESIRABLE TO INCLUDE THE PID OF THE LAST PROCESS ON THE QUEUE IN CMQWD. (FOR EXAMPLE, LET THE LEFT BYTE BE THE PID OF THE LAST PROCESS ON THE QUEUE, THE RIGHT BYTE THE PID OF THE FIRST PROCESS ON THE QUEUE.) THIS MAY NOT BE NECESSARY, HOWEVER, SINCE THIS QUEUE RARELY SEEMS TO HAVE MORE THAN ONE PROCESS TABLE ON IT. NOTE MAKING THIS QUEUE VIRTUAL IMPLIES THE INPUT PID MUST SPECIFY AN UNBOUND PROCESS.

CTBLK - BLOCK A PROCESS (COREM)

INPUT: AC2 = PROCESS TABLE, SAME AS NOW

TWO INTERNAL MODIFICATIONS ARE NECESSARY. AS USUAL, THE QUEUE WILL USE PIDS FOR LINKS INSTEAD OF PROCESS TABLE ADDRESSES, AND THE QUEUE WILL BE VIRTUAL. HOWEVER, CTBLK MAY NOT BE ABLE TO MOVE THE PROCESS TO THE BLOCKED QUEUE, BECAUSE THE "DO NOT MAKE VIRTUAL" BIT (PFNVT) MAY BE SET (E.G. DUE TO A SYSTEM CALL IN PROGRESS). IN SUCH A CASE, CTBLK WILL LEAVE THE PROCESS ON ELQUE, AND RELY ON THE SCHEDULER TO CALL CTBLK AGAIN WHEN THE PROCESS CAN BE MADE VIRTUAL. OTHERWISE, CTBLK WILL UNBIND THE PROCESS IF PFBVP IS SET, DEQUEUE IT FROM THE APPROPRIATE QUEUE AND ENQUE IT TO THE BLOCKED QUEUE.

IELSC - SCAN INELIGIBLE QUEUES (CMOV1)

INTERNALLY IELSC MUST BE CHANGED TO TAKE INTO ACCOUNT THAT IESWP AND IFRES, THE TWO INELIGIBLE QUEUES, ARE NOW VIRTUAL QUEUES. IELSC MUST NOW CALL THE BALANCE ROUTINE (BLNCE) BEFORE SCANNING IESWP TO SEE IF THE SYSTEM IS SO HEAVILY LOADED THAT ANOTHER SWAPPABLE PROCESS SHOULD NOT BE MADE ELIGIBLE EVEN IF IT WILL FIT.

BSCAN - SCAN BLOCKED QUEUE (CMOV1)

THIS MUST ALSO BE CHANGED TO REFLECT THE FACT THAT THE BLOCKED QUEUE IS NOW A VIRTUAL QUEUE. RECALL FROM THE DISCUSSION OF CTBLK, HOWEVER, THAT A PROCESS MAY BE BLOCKED WHILE ON ELQUE FOR A WHILE UNTIL IT CAN BE MADE VIRTUAL.

RUBLK - UNBLOCK A PROCESS (SGSUB)

INPUT: AC2 = PID, NOT PROCESS TABLE AS NOW

THIS ROUTINE IS CALLED TO UNBLOCK A PROCESS THAT IS IN CORE CURRENTLY. IT MUST KNOW THAT BLKQ IS NOW A VIRTUAL QUEUE, AND INSTEAD OF ENQUEING UNBLOCKED PROCESSES TO ELQUE SHOULD ENQUEUE THEM TO VELQUE. RUBLK MUST ALSO BE AWARE THAT THE PROCESS COULD STILL BE ON ELQUE, IN WHICH CASE NO DEQUEUEING OR ENQUEUEING IS NECESSARY.

DQBLQ - DEQUEUE A PROCESS FROM THE BLOCKED QUEUE (SGSUB)

INPUT: AC2 = PID INSTEAD OF PROCESS TABLE

THIS ROUTINE MUST BE MADE COGNIZANT OF VIRTUAL QUEUES, I.E. THAT BLKQ IS VIRTUAL. CARE MUST BE TAKEN SO THAT THIS ROUTINE IS NOT CALLED IF THE BLOCKED PROCESS IS NOT ON BLKQ.

PRPR - SELECT A PROCESS FOR PREEMPTION (COREM)

PRPR SCANS THE BLOCKED QUEUE FIRST IN SELECTING A CANDIDATE FOR PREEMPTION, THUS IT MUST INCORPORATE THE CHANGES NECESSARY TO SCAN A VIRTUAL QUEUE. AFTER THE SCAN OF BLKQ, INSTEAD OF SCANNING ELQUE AS NOW, PRPR WILL HAVE TO SCAN VELQUE, SINCE PROCESSES ON VELQUE WILL STILL HAVE MEMORY ALLOCATED TO THEM. ELQUE WILL ONLY BE SCANNED IF NOT ENOUGH MEMORY CAN BE TAKEN FROM PROCESSES ON BLKQ AND VELQUE. PRPR MUST CHECK THAT BIT PFNVT IS OFF BEFORE ENQUEING THE PROCESS TO THE CORE MANAGER, RATHER THAN MERELY CHECKING OFFSET PSQCT IN THE PROCESS TABLE EXTENDER.

TSPRC - PROCESS END OF TIME SLICE (COREM)

TIME SLICE END HAS BEEN CHOSEN AS THE TIME FOR MOVING PROCESSES BACK AND FORTH FROM VELQUE TO ELQUE. THUS TSPRC MUST BE CHANGED TO CHECK IF THERE ARE PROCESSES ON THE VIRTUAL ELIGIBLE QUEUE, AND IF SO, TO CALL MKINEL (IF NECESSARY) AND UVIRP TO REMOVE A PROCESS FROM ELQUE AND MKEL AND BVIRP TO MOVE A PROCESS FROM ELQUE TO VELQUE.

TSUP - PROCESS END OF SUB-SLICE (COREM)

THIS CODE, WHICH SHUFFLES PROCESSES OF EQUAL PRIORITY AT THE END OF EACH 32. MILLISECOND SUB-SLICE, NEED ONLY BE ALTERED TO UPDATE OFFSET PSSEL, THE NUMBER OF SUB-SLICES A PROCESS HAS RUN SINCE LAST PUT ON ELQUE, IN THE PROCESS TABLE OF THE EXECUTING PROCESS.

PRBAG - MARK A PROCESS FOR PREEMPTION (COREM)

THE TEST USED TO DECIDE IF THE PROCESS CAN BE ENQUEUED TO THE CORE MANAGER VIA CMENQ MUST BE CHANGED TO CHECK PFNVT RATHER THAN PSQCT, AS IN PRPR.

