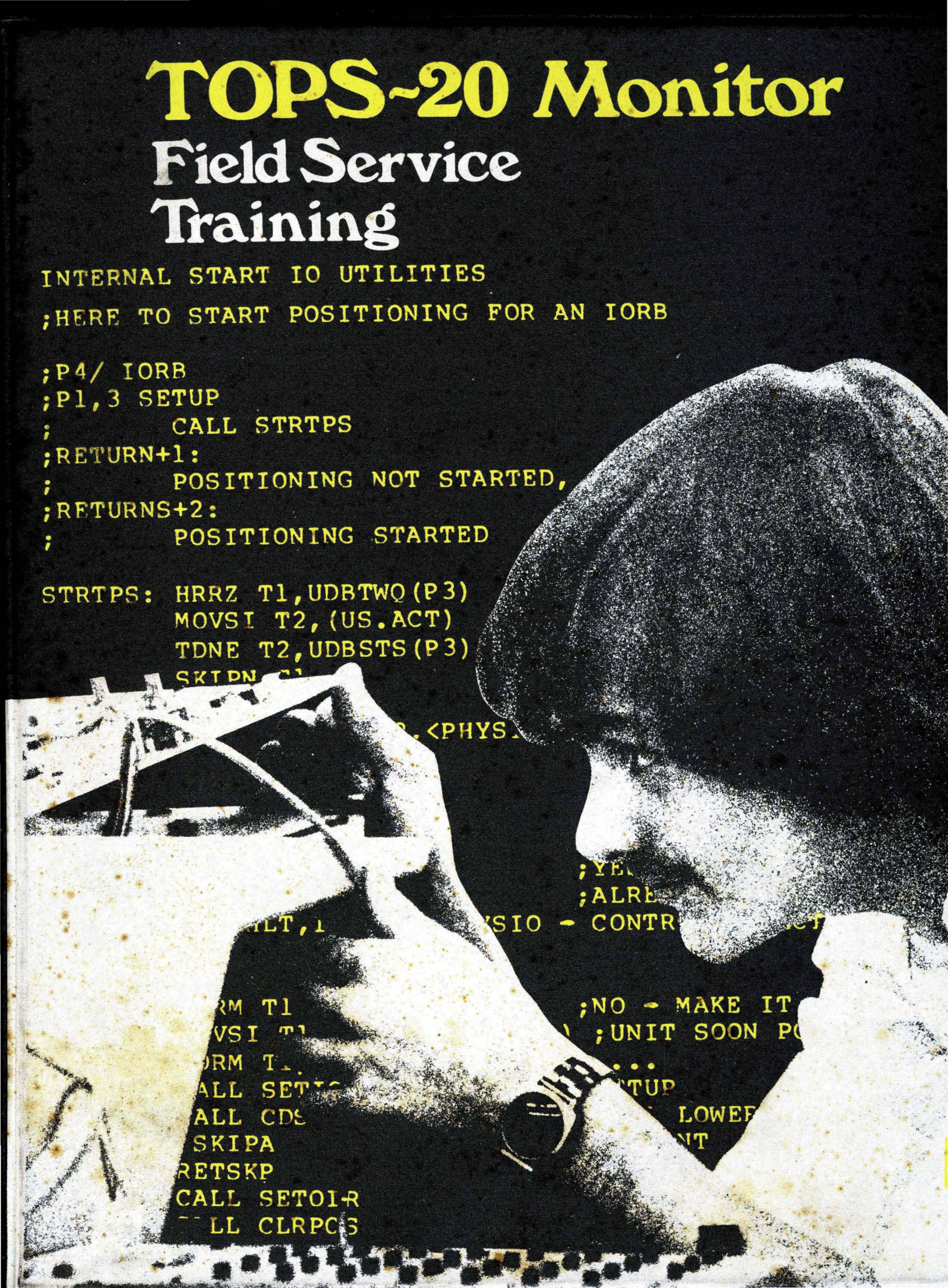
TOPS-20 Monitor
Field Service Training



TOPS-20 MONITOR

Course Number: J1183-A

Educational Services
Digital Equipment Corporation
Marlboro, Massachusetts

NOTE

The material in this course is intended for internal use only by Digital personnel. The material in this course is not to be duplicated nor distributed to customers or prospective customers.

This page is for notes.

CONTENTS

Student Guide.....SG
Software Introduction .....SI
Monitor Overview .....MO
Coding Conventions CC
SYSERR .....SER
Troubleshooting .....TS
PHYSIO - Disk / Tape .....DT
Front End/Terminals .....FE
Storage Management .....SM
Monitor Tables .....
Monitor Logic Manual .....
Miscellaneous .....MIS
Appendix I .....APP
Appendix II .....
Appendix III .....

TOPS-20 MONITOR

Student Guide

CONTENTS

Course Description SG-1

Prerequisites SG-1

Course Objectives SG-1

Resources SG-2

Course Outline SG-3

Course Evaluation Form SG-9

Appendix A - Course Map SG-13

This page is for notes.

COURSE DESCRIPTION

This course is addressed to regional and district support personnel and is designed to help in their task of determining and isolating system faults occurring in the TOPS-2Ø operating system. The approach focuses on troubleshooting, fault-finding, and error analysis. Following an overview of the major monitor structures and coding conventions is a general discussion of debugging techniques, including the BUGHLT features of the TOPS-2Ø system. Several of the available tools are covered thereafter, for example, MDDT and SYSERR. The remainder of the course is devoted to the main functions of TOPS-2Ø.

PREREQUISITES

The regional or district support specialist is expected to have completed the courses leading to the the Diagnostics course in this curriculum, including Software Concepts, ALP, and Specialist.

COURSE OBJECTIVES

Upon completion of this course, the student will be able to:

1. Describe how the monitor drives the hardware and,
2. Compare this to the way the diagnostics drive it.
3. Identify the portions of the data base related to a given problem and,

4. Use FILDDT to examine these parts of the monitor's data base to extract information related to the problem.

5. Use (or create) the tools to determine which of the devices are causing system problems (even when the diagnostics identify none).

6. Determine what is/was happening with/to the hardware.

COURSE RESOURCES

Each student should be given a copy of:

1. The course materials book including:
    1. The Student Guide
    2. The Modules
    3. The Monitor Tables
    4. The Monitor Logic Manual

2. DECsystem-10/DECSYSTEM-20 Hardware Reference Manual
3. TOPS-10 and TOPS-20 SYSERR Manual
4. TOPS-20 Microfiche Assembly Listing

COURSE OUTLINE

Software Introduction

I. Review of Operating System Principles
   A. Approaches
   B. Functions
      1. Scheduling
      2. Storage Management

   C. Virtual Memory
      1. Paging

   D. File System
   E. Interrupt Handling
   F. Accounting

II. TOPS-20 Hardware/Software Interface
   A. Virtual Address Translation - General
   B. User Page Map
   C. Hardware Page Table - Addressing
   D. Pointer Types
   E. Storage Addresses
   F. KL Paging
   G. Process Overhead Pages
   H. Special/Shared Page Table (SPT)
   I. Summary of Paging

Monitor Overview

I. Monitor Calls

II. Storage Management
   A. Block Diagram

III. Pager
   A. Hierarchical Storage Considerations
   B. Implementation - Mapping
   C. Inter-Level Data Flow
   D. Updating Lower Levels

IV. Scheduler

V. File System
   A. Data Structure

VI. Job/Fork Structure

VII. Disk And Magtape Service
   A. Hardware Principles
   B. Monitor Modules
   C. I/O Requests

VIII. Front End Service
   A. TTY Input
   B. TTY Line Buffers And Echoing
   C. Line Printer Output

IX. Appendices

Coding Conventions

I. Using MACSYM
   A. Symbol Definitions
   B. Macros To Manipulate Field Masks
   C. Instructions Using Field Masks
   D. DEFSTR -- MSKSTR Data Structure Facilities
      1. LOAD
      2. STOR
      3. Examples

   E. Subroutine Conventions
   F. Named Variable Facilities
   G. Miscellaneous

II. TOPS-20 Coding Standards
   A. Subroutine Calling - JSYS
   B. Subroutine Calling
   C. AC Definitions
   D. AC Saving and Restoration
   E. Subroutine Documentation
   F. Multi-line Literals
   G. Numbers

III. Appendices

SYSERR

I. The SYSERR Program
   A. Running the SYSERR Program
   B. Examples of SYSERR Output

II. SYSERR Module Internals
   A. SYSERR Block format
      1. Header
      2. Data

   B. Creating a SYSERR Entry
   C. The Job Ø SYSERR Task
   D. The SYSERR JSYS

Troubleshooting

I. CTY Output
   A. Explanation of KLERR Output
   B. Sample KLERR Output

II. Getting a DUMP
   A. How To Get a Dump
   B. Where BOOT Lands

III. SYSERR
   A. Overview of SYSERR Functions and Data Base
   B. Queued SYSERR Blocks In A Crash
   C. Moving SYSERR Blocks From a Crash To ERROR.SYS

IV. BUGHLT
   A. BUG Macro
   B. BUGHLT Contents

V. Push Down Lists And Related Data Bases
   A. How To Look At a Stack
   B. Push Down List / Machine State
   C. Stack Usage For Local Storage
   D. Stack Adjustment

VI. Machine States and Relevant Data Bases
   A. PC Storage
   B. AC Storage

C. Fork Scheduled, Or Not
D. Fork NOSKED
E. Extended vs. Non-extended Addressing
F. Sizes (Resident, Non-resident, Total)
G. MDDT Page
H. Relevant Data Base for Each Machine State

VII. DDT's
A. FILDDT
B. Relevant DDT/FILDDT Commands
C. MDDT
D. EDDT

PHYSIO - Disk/Tape

I. PHYSIO
A. Data Structure
B. Queueing an IORB
C. Scheduling an IORB
D. Starting I/O
E. Interrupt Handling

II. Disk Allocation
A. Data Structure (DSKBTTBL)
B. Space allocation
C. Space Deallocation
D. Drum Allocation
E. BAT Blocks

III. DISK Dependent I/O
A. Data Structure
B. Disk-Dependent Code
C. Disk Interrupts
D. Disk Errors and Abnormal Conditions

IV. MAGTAPE Dependent I/O
A. Magtape Data Base
B. Magtape IORB
C. CDB, KDB, and UDB
D. Interface to PHYSIO
E. Magtape I/O Wait
F. CLOSF Device-Dependent Functions
G. Magtape Interrupts
H. Error and Abnormal Conditions

Front End/Terminals

I. TTY/PTY Device-Dependent Code
   A. TTY Data Base
   B. TTYIN - TTY-Dependent Input
   C. TTYOUT - TTY-Dependent Output
   D. TTCH7 - 20 ms. Overhead Task

II. DTE Device-Dependent TTY Code
   A. DTE Data Base
   B. DTE Terminal Output
   C. DTE Interrupts

Storage Management

I. Storage Management
   A. Introduction
   B. Data Structures
   C. CST Tables
   D. SPT and Parallel Tables
   E. Working Set Management
   F. System-Wide Page Management
   G. Page Faulting
   H. Adjustment Of the Balance Set
   I. SWPIN and SWPOUT

II. JSB/PSB Space
   A. Context Switching the JSB and PSB
   B. JSB and PSB Maps
   C. Use of JSB Space
   D. Use of PSB Space

This page is for notes.

COURSE EVALUATION FORM

After finishing the course, please complete and return this form to the course instructor.

Additional comments, whether they deal with a specific question on the evaluation or are of a general nature, are most helpful to Digital in improving courses.

1. The most valuable aspect of this course was...

2. The least valuable aspect of this course was...

3. If you were to add or delete material to or from the course, what would it be? Why?

Please make any additional comments you wish on the next page.

Additional Comments:

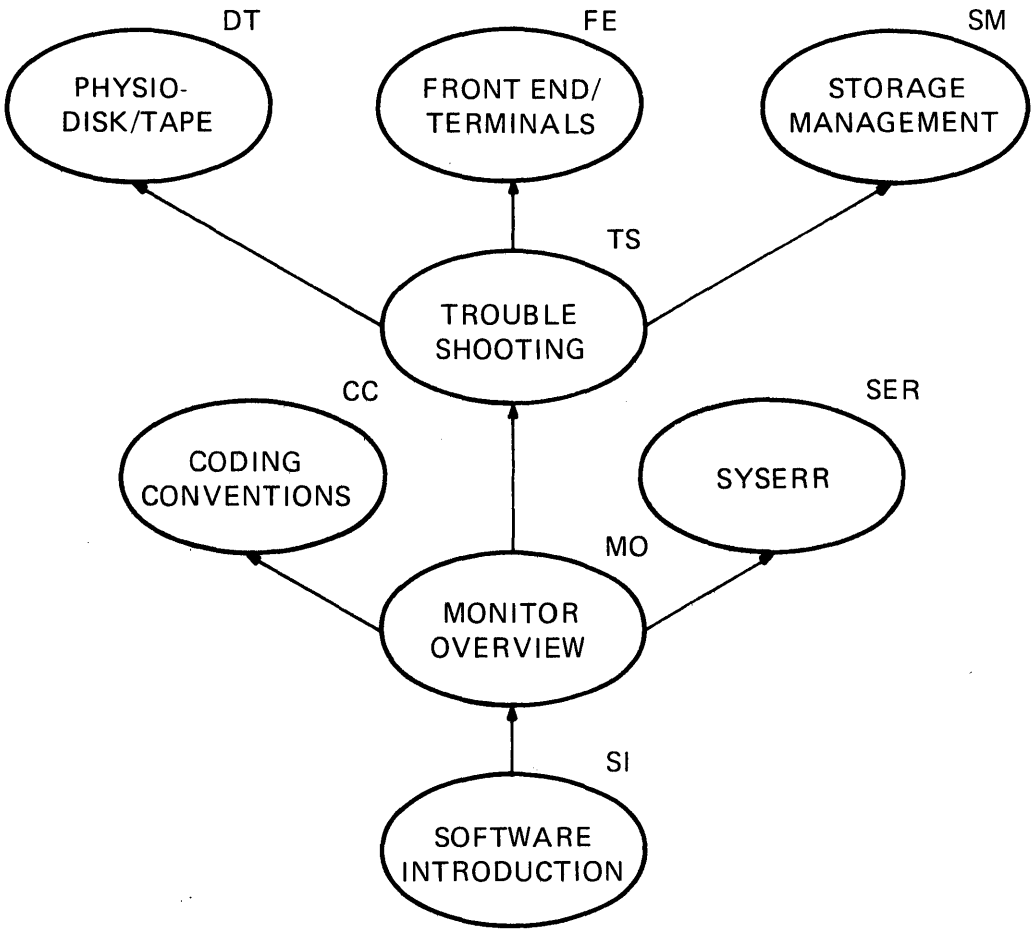
COURSE EVALUATION (cont.)

For each module and question intersection in the grid below, insert a number from 1 to 5, where 1 indicates "very little" or "poor" and 5 indicates "very much" or "excellent".

Questions:	Modules: SIMOCCSERTSDTFESMALL
My effort	
Demand on my time	
Content matched my need to know	
Learning objectives met	
Reference materials	
Module exercises	
Module labs	
Module test	
Enough time to cover material	
Did module stimulate ideas?	
Overall quality	
Did I meet prerequisites?	
Would I recommend this course?	

APPENDIX A

COURSE MAP



MR-2717

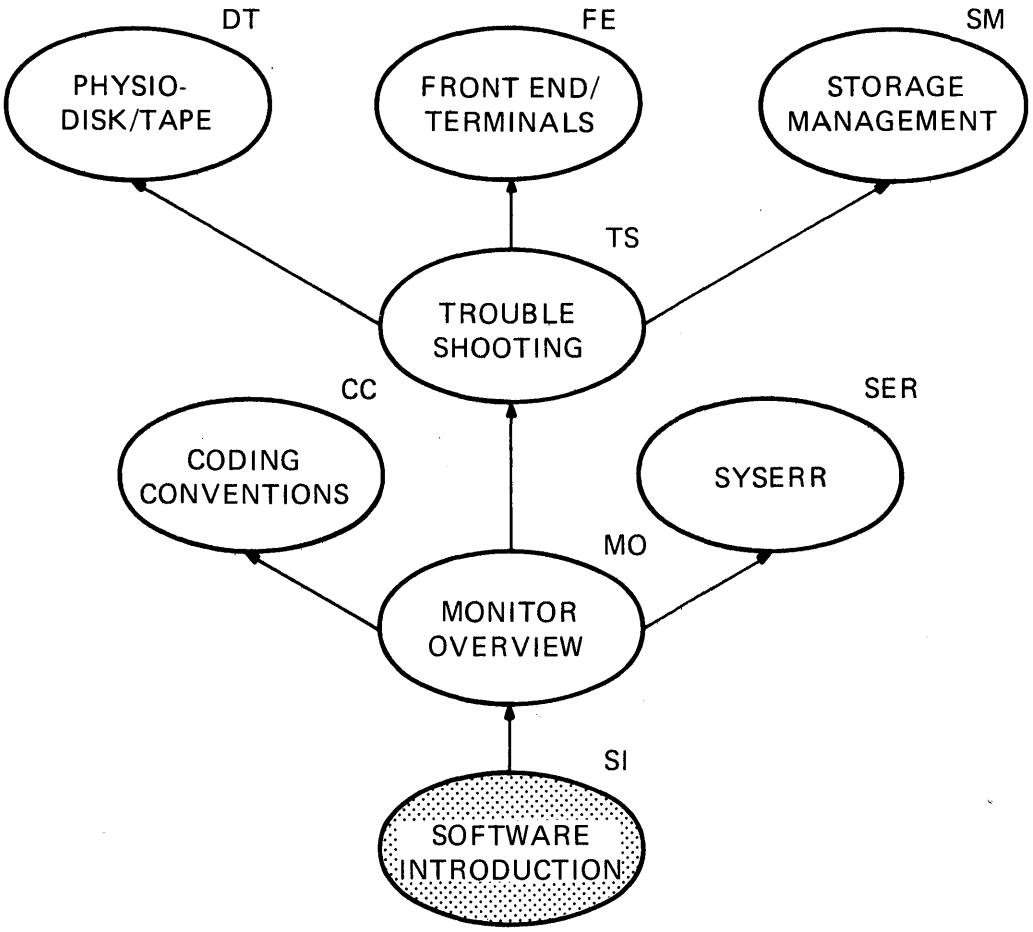
This page is for notes.

TOPS-20 MONITOR

Software Introduction

<<For Internal Use Only>>

COURSE MAP



MR-2717

This page is for notes.

Software Introduction

INTRODUCTION

Over the years the operating system has evolved into what some people say is the most complex piece of software ever devised. However, it is not fair to speak of operating systems only in terms of software because today hardware is often designed with particular aspects of the operating system in mind.

The operating system acts as the interface between the computer user and the hardware. The system must perform many tasks including multiprogramming, scheduling, memory management, file management, spooling, and device handling. In so doing, the operating system provides various services to computer users. Several types of operating systems have been developed to handle different user needs. These include batch, timesharing, and real time systems.

This module briefly reviews the various types of operating systems and the functions they perform. The emphasis, however, will be on time-sharing systems such as TOPS-20. Many functions will be related directly to hardware features (such as address translation and device handling) to give you a feel for the operating system's role in relation to the hardware. TOPS-20 virtual address translation will be covered in detail in this module.

The operating system not only serves the users and controls the hardware, it also detects and reports error conditions in the machine. To understand what caused an error which is reported by the operating system reports, you must understand what the operating system expects of the machine.

LEARNING OBJECTIVES

Upon completion of this module, the student will be able to:

1. Discuss the philosophies of batch, timesharing, and real-time operating systems.

2. Describe the following functions of an operating system:

    1. Scheduling

    2. File organization

    3. Memory management

    4. Device handling

    5. Accounting

3. Describe the functions of, and the actions performed by, the following portions of the virtual memory paging system:

    1. The Microcode

    2. A Page Map

    3. The Hardware Page Table

    4. The Three Pointer Types

    5. Storage Addresses

    6. The Base Registers

    7. The Special/Shared Page Table (SPT)

    8. The User and Executive Process Tables

MODULE OUTLINE

Software Introduction

I. Review of Operating System Principles
   A. Approaches
   B. Functions
      1. Scheduling
      2. Storage Management

   C. Virtual Memory
      1. Paging

   D. File System
   E. Interrupt Handling
   F. Accounting

II. TOPS-20 Hardware/Software Interface
   A. Virtual Address Translation - General
   B. User Page Map
   C. Hardware Page Table - Addressing
   D. Pointer Types
   E. Storage Addresses
   F. KL Paging
   G. Process Overhead Pages
   H. Special/Shared Page Table (SPT)
   I. Summary of Paging

This page is for notes.

REVIEW OF OPERATING SYSTEM PRINCIPLES

An operating system can be summarized as a manager of resources. These resources consist of the processor, the memory, disk storage, and the attached devices. The operating system must give some portion of these resources to each process that runs on the machine; it must schedule the use of these resources when several processes are competing for them and collect them when a process is finished.

SMALL SYSTEMS

A small operating system provides basic services for the user's programs. These include device handling routines and a simple scheduling approach (the allocation of resources may simply consist of giving everything to the executing process). The number of devices supported is usually small and the variety of devices is restricted.

LARGE SYSTEMS

A large operating system provides a wide range of services to the user. A sophisticated scheduler is used with resources shared or divided among several active processes. Protection for the system is substantial and the accounting keeps track of nearly everything. A large number of various devices are supported with a sophisticated file manipulation capability built in. Network communications between computers are often supported.

There is no clear dividing line between large and small systems. Almost any operating system may have some attributes which are characteristic of larger systems and some which are not. Both TOPS-10 and TOPS-20 would generally be considered large systems.

Approaches

In addition to coming in different sizes, operating systems have various philosophies on the treatment of users and on the processing of jobs. The variety of philosophies results from computer users' differing ideas on what to do with a computer and varied types of processing, each making different demands on processors. Some users want a system capable of reasonable turnaround time on jobs that require a great deal of input or output (I/O bound) or require long arithmetic processing (compute bound). Others want a computer system designed to service a large number of concurrent users in an interactive environment. Still others want a system to use as a command and control device to regulate machinery and respond to changing situations when reaction time is limited.

BATCH

A batch operating system consists of three components: spooling system, scheduler, and dispatcher. These components work together to get a user job through the system. In an operating system that combines batch and timesharing, as in TOPS-1Ø and TOPS-2Ø, these components are slightly modified and not very distinct. The batch system for these two operating systems is called Galaxy.

TIMESHARING

A timesharing operating system is designed to give service to a considerable number of users at the same time. The object is to allow these users quick and frequent interaction with the computer in such a way that each user has the illusion of having the exclusive attention of the whole system. Since one processor cannot actually be handling more than one user job at a time, a service approach very different from that of a batch system must be used to produce this illusion. Time is shared, that is, instead of allowing a few users to monopolize the processor for long periods of time as might happen under batch, each user is given a small amount of time on a regular basis. The amount of time is called a time slice. If a strict

rotation of users is followed and there are N users, each user gets the processor 1/Nth of the time. This situation is referred to as round-robin processing.

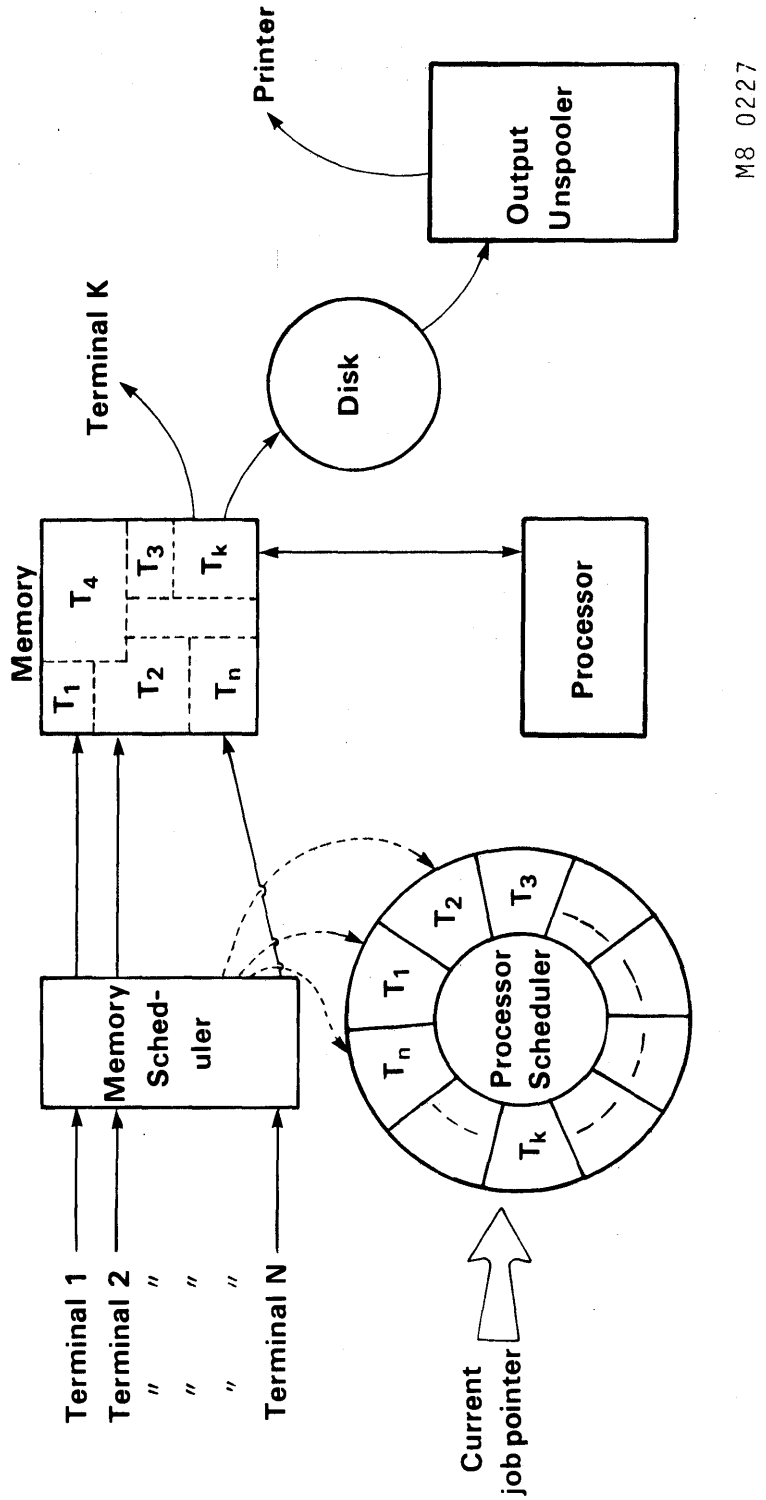


Figure SI-1. Timesharing Operating System

A large timesharing system necessarily requires a great deal of overhead, due to the fact that the system is performing so many functions for a large number of concurrent users. Their jobs may be short but that does not mean they are small. Memory management is a prime consideration with many users on the system. Changing from job to job, accounting, and communications between jobs all require time and space. Consequently, few timesharing systems can support a large number of users without visibly degraded response time.

Both TOPS-10 and TOPS-20 are principally timesharing operating systems. The Galaxy batch control system is added to form a hybrid system. On TOPS-20 and TOPS-10, batch jobs are treated the same as timesharing jobs (with a few restrictions) as far as the operating system is concerned.

In order to enable the operating system to treat the batch jobs in a manner similar to timesharing users, TOPS-10 and TOPS-20 set up pseudo-terminals (PTYs) for them to use. These are software simulations of terminals which the operating system treats much like real terminals. By using the pseudo-terminals, the commands in the control file can be handled in the same way as those typed at a real terminal. With a batch job, the complete text of the session is maintained in a log file for future reference by the programmer.

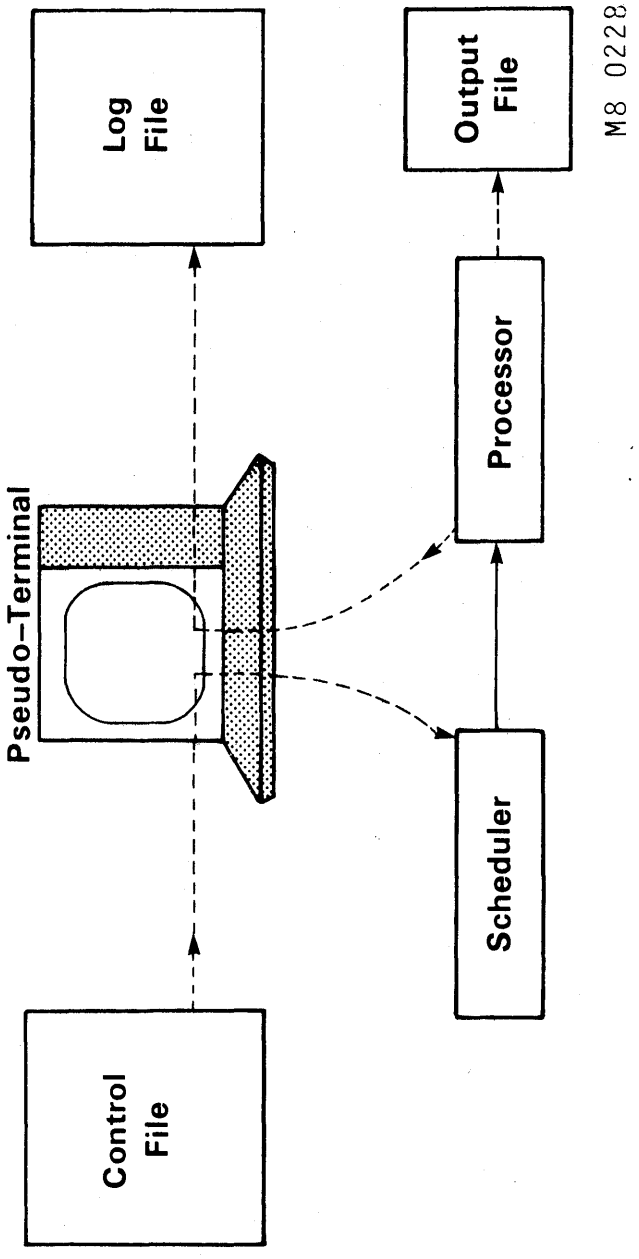


Figure SI-2. Use of Pseudo-Terminal

REAL TIME

A real time system can be organized in a manner similar to a timesharing system. However, there is one big difference -- the response time for the system cannot be allowed to degrade. Inputs to the system can be scanned in a round-robin-like arrangement as with timesharing; however, responses must be short. The processor cannot be tied up by any particular job for so long that processing needs of other jobs are not met.

When real time systems are combined with batch or regular timesharing, the real time functions must always be given priority. The operating system must maintain control if it is to provide continued service to all users. Necessarily then, the operating system must protect itself from the users and all users from each other.

OPERATION MODES

A simple way to protect the operating system from users is to have more than one operation mode. TOPS-1Ø and TOPS-2Ø have two modes: EXECUTIVE mode and USER mode. Only the operating system may run in executive mode and, as such, it can issue any instruction which the machine is capable of executing. All users run in user mode and must request certain actions to be done for them by the operating system. Only certain monitor calls and context switching require executive mode.

Functions

MEMORY ALLOCATION

One of the problems with maintaining several jobs at once is managing the memory. Since jobs vary greatly in size, there is a problem in fitting each into a fixed size memory. In scheduling the job to be run, the scheduler must determine if there is available space. One way of handling the allocation of memory is to force jobs to use memory in fixed size units called pages. Under this scheme, memory is divided into pages (each with the same number of contiguous words) and the allocation for each job is a number of pages.

On TOPS-2Ø, for example, a page consists of 512. words.

SCHEDULING

A portion of an operating system is its scheduler(s). The use of all resources in the system must be scheduled and these schedules are interrelated. On TOPS-2Ø a job may be composed of several processes (forks) and it is these processes which are scheduled.

PROCESS STATES

Processes can be placed into three categories: running, ready, and blocked. The running process is the one currently executing on the processor; for each processor there is only one running process.

A ready process is one that can be run but is not executing at the moment. Ready processes are those which have all their currently required resources ready. This implies that memory is allocated to them and devices assigned. There are usually a number of ready processes from which the running process is chosen.

A blocked process is one that has some or all of its resources but still requires something before it can be considered ready. The most frequent reason for being blocked is an I/O WAIT (i.e., the process is waiting for input or output to occur).

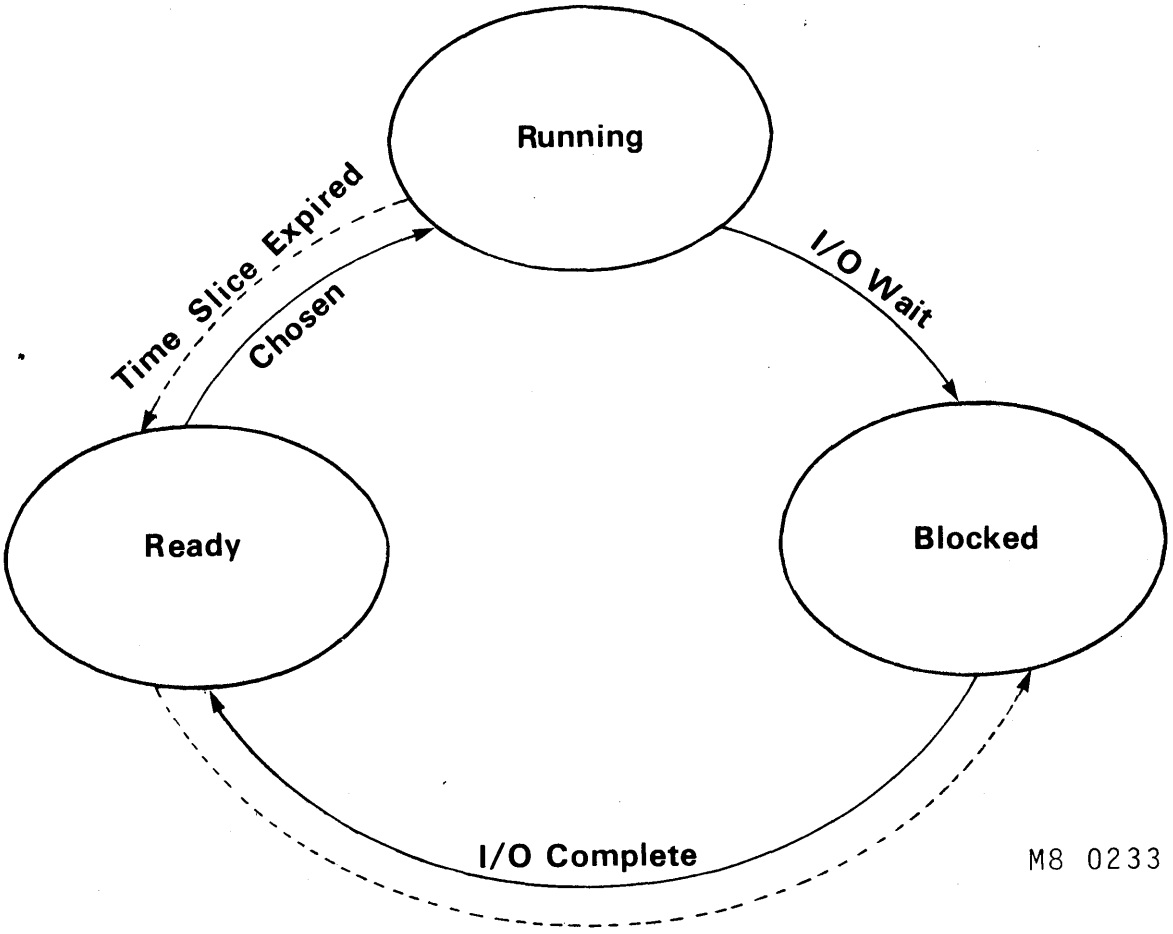


Figure SI-3. Process State Transitions

TOPS-2Ø uses the term Balance Set to refer to a group of processes (initially selected from the group of ready processes) which include the running process, some of the ready processes, and possibly some blocked processes. The group is called "balance" set because an attempt is made to balance the usage of physical memory with the needs of all the processes trying to use the processor. The blocked processes in the balance set are waiting for disk I/O and are included because they will be blocked for only a short time. The TOPS-2Ø scheduler selects the process to be run in the next time slice from among the processes in the balance set.

STORAGE MANAGEMENT

We have already discussed some of the problems of storage management connected with memory allocation and the restriction of a user's addressing capabilities. One way to solve or simplify many of these problems is to use virtual memory.

Virtual Memory

To understand virtual memory, we must look at the address space of a process. The address space is the size of physical memory (number of locations) which can be addressed using the machine instructions. The size of the address space is thus primarily dependent on the number of bits in the address portion of an instruction. On DECsystem-1Ø and DECSYSTEM-2Ø the address portion of an instruction, when used to access physical memory, is 18 bits long. Extended addressing adds 5 bits. Thus, 2**23, or 8388608, locations can be addressed. The usual size of physical memory is 256K or 512K.

This virtual address capability is independent of programs. Thus, the virtual memory can be considerably larger than the physical memory. Programs can be compiled or assembled and linked as though the entire virtual memory were available to each program. However, the loading of an entire program would be a problem because each program currently scheduled to use memory would have addresses ranging from zero to the size of the program (but there is only one address zero in physical memory).

One way to overcome this problem is to modify (relocate) each of the addresses according to its location in physical memory. Another solution to the loading problem is to use a mapping function to relate the virtual memory addresses within the executable module to the physical memory addresses. Instructions can only be executed when they are in physical memory. Consequently, the mapping must be applied to each instruction as it is executed. This mapping can be done with hardware assistance using process tables so that all programs can appear to begin at address zero.

There are several advantages in employing virtual memory address translation. First, virtual memory does not need to be mapped into contiguous sections of physical memory. If physical and virtual memory are both divided into pages, the virtual memory pages of a program can be mapped into pages scattered throughout physical memory.

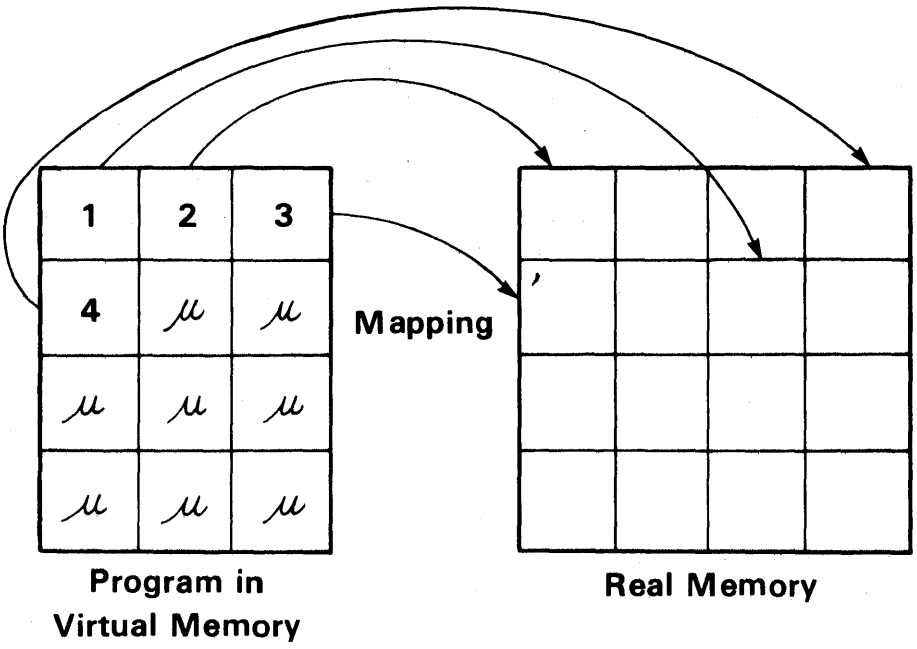


Figure SI-4. Virtual Memory Mapping

Second, only those pages of the program that are actually needed for execution have to be in physical memory. When more pages are needed, they can be placed anywhere, and when some pages are no longer needed, they can be removed. When a removed page needs to be in physical memory again, it can be placed anywhere. The address translation associated with virtual memory results in the program being executed as though the whole virtual memory were real, physical memory.

TOPS-10 and TOPS-20 both have virtual memory as an integral part of their operating systems. Each process has its own page table set up by the operating system and used by the hardware to translate virtual memory references into physical memory locations. However, the management of memory at the logical level in the two systems is quite different. TOPS-20 employs demand paging, whereas TOPS-10 uses swapping.

PAGING

The use of virtual memory with paging requires an effort to keep track of where the various pages of a process are. Some pages may be in physical memory, others on disk. When the process references a page that is not in physical memory, it must be fetched from the disk and placed into a free page frame in physical memory. The referencing of a non-resident page is referred to as a page fault. The detection of a page fault and the retrieval of the page are handled by the hardware (using the page table) and the operating system working together.

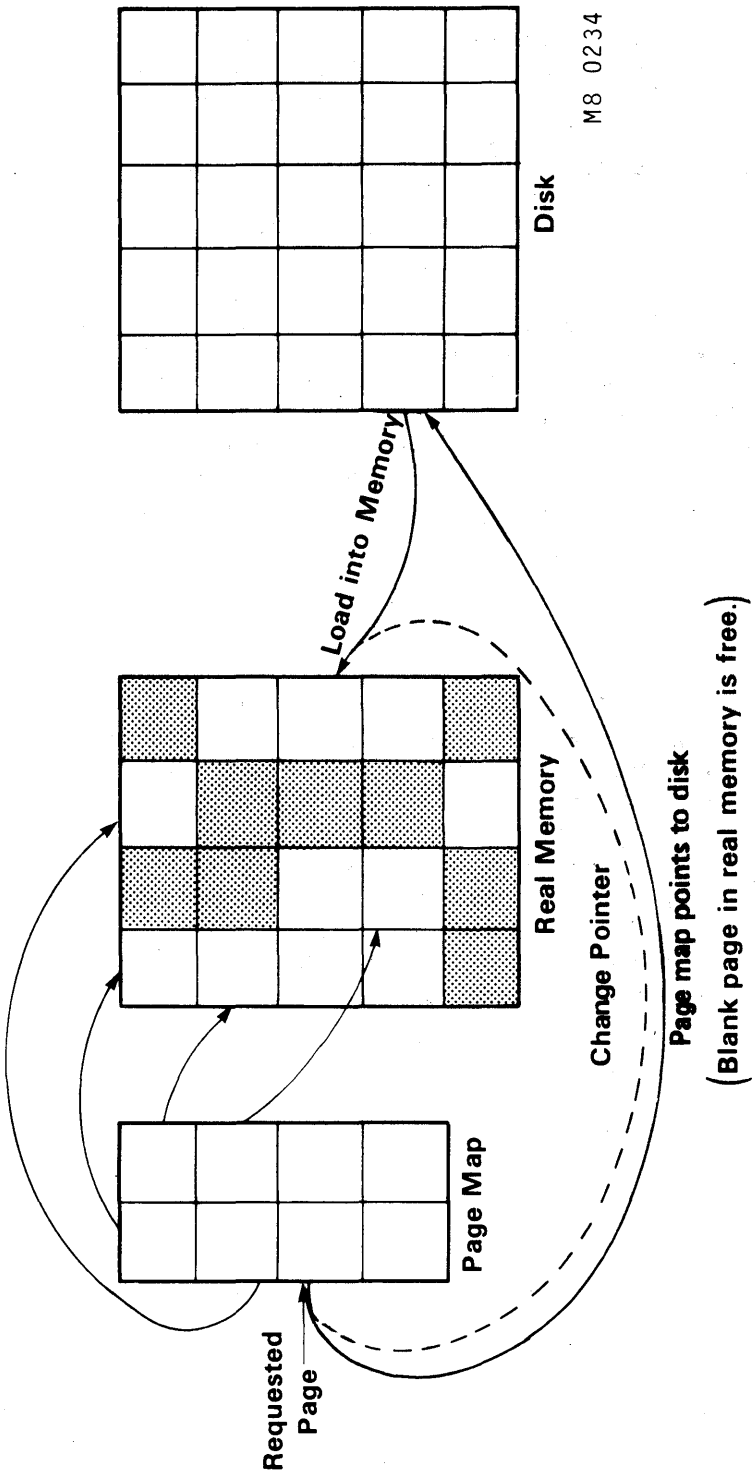


Figure SI-5. Page Fault Action

Virtual memory lends itself easily to the sharing of physical memory among processes. All that is necessary for two processes to share physical memory is for their page tables to map some virtual page to the same physical memory page. Since the use of sharing takes place dynamically (as the processes run), the operating system has to set up the sharing. The operating system fills in the page table entry and adjusts the physical memory usage statistics to show the page is shared.

A full presentation of TOPS-20 paging appears later in this module.

File System

The operating system maintains a list of all users of the system by directory name. This list is kept in a system directory which, on TOPS-20, is called ROOT-DIRECTORY. For each user in the system directory, there is a pointer to an index block for the user's directory. This index block consists of pointers to the various pages of the user's directory which are on disk. In the directory there is a file descriptor block for each file specifying owner, protection, date of creation, etc.

The storage of files is similar to that of user directories in that each file in the directory has a pointer to an index block. This index block contains pointers to the pages of the file, which may be scattered anywhere on the disk. If the file is so large that all the pointers to the pages will not fit into one index block, two levels of indexing can be used. With two levels of index blocks, the directory entry points to a super index block which contains pointers to index blocks.

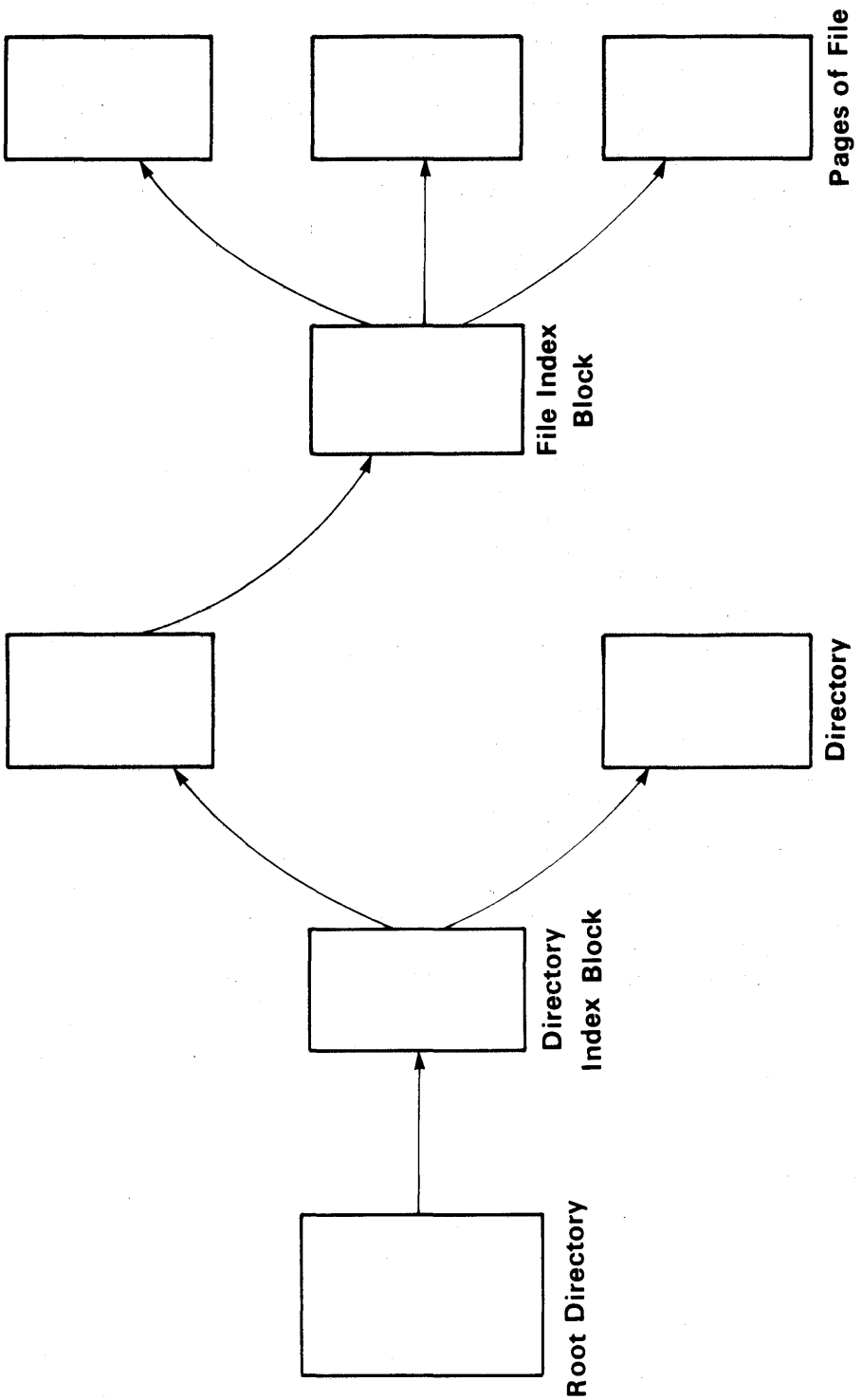


Figure SI-6. Access to a User File on TOPS-20

Interrupt Handling

THE INTERRUPT SYSTEM

The operating system must control and provide service to a wide variety of peripheral devices. Each of these devices interrupts the processor whenever it requires some service or has completed some action. Thus, the operating system can be bombarded with interrupts.

When several devices are competing for the processor's attention, the physical arrangement of the devices and their relative importance determines which device gets serviced first. Interrupt priorities are assigned to each device. The occurrence of multiple service requests is not an uncommon event; however, the different priorities immediately resolve these conflicts.

Figure SI-7 shows how normal processing can be suspended to service interrupts on several levels. Higher priority interrupts disrupt service to lower priority ones and lower priority interrupts must wait for the completion of higher priority interrupts.

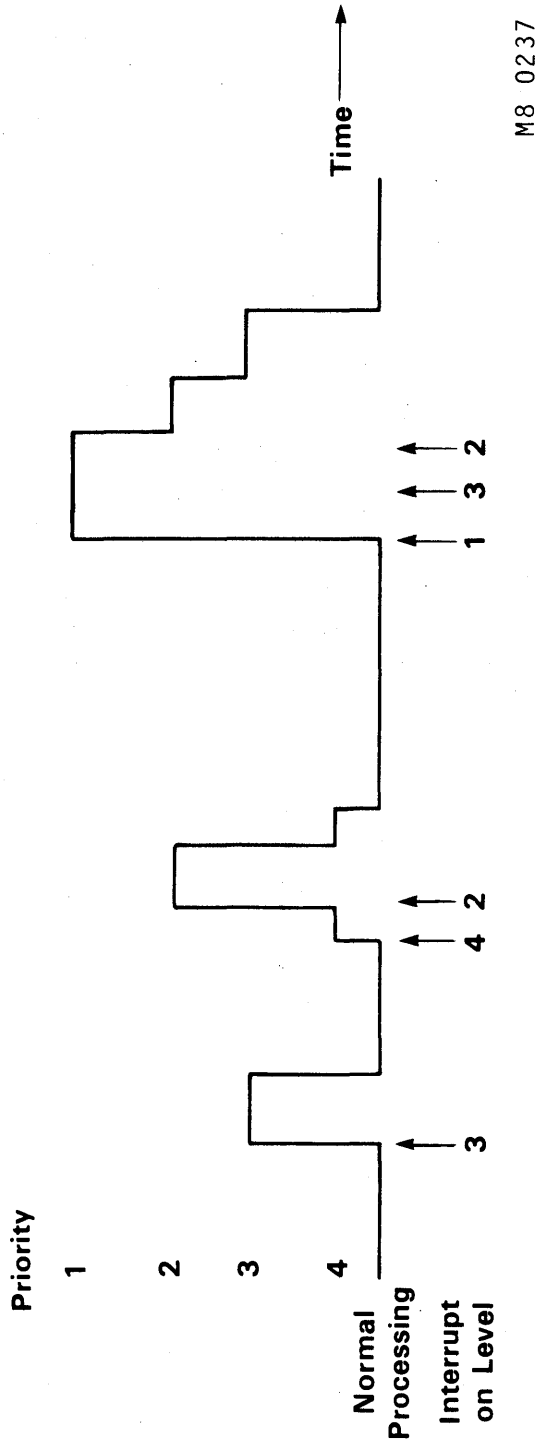


Figure SI-7. Priority Interrupt Processing (1 Highest Priority)

For the most part, interrupt service is the acknowledgement of some event (an I/O completion, an error, or a status change). An interrupt is usually a response to some request by the operating system for a device operation.

THE DISK MANAGER

Service to the disk requires special recognition since the operating system relies so heavily upon this device. Access to files, spooling, unspooling, compiling, loading, paging, swapping -- all require access to the disk. Thus, it is common for there to be several pending requests for access to the disk. In most operating systems, a disk manager determines the order in which these requests are processed.

The disk manager orders the requests in a queue for disk access so that average seek time (the time to find the right cylinder) and average latency time (time for the disk to rotate so that the desired sector is under the head) are minimized. The work of the disk manager is time critical because new requests are coming in constantly and the disk is always rotating.

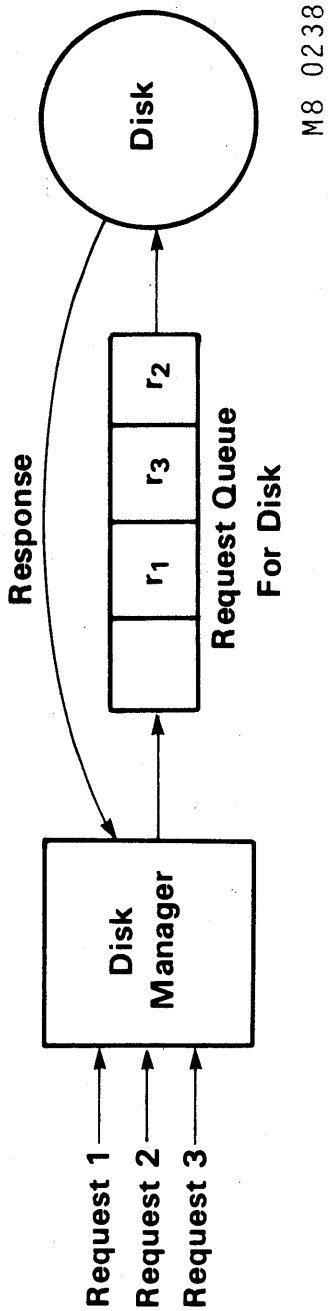


Figure SI-8. Disk Manager Activity

Accounting

It is essential for the people who pay for the computer system to know who is using it and for how long. Most operating systems have accounting facilities built in to keep track of the various ways that the computer is used. These facilities keep account of such things as processor time used, terminal connect time, amount of disk storage used, number of pages of printed output, and number of cards read. TOPS-1Ø and TOPS-2Ø accounting systems can keep track of all these things and can produce reports that can be used to charge the user.

Accounting systems charge different amounts for each kind of usage. Also, the accounting systems have the ability to treat each user separately so that individual rates, discounts, etc. can be applied. Most systems also have the ability to add charges for such things as consulting services and terminal rental, if appropriate.

TOPS-20 HARDWARE/SOFTWARE INTERFACE

Virtual Address Translation - General

The 18-bit virtual address is considered as two parts: the high-order nine bits are considered the page number, and the low-order nine bits are used as an index into a page. Remember that a page is 512. words long, and nine bits addresses 512. entries (i.e., words 0. through 511. in a given page).

A virtual address must be resolved to a physical address in the machine, and each address in physical memory is in one of the physical pages of memory. The hardware picks up the (9-bit) virtual page number, uses it as an index into a page table to determine where the physical page is, and then uses that physical page number (9-bits) along with the index from the virtual address to get the address in physical memory for the reference.

NOTE

Throughout this document the term "core" refers to physical memory, whether it is ferrite core or MOS semi-conductor memory.

Some of the characteristics of virtual memory are:

1. 256K of available memory for every user (even if the machine has less than 256K of physical memory)
2. Memory (virtual and physical) is divided into pages
3. Each page is 512. words long
4. Core (or MOS) is paged
5. The disk is paged

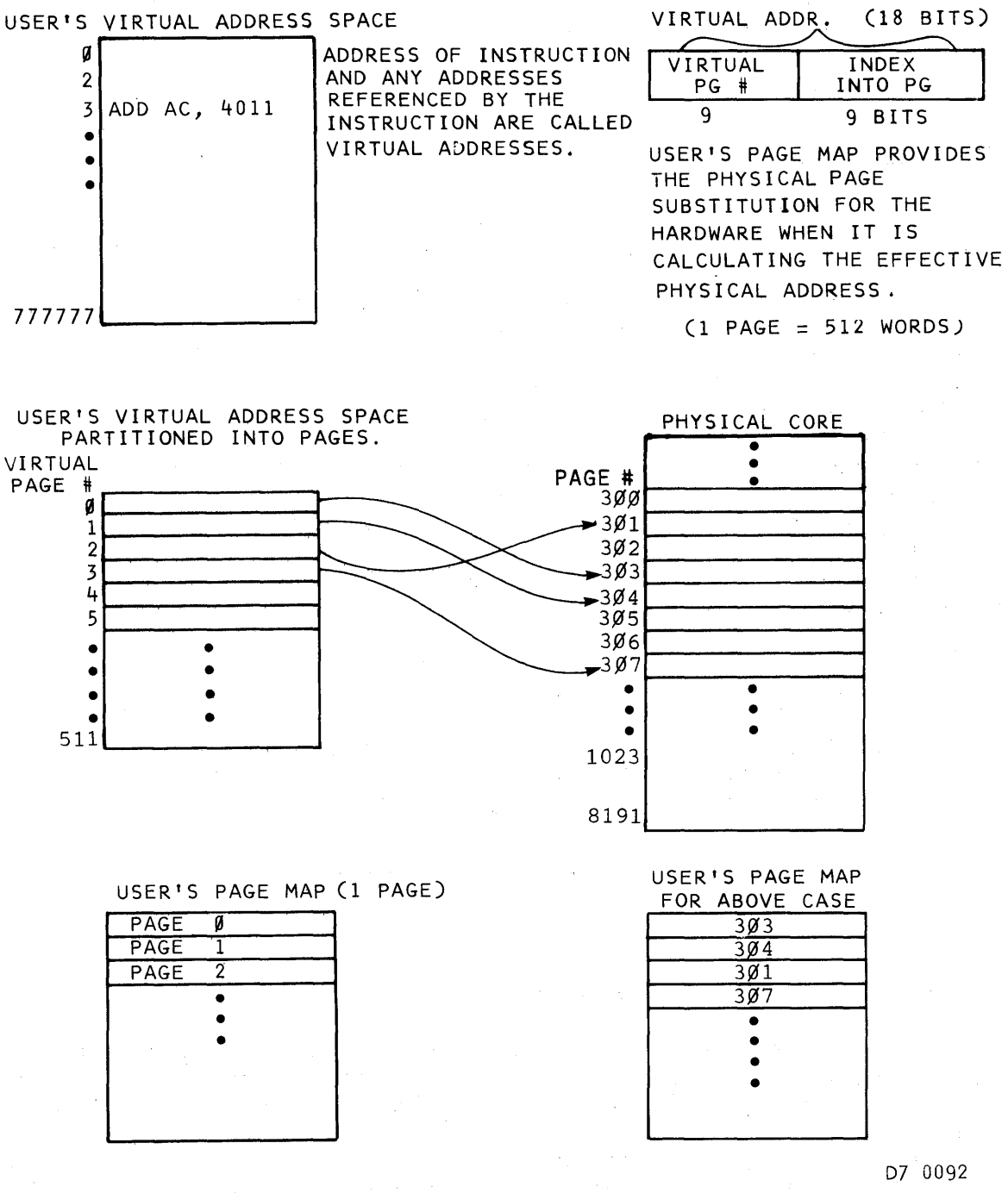


Figure SI-9. Page Mapping

Since each virtual page maps to a physical page, the hardware must have a means for determining where the physical page is. The hardware initially finds the mapping by using the User Page Map.

User Page Map

The User Page Map is a table 512. words long. Each entry in the table gives the hardware (firmware, microcode, etc.) the information it needs to determine where the information physically resides. The user's page map is one of the overhead pages which are guaranteed to be in core for each fork in the balance set. (The balance set is the group of processes which are eligible for running.) When the hardware first needs to resolve a virtual address for a user, the address is solved by getting the entry out of the user's page map. Note that in this first case, an extra reference to memory is necessary. The first reference is the only time this "extra" reference is necessary, since the hardware, once it gets the mapping the first time, puts that information into a "cache" of addresses called the Hardware Page Table.

NOTE

Actually, there are other times when the Hardware Page Table entry is invalid for some reason; these will be discussed later.

Hardware Page Table - Addressing

When an address reference occurs, the microcode first looks in the hardware page table to see if there is a valid entry for the specified page. If there is a valid entry, the memory reference is made to the physical page location gotten from the hardware page table (this is the number of the page in physical memory), and the index from the virtual address requested (this is used as the low-order nine bits

for the address). Otherwise, the microcode goes through the user's page table (in physical memory and requiring a physical memory reference) to get the physical page mapping. The mapping information is then placed in the hardware page table for future references.

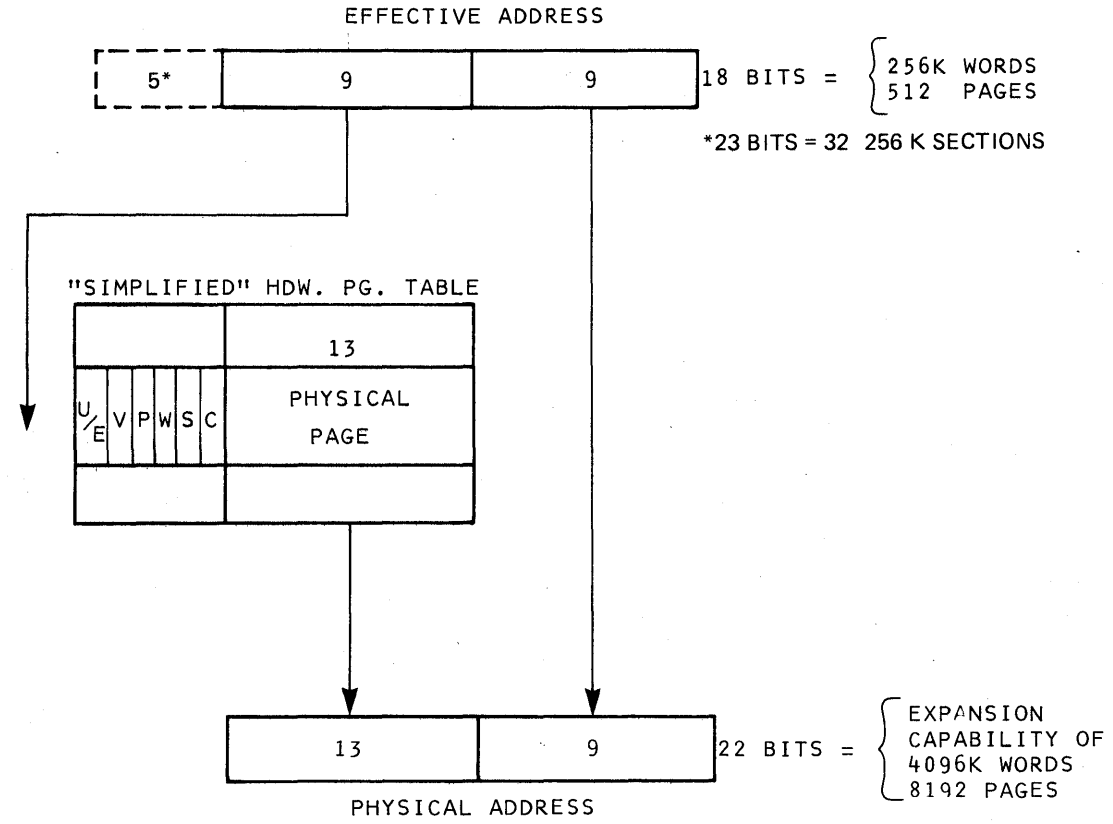
NOTE

In the KL-Model B processor, extended addressing is supported. In these machines, the page number information from the Hardware Page Table is composed of 13 bits of address data, the regular 9 bits, and the extra 4 for extended addressing of physical memory up to 4Ø96K words (8192 pages). The use of extended addressing will be discussed later.

In fact, there are two address spaces using the hardware page table simultaneously: the user, as we have just discussed, and the monitor. Since the hardware page table is only one page long, there is a high probability of conflict. To help out here, each entry in the hardware page table has a bit (user/exec bit) which tells the microcode whether the mapping information is either for the user space or monitor space. A further help is the fact that the monitor addresses are "hashed" so that, for example, page Ø of user space and page Ø of monitor space do not use the same hardware page table slot.

The entire hardware page table is cleared at context switch time. That is, when a different process is chosen to run, all of the mapping information in the hardware page table must be re-created.

Earlier we mentioned that the microcode first checks the Hardware Page Table to see if it contains a valid entry for the specified page. Along with the user/exec bit, and the fact that the table is cleared at context switch time, each page has an age stamp associated with it. Periodically these age stamps get incremented, and when this happens, the Hardware Page Table gets cleared.



U/E USER/EXEC MODE
V VALID
P PUBLIC/CONCEALED PG.
W WRITEABLE PG.
S SOFTWARE
C CACHEABLE PG.

* EXTENDED ADDRESSING INCLUDES A 5-BIT SECTION NUMBER.

D7 0088

Figure SI-10. Addressing Hardware Page Table

Pointer Types

What does an entry in the page table look like?

1. Each slot has access, user/exec, writable, cacheable bits.

2. Each slot has either an immediate pointer or a pointer to where to look next for the storage address.

3. There are three pointer types which the microcode understands:

    1. Immediate pointer -- storage address is here.

    2. Shared pointer -- the storage address must be gotten through the shared/special page table (SPT).

    3. Indirect pointer -- must look in the specified page table for the next pointer.

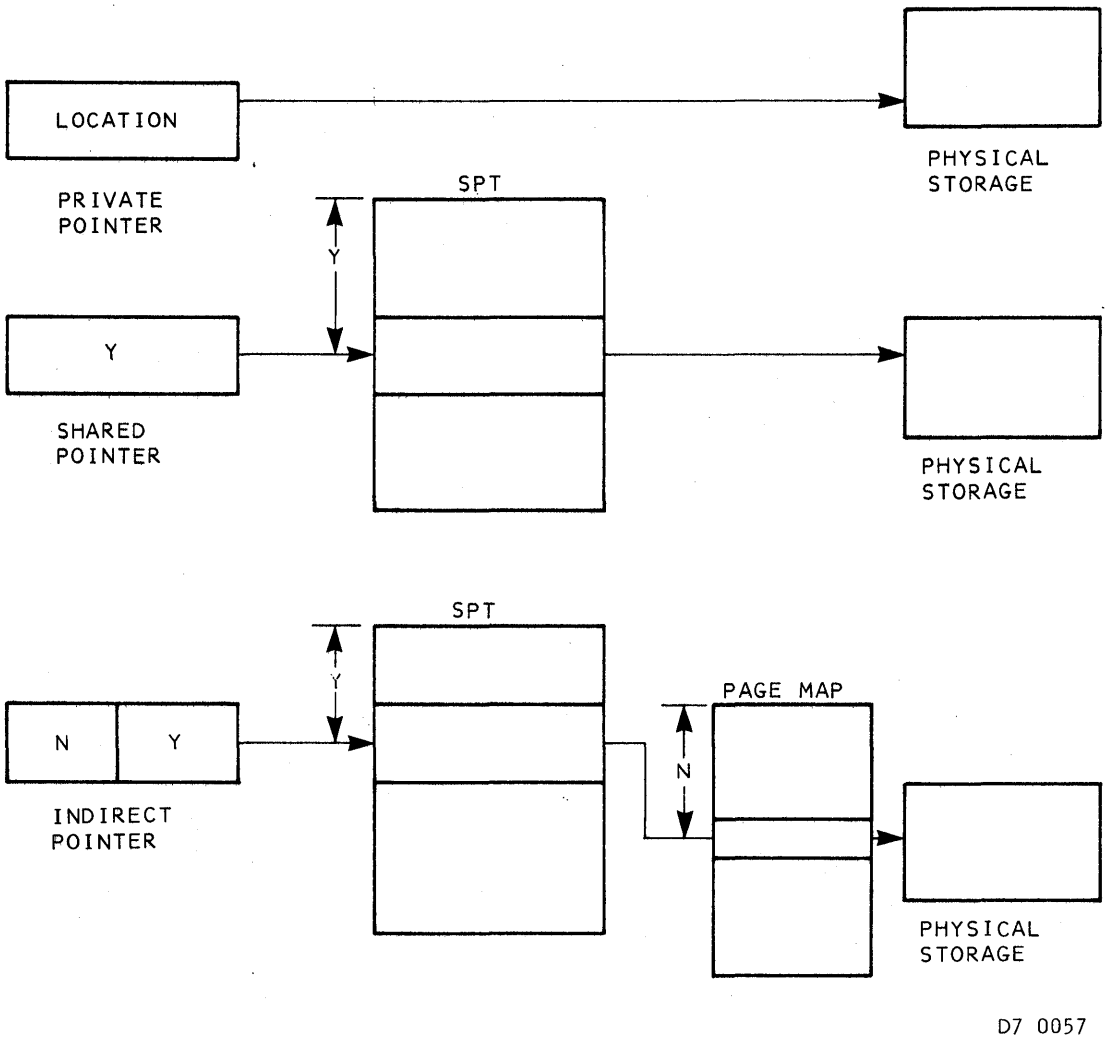


Figure SI-11. Pointer Types

- MEDIUM ON WHICH STORAGE EXISTS.
- LOCATION ON MEDIUM.
- CAN APPEAR IN A PAGE TABLE OR THE SPT.

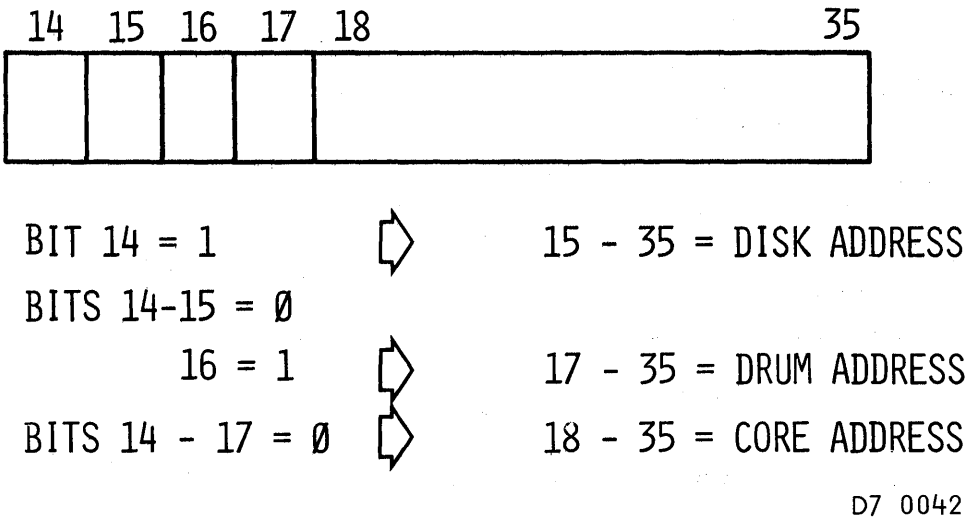


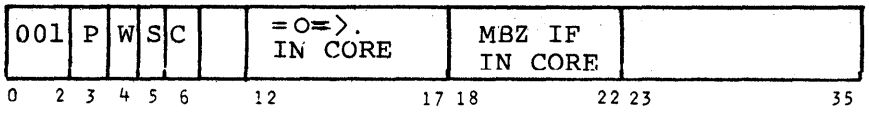
Figure SI-12. Storage Addresses

The pointer type is encoded in bits 0-2 of the page pointer The access bits are in bits 3-6. They are:

Pointer Type	
0	No Access
1	Immediate or Private
2	Shared
3	Indirect
4-7	Not Used, Reserved for future use by DEC

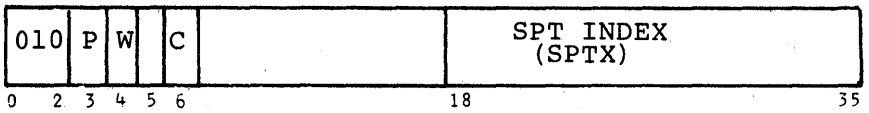
Access Bits	
P	Public/Concealed
W	Writeable
C	Cacheable
S	For Software

The immediate pointer holds a 13 bit physical page number in bits 23-35. This is also called a private pointer since it is private to the page table containing the pointer. This should not be confused with the public bit which describes the type of access allowed.



IMMEDIATE POINTER (CODE=1)

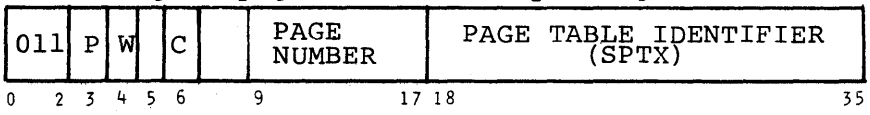
The shared pointer contains an index which addresses into the Special/Shared Pages Table (SPT). The SPT base register, SBR, (reserved AC block) points to the beginning of the SPT. The sum of the SBR and SPT index (SPTX) points to a word containing the storage address of the desired page. The line number from the virtual address is used to complete the reference.



SHARED POINTER (CODE=2)

Regardless of the number of page tables holding a particular shared pointer, the physical address is recorded only once in the SPT. Hence, the monitor may move the page with only one address to update.

The indirect pointer identifies both another page table and a new pointer within that page table. This allows one page to be exactly equivalent to another page in a separate address space. The object page is located by using the SPT index.



INDIRECT POINTER (CODE=3)

D7 0651

Figure SI-13. Page Pointers

Storage Addresses

Once the microcode determines the storage address, there is still work to do since the storage address may indicate that the page is not currently in core. There are three levels of storage for pages: core, drum, and disk. The microcode deals only with core storage addresses; if the referenced page is not in core, a page fault occurs and the monitor arranges for the desired page to be brought in. Remember that the term "drum" refers to the swapping space, which is, in fact, a reserved portion of the disk. Note that we are still looking for a whole page; we have not yet even considered the low-order nine bits of the requested virtual address. The format of a storage address is what the microcode uses to determine where a page is.

PAGE FAULTS

The term "page fault" indicates that, for some reason, the microcode was not able to access a page and had to call the monitor to make the page available. There are several reasons why a page may not be available:

1. The page is not in core.
2. The page is in core but marked to write out for replacement.
3. Null pointer -- the page does not exist.
4. Invalid age field.
5. Invalid access requested -- e.g., a write to a non-writable page.

STORAGE ADDRESS FORMAT

If bits 14-17 of a storage address are all Ø, the page is in core and bits 18-35 are the address. If bits 14-17 are not all Ø, the page is on disk or drum. In the latter case, bit 14 on indicates a disk address; otherwise, it is a drum address. If the storage address indicates memory,

the microcode copies the storage address into the hardware page table.

In summary:

1. The User Page Table has slots for all existent virtual addresses.

2. The entries in a User Page Table are either Ø (for non-used pages) or one of three types of pointers:

    1. Immediate

    2. Share

    3. Indirect

3. The translated address is copied into the Hardware Page Table if the Storage Address is core.

4. A page fault occurs if the Storage Address indicates a location other than core.

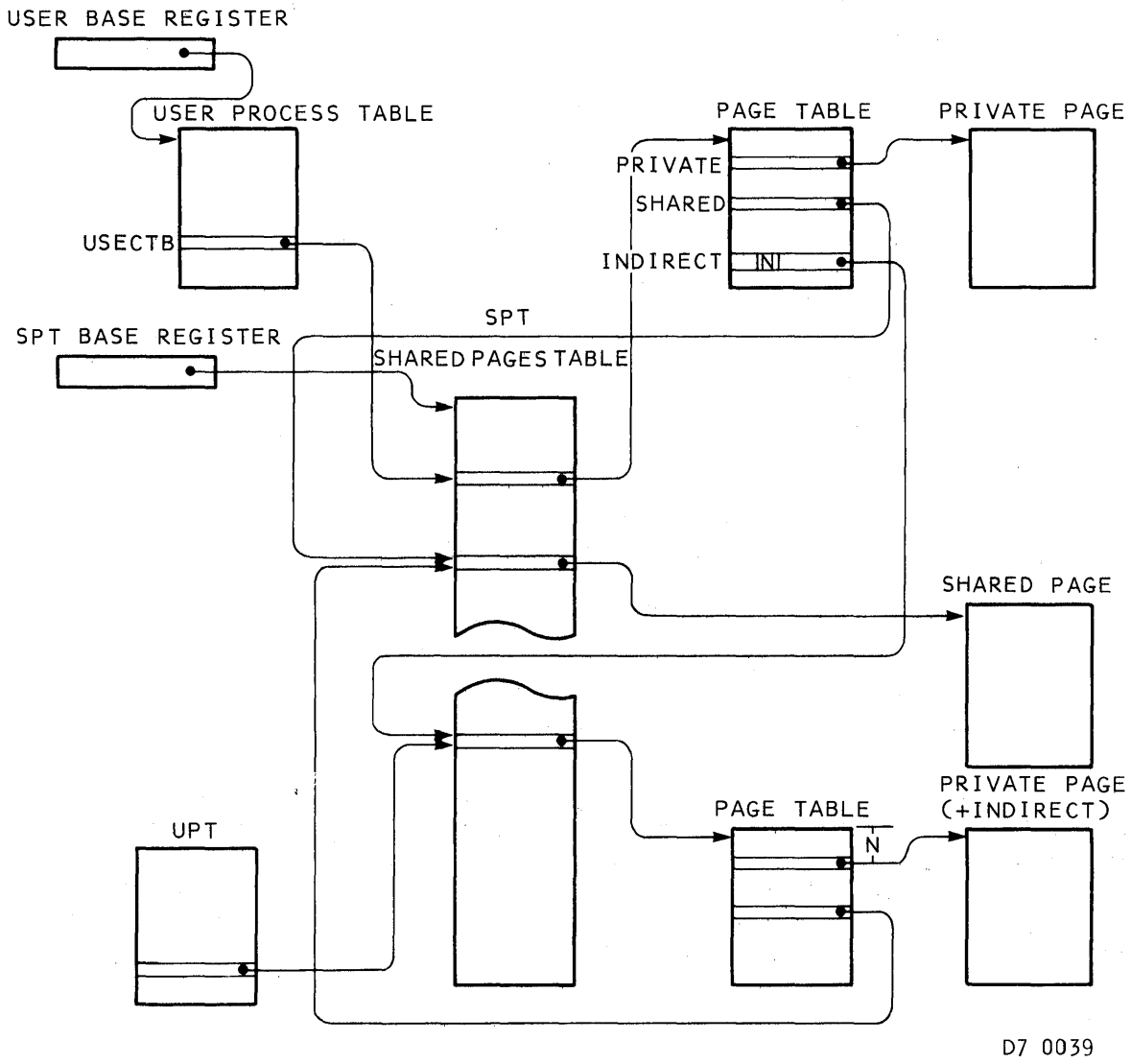


Figure SI-14. Layout of KL Paging

KL Paging

Two hardware registers are loaded for the microcodes' use: the Exec Base Register (EBR), which contains a pointer to the Exec Page Table (EPT), and the User Base Register (UBR), which points to the (UPT).

Process Overhead Pages

Each process needs several overhead pages: the UPT (two pages) and the User Page Map. The User Process Table (UPT) contains the information which the system needs to run a process. Included in the UPT is trap, context and scheduler information. Also in the UPT is a 32. word block (USECTB) which contains Page Map locations. There is one space allocated the page map of each of the possible 32. sections which may exist.

NOTE

At this time the TOPS-20 Monitor does NOT support extended addressing for users.

When the microcode needs to find a page for the user, the User Page Map is located from the section 0 (USECTB) slot in the UPT. (Remember that the physical address of the UPT has been put into the UBR.) A similar path is taken by the microcode for addressing in the monitor addressing space, except that the EBR is used. The EBR points to the EXEC Process Table which, in turn, has the map pointers at MSECTB.

Name: UPT

Description: User Process Table. A one page User Process Table is associated with the Scheduler and with each fork in the system. (Those associated with forks may be swapped out with the fork.) However, there is only one UPT known to the hardware/firmware at any one time. The UPT known is the one whose address is pointed to by the hardware User Base Register (UBR), which is set-up when a process is chosen to run.

The UPT contains the dispatch address for process events (i.e., traps) and the user's section map table.

Defined In: APRSRV

Referenced by: APRSRV, SCHED

FORMAT

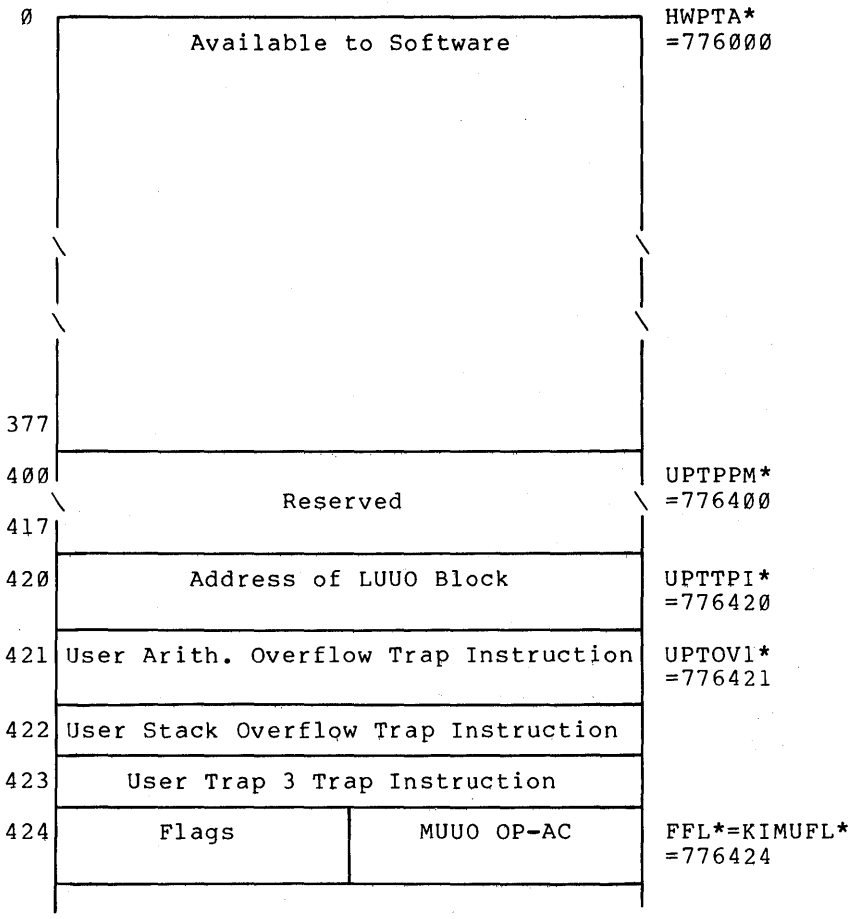


Figure SI-15. User Process Table

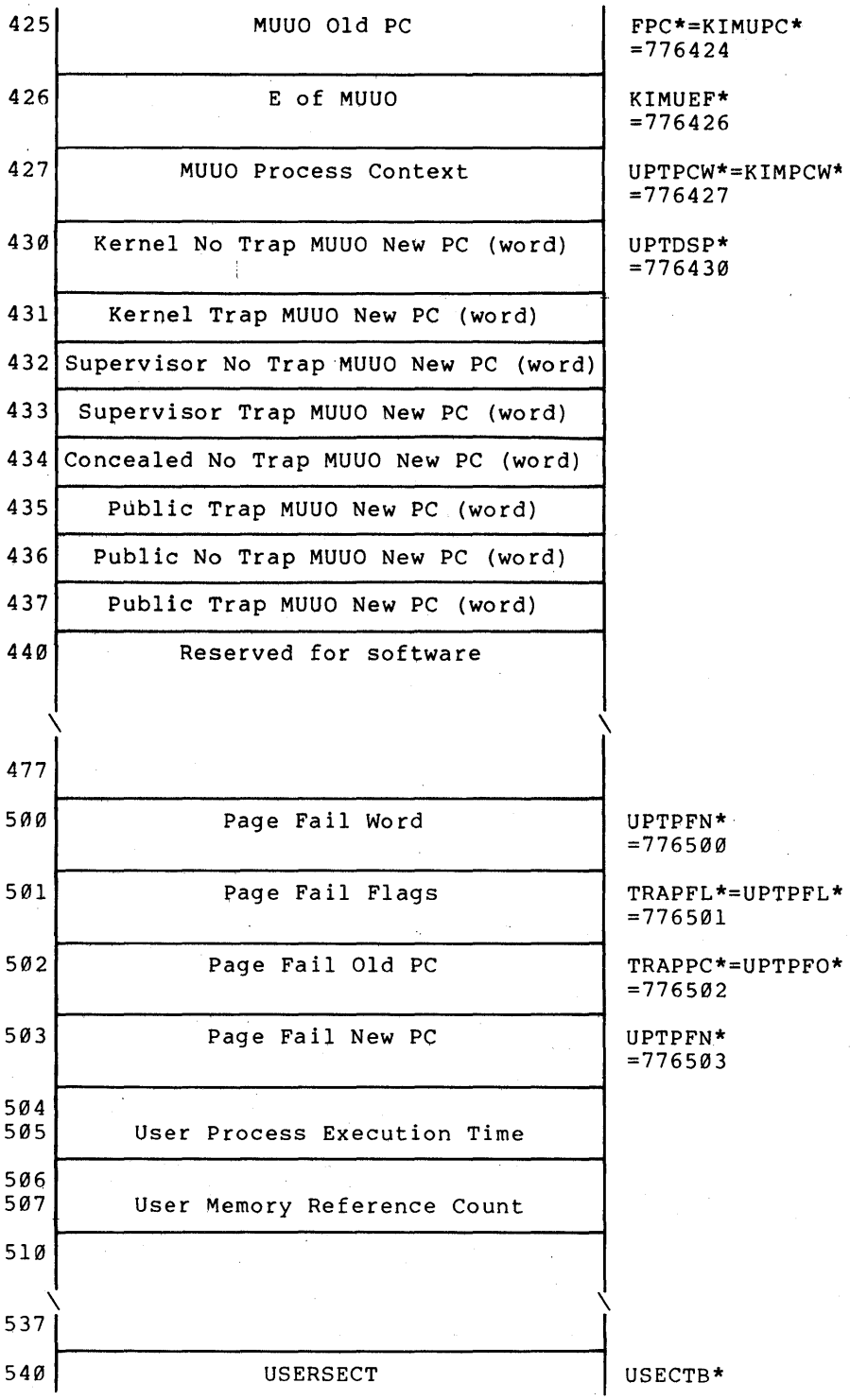
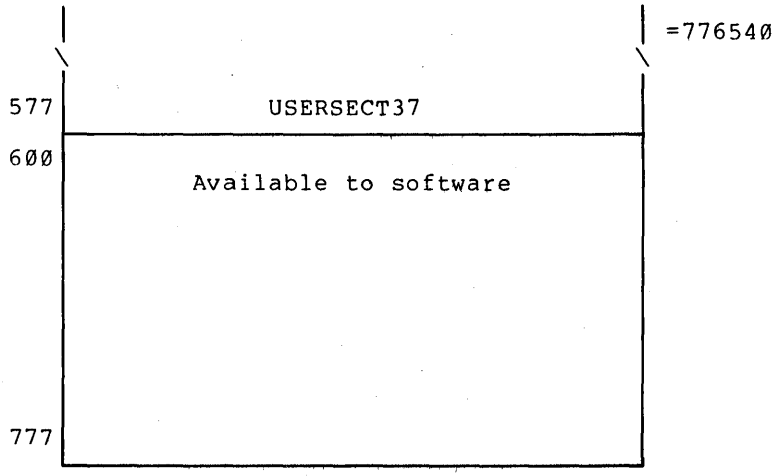


Figure SI-16. User Process Table (cont.)



Note: Approximately 1/4 of the UPT is used for hardware cells, leaving the rest available to software. The monitor currently uses this area to house the first page of the PSB table. (See PSB table description.)

* These are monitor virtual memory addresses and are used when the monitor wishes to reference the current fork's User Process Table.

Figure SI-17. User Process Table (cont.)

Name: EPT

Description: Executive Process Table. This memory resident table pointed to by the Executive Base Register (EBR), contains the vectored dispatch addresses for system events. All device interrupts pass control to a specific offset position in this table.

This table also includes the executive section map table, the time of day clock and arithmetic trap instructions which are executed when arithmetic conditions occur in executive mode.

Locations 444 to 457 are reserved for software and used by DTESRV.

Defined In: STG

Referenced by: APRSRV, DTESRV, MEXEC, PHYH11, PHYH2

FORMAT

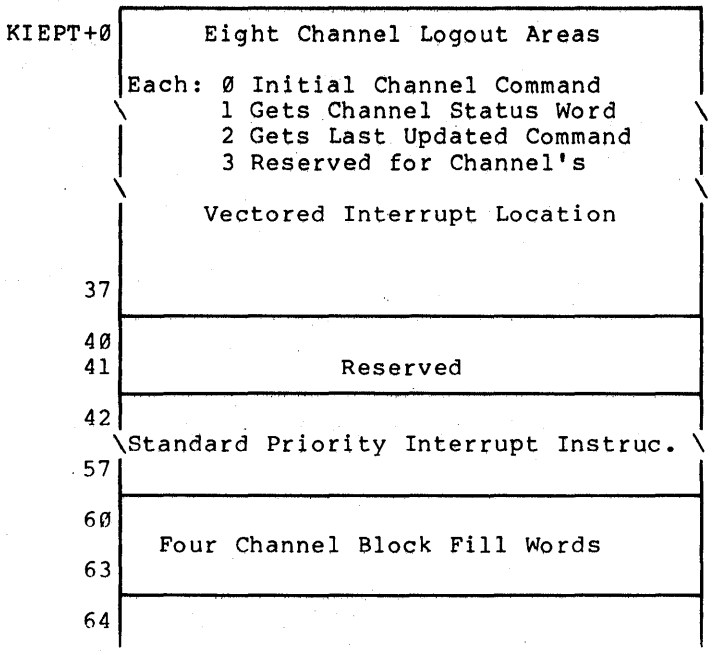


Figure SI-18. Executive Process Table

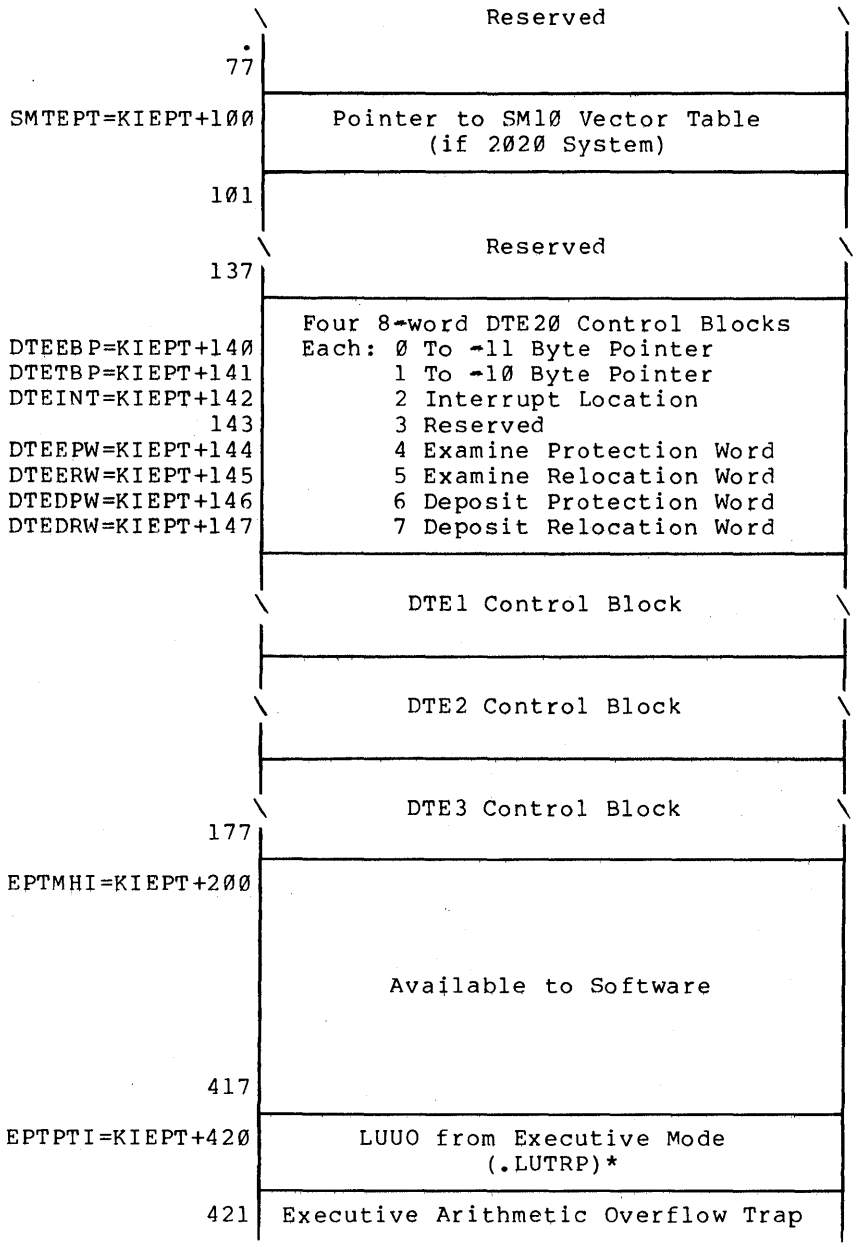


Figure SI-19. Executive Process Table (cont.)

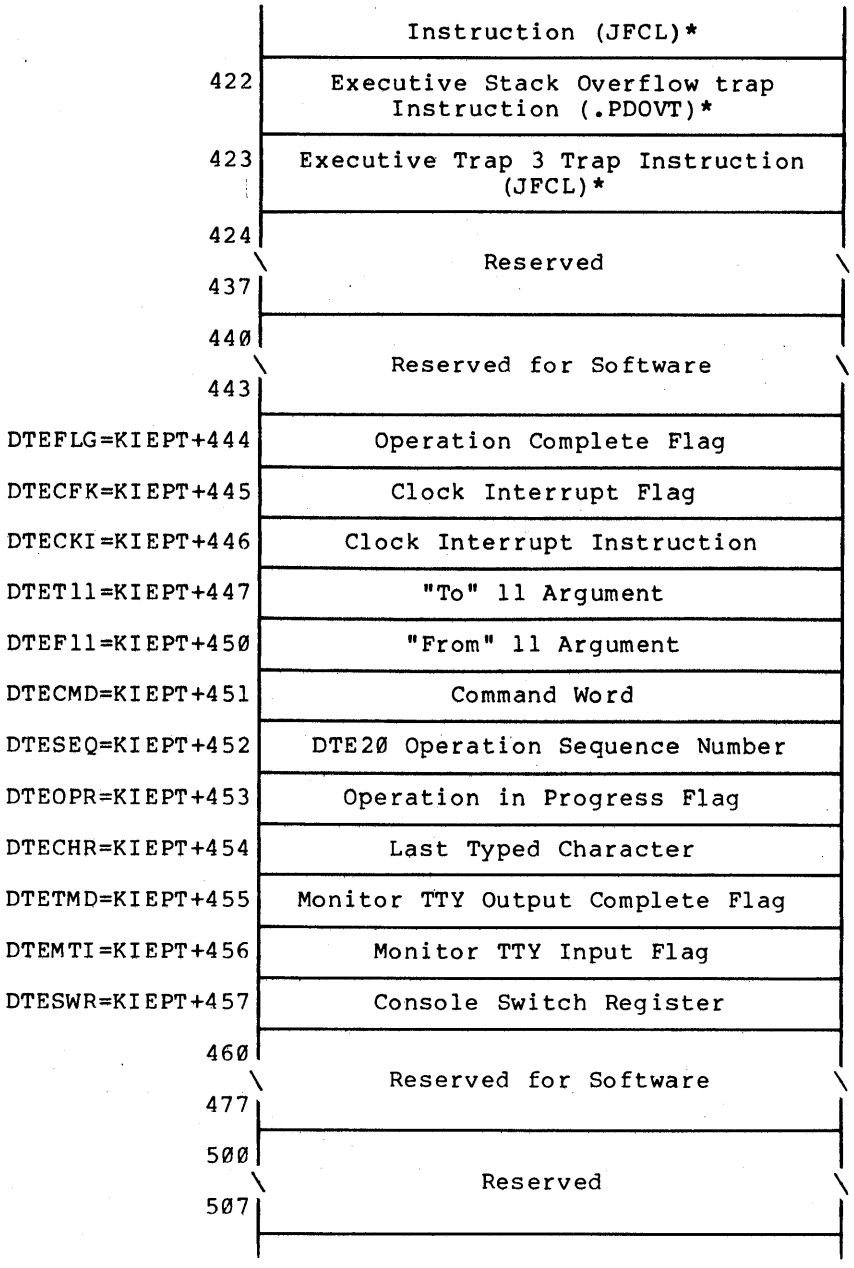
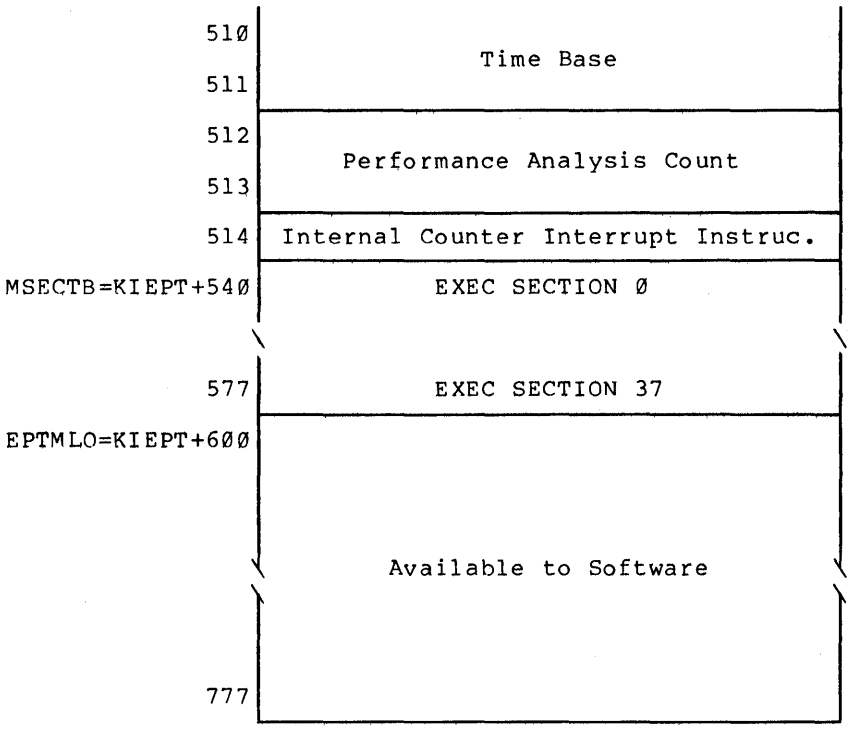


Figure SI-20. Executive Process Table (cont.)



* These values are placed into the table when the EPT is initialized at system startup.

Figure SI-21. Executive Process Table (cont.)

Special/Shared Page Table (SPT)

The monitor keeps track of the UPT and User Page Map pages in the SPT for each process on the system. The SPT is a resident table (that is, it is never swapped out) which is 3000 to 5000 (octal) words long. The SPT keeps track of a page's location since the page may be in core or swapped out. The SPT is also used to keep track of file pages and system overhead pages.

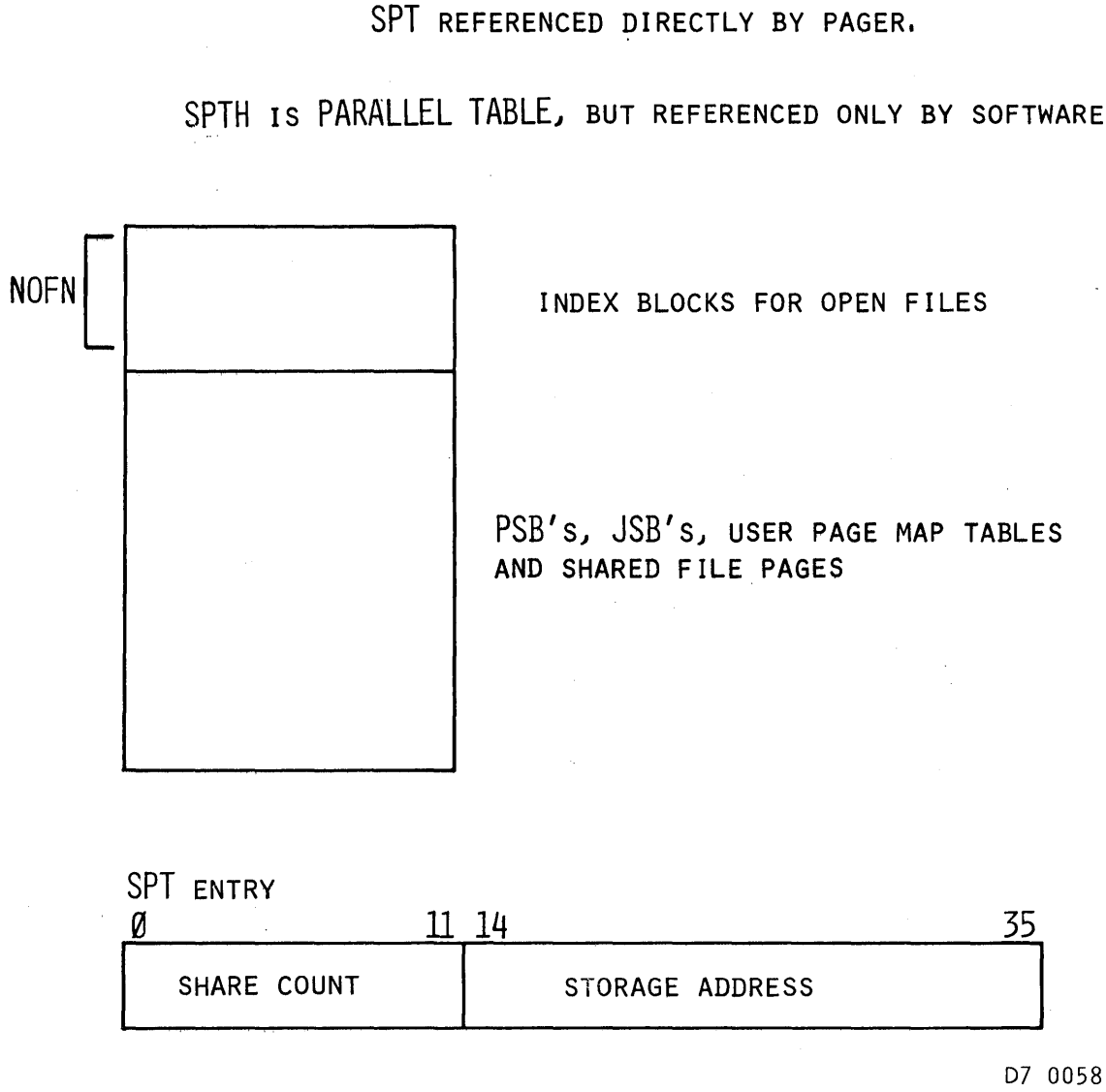


Figure SI-22. Shared Pages Table

Summary of Paging

The software sets up the page table locations and contents, the microcode does the address translation. How does the software tell the microcode where to look?

1. AC block 6 is set up by the software with the base addresses of:

    1. SPT Table

    2. CST Table (Core status table)

    3. Age stamp information (for removing old pages when necessary)

> **NOTE**
>
> The other used AC blocks are: 0 - monitor use, 1 - user, 7 - microcode use.

2. EBR is set up by the software

3. UBR is set up by the software

Software Introduction

LAB EXERCISES

When answering the lab exercises, write down the names of the tables where you found the answers. The labs will help you understand the monitor data base structure; so remember, where to look is more important than what you find there.

The exercises that are marked with a double star (**) are more difficult and are optional. If you have the time and motivation, do them.

TOOLS

FILDDT

FILDDT is a program that can be used to look at a crash or at the running monitor. Use FILDDT as in the example below to do this lab's exercises.

```
@ENABLE                              ;need enabled wheel
                                     ;capability to look at the
                                     ;running monitor.
$FILDDT                              ;start the program
FILDDT>LOAD <SYSTEM>MONITR.EXE       ;load the symbols
FILDDT>PEEK                          ;peek at the running monitor
```

At this point, the usual DDT commands allow you to look at the running monitor. You cannot change any locations (i.e., you have no write privileges). Also, your process will always be running when you look because the mechanism to look at the running monitor is like a JSYS whose function is to let you use DDT from monitor context.

Virtual Address Translation

The translation from virtual to physical addresses is done by the microcode. However, the page maps and tables the hardware uses are all set up by the monitor. Therefore, the monitor's page map, the current process's page map and the SPT table are all a part of the monitor's address space (so the monitor can add and delete pages for itself or for a process). Sections 0 and 1 of the monitor are both mapped through the same page table which begins at offset MMAP. The current process's page table is always mapped into the monitor's address space beginning at location UPTA.

RESOURCES

1. Read section 3.4 (TOPS-20 paging) of the Hardware Reference Manual.
2. Use the UPTA table in your monitor tables.
3. Refer to the Storage Addresses handout in your Student Guide.
4. Refer to the Page Pointers handout in your Student Guide.

EXERCISES

1. Using the page map at UPTA (which is your process's page map) find a share pointer, an immediate pointer, and in indirect pointer (if possible).
2. What is the storage address for each of the pointers?
3. What level of storage does the storage address indicate that page is on?
4. Look at MMAP; why do you think that all the pointers in the first portion of MMAP are private? **

Software Introduction

LAB SOLUTIONS

EXERCISES

1. Using the page map at UPTA (which is your process's page map) find a share pointer, an immediate pointer, and in indirect pointer (if possible).

   ANSWER: Bits 0-2 of the pointer contain the pointer type; 2 is a share pointer, 3 is an indirect pointer, and 1 is an immediate pointer.

2. What is the storage address for each of the pointers?

   ANSWER: A share pointer's storage address is in the indicated SPT slot; an immediate pointer's storage address is in bits 12-35 of the pointer itself; and the storage address of the indirect pointer is determined by the object page map.

```
UPTA/ 124000,,517       ;517 is the storage
                        ;address
UPTA+1/ 124000,,622     ;622 is the storage
                        ;address
UPTA+2/ 206000,,2167    ;SPT slot 2167 contains
                        ;the storage address
```

3. What level of storage does the storage address indicate each page is on?

   ANSWER: If bits 12-17 are 0, the storage address is a core address; if bits 12-14 are 0 but bit 16=1, the remainder of the word is a drum storage address; if bit 14 is on, the remainder of the word is a disk storage address.

```
UPTA/ 124000,,622       ;core address

UPTA+2/ 206000,,2167    ;SPT slot 2167 contains
                        ;the storage address
SPT+2167/ 110,,5174     ;disk address

UPTA+10/ 124003,,7034   ;drum address
```

4. Look at MMAP; why do you think that all the pointers in the first portion of MMAP are private? **

   ANSWER: This is the resident portion of the monitor, most of it read in by BOOT. You will also notice that the pages are always in core and that their core addresses correspond with their virtual addresses.

```
MMAP/ 124000,,0
MMAP+1/ 124000,,1
MMAP+2/ 124000,,2
MMAP+3/ 124000,,3
        .
        .
        .
        .
MMAP+32/ 124000,,32
        .
        .
        .
MMAP+55/ 124000,,55
```

MODULE TEST

1. What features of operating systems can be compared to determine whether a given system is large or small?

2. What are the basic aspects of batch, timesharing, and real time operating systems?

3. What advantages does virtual memory's address translation give an operating system?

4. What is a page fault and how is it handled?

5. How are files organized on TOPS-20?

6. What does the disk manager do?

This page is for notes.

TEST EVALUATION SHEET

1. Such things as scheduling approach, devices handled, protection, file organization, accounting, and network capabilities determine whether a system is large or small.

2. Batch works on long jobs, has sequential job submission, few concurrently running jobs, no interaction, and is often card-oriented. Timesharing includes many interactive users, round-robin scheduling, short jobs and is terminal-oriented. Real-time is interaction with fixed response time constraints.

3. Virtual memory address translation eliminates unusable pages by making all real memory usable by any process. It makes partial residency easy to maintain dynamically. It provides protection for the operating system and users and, finally, it reduces the work of the loader program.

4. A page fault occurs when reference is made to a process page that is not in real memory. When a page fault occurs, the operating system requests the page be loaded into real memory from the disk. The process becomes blocked until it arrives.

5. Files are accessed through directories. Each user has a directory which has a pointer to an index block for each file. The index block contains a pointer to each page of the file on disk. The pages of the file may be scattered anywhere on the disk.

6. The disk manager maintains a queue of disk requests. The order of requests is such that overall seek time and latency time are minimized.

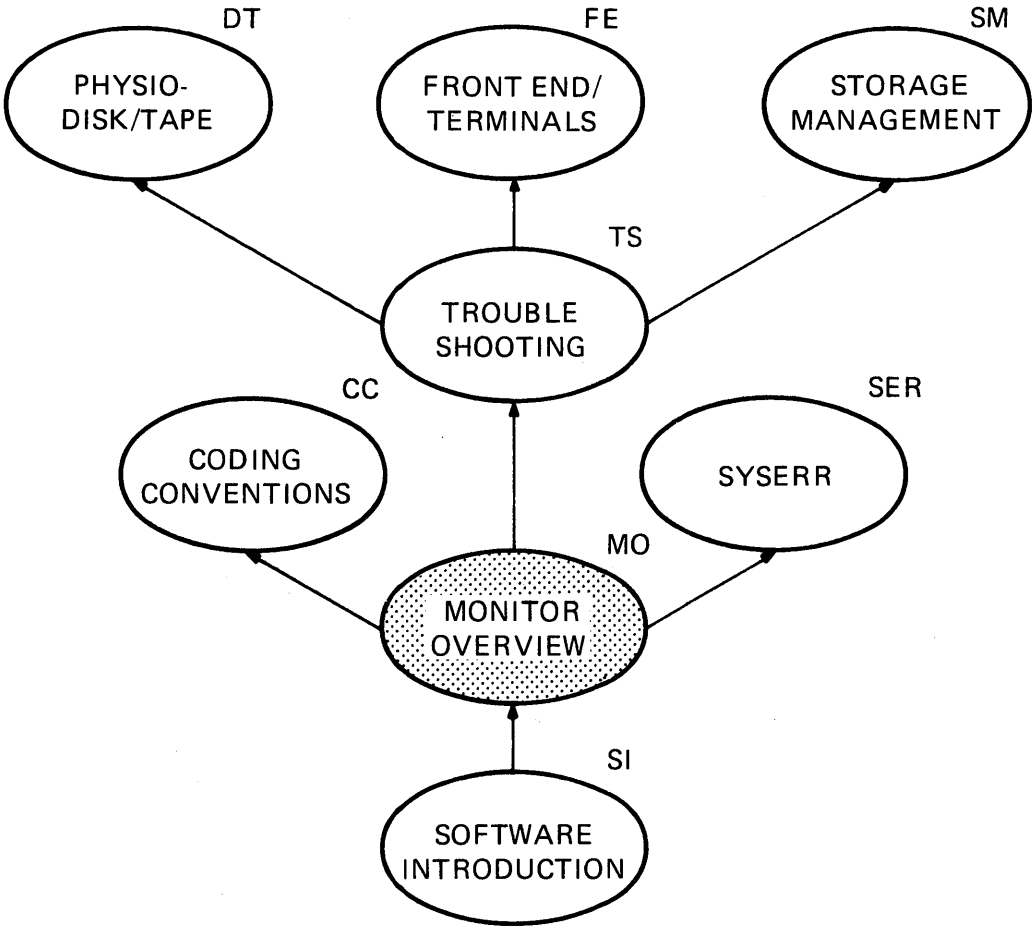
This page is for notes.

TOPS-20 MONITOR

Monitor Overview

<<For Internal Use Only>>

COURSE MAP



MR-2717

This page is for notes.

Monitor Overview

INTRODUCTION

The DECSYSTEM-20 consists of hardware and software designed to allow users to run a variety of programs efficiently and conveniently. It is specifically designed as a paged timesharing system. Normally, several active programs are run concurrently, with control switched from one to another by the monitor. Programs not using the CPU can still have active input and output devices. This overlapping of I/O with the processing of several programs permits efficient use of both the CPU and the I/O devices.

The DECSYSTEM-20 has several hardware features that facilitate multiprogram operation. The two basic modes of operation are: executive and user. The monitor runs in executive mode with no restrictions on its operations. In user mode, a program can access core memory only within areas assigned to it by the monitor. Also, certain instructions are not permitted in user mode. These include all I/O instructions and the instructions to control memory access and mode of operation.

LEARNING OBJECTIVES

Upon completion of this module, the student will be able to:

1. Name the major functional sections of the monitor.
2. Describe the function of each of those sections.
3. Identify the major data and monitor structures.
4. Describe the use and function of TOPS-2Ø Monitor Calls (JSYSs).

RESOURCES

Appendices A and B of this course.

MODULE OUTLINE

Monitor Overview

I. Monitor Calls

II. Storage Management
  A. Block Diagram

III. Pager
  A. Hierarchical Storage Considerations
  B. Implementation - Mapping
  C. Inter-Level Data Flow
  D. Updating Lower Levels

IV. Scheduler

V. File System
  A. Data Structure

VI. Job/Fork Structure

VII. Disk And Magtape Service
  A. Hardware Principles
  B. Monitor Modules
  C. I/O Requests

VIII. Front End Service
  A. TTY Input
  B. TTY Line Buffers And Echoing
  C. Line Printer Output

IX. Appendices

This page is for notes.

MONITOR CALLS

The monitor performs a number of services for user programs, including I/O operations. The instruction code, 104, provides the means for programs to request the monitor to perform these services. Each monitor call, referred to as a JSYS (Jump to SYStem), has a function code associated with it which is stored along with the operation code, 104, when the monitor call is assembled. The 104 operation code has no hardware function except to give control to the monitor. When a JSYS is executed, a routine in the monitor decodes the request and calls a subroutine to perform the requested operation. After the JSYS request has been processed, control is returned to the calling program along with indications of error conditions, if any.

Requests for service come from user programs in the form of JSYSs. A terminal request for system resources simply takes the form of input data to the EXEC program which translates the request into appropriate JSYSs. When a request is made to start up a program, an inferior fork of the EXEC is created, and the locations of the program's pages on disk are placed into the fork's page map table. No initial core is assigned to the fork's pages. Rather, pages in core are assigned on demand (i.e., when a page fault occurs) when the process references them. The most frequent requests for service from programs are the I/O JSYSs. These JSYSs allow a process to access data by file name on a byte, string, or page basis without being concerned about the physical location of the data. The monitor computes physical addresses on disk, starts I/O transfers, and handles the resulting I/O interrupts.

Control functions are performed as necessary by the monitor, according to algorithms which attempt to give optimal overall system performance. One of the most important of these functions is dividing the available CPU time among the active processes. A running process must be stopped when a clock tick occurs, and its computational state must be preserved so that it may be started at a later time. The monitor must decide which pages of user programs to keep in physical core and which to swap out to the

swapping device. In addition, it must decide where to put user pages in physical core when they are swapped back in, replacing other user pages if necessary. The replaced pages will be preserved first if needed (i.e., modified pages swapped out).

STORAGE MANAGEMENT

Block Diagram

The functions of scheduling and storage management are handled by a number of interrelated modules of the DECSYSTEM-20 monitor, each with a specific set of operations to perform. The following diagram gives the major modules of the scheduler and storage management and their communication paths with each other.

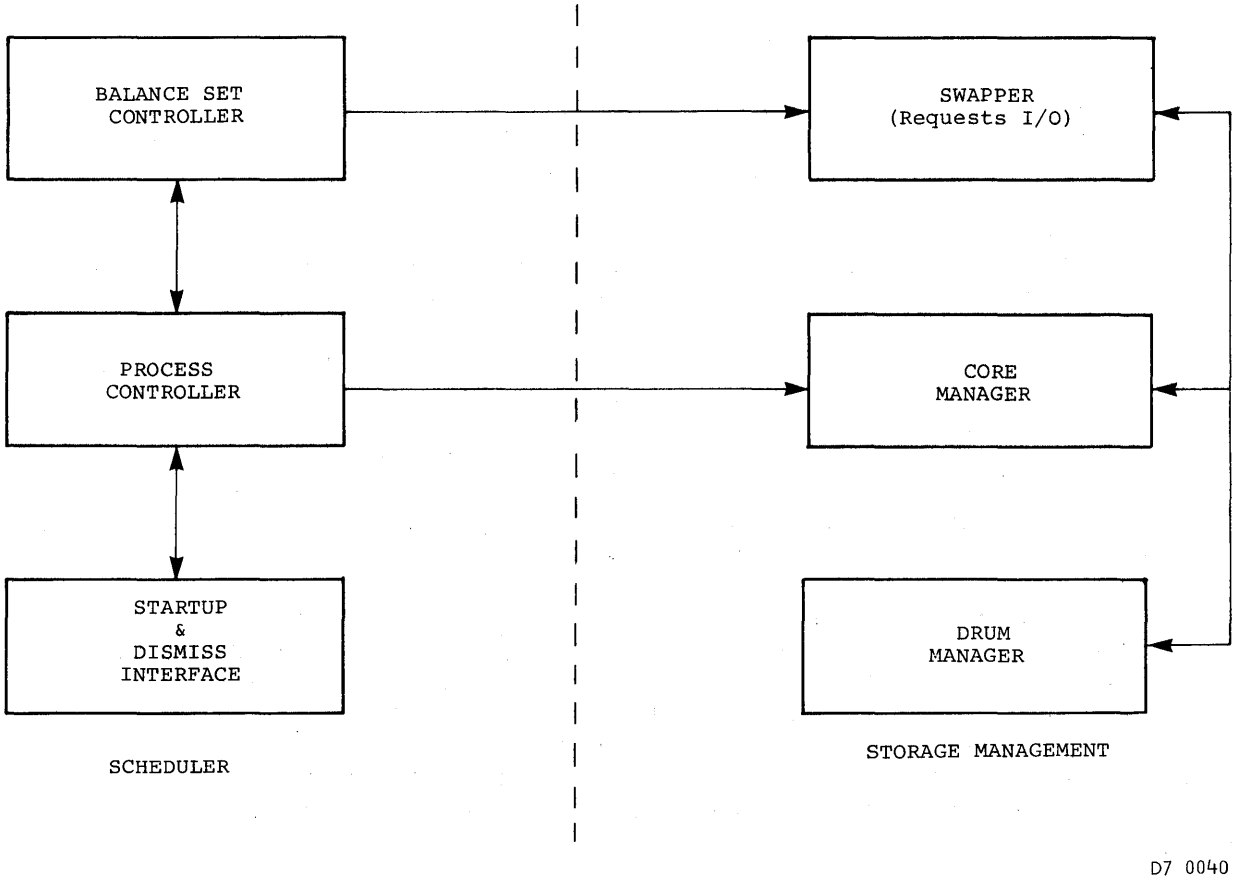


Figure MO-1. Block Diagram

The Swapper handles communication between the secondary storage devices (drum or disk) and core memory. Upon receiving a request from the Scheduler or Core Manager to move pages into and out of core, the swapper constructs an I/O request and calls the device-dependent module to start the I/O.

The Drum Manager is responsible for both assigning storage on the swapping drum and selecting pages to be moved to the disk in the event the drum becomes full.

The Core Manager selects core pages to be used for swap reads from the drum or disk, performs some "aging" operations, and handles the selection of core pages to be swapped to the drum. It has principal use and control of the Core Status Table (CST) which reflects at all times the current state of each page of core memory. The CST is also modified by the paging hardware, recording information about the activity of the running process.

The core manager is invoked when a page fault occurs. If the working set of the faulted process can be increased by one, the core manager will assign a page from the replaceable queue (linked list of free pages) and call the swapper to swap in the faulted page. If the working set size cannot be increased, garbage collecting for the process takes place. That is, the fork's working set will be decreased by swapping out the pages least recently referenced. Whether the fork's working set size becomes too large or not, the fork's working set is periodically examined for "old" pages.

If a process page faults and can legally be granted another page and none are available on the replacement queue, the fork is put into a wait state and the scheduler is called. The scheduler will detect the shortage of pages and call the core manager to global garbage collect on forks no longer in the balance set. The core manager will then invoke the swapper to swap out the collected pages.

PAGER

The pager is placed logically between the processor and the core memories and translates each memory address received from the processor into a physical core address which is sent to the memories. Control signals allow the pager to know what type of access the processor is making (read, write, or execute), and allow the pager to signal the processor when, for some reason, a reference cannot be completed (e.g., when the page is not in core). The virtual addresses received from the processor are 18 bits, and the page size is 512 words, so the pager is, in fact, translating the high-order 9 bits of address, and passing the low-order 9 bits through unchanged.

The pager uses a 512-word hardware page table (indexed by virtual page number) to hold physical page information of recently referenced virtual pages, but the source of this information is always a "page table" in core memory. Page tables contain (or point to) the physical storage address, if any, of each page of a virtual memory. Thus, each process' virtual memory is represented by one page table. Page table entries are of one word; hence, a page table for a 256K virtual memory is 512 words, or exactly one page long.

The pager references the page table of the relevant process, using the 9 high-order virtual address bits as an index, whenever the hardware page table fails to contain the physical information for the requested virtual address. The pager is capable of interpreting three types of page table entries. The first is called a "private" pointer and contains a physical storage address. If this is a core address, the pager will load the hardware page table with the information and complete the requested reference. If it is any other address, the pager will initiate a trap to the monitor for appropriate action. The second type of page table entry is called a "shared" pointer which contains an index into a system table at a fixed location. This "Shared/Special Pages Table" (SPT) contains the physical storage address, and the details of its functions are described below.

The third type of page table entry is the "indirect" pointer which contains a page number and SPT index. The SPT index is used to index into the SPT table to pick up an

address of a page table. This page table is indexed by the page number given in the indirect pointer to obtain the physical storage address. This pointer allows one page to be exactly equivalent to another page in a separate space.

One other fixed table, called the Core Status Table Ø (CSTØ), is used by the pager. For each page of physical core, this table contains information about recent references and notes if the page has been modified.

Hierarchical Storage Considerations

In any system using hierarchical (multi-level) storage, one is concerned with the movement of data between the various levels, the location of current "up-to-date" copy, the updating of lower levels, etc. It is usually considered essential that the address of the currently valid copy of an item of storage resides in only one place. This tends to conflict with the goal of sharing, which requires that items of storage be made available to multiple processes simultaneously. Replication of addresses would appear to admit the possibility of unresolvable phase errors, and the updating problem itself would unnecessarily complicate the software.

DECSYSTEM-2Ø's solution to the basic storage management problem is the shared pages table scheme. In this scheme, storage addresses (for shared elements) again reside in only one place, a fixed table called the shared/special pages table. Processes using an element of storage are given a fixed index 'Y' which identifies the SPT entry holding the current address, but an entry cannot be deleted from the SPT as long as pointers to it exist. Therefore, a share count is required for each entry to record the number of pointers to it that have been created; this count is kept in the SPT.

Implementation - Mapping

The following shows how the DECSYSTEM-20 implements the file mapping operations discussed in the previous section, and how data flows between the several levels of storage. The DECSYSTEM-20 storage hierarchy consists of three levels: core, swapping, and file.

As described previously, named memory consists of pages within files. A sample file with two of its pages is shown in Figure MO-2. The basic structure of the file is an index block containing the storage addresses of all of the data pages. In fact, this index block is a page table, initially containing private pointers. Assume a starting point where none of the file pages are mapped in any process, so the only place for the storage address of each of these file pages is logically and properly the index block of the file that owns them.

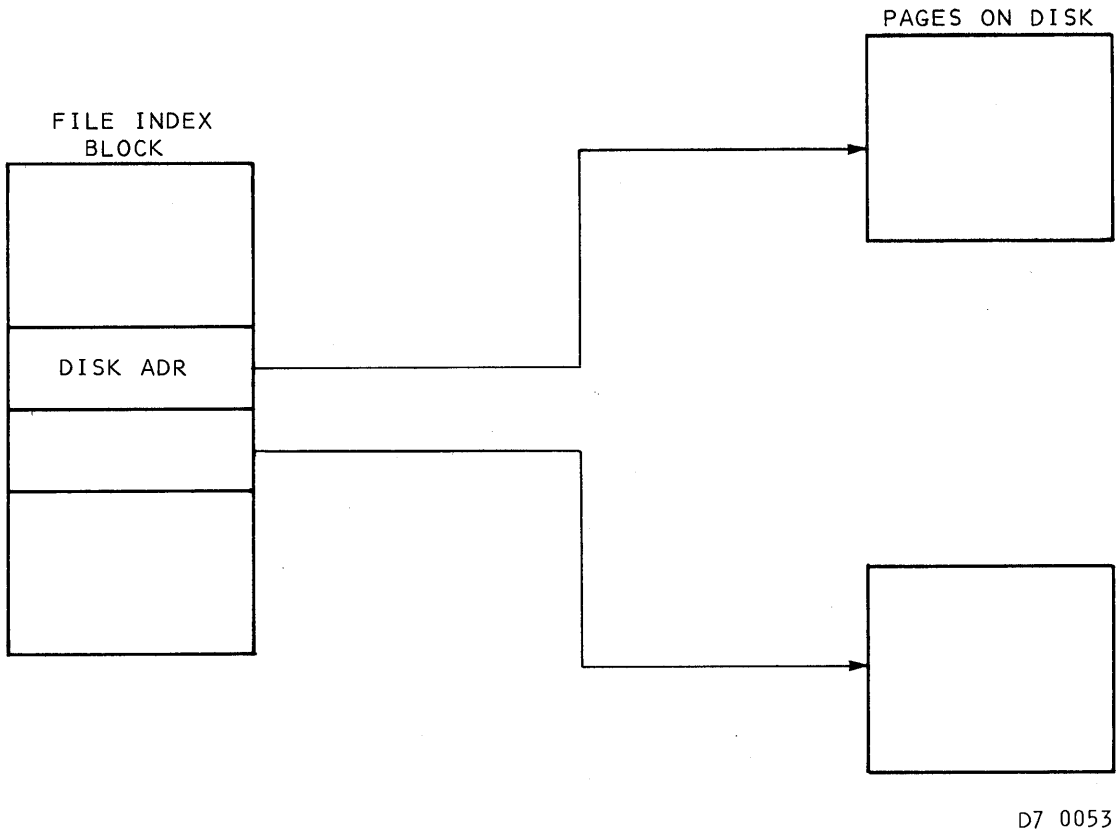


Figure MO-2. File Structure

Next, a process requests that one of these file pages be mapped into its address space. The monitor uses the JFN portion of the identifier to locate the file index block, and the PN portion to select the appropriate entry within it. Although our aim here is to have just one process using the page, we see that, in fact, the page must become shared at this point (that is, shared between the file and the process). Therefore, the monitor will assign a slot in the SPT and place in it the disk address obtained from the file's in-core copy of the index block. Simultaneously, it creates a shared pointer which points to that SPT slot and places a copy in both the file's in-core copy of the index block and the process page table. The share count for the SPT slot is set to reflect the fact that the page is in use twice: once by the file, and once by the process.

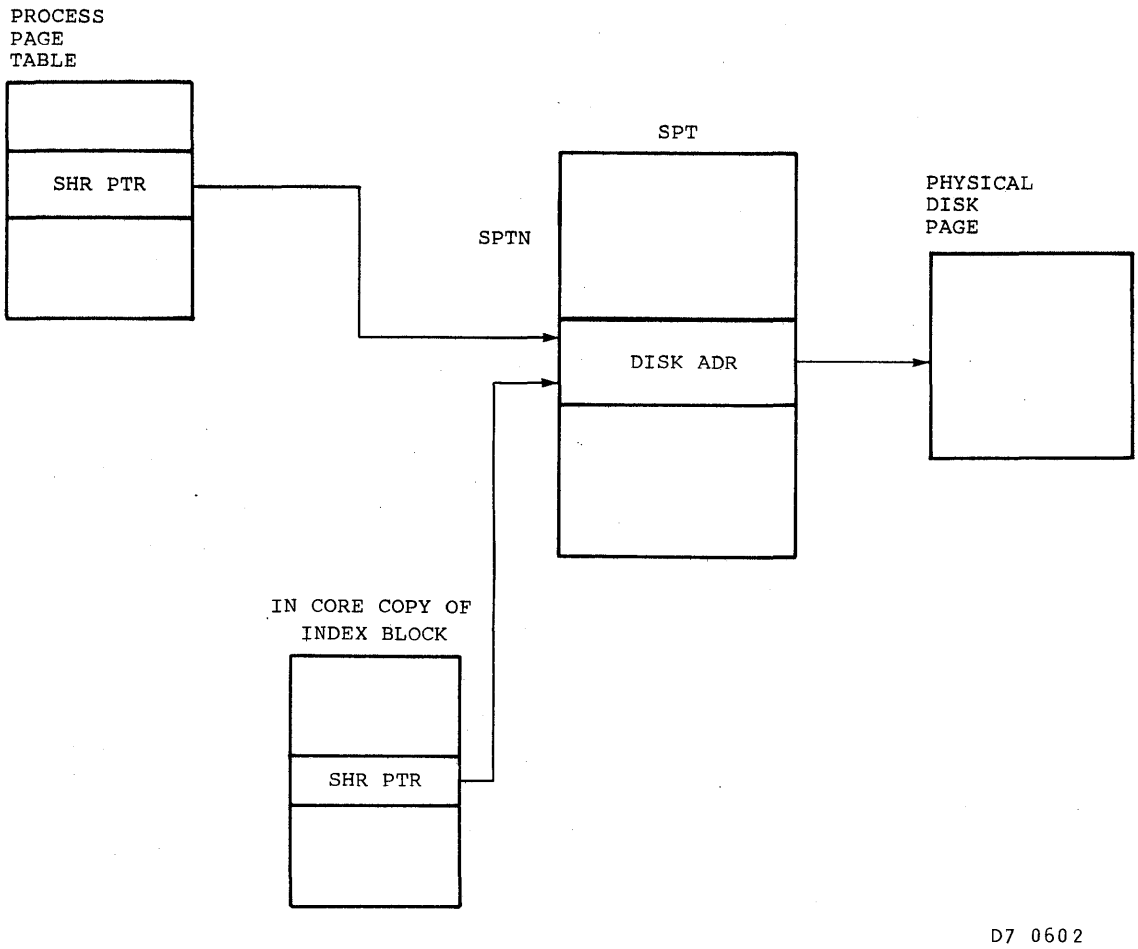


Figure MO-3A. One Process Maps a File Page

A second process wishing to use the same page proceeds in the same manner, but now it is only necessary to create another copy of the shared pointer and increment the share count. This situation is shown in Figure MO-3B. The subsequent reduction of the share count to 1 (when all processes unmap the page) will indicate that the SPT entry may be reclaimed.

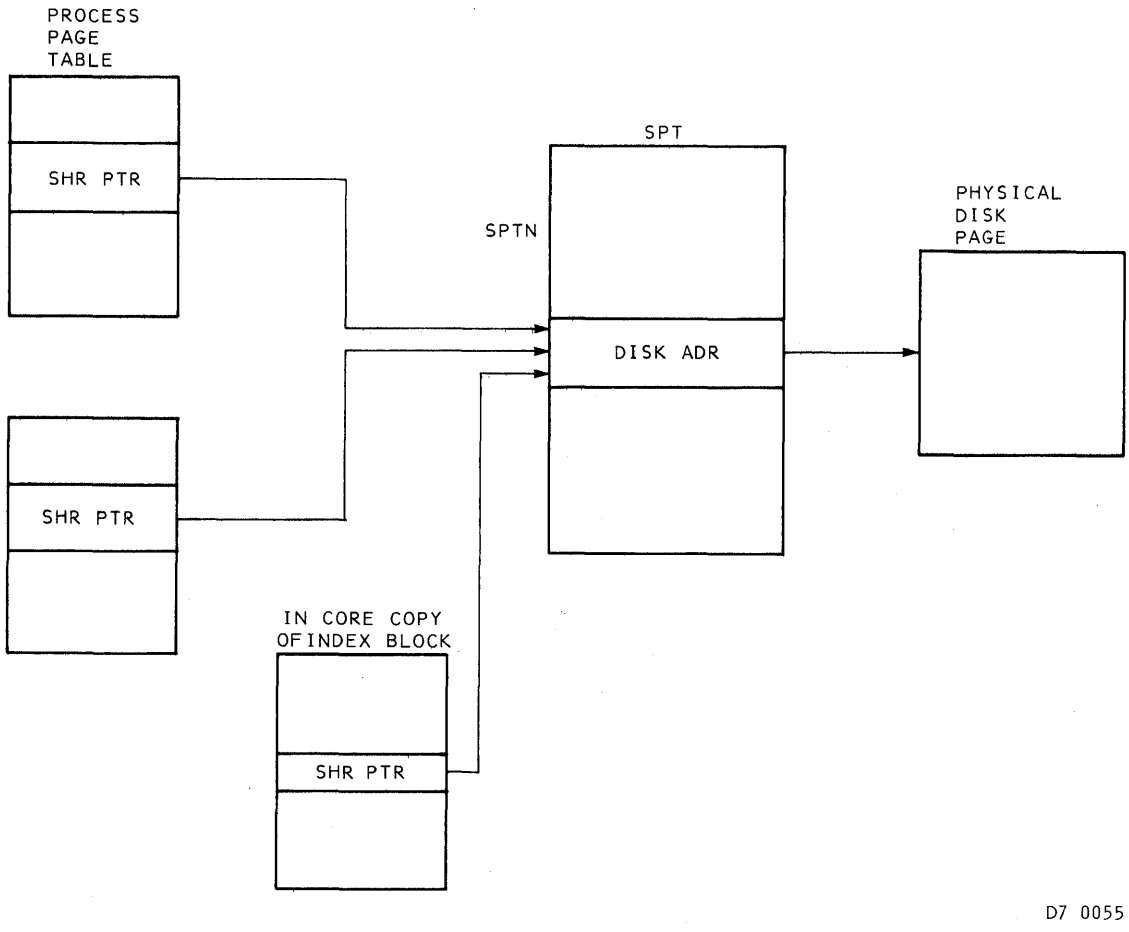


Figure MO-3B. Two Processes Map a File Page

Some additional bookkeeping is necessary in order to keep track of the owner of the page and to note the fact that the file index block is in use. This is shown in Figure MO-4. The table labeled SPTH is parallel to and the same length as the SPT. For our example file page which was assigned to slot 'SPTN', the parallel entry in the SPTH records the owning page table of the page. This is shown as OFN and PN. The OFN (Open File Number) is the monitor's internal equivalent of the user's JFN, except that it identifies open files over the domain of all jobs in the system. The OFN is actually an index into a portion of the SPT which is reserved for index blocks, and the PN is the page number supplied by the user. The OFN portion of the SPTH holds the home addresses of the index blocks currently in use.

The monitor must always open files on the basis of the storage address of the index block as obtained from the file directory, and a search of this part of the SPTH is necessary to determine if the file is already open.

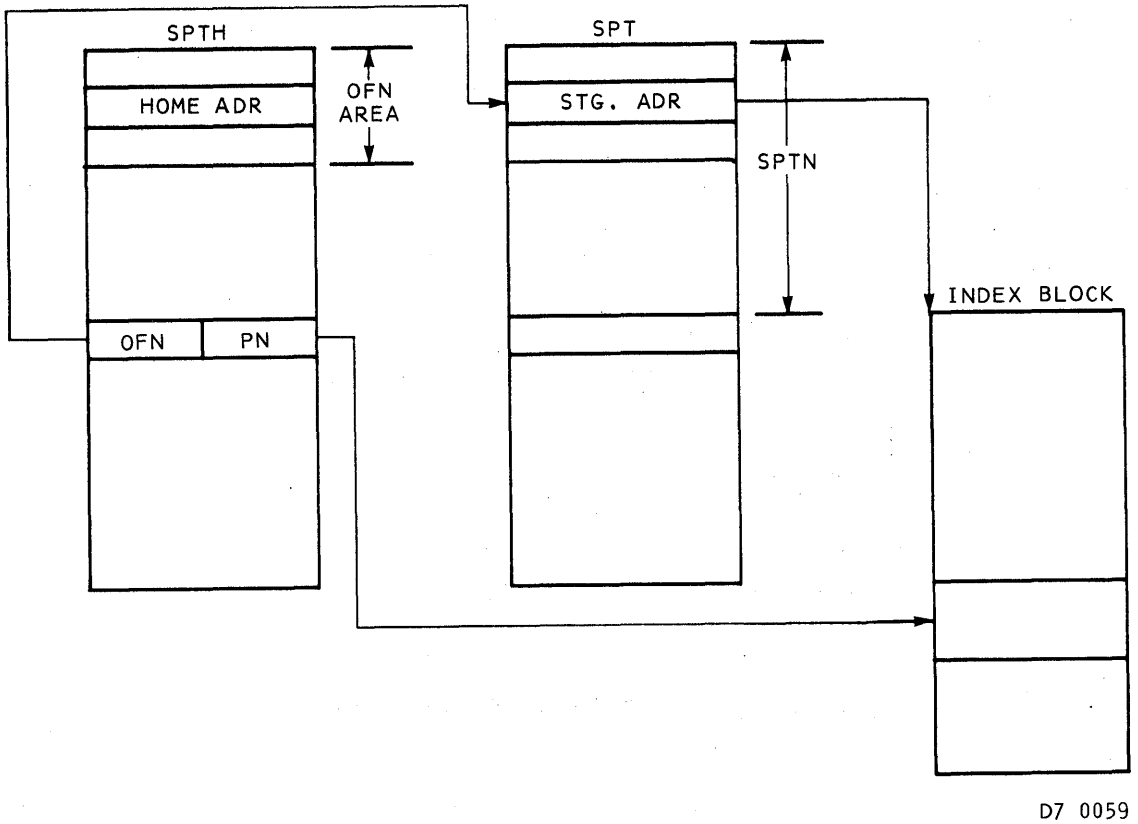


Figure MO-4. Ownership Back Pointers

Inter-Level Data Flow

Next, we show what happens when one of the processes references the file page which has been mapped. This is shown in Figure MO-5. The pager interprets the shared pointer found in the process map, and references the SPT. It finds, however, that the page is not in core, and so, traps to the monitor. The monitor in turn selects a page of real core and initiates a read of the disk to bring in the page. The SPT slot is then changed to indicate that the page is in core.

Two tables record the state of physical core. These are the Core Status Tables (CST1 and CST2). For each page of physical core, CST1 holds the physical address of the next lower level of storage for the page. In our example, this is a disk address because the page is just being read from the disk. CST2 records the name of the page table holding the pointer to that core page, which in this case is an SPT index.

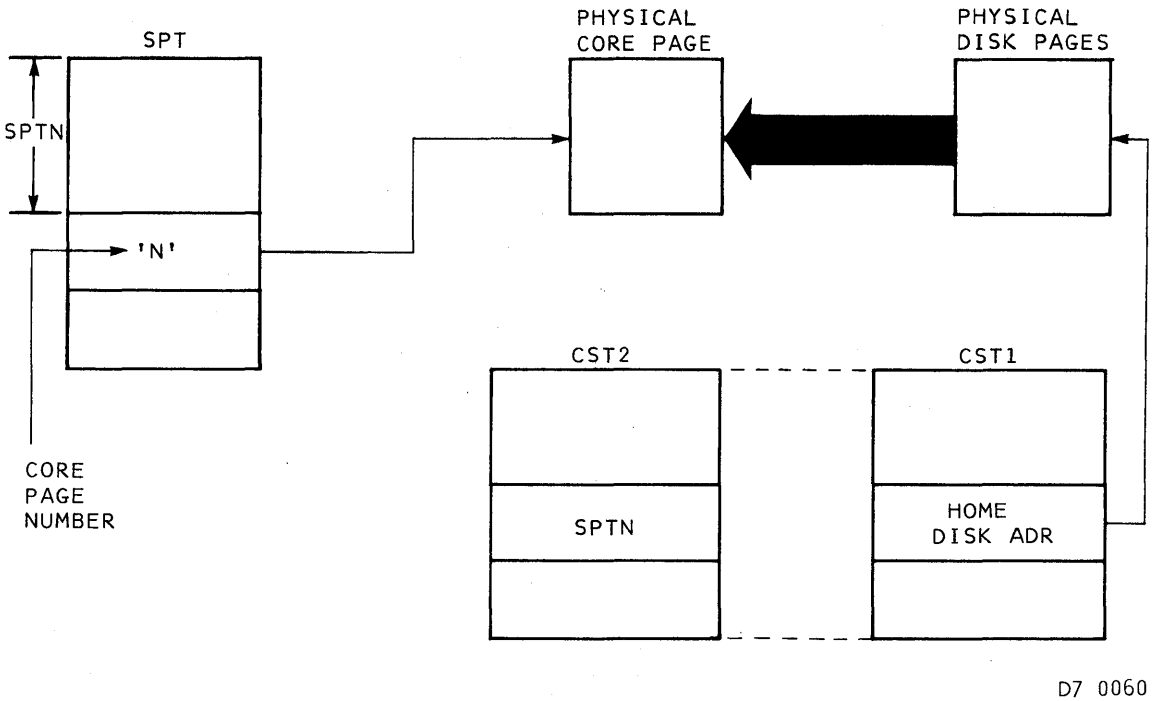


Figure MO-5. A Page is Referenced and Brought into Core

Then, we consider what is necessary for the monitor to swap the page onto the drum. It is important to note that during the course of the drum write (including latency) and for a period of time thereafter, the core page still contains a current copy of the data, and so we may properly leave the SPT slot pointing to it. This will prove useful in the event that a process makes another reference to the page during this time (since the page will not have to be read into core again). Thus, to begin the swapout, the monitor selects a free drum page, initiates the drum-write operation, and updates CST1 to reflect the fact that the next lower level of storage is now the drum.

However, we cannot discard the home address of the page, so one other table is required. The DST (Drum Status Table) serves a function for the swapping level of storage equivalent to that of the CST for core. That is, for each page in use on the drum, the DST holds the address of the next lower level of storage. It also records whether the copy on the drum has been modified with respect to the copy on the disk so that the monitor will know whether a write is necessary at some time to update the disk copy. The picture of a file page with copies on all levels of storage is now complete. (Figure MO-6).

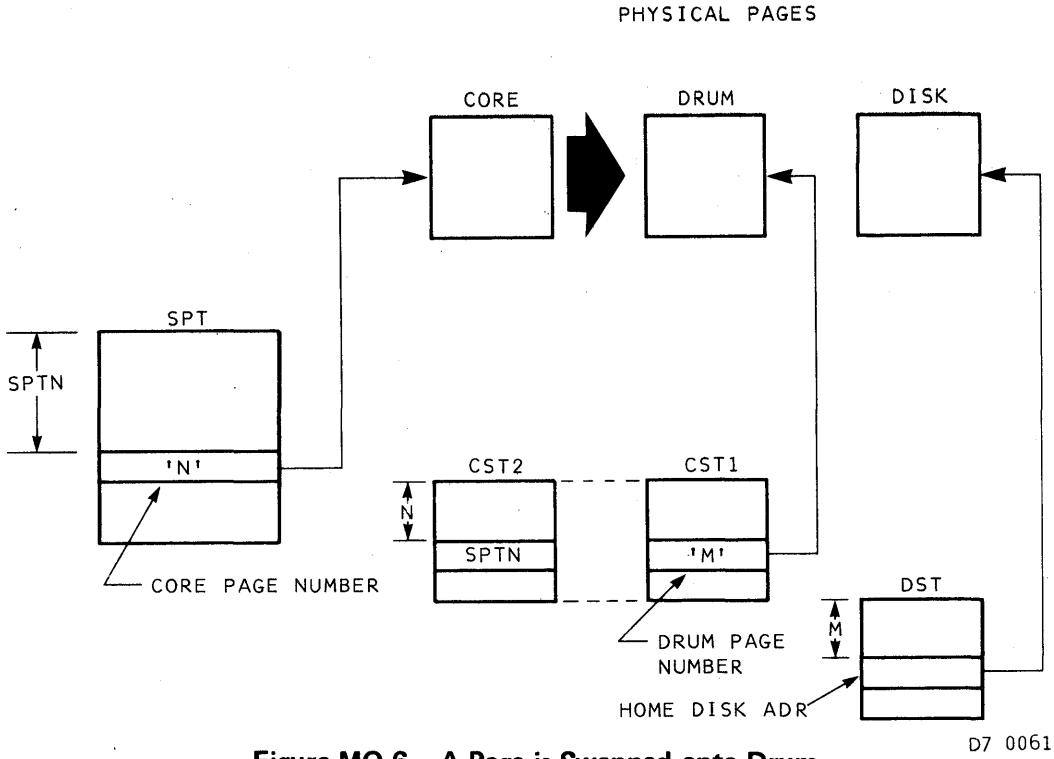


Figure MO-6. A Page is Swapped onto Drum

One final step is shown in Figure MO-7. If the page remains unreferenced for some period of time, the monitor will want to use the core page for another purpose. To do this, the monitor will move the drum address from CST1 of the page being reclaimed to the SPT slot, and succeeding attempts to reference the page will reveal that it is no longer in core.

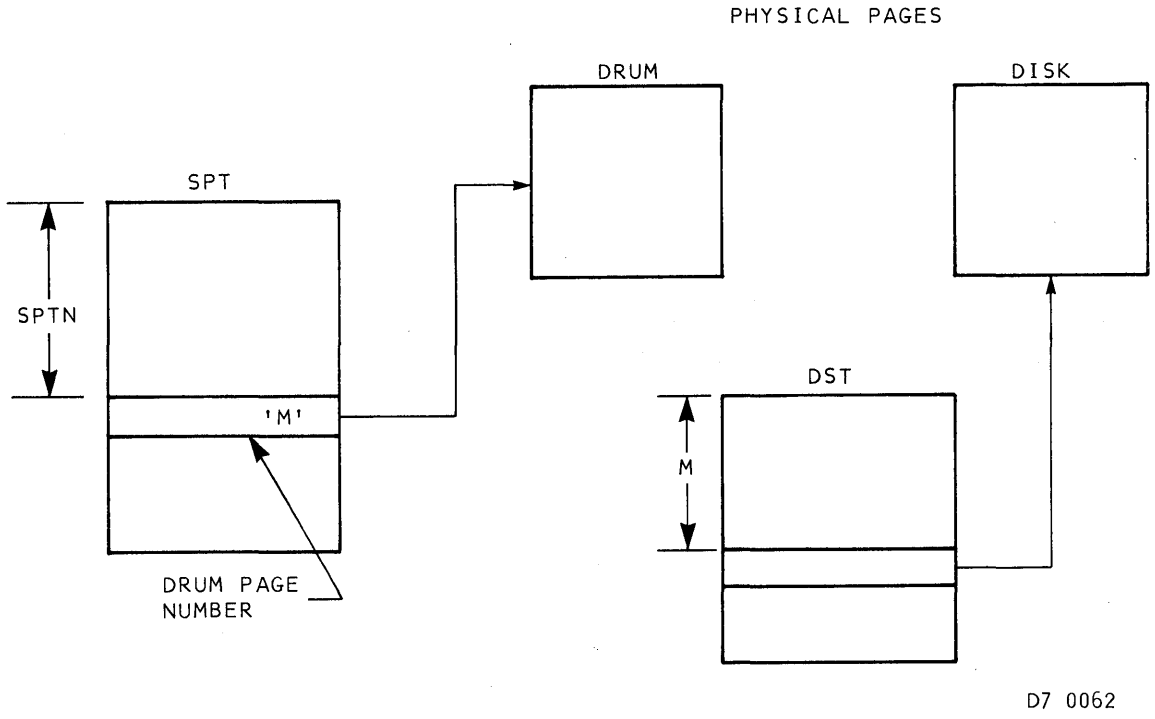


Figure MO-7. A Core Page is Released

Updating Lower Levels

As long as the page remains mapped by one or more processes, the share count will keep the SPT slot in use, and the page will be moved between the drum and core as needed. This suggests that some procedure may be necessary to periodically update the home (disk) copy of pages -- a necessity both to guard against loss due to system crash, and because some files are mapped when the system starts up and are never unmapped (e.g., the disk assignment bit table). In the DECSYSTEM-20 a special system process (DDUMP) takes this responsibility. DDUMP periodically scans the open files, finding pages that have been changed since being read from the disk. File pages are backed up to the disk by setting a request bit in the CST which causes the swapper to move the page to the disk instead of the drum. File index blocks must also be updated but require a different procedure. For these, the backup process constructs an image of the index block as it would appear with no pages shared. That is, it finds the home address of each page and puts it in the index block in the form of a private pointer. This copy is then written on the disk.

SCHEDULER

There are two main goals of the TOPS-20 Scheduler:

1. To provide rapid response to interactive users and "fair share" service to compute-bound users of the system.

2. To make efficient use of the machine's principal resources: CPU and core.

The actions of the Scheduler affect the utilization of all of the resources in the system, primarily core and CPU service. The scheduling algorithm includes procedures to affect the scheduling as a result of I/O or other non-CPU activity. The fact that TOPS-20 is a paging system greatly increases the activity on the swapping channels, which imposes an even greater demand on the scheduling procedures to interrelate the use of core with the allocation of the CPU.

In trying to allocate equal CPU time to processes on the system, the scheduler would like to know how much time a process is going to use when it makes a request for CPU service. The scheduler can only guess at the future behavior of a process based on its past behavior (but that guess may be wrong).

The most significant piece of data from the recent history of a process is the amount of time it has used since its last request for service. Two observations from monitoring process activity that can be used to predict time completion are:

1. Within any short period of time, the longer a process has run, the closer it is to completion.

2. The number of processes completed during any fixed period of time decreases as the total runtime increases; so, the longer a process has run, the less likely it is to finish.

Taking into account these observations, the runtime is broken up into separate regions (queues) so that:

- If two processes are widely separated in accumulated runtime (i.e., are in different queues), the one with the lesser time is preferred.
- If two processes are closely spaced (i.e., are in the same queue), the one with the greater time is preferred.

This scheduling philosophy has three parameters: the factor by which the time on each queue is greater than the last, the amount of time allowed in the first queue, and the number of queues.

The values chosen by TOPS-20 are graphically shown below where a process runs from its queue for a given quantum of time before being requeued to the end of the next lowest priority run queue. Processes completing their quantum in MAXQ stay in MAXQ.

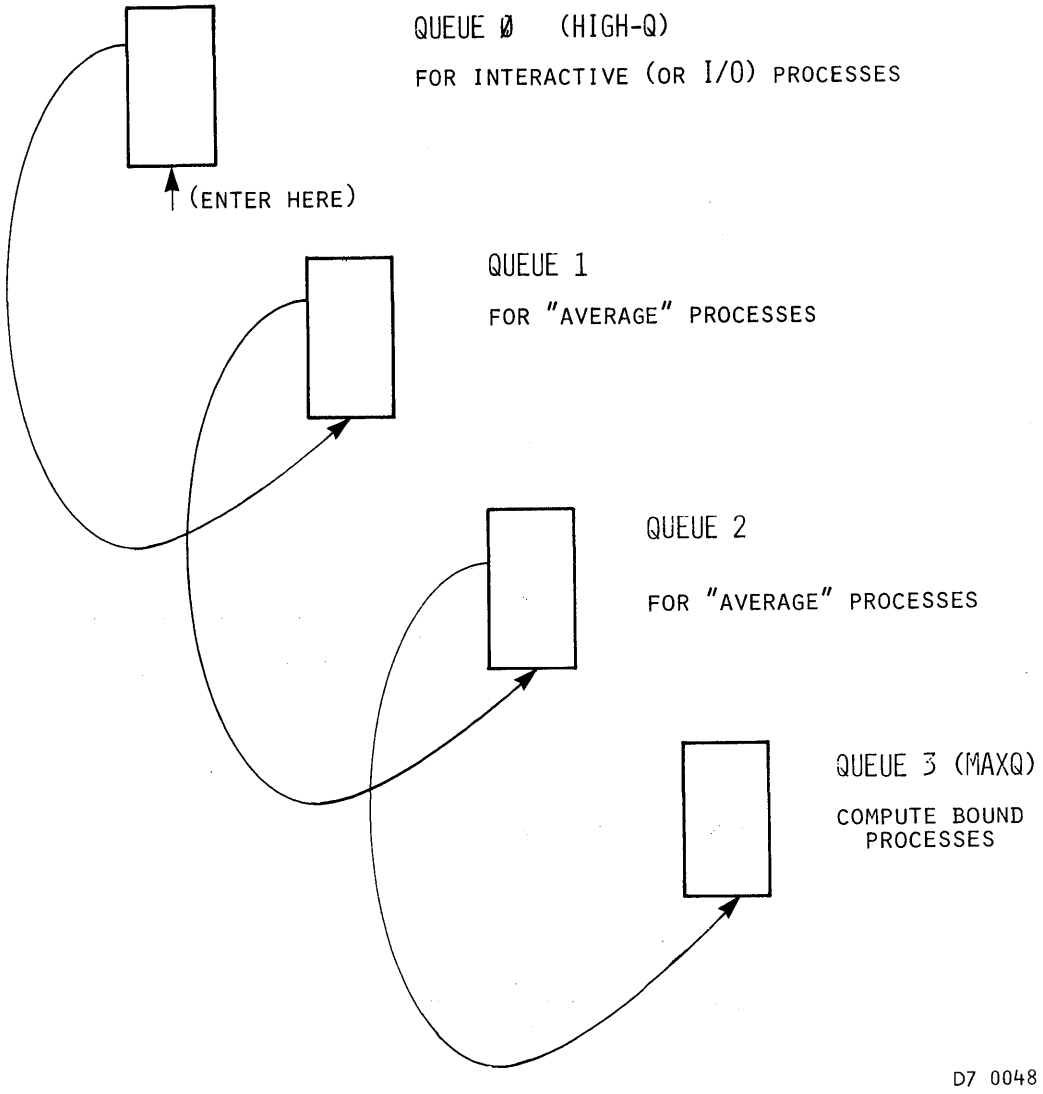


Figure MO-8. Scheduler Queue Structure

While a process is active in the system, it will be either on the go list (GOLST) or on one of the wait lists.* The contents of the fork's entry in FKPT supply the list name and the link to the next fork on the list. The table FKPT holds the chain of fork pointers linked in a forward direction for each list. The chain pointed to by the contents of GOLST is called the GOLST.

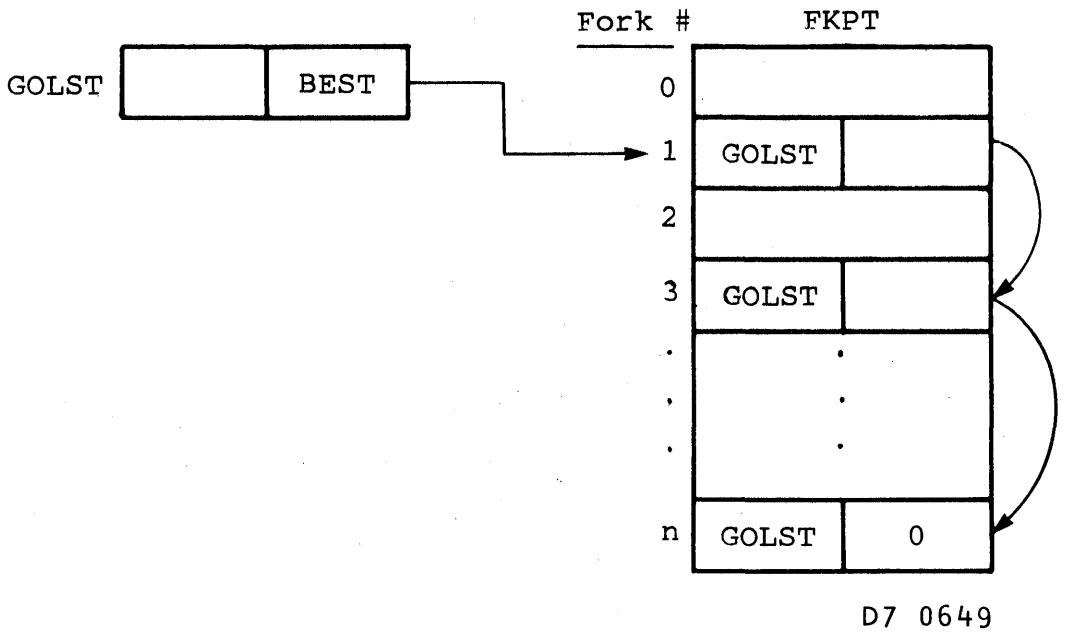


Figure MO-9. GOLST Structure

*The wait lists are: TTLIST, FRZLST, TRMLST, CLKLST, WT2LST, WTLST.

Each fork is associated with one of the run queues, Q0 through Q3. One can think of the run queues as a linkage of forks on the GOLST, where each fork's queue number is stored in the table FKQ2. When a fork is removed from the GOLST and put onto a wait list, its run queue number is still remembered as well as the time it was placed on the run queue. This time is kept in the table FKTIME.

When the fork's wait time is over, it must be placed back on the GOLST, its placement determined by:

1. The queue number of either the former queue or new queue given to the fork as described in the previous section.

2. The fork's elapsed real time in the run queue before being placed on the wait list.

FILE SYSTEM

Data Structure

File names and pointers to files are kept in a directory. Directories are also named, with their names kept in a file called the ROOT-DIRECTORY. Each entry in the ROOT-DIRECTORY relates the name of the directory to its location. Two home blocks (one used as a backup) point to the ROOT-DIRECTORY. Whenever a directory is being referenced, the entire directory file is mapped into the monitor's process address space. Pages are then faulted in as needed. As far as disk page allocation is concerned, the monitor keeps a disk bit table to keep track of which pages are free and in use. This table is kept in a file on the disk. Figure MO-10 illustrates a structural overview of the file system.

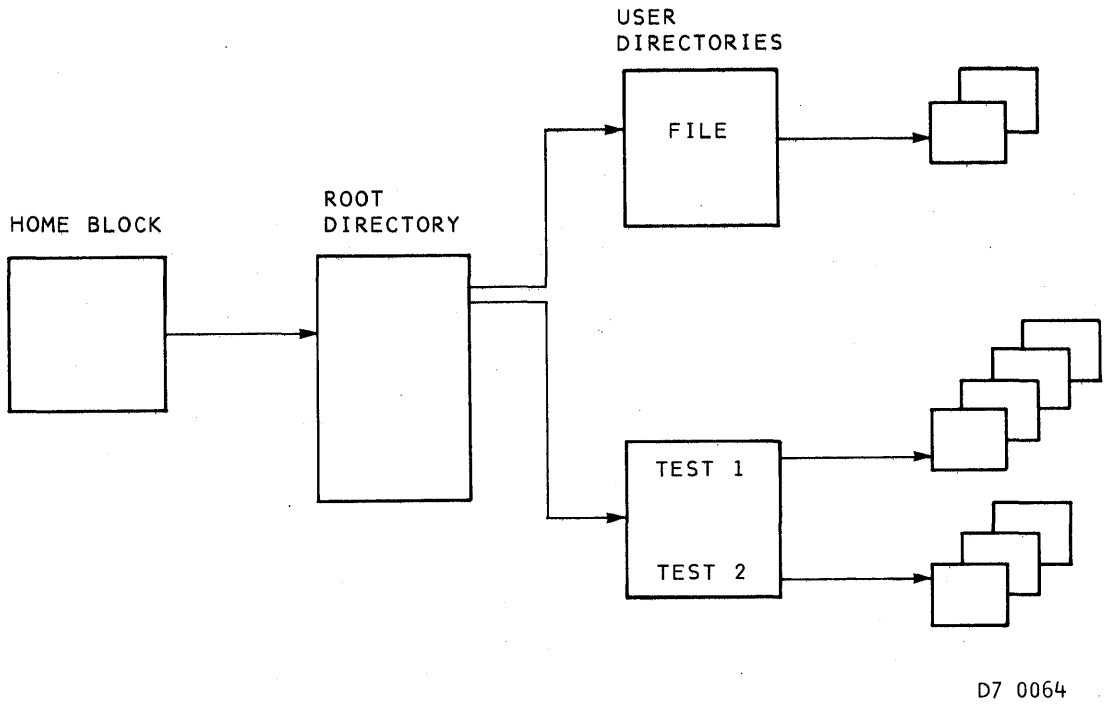


Figure MO-10. Extended File Structure

Each file on the disk (ROOT-DIRECTORY, user directories, and user files) has at least one index block of 512 words which is essentially a page table where each entry gives the location of one page of the file. A file may be logically as well as physically non-contiguous. A file that has page numbers $\leq$511 is called a short file. Directory files are always short files. Files having page numbers >511 are called long files. The location of index blocks for a long file is given in a higher level index block called the Super Index Block. The Super Index Block, being 512 words long, allows a user file to have 512*512 pages. See Figure MO-11 for a more detailed diagram of Figure MO-10.

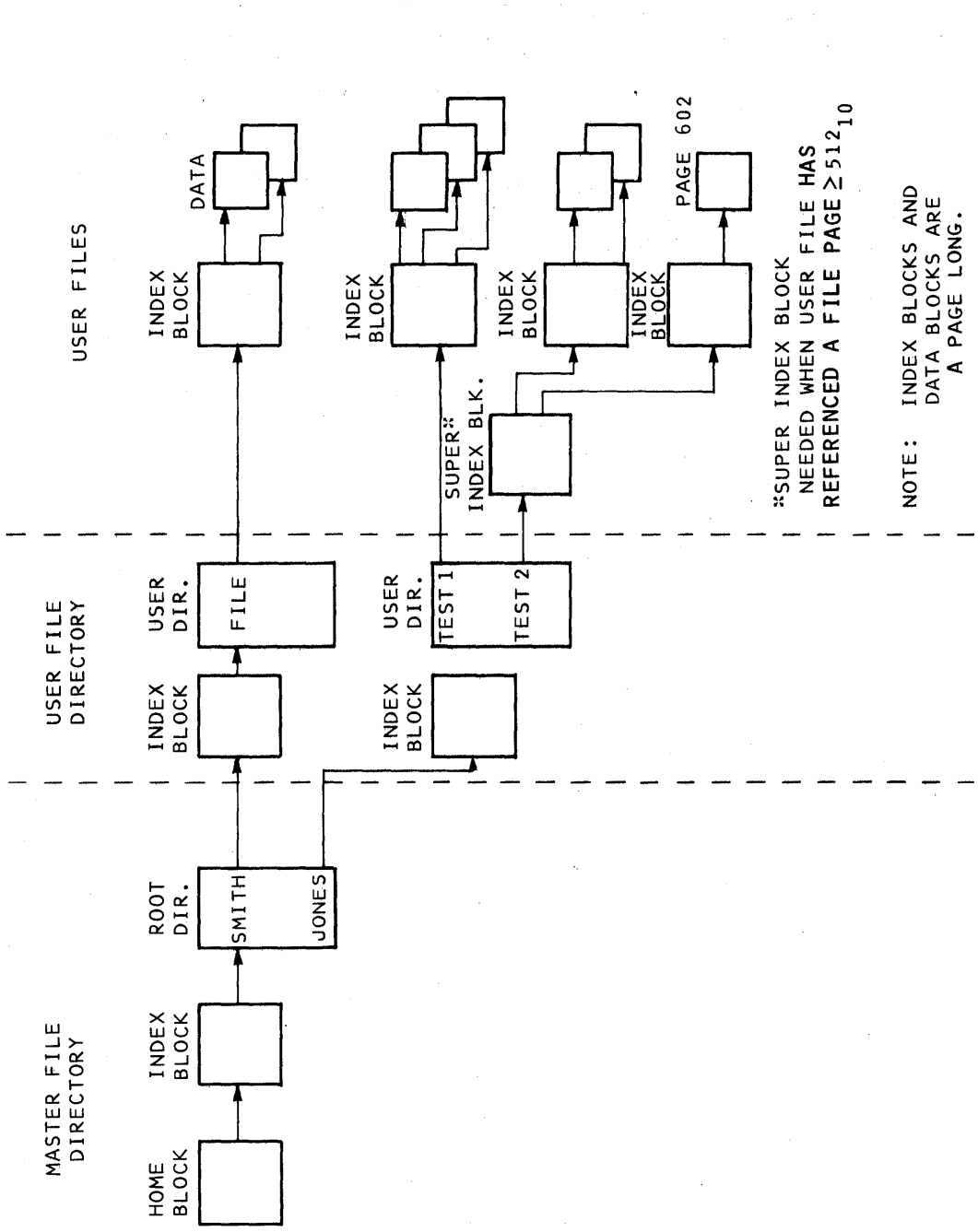
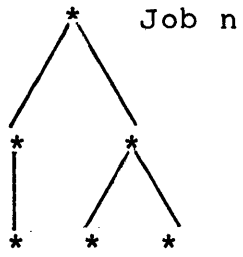


Figure MO-11. Full File Structure

JOB/FORK STRUCTURE

The monitor assigns a job number to a user upon login to a DECSYSTEM-2Ø. A job is a set of active resources normally under control of a single user. This set of resources is composed of one or more hierarchically-related processes (forks) that can communicate with each other in defined ways and that may contain several running and/or suspended programs. The set must always contain at least one process, with the Command Processor (the EXEC) being the most superior.



Except for the top-most process, a process has one immediate superior process and can have one or more inferior processes. A process can communicate with other forks of the structure by sharing memory, by the pseudo-interrupt system, or by direct control (superior to inferior only where the superior fork may confer or withhold privileges, suspend, continue or terminate inferior processes).

DISK AND MAGTAPE SERVICE

Hardware Principles

Each disk and magtape unit is connected to a controller (RH20) and all communications to that unit must go through the controller. The controller is connected to the CPU by the EBUS. Setup and startup operations are communicated over the EBUS to the controller. Internal data channels contained in the memory control logic of the processor are used for transferring data between memory and the controllers.

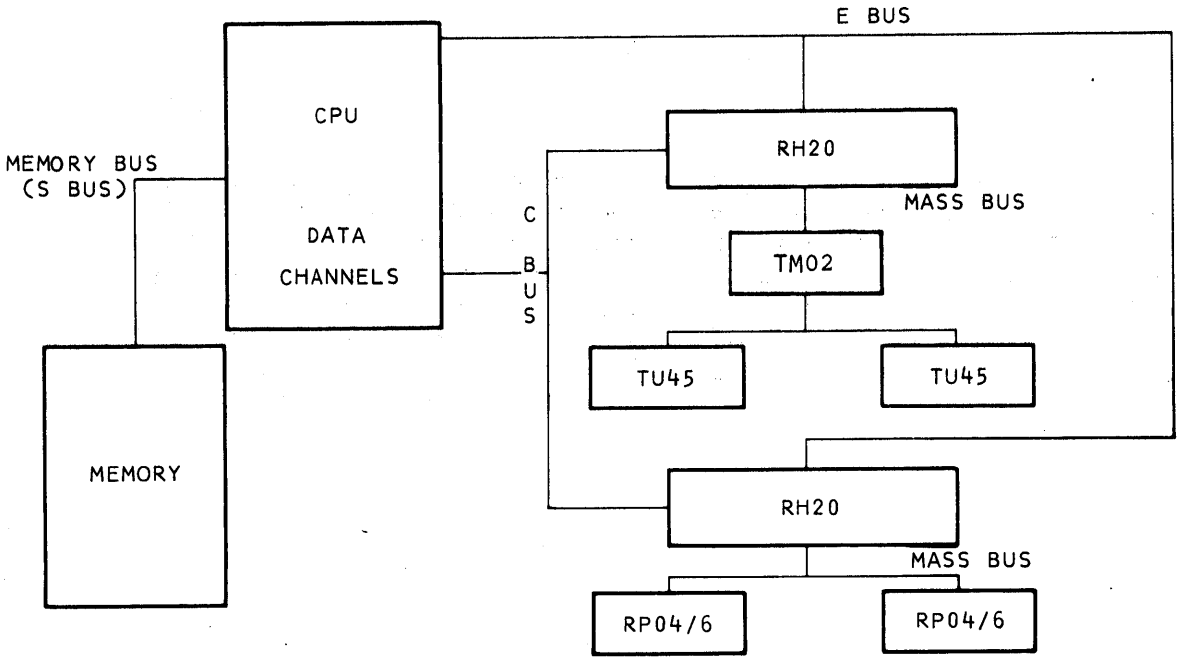


Figure MO-12. Hardware Configuration

Each RH2Ø is associated by the hardware with one of eight data channels. Each channel has its own 16-word buffer in the processor's memory control unit for data transfer and its own 4-word data channel area in the CPU's Executive Process Table (EPT) for holding the:

- Channel Command List Pointer
- Status when transfer done,, Adr. of Last Chan. Command Used
- Word Count,, Adr. of last data word transferred
- Not Used by Data Channel Hardware. (However, the system initialization software programs the RH2Ø to trap here on an interrupt, having stored the channel's vectored interrupt instruction in this word.)

Data transfers will be to or from consecutive locations on the disk/magtape but may be from or to non-contiguous core areas. The length and address of each of these core areas are on the channel command list. The list is pointed to by the channel's 4-word block in the EPT as shown above. The total length of the areas on the channel command list determine the number of words transferred.

All controllers can transfer data simultaneously, since each controller is connected to the memory system with its own built-in channel. Only one device per controller can transfer data at any one time, interrupting when done. But, non-data transfer commands (i.e., Seek, Rewind, etc.) can overlap and may be issued to any drive at any time. Also, non-data transfer commands interrupt on completion even during a transfer.

Each controller has a look-ahead command register, which enables the software to preload the next transfer request during the current transfer. Thus, the next transfer can begin with the next sector on the same device with no rotational delay.

When the controller has finished an operation, it interrupts the CPU via a vectored interrupt to the Exec Process Table (fourth word of its data channel area) so the CPU does not have to poll a series of devices to determine which device caused the interrupt.

Error checking is provided for both the channel and device data paths and the controller will terminate a command if certain errors are detected, marking the status word appropriately in the 4-word data channel area.

Monitor Modules

The TOPS-20 disk and magtape software have structures that parallel the hardware. At each level there is a module corresponding to a hardware device which interfaces to the rest of the monitor. All requests for disk or magtape operations are made to PHYSIO from PAGEM or MAGTAP, respectively (see Figure MO-13). PHYSIO handles physical I/O requests for the channel and device-independent portions of the initiation, termination and error-logging functions. Some of its responsibilities include:

- Manipulating device request queues, one per unit (RP04/6, TU45)
- Channel scheduling, (i.e., selection of the next operation to be done on a given channel)
- Seek scheduling (i.e., selection of the best request for seeking)
- Detecting timeouts associated with I/O operation
- Issuing "off-line" messages to the operator
- Initiating I/O retries. Errors are indicated in returns from the channel routine. If an error is to be logged, it makes a request to the error-logging routine
- Interrupt handling

PHYH2 is the primary downward interface to PHYSIO and handles the RH20 controller-dependent portions of initiations, termination, and error recovery. It is called on system startup to determine which devices are present and to initializa them via the appropriate device routines. It is likewise called to start I/O requests and when device requests have timed-out. When interrupts occur, it provides the principal interrupt analysis and calls the appropriate device routine.

Modules PHYP4 and PHYM2 contain the device-dependent code for the disk (RP04/6) and magtape (TU45), respectively.

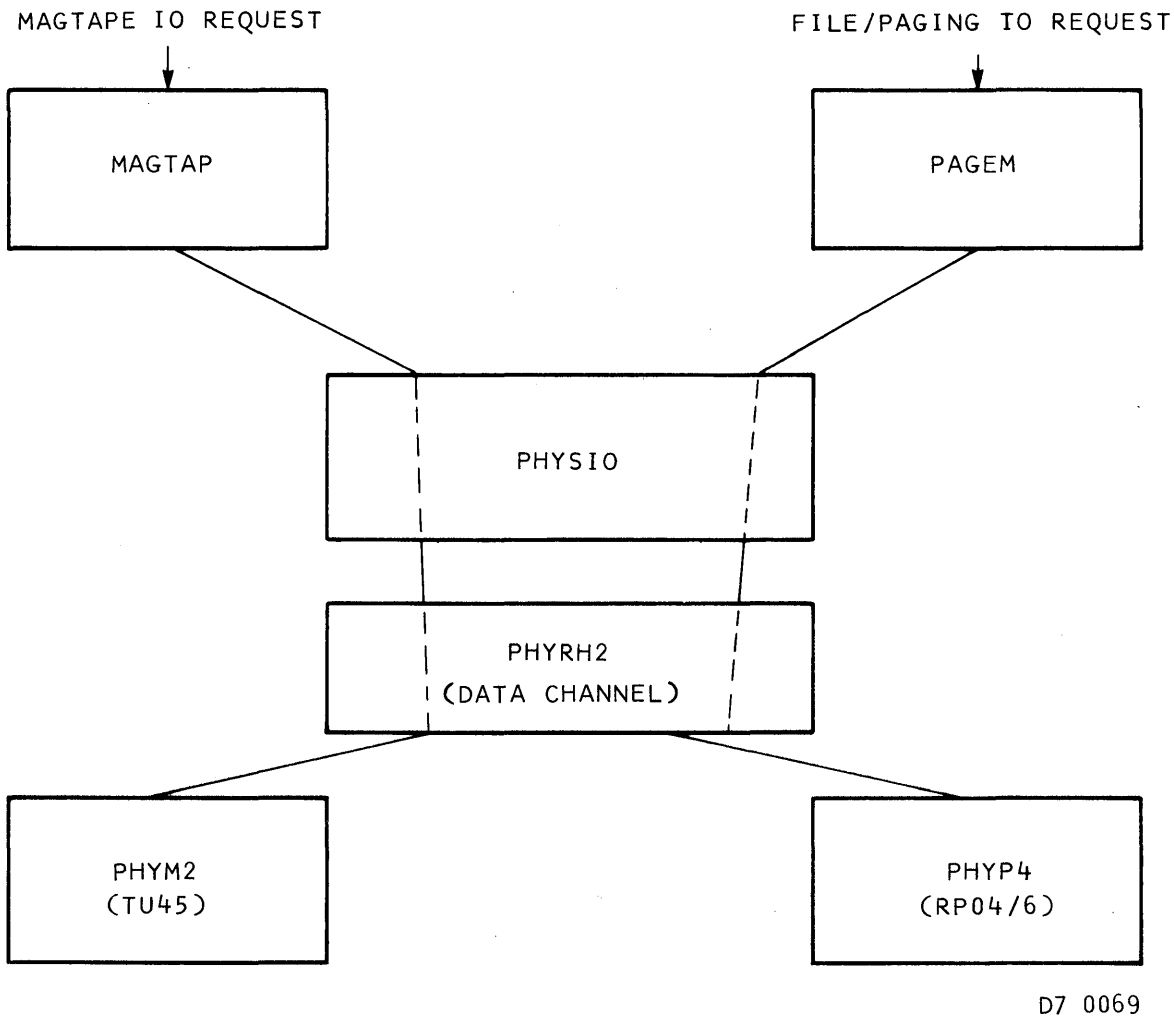


Figure MO-13. Physical I/O Request Paths

I/O Requests

I/O Request Blocks (IORBs) are built when an I/O function is requested from disk or magtape. The status word in this block contains the function (i.e., read, write, seek, backspace, etc.) and a bit indicating whether it is a disk or a magtape request (i.e., for PAGEM or MAGTAP, respectively).

Assume this is a read request for the disk. PAGEM calls PHYSIO with the IORB request, the physical core address and the device page address. PHYSIO determines the source unit of the request and places it in that unit's queue. If PHYSIO determines the device is idle (i.e., no other requests are pending) and needs positioning, it initiates a SEEK by calling the controller-dependent module, PHYR2, which in turn calls the RP04/6 driver. If the unit is positioned and the channel is idle, PHYSIO executes a Channel Schedule cycle which would initiate I/O for this request.

Figure MO-14 illustrates the data structure linkages for disk and magtape.

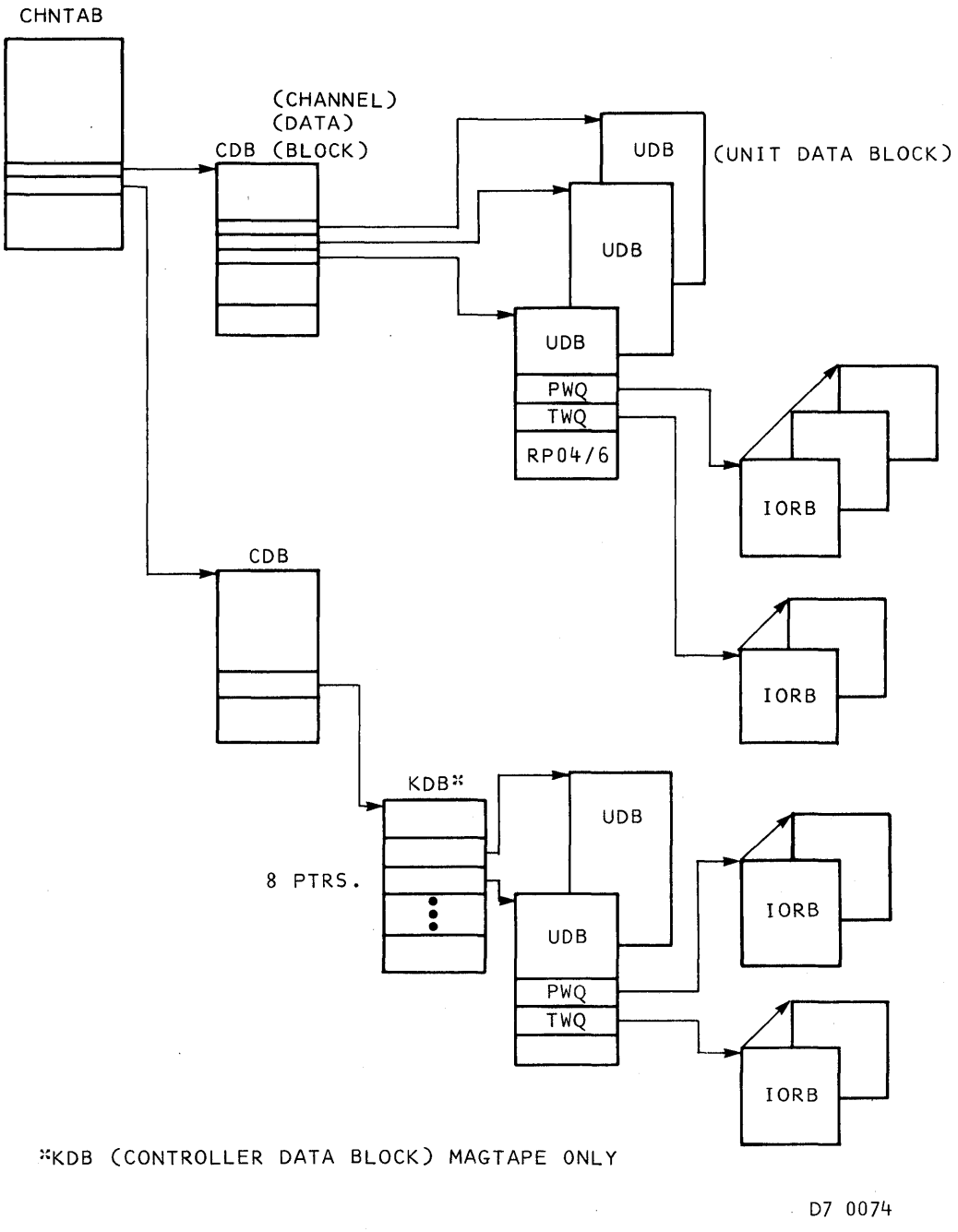


Figure MO-14. PHYSIO Data Structure Linkages

FRONT END SERVICE

This section is designed to cover the functional flow of information between the two processors of the DECSYSTEM-20, the PDP-11, and the KL10. All unit record equipment is attached directly to the PDP-11 Front-End Processor. Input and output data is buffered by the Front-End Processor to remove some of the burden of I/O handling from the KL10. A program running on the KL10 will pass an entire block of line printer data to the Front-End Processor, which will perform the necessary sequences to cause it to be printed. The communication link between the two processors is the UNIBUS device DTE20. As a UNIBUS device, it appears to the PDP-11 operating system, RSX20F, as a standard UNIBUS peripheral and the microcode on the KL10 side sees the DTE20 as a device on the EBUS.

TRANSFER BLOCKS

The primary protocol builds two types of transfer blocks:

1. Direct block - data is contained within the block

2. Indirect block - data is stored as a seperate buffer

Both blocks contain a count, function, device and a byte pointer. The device may be a pseudo-device similar to a PTY. The count indicates directly the number of data items to be transferred. The function field is used to indicate to the receiver the method to be used in storing this data. These blocks are 16-bit words to match the PDP-11 format and are referred to as a control packet. A control packet may be passed in either direction and may occur simultaneously to the transfer of a control packet in the opposite direction. These two blocks are described in Figure MO-15.

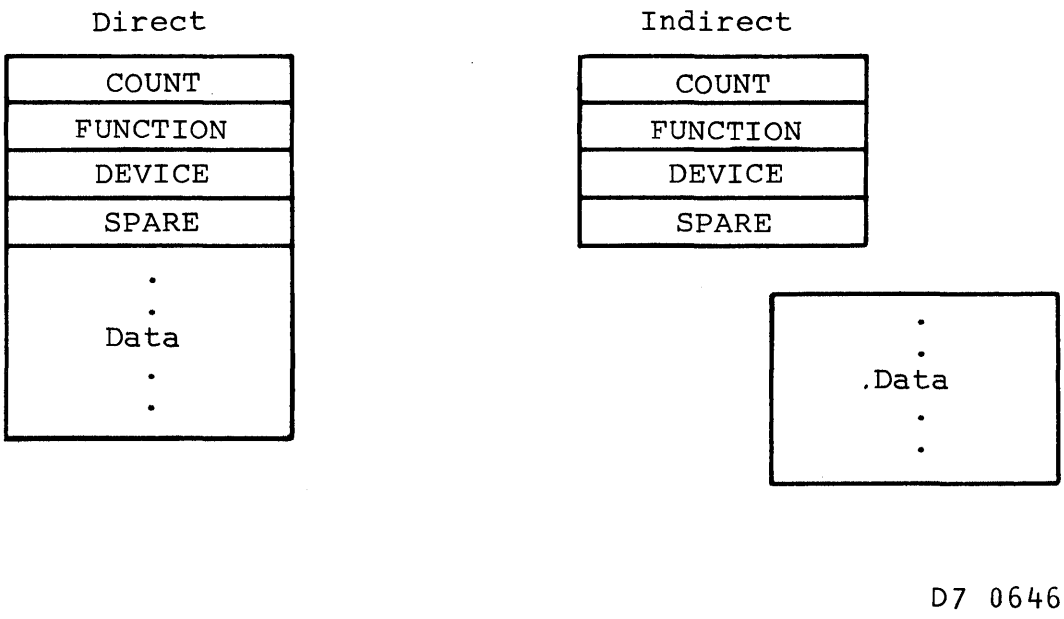


Figure MO-15. Transfer Blocks

TTY Input

Characters typed at the terminal are passed from the Front-End Processor to the KL10 processor where they are stored in a "Big Buffer". The monitor periodically sorts these characters by line number into line buffers until a wake-up character is recognized. This causes the requesting process to empty the line buffer and reset the required pointers to allow the next line to be accepted. The same type of mechanism is used to build output line buffers to be passed to the Front-End processor through its interface to the terminal (except that they may be indirect packets).

The block built by the Front-End contains a count, device, and function designator. The data received from the terminals is appended to this header to form a Direct Transfer Block. An example of this block is shown below.

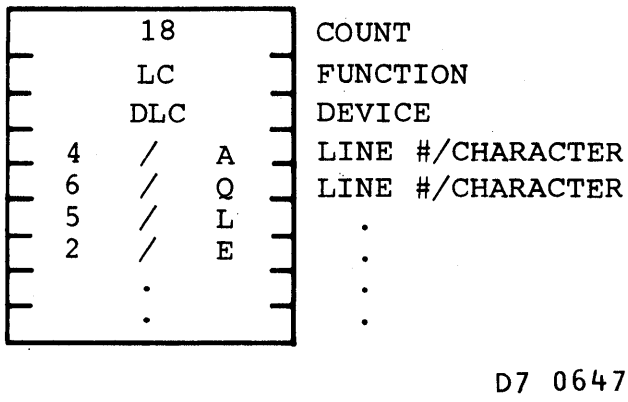


Figure MO-16. Direct Transfer Block (Packet)

This block is built by the operating system on the 11 side (RSX20F) and is transferred to a fixed area in the KL10 memory under control of the micro-code. The DTE20 connects the two processors and holds the transfer parameters to be used in requesting a transfer to the KL10. A protocol handler on the KL10 side takes the data from this fixed area or region in KL10 memory and appends it to the circular (ring) buffer known as "TTY Big Buff". Every 20 milliseconds the monitor empties this buffer into the line buffers. If a line buffer becomes full, further typing on the terminal will cause the character to be echoed as a bell. If a terminal (normally a buffered terminal) exceeds the space allotted to it in the input buffer, an XOFF will be sent to the terminal. This eliminates buffer overrun and allows the terminal to continue upon receipts of an XON from the monitor. This sequence is further illustrated in Figure MO-17.

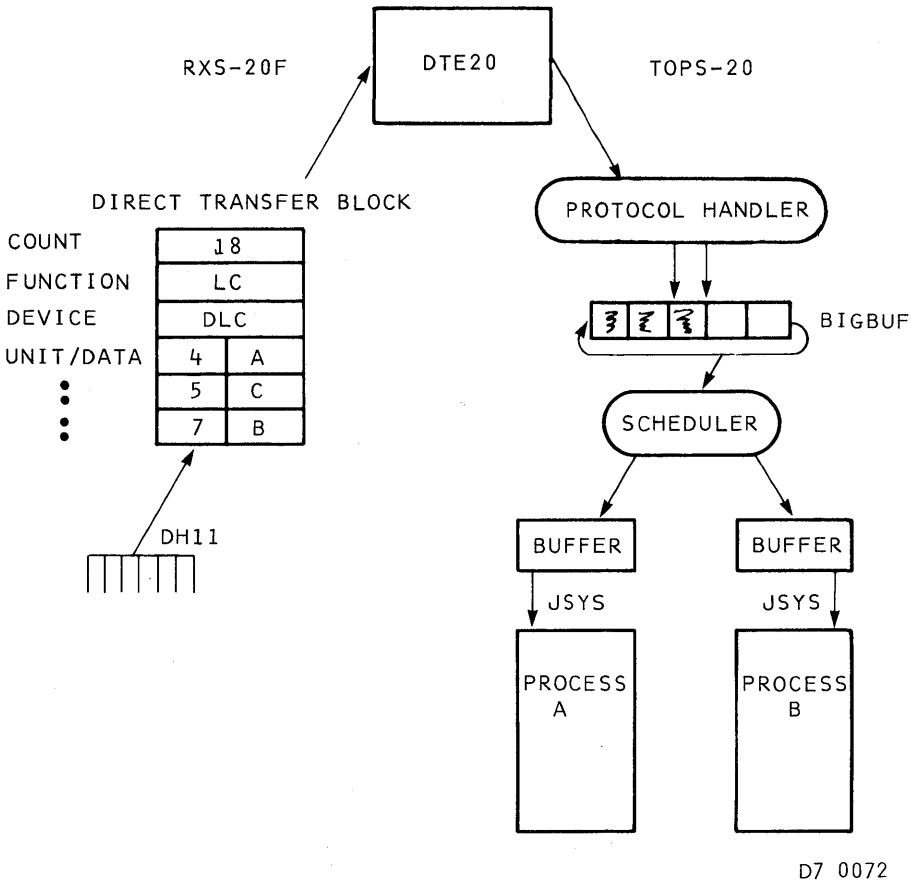


Figure MO-17. TTY Input Overview

An arbitrary limit of 100 characters has been set for a transfer block. Several direct transfer blocks may be linked together (appended to each other) and may be terminated by an indirect block. These blocks are passed between processors as a single request by setting the byte count in the DTE20 to be the sum of all blocks to be transferred.

Interrupts from the DTE20 indicate completion of the transfer into KL10 memory. The module DTESRV recognizes that a block is from a TTY by the device field. During the next 20 milli-second scheduler cycle, TTYSRV is called to sort the characters from "Big Buffer" into line buffers. A process blocked for input will remain in this state until a wake-up character is received. The set of wake-up characters consists of control characters, punctuation, letters and/or numbers as indicated by the parameters of the JSYS used.

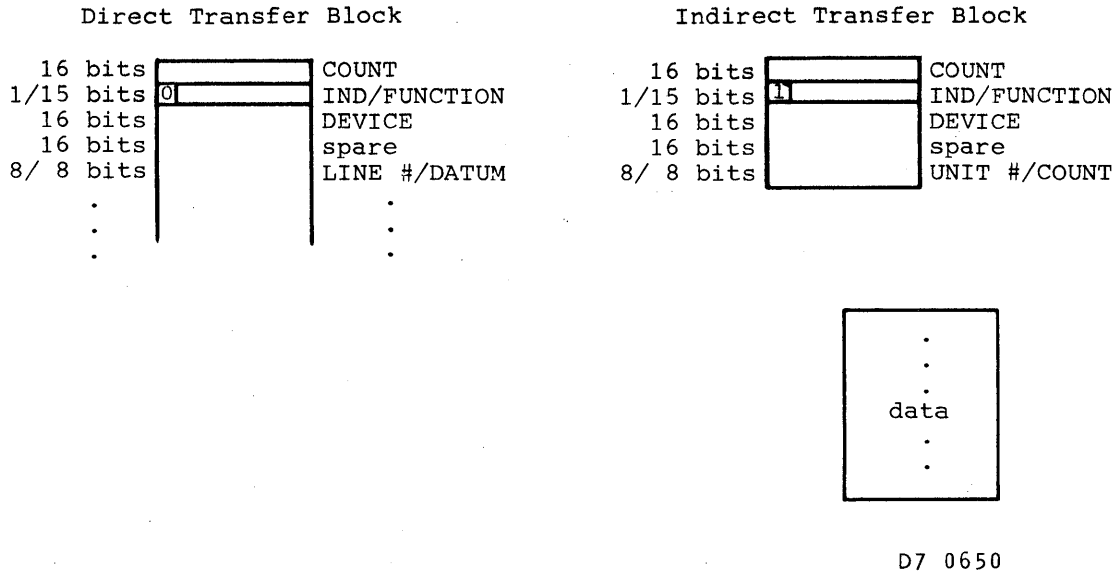


Figure MO-18. Detailed Transfer Blocks

TTY Line Buffers and Echoing

Two character buffers are assigned to each line that becomes active. One buffer is the input buffer which accumulates data as characters are moved from Big Buffer. A copy of each input character will be placed into the other buffer (the output buffer) when the character is to be echoed. When the input buffer is full, subsequent characters will be echoed as the bell control character. The output buffer consists of program characters to be printed on the terminal and input characters to be echoed.

Echoing, if not disabled, is either performed at the time the character is typed (immediate mode) or when it is passed to the program (deferred). The deferred mode allows immediate echoing if the process is in teletype input wait when the character is typed. Deferred is the standard echo mode and it produces a correctly ordered typescript (i.e., program input and output appear in the order they occur).

Line Printer Output

Data to be passed to the printer is represented by a header and an address. These are used by the protocol handler to deliver the data in the proper format to the Front-End, eliminating the requirement to BLT the data to a fixed area. The protocol between processors initiates two transfers: one to set up the Front-End to receive, and one to initiate the data transfer. Both are handled by the DTE-20 operating in primary protocol using Indirect Transfer Blocks and the doorbell mechanism.

Since line printer data and terminal output may consist of blocks of information that can be passed as contiguous data streams, the majority of KL10 to -11 transfers will use the indirect mechanism. (Device data for a specific device is also implemented by the indirect mechanism.) Transfers from the -11 to the KL10 use the direct mechanism supplying the line number and datum within the packet.

An overview of this process is shown in Figure MO-19. The JSYS that initiates a line printer transfer invokes a device driver. This device-dependent routine passes to the protocol handler the parameters necessary to transfer the block through the Front-End (to be handled and sequenced by

RSX2ØF) to the printer. The block built contains a count, function, and specific device. The KL1Ø wakes up the -11 to this transaction by ringing the 11's doorbell. At the time the doorbell is rung, the packet has already been set up and stored in the DTE2Ø. When the -11 acknowledges receipt, the data packet can then be set up and sent.

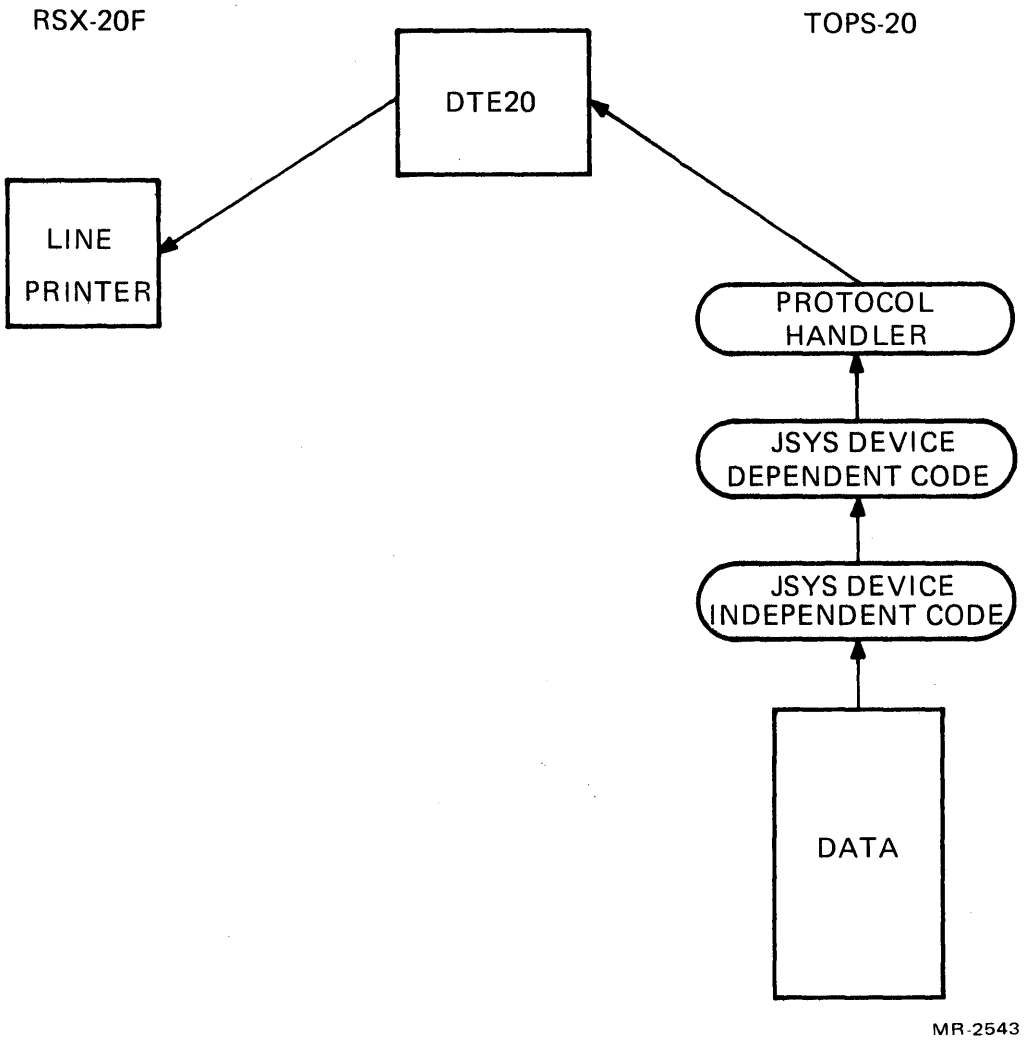


Figure MO-19. Line Printer Overview

This page is for notes.

MODULE TEST

When answering the lab exercises, write down the names of the tables where you found the answers. The labs will help you understand the monitor data base structure; so remember, where to look is more important than what you find there.

The exercises marked with a double star (**) are more difficult and are optional.

Answering some of the exercises requires use of the listings; do not assume that the answer is in the tables. After all, this is not a cookbook course.

TOOLS

FILDDT

FILDDT is a program that can be used to look at a crash or at the running monitor. Use FILDDT as in the example below to do this lab's exercises.

```
@ENABLE                              ;need enabled wheel
                                     ;capability to look at the
                                     ;running monitor.
$FILDDT                              ;start the program
FILDDT>LOAD <SYSTEM>MONITR.EXE       ;load the symbols
FILDDT>PEEK                          ;peek at the running monitor
```

At this point, the usual DDT commands allow you to look at the running monitor. You cannot change any locations (i.e., you have no write privileges). Also, your process will always be running when you look because the mechanism to look at the running monitor is like a JSYS whose function is to let you use DDT from monitor context.

EPT (Executive Process Table)

Introduction

The EPT is the hardware/software interface for most system-wide information. Some EPT locations must contain hardware-defined information. These locations are set up by the software and used by the hardware. For example, devices interrupt to locations in the EPT; the monitor's section pointers are in the EPT; and the exec mode overflow trap instructions are in the EPT. The software loads the base address of the EPT into a hardware register called the EBR (exec base register) so the hardware knows where the EPT begins.

The EPT is one page in length, but only a small percentage of the locations have hardware-defined uses. The remaining EPT locations are used solely by the software. Note also that the hardware uses of the table are different for TOPS-20 and TOPS-10; that is, the microcode differs in its use of the EPT.

The EPT is in the monitor's address space beginning at location KIEPT.

RESOURCES

1. Read and use as a reference section 3.4 of the Hardware Reference Manual; this section describes TOPS-20 paging and process tables.

2. The EPT is pictured in the Student Guide.

3. The EPT table is in the Monitor Tables.

EXERCISES

Vectored Interrupt Locations

The devices on the system use vectored interrupts; these interrupts all vector into the EPT (but do not need to). Vectored interrupts make polling all devices on the

same priority level unnecessary since each device can vector to its own assigned location. See section 3.1 (Priority Interrupts) of the Hardware Reference Manual for more information.

NOTE

Each interrupt location uses the instruction JRST 7,XXX (mnemonic XPCW). See the section on the JRST instruction (page 2-70) in the Hardware Reference Manual for information on how this instruction works.

1. Where does each DTE-20 interrupt and where does control pass?

2. Where do RH20s interrupt and where does control pass for each RH20?

3. Where does the interval timer interrupt and where does control pass?

Standard Interrupt Locations

Standard interrupts come in to the EPT at offset 40 + 2N where N is the priority interrupt level.

1. What is in each of the standard interrupt locations?

2. Where does control pass for each standard interrupt location and what processing happens there?

3. Which priority levels do you believe use standard interrupts?

Exec Mode Trap Instructions

The Exec mode PDL overflow and arithmetic overflow trap instructions are in the EPT. This is where the processor traps when one of these conditions occurs. The trap instruction in the appropriate location is executed.

1. What is the Exec mode arithmetic overflow trap instruction?

2. What will happen when this instruction is executed?

3. What is the Exec mode stack overflow instruction?

4. What will happen when this instruction is executed?

5. What is the trap 3 trap instruction?

Exec Mode Section Pointers

The Exec mode section pointers are in the EPT beginning at location MSECTB. The offset from MSECTB is the section number. On a model A machine, only section 0 can exist. On a model B machine, sections 0-5 are used. Refer to the section on extended addressing in your Student Guide for information on the use of each section on a model B machine. Note that the section pointers are "KL paging" format pointers.

1. How many sections are in use on the machine you are doing your lab with?

2. What kind of pointer(s) is/are in use for each section?

DTE-20 Control Blocks **

The EPT has space for four DTE control blocks, one for each possible DTE. Each block is eight words long. Refer to the EPT table in your Monitor Tables for a detailed layout of this area.

1. What are the examine protection and relocation words used for? What are they set up as in the running monitor? **

2. What are the deposit protection and relocation words used for? What are they set up as in the running monitor? **

3. What are the To -11 and To -10 byte pointers for? **

Channel Logout Areas

There is a channel logout area for each possible RH20 on the system. Refer to the EPT table in your Monitor Tables; do you know what each word is used for?

UPT

Overview

The UPT is the hardware/software interface for most process-specific information. The UPT is one page in length but only a few of the locations have hardware-defined uses. The remainder of the UPT page is used for software purposes only. The UPT page is called the PSB by the software. This page is called the UPT when referring to its hardware functions and the PSB when referring to its software functions.

Since the UPT/PSB page has process-specific information, it is context switched. The current process's UPT/PSB page is always mapped into the monitor's address space beginning at location PSB.

RESOURCES

1. Hardware Reference Manual.
2. UPT picture in the Student Guide.
3. PSB table in the Monitor Tables.

EXERCISES

Overflow Trapping

User push down overflow and user arithmetic overflow trap to locations in the UPT.

1. What is the stack overflow trap instruction?
2. What is the arithmetic overflow trap instruction?
3. What will happen when the stack overflow trap instruction is executed?

4. Using the information in the UPT, figure out what routine gets executed on a user-mode stack overflow. And then, using the microfiche, trace the routine; what does the routine do?

Page Fault Words

When a page fault occurs, the page fail word and the PC of the page fault are stored in the EPT; the processor picks up the new PC from the page fail new PC word (and switches to Exec mode if not already in Exec mode). Read the section entitled Page Failure on page 3-34 of the Hardware Reference Manual for more information.

1. What is in each of the page fault words?

2. Where does page fault handling begin?

3. What routine handles a page fault for the scheduler and what does this routine do when a page fault occurs? **

NOTE

When a page fault occurs, the processor always traps to the UPT, regardless of the mode the processor is in. Therefore, the scheduler has its own UPT which is context switched for the scheduler when it is running. A copy of this UPT is always a part of the monitor's address space (even when it is not being used as the current UPT); this copy begins at address SKHWPT.

User Section Pointers

The user section pointers begin at address USECTB. Currently, only one user section is supported. Note that these pointers are standard KL paging pointers.

1. What is the current pointer for user section 0?

2. What is the storage address for the user's page map?

Accounting Meters

For information on the accounting meters, read the section entitled User Accounts, page 3-52 of the Hardware Reference Manual. Note, however, that TOPS-20 does not currently use the accounting meters.

TEST EVALUATION SHEET

EPT (Executive Process Table)

EXERCISES

Vectored Interrupt Locations

1. Where does each DTE-20 interrupt and where does control pass?

   ANSWER: Each DTE interrupts into the third word of its DTE-20 control block in the EPT. For each DTE on the system, the DTE-20 control block is set up dynamically; the interrupt location contains a JRST 7,DTETRP+4n where n is the DTE-20 number. The old PC is stored in the first two locations of DTETRP+4n and the new PC is picked up from the second two locations of DTETRP+4n; the new PC is DTETRP+4n+5, which is always a JSR to SVDTRJ. Location SVDTRJ does a JRST to SVDTAC, which is where DTE interrupt processing begins.

```
KIEPT+142+4n/ JRST 7,DTETRP+4n

DTETRP+4n/ old PC flags
         / old PC
         / new PC flags
         / new PC = .+1
         / JSR SVDTRJ

SVDTRJ+1 / JRST SVDTAC
```

2. Where do RH20s interrupt and where does control pass for each RH20?

   ANSWER: Each RH20 vectors into the fourth word of its channel logout area in the EPT; that location contains a JRST 7,CDB-6 (i.e., six locations in front of the channel's CDB). Register P1 is saved in the CDB and then control is passed to routine PHYINT in PHYSIO.

```
KIEPT+3+4n/ JRST 7,CDB-6

CDB-6     / old PC flags
          / old PC
          / new PC flags
          / new PC = .+1
          / MOVEM P1,.+2+CDBSVQ
          / JSP P1,PHYINT
```

3. Where does the interval timer interrupt and where does control pass?

   ANSWER: The interval timer does a vectored interrupt to KIEPT+514; this location contains a JRST 7,TIMINT. The new PC is TIMIN0, which is in APRSRV.

```
KIEPT+514/ JRST 7,TIMINT

TIMINT    / old PC flags
          / old PC
          / new PC flags
          / new PC = TIMIN0
```

Standard Interrupt Locations

1. What is in each of the standard interrupt locations?

   ANSWER:

```
KIEPT+40/ 0
KIEPT+41/ 0
KIEPT+42/ 0
KIEPT+43/ 0
KIEPT+44/ 0
KIEPT+45/ 0
KIEPT+46/ JRST 7,PIAPRX          ;level 3
KIEPT+47/ 0
KIEPT+50/ JRST 7,PI4R            ;level 4
KIEPT+51/ 0
KIEPT+52/ JRST 7,PI5R            ;level 5
KIEPT+53/ 0
KIEPT+54/ JRST 7,PI6R            ;level 6
KIEPT+55/ 0
KIEPT+56/ JRST 7,PISC7R          ;level 7
```

2. Where does control pass for each standard interrupt location and what processing happens there?

   ANSWER:

   Channel 3 - new PC at PIAPRX+3 = PIAPR+1 in APRSRV; this routine handles interrupts for the APR device.

   Channel 4 - new PC at PI4R+3 = PISC4+1 in STG; this routine saves a few ACs, restores them and dismisses the interrupt. However, note that there is some code that is not assembled.

   Channel 5 - new PC at PI5R+3 = PISC5+1 in STG; this routine saves a few ACs, restores them and dismisses the interrupt. However, note that there is some code that is not assembled; if the machine is 2020, SMFLG=1 and the call to UNBCH5 is assembled.

   Channel 6 - new PC at PI6R+3 = PISC6+1 in STG; this routine saves a few ACs, restores them and dismisses the interrupt. However, note that there

is some code that is not assembled; if the machine is a 2020, SMFLG=1 and the call to DZCTIN is assembled.

Channel 7 - new PC at PISC7R+3 = PISC7+1 in SCHED; this routine handles the "software clock" that drives the overhead cycle.

3. Which priority levels do you believe use standard interrupts?

   ANSWER: For standard 2050/2060 monitors, only levels 3 and 7 are used. Level 3 handles the APR device (for memory parity errors, etc.) and level 7 is the "software clock" that drives the scheduler. On a 2020, since there is no front end, more priority interrupt levels are used to handle devices that the front end handles on a 2050/2060 or that do not exist on a 2050/2060. For a 2020, the Unibus adapter is on channel 5 and uses routine UNBCH5; terminal interrupts are on channel 6 and are handled by routine DZCTIN.

Exec Mode Trap Instructions

1. What is the Exec mode arithmetic overflow trap instruction?

   ANSWER: Exec mode arithmetic overflow traps to offset 421 in the EPT. This location contains a JFCL.

2. What will happen when this instruction is executed?

   ANSWER: Nothing; it is a no-op.

3. What is the Exec mode stack overflow instruction?

   ANSWER: Exec mode stack overflow traps to offset 422 in the EPT. This location contains: .PDOVT = 42000,,0

4. What will happen when this instruction is executed?

   ANSWER: 42 is an illegal op-code and is therefore handled as a UUO. However, since it is executed from a trap instruction, the new PC is picked up from the Kernel trap MUUO new PC word in the UPT (offset 431). Offset 431 in the UPT (PSB) contains: 1,,KITRPS. Therefore, when a push down list overflow occurs in Exec mode, routine KITRPS is executed. This routine executes a MONPDL BUGHLT.

5. What is the trap 3 trap instruction?

   ANSWER: If the arithmetic overflow and stack overflow PC flags are on at the same time, the machine traps to the trap 3 trap instruction. This never happens unless the PC flags are deliberately set.

Exec Mode Section Pointers

1. How many sections are in use on the machine you are doing your lab with?

   ANSWER: Section pointers are set up for sections Ø-3 whether the machine is a model A or a model B machine. However, only section Ø is used on a model A machine. Sections Ø and 1 have the same section pointer. Section 2 has an indirect pointer through location DRMAP in the current PSB; DRMAP contains a zero for model A machines. Section 3 has an indirect pointer through location IDXMAP in the current PSB; IDXMAP contains a zero for model A machines. Location DIRORA (the directory origin) contains [DRSECN,,Ø] for Model B machines and [MSEC1,,740ØØØ] for Model A machines. Location EXADDR is used as a flag to note whether this is a model A or model B machine; if EXADDR contains Ø, this is a model A machine.

2. What kind of pointer(s) is/are in use for each section?

   ANSWER : Sections 0 and 1 have share pointers to MMAP (contents of MMSPTN). Sections 2 and 3 have indirect pointers through locations in the PSB. Section 4 is used to map the bit table; the bit table is only mapped while it is being updated.

DTE-20 Control Blocks **

1. What are the examine protection and relocation words used for? What are they set up as in the running monitor? **

   ANSWER: The examine relocation word is the first word of KL memory that a DTE-20 has read access to; the examine protection word is the size of the region (beginning at the examine relocation word) that this DTE-20 can read.

```
KIEPT 144/ 225
KIEPT 145/ COMBUF+3

KIEPT 144+10/ 226
KIEPT 145+10/ COMBUF+2
```

2. What are the deposit protection and relocation words used for? What are they set up as in the running monitor? **

   ANSWER: The deposit relocation word is the first word of KL memory that a DTE-20 has write access to; the deposit protection word is the size of the region (beginning at the deposit relocation word) that this DTE-20 can write into.

```
KIEPT 146/ 30
KIEPT 147/ COMBAS+60

KIEPT 146+10/ 30
KIEPT 147+10/ COMBAS+110
```

3. What are the To -11 and To -1Ø byte pointers for? **

   ANSWER: The To -11 byte pointer is used as the byte pointer to the data for indirect transfers; the To -1Ø byte pointer is used as the byte pointer for where to put the data transferred from the PDP-11.

Channel Logout Areas

There is a channel logout area for each possible RH2Ø on the system. Refer to the EPT table in your Monitor Tables; do you know what each word is used for?

ANSWER: The first three words contain channel command list information for the last transfer on that channel. Word 4 is the vectored interrupt location for the channel.

UPT

EXERCISES

Overflow Trapping

1. What is the stack overflow trap instruction?

   ANSWER: Offset 422 in the UPT (PSB) is the stack overflow trap location; it contains .PDOVT = 420000,,0

2. What is the arithmetic overflow trap instruction?

   ANSWER: Offset 421 in the UPT (PSB) is the arithmetic overflow trap location; it contains JFCL (unless the user has enabled the PSI system to handle arithmetic overflow).

3. What will happen when the stack overflow trap instruction is executed?

   ANSWER: Op-code 42 is an illegal instruction and is handled as a UUO except that it is being executed as a trap instruction; therefore, the new MUUO PC is picked up from the concealed trap MUUO new PC word which is offset 435 in the UPT. This address is KITRPU.

4. Using the information in the UPT, figure out what routine gets executed on a user-mode stack overflow. And then, using the microfiche, trace the routine; what does the routine do?

   ANSWER: KITRPU in APRSRV is executed; this routine looks to see the if the PDL overflow channel is enabled for this fork or its superior and initiates the interrupt.

Page Fault Words

1. What is in each of the page fault words?

   ANSWER: The page fault words contain the following:

   UPTPFN= PSB+500 / page fail word --contains failure type code and virtual address that page faulted.

   UPTPFL= PSB+501 / page fail flags (old PC)

   UPTPFO= PSB+502 / old PC

   UPTPFN= PSB+503 / new PC = PGRTRP

2. Where does page fault handling begin?

   ANSWER: PGRTRP in PAGEM

3. What routine handles a page fault for the scheduler and what does this routine do when a page fault occurs? **

   ANSWER: Offset 503 of the SKHWPT is the new PC for page fault handling; this location contains address KIPFS. This routine executes a BUGHLT because the scheduler is currently not expected to page fault.

User Section Pointers

1. What is the current pointer for user section 0?

   ANSWER: USECTB+0 is the section pointer for section 0; it contains a share pointer to the page map for section 0.

2. What is the storage address for the user's page map?

   ANSWER: The SPT slot indicated by the share pointer contains the storage address for the user's page map.

This page is for notes.

APPENDIX A

MONITOR MODULES

I. MONITOR KERNEL

A. Management & Allocation

APRSRV
FORK
LDINIT
PAGEM
POSTLD
SCHED
SWPALC

B. Device Drivers

CDRSRV
DTESRV
IMPANX
LINEPR
PHYH2
PHYM2
PHYP4
PHYSIO
TTYSRV

II. FILE SYSTEM

DEVICE
DIRECT
DISC
DSKALC
FESRV
FILINI
FILMSC
FREE
GTJFN
IMPDV
IO
JSYSA
JSYSF
LOGNAM
LOOKUP
MAGTAP
MSTR
NETWRK
NSPSRV

III. UTILITY MODULES

BOOT
COMND
DATIME
DIAG
EDDT
ENQ
FUTILI
IPCF
MDDT
MEXEC
MFLIN
MFLOUT
MR
SYSERR
TAPE
TIMER

IV. DATA BASE

GLOBS
PARAMS
PHYPAR
PROLOG
SERCOD
STG

MONITOR KERNEL

A. Management & Allocation

APRSRV

This is the processor dependent service module which contains the initialization code for paging, MUUO handlers, and the priority interrupt system as well as for the clocks, APR, and DTE devices. Interrupt handling for these devices, pager control routines, and pre- and post- JSYS handling is also performed here.

FORK

Fork controlling JSYSs and support code.

LDINIT

At load time, this module defines storage PCs for the JSYS dispatch table JSTAB.

PAGEM

TOPS-20 page management code; core management routines, swapper routines, pager trap logic, OFN control, and CST and SPT initialization.

POSTLD

This code runs immediately following the loading of the monitor and performs functions outside the capabilities of LINK. It will move the symbol table to its runtime location, pruning it as necessary. It will then build the MONITOR.EXE file, write a BUGSTR text file and delete itself from core.

SCHED

This module contains the Channel 7 interrupt routine (which performs context switching), the TOPS-20 scheduler, the job/fork initialization/dismiss

routines, and the Program Software Interrupt (PSI) analysis and resolution routines.

SWPALC

This is the swapping space allocator which handles a device of some number (SWPSEC) of sectors, and some number (DRMMXB) of tracks. It has a resident bit table used to allocate swapping storage.

B. Device Drivers

CDRSRV

Card Reader Service.

DTESRV

DTE Service Driver; protocol handler for requests to and from the Front-End.

IMPANX

This module contains the device dependent code for the Interface-Message-Processor (IMP) device connected to a -10/-20 on the ARPA-network.

LINEPR

Lineprinter service.

PHYH2

Channel-dependent code for RH20 controller at direct I/O level.

PHYM2

Channel-dependent code for TM02/TM45 magtapes at direct I/O level.

PHYP4

Device-dependent code for RP04/6 disks at direct I/O level.

PHYSIO

This module handles the channel and driver I/O routines. It is responsible for queueing I/O requests into their proper queue, choosing the "best" request for seeking and/or transferring and starting I/O.

TTYSRV

This is the terminal service module containing the TTY I/O drivers, special control character conversion routines, terminal JSYS routines and the interface to the primary and secondary protocols in DTESRV. Its device dispatch table is contained in FILMSC.

FILE SYSTEM

DEVICE

Device and initialization look-up code.

DIRECT

Disk file and directory management code.

DISC

This module contains the pre-PHYSIO disk-dependent routines for I/O JSYSs and a dispatch table of vectored addresses (DSKDTB) which points to them.

DSKALC

Drive-independent code for disk block allocation.

FESRV

Device code for FE devices. This code contains the device-dependent routines for the FE pseudo-devices FEØ-FE3.

FILINI

This module contains code to initialize the file system at system startup.

FILMSC

This module contains miscellaneous routines for the PTY, TTY STRING and NULL I/O devices and also includes a device dispatch table for each of these devices.

FREE

Job storage free area management.

GTJFN

Contains the code for GTJFN, and the JSYSs supporting look-up, recognition, and creation of file names.

IMPDV

This module contains the Interface-Message-Protcol (IMP) device-independent code. It runs cyclically as a separate fork (i.e., under JOBØ) and handles the interface to the ARPA network by monitoring network activity and managing the message queues.

IO

Contains most of the device-independent sequential, random, and dump input/output routines for BIN, BOUT, SIN, SOUT, DUMPI, and DUMPO.

JSYSA

Random JSYSs for system and directory access, device allocation, job parameter settings, system accounting (EFACT) and file/fork mapping (PMAP).

JSYSF

Contains code that implements various file system JSYSs.

LOGNAM

Contains the logical name definition and recognition JSYSs and routines.

LOOKUP

Device-independent file name look-up.

MAGTAP

This module contains the pre-PHYSIO magtape-dependent routines for I/O JSYSs and a dispatch table of vectored addresses (MTADTB) which points to them.

MSTR

Contains the code to implement the mountable structure JSYS, MSTR.

NETWRK

This module contains the interface for all standard I/O JSYSs that communicate with the ARPA-network. It also provides a finite state machine of various events associated with a connection for the network control program (NSP).

NSPSRV

This module contains the control routines and JSYS interfaces for the host-to-host protocol of DECnet known as NSP, which allows communication between processes on hosts by means of logical links.

UTILITY MODULES

BOOT

BOOTSTRAP for the system.

COMND

Code for the COMND JSYS, which is used by user programs for consistent command parsing.

DATIME

Code for the date and time conversion JSYSs.

DIAG

This module contains code to support the DIAG JSYS for the KL1Ø.

EDDT

Exec mode DDT is loaded as part of the resident monitor and used for debugging basic monitor functions.

ENQ

This module implements the ENQ/DEQ facility to control simultaneous access to user specified sharable resources.

FUTILI

Contains routines to copy strings to/from JSBs and routines to get a yes/no answer from CTY.

IPCF

Code for the system interprocess communications facility; Code for [SYSTEM] IPCC.

MDDT

This is a version of DDT which runs in the monitor space for debugging processes on TOPS-20. This version runs under timesharing.

MEXEC

This module contains the MINI-EXEC which is a limited command interpreter for certain system loading/maintenance functions and swappable monitor bootstrap procedures. It is part of the swappable monitor and also contains many JSYS routines.

MFLIN

Floating point input and conversion JSYSs.

MFLOUT

Floating point output and conversion JSYSs.

MR

Floating point double-precision arithmetic routines.

SYSERR

Error reporting module for field service.

TAPE

This module contains the tape-table handler and record processor.

TIMER

This module implements the TIMER JSYS and all of its support. This includes scheduler clock routines (called from CLK2CL) and the code to kill the pending clock belonging to a dying fork (KSELF).

DATA BASE MODULES

GLOBS

All globals are defined as external here with QEXT macro. (Note, do not confuse this module with the global cross-reference file, GLOB, produced at monitor assembly time.)

PARAMS

This module contains one of the parameter files, PARBCH, PARSML, PARMIN, PARMED, PARBIG, depending on the size of the monitor required by the installation. The parameters in these files affect the space allocated for swapping and monitor resident tables. The assembled executable code is not affected.

PHYPAR

Universal file for PHYSIO and associated modules. It contains the definitions for the Channel Data Block, Channel Dispatch Table, Unit Data Block, Unit Dispatch Table, and the Input/Output Request Block.

PROLOG

This is a file of parameters, storage assignments, and macro definitions. The major regions of the monitor address space are defined as well as macros affecting PI bug strings, pseudo-interrupts, and scheduling. All PSB and JSB storage defined by the monitor at assembly time is specified here.

SERCOD

This module contains the error codes and fields for SYSERR, a program which produces hardware performance reports for field service personnel.

STG

The bulk of the monitor storage, both resident and nonresident, is defined in this module.

This page is for notes.

APPENDIX B

MONITOR'S VIRTUAL ADDRESS SPACE

Every user job in TOPS-20 comprises one or more processes. Each process has two memory maps which describe the virtual core assignment for the process when it is executing in user or monitor mode. The user page map completely describes the user space for the currently running process; the executive page map describes the mapping for the monitor's address space. The user space and

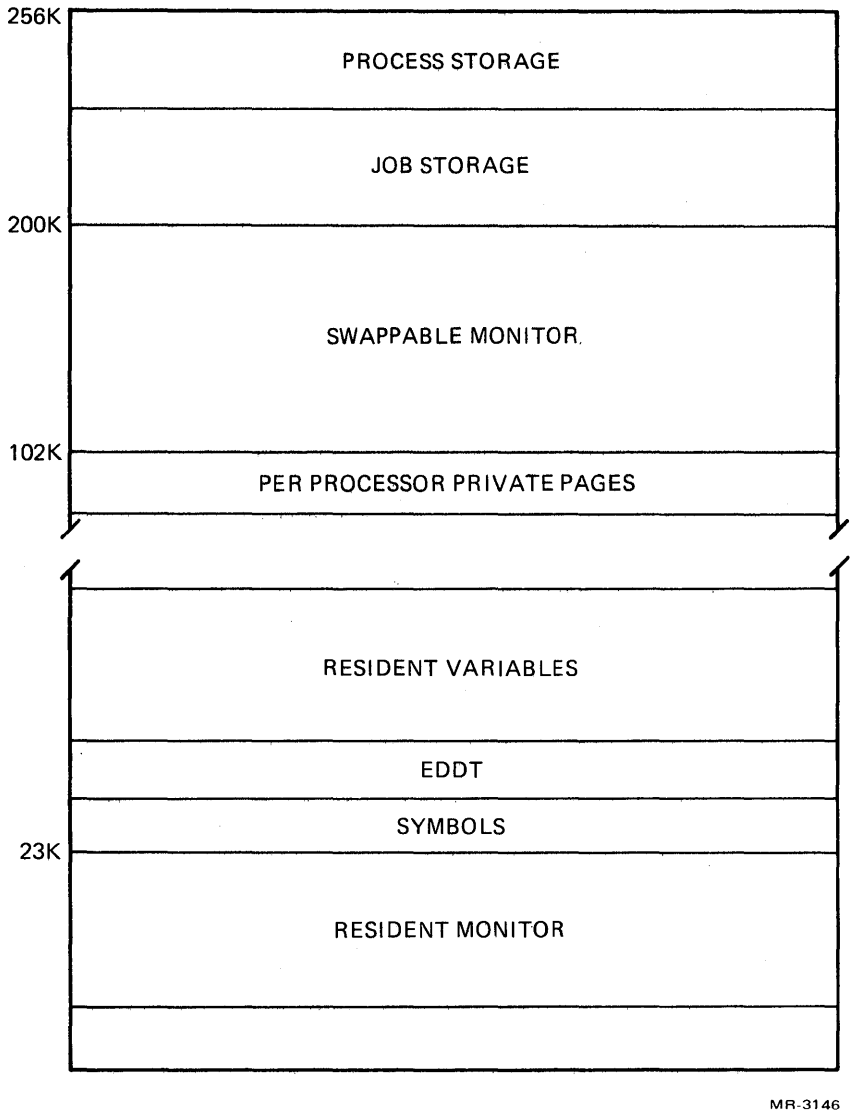


Figure B-1. Monitor Virtual Memory Map

the monitor space are each 256K. New maps are loaded when a new process is selected to run, with the user map reflecting the current allocation of the user's virtual address space, and the monitor map reflecting the per-process allocation of the monitor's address space. Notice, however, that the major portion of the monitor's page map is used to map the resident and non-resident parts of the TOPS-20 monitor. Figure B-1 is a simplified diagram of the monitor's virtual memory map.

The monitor consists of two logical sections: resident and non-resident code. Resident code is non-swappable and contains the scheduler, pager, basic interrupt and JSYS dispatch handlers and tables. The non-resident portion on TOPS-20 consists of swappable code and data which may or may not be in core at any given time (depending on system utilization). The RESCD macro causes code to be placed in the resident portion of the monitor. The SWAPCD macro will allow it in the swappable portion. The resident portion of the monitor is the same for all processes and will be discussed first.

RESIDENT MONITOR

Page 1 of the monitor is the JSYS dispatch table: 1000 words long with one entry for each JSYS. The right half of each entry contains the monitor PC describing where to start executing. Page 2 is the Executive Process Table. Page 3 is the scheduler page table which is used by the scheduler when it is called to select a new process to run.

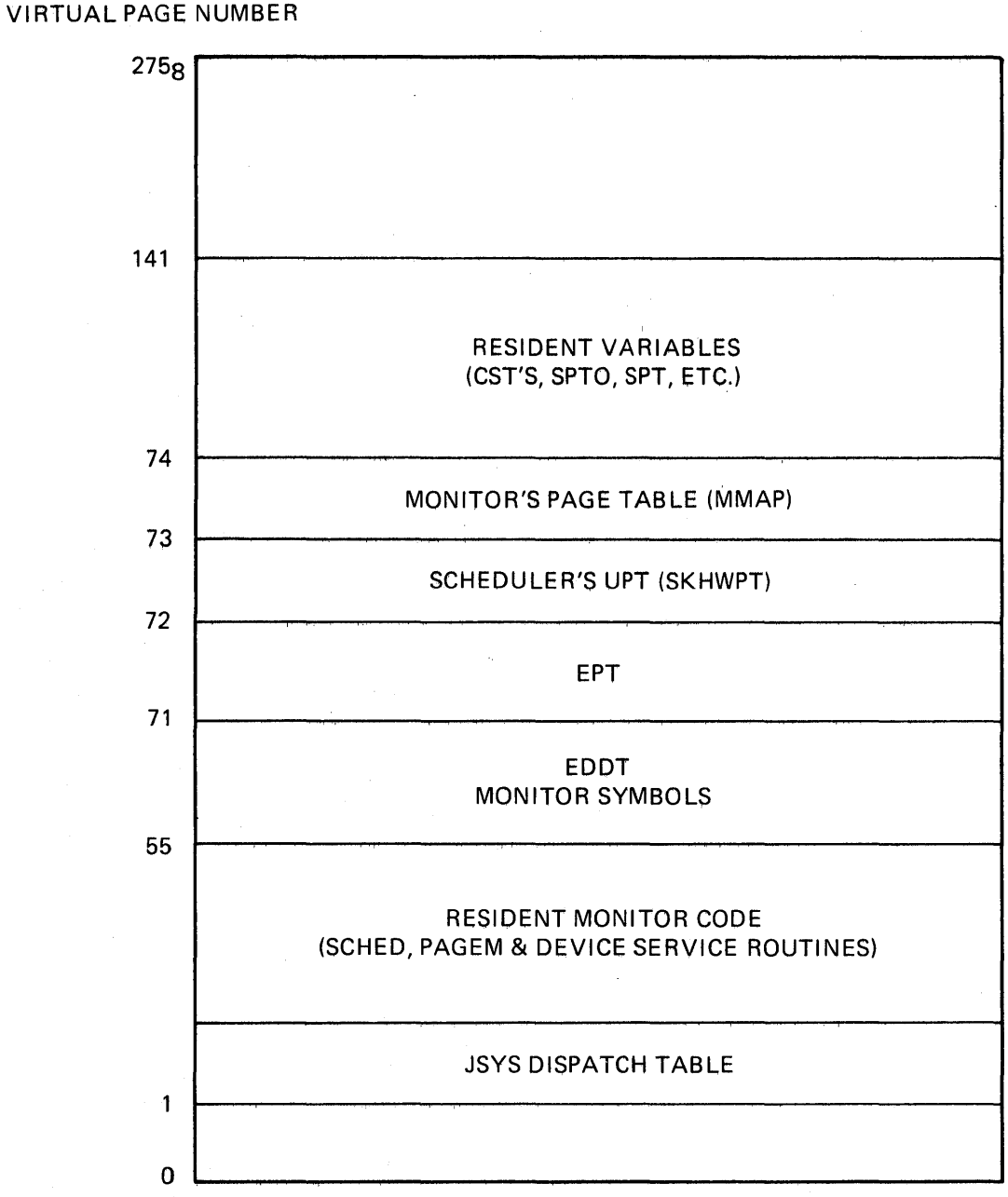


Figure B-2. Resident Monitor

Pages 4-60 octal house the resident variables (i.e., those defined by the RS macro). Included here are the SPT, SSPT, and CST tables used by the paging hardware. The size of this area will vary depending on the size of the system.

Resident monitor code sits between pages 60 and 140 and contains the paging and scheduling software and peripheral device interrupt and error processing routines. The symbol MONCOR points to the highest location used by the resident monitor code. Above this sits an optional area consisting of EDDT and symbols. Since this area is 30-40 pages, it is only in core when the system is being debugged.

NON-RESIDENT MONITOR

The swappable code of the monitor really starts at page 270 (see Figure B-3). Pages 270-276 belong to the per-processor private page area. Since the current implementation supports only one CPU, this area can be used by the swapper for temporary pages. Pages 277-323 contain swappable variables the monitor needs and are reserved by the NR macro. A free space pool also sits here and includes temporary storage for ENQ/DEQ and IPFC. Pages 324-527 contain the swappable monitor containing the bulk of the JSYS code, MDDT and symbols, and the MINI-EXEC. This area is write-protected. Pages 530-617 are swappable page storage reservable by the NP macro. Here lies a one-page buffer for each line printer and any other device that needs one.

Pages 620-717 are the Job Storage Block, an area that is context-switched whenever a new job is run but is shared among all forks in a job. This area contains descriptor blocks (approximately 20 words in length) for every JFN known to the user. These blocks reflect the current state of the opening of a file. Pointers to filename, type, version, directory and device names, and pointers to the OFN and wildcard masks are stored. The GTJFN JSYS builds these blocks; the actual ASCII strings are stored in a free space pool in the JSB. Information is kept in the JFN for File I/O as well (i.e., the file's page, byte position, status, length, etc.).

Pages 720-777 comprise the Per-Process Storage Area. This virtual core is switched every time a new process is run and is not the same for all forks in a job. Included here from pages 740-770 is the directory currently being scanned, which is PMAPed directly into this space for use by GTJFN and other related JSYSs. Pages 720-737 are used for special utility fork and swapper functions and the IDXFIL table. Page 777 is the Process Storage Block which contains a per-process pushdown list and a monitor page table for pages 720 and up. Also included here is space for saving multiple sets of ACs needed in the situation of nested JSYS calls, and tables for holding local fork handles.

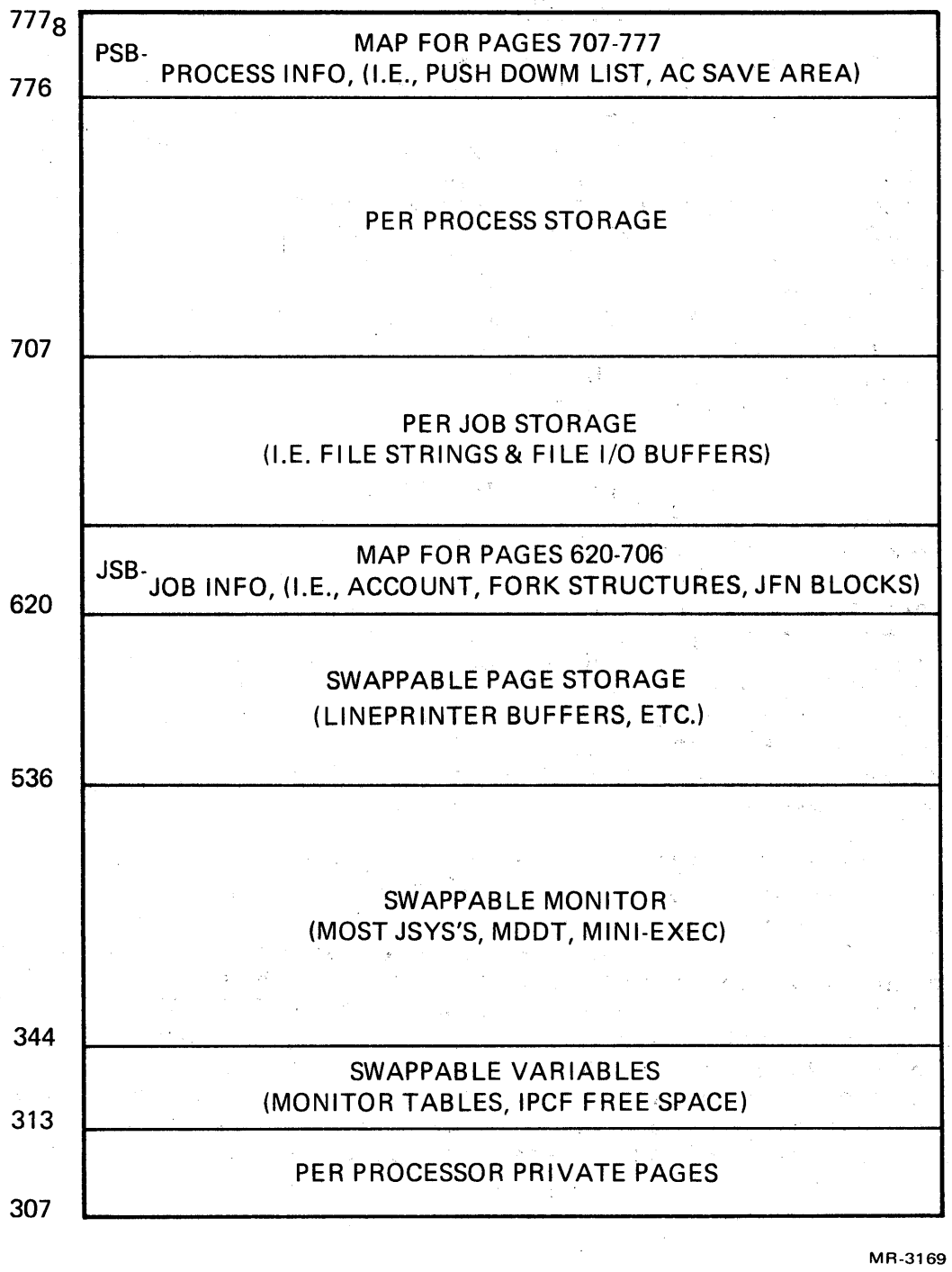


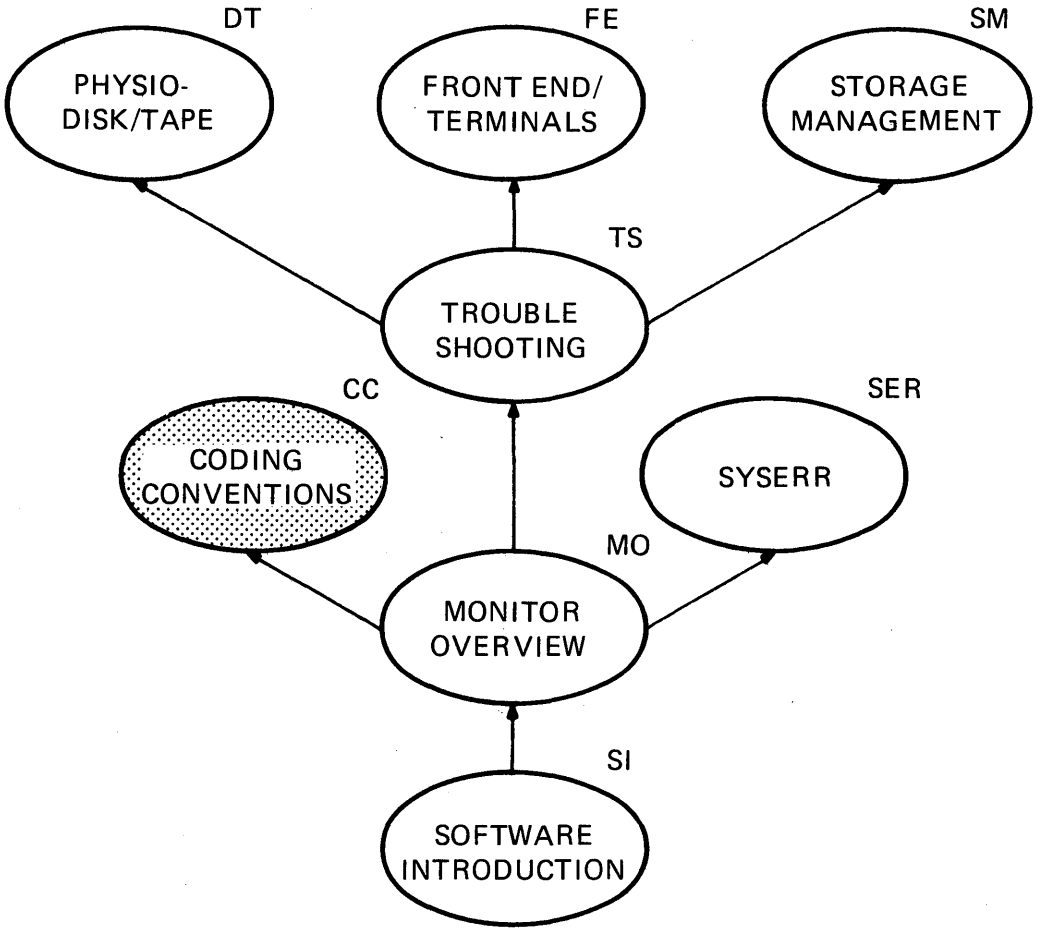
Figure B-3. Nonresident Monitor

TOPS-20 MONITOR

Coding Conventions

<<For Internal Use Only>>

COURSE MAP



MR-2717

This page is for notes.

Coding Conventions

INTRODUCTION

There is a set of standard macro and symbol definitions used in writing monitor code (and some support programs). This module will cover the use of the more common of these macros and other TOPS-20 coding and naming conventions. Knowledge of these conventions will help greatly in the reading of TOPS-20 monitor listings. A listing of MACSYM will be found at the end of this module.

LEARNING OBJECTIVES

Upon completion of this module, the student will be able to:

Determine the code generated and/or the function applied by the various macros and pseudo-ops used in the TOPS-20 monitor.

MODULE OUTLINE

CODING CONVENTIONS

I. Using MACSYM
   A. Symbol Definitions
   B. Macros To Manipulate Field Masks
   C. Instructions Using Field Masks
   D. DEFSTR -- MSKSTR Data Structure Facilities
      1. LOAD
      2. STOR
      3. Examples

   E. Subroutine Conventions
   F. Named Variable Facilities
   G. Miscellaneous

II. TOPS-20 Coding Standards
   A. Subroutine Calling - JSYS
   B. Subroutine Calling
   C. AC Definitions
   D. AC Saving and Restoration
   E. Subroutine Documentation
   F. Multi-line Literals
   G. Numbers

III. Appendices

This page is for notes.

USING MACSYM

MACSYM is available on SYS: in two forms: MACSYM.UNV and MACREL.REL. The first of these is the universal file of macro and symbol definitions; the second is a file of small support routines used by certain facilities (e.g., stack variables). The universal file is normally obtained at assembly time by the source statement

```
SEARCH MACSYM
```

The object file, if necessary, may be obtained by the source statement

```
.REQUIRE SYS:MACREL
```

which instructs LINK to load the object file along with the main program. The file is loaded only once (even if the .REQUIRE appears in several source modules) and no explicit LINK command need be given.

Symbol Definitions

Conventions observed regarding the construction of symbols are as follows ("x" represents any alphanumeric character):

xxxxx. an opdef or macro definition

.xxxxx a constant value

xx%xxx a mask, i.e., a bit or bits specifying a field

Symbols containing multiple periods may be used internally by some macros.

Symbols containing "$" are not used or defined by DEC and are reserved for customer use.

The following definitions are available in MACSYM and are arranged into groups as shown.

MISCELLANEOUS CONSTANTS (SYMBOLS)

```
.INFIN = 377777,,777777 ;plus infinity

.MINFI = 400000,,0 ;minus infinity

.LHALF = 777777,,0 ;left half

.RHALF = 0,,777777 ;right half

.FWORD = 777777,,777777 ;full word
```

CONTROL CHARACTERS (SYMBOLS)

Symbols are defined for all control character codes 0-37 and 175-177. The following are the commonly used characters; see source listing for others.

```
.CHBEL = 07 ;bell

.CHBSP = 10 ;backspace

.CHTAB = 11 ;tab

.CHLFD = 12 ;linefeed

.CHFFD = 14 ;formfeed

.CHCRT = 15 ;carriage return

.CHESC = 33 ;escape

.CHDEL = 177 ;delete (rubout)
```

```
                PC FLAGS (MASK SYMBOLS)

PC%OVF = 1B0 ;overflow

PC%CY0 = 1B1 ;carry 0

PC%CY1 = 1B2 ;carry 1

PC%FOV = 1B3 ;floating overflow

PC%BIS = 1B4 ;first part done (byte increment
                suppress)

PC%USR = 1B5 ;user mode

PC%UIO = 1B6 ;user IO mode

PC%LIP = 1B7 ;last instruction public

PC%AFI = 1B9 ;ADDRESS FAILURE INHIBIT

PC%ATN = 1B10 ;apr trap number

PC%FUF = 1B11 ;floating underflow

PC%NDV = 1B12 ;no divide
```

Macros to Manipulate Field Masks

Many of the symbols in MACSYM and MONSYM define flag bits and fields. A field mask is a full-word value with a single contiguous group of 1's in the field. For example, 000000,,777000 defines a field consisting of bits 18-26. The following macros may be used in expressions to deal with these masks.

WID(MASK)

Width - computes the width of the field defined by the mask, i.e., the number of contiguous 1-bits. The value is not defined if the mask contains non-contiguous 1-bits.

POS(MASK)

Position - computes the position of the field defined by the

mask. The position of a field is always represented by the bit-number of the bit furthest to the right of the field, regardless of the field's width. This is sufficient to specify the entire field in the case of flags (1-bit fields).

POINTR(LOC,MASK)

Byte pointer - constructs a byte pointer to location LOC which references the byte defined by MASK. For example, POINTR(100,77) = POINT 6,100,35 = 000600,,100

FLD(VAL,MASK)

Field value - Places the value VAL into the field defined by MASK. For example, FLD(3,700) = 0,,000300

.RTJST(VAL,MASK)

Right-justify - Shifts VAL right so that the field defined by MASK is moved to the low-order bits of the word. For example, .RTJST(300,700) = 3

MASKB(LBIT,RBIT)

Mask - constructs a mask word which defines a field from bit LBIT to bit RBIT inclusive. E.g., MASKB(18,26) = 0,,777000.

Instructions Using Field Masks

The following mnemonics are similar to certain machine instructions used to move and test bits and fields. These macros select the most efficient instruction for the mask being used.

MOVX AC,MASK

Load AC with a constant. MASK may be any constant. This assembles one of the following instructions: MOVEI, MOVSI, HRROI, HRLOI, or MOVE literal.

TXmn AC,MASK

where m is: N, Z, O, C

n is: E, N, A, null

There are 16 definitions of this form which include all of the modification and testing combinations of the test instructions (i.e., TXNN, TXNE, TXO, TXON, etc.). A TL, TR, or TD literal is assembled as appropriate.

```
IORX AC,MASK

ANDX AC,MASK

XORX AC,MASK
```

These are equivalent to certain TX functions but are provided for mnemonic value.

```
JXm AC,MASK,ADDRESS
```

This is a set of four definitions which jump to ADDRESS if the field specified by MASK meets a certain condition. The condition (m) may be:

E - jump if all masked bits are Ø

N - jump if not all masked bits are Ø

O - jump if all masked bits are 1

F - jump if not all masked bits are 1 (false)

These macros will assemble into one, two, or three instructions as necessary to effect the specified result. E.g., JXN T1,1BØ,FOO = JUMPL T1,FOO

JXE T1,77Ø,FOO = TRNN T1,77Ø JRST FOO

DEFSTR - MSKSTR Data Structure Facilities

This set of macros provides a comprehensive facility for the definition and use of data structures. It is an extension of some of the techniques represented by the field mask facilities above. Typically, a data structure definition will include some information about the location

of the data in memory as well as its position within a word. These facilities are intended to provide the following advantages:

1. Data items may be referenced mnemonically. For example, two data items in the same word would be given different names rather than merely being known as the left half or right half of the word.

2. Should the need arise, storage formats may be changed without incurring the expense of a search of the code to change each reference.

DEFSTR -- MSKSTR

These macros both define a data structure called NAME:

```
DEFSTR NAME,LOCATION,POSITION,SIZE

MSKSTR NAME,LOCATION,MASK
```

LOCATION specifies the memory location of the desired word and consists of address, index, and indirect fields in the usual form, i.e., @address(index). Any of the fields may be omitted if not needed, and the entire location argument may be null in some circumstances. The remaining arguments define the desired field. DEFSTR specifies the field in terms of its position (right-most bit number) and size (number of bits), while MSKSTR specifies the field by a full-word mask as described earlier. Normally, the actual storage to be used is declared separately, e.g., by a BLOCK statement.

As a simple example, consider an array of full-word data items. We wish to use the name FOO for the data itself, so we declare the actual storage by some other name, e.g.,

```
FOO1:  BLOCK n
```

Then, we declare the structure by

```
DEFSTR FOO,FOO1(FOOX),35,36
```

This says that we declare a data item called FOO, that the items are addressed by FOO1(FOOX) (assuming that the index

is kept in register FOOX), that the items are 36-bit quantities with the right-most bit in bit 35 (i.e., full words). If instead, we wish to declare that each word of FOO1 consists of an item in the left half and two 9-bit items in the right half, we could write:

```
DEFSTR FIRSTD,FOO1(FOOX),17,18  ;LH item

DEFSTR SECOND,FOO1(FOOX),26,9   ;one 9-bit item

DEFSTR THIRDD,FOO1(FOOX),35,9   ;another 9-bit item
```

LOAD

Data items defined with DEFSTR or MSKSTR may be referenced in a general way. At each instance, additional location information may be given if necessary. A set of reference functions (macros) is defined for most common operations, some affecting AC and memory, others only memory. For example, the LOAD function loads a data item into an AC and is written as

```
LOAD AC,NAME,LOCATION
```

where: AC is the AC to be loaded

NAME is the structure name as defined with DEFSTR

LOC is the location specification in addition to that declared in the structure definition. This field may be null in some cases.

Taking the sample definitions above, we may write

```
LOAD T1,FOO
```

which would assemble into

```
MOVE T1,FOO1(FOOX)
```

or

```
LOAD T1,SECOND = LDB T1,[POINT 9,FOO1(FOOX),26]

LOAD T1,FIRSTD = HLRZ T1,FOO1(FOOX)
```

Note that the macro compiles the most efficient instruction available to reference the specified field.

The optional third argument is provided to allow some of the location information to be specified at each instance. For example, if the definition is

```
DEFSTR FOO,FOO1,35,36
```

The index may be specified at each instance, for example,

```
LOAD T1,FOO,(XX)

LOAD T2,FOO,(T1)
```

The specification given in the definition is concatentated with the specification given in the reference.

STOR

The following reference functions are presently defined:

```
LOAD AC,NAME,LOC  load data item into AC

STOR AC,NAME,LOC  store data item from AC into memory
```

The data item is right-justified in the AC.

```
SETZRO NAME,LOC set the data item to zero

SETONE NAME,LOC set the data item to all ones

SETCMP NAME,LOC complement the data item

INCR NAME,LOC increment the data item

DECR NAME,LOC decrement the data item
```

For functions not specifically provided, the following may be used:

```
OPSTR OP,NAME,LOC

OPSTRM OP,NAME,LOC
```

OP is any machine instruction written without an address field. It will be assembled so as to reference the specified data structure. OPSTR is used if memory is not modified; OPSTRM is used if memory is modified. For example,

```
OPSTRM <ADDM T1,>,FOO
```

to add the quantity in T1 to the data item FOO.

The following test and transfer functions are presently defined:

```
JE NAME,LOC,ADDR jump to ADDR if data is 0

JN NAME,LOC,ADDR jump to ADDR if data is not 0
```

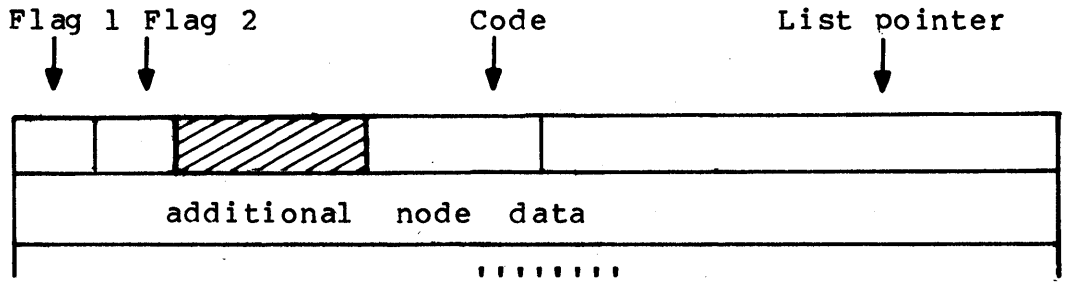
The following test and transfer functions take a list of structure names (surrounded by angle-brackets) or a single structure name. They compile code to test each data item in the order given, and will stop as soon as the result of the function is known (e.g., AND encounters a false term).

```
JOR NAMLST,LOC,ADDR    jump to ADDR if any data item is
                         true (non-0)

JAND NAMLST,LOC,ADDR   jump to ADDR if all data items are
                         true (non-0)

JNOR NAMLST,LOC,ADDR   jump to ADDR if all data items are
                         false (0)

JNAND NAMLST,LOC,ADDR  jump to ADDR if any data item is
                         false (0)
```

These functions optimize multiple fields in the same word if they are adjacent in the structure list. If the final location is an accumulator, further optimization is done.

EXAMPLES

As an example of the data structure facility, consider the typical case of data organized into unit blocks with pointers to other blocks. Such a block may appear as:



We assume that n-word blocks will be allocated from a free pool at execution time. The structure of the block is declared as follows:

```
MSKSTR FLAG1,Ø,1BØ

MSKSTR FLAG2,Ø,1B1

DEFSTR CODE,Ø,17,9

DEFSTR LINK,Ø,35,18

DEFSTR NODDAT,1,35,36
```

Note that the location field contains only the offset address of the word within the block; the address of the block will be specified in an index at each reference. References would appear as follows:

```
LOAD T1,LINK,(T1) ;step to next node in list

STOR T2,CODE,(T1) ;set new block code

JE FLAG1,(T1),FLOFF ;jump if flag1 is off

JAND <FLAG1,FLAG2>,(T1),FLGSON  ;jump if flag1 and
                                ;flag2 are both on
```

;Q-BLOCK FORMAT

ENQLJQ: BACK POINTER TO LAST Q-BLOCK FOR JOB		ENQNJQ: FORWARD POINTER TO NEXT Q-BLOCK FOR JOB
ENQLLQ: BACK POINTER TO LAST Q-BLOCK		ENQNLQ: FORWARD POINTER TO NEXT Q-BLOCK
ENQFLG: FLAGS	ENQCHN: PSI CHANNEL	ENQFRK: FORK TO INTERRUPT WHEN REQUEST IS LOCKED
ENQNR: # OF RESOURCES REQUESTED FROM POOL		ENQID: REQUEST ID CODE
ENQLRQ: BACK POINTER TO LAST Q-BLOCK OF REQUEST		ENQFQ: FORWARD POINTER TO NEXT Q-BLOCK OF REQUEST
ENQLBP: POINTER TO LOCK-BLOCK		ENQGRP: GROUP # FOR SHARABLE REQUESTS
ENQNST: NEST COUNT		ENQJFN: JFN OF REQUEST OR -1, -2, OR -3
		ENQMSK: POINTER TO THE MASK BLOCK

;LOCK-BLOCK FORMAT

ENQLHC: BACK POINTER TO LAST LOCK-BLOCK ON HASH CHAIN	ENQNHC: POINTER TO NEXT LOCK-BLOCK ON HASH CHAIN
ENQLLQ: BACK POINTER TO LAST Q-BLOCK ON QUEUE	ENQNLQ: FORWARD POINTER TO FIRST Q-BLOCK ON QUEUE
ENQFLG: FLAGS	ENQLVL: LEVEL NUMBER OF THIS LOCK
ENQTR: TOTAL # OF RESOURCES IN THIS POOL	ENQRR: REMAINING NUMBER OF RESOURCES IN THIS POOL
ENQTS: TIME STAMP TIME OF LAST REQUEST LOCKED	
ENQFBP: FREE BLOCK POINTER TO FREE Q-BLOCK	ENQLT: LONG TERM LOCK LIST FOR THIS JOB
ENQOFN: OFN, OR -2, OR -3, OR 400000 + JOB NUMBER	ENQLEN: LENGTH OF THIS LOCK-BLOCK
ENQNMS: NUMBER OF WORDS IN THE MASK BLOCK	
ENQTXT: ASCIZ STRING OR 500000 + USER CODE	

```
DEFSTR(ENQLJQ,Ø,17,18)   ;BACK POINTER TO LAST Q FOR JOB
DEFSTR(ENQNJQ,Ø,35,18)   ;FORWARD POINTER TO NEXT Q FOR JOB
                         ;ENQNJQ AND ENQLJQ MUST BE IN WORD Ø
DEFSTR(ENQLLQ,1,17,18)   ;BACK POINTER TO LAST Q IN LOCK QUEUE
DEFSTR(ENQNLQ,1,35,18)   ;FORWARD POINTER TO NEXT Q OF LOCK
DEFSTR(ENQFLG,2,11,12)   ;FLAGS OF BLOCK (EITHER LOCK OR Q)
DEFSTR(ENQCHN,2,17,6)    ;PSI CHANNEL #, -1 MEANS JOB BLOCKED
DEFSTR(ENQFRK,2,35,18)   ;FORK NUMBER OF CREATOR OF Q-BLOCK
DEFSTR(ENQNR,3,17,18)    ;# OF RESOURCES REQUESTED
DEFSTR(ENQID,3,35,18)    ;ID OF ENQ REQUEST
DEFSTR(ENQLRQ,4,17,18)   ;BACK POINTER TO REST OF REQUEST
DEFSTR(ENQFQ,4,35,18)    ;FORWARD POINTER TO REST OF REQUEST
DEFSTR(ENQLBP,5,17,18)   ;POINTER TO LOCK-BLOCK OF THIS Q
DEFSTR(ENQGRP,5,35,18)   ;GROUP NUMBER OF SHARABLE REQUEST
DEFSTR (ENQJFN,6,35,18)  ;JFN OF ENQ REQUEST
DEFSTR (ENQNST,6,17,18)  ;NEST COUNT
DEFSTR (ENQMSK,7,35,18)  ;POINTER TO MASK BLOCK

DEFSTR(ENQLHC,Ø,17,18)   ;BACK POINTER TO LAST LOCK IN HASH LIST
DEFSTR(ENQNHC,Ø,35,18)   ;FORWARD PNTR TO NEXT LOCK ON HASH LIST
                         ;ENQNHC AND ENQLHC MUST BE IN WORD Ø
DEFSTR(ENQLVL,2,35,18)   ;LEVEL NUMBER OF LOCK
DEFSTR(ENQTR,3,17,18)    ;TOTAL # OF RESOURCES IN POOL
DEFSTR(ENQRR,3,35,18)    ;# OF RESOURCES REMAINING IN POOL
DEFSTR(ENQTS,4,35,36)    ;TIME STAMP OF LAST REQUEST TO BE LOCKED
DEFSTR(ENQFBP,5,17,18)   ;POINTER TO FREE Q-BLOCK
DEFSTR(ENQLT,5,35,18)    ;LONG TERM LOCK LIST FOR THIS JOB
        .ENQLT==5        ;OFFSET OF LOCK LIST ELEMENT
DEFSTR(ENQOFN,6,17,18)   ;OFN, -2, -3, OR 4ØØØØØ+JOB NUMBER
DEFSTR(ENQLEN,6,35,18)   ;LENGTH OF LOCK-BLOCK
DEFSTR(ENQNMS,7,17,18)   ;NUMBER OF WORDS IN THE MASK BLOCK
DEFSTR(ENQTXT,1Ø,35,36)  ;FIRST WORD OF TEXT OR USER CODE
        .ENTXT==1Ø       ;INDEX INTO LOCK-BLOCK FOR TEXT BLOCK
DEFSTR(ENQOTA,ENQLST,8,9)        ;ENQ/DEQ QUOTA
DEFSTR(ENQCNT,ENQLST,17,9)       ;COUNT OF REQUESTS QUEUED UP
```

```
ENQ     MACRO %53(1042) 13:36 20-Oct-78 Page 12
ENQ     MAC     16-Nov-77 14:58


                                        ;DEQ FUNCTION 0

                                        ;ACCEPTS IN T1/  0 = INTERNAL MONITOR CALL
                                        ;               -1 = JSYS CALL  (READ ARGUMENTS FROM USER SPACE)

000274'02  265 01 0 00 000043'  DEQFN0: JSP T1,SETVAR           ;SET UP GLOBAL VARIABLES
000275'02  265 16 0 00 000047*          STKVAR <DQFN0T,DQFN0Q>
000276'02  000002  000002
000277'02  402 00 0 17 777776           SETZM DQFN0T            ;INITIALIZE ERROR COUNTER
000300'02  322 01 0 00 000305'          JUMPE T1,DQFN0D         ;IF MONITOR CALL, ARGS ARE SET UP
000301'02  260 17 0 00 000704'          CALL VALARG             ;VALIDATE THE ARGUMENT BLOCK
000302'02  263 17 0 00 000000            RET                    ;ILLEGAL ARGUMENT BLOCK
000303'02  260 17 0 00 000734'  DQFN0A: CALL VALREQ             ;VALIDATE THIS LOCK SPECIFICATION
000304'02  254 00 0 00 000340'           JRST DQFN0B            ;ERROR
000305'02  260 17 0 00 002026'  DQFN0D: CALL HASH               ;HASH THIS REQUEST
000306'02  254 00 0 00 000340'           JRST DQFN0B            ;ERROR DURING HASH
000307'02  260 17 0 00 001033'          CALL FNDLOK             ;FIND THE LOCK-BLOCK
000310'02  254 00 0 00 000340'           JRST DQFN0B            ;NO SUCH LOCK-BLOCK
000311'02  135 02 0 00 002256'          LOAD T2,ENQFLG,(T1)     ;GET FLAGS OF THE LOCK BLOCK
000312'02  603 10 0 00 040000           TXNE P1,EN%LTL          ;IS THIS A LONG TERM LOCK
000313'02  660 02 0 00 000040           TXO T2,EN.LTL           ;YES, REMEMBER THIS IN THE LOCK BLOCK
000314'02  137 02 0 00 002256'          STOR T2,ENQFLG,(T1)
000315'02  260 17 0 00 001055'          CALL FNDQ               ;FIND THE Q-BLOCK FOR THIS FORK
000316'02  254 00 0 00 000340'           JRST DQFN0B            ;COULD NOT FIND THE Q-BLOCK
000317'02  202 01 0 17 777775           MOVEM T1,DQFN0Q         ;SAVE THE Q-BLOCK ADDRESS
000320'02  554 02 0 01 000006           LOAD T2,ENQNST,(T1)     ;GET NEST COUNT
                                        JUMPG T2,[DECR ENQNST,(T1)
000321'02  327 02 0 00 002274'                  JRST DQFN0C]    ;THIS WAS A NESTED ENQ, DONT DEQ IT
000322'02  554 02 0 01 000003           LOAD T2,ENQNR,(T1)      ;GET NUMBER LOCKED IN ORIGINAL ENQ
                                        JUMPE T2,[CALL DEQMSK   ;IF 0, SEE IF DEQ'ING A MASK
                                                  JRST DQFN0E   ;NOT COMPLETELY DEQUEUED
                                                 MOVE T1,DQFN0Q ;OK TO DELETE THIS Q-BLOCK
                                                 CALL SQDEQ     ;GO DELETE THIS Q-BLOCK
000323'02  322 02 0 00 002277'                   JRST DQFN0C]   ;STEP TO NEXT REQUEST
000324'02  275 02 0 12 000000           SUBI T2,0(P3)           ;SEE IF DEQ'ING ALL RESOURCES
                                        JUMPL T2,[MOVEI T1,ENQX12
000325'02  321 02 0 00 002304'                   JRST DQFN0B]   ;DEQ'ING TOO MANY RESOURCES
                                        JUMPE T2,[CALL SQDEQ    ;DEQ'ING ALL OF THEM, DELETE Q-BLOCK
000326'02  322 02 0 00 002302'                   JRST DQFN0C]
000327'02  506 02 0 01 000003           STOR T2,ENQNR,(T1)      ;PUT BACK NEW # OF RESOURCES LOCKED
000330'02  554 01 0 01 000005           LOAD T1,ENQLBP,(T1)     ;GET ADDRESS OF LOCK BLOCK
000331'02  550 02 0 01 000003           LOAD T2,ENQRR,(T1)      ;GET # OF REMAINING RESOURCES
000332'02  271 02 0 12 000000           ADDI T2,0(P3)           ;UPDATE THE COUNT
000333'02  542 02 0 01 000003           STOR T2,ENQRR,(T1)      ;STORE NEW COUNT OF REMAINING RESOURCES
000334'02  200 01 0 17 777775   DQFN0E: MOVE T1,DQFN0Q         ;GET Q-BLOCK ADDRESS
000335'02  554 01 0 01 000005           LOAD T1,ENQLBP,(T1)     ;GET ADDRESS OF THE LOCK BLOCK
000336'02  260 17 0 00 001700'          CALL LOKSKD             ;GO SCHEDULE THIS LOCK
000337'02  254 00 0 00 000341'          JRST DQFN0C             ;DONT COUNT UP ERROR COUNTER
000340'02  202 01 0 17 777776   DQFN0B: MOVEM T1,DQFN0T         ;SAVE THIS ERROR CODE
000341'02  270 05 0 15 000003   DQFN0C: ADD Q1,EDSTP            ;STEP TO THE NEXT LOCK REQUEST
000342'02  327 05 0 00 000303'          JUMPG Q1,DQFN0A         ;LOOP BACK FOR ALL LOCKS
000343'02  337 01 0 17 777776           SKIPG T1,DQFN0T         ;ANY ERRORS SEEN?
000344'02  254 00 0 00 000171*          RETSKP                  ;NO, DEQUEUING COMPLETED
000345'02  263 17 0 00 000000           RET                     ;YES, RETURN ERROR CODE IN T1
```

Subroutine Conventions

The following definitions are used to make subroutine mechanics mnemonic. Reference is made to these conventions elsewhere in this document.

CALL address

Call subroutine at address; equivalent to PUSHJ P,address

RET

Return from subroutine; equivalent to POPJ P,

RETSKP

Return from subroutine and skip; equivalent to

```
JRST [AOS Ø(P) RET]
```

CALLRET address

Call the subroutine at address and return immediately thereafter; equivalent to

```
CALL address RET
RETSKP
```

CALLRET assembles as JRST but should be treated as if it assembles into several instructions and cannot be skipped over.

AC CONVENTIONS

The facilities described here assume (in some cases) the following AC conventions:

AC1-AC4	temporary, may be used to pass and return values
ACØ,AC5-AC15	preserved, i.e., saved and restored if used by subroutine

AC16 temporary, used as scratch by some MACSYM facilities

AC17 stack pointer

Named Variable Facilities

A traditional deficiency of machine language coding environments is a lack of facilities for named transient storage ("automatic", etc.). Sometimes, permanent storage is assigned (e.g., by BLOCK statements) when no recursion is expected. More often, ACs are used for a small number of local variables. In such a case, the previous contents must usually be saved, and a general mnemonic (e.g., T1, A, X) is usually used. In some cases, data on the stack is referenced, for example,

```
MOVE T1,-2(P)
```

But this statement is non-mnemonic and likely to fail if additional storage is added to or removed from the stack.

The facilities described here provide local named variable storage. Two of these allocate the storage on the stack; the third allocates it in the ACs.

STKVAR NAMELIST

This statement allocates space on the stack and assigns local names. The list consists of one or more symbols separated by commas. Each symbol is assigned to one stack word. If more than one word is needed for a particular variable, a size parameter may be given enclosed with the symbol in angle-brackets. For example,

```
STKVAR <AA,BB>

STKVAR <AA,<BB,3>>
```

Variables declared in this way may be referenced as ordinary memory operands, for example,

```
MOVE T1,AA

DPB T1,[POINT 6,BB,5]
```

Each variable is assembled as a negative offset from the current stackllocation, for example,

```
MOVE T1,AA = MOVE T1,-2(P)
```

Hence, no other index may be given in the address field. Indirection may be used if desired.

There is no explicit limit to the scope of the variables defined by STKVAR, but the following logical constraints must be observed:

1. The stack pointer must not be changed within the logical scope of the variables, e.g., by PUSH or PUSHJ instructions. This also implies that the variables may not be referenced within a local subroutine called from the declaring routine.

2. The declaring routine must return with a RET or RETSKP. This will cause the stack storage to be automatically deallocated.

STKVAR assumes that the stack pointer is in P, and it uses .A16 (AC16) as a temporary.

TRVAR NAMELIST

This statement allocates stack space and assigns local names. It is equivalent to STKVAR, except that it uses one additional preserved AC and eliminates some of the scope restrictions of STKVAR. In particular, it uses .FP (AC15) as a frame pointer. .FP is setup (and the previous contents saved) at the same time as the stack space is allocated. References to the variables use .FP as the index rather than P. This provides for additional storage to be allocated on the stack and allows the variables to be referenced from local subroutines. Note that all such subroutines (i.e., all variable references) must appear after the declaration

in the source. STKVAR may be used within TRVAR, e.g., by a local subroutine.