SUMMARY: CHANGES BY MODULE

THE AOS SOURCE MODULES THAT WILL BE AFFECTED BY THE SCHEDULER AND CORE MANAGER CHANGES DISCUSSED IN THIS SPECIFICATION ARE LISTED HERE, ALONG WITH A SUMMARY OF THE CHANGES TO THOSE MODULES. NOTE THE CHANGES NECESSARY TO THE CODE THAT PROCESSES THE VARIOUS SYSTEM CALLS, INCLUDING THE CHANGES TO IMPLEMENT THE ?DELAY FUNCTION, ARE NOT LISTED, NOR ARE THE CHANGES NECESSARY TO IMPLEMENT THE "OTHER DATA BASE CHANGES" DESCRIBED IN THIS DOCUMENT.

CMOV1:
- -MODIFY ROUTINES IELSC, BSCAN

COREM:
- -MODIFY ROUTINES FITER, CMENQ, CTBLK, PRPR, PRBAG, TSPRC, TSUP
- -ADD ROUTINES VPENQ, VPDEQ, BLNCE
- -ADD CODE TO BIND AND UNBIND SWPTB TO PROCESS BEING SWAPPED

I/O POST PROCESSORS:
- -TURN OFF PFNVT IF PSQCT DECREMENTED TO ZERO

INTS:
- -UPDATE PSSEL AT SUB-SLICE END

IOOV:
- -TURN ON PFNVT WHEN PSQCT IS INCREMENTED

MAPER:
- -ADD ROUTINE GPTBL

PARS:
- -DEFINE BITS PFNVT AND PFBVP
- -DEFINE OFFSET PSSEL

PROC2:
- -ADD CODE TO EXPAND VPTTB
- -CALL UVIRP TO UNBIND PROCESS FROM TEMPORARY RESIDENT PROCESS TABLE

SCHED:
- -ADD ROUTINES BVIRP, UVIRP, MKEL, MKINEL
- -ADD CODE TO TRY TO PUT BLOCKED PROCESSES ON BLKQ
- -TURN ON PFNVT WHEN STARTING A SYSTEM CALL

SCPRC:
- -TURN OFF PFNVT AFTER PROCESSING A CALL IF PSQCT IS NOW ZERO

SGSUB:

-MODIFY ROUTINES DQBLQ, RUBLK

SINIT:

-ADD CODE TO ALLOCATE VIRTUAL PROCESSORS AND INITIALIZE VPCHN, VPCNT, VPFCN
-ADD CODE TO INITIALIZE VPTTB

STABLE:

-DEFINE VPTTB
-DEFINE VELQUE, VPCHN, VPCNT, VPFCN, VELCN

STKS:

-DEFINE SWPTB

VIRTUALIZING CBASE: A PROPOSAL AND DESIGN SPECIFICATION

I. INTRODUCTION

THE DATA BASE AOS USES TO KEEP TRACK OF EACH 1 KW PAGE OF PHYSICAL MEMORY IS CALLED CBASE AND IS DEFINED IN MODULE MEMAP.SR. ACTUALLY, CBASE IS THE BASE OF THE MEMORY MAP DEFINED IN MEMAP, AND THE MEMORY MAP CONTAINS ONE FOUR-WORD ENTRY FOR EACH PHYSICAL PAGE IN THE CURRENT MEMORY CONFIGURATION. THESE ENTRIES ARE CALLED CORE MAP ENTRIES, OR "CMES". THE PROBLEM IS THAT THE MEMORY MAP COMES OUT OF AOS'S DYNAMIC MEMORY AREA, THUS THE LARGER THE AMOUNT OF PHYSICAL MEMORY, THE LARGER THE MEMORY MAP AND THE SMALLER THE AMOUNT OF DYNAMIC MEMORY LEFT. THIS IS MORE THAN UNFORTUNATE, BECAUSE IT MEANS A LARGER MEMORY SYSTEM WILL ACTUALLY BE ABLE TO SUPPORT LESS ACTIVITY BEFORE THE DYNAMIC MEMORY IS EXHAUSTED. NOTE THAT IN A SYSTEM WITH 128 KW OF MEMORY, THE MEMORY MAP REQUIRES 4 * 128 = 512 WORDS, BUT WITH 512 KW OF MEMORY THIS INCREASES TO 4 * 512 = 2048 WORDS OR 2 PAGES. SINCE OTHER MEASUREMENTS HAVE SHOWN THAT A MINIMAL SYSTEM WITH 256 KW OF MEMORY ONLY HAS ABOUT 5200 (OCTAL) WORDS OF DYNAMIC MEMORY, AOS SIMPLY CANNOT AFFORD TO USE UP THE ADDITIONAL 1024 WORDS THAT WOULD BE REQUIRED TO SUPPORT 512 KW OF PHYSICAL MEMORY. NOTE THIS PROBLEM WILL BE MUCH MORE SEVERE IF THE ECLIPSE IS EXTENDED TO ALLOW THE FULL 1024 KW OF MEMORY THAT THE MAP WILL ALLOW.

THIS DOCUMENT PROPOSES TO SOLVE THIS DIFFICULTY BY MAKING THE MEMORY MAP A VIRTUAL DATA BASE THAT IS MAPPED ONLY WHEN NEEDED. THIS HAS TWO IMPLICATIONS: FIRST, ANY SIZE PHYSICAL MEMORY CAN BE SUPPORTED EQUALLY WELL AND WITH NO EFFECT ON THE AMOUNT OF DYNAMIC MEMORY AVAILABLE TO AOS; SECOND, THE AMOUNT OF DYNAMIC MEMORY SPACE WILL ACTUALLY BE INCREASED. THE SCOPE OF THE REQUIRED CHANGES ARE DETAILED IN WHAT FOLLOWS; THE CHANGES ARE NOT MAJOR AND THE EFFECTS ARE FOR THE MOST PART VERY LOCALIZED. THESE CHANGES WILL HAVE TO BE MADE IN BOTH DEMAND PAGED AND NON-DEMAND PAGED AOS.

II. INTERNAL STRUCTURE

BASICALLY THE ONLY CHANGE TO AOS IS TO ALLOCATE CBASE IN DISASSOCIATED MEMORY THAT IS MAPPED WHENEVER A REFERENCE IS MADE TO A CME. SUCH A CHANGE AFFECTS PRIMARILY THE MEMORY MANAGEMENT CODE IN AOS. THE ACTUAL ROUTINES AND MODULES INVOLVED ARE DESCRIBED IN DETAIL IN THE NEXT SECTION.

SINIT WILL NOW ALLOCATE THE NUMBER OF 1 K PAGES NECESSARY TO HOLD CBASE AFTER IT HAS SIZED MEMORY TO DETERMINE THE AMOUNT OF PHYSICAL MEMORY ACTUALLY PRESENT ON THE SYSTEM. IT MUST FILL IN A TABLE SO THAT THESE PAGES CAN BE MAPPED AS REQUIRED.