STKVAR and TRVAR declarations are normally placed at the beginning of a routine. They need not be the first statement. If a routine has two or more entry points, a single declaration may be placed in the common path, or several identical declarations may be used in each of the separate paths. Care must be taken that control passes through only one declaration before any variables are referenced. For example,

```
;MAIN ROUTINE

ENT1:   TXO F,FLAG          ;entry 1, set flag
        JRST ENT0           ;join common code

ENT2:   TXZ F,FLAG          ;entry 2, clear flag
ENT0:   TRVAR <AA,BB>       ;common code, declare locals
        ..
        CALL LSUBR          ;call local subroutine
        ..
        RET

;LOCAL SUBROUTINE

LSUBR:  STKVAR <CC>         ;local subroutine, declare
                            ; locals
        MOVE T1,AA          ;reference outer routine
                            ; variable
        MOVEM T1,CC         ;reference local variable
        ..
        RETSKP              ;skip return
```

ASUBR NAMELIST

This statement is used to declare formals for a subroutine. The namelist consists of from one to four variable names. The arguments are passed to the subroutine in ACs T1 to T4, and values may be returned in these same ACs. ASUBR causes these four ACs to be stored on the stack (regardless of how many formals are declared), and defines the variable names as their corresponding stack locations. The return does not restore T1-T4. The same frame pointer AC is used by both ASUBR and TRVAR; hence, these declarations may not be used

within the same routine. The scope rules are the same here as for TRVAR.

ACVAR NAMELIST

This statement declares local storage which is allocated from the set of preserved ACs. An optional size parameter may be given for each variable. The previous contents of the ACs are saved on the stack and automatically restored on the next return. Variables declared by ACVAR may be referenced as ordinary AC operands.

Miscellaneous

TMSG string

Type literal string; uses AC1, outputs to primary output. For example,

```
TMSG <TYPE THIS TEXT>
```

JSERR

Handle unexpected JSYS error; type "?JSYS ERROR: message". This is a single instruction subroutine call which always returns +1.

JSHLT

Handle unexpected fatal JSYS error; same as JSERR except it does a HALTF instead of returning.

MOD.(DEND,DSOR)

Modulo - In assembly-time expression; this gives the remainder of DEND divided by DSOR; e.g., MOD. 10,3 = 1.

TOPS-20 CODING STANDARDS

Subroutine Calling - JSYS

Monitor-call JSYSs may be used in user or monitor code. All ACs are preserved over a JSYS call unless an explicit statement to the contrary appears in the JSYS description. ACs are changed over a JSYS call only when values are to be returned to the caller.

The JSYS name appears as the opcode in the statement performing the call. The JSYS mnemonic includes the instruction field, so no other fields are supplied by the user.

Unimplemented JSYSs will invoke the illegal instruction sequence (with error code ILINS2). Defined and implemented JSYSs will return to caller +1 upon success, or will invoke the illegal instruction sequence upon failure. The illegal instruction sequence recognizes an ERJMP or ERCAL following the failing JSYS and causes the appropriate action. If that instruction is not an ERJMP or ERCAL, an illegal instruction interrupt is requested which will be handled by the executing fork if enabled, or otherwise, it forces fork termination. See paragraph below on JSYS returns for proper indication of JSYS failure.

All constant values, bits, and fields of JSYS arguments will have mnemonics defined according to the rules in MONSYM. The JSYS code itself uses these symbols for loading arguments, testing bits, etc.

When writing code to implement a JSYS, the following conventions are observed:

1. The entry point of the JSYS is defined by a global tag consisting of a DOT concatenated with the symbolic name of the JSYS; e.g., .GTJFN::.

2. The first statement of the JSYS code is MCENT (Monitor Context ENTry). This establishes the normal JSYS context for a "slow" JSYS. At this writing, MCENT is a null macro and the JSYS entry procedure is invoked automatically. The use of

MCENT is required so that this implementation may be changed in the future if necessary.

3. All caller ACs are automatically preserved by the entry and exit procedures. Therefore, JSYS routines are specifically required NOT to save and restore the ACs. The contents of the caller's ACs 1-4 are copied into the callee's ACs. However, no callee ACs are copied back to the caller's AC block on return; one of the "previous context" instructions (i.e., UMOVE, UMOVEM, XCTU [instruction], etc.) must be used to return any values to the caller. For example,

```
UMOVEM T1,T1        ;store monitor T1 into user T1
```

A "previous context" instruction may also be used at any time to fetch the original contents of the caller's ACs unless they have been explicitly changed by a previous context store operation. For example,

```
UMOVE T2,T1         ;load user T1 into monitor T2
```

4. Return from JSYS code should be effected by the statement:

```
MRETNG              ;Monitor RETurn Good
```

This transfers to the JSYS exit sequence (returning caller +1) and should be used to indicate successful completion of the JSYS. If the JSYS could not be completed successfully, the following statement should be used:

```
ITERR errcod        ;causes an Instruction Trap
                    ;ERRor, leaves
                    ;the error code in LSTERR
```

Other statements are defined which effect JSYS returns according to a previous convention. They are:

```
RETERR errcod     ;RETurn ERRor, return
                  ;caller +1 with error code
                  ;left in AC1 and LSTERR

EMRETN errcod     ;Error Monitor RETurn, return
                  ;caller +1 with error code left
                  ;in LSTERR
```

These should not be used in new JSYS code but may be needed if existing JSYSs are modified.

All error returns include an error code (mnemonic) that will be defined in MONSYM.MAC. If the appropriate error code has already been loaded into AC1, the errcod field may be omitted from the above with the contents of AC1 taken as the error code. No JSYS shall return other than +1 or instruction trap; therefore, no occurrence of AOS 0(P) should ever be required in JSYS code.

When invoking a JSYS error return, it is not necessary to "pop" temporary quantities from the stack. The successful return, however, should be given only when the stack is properly cleared.

Subroutine Calling

The allocation of ACs for all inter- and intra-module subroutine calls are:

ACs 1,2,3,4 -- General temporary, may be destroyed by a subroutine.

ACs 0, 5-15 -- Preserved, not changed by a subroutine (or saved and restored, if necessary).

AC 16 -- Temporary, used by the JSYS call/return procedure and reserved for use by other call/return procedures.

AC 17 -- Global stack pointer

Call and return are effected by PUSHJ P, and POPJ P, respectively. A set of assembler mnemonics has been defined for subroutine mechanics as follows:

'CALL' (= PUSHJ P,) is used to call subroutines, e.g., CALL SUBR.

'RET' (= POPJ P,) is used to return +1 from subroutines.

'RETSKP' is used to return +2 from subroutines. RETSKP is equivalent to:

```
JRST [  AOS 0(P)
        RET]
```

'RETBAD errcod' is used to return +1 with an error code from a subroutine. The error code field is optional as with JSYS error returns above. RETBAD is equivalent to:

```
JRST [  MOVEI A,ERRCOD
        RET]
```

'CALLRET' may be used to call a subroutine and return immediately thereafter. It is an abbreviation for

```
        CALL SUBR
        RET
```

or

```
        CALL SUBR
         RET
        RETSKP
```

Note that CALLRET is not guaranteed to be a single instruction; therefore, it may not be skipped over. The other returns above are guaranteed to be single instructions.

These mnemonics are used to emphasize the FUNCTION being performed (calling, returning) rather than the mechanics of the function (pushing, jumping, etc.). Also, these mnemonics could still be used even if a more general calling standard were adopted at some time in the future.

Return may also be effected by transferring control to the global tag R or RSKP, for example,

```
JUMPE A,R          ;equivalent to JUMPE A,[RET]

JUMPN A,RSKP       ;equivalent to JUMPN A,[RETSKP]
```

The general temporaries are used for passing arguments to subroutines and returning values. AC1 is used for a single argument routine, ACs 1 and 2 for a two-argument routine, etc.

A routine defined to return caller +2 (skip) upon success and caller +1 (noskip) upon failure is acceptable. Returns greater than caller +2 are not permitted.

AC Definitions

The following mnemonics have been chosen to be consistent with the AC use conventions above. The preserved ACs are divided into three groups: F (1 AC) intended for Flags, and Q1-Q3 and P1-P6 for general use. The ACs within each group are consecutive.

0 - F	10 - P1
1 - T1	11 - P2
2 - T2	12 - P3
3 - T3	13 - P4
4 - T4	14 - P5
5 - Q1	15 - P6
6 - Q2	16 - CX
7 - Q3	17 - P

The programmer should assume that each group (Tn, Qn, Pn,) is in ascending order (e.g., that T2=T1+1) though the specific assignment of numbers may change. Explicit numeric offsets from AC symbols (e.g., T1+1) should NEVER be used. Instructions using more than one AC (e.g., DIV, JFFO) must be given an AC operand so that the other AC(s) implicitly affected are in the same group. For example, T3 (and T4) is OK for IDIV because T3+1=T4, but Q3 is not because Q3+1=??.

AC Saving and Restoration

Several facilities in the monitor save and automatically restore ACs. Each of these will save all of the indicated ACs on the stack at the point of execution and will place a dummy return on the stack which causes these ACs to be restored automatically when the current routine returns. Use of these facilities eliminates the need for matching PUSH/POP pairs at the entry at exits of routines and eliminates the bugs which often arise from an unmatched PUSH or POP. The available macros are:

SAVEQ - saves ACs Q1-Q3

SAVEP - saves ACs P1-P6

SAVEPQ - saves ACs Q1-Q3 and P1-P6

SAVET - saves ACs T1-T4

Defining a different mnemonic for a preserved AC may be of value when the AC is used for a specific function by a large body of code. However, this may cause confusion because two different symbols may refer to the same AC without the knowledge of the programmer. In smaller programs, use of certain ACs can be restricted to specific functions, and a global definition is appropriate. However, a timesharing monitor is too large to accommodate all of the possible dedicated ACs.

Therefore, when a specific function-oriented AC definition is made, it shall be explicitly decided which modules shall use the definition. Within these modules, the usual name for the AC must be purged so that there is no possibility of using two different symbols for the same AC.

Only preserved ACs may be used for special definitions. Parameters to subroutines may be passed in functionally defined ACs in the following cases:

1. On an intra-module call where the contents of the AC is appropriate to its function definition.

2. On an inter-module call where the same definition exists in both modules and the AC is being used for its intended function.

A parameter may NOT be passed in a preserved AC unless both caller and callee know it by the same name, and that name must be a specific one related to the function the AC is performing.

The procedure for declaring a functionally defined AC is:

```
DEFAC NEWAC,OLDAC
```

This must be done at the beginning of an assembly; it defines NEWAC to be equal to OLDAC. OLDAC must be the mnemonic for one of the regular preserved ACs; this mnemonic will be purged and therefore unavailable for use in the current assembly.

An AC with a special definition should not be used for other purposes; for example, "JFN" should not be used to hold some quantity other than a JFN merely because it happens to be available.

Subroutine Documentation

The following is a suggested format for documenting the calling sequence of a JSYS or subroutine. A description of this sort should appear at the beginning of every subroutine, no matter how short it may be.

```
;name of subroutine - function of subroutine, etc.
; T1/ description of first argument
; T2/ description of second argument
;   ...
;       CALL NAME or JSYSNAME
; RETURN +1: conditions giving this return,
;  T1/ value(s) returned
; RETURN +2: conditions and values as above.
```

1. The arguments, if any, should be documented as the contents of registers and/or variables as shown. MONSYM mnemonics should be used when available (for example, at JSYS entry points).

2. The actual instruction to do the call should be shown. It will be "CALL subname" in the case of internal subroutines, and the single-word JSYS name in the case of a JSYS entry point.

3. The return(s) should be noted as shown; "ALWAYS" or "NEVER" may be used as the condition where appropriate; the +2 return need not be shown if it does not exist; values returned should be described in the same form as arguments.

Examples:

```
;SIN - COMPUTES SINE OF AN ANGLE
; T1/ ANGLE IN RADIANS, FLOATING POINT
;       CALL SIN
; RETURN +1: FAILURE, UNNORMALIZED NUMBER OR OUT OF
RANGE
; RETURN +2: SUCCESS, T1/ VALUE, FLOATING POINT

SIN::   ..

          - - - - - - - -

;GJINF - GET JOB INFORMATION JSYS
;       GJINF
; RETURN +1: ALWAYS,
;  T1/ LOGGED-IN DIRECTORY NUMBER
;  T2/ CONNECTED DIRECTORY NUMBER
;  T3/ JOB NUMBER
;  T4/ TERMINAL NUMBER OR -1 IF DETACHED

.GJINF:: ..
```

Multi-Line Literals

The use of multi-line literals is encouraged as a technique for making code more readable and easier to follow. The following additional rules apply:

1. The opening bracket for a multi-line literal should occur in the position the first character of the address field would have occupied if the instruction had an ordinary address. For example,

```
SKIPGE FOO
JRST [
```

2. The first and all following instructions within the literal shall begin at the second tabstop. For example,

```
JRST [  MOVE A,MUMBLE   ;COMMENT
        JRST FIE]       ;COMMENT
```

   The tab between the open bracket and the first opcode may be omitted if the line position is already at or beyond the second tab stop. For example,

```
JUMPGE A,[MOVE A,MUMBLE
```

3. The closing bracket shall follow the last field of the last instruction (as above), and shall be before the comment on the same line.

4. Nesting of multi-line literals to a depth greater than one is discouraged because of awkward formatting problems.

5. Tags may not appear in multi-line literals.

6. There are no hard and fast rules concerning when to use or not use multi-line literals. However, a literal longer than about 1Ø lines is suspect.

7. Use of ".+1" is legal in a literal to return to the main sequence.

FLOW OF CONTROL - BRANCH CONVENTIONS

Jumps, where possible, should be used to tags forward in the code (except in the case of loops). The tops of a loop should be identified by a comment.

The expressions ".+1" and ".-1" are the only legal uses of "." (this location). All other potential uses should be avoided in favor of an explicitly defined tag.

"Global" jumps should be avoided altogether. Higher-level languages do not permit them, and with good reason. The only exceptions are jumps to well-defined and published exit sequences, for example, R, RSKP (see subroutine conventions, above).

Numbers

In general, there should be no occasion to use a literal number in in-line code. All parameters, bit definitions, CONO/CONI codes, etc. should be defined mnemonically at appropriate places. It is much easier to err by not using mnemonics enough rather than in using them too much; therefore, when in the slightest doubt, define a mnemonic.

APPENDIX A

LIVING IN AN IMPERFECT WORLD

Much of the present TOPS-20 code does not conform to this standard since it was written before the standard's existence. Although a great deal of systematic editing has already been done to improve the code, obvious irregularities exist. In general, new code should conform exactly to this standard even if it is being integrated with old code. The following are some specific problems that may arise, with recommended solutions:

1. AC Mnemonics

Some code uses absolute numeric ACs. If new code is being integrated into a sequence that uses numeric ACs, editing the existing code to use the standard mnemonics is desirable, particularly for the preserved ACs. If the programmer cannot take the time to do this, the mnemonics T1-T4 should be used for ACs 1-4; other ACs should be referenced in the same way as is done by the existing code.

Some code uses mnemonics A,B,C,D for the temporary ACs. These mnemonics should be used for new code being integrated into such code, or all references can be edited to use the standard mnemonics.

You may write some code using the standard mnemonics for preserved ACs and then discover that the module into which you wish to put this code has redefined some of these ACs. The solution is one or a combination of the following:

1. Move the new code to a module which does not redefine the preserved ACs.

2. Use different preserved ACs -- ones which have not been redefined. (Note it is not acceptable to use an AC with a special definition for other than its special purpose.)

Clearly, code needing some of the special definitions must be placed in a module which has these ACs defined and must therefore use only the other preserved ACs.

Note that a value which usually resides in a special AC need not ALWAYS reside there. For example, if code in JSYSF needs to call a routine in PAGEM and pass a JFN index as an argument, the JFN should be loaded into T1-T4 for the call since PAGEM does not have JFN defined and cannot accept an argument in it.

2. Stack Handling

Use of the several stack variable facilities defined in MACSYM is recommended. However, some old code uses explicit PUSH and POP and references of the form -n(P). When notable modifications must be made to such code, edit it to use STKVAR or TRVAR.

MODULE TEST

For this lab, you are to use the coding conventions and macros covered in the Coding Conventions module.

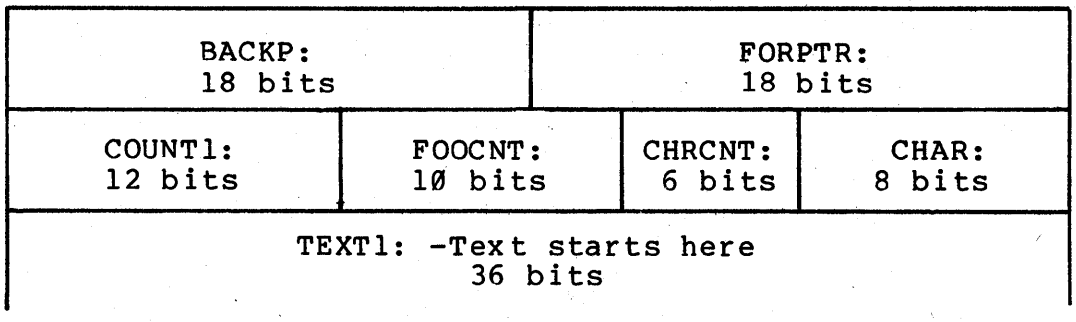
The exercises marked with a double star (**) are optional.

Using MACSYM

Using any editor with which you are familiar, demonstrate, by writing as though for a program, the call required to assure that the definitions in MACSYM are available.

Data Structure Facilities

With the sample data structure given below, use the data structure macros (e.g., DEFSTR - MSKSTR etc., and the field mask definitions) to define names for each of the specified fields.



LOAD - STOR

Assuming the above macros, write the code necessary to:

1. Get FOOCNT into AC1 (using the standard name of AC1).

2. Jump to location WEX if CHRCNT is not Ø.

3. Subtract 1 from CHRCNT.

   ** Assume the location of the above table was contained in T2:

4. What would the answers to the above three questions be? **

Using The Coding Conventions

Using the conventions and standards of the Coding Conventions module, write and compile a subroutine (assume it refers to the data structure given above and that it is called with CALL) with documentation, which will:

1. Define TEM1 and TEM2 as stack variables.

2. Save the temporary storage ACs.

3. Put CHAR into TEM1.

4. Add 167 (octal) to BACKP.

5. Jump to location PMW (in the subroutine) if FORPTR is not Ø.

6. If FORPTR is not Ø (at PMW) add 4 to it and give return 2.

7. Otherwise, give return 1.

TEST EVALUATION SHEET

Results of the laboratory exercises will be discussed in class.

This page is for notes.

```
MACSYM  COMMON MACROS AND SYMBOLS       MACRO %53A(1072) 13:55 29-Dec-78 Page 1
MACSYM  MAC      8-Nov-77 10:47

                                        ;<3-UTILITIES>MACSYM.MAC.6,  8-Nov-77 10:47:32, EDIT BY KIRSCHEN
                                        ;MORE COPYRIGHT UPDATING...
                                        ;<3-UTILITIES>MACSYM.MAC.5, 26-Oct-77 11:06:30, EDIT BY KIRSCHEN
                                        ;UPDATE COPYRIGHT FOR RELEASE 3
                                        ;<3-UTILITIES>MACSYM.MAC.4, 21-Sep-77 15:49:41, EDIT BY OSMAN
                                        ;MOVE "PURGE" TO AFTER DEFINITION OF .RLEND
                                        ;<3-UTILITIES>MACSYM.MAC.3, 21-Sep-77 15:35:48, EDIT BY OSMAN
                                        ;ADD .RLEND
                                        ;<3-UTILITIES>MACSYM.MAC.2, 22-Jun-77 15:40:57, EDIT BY MURPHY
                                        ;ADDED SETMI (XMOVEI) TO SAVEAC
                                        ;<2-UTILITIES>MACSYM.MAC.7, 27-Dec-76 17:06:19, EDIT BY HURLEY
                                        ;<2-UTILITIES>MACSYM.MAC.6, 11-Oct-76 13:01:04, EDIT BY MURPHY
                                        ;<2-UTILITIES>MACSYM.MAC.5,  6-Oct-76 11:45:47, EDIT BY MURPHY
                                        ;<2-UTILITIES>MACSYM.MAC.4,  6-Oct-76 10:41:20, EDIT BY MILLER
                                        ;<2-UTILITIES>MACSYM.MAC.3,  6-Oct-76 10:30:31, EDIT BY MILLER
                                        ;CHECK FOR ALREADY DEFINED STKVAR'S AND TRVAR'S
                                        ;<2-UTILITIES>MACSYM.MAC.2, 15-Sep-76 14:21:57, EDIT BY MURPHY
                                        ;ADDED FMSG, PERSTR, SAVEAC
                                        ;<1A-UTILITIES>MACSYM.MAC.54, 10-MAY-76 14:01:20, EDIT BY HURLEY
                                        ;<1A-UTILITIES>MACSYM.MAC.50,  8-APR-76 11:16:25, EDIT BY HURLEY
                                        ;<1A-UTILITIES>MACSYM.MAC.49,  8-APR-76 11:11:35, EDIT BY HURLEY
                                        ;TCO 1244 - ADD .DIRECT .XTABM FOR MACRO 50 ASSEMBLIES



                                        ;THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY ONLY BE USED
                                        ;  OR COPIED IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE.
                                        ;
                                        ;COPYRIGHT (C) 1976, 1977, 1978 BY DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASS.

                                        ;VERSION 1

                                        IFNDEF REL,<REL==0>             ;UNIVERSAL UNLESS OTHERWISE DECLARED
                                           IFE REL,<
                                                UNIVERSAL MACSYM        COMMON MACROS AND SYMBOLS
                                           >
                                           IFN REL,<
                                                TITLE MACREL            SUPPORT CODE FOR MACSYM
                                                SEARCH MONSYM
                                                SALL
                                                IFNDEF .PSECT,<
                                                .DIRECT .XTABM>
                                           >

                                        ;THE STANDARD VERSION WORD CONSTRUCTION
                                        ; VERS - PROGRAM VERSION NUMBER
                                        ; VUPDAT - PROGRAM UPDATE NUMBER (1=A, 2=B ...)
                                        ; VEDIT - PROGRAM EDIT NUMBER
                                        ; VCUST - CUSTOMER EDIT CODE (0=DEC DEVELOPMENT, 1=DEC SWS, 2-7 CUST)

                                        DEFINE PGVER.   (VERS,VUPDAT,VEDIT,VCUST)<
                                                ..PGV0==.               ;;SAVE CURRECT LOCATION AND MODE
                                                .JBVER=:137             ;;WHERE TO PUT VERSION
                                                LOC     .JBVER          ;;PUT VERSION IN STANDARD PLACE
```

```
MACSYM  COMMON MACROS AND SYMBOLS       MACRO %53A(1072) 13:55 29-Dec-78 Page 1-1
MACSYM  MAC      8-Nov-77 10:47

                                                BYTE    (3)VCUST(9)VERS(6)VUPDAT(18)VEDIT
                                                .ORG    ..PGV0          ;;RESTORE LOCATION AND MODE
                                        >

                                        ;MASKS FOR THE ABOVE

          700000  000000                VI%WHO==:7B2                    ;Customer edit code
          077700  000000                VI%MAJ==:777B11                 ;Major version number
          000077  000000                VI%MIN==:77B17                  ;Minor version/update
                  777777                VI%EDN==:777777B35              ;Edit number
                                        ;ADDED VI%XXX
```

```
MACSYM  COMMON MACROS AND SYMBOLS       MACRO %53A(1072) 13:55 29-Dec-78 Page 2
MACSYM  MAC      8-Nov-77 10:47         MISC CONSTANTS

                                                SUBTTL MISC CONSTANTS

                                        ;MISC CONSTANTS

                                        .INFIN==:377777,,777777         ;PLUS INFINITY
                                        .MINFI==:1B0                    ;MINUS INFINITY
                                        .LHALF==:777777B17              ;LEFT HALF
                                        .RHALF==:777777                 ;RIGHT HALF
                                        .FWORD==:-1                     ;FULL WORD
```

```
MACSYM  COMMON MACROS AND SYMBOLS       MACRO %53A(1072) 13:55 29-Dec-78 Page 3
MACSYM  MAC     8-Nov-77 10:47          MISC CONSTANTS


                                        SUBTTL  SYMBOLS FOR THE CONTROL CHARACTERS

                000000          .CHNUL==:000                    ;NULL
                000001          .CHCNA==:001
                000002          .CHCNB==:002
                000003          .CHCNC==:003
                000004          .CHCND==:004
                000005          .CHCNE==:005
                000006          .CHCNF==:006
                000007          .CHBEL==:007                    ;BELL
                000010          .CHBSP==:010                    ;BACKSPACE
                000011          .CHTAB==:011                    ;TAB
                000012          .CHLFD==:012                    ;LINE-FEED
                000013          .CHVTB==:013                    ;VERTICAL TAB
                000014          .CHFFD==:014                    ;FORM FEED
                000015          .CHCRT==:015                    ;CARRIAGE RETURN
                000016          .CHCNN==:016
                000017          .CHCNO==:017
                000020          .CHCNP==:020
                000021          .CHCNQ==:021
                000022          .CHCNR==:022
                000023          .CHCNS==:023
                000024          .CHCNT==:024
                000025          .CHCNU==:025
                000026          .CHCNV==:026
                000027          .CHCNW==:027
                000030          .CHCNX==:030
                000031          .CHCNY==:031
                000032          .CHCNZ==:032
                000033          .CHESC==:033                    ;ESCAPE
                000034          .CHCBS==:034                    ;CONTROL BACK SLASH
                000035          .CHCRB==:035                    ;CONTROL RIGHT BRACKET
                000036          .CHCCF==:036                    ;CONTROL CIRCONFLEX
                000037          .CHCUN==:037                    ;CONTROL UNDERLINE

                000175          .CHALT==:175                    ;OLD ALTMODE
                000176          .CHAL2==:176                    ;ALTERNATE OLD ALTMODE
                000177          .CHDEL==:177                    ;DELETE
```

```
MACSYM  COMMON MACROS AND SYMBOLS       MACRO %53A(1072) 13:55.29-Dec-78 Page 4
MACSYM  MAC      8-Nov-77 10:47         SYMBOLS FOR THE CONTROL CHARACTERS


                                               SUBTTL  HARDWARE BITS OF INTEREST TO USERS

                                       ;PC FLAGS

          400000  000000               PC%OVF==:1B0                    ;OVERFLOW
          200000  000000               PC%CY0==:1B1                    ;CARRY 0
          100000  000000               PC%CY1==:1B2                    ;CARRY 1
          040000  000000               PC%FOV==:1B3                    ;FLOATING OVERFLOW
          020000  000000               PC%BIS==:1B4                    ;BYTE INCREMENT SUPPRESSION
          010000  000000               PC%USR==:1B5                    ;USER MODE
          004000  000000               PC%UIO==:1B6                    ;USER IOT MODE
          002000  000000               PC%LIP==:1B7                    ;LAST INSTRUCTION PUBLIC
          001000  000000               PC%AFI==:1B8                    ;ADDRESS FAILURE INHIBIT
          000600  000000               PC%ATN==:3B10                   ;APR TRAP NUMBER
          000100  000000               PC%FUF==:1B11                   ;FLOATING UNDERFLOW
          000040  000000               PC%NDV==:1B12                   ;NO DIVIDE
```

```
MACSYM  COMMON MACROS AND SYMBOLS       MACRO %53A(1072) 13:55 29-Dec-78 Page 5
MACSYM  MAC      8-Nov-77 10:47         HARDWARE BITS OF INTEREST TO USERS


                        SUBTTL  MACROS FOR FIELD MASKS

                ;STANDARD MACROS

                ;MACROS TO HANDLE FIELD MASKS

                ;COMPUTE LENGTH OF MASK, I.E. LENGTH OF LEFTMOST STRING OF ONES
                ;REMEMBER THAT ^L DOES 'JFFO', I.E. HAS VALUE OF FIRST ONE BIT IN WORD

                ;COMPUTE WIDTH OF MASK, I.E. LENGTH OF LEFTMOST STRING OF ONES

                DEFINE WID(MASK)<<^L<-<<MASK>_<^L<MASK>>>-1>>>

                ;COMPUTE POSITION OF MASK, I.E. BIT POSITION OF RIGHTMOST ONE IN MASK

                DEFINE POS(MASK)<<^L<<MASK>&<-<MASK>>>>>

                ;CONSTRUCT BYTE POINTER TO MASK

                DEFINE POINTR(LOC,MASK)<<POINT WID(MASK),LOC,POS(MASK)>>

                ;PUT RIGHT-JUSTIFIED VALUE INTO FIELD SPECIFIED BY MASK

                DEFINE FLD(VAL,MSK)<<VAL>B<POS(MSK)>> .

                ;MAKE VALUE BE RIGHT JUSTIFIED IN WORD.

                DEFINE .RTJST(VAL,MSK)<<VAL>B<^D70-POS(MSK)>>

                ;CONSTRUCT MASK FROM BIT AA TO BIT BB. I.E. MASKB 0,8 = 777B8

                DEFINE MASKB (AA,BB)<1B<<AA>-1>-1B<BB>>

                ;MODULE - GIVES REMAINDER OF DEND DIVIDED BY DSOR

                DEFINE MOD. (DEND,DSOR)<<DEND-<DEND/DSOR>*DSOR>>
```

```
MACSYM  COMMON MACROS AND SYMBOLS       MACRO %53A(1072) 13:55 29-Dec-78 Page 6
MACSYM  MAC      8-Nov-77 10:47         MACROS FOR FIELD MASKS

                                        SUBTTL MOVX

                                ;MOVX - LOAD AC WITH CONSTANT

                                DEFINE MOVX (AC,MSK)<
                                   ..MX1==MSK                   ;;EVAL EXPRESSION IF ANY
                                IFDEF .PSECT,<
                                   .IFN ..MX1,ABSOLUTE,<
                                        MOVE AC,[MSK]>
                                   .IF ..MX1,ABSOLUTE,<
                                        ..MX2==0                ;;FLAG SAYS HAVEN'T DONE IT YET
                                        IFE <..MX1>B53,<
                                          ..MX2==1
                                          MOVEI AC,..MX1>       ;;LH 0, DO AS RH
                                        IFE ..MX2,<             ;;IF HAVEN'T DONE IT YET,
                                        IFE <..MX1>B17,<
                                          ..MX2==1
                                          MOVSI AC,(..MX1)>>    ;;RH 0, DO AS LH
                                        IFE ..MX2,<             ;;IF HAVEN'T DONE IT YET,
                                        IFE <<..MX1>B53-^O777777>,<
                                          ..MX2==1
                                          HRROI AC,<..MX1>>>    ;;LH -1
                                        IFE ..MX2,<             ;;IF HAVEN'T DONE IT YET,
                                        IFE <<..MX1>B17-^O777777B17>,<
                                          ..MX2==1
                                          HRLOI AC,(..MX1-^O777777)>> ;;RH -1
                                        IFE ..MX2,<             ;;IF STILL HAVEN'T DONE IT,
                                          MOVE AC,[..MX1]>      ;;GIVE UP AND USE LITERAL
                                        >>

                                IFNDEF .PSECT,<
                                        ..MX2==0                ;;FLAG SAYS HAVEN'T DONE IT YET
                                        IFE <..MX1>B53,<
                                          ..MX2==1
                                          MOVEI AC,..MX1>       ;;LH 0, DO AS RH
                                        IFE ..MX2,<             ;;IF HAVEN'T DONE IT YET,
                                        IFE <..MX1>B17,<
                                          ..MX2==1
                                          MOVSI AC,(..MX1)>>    ;;RH 0, DO AS LH
                                        IFE ..MX2,<             ;;IF HAVEN'T DONE IT YET,
                                        IFE <<..MX1>B53-^O777777>,<
                                          ..MX2==1
                                          HRROI AC,<..MX1>>>    ;;LH -1
                                        IFE ..MX2,<             ;;IF HAVEN'T DONE IT YET,
                                        IFE <<..MX1>B17-^O777777B17>,<
                                          ..MX2==1
                                          HRLOI AC,(..MX1-^O777777)>> ;;RH -1
                                        IFE ..MX2,<             ;;IF STILL HAVEN'T DONE IT,
                                          MOVE AC,[..MX1]>      ;;GIVE UP AND USE LITERAL
                                >
                                        PURGE ..MX1,..MX2>
```

```
MACSYM  COMMON MACROS AND SYMBOLS       MACRO %53A(1072) 13:55 29-Dec-78 Page 7
MACSYM  MAC      8-Nov-77 10:47         MOVX

   221                                  ;VARIENT MNEMONICS FOR TX DEFINITIONS
   222
   223                                  DEFINE IORX (AC,MSK)<
   224                                          TXO AC,<MSK>>
   225
   226                                  DEFINE ANDX (AC,MSK)<
   227                                          TXZ AC,<^-<MSK>>>
   228
   229                                  DEFINE XORX (AC,MSK)<
   230                                          TXC AC,<MSK>>
```

```
MACSYM  COMMON MACROS AND SYMBOLS       MACRO %53A(1072) 13:55 29-Dec-78 Page 8
MACSYM  MAC      8-Nov-77 10:47         MOVX


                        SUBTTL TX -- TEST MASK

;CREATE THE TX MACRO DEFINITIONS

;THIS DOUBLE IRP CAUSES ALL COMBINATIONS OF MODIFICATION AND TESTING
;TO BE DEFINED

DEFINE ..DOTX (M,T)<
        IRP M,<
        IRP T,<
          DEFINE TX'M'T (AC,MSK)<
                ..TX(M'T,AC,<MSK>)>>>>

        ..DOTX (<N,O,Z,C>,<,E,N,A>)^ ;DO ALL DEFINITIONS
        PURGE ..DOTX

;..TX
;ALL TX MACROS JUST CALL ..TX WHICH DOES ALL THE WORK

DEFINE ..TX(MT,AC,MSK)<
   ..TX1==MSK                   ;;EVAL EXPRESSION IF ANY
IFDEF .PSECT,<
   .IFN ..TX1,ABSOLUTE,<
        TD'MT AC,[MSK]>
   .IF ..TX1,ABSOLUTE,<         ;;MASK MUST BE TESTABLE
        ..TX2==0                ;;FLAG SAYS HAVEN'T DONE IT YET
        IFE <..TX1&^O777777B17>,<
          ..TX2==1              ;;LH 0, DO AS RH
          TR'MT AC,..TX1>
        IFE ..TX2,<             ;;IF HAVEN'T DONE IT YET,
        IFE <..TX1&^O777777>,<
          ..TX2==1              ;;RH 0, DO AS LH
          TL'MT AC,(..TX1)>>
        IFE ..TX2,<             ;;IF HAVEN'T DONE IT YET,
          IFE <<..TX1>B53-^O777777>,< ;;IF LH ALL ONES,
            ..TX3 (MT,AC)>>     ;;TRY Z,O,C SPECIAL CASES
        IFE ..TX2,<             ;;IF STILL HAVEN'T DONE IT,
          TD'MT AC,[..TX1]>     ;;MUST GIVE UP AND USE LITERAL
        PURGE ..TX1,..TX2>>
IFNDEF .PSECT,<
        ..TX2==0                ;;FLAG SAYS HAVEN'T DONE IT YET
        IFE <..TX1&^O777777B17>,<
          ..TX2==1              ;;LH 0, DO AS RH
          TR'MT AC,..TX1>
        IFE ..TX2,<             ;;IF HAVEN'T DONE IT YET,
        IFE <..TX1&^O777777>,<
          ..TX2==1              ;;RH 0, DO AS LH
          TL'MT AC,(..TX1)>>
        IFE ..TX2,<             ;;IF HAVEN'T DONE IT YET,
          IFE <<..TX1>B53-^O777777>,< ;;IF LH ALL ONES,
            ..TX3 (MT,AC)>>     ;;TRY Z,O,C SPECIAL CASES
        IFE ..TX2,<             ;;IF STILL HAVEN'T DONE IT,
          TD'MT AC,[..TX1]>     ;;MUST GIVE UP AND USE LITERAL
        PURGE ..TX1,..TX2>>
```

```
MACSYM  COMMON MACROS AND SYMBOLS       MACRO %53A(1072) 13:55 29-Dec-78 Page 9
MACSYM  MAC      8-Nov-77 10:47         TX -- TEST MASK

                                        ;SPECIAL CASE FOR LH ALL ONES

                                        DEFINE ..TX3 (MT,AC)<
                                                IFIDN <MT><Z>,<         ;;IF ZEROING WANTED
                                                  ..TX2==1
                                                  ANDI AC,^-..TX1>      ;;CAN DO IT WITH ANDI
                                                IFIDN <MT><O>,<         ;;IF SET TO ONES WANTED
                                                  ..TX2==1
                                                  ORCMI AC,^-..TX1>     ;;CAN DO IT WITH IORCM
                                                IFIDN <MT><C>,<         ;;IF COMPLEMENT WANTED
                                                  ..TX2==1
                                                  EQVI AC,^-..TX1>>     ;;CAN DO IT WITH EQV
```

```
MACSYM  COMMON MACROS AND SYMBOLS       MACRO %53A(1072) 13:55 29-Dec-78 Page 10
MACSYM  MAC      8-Nov-77 10:47         TX -- TEST MASK


        SUBTTL JX -- JUMP ON MASK

;JXE -- JUMP IF MASKED BITS ARE EQUAL TO 0
;JXN -- JUMP IF MASKED BITS ARE NOT EQUAL TO 0
;JXO -- JUMP IF MASKED BITS ARE ALL ONES
;JXF -- JUMP IF MASKED BITS ARE NOT ALL ONES (FALSE)

DEFINE JXE (AC,MSK,BA)<
        ..JX1==MSK              ;;EVAL EXPRESSION IF ANY
IFDEF .PSECT,<
        .IFN ..JX1,ABSOLUTE,<PRINTX MSK NOT ABSOLUTE
                ..JX1==0>
        .IF ..JX1,ABSOLUTE,<
        .IF0 <<..JX1>-1B0>,<    ;;IF MASK IS JUST B0,
                JUMPGE AC,BA>,<
        .IF0 <<..JX1>+1>,<      ;;IF MASK IF FULL WORD,
          JUMPE AC,BA>,<        ;;USE GIVEN CONDITION
                TXNN (AC,..JX1)
                JRST BA>>>
        PURGE ..JX1>
IFNDEF .PSECT,<
        .IF0 <<..JX1>-1B0>,<    ;;IF MASK IS JUST B0,
                JUMPGE AC,BA>,<
        .IF0 <<..JX1>+1>,<      ;;IF MASK IF FULL WORD,
          JUMPE AC,BA>,<        ;;USE GIVEN CONDITION
                TXNN (AC,..JX1)
                JRST BA>>>
        PURGE ..JX1>

DEFINE JXN (AC,MSK,BA)<
        ..JX1==MSK              ;;EVAL EXPRESSION IF ANY
IFDEF .PSECT,<
        .IFN ..JX1,ABSOLUTE,<PRINTX MSK NOT ABSOLUTE
                ..JX1==0>
        .IF ..JX1,ABSOLUTE,<
        .IF0 <<..JX1>-1B0>,<    ;;IF MASK IS JUST B0,
                JUMPL AC,BA>,<
        .IF0 <<..JX1>+1>,<      ;;IF MASK IF FULL WORD,
          JUMPN AC,BA>,<        ;;USE GIVEN CONDITION
                TXNE (AC,..JX1)
                JRST BA>>>
        PURGE ..JX1>
IFNDEF .PSECT,<
        .IF0 <<..JX1>-1B0>,<    ;;IF MASK IS JUST B0,
                JUMPL AC,BA>,<
        .IF0 <<..JX1>+1>,<      ;;IF MASK IF FULL WORD,
          JUMPN AC,BA>,<        ;;USE GIVEN CONDITION
                TXNE (AC,..JX1)
                JRST BA>>>
        PURGE ..JX1>
```

```
MACSYM  COMMON MACROS AND SYMBOLS       MACRO %53A(1072) 13:55 29-Dec-78 Page 11
MACSYM  MAC      8-Nov-77 10:47         JX -- JUMP ON MASK


DEFINE JXO (AC,MSK,BA)<
        ..JX1==MSK                      ;;EVAL EXPRESSION
IFDEF .PSECT,<
        .IFN ..JX1,ABSOLUTE,<PRINTX MSK NOT ABSOLUTE
                ..JX1==0>
        .IF ..JX1,ABSOLUTE,<
        .IF0 <<..JX1>-1B0>,<
                JUMPL AC,BA>,<
        ..ONEB (..BT,MSK)               ;;TEST MASK FOR ONLY ONE BIT ON
        .IF0 ..BT,<
          SETCM .SAC,AC                 ;;GENERAL CASE, GET COMPLEMENTS OF BITS
          JXE (.SAC,..JX1,BA)>,<        ;;JUMP IF BITS WERE ORIGINALLY ONES
            TXNE AC,..JX1               ;;TEST AND JUMP
            JRST BA>>>
        PURGE ..JX1>
IFNDEF .PSECT,<
        .IF0 <<..JX1>-1B0>,<
                JUMPL AC,BA>,<
        ..ONEB (..BT,MSK)               ;;TEST MASK FOR ONLY ONE BIT ON
        .IF0 ..BT,<
          SETCM .SAC,AC                 ;;GENERAL CASE, GET COMPLEMENTS OF BITS
          JXE (.SAC,..JX1,BA)>,<        ;;JUMP IF BITS WERE ORIGINALLY ONES
            TXNE AC,..JX1               ;;TEST AND JUMP
            JRST BA>>>
        PURGE ..JX1>

DEFINE JXF (AC,MSK,BA)<
        ..JX1==MSK                      ;;EVAL EXPRESSION
IFDEF .PSECT,<
        .IFN ..JX1,ABSOLUTE,<PRINTX MSK NOT ABSOLUTE
                ..JX1==0>
        .IF ..JX1,ABSOLUTE,<
        .IF0 <<..JX1>-1B0>,<
                JUMPGE AC,BA>,<
        ..ONEB (..BT,MSK)               ;;TEST MASK FOR ONLY ONE BIT ON
        .IF0 ..BT,<
          SETCM .SAC,AC                 ;;GENERAL CASE, GET COMPLEMENT OF BITS
          JXN (.SAC,..JX1,BA)>,<        ;;JUMP IF SOME ZEROS ORIGINALLY
            TXNN AC,..JX1               ;;TEST AND JUMP
            JRST BA>>>
        PURGE ..JX1>
IFNDEF .PSECT,<
        .IF0 <<..JX1>-1B0>,<
                JUMPGE AC,BA>,<
        ..ONEB (..BT,MSK)               ;;TEST MASK FOR ONLY ONE BIT ON
        .IF0 ..BT,<
          SETCM .SAC,AC                 ;;GENERAL CASE, GET COMPLEMENT OF BITS
          JXN (.SAC,..JX1,BA)>,<        ;;JUMP IF SOME ZEROS ORIGINALLY
            TXNN AC,..JX1               ;;TEST AND JUMP
            JRST BA>>>
        PURGE ..JX1>
```

```
MACSYM  COMMON MACROS AND SYMBOLS       MACRO %53A(1072) 13:55 29-Dec-78 Page 12
MACSYM  MAC      8-Nov-77 10:47         JX -- JUMP ON MASK

401
402                                             SUBTTL SUBFUNCTION MACROS
403
404                                     ;.IFØ CONDITION, ACTION IF CONDITION Ø, ACTION OTHERWISE
405
406                                     DEFINE .IFØ (COND,THEN,ELSE)<
407                                             ..IFT==COND             ;;GET LOCAL VALUE FOR CONDITION
408                                             IFE ..IFT,<
409                                             THEN
410                                             ..IFT==Ø>               ;;RESTORE IN CASE CHANGED BY NESTED .IFØ
411                                             IFN ..IFT,<
412                                             ELSE>>
413
414                                     ;CASE (NUMBER,<FIRST,SECOND,...,NTH>)
415
416                                     DEFINE .CASE (NUM,LIST)<
417                                             ..CSN==NUM
418                                             ..CSC==Ø
419                                             IRP LIST,<
420                                             IFE ..CSN-..CSC,<
421                                               STOPI
422                                               ..CAS1 (LIST)>
423                                             ..CSC==..CSC+1>>
424
425                                     DEFINE ..CAS1 (LIST)<
426                                             LIST>
427
428                                     ;TEST FOR FULL WORD, RH, LH, OR ARBITRARY BYTE
429
430                                     DEFINE ..TSIZ (SYM,MSK)<
431                                             SYM==3                  ;;ASSUME BYTE UNLESS...
432                                             IFE <MSK>+1,<SYM=Ø>     ;;FULL WORD IF MASK IS -1
433                                             IFE <MSK>-^O777777,<SYM==1> ;;RH IF MASK IS 777777
434                                             IFE <MSK>-^O777777B17,<SYM==2>> ;;LH IF MAST IS 777777,,Ø
435
436                                     ;TEST FOR LOC BEING AN AC -- SET SYM TO 1 IF AC, Ø IF NOT AC
437
438                                     DEFINE ..TSAC (SYM,LOC)<
439                                             IFNDEF .PSECT,<
440                                             SYM==Ø                  ;;ASSUME NOT AC UNLESS...
441                                             ..TSA1==<Z LOC>         ;;LOOK AT LOC
442                                               IFE ..TSA1&^O777777777760,<SYM==1> ;;AC IF VALUE IS Ø-17
443                                             >
444                                             IFDEF .PSECT,<
445                                             SYM==Ø                  ;;ASSUME NOT AC UNLESS...
446                                             ..TSA1==<Z LOC>         ;;LOOK AT LOC
447                                             .IF ..TSA1,ABSOLUTE,<   ;;SEE IF WE CAN TEST VALUE
448                                               IFE ..TSA1&^O777777777760,<SYM==1>> ;;AC IF VALUE IS Ø-17
449                                             PURGE ..TSA1>>
450
451                                     ;FUNCTION TO TEST FOR MASK CONTAINING EXACTLY ONE BIT. RETURNS
452                                     ;1 IFF LEFTMOST BIT AND RIGHTMOST BIT ARE SAME
453
454                                     DEFINE ..ONEB (SYM,MSK)<
455                                             SYM==<<<-<MSK>>&<MSK>>&<1B<^L<MSK>>>>>
```

```
MACSYM  COMMON MACROS AND SYMBOLS       MACRO %53A(1072) 13:55 29-Dec-78 Page 12-1
MACSYM  MAC      8-Nov-77 10:47         SUBFUNCTION MACROS


   456
   457                                  ;DEFAULT SCRACH AC
   458
   459                  000016          .SAC=16
```

```
MACSYM  COMMON MACROS AND SYMBOLS       MACRO %53A(1072) 13:55 29-Dec-78 Page 13
MACSYM  MAC      8-Nov-77 10:47         SUBFUNCTION MACROS


                        SUBTTL DEFSTR -- DEFINE DATA STRUCTURE

                ;DEFINE DATA STRUCTURE
                ; NAM - NAME OF STRUCTURE AS USED IN CODE
                ; LOCN - ADDRESS OF DATA
                ; POS - POSITION OF DATA WITHIN WORD (RIGHTMOST BIT NUMBER)
                ; SIZ - SIZE OF DATA (IN BITS) WITHIN WORD

                DEFINE DEFSTR (NAM,LOCN,POS,SIZ)<
                        NAM==<-1B<POS>+1B<POS-SIZ>> ;;ASSIGN SYMBOL TO HOLD MASK
                        IF1,<IFDEF %'NAM,<PRINTX ?NAM ALREADY DEFINED>>
                        DEFINE %'NAM (OP,AC,Y,MSK)<
                        OP (<AC>,LOCN''Y,MSK)>> ;;DEFINE MACRO TO HOLD LOCATION

                ;ALTERNATE FORM OF DEFSTR -- TAKES MASK INSTEAD OF POS,SIZ

                DEFINE MSKSTR (NAM,LOCN,MASK)<
                        NAM==MASK               ;;ASSIGN SYMBOL TO HOLD MASK
                        IF1,<IFDEF %'NAM,<PRINTX ?NAM ALREADY DEFINED>>
                        DEFINE %'NAM (OP,AC,Y,MSK)<
                        OP (<AC>,LOCN''Y,MSK)>> ;;DEFINE MACRO TO HOLD LOCATION

                ;..STR0 - PROCESS INSTANCE OF STRUCTURE USAGE, SINGLE STRUCTURE CASE.

                DEFINE ..STR0 (OP,AC,STR,Y)<
                        IFNDEF STR,<PRINTX STR IS NOT DEFINED
                          OP (<AC>,Y,.FWORD)>   ;;RESERVE A WORD, ASSUME WORD MASK
                        IFDEF STR,<
                        IFNDEF %'STR,<
                          OP (<AC>,Y,STR)>      ;;ASSUME NO OTHER LOCN
                        IFDEF %'STR,<
                          %'STR (OP,<AC>,Y,STR)>>> ;;DO IT
```

```
MACSYM  COMMON MACROS AND SYMBOLS       MACRO %53A(1072) 13:55 29-Dec-78 Page 14
MACSYM  MAC      8-Nov-77 10:47         DEFSTR -- DEFINE DATA STRUCTURE

                                        ;..STR1, ..STR2, ..STR3, AND ..STR4 ARE INTERNAL MACROS FOR PROCESSING
                                        ;INSTANCES OF STRUCTURE USAGE.

                                        DEFINE ..STR1 (OP,AC,STR,Y,CLL)<
                                                ..NS==0                 ;;INIT COUNT OF STR'S
                                                IRP STR,<..NS=..NS+1>   ;;COUNT STR'S
                                                IFE ..NS,<PRINTX ?EMPTY STRUCTURE LIST, OP>
                                                IFE ..NS-1,<            ;;THE ONE CASE, CAN DO FAST
                                                  ..STR0 (OP,<AC>,<STR>,Y)>
                                                IFG ..NS-1,<            ;;MORE THAN ONE, DO GENERAL CASE
                                                ..ICNS                  ;;INIT REMOTE MACRO
                                                ..CNS (<CLL (OP,<AC>,,>) ;;CONS ON CALL AND FIRST ARGS
                                                IRP STR,<               ;;DO ALL NAMES IN LIST
                                                  IFNDEF STR,<PRINTX STR NOT DEFINED>
                                                  IFDEF STR,<
                                                  IFNDEF %'STR,<
                                                  ..CNS (<,STR,Y>)>     ;;ASSUME NO OTHER LOCN
                                                  IFDEF %'STR,<
                                                  %'STR (..STR2,,Y,STR)> ;;STR MACRO WILL GIVE LOCN TO ..STR2
                                                  ..CNS (<)>)           ;;CLOSE ARG LIST
                                                  ..GCNS                ;;DO THIS AND PREVIOUS NAME
                                                  ..ICNS                ;;REINIT CONS
                                                  ..CNS (<CLL (OP,<AC>>) ;;PUT ON FIRST ARGS
                                                  IFNDEF %'STR,<
                                                  ..CNS (<,STR,Y>)>     ;;ASSUME NO OTHER LOCN
                                                  IFDEF %'STR,<
                                                  %'STR (..STR2,,Y,STR)>>> ;;PUT ON THIS ARG, END IRP
                                                ..CNS (<,,)>)           ;;CLOSE ARG LIST
                                                ..GCNS>>                ;;DO LAST CALL
```

```
;..STR2 -- CALLED BY ABOVE TO APPEND STRUCTURE NAME AND LOC TO ARG LIST

DEFINE ..STR2 (AA,LOC,STR)<
        ..CNS (<,STR,LOC>)>     ;;CONS ON NEXT ARG PAIR

;..STR3 -- CHECK FOR ALL STRUCTURES IN SAME REGISTER

DEFINE ..STR3 (OP,AC,S1,L1,S2,L2)<
        IFDIF <L1><L2>,<
          IFNB <L1>,<
            OP (<AC>,L1,..MSK)  ;;DO ACCUMULATED STUFF
            IFNB <L2>,<PRINTX S1 AND S2 ARE IN DIFFERENT WORDS>>
          ..MSK==0>             ;;INIT MASK
        IFNB <L2>,<
          ..MSK=..MSK!<S2>>>

;..STR4 -- COMPARE SUCCESSIVE ITEMS, DO SEPARATE OPERATION IF
;DIFFERENT WORDS ENCOUNTERED

DEFINE ..STR4 (OP,AC,S1,L1,S2,L2)<
        IFDIF <L1><L2>,<        ;;IF THIS DIFFERENT FROM PREVIOUS
          IFNB <L1>,<
            OP (<AC>,L1,..MSK)> ;;DO PREVIOUS
          ..MSK==0>             ;;REINIT MASK
        IFNB <L2>,<
          ..MSK=..MSK!<S2>>>    ;;ACCUMULATE MASK

;..STR5 - SAME AS ..STR4 EXCEPT GIVES EXTRA ARG IF MORE STUFF TO
;FOLLOW.

DEFINE ..STR5 (OP,AC,S1,L1,S2,L2)<
        IFDIF <L1><L2>,<        ;;IF THIS DIFFERENT FROM PREVIOUS,
          IFNB <L1>,<
            IFNB <L2>,<         ;;IF MORE TO COME,
                OP'1 (AC,L1,..MSK)> ;;DO VERSION 1
            IFB <L2>,<          ;;IF NO MORE,
                OP'2 (AC,L1,..MSK)>> ;;DO VERSION 2
          ..MSK==0>             ;;REINIT MASK
        IFNB <L2>,<
          ..MSK=..MSK!<S2>>>    ;;ACCUMULATE MASK
```

```
MACSYM  COMMON MACROS AND SYMBOLS       MACRO %53A(1072) 13:55 29-Dec-78 Page 16
MACSYM  MAC      8-Nov-77 10:47         DEFSTR -- DEFINE DATA STRUCTURE


                                        ;'REMOTE' MACROS USED TO BUILD UP ARG LIST

                                        ;INITIALIZE CONS -- DEFINES CONS

                                        DEFINE ..ICNS <
                                           DEFINE ..CNS (ARG)<
                                                ..CNS2 <ARG>,>

                                           DEFINE ..CNS2 (NEW,OLD)<
                                                DEFINE ..CNS (ARG)<
                                                  ..CNS2 <ARG>,<OLD'NEW>>>
                                           >

                                        ;GET CONS -- EXECUTE STRING ACCUMULATED

                                        DEFINE ..GCNS <
                                           DEFINE ..CNS2 (NEW,OLD)<
                                                  OLD>                  ;;MAKE ..CNS2 DO THE STUFF
                                                ..CNS ()>               ;;GET ..CNS2 CALLED WITH THE STUFF
```

```
MACSYM  COMMON MACROS AND SYMBOLS       MACRO %53A(1072) 13:55 29-Dec-78 Page 17
MACSYM  MAC      8-Nov-77 10:47         DEFSTR -- DEFINE DATA STRUCTURE


                                        ;SPECIFIC CASES

                                        ;LOAD, STORE
                                        ; AC - AC OPERAND
                                        ; STR - STRUCTURE NAME
                                        ; Y - (OPTIONAL) ADDITIONAL SPECIFICATION OF DATA LOCATION

                                        DEFINE LOAD (AC,STR,Y)<
                                                ..STRØ (..LDB,AC,STR,Y)>

                                           DEFINE ..LDB (AC,LOC,MSK)<
                                                ..TSIZ (..PST,MSK)
                                                .CASE ..PST,<<
                                                        MOVE AC,LOC>,<
                                                        HRRZ AC,LOC>,<
                                                        HLRZ AC,LOC>,<
                                                        LDB AC,[POINTR (LOC,MSK)]>>>

                                        DEFINE STOR (AC,STR,Y)<
                                                ..STRØ (..DPB,AC,STR,Y)>

                                           DEFINE ..DPB (AC,LOC,MSK)<
                                                ..TSIZ (..PST,MSK)
                                                .CASE ..PST,<<
                                                        MOVEM AC,LOC>,<
                                                        HRRM AC,LOC>,<
                                                        HRLM AC,LOC>,<
                                                        DPB AC,[POINTR (LOC,MSK)]>>>

                                        ;SET TO ZERO

                                        DEFINE SETZRO (STR,Y)<
                                                ..STR1 (..TQZ,,<STR>,Y,..STR4)>

                                           DEFINE ..TQZ (AC,LOC,MSK)<
                                                ..TSIZ (..PST,MSK)      ;;SET ..PST TO CASE NUMBER
                                                .CASE ..PST,<<
                                                        SETZM LOC>,<    ;;FULL WORD
                                                        HLLZS LOC>,<    ;;RH
                                                        HRRZS LOC>,<    ;;LH
                                                  ..TSAC (..ACT,LOC)    ;;SEE IF LOC IS AC
                                                  .IFØ ..ACT,<
                                                        MOVX .SAC,MSK   ;;NOT AC
                                                        ANDCAM .SAC,LOC>,<
                                                        ..TX (Z,LOC,MSK)>>>>
```

```
MACSYM  COMMON MACROS AND SYMBOLS       MACRO %53A(1072) 13:55 29-Dec-78 Page 18
MACSYM  MAC      8-Nov-77 10:47         DEFSTR -- DEFINE DATA STRUCTURE


                                        ;SET TO ONE

                                        DEFINE SETONE (STR,Y)<
                                                ..STR1 (..TQO,,<STR>,Y,..STR4)>

                                           DEFINE ..TQO (AC,LOC,MSK)<
                                                ..TSIZ (..PST,MSK)
                                                .CASE ..PST,<<
                                                        SETOM LOC>,<
                                                        HLLOS LOC>,<
                                                        HRROS LOC>,<
                                                  ..TSAC (..ACT,LOC)
                                                  .IFØ ..ACT,<
                                                        MOVX .SAC,MSK
                                                        IORM .SAC,LOC>,<
                                                        ..TX (O,LOC,MSK)>>>>

                                        ;SET TO COMPLEMENT

                                        DEFINE SETCMP (STR,Y)<
                                                ..STR1 (..TQC,,<STR>,Y,..STR4)>

                                           DEFINE ..TQC (AC,LOC,MSK)<
                                                ..TSIZ (..PST,MSK)
                                                .IFØ ..PST,<            ;;IF FULL WORD,
                                                        SETCMM LOC>,<   ;;CAN USE SETCMM
                                                  ..TSAC (..ACT,LOC)    ;;OTHERWISE, CHECK FOR AC
                                                  .IFØ ..ACT,<
                                                        MOVX .SAC,MSK
                                                        XORM .SAC,LOC>,<
                                                        ..TX(C,LOC,MSK)>>>
```

```
MACSYM  COMMON MACROS AND SYMBOLS       MACRO %53A(1072) 13:55 29-Dec-78 Page 19
MACSYM  MAC      8-Nov-77 10:47         DEFSTR -- DEFINE DATA STRUCTURE


661
662                                     ;INCREMENT, DECREMENT FIELD
663
664                                     DEFINE INCR (STR,Y)<
665                                             ..STR0 (.INCR0,,<STR>,Y)>
666
667                                        DEFINE .INCR0 (AC,LOC,MSK)<
668                                             ..PST==MSK&<-MSK>          ;;GET LOWEST BIT
669                                             .IF0 ..PST-1,<
670                                                     AOS LOC>,<         ;;BIT 35, CAN USE AOS
671                                                     MOVX .SAC,..PST ;;LOAD A ONE IN THE APPROPRIATE POSITION
672                                                     ADDM .SAC,LOC>>
673
674                                     DEFINE DECR (STR,Y)<
675                                             ..STR0 (.DECR0,,<STR>,Y)>
676
677                                        DEFINE .DECR0 (AC,LOC,MSK)<
678                                             ..PST==MSK&<-MSK>
679                                             .IF0 ..PST-1,<
680                                                     SOS LOC>,<         ;;BIT 35, CAN USE SOS
681                                                     MOVX .SAC,-..PST ;;LOAD -1 IN APPROPRIATE POSITION
682                                                     ADDM .SAC,LOC>>
683
684                                     ;GENERAL DEFAULT, TAKES OPCODE
685
686                                     DEFINE OPSTR (OP,STR,Y)<
687                                             ..STR0 (.OPST1,<OP>,<STR>,Y)>
688
689                                        DEFINE .OPST1 (OP,LOC,MSK)<
690                                             ..TSIZ (..PST,MSK)
691                                             .IF0 ..PST,<
692                                                     OP LOC>,<          ;;FULL WORD, USE GIVEN OP DIRECTLY
693                                                     ..LDB .SAC,LOC,MSK ;;OTHERWISE, GET SPECIFIED BYTE
694                                                     OP .SAC>>
695
696                                     DEFINE OPSTRM (OP,STR,Y)<
697                                             ..STR0 (.OPST2,<OP>,<STR>,Y)>
698
699                                        DEFINE .OPST2 (OP,LOC,MSK)<
700                                             ..TSIZ (..PST,MSK)
701                                             .IF0 ..PST,<
702                                                     OP LOC>,<          ;;FULL WORD, USE OP DIRECTLY
703                                                     ..LDB .SAC,LOC,MSK
704                                                     OP .SAC
705                                                     ..DPB .SAC,LOC,MSK>>
```

```
MACSYM  COMMON MACROS AND SYMBOLS       MACRO %53A(1072) 13:55 29-Dec-78 Page 20
MACSYM  MAC      8-Nov-77 10:47         DEFSTR -- DEFINE DATA STRUCTURE


                                ;JUMP IF ALL FIELDS ARE 0 (ONE REGISTER AT MOST)

                                DEFINE JE (STR,Y,BA)<
                                        ..STR1 (..JE,<BA>,<STR>,Y,..STR3)>

                                   DEFINE ..JE (BA,LOC,MSK)<
                                        ..TSAC (..ACT,LOC)      ;;SEE IF AC
                                        .IF0 ..ACT,<
                                          ..TSIZ (..PST,MSK)    ;;SEE WHICH CASE
                                          .CASE ..PST,<<
                                                SKIPN LOC       ;;FULL WORD, TEST IN MEMORY
                                                JRST BA>,<
                                                HRRZ .SAC,LOC   ;;RIGHT HALF, GET IT
                                                JUMPE .SAC,BA>,<
                                                HLRZ .SAC,LOC   ;;LEFT HALF, GET IT
                                                JUMPE .SAC,BA>,<
                                                MOVE .SAC,LOC   ;;NOTA, GET WORD
                                                JXE (.SAC,MSK,<BA>)>>>,<
                                          JXE (LOC,MSK,<BA>)>>

                                ;JUMP IF NOT ALL FIELDS ARE 0 (ONE REGISTER AT MOST)

                                DEFINE JN (STR,Y,BA)<
                                        ..STR1 (..JN,<BA>,<STR>,Y,..STR3)>

                                   DEFINE ..JN (BA,LOC,MSK)<
                                        ..TSAC (..ACT,LOC)      ;;SEE IF AC
                                        .IF0 ..ACT,<
                                          ..TSIZ (..PST,MSK)
                                          .CASE ..PST,<<
                                                SKIPE LOC       ;;FULL WORD, TEST IN MEMORY
                                                JRST BA>,<
                                                HRRZ .SAC,LOC   ;;RIGHT HALF, GET IT
                                                JUMPN .SAC,BA>,<
                                                HLRZ .SAC,LOC   ;;LEFT HALF, GET IT
                                                JUMPN .SAC,BA>,<
                                                MOVE .SAC,LOC   ;;NOTA, GET WORD
                                                JXN (.SAC,MSK,<BA>)>>>,<
                                          JXN (LOC,MSK,<BA>)>>
```

```
MACSYM  COMMON MACROS AND SYMBOLS       MACRO %53A(1072) 13:55 29-Dec-78 Page 21
MACSYM  MAC      8-Nov-77 10:47         DEFSTR -- DEFINE DATA STRUCTURE


  746
  747                                   ;JOR - JUMP ON 'OR' OF ALL FIELDS
  748
  749                                   DEFINE JOR (STR,Y,BA)<
  750                                           ..STR1 (..JN,<BA>,<STR>,Y,..STR4)>
  751
  752                                   ;JNAND - JUMP ON NOT 'AND' OF ALL FIELDS
  753
  754                                   DEFINE JNAND (STR,Y,BA)<
  755                                           ..STR1 (..JNA3,<BA>,<STR>,Y,..STR4)>
  756
  757                                      DEFINE ..JNA3 (BA,LOC,MSK)<
  758                                           ..TSAC (..ACT,LOC)
  759                                           .IF0 ..ACT,<
  760                                             SETCM .SAC,LOC        ;;NOT AC, GET COMPLEMENT OF WORD
  761                                             JXN (.SAC,MSK,<BA>)>,<        ;;JUMP IF ANY BITS ORIGINALLY OFF
  762                                             JXF (LOC,MSK,<BA>)>>  ;;DO AC CASE
```

```
MACSYM  COMMON MACROS AND SYMBOLS       MACRO %53A(1072) 13:55 29-Dec-78 Page 22
MACSYM  MAC      8-Nov-77 10:47         DEFSTR -- DEFINE DATA STRUCTURE


;JAND - JUMP ON 'AND' OF ALL FIELDS

DEFINE JAND (STR,Y,BA,%TG)<
        ..STR1 (..JAN,<%TG,<BA>>,<STR>,Y,..STR5)
%TG:>

   DEFINE ..JAN1 (BA1,BA2,LOC,MSK)<
        ..JNA3 (BA1,LOC,MSK)>   ;;DO JUMP NAND TO LOCAL TAG

   DEFINE ..JAN2 (BA1,BA2,LOC,MSK)<
        ..TSAC (..ACT,LOC)
        .IF0 ..ACT,<
          SETCM .SAC,LOC        ;;NOT AC, GET COMPLEMENT OF WORD
          JXE (.SAC,MSK,<BA2>)>,<       ;;JUMP IF ALL BITS ORIGINALLY ONES
          JXO (LOC,MSK,<BA2>)>> ;;DO AC CASE

;JNOR - JUMP ON NOT 'OR' OF ALL FIELDS

DEFINE JNOR (STR,Y,BA,%TG)<
        ..STR1 (..JNO,<%TG,<BA>>,<STR>,Y,..STR5)
%TG:>

   DEFINE ..JNO1 (BA1,BA2,LOC,MSK)<
        ..JN (BA1,LOC,MSK)>     ;;DO JUMP OR TO LOCAL TAG

   DEFINE ..JNO2 (BA1,BA2,LOC,MSK)<
        ..JE (<BA2>,LOC,MSK)>   ;;DO JUMP NOR TO GIVEN TAG

;TEST AND MODIFY GROUP USING DEFINED STRUCTURES.  TEST-ONLY AND
;MODIFY-ONLY PROVIDED FOR COMPLETENESS.

DEFINE ..DOTY (M,T)<            ;;MACRO TO DEFINE ALL CASES
        IRP M,<
        IRP T,<
          DEFINE TQ'M'T (STR,Y)<
            ..STR1 (..TY,M'T,<STR>,Y,..STR3)>>>>

        ..DOTY (<N,O,Z,C>,<,E,N,A>)^ ;DO 16 DEFINES
        PURGE ..DOTY

;ALL TY MACROS CALL ..TY AFTER INITIAL STRUCTURE PROCESSING

DEFINE ..TY (MT,LOC,MSK)<
        ..TSAC (..ACT,LOC)      ;;SEE IF LOC IS AC
        .IF0 ..ACT,<
                PRINTX ?TQ'MT - LOC NOT IN AC>,<
                TX'MT LOC,MSK>>
```

```
MACSYM  COMMON MACROS AND SYMBOLS       MACRO %53A(1072) 13:55 29-Dec-78 Page 23
MACSYM  MAC      8-Nov-77 10:47         DEFSTR -- DEFINE DATA STRUCTURE

                                              SUBTTL CALL, RET, JSERR

                                           IFE REL,<
                                              EXTERN JSERR0,JSHLT0,R,RSKP>

                                        ;CALL AND RETURN

                    000001              .AC1==1                         ;ACS FOR JSYS ARGS
                    000002              .AC2==2
                    000003              .AC3==3
                    000016              .A16==16                        ;TEMP FOR STKVAR AND TRVAR
                    000017              P=17                            ;STACK POINTER

          260740    000000              OPDEF CALL [PUSHJ P,0]
          263740    000000              OPDEF RET [POPJ P,0]

                                        ;ABBREVIATION FOR  CALL, RET, RETSKP

          254000    000000              OPDEF CALLRET [JRST]

                                        DEFINE RETSKP <
                                              JRST RSKP>

                                        ;MACRO TO PRINT MESSAGE ON TERMINAL

                                        DEFINE TMSG ($MSG)<
                                              HRROI .AC1,[ASCIZ \$MSG\]
                                              PSOUT>

                                        ;MACRO TO OUTPUT MESSAGE TO FILE
                                        ; ASSUMES JFN ALREADY IN .AC1

                                        DEFINE FMSG ($MSG)<
                                              HRROI .AC2,[ASCIZ \$MSG\]
                                              MOVEI .AC3,0
                                              SOUT>

                                        ;MACRO TO PRINT MESSAGE FOR LAST ERROR, RETURNS +1

                                        DEFINE PERSTR ($MSG)<
                                           IFNB <$MSG>,<
                                              TMSG <$MSG>>
                                              CALL JSMSG0>

                                        ;MACRO TO PRINT JSYS ERROR MESSAGE, RETURNS +1 ALWAYS

                                        DEFINE JSERR<
                                              CALL JSERR0>

                                        ;MACRO FOR FATAL JSYS ERROR, PRINTS MSG THEN HALTS

                                        DEFINE JSHLT<
                                              CALL JSHLT0>
```

```
MACSYM  COMMON MACROS AND SYMBOLS       MACRO %53A(1072) 13:55 29-Dec-78 Page 23-1
MACSYM  MAC      8-Nov-77 10:47         CALL, RET, JSERR

   866                                  ;PRINT ERROR MESSAGE IF JSYS FAILS
   867
   868                                  DEFINE ERMSG(TEXT),<
   869                                         ERJMP   [TMSG <? TEXT>
   870                                                  JSHLT]
   871                                  >
   872
   873                                  ;MAKE SYMBOLS EXTERN IF NOT ALREADY DEFINED
   874
   875                                  DEFINE EXT (SYM)<
   876                                     IF2,<
   877                                         IRP SYM,<
   878                                         IFNDEF SYM,<EXTERN SYM
   879                                         SUPPRE SYM>>>>
```

```
MACSYM  COMMON MACROS AND SYMBOLS       MACRO %53A(1072) 13:55 29-Dec-78 Page 24
MACSYM  MAC      8-Nov-77 10:47         CALL, RET, JSERR


                        SUBTTL SUPPORT CODE FOR JSERR

   IFN REL,<

A=1
B=2
C=3
D=4

;JSYS ERROR HANDLER
;       CALL JSERRØ
; RETURNS +1: ALWAYS, CAN BE USED IN +1 RETURN OF JSYS'S

JSERRØ::MOVEI A,.PRIIN
        CFIBF                   ;CLEAR TYPAHEAD
        MOVEI A,.PRIOU
        DOBE                    ;WAIT FOR PREVIOUS OUTPUT TO FINISH
        TMSG <
? JSYS ERROR: >
JSMSGØ::MOVEI A,.PRIOU
        HRLOI B,.FHSLF          ;SAY  THIS FORK ,, LAST ERROR
        SETZ C,
        ERSTR
         JFCL
         JFCL
        TMSG <
>
        RET

;FATAL JSYS ERROR - PRINT MESSAGE AND HALT
;       CALL JSHLTØ
; RETURNS: NEVER

JSHLTØ::CALL JSERRØ             ;PRINT THE MSG
JSHLT1: HALTF
        TMSG <PROGRAM CANNOT CONTINUE
>
        JRST JSHLT1             ;HALT AGAIN IF CONTINUED
   >                            ;END OF IFN REL,
```

```
MACSYM  COMMON MACROS AND SYMBOLS       MACRO %53A(1072) 13:55 29-Dec-78 Page 25
MACSYM  MAC      8-Nov-77 10:47         SUPPORT CODE FOR JSERR

920
921                                             SUBTTL STKVAR - STACK VARIABLE FACILITY
922
923                                     ;MACRO FOR ALLOCATING VARIABLES ON THE STACK. ITS ARGUMENT IS
924                                     ;A LIST OF ITEMS.  EACH ITEM MAY BE:
925                                     ; 1. A SINGLE VARIABLE WHICH WILL BE ALLOCATED ONE WORD
926                                     ; 2. A VARIABLE AND SIZE PARAMETER WRITTEN AS <VAR,SIZ>.  THE
927                                     ;       VARIABLE WILL BE ALLOCATED THE SPECIFIED NUMBER OF WORDS.
928                                     ;RETURN FROM A SUBROUTINE USING THIS FACILITY MUST BE VIA
929                                     ;RET OR RETSKP.  A DUMMY RETURN WHICH FIXES UP THE STACK IS PUT ON
930                                     ;THE STACK AT THE POINT THE STKVAR IS ENCOUNTERED.
931                                     ;WITHIN THE RANGE OF A STKVAR, PUSH/POP CANNOT BE USED AS THEY WILL
932                                     ;CAUSE THE VARIABLES (WHICH ARE DEFINED AS RELATIVE STACK LOCATIONS)
933                                     ;TO REFERENCE THE WRONG PLACE.
934                                     ;TYPICAL USE:   STKVAR <AA,BB,<QQ,5>,ZZ>
935
936                                        IFE REL,<
937                                             EXTERN .STKST,.STKRT>
938
939                                     DEFINE STKVAR (ARGS)<
940                                             ..STKR==10              ;;REMEMBER RADIX
941                                             RADIX 8
942                                             ..STKN==0
943                                             IRP ARGS,<
944                                               .STKV1 (ARGS)>
945                                             JSP .A16,.STKST
946                                              ..STKN,,..STKN
947                                             RADIX ..STKR
948                                             PURGE ..STKN,..STKR,..STKQ
949                                        >
950
951                                     ;INTERMEDIATE MACRO TO PEAL OFF ANGLEBRACKETS IF ANY
952
953                                     DEFINE .STKV1 (ARG)<
954                                             .STKV2 (ARG)>
955
956                                     ;INTERMEDIATE MACRO TO CALCULATE OFFSET AND COUNT VARIABLES
957
958                                     DEFINE .STKV2 (VAR,SIZ)<
959                                             IFB <SIZ>,<..STKN==..STKN+1>
960                                             IFNB <SIZ>,<..STKN==..STKN+SIZ>
961                                             ..STKQ==..STKN+1
962                                             .STKV3 (VAR,\..STKQ)>
963
964                                     ;INNERMOST MACRO TO DEFINE VARIABLE
965
966                                     DEFINE .STKV3 (VAR,LOC)<
967                                        IFDEF VAR,<.IF VAR,SYMBOL,<PRINTX STKVAR VAR ALREADY DEFINED>>
968                                             DEFINE VAR<-^O'LOC(P)>
969                                             $'VAR==<Z VAR>>         ;SYMBOL FOR DDT
```

```
MACSYM  COMMON MACROS AND SYMBOLS       MACRO %53A(1072) 13:55 29-Dec-78 Page 26
MACSYM  MAC      8-Nov-77 10:47         STKVAR - STACK VARIABLE FACILITY

   970
   971                                     IFN REL,<
   972
   973                                  ;COMMON ENTRY AND EXIT ROUTINE FOR STACK VARIABLE
   974
   975                                  .STKST::ADD P,0(.A16)            ;BUMP STACK FOR VARIABLES USED
   976                                          JUMPGE P,STKSOV          ;TEST FOR STACK OVERFLOW
   977                                  STKSE1: PUSH P,0(.A16)           ;SAVE BLOCK SIZE FOR RETURN
   978                                          PUSHJ P,1(.A16)          ;CONTINUE ROUTINE, EXIT TO .+1
   979                                  .STKRT::JRST STKRT0              ;NON-SKIP RETURN COMES HERE
   980                                          POP P,.A16               ;SKIP RETURN COMES HERE-RECOVER COUNT
   981                                          SUB P,.A16               ;ADJUST STACK TO REMOVE BLOCK
   982                                          AOS 0(P)                 ;NOW DO SKIP RETURN
   983                                          RET
   984
   985                                  STKRT0: POP P,.A16               ;RECOVER COUNT
   986                                          SUB P,.A16               ;ADJUST STACK TO REMOVE BLOCK
   987                                          RET                      ;DO NON-SKIP RETURN
   988
   989                                  STKSOV: SUB P,0(.A16)            ;STACK OVERFLOW- UNDO ADD
   990                                          HLL .A16,0(.A16)         ;SETUP TO DO MULTIPLE PUSH, GET COUNT
   991                                  STKSO1: PUSH P,[0]               ;DO ONE PUSH AT A TIME, GET REGULAR
   992                                          SUB .A16,[1,,0]          ; ACTION ON OVERFLOW
   993                                          TLNE .A16,777777         ;COUNT DOWN TO 0?
   994                                          JRST STKSO1              ;NO, KEEP PUSHING
   995                                          JRST STKSE1
   996                                     >                             ;END OF IFN REL,
```

```
MACSYM  COMMON MACROS AND SYMBOLS       MACRO %53A(1072) 13:55 29-Dec-78 Page 27
MACSYM  MAC      8-Nov-77 10:47         STKVAR - STACK VARIABLE FACILITY

                                               SUBTTL TRVAR - TRANSIENT VARIABLE FACILITY

                                        ;TRANSIENT (STACK) VARIABLE FACILITY - EQUIVALENT TO STKVAR
                                        ;EXCEPT ALLOWS VARIABLES TO BE USED WITHIN LOWER LEVEL ROUTINES
                                        ;AND AFTER OTHER THINGS HAVE BEEN PUSHED ON STACK.
                                        ;N.B. USES .FP AS FRAME POINTER - MUST NOT BE CHANGED WHILE
                                        ;VARIABLES IN USE.

                    000015              .FP==15                         ;DEFAULT FRAME POINTER

                                           IFE REL,<
                                                EXTERN .TRSET,.TRRET,.ASSET,.ASRET>

                                        DEFINE TRVAR (VARS)<
                                                ..TRR==10               ;;REMEMBER CURRENT RADIX
                                                RADIX 8
                                                ..NV==1                 ;;INIT COUNT OF STACK WORDS
                                                IRP VARS,<
                                                   .TRV1 (VARS)>        ;;PROCESS LIST
                                                JSP .A16,.TRSET         ;;ALLOCATE STACK SPACE, SETUP .FP
                                                 ..NV-1,,..NV-1
                                                RADIX ..TRR             ;;RESTORE RADIX
                                                PURGE ..TRR,..NV>       ;;CLEAN UP

                                        DEFINE .TRV1 (VAR)<
                                                .TRV2 (VAR)>            ;;PEEL OFF ANGLEBRACKETS IF ANY

                                        DEFINE .TRV2 (NAM,SIZ)<
                                                .TRV3 (NAM,\..NV)       ;;DEFINE VARIABLE
                                                IFB <SIZ>,<..NV=..NV+1>
                                                IFNB <SIZ>,<..NV=..NV+SIZ>>

                                        DEFINE .TRV3 (NAM,LOC)<
                                           IFDEF NAM,<.IF NAM,SYMBOL,<PRINTX TRVAR NAM ALREADY DEFINED>>
                                                DEFINE NAM<^O'LOC(.FP)>
                                                $'NAM==<Z NAM>>         ;;SYMBOL FOR DDT

                                        ;AC SUBROUTINE - ENTRY FOR SUBROUTINE CALLED WITH 1-4 ARGS IN ACS T1-T4.
                                        ;USES .FP AS FRAME PTR LIKE TRVAR

                                        DEFINE ASUBR (ARGS)<
                                                ..TRR==10               ;;SAVE RADIX
                                                RADIX 8
                                                ..NV==1                 ;;INIT ARG COUNT
                                                IRP ARGS,<
                                                   .TRV1 (ARGS)>        ;;DEFINE ARG SYMBOL
                                                IFG ..NV-5,<PRINTX ?TOO MANY ARGUMENTS: ARGS>
                                                JSP .A16,.ASSET         ;;SETUP STACK
                                                RADIX ..TRR             ;;RESTORE RADIX
                                                PURGE ..TRR,..NV>
```

```
1048
1049                            IFN REL,<
1050                    ;SUPPORT ROUTINE FOR TRVAR
1051
1052                    .TRSET::PUSH P,.FP              ;PRESERVE OLD .FP
1053                            MOVE .FP,P              ;SETUP FRAME PTR
1054                            ADD P,Ø(.A16)           ;ALLOCATE SPACE
1055                            JUMPGE P,TRSOV
1056                    TRSET1: PUSHJ P,1(.A16)         ;CONTINUE ROUTINE, EXIT VIA .+1
1057                    .TRRET::JRST [  MOVEM .FP,P     ;CLEAR STACK
1058                                    POP P,.FP       ;RESTORE OLD .FP
1059                                    POPJ P,]
1060                            MOVEM .FP,P             ;HERE IF SKIP RETURN
1061                            POP P,.FP
1062                            AOS Ø(P)                ;PASS SKIP RETURN
1063                            POPJ P,
1064
1065                    TRSOV:  SUB P,Ø(.A16)           ;STACK OVERFLOW - UNDO ADD
1066                            HLL .A16,Ø(.A16)        ;GET COUNT
1067                    TRSOV1: PUSH P,[Ø]              ;DO ONE PUSH AT A TIME, GET REGULAR
1068                            SUB .A16,[1,,Ø]         ; ACTION ON OVERFLOW
1069                            TLNE .A16,777777        ;COUNT TO Ø?
1070                            JRST TRSOV1             ;NO, KEEP PUSHING
1071                            JRST TRSET1             ;CONTINUE SETUP
1072
1073                    ;SUPPORT ROUTINE FOR ASUBR
1074
1075                    .ASSET::PUSH P,.FP              ;SAVE .FP
1076                            MOVE .FP,P              ;SETUP FRAME POINTER
1077                            ADD P,[4,,4]            ;ADJUST STACK
1078                            JUMPGE P,[SUB P,[4,,4]  ;PROBABLE OVERFLOW
1079                                    PUSH P,A        ;DO WITH PUSH, GET INTERRUPT...
1080                                    PUSH P,B
1081                                    PUSH P,C
1082                                    PUSH P,D
1083                                    JRST ASSET1]
1084                            DMOVEM A,1(.FP) ;SAVE ARGS
1085                            DMOVEM C,3(.FP)
1086                    ASSET1: PUSHJ P,Ø(.A16)         ;CONTINUE ROUTINE
1087                    .ASRET:: JRST [ MOVEM .FP,P     ;NO-SKIP RETURN, CLEAR STACK
1088                                    POP P,.FP
1089                                    POPJ P,]
1090                            MOVEM .FP,P             ;SKIP RETURN, CLEAR STZCK
1091                            POP P,.FP
1092                            AOS Ø(P)
1093                            POPJ P,
1094                       >                            ;END OF IFN REL,
```

```
MACSYM  COMMON MACROS AND SYMBOLS       MACRO %53A(1072) 13:55 29-Dec-78 Page 29
MACSYM  MAC     8-Nov-77 10:47          TRVAR - TRANSIENT VARIABLE FACILITY


                                ;AC VARIABLE FACILITY

                                   IFE REL,<
                                        EXTERN .SAV1,.SAV2,.SAV3,.SAV4,.SAV8>

           000005               .FPAC==5                        ;FIRST PRESERVED AC
           000010               .NPAC==10                       ;NUMBER OF PRESERVED ACS

                                DEFINE ACVAR (LIST)<
                                        ..NAC==0                ;;INIT NUMBER OF ACS USED
                                        IRP LIST,<
                                          .ACV1 (LIST)>         ;;PROCESS ITEMS
                                        .ACV3 (\..NAC)>         ;;SAVE ACS USED

                                DEFINE .ACV1 (ITEM)<
                                        .ACV2 (ITEM)>           ;;PEEL OFF ANGLEBRACKETS IF ANY

                                DEFINE .ACV2 (NAM,SIZ)<
                                        NAM=.FPAC+..NAC         ;;DEFINE VARIABLE
                                        IFB <SIZ>,<..NAC=..NAC+1>
                                        IFNB <SIZ>,<..NAC=..NAC+SIZ>>

                                DEFINE .ACV3 (N)<
                                        IFG N-.NPAC,<PRINTX ?TOO MANY ACS USED>
                                        IFLE N-4,<
                                          JSP .A16,.SAV'N>      ;;SAVE ACTUAL NUMBER USED
                                        IFG N-4,<
                                          JSP .A16,.SAV8>>      ;;SAVE ALL

                                   IFN REL,<
                                ;SUPPORT ROUTINES FOR AC VARIABLE FACILITY

                                .SAV1:: PUSH P,.FPAC
                                        PUSHJ P,0(.A16)         ;CONTINUE PROGRAM
                                         SKIPA
                                        AOS -1(P)
                                        POP P,.FPAC
                                        POPJ P,

                                .SAV2:: PUSH P,.FPAC
                                        PUSH P,.FPAC+1
                                        PUSHJ P,0(.A16)
                                         SKIPA
                                        AOS -2(P)
                                        POP P,.FPAC+1
                                        POP P,.FPAC
                                        POPJ P,

                                .SAV3::
                                .SAV4:: PUSH P,.FPAC
                                        PUSH P,.FPAC+1
                                        PUSH P,.FPAC+2
                                        PUSH P,.FPAC+3
                                        PUSHJ P,0(.A16)
```

```
MACSYM  COMMON MACROS AND SYMBOLS       MACRO %53A(1072) 13:55 29-Dec-78 Page 29-1
MACSYM  MAC      8-Nov-77 10:47         TRVAR - TRANSIENT VARIABLE FACILITY

  1150                                          SKIPA
  1151                                          AOS -4(P)
  1152                                          POP P,.FPAC+3
  1153                                          POP P,.FPAC+2
  1154                                          POP P,.FPAC+1
  1155                                          POP P,.FPAC
  1156                                          POPJ P,
  1157
  1158                                  .SAV8:: ADD P,[10,,10]
  1159                                          JUMPGE P,[HALT .]
  1160                                          DMOVEM .FPAC,-7(P)
  1161                                          DMOVEM .FPAC+2,-5(P)
  1162                                          DMOVEM .FPAC+4,-3(P)
  1163                                          DMOVEM .FPAC+6,-1(P)
  1164                                          PUSHJ P,0(.A16)
  1165                                           SKIPA
  1166                                          AOS -10(P)
  1167                                          DMOVE .FPAC+6,-1(P)
  1168                                          DMOVE .FPAC+4,-3(P)
  1169                                          DMOVE .FPAC+2,-5(P)
  1170                                          DMOVE .FPAC,-7(P)
  1171                                          SUB P,[10,,10]
  1172                                          POPJ P,
  1173                                     >
```

```
MACSYM  COMMON MACROS AND SYMBOLS       MACRO %53A(1072) 13:55 29-Dec-78 Page 30
MACSYM  MAC      8-Nov-77 10:47         TRVAR - TRANSIENT VARIABLE FACILITY


                                        ;AC SAVE FACILITY - COMPILES OPEN PUSH'S
                                        ;       SAVEAC <LIST-OF-ACS>
                                        ;DUMMY ROUTINE PUT ON STACK TO CAUSE AUTOMATIC RESTORE. SUPPORTS
                                        ; +1 OR +2 RETURNS.

                                        DEFINE SAVEAC (ACS)<
                                                .NAC==0
                                                IRP ACS,<
                                                  PUSH P,ACS            ;;SAVE AN AC
                                                  .NAC=.NAC+1>          ;;COUNT THEM
                                                .N1==.NAC
                                                SETMI .A16,[CAIA        ;;STACK DUMMY RETURN
                                                        AOS -.N1(P)     ;;HANDLE SKIP RETURN
                                                  IRP ACS,<
                                                        .N1=.N1-1
                                                        MOVE ACS,-.N1(P)>  ;;RESTORE AN AC
                                                        SUB P,[.NAC,,.NAC]  ;;CLEAR STACK
                                                        POPJ P,]        ;;FINAL RETURN
                                                PUSH P,.A16>

                                           IFN REL,<
                                        ;STANDARD RETURNS

                                        RSKP::  AOS 0(P)
                                        R::     RET
                                           >                            ;END OF IFN REL,

000000'                                 LIT                             ;MAKE SURE LITERALS COME BEFORE END MARK
                                           IFN REL,<
                                        .RLEND==:.-1                    ;MARK END OF CODE IN MACREL
                                           >
                                          IF2,<PURGE REL>               ;FLUSH REL FROM UNIV FILE
                                                END

NO ERRORS DETECTED

PROGRAM BREAK IS 000000
CPU TIME USED 00:03.083

24P CORE USED
```

```
MACSYM  COMMON MACROS AND SYMBOLS        MACRO %53A(1072) 13:55 29-Dec-78 Page S-1
MACSYM  MAC      8-Nov-77 10:47          SYMBOL TABLE

CALL    260740  000000          .CHCRB          000035  sin
CALLRE  254000  000000          .CHCRT          000015  sin
JSERRØ          000000  ext     .CHCUN          000037  sin
JSHLTØ          000000  ext     .CHDEL          000177  sin
P               000017          .CHESC          000033  sin
PC%AFI  001000  000000  sin     .CHFFD          000014  sin
PC%ATN  000600  000000  sin     .CHLFD          000012  sin
PC%BIS  020000  000000  sin     .CHNUL          000000  sin
PC%CYØ  200000  000000  sin     .CHTAB          000011  sin
PC%CY1  100000  000000  sin     .CHVTB          000013  sin
PC%FOV  040000  000000  sin     .FP             000015  spd
PC%FUF  000100  000000  sin     .FPAC           000005  spd
PC%LIP  002000  000000  sin     .FWORD  777777  777777  sin
PC%NDV  000040  000000  sin     .INFIN  377777  777777  sin
PC%OVF  400000  000000  sin     .LHALF  777777  000000  sin
PC%UIO  004000  000000  sin     .MINFI  400000  000000  sin
PC%USR  010000  000000  sin     .NPAC           000010  spd
R               000000  ext     .RHALF          777777  sin
RET     263740  000000          .SAC            000016
RSKP            000000  ext     .SAV1           000000  ext
VI%EDN          777777  sin     .SAV2           000000  ext
VI%MAJ  077700  000000  sin     .SAV3           000000  ext
VI%MIN  000077  000000  sin     .SAV4           000000  ext
VI%WHO  700000  000000  sin     .SAV8           000000  ext
.A16            000016  spd     .STKRT          000000  ext
.AC1            000001  spd     .STKST          000000  ext
.AC2            000002  spd     .TRRET          000000  ext
.AC3            000003  spd     .TRSET          000000  ext
.ASRET          000000  ext
.ASSET          000000  ext
.CHAL2          000176  sin
.CHALT          000175  sin
.CHBEL          000007  sin
.CHBSP          000010  sin
.CHCBS          000034  sin
.CHCCF          000036  sin
.CHCNA          000001  sin
.CHCNB          000002  sin
.CHCNC          000003  sin
.CHCND          000004  sin
.CHCNE          000005  sin
.CHCNF          000006  sin
.CHCNN          000016  sin
.CHCNO          000017  sin
.CHCNP          000020  sin
.CHCNQ          000021  sin
.CHCNR          000022  sin
.CHCNS          000023  sin
.CHCNT          000024  sin
.CHCNU          000025  sin
.CHCNV          000026  sin
.CHCNW          000027  sin
.CHCNX          000030  sin
.CHCNY          000031  sin
.CHCNZ          000032  sin
```

JSERR0	815#													
JSHLT0	815#													
P	823#	825	826											
PC%AFI	128#													
PC%ATN	129#													
PC%BIS	124#													
PC%CY0	121#													
PC%CY1	122#													
PC%FOV	123#													
PC%FUF	130#													
PC%LIP	127#													
PC%NDV	131#													
PC%OVF	120#													
PC%UIO	126#													
PC%USR	125#													
R	815#													
REL	34	35	38	814	883	936	971	1008	1049	1098	1125	1195	1203	1206
RSKP	815#													
VI%EDN	65#													
VI%MAJ	63#													
VI%MIN	64#													
VI%WHO	62#													
.A16	822#													
.AC1	819#													
.AC2	820#													
.AC3	821#													
.ASRET	1009#													
.ASSET	1009#													
.CHAL2	113#													
.CHALT	112#													
.CHBEL	86#													
.CHBSP	87#													
.CHCBS	107#													
.CHCCF	109#													
.CHCNA	80#													
.CHCNB	81#													
.CHCNC	82#													
.CHCND	83#													
.CHCNE	84#													
.CHCNF	85#													
.CHCNN	93#													
.CHCNO	94#													
.CHCNP	95#													
.CHCNQ	96#													
.CHCNR	97#													
.CHCNS	98#													
.CHCNT	99#													
.CHCNU	100#													
.CHCNV	101#													
.CHCNW	102#													
.CHCNX	103#													
.CHCNY	104#													
.CHCNZ	105#													

```
.CHCRB   108#
.CHCRT    92#
.CHCUN   110#
.CHDEL   114#
.CHESC   106#
.CHFFD    91#
.CHLFD    89#
.CHNUL    79#
.CHTAB    88#
.CHVTB    90#
.FP     1006#
.FPAC   1101#
.FWORD    75#
.INFIN    71#
.LHALF    73#
.MINFI    72#
.NPAC   1102#
.RHALF    74#
.SAC     459#
.SAV1   1099#
.SAV2   1099#
.SAV3   1099#
.SAV4   1099#
.SAV8   1099#
.STKRT    937#
.STKST    937#
.TRRET  1009#
.TRSET  1009#
```

ACVAR	1104#
ANDX	226#
ASUBR	1038#
CALL	825#
CALLRE	830#
DECR	674#
DEFSTR	469#
ERMSG	868#
EXT	875#
FLD	156#
FMSG	844#
INCR	664#
IORX	223#
JAND	766#
JE	709#
JN	729#
JNAND	754#
JNOR	782#
JOR	749#
JSERR	858#
JSHLT	863#
JXE	306#
JXF	376#
JXN	328#
JXO	350#
LOAD	591#
MASKB	164#
MOD.	168#
MOVX	174#
MSKSTR	477#
OPSTR	686#
OPSTRM	696#
PERSTR	851#
PGVER.	52#
POINTR	152#
POS	148#
RET	826#
RETSKP	832#
SAVEAC	1180#
SETCMP	649#
SETONE	632#
SETZRO	615#
STKVAR	939#
STOR	602#
TMSG	837#
TQC	802#
TQCA	802#
TQCE	802#
TQCN	802#
TQN	802#
TQNA	802#
TQNE	802#
TQNN	802#

```
TQO         802#
TQOA        802#
TQOE        802#
TQON        802#
TQZ         802#
TQZA        802#
TQZE        802#
TQZN        802#
TRVAR      1011#
TXC         246#
TXCA        246#
TXCE        246#
TXCN        246#
TXN         246#
TXNA        246#
TXNE        246#
TXNN        246#
TXO         246#
TXOA        246#
TXOE        246#
TXON        246#
TXZ         246#
TXZA        246#
TXZE        246#
TXZN        246#
WID         144#
XORX        229#
..CAS1      425#
..DOTX      239#    245     246
..DOTY      795#    801     802
..DPB       605#
..GCNS      579#
..ICNS      568#
..JAN1      770#
..JAN2      773#
..JE        712#
..JN        732#
..JNA3      757#
..JNO1      786#
..JNO2      789#
..LDB       594#
..ONEB      454#
..STR0      485#
..STR1      496#
..STR2      525#
..STR3      530#
..STR4      542#
..STR5      553#
..TQC       652#
..TQO       635#
..TQZ       618#
..TSAC      438#
..TSIZ      430#
```

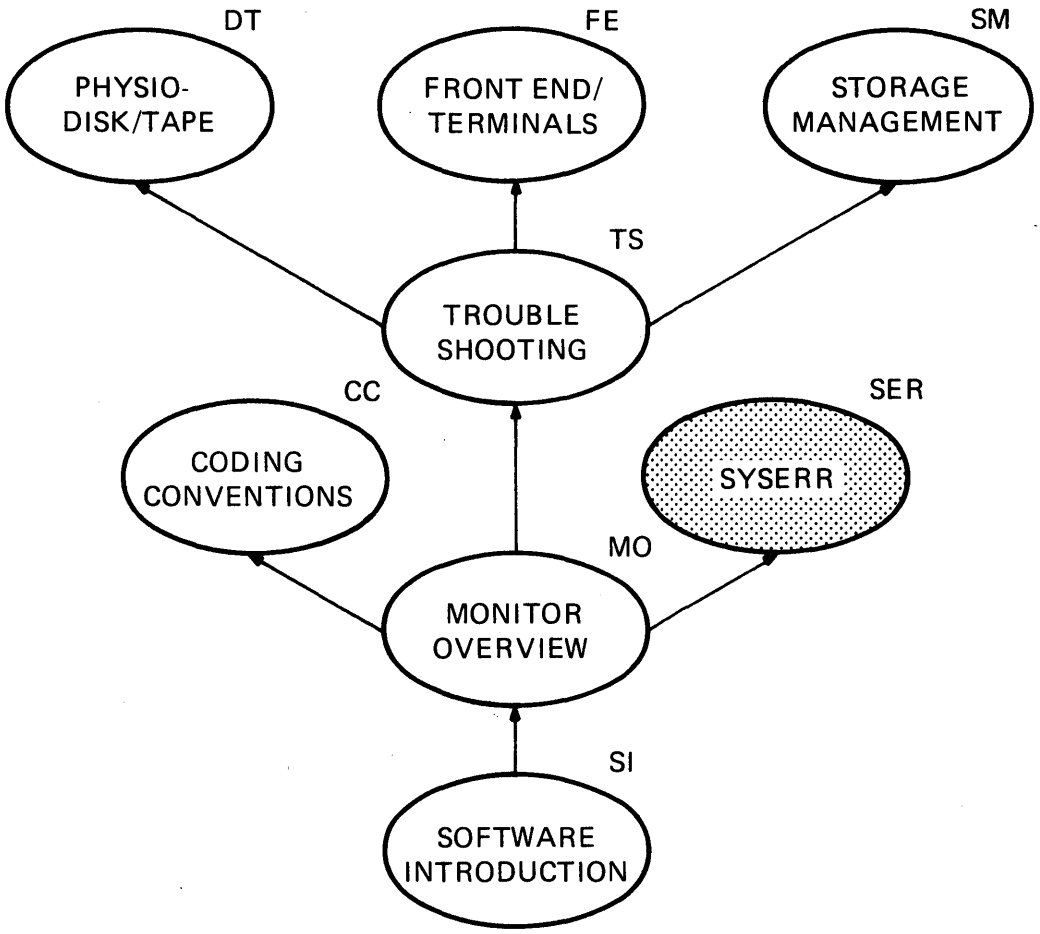
```
..TX      251#
..TX3     288#
..TY      8Ø6#
.ACV1    1110#
.ACV2    1113#
.ACV3    1118#
.CASE     416#
.DECRØ    677#
.IFØ      4Ø6#
.INCRØ    667#
.OPST1    689#
.OPST2    699#
.RTJST    16Ø#
.STKV1    953#
.STKV2    958#
.STKV3    966#
.TRV1    1Ø22#
.TRV2    1Ø25#
.TRV3    1Ø3Ø#
```

TOPS-20 MONITOR

SYSERR

<<For Internal Use Only>>

COURSE MAP



MR-2717

This page is for notes.

SYSERR

INTRODUCTION

SYSERR is the name of a user error reporting program and of a monitor module. The SYSERR program takes a file, ERROR.SYS, and produces a report of the errors indicated by the entries in that file. The monitor SYSERR module contains the code which puts those entries in the ERROR.SYS file. This module addresses both of these aspects of SYSERR, starting with a discussion of the use of the SYSERR program, and continuing with a view of the monitor's internal SYSERR data base. The logic flow of a SYSERR entry creation is presented, and finally, there is a presentation of the tools which allow a privileged user to create ERROR.SYS entries to be reported by the SYSERR program.

LEARNING OBJECTIVES

Upon completion of this module, the student will be able to:

1. Given typical SYSERR output, describe the information contained.
2. Given a sample SYSERR message, tell which data was used in the monitor to generate the report.
3. Use SYSERR to gather information relevant to a specific problem.

RESOURCES

1. TOPS-10 and TOPS-20 SYSERR Manual AA-D533A-TK
2. DECSYSTEM-20 Monitor Calls Reference Manual AA-4166C-TM

MODULE OUTLINE

SYSERR

I. The SYSERR Program
   A. Running the SYSERR Program
   B. Examples of SYSERR Output

II. SYSERR Module Internals
   A. SYSERR Block format
      1. Header
      2. Data

   B. Creating a SYSERR Entry
   C. The Job 0 SYSERR Task
   D. The SYSERR JSYS

This page is for notes.

THE SYSERR PROGRAM

OVERVIEW OF SYSERR

SYSERR is a program which produces reports containing system error information. TOPS-20 collects information such as the time and reason for each monitor reload, error status information for all HARD (non-recoverable) and SOFT (recoverable) errors on devices such as disk and magtape, the details of each BUGINF, BUGCHK and BUGHLT, the occurrence of memory parity errors and the memory locations involved, and console front-end reloads. This data is recorded on disk in file PS:<SYSTEM>ERROR.SYS.

If the TOPS-20 monitor cannot continue due to a BUGHLT, it collects the error information and then halts. During the reload, a dump is taken of the contents of main memory and saved on disk in PS:<SYSTEM>DUMP.EXE. Program SETSPD looks in this crash dump file and extracts the error information. As successive crashes occur, the DUMP.EXE file is copied to successive generations of DUMP.CPY by SETSPD.

If the KL CPU (DECSYSTEM-2040/50/60 only) halts due to some error, the console front-end task KLERR takes a snapshot of the KL CPU. This information is written in a file called KLERR0.SNP in the console front-end files area (usually on the dual-ported disk in the FILES-11 area, but written on floppy disk if it is the front-end device). After the system comes back up, SYSJOB appends this information to ERROR.SYS. In these ways, ERROR.SYS accumulates a history of system errors of various types. Note that you should periodically determine the size of ERROR.SYS. If it is very large (e.g., more than 1000 pages), save ERROR.SYS on magtape, delete and expunge, and let it build up again.

The SYSERR program uses ERROR.SYS as an input file and creates a large variety of reports, depending on the commands you give. Basically, you can select error reports according to device, type of error, amount of detail, and time frame. This is done by specifying switches in commands to SYSERR. You can also obtain an error summary covering all devices and error types for a selected time period.

Running the SYSERR Program

To run SYSERR, type

```
@SYSERR

FOR HELP, TYPE "/HELP"
*
```

When SYSERR is ready for a command, it replies with an asterisk. The command format is

```
*output-filespec = input-filespec/switch1/switch2....
```

where "output-filespec" is the report to be created and the input file is usually SYSTEM:ERROR.SYS. If you do not specify an input file, SYSERR will use PS:<SYSTEM>ERROR.SYS by default. You may need to enable privileges in order to read ERROR.SYS. If you omit the output file specification, the operating system will give the report a default name (determined by the switches) with file type .LST. The switches specify the device or type of error you wish to report. The switches also specify the time frame. Some of the common switches are:

```
/ALL            !List all errors.
/ALLSUM         !List the summary only.
/CPUALL         !List all processor related errors.
/MASALL         !List all MASSBUS device errors (TU45,
                !RP06, etc.).
/DEV:name       !List errors for the specified device only.
                !This allows you to select a particular
                !magtape drive or disk drive.
/DEV:type       !List errors for the specified type of
                !device, for example, TU45, LP20, the CPU,
                !etc.
/DETAIL         !List all information instead of a brief
                !listing.
/BEGIN:mm-dd-yy:hh:mm:ss      !begin the listing on the
                              !date specified. You may
                              !also use the format
                              !/BEGIN:-nD to obtain all
                              !errors for the past n days.
/END:mm-dd-yy:hh:mm:ss        !end the listing on the date
                              !specified. To obtain errors
```

```
!up to n days ago, use the
!format /END:-nD.
```

To obtain a summary of all errors, use the switch /ALLSUM. This report can be obtained and printed each day. It is useful for keeping track of such errors as HARD disk errors, which can indicate a serious problem with a disk pack. Whatever switches are used, the error summary is always a part of each report.

Examples of SYSERR Output

Figures SER-1, SER-2, and SER-3 at the end of this section show examples of output produced by SYSERR. Figure SER-1 shows two entries: a MASSBUS device error and a BUGINF. In the MASSBUS device error entry, the unit name is DP26Ø. This name indicates a drive on channel 2 (RH2Ø controller number 2) with unit number 6. The Ø has no significance for disk. The unit type is RPØ6, the name of the structure SNARK:. LBN stands for the Logical Block Number on the pack being addressed when the error occurred, and is translated to cylinder, surface and sector to provide physical location. The last line indicates that the error was recoverable (SOFT error). The line before the last shows that the operation was retried twice before succeeding. The operation being tried is also indicated following OPERATION AT ERROR:. If an error is not recoverable after a reasonable number of retries, the attempt to retry ceases and the error is classified as non-recoverable (HARD error).

A hard error usually involves loss of data, or failure for some user or system operation. It is important to keep statistics of the hard and soft errors which occur on each disk pack. In this way, you can detect a bad pack or one about to go bad. The physical location of the errors (given in terms of cylinder, surface and sector) can also be used to locate a bad spot (scratch, etc.) on the pack. If errors keep occurring in the same physical location, a bad spot is indicated. If many hard errors occur at once in a variety of locations, the pack may have experienced a head crash.

The second example in Figure SER-1 is for a BUGINF

whose name was DN20ST. The user and program name are useful in determining the cause of the problem, especially in the case of BUGCHKs and BUGHLTs. If the same user and program are specified each time a BUGHLT occurs, a good place to start in investigating the cause of the crashes would be with the user and user program.

Figure SER-2 shows an entry for a MASSBUS device error caused by a tape operation. The device unit name is MT310. This indicates a drive on channel 3 (RH20 controller 3), TM02 unit number 1, the drive being logical unit 0 on the TM02. The position on the tape when the error occurred is given in terms of the file and record numbers. The user and program are also given. In this example, the error was recoverable. This type of entry can be used to track down bad tapes or a malfunctioning drive.

Figure SER-3 shows two portions of an error summary. Under FILE ENVIRONMENT are the input and output file names and the switches used when running SYSERR. The input file used to create this report was SYSTEM:ERROR.SYS. The total number of errors in the categories BUGHLT-BUGCHK, MASSBUS DEVICE and FRONT END DEVICE are given. There is also a breakdown of the BUGHLT-BUGCHK types. The hardware detected error summary for DP260 is another portion of the error summary, showing the total number of such errors, both hard and soft. Similar summaries are given for magtape drives.

For further information about SYSERR and the reports it creates, refer to the DECSYSTEM-20 Operator's Guide and the DECSYSTEM-2020 Operator's Guide

```
SYSTEM ERROR REPORT COMPILED ON Monday, May 8, 1978 14:45:45    PAGE 1
**********************************************
MASSBUS DEVICE ERROR
 LOGGED ON Mon  8 May 78 14:01:20      MONITOR UPTIME WAS 14:29:47
        DETECTED ON SYSTEM # 2102.
        RECORD SEQUENCE NUMBER: 1438.
**********************************************

        UNIT NAME:      DP260
        UNIT TYPE:      RP06
        UNIT SERIAL #:  0597.
        VOLUME ID:      SNARK
        LBN:            1115764  =
        CYL:   794.     SURF:   17.     SECT:   8.
        OPERATION AT ERROR:      DEV.AVAIL., GO +  READ DATA(70)
        FINAL ERROR STATUS:     200000,7
        RETRIES PERFORMED:      2.
        ERROR:  RECOVERABLE DRIVE EXCEPTION,CHN ERROR, IN CONTROLLER CONI
                DCK, IN DEVICE ERROR REGISTER

=====REST OF INFORMATION AVAILABLE BY USING /DETAIL SWITCH=====


SYSTEM ERROR REPORT COMPILED ON Monday, May 8, 1978 14:46:49    PAGE 4
**********************************************
TOPS20 BUGHLT-BUGCHK
 LOGGED ON Mon  8 May 78 12:48:01      MONITOR UPTIME WAS 13:16:28
        DETECTED ON SYSTEM # 2102.
        RECORD SEQUENCE NUMBER: 1415.
**********************************************

ERROR INFORMATION:
        DATE-TIME OF ERROR:     Mon  8 May 78 12:47:57
        # OF ERRORS SINCE RELOAD:       84.
        FORK # & JOB #:         117,0
        USER'S LOGGED IN DIR:   OPERATOR
        PROGRAM NAME:           SYSJOB
        ERROR:          BUGINF
        ADDRESS OF ERROR:       502633
        NAME:                   DN20ST
        DESCRIPTION:            DTESRV- DN20 STOPPED

=====REST OF INFORMATION AVAILABLE BY USING /DETAIL SWITCH=====
```

Figure SER-1. Output from SYSERR

```
SYSTEM ERROR REPORT COMPILED ON Monday, May 8, 1978 14:41:54    PAGE 1

************************************************
MASSBUS DEVICE ERROR
 LOGGED ON Sat  6 May 78 14:56:28      MONITOR UPTIME WAS  7:44:04
        DETECTED ON SYSTEM # 2102.
        RECORD SEQUENCE NUMBER: 917.
************************************************

        UNIT NAME:      MT310
        UNIT TYPE:      TU45
        UNIT SERIAL #:  0148.
        VOLUME ID:
        LOCATION:    RECORD  #  12.    OF FILE #  0.
        USER'S LOGGED IN DIR:   OPERATOR
        USER'S PGM:     EXEC
        OPERATION AT ERROR:      DEV.AVAIL.  GO +  WRITE FWD.(60)
        FINAL ERROR STATUS:     0,3
        RETRIES PERFORMED:      0.
        ERROR:  RECOVERABLE DRIVE EXCEPTION, IN CONTROLLER CONI
                COR/CRC, IN DEVICE ERROR REGISTER

=====REST OF INFORMATION AVAILABLE BY USING /DETAIL SWITCH=====
```

Figure SER-2. Output from SYSERR

```
SYSTEM ERROR REPORT COMPILED ON Monday May 8, 1978 14:46:50      PAGE 23
        - SYSTEM SUMMARY FOR SYSTEM # 2102.
**********************************************************************


FILE ENVIRONMENT
        SYSERR VERSION 10(546)
        INPUT FILES:     SYS:ERROR.SYS  CREATED: Mon  8 May 78  2:06:40PM
        OUTPUT FILE:     DSK:ERR.5
        SWITCHES:        /CPUALL /BEGIN:  8-May-78 AT 12:46:39
        DATE OF FIRST ENTRY PROCESSED:  Tue 18 Apr 77  1:33:36PM
        DATE OF LAST ENTRY PROCESSED:   Mon  8 May 78  2:06:40PM
        NUMBER OF ENTRIES PROCESSED:    1439.
        # OF INCONSISTENCIES DETECTED IN ERROR FILE:    0.

ENTRY OCCURRENCE COUNTS
        TOTAL TOPS20 BUGHLT-BUGCHK:             1.
        TOTAL MASSBUS DEVICE ERROR:             6.
        TOTAL FRONT END DEVICE REPORT:          21.

TOPS20 BUGHLT-BUGCHK
        BUGHLT/BUGCHK BREAKDOWN:
                DN20ST  1.



SYSTEM ERROR REPORT COMPILED ON Monday May 8, 1978 14:46:52
PAGE 24 - MASSBUS SYSTEM ANALYSIS(RH20)


                          HARDWARE DETECTED
              PAR         LWC  SWC  CHN  RES        OVR
              ERR  EXC    ERR  ERR  ERR  ERR  RAE  RUN
DP260 HARD
      SOFT          6.              3.
```

Figure SER-3. Output from SYSERR

SYSERR MODULE INTERNALS

Error reporting, often called SYSERR, is really composed of three steps:

1. A request to generate a SYSERR block is created and queued via the SYERR JSYS or, if generated by the monitor, requested generally by calling the SYERR routines directly.

2. The SYSERR fork (part of Job 0) writes all queued requests to the file PS:<SYSTEM>ERROR.SYS.

3. The SYSERR user program reads the ERROR.SYS file and generates the error reports and summaries.

SYSERR Block Format

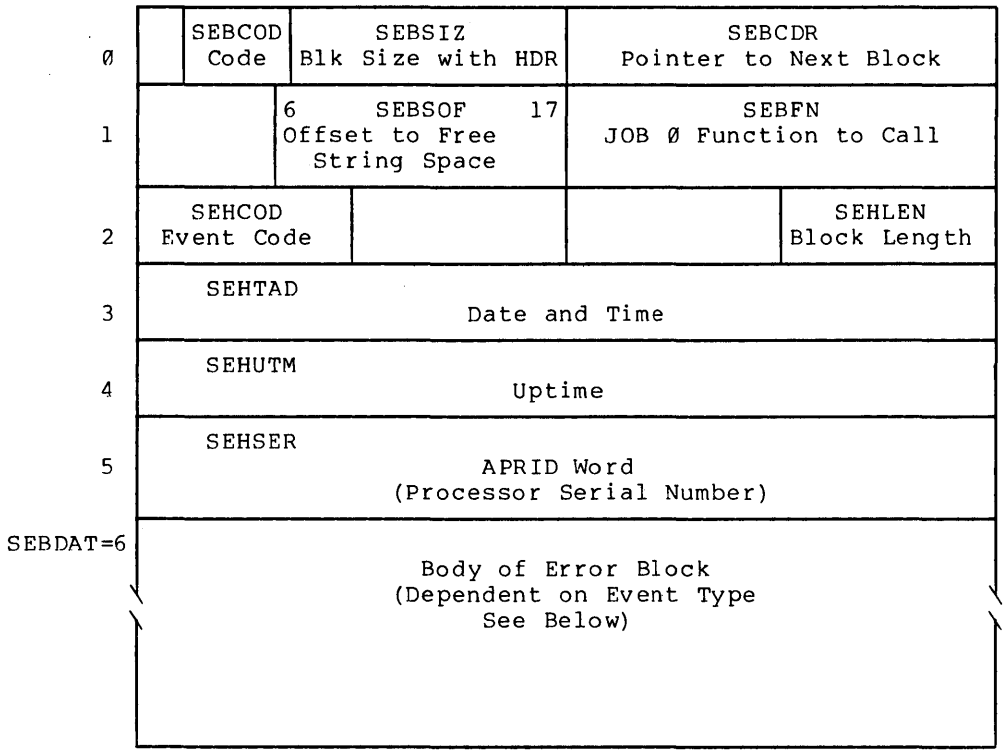
HEADER

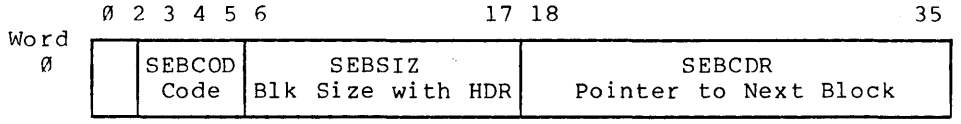
Each entry in the ERROR.SYS file is made up of two parts: the header, which has a fixed length and format (See Figures SER-4 and SER-5), and a data portion, whose content is dependant on the type of event being reported. See Figure SER-6 for an example. Note that the diagram shown in Figure SER-4 and SER-5, the first two words are part of the internal header, but do NOT appear in the ERROR.SYS file.

The fixed information in the header includes the event code (the type given here tells the user SYSERR reporting program the type and format of the data portion of the message), and the block length (not including the two words of monitor header). The block lengths of the standard event types are fixed on an ad hoc basis. The other standard words are Universal format date and time of the blocks creation, the uptime at that point, and APR serial number of the system generating the report.

SEBBFR: SYSERR BUFFER BLOCKS

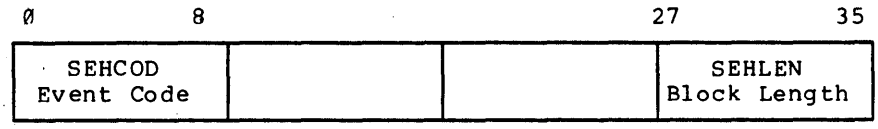
SYSERR BLOCK FORMAT





Bits	Pointer	Meaning
3-5	SEBCOD	State Code SBCFRE=Ø on Free List SBCREL=1 Released SBCACT=2 Active
6-17	SEBSIZ	Block Size Including Header
18-35	SEBCDR	Pointer to Next in List

Figure SER-4. SYSERR Block Format Header



Bits	Pointer	Meaning
0-8	SEHCOD	Event Code (i.e., Block Type) SEC%RL=101 System Reload SEC%BG=102 BUGHLT/BUGCHK/BUGINF SEC%FE=130 Front End Error SEC%11=131 F.E. Reload Entry (Gives -11 Reboot Info.) SEC%PT=160 Processor Parity Trap SEC%PI=161 Processor Parity Intrp. SEC%MB=111 Massbus Device Error
27-35	SEHLEN	Block Length (Including Header) RL%LEN - System Reload Block Length BG%LEN - BUGHLT/CHK/INF Block Length FE%LEN - F.E. Errors Blk Length R1%LEN - F.E. Reload Entry Blk Length PT%LEN - Proc. Parity Trap Blk Length PI%LEN - Proc. Parity Interrupt Blk Lgh MB%LEN - Massbus Dev. Err Blk Length

Figure SER-5. Expansion of Word 2 of Header

DATA

The data portion of an entry is dependent on the event type being reported. When the monitor is generating a SYSERR block, it takes specific information from specific locations and puts that data into the SYSERR block in a pre-defined order. The user SYSERR reporting program is coded to know what information the monitor has placed in which word, and is thus able to format that data in a more meaningful way. The example in figure SER-6 shows a typical data portion.

Event Type 101
System Reloaded Error Block Data

RL%SVN=0	ASCII Byte Pointer to System Name
RL%STD=1	Time of System Build (Univ. Format)
RL%VER=2	System Version Number
RL%SER=3	APR Serial Number
RL%OPR=4	ASCII Byte Pointer to "Why Reload"
RL%HLT=5	BUGHLT Address (if Auto-Reloaded)
RL%FLG=6	Flags
	Monitor Name (Text)
	"Why Reload" Answer String (Text)
RL%LEN=61	

Figure SER-6. Sample SYSERR Block Data

Figures SER-4, SER-5 and SER-6 were taken from the TOPS-20 Monitor Tables, which has diagrams of the several event types data formats. The TOPS-10 and TOPS-20 SYSERR Manual also has descriptions of the ERROR.SYS entry formats.

INTERNAL QUEUE STRUCTURE

The monitor SYSERR blocks are taken from a queue of free blocks reserved for them, and through the calling of various monitor routines, these blocks are queued for the SYSERR fork to write to ERROR.SYS. In referring to Figure SER-4, note that the right half of the first word is reserved for a pointer to the next block (if any). Location

24 of the machine contains a pointer to the first queued block. When the SYSERR fork is awakened, that task checks to see if location 24 (SEBQOU) is non-zero. If there is an address there, that SYSERR block and any other queued SYSERR blocks are appended to PS:<SYSTEM>ERROR.SYS.

NOTE

BUGHLT is a special case where the SYSERR block is created and queued but the SYSERR fork is not started, and the information is not written to PS:<SYSTEM>ERROR.SYS until the system comes up after the crash.

Creating a SYSERR Entry

The following are the steps taken by the monitor in creating a SYSERR block. The calls described are generally called individually by the monitor, but they are the same ones as called by the SYERR JSYS (discussed below).

1. Generate SYSERR entry
    1. Put information in header
    2. Put data in data portion, (immediately following the header)
2. Call the internal monitor routines (or SYERR JSYS)
    1. ALSEB - allocate a SYSERR block
    2. SEBCPY - copy data to block
    3. QUESEB - queue the SYSERR block; that is, put address into SEBQOU or on end of queue of existing blocks. This call also wakes the Job Ø SYSERR task through an AOS @SECHKF.

The Job Ø SYSERR Task

The Job Ø SYSERR task executes code in the module SYSERR at SEBCHK, which checks the queue at SEBQOU (location 24). For each entry in the queue, SYSERR will:

1. Unlink and remove the block from the queue
2. Write the block to PS:<SYSTEM>ERROR.SYS
3. Call RELSEB to release the block for reallocation

BUGHLT is a special case. The BUGHLT code generates and queues a BUGHLT SYSERR block (event type 1Ø2), then shuts the system down. This block is NOT written out to ERROR.SYS at the time of the crash. When the system reloads, SETSPD, a Job Ø task, reads the dump and does a SYERR JSYS for any queued SYSERR blocks. (There will always be a BUGHLT block, but there may also be others which had not yet been written before the crash.) Refer to the Monitor Tables descriptions or the SYSERR Manual for a breakdown of the contents of the BUGHLT SYSERR error block format.

The SYERR JSYS

The SYERR JSYS is available to the privileged user who wishes to make entries in ERROR.SYS for the SYSERR program to report. WHEEL, OPERATOR, or MAINTENANCE capabilities must be enabled to execute the SYERR JSYS. The monitor makes no check of the event code (other than Ø is illegal). When the SYSERR program detects an unknown event code, the output contains a message to that effect and the contents of the block are reported in octal. However, event code 117 (17 on TOPS-1Ø) is defined in the SYSERR program and is intended for this special use. When event code 117 is detected, the SYSERR program labels it as "Software Requested Data" and produces a report with the data as both octal and SIXBIT. The data reported may be gathered by a monitor patch, the PEEK JSYS, the SNOOP JSYS, etc. See the SYSERR Manual under "Software Requested Data", and the Monitor Calls Manual for further information.

This page is for notes.

EXERCISES

1. With the SYSERR Manual and the sheet of sample SYSERR report:

    1. Tell the event type of the error report and the type of error.

    2. Describe how the monitor selected the values reported.

2. Describe how you, as a user, might wish to use the SYERR JSYS.

This page is for notes.

EXERCISE SOLUTIONS

Check with your instructor and your classmates for answers.

This page is for notes.

SYSERR

LAB EXERCISES

When answering the lab exercises, write down the names of the tables where you found the answers. The labs will help you understand the monitor data base structure; so remember, where to look is more important than what you find there.

The exercises marked with a double star (**) are more difficult and are optional. If you have the time and motivation, do them.

Some of the exercises require use of the listings to find the answer; do not assume that the answer is in the tables.

TOOLS

FILDDT

To use FILDDT on a crash, use the GET command instead of the PEEK command. For the following exercise, the crash you are to look at is in a file called <MONITOR-INTERNALS>SYSERR.CRSH. Use FILDDT as in the example below to do the following exercise.

```
@ENABLE
$FILDDT
FILDDT>LOAD <MONITOR-INTERNALS>R3-MONITOR.EXE   ;get symbols
FILDDT>GET <MONITOR-INTERNALS>SYSERR.CRSH
```

At this point, the usual DDT commands allow you to look at the crashed monitor. Note the following things:

1. Only those pages that were in core at the time of the crash are a part of the crash dump.

2. BOOT has overwritten a part of the monitor--currently, it overwrites a part of APRSRV.

3. You must tell FILDDT to use the monitor's page table if you want to look at the monitor's address space. By default, you are looking at physical addresses when you look at a crash with FILDDT. FILDDT knows how to simulate TOPS-20 paging; the command that causes FILDDT to use TOPS-20 paging and which specifies the page map to use is: n$U where n is the SPT slot belonging to the page table FILDDT should use to do the address translation. For most cases, you want FILDDT to use MMAP, which is the monitor's page map for sections 0 and 1. The SPT slot belonging the the monitor's page map is in location MMSPTN. Location MMSPTN contains a 403 for standard monitors; however, you should check to be sure. If the limit on open files has been changed for a monitor, the SPT slot belonging to MMAP is also changed. To set monitor context (i.e., to use MMAP) do the following:

```
MMSPTN/ 403      ;MMAP's SPT slot
403$U
```

Queued SYSERR Blocks

If there were any SYSERR blocks queued to be written at the time of the crash, location SEBQOU= 24 will be the queue header. The right half of the first word of each SYSERR block will contain a pointer to the next block or 0 if there are no more queued blocks. Normally, the BUGHLT block will still be queued up and will be written to ERROR.SYS when the SYSERR fork starts up again. If the system gets a KEEP-ALIVE CEASED, there can be SYSERR blocks left in the queue.

RESOURCES

1. SYSERR related tables in the Monitor Tables.

EXERCISES

1. Beginning at SEBQOU, trace the queue of SYSERR blocks; use the tables to determine if each block is active.

2. What type of block is each queued block?

3. Find a processor parity interrupt error block and match the information stored there with the information the tables say is stored in that block type. **

This page is for notes.

SYSERR

LAB SOLUTIONS

EXERCISES

1. Beginning at SEBQOU, trace the queue of SYSERR blocks; use the tables to determine if each block is active.

   ANSWER:

```
24/ SEBBFR+67

SEBBFR+67/ 20104,,SEBBFR+173

SEBBFR+173/ 20104,,SEBBFR+277

SEBBFR+277/ 20067,,SEBBFR+366

SEBBFR+366/ 20104,,SEBBFR+472

SEBBFR+472/ 20104,,0            ;last queued
                                ;block
```

   Each queued block points to the next queued block; the last queued block has a zero in the right half indicating there are no more queued blocks. Bits 3-5 contain the state code; a value of 2 means active. Each of these blocks is active.

2. What type of block is each queued block?

   ANSWER: Word 2 of each block has the block type in bits 0-8.

```
SEBBFR+67+2/ 102000,,0    ;type= 102 = SEC%BG

SEBBFR+173+2/ 102000,,0   ;type= 102 = SEC%BG

SEBBFR+277+2/ 161000,,0   ;type= 161 = SEC%PI

SEBBFR+366+2/ 102000,,0   ;type= 102 = SEC%BG

SEBBGR+472+2/ 102000,,0   ;type= 102 = SEC%BG
```

Block type SEC%BG labels a block as a BUGHLT/BUGINF/BUGCHK type. Block type SEC%PI labels a block as a process parity interrupt type.

3. Find a processor parity interrupt error block and match the information stored there with the information the tables say is stored in that block type. **

   ANSWER: Use the SYSERR block tables to compare the two.

MODULE TEST

The module test for this module is in two parts. First, obtain a sample SYSERR report from your instructor, and with it (using any available resources) determine where in the monitor each of the reported data came from. You may use either the micro-fiche, the running monitor with FILDDT, or any of the class lab system crash files, (also using FILDDT).

Second, locate and describe the unreported SYSERR entries in a system crash file using FILDDT. See your instructor for the name of the crash file to use.

This page is for notes.

TEST EVALUATION SHEET

The results of these problems will be discussed in class after the laboratory session.

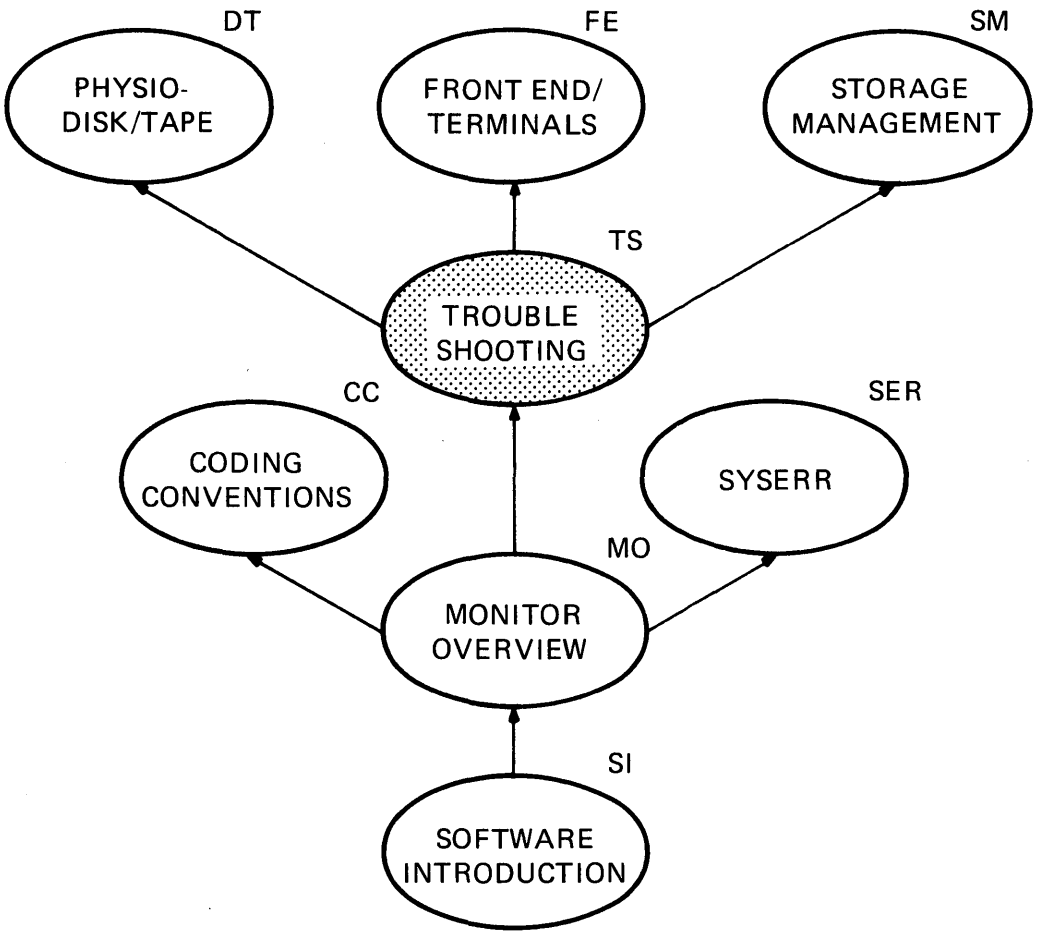
This page is for notes.

TOPS-20 MONITOR

Troubleshooting

<<For Internal Use Only>>

COURSE MAP



MR-2717

This page is for notes.

Troubleshooting

INTRODUCTION

This module contains information on debugging and crash analysis, with the final section devoted to MDDT, EDDT, and FILDDT.

There is a great difference between analyzing a crash and debugging one. While analysis simply tells you what happened, debugging gives you reasons why (and thus, implies remedies).

Successful crash analysis depends heavily on how well you know the data base and whether you can discover inconsistencies that give you clues about what happened. The information in this module shows you how to use available tools in looking at a crash and how to find basic information about the state of the machine at the time of the crash. Further analysis of a crash requires that you are able to propose a reason for what happened that matches the state of the data base.

LEARNING OBJECTIVES

Upon completion of this module, the student will be able to:

1. Identify the mode of the operating system at the time of a crash.
2. Determine which, if any, fork was running at that time.
3. Deduce, from the stack, which of the save macros put what on the stack.
4. Elicit a dump.
5. Extract saved SYSERR blocks from a crash.
6. Designate the relevent portions of the data base.

RESOURCES

DECsystem-10/DECSYSTEM-20 Hardware Reference Manual

MODULE OUTLINE

Troubleshooting

I. CTY Output
   A. Explanation of KLERR Output
   B. Sample KLERR Output

II. Getting a DUMP
   A. How To Get a Dump
   B. Where BOOT Lands

III. SYSERR
   A. Overview of SYSERR Functions and Data Base
   B. Queued SYSERR Blocks In A Crash
   C. Moving SYSERR Blocks From a Crash To ERROR.SYS

IV. BUGHLT
   A. BUG Macro
   B. BUGHLT Contents

V. Push Down Lists And Related Data Bases
   A. How To Look At a Stack
   B. Push Down List / Machine State
   C. Stack Usage For Local Storage
   D. Stack Adjustment

VI. Machine States and Relevant Data Bases
   A. PC Storage
   B. AC Storage
   C. Fork Scheduled, Or Not
   D. Fork NOSKED
   E. Extended vs. Non-extended Addressing
   F. Sizes (Resident, Non-resident, Total)
   G. MDDT Page
   H. Relevant Data Base for Each Machine State

VII. DDT's
   A. FILDDT
   B. Relevant DDT/FILDDT Commands
   C. MDDT
   D. EDDT

This page is for notes.

CTY OUTPUT

Collect any CTY output that is relevant to the crash. This should include the KLERR printout and the BUGHLT (as well as recent BUGCHKs and BUGINFs, if any). If the machine got a KEEP-ALIVE CEASED, the KLERR output is the only reliable information you get. (See the section on the BUGHLT location for an explanation.)

Explanation of KLERR Output

KLERR includes the PC, the last memory fetch and information on the PI system. The PI information includes the following:

PI STATE: ON or OFF	-- indicates whether the PI system is on or not.
PI ON: n	-- n indicates which of the 7 channels are enabled.
PI HLD: n	-- n indicates which of the 7 channels have an interrupt in progress.
PI GEN: n	-- n indicates which of the 7 channels have a pending interrupt.

Sample KLERR Output

Here is an example of the KLERR output on a KEEP-ALIVE CEASED error:

```
%DECSYSTEM-20 NOT RUNNING


KEEP ALIVE CEASED
KLERR -- VERSION V02-02 RUNNING
```

```
KLERR -- KL NOT IN HALT LOOP
KLERR -- KL ERROR OTHER THAN CLOCK ERROR STOP
KLERR -- KL VMA: 000000 035717    PC: 000000 035717
KLERR -- PI STATE: ON, PI ON: 177, PI HLD: 004, PI GEN: 001
KLERR -- EXIT FROM KLERR
```

GETTING A DUMP

DUMP.EXE is a pre-allocated file into which BOOT writes the dump. When the system comes up, SETSPD copies DUMP.EXE to DUMP.CPY.

How to Get a Dump

If the system does an auto-reload, the console front end will give BOOT the commands to get a dump. If the auto-reload does not work for some reason, you can force a dump by typing /d to the BOOT> Prompt.

```
KLI -- VERSION VB06-07 RUNNING
KLI -- ALL CACHES ENABLED
KLI -- BOOTSTRAP LOADED AND STARTED
?DUPL STR UNI?DUPL STR UNI
                                ;problem because two PS:
                                ;structures on line.
BOOT>/d                         ;request a dump (after the
                                ;problem is corrected).

BOOT>                           ;type CR for default monitor
```

Where BOOT Lands

The console front end loads BOOT into KL memory. Of course, this overwrites whatever used to be in that part of memory. Therefore, BOOT is always loaded into a part of the monitor that contains pure code (i.e., so no data is destroyed). Currently, BOOT is brought in on top of a part of APRSRV. If you need to look at code that is loaded where BOOT lands, you must go to the listings. BOOT also uses some of high core to build the EXE directory for the file DUMP.EXE; for example, on a machine with 256K, no page above 761 is dumped.

SYSERR

Overview of SYSERR Functions and Data Base

SYSERR is a program that reads PS:<SYSTEM>ERROR.SYS and generates reports on hardware errors, system crashes, front end reloads, etc. Entries in the file ERROR.SYS are written via the SYERR JSYS. For a description of each of the types of entries, see the monitor tables.

When an ERROR.SYS entry is desired, the caller (which is one of the system programs such as QUASAR or the monitor itself) builds a SYSERR block as described in the monitor tables and does the SYERR JSYS. The SYERR JSYS adds the block to a queue in the monitor's address space; the queue header is SEBQOU (location 24). It then wakes up the Job Ø task which processes the queue and writes the queued entries to ERROR.SYS.

Queued SYSERR Blocks In a Crash

A BUGHLT entry is generated when the system BUGHLTs; this entry is queued but not written to ERROR.SYS. The system is considered to be in an unsafe state at the time of the crash. When the system comes back up, code in SETSPD is called to move any queued SYSERR blocks in the dump to ERROR.SYS.

The queue header is location 24 (called SEBQOU); each SYSERR block consists of the standard SYSERR header followed by the information in the specific block type.

Sometimes it is useful to look at the queued SYSERR blocks in a crash, particularly the BUGHLT block. The BUGHLT block contains certain status information at the time of the crash. The status words are described below:

1. CONI APR,

   Read the status of the processor error and sweep flags. This information is stored in offset BG%APS of the BUGHLT block. The flags and status information returned by a CONI APR are described in the Hardware Reference Manual.

2. CONI PAG,

   This information is stored in offset BG%PGS of the BUGHLT block. A CONI PAG reads the system status of the pager. If TOPS-20 paging is on, there is a 1 in bit 21. Bits 23-35 contain the contents of the EBR (the address of the EPT). For a description of all the fields, see the Hardware Reference Manual.

3. DATAI PAG,

   This information is stored in offset BG%PGD of the BUGHLT block. A DATAI PAG returns the process status of the pager. DATAI PAG, returns the current and previous contex AC blocks, and the address of the UPT. For a complete description of the fields returned by a DATAI PAG, see the Hardware Reference Manual.

4. CONI PI,

   This information is stored in offset BG%PIS of the BUGHLT block. A CONI PI returns the status of the priority interrupt system; it indicates which levels are on, whether the PI system is on, and on which levels interrupts are currently being held. For a complete description, see the Hardware Reference Manual.

Moving SYSERR Blocks From a Crash to ERROR.SYS

As stated before, SETSPD moves queued SYSERR blocks from the crash to ERROR.SYS. A Job 0 task starts the SETSPD program at START3; this code copies DUMP.EXE to DUMP.CPY and then issues a SYERR JSYS for each queued SYSERR block in the crash.

BUGHLT

Location BUGHLT contains the location the BUGHLT came from. The latter contains an XCT BUGHLT-name. All BUGHLT code is generated by the BUG macro defined in PROLOG.

BUG Macro

```
DEFINE BUG(TYP,TAG,STR,REGS,%NAM,%STR)<
        XCT [TAG::      JSR BUG'TYP
                IRP REGS,<
        Z REGS>
        SIXBIT /TAG/]
        .PSECT BGSTR
%STR:   ASCIZ \STR\
        .ENDPS BGSTR
        .PSECT BGPTR
        XWD TAG,%STR
        .ENDPS BGPTR
>
```

This is an example of a call to the BUG macro:

```
        BUG(HLT,JØNRUN,<JOB Ø NOT RUN FOR TOO LONG,...
                ...PROBABLE SWAPPING HANGUP>)
```

If a JØNRUN BUGHLT occurred, the data base would look like this:

```
        BUGHLT/ CAIA CLK2+6     ;address the BUGHLT came from

        CLK2+6/ XCT JØNRUN      ;generated by BUG macro

        JØNRUN/ JSR BUGHLT
              / SIXBIT \JØNRUN\
```

BUGHLT Contents

Again, location BUGHLT is set up with the location the BUGHLT came from if the machine BUGHLTed. That location contains an XCT BUGHLT-name. All the BUGHLTs are listed in the Operator's Guide with a short descriptive phrase. (Appendix I of this course contains a list of all BUGHLTs, BUGCHKS and BUGINFs.)

If there is a zero in location BUGHLT, the machine probably got a KEEP-ALIVE CEASED. This happens if either a "clock error stop" or "deposit/examine failure" occurs. Both of these errors are hardware failures and Field Service should be called. Although these two errors are the probable causes of KEEP-ALIVE CEASED, you cannot rule out the possibility of a software bug. Get the KLERR output from the CTY. It will have the PC and PI state.

Because the console front end simply reloads for a KEEP-ALIVE CEASED, the information in the dump is not dependable because the cache has not been written out, the ACs have not been saved, etc. (These functions are normally done by the BUGHLT code.) The only valid information is the CTY output from KLERR.

PUSH DOWN LISTS AND RELATED DATA BASES

In general, the pushdown list in use at the time of the crash implies what was going on. For example, if the scheduler was running, SKDPDL is the push down list. If a page fault was in progress, TRAPSK is the push down list, and the former P is saved.

How to Look at a Stack

1. P contains the current stack pointer.

2. If an entry was made on the stack by a PUSHJ, the entry will look like a PC. This is not a hard and fast rule, but it can help. A user mode PC usually has bits 1,2, and 3 on and a monitor PC has bits 1 and 2 on.

3. If a return address is still on the stack (i.e., the entry is at an address less than the stack pointer), you have not returned from the routine.

4. The monitor uses the stack for temporary storage. The macros STKVAR, TRVAR, etc. leave recognizable entries on the stack. Knowing these conventions helps you recognize which stack locations are being used as temporary storage.

Push Down List/Machine State

The monitor uses different stacks to do different things. Register P is the stack pointer and indicates which stack was in use at the time of the crash. The stacks and their uses are listed below:

1. UPDL -- Used when running in Exec mode for the user, that is, when doing a JSYS. Also used by the Job Ø tasks that run in exec mode.

2. TRAPSK -- Used for page fault handling.

3. PIPDB -- Used for software interrupt handling.

4. SKDPDL -- Used by the scheduler for the overhead cycle.

5. DTESTK -- DTE interrupt level stack (PI level 6).

6. PHYPDL -- Used by PHYSIO when queueing an IORB.

7. PHYIPD -- Used when PHYSIO is handling an interrupt.

8. MEMPP -- Used when handling APR interrupts.

Stack Usage for Local Storage

Several macros that provide local storage use the stack. What they put on the stack is usually recognizable. (See MACSYM.MEM for further information.)

1. STKVAR

   STKVAR uses the stack as temporary storage; the local variables have names that are really stack locations. STKVAR uses n stack locations for local variables (where n is the number of local variables requested) a count of local variables, the return address .STKRT. On the stack you will see:

```
/ local variable
/ local variable
/ .
/ .
/ local variable n
/ n,,n
          ;count of local variables (used to
          ;adjust the stack)
/ .STKRT ;routine to clean up the stack and
          ;return
```

Therefore, when you find .STKRT on the stack, the word before it is the count of local variables which tells you how many locations on the stack are in use by STKVAR.

2. TRVAR

TRVAR uses the stack in much the same way as STKVAR does, but it also uses AC15, the current contents of which is pushed on the stack first. The stack locations it uses look like this:

```
/ AC15
/ local variable
/ local variable
/ .
/ .
/ local variable n
/ n,,n
/ .TRRET
```

Therefore, when you find .TRRET on the stack, the word before it is the count of local variables, with register 15 stored on the stack in front of the local variables.

3. ASUBR

ASUBR saves AC15, ACs 1-4, followed by the return address .ASRET, which is a routine to clean up the stack. When you see the address .ASRET on the stack, you can expect the following in this part of the stack:

```
/ AC15
/ AC1
/ AC2
/ AC3
/ AC4
/ .ASRET
```

4. ACVAR

ACVAR can save AC5, AC5 and AC6, AC5-AC7, AC5-AC1Ø, or AC5-AC14, depending on the arguments given. In each case, the return address to clean up the stack is the last item pushed on the stack by the ACVAR macro; the return address stored on the stack is the clue to what else was pushed on the stack. Each of the possible cases is listed below:

1. AC5 saved

```
/ AC5
/ .SAV1+2          ;return address
```

2. AC5 and AC6 saved

```
/ AC5
/ AC6
/ .SAV2+3          ;return address
```

3. AC5, AC6, AC7, and AC1Ø saved

```
/ AC5
/ AC6
/ AC7
/ .SAV3+4          ;return address
```

4. AC5, AC6, AC7, and AC1Ø saved

```
/ AC5
/ AC6
/ AC7
/ AC1Ø
/ .SAV4+5          ;return address
```

5. AC5 through AC14 saved

```
/ AC5
/ AC6
/ AC7
/ AC10
/ AC11
/ AC12
/ AC13
/ AC14
/ .SAV8+7          ;return address
```

5. SAVEAC

SAVEAC takes a list of ACs to be saved as an argument. It pushes the list of ACs on the stack, followed by the address of a literal which is the routine that restores the stack. One of the instructions in the literal does a SUB P,[.NAC,,.NAC]. This macro does not leave easily recognizable data on the stack, but if you find a return address on the stack that is a literal that does the following, SAVEAC was used. (If you look at the code in the literal, you will be able to tell which ACs were pushed on the stack and how many there were. .NAC is the count of ACs pushed.

```
/ AC
/ AC
/ .
/ .
/ last AC saved
/ address of literal to restore stack
```

The literal to restore the stack looks (approximately) like this:

LIT= address of literal to restore stack for this example.

```
LIT-1/ 3,,3      ;count of ACs saved =3
LIT/ CAIA 0
   / AOS -N(P)
   / MOVE 1,-2(17) ;restore AC1
   / MOVE 5,-1(17) ;restore AC5
   / MOVE 10,0(17) ;restore AC10
   / SUB 17,LIT-1  ;reclaim stack locations
   / POPJ P,       ;return to callee
```

6. SAVEP

This macro calls the routine SAVP (in APRSRV) to save the ACs P1-P6 on the stack, followed by the address RESTP, which is the routine to restore the ACs.

```
/ P1
/ P2
/ P3
/ P4
/ P5
/ P6
/ RESTP
```

7. SAVEQ

This macro calls the routine SAVQ (in APRSRV) to save the ACs Q1-Q3 on the stack, followed by the address RESTQ, which is the routine to restore the ACs.

```
/ Q1
/ Q2
/ Q3
/ RESTQ
```

8. SAVEPQ

This macro calls the routine SAVPQ (in APRSRV) to save the ACs Q1-Q3 and P1-P6 on the stack, followed by the address RESTPQ, which is the routine to restore the ACs.

```
/ Q1
/ Q2
/ Q3
/ P1
/ P2
/ P3
/ P4
/ P5
/ P6
/ RESTPQ
```

9. SAVET

This macro calls the routine SAVT (in APRSRV) to save the ACs T1-T4 on the stack, followed by the address RESTT, which is the routine to restore the ACs.

```
/ T1
/ T2
/ T3
/ T4
/ RESTT
```

Stack Adjustment

Many times the stack pointer is adjusted. Table BHC, indexed by n, contains n,,n which may be added to or subtracted from the stack pointer.

MACHINE STATES AND RELEVANT DATA BASES

PC Storage

1. PC at the time of the crash.

   Location BUGHLT contains the PC at the time of the crash.

2. PC when JSYS began.

   Two copies of the PC are saved on the stack.

3. PFL/PPC

   Current PC of process when the process was last context switched. May be either an exec or user mode PC.

4. PIFL/PIPC

   The exec mode PC is saved here while the software interrupt code is in progress.

5. Temporary PC storage

   When the system is changing state, it must always be prepared for a context switch. This is a concern when a JSYS is starting, when a process blocks, and when a software interrupt begins. In each case, the PC is temporarily stored in case of a context switch while the state change is in progress.

   1. SKDFL/SKDPC - PC is saved here while process is blocking.

   2. MONFL/MONPC - PC is saved here while the nested JSYS is starting.

   3. ENSKR/ENSKR+1 - PC is saved here while it is entering the scheduler via the ENTSKD macro. This is the PC the ENTSKD macro was called with.

AC Storage

AC STORAGE IN THE PSB

Each process's PSB contains several storage areas for saving ACs. ACs are saved in the PSB in these cases:

1. Nested JSYS (JSYS called by a JSYS).

   When a user called JSYS is in progress, AC block Ø contains the monitor's ACs (the current JSYS code ACs) and AC block 1 contains the user mode ACs. If the JSYS code does a JSYS, AC block 1 (user mode ACs) are saved in the UACB area and the AC block Ø ACs are moved to AC block 1. For each level of JSYS, the AC block 1 ACs are pushed onto the UACB stack and the AC block Ø ACs are moved to AC block 1. Therefore, the AC block 1 ACs are always the previous context ACs; i.e., the ACs when the JSYS was called.

   If a nested JSYS is in progress, the user mode ACs are the first stacked ACs in UACB. If the nesting is more than one level deep, each subsequent JSYSs calling ACs are also saved in UACB; the current JSYS ACs are saved in UAC if the process is not currently running, or in BUGACU if the process was running at the time of the crash. The maximum nesting level for JSYSs is 5; this limit is dependent on how much storage is reserved for AC stacking in UACB.

   ACBAS is the "pointer" for the AC stack UACB, but is not stored as an address. The contents of ACBAS must be shifted left 4 places to make it an address. The resulting address is the first saved AC for the last pushed AC block (i.e., the saved ACs for the next higher level of nesting). If there are no saved ACs pushed on the stack, ACBAS contains its initial value of <UACB>B39-1=37677; if ACBAS contains anything else, there are pushed AC blocks saved in UACB.

2. Process is context switched while running in user mode.

   The current ACs (i.e., the user mode ACs) are saved in block UAC.

3. Process is context switched while running in exec mode.

   The current ACs are saved in block PAC. The previous context ACs are saved in block UAC. The ACs saved in UAC are the user mode ACs unless a nested JSYS is in progress; in this case, the ACs saved in UAC are the ACs the nested JSYS was called with. The user mode ACs for this case are saved on the AC stack called UACB; the user mode ACs are the first saved ACs on UACB.

4. Software interrupt processing.

   The exec mode ACs are saved in block PIAC while a software interrupt is in progress.

AC STORAGE AT THE TIME OF THE CRASH

1. BUGACS

   Exec mode ACs at the time of the crash, copied to current ACs when using FILDDT.

2. BUGACU

   Previous context ACs at the time of the crash. These are the user mode ACs unless a nested JSYS was in progress, i.e., if a JSYS called from a JSYS. If a nested JSYS was in progress at the time of the crash, BUGACU contains the ACs the current JSYS was called with. In such a case, the user mode ACs are saved in the AC stack called UACB.

SUMMARY OF AC STORAGE

1. UAC

   Previous context ACs are saved here when the user is context switched. For the currently scheduled process, UAC contains the ACs from the last time the process was dismissed. Once again, if a nested JSYS was in progress, the UACs contain the ACs the JSYS was called with. In such a case, the user mode ACs are saved in the AC stack called UACB.

2. UACB and ACBAS

   Pushed AC blocks when a nested JSYS is in progress.

3. PAC

   Exec mode ACs saved here for process when it is dismissed.

4. PIAC

   Exec mode ACs saved here when a software interrupt is in progress.

5. BUGACS

   Exec mode ACs at time of crash.

6. BUGACU

   Previous context ACs at time of crash.

Fork Scheduled, or Not

If a fork is scheduled, location FORKX contains the fork's system fork number. The scheduled fork's PSB and per-process pages, JSB and per-job storage, and the page table are all mapped into the monitor's address space. If no fork is currently scheduled, location FORKX contains a -1.

Fork NOSKED

If a fork is NOSKED, its fork number is stored in SSKED; if there is no NOSKED fork, SSKED contains -1.

Extended vs. Non-Extended Addressing

If the machine supports extended addressing, flag EXADDR contains a 1; if the machine does not support extended addressing, EXADDR contains Ø.

Sizes (Resident, Non-Resident, Total)

MONCOR/ number of pages in resident monitor

TOTRC/ total number of swappable core pages

NHIPG/ highest physical core page number

MDDT Page

When MDDT is in use for a process, DDTPPG (currently page 774) exists. If the running process's page 774 exists, that process has been using MDDT. (You might suspect the crash was caused by an accidental deposit in MDDT, for example.)

Relevant Data Base for Each Machine State

JSYS

1. Stack

   UPDL

2. Initial stack setup

   Initial UPDL setup for JSYS if from user mode:

   / PC at time of JSYS
   / PC flags at time of JSYS

```
/ PC at time of JSYS
/ PC flags at time of JSYS
```

Initial UPDL setup for JSYS if from exec mode (nested):

```
/ INTDF
/ MPP (for higher level JSYS)
/ PC at time of JSYS (return PC)
/ PC flags at time of JSYS
```

3. Previous PC

The return PC is pushed on to the stack; MPP is the stack pointer for the return PC.

4. Saved ACs

ACs saved in UACB if the JSYS is nested. See section on AC STORAGE for a description of UACB.

5. Saved stack pointer

If the JSYS was from user mode, this is not relevant. If the JSYS is nested, the previous JSYS also used this stack and the MPP pointer can be used to determine where the stack pointer was when this JSYS began:

```
Previous
stack ptr-> /
            / INTDF
            / MPP (for higher level JSYS)
            / MONPC
   MPP ->   / PC at time of JSYS (return PC)
```

6. AC usage

There is no standard for AC usage to which all JSYSs conform.

7. Related storage

   1. MPP -- points to:

      --return PC for JSYS

      --last location of initial setup for this JSYS

   2. FPC = KIMUPC -- dispatch address for JSYS

   3. KIMUU1 -- last UUO from user in format:

      KIMUU1/ flags,,opcode
            / JSYS number

   4. INTDF

      Indicates if the process is NOINT, and to how many levels. Set to -1 if the process is not NOINT, greater than or equal to zero if the process is NOINT.

PAGE FAULT

1. Stack

   TRAPSK

2. Initial stack setup

   The initial stack setup differs for each of three cases:

   --page fault from user mode

   --page fault from exec mode

   --recursive page fault

1. Stack setup upon page fault from user mode:

```
/runtime
/return PC
/return PC flags
```

2. Stack setup upon page fault from exec mode:

```
/ AC1
/ AC2
/ AC3
/ AC4
/ AC7
/ AC16
/ TRAPSW
/ runtime
/ PC
/ PC flags
```

3. Stack setup upon recursive page fault:

```
/ AC1
/ AC2
/ AC3
/ AC4
/ AC7
/ AC16
/ TRAPSW
/ PC
/ PC flags
```

3. Previous PC

Saved on stack; see initial stack setup for location.

4. Saved ACs

The ACs that are saved are kept on the stack. See the initial stack setup to learn where each AC is saved.

5. Saved stack pointer

   TRAPAP

6. AC usage

   Differs for each type of page fault.

7. Related storage

   1. TRPID --identity of the page causing the trap in the form PTN,,PN or PTN

      This is the identity of the page the page fault handler is working on. TRPID contains the page's page table identity while the page's page table is brought into core (if the page table was not in core).

   2. TRPPTR

      Storage address of the page the page fault handler is working on.

   3. TRAPSW (copy of TRAPSØ)

   4. TRAPC

      Ø if the first level page fault; if greater than Ø, it indicates the level of recursion.

   5. TRAPFL/TRAPPC = UPTPFL/UPTPFO

      Flags and PC at time of page fault.

   6. TRAPSØ = UPTPFW

      Page fail word; contains the ADDRESS that page faulted.

SCHEDULER

1. Stack

   SKDPDL

2. Initial stack setup

   None.

3. Previous PC

   Saved in PSB for a process upon a context switch to the scheduler; the PC is saved in PFL/PPC of the process's PSB. If FORKX contains a fork number, it is the number of the fork running when the scheduler was invoked. If FORKX is not set up, you cannot determine which fork was running last.

4. Saved ACs

   The process's ACs are saved in block PAC (exec mode ACs) and block UAC (previous context ACs) of the PSB for the process. If FORKX is not setup, you cannot determine which process was running last.

5. Saved stack pointer

   In the saved ACs.

6. AC usage

   FX/ -1 if no fork was chosen, or the system fork number of the chosen fork

7. Related storage

   1. FORKX - FORKX contains a -1 if no fork is chosen, or the fork number of the chosen fork.

   2. Temporary storage while entering scheduler.

PHYSIO QUEUEING LEVEL

1. Stack

   PHYPDL

2. Initial stack setup

   The P and Q ACs are saved on the stack by the macro SAVEPQ; the ACs are saved in order of Q1 through Q3 followed by P1 through P6. See the section on Stack Usage for Local Storage for the format.

3. Previous PC

   Since PHYSIO is called with a PUSHJ, the previous PC is the top of the saved stack.

4. Saved ACs

   ACs Q1-Q3 and P1 through P6 are saved on the stack. See the section in Stack Usage for Local Storage entitled SAVEPQ.

5. Saved stack pointer

   The previous stack pointer is saved in PHYSVP.

6. AC usage

   P4/ address of IORB being queued

   P1/ address of CDB

   P3/ address of UDB

   P2/ address of KDB or Ø if no KDB

7. Related storage

   None.

PHYSIO INTERRUPT LEVEL

1. Stack

   PHYIPD

2. Initial stack setup

   None.

3. Previous PC

   The previous PC is saved by the XPCW instruction in a two word block, beginning at the CDB-6.

4. Saved ACs

   PHYACS -- block where ACs saved

5. Saved stack pointer

   The saved stack pointer is in PHYACS+17.

6. AC usage

   P1/ address of CDB

   P2/ address of KDB or 0 if none

   P3/ address of UDB

   P4/ IORB address or argument indicating action code:

   P4<0 schedule a channel cycle (P4) = -1
   P4=0 dismiss interrupt
   P4>0 housekeep current request (contains IORB address)

7. Related storage

   Home block check funtion. In STG, starts at CHBUDB.

APR INTERRUPT LEVEL

1. Stack

   MEMPP

2. Initial stack setup

```
/UPTPFO= TRAPPC
/UPTPFL= TRAPFL
/UPTPFN
/UPTPFW= TRAPSØ
```

3. Previous PC

   Saved as a double-word PC by XPCW in locations PIAPRX and PIAPRX1.

4. Saved ACs

   MEMPA -- block where ACs Ø-1Ø are saved.

NOTE

Release 3A uses a different AC block while at APR interrupt level; therefore, no ACs are saved.

5. Saved stack pointer

   MEMAP -- previous stack pointer saved there.

6. AC usage

   None.

7. Related storage

   1. Sets "local" page fail routine to MEMPTP.

DTE INTERRUPT LEVEL

1. Stack

   DTESTK

2. Initial stack setup

   None.

3. Previous PC

   Saved in DTETRA.

4. Saved ACs

   DTEACB -- block where ACs are saved.

5. Saved stack pointer

   Previous stack pointer is saved in DTEACB+17.

6. AC usage

   F/ result of CONI DTEn,

   A/ DTE number of DTE that caused interrupt

   3/ count (if RSX20F protocol)

   4/ Byte pointer (if RSX20F protocol)

7. Related storage

   None.

PSI HANDLING

1. Stack

   PIPDB

2. Initial stack setup

   None.

3. Previous PC

   PIDL/PIPC

4. Saved ACs

   PIAC -- block where ACs are saved.

5. Saved stack pointer

   Previous stack pointer is saved in PIAC+17.

6. AC usage

   FX/ interrupt flags from FKINT

7. Related storage

   None.

JOB Ø EXEC MODE TASKS

1. Stack

   UPDL

2. Initial stack setup

   None.

3. Previous PC

   Since these are scheduled processes, this is not relevant.

4. Saved ACs

   Since these are scheduled processes, this is not relevant.

5. Saved stack pointer

   Since these are scheduled processes, this is not relevant.

6. AC usage

   None.

7. Related storage

   1. How can you tell this use of UPDL from a JSYS?

      If the FKJOB entry for the running fork is Job 0, the current process is probably a Job 0 task as opposed to a JSYS in progress. If the PC is in a Job 0 routine, this also indicates a Job 0 task.

USER MODE

The system never BUGHLTs in user mode, but it could KEEP-ALIVE CEASE. The PC is from user mode if the flag UMODF is set in the PC.

DDT's

FILDDT

The latest version of DUMP.CPY is the last crash. The program FILDDT is used to analyze a crash.

HOW TO USE FILDDT ON A CRASH

To look at a crash with FILDDT you need the dump and the monitor file it came from (for symbols). For example:

```
@ENABLE
$FILDDT
FILDDT>LOAD <SYSTEM>MONITR.EXE  ;load symbols
FILDDT>GET <SYSTEM>DUMP.CPY     ;load dump
```

The ACs contain their contents at the time of the dump. By default you look at physical (not virtual) addresses.

$U COMMAND

FILDDT can simulate KL paging. If you want to look at a particular address space, use the n$U (altmode U) command. The n is the address space's page table's SPT slot. Usually, you wish to look at the monitor's address space. MMAP's SPT slot is in location MMSPTN (usually it is 403, but you should check the contents of MMSPTN if you are looking at an unfamiliar monitor). In the part of the monitor that BOOT loads, there is a one-to-one correspondence between physical and virtual addresses; MMSPTN is in this part of the monitor's address space.

If you wish to look at some fork's address space, find its page table's SPT slot in the left half of FKPGS, indexed by fork number.

If you wish to return to physical addressing (i.e., no KL paging simulation), type $U (no n argument).

Relevant DDT/FILDDT Commands

These are standard DDT commands; however, you may not be familiar with them. They are included here along with examples of their use.

QUESTION MARK (?)

If you type a symbol followed by a question mark, DDT tells you which module(s) that symbol appears in; the module name is followed by a G if the symbol is global. A local symbol may be defined in more than one module.

This facility can be used to locate symbols, like GLOB, but faster.

```
BUGSTO?
APRSRV  ;symbol is local and defined in  APRSRV

SPT?
STG G   ;symbol is global and defined in STG
```

UNDERSCORE

A value followed by an underscore is a request to DDT to find a symbol with that value.

This facility can be used to locate the symbolic address of a value.

```
14156_LSCHED+5

1Ø14ØØ_SPT
```

EFFECTIVE ADDRESS SEARCH ($E)

The $E command is used to search for all locations where the effective address, following all indirect and index-register chains to a maximum length of 64 (base 1Ø) equals the address being searched for.

The format of the command is a<b>c$E; a<b> is the range and is optional. If no range is specified, the whole address space is assumed. The c argument is the address to search for.

```
MMSPTN$E
PGRI10+3/ MOVEM T1,MMSPTN
FPTA4/ SKIPA T1,MMSPTN
MLKPGM+2/ CAMN T2,MMSPTN
SWPER3+2/ CAMN T2,MMSPTN
GSMLER+11/ HRL T1,MMSPTN
BSMGP1+2/ HRL T1,MMSPTN
212777/ HRL T1,MMSPTN
SNPF0A+15/ HRL T1,MMSPTN
SNPF5B+10/ HRL T1,MMSPTN
UT1LL+1/ HRL T1,MMSPTN

JSB<JSB+5>0$E
JOBMAP+2/ 0
JOBMAP+3/ 0
JOBMAP+5/ 0
```

WORD SEARCH ($W)

Word search compares each storage word with the word being searched for in those bit positions where the mask, located at $M, has ones. The mask word contains all ones unless set by the user. If the comparison shows equality, the word search types out the address and the contents of the location; if the comparison results in inequality, the word search types out nothing.

The format of the command is a<b>c$W. a<b> is the range and c is the quantity searched for. To set the mask, type n$M where n is the quantity to be placed in the mask word.

Suppose we wish to find all share pointers in the current user's page map between pages 0 and 10. In this case, store a 7 (for pointer type) in bits 0-2 of the mask. The command is UPTA<UPTA+10>200000,,0$W and works as follows:

```
7000000,,0$M

UPTA<UPTA+10>200000,,0$W
UPTA+2/ 206000,,1244
UPTA+4/ 206000,,1242
```

NOT WORD SEARCH ($N)

Not word search works like word search, the only difference is that it types out the contents of the register when the comparison is an inequality, and types nothing when an equality is found.

Not word search is commonly used to type out all non-zero locations in some range. Suppose you wish to find all existent (non-zero) entries in the JSB map; you would type:

```
-1$M
JOBMAP<JOBMAP+66>0$N
JOBMAP/ 224000,,635
JOBMAP+1/ 124003,,7044
JOBMAP+4/ 124003,,2764
JOBMAP+6/ 124003,,7050
```

MDDT

MDDT is a part of the monitor that allows you to look at the running monitor with the standard DDT commands; your process is always the running process when you use MDDT. You can also call monitor routines to map pages, etc.; however, extreme caution should be taken when using MDDT. If you change any locations, you can crash the monitor. It is a good practice to type carriage return immediately after you open any location to prevent accidental deposits into memory.

You can enter MDDT in either of two ways. In the first example, the running fork will be the top fork of your job, i.e., the EXEC. In the second example, the running fork will be the fork running user level DDT.

```
@ENABLE
```

```
$^EQUIT
MX>/
MDDT


@ENABLE
$SDDT
JSYS 777$X
MDDT
```

You can use either method to enter MDDT. Return from MDDT by calling the routine MRETN. Do this by typing:

```
MRETN$G
```

EDDT

While you are in EDDT, timesharing ceases.

Loading the Monitor With EDDT

Switch registers: 0,1,2,7

```
BOOT>/L
BOOT>/G141
EDDTF/ 1
DBUGSW/ 2
143$G
```

This page is for notes.

MODULE TEST

When answering the lab exercises, write down the names of the tables where you found the answers. The labs will help you understand the monitor data base structure. So remember, where to look is more important than what you find there.

The exercises marked with a double star (**) are more difficult and are optional. As the course progresses, you may feel more comfortable about these portions; feel free to come back to them later.

ILLUUO Crash

TOOLS

This set of exercises uses a crash named <MONITOR-INTERNALS>ILLUUO.EXE. The monitor you should use to load your symbols is named <MONITOR-INTERNALS>ILLUUO-MONITR.EXE. Do not forget to set monitor context!

EXERCISES

You should attempt to analyze why the ILLUUO crash occurred. The following questions should help you look in the right directions. Good luck!

1. What is the BUGHLT? What does it mean to get this BUGHLT type?

2. What was the PC that caused the BUGHLT?

3. What was the instruction that trapped as an illegal UUO?

4. What stack was in use?

5. What mode was the processor in when the illegal UUO occurred?

6. How did it get to that instruction? **

TEST EVALUATION SHEET

EXERCISES

1. What is the BUGHLT? What does it mean to get this BUGHLT type?

   ANSWER: The BUGHLT is an ILLUUO. This means the monitor executed an illegal instruction, which, in turn, generally means that either the monitor somehow started executing data or its ACs or that some code was garbaged.

2. What was the PC that caused the BUGHLT?

   ANSWER: Since it is a UUO, the PC is stored in the MUUO old PC word; address KIMUPC (this is offset 424 in the UPT/PSB page). Remember that this is the updated PC, so it is usually 1 greater than the UUO.

```
KIMUPC/ 304000,,1
```

3. What was the instruction that trapped as an illegal UUO?

   ANSWER: The updated PC was 1; so the illegal UUO is in AC0.

```
0/ 0    ;certainly looks illegal
```

4. What stack was in use?

   ANSWER: The stack in use tells us what was generally going on.

```
P/ UPDL+125,,UPDL+30    ;JSYS in progress
```

5. What mode was the processor in when the illegal UUO occurred?

   ANSWER: The PC flags have bit UMODF on if it was user mode; but a ILLUUO cannot happen anyway unless it was monitor mode.

```
KIMUPC/ 304000,,1

UMODF= 10000,,0          ;not user mode
```

6. How did it get to that instruction? **

   ANSWER: The best way to figure this out is to look at the stack. There are these basic facts that help to make sense of the stack:

   1. P contains the current stack pointer.

   2. MPP contains the stack pointer at the time the last JSYS began. If MPP does not point to the start of the stack plus 1 (UPDL+1), there is a JSYS that called a JSYS.

   3. If an entry was made on the stack by a PUSHJ, it will look like a PC. This is not a hard and fast rule, but it can help. A user mode PC usually has bits 1,2, and 3 on and a monitor PC has bits 1 and 2 on.

   4. When a JSYS begins, it pushes the old PC on the stack twice.

   5. If a return address is still on the stack (i.e., has an address less than the stack pointer), then you have not returned from the routine.

   6. The monitor uses the stack for temporary storage. The macros STKVAR, TRVAR, etc. leave recognizable things on the stack. Knowing these conventions helps you recognize which stack locations are being used as temporary storage.

Using these points, try to fit the pieces together. Here is the stack and deductions from it:

```
P/ UPDL+125,,UPDL+30          ;current stack pointer

MPP/ UPDL+76,,UPDL+1          ;only one JSYS

UPDL/ 310000,,442531          ;PC at time of JSYS
    / 310000,,442531          ;2nd copy of PC
    / 0
    / 1,,1
    / MMAP,,.STKRT            ;looks like temporary
                              ;storage macro (STKVAR)
    / CAIA SOUT1+1            ;looks like a PC
                               ;and the address
                              ;implies this
                              ;is a SOUT jsys.
    / MMAP+400,,SWPMLK+3
    / JSTAB+161
UPDL+10/ CAIA SOUTB+6         ;looks like a PC
    / CAIA BYTOUA+13
    / CAIA NOPGT0+15
    / -1
    / 1,,160000
    / -1
    / JSTAB+161
    / 4,,4                    ;4 temporary
                              ;storage locations
UPDL+20/ CAIA .STKRT          ;another STKVAR
    / CAIA NEWWNA+6           ;looks like a PC
    / -1
    / JSTAB+160
    / CAIA JFNOF5+20          ;looks like a PC
    / UPDL+77,,UPDL+2
    / T1
    / CAIA .ASRET             ;looks like a PC
UPDL+30/ JSTAB+563,,10        ;this is our top
                              ;of stack
    / -1                      ;since this is the
                              ;last location
                              ;popped off the stack;
    / CAIA .TRRET             ;routine for return
                              ;with TRVAR
    / CAIA GTFDB2+14
    / CAIA USTDIR+1
```

P and the stack indicate there is a SOUT going on; MPP indicate it is not a nested JSYS. The PC at the time of the JSYS (in UPDL) is a user mode PC because UMODF is on.

Locations UPDL+2 through UPDL+4 are STKVAR locations; .STKRT is the return routine STKVAR sets up and the location before it (containing 1,,1) is the count of temporary storage locations, that is, 1.

It looks as if the SOUT called NEWWNA to change the file window page. That code called JFNOF5, which made a call to NEWLFP; this is indicated by the entry in UPDL+24, which is the return address from NEWLFP. Looking at the code, and what gets called, it seems that NEWLFP went to NEWFLL and called NEWLFS, which failed. In the literal at NEWFLL+3, it calls and returns from USTDIR, and then it adjusts the stack. This causes it to POPJ to -1, the next higher entry on the stack.

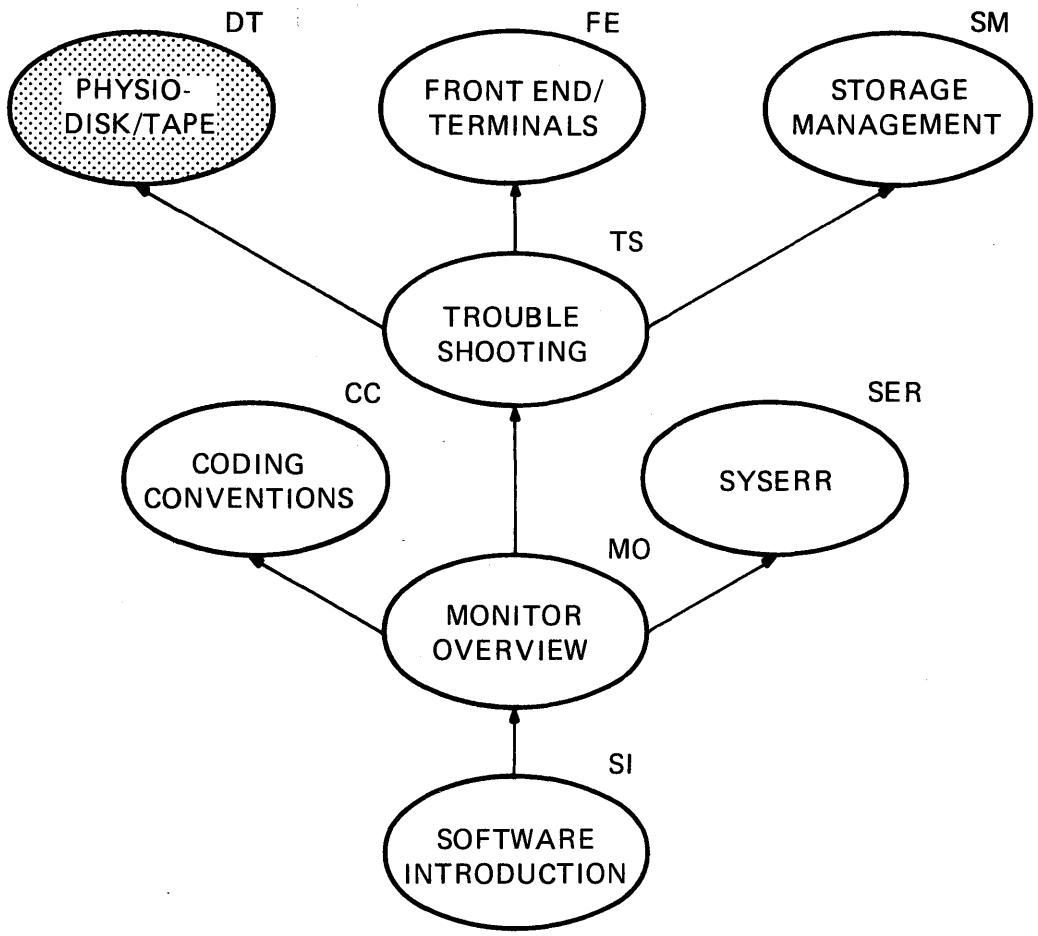
So it adjusted the stack by 1, which in this case, it should not have. The code apparently expected one more thing to be on the stack.

TOPS-20 MONITOR

PHYSIO - Disk/Tape

<<For Internal Use Only>>

COURSE MAP



MR-2717

This page is for notes.

PHYSIO - Disk/Tape

INTRODUCTION

This module covers, in depth, the flow of the physical I/O for Magnetic Tape and Disk transfers. The flow starts with the generation of disk/magtape I/O Request Blocks (IORBs), through their queueing to the calling of the device-dependent, unit specific code. The channel and unit data bases are discussed, along with a description of the algorithms used for the selection of the appropriate transfer to be executed. Interrupt and error processing are also addressed.

LEARNING OBJECTIVES

Upon completion of this module, the student will be able to:

1. Describe the overall structure of the Physical I/O data base.
2. Identify the data contained in each of the structures relating to the Physical I/O data base.
3. List the various error counts and specify the location of each.
4. Describe the disk allocation algorithms and the relevant portions of the data base.

RESOURCES

1. Monitor Tables
2. Micro-fiche of the Monitor

MODULE OUTLINE

PHYSIO-Disk/Tape

I. PHYSIO
   A. Data Structure
   B. Queueing an IORB
   C. Scheduling an IORB
   D. Starting I/O
   E. Interrupt Handling

II. Disk Allocation
   A. Data Structure (DSKBTTBL)
   B. Space allocation
   C. Space Deallocation
   D. Drum Allocation
   E. BAT Blocks

III. DISK Dependent I/O
   A. Data Structure
   B. Disk-Dependent Code
   C. Disk Interrupts
   D. Disk Errors and Abnormal Conditions

IV. MAGTAPE Dependent I/O
   A. Magtape Data Base
   B. Magtape IORB
   C. CDB, KDB, and UDB
   D. Interface to PHYSIO
   E. Magtape I/O Wait
   F. CLOSF Device-Dependent Functions
   G. Magtape Interrupts
   H. Error and Abnormal Conditions

This page is for notes.

PHYSIO

Data Structure

IORB (I/O REQUEST BLOCK)

An IORB is built by the disk or magtape dependent code. Short IORBs are built for most disk I/O requests; magtape I/O always uses long IORBs.

CHANNEL DATA BLOCK

The channel data block, or CDB, contains channel-dependent status information. All interrupts for the channel begin processing in the CDB and are dismissed by code in the CDB; interrupts come in at offset CDBINT=-6 and are dismissed at offset CDBJEN. Offset CDBDSP has the address of the channel dispatch table (RH2DSP). The KDB addresses for magtape, or the UDB addresses for disk, begin at offset CDBUDB. The channel number is in offset CDBADR.

Each CDB has a device-dependent portion beginning at CDBDDP. For RH2Øs, this portion is a 4-word block with the CONI, CONO, DATAI, and DATAO instructions in it. For RH11s, the portion contains the device registers, the UNIBUS status register address, and the UNIBUS bus address base address.

CONTROLLER DATA BLOCK

The controller data block, or KDB, exists for magtape controllers. Disk units each have their own controller built in; there is no KDB for disk. The KDB has the addresses of the unit data blocks for each magtape unit on the controller; one controller can handle a maximum of eight units.

Offset KDBDSP has the address of the KDB function dispatch table. Offset KDBUDB is the first of eight possible UDB addresses, and each KDB has a device-dependent portion beginning at offset KDBDDP. I/O instructions and current unit status information is kept in this portion.

UNIT DATA BLOCK

Each magtape and disk unit has a unit data block, or UDB. Each UDB has unit-dependent status information for the unit. Offset UDBPWQ is the header for the unit's position wait queue, and offset UDBTWQ is the header for the unit's transfer wait queue.

Each UDB has a device-dependent portion beginning at offset UDBDDP. RP04 and RP06 devices have the hardware I/O instructions here, plus the twenty drive registers. Magtape devices have the slave address on the slave bus, error information for the drive, and the tape cleaner flag stored here also.

FINDING EACH DATA STRUCTURE

The CHNTAB table, indexed by channel number, contains the address of the channel data block, or CDB. For disk, offset CDBUDB of the CDB is the first of eight pointers for the eight possible unit data blocks, or UDBs; if the unit exists, the pointer has the address of the unit's UDB. For magtape units, offset CDBUDB contains the address of the KDB, or controller data block. The KDB has the pointers to the magtape UDBs, beginning at offset KDBUDB; each controller can support a maximum of eight units.

For magtape units, MTCUTB, indexed by unit number, contains the CDB address for the unit in the left half and the UDB address for the unit in the right half.

RH20 CHANNEL DISPATCH TABLE

Table PHYCHT contains the addresses of the channel dispatch tables for each channel supported. Currently, this table has one entry and all systems have the address RH2DSP. However, the RH2DSP table is in PHYH2 or PHYH11, and only one of these modules will exist in a monitor; PHYH11 is the 2020 channel-dependent module and PHYH2 is in all other TOPS-20 monitors.

The channel dispatch table offsets are described in the CDS table in the Monitor Table Descriptions section. Each offset serves a different function. For example, offset

CDSINI initializes the channel and builds the CDB.

UNIT DISPATCH TABLE

Table PHYUNT has the dispatch table addresses for all supported unit types. The possible dispatch tables are:

1. RS4DSP -- RSØ4 dispatch table

2. RP4DSP -- RPØ4/RPØ6 dispatch table

3. TM2DSP -- TMØ2 dispatch table

The KDB, at offset KDBDSP, contains the KDB dispatch table -- TM2DSP, the same table referred to above as the unit dispatch table.

Many channel-dependent functions (i.e., routines dispatched to through the channel dispatch table) will also dispatch through the unit dispatch table. For example, RH2DSP plus CDSSIO (start I/O offset) does some channel-dependent functions and then dispatches through offset UDSSIO of the TM2DSP table, or the RP4DSP table, depending on the unit type.

Queueing an IORB

The IORB is added to either the transfer or position wait queue for the relevant unit. If the unit's channel is inactive, the request is serviced; that is, positioning or transfer is started, depending on the request type.

QUEUING A MAGTAPE IORB

For magtape, routine MTAIRQ is called to set up the IORB transfer list and fill in other IORB fields. MTAIRQ then calls QUEIRB (in PHYSIO) which calls routine PHYSIO (in PHYSIO) to add the IORB to the unit's transfer wait queue or position wait queue.

Magtape requests must be serviced in the order they are requested. To insure this happens, requests are handled as follows:

1. In a transfer request with the position wait queue (PWQ) empty, add the request to the transfer wait queue (TWQ).

2. In a position request, add the request to the PWQ.

3. In a transfer request with the position wait queue not empty, add the request to the PWQ.

For an explanation of how this affects scheduling, see the section on how IORB's are scheduled for magtape.

QUEUING A DISK IORB

For disk, read operations are triggered via the page fault mechanism. Write operations are triggered by UFPGS JSYS, un-mapping a file page, DDMP action, or by one of the garbage collectors (XGC or GCCOR).

Read requests call SWPIN; write requests call SWPOUT. These routines do any necessary overhead operations (such as mapping the index block, getting a page from RPLQ, setting up the core management data base, putting the page on RPLQ, assigning drum space, etc.) and then call DSKIO to set up a short IORB. (Index block reads and writes go through UDSKIO).

For disk requests, routine PHYSIO will add the IORB to either the unit's position wait queue or transfer wait queue. If the unit is positioned at the same cylinder as the requested cylinder and the fairness count for the unit has not expired, the request is added to the unit's transfer wait queue; otherwise, the request is added to the unit's position wait queue.

If the channel is not active, positioning or data transfer (depending on whether the request was added to the PWQ or to the TWQ) is started. If the channel is busy, the request is scheduled at interrupt level; if the request is added to the TWQ for disk, it may be stuffed into the backup register. A request is added to the backup register if:

1. The primary command is active.

2. It is for the cylinder at which the unit is positioned.

3. Full latency optimization is enabled.

4. Its sector address is within minimum latency time.

Scheduling an IORB

Most scheduling of IORBs is done at interrupt level. When a transfer done interrupt occurs for a unit, another transfer is started for the channel. Each unit has a transfer wait queue; however, the channel can handle only one data transfer at a time. When a disk unit's transfer wait queue is empty, a new cylinder is selected via the scan algorithm, and positioning is started for that unit. Note that positioning can be started for a unit (or units) with data transfer started on another unit; that is, positioning and data transfer can go on simultaneously for units on the same channel. However, the positioning requests must be started before any data transfer is requested because once a data transfer begins, the channel is busy until the transfer done interrupt occurs.

The algorithms used to choose a transfer request for disk and magtape are as follows.

For magtape:

Requests must be serviced in the order they are given. Choose the next request on the TWQ. If the TWQ is empty, look at the first request on the PWQ. If it is a positioning request, start positioning. An empty TWQ, with the first request on the PWQ being a transfer request, implies that the last serviced request for this unit was a position request; move all transfer requests (up to the next positioning request) from the PWQ to the TWQ. Service the first request on the TWQ. The unit is chosen in round-robin fashion.

For disk:

1. Choose the best latency request from TWQ for each unit and, then, the best latency for all units on the channel.

2. If the fairness count for best latency across units has expired, step one unit from where marker CDBCUN (in the CDB) points.

3. If a unit's TWQ is empty, start positioning the unit using the SCAN algorithm with read preference (next higher numbered cylinder read request; if none, take next higher numbered write request; if none, start algorithm over at lowest cylinder). IORBs in the PWQ for the cylinder positioned to are moved to the TWQ. Note: the channel can simultaneously transfer data and postion units.

When a request has been chosen, start I/O.

If the primary command is active when the interrupt comes in, it is likely that there was a backup register request which became the primary command when the request (whose interrupt is being serviced) was finished transferring. Moving the backup register command to the primary command is done by the hardware if the backup register has a command in it when the primary command finishes. When the routine that schedules transfers at interrupt level sees that the primary command is active, it looks for a transfer request in the TWQ whose sector address is less than minimum latency; if there is such a request, it is stuffed in the backup register.

Starting I/O

The routine in offset CDSSIO of the channel dispatch table is called to start I/O for the channel.

Interrupt Handling

All disk and magtape interrupts vector to the fourth word of the channel's data logout area in the EPT; this is the vectored interrupt location, which always contains:

JRST 7,CDB -6 (where CDB is that channel's CDB).

This instruction stores the old PC and flags in the first two locations and picks up the new PC from the next two locations. The PC picked up is CDB-1; the instruction in this location saves AC 10 and then does a JSP 10,PHYINT. Routine PHYINT does the standard pre-processing for disk and magtape.

PHYINT calls channel-dependent code through offset CDSINT of the channel dispatch table to analyze the cause of the interrupt. The channel-dependent routine for RH20s is RH2INT. Lower level routines called by RH2INT (i.e., unit dependent routines) return an argument in accumulater P4; this argument is passed to PHYINT to indicate whether to dismiss the interrupt (argument is zero), schedule another channel cycle right away (argument less than zero), or housekeep the current request (argument greater than zero). For example, the argument returned can have the following meanings:

1. Request to dismiss (P4=0) -- the done flag is on and the channel is not occupied.

2. Request for immediate channel cycle (P4 < 0) -- positioning done interrupt has occurred and there is no transfer in progress.

3. Housekeep current request (P4 > 0) -- Transfer done interrupt occurred and transfer done requires housekeeping before another request can be scheduled.

DISK ALLOCATION

Data Structure (DSKBTTBL)

Each structure on the system has a file, called STR:<ROOT-DIRECTORY>DSKBTTBL, containing the current structure allocation information. This file is divided into two parts. The first portion holds a word containing a count of free pages for each cylinder on the structure. The second portion has one bit for each page on the structure; if the bit for a page is on, the page is free.

This file must be mapped into the monitor's address space in order to allocate space on a structure. For Model B processors using extended addressing, it is mapped into section BTSEC (BTSEC = 4, currently) beginning at address 0. For systems not using extended addressing, the bit table is mapped into section 0 beginning at location BTB. The bit table is mapped only when space is being allocated or deallocated.

Space Allocation

The routine DSKASN in DSKAL1 allocates space on the disk. The caller can request a specific cylinder; when a new page is allocated for an existing file, the caller requests a page from the same cylinder that the last file page was allocated from.

If a cylinder is requested, and no page is free on that cylinder, this algorithm is used to select a page:

1. Look for the next higher cylinder on the current unit with any free page.

2. If no page is found on this unit, step to other units in the current structure.

If no cylinder is requested, the search for a page to assign begins at the cylinder number stored in location SDBLCA of the SDB; this location contains the cylinder from which a page was last allocated. The algorithm works as follows:

1. Start with the last cylinder a page was assigned from (contents of SDBLCA).

2. Look at its cylinder number (across all units) for at least SDBMFP pages. (SDBMFP is in the SDB and contains the minimum free pages below which DSKASN changes its assignment algorithm. SDBMFP is initialized to the value stored in table MINFPG for this structure's unit type.)

3. If there are less than SDBMFP pages free on this cylinder, increment the cylinder number and look at that cylinder for SDBMFP pages across all units.

4. If no cylinder has at least SDBMFP pages, choose the cylinder with the most pages and set SDBMFP to this value.

Once a cylinder is chosen, the count of free pages on the cylinder is decremented in the first part of DSKBTTBL and the bit for the chosen page is turned off. DSKBTTBL will always be updated before the index block of a file is updated.

Space Deallocation

Routine DSKDEA is called to release a disk page. The bit is set to 1 for the page and its cylinder's free page count is incremented.

Drum Allocation

Table DRMCNT has a one word entry for each cylinder in the swapping space; table DRMBBT has one bit for each page in the swapping space. Location DRMFRE is the total number of free pages on the swapping space. Routine DRMASN assigns swapping space while DASDRM de-assigns it.

Routine DRMASN is called to assign a page anywhere on the drum and DRMASA is called to assign a specific address. DRMASN looks for the first free page of swap space, beginning where a page was last assigned. The cylinder a page was last assigned from is remembered in location DRMBN0. Routine DRMASA tries to assign the requested page; if the page is not free, it gives an error return. For most cases, DRMASN is the routine called to assign a page. DRMASA is called: for swappable monitor space when the system is initialized, by GCCOR when it writes a group of pages, and to mark bad spots in the swapping space (to keep them from being allocated).

BAT Blocks

Any page in a bad spot on the disk is marked in the BAT blocks and is also marked as assigned in the bit table for disk or drum. When the disk is formatted, the BAT blocks are allocated and any bad spots on the disk at this time are marked in the BAT blocks. If, at any later time, a page gets a hard read or a hard write error, the file's FDB is marked. When the page is released from the file, the page is marked in the BAT blocks.

The channel-dependent routine also records error information so that PHYINT can see if error recovery is in progress or should be started.

If the interrupt is a positioning complete or transfer done interrupt, the IORB is removed from the PWQ or TWQ and the IORB is posted as done. If the interrupt is from a disk unit and the backup register had a request, the backup register request is serviced while the interrupt handling is in progress.

DISK DEPENDENT I/O

Data Structure

FILE WINDOW PAGE

All disk I/O is done by mapping a file page to or from a virtual address space. The PMAP JSYS is the way a user maps a file page. The sequential and dump mode JSYSs also use page mapping for disk I/O. However, this mapping is invisible to the user. The page will be mapped through a page called the file window page. There can be one file window page per JFN.

File window pages are allocated in the JSB space, and the address of a file window page is stored in offset FILWND of the JFN block. When the user executes an I/O JSYS (sequential or dump mode) to a disk file, the data is moved from the user's address space to the file window page (if writing) or from the file window page to the user's address space (if reading). When the window page is either full or empty, it is unmapped and the next page of the file is mapped. If the file is being written, unmapping the page insures that the updated information is on disk.

JFN BLOCK ENTRIES

FILBYT is the current byte pointer into the file window page. FILCNT is the number of bytes left in the file window page. FILCNT is decremented for each byte removed from the file window page (if reading) or added to the file window page (if writing). When FILCNT goes to zero, the next page of the file is mapped to the file window page. FILBYN is the byte number of the last byte read or written. FILLEN is the total length of the file in bytes. If the file is being written and FILBYN becomes greater than FILLEN, FILLEN is set to FILBYN. The OFN of the current index block is stored in the left half of FILOFN.

A long file is one with a data page whose page number is greater than 777. When a file is long it has a super index block (also called a page table table). This is required because with only one index block, there are slots

for only pages 0-777. Note that a file can be long but have only one page whose page number is greater than 777. When a long file is open, the OFN of the super index block is stored in the right half of FILOFN. FILCOD has the OFN of index block 0; this is the unique identity ENQ/DEQ uses for a file. When a short file goes long, index block 0 is used because it is the only overhead page that is constant. Index block 0 always exists for a file, even if there are no pages in the file with page numbers between 0 and 777.

The left half of FILLFW contains the count of pages mapped. The file cannot be closed until there are no pages mapped. FILFDB has the address of the file's FDB, the offset from the beginning of the directory.

DISK IORBS

Disk file data I/O requests use short IORBs. A short IORB is an entry in table CST5. The offset into CST5 is the physical core page that is to be read into or written out. An entry in CST5 that is a queued IORB has the link to the next IORB in the right half and status bits in the left half. (In other words, a short IORB is one word long and has the same format as the first word of a long IORB.) Status bit IS.SHT is on if the IORB is short. The parallel entry in CST3 is the disk address to be read from or written to. Routine DSKIO creates short IORBs.

Index blocks and other disk pages that are not data pages of a TOPS-20 file (such as front end file pages, and home blocks) use long IORBs. Routine UDSKIO creates long IORBs for disk pages. The DSKOP JSYS also creates a long disk IORB when necessary. See the tables descriptions for the layout of a long disk IORB.

Disk requests (IORBs) will always be for one page at a time.

Disk-Dependent Code

The disk-dependent code for sequential and dump mode JSYSs is called through offsets in table DSKDTB. Each time the file window page is full or empty (when FILCNT goes to zero), the disk-dependent code is called to map a new page

into the file window page.

MAPPING A NEW PAGE INTO THE FILE WINDOW PAGE

A combination of the following factors determines what happens when a new page needs to be mapped:

1. The new page does or does not exist
2. The user is reading or writing
3. The file is long
4. The file is going long

If the new page exists and is in the same index block as the previous page, the new page is mapped and control returns to the caller.

If the new page exists but is in a different index block than the previous page, an OFN is needed for that index block. The previous OFN is released and the new OFN is stored in the left half of FILOFN.

If the new page does not exist, and the user is reading, a private core page of zeroes is set up in the file window page. The user will see data as zeroes when reading from non-existent pages before the EOF mark.

If the new page does not exist, and the user is writing, the directory's quota must be checked. If the page to be mapped is not in the same index block, an OFN is needed for the index block. The OFN is stored in FILOFN and the OFN previously in use is released.

If the file is going long, a super index block is created, an OFN assigned for the super index block and stored in the right half of FILOFN, and the super index block is mapped to a page of the JSB space with its mapped address stored in the right half of FILLFW.

Disk Interrupts

All disk and magtape interrupts vector to the fourth word of the channel's data logout area in the EPT; however, all disk and magtape interrupt handling begins at PHYINT (in PHYSIO). See the PHYSIO section for a description of what happens in the CDB before dispatching to location PHYINT in PHYSIO. After standard pre-processing, long disk IORBs dispatch to UDIINT, the interrupt address set up in the right half of offset IRBIVA in the IORB. If flag IS.SHT (short IORB) is set, go directly to SWPDON. Since the interrupt location cannot be specified for short IORBs, it is defaulted to SWPDON.

Disk Errors and Abnormal Conditions

Before each transfer for a long disk IORB is started, routine UDISIE is called to see if the unit is offline. If so, the transfer is aborted. This check is not made for short IORBs.

MAGTAPE DEPENDENT I/O

The magtape device-dependent code is called through offsets in table MTADTB; each function has an offset in MTADTB.

Magtape Data Base

Of the several levels of magtape data base, the principle ones are for the MAGTAP and PHYSIO modules. The MAGTAP data base looks at the world from a magtape drive point of view and has tables with per-drive entries. The PHYSIO data base works with disk and magtape I/O requests; PHYSIO maintains the position and transfer wait queues and schedules requests from them. I/O requests for disk and magtape have formats defined by PHYSIO; a request is called an I/O request block, or IORB. A magtape IORB is built by MAGTAP and passed to PHYSIO to be queued and scheduled. There is also a small amount of data that must be maintained for the channels, controller, and units.

MAGTAPE BUFFERS AND RELATED DATA BASE

For sequential I/O, buffers are built in the JSB space of the job. The buffer size is determined by the record size the user has set at the time of the first read or write. For record sizes larger than a page, a non-contiguous set of pages makes up the buffer. Each magtape unit has a list of buffer page pointers. The maximum size of a record is 2Ø (octal) pages and there are two buffers; therefore, the buffer page pointer list needs 4Ø (octal) entries. The buffers are built when the first sequential I/O request is made.

Table MTANR2, indexed by unit number, has the address of the list of buffer page pointers for that drive. Table MTANR3, indexed by drive number, has fields MTCUB (current user buffer) and MTCSB (current service routine buffer). Table MTANR4, indexed by drive number, has a field MTCUP (current user page) for storing the offset into the buffer page pointer area.

USE OF JFN BLOCK LOCATIONS

FILLEN is set to the number of bytes in a record (the record size). FILCNT is set to the number of bytes in the current page of the buffer (this will be less than a full page of bytes if the record size is less than a page or if this is the last page of the buffer and it is only partially used). When FILCNT goes to zero, it is time to step to the next buffer page or, if this was the last page of the buffer, read or write the next record. The magtape-dependent code is called whenever FILCNT goes to zero; it determines if it should set up the JFN block to point to the next buffer page or, if this was the last page of the buffer, step to the next record. FILLEN is the number of bytes in the whole record, so if FILBYN has reached FILLEN, it is time to step to the next record.

When the magtape-dependent code returns to the device-independent code: FILCNT contains the number of bytes in this buffer page, FILBYT is a byte pointer to the beginning of this buffer page, and FILBYN is set to zero. FILLEN is the number of bytes in the whole record.

Magtape IORB

Magtapes use long IORBs. These have a 7-word header, followed by the number of hardware bytes in the record, followed by the channel command list.

CDB, KDB, and UDB

The Channel Data Block (CDB) contains channel-dependent information. There is one CDB for each channel. The six locations immediately preceeding the CDB for each channel contain the interrupt storage and vectors for the specific channel. (This is the space pointed to by the interrupt locations in the channel logout area of the EPT.) The CDB contains status information, channel timers, error counts, etc. See the table descriptions for a full breakdown of the CDB Table.

The Kontroller Data Block (KDB) exists only for TM02 tapes. The KDB contains MASSBUS addresses and the UDB table for these units.

The Unit Data Block (UDB) contains unit specific information. Among the data in the UDB are the error counts, timers, and pointers to the units' queued IORBs. See the table descriptions for the specific contents of the UDB.

Interface to PHYSIO

When a record needs to be read in or written out, MAGTAP uses the IORB pointed to by field MTCIRB in table MTANR4, indexed by drive number. The pages of the buffer are locked in core and the physical page numbers of the buffer pages are written into the left half of the MTBUF entries. MAGTAP then builds an I/O list in the IORB, beginning at offset MTIRBL+1. This I/O list is composed of channel command words, or CCWs. MAGTAP then calls PHYSIO to queue the request.

Magtape I/O Wait

If the other buffer is also queued (that is, unavailable to the user to fill or empty), the process must block. The process is put into balance set wait for a maximum of 50 ms. A standard wait list test is also set up; the blocking reason is the IORB number and the test routine is MTARWT. If 50 ms. expires before the record is ready, the process is moved to WTLST.

CLOSF Device-Dependent Functions

When a CLOSF is issued for a magtape, several device-dependent functions must be performed. If the tape is open for read, the tape is left positioned after the tape mark. If the tape is open for write, any partial buffers are forced out, two tape marks are written, the tape is positioned between the tape marks, and any remaining IORBs are flushed.

Magtape Interrupts

All disk and magtape interrupts vector to the channel's data logout area in the EPT; however, interrupt handling for both disk and magtape begins at location PHYINT (in PHYSIO). See the PHYSIO description for an explanation of how control passes from the EPT to PHYINT. After standard pre-processing, magtape interrupts dispatch to MTAINT, the interrupt address set up in the right half of offset IRBIVA of the IORB.

Routine MTAINT unlocks the buffer pages associated with the IORB (they are locked when the IORB is queued because they must be in core for the data transfer). It also checks for errors and aborts any other queued IORBs for this drive if there are errors.

Error and Abnormal Conditions

Before each magtape transfer is started, routine MTCHKA is called to check if the abort flag has been turned on for the IORB. If the abort flag is on, the transfer is aborted. MTCHKA is the address set up in the right half of offset IRBIVA of all magtape IORBs.

MODULE TEST

When answering the lab exercises, write down the names of the tables where you found the answers. The labs will help you understand the monitor data base structure. So remember, where to look is more important than what you find there.

The exercises marked with a double star (**) are more difficult and are optional. If you have the time and/or motivation, do them.

I/O - Magtape Dependent Data Base

Since there are not enough magtape drives to go around for the whole class, this part of the lab uses a "set up crash"; that is, while using a magtape and while the monitor was in a desired state, the system was deliberately crashed.

Tools

For the following exercises, use FILDDT on the crash named <MONITOR-INTERNALS>TAPE.CPY. For symbols, use the monitor named <MONITOR-INTERNALS>R3-MONITOR.EXE. Be sure to set monitor context!

EXERCISES

1. Which magtape drives are in use?

2. Are there any magtape buffers set up? What is the buffer size?

3. Which pages of the JSB space are in use as magtape buffers?

4. Are the pages locked in core? **

5. Which buffer and which page of the buffer is the sequential I/O code currently using?

6. Is there a queued magtape IORB? If so, what is the transfer list for the IORB? **

7. What is the status of each magtape drive?

I/O - PHYSIO Dependent Data Base

Because much of the PHYSIO data base changes rapidly, the exercises for this section make use of a crash.

Tools

The crash for this part of the lab is <MONITOR-INTERNALS>BADBTB.CPY. For symbols, use the monitor in the file <MONITOR-INTERNALS>R3-MONITOR.EXE. Do not forget to set monitor context!

EXERCISES

1. Find the location of each channel data block.
2. Find the location of each unit data block.
3. Which channel is each UDB associated with?
4. How many units are on channel Ø?
5. Which channels have disk units on them? **
6. Which UDBs are for units that are part of the structure PS:? **
7. Find the TWQ for unit Ø of the public structure.
8. Find the UDB for MTAØ:
9. Were any magtape units in use at the time of the crash?
10. Trace the TWQ for each disk unit.
11. Trace the PWQ for each unit of PS:
12. Are there any long IORBs in the PWQ for the units of PS:? **
13. What routine is called to start I/O for a disk unit? For a magtape unit? What module is each routine in? **

This page is for notes.

TEST EVALUATION SHEET

I/O Magtape Dependent Data Base

EXERCISES

1. Which magtape drives are in use?

   ANSWER: Table MTASTS, indexed by drive number, contains a drive's status. When a drive is not in use, its status is 0. Location MTINDX contains the highest real unit number in the right half.

```
MTINDX/ -7,,1

MTASTS/ 0
MTASTS+1/ 410000,,400000
```

2. Are there any magtape buffers set up? What is the buffer size?

   ANSWER: Table MTANR2, indexed by drive number, contains the pointer to the buffer pages list. If the buffer is set up, the buffer pages list contains the JSB space virtual address for the assigned pages in the right half of each entry on the list. The buffer size is in the right half of MTANR1, indexed by drive number.

```
MTANR2+1/ 444400,,127325

127325/ 674,,627000     ;page 627 is a buffer
                        ;page
      / 677,,630000     ;page 630 is a buffer
                        ;page
      / 700,,631000     ;page 631 is a buffer
                        ;page
      / 701,,632000     ;page 632 is a buffer
                        ;page
      / 0

MTANR1+1/ 10041,,1006   ;size is 1006
```

Since the size is 1006, and since there are always 2 buffers assigned, pages 627 and 630 are for buffer 1 and pages 631 and 632 are for buffer 2.

3. Which pages of the JSB space are in use as magtape buffers?

   ANSWER: As stated above, pages 627, 630, 631, and 632 are in use as magtape buffers.

4. Are the pages locked in core? **

   ANSWER: First find the page's current storage address; if the page is in core, table CST1, indexed by page number, contains the lock count.

```
MMAP+627/ 324007,,400    ;mapped indirect
                         ;through slot 7
                         ;of the JSB map
MMAP+630/ 324010,,400    ;mapped indirect
                         ;through slot 10
                         ;of the JSB map
MMAP+631/ 324011,,400    ;mapped indirect
                         ;through slot 11
                         ;of the JSB map
MMAP+632/ 324012,,400    ;mapped indirect
                         ;through slot 12
                         ;of the JSB map

JSB+7/ 124003,,1474      ;page on drum

JSB+10/ 124003,,1510     ;page on drum

JSB+11/ 124000,,566      ;page in core (so
                         ;might be locked)

JSB+12/ 124003,,1520     ;page on drum


CST1+566/ 3,,1514        ;lock count = 0
```

No page is locked in core; only one page is in core.

5. Which buffer and which page of the buffer is the sequential I/O code currently using?

   ANSWER: Table MTANR4, indexed by drive number, contains the IORB, if any, that the service routine is using in the left half and the current user page in the right half. The current user page is the buffer page that the JFN block is currently moving bytes to or from; that is, the page of the buffer the sequential I/O code is currently using.

```
MTANR4+1/ 127330          ;this is the offset
                          ;in the buffer pages
                          ;list which contains
                          ;the address of the
                          ;current buffer page.

127330/ 701,,632000       ;currently on page
                          ;632
```

   Note that it is the second page of the second buffer.

6. Is there a queued magtape IORB? If so, what is the transfer list for the IORB? **

   ANSWER: We need to look at the TWQ and PWQ for the unit; table MTCUTB, indexed by drive number, contains the unit's UDB address. The TWQ and PWQ headers are in the UDB.

```
MTCUTB+1/ 555253,,555435      ;555435 is
                              ;the UDB for
                              ;unit 1

555435+UDBPWQ/ 555463,,0      ;this is an
                              ;empty queue.
                              ;When the
                              ;queue is
                              ;empty, the
                              ;tail pointer
555435+UDBPWQ=555463          ;points to
                              ;the header
                              ;itself.
```

```
555435+UDBTWQ/ 555464,,0        ;this is also
                                 ;an empty
                                 ;queue.

555435+UDBTWQ= 555464
```

7. What is the status of each magtape drive?

   ANSWER: Table MTASTS, indexed by unit number, contains the status.

```
MTASTS/ 0          ;not in use

MTASTS+1/ 410000,,400000

OPN= 400000,,0
BUFA=100000,,0
MT%ILW=400000
```

   Drive 1 is open, has buffers assigned, and is write locked.

I/O - PHYSIO Dependent Data Base

EXERCISES

1. Find the location of each channel data block.

   ANSWER: Each channel data block is pointed to by an entry in CHNTAB; the offset from CHNTAB is the channel number.

```
CHNTAB/ 555007          ;address of CDB for
                        ;channel 0
      / 555253          ;address of CDB for
                        ;channel 1
```

2. Find the location of each unit data block.

   ANSWER: Beginning at offset CDBUDB in each CDB, there is space reserved for pointers to the channel's UDBs; there can be as many as eight entries and if a channel has less than eight units, those entries are 0. If the channel has magtape units, the pointer is to the KDB and the KDB points to the magtape unit's UDBs.

```
555007+CDBUDB/ 555065   ;UDB for unit 0
             / 555155   ;UDB for unit 1

555253+CDBUDB/ 555331   ;pointer to KDB
                        ;(channel 1)
                        ;used for magtapes.

555331+KDBUDB/ 555365   ;UDB for MTA unit 0
             / 555435   ;UDB for MTA unit 1
```

3. Which channel is each UDB associated with?

   ANSWER: Offset UDBCDB of each UDB contains the unit's CDB address.

```
555155+UDBCDB/ 555007   ->
                                share same CDB
555065+UDBCDB/ 555007   ->

555365+UDBCDB/ 555253   ->
                                share same CDB
555435+UDBCDB/ 555253   ->
```

4. How many units are on channel 0?

   ANSWER: Two units because there are two UDBs pointed to by channel 0's CDB.

5. Which channels have disk units on them? **

ANSWER: Channel Ø has two disk units. Each UDB has a status word at offset UDBSTS with the unit type in bits 31-35.

```
555155+UDBSTS/ 4ØØ4,,146        ;type = 6

555Ø65+UDBSTS/ 44ØØ4,,146       ;type = 6

555365+UDBSTS/ 4ØØØØ2,,143      ;type = 3

555435+UDBSTS/ 4ØØ1Ø2,,143      ;type = 3


.UTT16= 3       ;TU45

.UTRP6= 6       ;RPØ6
```

We have two RPØ6s and two TU45s. Note that for historical reasons the TU45s have TU16-like mnemonics!

6. Which UDBs are for units that are part of the structure PS:? **

ANSWER: The structure data block has a list of pointers to the structure's UDBs (for its units) beginning at SDBUDB.

```
STRTAB/ SDDBLØ          ;PS's SDB

SDDBLØ+SDBUDB/ 3ØØØØØ,,555Ø65
              / Ø
```

7. Find the TWQ for unit Ø of the public structure.

ANSWER: The previous question located the UDB for unit Ø of PS: The TWQ for unit Ø begins at offset UDBTWQ.

```
555Ø65+UDBTWQ/ CST5+647,,CST5+647   ;tail,,
                                    ;head of
                                    ;unit's
                                    ;TWQ
```

8. Find the UDB for MTAØ:

   ANSWER: Table MTCUTB, indexed by magtape unit number, contains that unit's CDB address in the left half and the unit's UDB address in the right half.

```
MTCUTB/ 555253,,555365  ;CDB,,UDB for unit Ø
```

9. Were any magtape units in use at the time of the crash?

   ANSWER: Table MTASTS, indexed by unit number, has that unit's status. There are status bits to say the drive is open, doing a JSYS, etc.

```
MTASTS/ Ø               ;not in use
      / Ø               ;not in use
```

1Ø. Trace the TWQ for each disk unit.

   ANSWER: Each unit's TWQ header is at offset UDBTWQ of the unit's UDB. The queue is a linked list. From previous exercises, we know that the disk unit UDBs begin at 555155 and 555Ø65.

```
555155+UDBTWQ/ CST5+3Ø7,,CST5+3Ø7   ;only one
                                     ;page on queue
                                    ;it is page 3Ø7

CST5+3Ø7/ 4ØØØØ3,,Ø                ;no next IORB


555Ø65+UDBTWQ/ CST5+647,,CST5+647   ;also only one
                                     ;page on queue
                                    ;it is core
                                     ;page 647

CST5+647/ 4ØØØØ1,,Ø
```

11. Trace the PWQ for each unit of PS:

ANSWER: From previous exercises, we know PS: has only one unit whose UDB begins at 555065. A PWQ's header is at offset UDBPWQ.

```
555065+UDBPWQ/ CST5+670,,CST5+513

CST5+513/ 400001,,CST5+737

CST5+737/ 400001,,CST5+717

CST5+717/ 400001,,CST5+616

CST5+616/ 400001,,CST5+622

CST5+622/ 400001,,CST5+321

CST5+321/ 400001,,CST5+551

CST5+551/ 400001,,CST5+670

CST5+670/ 400001,,0              ;end of queue
```

12. Are there any long IORBs in the PWQ for the units of PS:? **

ANSWER: No, there are no long IORBs. If there were a long IORB, it would be allocated from the IORB pool and all the IORBs on the queue would be in CST5.

13. What routine is called to start I/O for a disk unit? For a magtape unit? What module is each routine in? **

ANSWER: To start I/O, PHYSIO first calls the channel-dependent level code in PHYH2; this code is dispatched to through offset CDSSIO in the channel's CDS table. This routine does the channel-dependent functions and then dispatches to unit-dependent code through the unit's UDS table at offset UDSSIO; the UDS table for magtape is in module PHYM2 and for disk, in module PHYP4. Table PHYCHT contains the addresses of the channel dispatch tables for each channel supported; currently, this is only RH2DSP for RH2Øs. Table PHYUNT has the dispatch addresses for all supported unit types; in this monitor, RP4DSP (for RPØ6s) and TM2DSP (for magtape) are of interest.

```
PHYCHT/ 1,,RH2DSP          ;RH2Ø dependent
                           ;dispatch table

RH2DSP+CDSSIO/ JRST RH2SIO ;dispatch here
                           ;to start I/O
                           ;for a channel.

PHYUNT/ 1,,RP4DSP          ;RPØ4/6 dependent
                           ;dispatch table
        4,,TM2DSP          ;TMØ2 dependent
                           ;dispatch table

RP4DSP+UDSSIO/ JRST RP4SIO  ;dispatch here
                            ;to start I/O
                            ;for an RPØ4/6

TM2DSP+UDSSIO/ JRST TM2SIO ;dispatch here to
                           ;start I/O
                           ;for a TMØ2
```

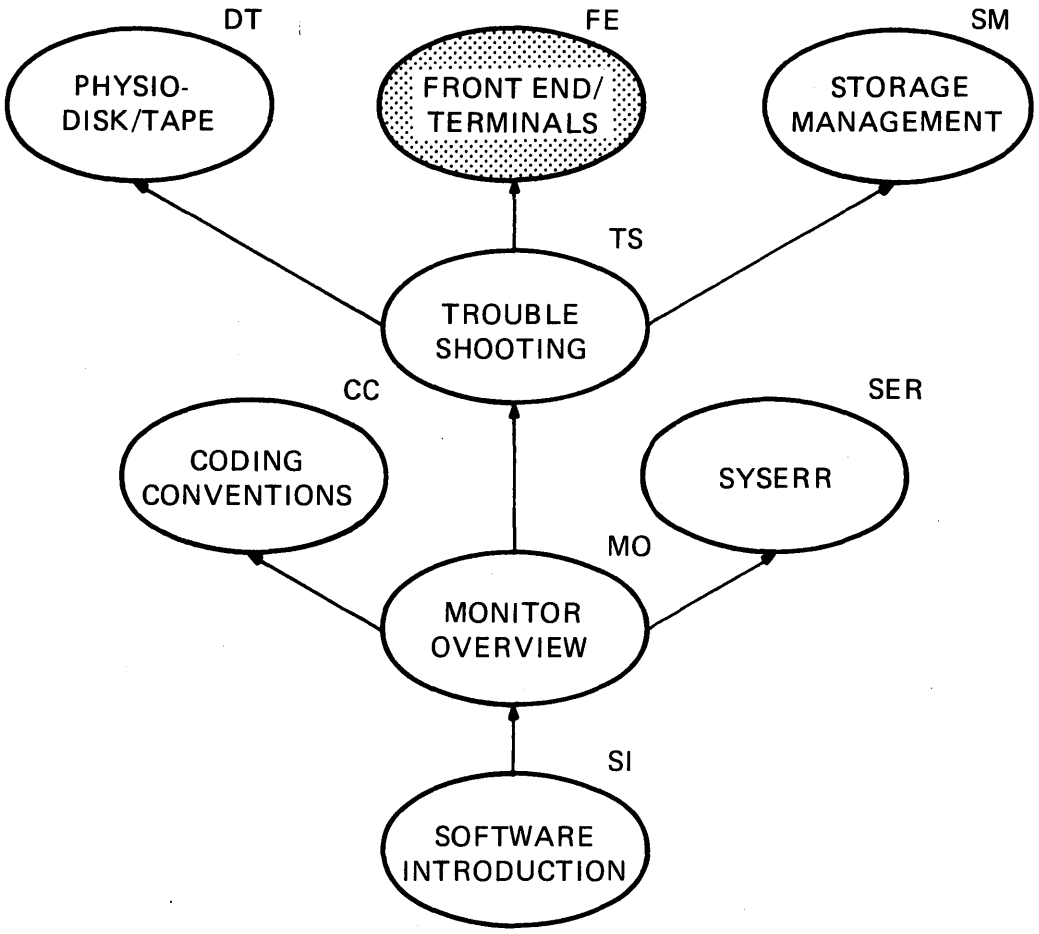
This page is for notes.

TOPS-20 MONITOR

Front End/Terminals

<<For Internal Use Only>>

COURSE MAP



MR-2717

This page is for notes.

Front End/Terminals

INTRODUCTION

This module covers data flow between the host processor and the front end in terms of the DTE formats for the I/O. Also addressed is the terminal handling data base and flow both within the host processor and the DTE. This module covers the -10 side of these transactions in depth. The processing within the -11 is not addressed here.

LEARNING OBJECTIVES

Upon completion of this module, the student will be able to:

1. Describe the format of the three types of DTE transfer packets.
2. Identify the portions of the terminal data base in terms of the tables and the routines effecting terminal I/O.
3. List the modules used to configure the monitor for terminal I/O with a given system.

RESOURCES

1. Monitor Tables
2. Micro-fiche of the Monitor

MODULE OUTLINE

Front End/Terminals

I. TTY/PTY Device-Dependent Code
   A. TTY Data Base
   B. TTYIN - TTY-Dependent Input
   C. TTYOUT - TTY-Dependent Output
   D. TTCH7 - 2Ø ms. Overhead Task

II. DTE Device-Dependent TTY Code
   A. DTE Data Base
   B. DTE Terminal Output
   C. DTE Interrupts

This page is for notes.

TTY/PTY DEVICE-DEPENDENT CODE

The terminal device-dependent code is called for each character for TTYs and PTYs from the JSYS level code. The device-dependent code exists on several levels; the first level is dispatched to via the dispatch tables TTYDTB for TTYs and PTYDTB for PTYs. Calls to subsequent levels come from the current level.

This code is initially dispatched to through offsets in TTYDTB for input and output. The TTCH7 code is called cyclically from the scheduler overhead cycle; the TTCH7 routine moves characters from BIGBUF to their respective line buffers.

TTY Data Base

DEVICE TABLES AND DEVICE DISPATCH TABLES

Table TTLINV contains the addresses of the device-dependent dispatch tables for each type of line that can be supported by a TOPS-20 monitor.

```
TTLINV: TTFEVT  --console front end lines
        TTMCVT  --MCB lines (not used -- future table for
                  command terminal support on DECnet nodes)
        TTPTVT  --PTY lines
        TTDCVT  --DC10 lines (not used -- historical)
        TTNTVT  --network virtual lines
        TTDZVT  --DZ11 lines
```

A subset of the dispatch tables will exist for a particular hardware configuration. Here is a list of standard configurations and the dispatch tables that would exist for such configurations:

2040 or 2050 -- TTFEVT and TTPTVT
2020 -- TTDZVT and TTPTVT
ARPA net system -- TTFEVT, TTPTVT and TTNTVT

Each entry of a dispatch table is the transfer vector of a particular line-type-dependent function. Each function has the same offset into each dispatch table. Each line type that does not exist in a monitor will dispatch through a

dummy vector table called TTDMVT, with its table name set equal to TTDMVT. Entries in that table return immediately, give an error, or bughalt, depending on the function.

DATA BASE FOR EACH LINE

Tables indexed by line number (resident tables):

TTSTAT - flags and line type (offset into TTLINV table)
TTCSAD - address of special request routine
TTCSTM - time to call special request routine

Tables indexed by line number (non-resident tables):

TTSPWD - input and output line speed
TTACTL - address of dynamic data for line

Dynamic data block for each line:

The dynamic area is created for a line when it becomes fully active. The dynamic area length (TTDDL2 = 26 (octal)) contains items such as:

number of input and output buffers for the line
number of characters in input and output buffers for the line
terminal data mode
byte pointer for removing/adding a character from/to an input or output buffer
terminal characteristics

Sometimes, output needs to be sent to an inactive line, as in one of these cases: a send-all message, "Logging in on local terminals is not allowed", or sending a "ding" when any character but CTRL/C is typed on an inactive line. There are two shortened forms of the dynamic data block that are built for these cases: send-all block and message block. The send-all block is the smallest block; it is not used on KL systems because the front end does the send-all. This block is used on the 2020.

Line buffer allocation:

TTFREB - head of free list for TTY buffers
TTFREC - free buffer count; initialized to NTTBF = 214 (octal)
TTSIZ - 20 (octal) buffer size

Input and output buffers for each line:

When buffers are needed, they are assigned from the free list TTFREB. Each of these buffers is 20 (octal) words long. Routine TTGTBF assigns the requested number of buffers and links them circularly, (that is, the last buffer points to the first). The dynamic data block for the line points to the first buffer and also has a byte pointer for the current character. The output buffer contains characters to be echoed as well as output characters. Currently, two input buffers and two output buffers are assigned for each active line; that is, a line's output buffer consists of two buffers assigned from TTFREB and a line's output buffer consists of two buffers assigned from TTFREB. Each character is stored in a 9-bit byte; therefore, four characters are stored in each word.

Several fields in the dynamic data block for a line are used for byte pointers into the line buffers and as character counts in those buffers. The dynamic data block fields relevant to line buffers are:

TTNIN / count of input buffers
TTNOU / count of output buffers
TTOCT / number of characters in output buffer
TTOOUT / byte pointer for removing characters from output buffer
TTOIN / byte pointer for adding characters to output buffer
TTICT / number of characters in input buffer
TTIOUT / byte pointer for removing characters from input buffer
TTIIN / byte pointer for adding characters to input buffer

TTYIN - TTY-Dependent Input

This is the routine for handling BIN, SIN, and SINR JSYSs. TTYIN is the dispatch address in TTYDTB + BIND.

The code first translates the JFN, TTY device number, or -1 (implies the controlling job's terminal), to an actual line number and checks if it is legal for this process to read from the line. If not, an error is returned.

At this point, one of three choices is made:

1. If the user just issued a BKJFN, repeat the last character.

2. If the terminal is set for binary mode, dispatch for binary input at TCIB.

3. If the terminal is not set for binary mode, dispatch to TCI.

Routines TCI and TCIB both pick up the next character from the input buffer if there is one. If there is no character in the input buffer, the process is blocked with an MDISMS. The MDISMS code adds the fork to TTILST with test data equal to the line number and test routine set to TCITST.

If the process goes into TTY input wait, when it wakes up will be determined by its wake up class. The decision to wake up the process is made when characters are being moved from BIGBUF to their individual line buffers. See the section on TTCH7 for more details.

If deferred echoing is in effect, the character is echoed now (that is, as it is removed from the line buffer and read by the process) if it has not already been echoed. Each character in the input buffer has a flag indicating whether or not it has been echoed; this flag is TTXECO and is on if the character has been echoed. The character will already have been echoed if immediate echoing is on, or, in the case of deferred echoing, if the line was in TTY input wait when the character was moved from BIGBUF to its line buffer. Binary mode does not echo. Therefore, the character is echoed at the time the character is read from its input buffer only if echoing is on and if the TTXECO flag for the character is not on.

If an echo is needed, the character is sent by placing it in the line's output buffer, as with any other character to be output to the line. Deferred echoing assures a clean typescript, that is, characters are not added to the output buffers unless the line is in input wait or the process is reading characters from the line buffer. Therefore, echoed characters will not be scrambled with output.

Some characters are treated specially. For example, if the character being read is a carriage return, it is echoed as a carriage return and a line feed. The TTYIN code also handles such things as raising the case of the character before echoing it if terminal-raise is set.

If the character picked up from the input buffer is an enabled interrupt character, the character is flushed and the code tries again for the next character. An enabled interrupt character is one of the class of characters (such as control characters) that a process can enable a software interrupt on; when an enabled interrupt character is typed, the process gets a software interrupt. In the case of interrupt characters, the interrupt is initiated at the time the character is moved from BIGBUF to its line buffer.

An XOFF or XON will be sent to the line if necessary. The XOFF is sent a few characters before the input buffer is full.

It is an undefined condition to have more than one fork of a job in TTY input wait at any given time. It is impossible to determine which process will receive the character. If this situation occurs, however, the conflict is resolved as follows: all forks in TTY input wait that are inferior to the fork receiving the character are halted; all forks in TTY input wait that are not inferior to the fork receiving the character are put in fork wait.

TTYOUT - TTY-Dependent Output

This is the dispatch address in offset BOUTD of TTYDTB for SOUT, BOUT, and SOUTR JSYSs.

First, determine if the terminal is in binary mode. If it is, transfer to TCOB; if it is not, transfer to TCO. Routine TCOB does not use the CCOC (control character output

control) words for control character translation.

Each terminal has two control character output control words. Each word consists of 2-bit bytes, one byte for each of the control characters (ASCII codes 0-37). The bytes are interpreted as follows:

00: ignore (send nothing) for the character
01: indicate by X (where X is the character)
10: send actual code to terminal
11: simulate format action for character

The terminal data mode may be set so that the CCOC words are used:

1. For both echoing and output (the normal mode)

2. For neither echoing nor output (binary mode)

3. For echo translation only

The CCOC words themselves can be set for the translation of each character for ASCII codes 0-37.

Routine TCO must first check the data mode of the terminal. If the data mode is set for no output translation, routine TCOY is called to output the actual character code.

If the character code is not between 0 and 37, TCOY is called to output the character directly.

If the character is between 0 and 37, several more checks must be made to determine what to do. If the character is an escape, it is handled as a special case: if the corresponding CCOC byte is equal to 01, send ^[; if the CCOC byte is equal to 11, send a $; if the CCOC byte equals 00, flush the character, and if 10, send the character.

If the CCOC byte for the character is a zero, the character is flushed; that is, nothing is sent to the terminal.

If the CCOC byte for the character is 01, the character is converted to its printing equivalent (i.e., ^X) and TCOY is called to print the ^ first and then the X.

If the CCOC bits for the character are set to 10 (send actual character) or to 11 (simulate format action), call TCOY to output the character. In simulating format action, device-dependent code for the terminal type will be called (according to the hardware terminal characteristics). For example, a form feed is handled differently on a VT50 than on an LA36.

The code beginning at TCOE is also called for echoing a character. TCOE will be called from TCI or TTCH7 (the overhead cycle task to move characters from BIGBUF to their individual line buffers), depending on when the character is echoed.

TCOY

Routine TCOY is the second level TTY device-dependent routine. It handles a device's idiosyncracies and spacing (e.g., adding a CRLF sequence if at right margin). Here, also, lower case is converted to upper case for terminals that do not support lower case and ^ is placed in front of the character if the character is upper case and terminal-indicate is turned on.

Special characters such as form feed or tab are handled based on the terminal characteristics by dispatching through table CHITAB. Also, any necessary padding is done here.

TCOUT

TCOY calls TCOUT, which is the third level TTY-dependent output routine. TCOUT handles the parity bit by dispatching through offset TTVT12 of the TTXXVT table for terminal type XX.

If the line is currently linked to another terminal, routine TTLNK3 is called to output the character to the other terminal as well.

If output is currently control Oed, return because output is to be discarded.

If there is no room in the output line buffer for this character, set up a scheduler test of line-number,, TCOTST and give an error return. The calling code will MDISMS and put the fork on WTLST. The fork will become runnable again when the characters currently in the output buffer have been sent to the terminal. WTLST is checked for processes that have become runnable in one of the long cycle tasks.

If there are no output buffers for the line, routine TTGTBF is called to assign two output buffers for the line.

The character is then stored in the output buffer using the byte pointer in offset TTOOUT of the line's dynamic data area and the count of characters in the output buffer is incremented.

If output is not active on the line, routine STRTOU is called to start output. This routine dispatches through the TTVT13 offset of the TTXXVT table where XX is the terminal type.

Terminal type:	Dispatch address in TTXXVT+TTVT13:
TTFEVT	STRTO1 -- calls DTESRV to send the character out to the line. See section on TTFEVT functions.
TTPTVT	STRTØ2 -- converts internal line number to PTY number.

TTCH7 - 20 ms. Overhead Task

TTCH7 moves characters from BIGBUF to their respective line buffers. Characters are added to BIGBUF at interrupt level by either the DTE interrupt code for systems with front end lines, or, for the 2020, by the DZ11 interrupt code. For systems using RSX20F front end lines, the DTE transfers packets of characters to BIGBUF from the front end. For 2020 systems, the characters are added to BIGBUF one at a time when the interrupt from the character is handled.

An entry in BIGBUF is one word with the line number in the left half and the character in the right half. Cell TTBIGC contains the count of characters currently in BIGBUF. TTBIGO has the address of the next character to be removed from BIGBUF.

Routine TTCH7 is called cyclically from the scheduler overhead cycle, currently during the short clock cycle.

If TTBIGC, the count of characters in BIGBUF, is non-zero, pick up the next entry from BIGBUF using pointer TTBIGO. This will be an entry in the form line number,, character. Decrement TTBIGC, which is the count of characters in BIGBUF.

Dispatch through offset TTVT26 of TTXXVT line-type-dependent table, where XX is the line type. These routines return immediately for all supported line types and it exists for historical reasons only.

Some entries in BIGBUF may not be characters destined for input line buffers. Such entries will have the line number in the left half and either flag TTOIRQ, DLSRCF, or DLSCXF in the right half. These entries are placed in BIGBUF to flag conditions that are noted at DTE interrupt level but cannot be handled due to interrupt routine time constraints. If flag TTOIRQ is on in an entry, the line's output buffer was emptied and the process has asked for an interrupt when the output buffer became empty. When this type of entry is picked up from BIGBUF, a software interrupt for terminal code .TICTO is generated for the process. Flag DLSXCF indicates carrier transition was noted; flag DLSRCF is ignored. After handling any one of these conditions, loop back to TTCH7 for more characters.

If the entry picked up from BIGBUF is an input character, call routine TTCHI to place the character in the appropriate line buffer. TTCHI will initiate an interrupt if the character is an enabled interrupt character, echo the character if appropriate, and wake up the process if it was in TTY input wait and the character is in the wake up class set for the terminal. After returning from TTCHI, loop to TTCH7 for the next character. If there are no more characters, control will transfer to TTCH7X to handle any special line requests.

When BIGBUF is empty, control is transferred to TTCH7X and if there are any special requests queued, they are handled here. If there are any requests, cell TTQCNT will contain the count of requests. Special requests are set up for front end lines, primarily when the buffer space for TTY packets is full. For the 2020, the queued requests will be checking for data set line conditions, such as carrier on.

Requests are handled for a maximum of eight lines beginning with the line number in TTCQLN. Routine DOLINE is called to handle each line's request. The requested function's routine address is in field TROUT of offset TTCSAD in the line's dynamic data block. The function may have a designated time at which it is to be performed; the time is stored in the parallel table TTCSTM.

DTEQS IS A TABLE CONTAINING THE QUEUE HEADER FOR EACH DTE. DTEQS + DTE NUMBER IS THE QUEUE HEADER FOR A PARTICULAR DTE.

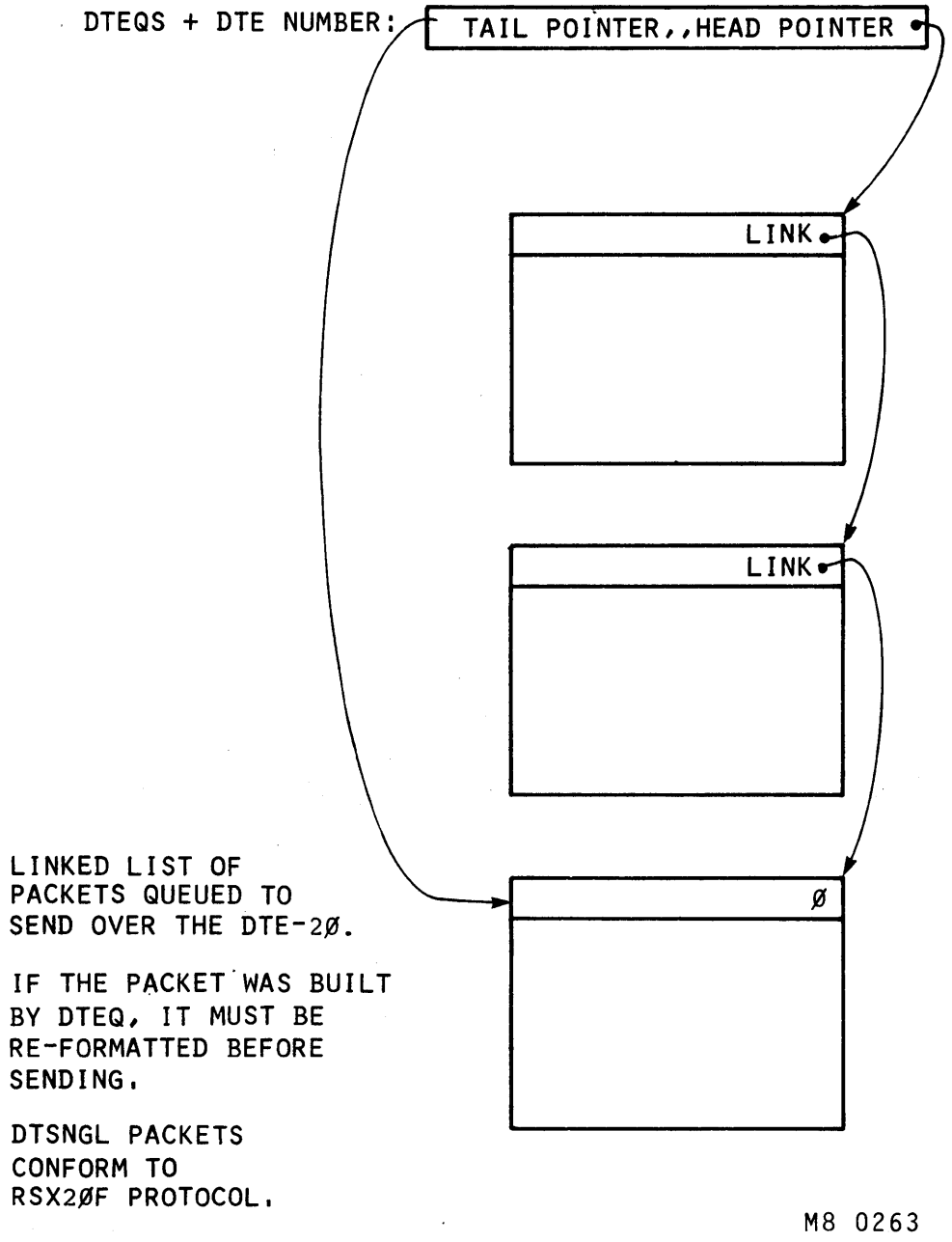


Figure FE-1. DTE Packet Queue

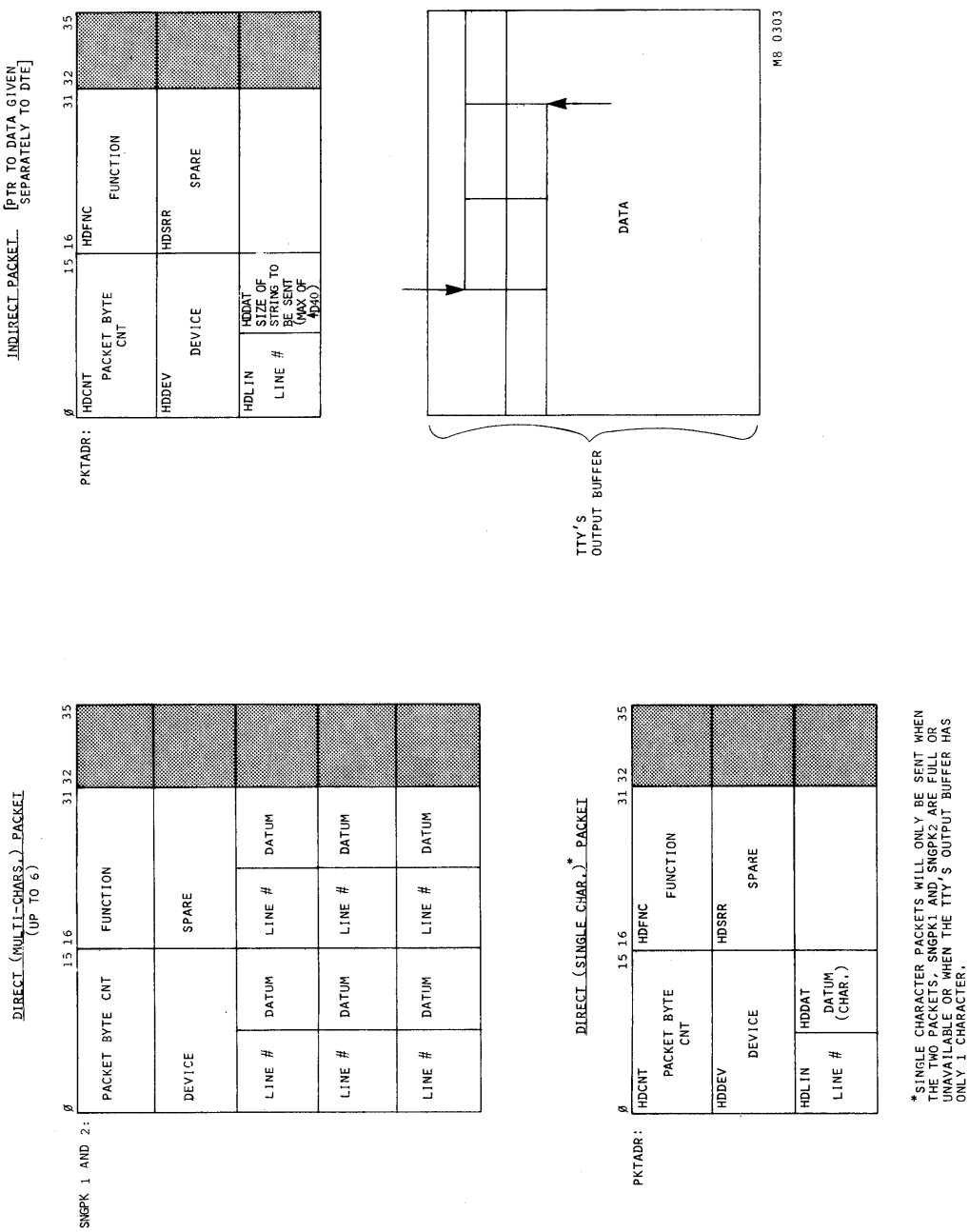


Figure FE-2. Packet Types Sent Over DTE (RSX20F Protocol)

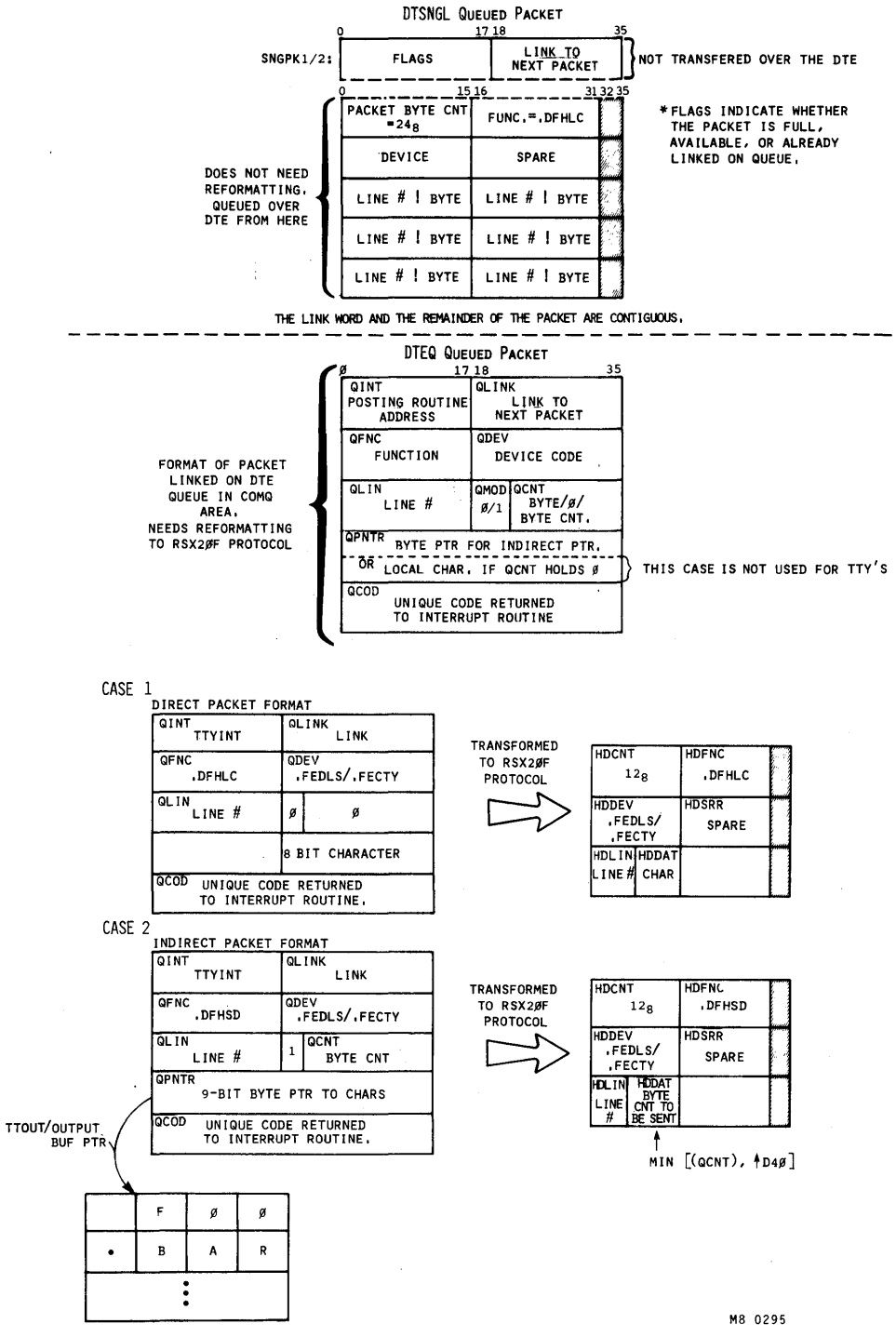


Figure FE-3. DTE Packets

DTE DEVICE-DEPENDENT TTY CODE

For front end lines, characters are assembled as packets and tranferred over the DTE. For output, packets consist of characters from output line buffers to be sent by the front end to the terminal lines. For input, the front end gathers characters from the terminals, sends them over the DTE, and the DTE interrupt code places them in BIGBUF. TTCH7, one of the 20 ms. scheduler tasks, moves the characters from BIGBUF to their line buffers, as described in the section above.

DTE Data Base

Data is sent across the DTE in packets composed of header or overhead information, and the characters to be transferred. Packets may be either direct or indirect. If the header block is followed immediately by the characters, the packet is a direct packet. If the header block and characters are not contiguous, the packet is an indirect packet.

When a packet is sent to the front end, its header will always have the following format:

HDCNT - count in bytes	HDFNC - function	
HDDEV - device	HDSPR - spare	
HDLIN|HDDAT or HDDT1		

Some packet headers need to be reformatted to conform to RSX20F protocol before they are sent. All packets queued by the routine DTEQ need reformatting, but packets queued by DTSNGL do not.

Each DTE has a transfer wait queue. The header for each queue is in table DTEQS and the header for a particular DTE is in DTEQS, indexed by DTE number. Each DTE queue header word has this format:

```
tail pointer,,head pointer
```

The queue is a linked list and the first word of each packet contains the link to the next packet (or a 0 if the packet is last on the queue).

Now, we will look at how TOPS-20 initiates DTE transfers for TTY output, and later, at the interrupt code that fills BIGBUF.

DTE Terminal Output

Three different data structures, or packets, are built to transfer characters from an output line buffer over the DTE to the front end. The type of packet used depends on whether the line has output that is currently active (i.e, output queued or being transferred to the front end by the DTE) and whether there are any packets available for use.

If the line currently has no active output, routine STRT01 will call DTSNGL. DTSNGL has two standard packets that it fills and queues; their purpose is to start I/O for lines which are inactive. These standard packets have the format shown above. (See DTSNGL Queued Packet.)

If there is no room in either of DTSNGL's standard packets, routine DTEQ is called to build a packet for that single character. This packet has the format shown above. (Direct Packet Format)

If DTEQ is unable to get space for a packet, and cannot block to wait for the space, a request to have the packet built later is set up. Blocking is not possible for one of the following reasons:

1. The process is NOSKED
2. The request to queue the packet was made by the scheduler (TTCH7 20 ms. task)
3. The request to queue the packet was made at DTE interrupt level

When the TTCH7 task has finished emptying BIGBUF, it services special requests for terminal lines; a maximum of eight requests will be serviced in one call to TTCH7. A special line request is made by storing the address of the

routine to be called in table TTCSAD (indexed by line number) the time to call the routine in TTCSTM (also indexed by line number) and incrementing TTQCNT, which is the count of special line requests.

When a packet for a front end line cannot be queued because there is no packet space, a special line request is set up for that line. TTSN10 is the routine address stored when the request is to queue the packet; this is the only type of special request made for a front end line. TTCH7 calls the routine TTSN10, which tries to get packet space to queue the packet. DZ11 lines (on 2020 systems) use the special request mechanism for carrier off line and other related conditions handled by the front end on systems with a front end.

Once a line has been activated by one of the methods above (i.e., the first character has been output), the TTYINT routine, which is called when a DTE transfer done interrupt occurs for a TTY packet, will queue the next packet for the line if there are any characters in the line's output buffer. If the packet whose transfer was completed was a DTSNGL packet, TTYINT will be called for each line that had a character in the packet. If there are at least two characters in a line's output buffer, TTYINT will call DTEQ to build a string data packet. String data packets are indirect packets and have the indirect packet format. String data packets hold a maximum of 40 (decimal) characters; this limit is imposed by RSX20F buffer space. Note that the packet built is an indirect packet. The characters will be in the line's output buffer and the packet header byte pointer will point there.

The normal sequence of events for a line that is doing output is:

1. The line buffers are assigned only while they have characters in them; therefore, the first two output line buffers must be assigned.

2. The FIRST character is queued via DTSNGL. This will add the character to one of DTSNGL's standard packets. TTOTP in the line's dynamic data block is set to indicate the line is active.

3. When the transfer done interrupt occurs, TTYINT will pick up any characters in the line's output buffer and call DTEQ to build a string data packet for the characters.

NOTE

TTYINT is called for each line that has a character in the DTSNGL packet; therefore, a packet is built and queued for each line (if it has characters to output).

4. Each time a transfer done interrupt occurs, TTYINT will build an indirect (string data) packet for the line if there are at least two characters in the line's output buffer.

5. If there is only one character in the line's output buffer, DTSNGL will be called to add that character to one of its standard packets. This is done to keep to a minimum the number of single character packets that are sent.

6. If the line has no characters, its state is changed to inactive. Flag TTOTP indicates whether or not a line is active.

7. The line is now inactive and the output buffers are released. When there is a character for the line, the output buffers must be allocated and the line must be reactivated as described before.

A TTOTP flag indicating that a line is active means that at the time a character is added to the line buffer, there is no need to worry about getting the character out because the TTYINT routine will queue another packet when the currently active packet for the line gets a transfer done interrupt.

Note that DTEQ called to send a single character (because the DTSNGL standard packets are full) is really a special case of the usual call to DTEQ by TTYINT; that is,

all calls to DTEQ will construct a packet with characters in it for a single line. If only one character is being sent, DTEQ will use a direct packet; if multiple characters are being sent, an indirect packet is built.

If a character is being echoed from TTCH7, and the line must be activated, and if there is no room for the character in one of the DTSNGL standard packets, DTEQ is not called. Instead, a special request is added to the queue that is examined when TTCH7 has finished emptying BIGBUF.

When TTYINT sets up a string data packet for DTEQ, it will never send beyond an escape character. An escape character is a a 9-bit byte with its high order bit on. The only defined escape character right now is stop output, which is used for terminal page (so it will not output more than one screen at a time). Routine FNDEND is called to scan the output buffer, and if an escape character is found, only the characters up to the escape character are sent in this packet. If the next character in the output buffer is the stop output escape character, output cannot be reactivated until a CTRL/Q is typed by the user.

TTYINT will be called for all TTY output transfer done interrupts. It is set up in the packet as the posting routine to call on transfer done. If the packet was queued by DTEQ, there will always be a posting routine address in the left half of the first word of the packet; the posting address is passed to DTEQ by the caller. DTSNGL character packets also call TTYINT, although TTYINT is not set up in their packet header as the routine to call. If the two high-order bits in the first word of the packet are on, the packet is a DTSNGL, and TTYINT is called as the posting routine.

TTYINT will call DTEQ to send the characters, if there are at least two. If there is only one character, TTSND is called which, in turn, calls DTSNGL to add the character to one of its standard packets.

All packets, whether TTY output or line printer output, are put on queue DTEQS. Each DTE has its own offset into the queue. Routine DTESKD is called to start the transfer on the top packet in the queue, which is the active packet.

DTSNGL packet headers and DTEQ packet headers do not have the same format. DTSNGL builds packets that conform to queued protocol, while the packets built by DTEQ must have their headers re-formatted to conform to queued protocol. Routine DTESKD takes the top packet from DTEQS, re-formats it if it was built by DTEQ, and sends it to the DTE. DTESKD knows the packet is a DTSNGL packet if bits 0 and 1 of the first word of the packet are on; a DTSNGL packet is sent as it is.

DTE Interrupts

All DTE interrupts come to INTDTE. This routine checks the reason for the interrupt and dispatches on type:

1. TO10DN -- TO-10 transfer done (input)
2. DINGME -- 10 doorbell
3. TO11DN -- TO-11 transfer done (output)

10 TO 11 TRANSFER DONE INTERRUPT (OUTPUT)

TO11DN will be called when a transfer over the DTE to the 11 is complete; that is, when an output request finishes. There are two basic tasks to be completed in this case:

1. Schedule the next DTE transfer
2. Post the packet

Routine DTESKD schedules 10 to 11 DTE transfers. The first entry in a DTE's queue will be sent. (This will be the entry pointed to by the right half of DTEQS, indexed by DTE number. Remember that the DTE can transfer simultaneously in both directions, so it will not be busy in the to 11 direction when DTESKD is called.)

The packet header will have the name of the posting routine to call (in offset QINT) unless the packet was queued by DTSNGL. For TTY output, this will always be set

up to TTYINT. The posting of TTY packets includes some housekeeping and queuing of other packets that need to be sent for the line (or lines) in the packet whose transfer was just completed.

Housekeeping means updating the relevant data structures to say the data has been successfully sent. In the case of TTY transfers, this means updating the count of characters in the line's output buffer (field TTOCT in the line's dynamic data block) , updating the byte pointer to take characters from the output buffer (field TTOOUT in the line's dynamic data block), and, if there are no more characters in the output buffer, setting the line to inactive (field TTOTP set to zero). If the output buffers are empty, they are released at this time.

If there is only one character in the output line buffer, send the single character by calling TTSND (which will call DTSNGL). To activate a line, see the mechanism described in the section above on DTE Terminal output.

If there is more than one character, set TTYINT as the interrupt handler and call DTEQ to send the characters. See the description of DTEQ in the DTE output section.

11 TO 1Ø TRANSFER DONE INTERRUPT (INPUT)

TO1ØDN will be called when a transfer over the DTE from the 11 to the 1Ø is finished. The major task of this code is to put the packet in its proper buffer. For TTYs, this will mean putting the characters into BIGBUF.

The TO1ØDN code dispatches, depending on the function code in the packet through table FNCTBL in DTESRV. TTY character input dispatches to TAKLC, which calls routine DTESTO for each character. DTESTO calls BIGSTO to store the character into BIGBUF, using pointer TTBIGI to place characters. Each entry in BIGBUF is one word containing a line number in the left half, and the character in the right half. Since BIGBUF is a circular buffer, if TTBIGI reaches TTBIGO, BIGBUF is full and the monitor BUGCHK's. When all characters in the packet have been moved to BIGBUF, the interrupt can be dismissed.

MODULE TEST

When answering the lab exercises, write down the names of the tables where you found the answers. The labs will help you understand the monitor data base structure. So remember, where to look is more important than what you find there.

The exercises marked with a double star (**) are more difficult and are optional. If you have the time and/or motivation, do them.

I/O - TTY Dependent Data Base

TOOLS

Use the PEEK command of FILDDT; i.e., look at the running monitor to answer these exercises.

EXERCISES

1. What TTY line types exist in this monitor?
2. What is the address of your terminal line's dynamic data block?
3. Is anyone linked to your line? **
4. What is your line speed set to?
5. What line-type is your line?
6. How many line buffers are currently on the free list? **
7. What device-dependent routine is called to send a character for your line's line-type? **

DTE - DTE Dependent Data Base

This set of exercises also uses a deliberate crash.

TOOLS

Use the crash <MONITOR-INTERNALS>DTEQS.CPY for this set of exercises. For symbols, use the monitor <MONITOR-INTERNALS>R3-MONITOR.EXE. Do not forget to set monitor context!

EXERCISES

1. Are there any packets queued to be sent over the DTE?

2. How many characters are in each of the DTSNGL packets?

3. Which routine (DTEQ or DTSNGL) built each of the queued packets?

TEST EVALUATION SHEET

I/O - TTY Dependent Data Base

EXERCISES

1. What TTY line types exist in this monitor?

   ANSWER: If a line type exists, it will have a line-type-dependent dispatch table pointed to by the appropriate entry in TTLINV. If a line does not exist, the offset in TTLINV for that line-type will point to the dummy table TTDMVT.

```
TTDMVT=43300

TTLINV/ 400000,,TTFEVT = 400000,,43354
      / 400000,,TTNTVT = 400000,,43300 ;does not
                                       ;exist
      / 400000,,TTPTVT = 400000,,44235
      / 400000,,TTNTVT = 400000,,43300 ;does not
                                       ;exist
      / 400000,,TTNTVT = 400000,,43300 ;does not
                                       ;exist
      / 400000,,TTNTVT = 400000,,43300 ;does not
                                       ;exist
```

   The table TTLINV says that offset 0 is for front end lines and offset 2 is for PTYs; therefore, these two types of lines exist for this monitor.

2. What is the address of your terminal line's dynamic data block?

   ANSWER: Given your job number, you can get your line number from JOBPT, indexed by job number, (in the left half). Table TTACTL, indexed by line number, contains the address of the line's dynamic data block.

```
$INFORMATION JOB
Job 26, User DONALEEN, CD:<DONALEEN>,
```

```
Account LSCD, TTY32
$CONT

JOBPT+26./ 32,,56

TTACTL+32/ 556235        ;line's dynamic data
                         ;block
```

3. Is anyone linked to your line? **

   ANSWER: If there are any lines linked to this one, the line numbers are stored in the line's dynamic data block at offset TTLINK; there are four 9-bit fields in this word to store a maximum of four links.

```
556235+TTLINK/ -1        ;no links
```

4. What is your line speed set to?

   ANSWER: The line speed for a terminal is stored in table TTSPWD, indexed by line number.

```
TTSPWD+32/ 15Ø,,96ØØ     ;(decimal)
```

5. What line-type is your line?

   ANSWER: A terminal's line-type is stored in bits 12-17 of table TTSTAT, indexed by line number.

```
TTSTAT+32/ 24ØØØ         ;this line's type is
                         ;Ø - i.e., this is
                         ;a front end line.
```

6. How many line buffers are currently on the free list? **

   ANSWER: Location TTFREC contains the count of free buffers.

```
TTFREC/ 154
```

7. What device-dependent routine is called to send a character for your line's line-type? **

   ANSWER: Table TTLINV, indexed by line type, contains the address of the line-type-dependent function table for a line type; that table, at offset TTVT32 contains the dispatch address to send a character to a line.

```
TTLINV/ 400000 TTFEVT          ;vector table for
                               ;this line type

TTFEVT+TTVT23/ 400000 TTSND1   ;TTSND1 is the
                               ;routine to send
                               ;a character to
                               ;this line type.
```

DTE - DTE Dependent Data Base

EXERCISES

1. Are there any packets queued to be sent over the DTE?

   ANSWER: Table DTEQS, indexed by DTE, contains the queue header for packets queued to be sent over the DTE.

```
DTEQS/ 126040,,126033          ;tail,,head
                               ;of queue
     / 0
```

2. How many characters are in each of the DTSNGL packets?

   ANSWER: The DTSNGL packets are called SNGPK1 and SNGPK2; SNGPK1 is queued to be sent (and therefore, can be expected to have valid unsent data in it). The byte count for a packet includes the header portion, but not the first word because this is only for queueing and is not sent over the DTE. An empty packet (i.e, header portion only) contains 10 bytes. There is room in a DTSNGL packet for six characters total; each character

takes two bytes -- one byte for the character and one byte for its line number. Therefore, a full packet can contain 10 header bytes plus 14 character bytes equals 24 bytes. Flag SNGAVL in the header word indicates there is room in the packet.

```
SNGPK1/ 700001,,126052  ;SNGAVL= bit 1;
                        ;SNGACT= bit 2
      / 0,12,0,4        ;(type out set to
                        ;8-bit bytes)
                        ;there are 12 bytes
                        ;in this packet.
                        ;Therefore, there is
                        ;one character.
      / 0,4,0,0
      / 17,215,5,317    ;line number and
                        ;character
      / 0,0,0,0

SNGPK2/ 0               ;packet is not
                          ;queued
```

3. Which routine (DTEQ or DTSNGL) built each of the queued packets?

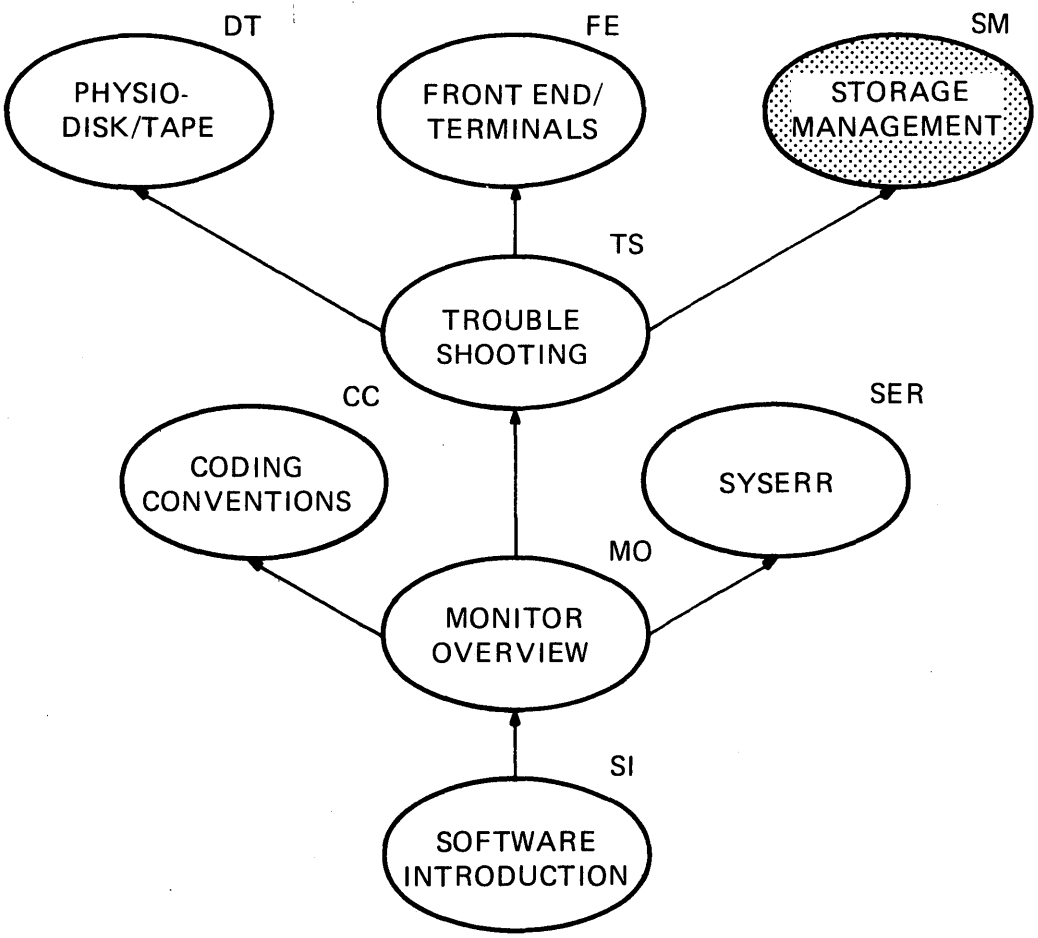
   ANSWER: Only packets SNGPK1 and SNGPK2 are used by DTSNGL; therefore, only SNGPK1 of the queued packets was built by DTSNGL.

TOPS-20 MONITOR

Storage Management

<<For Internal Use Only>>

COURSE MAP



MR-2717

This page is for notes.

Storage Management

INTRODUCTION

This module covers the portion of TOPS-20 which is responsible for storage management. The areas addressed by storage management include: the physical location of each page, the movement of pages between the several levels of storage, the manangement of reserved space, garbage collection, working set control and balance set adjustment. The emphasis in this module is on data base tables associated with the tasks mentioned above, and refers to the table descriptions section of the course materials, along with the micro-fiche.

LEARNING OBJECTIVES

Upon completion of this module, the student will be able to:

1. Describe the uses of the data contained in the various storage management related tables, specifically, the CST tables, the SPT and parallel tables, and the JSB/PSB space.
2. List the uses of the dynamically allocated monitor space.
3. Describe the uses of the JSB and the PSB in context switching.

RESOURCES

1. The Monitor Table Descriptions
2. The TOPS-20 Micro-fiche

MODULE OUTLINE

Storage Management

I. Storage Management
   - A. Introduction
   - B. Data Structures
   - C. CST Tables
   - D. SPT and Parallel Tables
   - E. Working Set Management
   - F. System-Wide Page Management
   - G. Page Faulting
   - H. Adjustment Of the Balance Set
   - I. SWPIN and SWPOUT

II. JSB/PSB Space
   - A. Context Switching the JSB and PSB
   - B. JSB and PSB Maps
   - C. Use of JSB Space
   - D. Use of PSB Space

This page is for notes.

STORAGE MANAGEMENT

Introduction

Storage management is responsible for each page's location, changing a page's level of storage, garbage collection, and reserving storage for forks. These responsibilities are implemented in:

1. Working set management
2. System wide page management
3. Page faulting
4. Adjustment of the balance set

Data Structures

PAGE TABLES

Both forks and files have page tables. (A file's page table is also called its index block). While a file (or a section of a long file) is in use, its index block is copied into core; this copy cannot be written to disk (but can be swapped to "drum") and is called the in-core copy of the index block. Each page of the file that is currently mapped has a share pointer or an immediate pointer in the in-core copy of the index block.

STORAGE ADDRESSES

The page table entry for a page tells you where to find the page's storage address. If the pointer for the page is an immediate pointer, the storage address is in bits 14-35 of the page table entry. If the pointer for the page is a share pointer, the storage address is in bits 14-35 of the SPT slot that belongs to the page.

CST Tables

The CST tables are parallel tables; there is one word entry per page of physical core in each table. The amount of physical core determines the size of these tables. The CST tables are all resident.

CST0

CST0 is the only CST table that both the microcode and the software use. The microcode writes into the CST0 table and the software uses the information. A CST0 table entry is divided into three fields: the age, the PUR (process use register), and the modify bit. See the CST0 table in the Monitor Tables for a diagram.

The age field is in bits 0-8; when a page is referenced, the running fork's current age is stamped in the age field of the page's CST0 entry (if the fork has not referenced the page since its last age increment -- or since it was context switched).

The Hardware Reference Manual incorrectly states that the age is in bits 0-5. The age field is actually in bits 0-8, but if the value stored in this field is not at least 10, the microcode traps to the software. Note that a number smaller than 10 in this field would be only in bits 6-8.

The PUR (process use register) is in bits 9-32. The PUR is used to indicate if a page has been referenced by more than one process; that is, whether or not the page is shared. Each process in the balance set is assigned a bit (called its core number) in the range of bits 9-32. When a process references a page, its core number is inclusively ORed into the page's PUR in its CST0 entry; this happens only if the process has not referenced the page since its last age stamp increment (or since it was context switched).

The modify bit indicates if a page has been modified.

CST1

The lock count and next level of storage for a page are stored in its CST1 entry. A page cannot be swapped out while its lock count is non-zero. The overhead pages for a fork are locked when the fork enters the balance set, and

pages being read or written are locked when the page is queued. The lock count can be greater than one; for example, if two forks from the same job are in the balance set, the job's JSB is locked twice.

CST2

The CST2 entry for a page indicates its "home" -- either its owning SPT slot, or its owning page table's SPT slot and its offset in that page table. The CST2 entry for a page is the internal identity on that page; it uniquely identifies any page on the system. If a core page is not assigned, its CST2 entry is zero.

CST3

The CST3 table is used for multiple purposes, depending on the state of the page. If the page is on the RPLQ, its CST3 entry contains the list pointers for the RPLQ. If the page is on the deletion queue or special memory queue, its CST3 entry is the list pointer for that queue. When a core page is queued to be read from disk or written to disk, its CST3 entry contains the local disk address.

CST5

The CST5 table is used for short IORBs. If a page is linked on a disk unit's transfer wait queue or position wait queue, the entry in CST5 for the core page being read to or from contains status flags and the pointer to the next item in the queue.

SPT and Parallel Tables

The SPT and SPTH are parallel tables. SPTO is parallel to the OFN portion.

SPT

The SPT contains the storage addresses for all fork page tables, JSBs, PSBs, in-core copies of index blocks, and file pages with share pointers in bits 14-35. The OFN portion of the SPT also contains the ALOCX index for the OFN in bits 0-11; the remainder of the SPT table keeps the share count on the SPT slot in bits 0-11. For a shared file

page, this is the number of pointers to the SPT slot.

SPTH

The SPTH table contains flags and the disk address of the index block in the OFN portion. The remainder of the SPTH table slots are used to identify the page's origin. If the page is a file page, its SPTH slot contains the owning OFN and the offset (page number) into that index block. If the page is not a file page, its SPTH slot is the owning fork number.

SPTO

The SPTO table contains the OFN's share count and the structure number the file is from.

DISK BIT TABLE

Each structure has a file in its <ROOT-DIRECTORY> called DSKBTTBL.BIN. This file contains one bit for each page in the structure. The file is mapped whenever a page is allocated or deallocated on that structure. If the bit for a page is Ø, the page is allocated; if it is a one, the page is free. See the write-up on disk allocation for more information.

FORK AGING

A fork always has an age associated with it based on the amount of time it has run. When a fork is created, it is assigned an age of 1ØØ; each time it has accumulated AGTICK ms. (currently 4Ø ms.), the fork's age is incremented. The fork's current age is stored in the FKNR table, indexed by the system fork number.

RPLQ

RPLQ, which is linked through the CST3 table, is the header for the linked list of available core pages. NRPLQ is the count of pages on this free list. A page is added to the head of the RPLQ if it is not expected to be referenced again; for example, deleted pages are added to the front of

the RPLQ. A page is added to the end of the RPLQ if there is a high probability the page will be referenced again; for example, index blocks are added to the end of the RPLQ.

When the system needs a core page, it takes one from the RPLQ. For example, when a process page faults (for not-in-core), the system takes a page from the RPLQ to swap the page into. When NRPLQ (the number of pages on the RPLQ) gets low (currently NRPMIN), the overhead cycle does a global garbage collect (GCCOR).

WORKING SETS

Each fork has both a reserve working set size and a current working set size. The current working set size is the count of pages currently in core. It is stored in table FKPGS and indexed by the system fork number. The reserve working set size is the count of pages reserved for the fork when it is in the balance set; the reserve working set size is stored in table FKNR, indexed by the system fork number. The reserve working set is always at least as large as the current working set.

Working Set Management

Pages are removed from a fork's working set by the local garbage collector (called XGC). The local garbage collector is invoked when a process page faults and the fork has run at least 2000. ms. since its last local garbage collection. The CST tables are scanned for pages to collect. A page is removed if:

1. The last reference to the page was more than 3 seconds ago.

2. It is private. The PUR must have only this fork's core number stored.

3. It is not locked.

4. It does not have write in progress.

System-Wide Page Management

GCCOR

The overhead cycle checks if the RPLQ has at least NRPMIN pages; if the free list is low or if a fork is being deleted, the global garbage collector (GCCOR) is called to collect pages for the free list. GCCOR scans the CST tables and takes pages:

1. Whose owning fork is outside the balance set. If the PUR indicates the page is shared and the owner is on the GOLST, it is left. If the owner is not on GOLST and there are sharers, the page is collected anyway.

2. That are not locked.

3. That are not being written.

GCCOR collects a maximum of 1/2 of swappable core. Note that even if a page is added to the free list, its owner can still reclaim the page. See the diagram on re-claiming a page from the RPLQ.

DDMP

DDMP is a Job 0 fork that runs in exec mode; its task is to move file pages from the swapping space back to disk. This both cleans up the swapping space and updates the disk copy of the page. DDMP is invoked both cyclically (every minute) and when the swapping space gets low.

Page Faulting

Although page faulting often refers to a page that is not in core, there are actually several types of page faults. A page fault occurs when the microcode traps to the monitor because it cannot complete a reference for one of several reasons. Page faults are divided into hard and soft types; if flag TWHPFF is on in TRAPSW, it is a hard page fault. The soft page fault types are:

1. NIC - not in core. When the microcode did the address translation, it found the storage address was not in core.

2. TRPØ - illegal age. The age field contains zeroes in bits Ø-5. This means the page is in a special state. (For example, a read may be in progress.) See the CSTØ table for an explanation of each of the special states.