MOST PLACES IN THE SYSTEM THAT CURRENTLY REFERENCE A CME DO SO BY CONSTRUCTING THE CME ADDRESS FROM THE PHYSICAL PAGE NUMBER. THE PAGE NUMBER IS MULTIPLIED BY FOUR (BY SHIFTING RIGHT TWO) AND THEN ADDED TO CBASE, THE STARTING ADDRESS OF THE MEMORY MAP. ALL SUCH PLACES WILL NOW MERELY CALL A ROUTINE (GET CORE MAP ENTRY ADDRESS - GCMEA) THAT WILL TAKE THE PHYSICAL PAGE NUMBER AS INPUT AND RETURN A MAPPED ADDRESS THAT CAN BE USED TO REFERENCE THE CME, DOING THE NECESSARY MAPPING. NOTE CONCEPTUALLY THIS IS MUCH CLEANER, AS IT REMOVES FROM THE CALLER THE KNOWLEDGE OF THE STRUCTURE OF THE MEMORY MAP.

THE 66000 SLOT WILL BE USED TO MAP CME'S. THIS IS BECAUSE THIS SLOT IS RARELY USED, AND THEN ONLY TEMPORARILY. NOTE THIS SLOT IS NOT PART OF A CONTROL BLOCK'S CONTEXT, THUS CALLERS OF GCMEA CANNOT PEND AND EXPECT THE CME TO STILL BE MAPPED AFTER THEY HAVE BEEN UNPENDED. THIS SHOULD NOT BE A PROBLEM AS CME'S ARE TYPICALLY REFERENCED FOR ONLY A FEW INSTRUCTIONS.

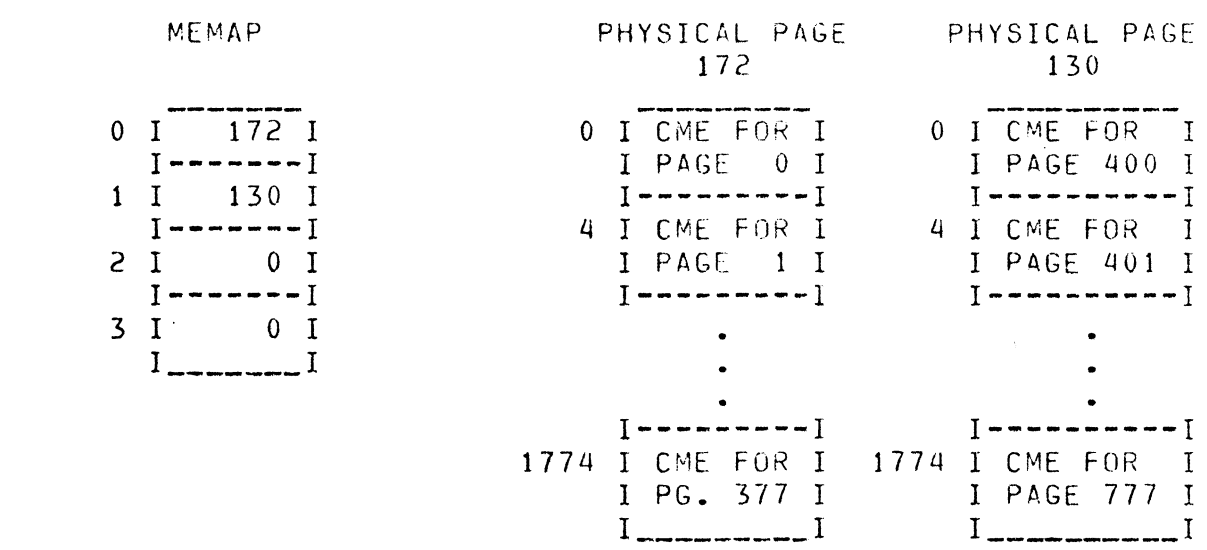
THE ONLY OTHER SIGNIFICANT CHANGES INTRODUCED ARE THE VIRTUALIZING OF TWO MEMORY CHAINS, FMCHN AND CANCH. THE FREE MEMORY CHAIN, FMCHN, IS A SINGLY LINKED LIST OF CME'S OF FREE PAGES. THE CANDIDATE CHAIN, CANCN, IS A DOUBLY LINKED, LEAST RECENTLY USED (LRU) CHAIN OF SHARED PAGES. BOTH FMCHN AND CANCN, WHICH ARE NOW CME ADDRESSES, WILL BE CHANGED TO BE PAGE NUMBERS, AND THE LINK WORDS IN THE CME'S THEMSELVES WILL BE CHANGED FROM CME ADDRESSES TO PAGE NUMBERS.

III. DESIGN SPECIFICATION

THIS SECTION DETAILS THE DATA BASE AND CODE CHANGES NECESSARY TO IMPLEMENT THIS PROPOSAL. IT IS COMPLETE ENOUGH THAT IT CAN BE USED BY AN IMPLEMENTOR TO ACTUALLY CARRY OUT THE REQUIRED MODIFICATIONS.

NEW DATA BASES

THE ONLY NEW DATA BASE REQUIRED IS THE MEMORY MAP TABLE, DESCRIBING WHICH 1 K PAGE HAS BEEN ALLOCATED TO HOLD EACH GROUP OF 256. CME'S. THIS TABLE WILL BE CALLED MEMAP, AND DEFINED IN MODULE MEMAP.SR, REPLACING THE CURRENT DEFINITION THERE OF CBASE. MEMAP WILL HAVE ONE ENTRY FOR EACH 256. PAGES OF PHYSICAL MEMORY IN THE SYSTEM, AND MEMAP (I) DENOTES THE PAGE NUMBER OF THE PHYSICAL PAGE OF MEMORY ALLOCATED TO HOLD THE CME'S FOR PAGES 0-155, 256-512, ETC. THUS THE STRUCTURE OF MEMAP IS (ALL NUMBERS IN OCTAL):



THUS, IF N = PHYSICAL PAGE #, THEN

N/256. = MEMAP ENTRY DEFINING PHYSICAL PAGE THE CME FOR PAGE N IS IN
4 * (N MOD 256.) = OFFSET OF CME WITHIN THAT PAGE

THE MAXIMUM LENGTH OF MEMAP IS FOUR, SINCE FOUR PAGES WILL HOLD 1024. CME'S, THE MAXIMUM AMOUNT THE CURRENT ECLIPSE MAP WILL EVER PERMIT. OBVIOUSLY, A MEMAP ENTRY OF 0 MEANS NO PAGE IS ALLOCATED (PHYSICAL PAGE 0 CAN NEVER BE ALLOCATED) BECAUSE THERE IS NOT THAT MUCH PHYSICAL MEMORY.

MODIFIED DATA BASES

THE FREE MEMORY CHAIN - FMCHN

LOCATION FMCHN IN STABLE, WHICH CURRENTLY POINTS TO THE CME OF THE FIRST FREE PAGE, WILL NOW BE THE PAGE NUMBER OF THE FIRST FREE PAGE. THE CHAIN OF FREE PAGES, SINGLY LINKED THOUGH OFFSET CMPFL IN THE CME, WILL BE CHANGED TO USE PAGE NUMBERS AS LINKS RATHER THAN CME ADDRESSES.