3. NPG - null pointer. The page table entry is zero. The page must be created.

4. WCPY - copy on write. There has been a write reference to a page with copy-on-write access set.

5. ILWR - illegal write. A write-protected page has had a write reference.

6. ILRD - illegal read. A reference was made to a read-protected page or to a page from a file on a structure that has been dismounted.

Adjustment of the Balance Set

The balance set defines which processes have reserved core. The balance set is adjusted when one of the following is true:

1. SUMNR is greater than MAXNR. The number of pages needed by balance set processes exceeds available core.

2. It is time for the periodic adjustment of the balance set.

These factors determine whether a process is added to the balance set:

1. GOLST priority

2. If it will fit in its partition.

SWPIN and SWPOUT

The routines SWPIN and SWPOUT are called for all storage management functions that require a page to be read or written. These routines call DSKIO or DRMIO to set up the request that is passed to PHYSIO to queue the page. The SWPOUT routine may not need to write the page if the page has not been modified. Also, the SWPOUT routine can decide to swap a file page to its disk home if the swapping space is low. The following system routines and JSYSs call SWPIN and/or SWPOUT:

1. Page fault not-in-core calls SWPIN.
2. Page fault null pointer calls SWPIN to assign and zero a core page if a private page is created.
3. Page fault copy-on-write calls SWPIN to assign a core page and copy the source page to the new page.
4. GCCOR calls SWPOUT for collected pages.
5. XGC calls SWPOUT for collected pages.
6. DDMP calls SWPIN to read pages in from the "drum" and then calls SWPOUT to write them to their homes on disk.
7. AJBALS calls SWPIN to read in a fork's overhead pages when it is added to the balance set.
8. UFPGS JSYS calls SWPOUT to update file pages on the disk.
9. CLOSF JSYS can call SWPOUT to write any modified file pages to the disk.
10. PMAP JSYS can call SWPIN or SWPOUT, depending on the requested function.

JSB/PSB SPACE

Each job has a JSB (Job Storage Block) and each process has a PSB (Process storage Block). The JSB and PSB associated with a fork are context switched with that fork. Also, all virtual pages between 620 and 777 of the monitor's address space are context switched with the process. And, for Model B machines, sections 2 and 3 are context switched with the process. How these parts of the monitor are used and context switched is explained below.

Pages 620-706 are called the JSB space and contain job specific information. Pages 707-777 are called the PSB space and contain process specific information. On model B machines, sections 2 and 3 also contain process specific information; namely, the process's currently mapped directory and the IDXFIL for the structure the process accessed last.

Context Switching the JSB and PSB

Each JSB and PSB has its own SPT slot. When a process is running, its JSB is currently mapped to page 620 of the monitor's map and its PSB is currently mapped to pages 776 and 777 of the monitor's map. Therefore, to context switch, all that is necessary is to set up share pointers in MMAP slots 620, 776, and 777 for the fork's JSB and PSB pages. By changing the map slots, the virtual address space of the monitor is changed to include the current fork's PSB and JSB pages. The fork tables FKJOB, FKPGS, and FKCNO (all indexed by fork number) contain the SPT offsets for a fork's JSB and PSB pages.

However, as discussed above, many more pages than just the JSB and PSB proper are context switched with the process. It is possible to change both the MMAP entries for each of the pages that is context switched and the section pointers for sections 2 and 3; but the cost in time would be exorbitant.

JSB and PSB Maps

To simplify context switching, part of the JSB is used as a page map for the JSB space pages and part of the PSB is used as a page map for the PSB space pages. The MMAP slots for these pages are indirect pointers through the JSB and PSB, respectively.

To avoid having to set up SPT offsets for the JSB and PSB, the monitor reserves SPT slot 400 for the current JSB, and SPT slot 401 for the current PSB page containing the PSB space map. When a process is context switched, the monitor gets its JSB and PSB SPT offsets from the FKJOB and FKPGS tables, picks up the current storage addresses from the relevant SPT slots, and copies the JSB's storage address to SPT slot 400 and the PSB's storage address to SPT slot 401.

Suppose fork 71 needs to be context switched, and the FKPGS and FKJOB entries for the fork contain the following:

```
FKJOB+71/ xx,,503       ;503 is the SPT offset for
                        ;the JSB belonging to this
                        ;fork's job

FKPGS+71/ xx,,451       ;451 is the SPT offset for
                        ;the PSB page
```

The storage addresses for these pages are in their SPT offsets. Suppose the SPT offsets contain the following:

```
SPT+503/ 300,,213       ;the page is currently in
                        ;core page 213

SPT+451/ 100,,451       ;this page is currently in
                        ;core page 451
```

To context switch, copy the current storage addresses above to SPT slots 400 and 401, respectively.

What must MMAP look like to make this work? MMAP+620 contains a share pointer for SPT slot 400. MMAP+621 through MMAP+706 contain indirect pointers through the JSB map in the JSB (that is, indirect pointers through SPT slot 400). The JSB map begins at 620000, the start of the JSB. Therefore, the JSB space is represented in MMAP as:

```
MMAP+621/ 3xxxx1,,400
MMAP+622/ 3xxxx2,,400
MMAP+623/ 3xxxx3,,400
        .
        .
        .
MMAP+706/ 3xxx66,,400          ;indirect through
                               ;offset 66 in the
                               ;JSB.
```

For the PSB, things are a little more complex. Remember that the PSB is also the UPT; therefore, several locations in the PSB have hardware-defined uses. For this reason, the PSB map cannot begin at the start of the PSB page. The PSB map begins at offset 666 of the PSB. Therefore, the PSB space is represented in MMAP as:

```
MMAP+707/ 3xx666,,401
MMAP+710/ 3xx667,,401
        .
        .
        .
MMAP+775/ 3xx754,,401
MMAP+776/ 2xxxxx,,401          ;share pointer to
                               ;PSB page itself
MMAP+777/ 3xx756,,401
```

Use of JSB Space

The JSB space is divided into reserved and dynamically allocated areas.

RESERVED JSB SPACE

Beginning at location FILSTS, space for 140 JFN blocks is reserved. Each JFN block has a size of MLJFN. The JSB page itself has defined uses for each location; see the Monitor Tables for a description of the JSB.

DYNAMICALLY ALLOCATED SPACE

The JSB space is allocated both a page at a time and an n-word block at a time.

JBCOR - is an allocation table with one bit for each page in the JSB space. If the bit is on, the page is free. If the bit is off, the page is in use. Pages 62Ø-7Ø6 are represented here.

JSBFRE - is the head of the blocks of free storage.

Routine ASGPAG is called to allocate a page of storage. Pages of JSB space are allocated for:

1. Magtape buffers.
2. File window pages.
3. To add more space to the free list, JSBFRE.
4. Mapping the super index block of a file.
5. Mapping the EXE file directory page and other such uses when a temporary page is needed.

Routine ASGJFR is called to allocate a free block; the caller passes the desired block size. If there is no block of the desired size, ASGJFR calls ASGPAG to assign another page. Blocks are used to store name strings for the JFN block. Name strings have this format:

```
-1,,n   ;n is size of block including this word
 .
 .      ;n-1 words containing name in ASCIZ.
```

Use of PSB Space

The PSB space is used differently for Model A and Model B machines.

MODEL A VERSUS MODEL B MACHINES

On Model B machines, directories are mapped to section 2; on Model A machines, directories are mapped to page 740000 of section 0 (the only section). On Model B machines, IDXFIL is mapped to section 3; on Model A machines, IDXFIL is mapped to page 720000 of section 0. Pages 720000 through 770000 of sections 0 (and 1) are unused on Model B machines.

PSB SPACE LAYOUT

1. CXBPG- page 707; used to temporarily map index blocks.

2. CPTPG- page 710; used to temporarily map fork page tables.

3. CPYPG - page 711; used to map the source page while a copy-on-write page fault is in progress.

4. FPG0, FPG1, FPG2, FPG3 - pages 712 through 715; called temporary fork pages; currently unused.

5. PSIPG - pages 716 and 717; meant for temporary PSI storage; currently unused.

6. IDXFIL - pages 720 through 737; IDXFIL mapped here on Model A machines; unused on Model B machines.

7. Directory pages - pages 740 through 770; directory mapped here on Model A machines; unused on Model B machines.

8. DDTPG - page 774; MDDT page; used while MDDT is in use.

DRMAP

For Model B machines, MSECTB+2 (the section pointer for section 2) contains an indirect pointer through location DRMAP of the current PSB. DRMAP contains a share pointer for the index block of the currently mapped directory. For example:

MSECTB+2/ 324643,,4Ø1

DRMAP=PSB+643/ 224ØØØ,,OFN of mapped directory

Therefore, when the PSB is context switched, the currently mapped directory is also context switched.

IDXMAP

For Model B machines, MSECTB+3 (the section pointer for section 3) contains an indirect pointer through location IDXMAP of the current PSB. IDXMAP contains a share pointer for the index block of the relevant structure's IDXFIL. For example:

MSECTB+3/ 324643,,4Ø1

IDXMAP=PSB+644/ 224ØØØ,,OFN of IDXFIL

MODULE TEST

When answering the lab exercises, write down the names of the tables where you found the answers. The labs will help you understand the monitor data base structure; so remember, where to look is more important than what you find there.

The exercises marked with a double star (**) are more difficult and are optional. If you have the time and/or motivation, do them.

Storage Management Data Structures

Tools

Use the crash <MONITOR-INTERNALS>SM.CPY for these exercises.

EXERCISES

1. Find an in-core page in the running fork's page table: Is it locked in core? If so, what is the lock count for the page?

2. Is the fork's page table locked in core? **

3. Does the fork have an in-core page with no backup address on the drum?

4. Find an in-core page with its own SPT slot.

5. Is there an in-core page with write in progress? With read in progress?

6. What page is at the head of the RPLQ?

7. How many pages are on the RPLQ?

8. Find a mapped file page in the running fork's page map. What is the owning OFNs backup level of storage? **

9. Using CST2, find a core page that is not currently assigned. **

10. What is the running fork's current age?

11. What is the running fork's current working set size?

12. What is the running fork's reserve working set size?

13. How many pages are on the swapping space free list? **

14. Does the running fork have any mapped file page with an indirect pointer?**

15. How many of the pages of the running fork's JSB space actually exist? **

16. What are the SPT offsets for the running fork's JSB, PSB and user page table?

17. Verify that the storage addresses for the running fork's JSB and PSB match the storage address in slots 400 and 401.

Internal Mapping Using MDDT **

MDDT is a part of the monitor that allows you to look at the running monitor with the standard DDT commands; your process is always the running process when you use MDDT. You can also call monitor routines to map pages; however, extreme caution should be taken when using MDDT. If you change any locations, you can crash the monitor. It is a good practice to type carriage return immediately after you open any location to prevent accidental deposits into memory. This part of the lab uses MDDT to map directories and/or other pages using their internal identities.

Tools

You can enter MDDT in either of two ways. In the first example, the running fork will be the top fork of your job (i.e., the EXEC). In the second example, the running fork will be the fork running user level DDT.

```
@ENABLE
$^EQUIT          ;go to the mini-exec
MX>/             ;enter MDDT
MDDT
```

```
@ENABLE
$SDDT            ;you can use SDDT or UDDT
JSYS 777$X       ;jsys 777 causes you to enter MDDT
MDDT
```

You can use either method to enter MDDT. You can return from MDDT by calling the routine MRETN. You do this by typing:

```
MRETN$G
```

You will also need the writeup in the DEBUGGING section called Mapping Page 677 and from your Student Guide.

EXERCISES

Map the pages in the following exercises to page 677. This is a page out of your job's JSB space and is the traditional page to use.

1. Map some other page table. It can be the page table of one of your friend's forks or any page table you choose.

NOTE

Hint: If you know the fork number of the fork whose page table you want to map, where would you find the page table's internal identity?

2. Map some other JSB. For example, ask your neighbor what his/her fork number is and then find out what the corresponding JSB's internal identity is.

3. Map an OFN.

NOTE

Suggestion: You may wish to map the OFN of the program running in your fork and verify that the in-core-copy of the index block has share pointers that match the share pointers for your fork's mapped pages from the file.

4. Map a page of some other process. **

5. Map a JSB space page from some other JSB. **

6. Un-map the last page.

NOTE

If you do not un-map the page, the job will hang when you try to LOGOUT because the JBCOR table does not have the page allocated.

Page Fault Crash **

The crash for this exercise is BADBTB.CPY. Try and figure out why the system crashed. The questions below should help point you in the right direction. To look at the crash, do the following:

```
@ENABLE
$FILDDT
FILDDT>LOAD <MONITOR-INTERNALS>R3-MONITOR.EXE
FILDDT>GET <MONITOR-INTERNALS>BADBTB.CPY
```

1. Why does a BADBTB crash happen?

2. What push down list is in use (i.e., what was generally going on when the system crashed)?

3. What instruction made the reference that caused the BADBTB?

This page is for notes.

TEST EVALUATION SHEET

Storage Management Data Structures

EXERCISES

1. Find an in-core page in the running fork's page table. Is it locked in core? If so, what is the lock count for the page?

   ANSWER: The running fork's page table is mapped to the monitor's address space beginning at UPTA. Look for a page whose storage address indicates it is in core; that is, bits 12-17 of the storage address are 0. The CST1 entry for that page contains the lock count in bits 0-11.

```
UPTA/ 124000,,742       ;this is an immediate
                        ;pointer, therefore,
                        ;the storage address
                        ;is in bits 12-35.
                        ;Bits 12-17 are 0,
                        ;indicating the page
                        ;is in core.

CST1+742/ 3,,12704      ;page is not locked
                        ;in core lock count
                        ;is zero.
```

2. Is the fork's page table locked in core? **

   ANSWER: The fork's page table is mapped at UPTA. Currently UPTA=775. Therefore, we want to trace down MMAP+775; see if the page is in core, and if so, is it locked.

```
UPTA= 775000

MMAP+775/324754,,401    ;mapped indirect
                        ;through offset 754
                        ;of SPT slot 401
```

```
                                    ;(which is the first
                                    ;page of the PSB).

        PSB+754/ 224000,,1131       ;share pointer so
                                    ;storage address in
                                    ;SPT slot 1131

        SPT+1131/ 100,,250          ;storage address is
                                    ;core page 250

        CST1+250/ 503,,12524        ;lock count = 5
```

3. Does the fork have an in-core page with no backup address on the drum?

   ANSWER: The next level of storage back up address for a core page is in its CST1 entry; if the page has a copy on the drum, the CST1 entry has a drum address. So, you want to look for a CST1 entry with no drum address.

```
        UPTA+4/ 124000,,210

        CST1+210/ 1,,0              ;not a drum address.
                                    ;Therefore, this page
                                    ;has no drum copy.
```

4. Find an in-core page with its own SPT slot.

   ANSWER: If an in-core page has its own SPT slot, the left half of its CST2 entry is zero and the right half is the SPT slot.

```
        CST2+56/ 536                ;has own SPT slot =536
        CST2+57/403,,45             ;does not have own SPT
                                    ;slot
        CST2+60/ 520                ;has SPT slot =520
```

5. Is there an in-core page with write in progress? With read in progress?

   ANSWER: If a page has read or write in progress, its age field in its CST0 entry is equal to PSRIP

or PSWIP, respectively. See the CST0 table for a description of the age field states.

```
CST0+615/ 4010,,0          ;this page has write
                           ;in progress.
```

There is no page with read in progress.

6. What page is at the head of the RPLQ?

   ANSWER: RPLQ is the queue header for the linked list of free pages. The replaceable queue is a linked list through CST3. The page number is its offset from CST3.

```
RPLQ/ 77403,,77611         ;tail,,head of free
                           ;list

CST3=77000

77611-77000=611            ;page 611 is the head
                           ;of the free list.
```

7. How many pages are on the RPLQ?

   ANSWER: Location NRPLQ contains the count of free pages.

```
NRPLQ/ 30                  ;there are 30 free
                           ;pages
```

8. Find a mapped file page in the running fork's page map. What is the owning OFN's backup level of storage?

   ANSWER: A mapped file page that has a share pointer has the owning OFN stored in its SPTH entry.

```
UPTA+5/ 204000,,517

SPTH+517/ 17,,2            ;OFN in left half

SPT+17/ 403,,15770         ;OFNs currently on
```

```
                                        ;drum.

        SPTH+17/ 10,,577554             ;therefore next level
                                        ;of storage is disk.
                                        ;SPTH entry for an
                                        ;OFN always contains
                                        ;the disk address.
```

9. Using CST2, find a core page that is not currently assigned. **

   ANSWER: If a core page is unassigned, its CST2 entry is 0.

```
        CST2+71/ 0                      ;unassigned page
```

10. What is the running fork's current age?

    ANSWER: A fork's current age is stored in it FKNR entry in bits 9-17.

```
        FORKX/ 40

        FKNR+40/ 100127,,166            ;current age is 127
```

11. What is the running fork's current working set size?

    ANSWER: A fork's current working set size is in its FKWSP entry.

```
        FKWSP+40/ 166                   ;current size
```

12. What is the running fork's reserve working set size?

    ANSWER: A fork's reserve working set size is in its FKNR entry, right half.

```
        FKNR+40/ 100127,,166            ;reserve size = 166
```

13. How many pages are on the swapping space free list? **

ANSWER: Location DRMFRE contains the number of free swapping pages. Routine DRMASN assigns swapping space and uses DRMFRE.

```
DRMFRE/ 3325
```

14. Does the running fork have any mapped file page with an indirect pointer?**

ANSWER: If the running fork has any indirect file page pointers, the page map entry has an indirect pointer through an OFN; that is, the SPT slot is less than NOFN. Search the page table (i.e., UPTA) for indirect pointers whose right half is less than NOFN. In this case, there are none.

15. How many of the pages of the running fork's JSB space actually exist? **

ANSWER: The JSB space pages are all mapped indirectly through the JSB map; if a page actually exists, its JSB space map slot contains a pointer and if the page does not exist its JSB map slot contains a zero.

```
JSB<JSB+66>0$N


JSB/ 224000,,1132
JOBMAP+4/  124003,,12574
JOBMAP+6/  124000,,241
```

16. What are the SPT offsets for the running fork's JSB, PSB and user page table?

ANSWER: Table FKJOB indexed by fork number contains the JSB's SPT offset in the right half. Table FKPGS, indexed by fork number, contains the user page table SPT offset in the left half and the first page of the PSB's SPT offset in the right half. Table FKCNO, indexed by fork number,

contains the second page of the PSB's SPT offset in the left half.

```
FORKX/ 40

FKJOB+40/ 13,,1132      ;JSB's SPT offset=1132

FKPGS+40/ 1131,,454     ;user page table SPT
                        ;offset,,PSB first
                        ;SPT offset

FKCNO+40/ 1130,,15      ;PSB second page SPT
                        ;offset=1130
```

17. Verify that the storage addresses for the running fork's JSB and PSB match the storage address in slots 400 and 401.

    ANSWER:

```
SPT+1132/ 100,,200      ;JSB storage address
SPT+400/ 200,,200       ;matches SPT slot 400

SPT+454/ 100,,464       ;PSB storage address
SPT+401/ 200,,464       ;matches SPT slot 401
```

Internal Mapping Using MDDT **

EXERCISES

1. Map some other page table. It can be the page table of one of your friend's forks or any page table you choose.

   ANSWER: Every page table has its own SPT slot; therefore, its internal identity is that SPT slot. If you know the fork number, its SPT slot is stored in the left half of FKPGS, indexed by fork number. Therefore, you should call SETMPG with that SPT slot as the argument in AC1.

2. Map some other JSB. For example, ask your neighbor what his/her fork number is and then find out what the corresponding JSB's internal identity is.

   ANSWER: Every JSB also has its own SPT slot; its SPT slot is stored in FKJOB indexed by fork number (for every fork in the job). This SPT slot is a JSB's internal identity and is the argument you give in AC1 for SETMPG.

3. Map an OFN.

   ANSWER: The OFN (i.e., its SPT slot) is its internal identity; the OFN is the argument in AC1 when you call SETMPG.

4. Map a page of some other process. **

   ANSWER: To map a page of another process, you need the process page table's SPT slot in the left half and the page number in the right half.

5. Map a JSB space page from some other JSB. **

   ANSWER: A JSB page's internal identity is the JSB's SPT slot in the left half and the page's offset in the JSB map in the right half.

6. Un-map the last page.

   ANSWER: To unmap a page, call SETMPG with AC2 set up as usual and a 0 in AC1.

Page Fault Crash **

EXERCISES

1. Why does a BADBTB crash happen?

   ANSWER: A BADBTB happens when a page where the bit table is mapped is referenced when the bit table is not currently mapped.

2. What push down list is in use (i.e., what was generally going on when the system crashed)?

   ANSWER: The push down list that is in use tells you what type of thing the monitor was doing at the time of the crash.

```
p/ -36,,TRAPSK+23        ;a page fault was in
                         ;progress.
```

3. What instruction made the reference that caused the BADBTB?

   ANSWER: We want the instruction that page faulted. The address of the instruction that page faulted and the page fault address are stored in the page fault old PC (UPTPFO) and the page fault word (TRAPSØ), respectively.

```
TRAPSØ/ 1ØØØ,,551ØØØ     ;address that page
                          ;faulted
UPTPFL/ JUMP Ø             ;flags
UPTPFO/ 26255              ;address of
                           ;instruction that
                           ;page faulted
                           ;i.e., the effective
                           ;address of its
                           ;contents is 551ØØØ.


BTB=54ØØØØ                 ;base address where
                           ;bit table mapped

BTBLEN=15ØØØ               ;size in words of
                           ;bit table reserved
                           ;area.
```

   Therefore, we see the reference is definitely in the area. Now, why was the reference made? Was the code actually trying to reference the bit table and did not map it first, or was it a garbaged reference?

```
26255/ LDB T1, 25643     ;this is the
```

```
                                ;instruction that
                                ;caused the fault

        25643/ 3002,,101400     ;this is the byte
                                ;pointer.  Note
                                ;it is indexed by AC2

        SPT= 101400             ;this is the SPT base
                                ;address

        551100-101400= 447500   ;this must have been
                                ;the contents of AC2.
```

Using listings/ microfiche and FILDDT, find where the instruction is.

```
        SECG37+22/ JRST GETTP1
                 / HRRZ T2,T1           ;looks like
                                        ;the start
                                        ;of a literal
                 / MOVEM T2,TRPID
                 / LDB T1,25643         ;here is our
                                        ;instruction


        0,,-1$M             ;set the mask for the right
                            ;half

        SECG37+23$W

        GETTP1+27/ JRST SECG37+23       ;here is
                                        ;where we
                                        ;came from
                                        ;i.e, how we
                                        ;got to the
                                        ;literal
```

So, this page fault came from the routine GETTPD, which is page fault code itself. We have a page fault within a page fault. Look on the stack for evidence of what the old page fault was and what was happening to cause this page fault. Also, remembering what our byte pointer looks like, that is, SPT table base address indexed by an AC, implies that the LDB was trying to pick up some information from an SPT slot; however, 447500 is

not a legal SPT slot. Also, location TRAPC is zero if this is not a recursive page fault and if TRAPC is positive, the number indicates the level of recursion.

```
TRAPC/ 1      ;one level of recursion

P/ -36,,TRAPSK+23

TRAPSK+23/ JUMP 0    ;top of stack now.


TRAPSK/ 10
      / 447500           ;looks like our index
      / 224000,,447500   ;share pointer with
                          ;our index!
      / 2000,,0
      / 221120,,531
      / 300000,,CHKDM0+1 ;looks like a PC
      / 401000,,6070
      / 32,,541636
TRAPSK+10/ MRP4+12
      / 300000,,0
      / 300000,,TRPRST+1 ;looks like a PC
      / 447500             ;our index again
      / 224000,,447500
      / 1000,,707010
      / MOVE T1,PIPGA(T1)
      / 75
TRAPSK+20/ 300000,,CHKDM0+1 ;looks like a PC
      / 1000,,707010
      / 26255
      / 320000,,0
```

Using your crash writeup, look at what gets pushed on the stack when a page fault begins. In order of appearance, we get AC1, AC2, AC3, AC4, AC7, AC16, TRAPSW, the runtime (if not recursive fault), the return PC, and the flags. So, stack locations 0-11 are these locations at the time of the first page fault. Note that the first page fault came from MRP4+12. Then, we have something on the stack that looks like a return address. It is the return from the call to GETTPD in the routine TRPRST. Then, we have the set up from the second page fault. Note that AC2 contained a 447500 in the right half, as we suspected. And note that it looks very much like a share pointer, also as we expected. Location TRAPSW is the previous saved TRAPS0, that is, the page fault word. So the previous page fault was on address 707010. See why it faulted.

```
MMAP+707/ 324666,,401

PSB+666/ 224000,,447500          ;here is the
                                 ;share pointer
                                 ;we saw on
                                 ;the stack.
```

It is not a legal share pointer because 447500 is not a legal offset into the SPT. So, somehow, the PSB page map got a bad share pointer.

The original page fault was from MRP4+12. See what that routine was doing, how it got called and why.

```
MRP4+12/ MOVE T3,CXBPGA(T1) ;here is the
                            ;reference to
                            ;page 707 that
                            ;that originally
                            ;page faulted.
```

When a page fault starts, the old P is saved in location TRAPAP.

```
TRAPAP/ UPDL+26,,UPDL+25
```

So, there was a JSYS going on and the JSYS code called MRP4. See what JSYS it was.

```
MPP/ UPDL+10,,UPDL+7     ;this indicates that
                         ;a JSYS has called a
                         ;JSYS.


UPDL/ 5623
    / CAM 104000
    / 5623
    / CAM 104000
    / -1
    / UPDL+4,,UPDL+3
    / FFFFPL+1
    / MEMPS1+3,,104000
 UPDL+10/ UPDL+5,,UPDL+4
    / 137
    / .TRRET
    / CAI RPACS1+10
    / 1
    / 1
    / IMULI T2,@531
    / 0
UPDL+20/ 137
    / DSKDTB
    / HRR DSKDTB
    / 0
```

If we look at the stack, it seems that an FFFFP JSYS called an RPACS.

```
RPAC1+7/ CALL MRPACS
```

The routine MRPACS calls MRP4, which is where the original page fault came from. Inspection of the code seems to indicate that all traces of who messed up the map entry for page 707 are gone. The code in MRP4 calls SETXB1 to map the index block and it looks as though the routine set up a bad share pointer somehow.

Monitor Tables

DECsystem-2Ø Monitor Tables

Table Of Contents

(ALOC1) Allocation 1 Table. 1

(ALOC2) Allocation 2 Table. 2

(BALSET) Balance Set Table. 3

(BAT) Bad Allocation Table. 4

(BSPT) Balance Set Process Table. 6

(BSQ) Balance Set Quantum Table. 7

(BTB) Bit Table for Disk. 8

(BUG-HLT/CHK/INF-STORAGE-AREA)
BUGHLT, BUGCHK, and BUGINF Storage Area. 1Ø

(CDB) Channel Data Block. 11

(CDR-Storage-Area).
Card Reader (Physical) Storage Area. 15

(CDS) Channel Dispatch Service Routine Table. 18

(CHNTAB) Channel Table. 19

(CSTØ) Core Status Table Ø. 2Ø

(CST1) Core Status Table 1. 22

(CST2) Core Status Table 2. 23

(CST3) Core Status Table 3. 25

(CST5) Core Status Table 5. 27

(DEVCHR) Device Characteristics Table. 28

(DEVCH1) Device Characteristics Table 1. 29

(DEVCH2) Device Characteristics Table 2. .....3Ø

(DEV'DTB) Device Dispatch Table. 31

(DEVDSP) Device Dispatch Table Addresses. 33

(DEVNAM) Device Name Table. 34

(DEVUNT) Device Unit Table. 35

(DIRECTORY) Directory Format. 36

(DRMBBT) Drum Bit Table. 44

(DRMCNT) Drum Count Table. 45

(DSKSIZ) Disk Size Pointer Table. 46

(DSKSZ'n) Disk Size Table. 47

(DSKUTP) Disk Unit Type. 48

(DST) Drum Status Table. 49

(DTE-STORAGE-AREA) DTE Storage Area. 50

(DTEDTV) DTE Protocol Device Dispatch Table. 61

(END/DEQ - STORAGE AREA).
Enqueue/Dequeue Storage Area. 62

(ENQ-LOCK-BLOCK) Enqueue Lock Block 63

(EPT) Executive Process Table. 65

(EXEC-PG-MAP-TBL) Executive Page Map Table. 69

(FDB) File Description Block. 71

(FE-STORAGE-AREA) Front End Storage Area. 74

(FKCNO) Fork Core Number Table. 76

(FKINT) Fork Interrupt Table. 77

(FKINTB) Fork Interrupt Buffer Table. 79

(FKJOB) Fork Job Table. 80

(FKJTQ) Fork JSYS Trap Queue. 81

(FKNR) Fork Number of Reserve Pages Table. 82

(FKPGS) Fork Page and Process Storage Table. 83

(FKPGST)
Fork BALSET Wait Satisfied Test Table. 84

(FKPT) Fork List Pointer Table. 85

(FKQ1) Fork Run Queue Table 1. 86

(FKQ2) Fork Run Queue Table 2. 87

(FKSTAT) Fork Status Table. 88

(FKTIME) Fork Time Table. 89

(FKWSP)
Fork Working Set (In-Memory Size) Table. 90

(HOM) Home Block 91

(HOME) Home Table 93

(HOMTAB)
Logical Unit's Channel and Unit Table. 94

(IDXFIL) Index Table File. 95

(INDEX) Index Block Table. 97

(INIDEV) Initialization Device Routines. 98

(INIDV1)
Front End Initialization Device Routines. 99

(INIDVT) Device Initialization Table. 100

(IORB) I/O Request Block. 103

(IPCF-MESSAGE-HEADER). IPCF Message Header. 107

(IPCF-PID-HEADER). IPCF Process ID Header. 108

(IPCF-STORAGE-AREA) Inter-Process Communication Facility Storage Area. 109

(JOBDIR) Job Directory Table. 110

(JOBNAM) Job Name Table. 111

(JOBPNM) Job Program Name. 112

(JOBPT) Job Process Table. 113

(JOBRT) Job Runtime Table. 114

(JOBRTL) Job Runtime Limit. 115

(JSB) Job Storage Block. 116

(KDB) Kontroller Data Block (TM02 only). 125

(LOGICAL-NAME-DEFINITION). Logical Name Definition Block. 126

(LOGICAL-NAMES-LIST) Logical Names List. 127

(LPT-STORAGE-AREA) Line Printer Storage Area. 128

(MTA-STORAGE-AREA) Magtape Storage Area. 134

(NAMUTP) Name Unit Type Pointers. 138

(NBQ) Negative Balance Set Hold Quantum. 139

(NBW) Balance Set Wait Time 140

(OFNLEN) Open File Length Table. 141

(PHYCHT) PHYSIO Channel Dispatch Tables. 142

(PHYUNT) PHYSIO Unit Dispatch Tables. 143

(PIDCNT) Process ID Count Table. 144

(PIDTBL) Process ID Table. 145

(PSB) Process Storage Block. 146

(PTYSTS) Pseudo Terminal Status Table. 153

(QBLOCK) Queue Block. 154

(RES-FREE-SPACE) Resident Free Space Storage. 156

(SCDRQB) Scheduler Request Table. 158

(SDB) Structure Data Block. 159

(SNAMES) Subsystem Names. 162

(SNBLKS) Sybsystem Blocks. 163

(SPFLTS) Subsystem Page Faults. 164

(SPT) Special Pages Table. 165

(SPTH) Special Pages Table Home Information 168

(SPTO) Special Pages Table 0. 170

(SSIZE) Subsystem Working Set Size. 171

(STIMES) Subsystem Runtimes. 172

(STRTAB) Sturcture Data Block Table. 173

(SWAP-FREE-SPACE)
Swappable Free Space Pool Format. 174

(SYNMTB) System Logical Name Table. 175

(SYS-STARTUP-VECTORS)
System Startup Transfer Vectors. 176

(SYSERR-STORAGE-AREA)
Syserr Storage Area. 177

(TT-LINE-DYN-DATA-BLK)
Teletype Line Dynamic Data Block. 188

(TTACTL) Teletype Active Line Table. 191

(TTBUFS) Teletype Buffers. 192

(TTCSAD)
Terminal Call Special Request Address Table. 193

(TTCSTM)
Terminal Call Special Request Time Table. 194

(TTLINV) Terminal Type Line Vector Table. 195

(TTSPWD) Terminal Speed Word Table. 196

(TTSTAT) Teletype Status Table. 197

(TTXXVT)
Teletype Device Specific Vector Table. 198

(TTY-STORAGE-AREA)
Teletype Storage Area. 200

(UDB) Unit Data Block. 202

(UDIORB) UDSKIO IORB Pool. 205

(UDS) Unit Dispatch Service Routine Table. 206

(UPT) User Process Table. 207

(USER-PG-MAP-TBL)
User Page Map Table. 210

Name: ALOC1

Description: Allocation 1 Table. This table of length NOFN (size of OFN area in SPT) is used to help enforce disk quotas for each active directory.

Defined in: STG

Reference by: PAGEM

Format

ALOC1

ADIRN Directory No.	ODIRC OFN Directory Count
. . . . .	

Note: Each SPT entry in the OFN area contains an index into this table.

Name: ALOC2

Description: Allocation 2 Table. This table of length NOFN (size of OFN area is SPT) is used in disk quota enforcement for each active directory.

Defined in: STG

Referenced by: DISC, PAGEM

Format

ALOC2

PGLFT Count of Pages Left for This Directory (may be negative)
. . . . .

Note: Each SPT entry in the OFN area contains an index into this table.

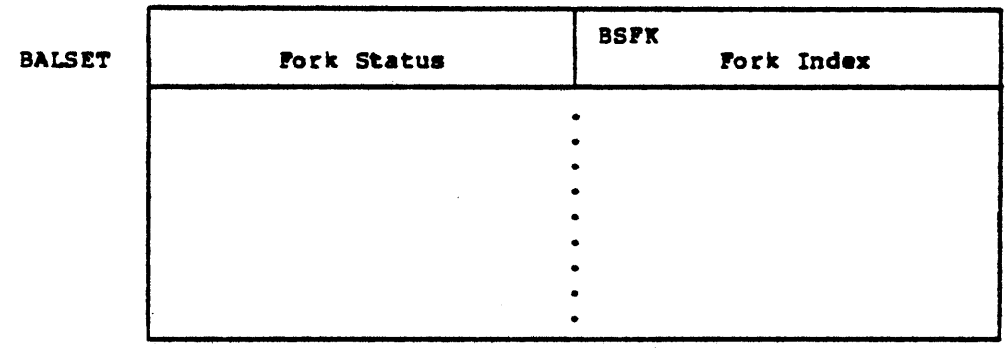
Name: BALSET

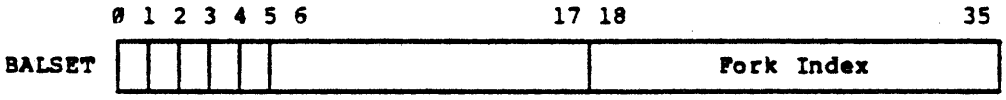
Description: Balance Set Table. This table contains the set of most eligible forks for CPU service whose combined working set sizes are balanced with the amount of physical core available. Only forks in this table can be chosen to run. Position in this table is arbitrary and has no effect on run priority (Position on GOLST determines this).

Defined in: SCHED

Referenced by: PAGEM

Format





Symbol	Bits	Content
BSWTB	0	If 1, fork waiting for I/O (disk or drum)
BSNSK	1	If 1, fork is NOSKED (no scheduling of other forks allowed) or NOSWAP
BSNUL	2	If 1, free BALSET slot (Deleted entry)
BSHLD	4	If 1, fork being held in Balance Set

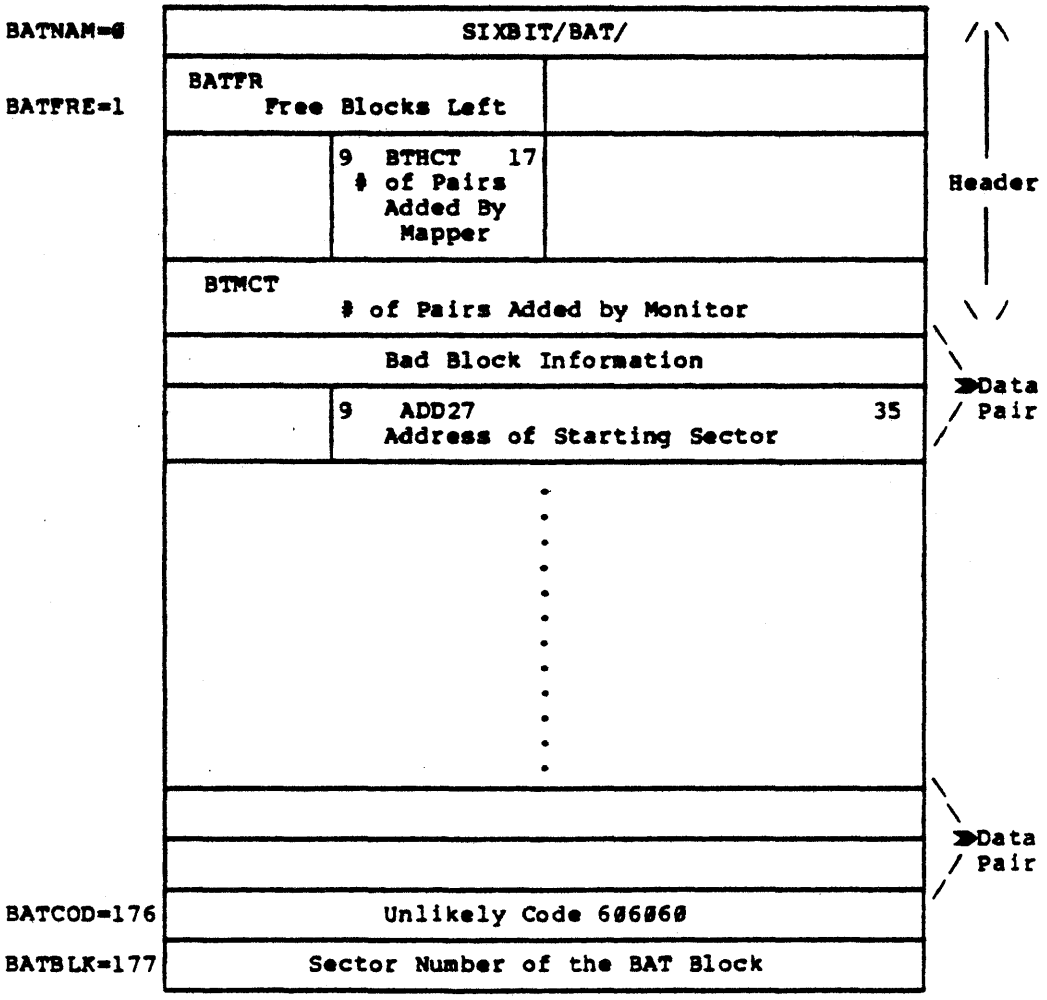
Name: BAT

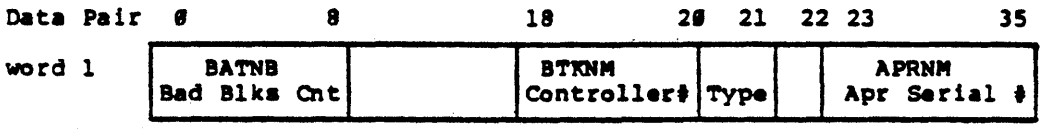
Description: Bad Allocation Table. The BAT Block is one sector in length (128 words). It consists of 4 words of header, followed by data; each 2 word data entry indicates the bad spots on the disk.

Defined in: PROLOG

Referenced by: DSKALC, DISC, DEVICE, PHYH2

Format





Bits	Pointer	Content
0-8	BATNB	Count of Bad Blks in Pair
18-20	BTKNB	Massbus Controller #
21	BADT	Type field in BAT Pair
23-35	APRNM	APR Serial #

word 2

Bits		
18-35	ADD18	Old style disk address of starting sector
9-35	ADD27	New style address of starting sector

Name: BSPT

Description: Balance Set Process Table. This table is a doubly linked list of all forks in the balance set and is parallel to the BALSET table. The pointers to the beginning and end of the list are kept in the resident storage address, BALLST. The Scheduler scans this ordered list from the top for the next eligible fork to run. When a fork has exhausted it's balance set quantum (See BSQ table), it will be requeued to the bottom of the BSPT list, allowing the Scheduler to give round-robin CPU service to the balance set processes.

Compute-bound processes (Q3 forks) are placed at the end of the BSPT list when they enter the balance set. Interactive processes (Q0, Q1, Q2 forks) are initially favored in their placement on the BSPT list as they are placed above Q3 forks and above any forks that have run more than their balance set quantum. (i.e., have been requeued once).

Once interactive processes have exhausted their balance set quantums, however, they are requeued to the bottom and compete equally with other forks for CPU service.

Defined in: STG

Referenced by: SCHED

FORMAT

BSPT

Backward List Ptr.	Forward List Ptr.
.	.
.	.
.	.
.	.
.	.

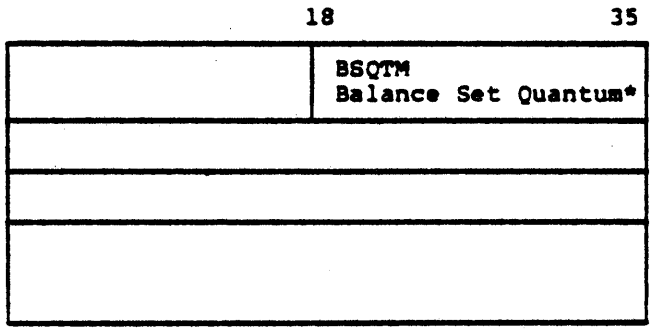
Note: The end of the list will be marked by the address, BALLST.

Name: BSQ

Description: Balance Set Quantum Table. This table is parallel to the BSPT table and holds the balance set quantum for a fork in the Balance Set. When the balance set quantum has expired for a fork, the fork is requeued to a lower run priority in the BSPT table and its quantum reinitialized.

Defined in: STG

Referenced by: SCHED



* The initialization value is currently 200 ms.

Name: BTB*

Description: Bit Table for Disk. This table has mapped into it pointers to the file STRNAM:<ROOT-DIRECTORY> DSKBTTBL, when pages are allocated or deallocated from the disk unit(s) belonging to structure, STRNAM. The bit table file as shown below indicates which pages are assigned (bits off) and which are available (bits on).

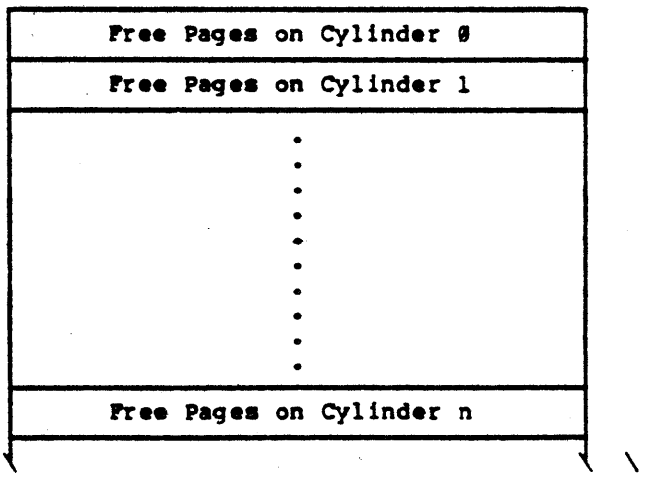
It consists of two parts; the top half contains the number of free pages for each cylinder in the structure and the bottom half contains a bit map (1 bit per page) for all pages of each cylinder in the structure.

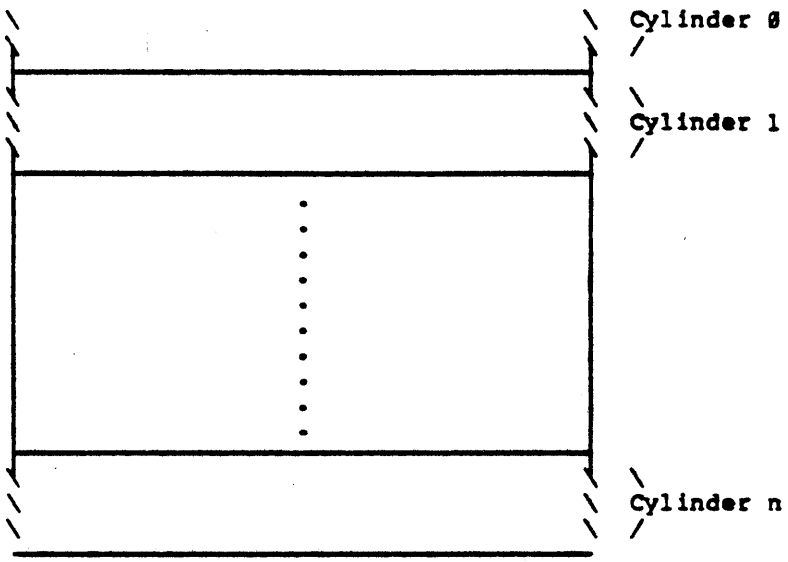
At initialization time, all of this structure's pages belonging to the Home blocks, swapping space and pointed to by the BAT blocks are assigned in the Bit Table file.

Defined in: STG

Referenced by: DSKALC

Format





Note:
In the bit map each cylinder starts on a word boundary and contains as many full words as are needed for all of its pages.

*For Systems which have sectioning, the BTB table does not hold the maps for the disk bit table file. Rather, the monitor will map the disk bit table file for a structure into section four of the monitor's address space when it needs to allocate or deallocate disk pages. That is, the index block of this file will be the page table pointed to by the monitor's section pointer for section four.

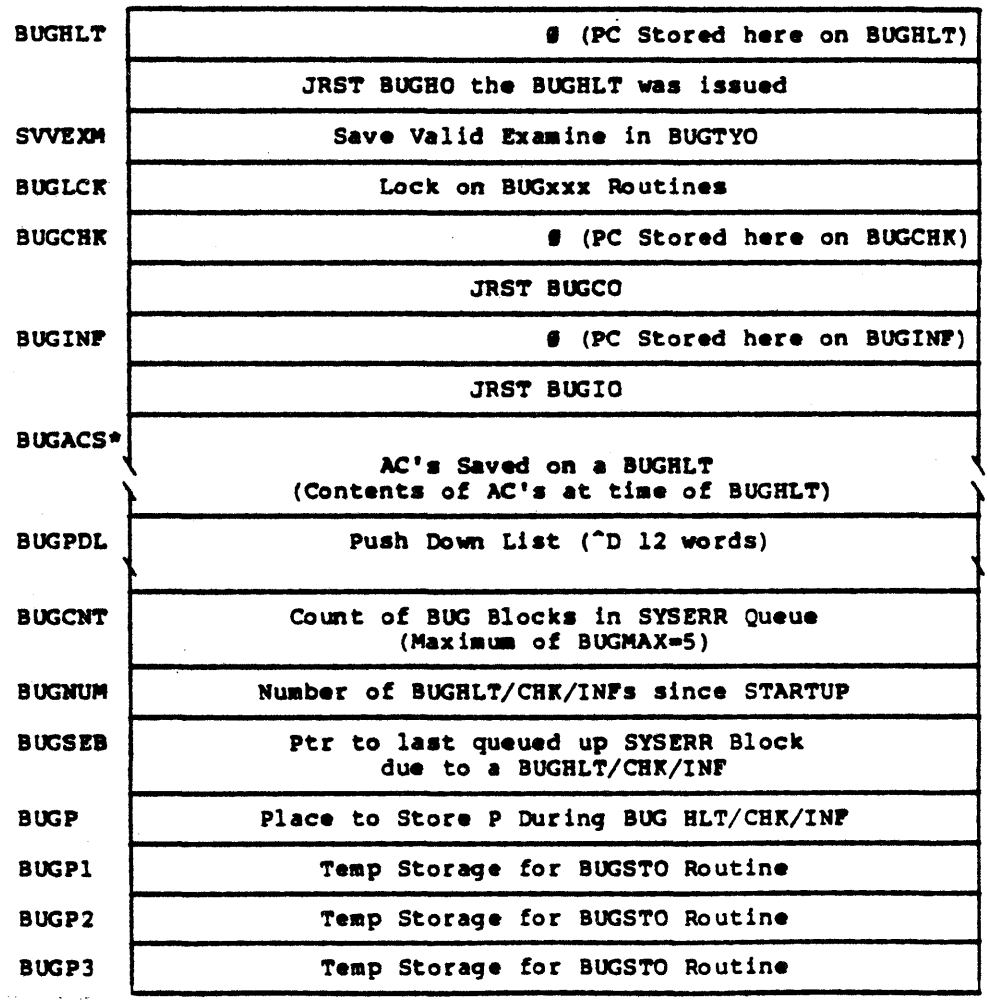
Name: BUG-HLT/CHK/INF-STORAGE-AREA

Description: BUGHLT, BUGCHK, and BUGINF Storage Area. This resident storage is used to hold such information as the push down list, PC, ACs and dispatch address when a BUGHLT/BUGCHK/BUGINF occurs. BUGSEB holds the pointer to the last queued up SYSERR block. (See SYSERR-STORAGE-AREA)

Defined in: STG

Referenced by: APRSRV, DEVICE, DIAG, DIRECT, DISC, DSKALC, DTESRV, EWQ, FESRV, FILINI, FILMSC, FORK, FREE, FUTILI, GTJFN, IMPPHY, IMPDV, IO, IPCF, JSYSA, JSYSF, LINPR, LOGNAM, LOOKUP, MAGTAP, MEXEC, MSTR, NETWRK, NSPSRV, PAGEM, PHYH11, PHYH2, PHYM2, PHYP4, PHYSIO, SCHED, STG, SWPALC, SYSERR, TAPE, TTYSRV

FORMAT



Name: CDB

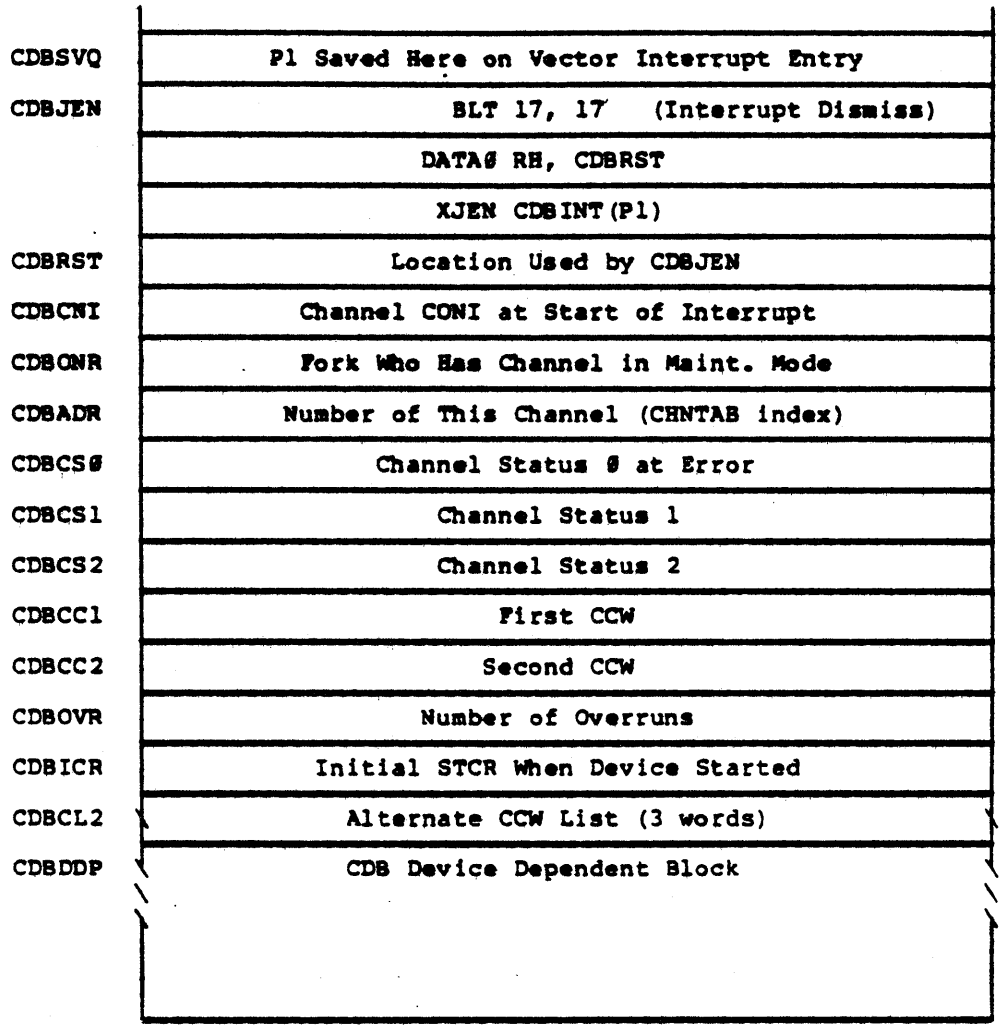
Description: Channel Data Block. This table, one per channel, contains channel dependent instructions and data, pointers to the units (i.e. UDBs) belonging to the channel and information about the currently active unit. When the channel interrupts, control passes (via a JSP instruction) to CDBINT. The CDB address is stored in AC, P1, and the principal analysis routine, PHYINT, is called.

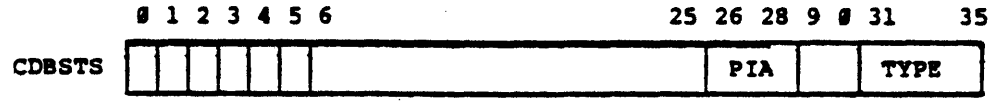
Defined in: PHYPAR

Referenced by: PHYSIO, PHYH2, PHYM2, PHYP4

Format

CDBINT	0(2-word PC stored on interrupt)	
-5	0	
-4	(Flags) 0	
-3	0,, .+1	
-2	MOVEM P1, .+2+CDBSVQ	
-1	JSP P1, PHYINT	
CDBSTS=0	Status and Configuration Information	
CDBMBW=1	Memory Bandwidth Scheduling Information	
CDBODT	Overdue Timer when Data Transfer Active	
CDBICP	EXEC Virtual Adrs (EPT Adrs) of Data Logout Area,, Interrupt Vector	
CDBIUN	Initial AOBJN Pointer to UDB Table	
CDBCUN	Current AOBJN Pointer to UDB Table	
CDBDSP	Unit Utilities Dispatch	Main Entry Dispatch (Channel Dispatch Table)
CDBFCT	Fairness Count for Latency	
CDBPAR	Channel Memory Parity Errors	
CDBNXM	Channel NXMs	
CDBXFR	Currently Transferring UDB	
CDBCCL	Channel Command List (3 words)	
CDBUDB	UDB Table (8 words)	


CDBSVQ
P1 Saved Here on Vector Interrupt Entry
CDBJEN
BLT 17, 17 (Interrupt Dismiss)
DATAØ RH, CDBRST
XJEN CDBINT(P1)
CDBRST
Location Used by CDBJEN
CDBCNI
Channel CONI at Start of Interrupt
CDBCNR
Fork Who Has Channel in Maint. Mode
CDBADR
Number of This Channel (CHNTAB index)
CDBCSØ
Channel Status Ø at Error
CDBCS1
Channel Status 1
CDBCS2
Channel Status 2
CDBCC1
First CCW
CDBCC2
Second CCW
CDBOVR
Number of Overruns
CDBICR
Initial STCR When Device Started
CDBCL2
Alternate CCW List (3 words)
CDBDDP
CDB Device Dependent Block



Symbol	Bits	Content
CS.OFL	0	Offline
CS.AC1	1	Primary command active
CS.AC2	2	Secondary command active
CS.MAI	3	Channel is in maint. mode
CS.MRQ	4	Maint. mode requested for a unit
CS.ERC	5	Error recovery in progress
CS.STK	6	Channel Support Command Stacking
CS.ACL	7	Alternate CCW List is Current
	26-28	PIA field
	31-35	Channel type field

CDBDSP
See Tables, UDS and CDS

CDBDDP
CDB Device Dependent Block for the RH20 Controller

CDBDDP=RH2CNI	CONI RH2, T1
	CONO RH2, T2
	DATAI RH2, T1
	DATAO RH2, T2

CDB Device Dependent Block for the RH11 Controller

CDBDDP=RC1CS1	Control Status 1
RC1DS	Drive Status Register
RC1ER1	Error Register 1
RC1MR	Maintenance Register
RC1AS	Attention Summary Register
RC1DA	Desired Address Register
RC1DT	Driver Type Register
RC1LA	Look Ahead Register
RC1SN	Serial Number Register
RC1OF	Offset Register
RC1DC	Desired Cylinder Register
RC1CA	Current Cylinder Register
RC1ER2	Error Register 2
RC1ER3	Error Register 3
RC1EC1	ECC Register 1
RC1EC2	ECC Register 2
RC1WC	Word Count Register
RC1BA	Current Address Register
RC1CS2	Control and Status Register 2
UBADSW	Unibus Status Register Address
UBBASA	Unibus Bus Addr. Base Addr.

Name: CDR-STORAGE-AREA

Description: Storage Area for Card Readers (physical). Each entry (except for CDRLCK and CDRCNT) is CDRN words long where CDRN equals the number of card readers on the system.

Defined in: STG

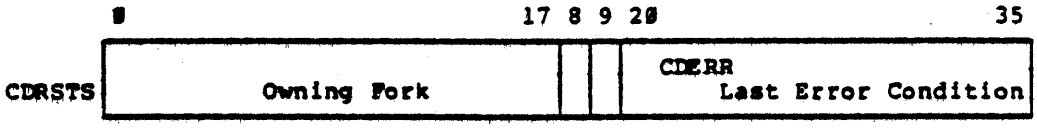
Referenced by: CDRSRV

Format

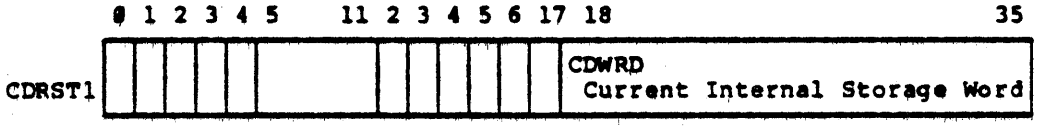
CDRCT1	Buffer Count
CDRCKT	Word for Scheduler Test
CDRSTS	Status Word
CDRST1	Second Status Word
CDRST2	Third Status Word
CARDCT	Count of Cards Read
CARDER	Number of "Hardware" Errors
CDRLCK	CDR Lock Word
CDRCNT	Count of CDRs Opened

The Non-resident area contains:

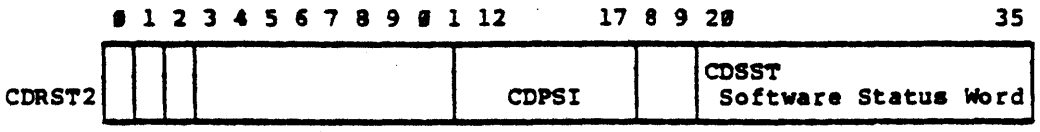
CDRBUF	Card Reader Buffer (1 page)



Bits	Pointer	Content
0-17		Owning fork
18	CDOL	If one, cards in reader
19	CDBLK	Waiting for a card
20-35	CDERR	Last error condition



Bits	Pointer	Content
0	CDAII	CDR opened in ASCII
1	CDATN	CDR needs attention
2	CDMSG	Suppress system messages
3	CDOPN	CDR is open
4	CDER	Error in this CDR
12	CDCNT	Count of bytes now in buffer
13	CDEOF	EOF button was pushed
14	CDBUF	Buffer for process level
15	CDPIR	Process needs interrupt
16	CDBFI	Buffer for PI level
17	CDDON	If one, doing a buffer by process



Bits	Pointer	Content
0	CDSHA	"Status has arrived" flag
1	CDMWS	MTOPR is waiting for status to arrive
2	CDRLD	Front end has reloaded
12-17	CDPSI	PSI chan. no. for on-line transitions
20-35	CDSST	Software status word

Symbol	Bits	Content
.DVFFE	28	Device has a fatal, unrecoverable error
.DVFLG	29	Error logging info. follows
.DVFEF	30	EOF
.DVFIP	31	I/O in progress
.DVFSE	32	Software cond.
.DVFHE	33	Hardware error
.DVFOL	34	Offline
.DVFNX	35	Nonexistent device

Name: CDS

Description: Channel Dispatch Service Routine Table. This table contains vectored addresses to channel dependent functions, and is given in its generalized form. The specific channel dispatch table for the RH2Ø begins at RH2DSP in PHYH2. See PHYPAR for definitions of arguments given and returned on calls to these channel routines.

Defined in: PHYPAR

Referenced by: PHYH2, PHYM2, PHYP4, PHYSIO, STG

Format

CDSINI=Ø	Initialize and Build Data Structure
CDSSTK=1	Stack Second Channel Command, Skip if OK
CDSSIO=2	Start I/O on IORB (skip if started O.K.)
CDSPOS=3	Do Positioning to Idle Unit (skips if O.K.)
CDSLTM=4	Return Latency and Best Request (i.e. best IORB)
CDSINT=5	Interrupt Entry
CDSCCW=6	Generate Single CCW Entry
CDSHNG=7	Hung Reset
CDSRST=1Ø	On Restart, Reset Channel and All Devices
CDSCHK=11	Periodic Check Entry, PIA, etc.

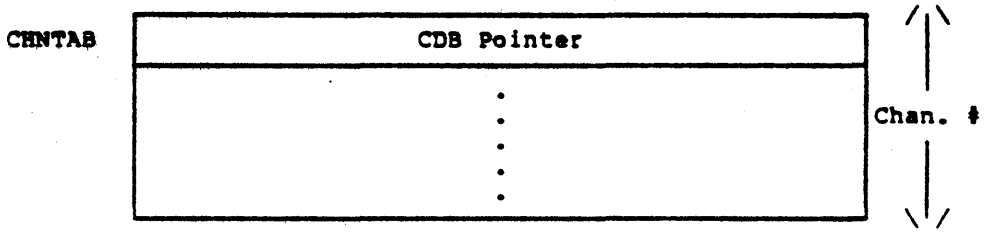
Name: CHNTAB

Description: Channel Table, indexed by channel number, contains channel data block (CDB) pointers.

Defined in: STG

Referenced by: DSKALC, PHYH2, PHYSIO

Format



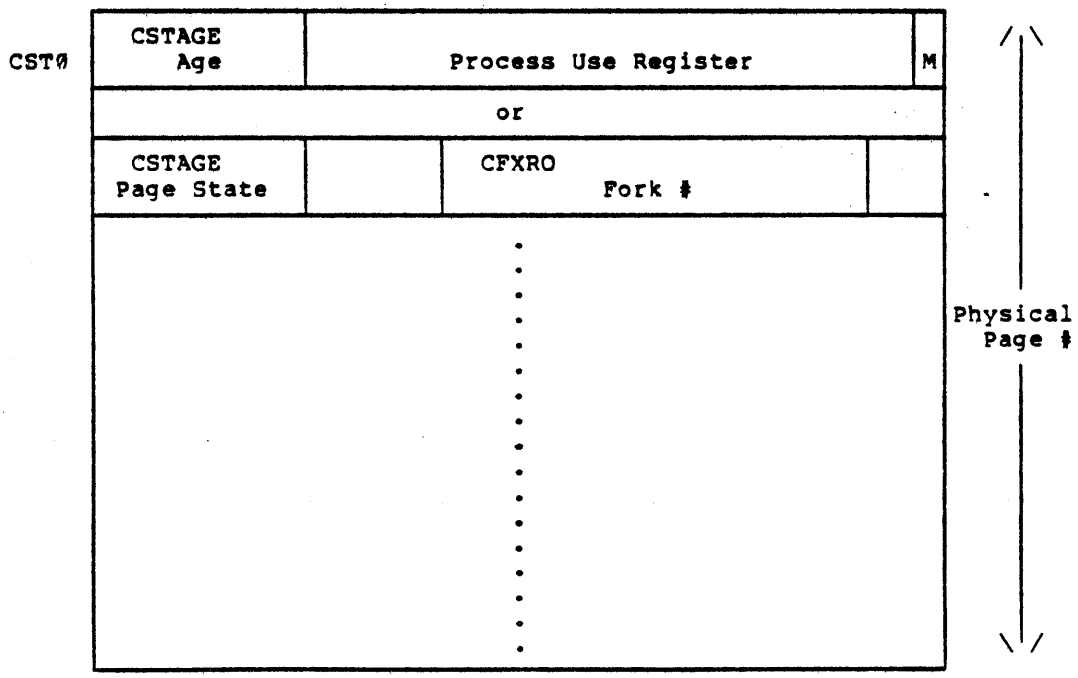
Name: CST∅

Description: Core Status Table ∅ (sometimes referred to as CST). Each entry in this table, indexed by physical page #, is principally defined by the pager. If the page is in use, the entry contains the age stamp for the page, which processes have referenced it and whether the page has been modified. The age stamp field is used to show the page's state if it is not assigned to a process.

Defined in: STG

Referenced by: APRSRV, DSKALC, PAGEM

Format



Symbol	Bits	Pointer	Content
	0-8	CSTAGE	If page in use, contents of pager age register (>= 100) at last age register reload
PUFLD	9-32		Process use register if age field indicates page is in use (i.e., age >=100). Bit n is 1 if process with core number n has referenced it. (Core number is stacked in the FKCNO table)
CORMB	35		This is the "modified" bit which is set by the pager on any write reference. This bit will be 1 if the page has been written since the last operation.
AGEMSK	0-8	CSTAGE	If page not in use, this field indicates (right-justified) the page state as follows: PSRPQ = 0 On replaceable queue PSDEL = 1 To be put on replaceable queue PSRDN = 2 Read completed PSWIP = 4 Write in progress PSRIP = 6 Read in progress PSSPQ = 7 Page on special memory queue PSASN = 10 Page assign to process if age field >= PSASN. (The age field should always be strictly greater than 10 as it is initialized to 100 and increases in value as process runs.)
PSTFLD	15-32	CFXRD	Number of fork which initiated read if page not in use (i.e. age field < 10).
	33-34	CSTPST	Special page state PSTAVL=.MCPSA=0 Available for RPLQ when freed PSTSPM=.MCPSS=1 Place on SPMQ when freed PSTOFL=.MCPSO=2 Offline-action as PSTSPM PSTERR=.MCPSE=3 Offline due to error action as PSTSPM

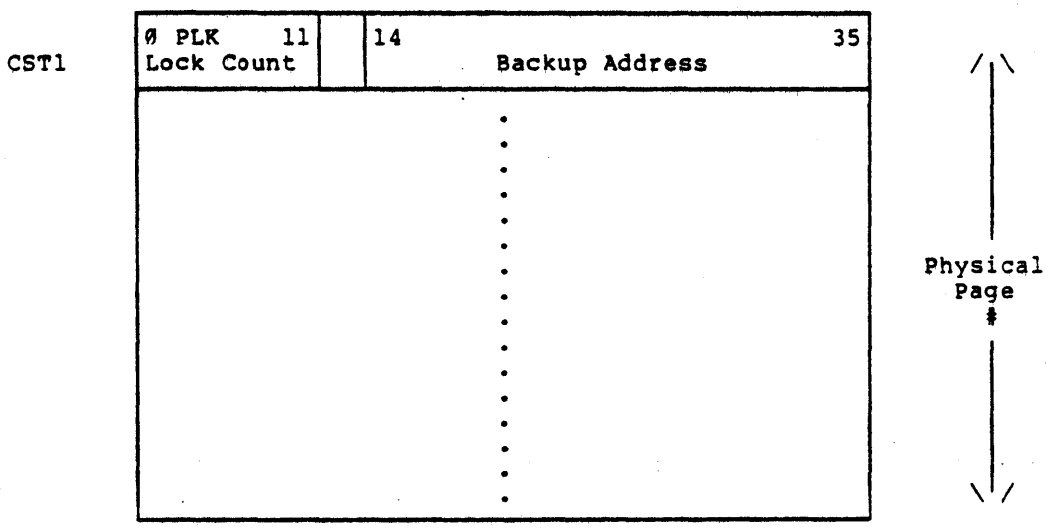
Name: CST1

Description: Core Status Table 1. This table, indexed by physical core page number, is referenced only by the software and is parallel to CSTØ. It contains the lock count which indicates the number of system events requiring the page be locked in core (i.e., page table contains other core addresses) and the backup address (next level of storage) for each page in core (1ØØØØØ if unassigned).

Defined in: STG

Referenced by: PAGEM, PHYSIO, SCHED

Format



Note: If the lock count is non-zero, the page will not be considered for swapping.

Name: CST2

Description: Core Status Table 2 (Home Map Location). This table, indexed by physical page number, is referenced only by the software and is parallel to CST0. It contains the home map location for the page (i.e., the page table which contains the core address pointing to the page).

If the left half is 0, the home map is the SPT and the right half contains the SPT index. If the left half is not 0, the home map is a page table or index block, where PTN is the SPT index of that map and PN is the page number within that map.

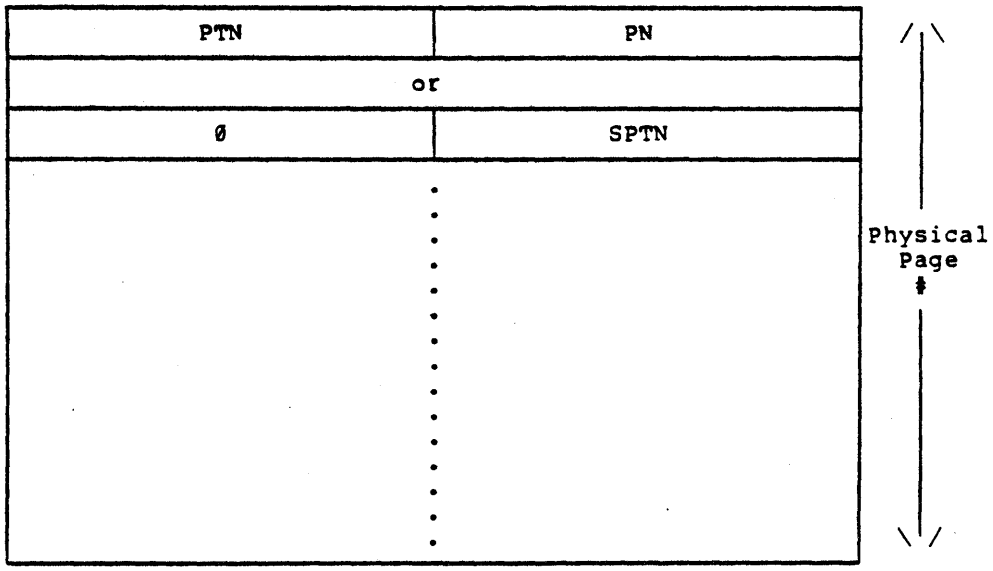
(See the SPT and SPTH table descriptions.)

Defined in: STG

Referenced in: PAGEM

Format

CST2



Note: The SPTN/PTN value (both SPT indexes) is used to specify the kind of page represented in the CST2 table. For example, if the SPTN in the second format above is greater than or equal to NOFN (length of the OFN area), the process' page is a file page pointed to by a shared pointer or fork overhead page. Otherwise (i.e., SPTN<NOFN), it's an index block page.*

Likewise, if the PTN value in the first format above is greater than or equal to NOFN, the page is a private process page (i.e., pointed to by a direct pointer from the process' map). Otherwise (i.e., PTN<NOFN), it is a process' file page pointed to by an indirect pointer through the file's own page table, the index block.*

* In both of these cases when an index block is involved (i.e., SPTN/PTN< NOFN), it is common to find in the monitor listings the symbolic notation, OFN, replacing SPTN/PTN.

Name: CST3

Description: Core Status Table 3. This table, indexed by physical core page number, is referenced only by the software and is parallel to CST0. An entry in this table is used for a variety of purposes, generally as a list pointer for groups of pages on various queues.

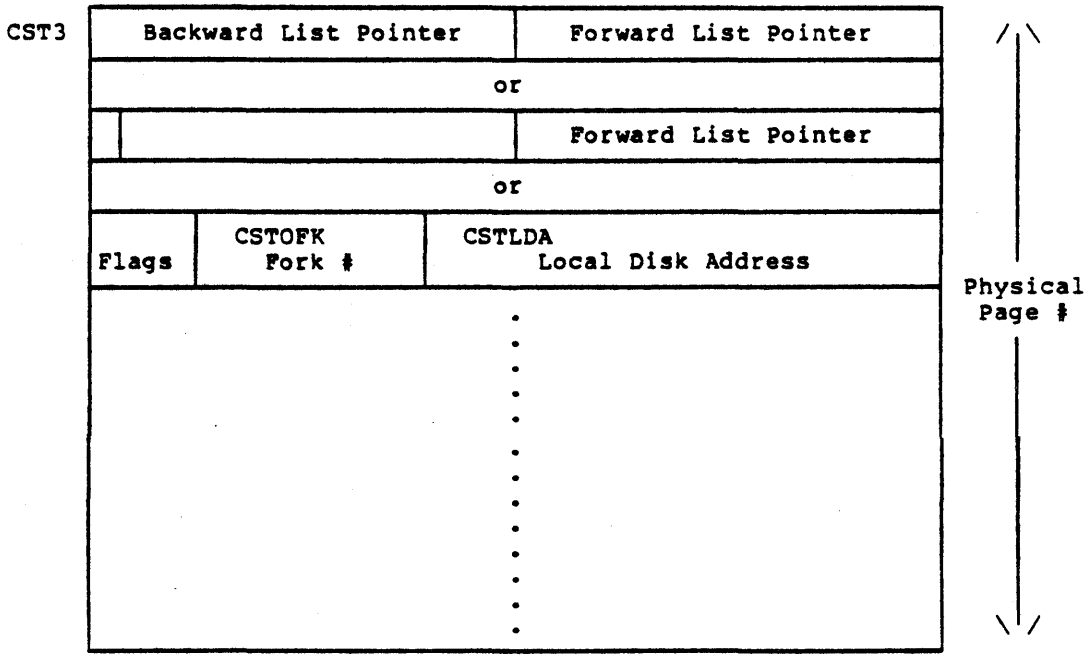
For example, when on the replaceable queue, the left half and right half contain backward and forward list pointers, respectively. When on a swapping device queue, the right half contains a forward list pointer and B0 is 1 if write and 0 if read. Other queues threaded through this table are the deletion and special memory queues.

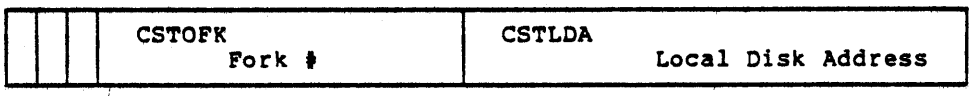
When the page is in use (not linked on one of the queues), it contains the local disk address for PHYSIO and the fork # assigned to the page.

Defined in: STG

Referenced by: PAGEM, PHYSIO, SCHED

Format





Symbol	Bits	Pointer	Content
DWRBIT	Ø		Set if write in progress. The bit is cleared by the swapper when the write completes.
SWPERR	1		Set if an unrecoverable error occurred when this page read in from disk/drum
DSKSWB	2		Swap to disk requested by DDMP (periodic routine that trickles file pages to the disk) or by monitor when certain monitor calls are issued, e.g., CLOSF
	3-14	CSTOFK	Process to which this page is assigned (7777 is not assigned).
	15-35	CSTLDA	Local disk address for PHYSIO

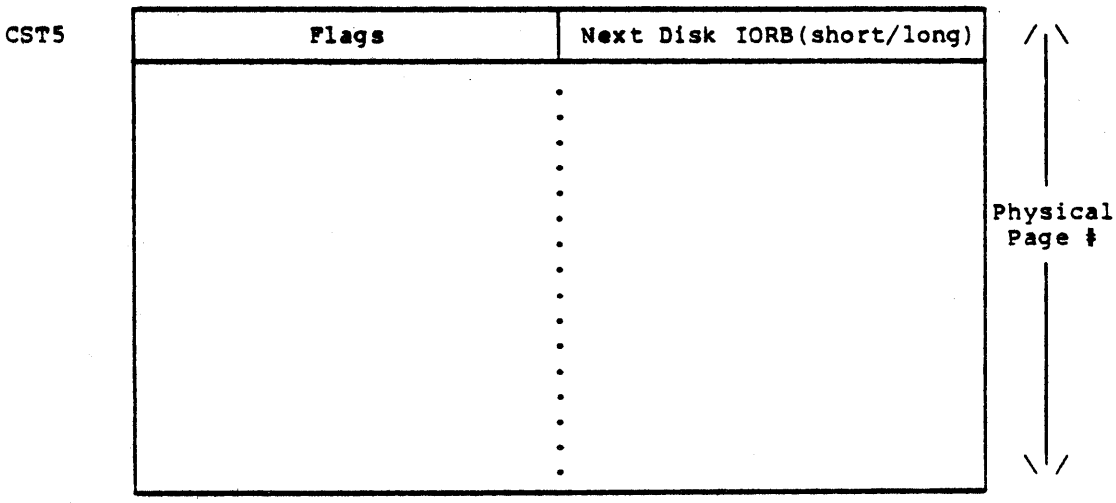
Name: CST5

Description: Core Status Table 5. This table, indexed by physical core page number, is referenced only by the software and is parallel to CST0. It is a table of short IORBs. (See the IORB table description for format of the IRBSTS entry.)

Defined in: STG

Referenced by: PHYSIO

Format



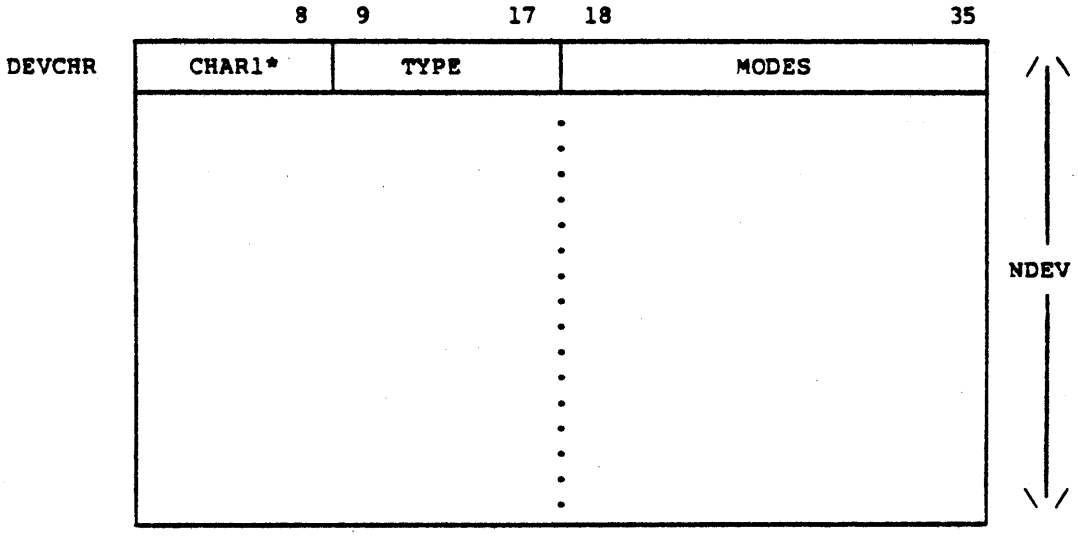
Name: DEVCHR

Description: Device Characteristics Table. This table contains indexed information about each device unit and is initialized from the INIDVT table at system startup time. See INIDVT table for bit definitions.

Defined in: STG

Referenced by: DEVICE, DISC, GTJFN, JSYSA, JSYSF, MEXEC

Format



*CHAR1
Bit 5 can take on another meaning (i.e., DVOPN=1B5; File Open on Device) than the one described in INIDVT.

Note: A resident word, DEVLCK (Free is -1), is used to lock the data base in the DEVXXX Tables when they are being manipulated.

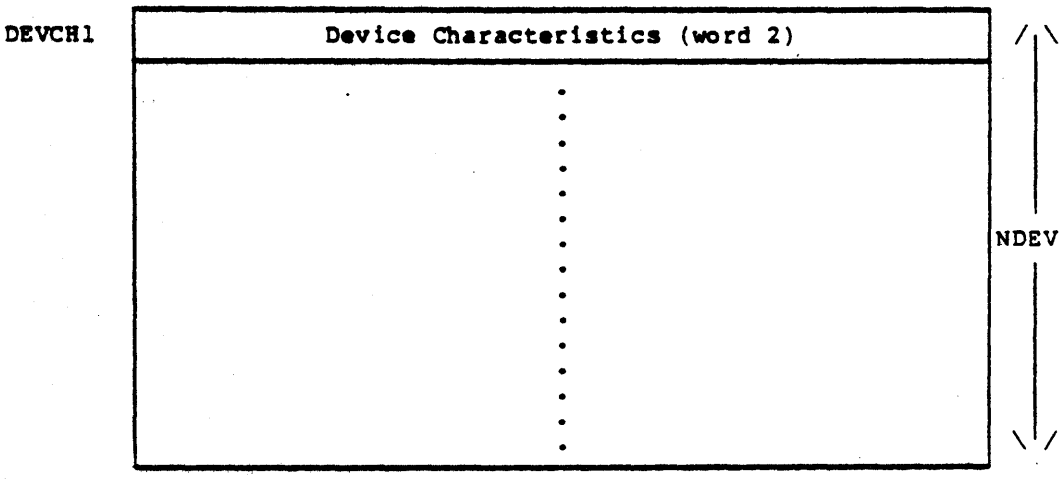
Name: DEVCH1

Description: Device Characteristics Table 1. This table, contains another word of information about each device unit in the system and is initialized from the INIDVT table at system start up time.

Defined in: STG

Referenced by: DEVICE, DISC, GTJFN, JSYSA, JSYSF

Format



Symbol	Bit	Content
D1%SPL	0	Device is spooled
D1%ALC	1	Device is under control of allocator
D1%VVL	2	Volume valid
D1%NIU	3	Device slot not is use
D1%INI	4	Device is being initialized currently for structures only)

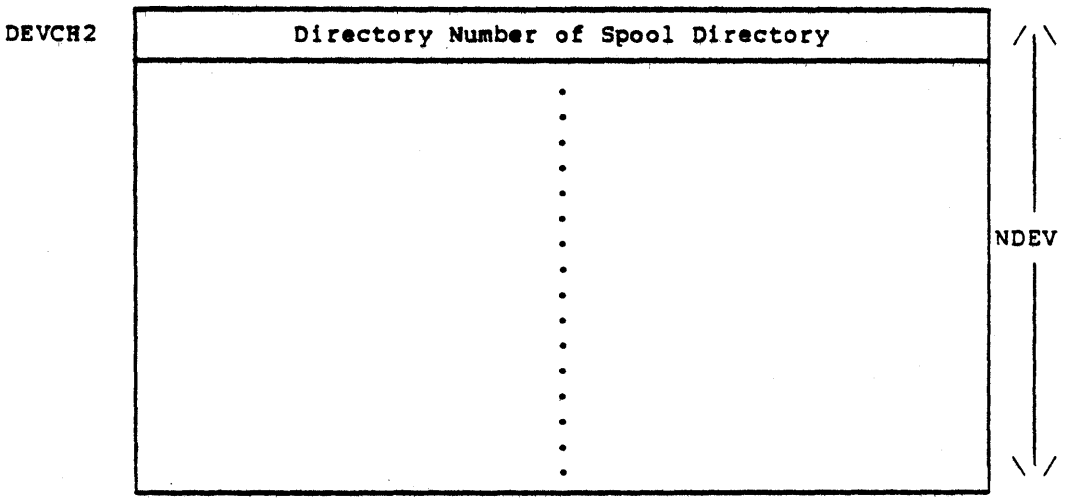
Name: DEVCH2

Description: Device Characteristcs Table 2. This table contains spool directory information and is parallel to the DEVCH1 table.

Defined in: STG

Referenced by: DISC, JSYSA

Format



Name: DEV`DTB

Description: Dispatch Table. Each device has its own dispatch table that conforms to the format described below. An error return dispatch address is placed in those words which have no corresponding device function. The naming convention for these tables is the device name concatenated with DTB (i.e. MTADTB, DSKDTB, TTYDTB, etc.)

Defined in: PROLOG

Referenced by:

Format

DLUKD=0	Directory Setup
NLUKD=1	Name Lookup
ELUKD=2	Extension Lookup
VLUKD=3	Version Lookup
PLUKD=4	Protection Insertion
ALUKD=5	Account Insertion
SLUKD=6	Status Modification
OPEND=7	Open File
BIND=10	Sequential Byte Input
BOUTD=11	Sequential Byte Output
CLOSD=12	Close File
REND=13	Rename File
DELD=14	Delete File
DMPID=15	Dump Mode Input
DMPOD=16	Dump Mode Output

MNTD=17	Mount
DSMD=20	Dismount
INDD=21	Initialize a Directory
MTPD=22	MTAPE Operations
GDSTD=23	Get Device Status
SDSTD=24	Set Device Status
RECOUT=25	Force Record Out (SOUTR)
RFTADD=26	Read File Time and Date
SFTADD=27	Set File Time and Date
JFNID=30	Set JFN for Input
JFNOD=31	Set JFN for Output
ATRD=32	Check Attribute

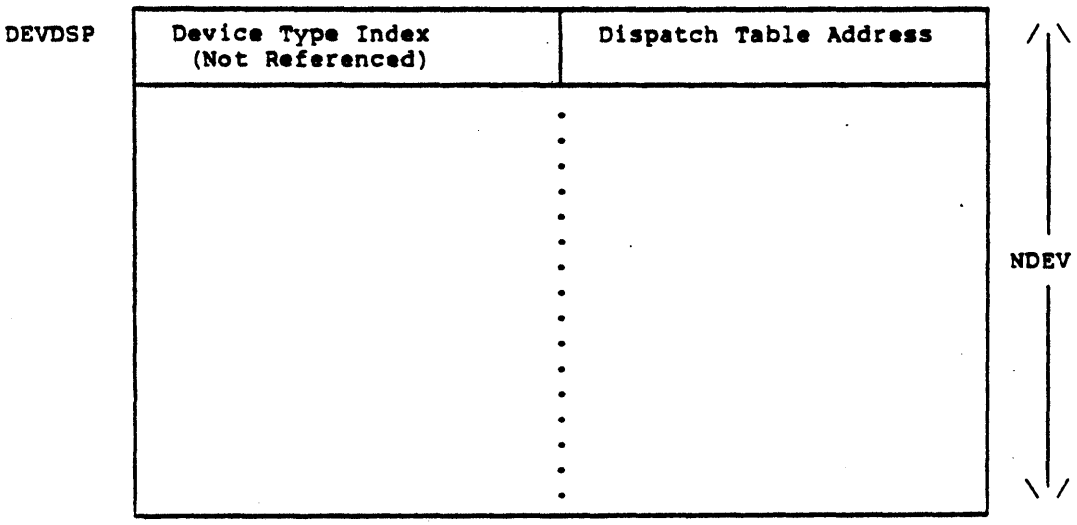
Name: DEVDSP

Description: Device Dispatch Table. This table contains the device dispatch table for each device unit and is initialized from the INIDVT table at system startup time.

Defined in: STG

Referenced by: DEVICE, JSYSF

Format



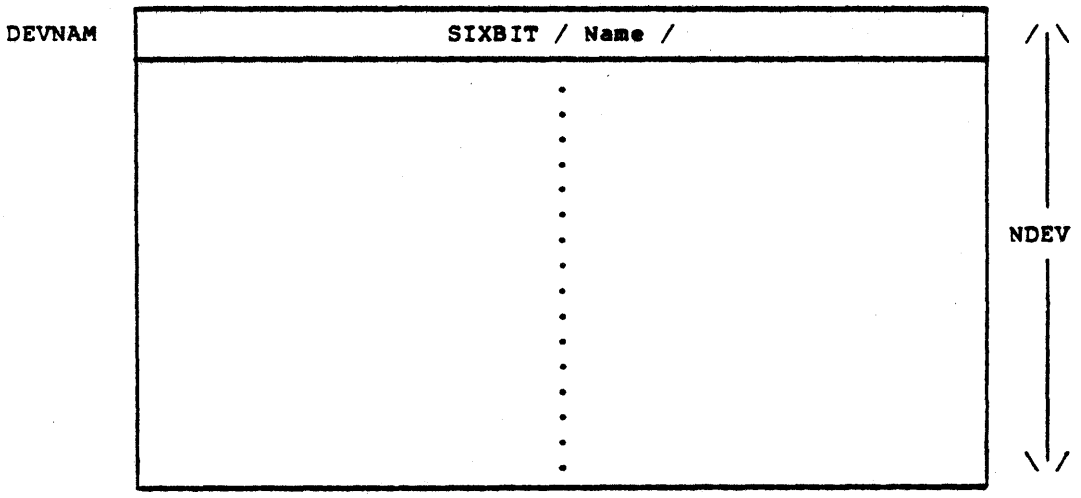
Name: DEVNAM

Description: Device Name Table. This table contains the SIXBIT device name for each device unit. The generic device name is obtained from INIDVT, modified to include unit number (if device has units) and stored in this table at system start up time.

Defined in: STG

Referenced by: DEVICE, JSYSA, MEXEC

Format



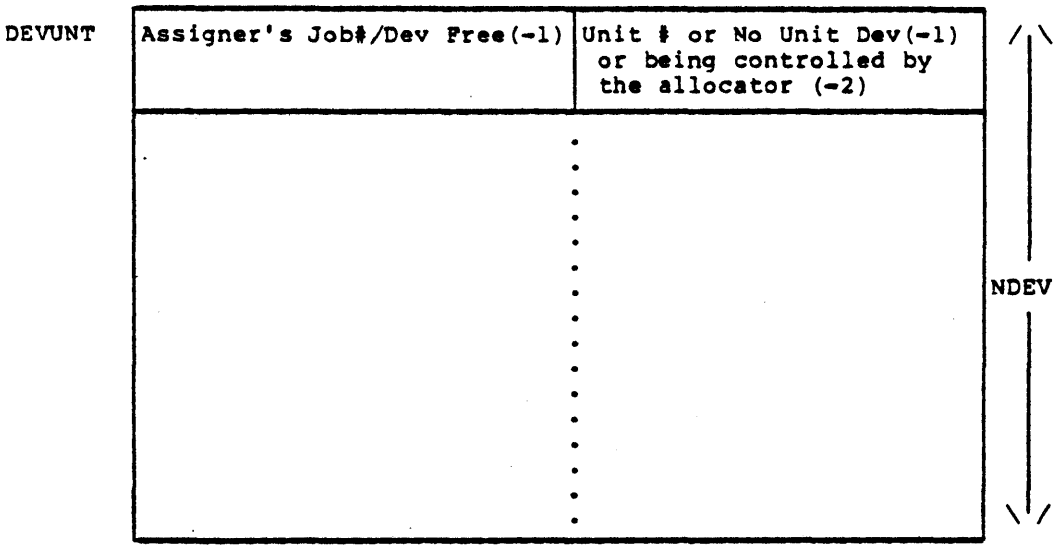
Name: DEVUNT

Description: Device Unit Table. This table, contains infomation about the job associated with a unit, where the unit information in DEVUNT is built at system startup utilizing data from the INIDVT table.

Defined in: STG

Referenced by: TTYSRV, DEVICE, JSYSA, JSYSF, MEXEC

Format



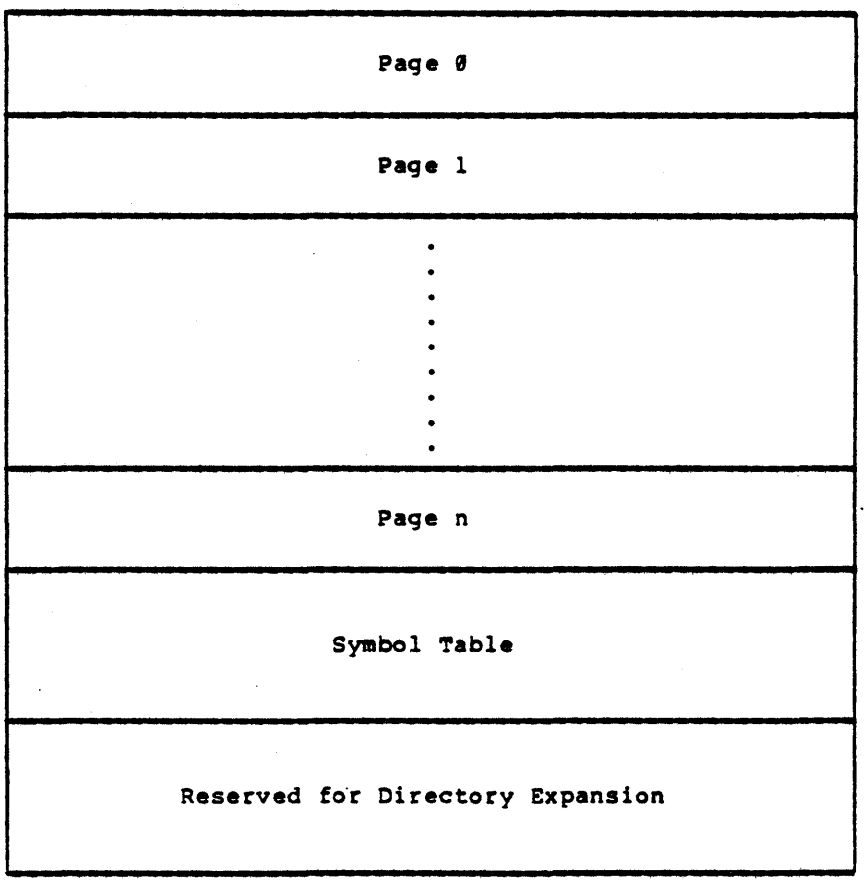
Name: DIRECTORY

Description: Directory Format. The following illustrations show the format of a TOPS-20 directory.

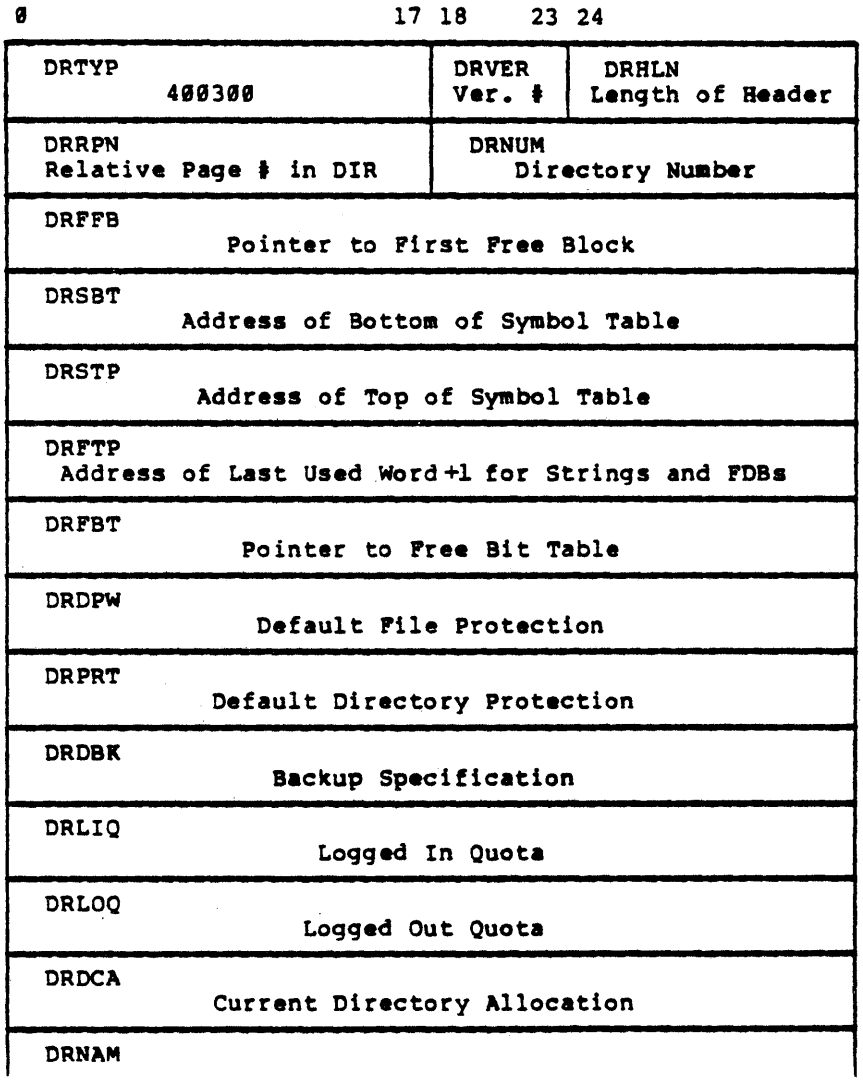
Defined: PROLOG

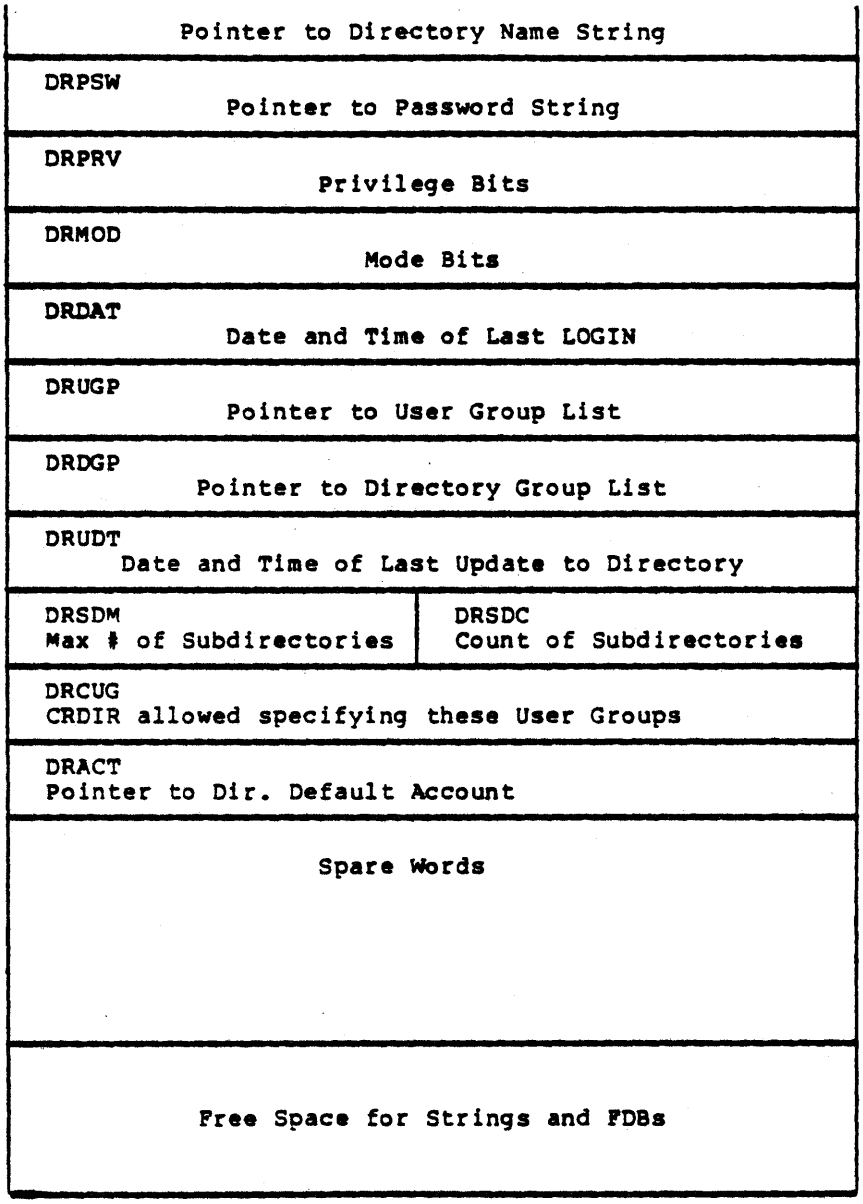
Referenced by: DIRECT, DISC, DSKALL

Overview of a Directory



First Page of a Directory



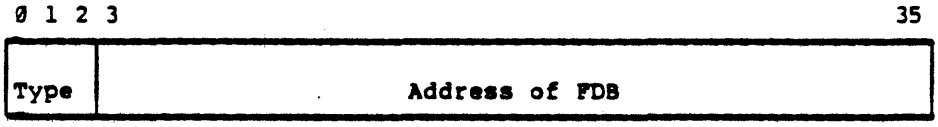

Pointer to Directory Name String
DRPSW
Pointer to Password String
DRPRV
Privilege Bits
DRMOD
Mode Bits
DRDAT
Date and Time of Last LOGIN
DRUGP
Pointer to User Group List
DRDGP
Pointer to Directory Group List
DRUDT
Date and Time of Last Update to Directory
DRSDM
Max # of Subdirectories
DRSDC
Count of Subdirectories
DRCUG
CRDIR allowed specifying these User Groups
DRACT
Pointer to Dir. Default Account
Spare Words
Free Space for Strings and FDBs

Subsequent Directory Pages

DRTYP 400300	DRVER Ver. #	DRHLN Length of Header
DRRPN Relative Page # in DIR	DRNUM Directory Number	
DRFFB Pointer to First Free Block		
Free Space for Strings and FDBs		

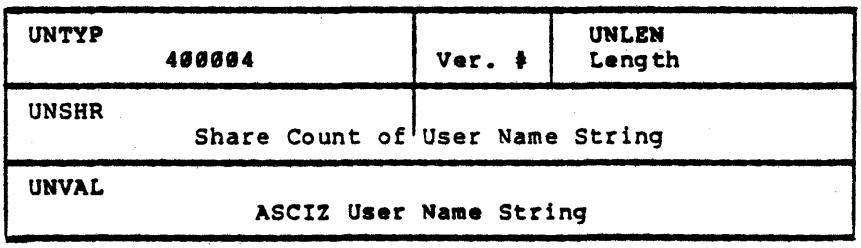
Symbol Table

SYMTY 400400		SYMDN Dir. # of Sym.Tbl.
-1		
SYMET Type	SYMAD Address of FDB	
SYMVL First 5 Characters of Name, Account or User Name for last writer/author		
. . . . .		

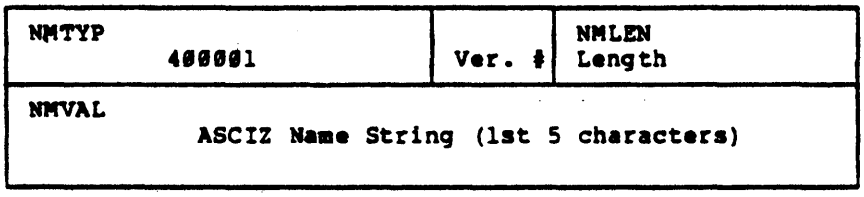


Bits	Pointer	Content	
0-2	SYMET	Entry Type	
		0 = .ETNAM	Name
		2 = .ETUNS	User Name
		4 = .ETACT	Account
3-35	SYMAD	Address of FDB	

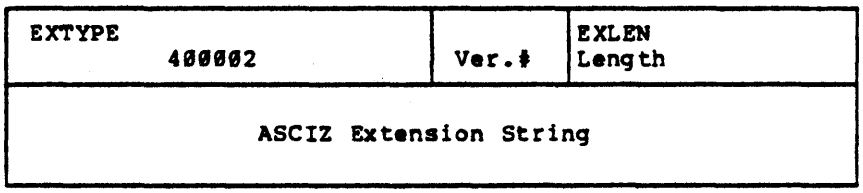
User Name String



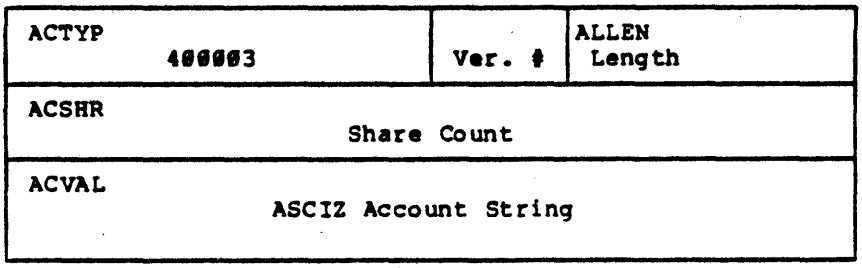
Name String



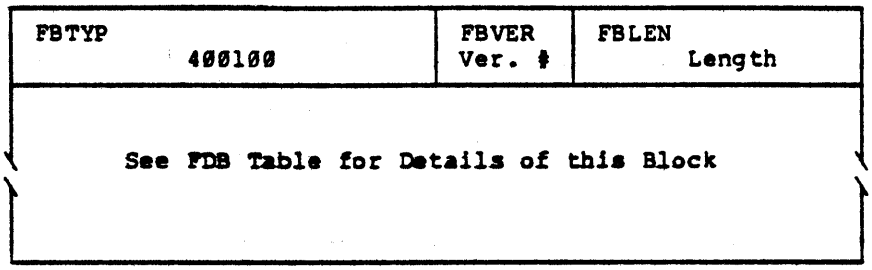
Extension String



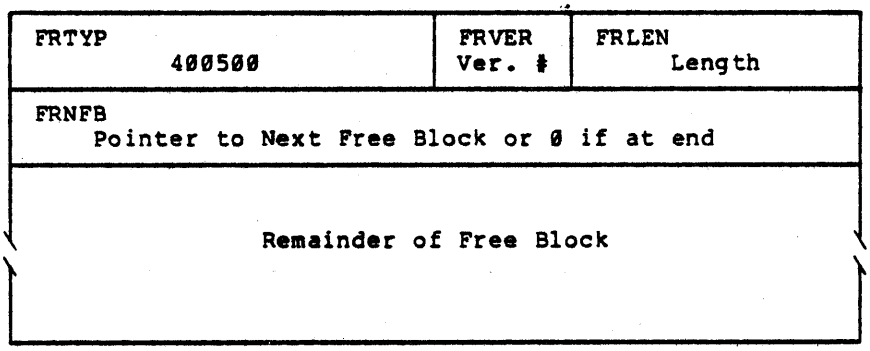
Account String



File Descriptor Block (FDB)



Free Space



Free Storage Bit Table

400600	Ver. #	Length
Bit Table Containing 1 Bit per Directory Page 0 = No Room on the Page 1 = There is Room on the Page		

Group List

400700	Ver. #	Length
Group #	Group #	
Group #	0	

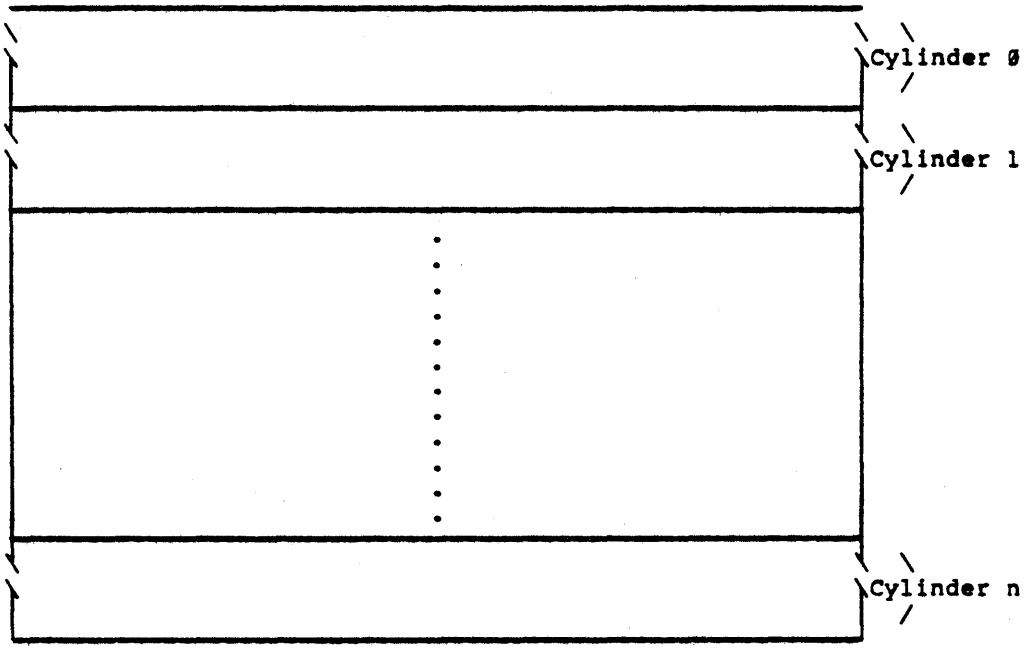
Name: DRMBBT

Description: Drum Bit Table. This bit table indicates which pages are in use and which pages are available in the swapping area.

Defined in: STG

Referenced by: SWPALC

Format



Note: The bit map for each cylinder starts on a word boundary and contains as many full words as are needed for all of its pages.

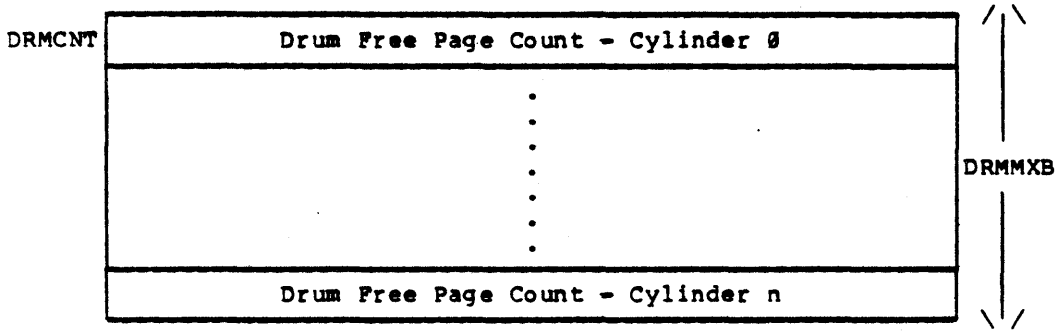
Name: DRMCNT

Description: Drum Count Table. This table, indexed by cylinder, records the free page count for the drum (logical swapping area). The total free page count for all the cylinders is kept in the storage word, DRMFRE.

Defined in: STG

Referenced by: SWPALC

Format



Name: DSKSIZ

Description: Disk Size Pointer Table. This table contains pointers to the disk size data tables. DSKSIZ is parallel to DSKUTP which contains codes for the known disk types. When an entry is added to DSKUPT, a corresponding entry must be added to DSKSIZ to point to the correct size data for that type of disk.

Defined in: STG

Referenced by: DSKALC

Format

DSKSIZ

Pointer to RP04 Table (DSKSZ0)
Pointer to RP05 Table (DSKSZ0)
Pointer to RP06 Table (DSKSZ1)
Pointer to RM03 Table (DSKSZ3)

Name: DSKSZ`n

Description: Disk Size Table (for type n). The resident table contains size data (for disks) based on type.

n = Ø for RPØ4 and RPØ5
n = 1 for RPØ6
n = 3 for RMØ3

Defined in: STG

Referenced by: DSKALC

Format

DSKSZ`n/SEGPAG=Ø	Sectors per Page
SECCYL=1	Sectors per Cylinder
PAGCYL=2	Pages per Cylinder
CYLUNT=3	Cylinders per Unit
SECUNT=4	Sectors per Unit
BTWCYL=5	No. of Bit Words in Bit Table per Cylinder
MINFPG=6	Minimum Free Pages for Free Choice Allocation
MAXFP=7	Pages per Unit for DSKASN turning point

Name: DSKUTP

Description: Disk Unit Type. This table contains the unit types used by the file system.

Defined in: PHYSIO

Referenced by: DSKALC

Format

DSKUTP

RP04 Disk Unit Code (.UTRP4 = 1)
RP05 Disk Unit Code (.UTRP5 = 5)
RP06 Disk Unit Code (.UTRP6 = 6)
RM03 Disk Unit Code (.UTRM3 = 11)

Name: DST

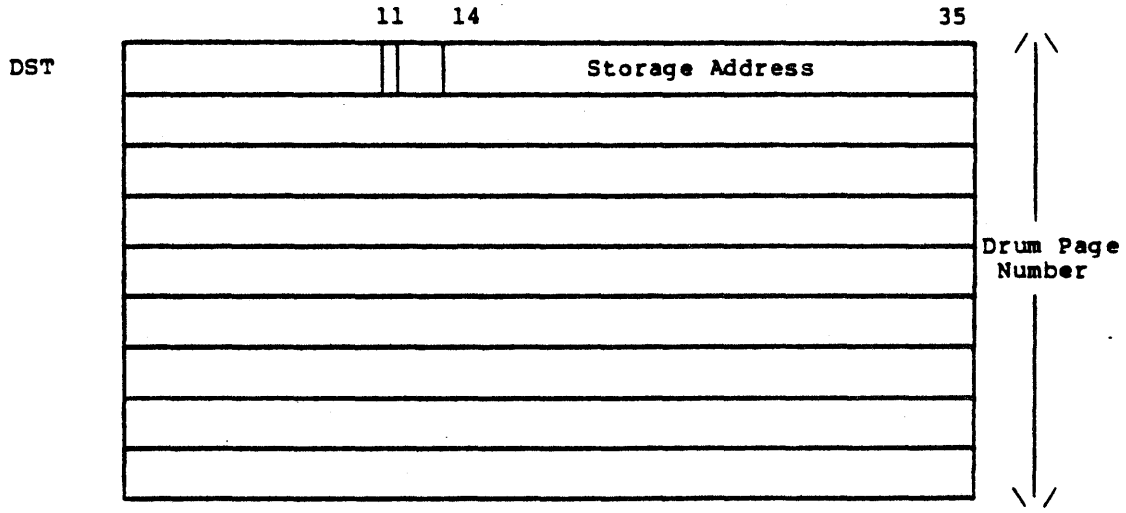
Description: Drum Status Table. This table is indexed as a function of the drum (swapping space) address. The routine GDSTX converts a drum address into a DST index. The DST holds the address of the next lower level of storage (usually disk) for the page stored at that address on the drum.

BWRBIT (bit 11) indicates if the page has been changed since being read from the lower level storage. The page will only be copied back on to the lower level storage if BWRBIT is set (i.e., page modified) when the page is no longer in use. A slot no longer in use contains a -1.

Defined in: STG

Referenced by: PAGEM

Format



Name: DTE-STORAGE-AREA

Description: DTE Storage Area. This storage area contains storage for each DTE. It contains the Communication Area for each processor in COMBAS, the linked output packet queues (pointed to by DTEQS), the DTE input buffers, and local storage (i.e., ACs, PC, & PDL) for the DTE Protocol Handler, DTESRV.

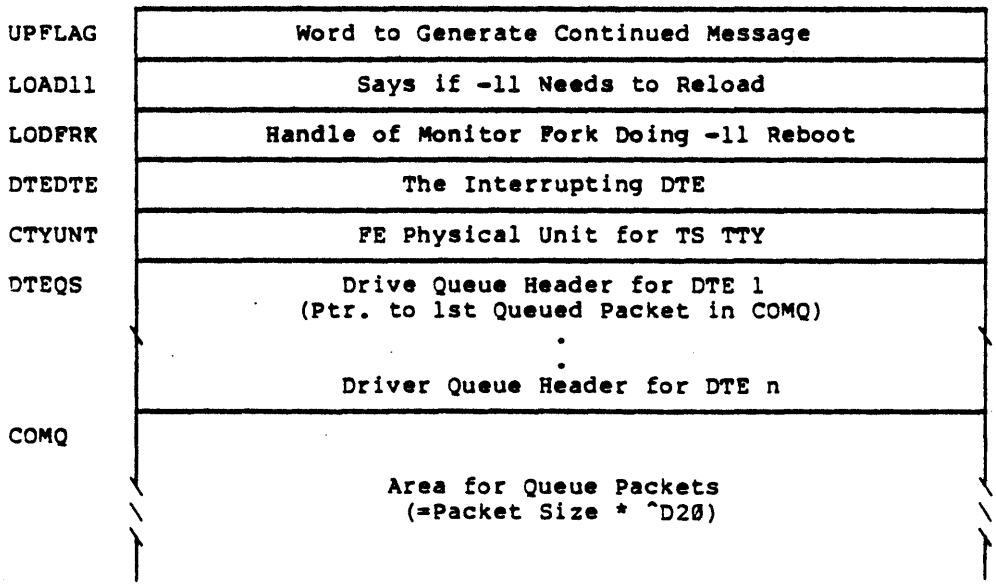
A packet in the COMQ area must be reformatted to RSX20F protocol and stored in PKTADR before being sent over the DTE. The before and after packet formats are described below.

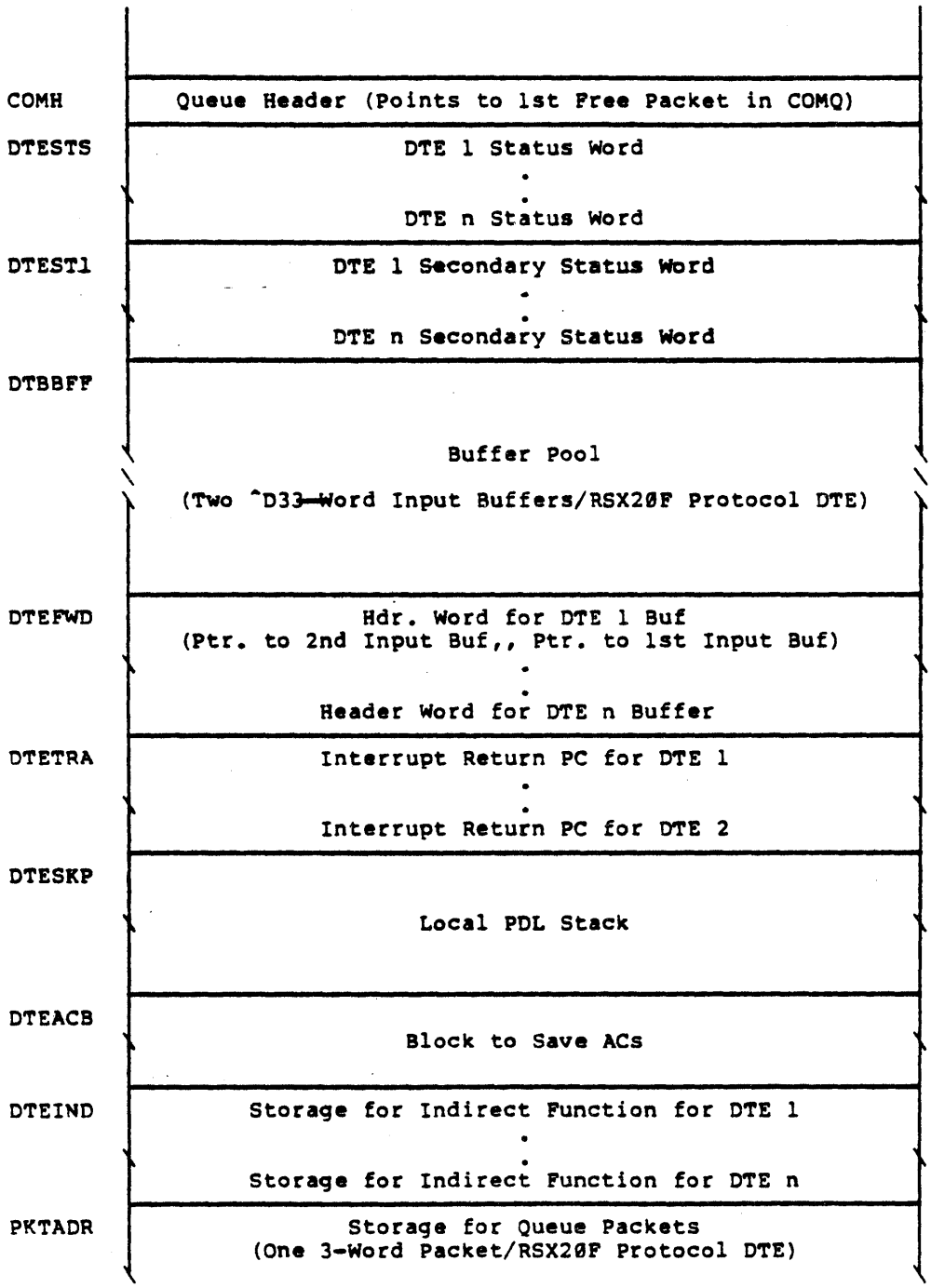
Two single packets, SNGPK1 and SNGPK2 (already formatted as direct packets to RSX20F protocol - See below) are set aside for the DTSNGL routine. This routine is responsible for activating lines and sending single characters over the DTE if the output buffer has only one character.

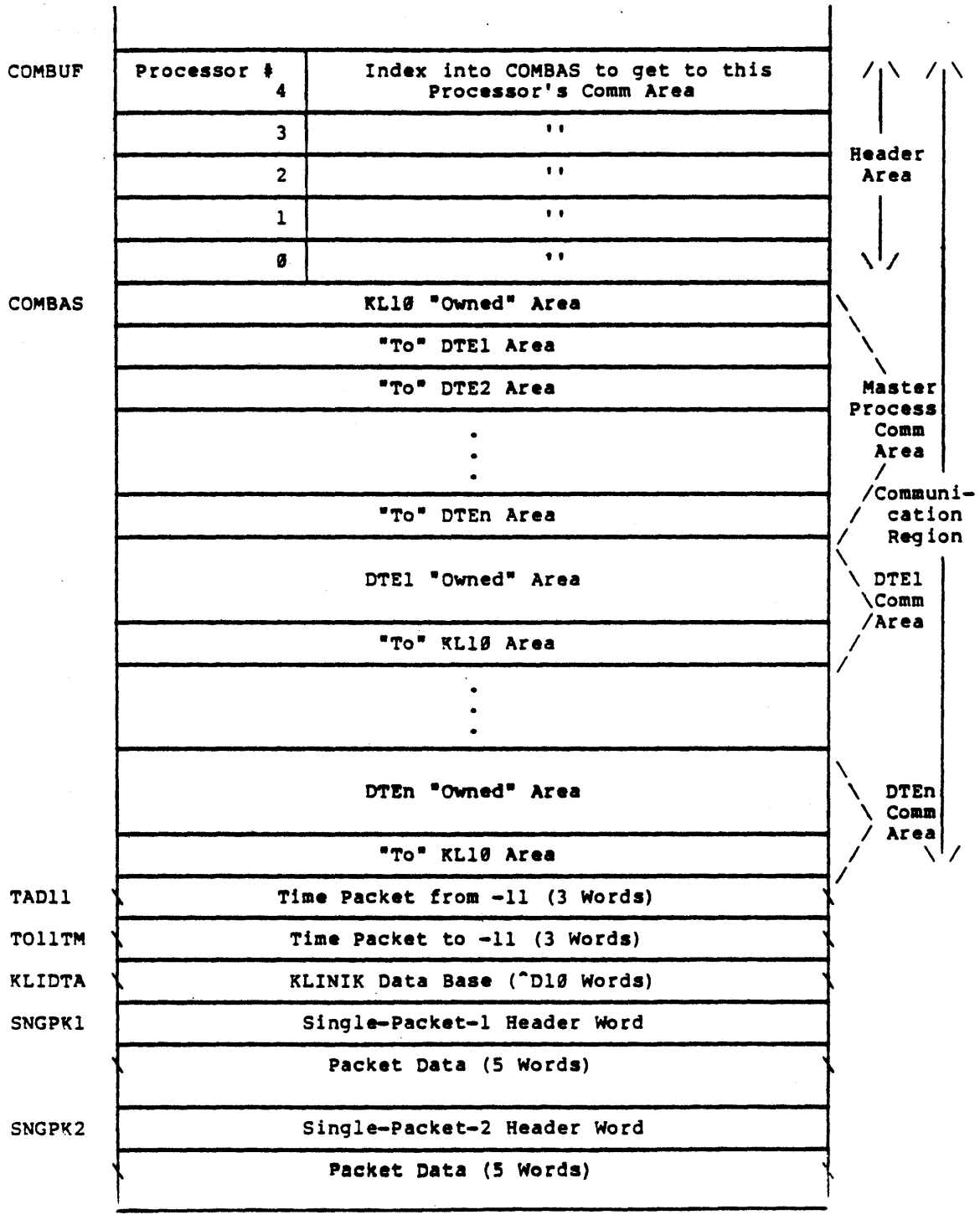
Normally output buffer characters are sent via indirect packets over the DTE, where the indirect packet (after being reformatted and stored in PKTADR) is sent first followed by the line's output buffer characters.

Defined in: STG

Referenced by: DTESRV, SPRSRV, MEXEC, SCHED




COMH
Queue Header (Points to 1st Free Packet in COMQ)
DTESTS
DTE 1 Status Word
DTE n Status Word
DTESTl
DTE 1 Secondary Status Word
DTE n Secondary Status Word
DTBBFF
Buffer Pool
(Two ^D33-Word Input Buffers/RSX20F Protocol DTE)
DTEFWD
Hdr. Word for DTE 1 Buf
(Ptr. to 2nd Input Buf,, Ptr. to 1st Input Buf)
Header Word for DTE n Buffer
DTETRA
Interrupt Return PC for DTE 1
Interrupt Return PC for DTE 2
DTESKP
Local PDL Stack
DTEACB
Block to Save ACs
DTEIND
Storage for Indirect Function for DTE 1
Storage for Indirect Function for DTE n
PKTADR
Storage for Queue Packets
(One 3-Word Packet/RSX20F Protocol DTE)

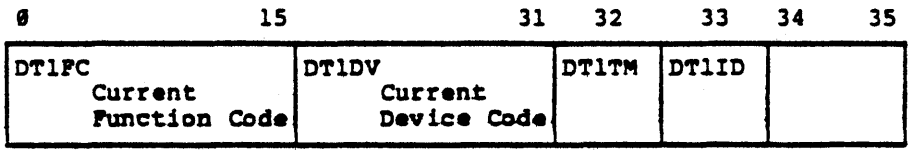

COMBUF
Processor # 4
Index into COMBAS to get to this Processor's Comm Area
3
''
2
''
1
''
0
''
Header Area
COMBAS
KL10 "Owned" Area
"To" DTE1 Area
"To" DTE2 Area
"To" DTEn Area
DTE1 "Owned" Area
"To" KL10 Area
DTEn "Owned" Area
"To" KL10 Area
Master Process Comm Area
Communi-cation Region
DTE1 Comm Area
DTEn Comm Area
TAD11
Time Packet from -11 (3 Words)
TO11TM
Time Packet to -11 (3 Words)
KLIDTA
KLINIK Data Base (^D10 Words)
SNGPK1
Single-Packet-1 Header Word
Packet Data (5 Words)
SNGPK2
Single-Packet-2 Header Word
Packet Data (5 Words)

DTESTS

0	1	2	3	4	5	6 17	18 29	30 35
DTERL DTE exists if set	DTEBF Which Buf is in use for RSX20F Protocol	DTPRV DTE Privi-ledged	DTRLD Set=>11 is being Reloaded	DTKAC Set=>11 is ill	DTSTI Status Packet is Split	DTEB1 Byte Count of Last Xfer.	DTEBC Byte Count Remaining for Subse-quent Xfer	DTEST DTE Status

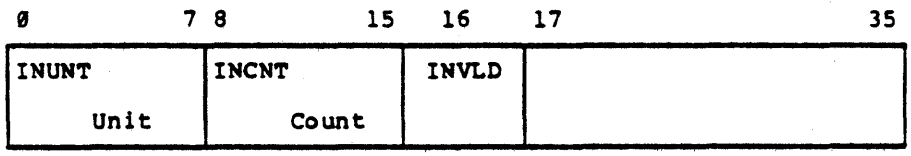
Bits	Pointer	Content
0	DTERL	If set, DTE exists
1	DTEBF	Says which buffer is in use for RSX20 of protocol
2	DTBLK	For MCB, to KL10 is blocked on free space
3	DTRLD	If set, -11 is being reloaded
4	DTKAC	If set, -11 is ill
5	DTSTI	Status pocket is split
6-17	DTEB1	Byte count of list transfer
18-29	DTEBC	Byte count remaining for subsequent transfer
30-35	DTEST	DTE status DTET10=1 KL10 is receiving last fragment of message DTE11=2 -11 is receiving bytes DTE11I=4 -11 is receiving an indirect queue entry DTE1F=10 KL10 is receiving 1st fragment of a message

DTEST1 Is parallel to DTESTS and contains current operation
data and special request bits for "To" -11 conditions.



Bits	Pointer	Contents
0-15	DT1FC	Current function code
16-31	DT1DV	Current device code
32	DT1TM	-11 wants time of day
33	DT1ID	Waiting for indirect setup

DTEIND (Storage for indirect packets)



Bits	Pointer	Contents
0-7	INUNT	Unit
8-15	INCNT	Count
16	INVLD	If set, says unit field is invalid

COMQ Area for queue packets where a packet (5 words in length) has the form:

Ø	QINT Int Loc for this Function	QLINK Link to Next Packet	
1	QFNC Function Word for this Request	QDEV DTE Dev. Code for this Request	
2	QLIN Device Unit #	QMODE indirect Data Must be Byte Mode	19 QCNT 35 Byte Count or Byte or Ø
3	QPNTR Byte Pointer for Indirect Operation or Local 8-bit Datum if QCNT = Ø		
4	QCOD Unique Code Returned to Interrupt Routine, TTYINT		

COMQ area is currently assembled for room of ^D2Ø packets.

PKTADR

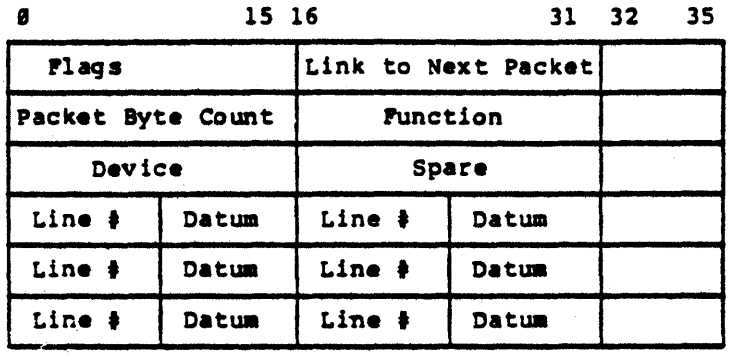
Storage for currently activated DTE packet for each DTE (Packet taken from the linked list of packets on the queue in COMQ and place here).