THE CANDIDATE CHAIN - CANCH

LOCATION CANCH IN STABLE, WHICH CURRENTLY POINTS TO THE CME OF THE FIRST (LEAST RECENTLY USED) PAGE ON THE CHAIN OF RELEASED SHARED PAGES, WILL NOW BE THE PAGE NUMBER OF THE FIRST PAGE ON THE CHAIN. THIS CHAIN IS DOUBLY LINKED USING OFFSETS CMPFL AND CMPBL IN THE CME; THESE LINKS WILL BECOME PAGE NUMBERS INSTEAD OF CME ADDRESSES. LOCATION CLEND, NOW A POINTER TO THE CME OF THE LAST PAGE ON CANCH, MUST ALSO BE CHANGED TO A PAGE NUMBER.

CBASE

THIS ENTRY, WHICH DEFINES THE STARTING ADDRESS OF THE MEMORY MAP, IS NO LONGER NEEDED AND WILL BE DELETED.

NEW SYSTEM PRIMITIVES

ONLY ONE NEW ROUTINE IS NEEDED, GCMEA - GET CORE MAP ENTRY ADDRESS, WHICH GIVEN A PHYSICAL PAGE NUMBER TRANSLATES IT INTO THE CORRESPONDING MAPPED CME ADDRESS, MAPS THAT CME, AND RETURNS THE MAPPED ADDRESS WHICH THE CALLER MAY USE TO REFERENCE THAT CME. A COMPLETE DESCRIPTION OF THIS ROUTINE USING THE PRESENTLY EMPLOYED AOS SYSTEM PRIMITIVE DOCUMENTATION TEMPLATE IS GIVEN ON A SEPARATE PAGE.

```
;:::DOC_START

;*****************************************************************
;                          GCMEA
;*****************************************************************

;FUNCTIONAL DESCRIPTION:          GET CORE MAP ENTRY ADDRESS

;AC0 IN:          NOT USED
;AC1 IN:          PHYSICAL PAGE # OF PAGE WHOSE CME DESIRED
;AC2 IN:          NOT USED
;AC3 IN:              "

;CALLING CONVENTIONS:    JSR@     .GCMEA
;                        <RETURN>

;ASSUMPTIONS:    NONE

;CAUTIONS:       USES 66000 MAP SLOT

;ERROR RETURN CONDITIONS:        NONE, BUT PANICS IF PAGE # INVALID

;AC0 OUT:         UNCHANGED
;AC1 OUT:             "
;AC2 OUT:         MAPPED CME ADDRESS
;AC3 OUT:         FRAME POINTER

;GLOBAL DATA BASES ACCESSED:
;        CNM66 - MAP SLOT 66000 MAP CONSTANT
;       *CUR66 - CURRENT CONTENTS OF 66000 SLOT
;        MEMAP - MEMORY MAP TABLE
;        HPAGE - # OF 1K PAGES IN SYSTEM

;ANY DATA CHANGED:       ITEMS MARKED WITH A * ARE CHANGED

;ROUTINES CALLED:
;                .BLD             .PNIC

;FILENAME:       MAPER.SR

;MISCELLANEOUS:
;        GCMEA COMPUTES THE MAPPED CORE MAP ENTRY ADDRESS AS
;        FOLLOWS:
;                LET N = PHYSICAL PAGE # PASSED IN AC1
;        THEN MEMAP (N/256.) CONTAINS THE PHYSICAL PAGE THE
;        CME IS IN.  THIS PAGE IS MAPPED INTO THE 66000 SLOT.
;        THE MAPPED ADDRESS IS:
;                CME ADDRESS = 66000 + 4*(N MOD 256.)

;DOCUMENTED BY: ANDY HUBER

;DOCUMENTATION DATE:     MARCH 5, 1978

;:::DOC_END
```

MODIFIED SYSTEM ROUTINES

THE FOLLOWING IS A COMPLETE LIST OF ALL SYSTEM MODULES THAT REFERENCE CME'S:

CLOSE	IOOV	MEMAP
MEMRY	SGSUB	SINIT
SOVLY	SRDB	SSOV6
SSOV9	STABLE	SWAPI
SWAPO	SZERO	

THE CHANGES REQUIRED IN EACH MODULE ARE DISCUSSED IN DETAIL BELOW.

CLOSE

ALL CLOSE DOES IS REFERENCE CME'S DESCRIBING PAGES OF SHARED FILES THAT ARE BEING CLOSED. THUS, CLOSE MUST MERELY BE CHANGED TO CALL GCMEA BEFORE REFERENCING A SHARED PAGE.

IOOV

IOOV CALLS SBSRH TO SEARCH AN FCB FOR A SHARED PAGE (I.E. TO SEE IF A SHARED PAGE IS ALREADY IN CORE AND SO NEED NOT BE READ) AND THEN PICKS UP THE SHARED PAGE DESCRIPTOR FROM THE CME SBSRH RETURNS. THIS CME ADDRESS IS SAVED AND USED AFTER A CALL TO SHREL TO RELEASE THE PREVIOUS CONTENTS OF THE SHARED SLOT. THIS MUST BE CHANGED SO THAT IOOV REMAPS THE CME AFTER THE CALL TO SHREL, SINCE SHREL CALLS RSBLK WHICH NOW MUST MAP THE CME OF THE PAGE BEING RELEASED.

MEMAP

THIS MODULE DEFINES CBASE CURRENTLY. THIS DEFINITION WILL BE DELETED, AND THE MEMAP TABLE DESCRIBED EARLIER WILL BE DEFINED IN ITS PLACE. ALTERNATIVELY, THIS MODULE COULD BE DELETED ALTOGETHER AND THE MEMAP TABLE DEFINED IN STABLE.

MEMRY

THIS IS THE PRIMARY MEMORY MANAGEMENT MODULE IN AOS, THUS MANY OF THE ROUTINES DEFINED HERE NEED TO BE CHANGED TO TAKE INTO ACCOUNT THE NEW STRUCTURE OF THE MEMORY MAP. EACH SUCH ROUTINE IS DESCRIBED SEPARATELY.

1. GSMEM, GSMNW, GSMRS

THESE THREE ROUTINES ALLOCATE DYNAMIC MEMORY SPACE (I.E. "GSMEM SPACE"), AND CALL GMBLK TO GET A 1 K PAGE. THE COMMON CODE IN THESE ROUTINES NEED BE CHANGED ONLY TO CALL GCMEA RATHER THAN CALCULATE THE CME ADDRESS OF THE ALLOCATED PAGE ITSELF.

2. GMBLK

THIS ROUTINE ALLOCATES A FREE 1 KW PAGE TO THE CALLER FROM FMCHN OR CANCH. IT RETURNS IN AC1 THE PHYSICAL PAGE NUMBER OF THE ALLOCATED PAGE. INTERNALLY, THIS ROUTINE MUST TAKE INTO ACCOUNT THE CHANGES MADE IN FMCHN AND CANCH.

3. RFBLK

THIS IS THE COMPLEMENT OF GMBLK, AND IS CALLED TO RELEASE TO THE SYSTEM A FREE 1 KW PAGE. INTERNALLY IT MUST BE CHANGED TO ACCOMODATE THE VIRTUAL FREE MEMORY CHAIN, FMCHN. NO CHANGES IN CALLING SEQUENCE ARE NECESSARY.

4. RMBLK, RSBLK

THESE ROUTINES RELEASE A SHARED 1 KW PAGE OF MEMORY TO THE HEAD AND TAIL OF CANCH, RESPECTIVELY. THEY MUST BE CHANGED TO REFLECT VIRTUAL CME'S AND THE VIRTUAL CANCN. NO EXTERNALLY VISIBLE CHANGES ARE REQUIRED. NOTE THAT THESE TWO ROUTINES PASS BACK THE CORE MAP ENTRY ADDRESS, THIS WILL, OF COURSE, NOW BE A MAPPED ADDRESS.

5. NQCAN

THIS INTERNAL ROUTINE ACTUALLY DOES THE ENQUEING OF A CME TO CANCH. IT NEEDS TO BE MADE AWARE OF THE CHANGES IN THE FORMAT OF CANCH.

6. GCORE

THIS ROUTINE GETS THE UNSHARED MEMORY FOR A PROCESS BY CALLING GMBLK. SINCE IT DOES NOT REFERENCE THE CME, NO CHANGES ARE NEEDED.

7. SHFLS

THIS ROUTINE FLUSHES A SHARED PAGE SPECIFIED BY CME ADDRESS TO DISK. SINCE THIS ROUTINE PENDS WHILE THE I/O IS IN PROGRESS, IT MUST BE CHANGED TO RE-MAP THE CME THE CALLER HAS SPECIFIED AFTER THE I/O COMPLETES SO THE CALLER CAN STILL REFERENCE THE CME. NO CHANGES IN THE CALLING SEQUENCE ARE ANTICIPATED.

8. GFCB

THIS ROUTINE ALLOCATES FCB'S, AND CALLS GMBLK. IT MUST BE CHANGED TO CALL GCMEA, INSTEAD OF COMPUTING THE CME ADDRESS OF THE ALLOCATED PAGE ITSELF.

9. RFCB

THIS ROUTINE RELEASES FCB'S, AND USES THE PHYSICAL PAGE NUMBER IN THE VIRTUAL FCB ADDRESS TO COMPUTE THE CME ADDRESS OF THE PAGE HOLDING THE FCB SO THE FCB HEADER CAN BE FOUND (IN OFFSET CMPPT). THIS MUST BE CHANGED TO CALL GCMEA.

THAT CONCLUDES THE CHANGES NECESSARY IN MODULE MEMRY.

SGSUB

ROUTINE CHKAR IN SGSUB COMPUTES THE CME ADDRESS OF SHARED PAGES IT WISHES TO CHECK. CHKAR NEEDS TO BE MODIFIED TO CALL GCMEA TO DO THIS ADDRESS COMPUTATION.

SINIT

SINIT MUST BE ALTERED TO INITIALIZE THE NEW MEMAP TABLE AFTER IT HAS SIZED MEMORY. BASED ON THE AMOUNT OF PHYSICAL MEMORY FOUND, SINIT WILL ALLOCATE 1-4 PAGES TO HOLD THE CME'S.

SOVLY

THE OVERLAY HANDLING MODULE CALLS GMBLK TO GET 1 KW PAGES TO BE USED FOR OVERLAYS, AND THEN UPDATES THE CME OF THE ALLOCATED PAGE TO REFLECT ITS CURRENT USAGE. THUS SOVLY MUST NOW CALL GCMEA TO COMPUTE THE CME ADDRESS.

SRDB

THIS IS THE SYSTEM SHARED READ CODE, AND THIS CODE SETS UP THE CME'S FOR SHARED PAGES IT READS IN. THIS CODE MUST USE GCMEA. ALSO, TWO ROUTINES DEFINED IN SRDB MUST BE MODIFIED:

1. SBSRH

   THIS ROUTINE SEARCHES AN FCB SHARED PAGE LIST FOR A GIVEN SHARED PAGE, AND RETURNS THE CME ADDRESS OF THE PAGE IF FOUND. THUS, SBSRH MUST BE CHANGED TO RETURN A MAPPED ADDRESS.

2. DQCHD

   THIS ROUTINE DEQUEUES A CME FROM CANCH, AND HENCE MUST BE CHANGED TO REFLECT THE FACT THAT CANCH IS NOW A VIRTUAL CHAIN.

STABLE

THE NEW DEFINITIONS OF CANCH, CLEND, AND FMCHN NEED TO BE DOCUMENTED IN THIS MODULE. NOTE NO CODE IS AFFECTED, ONLY COMMENTS, UNLESS IT IS DECIDED TO DEFINE THE NEW FORMAT MEMAP TABLE HERE INSTEAD OF IN MEMAP.SR.

SSOV6

THIS MODULE CONTAINS A ROUTINE CALLED CHKSA WHICH IS INVOKED BY AN ?SCLOSE IF THE ?SCLOSE SPECIFIED THAT PAGES OF THE FILE BEING CLOSED BE RELEASED IF NECESSARY. THIS ROUTINE MUST CALL GCMEA RATHER THAN COMPUTE THE CME ADDRESS ITSELF.

SSOV9

THE CODE IN THIS MODULE THAT IMPLEMENTS THE ?FLUSH SYSTEM CALL MUST BE CHANGED TO CALL GCMEA INSTEAD OF DOING THE COMPUTATION ITSELF.

SWAPI

THE SWAP IN CODE MUST BE CHANGED TO CALL GCMEA TO COMPUTE THE CME ADDRESS OF SHARED PAGES.

SWAPO

THE SWAP OUT CODE REFERENCES THE CME'S OF PAGES RELEASED. NO CHANGES ARE NECESSARY IF RSRLK IS CHANGED AS DESCRIBED EARLIER TO RETURN THE CORRECT, MAPPED CME ADDRESS.