The packet has the following form:

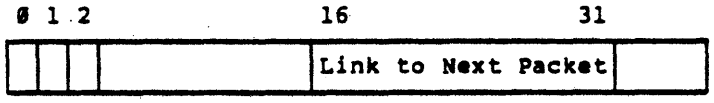
	Ø 15	16 31	32 35
Ø	HDCNT Count	HDFNC Function	
1	HDDEV Device Code	HDSPR Spare	
2	Ø 7 HDLIN Line # / 8 15 HDDAT Datum*		
	Ø or 15 HDDT1 Dataum for Single Datum Packet		

* Datum could be a character (direct packet case) or Max # of characters to be sent (indirect packet case)

SNGPK1/2

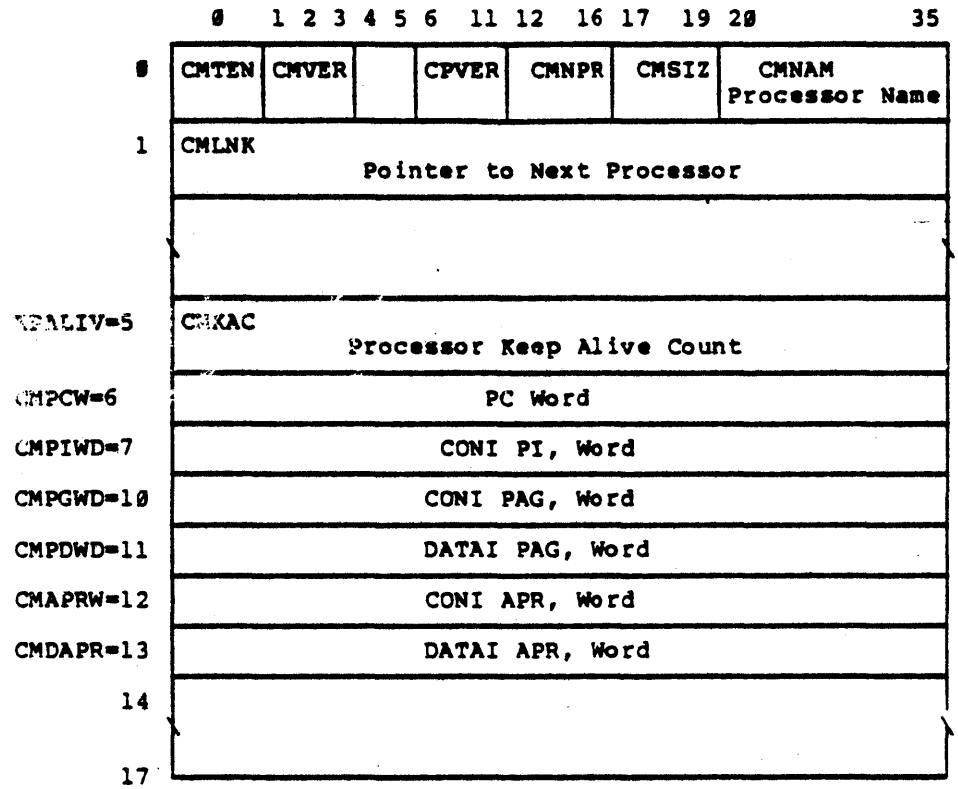


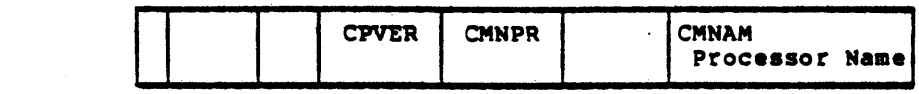
Header Word



Symbol	Bits	Contents
SNGONQ	0	On the DTE packet queue pointed to by DTEQS
SNGAVL	1	Packet has space available
SNGACT	2	Packet active (i.e. DTE processing it)
	16-31	Link to next packet

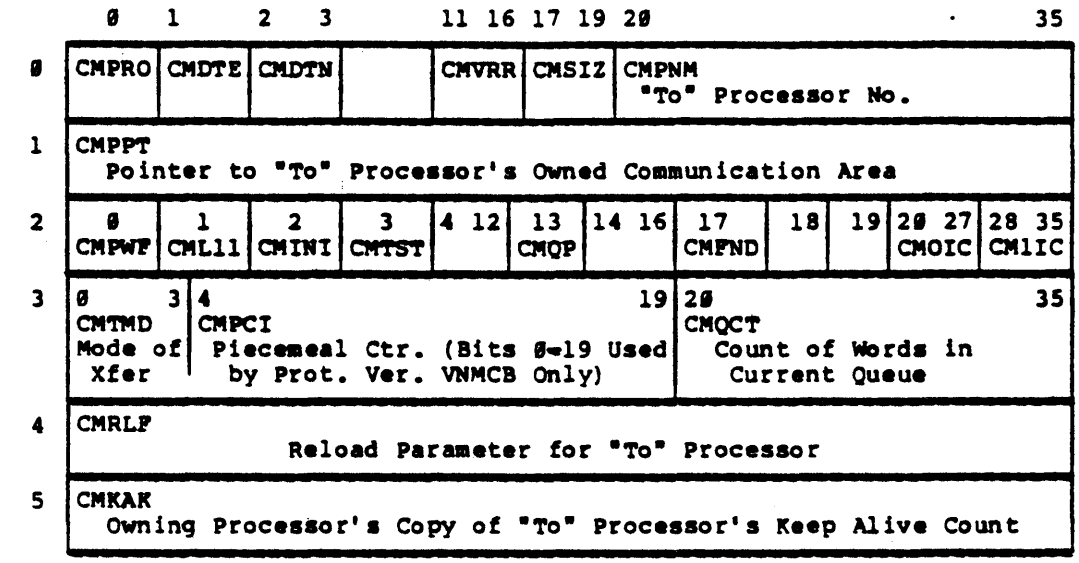
COMBAS
"Owned" Area Block Format

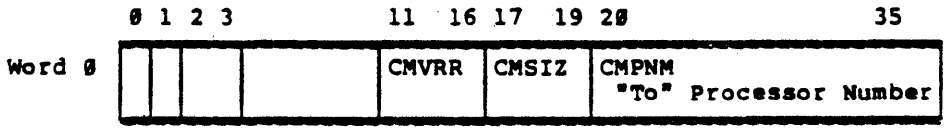




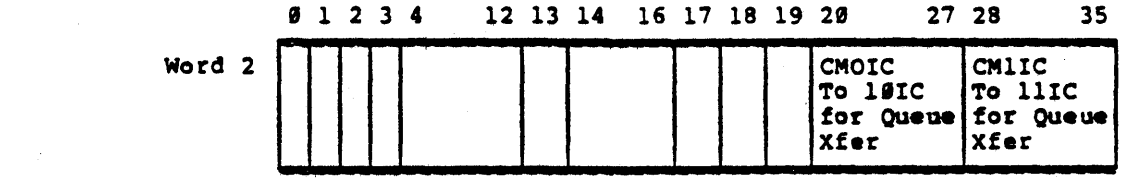
Bits	Pointer	Contents
0	CMTEN	Set if area belongs to KL10
1-3	CMVER	Communication area version number
6-11	CPVER	Protocol version number
12-16	CMNPR	Number of processors represented in this area (including owner)
17-19	CMSIZ	Size of area in 8-word blocks
20-35	CMNAM	Processor name (= serial number)

"To" Area Block Format





Bits	Pointer	Contents
0	CMPRO	If set, it implies connected to a KL10
1	CMDTE	If set, there is a DTE connecting this processor and owning processor
2-3	CMDTN	If CMDTE is set, this is the number of that connecting DTE
11-16	CMVRR	Protocol in use by the two processors
17-19	CMSIZ	Size of "to" area in 8-word blocks
20-35	CMPNM	"To" processor number



Symbol	Bits	Pointer	Contents
	0	CMPWF	Power fail indicator - 11
	1	CML11	Wants reload (set by -11)
	2	CMINI	Initialization bit for MCB protocol only
	3	CMTST	Valid examine if set (should always be set)
	13	CMQP	Set if using queued protocol
	17	CMFND	-11 doing full word transfer (set by -11)
CMIP	18		-11 doing indirect transfer
CMTOT	19		"Toit" bit. Set to 1 by KL10 in -11's section of -10's Comm area after -11 sets QMode bit or increments Q-count, and after -10 processes the doorbell. Cleared by KL10 after receiving T010DN. Assures -11 that the KL10 has not lost a T010DN interrupt
	20-27	CMOIC	-11s wrap around count of direct Q transfer
	28-35	CM1IC	KL10's wrap around count of direct Q transfers.

Name: DTEDTV

Description: DTE Protocol Device Dispatch Table. The entries with the dispatch address, TTYDTV are for the CTY, DL11, DH11 and DLS devices.

Defined in: STG

Referenced by: DTESRV

Format

DTEDTV

Reserved for Unknown Device
TTYDTV
TTYDTV
TTYDTV
TTYDTV
LPTDTV
CDRDTV
0 (Unknown Device)
FEDTV

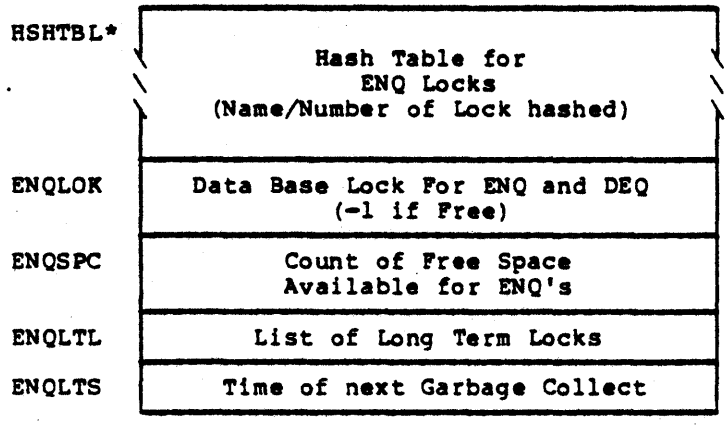
Name: ENQ/DEQ - STORAGE AREA

Description: Enqueue/Dequeue Storage Area. The non-resident local area for the ENQ/DEQ Facilty is illustrated first followed by the resident bit tables, ENFKTB and LCKDBT. A bit is set in the ENFKTB bit table if the fork should be woken up or interrupted because it owns a lock. (The Scheduler's wake-up test routine address is ENQTST.)

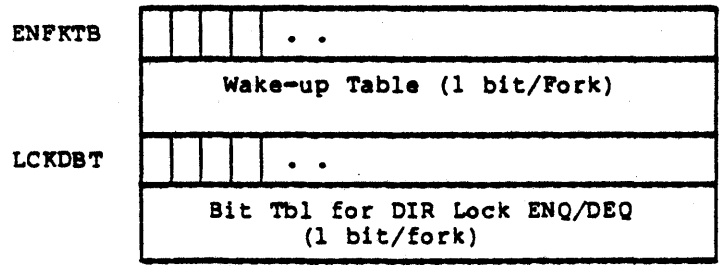
Defined in: STG

Referenced by: ENQ, IPCF, DIRECT

FORMAT



Resident Storage:



*The name (or identifying number) of a lock block is hashed to provide a number; This number, modulo the size of the hash table is used as an index into HSHTBL. If the hashing algorithm yields the same index for more than one lock block name, the lock blocks will be linked together; the HSHTBL entry will be the linked list header.

Name: ENQ-LOCK-BLOCK

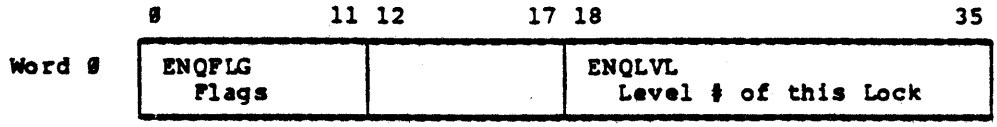
Description: Each resource is described in a lock block. The lock block is created at the time of the first request.

Defined in: ENQ

Referenced by: ENQ

FORMAT

0	ENQLHC: Back Pointer to Last Lock-Block on Hash Chain	ENQNHC: Pointer to Next Lock-Block on Hash Chain
1	ENQLLQ: Back Pointer to Last Q-Block on Queue	ENQNLQ: Forward Pointer to First Q-Block on Queue
2	ENQFLG: Flags	ENQLVL: Level Number of this Lock
3	ENQTR: Total # of Resources in this Pool	ENQRR: Remaining Number of Resources in this Pool
4	ENQTS: Time Stamp Time of Last Request Locked	
.ENQLT=5	ENQFBP: Free Block Pointer to Free Q-Block	ENQLT: Long Term Lock List for this Job
6	ENQOFN: OFN, or -2, or -3, or 400000 + Job Number	ENQLEN: Length of this Lock-Block
7	ENQNMS: Number of Words in the Mask Block	
.ENTXT=10	ENQTXT: ASCIZ String or 500000 + User Code	



Symbol	Bits	Pointer	Meaning
EN.LTL=40	6		Long Term Block
EN.INV=20	7		This Q-Block is invisible
EN.LOK=10	8		The Q-Block has the Lock Locked.
EN.TXT=4	9		This Block has a Text String Identity.
EN.EXC=2	10		Request is Exclusive
EN.LB=1	11		This is the Lock-Block
	13-17	ENQCHN	PSI Channel (-1 means job Blocked)
	18-35	ENQLVL	Level # of this lock.

Name: EPT

Description: Executive Process Table. This memory resident table pointed to by the Executive Base Register (EBR), contains the vectored dispatch addresses for system events. All device interrupts pass control to a specific offset position in this table.

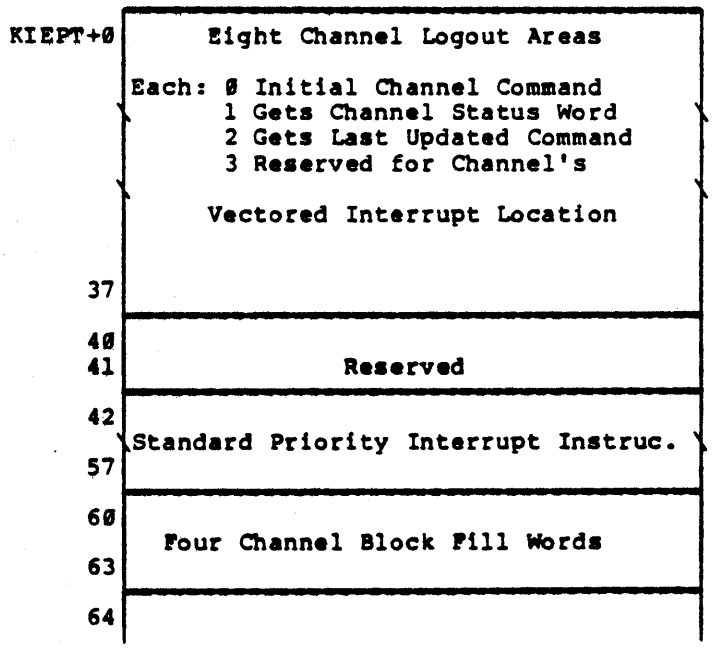
This table also includes the executive section map table, the time of day clock and arithmetic trap instructions which are executed when arithmetic conditions occur in executive mode.

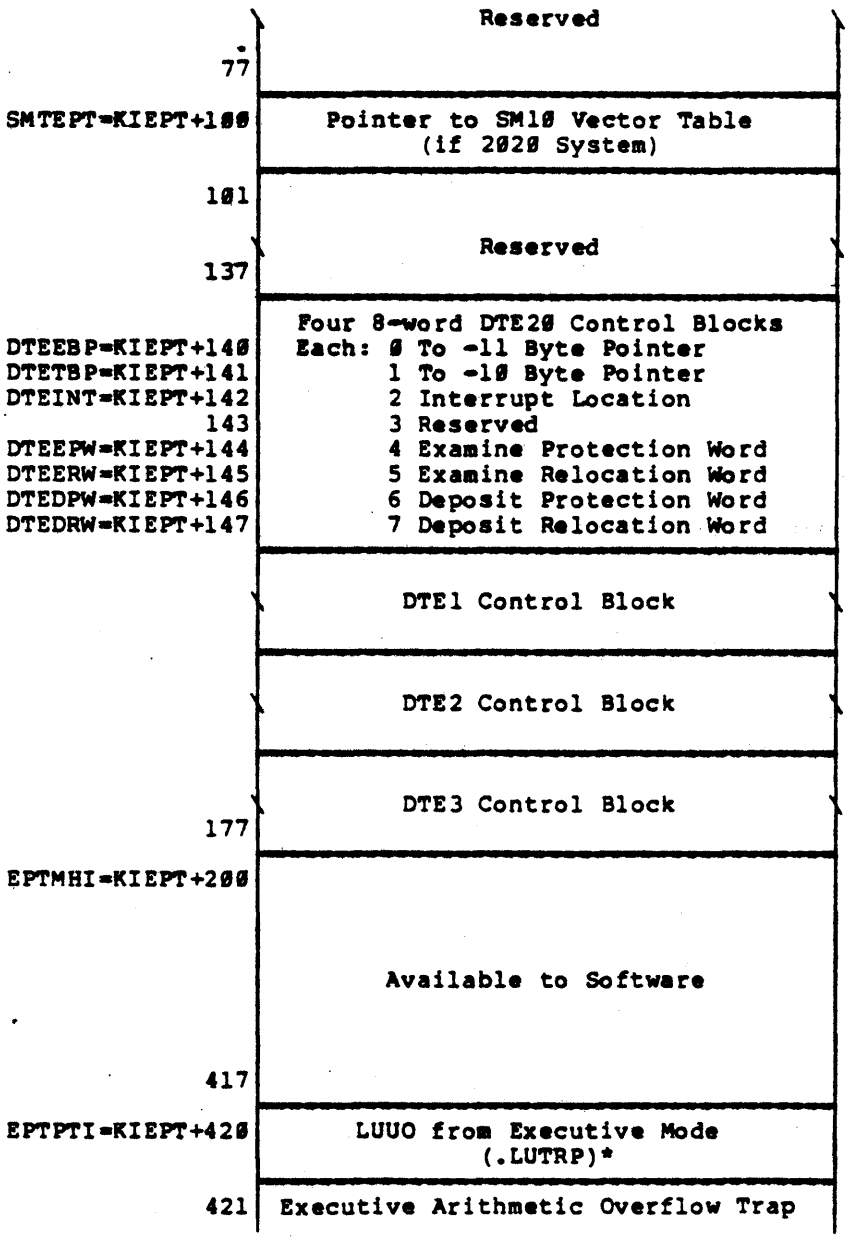
Locations 444 to 457 are reserved for software and used by DTESRV.

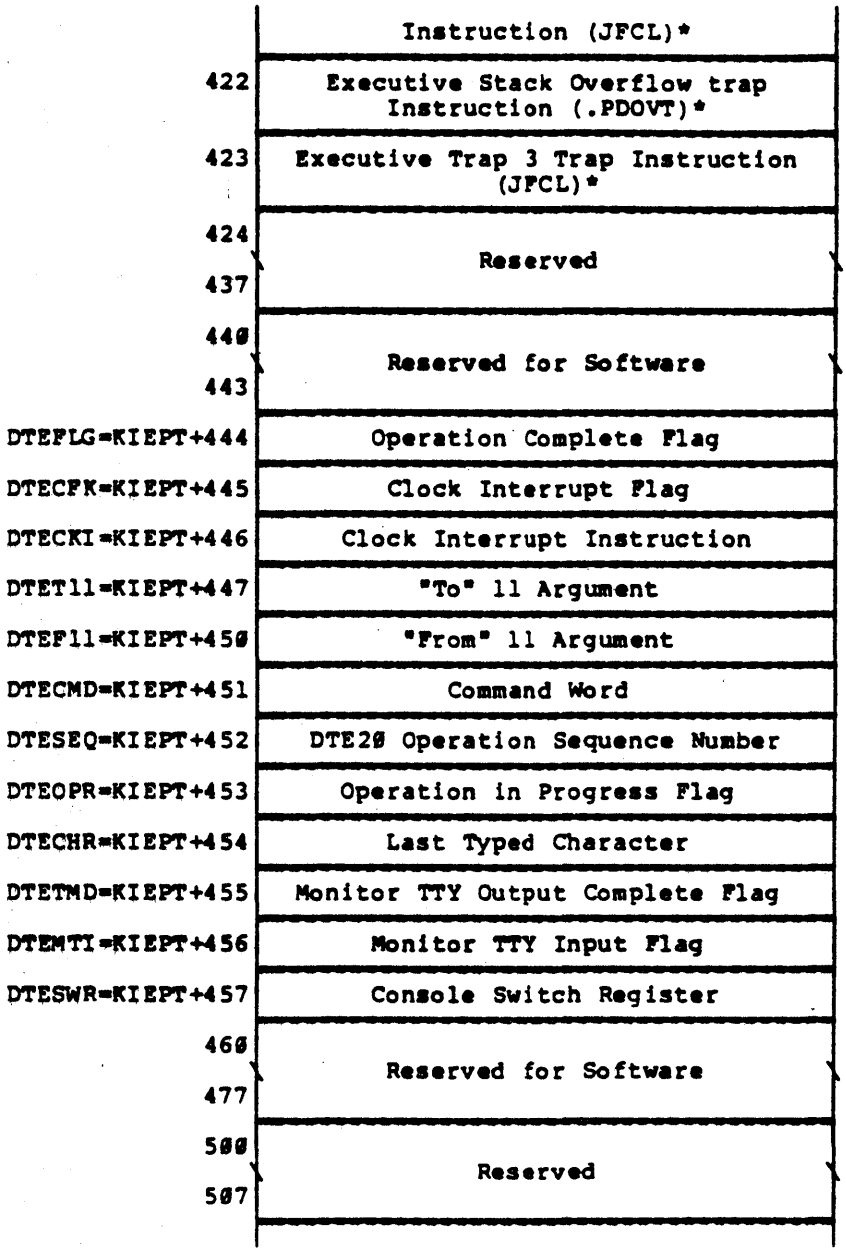
Defined In: STG

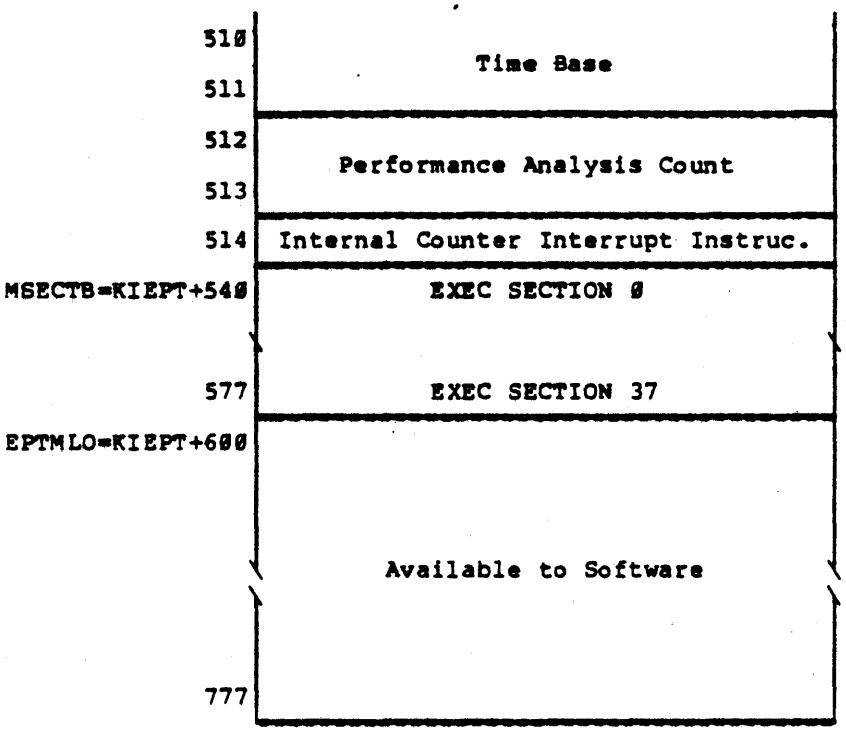
Referenced by: APRSRV, DTESRV, MEXEC, PHYH11, PHYH2

FORMAT




Reserved
77
SMTEPT=KIEPT+100
Pointer to SM10 Vector Table
(if 2020 System)
101
Reserved
137
Four 8-word DTE20 Control Blocks
DTEEBP=KIEPT+140
DTETBP=KIEPT+141
DTEINT=KIEPT+142
143
DTEEPW=KIEPT+144
DTEERW=KIEPT+145
DTEDPW=KIEPT+146
DTEDRW=KIEPT+147
Each: 0 To -11 Byte Pointer
1 To -10 Byte Pointer
2 Interrupt Location
3 Reserved
4 Examine Protection Word
5 Examine Relocation Word
6 Deposit Protection Word
7 Deposit Relocation Word
DTE1 Control Block
DTE2 Control Block
DTE3 Control Block
177
EPTMHI=KIEPT+200
Available to Software
417
EPTPTI=KIEPT+420
LUUO from Executive Mode
(.LUTRP)*
421
Executive Arithmetic Overflow Trap


Instruction (JFCL)*
422
Executive Stack Overflow trap Instruction (.PDOVT)*
423
Executive Trap 3 Trap Instruction (JFCL)*
424
437
Reserved
440
443
Reserved for Software
DTEFLG=KIEPT+444
Operation Complete Flag
DTECFK=KIEPT+445
Clock Interrupt Flag
DTECKI=KIEPT+446
Clock Interrupt Instruction
DTET11=KIEPT+447
"To" 11 Argument
DTEF11=KIEPT+450
"From" 11 Argument
DTECMD=KIEPT+451
Command Word
DTESEQ=KIEPT+452
DTE20 Operation Sequence Number
DTEOPR=KIEPT+453
Operation in Progress Flag
DTECHR=KIEPT+454
Last Typed Character
DTETMD=KIEPT+455
Monitor TTY Output Complete Flag
DTEMTI=KIEPT+456
Monitor TTY Input Flag
DTESWR=KIEPT+457
Console Switch Register
460
477
Reserved for Software
500
507
Reserved



* These values are placed into the table when the EPT is initialized at system startup.

Name: EXEC-PG-MAP-TABLE

Description: Executive Page Map Table. This 512-word memory resident table holds or points to other tables that hold all of the mapping information needed by the firmware to translate executive (monitor) virtual addresses in a given section into physical memory addresses. It is pointed to by an entry in the monitor's section table in the Executive Process Table (EPT).

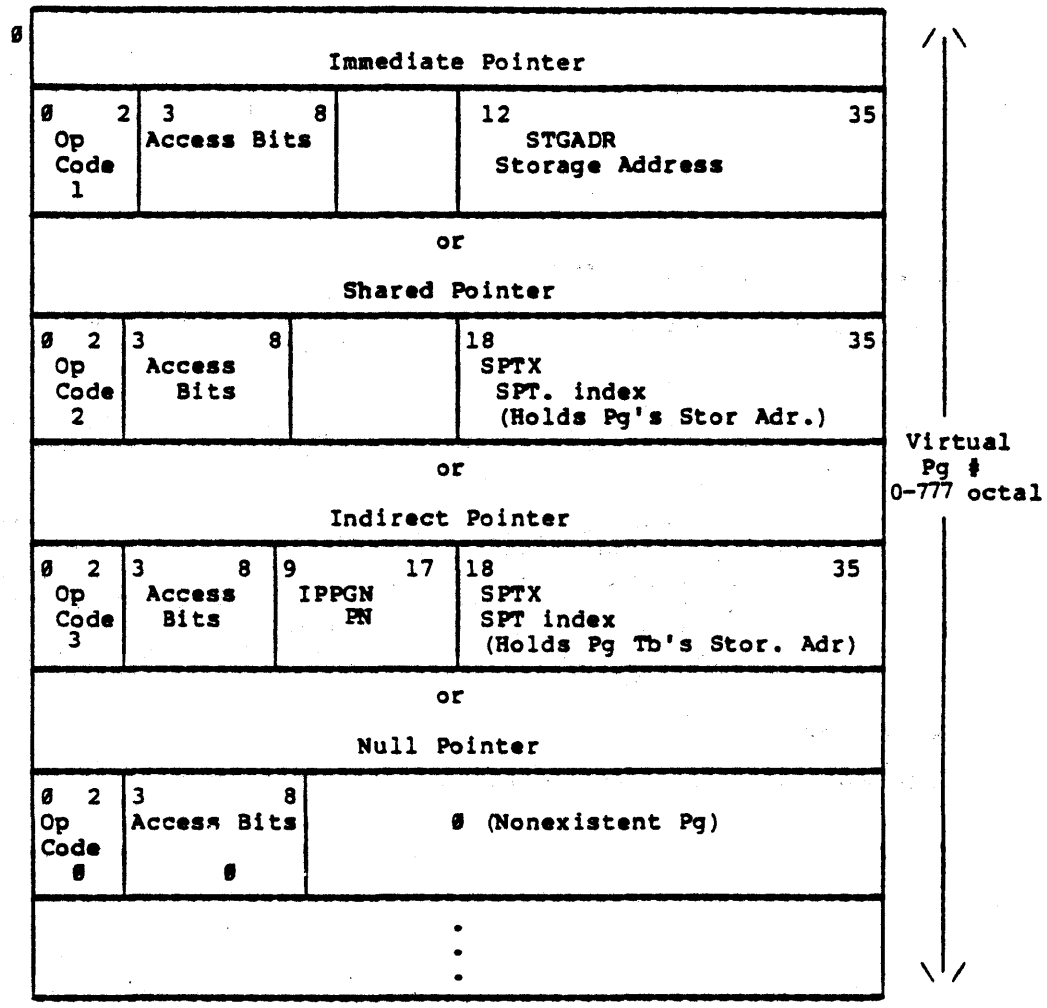
The four possible formats for an entry in this table (i.e., Immediate, shared, indirect or null pointers) are illustrated below. The details of these four possible pointer words as well as the mechanics of the virtual to physical translation process for a monitor page is identical to that described for the User-Page Map Table (See User-Pg-Map-Tbl)

Defined in: STG

Referenced by: All Monitor Modules

FORMAT

MMAP*



* Currently MMAP is the monitor's page map table for section Ø and section 1. The layout of the monitor's virtual address space for section Ø is described in Appendix B of the Monitor Structures Book.

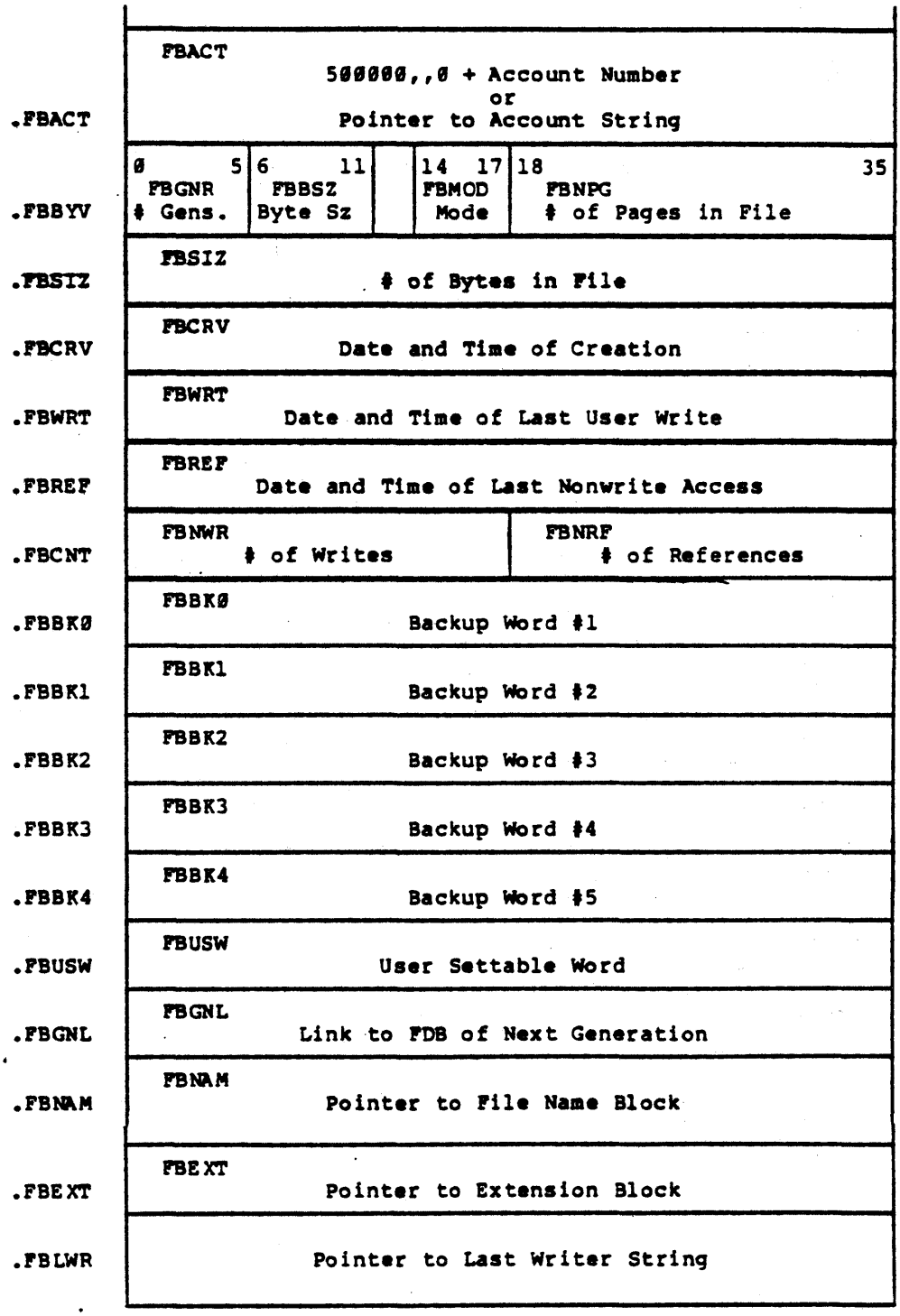
Name: FDB

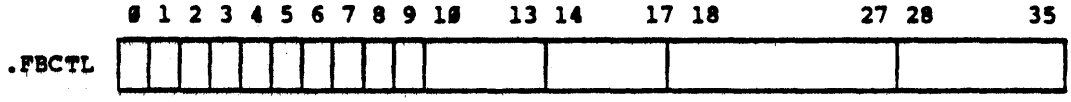
Description: File Description Block. All attributes of a file are stored in its description block (FDB) maintained in the file's directory. An FDB is built in the directory's free space area when a file is created. This table is referenced by the DIR table.

Defined in: PROLOG, MONSYM

Referenced by: DISC, DIRECT, DSKALC, GTJEN, JSYSA, JSYSF, FILINI, LINEPR, IO, SYSERR, DTESRV

	0 17	18 23	24 35
.FBHDR	FBTYP 400100	FBVER Ver. #	FBLEN Length
.FBCTL	FBFLG Flags		
.FBEXL	FBEXL Link to FDB of Next Extension		
.FBADR	FBADR Disk Address of File's Index Block		
.FBPRT	FBPRT 500000	File Access Bits	
.FBCRE	FBCRE Date and Time of Last Write to File		
.FBUSE	FBLW0 DIR # of Last Writer	FBAT0 DIR # of Author	
.FBAUT	FBAUT Pointer to Author String		
.FBLWR	FBLWR Pointer to Last Writer String		
.FBGEN	FBGEN Generation Number	FBDRN Dir.# (if it's a Dir File)	


FBACT
500000,,0 + Account Number
or
Pointer to Account String
.FBACT
0 5
FBGNR
Gens.
6 11
FBBSZ
Byte Sz
14 17
FBMOD
Mode
18 35
FBNPG
# of Pages in File
.FBBYV
FBSIZ
# of Bytes in File
.FBSIZ
FBCRV
Date and Time of Creation
.FBCRV
FBWRT
Date and Time of Last User Write
.FBWRT
FBREF
Date and Time of Last Nonwrite Access
.FBREF
FBNWR
# of Writes
FBNRF
# of References
.FBCNT
FBBK0
Backup Word #1
.FBBK0
FBBK1
Backup Word #2
.FBBK1
FBBK2
Backup Word #3
.FBBK2
FBBK3
Backup Word #4
.FBBK3
FBBK4
Backup Word #5
.FBBK4
FBUSW
User Settable Word
.FBUSW
FBGNL
Link to FDB of Next Generation
.FBGNL
FBNAM
Pointer to File Name Block
.FBNAM
FBEXT
Pointer to Extension Block
.FBEXT
Pointer to Last Writer String
.FBLWR



Symbol	Bits	Pointer	Content
FB%TMP	0		File is temporary
FB%PRM	1		File is permanent
FB%NEX	2		No extension for this file yes; file doesn't really exist.
FB%DEL	3		File is deleted
FB%NXF	4		File doesn't exist (first write not complete)
FB%LNG	5		Long file
FB%SHT	6		
FB%DIR	7		File is a directory
FB%NOD	8		File is not saved by backup system
FB%BAT	9		File may have bad pages
FB%FCF	14-17		File class field 0 = .FBNRM Not an RMS file 1 = .FBRMS RMS file

Note: See Monitor Call's Reference Manual (Chapter 2) for more information.

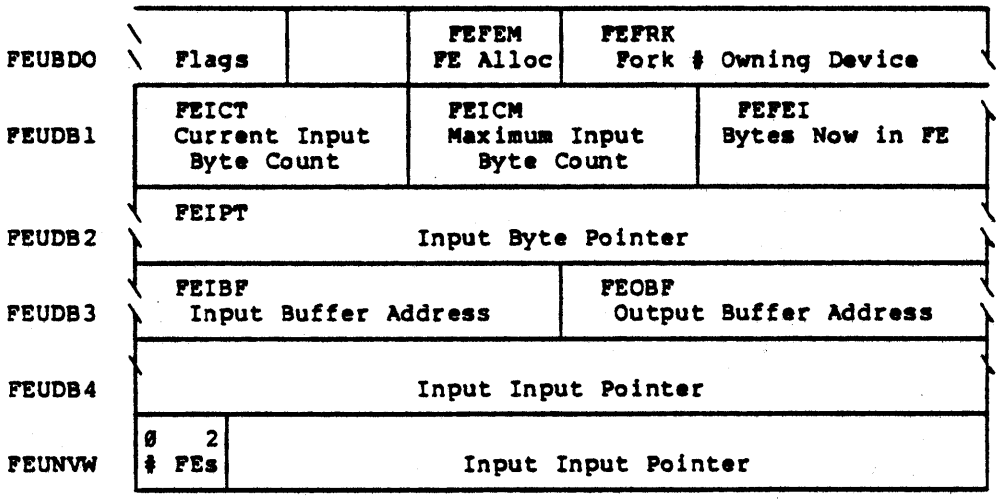
Name: FE-STORAGE-AREA

Description: Storage area for front end devices. Each entry is FEN words long (except FEUNVW), where FEN equals the number of front end devices.

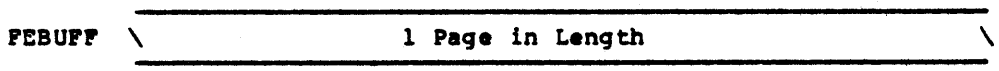
Defined in: STG

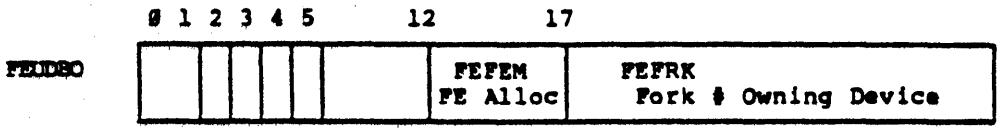
Referenced by: FESRV

Format



The buffer area for the front end is in the monitor's nonresident address space.





Bit(s)	Pointer	Content
0-1	FEDTE	DTE owning this device
2	FEBLK	Unit is blocked
3	FEPST	Waiting for DTE Post
4	FEEOF	Input EOF declared by FE
5	FEVDT	FE assignment is valid
12-17	FEFEM	FE Allocation
18-35	FEFRK	Fork # owning device

Name: FKCNO

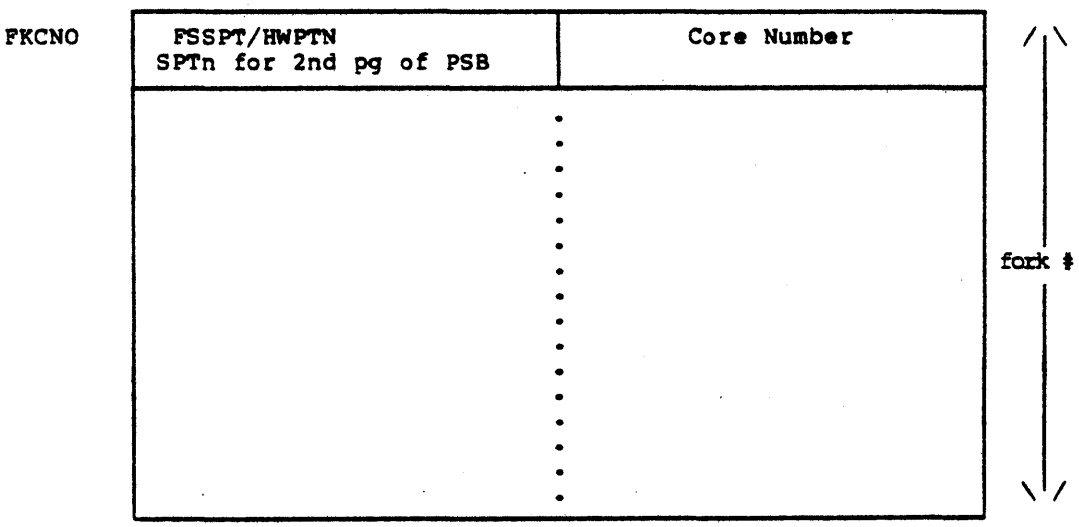
Description: Fork Core Number Table. This table, indexed by fork #, contains the core number (given when a process enters the balance set) for each fork. The core number is used to set a corresponding bit in the pager's process use register (PUR) when the fork is chosen to run.

The left half of this table is used to hold the SPT index for the second page of each fork's PSB table. (see FKPGS table for SPT index for first page of each fork's PSB).

Defined in: STG

Referenced by: PAGEM, SCHED

Format



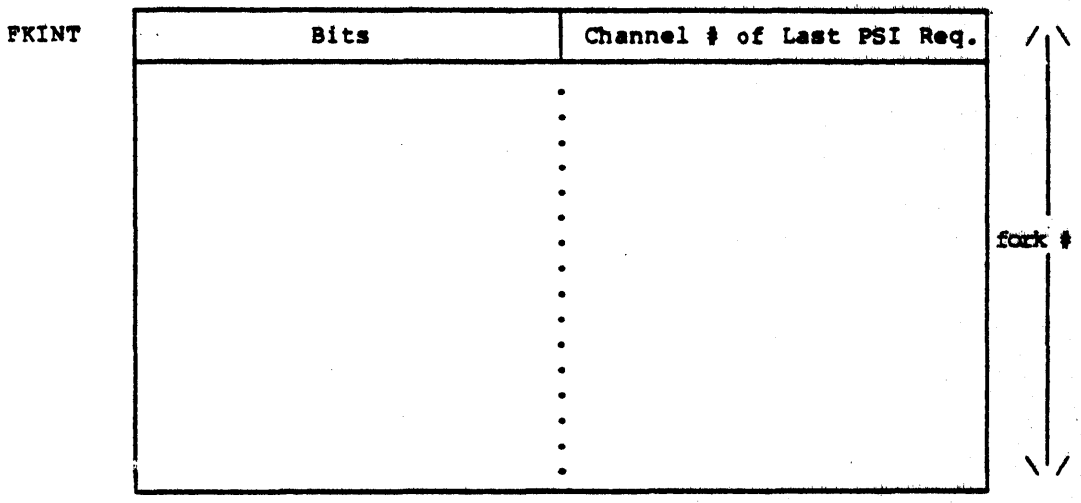
Name: FKINT

Description: Fork Interrupt Table. This table, indexed by fork #, contains the pseudo-interrupt communication register for each fork. The left half of each entry contains bits recording the type of request. The symbols for these requests have right half bit assignments (i.e. bits 20-31) but are tested against the left half of the table.

Defined in: STG

Referenced by: TTYSRV, FORK, MEXEC, SCHED

Format



Symbol	Bit	Content
	0	Interrupt Request(s) pending
	1	Fork not interruptable
NEWFKF(B20)	2	Initiate new fork
NEWJBF(1B21)	3	Initiate new job

PSIIF(1B22)	4	Channel interrupt requested in FKINTB
PSIT1F(1B23)	5	Terminal code Interrupt, Phase 1
PSIT2F(1B24)	6	Terminal code Interrupt, Phase 2
SUSFKR(1B25	7	Suspend fork request
PSIWTF(1B26)	8	Job was in wait state
PSILOB(1B27)	9	Logout job request
FRZB1(1B28)	10	Direct freeze has been done
FRZB2(1B29)	11	Indirect freeze has been done
PSICOB(1B30)	12	Carrier off action request
PSITLE(1B31)	13	Time Limit Exceeded interrupt
PSIJTR(1B32)	14	JSYS trap request
JTFRZB(1B33)	15	JSYS trap freeze
ADRBKF(1B34)	16	Address Break Request
ABFRZB(1B35)	17	Address Break Freeze

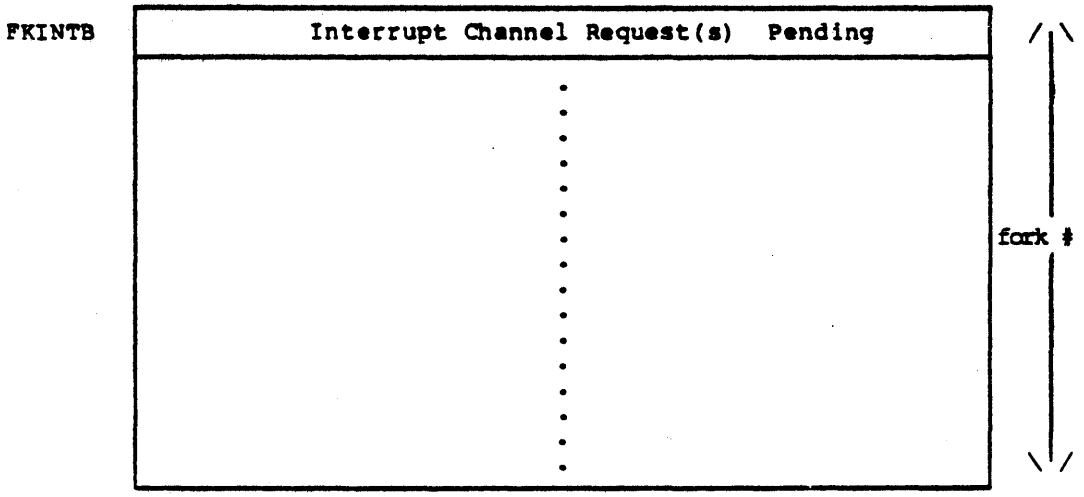
Name: FKINTB

Description: Fork Interrupt Buffer Table. This table, indexed by fork #, contains the buffer for the pseudo-interrupt channel requests pending for each fork since the fork's last PSI interrupt.

Defined in: STG

Referenced by: SCHED

Format



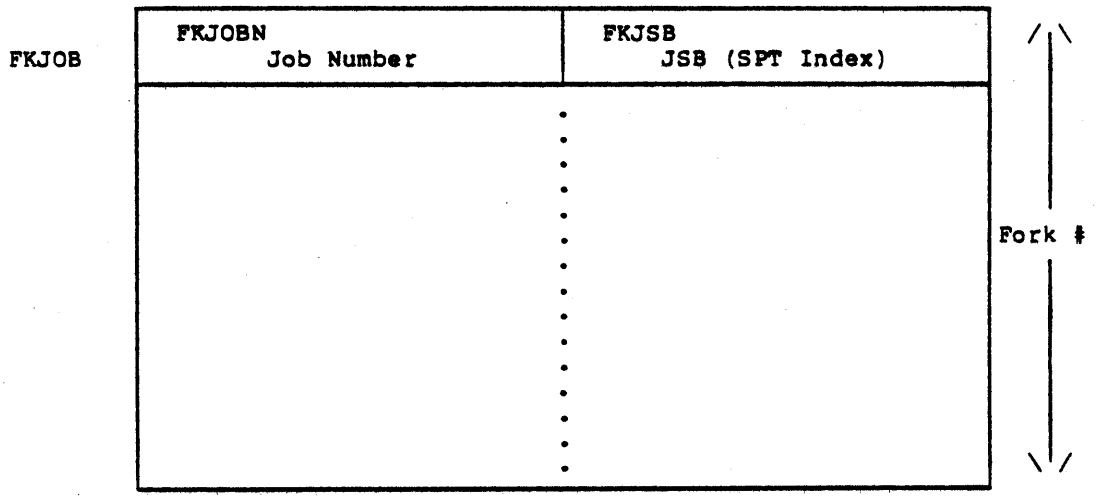
Name: FKJOB

Description: Fork Job Table. This table, indexed by fork #, holds each fork's job number and JSB address (SPT index).

Defined in: STG

Referenced by: APRSRV, DTESRV, ENQ, FILMSC, FORK, IPCF, MEXEC, PAGEM, PHYSIO, SCHED

Format



Name: FKJTQ

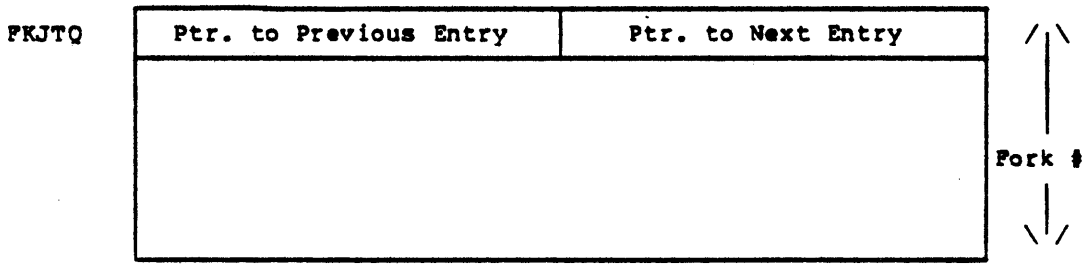
Description: Fork JSYS Trap Queue. This doubly linked list is a JSYS Traps Queue of forks waiting to program software interrupt (PSI) the monitor. JTLST points to the top fork on the linked JSYS traps queue in FKJTQ.

When a fork tries JTLOCK (in the JSB) and some other fork has the lock, the fork is added to FKJTQ and blocked. When the lock is cleared, the queue is scanned for the first fork (if any) waiting on the lock. That fork is removed from the queue and allowed to run.

Defined in: STG

Referenced by: SCHED

FORMAT



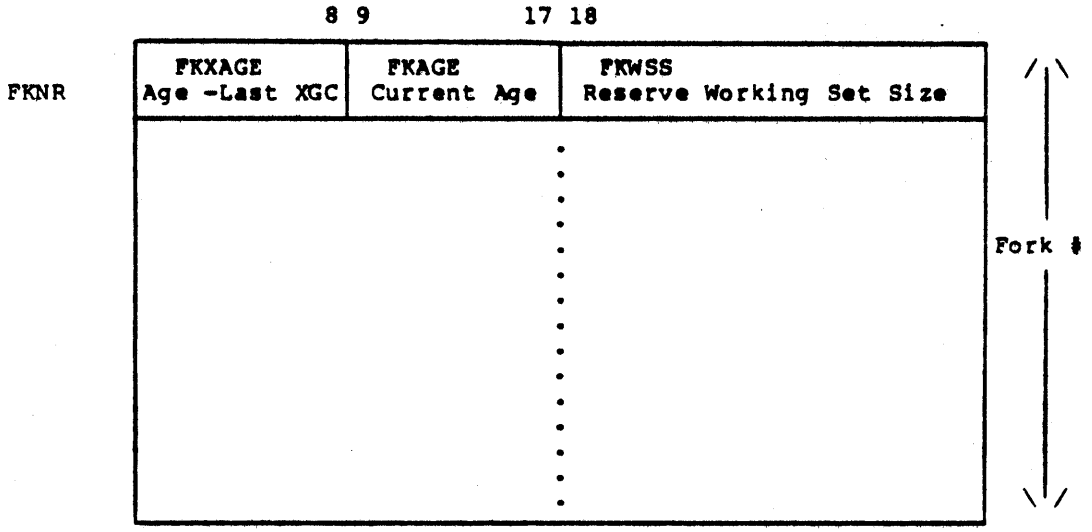
Name: FKNR

Description: Fork Number of Reserve Pages Table. This table, indexed by fork #, contains in the right half the current reserve working set size for each fork. Also, it contains in the left half the current age stamp (to be loaded into the pager's age resister when the fork is chosen to run) and the age stamp value at the last time local garbage collection (removal of less frequently used pages) took place for the fork.

Defined in: STG

Referenced by: PAGEM, SCHED

Format



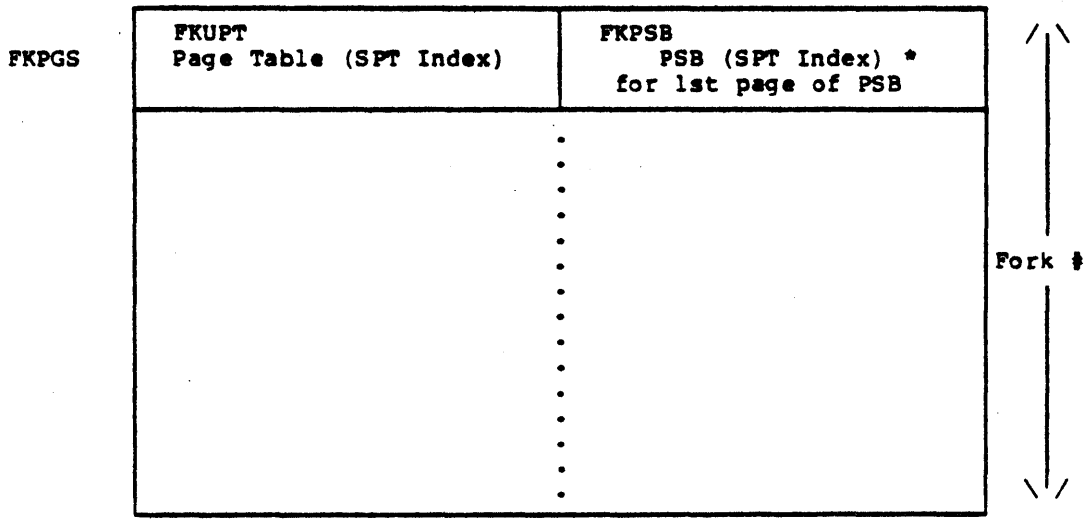
Name: FKPGS

Description: Fork Page and Process Storage Table. This table, indexed by fork #, contains the page table and PSB locations (SPT indexes) for each fork.

Defined in: STG

Referenced by: FORK, IPCF, JSYSA, PAGEM, SCHED

Format



* See FKCNO table description for SPT index for second page of PSB.

Name: FKPGST

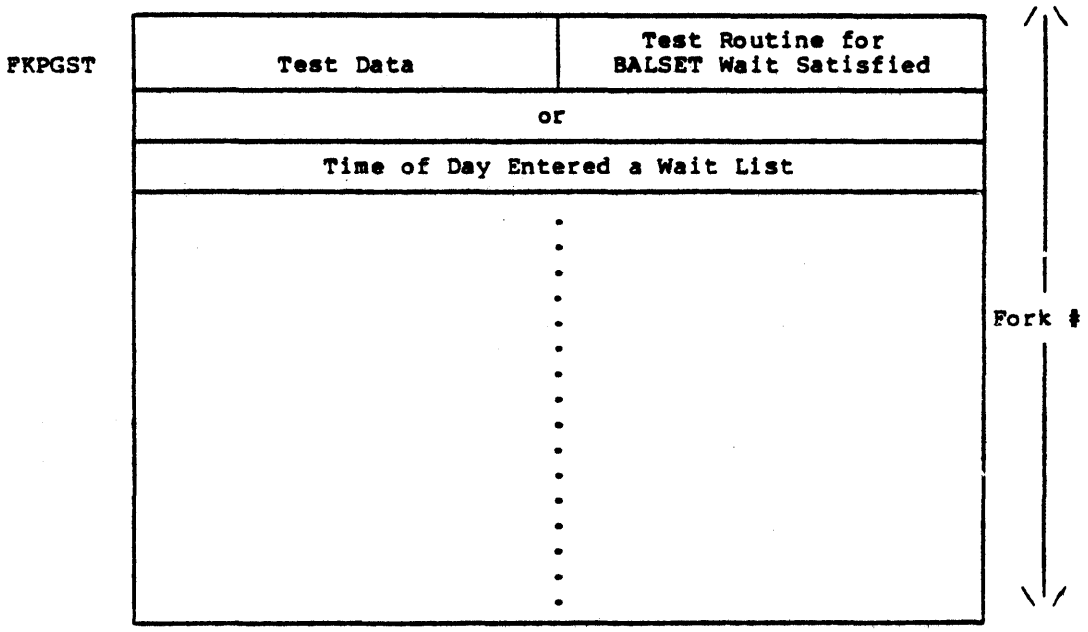
Description: This table, indexed by fork #, holds test routine information for forks in a balance set wait state. The test routine checks if wait satisfied has occurred.

For forks on a wait list (and therefore not in the balance set), this table contains the time of day the fork entered the list.

Defined in: STG

Referenced by: SCHED

Format



Name: FKPT

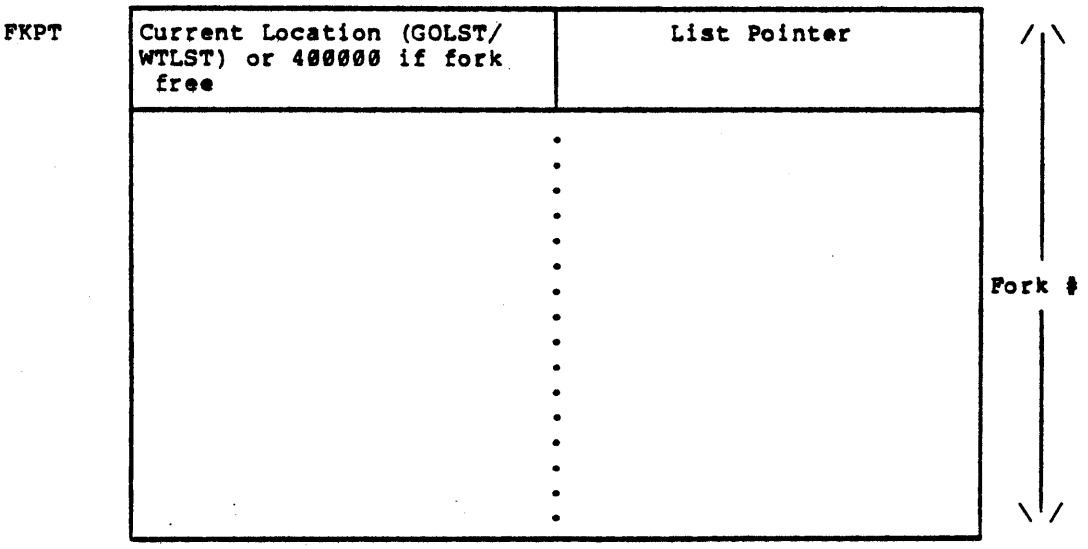
Description: Fork List Pointer Table. This table, indexed by fork #, gives the chain of forks for each list of forks in the system. That is, it holds the linked list of forks on TTILST, CLKLST, GOLST, etc. A fork is either on one of the wait-lists or the go-list.

The right half contains the list pointer to the next fork on the same list and the left half contains WTLST or GOLST. If the left half contains WTLST, the type of wait-list can be obtained from the right half of FKQ2.

Defined in: STG

Referenced by: APRSRV, FORK, MEXEC, PAGEM, SCHED

Format



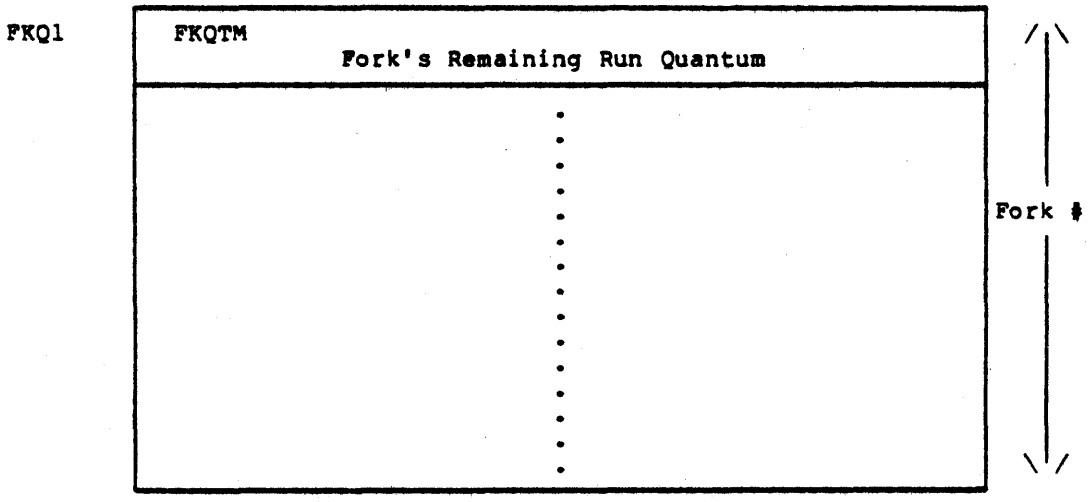
Name: FKQ1

Description: Fork Run Queue Table 1. This table, indexed by fork #, contains each fork's remaining run quantum. When this quantum expires, the process will be requeued to a lower run position and given the the quantum associated with that run queue.

Defined in: STG

Referenced by: SCHED

Format



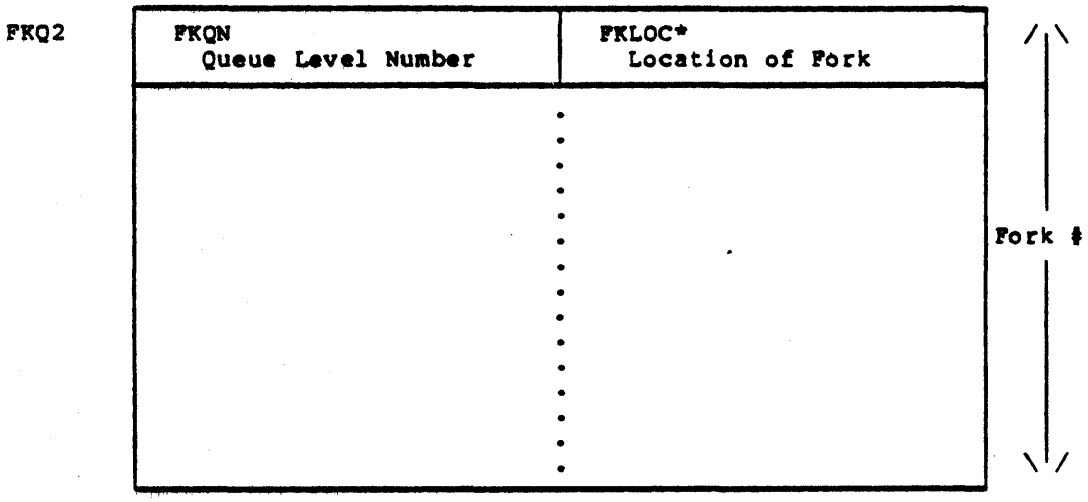
Name: FKQ2

Description: Fork Run Queue Table 2. This table, indexed by fork #, contains the queue level number and fork location [i.e., list address (TTILST, GOLST, etc.) or BALSET table index] for each fork.

Defined in: STG

Referenced by: PAGEM, SCHED

Format



* If FKLOC contains a number less than NPB (size of BALSET table), then the fork is in the balance set. Otherwise, it contains the list address of which list it's on.

Name: FKSTAT

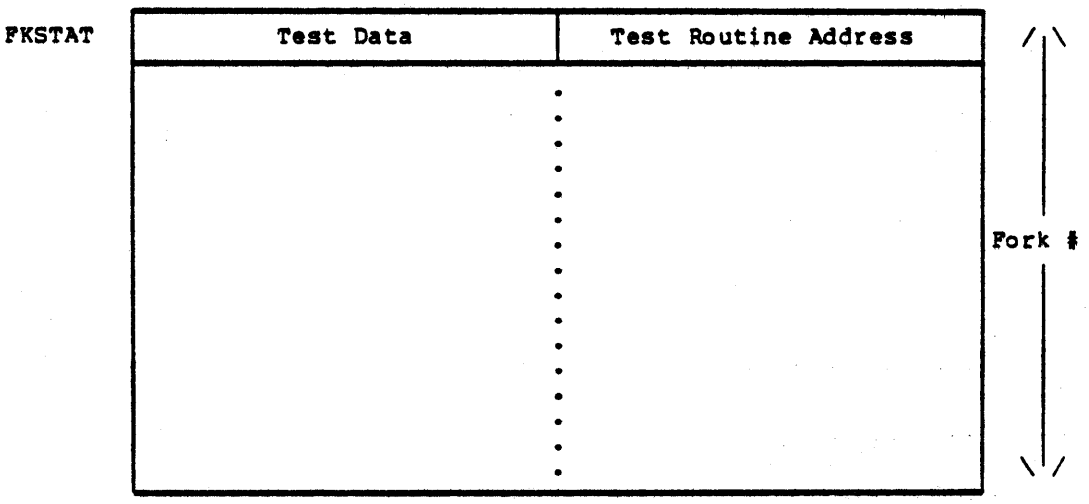
Description: Fork Status Table. This table, indexed by fork #, has useful information when a fork blocks and leaves the GOLST (i.e., LH (FKPT) contains the list address, WTLST).

The blocked fork's entry in this table will contain the address of the test routine which, when called, determines if wait satisfied has occurred for the fork.

Defined in: STG

Referenced by: DIRECT, FORK, SCHED

Format



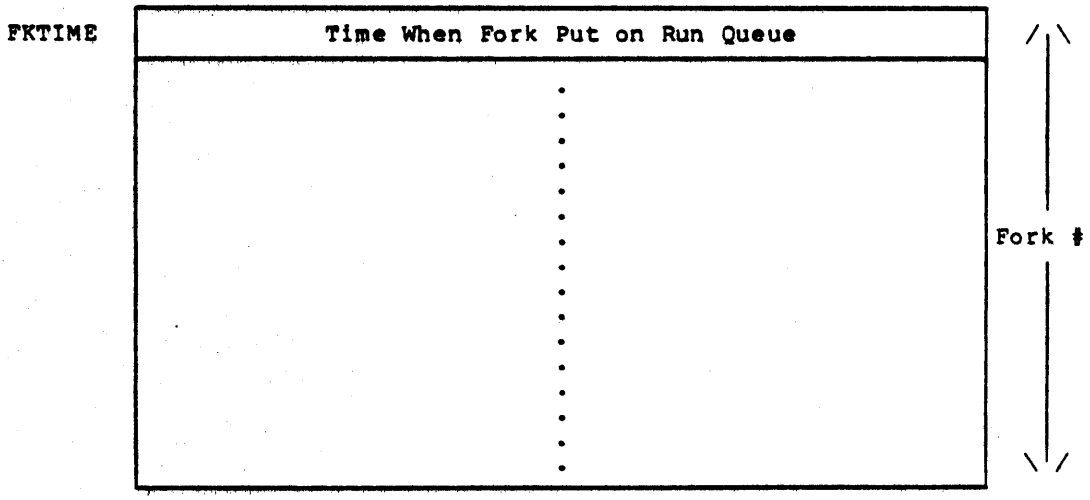
..ame: FKTIME

Description: Fork Time Table. This table, indexed by fork #, gives the time of day (TODCLK) at which each fork was put on its current run queue.

Defined in: STG

Referenced by: SCHED

Format



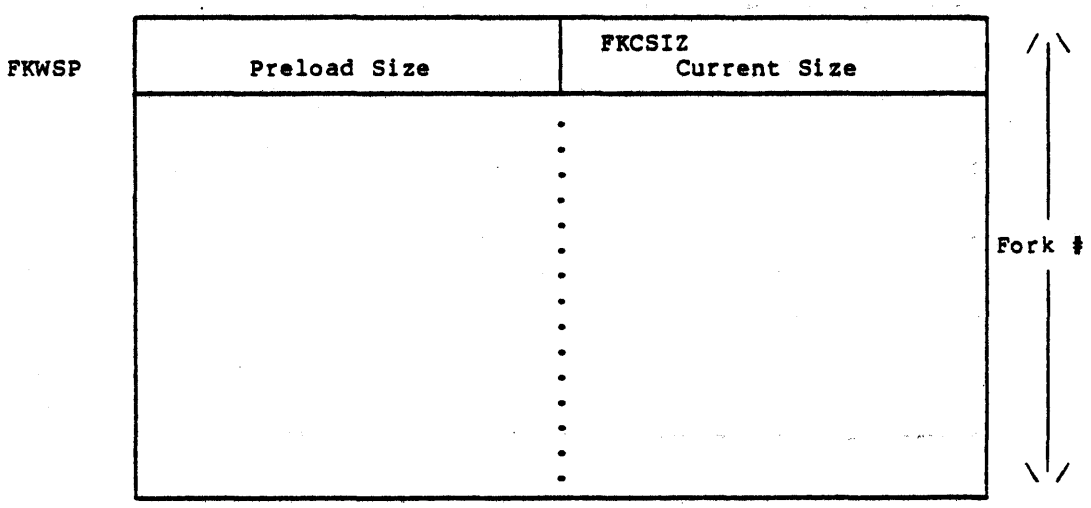
Name: FKWSP

Description: Fork Working Set (physical in-core size) Table. This table, indexed by fork #, contains in the right half the number of physical pages currently assigned to each fork. The left half is used to hold the pre-load size as determined by LDJOB when a fork enters the balance set.

Defined In; STG

Referenced by: PAGEM, SCHED

Format



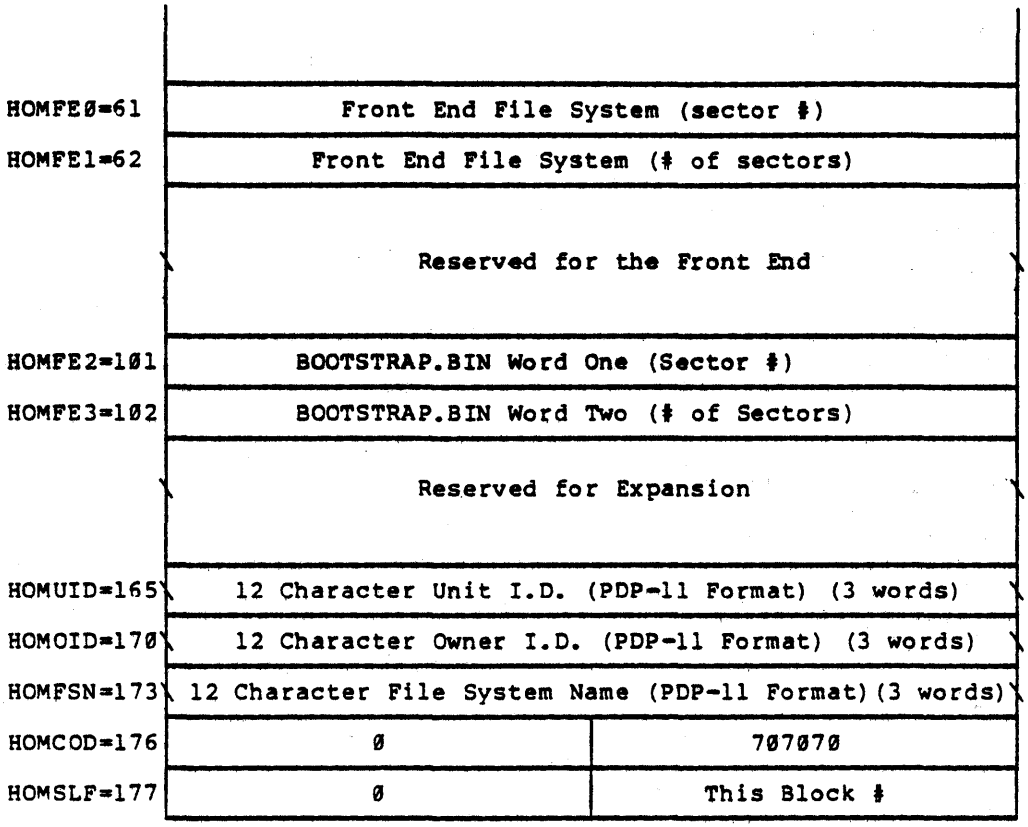
Name: HOM

Description: Home Block. Block on each disk unit which contains vital statistics that cannot be built in when a monitor is generated. These are primarily parameters of the unit and the STR to which it belongs.

Defined in: DSKALC

Referenced in: DSKALC, PHYSIO, JSYSA

HOMNAM=0	SIXBIT/HOM/	
HOMID=1	SIXBIT/Unit ID/	
HOMPHY=2	Physical Disk Address of This Home Block	Physical Disk Address of Other Home Block
HOMSNM=3	SIXBIT/Structure Name/	
HOMLUN=4	# of Packs in STR	Logical Unit # Within STR
HOMHOM=5	Block # of This Home Block	Block # of Other Home Block
HOMP4S=6	# of Pages for Swapping on This Structure	
HOMFST=7	First Swapping Track on Unit	
HOMRXB=10	Address of Index Block of ROOT-DIRECTORY	
HOMBXB=11	Address of Index Block of BACKUP COPY OF ROOT DIRECTORY	
HOMFLG=12	Flags	
HOMSIZ=13	Number of Sectors in This Unit	
HOMBTB=14	Number of Tracks in Structure	
HOMMID=15	Pack Unique Code	
	Reserved for Expansion	


HOMFE0=61
Front End File System (sector #)
HOMFE1=62
Front End File System (# of sectors)
Reserved for the Front End
HOMFE2=101
BOOTSTRAP.BIN Word One (Sector #)
HOMFE3=102
BOOTSTRAP.BIN Word Two (# of Sectors)
Reserved for Expansion
HOMUID=165
12 Character Unit I.D. (PDP-11 Format) (3 words)
HOMOID=170
12 Character Owner I.D. (PDP-11 Format) (3 words)
HOMFSN=173
12 Character File System Name (PDP-11 Format) (3 words)
HOMCOD=176
0
707070
HOMSLF=177
0
This Block #

Name: HOME

Description: Home Table. This table contains the disk pages for the HOME and BAT blocks and the 11 Bootstrap program.

Defined in: STG

Referenced by: DSKALC

Format

HOME

Ø (11 Bootstrap)
1 (Home Block)
2 (BAT Block)
3 Reserved
4 .
5 .
6 .
7 .
1Ø .
11 .
12 (Secondary Home Block)
13 (Secondary Home Block)

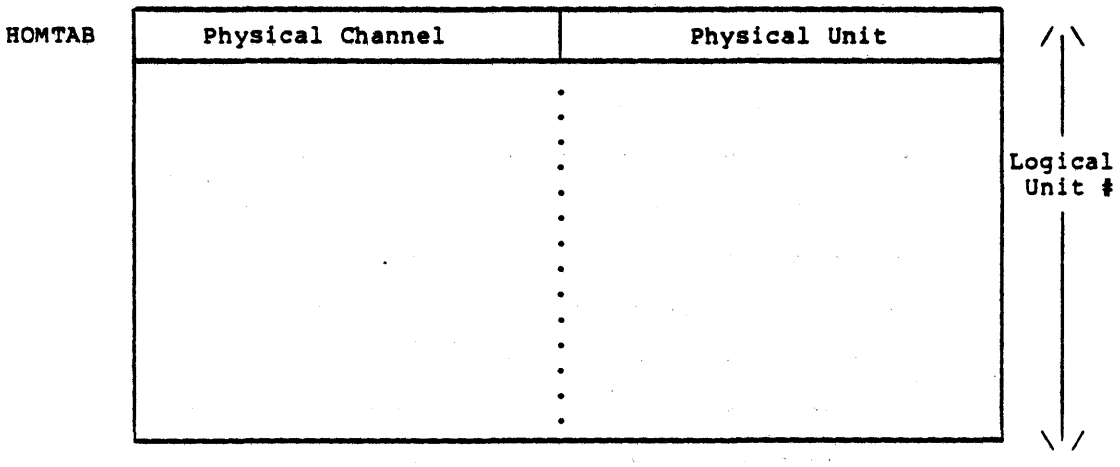
Name: HOMTAB

Description: This table contains the logical to physical mapping (channel and unit) per logical unit, and its length equals the maximum number of packs in a structure.

Defined in: STG

Referenced by: DSKALC

Format



Name: IDXFIL

Defined in: PROLOG

Referenced in: FILINI, DIRECT

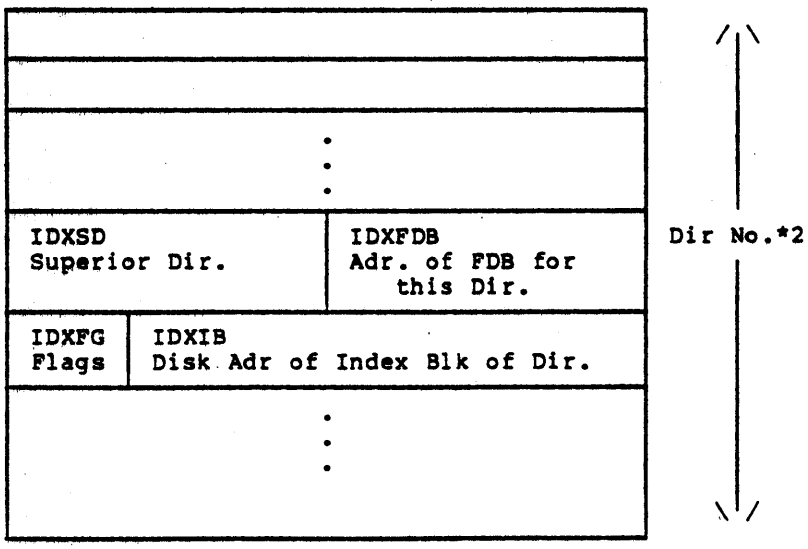
Description: The Index Table File of the structure currently mapped for a process. Each structure has an index table file. The file is indexed by 2* directory number as each entry is two words long. For each directory on the structure, an entry will contain the address of the FDB for the directory and the disk address of the index block for the directory.

The table, IDXFIL, in the PSB, is mapped from the index block of the file, STRNAM:<ROOT-DIRECTORY>INDEX-TABLE.BIN. The file pointed to by the IDXFIL map has the format shown below.

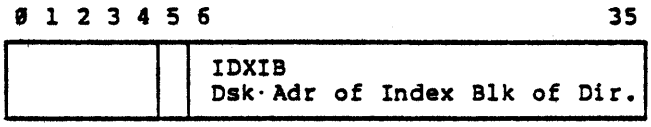
When a structure is mounted (physically) the system gets on OFN for this file and stores it in entry, STRIDX, in the SDB table for that structure. The table entries are created at this time (mount-time).

Format

IDXFIL:



Word 2 of Pair



Symbol	Bits	Pointer	Meaning
IDX%IV	5		If set, indicates that this IDX entry is invalid. (IDX%IV is set equal to 1, but is positioned at bit 5)
	6-35	IDXIB	Disk address of index block of directory.

Name: INDEX

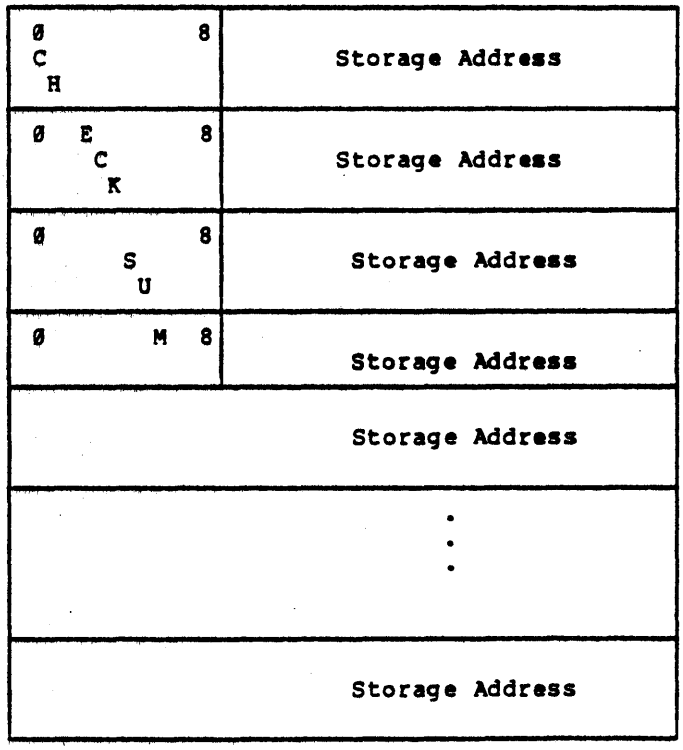
Description: The Index Block (1 page) exists for each disk file and contains pointers to where each of the file's pages resides on disk. If more than one index block is needed for non-directory files, a super index block (1 page) is created which points to the home disk address of each index block. (Note that the maximum file size is 512*512 pages.)

When the file is referenced, an in-core copy of the index block is maintained which keeps track of the file's active pages in the system. (i.e. Whether the pages are in-core, on the swapping area, or on disk.)

Defined in:

Referenced by: PAGEM, PHYSIO

Format



Name: INIDEV

Description: Initialize Devices. This table contains calls to initialize devices after loading the swappable monitor.

Defined in: STG

Referenced by: FILINI

Format

INIDEV	CALL MTA
	CALL LPT
	RET

Name: INIDV1

Description: Device initiation for front end devices.

Defined in: STG

Referenced by: DTESRV

Format

INIDV1	CALL FEINI
	CALL CDRINI
	CALL LPTINI
	RET

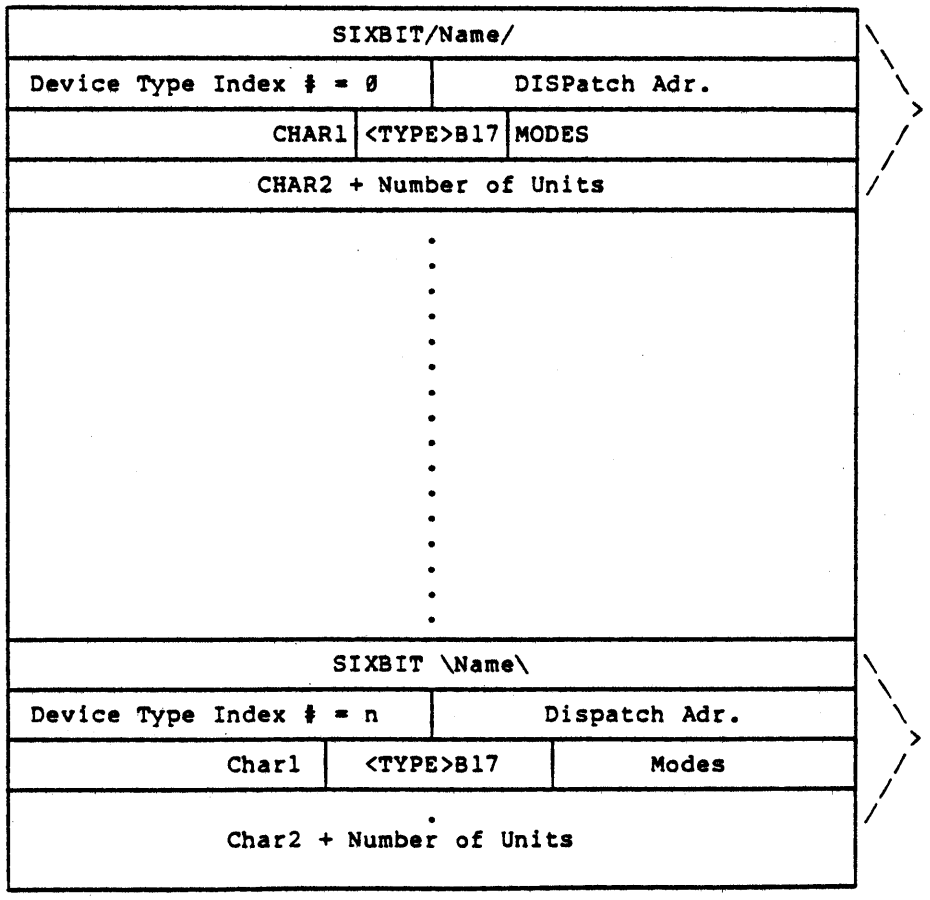
Name: INIDVT

Description: Device Initialization Table. This static table generated at assembly time, contains a 4 word block for each type of device on the system. It is used at system startup time to generate unit # of entries per device type in the device tables, DEVCHR, DEVCH1, DEVNAM, & DEVUNT. Thus, each unit of each device type has an entry in the device tables.

Defined in: STG, MONSYM

Referenced by: DEVICE

Format



CHAR1 can be a combination of the following:

Symbol	Bit	Meaning
DV%%OUT	Ø	Can do output
DV%%IN	1	Can do input
DV%%DIR	2	Has a directory
DV%%AS	3	Is assignable
DV%%MDD	4	Is a multiple directory device
DV%%AV *	5	Is available to this job
DV%%ASN *	6	Is assigned by ASND
DV%%MDV	7	Is a mountable device
DV%%MNT *	8	Is mounted

TYPE is one of the following:

Symbol	Value	Meaning
.DVDSK	Ø	Disk
.DVMTA	2	Magtape
.DVPTP	5	Spooled PTP
.DVLPT	7	Spooled & physical line printer
.DVCDR	1Ø	Spooled & physical card reader
.DVFE	11	Front End Device
.DVTTY	12	Terminal
.DVPTY	13	Pseudo TTY
.DVNUL	15	Null Device
.DVPLT	17	Spooled Plotter
.DVCDP	21	Spooled Card Punch

MODES can be a combination of the following:

Symbol	Bit	Meaning
DV%%M Ø	35	Can be opened in mode Ø
DV%%M1	34	" 1
DV%%M2	33	" 2
DV%%M3	32	" 3
DV%%M4	31	" 4
DV%%M5	3Ø	" 5
DV%%M6	29	" 6
DV%%M7	28	" 7
DV%%M1Ø	27	" 1Ø
DV%%M11	26	" 11
DV%%M12	25	" 12
DV%%M13	24	" 13
DV%%M14	23	" 14
DV%%M15	22	" 15
DV%%M16	21	" 16
DV%%M17	2Ø	" 17

CHAR2 can be a combination of the following:

Symbol	Bit	Meaning
D1%%SPL	Ø	Is spooled
D1%%ALC *	1	Is under control of allocator
D1%%VVL *	2	Volume valid
D1%%NIU *	3	Device slot not in use
D1%%INI *	4	Device is being initialized (currently for structures only)

* These bits are zero at assembly time and are set by the monitor when appropriate in their corres%-ponding device tables. (i.e. DEVCHR or DEVCH1)

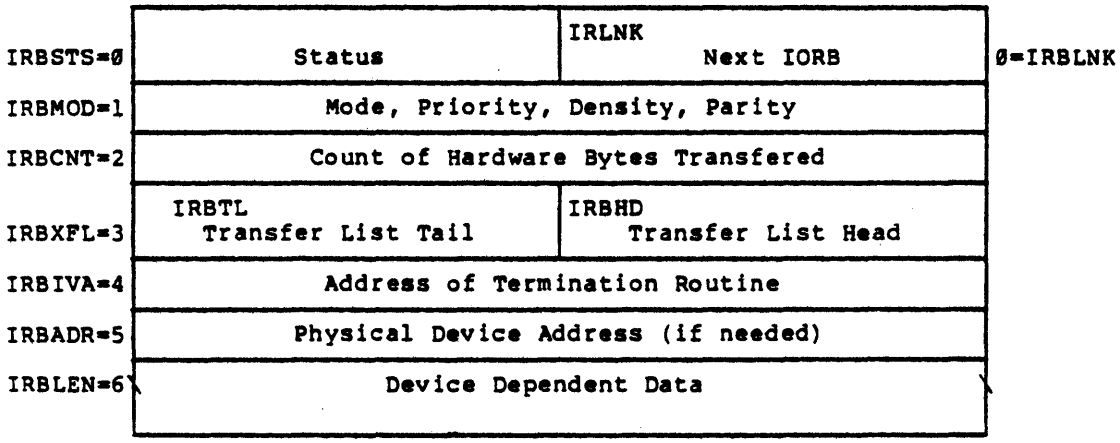
Name: IORB

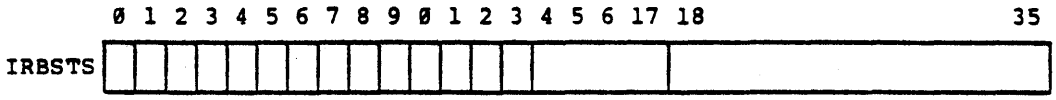
Description: I/O Request Block. Whenever a request for massbus I/O (i.e. DSK or MTA) occurs, an IORB is built for that request. It is of the long form described below for magtape requests and special disk I/O. However, the most common IORB format for disk I/O is a one word IORB, consisting of just the status word, IRBSTS, and stored in the CST5 table.

Defined: PHYPAR

Referenced by: PHYSIO, STG

Format

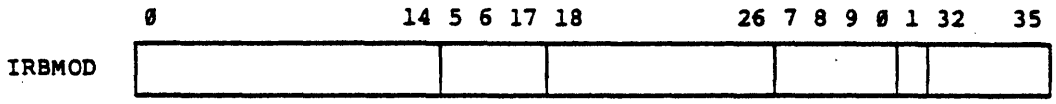




Symbol	Bits	Pointer	Content
IS.SHT	0		Short form (PAGEM) request
IS.DON	1		Done with this job
IS.ERR	2	ISERR	Error on this operation
IS.NRT	3		No more retries
IS.WGU	4		Wrong unit interrupted
IS.TPM	5	ISTPM	Hit tape mark
IS.EOT	6		On write only, hit physical EOT
IS.WLK	7		Write locked
IS.IER	8		Inhibit error recovery
IS.DER	9		Data error
IS.HER	10		Hardware error on device
IS.BOT	11		Hit BOT
IS.RTL	12		Record too long (buffer too small)
IS.IEL	13		Inhibit error logging
	14-17	IFSCN	Function code
	18-35	IRLNK	When referring to link

Function Codes for ISFCN

Symbol	Code	Function
IRFRED	1	Read data
IRFRDF	2	Read data and format (count, key, header)
IRFWRT	3	Write Data
IRFWTF	4	Write format
IRFSEK	5	Seek
IRFFSB	6	Forward space block
IRFBSB	7	Backspace block
IRFWTM	10	Write tape mark
IRFERG	11	Erase gap
IRFREW	12	Rewind
IRFRUN	13	Rewind and unload
IRFRDR	14	Read reverse
IRFRCR	15	Recovery read

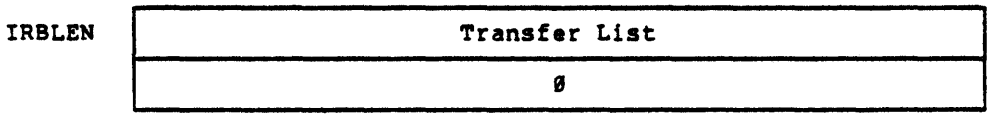


Bits	Pointer	Content
15-17	IRBDM	Data Mode
27-3Ø	IRBPRI	Priority
31	IRBPAR	Parity
32-35	IRBDN	Density

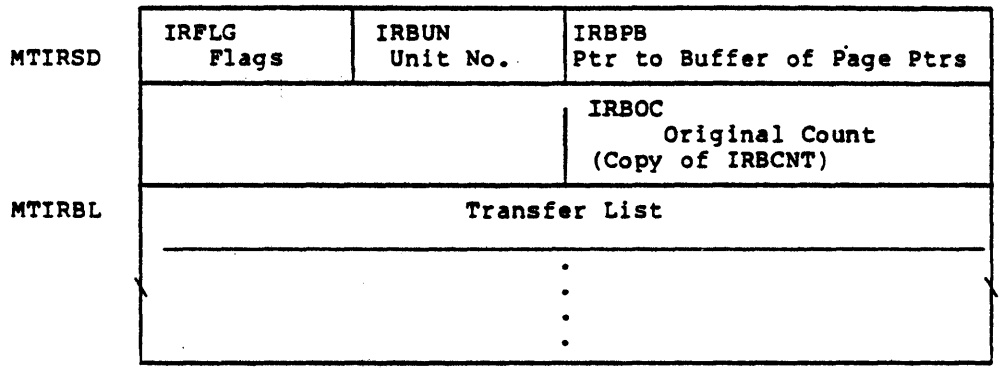
Data Modes for IRBDM

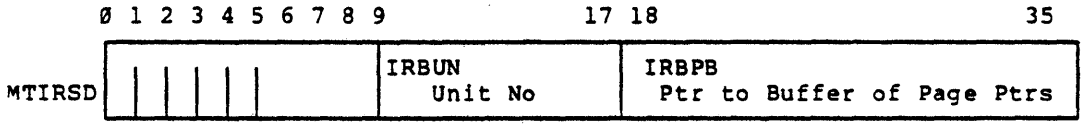
Symbol	Code	Meaning
IRMWRD	1	Word mode
IRM6BT	2	Six bit
IRM7BT	3	Seven bit
IRM8BT	4	Eight bit

If device is DSK, IRBLEN becomes:



If device is MTA, IRBLEN=MTIRSD becomes:





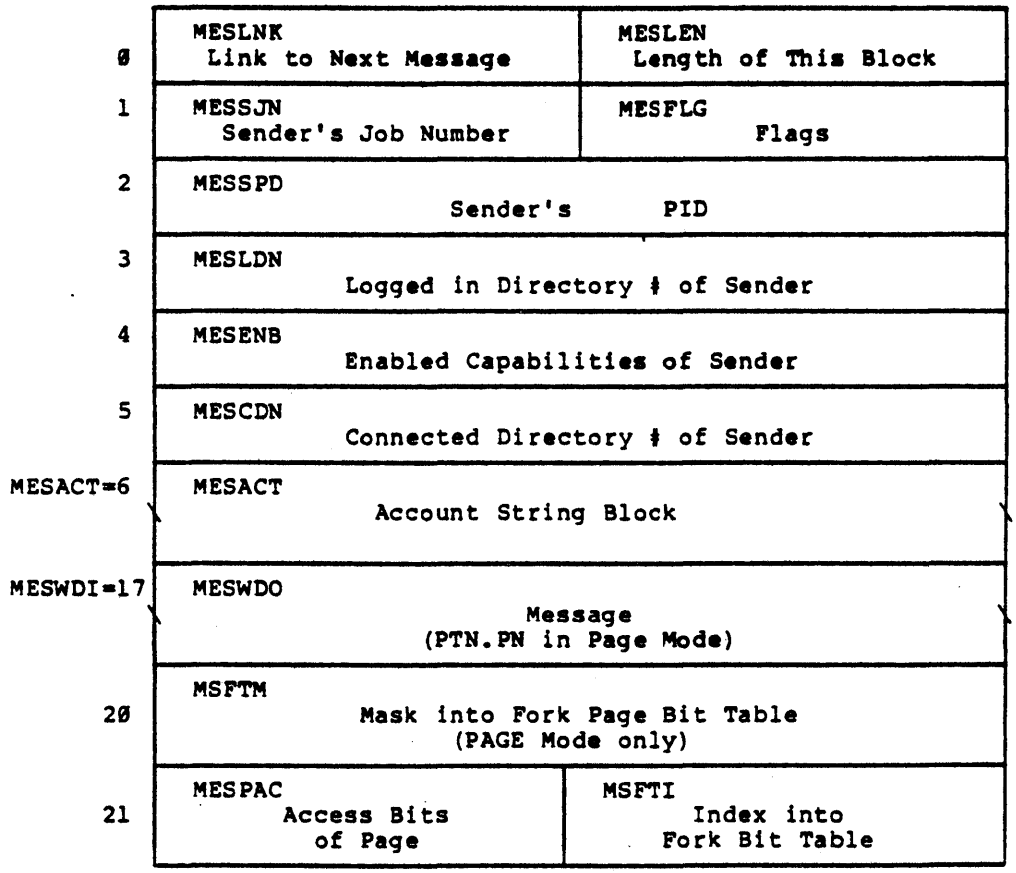
Bits	Pointer	Content
0	IRBFR	Buffer ready for use
1	IRBFQ	Current buffer flag
2	IRBFA	Active flag, IORB being filled or emptied by service routine
3	IRBAB	IORB aborted due to an error
4	IRBFF	IORB free
9-17	IRBUN	Unit number
18-35	IRBPB	Pointer to buffer of page pointers

Name: IPCF-Message-Header

Description: IPCF Message header. This table describes the format of the message header for message sent by the Inter-Process Communications Facility.

Defined in: IPCF

Referenced by: IPCF

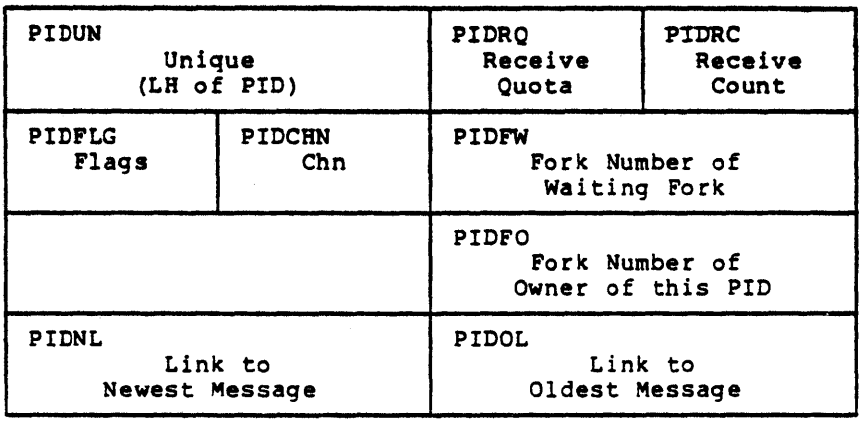


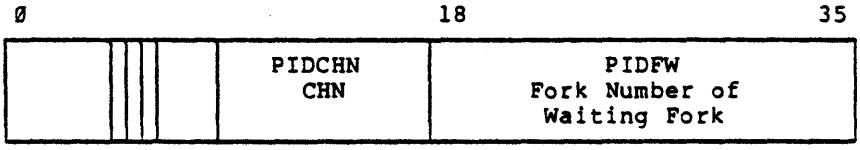
Name: IPCF-PID-HEADER

Description: Overhead information for each PID in use.

Defined in: IPCF

Referenced by: IPCF





Symbol	Bits	Pointer	Meaning
PD%JWP	8		PID is a Job-wide PID
PD%DIS	9		PID is disabled
PD%CHN	1Ø		A channel is set up to get interrupts
PD%NOA	11		No access by other forks
	12-17	PIDCHN	Channel # to interrupt waiting fork on.
	18-35	PIDFW	Fork waiting for message to this PID

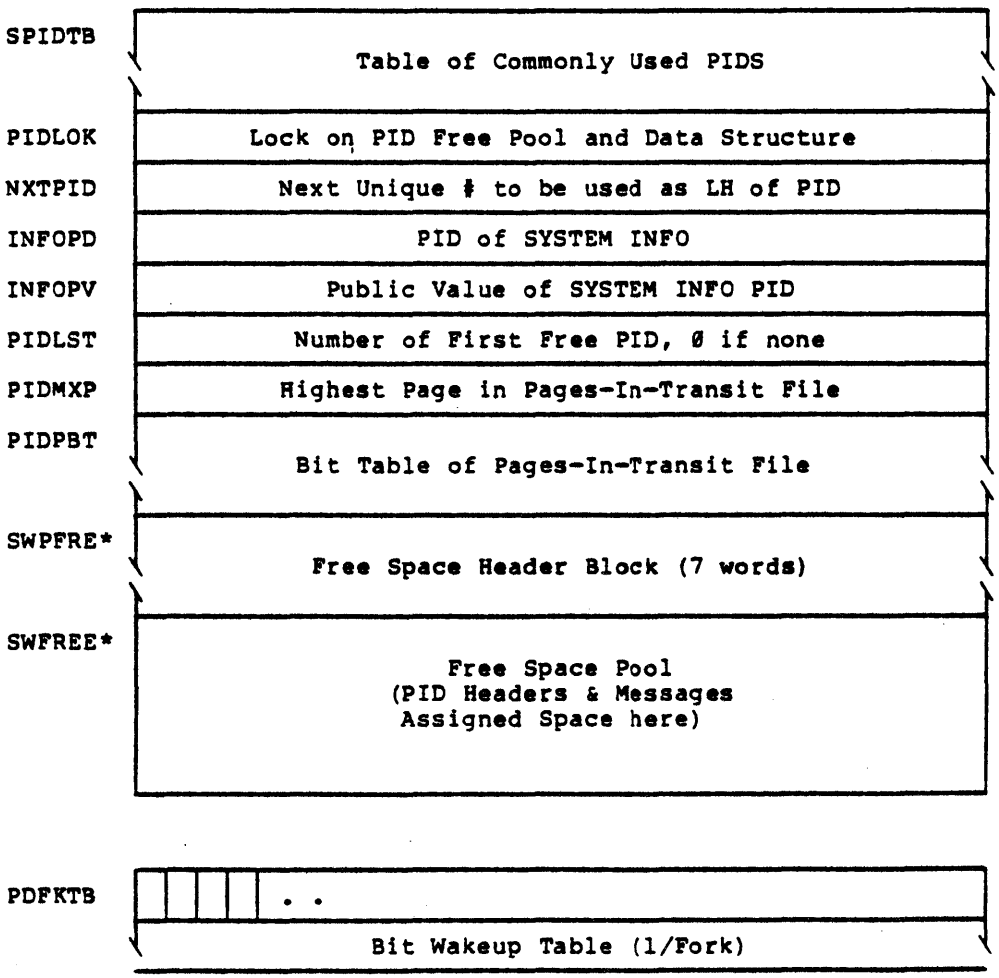
Name: IPCF-STORAGE-AREA

Description: Inter-Process Communication Facility Storage Area. This non-resident storage is described followed by the resident wake-up bit table (PDFKTB). See also the tables, PIDCNT and PIDTBL.

Defined in: STG

Referenced by: GTJFN, IPCF, LOGNAM, MEXEC

FORMAT



* See Swap-Free-Space-Pool Table.

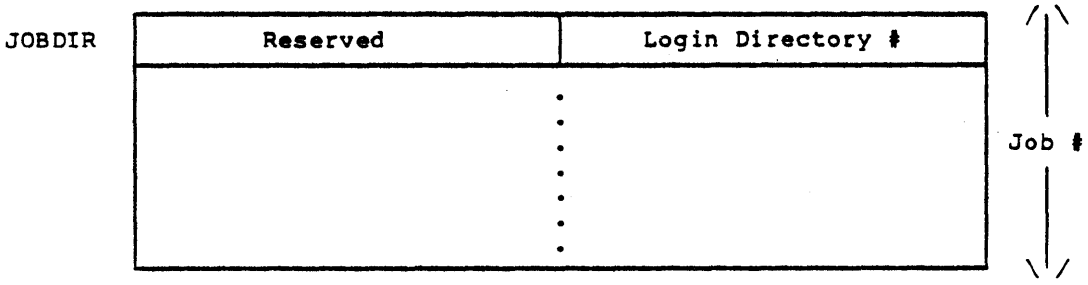
Name: JOBDIR

Description: Job Directory Table. This table, indexed by job #, contains the number of the login directory for each job.

Defined In: STG

Referenced by: APRSRV, TTYSRV, DIRECT, DISC, DTESRV, FILINI, IPCF, JSYSA, MAGTAP, MEXEC

Format



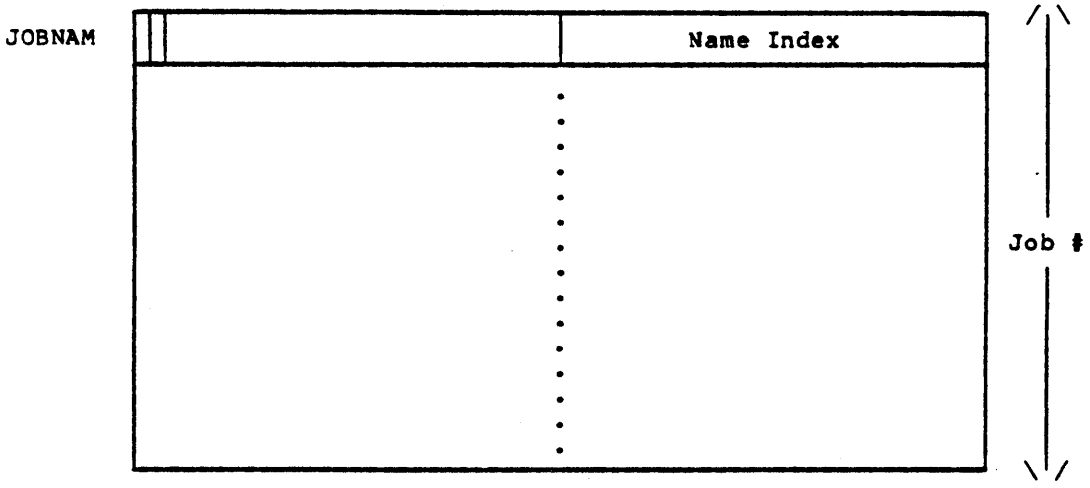
Name: JOBNAM

Description: Job Name Table. This table, indexed by job #, contains an index into the subsystem name tables (SNAMES, STIMES, etc.) indicating what subsystem, if any, each job is running. The name index is for statistics only and is not used by the monitor.

Defined In: STG

Referenced by: FORK, MEXEC, PAGEM, PHYSIO, SCHED

Format



Symbol	Bit	Pointer	Content
JWAKEF	0	HIBFL	Flag used by HIBER JSYS. If set, implies a wakeup signal to THIBR
	1	DIAFL	Job has DIAG resources
	18-35		Name index

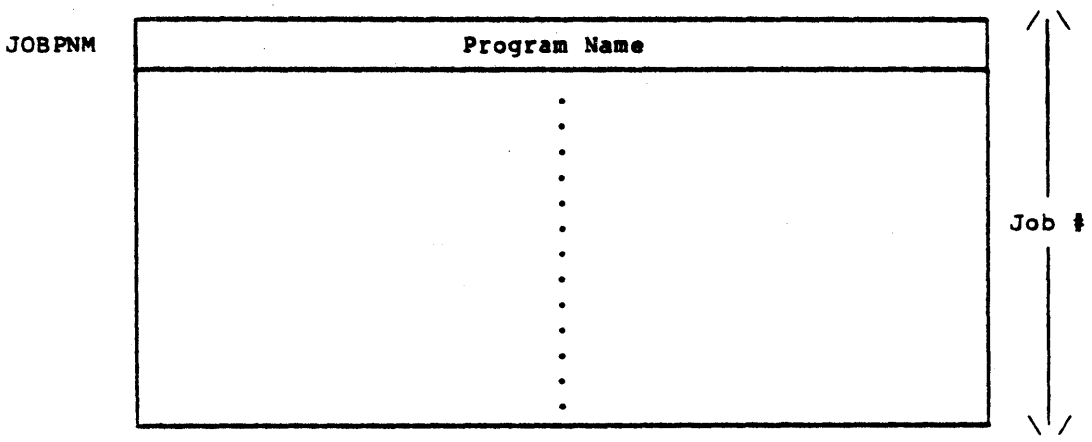
Name: JOBPNM

Description: Job Program Name. This table, indexed by job #, contains each job's program name.

Defined In: STG

Referenced by: DTESRV, IPCF, MAGTAP, MEXEC

Format



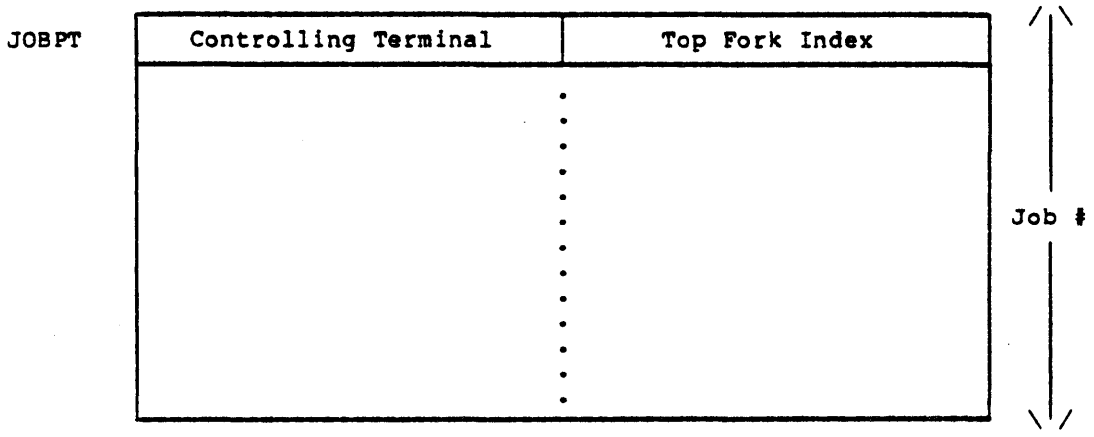
Name: JOBPT

Description: Job Process Table. This table, indexed by job #, contains the number of the controlling terminal, or -1 for a detached job, and the index of the top fork of the job.

Defined in: STG

Referenced by: TTYSRV, DEVICE, FILMSC, FORK, IO, IPCF, JSYSA, MEXEC, SCHED

Format



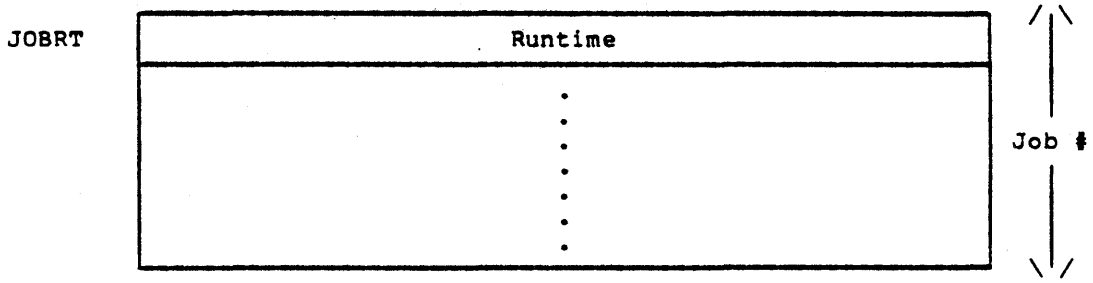
Name: JOBRT

Description: Job Runtime Table. This table, indexed by job #, contains the total runtime of each job (sum of all forks) in milliseconds. If a word contains a -1, the job does not exist.

Defined in: STG

Referenced by: ENQ, FORK, IPCF, JSYSA, MEXEC, SCHED

Format



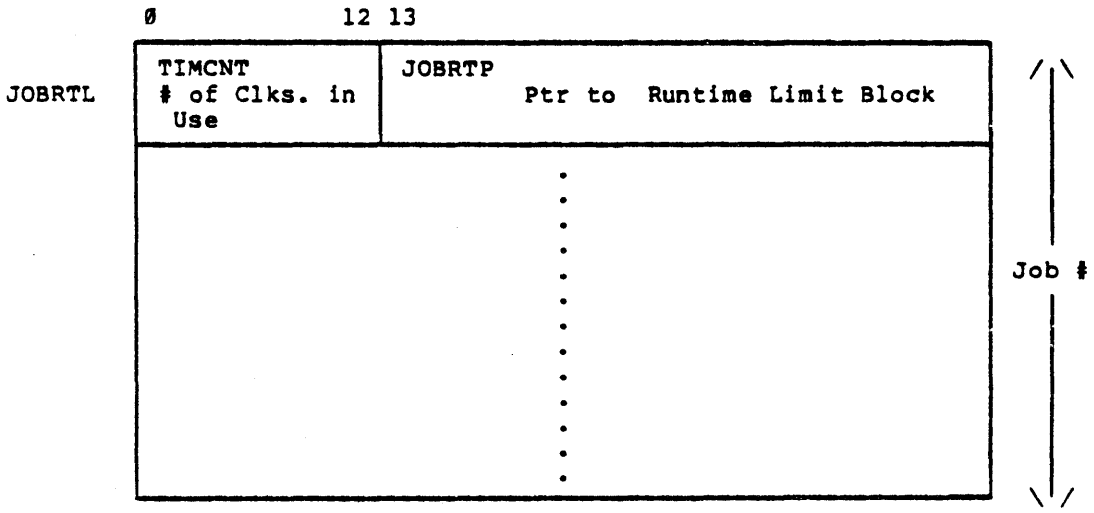
Name: JOBRTL

Description: Job Runtime Limit. This table, indexed by job #, contains the number of clocks (via TIMER JSYS) in use by each job and a pointer to the runtime limit TIMER block. A description of the TIMER block is described below. (See RES-FREE-SPACE).

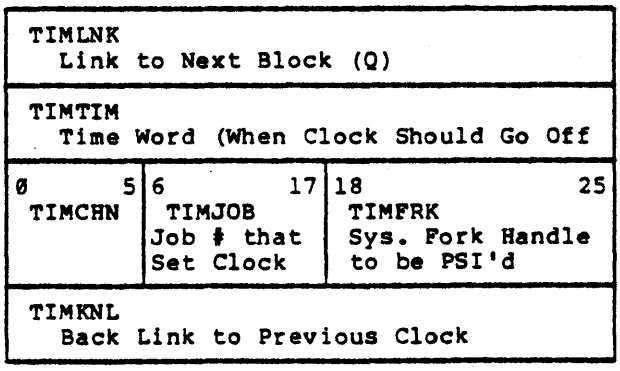
Defined in: STG

Referenced by: MEXEC, SCHED, TIMER

Format



TIMER Run Timer Limit Block



Name: JSB

Description: Job Storage Block. Each job has a Job Storage Block which holds per-job information such as the job's fork structure, line number of controlling TTY, terminal interrupts enabled and accounting and logical name information.

The JOBMAP map in the JSB points to all of the per-job storage (including the JSB page itself). When the monitor references this current job's storage area it uses virtual addresses 620000-706777. (The monitor's mapped slots in MMAP for virtual pages 620-706 point to the JOBMAP map via indirect pointers.)

JBCOR contains a bit table which keeps track of which pages in the Job Storage Area are in use (bit(s) = 0) and which are free (bit(s) = 1). The first several pages of this Job Common Area will always be allocated for the JSB page plus expansion pages for the JFN blocks and for the JSYS trap header word and trap blocks (See FKJTQ table). The first non-reserved page begins at FREJPA (=626000).

JSBFRE is the free block header. If a block of words (i.e., <512 words) is required, it is allocated from the JSFREE area in the JSB. Blocks in the JSFREE area are linked and when a block of words is required, the free list is search looking for a large enough block. If the free list area in JSFREE is depleted, a new page (space outside the JSB in the Job Storage Area) is allocated and its space added to the free list for block usage.

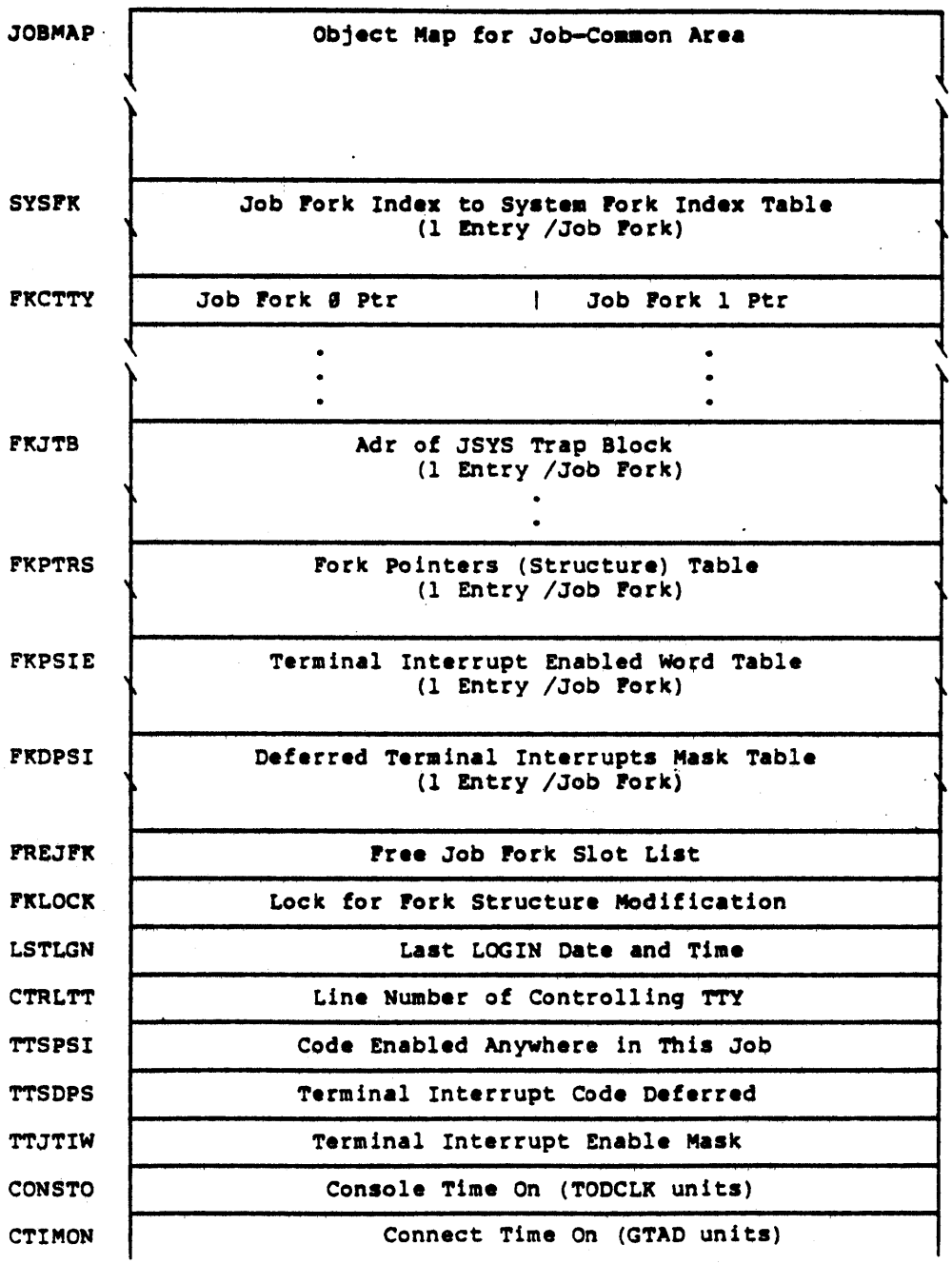
Pages are assigned from the bit table, JBCOR, by the routine, ASGPAG, and are used for temporary job pages such as file window pages, magtape buffer pages, mapping a super index block, getting more space for the free block storage linked in JSBFRE, and mapping EXE file directory pages.

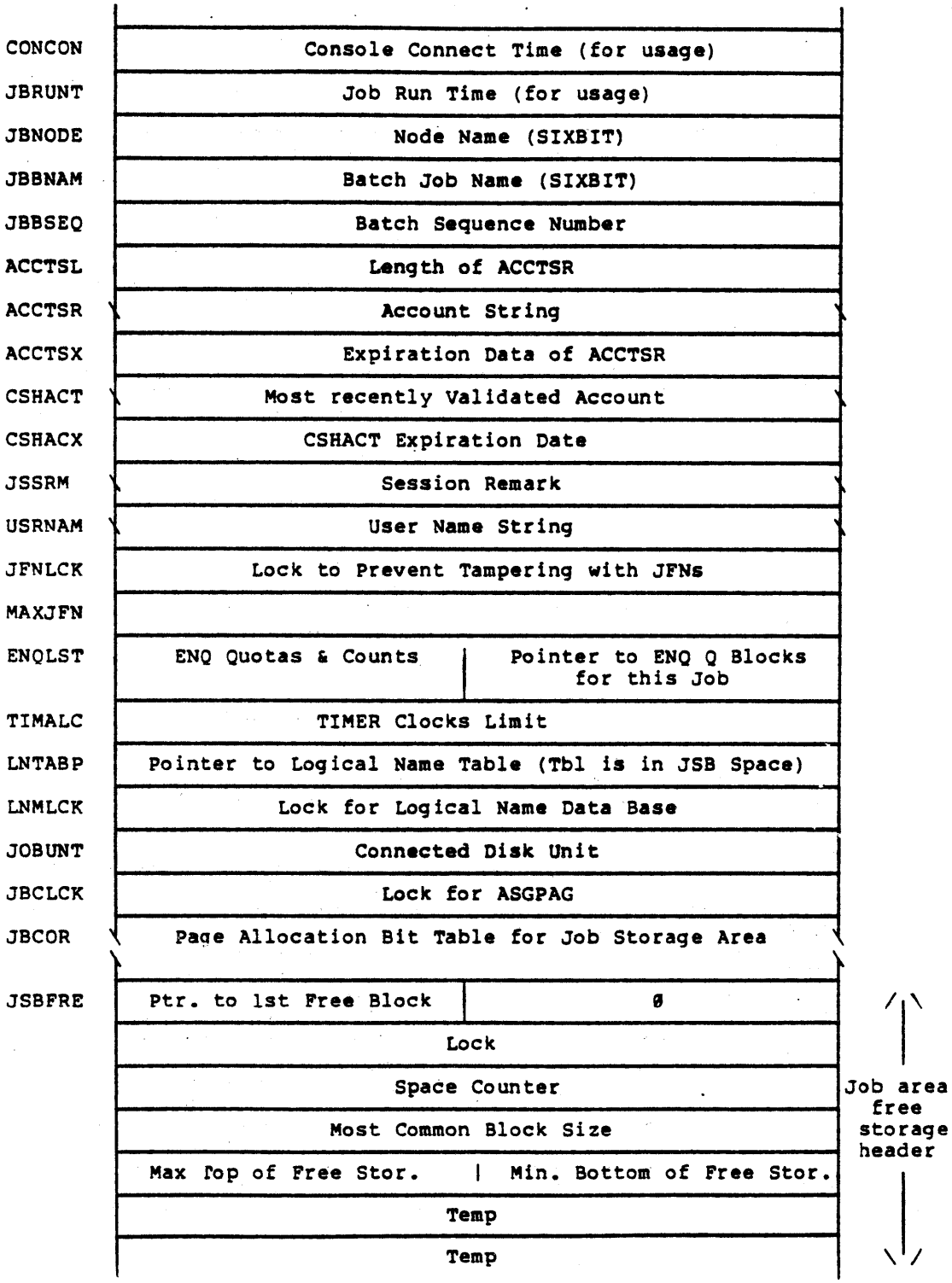
Blocks of words are assigned from the free list, headed by JSBFRE by the routine, ASGFRE, and are used to hold temporary storage such as name strings for JFN blocks, the job-wide Logical Names List, and the Logical Name Definition Blocks.

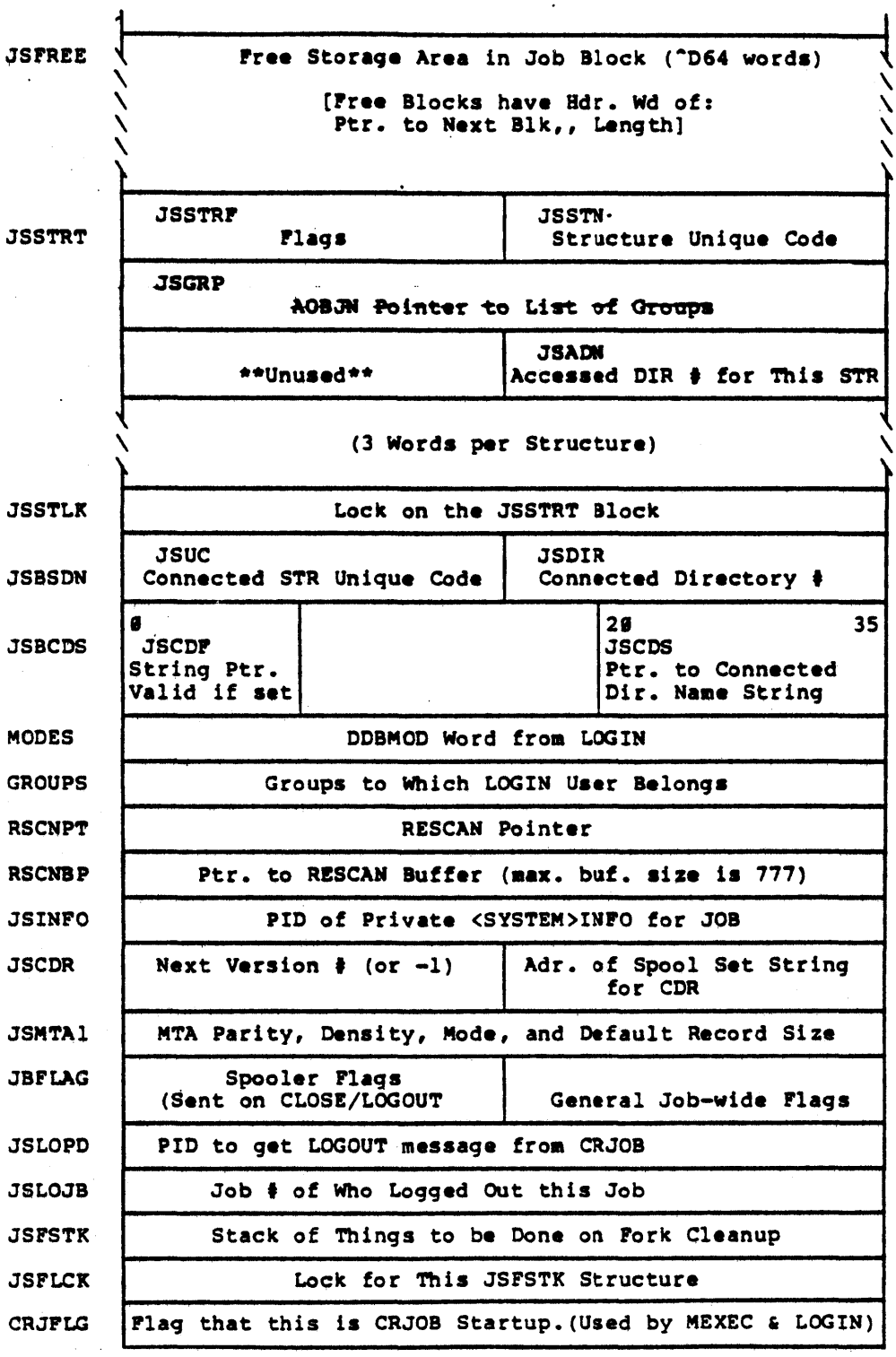
Defined in: STG

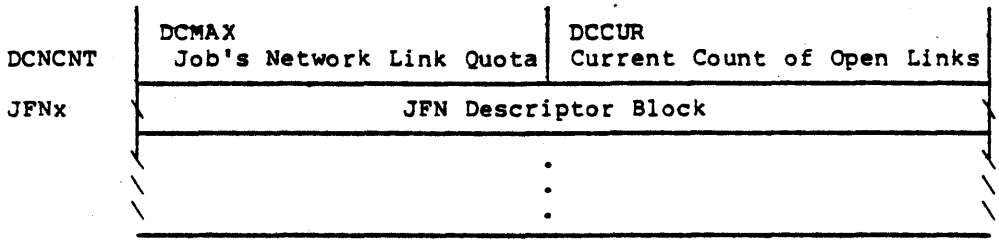
Referenced by: PAGEM, SCHED, FORK, POSTLD, PROLOG, DSKALC, DISC, JSYSA, JSYSF, FILINI, FREE, GTJFN, IO, IPCF, LOGNAM, MAGTAP, MEXEC, MSTR, NETWRK, NSPSRV, TAPE

Format


JOBMAP
Object Map for Job-Common Area
SYSFK
Job Fork Index to System Fork Index Table
(1 Entry /Job Fork)
FKCTTY
Job Fork 0 Ptr
Job Fork 1 Ptr
FKJTB
Adr of JSYS Trap Block
(1 Entry /Job Fork)
FKPTRS
Fork Pointers (Structure) Table
(1 Entry /Job Fork)
FKPSIE
Terminal Interrupt Enabled Word Table
(1 Entry /Job Fork)
FKDPSI
Deferred Terminal Interrupts Mask Table
(1 Entry /Job Fork)
FREJFK
Free Job Fork Slot List
FKLOCK
Lock for Fork Structure Modification
LSTLGN
Last LOGIN Date and Time
CTRLTT
Line Number of Controlling TTY
TTSPSI
Code Enabled Anywhere in This Job
TTSDPS
Terminal Interrupt Code Deferred
TTJTIW
Terminal Interrupt Enable Mask
CONSTO
Console Time On (TODCLK units)
CTIMON
Connect Time On (GTAD units)


CONCON
Console Connect Time (for usage)
JBRUNT
Job Run Time (for usage)
JBNODE
Node Name (SIXBIT)
JBBNAM
Batch Job Name (SIXBIT)
JBBSEQ
Batch Sequence Number
ACCTSL
Length of ACCTSR
ACCTSR
Account String
ACCTSX
Expiration Data of ACCTSR
CSHACT
Most recently Validated Account
CSHACX
CSHACT Expiration Date
JSSRM
Session Remark
USRNAM
User Name String
JFNLCK
Lock to Prevent Tampering with JFNs
MAXJFN
ENQLST
ENQ Quotas & Counts
Pointer to ENQ Q Blocks for this Job
TIMALC
TIMER Clocks Limit
LNTABP
Pointer to Logical Name Table (Tbl is in JSB Space)
LNMLCK
Lock for Logical Name Data Base
JOBUNT
Connected Disk Unit
JBCLCK
Lock for ASGPAG
JBCOR
Page Allocation Bit Table for Job Storage Area
JSBFRE
Ptr. to 1st Free Block
Ø
Lock
Space Counter
Most Common Block Size
Max Top of Free Stor.
Min. Bottom of Free Stor.
Temp
Temp
Job area free storage header


JSFREE
Free Storage Area in Job Block (^D64 words)
[Free Blocks have Hdr. Wd of:
Ptr. to Next Blk,, Length]
JSSTRT
JSSTRF
Flags
JSSTN·
Structure Unique Code
JSGRP
AOBJN Pointer to List of Groups
Unused
JSADN
Accessed DIR # for This STR
(3 Words per Structure)
JSSTLK
Lock on the JSSTRT Block
JSBSDN
JSUC
Connected STR Unique Code
JSDIR
Connected Directory #
JSBCDS
0
JSCDF
String Ptr.
Valid if set
20
35
JSCDS
Ptr. to Connected
Dir. Name String
MODES
DDBMOD Word from LOGIN
GROUPS
Groups to Which LOGIN User Belongs
RSCNPT
RESCAN Pointer
RSCNBP
Ptr. to RESCAN Buffer (max. buf. size is 777)
JSINFO
PID of Private <SYSTEM>INFO for JOB
JSCDR
Next Version # (or -1)
Adr. of Spool Set String
for CDR
JSMTA1
MTA Parity, Density, Mode, and Default Record Size
JBFLAG
Spooler Flags
(Sent on CLOSE/LOGOUT
General Job-wide Flags
JSLOPD
PID to get LOGOUT message from CRJOB
JSLOJB
Job # of Who Logged Out this Job
JSFSTK
Stack of Things to be Done on Fork Cleanup
JSFLCK
Lock for This JSFSTK Structure
CRJFLG
Flag that this is CRJOB Startup.(Used by MEXEC & LOGIN)



Each JFN uses a block of 19 words. (Since JFNs can grow beyond the end of the JSB into successive pages, the JFN blocks must be the last storage defined in the JSB.)