SZERO

ENTRY .CBASE, CONTAINING THE ADDRESS OF CBASE, CAN BE ELIMINATED. AN ENTRY .GCMEA CONTAINING THE ADDRESS OF THE GCMEA ROUTINE WILL BE ADDED.

```
                        .TITL   DSKBT   ;AOS DISK BOOTSTRAP

                ; TO BOOT A SYSTEM FROM THE INVISIBLE DISK SPACE OF AN AOS
                ; SYSTEM OR BOOTSTRAP DISK TO CORE
                ;
                ; THIS CODE WILL RESIDE IN PHYSICAL BLOCKS ZERO AND ONE OF AOS
                ; FORMATTED DISKS. THE DISK DRIVER PORTION OF THIS CODE IS
                ; BUILT DYNAMICALLY BY THE AOS INSTALLER AND DISK FORMATTER
                ;
                ; DSKBT IS LOADED VIA THE PROGRAM LOAD FUNCTION (ACTUALLY,
                ; ONLY THE FIRST HALF OF THE AOS DISK BOOTSTRAP IS LOADED BY
                ; THE PROGRAM LOADER. THE SECOND HALF IS LOADED BY THE FIRST
                ; HALF). DSKBT WILL EXECUTE FROM LOCATIONS 0 THRU 777
                ;
                ; CONSOLE DATA SWITCHES: 0 - UP (DATA CHANNEL)
                ;                        1 THRU 3 - UNIT NUMBER
                ;                        10 THRU 15 - DEVICE CODE
                ;
                ; NOTE - THE PROGRAM LOADER LOADS DSKBT FROM THE DISK WHOSE
                ;        UNIT NUMBER IS ZERO. HOWEVER, DSKBT LOADS THE SYSTEM
                ;        FROM THE DISK WHOSE UNIT NUMBER IS SPECIFIED IN THE
                ;        CONSOLE DATA SWITCHES
                ;
                ; RESULTING CORE LAYOUT: 0 - DEVICE CODE (LH), UNIT NUMBER (RH)
                ;                        1 - DISK ADDRESS OF SYSTEM'S ORIGIN
                ;                        2 THRU 76377 - 31.25K OF SYSTEM
                ;                        76400 THRU ? - GARBAGE
                ;
                ; THE SYSTEM IS ENTERED VIA LOCATION 2
                ;
                ; A SPECIAL INDIRECT ENTRY IN LOCATION 2 IS PROVIDED FOR AOS
                ; BOOTSTRAPPING. THE DEVICE CODE AND UNIT NUMBER ARE SET UP
                ; BY THE CALLER (AOS) AND AC1 HAS THE DISK ADDRESS OF THE
                ; SYSTEM'S ORIGIN ON DISK
                ;
                ; POSITION INDEPENDENT CODE

                        .ENT    DSKBT,DDLEN

                        .NREL

        000001          .TXTM   1

                ; DISK DRIVER ENTRY POINT MNEMONICS

        000003 RECAL=3
        000005 READ=5

                DSKBT:                  ;ORIGIN OF DISK BOOTSTRAP

00000'000000 LOC0:      0               ;DEVICE CODE FOR DISK DRIVER
00001'000000 LOC1:      0               ;UNIT NUMBER FOR DISK DRIVER
00002'000442 LOC2:      AOSBT-DSKBT     ;SPECIAL @ENTRY FOR AOS BOOTSTRAPPER
00003'000005 LOC3:      5.              ;DISK ERROR RETRY COUNT
00004'000000 LOC4:      0               ;DISK STATUS

00005'000400 .BT2:      BOOT2-DSKBT     ;POINTER TO DSKBT'S BETTER HALF
```

```
01
02 00006'062677 BOOT1:  IORST                   ;CLEAR THE WORLD
03 00007'102400         SUB 0,0                 ;CLEAR UNIT NUMBER - DESTROYED
04 00010'040771         STA 0,LOC1              ;BY PAGING DISK'S STATUS
0                ;
06               ; GET THE DEVICE CODE FROM THE CONSOLE DATA SWITCHES AND
07               ; PUT IT AWAY
08               ;
09 00011'060477         READS 0                 ;READ SWITCHES
10 00012'024414         LDA 1,DVCMK             ;DEVICE CODE MASK
11 00013'123400         AND 1,0                 ;MASK DEVICE CODE
12 00014'040764         STA 0,LOC0              ;PUT IT INTO .LOC 0
13               ;
14               ; LOAD THE OTHER HALF OF THE AOS DISK BOOTSTRAP
15               ;
16 00015'102400         SUB 0,0                 ;DISK ADDRESS 1
17 00016'105400         INC 0,1
18 00017'131300         MOVS 1,2                ;MEMORY ADDRESS 400
19 00020'004415         JSR DDORG+READ          ;DUMP IT IN
20 00021'000400         JMP .                   ;DISK ERROR.
21 00022'000400         JMP .                   ;"
22 00023'000400         JMP .                   ;"
23 00024'000400         JMP .                   ;"
24               ;
25 00025'002760         JMP @.BT2               ;THAT'S IT FOR THIS HALF.
26               ;
27 00026'000077 DVCMK:  77              ;DEVICE CODE MASK
28
29               ;
30               ; THE DISK DRIVER WILL LIVE HERE. LOCATION 377 HAS A JUMP
31               ; TO DSKBT'S START FOR THE PROGRAM LOADER
32               ;
33 00027'000347 DDLEN:  POEND-DDORG     ;MAX DISK DRIVER LENGTH
34               DDORG:                 ;DISK DRIVER ORIGIN
35               ;
36       000377'         .LOC    DSKBT+377
37               ;
38 00377'000006 POEND:  JMP BOOT1-DSKBT         ;FOR PROGRAM LOAD.
```

```
!0003 DSKBT
01                      BOOT2:
02                      ;
03                      ; RECALIBRATE THE DISK DRIVE (MIGHT BE FINISHED WITH IT)
04                      ;
0  00400'004033                 JSR DDORG+RECAL-DSKBT
06 00401'000513                 JMP DERR2               ;DISK ERROR.
07 00402'000512                 JMP DERR2               ;"
08 00403'000511                 JMP DERR2               ;"
09 00404'000510                 JMP DERR2               ;"
10                      ;
11                      ; GET THE UNIT NUMBER FROM THE CONSOLE DATA SWITCHES AND
12                      ; PUT IT AWAY
13                      ;
14 00405'060477                 READS 0                 ;READ SWITCHES
15 00406'024550                 LDA 1,UNMK              ;UNIT NUMBER MASK
16 00407'123400                 AND 1,0                 ;MASK UNIT NUMBER
17 00410'101120                 MOVZL 0,0               ;UNIT # TO BITS 13 THRU 15...
18 00411'101100                 MOVL 0,0
19 00412'101100                 MOVL 0,0
20 00413'101100                 MOVL 0,0
21 00414'101100                 MOVL 0,0
22 00415'040001                 STA 0,LOC1-DSKBT        ;PUT IT INTO .LOC 1
23                      ;
24                      ; RECALIBRATE THE DISK DRIVE (MIGHT BE A NEW DRIVE)
25                      ;
26 00416'004033                 JSR DDORG+RECAL-DSKBT
27 00417'000475                 JMP DERR2               ;DISK ERROR.
28 00420'000474                 JMP DERR2               ;"
29 00421'000473                 JMP DERR2               ;"
3  00422'000472                 JMP DERR2               ;"
31                      ;
32                      ; READ IN THE DIB AND SEE IF THERE IS A SYSTEM ON THIS DISK.
33                      ; IF THERE IS, GET THE DISK ADDRESS OF ITS ORIGIN
34                      ;
35 00423'102400                 SUB 0,0                 ;DISK ADDRESS OF DIB
36 00424'024531                 LDA 1,DIBSC
37 00425'030532                 LDA 2,C1000             ;MEMORY ADDRESS 1000 (SCRATCH)
38 00426'004035                 JSR DDORG+READ-DSKBT    ;READ IN DIB
39 00427'000456                 JMP DERR1               ;DISK ERROR.
40 00430'000455                 JMP DERR1               ;"
41 00431'000454                 JMP DERR1               ;"
42 00432'000453                 JMP DERR1               ;"
43 00433'025017                 LDA 1,IBSYS,2           ;IS THERE A SYSTEM OUT THERE?
44 00434'125102                 MOVL 1,1,SZC
45 00435'000404                 JMP SI                  ;YES.
46 00436'030557                 LDA 2,NSMES             ;NO, SAY SO
47 00437'004515                 JSR X
48 00440'063077                 HALT                    ;DEAD (CAN PRESS CONTINUE).
49 00441'125221 SI:             MOVZR 1,1,SKP           ;SAVE ITS ORIGIN...
50                      ;
51                      ;SPECIAL ENTRY FOR AOS BOOTSTRAP MECHANISM
52                      ;
53 00442'062677 AOSBT:  IORST                           ;CLEAR THE WORLD
54 00443'044515                 STA 1,SORG              ;SAVE SYSTEM'S ORIGIN
55                      ;
56                      ; LOAD 31.25K OF THIS SYSTEM ON DISK INTO LOCATIONS
57                      ; 1000 THRU 77377, A TOTAL OF 125. DISK BLOCKS
58                      ;
59 00444'020515                 LDA 0,NBK               ;BLK COUNT AND DISK ADDR (HIGH)
60                                                      ;DISK ADDRESS (LOW) IS IN AC1
```

```
 0004 DSKBT
01 00445'030512          LDA 2,C1000             ;MEMORY ADDRESS
02 00446'004035          JSR DDORG+READ-DSKBT    ;DUMP IT IN
03 00447'000436          JMP DERR1               ;DISK ERROR.
04 00450'000435          JMP DERR1               ;"
05 00451'000434          JMP DERR1               ;"
06 00452'000433          JMP DERR1               ;"
07                ;
08                ; RECALIBRATE THE DISK DRIVE (ALL DONE WITH IT)
09                ;
10 00453'004033          JSR DDORG+RECAL-DSKBT
11 00454'000440          JMP DERR2               ;DISK ERROR.
12 00455'000437          JMP DERR2               ;"
13 00456'000436          JMP DERR2               ;"
14 00457'000435          JMP DERR2               ;"
15                ;
16                ; PUT THE DEVICE CODE AND UNIT NUMBER INTO LOCATION 0 OF THE
17                ; SYSTEM IN CORE AND THE DISK ADDRESS OF THE SYSTEM'S ORIGIN
18                ; INTO LOCATION 1
19                ;
20 00460'020000          LDA 0,LOC0-DSKBT        ;DEVICE CODE
21 00461'101300          MOVS 0,0                ;TO LEFT BYTE
22 00462'024001          LDA 1,LOC1-DSKBT        ;UNIT NUMBER
23 00463'123000          ADD 1,0                 ;TO RIGHT BYTE
24 00464'030473          LDA 2,C1000             ;PUT INTO LOCATION 0
25 00465'041000          STA 0,0,2
26 00466'020472          LDA 0,SORG              ;DISK ADDRESS OF SYSTEM'S
27 00467'041001          STA 0,1,2               ;ORIGIN TO LOCATION 1
28                ;
29                ; COPY SOME CODE TO LOCATIONS 77400 THRU 77400+SSMOVE-1.
30                ; THIS CODE WILL MOVE THE SYSTEM INTO LOCATIONS 0 THRU 76377
31                ;
32 00470'024412          LDA 1,NSSMV             ;COUNT (NEGATIVE)
33 00471'030412          LDA 2,SMVORG            ;SOURCE
34 00472'034412          LDA 3,ACON              ;DESTINATION
35 00473'021000 MOVE1:   LDA 0,0,2               ;MOVE IT...
36 00474'041400          STA 0,0,3
37 00475'151400          INC 2,2
38 00476'175400          INC 3,3
39 00477'125404          INC 1,1,SZR
40 00500'000773          JMP MOVE1
41 00501'002403          JMP @ACON               ;GO THERE.
42                ;
43 00502'177764 NSSMV:   -SSMOVE          ;NEGATIVE WORD-SIZE OF SMOVE
44 00503'000643 SMVORG:  SMOVE-DSKBT      ;ORIGIN OF SMOVE
45 00504'077400 ACON:    77400            ;ORIGIN OF LAST PAGE IN 32K SYSTEM
46                ;
47                ; DISK ERROR, RECALIBRATE THE DISK DRIVE
48                ;
49 00505'020004 DERR1:   LDA 0,LOC4-DSKBT        ;GET DISK STATUS
50 00506'004033          JSR DDORG+RECAL-DSKBT
51 00507'000405          JMP DERR2
52 00510'000404          JMP DERR2
53 00511'000403          JMP DERR2
54 00512'000402          JMP DERR2
55 00513'000402          JMP DERR0               ;ALREADY HAVE DISK STATUS.
56                ;
57                ; DISK ERROR, SAY SO AND GIVE UP
58                ;
59 00514'020004 DERR2:   LDA 0,LOC4-DSKBT        ;GET DISK STATUS
60 00515'030435 DERR0:   LDA 2,ODMK              ;OCTAL DIGIT MASK
```

```
 0005 DSKBT
01 00516'034435 DER2L:  LDA 3,C60               ;ASCII ZERO
02 00517'105000         MOV 0,1                 ;GET NEXT OCTAL DIGIT
03 00520'147400         AND 2,1
04 00521'167000         ADD 3,1                 ;MAKE IT ASCII
0  00522'046426         STA 1,@DLOC             ;SAVE IT
06 00523'101220         MOVZR 0,0               ;GET NEXT OCTAL DIGIT
07 00524'101220         MOVZR 0,0
08 00525'101200         MOVR 0,0
09 00526'105000         MOV 0,1
10 00527'147400         AND 2,1
11 00530'167300         ADDS 3,1                ;MAKE IT ASCII, MOVE TO HIGH
12 00531'036417         LDA 3,@DLOC             ;GET THE OTHER IN THIS GROUP
13 00532'167000         ADD 3,1                 ;PUT INTO LOW
14 00533'046415         STA 1,@DLOC             ;PUT TWO DIGIT GROUP AWAY
15 00534'101200         MOVR 0,0                ;POSITION NEXT OCTAL DIGIT
16 00535'101200         MOVR 0,0
17 00536'101200         MOVR 0,0
18 00537'014411         DSZ DLOC                ;POINT TO NEXT TWO DIGIT GROUP
19 00540'014411         DSZ DCNT                ;ALL DONE?
20 00541'000755         JMP DER2L               ;NO, ANOTHER TWO DIGIT GROUP.
21 00542'030420         LDA 2,DEMES             ;YES, PRINT ERROR MESSAGE...
22 00543'004514         JSR TYPIT
23 00544'030443         LDA 2,DESTP
24 00545'004512         JSR TYPIT
25 00546'063077         HALT                    ;DEAD.
26 00547'000777         JMP .-1                 ;CAN'T CONTINUE.
27                      ;
28 00550'000612 DLOC:   DEST-DSKBT+2    ;POINTER TO CURRENT TWO DIGIT GROUP
29 00551'000003 DCNT:   3.              ;NUMBER OF TWO DIGIT GROUPS TO PROCESS
3  00552'000007 ODMK:   7               ;OCTAL DIGIT MASK
31 00553'000060 C60:    "0              ;ASCII ZERO
32                      ;
33 00554'000503 X:      JMP TYPIT       ;BRIDGE
34                      ;
35 00555'000003 DIBSC:  SCDIB           ;DISK ADDRESS OF DIB
36 00556'070000 UNMK:   70000           ;UNIT NUMBER MASK
37 00557'001000 C1000:  1000            ;OCTAL 1000
38 00560'000000 SORG:   0               ;DISK ADDRESS OF SYSTEM'S ORIGIN
39 00561'076400 NBK:    125.*400        ;# OF DISK BLOCKS IN 31.25K WORDS
40 00562'001346 DEMES:  .+1-DSKBT*2
41 00563'006412         .TXT '<15><12><12>FROM DISK BOOT: DISK ERROR, STATUS='
42       005106
43       051117
44       046440
45       042111
46       051513
47       020102
48       047517
49       052072
50       020104
51       044523
52       045440
53       042522
54       051117
55       051054
56       020123
57       052101
58       052125
59       051475
60       000000
```

```
        LICENSED MATERIAL - PROPERTY OF DATA GENERAL CORPORATION
 0006 DSKBT
01 00607'001420 DESTP:  DEST-DSKBT*2
02 00610'054130 DEST:   .TXT 'XXXXXX<15><12>'
03       054130
04       054130
0        006412
06       000000
07 00615'001434 NSMES:  .+1-DSKBT*2
08 00616'006412         .TXT '<15><12><12>FROM DISK BOOT: SYSTEM NOT
09       005106
10       051117
11       046440
12       042111
13       051513
14       020102
15       047517
16       052072
17       020123
18       054523
19       052105
20       046440
21       047117
22 00634'052040  INSTALLED<15><12>'
23       044516
24       051524
25       040514
26       046105
27       042015
28       005000
29
3'              ;
31              ; MOVE THE SYSTEM TO ITS PROPER LOCATION
32              ;
33 00643'024412 SMOVE:  LDA 1,NNPG              ;COUNT (NEGATIVE)
34 00644'030412         LDA 2,CTHOU             ;SOURCE
35 00645'176400         SUB 3,3                 ;DESTINATION
36 00646'021000 MOVE2:  LDA 0,0,2               ;MOVE IT...
37 00647'041400         STA 0,0,3
38 00650'151400         INC 2,2
39 00651'175400         INC 3,3
40 00652'125404         INC 1,1,SZR
41 00653'000773         JMP MOVE2
42              ;
43              ; ENTER THE SYSTEM VIA LOCATION 2
44              ;
45 00654'002002         JMP @2                  ;BYE-BYE.
46              ;
47 00655'101400 NNPG:   -76400          ;NEGATIVE NUMBER OF WORDS TO MOVE
48 00656'001000 CTHOU:  1000            ;OCTAL 1000
49       000014 SSMOVE=.-SMOVE          ;WORD-SIZE OF SMOVE
50
51              ;
52              ; SUBROUTINE TO TYPE A MESSAGE. CALL WITH A JSR. AC2 HAS
53              ; A BYTEPOINTER TO THE MESSAGE (NULL TERMINATION). AC0
54              ; IS DESTROYED AND AC1 IS SAVED
55              ;
56  )657'054417 TYPIT:  STA 3,TIRTN             ;SAVE RETURN ADDRESS
57 00660'151220 TTTYP:  MOVZR 2,2               ;GET NEXT CHAR IN MESSAGE...
58 00661'021000         LDA 0,0,2
59 00662'151013         MOV# 2,2,SNC
60 00663'101300         MOVS 0,0
```

```
 0007 DSKBT
01 00664'151100          MOVL 2,2
02 00665'034417          LDA 3,C177
03 00666'163400          AND 3,0
04 00667'151400          INC 2,2                    ;BUMP MESSAGE BYTEPOINTER
05 00670'101015          MOV# 0,0,SNR               ;END OF MESSAGE?
06 00671'002405          JMP @TTIRTN                ;YES, RETURN TO CALLER.
07 00672'061111          DOAS 0,TTO                 ;NO, TYPE THE CHARACTER
08 00673'063611          SKPDN TTO                  ;IS IT DONE?
09 00674'000777          JMP .-1                    ;NO, WAIT FOR IT.
10 00675'000763          JMP .TTYP                  ;YES, LOOP.
11                  ;
12 00676'000000 TTRTN:   0                  ;RETURN ADDRESS SAVEAREA
13 00677'000177 C177:    177                ;OCTAL 177 (CHARACTER MASK)
14
15 00700'000100          .BLK    DSKBT+1000-.       ;ZERO REST OF DISK BLOCK 1
16
17                       .END

**00000 TOTAL ERRORS, 00000 PASS 1 ERRORS
```