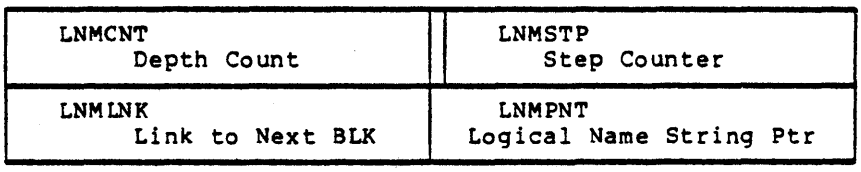
JFN descriptor block format:

FILBYT(1)=0	Byte Pointer to Current Window		
FILBYN(2)=1	Byte # of Current Byte		
FILAC	Ptr to Account String or Account #		
FILLEN	Total File Length in Bytes		
FILCNT	Bytes Remaining in Current Buffer		
FILLCK	File Lock Word		
FILWND(3)	Current Page #	Location of Current Window	
FILSTS	File Status Bits	Status	Mode
FILDEV	STR Structure Number	DEV`DTB (i.e.Dev Disp. Tbl)	
FILOFN(4)	OFN for This File	OFN of Long File PT Table	
FILLFW(5)	Count of Pages Mapped	Loc. of Page Table Table	
FILDDN	Ptr. to Device String Block	Directory #	
FILDNM	FILDIR Directory Name String	FILATL Ptr to Attribute List	
FILNEN	File Name String Blk. Ptr.	Ext. String Blk. Ptr.	
FILVER	Fork # of JFN Originator	Version #	
FILMS1(6)	FILDMS Directory Wild Mask	FILNMS Name Wild Mask	
FILMS2(7)	FILEMS Extension Wild Mask		
FILFDB(8)	Address of FDB in the Directory		
FILCOD	FILUC STR Unique Code	FILPO PTO OFN for Long File	

These definitions are used in the above positions only during the GTJFN procedure:

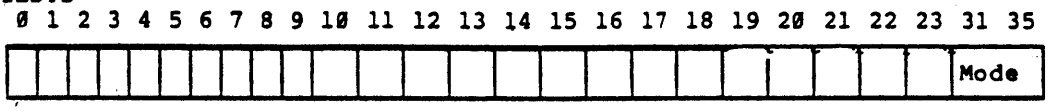
```
(1)  FILTMP / Ptr. to temp string block for default ,, Ptr. to
              temp string block
(2)  FILPRT / Ptr. to protection string or protection #
(3)  FILSKT / Arpanet connection no.,, Unused
            FILOPT / Byte Ptr. to Store String in GTJFN
       (4)  FILLIB / For DECNET, Ptr to LL Block
       (5)  FILLNM / Ptr. to RDTEXT buffer ,, Ptr. to logical name
                     chain **
       (6)  FILBFO / For DECNET, Output Buffer Ptr.
       (7)  FILIDX / Ø ,, Index into device tables for original
                     devices GTJFNed
                     {(i.e., doesn't change during spooling)}
            FILBFI / For DECNET, Ptr. to Input Buffer
       (8)  FILBCT / For DECNET, Ptr. to Counts
```

** Logical Name Header Format



Bits	Pointer	Content
Ø-17	LNMCNT	Depth count for logical names
18	LMMIDX	Index into logical name tables
19-35	LNMSTP	Step counter at time of chaining
Ø-17	LNMLNK	Link to next chain block
18-35	LNMPNT	Pointer to logical name string

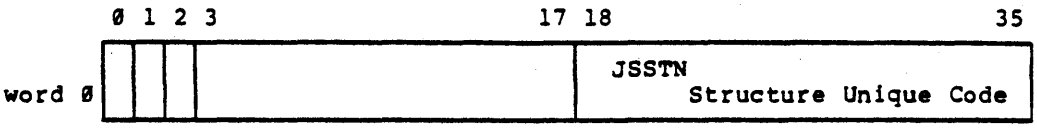
FILSTS



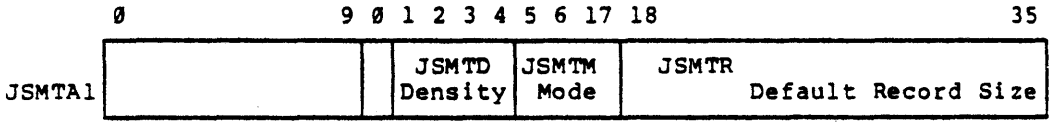
FILSTS

Symbol	Bits	Pointer	Meaning
OPNF	0		File is open
READF	1		File is OK to read
WRTF	2		File is OK to write
XCTF	3		File is OK to execute
RNDF	4		File is OK to reset ptr. (i.e., not append)
NONXF	5		Non-existent File, delete FDB on RLJFN
NWTF	6		No wait on DUMP I/O
LONGF	7		File is a long file
EOFF	8		End of file if read attempted
ERRF	9		Bytes read may be wrong
NAMEF	10		Name is associated with this JFN
ASTF	11		An * was typed in
ASGF/BLKF	12		JFN is being assigned or service routine wants to block; shares a bit position with ASGF
HLTF	13		Halt if I/O error
WNDF	14		A window page has been set up
ENDF/TRNSL	15		File is past end of maximum length. Bit timeshared with ENDF to say the JFN is in a transitional state and may not be accessed.
SIZE	16		Illegal to change size of byte
FRKF	17		File is restricted to fork in LH(FILVER)
PASLSN	18		Set to skip line number checking on ASCII files
SKIPBY	19		Set by BYTINA to remember that it has to discard.
XQTAF	20		Quota exceeded Flag
FILDUD	21		Suppress DDMP action if set
FILINP	22		Direction of I/O is input
FILOUP	23		Direction of I/O is output
	31-35	IOMODE	Mode File is opened in

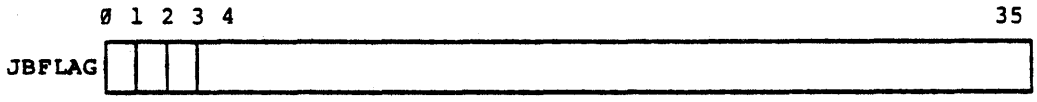
Structure 3-word Block (starts at JSSTRT)



Bit	Pointer	Content
0	JSSDM	Structure is dismounted
1	JSMCI	Mount count has been incremented by structure
2	JSXCL	Structure is mounted exclusively by the structure
18-35	JSSTN	Structure unique code



Bits	Pointer	Content
10	JSMTP	Parity
11-14	JSMTD	Density
15-17	JSMTM	Mode
18-35	JSMTR	Default record size (hardware bytes)



Symbol	Bits	Pointer	Content
SP%BAT	0		Job is being controlled by BATCH
SP%DFS	1		Spooling is deferred
SP%ELO	2		Job executed LOGOUT JSYS
SP%FLG	3		Job forced to LOGOUT by top Fork error
SP%OLO	4		Job logged out by other job
	18	JBMAX	Job has been in the mini-exec

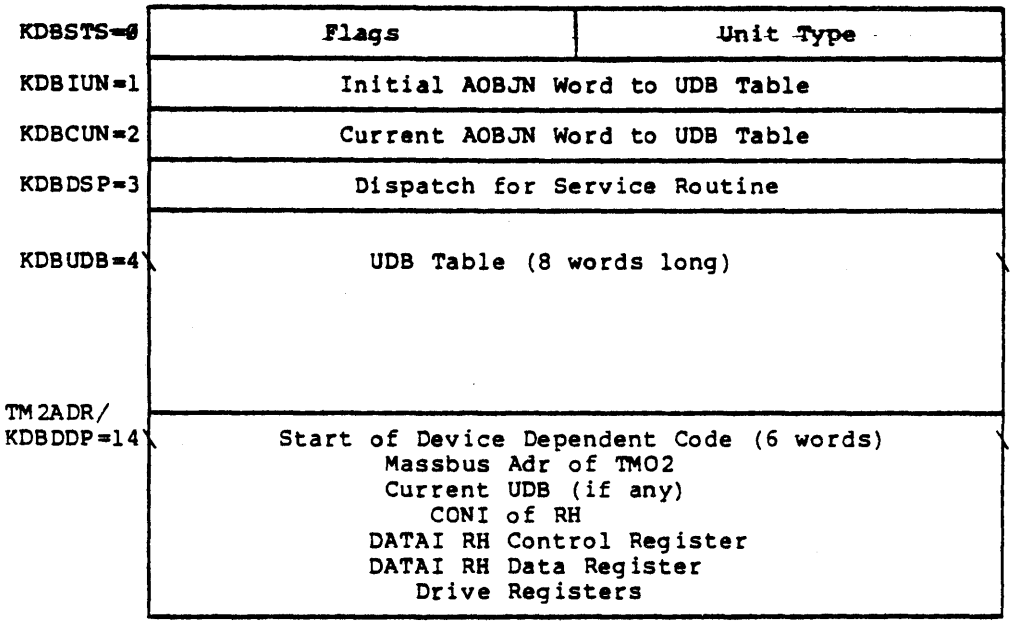
Name: KDB

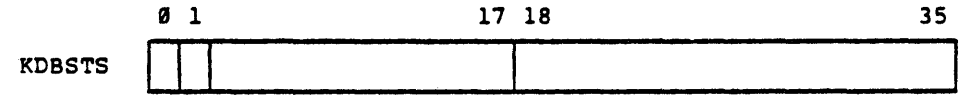
Description: Kontroller Data Block (TMØ2 only)

Defined in: PHYPAR

Referenced by: PHYM2

Format





Symbol	Bit	Pointer	Content
KS.ACT	1		Controller Active if set
	18-35		Unit type

Name: LOGICAL-NAME-DEFINITION

Description: Logical Name Definition Block. The block format given below is used for system and job-wide logical name definitions. The first definition block for a logical name is pointed to by its Logical Name List and is store in the swappable free space if a system logical name or in the JSB space if a job-wide logical name.

Defined in: LOGNAM

Reference by: LOGNAM

LNBLK=0	PTR TO NEXT DEFINITION (OR ZERO IF NONE)	SIZE OF THIS BLOCK (USUALLY 12)
LNDEV=1	ASCII BYTE PTR TO DEVICE BLOCK (IF ANY)	
LNDIR=2	ASCII BYTE PTR TO DIRECTORY BLOCK (IF ANY) (-3 MEANS STAR WAS TYPED)	
LNNAM=3	ASCII BYTE POINTER TO NAME BLOCK (IF ANY)	
LNEXT=4	ASCII BYTE POINTER TO FILE TYPE (IF ANY) (-2 MEANS A NULL FIELD WAS SPECIFIED)	
LNVER=5	500000,,0 + GENERATION NUMBER (IF ANY)	
LNACT=6	500000,,0 + ACCOUNT NUMBER -OR- ASCII BYTE POINTER TO ACCOUNT STRING (IF ANY)	
LNPRT=7	500000,,0 + FILE PROTECTION (IF ANY)	
LNTMP=10	0 IF PERMANENT OR -1 IF TEMPORARY (IF ANY)	
LNATR=11	PREFIX VALUE OF CURRENT PREFIX	PTR TO ATTRIBUTE CHAIN

Name: LOGICAL-NAMES-LIST

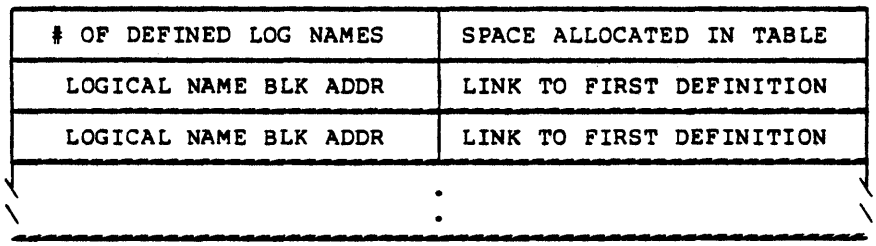
Description: List of Currently Defined Logical Names. The list described below is the format used for the system logical names list (pointed to by SYLNTB) and the job wide logical names list (pointed to by the JSB entry, LNTABP.)

The system logical names list is built in the swappable free space from the entries in SYNMTB at system initialization time. (See SWAP-FREE-SPACE and SYNMTB tables). A job's logical names list is built in the JSB space the first time a logical name is created.

An entry in a logical names list has a pointer to the logical name string (in ASCIZ) in the left half and a link to the first definition block in the right half (See LOGICAL-NAME-DEFINITION description).

Defined in: STG

Referenced by: LOGNAM



Name: LPT-STORAGE-AREA

Description: Storage area for line printers. Each entry in the resident area is LPTN words long, where LPTN equals the number of line printers on the system.

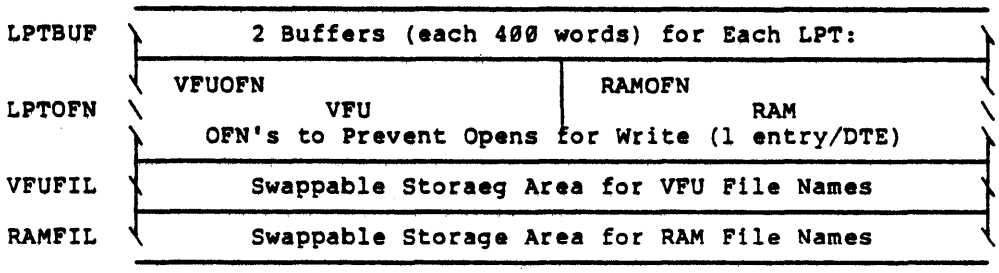
Defined in: STG

Referenced by: LINEPR

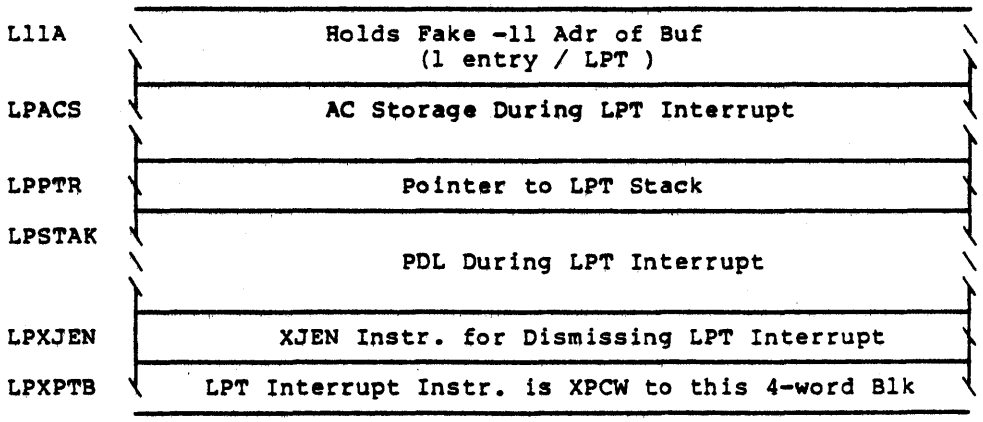
Format

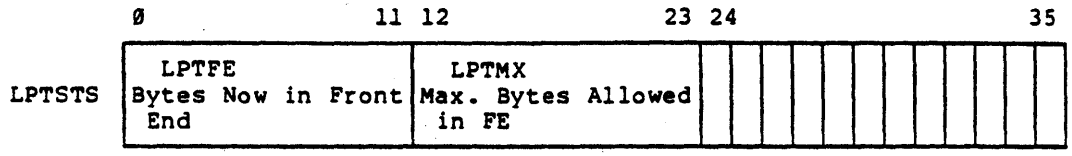
LPTTYP	Type of LPT Vector for Dev. Independence
LPTSTS	Status Word
LPTST1	Second Status Word
LPTST2	Third Status Word
LPTST3	Fourth Status Word
LPTERR	Last Error Word
LPTCNT	Buffer Counter
LPTCLS	LPTCHK Clock Switch
LPTCCW	BLKI/O Pointer
LPTICT	Interrupt Byte Count
LPTCKT	Interval for LPTTIM
LPTLCK	Lock on Opening LPT
PGDATA	Page Counter to be Sent to -11

The following LPT: storage items are in the nonresident area of the monitor.

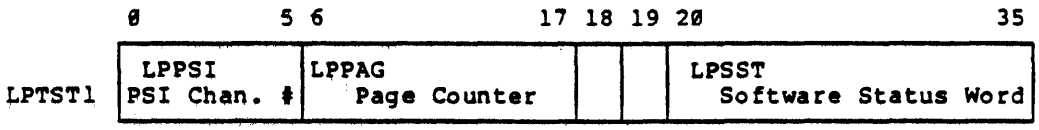


If the assembly flag, SMFLG, is set, indicating a 2020 Monitor, then the following additional storage is assembled in the resident area of the monitor.





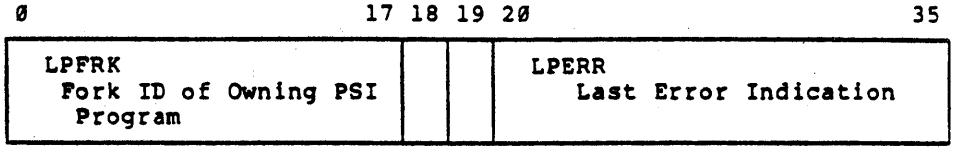
Symbol	Bits	Pointer	Content
	0-11	LPTFE	Bytes now in front end
	12-23	LPTMX	Max. bytes allowed in front end
LP%LHC	24	LPLHC	Loading has completed flag for RAM/VFU load
LP%HE	25	LPTHE	Hard error on this LPT:
LP%OBF	26	LPOBF	Output is being flushed
LP%MWS	27	LPMWS	MTOPR is waiting for a status to arrive
LP%ER	28	LPTER	LPT had an error
LP%OL	29	LPTOL	LPT on-line
LP%TBL	30	LPTBL	LPT is over allocation
LP%TWT	31	LPTWT	Request on Q.
LP%THN	32	LPTHN	DTEQ failed
LP%OPN	33	LPOPN	LPT is opened
LP%ALI	34	ALTI	Interrupt buffer pointer
LP%ALP	35	ALTP	Buffer Pointer



Symbol	Bits	Pointer	Content
	0-5	LPPSI	Channel # on which PSI's are desired.
	6-17	LPPAG	Page Counter
LP%LCP	18	LPLCP	Lower case printer
LP%SHA	19	LPSHA	Status has arrived
	20-35	LPSST	Software status word

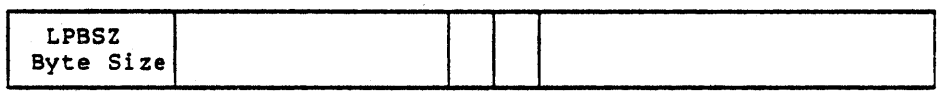
Symbol	Bits	Content
.DVFFE	28	Device has a fatal, unrecoverable error
.DVFLG	29	Error logging information follows
.DVFEF	30	EOF
.DVFIP	31	I/O in progress
.DVFSE	32	Software condition
.DVFHE	33	Hardware error
.DVFOL	34	Offline
.DVFNX	35	Nonexistent device

LPTERR



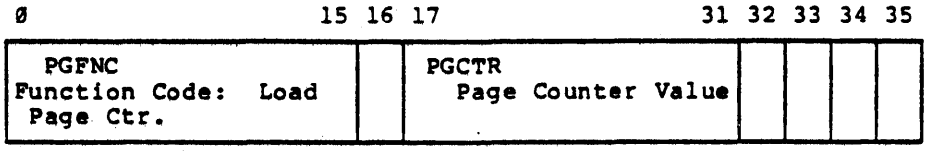
Symbol	Bits	Pointer	Content
	0-17	LPFRK	Fork ID of owning PSI process
LP%MSG	18	LPMSG	If on, suppress standard messages
LP%PCI	19	LPPCI	Page counter has interrupted
	20-35	LPERR	Last error indication

LPTCLS



Symbol	Bits	Pointer	Content
	0-5	LPBSZ	Byte size of OPENF
LP%RLD	16	LPRLD	Front end was reloaded
LP%NOE	17	LPNOE	Note occurrence of EOF

PGDATA



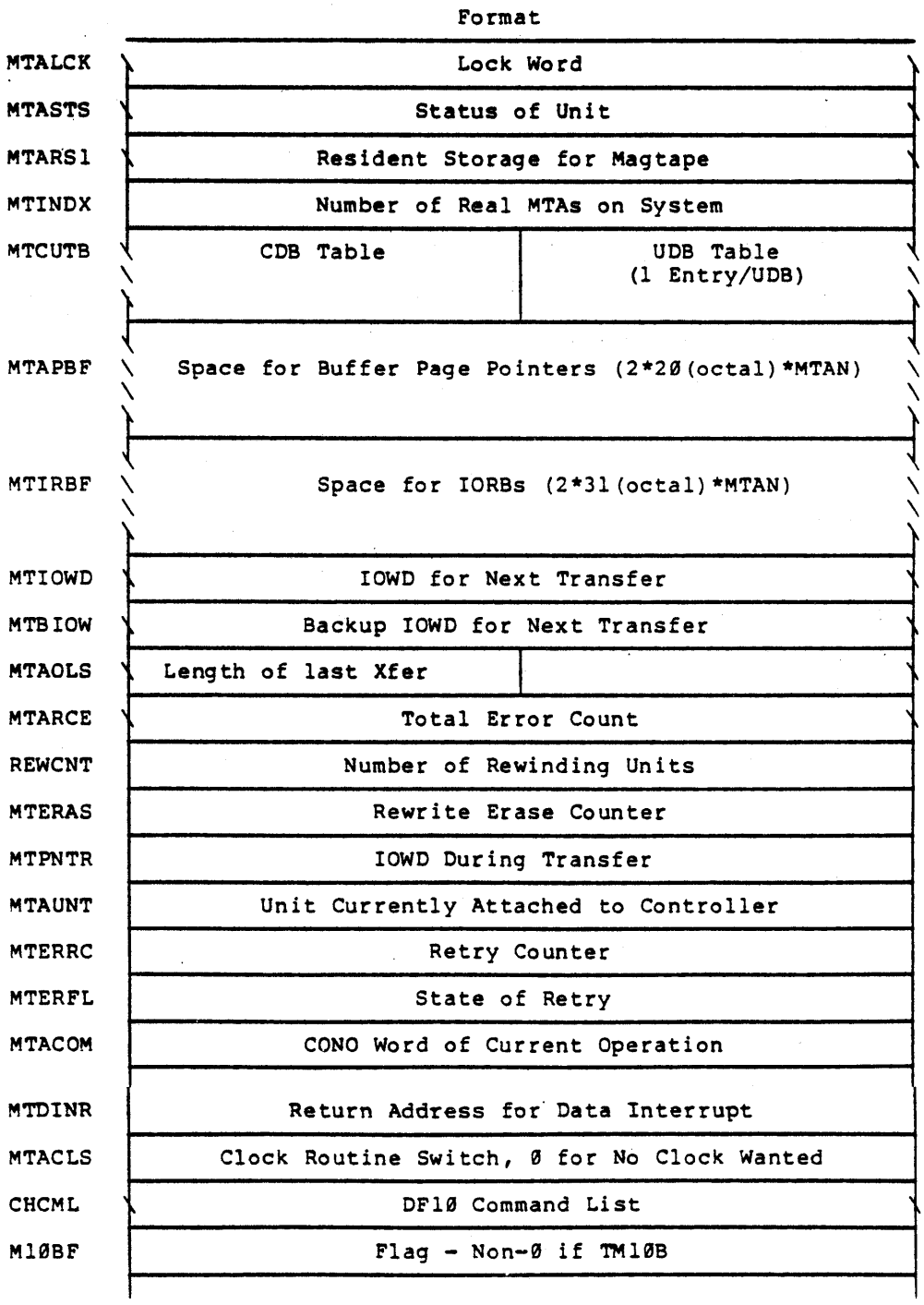
Symbol	Bits	Pointer	Content
	0-15	PGFNC	Function code: load page counter
	16	PGENB	Enable interrupts
	17-31	PGCTR	Page counter value
LP%IRP	32	LPIRP	Interrupt request pending
LP%RBR	33	LPRBR	RAM or VFU being reloaded
LP%LTR	34	LPLTR	Translation RAM requires reloading
LP%LVF	35	LPLVF	VFU requires reloading

Name: MTA-STORAGE-AREA

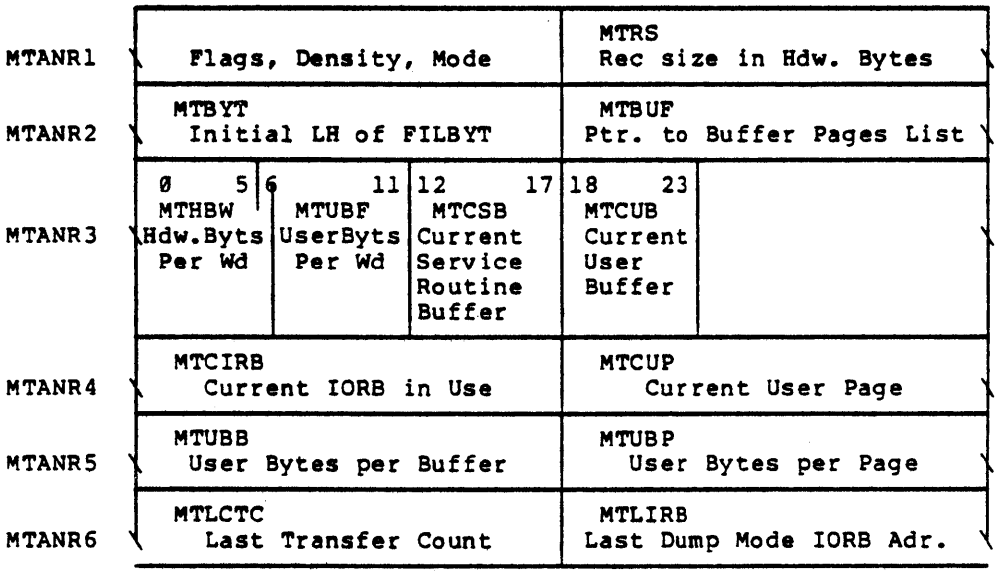
Description: Magtape storage area; each entry (unless otherwise noted) is MTAN words long where MTAN equals the number of magtape units on the system.

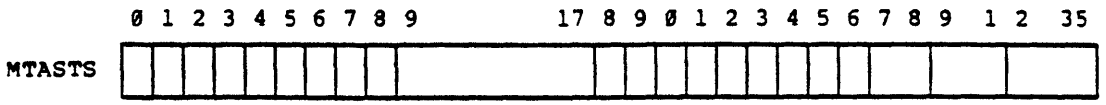
Defined in: STG

Referenced by: MAGTAP

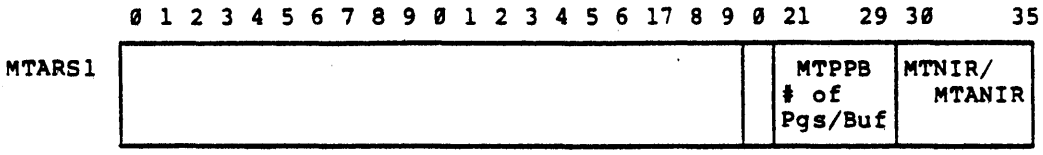


The following MTA storage items are in the nonresident area of the monitor and each item is MTAN words long.

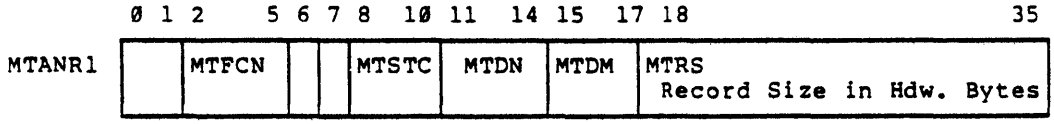




Symbol	Bits	Pointer	Content
OPN	0		Unit has been opened
OPND	2		Unit has been opened for dum mode
DMPWT	3		Waiting for a dump mode to finish
LTERR	4		Error Occurred on last dump mode
BUFA	5		Buffers have been assigned
CLOF	6		CLOSF in progress
MTOWT	7		MTOPR in progress
MTIEL	8		Inhibit error logging
MT%ILW	18		Write lock
MT%DVE	19		Hardware device error
MT%DAE	20		Data error
MT%SER	21		No error retry
MT%EOF	22		EOF
MT%IRL	23		Illegal record length
MT%BOT	24		Beginning of tape
MT%EOT	25		Physical end of tape
MT%EVP	26		Even Parity
MT%DEN	27-28		Density (0 is normal) .MTLOD=1 Low Density (200 BPI) .MTMED=2 Medium Density (556 BPI) .MTHID=3 High Density (800 BPI)
MT%CCT	29-31		Character Counter



Bits	Pointer	Content
20	ABORTF	An error occurred and IORBs aborted
21-29	MTPPB	Number of pages per buffer
30-35	MTNIR	Number of IORBs queued
30-35	MTANIR	Absolute version of MTNIR



Bits	Pointer	Content
0-1	MTNTM	Count of EOFs written
2-5	MTFCN	Last function performed
6	MTPAR	Parity
7	MTRBF	Reading backwards flag
8-10	MTSTC	CLOSF function counter
11-14	MTDN	Density
15-17	MTDM	Data mode
18-35	MTRS	Record size in hardware bytes

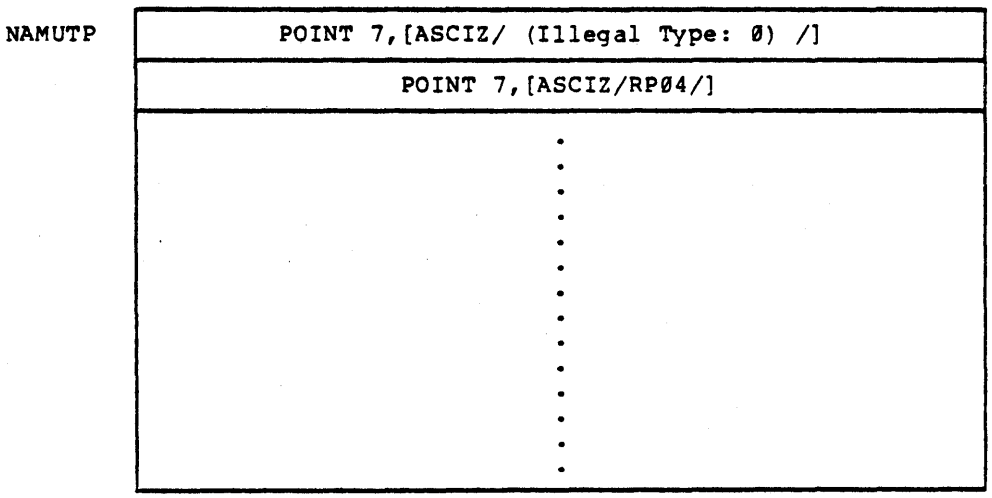
Name: NAMUTP

Description: Name Unit Type Pointers. This table contains pointers to unit type names for disks (i.e., RP06, RM03, etc.) and magtapes (i.e., TU45, TU77, etc.).

Defined in: PHYSIO

Referenced by: PHYSIO

Format



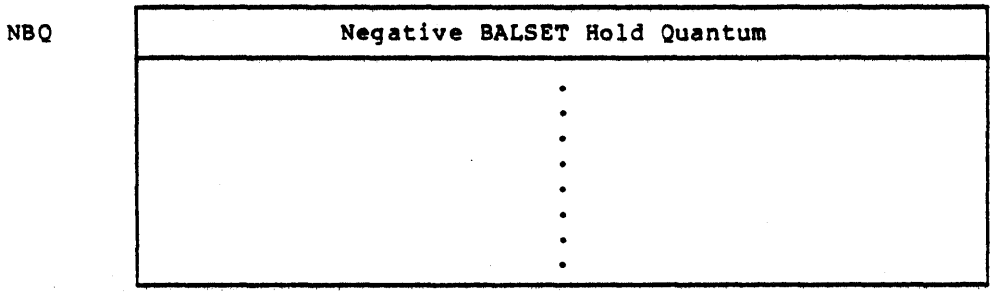
Name: NBQ

Description: Negative Balance Set Hold Quantum. This table parallels the BALSET table and maintains the minimum hold quantum for each fork in the balance set. When the fork's quantum has been exhausted, it becomes eligible for removal from the balance set.

Defined in: SCHED

Referenced by: SCHED

Format



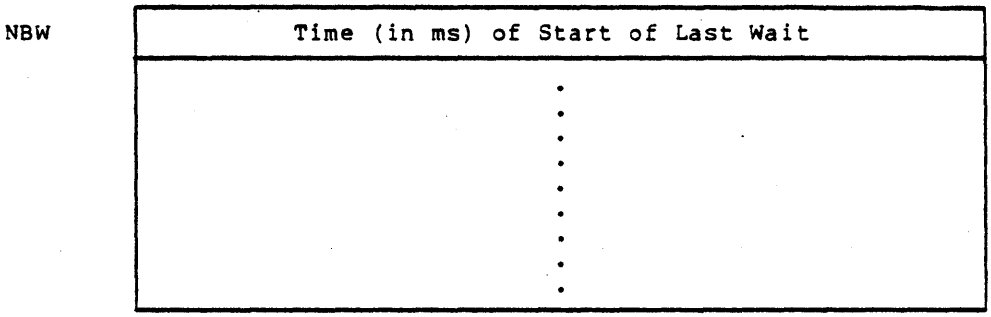
Name: NBW

Description: Balance Set Wait Time. This table records the time of day each fork in the balance set goes into a balance set wait state waiting for disk/drum I/O. When "wait satisfied" occurs, total wait time can be calculated and stored for system statistics. (Parallel table to BALSET)

Defined in: SCHED

Referenced by: SCHED

Format

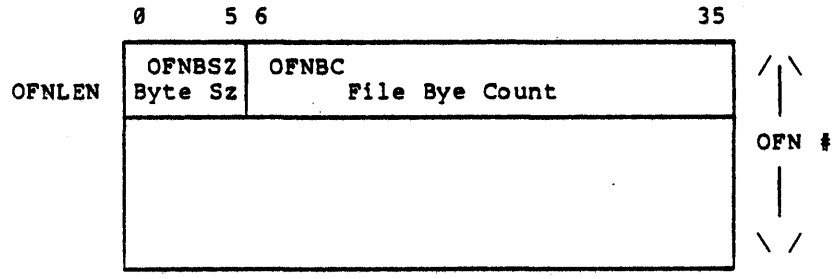


Name: OFNLEN

Description: Open File Length Table. This table, indexed by OFN, contains the current file byte size and file byte count for each open file. It is parallel to the OFN areas of the SPT and SPTH tables.

Defined in: STG

Referenced by: DISC, JSYSF, PAGEM



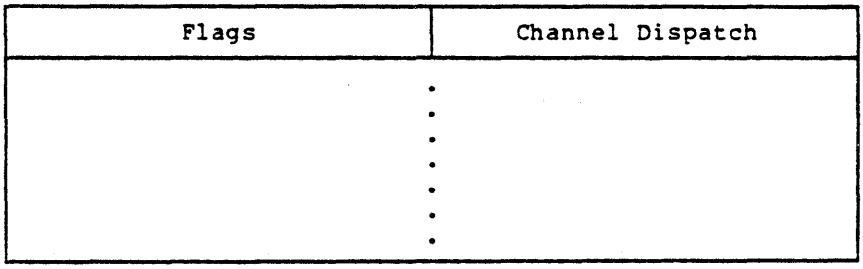
Name: PHYCHT

Description: This table contains the names of function dispatch tables for all supported channel types. Currently supported channel types are the RH20F and the RH11F. Both are supported by the same named dispatch table, RH2DSP, in their respective monitor modules, PHYH2 or PHYH11. However, only one of these modules is present in a given monitor.

Defined in: STG

Referenced by: PHYSIO

Format



Note: This table currently contains only one entry .CTRH2,,RH2DSP

Name: PHYUNT

Description: Table of known unit dispatch routines, (i.e., one for disk and one for magtape).

Defined in: STG

Referenced by: PHYH2

Format

PHYUNT

Type (DSK)=.UTRP4	DSK Unit Dispatch Adr. = RH2DSP
Type (MTA)=.UTTM2	MTA Unit Dispatch Adr. = TM2DSP

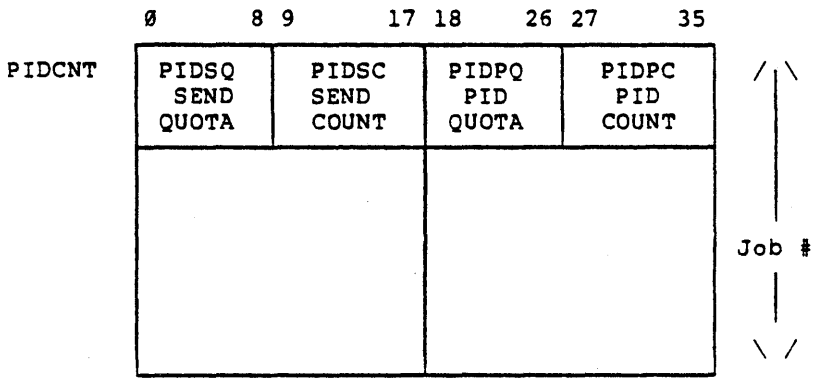
Name: PIDCNT

Description: Process ID Count Table. This non-resident table, indexed by job #, holds the send quota and count and the PID quota and count for each job.

Defined in: STG

Referenced by: IPCF

FORMAT

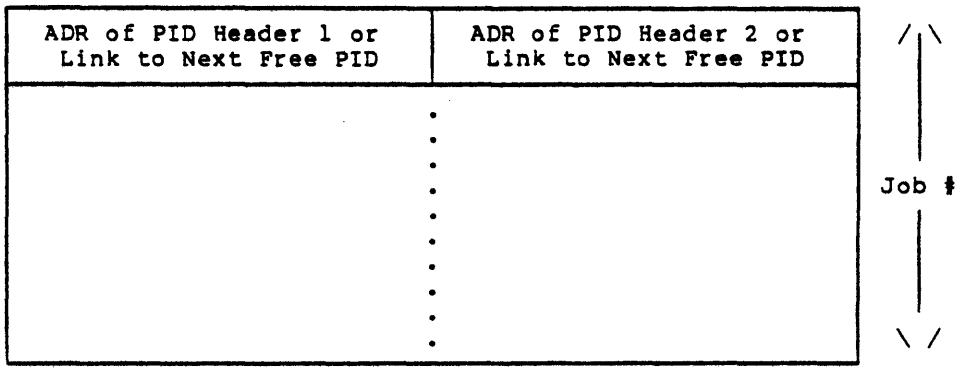


Name: PIDTBL

Description: Process ID Table. This non-resident table, indexed by job #, is a table of halfword entries, one entry per PID; the number of entries in the table is two times the number of jobs. (The monitor is assuming an average of 2 PIDs per job.) If a PID is in use, the table entry will contain the address of the PID header. PID's not in use are linked together; PIDLST is the free PID header.

Defined in: STG

Referenced by: IPCF



Name: PSB

Description: Process Storage Block. Each fork has a Process Storage Block which holds per-process information such as: the fork's PC and ACs when not running; the forks known to this process, and accounting, PSI, paging and directory information.

It also holds trapping information and the hardware cells for the fork's User Process Table (See UPT Table). Page 2 of the PSB houses the push down list used by the monitor when executing JSYSs. (i.e. in process context).

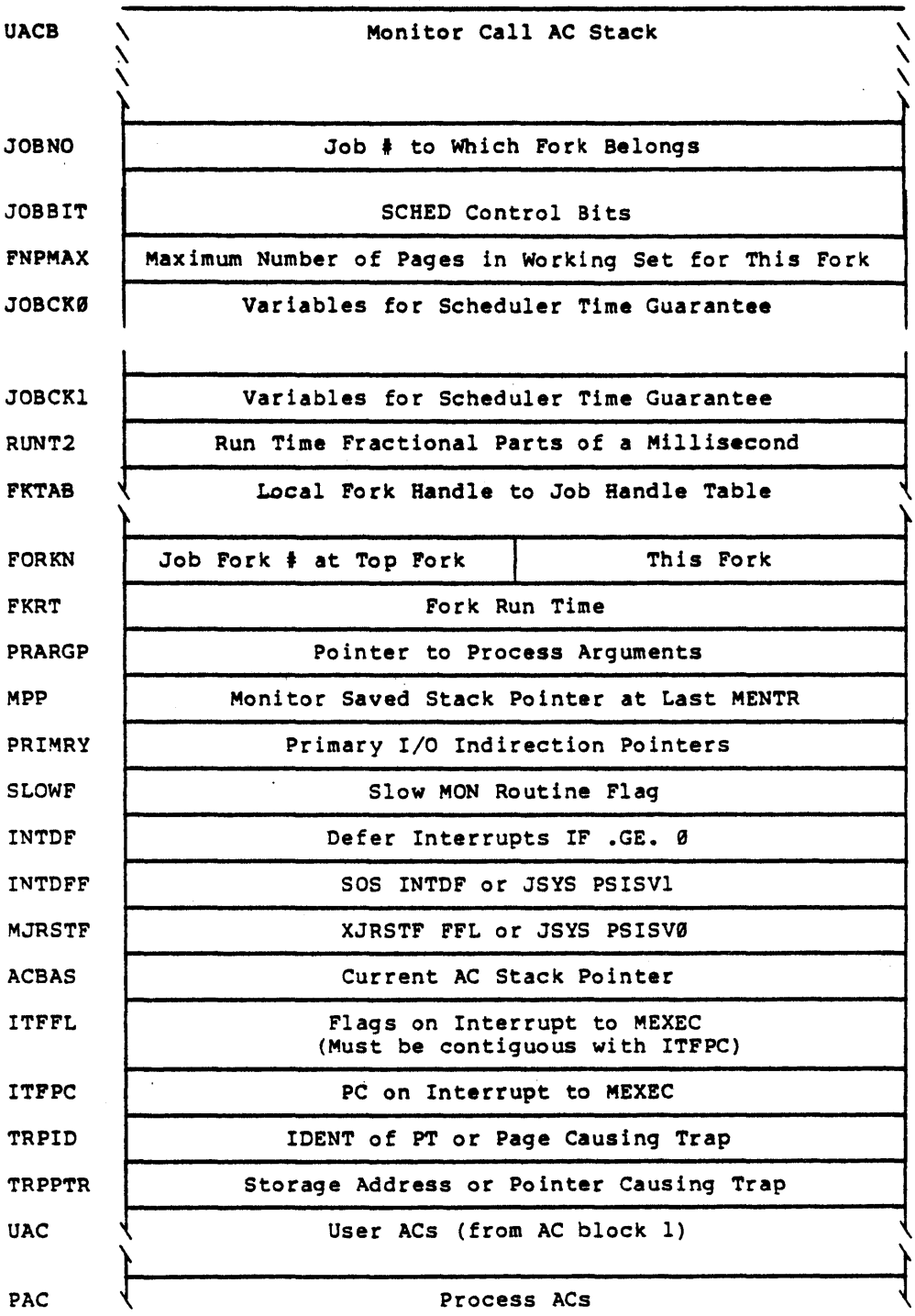
The PSBMAP map in the PSB points to all of the per-process storage area (including the PSB itself). When the monitor references the current fork's per-process area, it uses virtual addresses, 707000-777777. (The monitor's mapped slots in MMAP for virtual pages 707-777 point to the PSBMAP via indirect pointers).

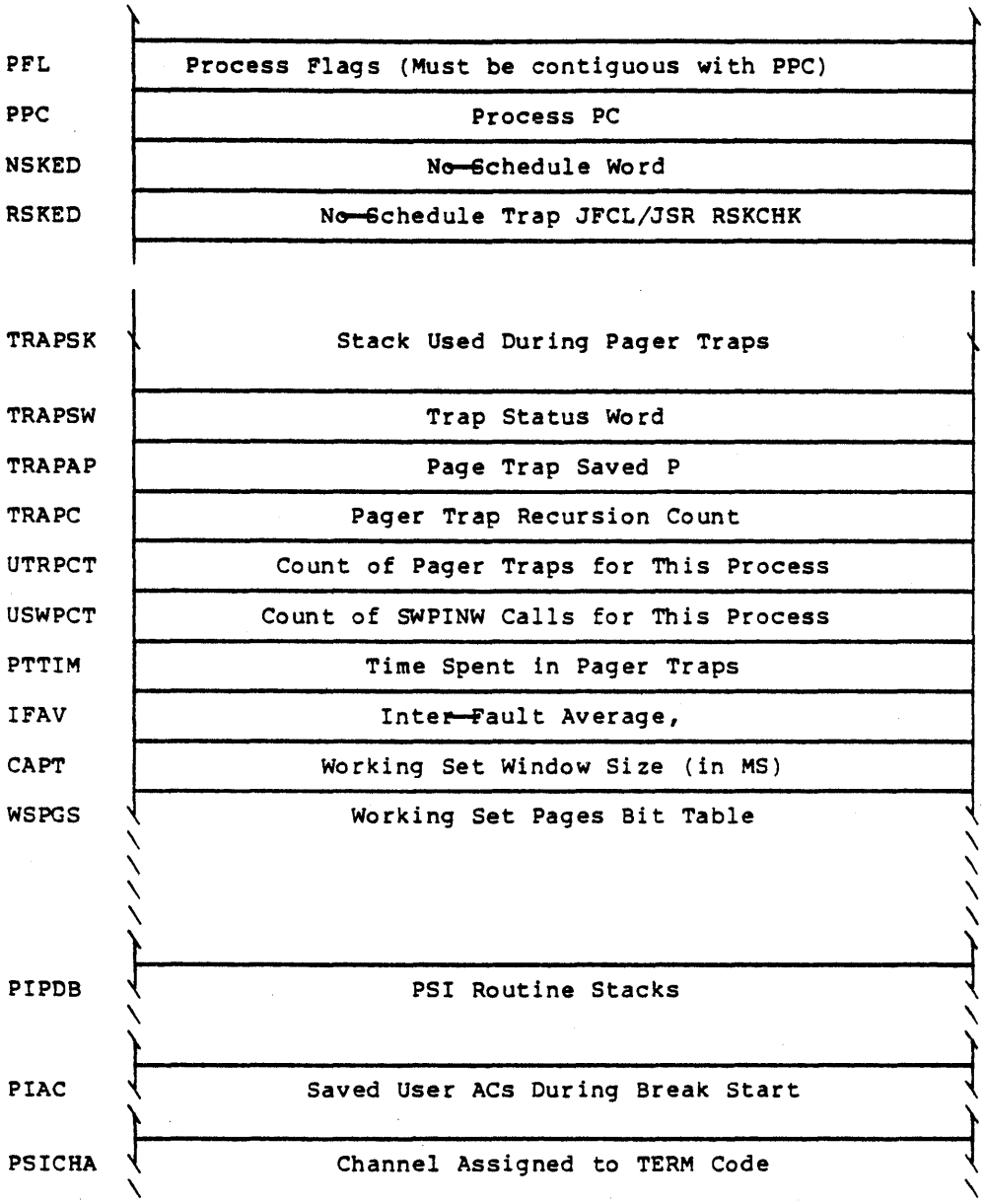
Monitor virtual pages in the per-process area are used for the PSB table, the User's Page Map Table, and IDXFIL. Some per-process pages are used temporarily by the Swapper and Map routines and by by the Program Software Interrupt (PSI) and fork utility routines.

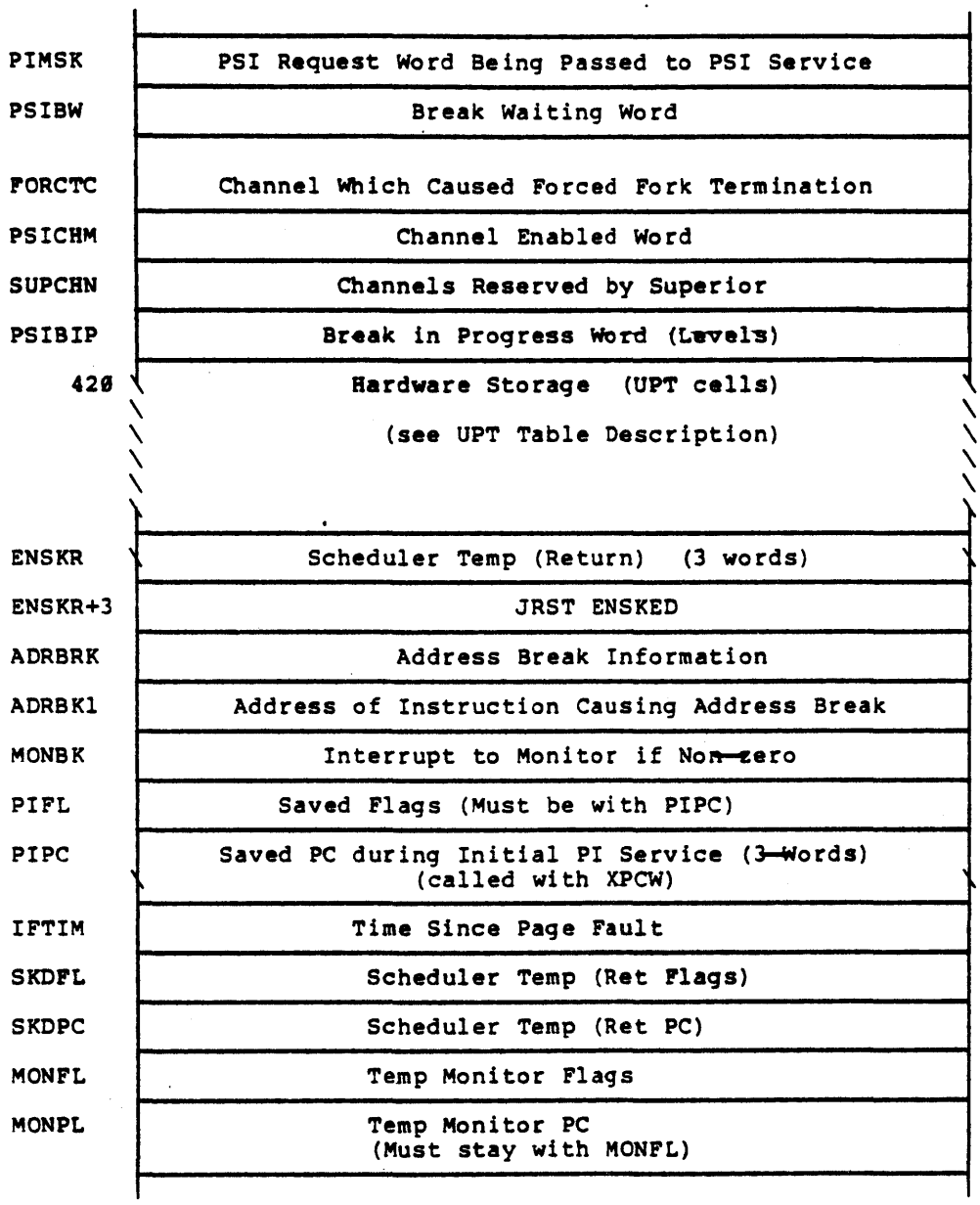
Defined in: PROLOG

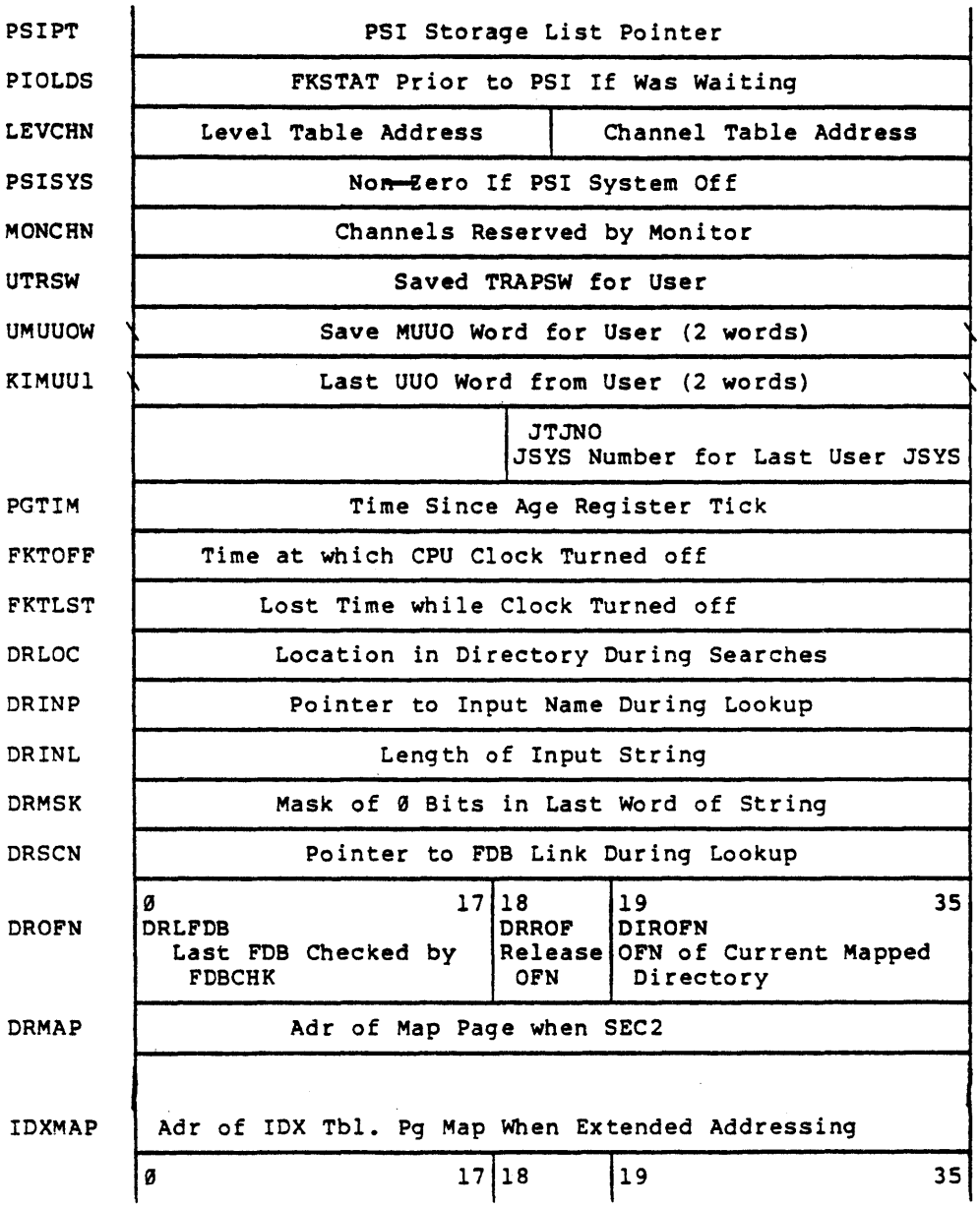
Referenced by: APRSRV, DATIME, DIRECT, DISC, DSKALC, DTESRV, ENQ, FESRV, FILINI, FILMSC, FORK, FREE, GTJFN, IO, IPCF, JSYSA, JSYSF, LINEPR, LOGNAM, MAGTAP, MEXEC, PAGEM, PHYSIO, POSTLD, SCHED, SYSERR

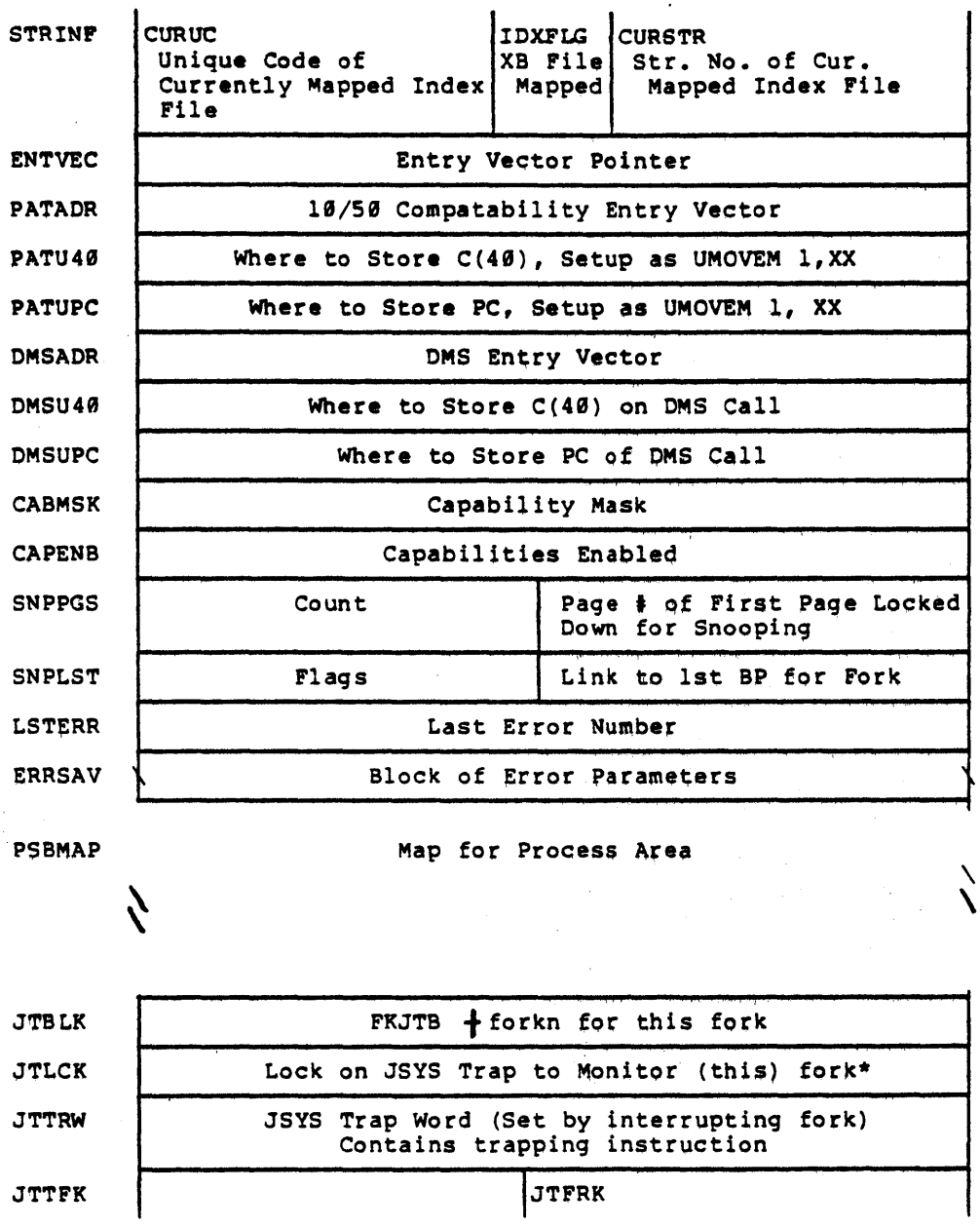
Format

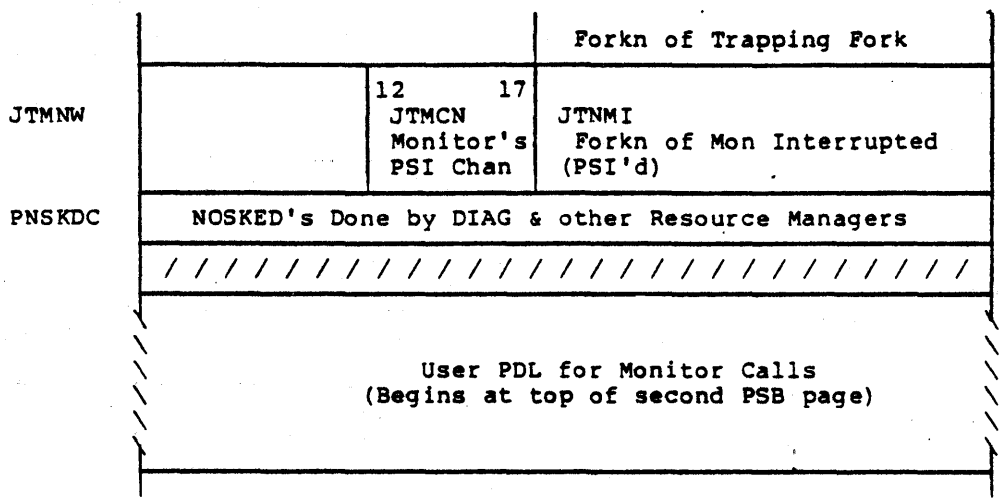

UACB
Monitor Call AC Stack
JOBNO
Job # to Which Fork Belongs
JOBBIT
SCHED Control Bits
FNPMAX
Maximum Number of Pages in Working Set for This Fork
JOBCKØ
Variables for Scheduler Time Guarantee
JOBCK1
Variables for Scheduler Time Guarantee
RUNT2
Run Time Fractional Parts of a Millisecond
FKTAB
Local Fork Handle to Job Handle Table
FORKN
Job Fork # at Top Fork
This Fork
FKRT
Fork Run Time
PRARGP
Pointer to Process Arguments
MPP
Monitor Saved Stack Pointer at Last MENTR
PRIMRY
Primary I/O Indirection Pointers
SLOWF
Slow MON Routine Flag
INTDF
Defer Interrupts IF .GE. Ø
INTDFF
SOS INTDF or JSYS PSISV1
MJRSTF
XJRSTF FFL or JSYS PSISVØ
ACBAS
Current AC Stack Pointer
ITFFL
Flags on Interrupt to MEXEC
(Must be contiguous with ITFPC)
ITFPC
PC on Interrupt to MEXEC
TRPID
IDENT of PT or Page Causing Trap
TRPPTR
Storage Address or Pointer Causing Trap
UAC
User ACs (from AC block 1)
PAC
Process ACs


PFL
Process Flags (Must be contiguous with PPC)
PPC
Process PC
NSKED
No-Schedule Word
RSKED
No-Schedule Trap JFCL/JSR RSKCHK
TRAPSK
Stack Used During Pager Traps
TRAPSW
Trap Status Word
TRAPAP
Page Trap Saved P
TRAPC
Pager Trap Recursion Count
UTRPCT
Count of Pager Traps for This Process
USWPCT
Count of SWPINW Calls for This Process
PTTIM
Time Spent in Pager Traps
IFAV
Inter-Fault Average,
CAPT
Working Set Window Size (in MS)
WSPGS
Working Set Pages Bit Table
PIPDB
PSI Routine Stacks
PIAC
Saved User ACs During Break Start
PSICHA
Channel Assigned to TERM Code


PIMSK
PSI Request Word Being Passed to PSI Service
PSIBW
Break Waiting Word
FORCTC
Channel Which Caused Forced Fork Termination
PSICHM
Channel Enabled Word
SUPCHN
Channels Reserved by Superior
PSIBIP
Break in Progress Word (Levels)
420
Hardware Storage (UPT cells)
(see UPT Table Description)
ENSKR
Scheduler Temp (Return) (3 words)
ENSKR+3
JRST ENSKED
ADRBRK
Address Break Information
ADRBK1
Address of Instruction Causing Address Break
MONBK
Interrupt to Monitor if Non-zero
PIFL
Saved Flags (Must be with PIPC)
PIPC
Saved PC during Initial PI Service (3 Words)
(called with XPCW)
IFTIM
Time Since Page Fault
SKDFL
Scheduler Temp (Ret Flags)
SKDPC
Scheduler Temp (Ret PC)
MONFL
Temp Monitor Flags
MONPL
Temp Monitor PC
(Must stay with MONFL)


PSIPT
PSI Storage List Pointer
PIOLDS
FKSTAT Prior to PSI If Was Waiting
LEVCHN
Level Table Address
Channel Table Address
PSISYS
Non-Zero If PSI System Off
MONCHN
Channels Reserved by Monitor
UTRSW
Saved TRAPSW for User
UMUUOW
Save MUUO Word for User (2 words)
KIMUU1
Last UUO Word from User (2 words)
JTJNO
JSYS Number for Last User JSYS
PGTIM
Time Since Age Register Tick
FKTOFF
Time at which CPU Clock Turned off
FKTLST
Lost Time while Clock Turned off
DRLOC
Location in Directory During Searches
DRINP
Pointer to Input Name During Lookup
DRINL
Length of Input String
DRMSK
Mask of Ø Bits in Last Word of String
DRSCN
Pointer to FDB Link During Lookup
DROFN
Ø
17
DRLFDB
Last FDB Checked by FDBCHK
18
DRROF
Release OFN
19
35
DIROFN
OFN of Current Mapped Directory
DRMAP
Adr of Map Page when SEC2
IDXMAP
Adr of IDX Tbl. Pg Map When Extended Addressing
Ø
17
18
19
35


STRINF
CURUC
Unique Code of Currently Mapped Index File
IDXFLG
XB File Mapped
CURSTR
Str. No. of Cur. Mapped Index File
ENTVEC
Entry Vector Pointer
PATADR
10/50 Compatability Entry Vector
PATU40
Where to Store C(40), Setup as UMOVEM 1,XX
PATUPC
Where to Store PC, Setup as UMOVEM 1, XX
DMSADR
DMS Entry Vector
DMSU40
Where to Store C(40) on DMS Call
DMSUPC
Where to Store PC of DMS Call
CABMSK
Capability Mask
CAPENB
Capabilities Enabled
SNPPGS
Count
Page # of First Page Locked Down for Snooping
SNPLST
Flags
Link to 1st BP for Fork
LSTERR
Last Error Number
ERRSAV
Block of Error Parameters
PSBMAP
Map for Process Area
JTBLK
FKJTB + forkn for this fork
JTLCK
Lock on JSYS Trap to Monitor (this) fork*
JTTRW
JSYS Trap Word (Set by interrupting fork) Contains trapping instruction
JTTFK
JTFRK



* JTLCK: Lock protects JTTRW and allows only one JSYS trap interrupt at a time to this monitor

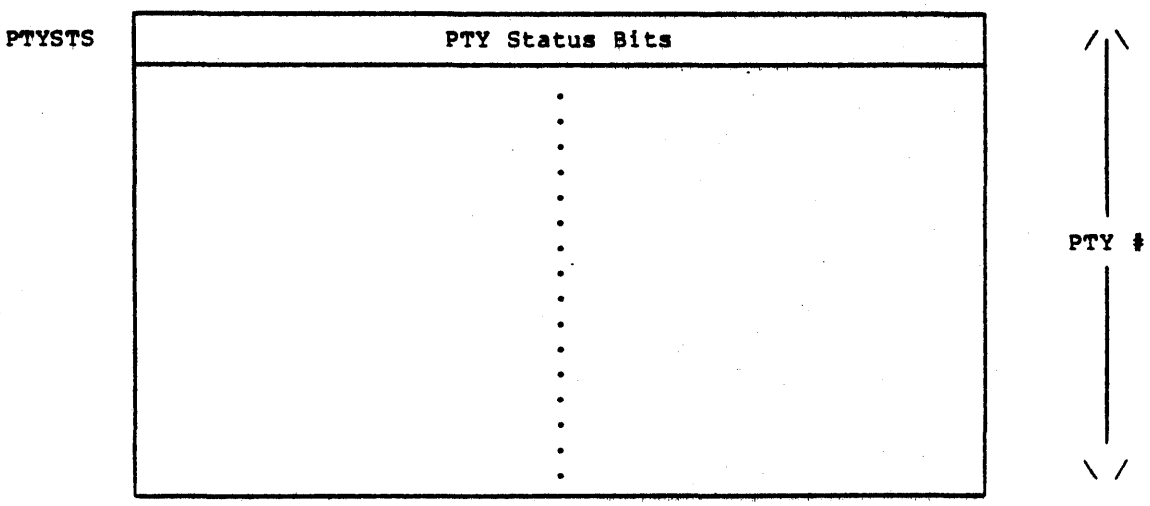
Name: PTYSTS

Description: Pseudo Terminal Status Table. This table, indexed by PTY, contains the PTY's status word.

Defined in: STG

Referenced by: FILMSC

Format



Name: Q-BLOCK

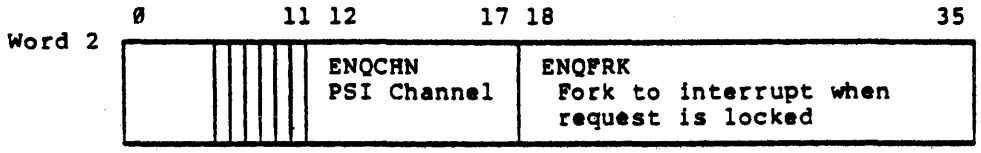
Description: The information for each ENQ request is stored in a Q-BLOCK. QBLOCK's are doubly linked for each job; the list header is in the right half of ENQLST in the JSB. Also, QBLOCK are doubly linked on a system wide list for each lock block; the list header is in the lock block.

Defined in: ENQ

Referenced by: ENQ

FORMAT

Ø	ENQLJQ: Back Pointer To Last Q-Block For Job		ENQNJQ: Forward Pointer To Next Q-Block For Job
1	ENQLLQ: Back Pointer To Last Q-Block		ENQNLQ: Forward Pointer To Next Q-Block
2	ENQFLG: Flags Either Lock or Q	ENQCHN: PSI Channel	ENQFRK: Fork to Interrupt When Request is Locked
3	ENQNR: # of Resources Requested From Pool		ENQID: Request ID Code
4	ENQLRQ: Back Pointer To Last Q-Block of Request		ENQFQ: Forward Pointer To Next Q-Block of Request
5	ENQLBP: Pointer to Lock-Block of this Q		ENQGRP: Group # For Sharable Requests
6	ENQNST: Nest Count		ENQJFN: JFN of Request or -1, -2, or -3
7			ENQMSK: Pointer to the Mask Block



Symbol	Bits	Pointer	Meaning
EN.LTL=40	6		Long Term Block
EN.INV=20	7		This Q-Block is invisible
EN.LOK=10	8		The Q-Block has the Lock locked.
EN.TXT=4	9		This Block has a Text String Identity.
EN.EXC=2	10		Request is Exclusive
EN.LB=1	11		This is the Lock-Block
	13-17	ENQCHN	PSI Channel (-1 means job blocked)
	18-35	ENQFRK	Fork # of Creator of Q-Block

Name: RES-FREE-SPACE

Description: Resident Free Space Storage. This area contains the resident free space bit table, RESBTB, which indicates which 4-word blocks of the resident free space pools are in use.

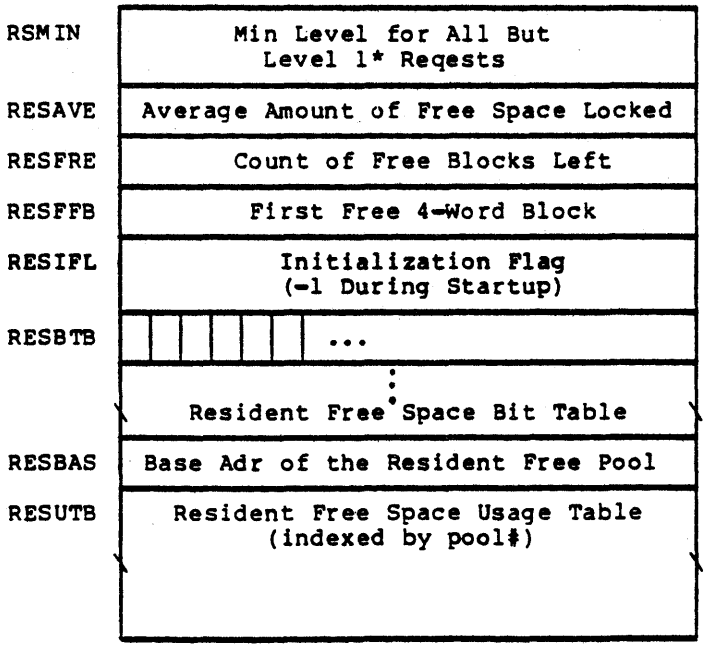
The resident free space (in RESFRP) is pooled by PHYSIO for building UDBs, CDBs, KDBs, and SDBs; by TTYSRV for terminal messages and line dynamic data blocks; by NETSRV for an input and output buffer for each active line; and by TIMER for the TIMER JSYS when it builds a job's run-time limit block. (See JOBRTL table).

This area also contains a resident free space usage table, RESUTB, which indexed by pool #, holds the amount of free space left for each pool.

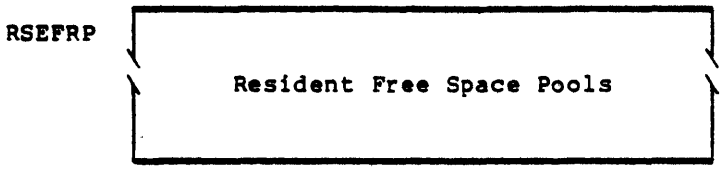
Defined in: STG

Referenced by: DSKALC, FREE, FESRV, NSPSRV, PHYSIO, TIMER, TTYSRV

FORMAT



The following storage is in the non-resident area of the monitor.



*Note: Requests for Resident Free Space are given priority levels where:
Level 1 - Has highest priority and monitor always tries to assign space. Page faults are not allowed.
Level 2 - Has second level priority where monitor will not assign space if free storage would go below RESMIN. Page faults are not allowed.
Level 3 - Has lowest priority and requests for this level are made in process context. Page Faults are allowed.

Name: SCDRQB

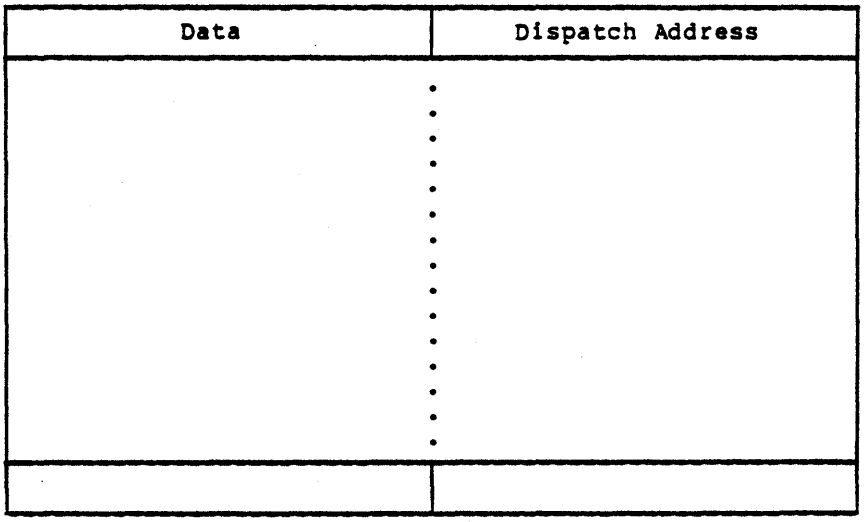
Description: Scheduler Request Table. During the scheduler's overhead cycle, the initial job startup request is placed in this table when the first CTRL/C is processed. The table is later examined in the same cycle and all entries are processed by dis patching to each entry's dispatch address.

Defined in: SCHED

Referenced by: SCHED, TTYSRV

Format

SCDRQB



Name: SDB

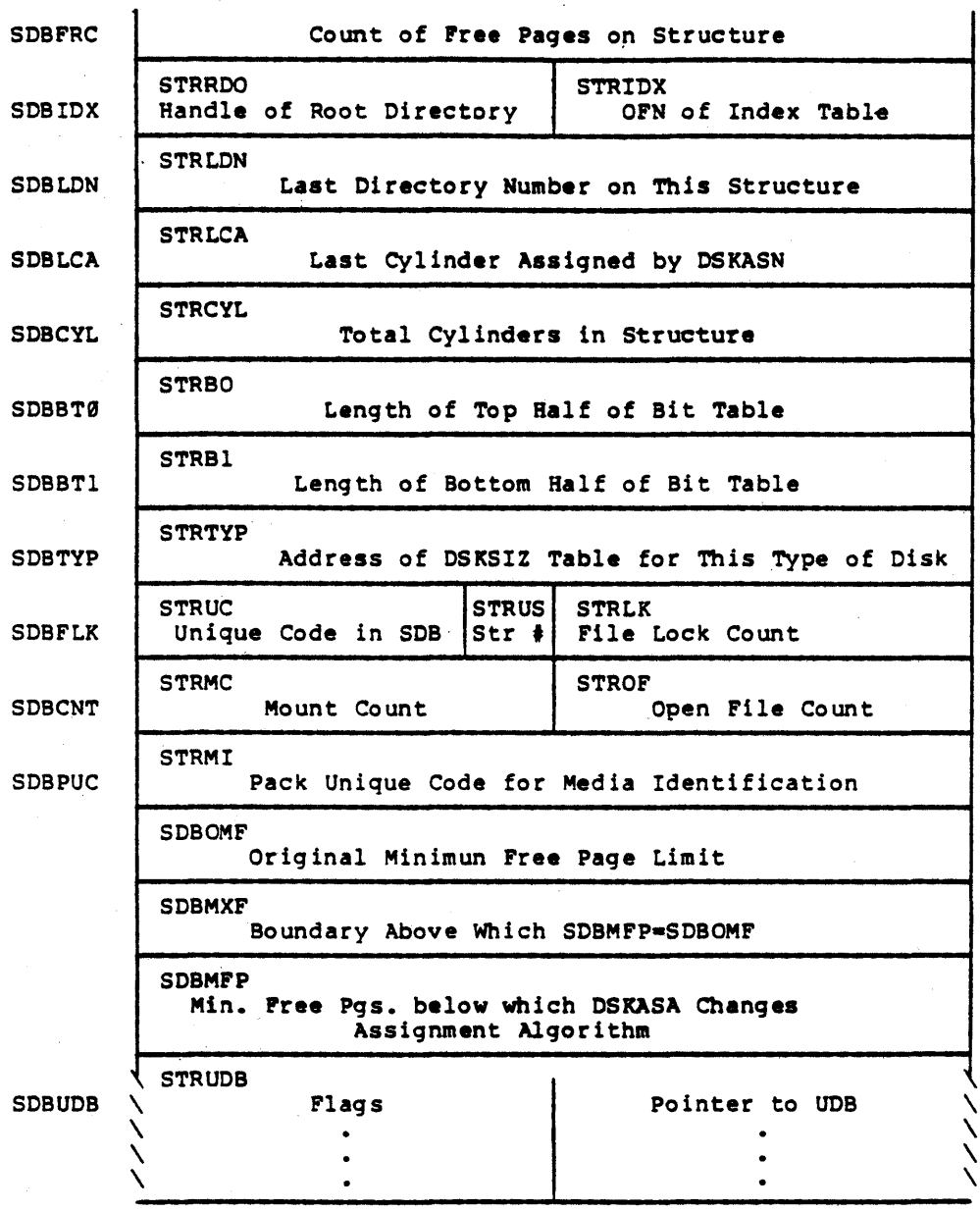
Description: Structure Data Block. This block, one per structure, contains information about the structure's units, master directory (i.e. Root-Directory), bit map for disk page allocation/deallocation, and assigned swapping area. It also contains mount and open-file information. SDBBLO is the name of the storage area reserved for handling the SDB for the Public Structure (PS).

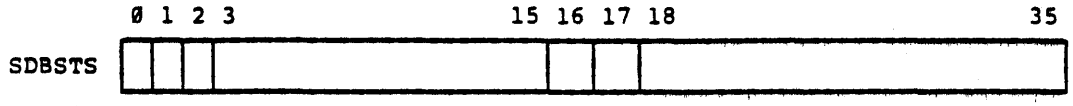
Defined in: STG

Referenced by: DEVICE, DIRECT, DSKALC, FILINI, FUTILI, IO, PHYSIO, JSYSA, JSYSF, MEXEC, MSTR

Format

SDBNAM=Ø	STRNAM Structure Name (in SIXBIT)	
SDBNUM=1	STRNUM Number of Units in Structure	
. . SDBSIZ .	STRSIZ Size (in sectors) of Each Unit in Structure	
SDBSTS	STRSTS Status Flags	STRJB Initing Fork #
SDBRXB	STRRXB Address of Root Directory Index Block	
SDBBXB	STRBXB Address of Backup Copy of Root Directory Index Block	
SDBNSS	STRNSS Number of Swapping Sectors per Unit	
SDBFSS	STRFSS First Swapping Sector per Unit	
SDBBTB	STRBTB OFN of Bit Table	
	STRFC	


SDBFRC
Count of Free Pages on Structure
SDBIDX
STRRDO
Handle of Root Directory
STRIDX
OFN of Index Table
SDBLDN
STRLDN
Last Directory Number on This Structure
SDBLCA
STRLCA
Last Cylinder Assigned by DSKASN
SDBCYL
STRCYL
Total Cylinders in Structure
SDBBTØ
STRBO
Length of Top Half of Bit Table
SDBBT1
STRB1
Length of Bottom Half of Bit Table
SDBTYP
STRTYP
Address of DSKSIZ Table for This Type of Disk
SDBFLK
STRUC
Unique Code in SDB
STRUS
Str #
STRLK
File Lock Count
SDBCNT
STRMC
Mount Count
STROF
Open File Count
SDBPUC
STRMI
Pack Unique Code for Media Identification
SDBOMF
Original Minimun Free Page Limit
SDBMXF
Boundary Above Which SDBMFP=SDBOMF
SDBMFP
Min. Free Pgs. below which DSKASA Changes Assignment Algorithm
SDBUDB
STRUDB
Flags
Pointer to UDB



Symbol	Bits	Pointer	Content
MSPS	0	STPS	Structure is public
MSDIS	1	STDIS	Structure is being dismounted
MSDOM	2	STDOM	Structure is domestic
	16	STIDX	Index table file OFN has been set up
	17	STCRD	Creating Root Directory on this Structure
	18-35	STRJB	Initializing job # (only legal user while structure is being initialized)

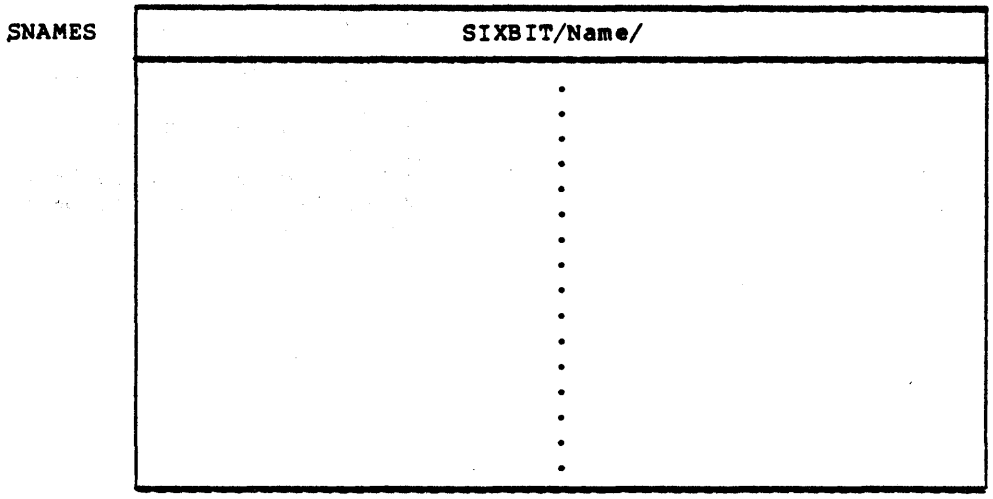
Name: SNAMES

Description: Subsystem Names. Each entry contains a subsystem program name.

Defined in: STG

Referenced by: MEXEC

Format



Name: SNBLKS

Description: Subsystem Blocks. Each entry contains the number of blocks for a subsystem program. This table is parallel to SNAMES.

Defined in: STG

Referenced by: MEXEC

Format

SNBLKS

Number of Blocks
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.
.

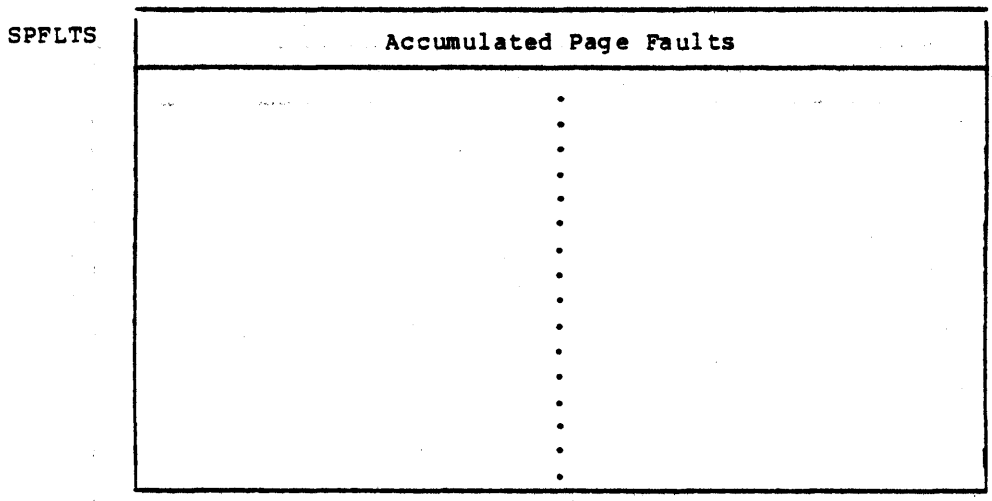
Name: SPFLTS

Description: Subsystem Page Faults. Each entry contains the accumulated number of page faults of a subsystem program. This is a parallel table to SNAMES.

Defined in: STG

Referenced by: MEXEC, PAGEM

Format



Name: SPT

Description: Special Pages Table. This table is pointed to by the firmware's SPT Base Register (an AC in an AC Block reserved for hardware/firmware registers) which is setup by the monitor at system initialization time.

It is referenced directly by the paging firmware (bits 12-35 only) when virtual to physical address translation takes place and shared and indirect pointers are involved.

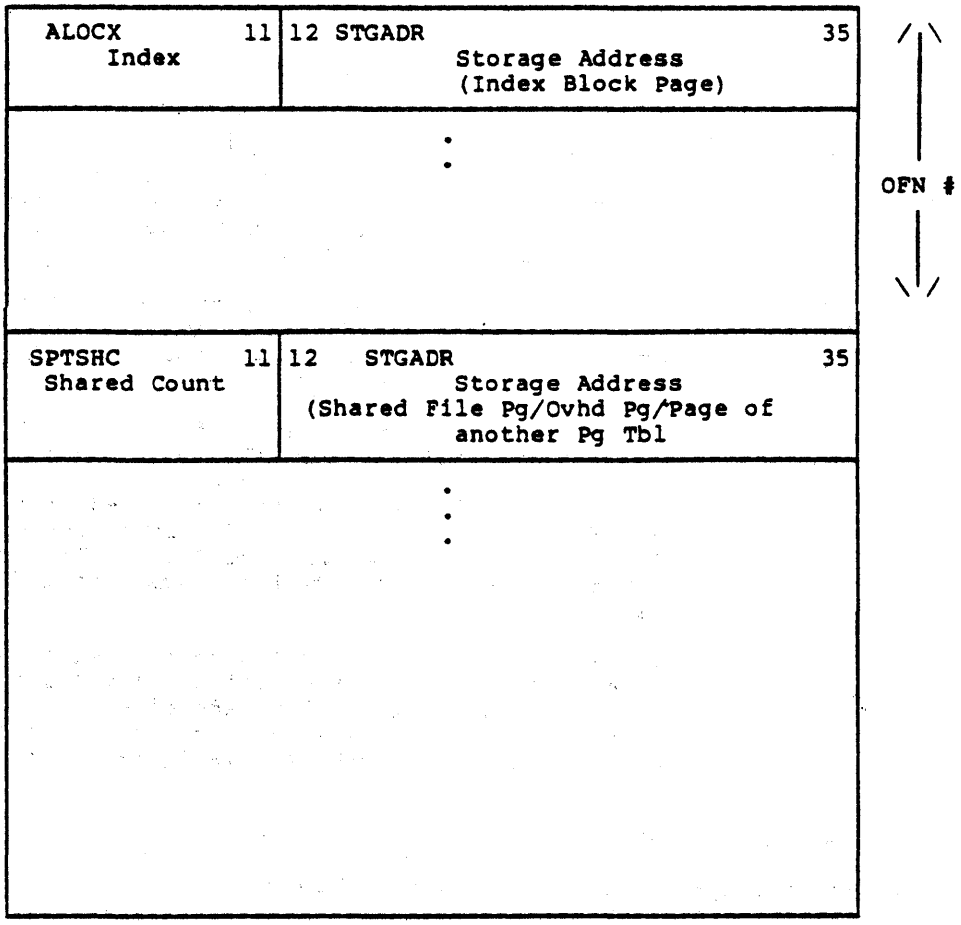
The first part of the table (of length NOFN) is used to point to index blocks in memory (or swapping area) for open files and an index into this part is often referred to as an OFN (Open File Number). The remainder of the table is used to point to PSBs, JSBs, UPTs, UPTAs, (User Page Map Tables), and shared file pages.

The ALOCX value in the OFN area is used as an index into the allocation tables (ALOC1 & ACOC2) to obtain information about the directory of the open file, (i.e., pages left in quota). The share count in the non -OFN area is indexed for each sharing of the page.

Defined in: STG

Referenced by: APRSRV, FORK, PAGEM, SCHED

Format


ALOCX 11
Index
12 STGADR 35
Storage Address
(Index Block Page)
OFN #
SPTSHC 11
Shared Count
12 STGADR 35
Storage Address
(Shared File Pg/Ovhd Pg/Page of
another Pg Tbl

STORAGE ADDRESS

SYMBOL	BITS	POINTER	CONTENTS
	12%-35	STGADR	Storage address (Interpretation follows)
NCORTM	12%-17		Non%-Core Test Mask yielding type of storage. Bits %<12%-17%>=Ø =%> Bits %<18%-35%>=Memory Pg Adr. Bits %<12%-17%>=Ø =%> Bits %<18%-35%>=Drum/DSK Adr.
DSKAB	14		Storage address is a disk address
DSKNB	15		Temporary bit used with DSKAB to say that disk address is newly assigned.
DRMAB	16		Storage address is a drum address
DRMOB	17		Used with DRMAB to indicate that the swapping area has overflowed to the disk file system. (Since TOPS%-2Ø currently uses only the disk file system for swapping, a drum storage address will always have bits 16 & 17 set.)
UAABC	17%-35		Temporary bit used by the monitor's page trap handler when a copy%-on%-write page trap has occurred. If the page to be copied is a drum address, it will be faulted in befor these bits are used, avoiding conflict over bit 17. These bits will signify to a lower level routine, SWPIN, that the page just gotten from the free list has no backup address and that it is to get a copy of another page.

Name: SPTH

Description: Special Pages Table Home Information. This table, parallel to the SPT table is referenced only by the software and is divided into two parts. The first part, indexed by OFN, is used to point to the home address of each open file (i.e., to its index block) and to hold status information about each OFN.

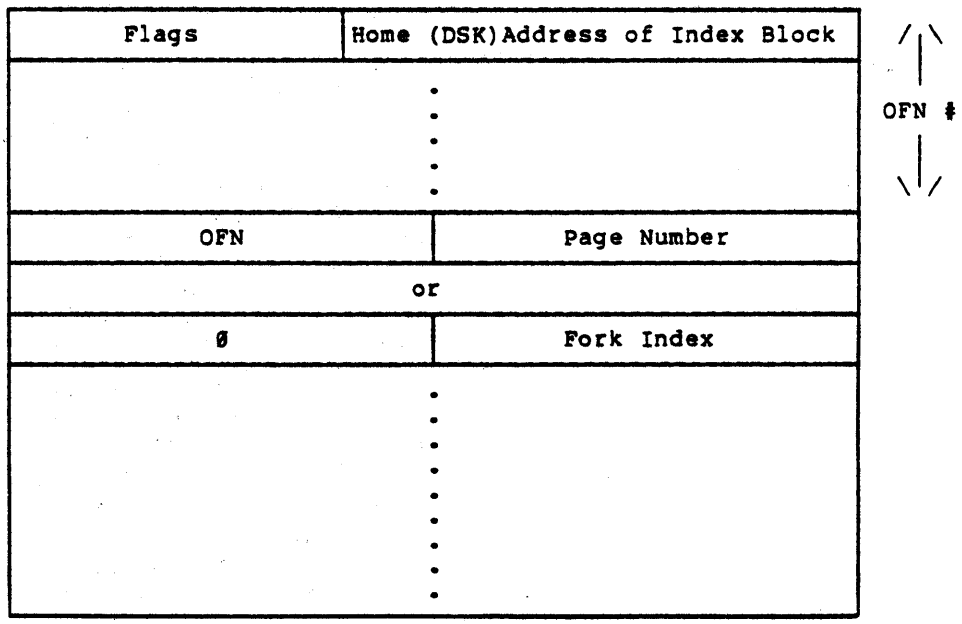
The second part is used mainly to show the page's origin. For a shared file, this is indicated by OFN ,, Page Number , where page number is within open file, OFN. For PSBs, JSBs, and UPTs, the SPTH word contains Ø ,, Fork Index. The free slots in this part are on a list chained through the SPT where the free list pointer resides in FRESPT.

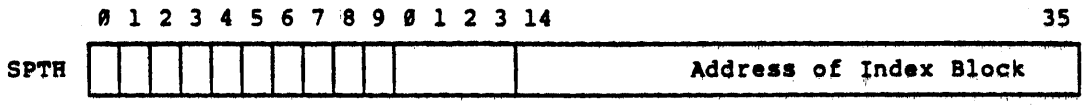
Defined in: STG

Referenced by: DISC, DSKALC, FILINI, PAGEM

Format

SPTH





Symbol	Bits	Content
*FILWB	1	File write bit in SPTH and ASOFN argument
*THAWB	2	Thawed bit
FILNB	3	"File new" bit
SPTLKB	4	LH of SPTH(OFN), XB(Index Block) in use by DDMP
OFNWRB	5	OFN has been modified
OFNBAT	6	Index block contains a bad block
OFNERR	7	Error in file (i.e., MPE)
OFNDMO	8	OFN is on a dismounted structure
OFNDUD	9	Suppress DDMP

* If a file is OPENed with thawed access (OFTHW), then both FILWB and THAWB will be set to 1. If OPEN'ed with restricted access, then the THAWB bit will be on and the FILWB will be off.

Note: A file is opened by searching the OFN part of SPTH for the index block address. If the address is found and the write and thawed bits are legal, it is a shared opening and the same index is used. If the address is not found, a new entry is made from one of the free (-1) slots in SPTH.

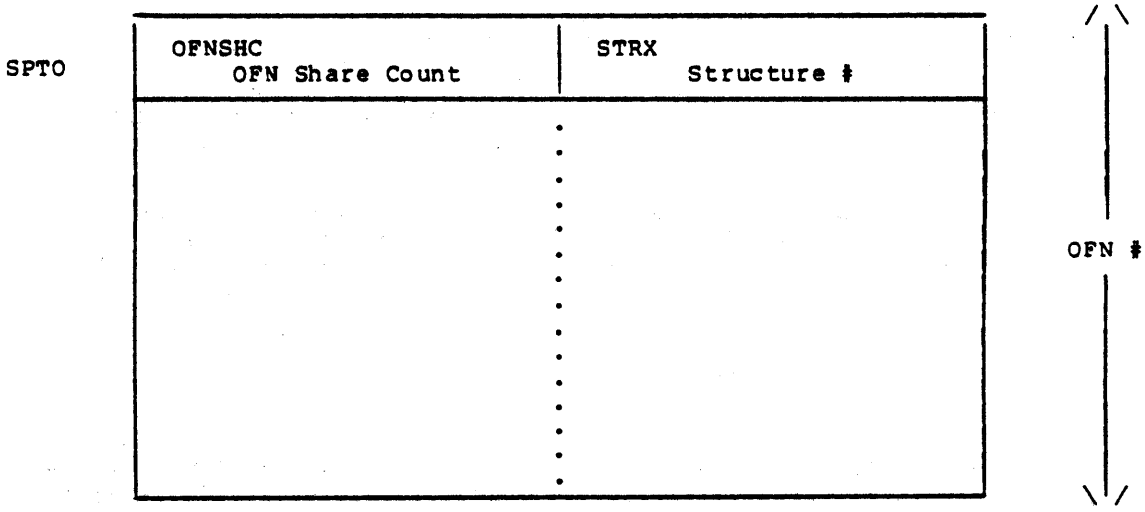
Name: SPTO

Description: Special Pages Table 0. This table is parallel to the OFN area of the SPT table and contains the structure number and open file share count for each open file. The OFN share count is indexed for each opening of the file and for each shared page within the open file.

Defined in: STG

Referenced by: PAGEM

Format



Name: SSIZE

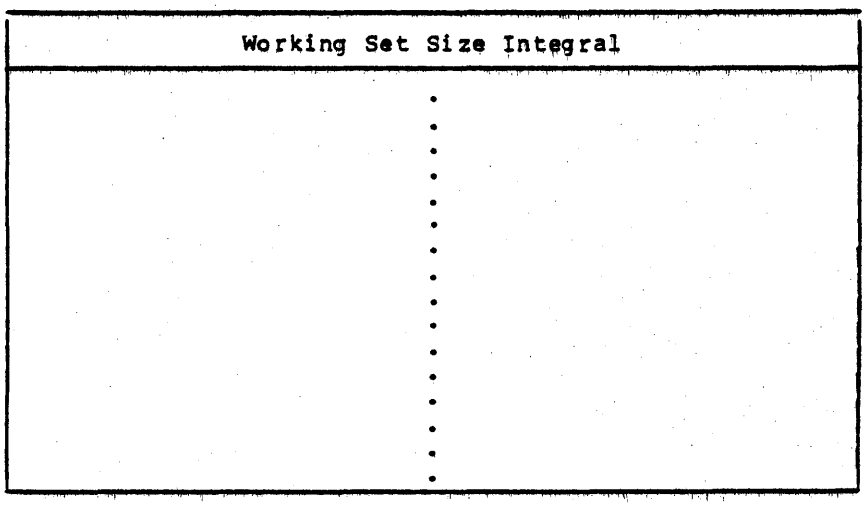
Description: Subsystem Working Set Size. Each entry contains the working set size integral for a subsystem program. This is a parallel table to SNAMES.

Defined in: STG

Referenced by: MEXEC

Format

SSIZE



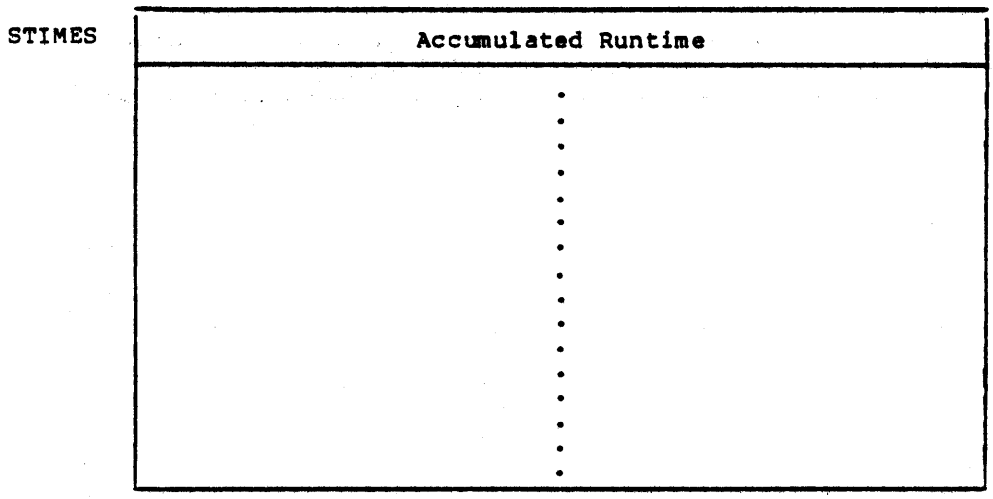
Name: STIMES

Description: Subsystem Runtimes. Each entry contains the accumulated runtime of a subsystem program. This is a parallel table to SNAMES.

Defined in: STG

Referenced by: MEXEC

Format



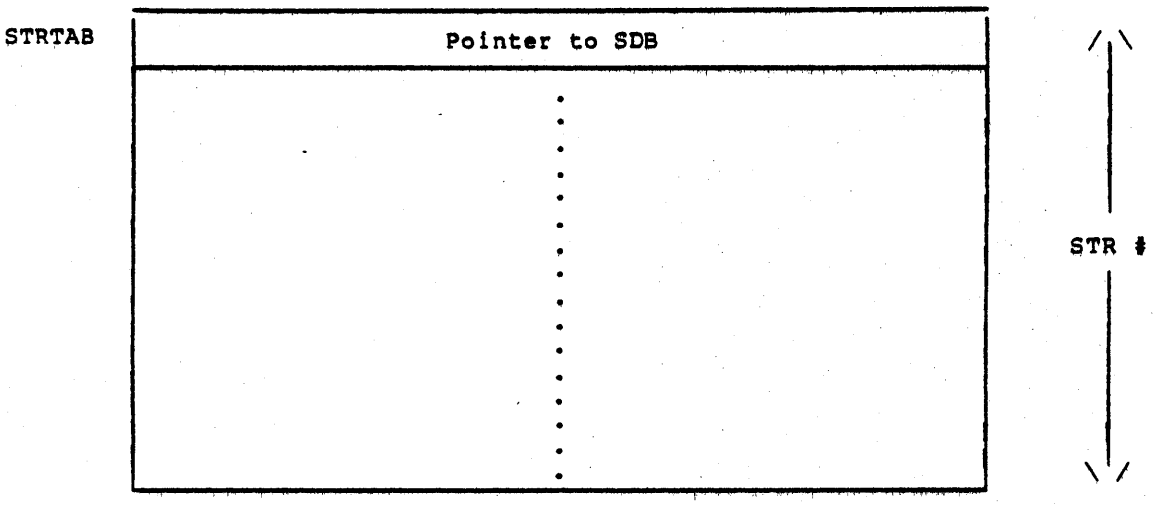
Name: STRTAB

Description: Structure Data Block Table. This table, indexed by structure number, contains pointers to each structure data block in the system.

Defined in: STG

Referenced by: DSKALC, PHYSIO

Format



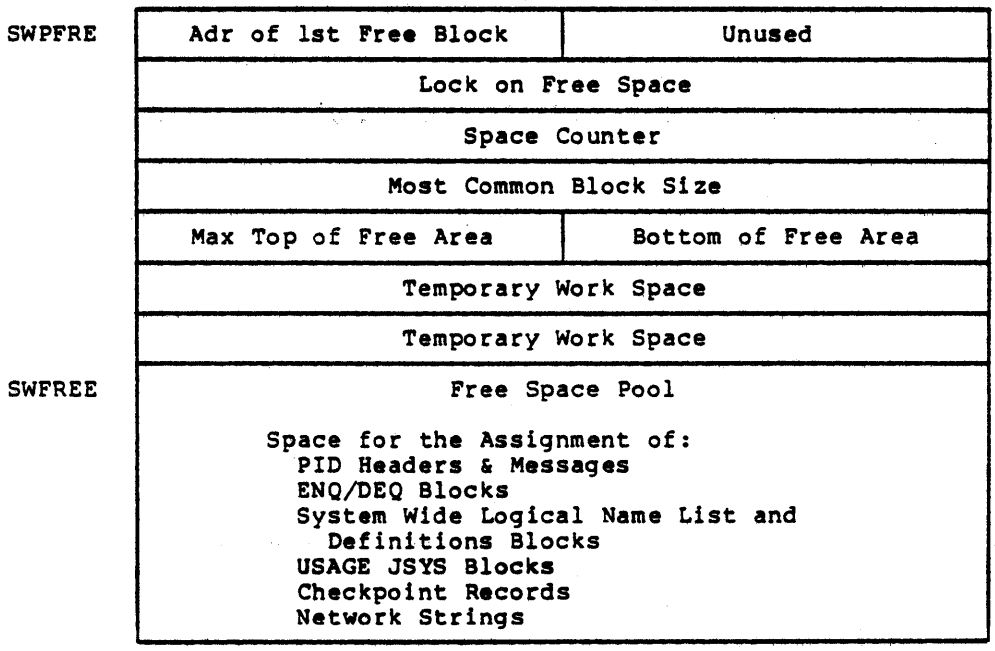
Name: SWAP-FREE-SPACE

Description: Swappable Free Space Pool Format. This table describes the header area that is used in the assignment and deassignment of swappable free space (by ASGFRE) and the usage of this space when assigned.

Defined in: STG

Referenced by: IPCF, LOGNAM

FORMAT



Name: SYNMTB

Description: System Logical Name Table: This table contains pointers to the initial ASCIZ strings for the system logical names.

Defined in: STG

Referenced by: LOGNAM

FORMAT

SYNMTB	XWD[ASCIZ/SYS/],[ASCIZ/PS:<SUBSYS>/]
	XWD[ASCIZ/HLP/],[ASCIZ/SYS:/]
	XWD[ASCIZ/SYSTEM,][ASCIZ/PS:<NEW-SYSTEM,PS:<SYSTEM>/]

Name: SYS-STARTUP-VECTORS

Description: Startup Transfer Vectors. This table, in resident locations 140-147, contains the startup vectors for the monitor as well as vectors to enter EDDT.

Defined in: STG

Referenced by: STG, POSTLD

Format

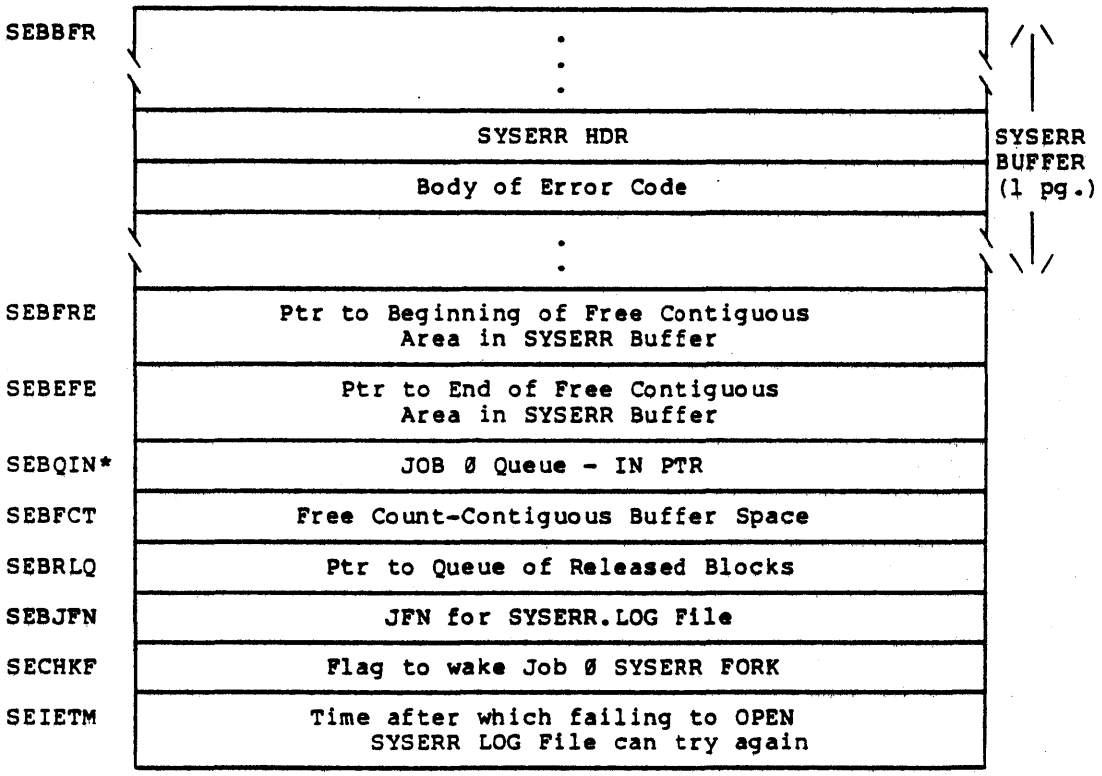
EVDDT=140	JRST DDTX	(EDDT)
	JRST SYSDDT	(Reset and go to EDDT)
EVDDT2=142	JRST DDTX	(Copy of EDDT in case other clobbered)
EVDDT2=143	JRST SYSLOD	(Initialize disk file system)
	XPCW RLODPC	(Keep alive execute address)
EVRST=145	JRST SYSRST	(Restart)
EVLDGO=146	JRST SYSGO	(Reload and start)
EVGO=147	JRST SYSGO1	(Start)

Name: SYSERR-STORAGE-AREA

Description: SYSERR STORAGE AREA. This area contains the buffer for all SYSERR error blocks which are later written by JOBØ into the SYSERR.LOG file. In and out pointers into the buffer area are maintained for JOBØ as well as pointers to the free and released SYSERR blocks.

Defined in: STG

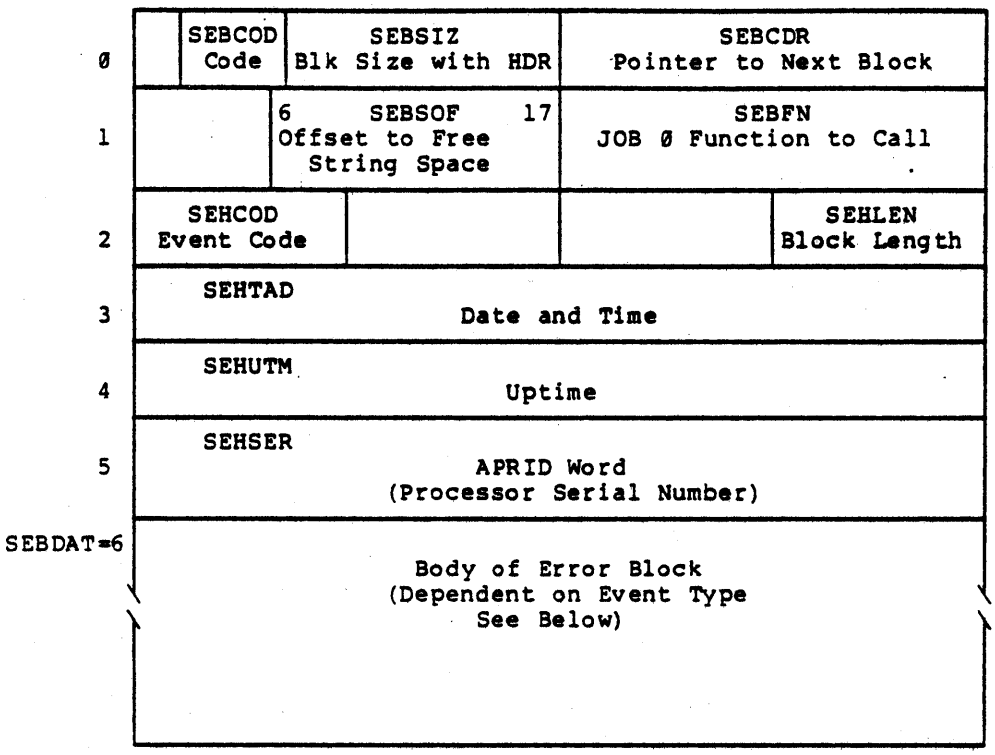
Referenced by: SYSERR

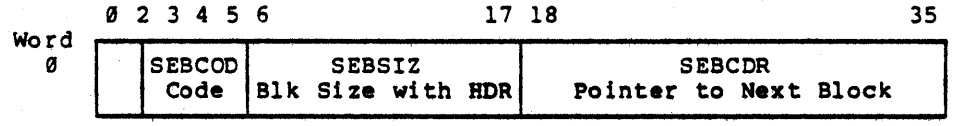


* Although the In-pointer is in this storage area the corresponding Out-pointer in SEBQUO is in a fixed place in lower core (i.e., location 24), so JOB Ø can queue up a BUGHLT block after a crash. One can examine the last SYSERR block by adding to the right half of the contents of SEBQUO, SEBDAT plus offset into SYSERR block.

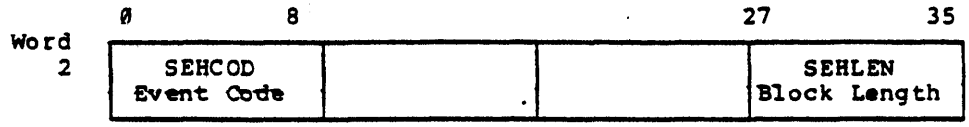
SEBBFR: SYSERR BUFFER BLOCKS

SYSERR BLOCK FORMAT





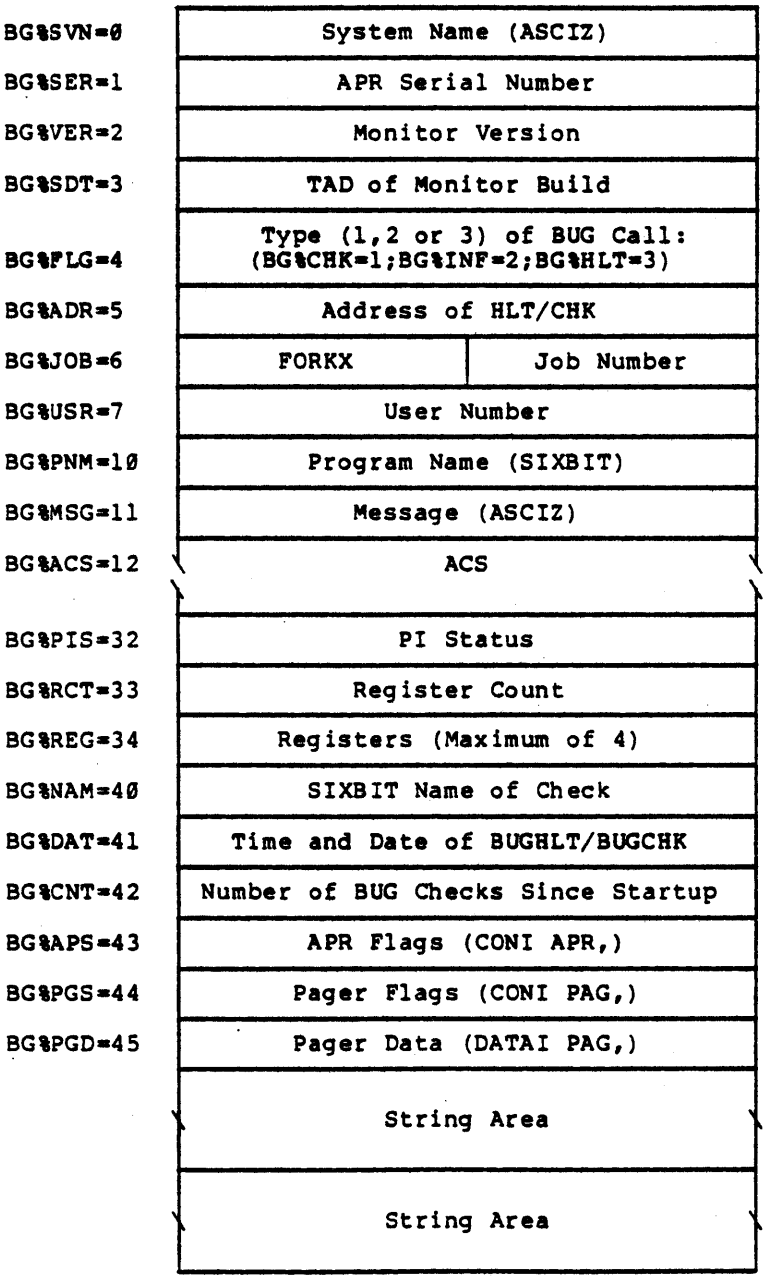
Bits	Pointer	Meaning
3-5	SEBCOD	State Code SBCFRE=0 on Free List SBCREL=1 Released SBCACT=2 Active
6-17	SEBSIZ	Block Size Including Header
18-35	SEBCDR	Pointer to Next in List



Bits	Pointer	Meaning
0-8	SEHCOD	Event Code (i.e., Block Type) SEC%RL=101 System Reload SEC%BG=102 BUGHLT/BUGCHK/BUGINF SEC%FE=130 Front End Error SEC%11=131 F.E. Reload Entry (Gives -11 Reboot Info.) SEC%PT=160 Processor Parity Trap SEC%PI=161 Processor Parity Intrp. SEC%MB=111 Massbus Device Error
27-35	SEHLEN	Block Length (Including Header) RL%LEN - System Reload Block Length BG%LEN - BUGHLT/CHK/INF Block Length FE%LEN - F.E. Errors Blk Length R1%LEN - F.E. Reload Entry Blk Length PT%LEN - Proc. Parity Trap Blk Length PI%LEN - Proc. Parity Interrupt Blk Lgh MB%LEN - Massbus Dev. Err Blk Length

Word 6 to End (Body of Error Block - Dependent on Event Type)
Event type 102

BUGHLT/CHK/INF Error Block Data



Event Type 101
System Reloaded Error Block Data

RL%SVN=0	ASCII Byte Pointer to System Name
RL%STD=1	Time of System Build (Univ. Format)
RL%VER=2	System Version Number
RL%SER=3	APR Serial Number
RL%OPR=4	ASCII Byte Pointer to "Why Reload"
RL%HLT=5	BUGHLT Address (if Auto-Reloaded)
RL%FLG=6	Flags
	Monitor Name (Text)
	"Why Reload" Answer String (Text)

RL%LEN=61

Event Type 130
Front End Errors Data Block

FE%FJB=0	Fork Number,,Job Number
FE%DIR=1	Directory Numbers
FE%ID=2	Front End Software Version
FE%NAM=3	SIXBIT Name of Program
FE%DEV=4	Protocol Device Code (1B0=Unknown)
FE%PTR=5	-Length of Data,,Start of Data
FE%DTE=6	DTE Number
FE%BYT=7	# of -11 Bytes in the Message

FE%LEN=10

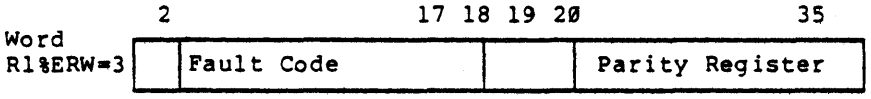
Event Type 131
Front End Reload ERROR BLOCK DATA

R1%NUM=0	-11 Number
R1%STS=1	Reload Status Bits
R1%FNM=2	File Name Pointer
R1%ERW=3	-11 Error Word
	String Area (^D20 words)

R1%LEN=30

Word
R1%STS=1

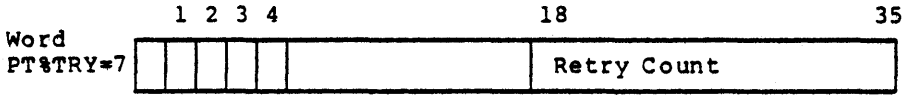
Symbol	Bits	Contents
.R1GTF	0	GTJFN failed for Dump File
.R1OPF	1	OPENF failed for Dump File
.R1DPF	2	Dump failed
.R110E	3	To -10 Error on Dump
.R111E	4	To -11 Error on Dump
.R1ASF	5	ASGPAG failed on Dump
.R1RLF	6	Reload failed
.R1PDF	7	-11 didn't Power Down
.R1POF	8	-11 didn't Power Up
.R1RMF	9	ROM did not ACKnowledge the -10
.R1BSF	10	-11 Boot Program didn't make it to the -11
.R1NRL	11	-11 took more than 1 Min. to Reload.
.R1RTC	33-35	Retry Count



Symbol	Bits	Contents
	2-17	Fault code as 3 RAD50 characters
	20-35	Parity Register Valid Only if it is Nonzero

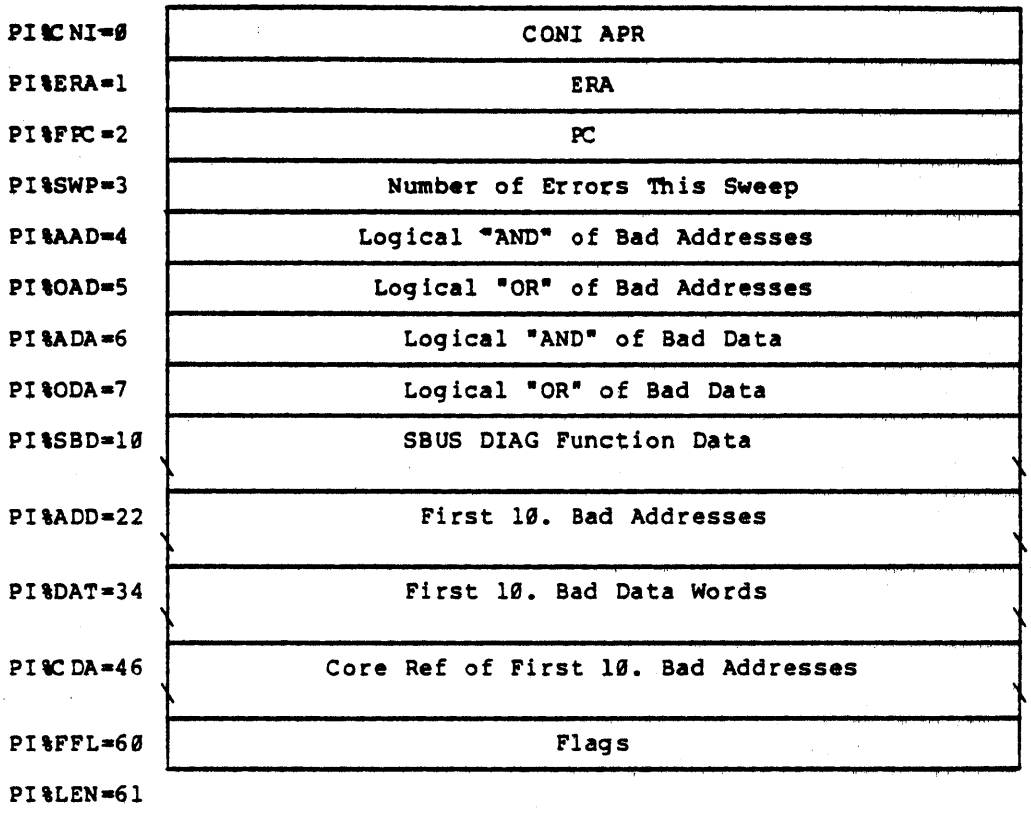
Event Type 160
Processor Parity Trap Error Block Data

PT%PFW=0	Page Fail Word	
PT%BDW=1	Bad Data Word	
PT%GDW=2	Good Data Word	
PT%USR=3	User Number	
PT%JOB=4	FORKX	JOBN
PT%PGM=5	Program Name (SIXBIT)	
PT%PMA=6	Physical Memory Address	
PT%TRY=7	Flags	Retry Count
PT%LEN=10		



Symbol	Bits	Contents
PT%HRO	1	Hard Error
PT%CCP	2	Cache Failure
PT%CCH	3	Cache in Use
PT%ESW	4	Error on Sweep to Core
	18-35	Retry Count

Event Type 161
Processor Parity Interrupt Error Data Block



Event Type 111
MASS BUS DEV Error Data Block

MB%NAM=0	Device Name (if available)
MB%VID=1	Volume ID (SIXBIT)
MB%TYP=2	Channel,,Device Type - See PHYPAR
MB%LOC=3	Location of Error - Sector or File,,Record
MB%FES=4	Final Error State - Device Dependant
MB%CNI=5	CONI Initial
MB%CIF=6	CONI Final
MB%SEK=7	Number of Seeks
MB%RED=10	Number of Blocks/Frames Read
MB%WRT=11	Number of Blocks/Frames Written
MB%FIL=12	Filename (Pointer)
MB%USR=13	User Making Request (Pointer)
MB%PGM=14	Program Running
MB%D1I=15	DATAI PTCR Initial
MB%D1F=16	DATAI PTCR Final
MB%D2I=17	DATAI PBAR Initial
MB%D2F=20	DATAI PBAR Final
MB%UDB=21	Unit Data Block for JOB 0 BAT Blocks
MB%IRS=22	IORB Status Word, IS.ERR if Hard (See PHYPAR)

MB%SRE=23	Soft Read Errors
MB%SWE=24	Soft Write Errors
MB%HRE=25	Hard Read Errors
MB%HWE=26	Hard Write Errors
MB%PS1=27	Position, CYL if Disk, File if Tape
MB%PS2=30	SURF/SEC or Record
MB%CS0=31	Channel Logout 0
MB%CS1=32	Channel Logout 1
MB%CS2=33	Channel Logout 2
MB%CC1=34	First CCW
MB%CC2=35	Second CCW
MB%MPE=36	Count of MPE
MB%NXM=37	Count of NXM
MB%FEC=40	Final Error Count
MB%CAD=41	Channel Address
MB%UAD=42	Unit Address
MB%SPE=43	Soft Positioning Errors
MB%HPE=44	Hard Positioning Errors
MB%OVR=45	Overruns
MB%ICR=46	Initial TCR
MB%REG=47	Units Massbus Registers in order with their: Final Contents,, Initial Error Contents
MB%LEN=67	

Name: TT-LINE-DYN-DATA-BLK

Description: Teletype Line Dynamic Data Block. This block pointed to by the line's entry in TTACTL, holds line specific data and is built when the line becomes active. It is deallocated when the line becomes inactive.

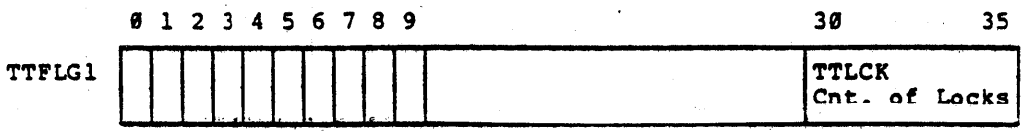
There are two shortened forms of the dynamic data block, one used for a SENDALL type of message and the other for sending a "ding" when any character but CTRL/C is typed on an inactive line.

Defined in: TTYSRV

Referenced by: TTYSRV

TTFLG1=0	Flags				
TTDAT1=1	3 8 TLTYP Line Type	9 17 TTTYP Terminal Type	18 35 TINTL Internal Line Number (index into static data)		
TTSAL1=2	TSALC Send All Character Count				
TTSAL2=3	TSALP Send All Byte Pointer				
TTDEV=4	Device dependent word (See Device modules for definitions)				
TTBFRC=5	0 7 TOWRN Wake Up Count	8 12 TTNIN # of Input Bufs	13 17 TTNOU # of Output Bufs	18 26 TIMAX Max bytes In Input Buf	27 35 TOMAX Max bytes In Output Buf
TTOCT=6	Number of Characters in Output Buffer				
TTOOUT=7	Pointer for Removing Char from Output Buffer				
TTOIN=10	Pointer for Entering Char into Output Buffer				

TTDAT2=11	7 8 TYLMD TTY data mode for Last Inp Char.	9 17 TYLCH Last Charcter Removed from Input Buffer (For Backup Char JSYS)	TPWID Page width
TTICT=12	Number Characters in Input Buffer		
TTIOUT=13	Pointer for Removing Char from Input Buffer		
TTIIN=14	Pointer for Entering Char into Input Buffer		
FCMOD1=15	Control Character Output Control Words		
FCMOD2=16	Possible Values for each Char. (2 Bits/Char) CCNONE = 0 Send nothing CCIND = 1 Indicate via ^ CCSEND = 2 Send Actual Code CCSIM = 3 Simulate Format Action		
TTDPSI=17	Bit for Terminal Code Set if Deferred Interrupt		
TTPSI=20	Bit for Terminal Code Set if Interrupt		
TTLINK=21	Lines linked to (9 bits per line)		
TTLPOS=22	0 17 TPGPS Current Line Position in Page	18 35 TLNPS Current Charcter Position within Line	
TTFLGS=23	0 TOFLG ^O was typed	10 17 TPLEN page length	
TTFORK=24	TCJOB Controlling Job Number	TWFRK Fork Number in Input Wait on this Line	
TTFRK1=25		TTPFK Fork which is Top Fork of a SCTTY Tree (-1 if None)	



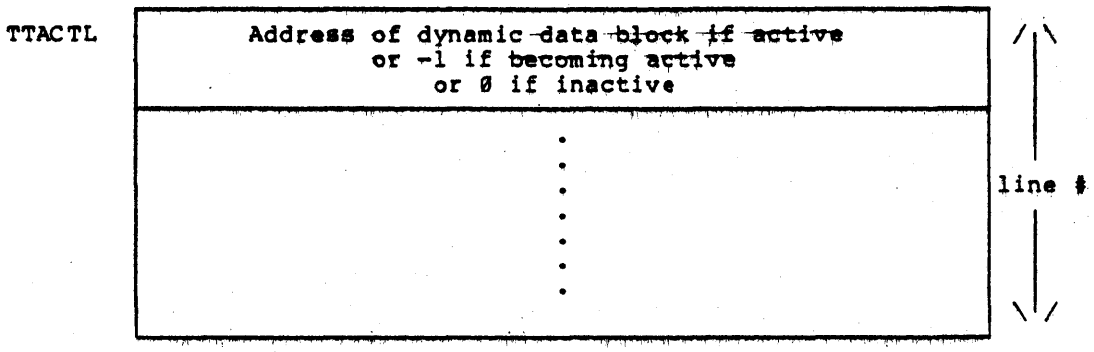
Symbol	Bits	Pointer	Contents
TT%SAL	0	TTSAL	Sendall being done to this line
TT%SHT	1	TTSHT	This is a short block
TT%MES	2	TTMES	This is a system message block
TT%OTP	3	TTOTP	Output is enroute to the line
TT%FWK	4	TTFWK	Forced wakeup
TT%SFG	5	TTSFG	CTRL/S was typed
TT%RFG	6	TTRFG	Repeat last character (BKJFN)
TT%WFG	7	TTWFG	Blocked on input
TT%PRM	8	TTPRM	Don't deallocate dynamic data
TT%BAC	9	TTBAC	Permanent and becoming active
TT%LCK	30-35	TTLCK	Count of locks on this block

Name: TTACTL

Description: Teletype Active Line Table. This resident table, indexed by line #, contains a pointer to each active line's dynamic data block.

Defined in: STG

Referenced by: MEXEC, TTYSRV



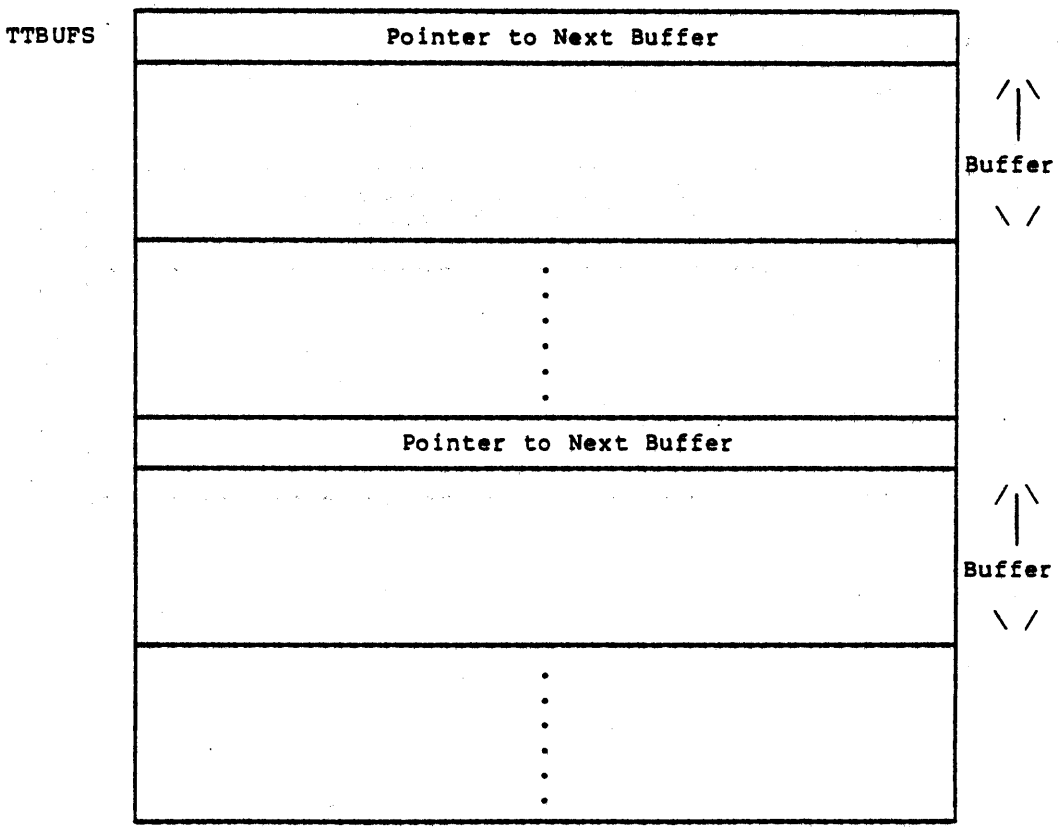
Name: TTBUFS

Description: Teletype Buffers. This storage area contains the input and output buffers for each line (TTY and PTY) on the system. Input and output pointers to each buffer are kept in the line's dynamic data block. These buffers are fixed length and are assigned on demand. When there is no character activity, the buffers are deassigned.

Defined in: STG

Referenced by: TTYSRV

Format



Note: The free buffers are linked and are pointed to by TTFREB.

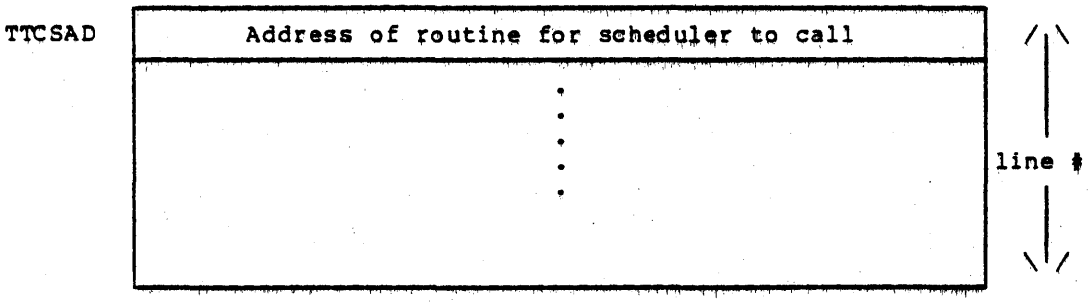
Name: TTCSAD

Description: Terminal Call Special Request Address Table. This resident table, indexed by line number, is used to hold the dispatch address of a scheduler routine for a special line request.

Special line requests are made when the DTEQ routine is unable to obtain space for a packet and cannot block to wait for the space. (i.e. process is NOSKED, or request made at interrupt or scheduler level). A special line request is made so that a packet will be queued later by the Scheduler. (See Table, TTCSTM).

Defined in: STG

Referenced by: TTYSRV, TROUT

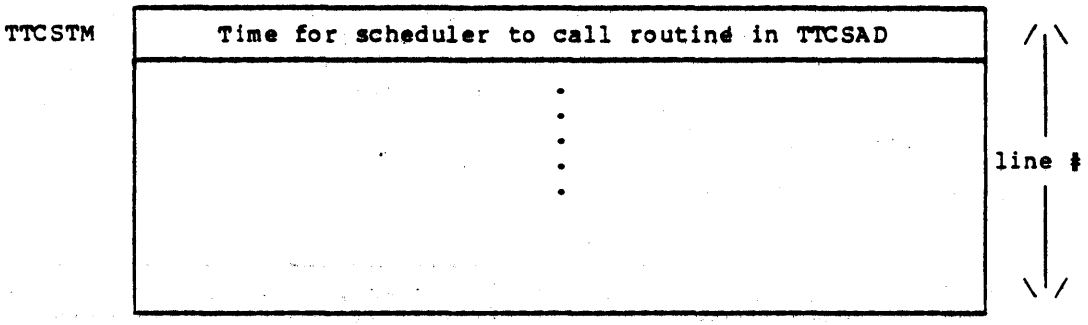


Name: TTCSTM

Description: Terminal Call Special Request Time Table. This resident table, indexed by line number, parallels the TTCSAD table and holds the time the Scheduler is to call the special request routine in TTCSAD.

Defined in: STG

Referenced by: TTYSRV, TTIME



Name: TTLINV

Definition: Terminal Type Line Vector Table. This vector table, indexed by device type, gives the device table addresses of the form, TTXXVT. These tables hold the device specific vectors for device functions. (See TTXXVT Table).