		7/05						
MOVE1	000473'	4/35	4/40					
MOVE2	000646'	6/36	6/41					
N	000561'	3/59	5/39					
NNPG	000655'	6/33	6/47					
NSMES	000615'	3/46	6/07					
NSSMV	000502'	4/32	4/43					
ODMK	000552'	4/60	5/30					
POEND	000377'	2/33	2/38					
READ	000005	1/50	2/19	3/38	4/02			
READS	060477	2/09	3/14					
RECAL	000003	1/49	3/05	3/26	4/10	4/50		
SCDIB	000003	5/35						
SI	000441'	3/45	3/49					
SKP	000001	3/49						
SKPDN	063600	7/08						
SMOVE	000643'	4/44	6/33	6/49				
SMVOR	000503'	4/33	4/44					
SNC	000003	6/59						
SNR	000005	7/05						
SURG	000560'	3/54	4/26	5/38				
SSMOV	000014	4/43	6/49					
STA	040000	2/04	2/12	3/22	3/54	4/25	4/27	4/36
		5/05	5/14	6/37	6/56			
SUB	102400	2/03	2/16	3/35	6/35			
SZC	000002	3/44						
SZR	000004	4/39	6/40					
TIRTN	000676'	6/56	7/06	7/12				
T1	000011	7/07	7/08					
T1T	000657'	5/22	5/24	5/33	6/56			
UNMK	000556'	3/15	5/36					
X	000554'	3/47	5/33					
.BT2	000005'	1/60	2/25					