Defined in: TTYSRV

Referenced by: TTYSRV

Format

TTLINV:

IFIW !	TTFEVT	FE Vector Table
IFIW !	TTMCVT*	MCB Vector Table
IFIW !	TTPTVT	PTY Vector Table
IFIW !	TTDCVT*	DC10 Vector Table
IFIW !	TTNTVT	Network Vector Table
IFIW !	TTDZVT	DZ11 Vector Table

NOTE

If a terminal type does not exist, its corresponding vector table address in TTLINV will be set equal to the vector table address, TTDMVT. This table vectors to routines which BUGHLT.

* Are currently not in use.

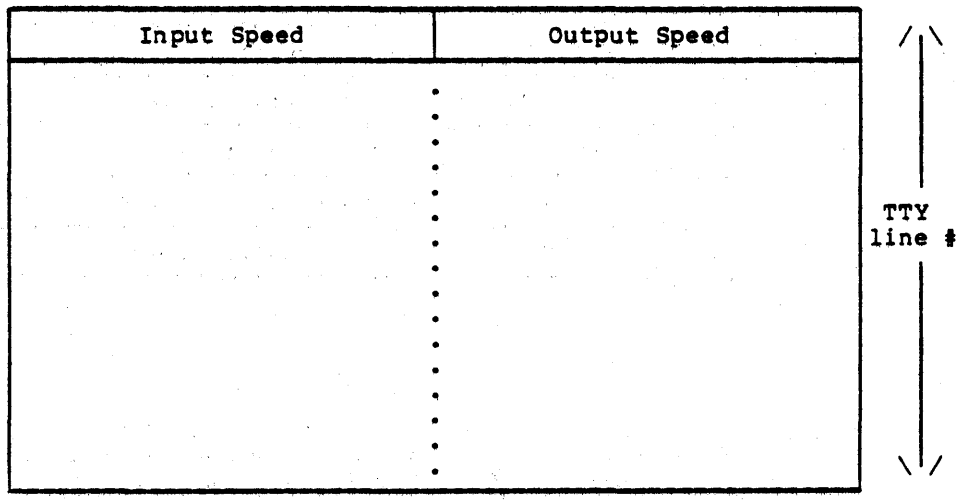
Name: TTSPWD

Description: Terminal Speed Word Table. This nonresident table, indexed by line number (TTYs only), contains the terminal speeds for each terminal.

Defined in: STG

Referenced by: TTYSRV

Format



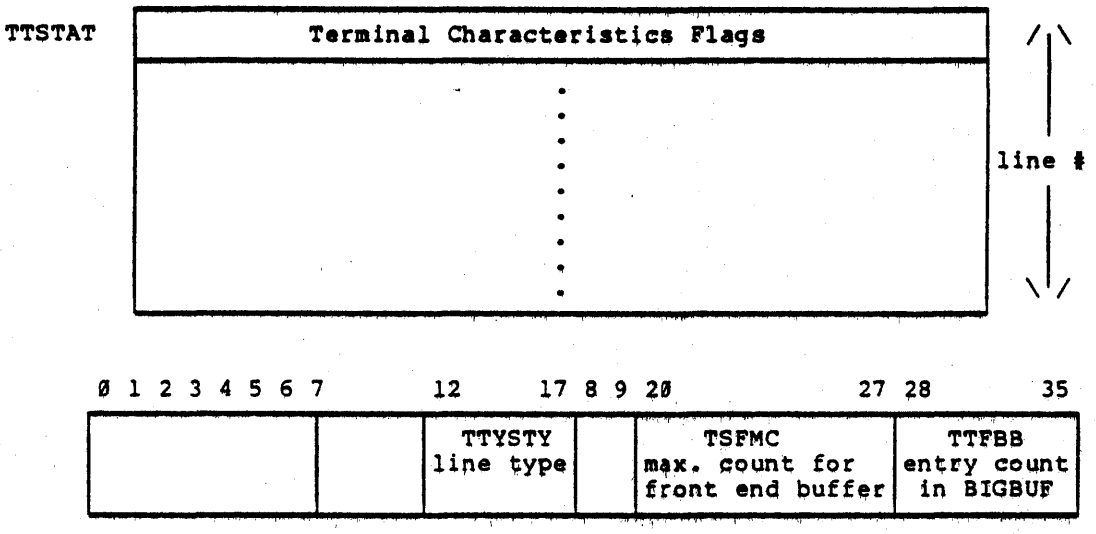
Name: TTSTAT

Description: Teletype Status Table. This resident table, indexed by line #, contains the terminal characteristic flags

Defined in: STG

Referenced by: TTYSRV

Format



Symbol	Bits	Pointer	Contents
TT%FEM	0	TTFEM	Line is remote
TT%NTS	1	TTNTS	Don't send system mesg.
TT%FXO	2	TTFXO	Line needs XON
TT%CON	3	TTCON	Carrier is on
TT%FSP	4	TTFSP	Line needs speed set
TT%FXF	5	TTFXF	Line needs XOF
TT%IGI	6	TTIGI	Ignore input when line is inactive
TT%AUT	7	TTAUT	Line is auto-speed
	12-17	TTYSTY	Line type which yields the offset into the TTLINV table
	20-27	TSFMC	Max count for front end buffer
	28-35	TTFBB	Entry count in Big Buf

Name: TTXXVT

Description: Teletype Device Specific Vector Table. TTXXVT is the vector table (in generalized format) for data and routine addresses for device XX. The offsets given below can be used when referencing a device specific vector table.

Teletype service consists of the module - TTYSRV - which contains all device-independent code, and one module for each line type. The latter modules contain all the device-dependent code for their line type.

Each device-dependent module, of name TTXXDV, has a transfer vector table of the name TTXXVT, where XX is unique for each line type (i.e., XX = FE/DZ/DC/MC/NT/PT); (See TTLINV Table).

Defined in: TTYSRV

Referenced by: TTYSRV

DDLEN=0	Length of dynamic data for this type
TT1LIN=1	First line of this type/-1 no lines
TTVT00=2	Initialize tables at system startup
TTVT01=3	Activate lines at startup or restart
TTVT02=4	Clear output buffer
TTVT03=5	Set line speed
TTVT04=6	Read line speed
TTVT05=7	Set terminal/non-terminal status
TTVT06=10	Read terminal/non-terminal status
TTVT07=11	STO JSYS
TTVT08=12	STPAR JSYS
TTVT09=13	CKPHYT - see if physical terminal
TTVT10=14	Process XON from terminal
TTVT11=15	Deassign TTY Data Base
TTVT12=16	TCOUT - add parity to character
TTVT13=17	Start output to line
TTVT14=20	Send XOFF to terminal

TTVT15=21	Send XON to terminal
TTVT16=22	TTCH7 - Process TTCS words
TTVT17=23	Handle carrier/on
TTVT18=24	Handle carrier/off
TTVT19=25	Hangup, reactivate remote line
TTVT20=26	Process XOFF from terminal
TTVT21=27	Handle CTRL/C from inactive line
TTVT22=30	BIGSTO - Store character in TTBBUF
TTVT23=31	TTSND - Send character to line
TTVT24=32	Detach job on this line
TTVT25=33	Handle overflow of TTBBUF
TTVT26=34	Remove character from TTBBUF
TTVT27=35	Do TTMSG for one line
TTVT28=36	Enable/Disable Datasets
TTVT29=37	TTCH7 after emptying TTBBUF
TTVT30=40	Clear input buffer
TTVT31=41	DOBE
TTVT32=42	Input GA
TTVT33=43	Set Initial Values for a line
TTVT34=44	SOBE
TTVT35=45	Wakeup if output buffer empty
TTVT36==46	Sendall for one line
TTVT37==47	Sendall for all lines
TTVT38==50	Adjust wakeup class
TTVTMX==51	Maximum number of vector entries

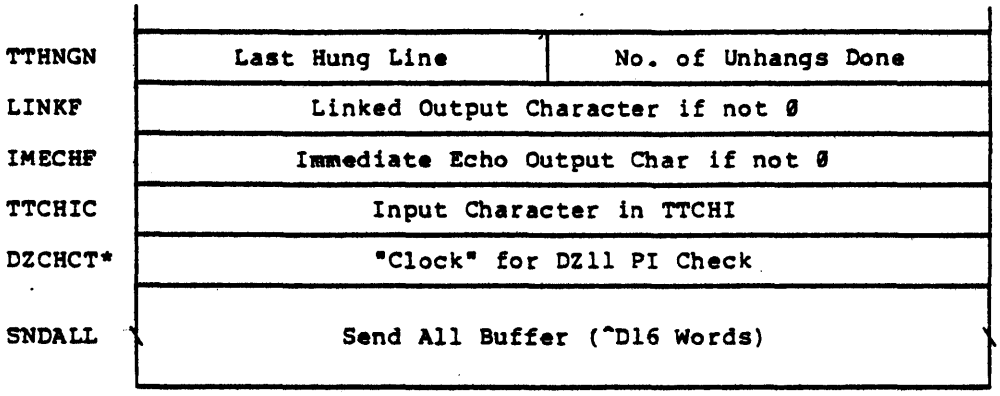
Name: TTY-STORAGE-AREA

Description: Teletype Storage Area. This resident area contains hung and special line information, the Big Buffer, and information about the Big Buffer. (See TT-LINE-DYN-DATA-BLK, TTACTL, TTBUFS, TTCSAD, TTCSTM, TTLINV, TTSPWD, TTSTAT, and TTXXVT Tables).

Defined in: STG

Referenced by: TTYSRV

CTYNIT	Unit No. on the .FEDLS Device by which the Front End Knows the CTY
TCOERR	TCOUT Sets this if Fails in Scheduler Context
SALCNT	Count of Lines Doing SENDALL
TTFREC	Count of Free Buffers
TTFREB	List of Free Buffers
TTBIGI	Input Index into Big Buffer
TTBIGO	Output Index into Big Buffer
TTBIGC	Char Count in Big Buffer
TTBBUF	Big Buffer (^D128 Words) Storage for all TTY Input Chars. Before Being Placed Into Individual Input Line Buffers in TTBUFS Area
TTQCNT	Count of Special Line Items
TTCQLN	Control of Current Line Number
TTHNGL	Line Being Examined for Hung
TTHNGT	Time at Which Line Will be Defined as Hung



* Only assembled if the assembly flag, SMFLG, does not equal zero. (i.e., Have a 2020 System).

Name: UDB

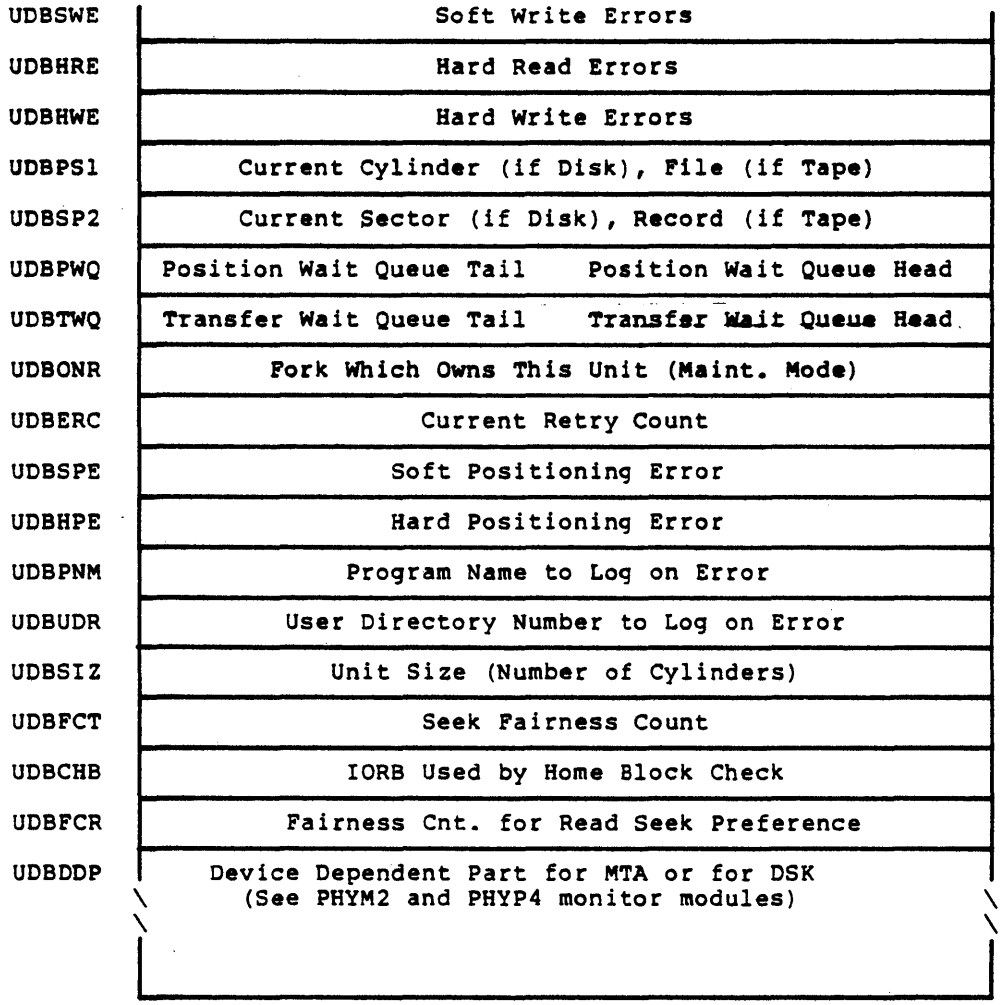
Description: Unit Data Block. This block, one per unit, contains information about the current activity on the unit.

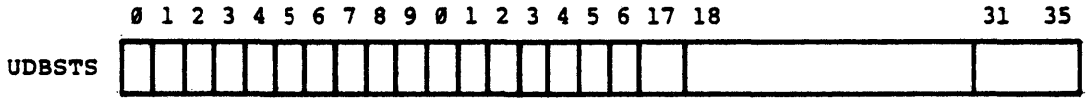
Defined in: PHYPAR

Referenced by: PHYSIO

Format

UDBSTS=0	Status and Configuration Information	
UDBMBW=1	Memory Bandwidth Scheduling Information	
DBODT	Overdue Timer for Seeks and the Like	
UDBERR	Error Recovery Status Word	
UDBERP	Error Reporting Work Area if Nonzero	
UDBDSP	Unit Routine Main Entry Dispatch	
UDBCDB	Secondary CDB	Primary CDB
UDBADR	Secondary Unit Address	Primary Unit Address
UDBAKA	Current CDB	Current Chain Address
UDBVID	Volume ID	
UDBSTR	Pointer to Structure Data Block	
UDBKDB	Pointer to KDB, if any	
UDBDSN	Drive Serial Number	
UDBSEK	Seeks	
UDBRED	Reads (Sectors if Disk, Frames if Tape)	
UDBWRT	Writes (Sectors if Disk, Frames if Tape)	
UDBSRE	Soft Read Errors	


UDBSWE
Soft Write Errors
UDBHRE
Hard Read Errors
UDBHWE
Hard Write Errors
UDBPS1
Current Cylinder (if Disk), File (if Tape)
UDBSP2
Current Sector (if Disk), Record (if Tape)
UDBPWQ
Position Wait Queue Tail
Position Wait Queue Head
UDBTWQ
Transfer Wait Queue Tail
Transfer Wait Queue Head
UDBONR
Fork Which Owns This Unit (Maint. Mode)
UDBERC
Current Retry Count
UDBSPE
Soft Positioning Error
UDBHPE
Hard Positioning Error
UDBPNM
Program Name to Log on Error
UDBUDR
User Directory Number to Log on Error
UDBSIZ
Unit Size (Number of Cylinders)
UDBFCT
Seek Fairness Count
UDBCHB
IORB Used by Home Block Check
UDBFCR
Fairness Cnt. for Read Seek Preference
UDBDDP
Device Dependent Part for MTA or for DSK
(See PHYM2 and PHYP4 monitor modules)



Symbol	Bits	Pointer	Content
US.OFS	0	USOFL	Offline or unsafe
US.CHB	1		Check home blocks before any normal I/O
US.POS	2		Positioning in progress
US.ACT	3		Active
US.BAT	4		Bad blocks on this unit
US.BLK	5		Lock bit for this units BAT blocks
US.PGM	6		Dual port switch in (A or B) (RP04,5,6)
US.MAI	7		Unit is in MAINT mode
US.MRQ	8		MAINT mode is requested on this unit
US.BOT	9		Unit is at BOT
US.REW	10		Unit is rewinding
US.WLK	11		Unit is write locked
US.MAL	12		MAINT mode allowed on this unit
US.OIR	13		Operator intervention required. Set at interrupt level, checked at SCHED.
US.OMS	14		Once a minute message to operator. Used in conjunction with US.OIR
US.PRQ	15		Positioning required on this unit
US.TAP	16		Tape type device
US.IDB	17		Tape - IDB seen on previous operation
	31 -35	USTYP	Unit Type

Type Code for USTYP

Symbol	Code	Unit
.UTRP4	1	RP04
.UTRS4	2	RS04
.UTT16	3	TU16
.UTTM2	4	TM02 (as unit)
.UTRP5	5	RP05
.UTRP6	6	RP06

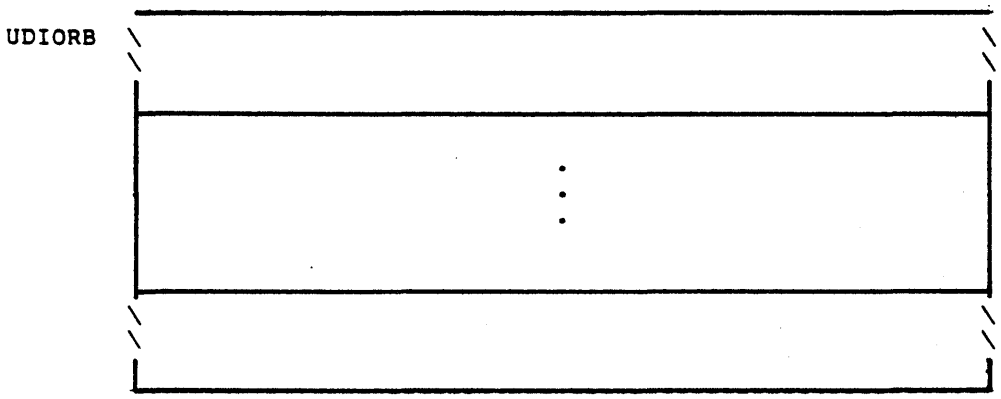
Name: UDIORB

Description: UDSKIO IORB Pool. The free IORBS are linked together in UDIORB and this list is pointed to by UIOLST.

Defined in: STG

Referenced by: PHYSIO

Format



Name: UDS

Description: Unit Dispatch Service Routine Table. This table, one per unit type, contains vectored addresses to unit dependent functions, and is given in its generalized form. The specific unit dispatch tables are RP4DSP (in PHYP4) for the disk device, and TM2DSP (in PHYM2) for the magtape device. See PHYPAR for definitions of arguments given and returned on calls to these unit routines.

Defined in: PHYPAR

Referenced by: PHYSIO, PHYH2, PHYM2(MTA), PHYP4(DSK), STG

Format

UDSINI=0	Initialize
UDSSIO=1	Start I/O on an IORB, skips if O.K.
UDSINT=2	Interrupt Routine (called on interrupts for XFER done)
UDSERR=3	Initiate Error Retry (skips if no more retrys)
UDSHNG=4	Hung Reset (called from TIMER to reset hung devices)
UDSCNV=5	Convert Unit Linear Address to CYL, SURF, SEC
UDSLTM=6	Return Latency or Best Request
UDSPOS=7	Start Positioning on IORB (skips if O.K.)
UDSATN=10	Attention Interrupt
UDSPRQ=11	Skip if Positioning Required
UDSSTK=12	Stack Second Command, Skip if OK

Name: UPT

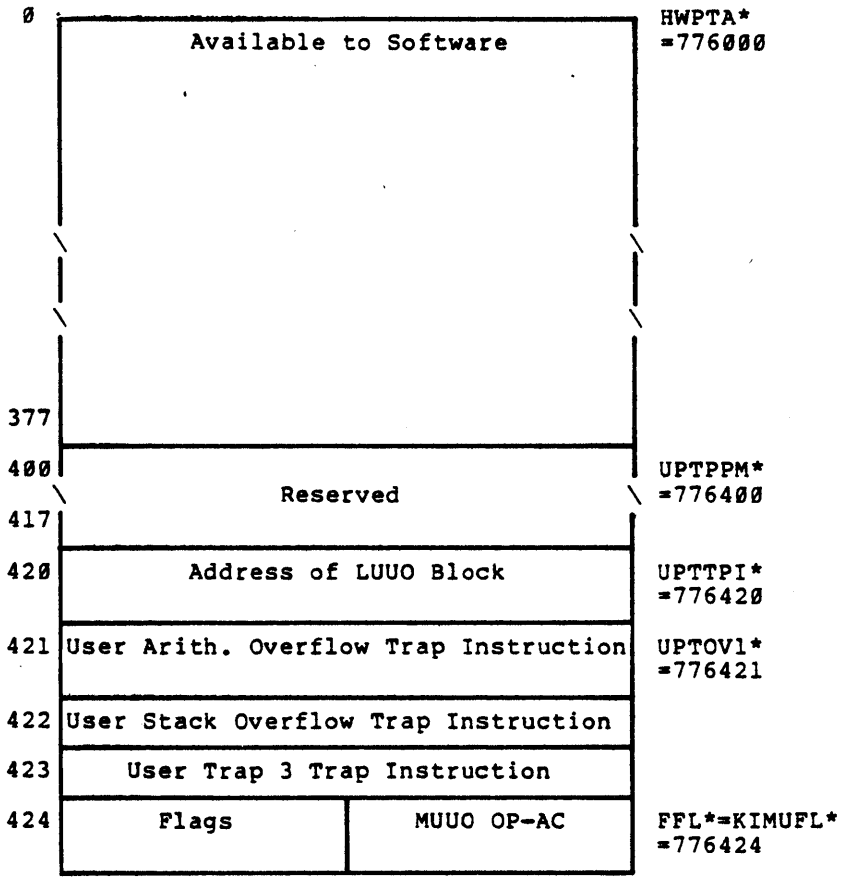
Description: User Process Table. A one page User Process Table is associated with the Scheduler and with each fork in the system. (Those associated with forks may be swapped out with the fork.) However, there is only one UPT known to the hardware/firmware at any one time. The UPT known is the one whose address is pointed to by the hardware User Base Register (UBR), which is set-up when a process is chosen to run.

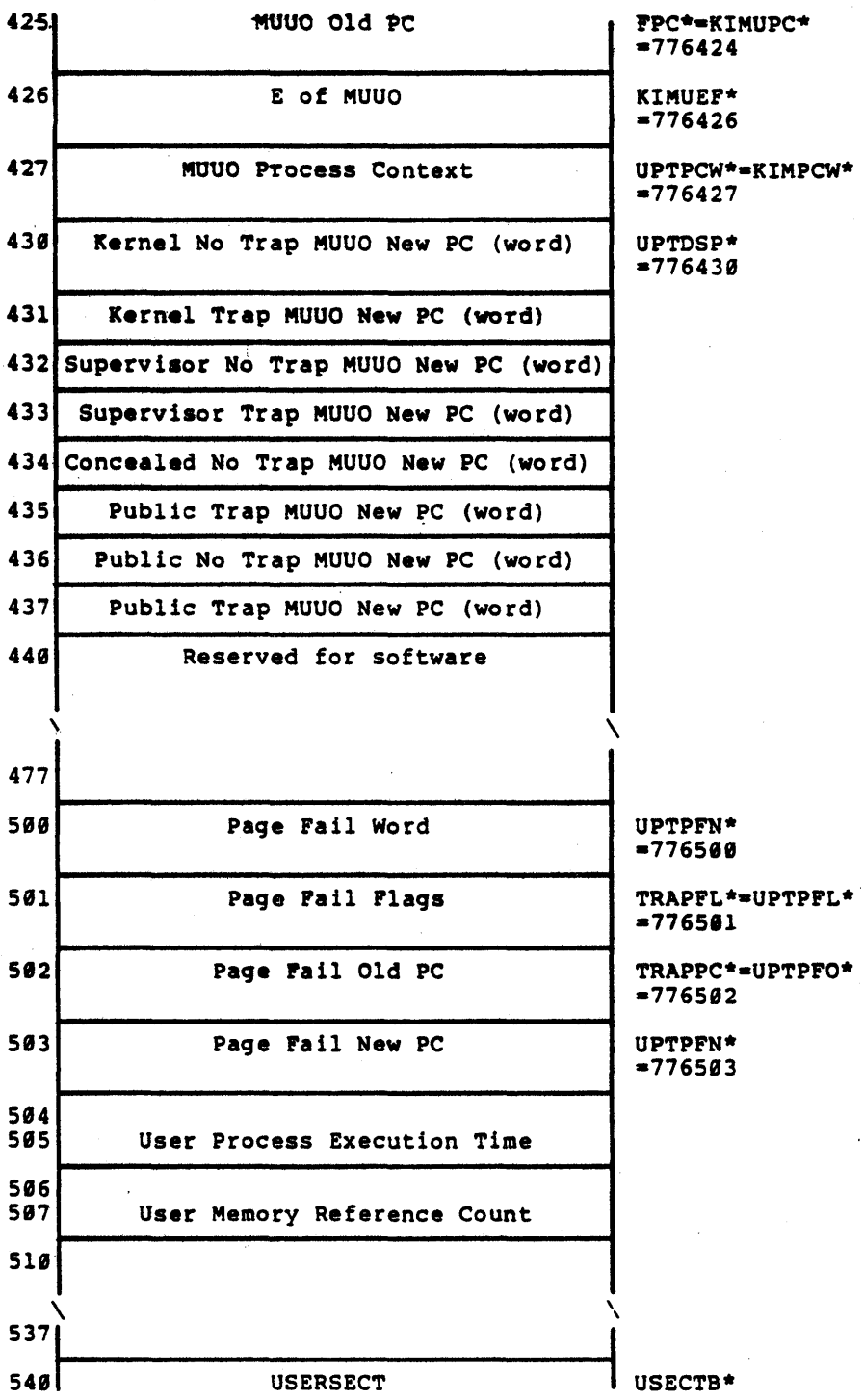
The UPT contains the dispatch address for process events (i.e., traps) and the user's section map table.

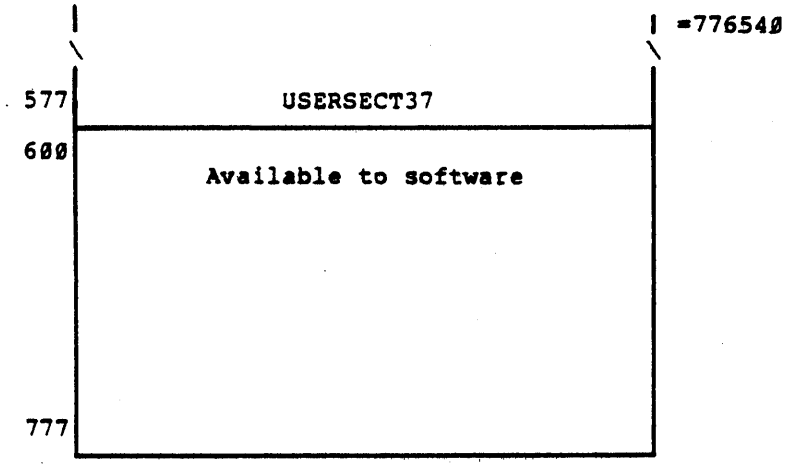
Defined In: APRSRV

Referenced by: APRSRV, SCHED

FORMAT




425
MUUO Old PC
FPC*=KIMUPC*
=776424
426
E of MUUO
KIMUEF*
=776426
427
MUUO Process Context
UPTPCW*=KIMPCW*
=776427
430
Kernel No Trap MUUO New PC (word)
UPTDSP*
=776430
431
Kernel Trap MUUO New PC (word)
432
Supervisor No Trap MUUO New PC (word)
433
Supervisor Trap MUUO New PC (word)
434
Concealed No Trap MUUO New PC (word)
435
Public Trap MUUO New PC (word)
436
Public No Trap MUUO New PC (word)
437
Public Trap MUUO New PC (word)
440
Reserved for software
477
500
Page Fail Word
UPTPFN*
=776500
501
Page Fail Flags
TRAPFL*=UPTPFL*
=776501
502
Page Fail Old PC
TRAPPC*=UPTPFO*
=776502
503
Page Fail New PC
UPTPFN*
=776503
504
505
User Process Execution Time
506
507
User Memory Reference Count
510
537
540
USERSECT
USECTB*



Note: Approximately 1/4 of the UPT is used for hardware cells, leaving the rest available to software. The monitor currently uses this area to house the first page of the PSB table. (See PSB table description.)

* These are monitor virtual memory addresses and are used when the monitor wishes to reference the current fork's User Process Table.

Name: USER-PG-MAP-TBL

Description: User Page Map Table. This 512-word swappable table, holds or points to other tables that hold all of the mapping information needed by the firmware to translate user mode virtual addresses in a given section into physical memory addresses. It is pointed to by an entry in the forks' section table in its User Process Table (UPT). (See UPT table description.)

The User Page Map, indexed by a 9 bit virtual page number (1), contains either the storage address for the virtual page if the page exists (immediate pointer) or a pointer to where the storage address resides in another table (shared or indirect pointer). The storage address can be a memory, swapping area, or disk page address.

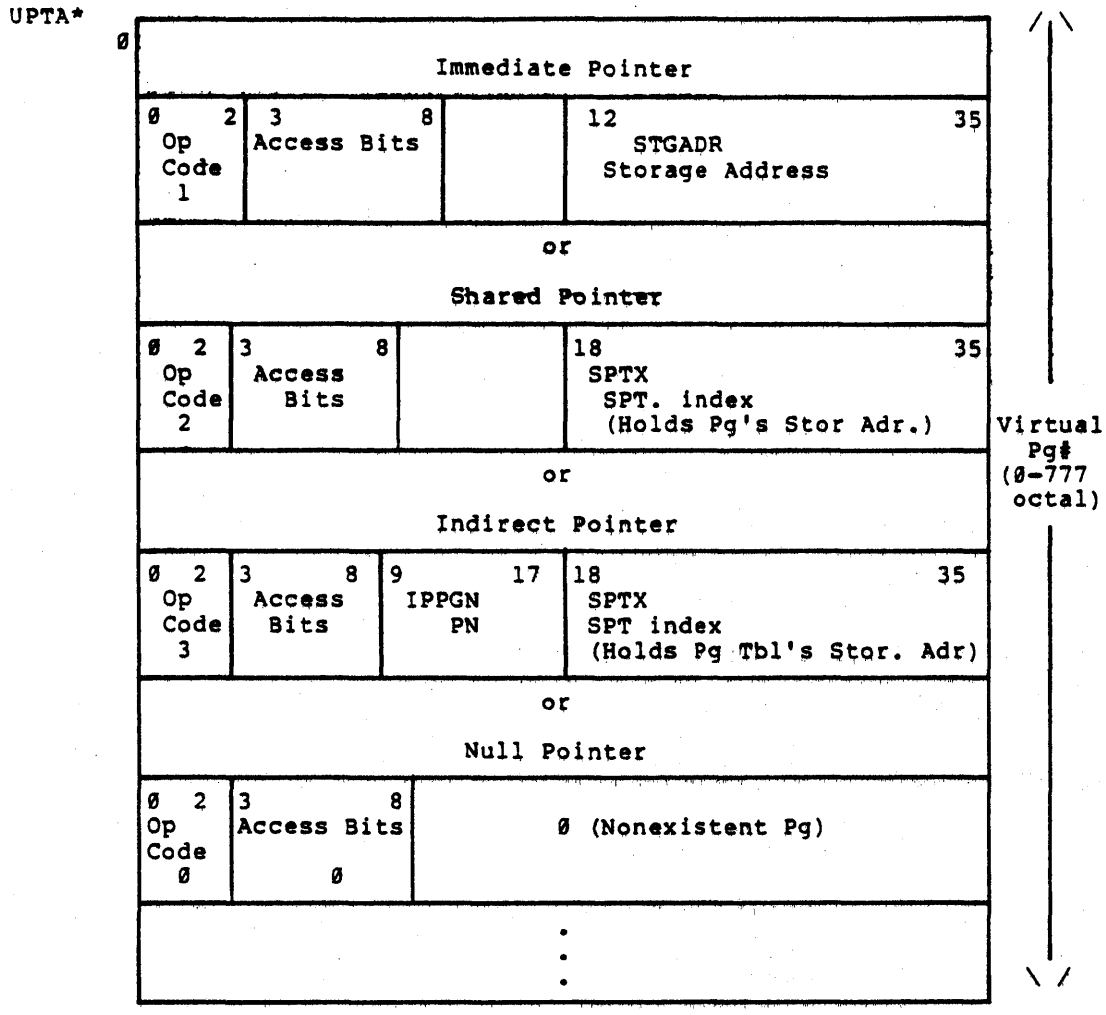
If the Storage address for the virtual page referenced by the process contains a memory page address (i.e., Storage Address Bits <12-17>=Ø), then the microcode, after copying this translation information along with the page's access bits into the CPU's Hardware Page Table (2), concatenates this memory page number with the index into the page to compose the complete physical address.

If the storage address for the virtual page referenced does not contain a memory address (i.e., Storage Address Bits <12-17> not equal to Ø), or the page is non-existant (i.e., Null Pointer word) or the page is being illegally accessed, the microcode will cause a page trap to the User Process Table (UPT). The monitor is then invoked to perform the analysis and resolution of the trap condition.

Defined in: PROLOG

Referenced by: DIAG, FORK, PAGEM, SCHED

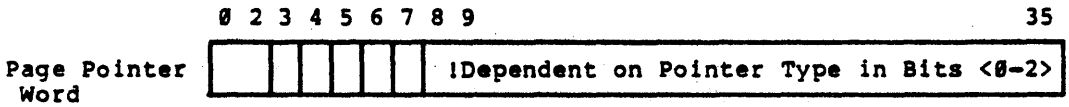
FORMAT



* UPTA is the monitor's symbol used when it wishes to reference the current user's page map table for section Ø.

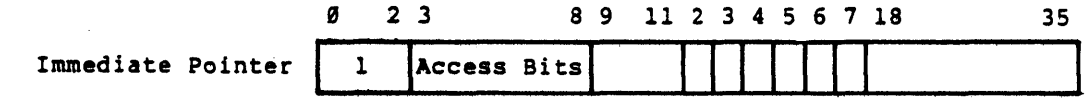
(1) A fork's virtual 18-bit address within a section is viewed as an address within a 512-word page, (i.e., virtual page number - 9 bits), Index into page 9-bits.

(2) This 512-word table is examined first by the microcode for the virtual page translation information. If not there, it then goes to the fork's UPTA in memory.



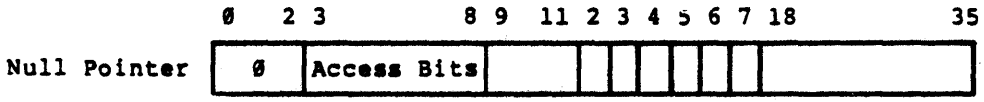
Symbol	Bits	Pointer	Meaning
PTRCOD	0-2		Operation Code for the page pointer type IMMCOD = 1 Immediate Pointer SHRCOD = 2 Share Pointer INDCOD = 3 Indirect
PTBUB	3		Public Bit
PTWR	4		Write Access
PTSOFT	5		Software Bit
PTCACH	6		Cache Bit
PTCPY	7		Copy-on-Write (Software Only)
PTLOK	8		Page is Locked (Immediate Pointer only)

Pointer Types:



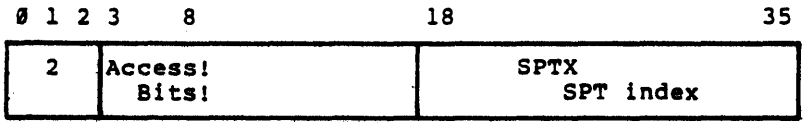
Symbol	Bits	Pointer	Meaning
	0-8		See above
	STORAGE ADDRESS		
	12-35	STGADR	Storage address (Interpretation follows)
NCORTM	12-17		Non-Core Test Mask yielding type of storage. Bits <12-17>=0 => Bits <18-35>=Memory Pg Adr. Bits <12-17> not equal 0 => Bits <18-35>=Drum/DSK Adr.
DSKAB	14		Storage address is a disk address
DSKNB	15		Temporary bit used with DSKAB to say that disk address is newly assigned.
DRMAB	16		Storage address is a drum address
DRMOB	17		Used with DRMAB to indicate that the swapping area has overflowed to the disk file system. (Since TOPS-20 currently uses only the disk file system for swapping, a drum storage address will always have bits 16 & 17 set.)

UAABC	17-35	Temporary bit used by the monitor's page trap handler when a copy-on-write page trap has occurred. If the page to be copied is a drum address, it will be faulted in before these bits are used, avoiding conflict over bit 17. These bits will signify to a lower level routine, SWPIN, that the page just gotten from the free list has no backup address and that it is to get a copy of another page.



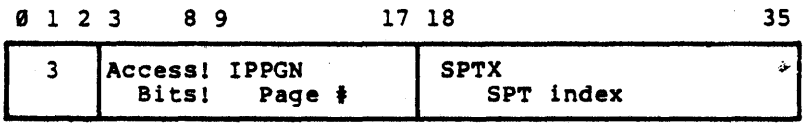
Symbol	Bits	Meaning
	0-8	These bits will have a value of 0 for the Null Pointer case.
UAAB	17	Temporary bit used by the monitor's page trap handler to say that the page has no assigned backup address on disk/drum.

Shared Pointer



Bits	Pointer	Meaning
0-8		See Above
18-35	SPTX	The SPT index is used to obtain from the SPT, the page's storage address.

Indirect Pointer



Bits	Pointer	Meaning
0-8		See Above
9-17	IPPGN	Page # whose value is used as an offset into the Page Table (pointed to by the SPT table address plus the SPT index in bits <18-35>) to obtain the page's translation information.
18-35	SPTX	The SPT index is used to obtain from the SPT the page table's storage address. The table's address plus the offset specified in bits <9-17> holds the virtual page's translation information.

Monitor Program Logic Manual

DIGITAL

The material in this document is for informational purposes and is subject to change without notice; it should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this document.

The software described in this document is furnished under a license and may only be used or copied in accordance with the terms of such license. Digital Equipment Corporation assumes no responsibility for the use or reliability of its software on equipment that is not supplied by Digital or its affiliated companies.

The following are trademarks of Digital Equipment Corporation, Maynard, Massachusetts:

COMPUTER LABS	COMTEX	DBMS-10
DBMS-11	DBMS-20	DDT
DEC	DECCOMM	DECsystem-10
DECSYSTEM-20	DECtape	DECUS
DIBOL	DIGITAL	EDUSYSTEM
FLIPCHIP	FOCAL	INDAC
LAB-8	MASSBUS	OMNIBUS
OS/8	PDP	PHA
RSTS	RSX	TYPESET-8
TYPESET-10	TYPESET-11	TYPESET-20
UNIBUS	DECSYSTEM-2020	

<<For Internal Use Only>>

DECSYSTEM-20 MONITOR FLOWCHARTS

I. Scheduler

II. Page Fault Handling

III. JSYS Calls - Device Independent Level

IV. JSYS Calls - Disk Dependent Level

V. JSYS Calls - Magtape Dependent Level

VI. Requesting DSK/MTA I/O & Interrupt Handling

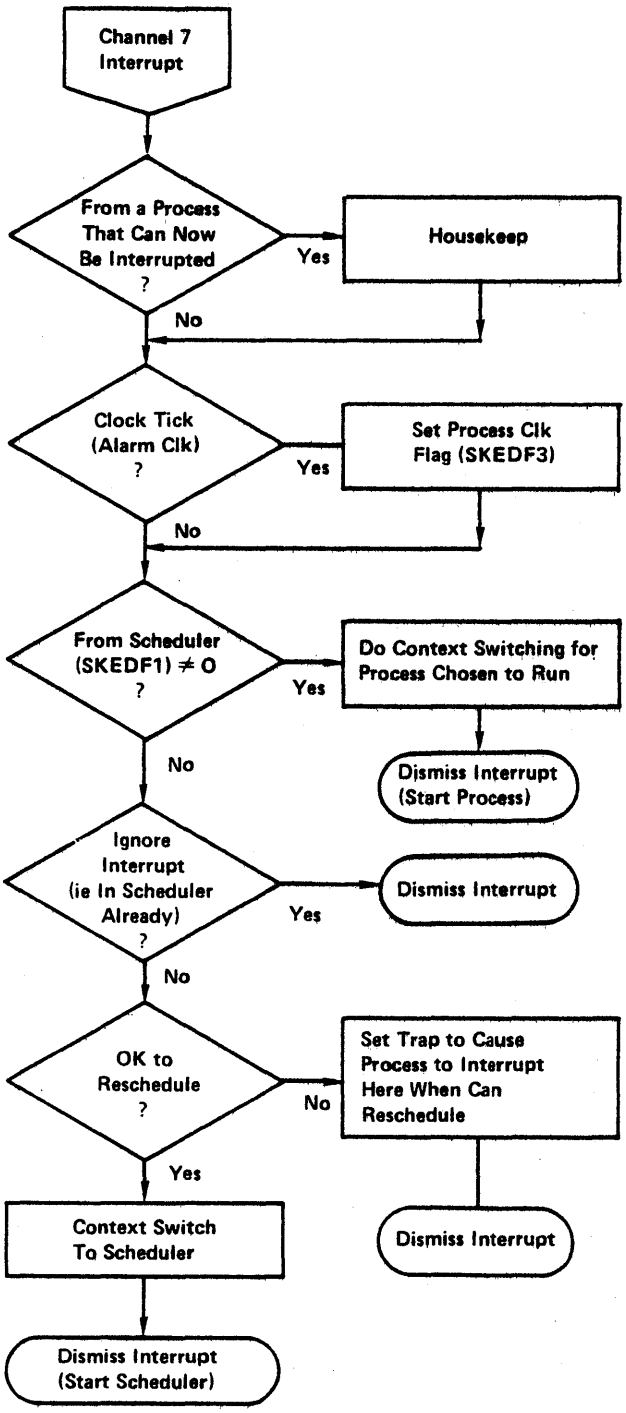
VII. JSYS Calls - TTY Dependent Level

VIII. Requesting TTY I/O & Interrupt Handling

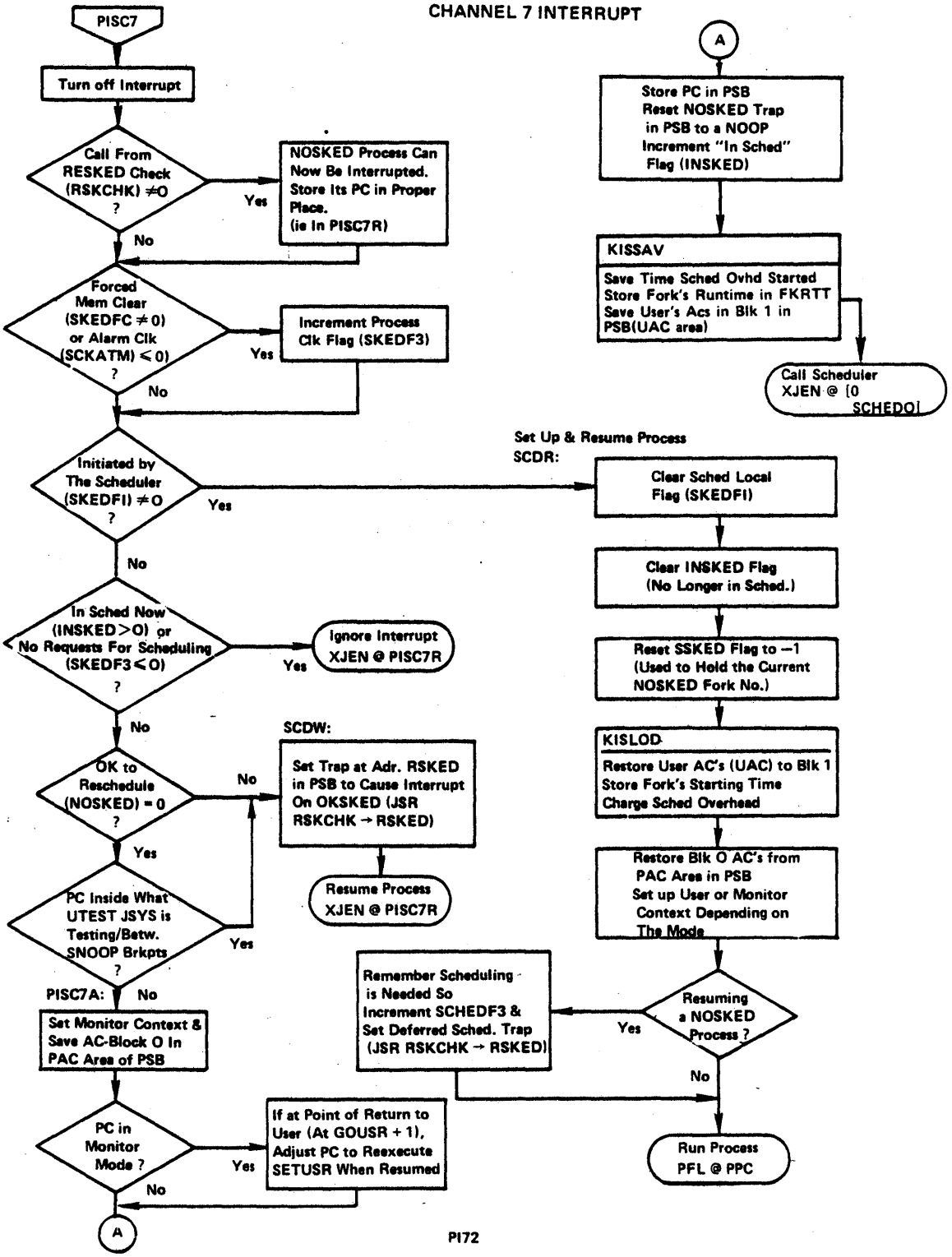
SCHEDULER FLOWCHARTS

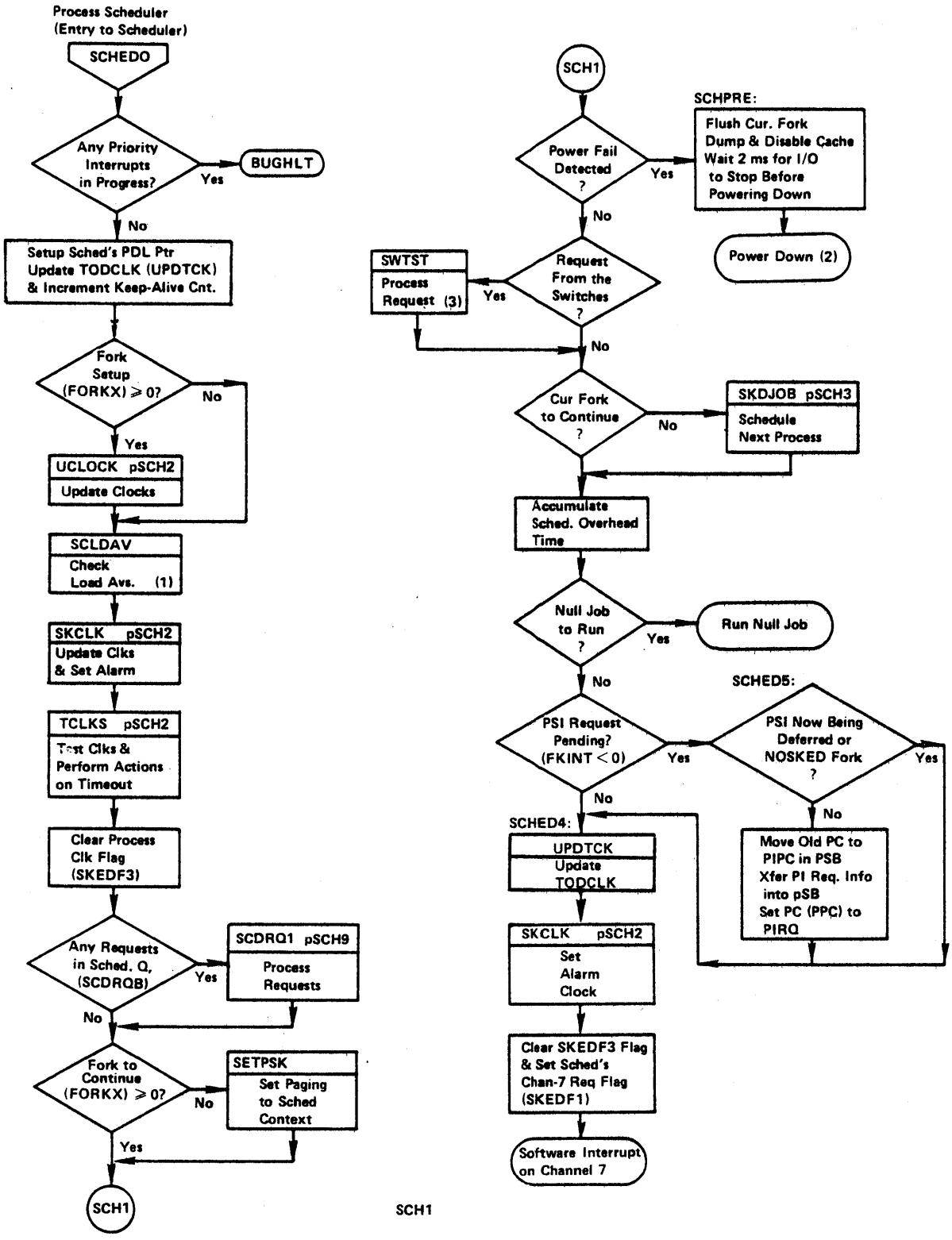
Channel 7 Interrupt - Context Switching Overview		PI71
PISC7 - Detailed Context Switching		PI72
SCHEDO - Process Controller		SCH1
UCLOCK - Process and System Accounting		SCH2
SKCLK - Update Clocks		SCH2
TCLKS - Test Clocks & Perform Action on Timeout		SCH2
SCDRQ1 - Process Requests in Scheduler's Queue		SCH9
JOBSRT - Job Startup		SCH9
SKDJOB - Select Process to Run		SCH3
GCCOR - Global Garbage Collect		SCH7
TSTBAL - Check if Balance Set Needs Adjustment		SCH5
AJBALS - Adjust Balance Set		SCH5

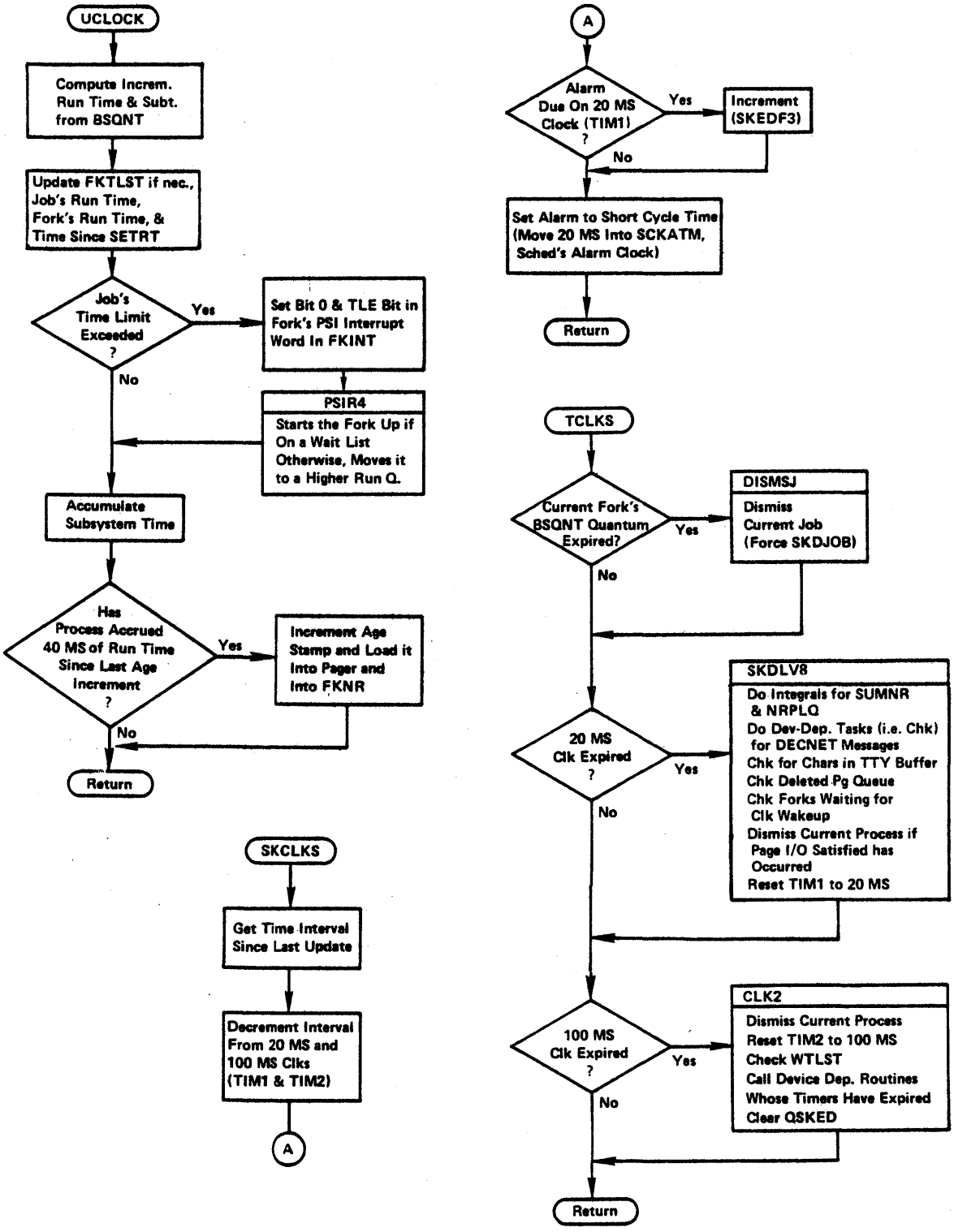
CHANNEL 7 INTERRUPT
AN OVERVIEW



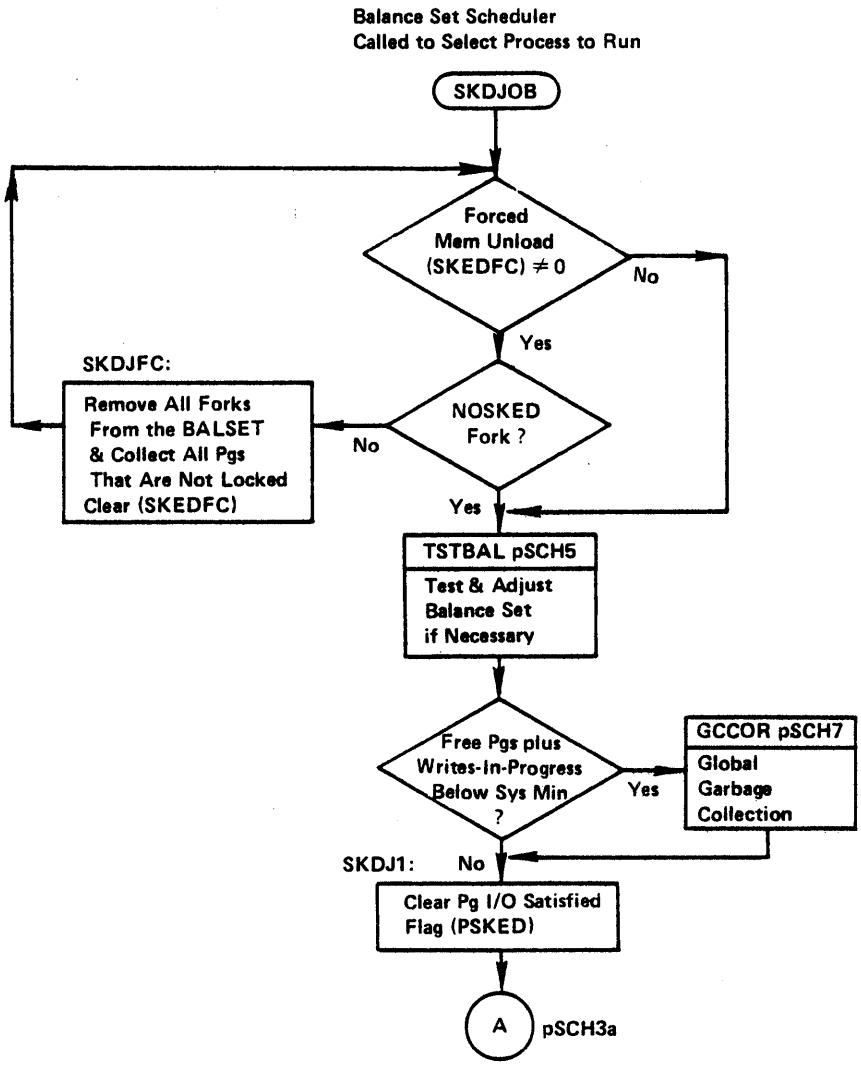
PI71


CHANNEL 7 INTERRUPT
PISC7
Turn off Interrupt
Call From RESKED Check (RSKCHK) ≠0 ?
Yes
NOSKED Process Can Now Be Interrupted. Store Its PC in Proper Place. (ie In PISC7R)
No
Forced Mem Clear (SKEDFC ≠ 0) or Alarm Clk (SCKATM) ≤ 0) ?
Yes
Increment Process Clk Flag (SKEDF3)
No
Initiated by The Scheduler (SKEDFI) ≠0 ?
Yes
No
In Sched Now (INSKED>0) or No Requests For Scheduling (SKEDF3≤0) ?
Yes
Ignore Interrupt XJEN @ PISC7R
No
SCDW:
OK to Reschedule (NOSKED) = 0 ?
No
Set Trap at Adr. RSKED in PSB to Cause Interrupt On OKSKED (JSR RSKCHK → RSKED)
Resume Process XJEN @ PISC7R
Yes
PC Inside What UTEST JSYS is Testing/Betw. SNOOP Brkpts ?
Yes
PISC7A:
No
Set Monitor Context & Save AC-Block 0 In PAC Area of PSB
PC in Monitor Mode ?
Yes
If at Point of Return to User (At GOUSR + 1), Adjust PC to Reexecute SETUSR When Resumed
No
A
A
Store PC in PSB
Reset NOSKED Trap in PSB to a NOOP
Increment "In Sched" Flag (INSKED)
KISSAV
Save Time Sched Ovhd Started
Store Fork's Runtime in FKRTT
Save User's Acs in Blk 1 in PSB(UAC area)
Call Scheduler XJEN @ [0 SCHED0]
Set Up & Resume Process
SCDR:
Clear Sched Local Flag (SKEDFI)
Clear INSKED Flag (No Longer in Sched.)
Reset SSKED Flag to −1 (Used to Hold the Current NOSKED Fork No.)
KISLOD
Restore User AC's (UAC) to Blk 1
Store Fork's Starting Time
Charge Sched Overhead
Restore Blk 0 AC's from PAC Area in PSB
Set up User or Monitor Context Depending on The Mode
Resuming a NOSKED Process ?
Yes
Remember Scheduling is Needed So Increment SCHEDF3 & Set Deferred Sched. Trap (JSR RSKCHK → RSKED)
No
Run Process PFL @ PPC
PI72

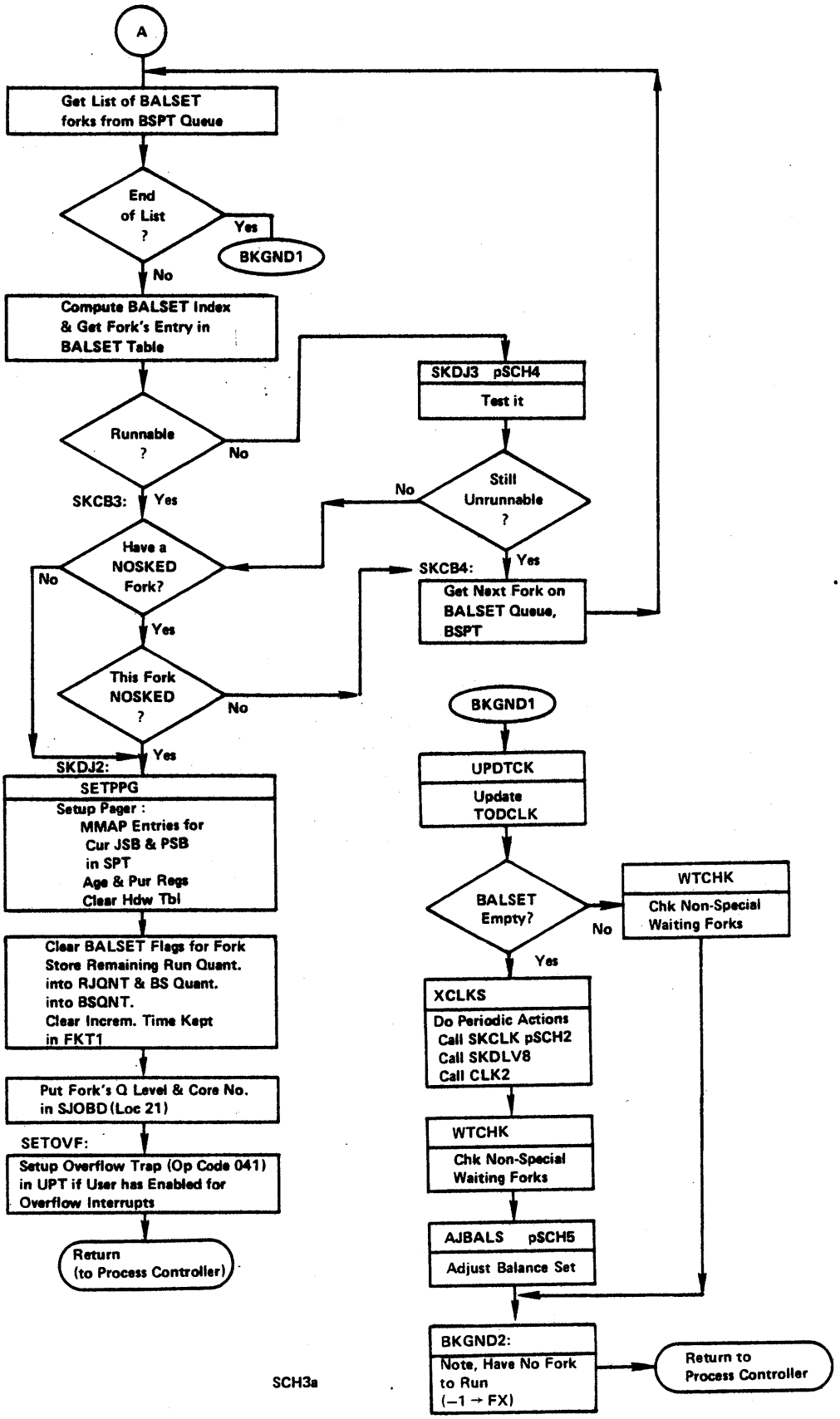

Process Scheduler
(Entry to Scheduler)
SCHEDO
Any Priority Interrupts in Progress?
Yes
BUGHLT
No
Setup Sched's PDL Ptr
Update TODCLK (UPDTCK)
& Increment Keep-Alive Cnt.
Fork Setup (FORKX) ≥ 0?
No
Yes
UCLOCK pSCH2
Update Clocks
SCLDAV
Check Load Avs. (1)
SKCLK pSCH2
Update Clks & Set Alarm
TCLKS pSCH2
Test Clks & Perform Actions on Timeout
Clear Process Clk Flag (SKEDF3)
Any Requests in Sched. Q, (SCDRQB)
Yes
SCDRQ1 pSCH9
Process Requests
No
Fork to Continue (FORKX) ≥ 0?
No
SETPSK
Set Paging to Sched Context
Yes
SCH1
SCH1
SCHPRE:
Power Fail Detected ?
Yes
Flush Cur. Fork
Dump & Disable Cache
Wait 2 ms for I/O
to Stop Before
Powering Down
Power Down (2)
No
SWTST
Process Request (3)
Yes
Request From the Switches ?
No
Cur Fork to Continue ?
No
SKDJOB pSCH3
Schedule Next Process
Accumulate Sched. Overhead Time
Null Job to Run ?
Yes
Run Null Job
No
SCHED5:
PSI Request Pending? (FKINT < 0)
Yes
PSI Now Being Deferred or NOSKED Fork ?
Yes
No
Move Old PC to PIPC in PSB
Xfer PI Req. Info into pSB
Set PC (PPC) to PIRQ
No
SCHED4:
UPDTCK
Update TODCLK
SKCLK pSCH2
Set Alarm Clock
Clear SKEDF3 Flag & Set Sched's Chan-7 Req Flag (SKEDF1)
Software Interrupt on Channel 7


UCLOCK
Compute Increm. Run Time & Subt. from BSQNT
Update FKTLST if nec., Job's Run Time, Fork's Run Time, & Time Since SETRT
Job's Time Limit Exceeded ?
Yes
Set Bit 0 & TLE Bit in Fork's PSI Interrupt Word In FKINT
No
PSIR4
Starts the Fork Up if On a Wait List Otherwise, Moves it to a Higher Run Q.
Accumulate Subsystem Time
Has Process Accrued 40 MS of Run Time Since Last Age Increment ?
Yes
Increment Age Stamp and Load it Into Pager and Into FKNR
No
Return
SKCLKS
Get Time Interval Since Last Update
Decrement Interval From 20 MS and 100 MS Clks (TIM1 & TIM2)
A
A
Alarm Due On 20 MS Clock (TIM1) ?
Yes
Increment (SKEDF3)
No
Set Alarm to Short Cycle Time (Move 20 MS Into SCKATM, Sched's Alarm Clock)
Return
TCLKS
Current Fork's BSQNT Quantum Expired?
Yes
DISMSJ
Dismiss Current Job (Force SKDJOB)
No
20 MS Clk Expired ?
Yes
SKDLV8
Do Integrals for SUMNR & NRPLQ
Do Dev-Dep. Tasks (i.e. Chk) for DECNET Messages
Chk for Chars in TTY Buffer
Chk Deleted Pg Queue
Chk Forks Waiting for Clk Wakeup
Dismiss Current Process if Page I/O Satisfied has Occurred
Reset TIM1 to 20 MS
No
100 MS Clk Expired ?
Yes
CLK2
Dismiss Current Process
Reset TIM2 to 100 MS
Check WTLST
Call Device Dep. Routines Whose Timers Have Expired
Clear QSKED
No
Return

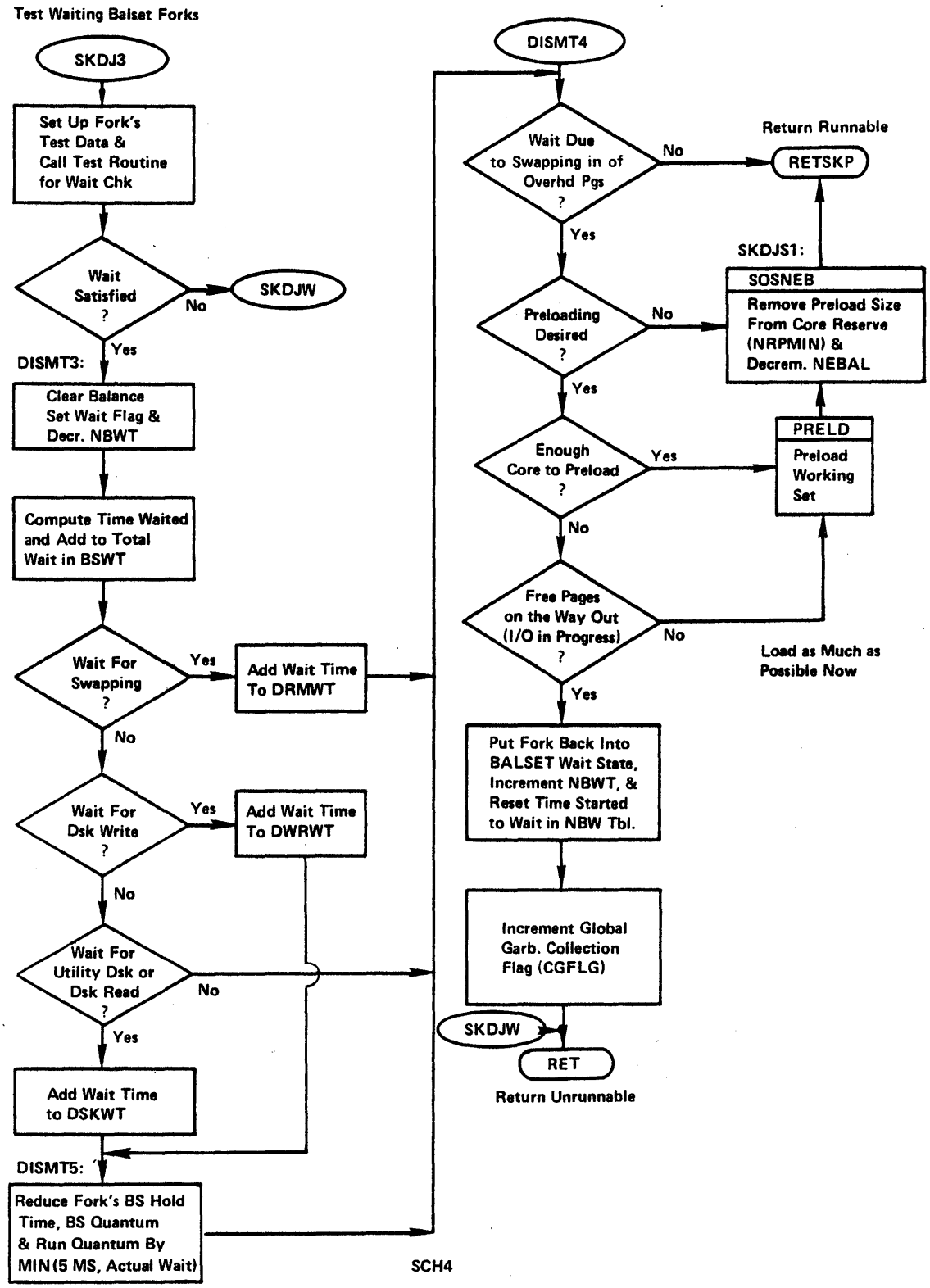
SCH2

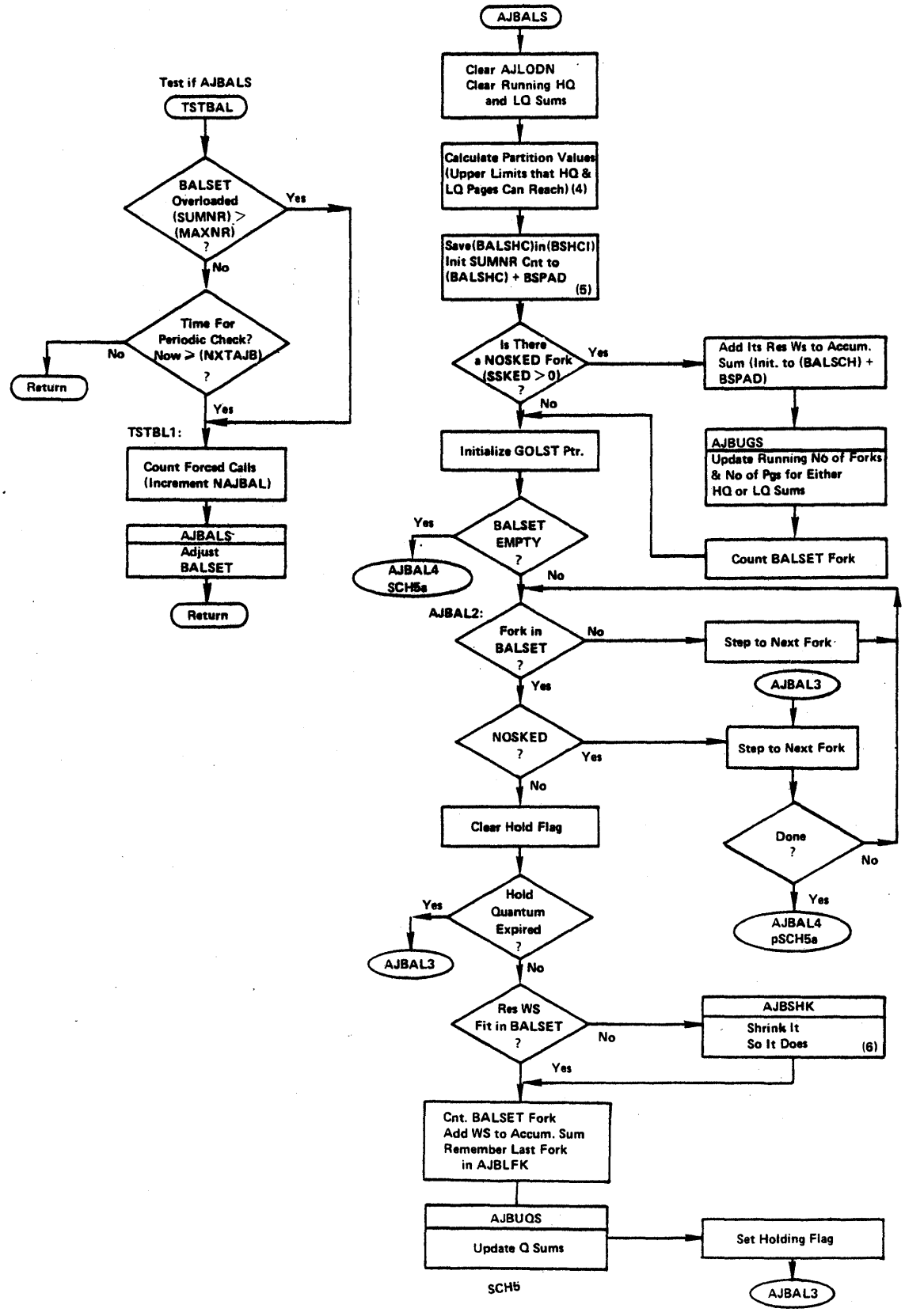

Balance Set Scheduler
Called to Select Process to Run
SKDJOB
Forced
Mem Unload
(SKEDFC) ≠ 0
No
Yes
SKDJFC:
Remove All Forks
From the BALSET
& Collect All Pgs
That Are Not Locked
Clear (SKEDFC)
NOSKED
Fork ?
No
Yes
TSTBAL pSCH5
Test & Adjust
Balance Set
if Necessary
Free Pgs plus
Writes-In-Progress
Below Sys Min
?
Yes
GCCOR pSCH7
Global
Garbage
Collection
SKDJ1:
No
Clear Pg I/O Satisfied
Flag (PSKED)
A
pSCH3a

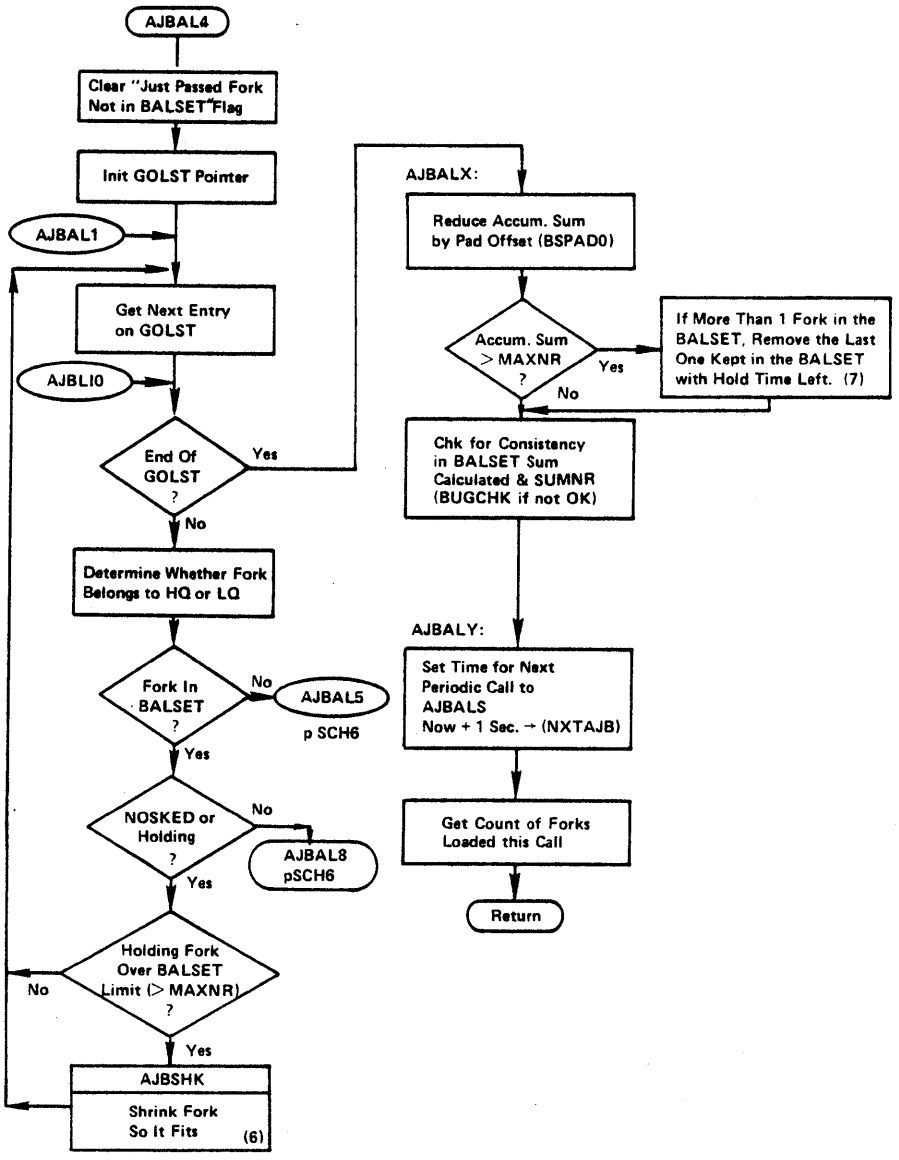
SCH3


A
Get List of BALSET forks from BSPT Queue
End of List ?
Yes
BKGND1
No
Compute BALSET Index & Get Fork's Entry in BALSET Table
Runnable ?
No
SKCB3:
Yes
Have a NOSKED Fork?
No
Yes
This Fork NOSKED ?
No
Yes
SKDJ2:
SETPPG
Setup Pager :
MMAP Entries for Cur JSB & PSB in SPT
Age & Pur Regs
Clear Hdw Tbl
Clear BALSET Flags for Fork
Store Remaining Run Quant. into RJQNT & BS Quant. into BSQNT.
Clear Increm. Time Kept in FKT1
Put Fork's Q Level & Core No. in SJOBD (Loc 21)
SETOVF:
Setup Overflow Trap (Op Code 041) in UPT if User has Enabled for Overflow Interrupts
Return (to Process Controller)
SKDJ3 pSCH4
Test it
Still Unrunnable ?
No
Yes
SKCB4:
Get Next Fork on BALSET Queue, BSPT
BKGND1
UPDTCK
Update TODCLK
BALSET Empty?
No
Yes
WTCHK
Chk Non-Special Waiting Forks
XCLKS
Do Periodic Actions
Call SKCLK pSCH2
Call SKDLV8
Call CLK2
WTCHK
Chk Non-Special Waiting Forks
AJBALS pSCH5
Adjust Balance Set
BKGND2:
Note, Have No Fork to Run
(−1 → FX)
Return to Process Controller

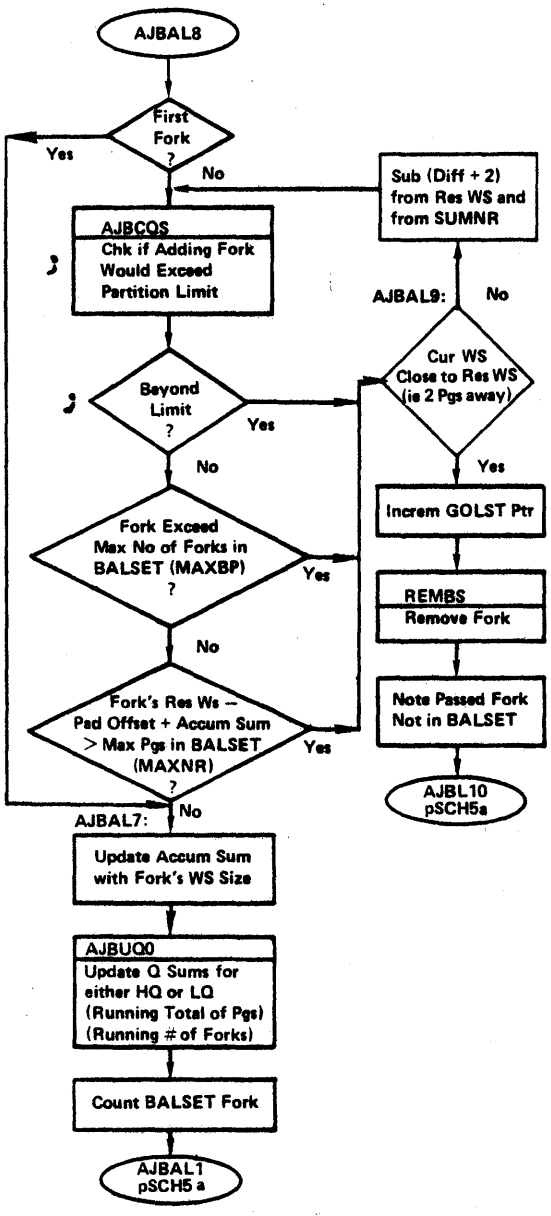
SCH3a

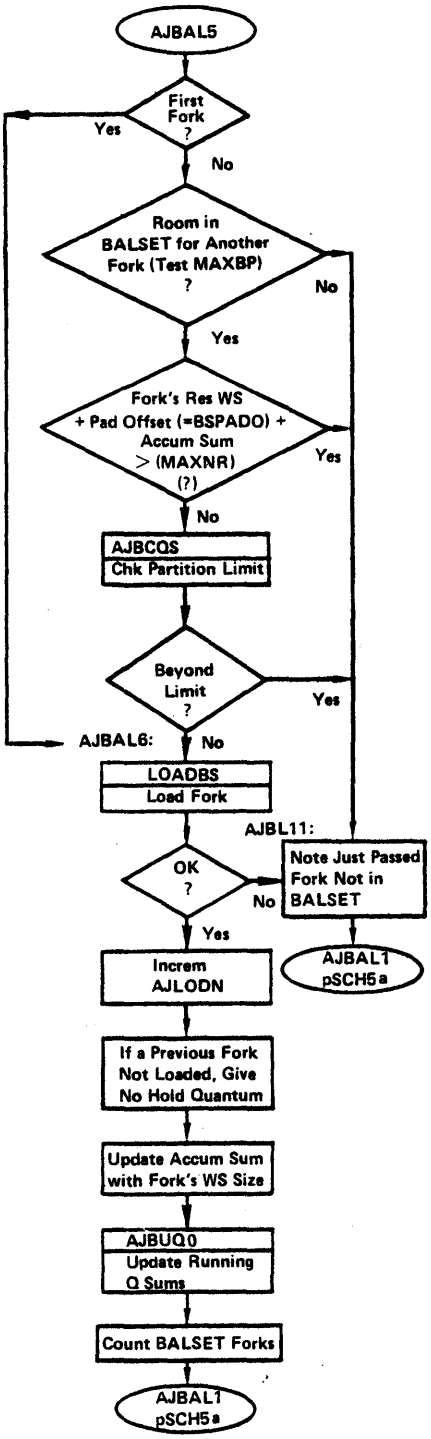

Test Waiting Balset Forks
SKDJ3
Set Up Fork's Test Data & Call Test Routine for Wait Chk
Wait Satisfied ?
No
SKDJW
Yes
DISMT3:
Clear Balance Set Wait Flag & Decr. NBWT
Compute Time Waited and Add to Total Wait in BSWT
Wait For Swapping ?
Yes
Add Wait Time To DRMWT
No
Wait For Dsk Write ?
Yes
Add Wait Time To DWRWT
No
Wait For Utility Dsk or Dsk Read ?
No
Yes
Add Wait Time to DSKWT
DISMT5:
Reduce Fork's BS Hold Time, BS Quantum & Run Quantum By MIN(5 MS, Actual Wait)
SCH4
DISMT4
Wait Due to Swapping in of Overhd Pgs ?
No
Return Runnable
RETSKP
Yes
SKDJS1:
SOSNEB
Remove Preload Size From Core Reserve (NRPMIN) & Decrem. NEBAL
Preloading Desired ?
No
Yes
PRELD
Preload Working Set
Enough Core to Preload ?
Yes
No
Free Pages on the Way Out (I/O in Progress) ?
No
Load as Much as Possible Now
Yes
Put Fork Back Into BALSET Wait State, Increment NBWT, & Reset Time Started to Wait in NBW Tbl.
Increment Global Garb. Collection Flag (CGFLG)
SKDJW
RET
Return Unrunnable


Test if AJBALS
TSTBAL
BALSET Overloaded (SUMNR) > (MAXNR) ?
Yes
No
Time For Periodic Check? Now > (NXTAJB) ?
No
Return
Yes
TSTBL1:
Count Forced Calls (Increment NAJBAL)
AJBALS
Adjust BALSET
Return
AJBALS
Clear AJLODN
Clear Running HQ and LQ Sums
Calculate Partition Values (Upper Limits that HQ & LQ Pages Can Reach) (4)
Save(BALSHC)in(BSHCI) Init SUMNR Cnt to (BALSHC) + BSPAD (5)
Is There a NOSKED Fork (SSKED > 0) ?
Yes
Add Its Res Ws to Accum. Sum (Init. to (BALSCH) + BSPAD)
No
Initialize GOLST Ptr.
AJBUGS
Update Running No of Forks & No of Pgs for Either HQ or LQ Sums
BALSET EMPTY ?
Yes
AJBAL4 SCH5a
Count BALSET Fork
No
AJBAL2:
Fork in BALSET ?
No
Step to Next Fork
AJBAL3
Yes
NOSKED ?
Yes
Step to Next Fork
No
Clear Hold Flag
Done ?
No
Yes
AJBAL4 pSCH5a
Hold Quantum Expired ?
Yes
AJBAL3
No
Res WS Fit in BALSET ?
No
AJBSHK
Shrink It So It Does (6)
Yes
Cnt. BALSET Fork
Add WS to Accum. Sum
Remember Last Fork in AJBLFK
AJBUQS
Update Q Sums
SCH5
Set Holding Flag
AJBAL3

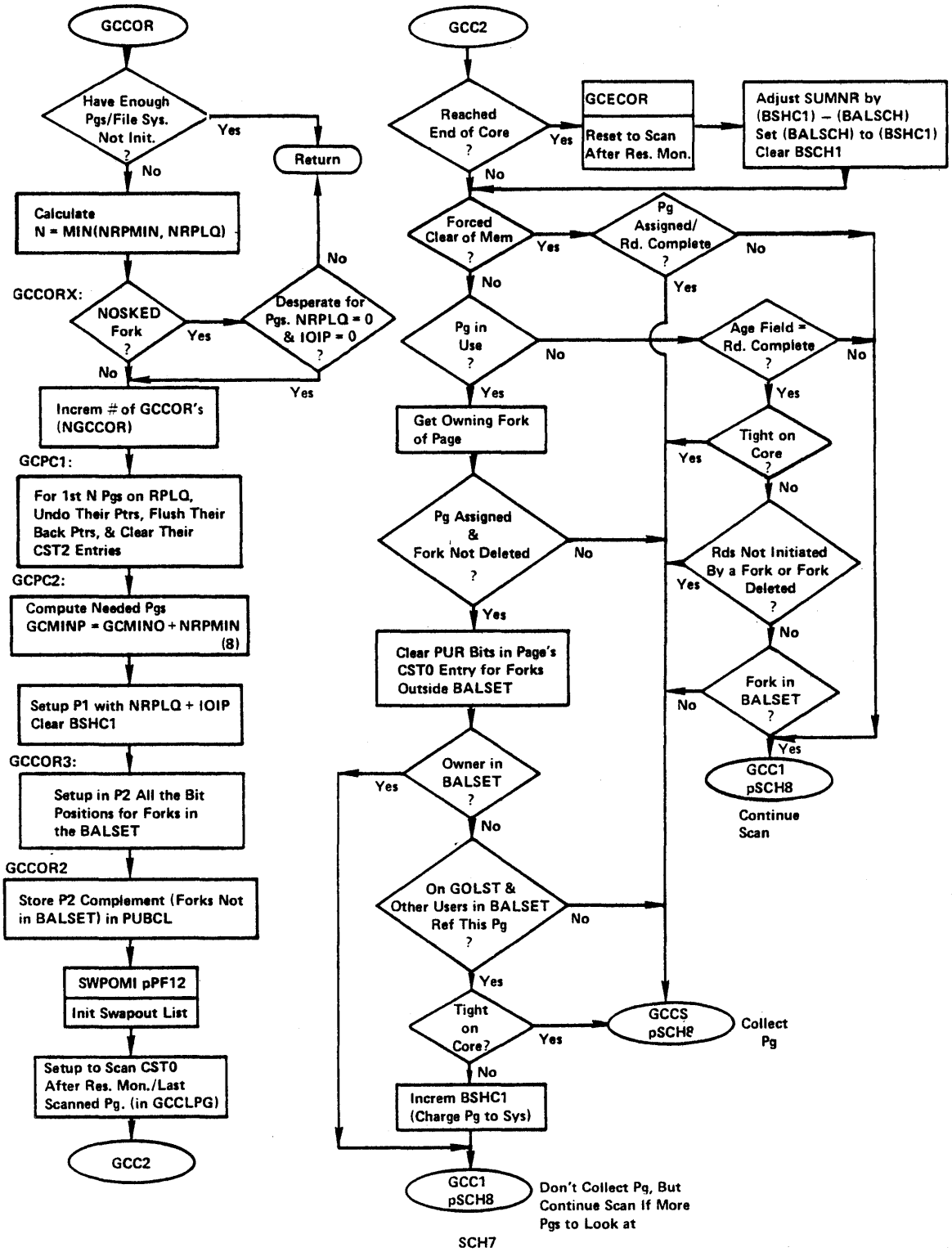

AJBAL4
Clear "Just Passed Fork Not in BALSET" Flag
Init GOLST Pointer
AJBAL1
Get Next Entry on GOLST
AJBLI0
End Of GOLST ?
Yes
No
Determine Whether Fork Belongs to HQ or LQ
Fork In BALSET ?
No
AJBAL5
p SCH6
Yes
NOSKED or Holding ?
No
AJBAL8
pSCH6
Yes
Holding Fork Over BALSET Limit (> MAXNR) ?
No
Yes
AJBSHK
Shrink Fork So It Fits (6)
AJBALX:
Reduce Accum. Sum by Pad Offset (BSPAD0)
Accum. Sum > MAXNR ?
Yes
If More Than 1 Fork in the BALSET, Remove the Last One Kept in the BALSET with Hold Time Left. (7)
No
Chk for Consistency in BALSET Sum Calculated & SUMNR (BUGCHK if not OK)
AJBALY:
Set Time for Next Periodic Call to AJBALS Now + 1 Sec. → (NXTAJB)
Get Count of Forks Loaded this Call
Return

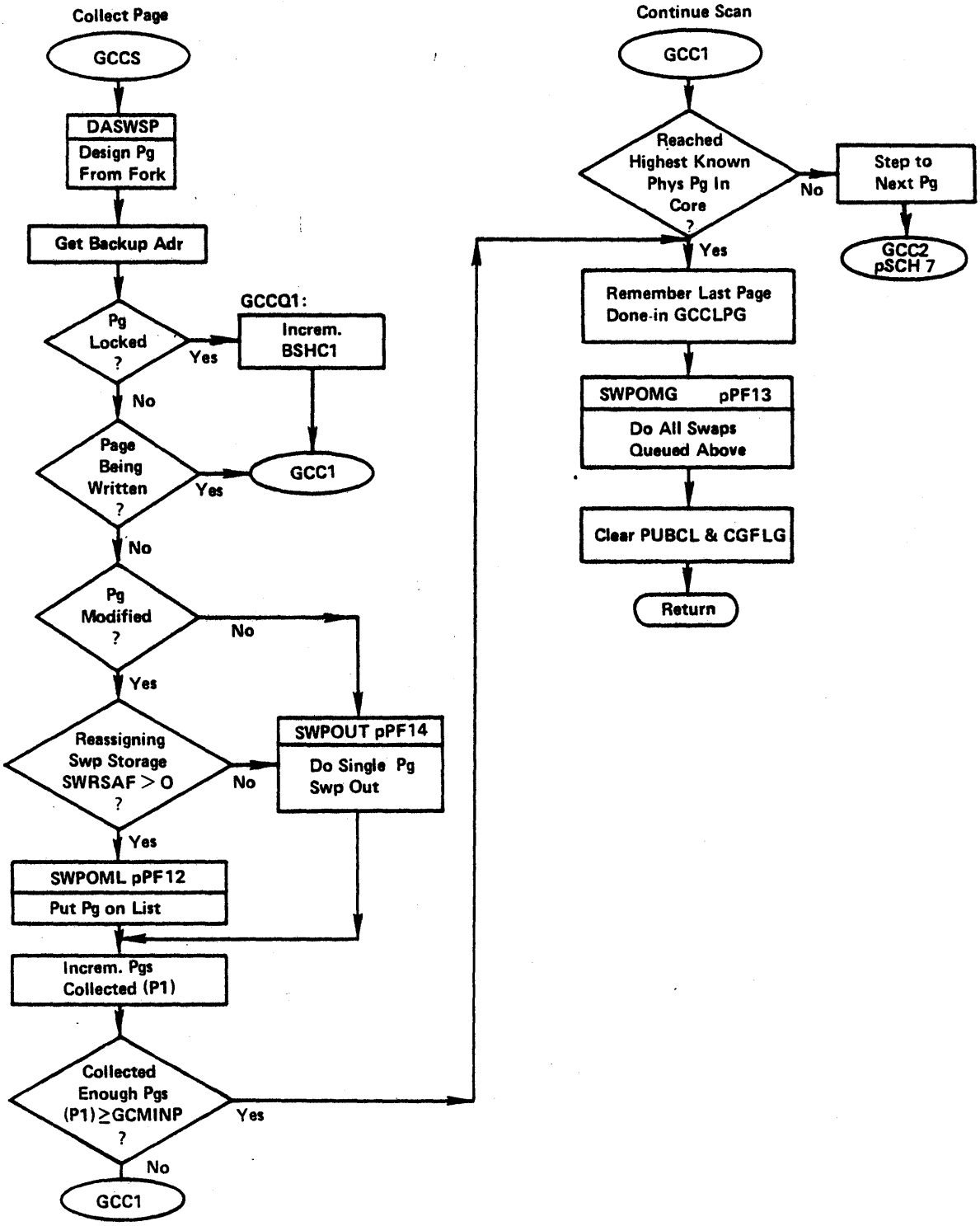
SCH5a


AJBAL8
First Fork ?
Yes
No
AJBCQS
Chk if Adding Fork Would Exceed Partition Limit
Beyond Limit ?
Yes
No
Fork Exceed Max No of Forks in BALSET (MAXBP) ?
Yes
No
Fork's Res Ws – Pad Offset + Accum Sum > Max Pgs in BALSET (MAXNR) ?
Yes
No
AJBAL7:
Update Accum Sum with Fork's WS Size
AJBUQ0
Update Q Sums for either HQ or LQ (Running Total of Pgs) (Running # of Forks)
Count BALSET Fork
AJBAL1 pSCH5a
Sub (Diff + 2) from Res WS and from SUMNR
AJBAL9:
No
Cur WS Close to Res WS (ie 2 Pgs away)
Yes
Increm GOLST Ptr
REMBS
Remove Fork
Note Passed Fork Not in BALSET
AJBL10 pSCH5a

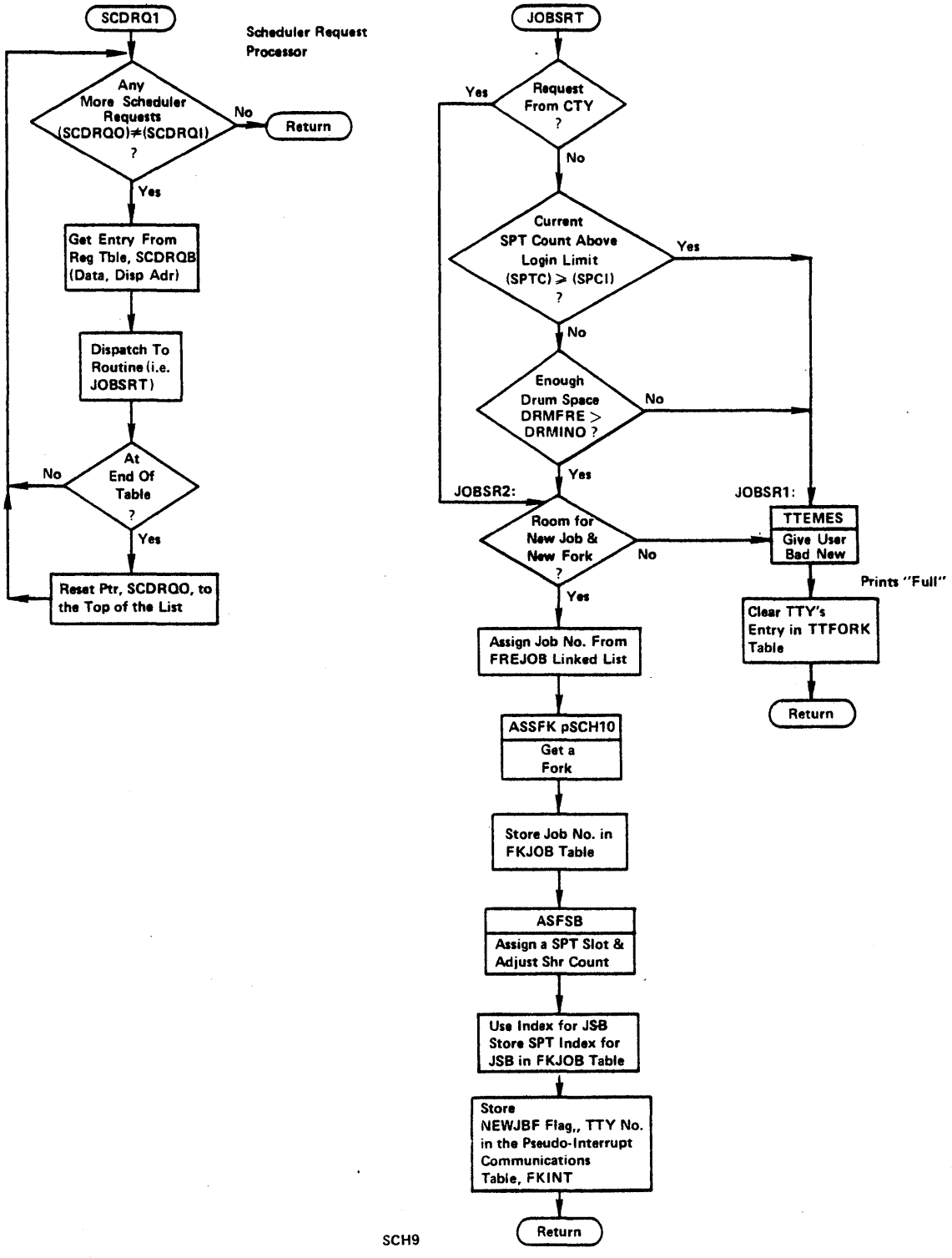

AJBAL5
First Fork ?
Yes
No
Room in BALSET for Another Fork (Test MAXBP) ?
No
Yes
Fork's Res WS + Pad Offset (=BSPADO) + Accum Sum > (MAXNR) (?)
Yes
No
AJBCQS
Chk Partition Limit
Beyond Limit ?
Yes
AJBAL6:
No
LOADBS
Load Fork
AJBL11:
OK ?
No
Note Just Passed Fork Not in BALSET
AJBAL1 pSCH5a
Yes
Increm AJLODN
If a Previous Fork Not Loaded, Give No Hold Quantum
Update Accum Sum with Fork's WS Size
AJBUQ0
Update Running Q Sums
Count BALSET Forks
AJBAL1 pSCH5a

SCH6

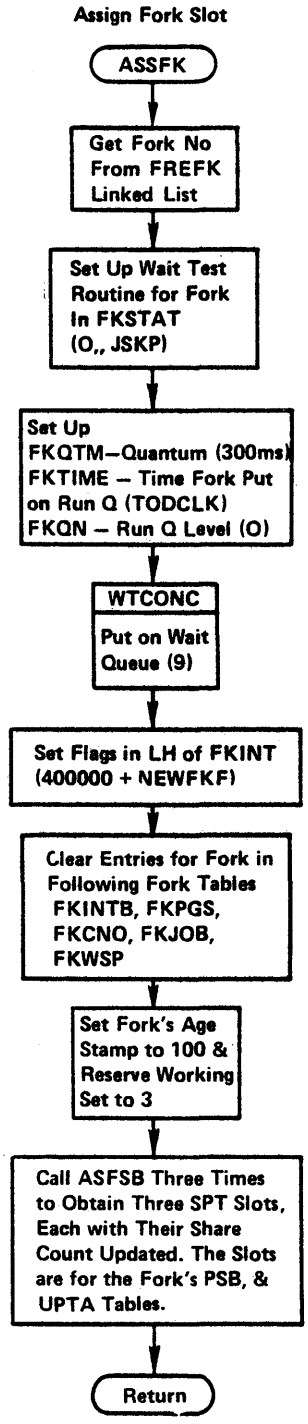

GCCOR
Have Enough Pgs/File Sys. Not Init. ?
Yes
Return
No
Calculate N = MIN(NRPMIN, NRPLQ)
GCCORX:
NOSKED Fork ?
Yes
Desperate for Pgs. NRPLQ = 0 & IOIP = 0 ?
No
Yes
No
Increm # of GCCOR's (NGCCOR)
GCPC1:
For 1st N Pgs on RPLQ, Undo Their Ptrs, Flush Their Back Ptrs, & Clear Their CST2 Entries
GCPC2:
Compute Needed Pgs GCMINP = GCMINO + NRPMIN (8)
Setup P1 with NRPLQ + IOIP Clear BSHC1
GCCOR3:
Setup in P2 All the Bit Positions for Forks in the BALSET
GCCOR2
Store P2 Complement (Forks Not in BALSET) in PUBCL
SWPOMI pPF12
Init Swapout List
Setup to Scan CST0 After Res. Mon./Last Scanned Pg. (in GCCLPG)
GCC2
GCC2
Reached End of Core ?
Yes
GCECOR
Reset to Scan After Res. Mon.
Adjust SUMNR by (BSHC1) – (BALSCH) Set (BALSCH) to (BSHC1) Clear BSCH1
No
Forced Clear of Mem ?
Yes
Pg Assigned/ Rd. Complete ?
No
Yes
No
Pg in Use ?
No
Age Field = Rd. Complete ?
No
Yes
Yes
Get Owning Fork of Page
Tight on Core ?
Yes
No
Pg Assigned & Fork Not Deleted ?
No
Rds Not Initiated By a Fork or Fork Deleted ?
Yes
Yes
No
Clear PUR Bits in Page's CST0 Entry for Forks Outside BALSET
Fork in BALSET ?
No
Yes
GCC1 pSCH8
Continue Scan
Owner in BALSET ?
Yes
No
On GOLST & Other Users in BALSET Ref This Pg ?
No
Yes
Tight on Core?
Yes
GCCS pSCH8
Collect Pg
No
Increm BSHC1 (Charge Pg to Sys)
GCC1 pSCH8
Don't Collect Pg, But Continue Scan If More Pgs to Look at
SCH7


Collect Page
GCCS
DASWSP
Design Pg From Fork
Get Backup Adr
GCCQ1:
Pg Locked ?
Yes
Increm. BSHC1
No
Page Being Written ?
Yes
GCC1
No
Pg Modified ?
No
Yes
Reassigning Swp Storage SWRSAF > 0 ?
No
SWPOUT pPF14
Do Single Pg Swp Out
Yes
SWPOML pPF12
Put Pg on List
Increm. Pgs Collected (P1)
Collected Enough Pgs (P1) ≥ GCMINP ?
Yes
No
GCC1
Continue Scan
GCC1
Reached Highest Known Phys Pg In Core ?
No
Step to Next Pg
GCC2 pSCH 7
Yes
Remember Last Page Done-in GCCLPG
SWPOMG pPF13
Do All Swaps Queued Above
Clear PUBCL & CGFLG
Return

SCH8


SCDRQ1
Scheduler Request Processor
Any More Scheduler Requests (SCDRQO)≠(SCDRQI) ?
No
Return
Yes
Get Entry From Req Tble, SCDRQB (Data, Disp Adr)
Dispatch To Routine (i.e. JOBSRT)
At End Of Table ?
No
Yes
Reset Ptr, SCDRQO, to the Top of the List
JOBSRT
Request From CTY ?
Yes
No
Current SPT Count Above Login Limit (SPTC) ≥ (SPCI) ?
Yes
No
Enough Drum Space DRMFRE > DRMINO ?
No
Yes
JOBSR2:
JOBSR1:
Room for New Job & New Fork ?
No
Yes
TTEMES
Give User Bad New
Prints "Full"
Clear TTY's Entry in TTFORK Table
Return
Assign Job No. From FREJOB Linked List
ASSFK pSCH10
Get a Fork
Store Job No. in FKJOB Table
ASFSB
Assign a SPT Slot & Adjust Shr Count
Use Index for JSB Store SPT Index for JSB in FKJOB Table
Store NEWJBF Flag,, TTY No. in the Pseudo-Interrupt Communications Table, FKINT
Return

SCH9


Assign Fork Slot
ASSFK
Get Fork No From FREFK Linked List
Set Up Wait Test Routine for Fork In FKSTAT (O,, JSKP)
Set Up
FKQTM—Quantum (300ms)
FKTIME — Time Fork Put on Run Q (TODCLK)
FKQN — Run Q Level (O)
WTCONC
Put on Wait Queue (9)
Set Flags in LH of FKINT (400000 + NEWFKF)
Clear Entries for Fork in Following Fork Tables FKINTB, FKPGS, FKCNO, FKJOB, FKWSP
Set Fork's Age Stamp to 100 & Reserve Working Set to 3
Call ASFSB Three Times to Obtain Three SPT Slots, Each with Their Share Count Updated. The Slots are for the Fork's PSB, & UPTA Tables.
Return

SCHIO

Scheduler Comments

SCHEDO:

(1) Running averages, exponentially weighed over intervals of 1, 5, and 15 minutes, are maintained for the number of runnable processes overall, as well as for those in High Run Queues and those in the Low Run Queues.

(2) Final phase of powerdown seq. clears the priority interrupt system and causes the system to loop in the ACs until power actually vanishes. If the power fail interrupt was spurious, the loop will time out after a few seconds and the system will be continued at address SYSRST.

(3) A very limited set of central functions for debugging purposes has been built into the Scheduler. To invoke a function, the appropriate bit or bits are set into loc 20 (SCTLW) via MDDT. The word is scanned from left to right (JFFO); the first bit found set on the scan selects the function.

Bit 0 Causes the scheduler to dismiss the current process and to stop timesharing. Useful to effect a clean manual transfer to Exec-mode DDT. System may be resumed at SCHEDO if no IOB reset is done.

Bit 1 Causes job specified by $(20)_{RH}$ to be run exclusively.

Bit 2 Forces running of Job 0 back-up function before halting the system.

If loc 30 (SHLTW) is set not equal to Ø, the system will crash. (Same as setting bit 2 of SCTLW word.)

AJBALS

(4) Upper Limit for LQ=MAXNR-MIN [Max HQ Reserve, HQ Load Avg.* (16)]

Upper Limit for HQ=MAXNR-MIN [Max LQ Reserve, No. of LQ forks * (32)]

(5) SUMNR reflects the number of timesharing pages in use. Its value after AJBALS equals the number of pages reserved for balance set members plus BALSHC (the number of pages shared, but not owned, by balance set members plus the number of locked pages).

BSPAD reflects the number of pages set aside for balance set members as their working set reserves grow. The real value of BSPAD is offset by a factor of BSPADO. When forks are trying to stay in the balance set, the adjustment algorithm allows the pad offset to be subtracted from the accumulated sum before it checks if the fork can fit.

i.e.,

$$(\text{BSPAD} + \sum_{i=1}^{n} \text{Res. WS}_i) - \text{BSPADO} + \text{Res. WS}_{n+1} \geq \text{MAXNR}$$

The adjustment algorithm does the opposite (i.e., adds the BSPADO factor) for forks trying to get into the balance set. The overall affect of this is to ensure (as much as possible) a certain number of pages be available for balance set forks.

(6) The shrink algorithm shrinks the fork's reserve working set by:

MIN [Reserve WS - Current WS, Accum. Sum + Fork's Res WS-MAXNR]

Notice that the fork's reserve working set will not be reduced below its current working set.

(7) This is the rare case of forks, with hold-time left, expanding. The lowest priority one is removed. If there is only one fork in the balance set, it is not removed. (Note: it is possible for one fork to be greater than MAXNR due to the BALSHC count changing).

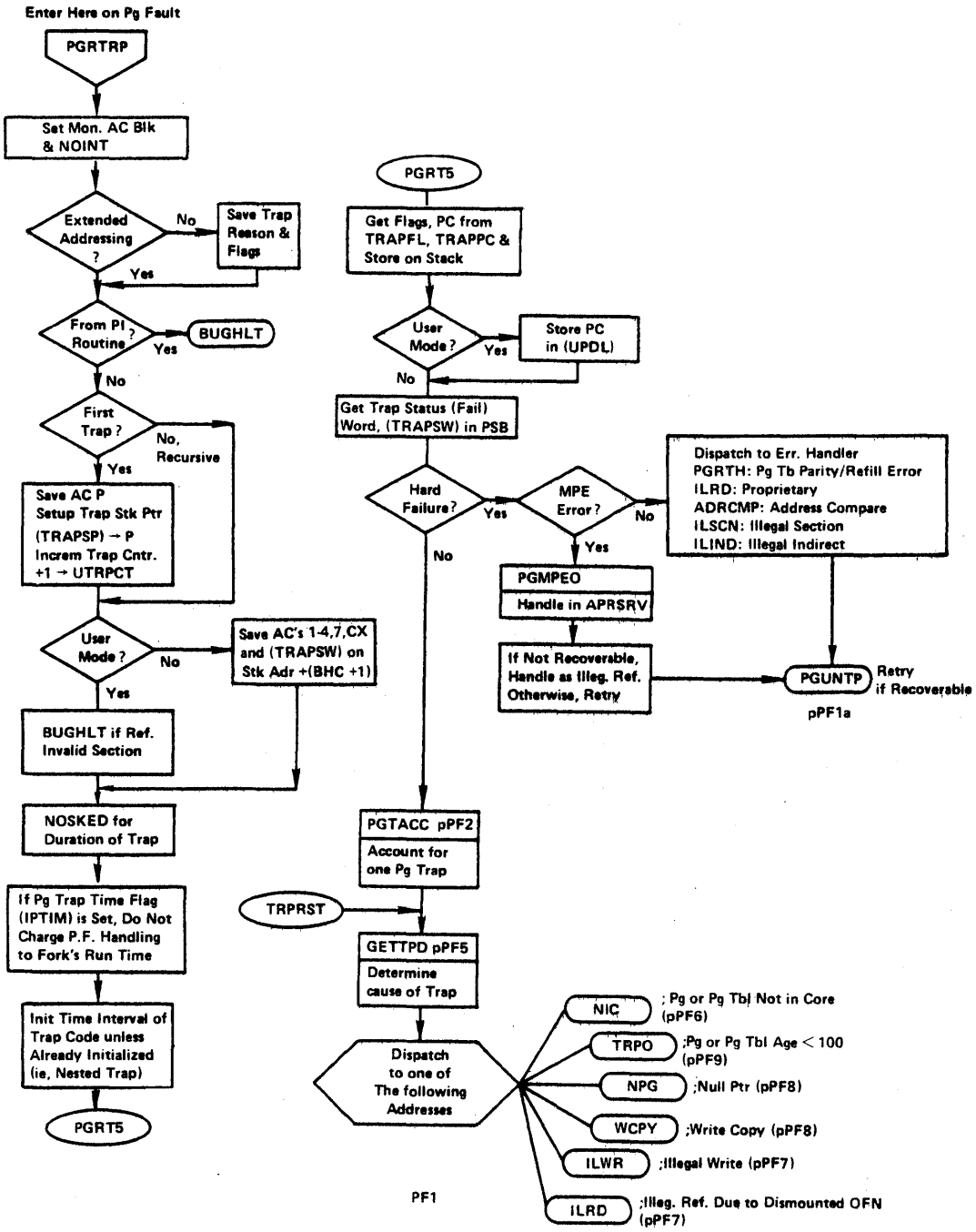
GCCOR

(8) If it is a forced clear, then GCMINO is made very large so all of core will be collected. However, its usual value is much lower. (Currently 64 decimal).

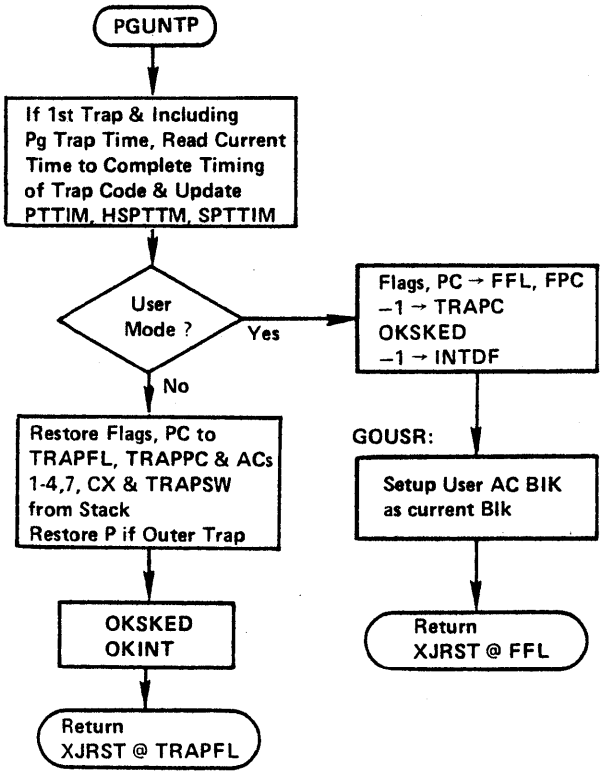
(9) The fork is actually placed on the GOLST at this time. WTCONC, after putting a fork on WTLST, checks if the wait condition is satisfied. The test routine, JSKP, gives a skip return indicating that the wait is satisfied. This causes UNBLK1 to be called which in turn calls SCHEDJ to unblock the fork and to requeue it from the WTLST to the GOLST.

PAGE FAULT HANDLING FLOWCHARTS

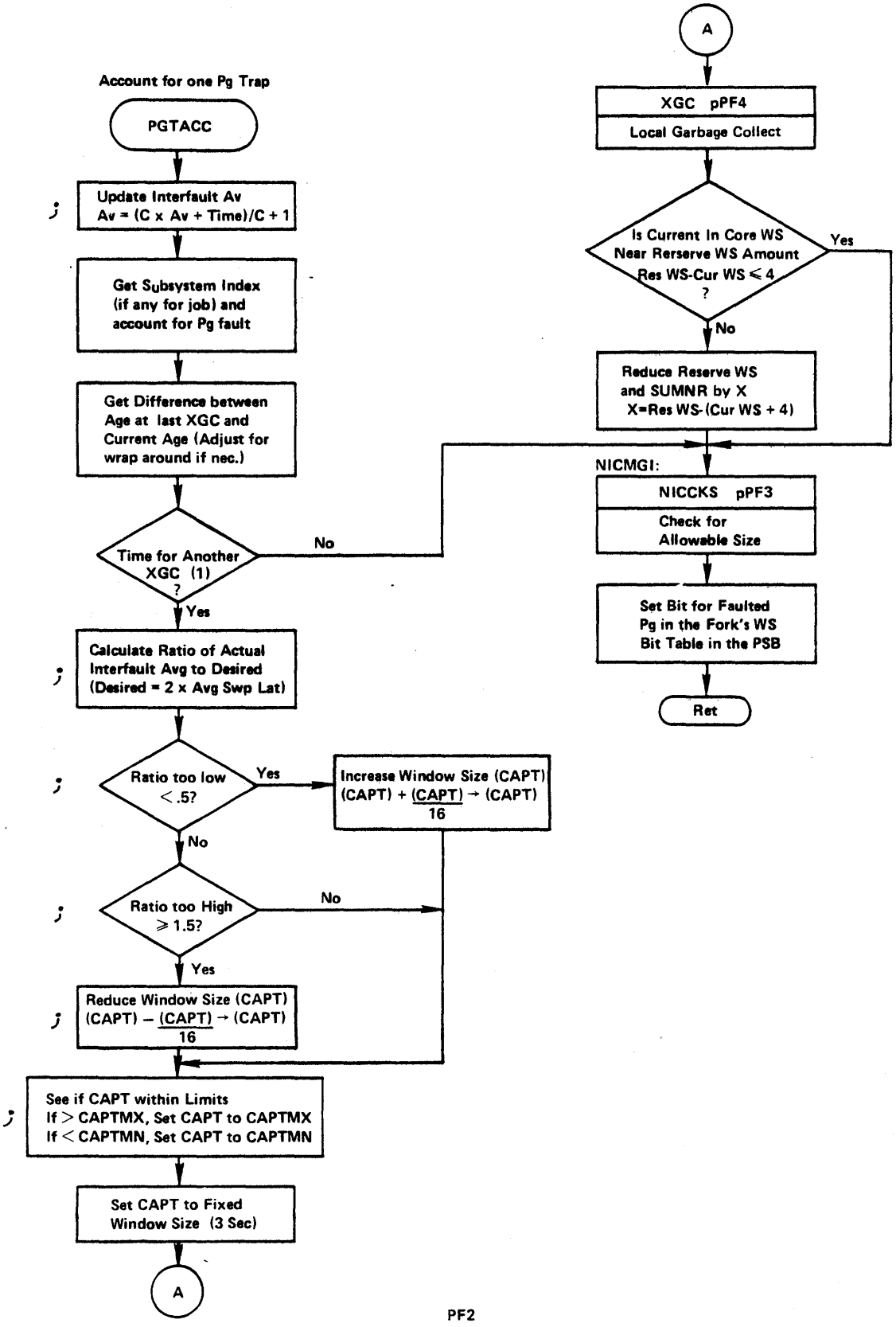
PGRTRP -	Performs the Principal Accounting, Analysis, and Resolution of Page Faults		PF1
	PGTACC -	Accounts for Page Traps	PF2
		XGC - Local Garbage Collection	PF4
		SWPOUT - Swapping Out a Page	PF14
		NICCKS - Check In-Core Size Limits	PF3
	GETTPD -	Determine Cause of Trap	PF5
	NIC -	Not in Core Trap	PF6
		SWPINW - Swap In and Wait	PF10
		SWPIN - Swap In a Page	PF11
	WCPY -	Write Copy Trap	PF8
	ILRD -	Illegal Read Trap	PF7
	ILWR -	Illegal Write Trap	PF7
	TRPO -	Age < 100 Trap	PF9


Enter Here on Pg Fault
PGRTRP
Set Mon. AC Blk & NOINT
Extended Addressing ?
No
Save Trap Reason & Flags
Yes
From PI Routine ?
Yes
BUGHLT
No
First Trap ?
No, Recursive
Yes
Save AC P
Setup Trap Stk Ptr
(TRAPSP) → P
Increm Trap Cntr.
+1 → UTRPCT
User Mode ?
No
Save AC's 1-4,7,CX and (TRAPSW) on Stk Adr +(BHC +1)
Yes
BUGHLT if Ref. Invalid Section
NOSKED for Duration of Trap
If Pg Trap Time Flag (IPTIM) is Set, Do Not Charge P.F. Handling to Fork's Run Time
Init Time Interval of Trap Code unless Already Initialized (ie, Nested Trap)
PGRT5
PGRT5
Get Flags, PC from TRAPFL, TRAPPC & Store on Stack
User Mode ?
Yes
Store PC in (UPDL)
No
Get Trap Status (Fail) Word, (TRAPSW) in PSB
Hard Failure?
Yes
MPE Error ?
No
Yes
Dispatch to Err. Handler
PGRTH: Pg Tb Parity/Refill Error
ILRD: Proprietary
ADRCMP: Address Compare
ILSCN: Illegal Section
ILIND: Illegal Indirect
PGMPEO
Handle in APRSRV
If Not Recoverable, Handle as Illeg. Ref. Otherwise, Retry
PGUNTP
Retry if Recoverable
pPF1a
No
PGTACC pPF2
Account for one Pg Trap
TRPRST
GETTPD pPF5
Determine cause of Trap
Dispatch to one of The following Addresses
NIC
; Pg or Pg Tbl Not in Core (pPF6)
TRPO
;Pg or Pg Tbl Age < 100 (pPF9)
NPG
;Null Ptr (pPF8)
WCPY
;Write Copy (pPF8)
ILWR
;Illegal Write (pPF7)
ILRD
;Illeg. Ref. Due to Dismounted OFN (pPF7)
PF1

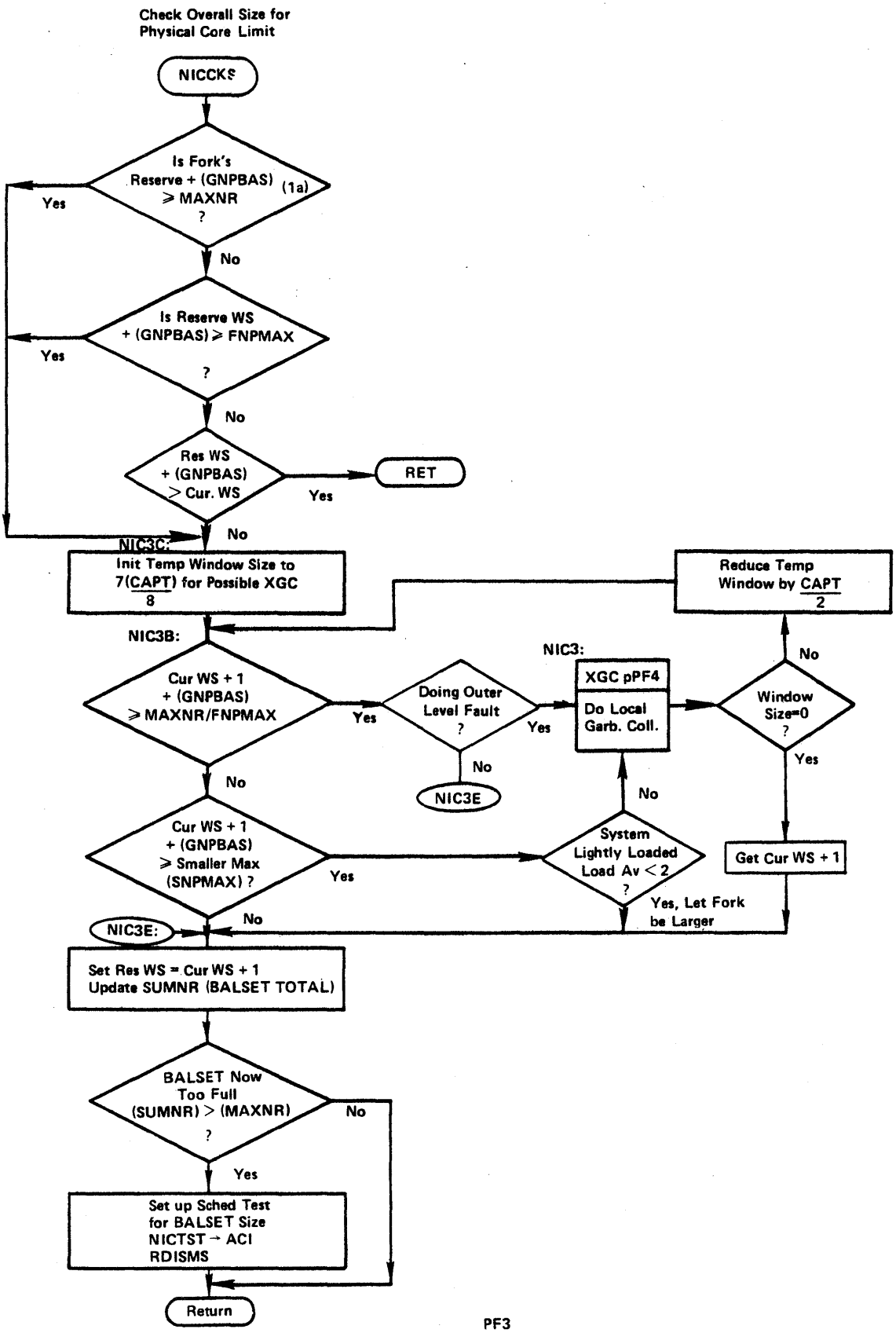
Page OK to Ref.

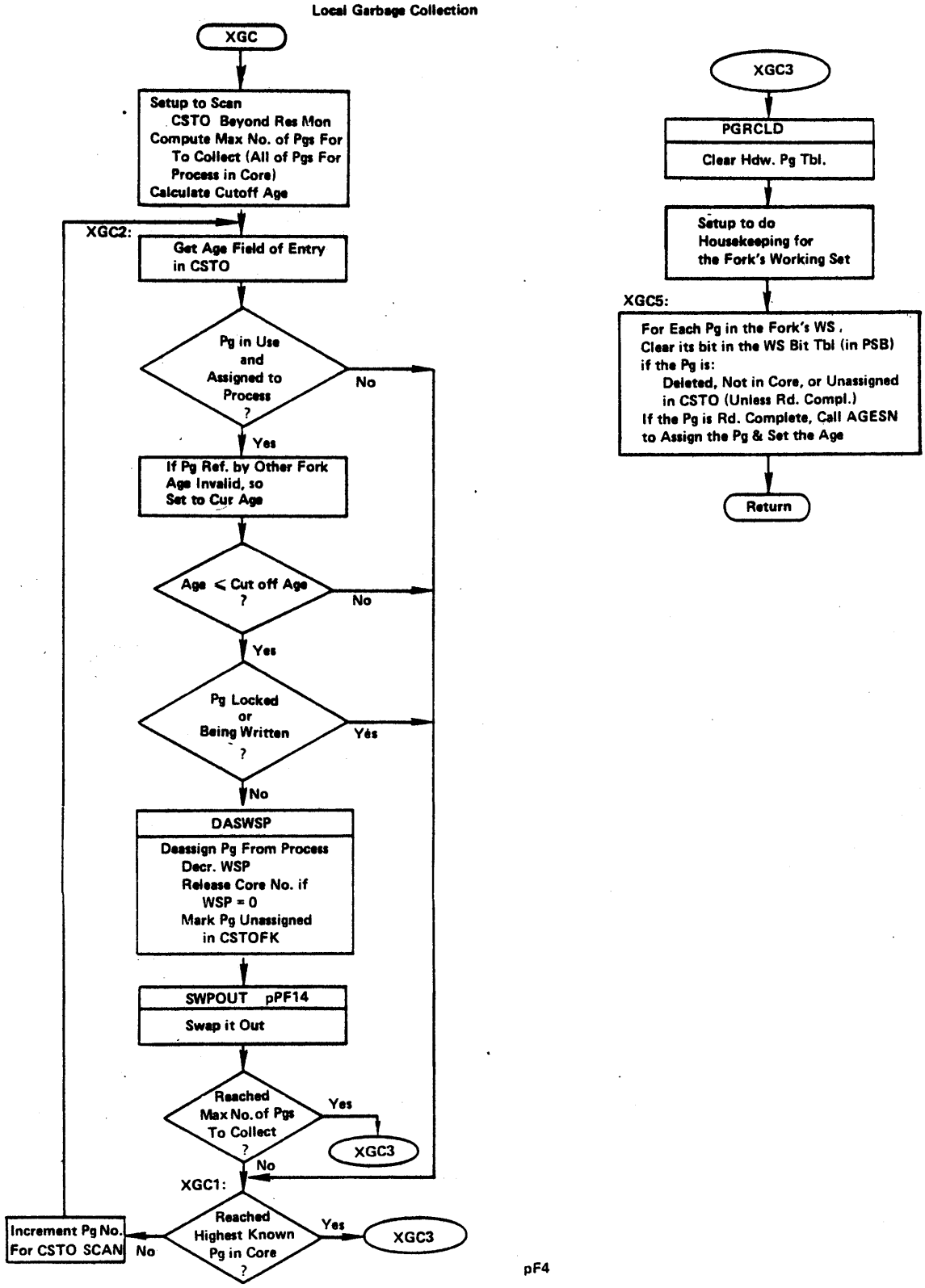


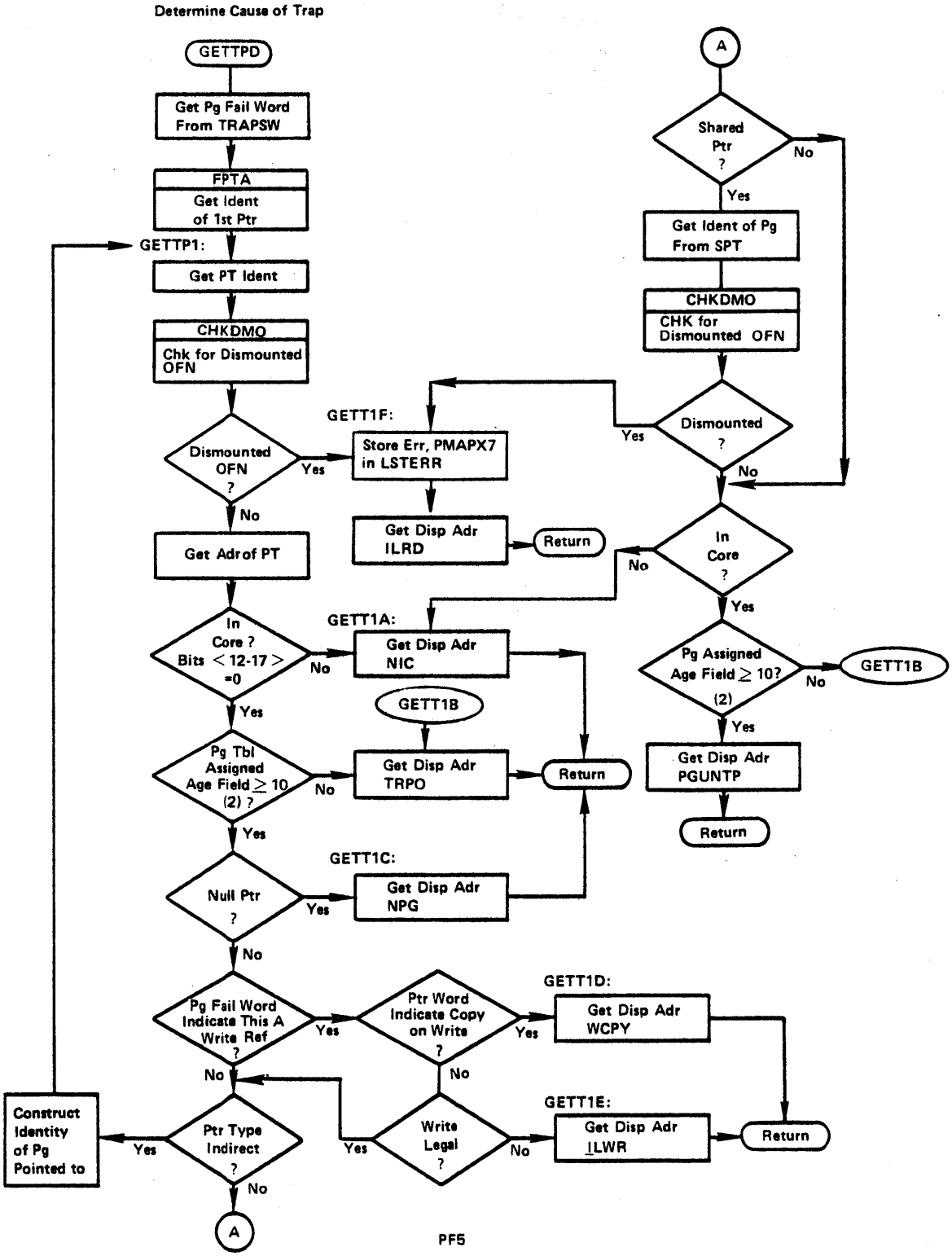
PF1a

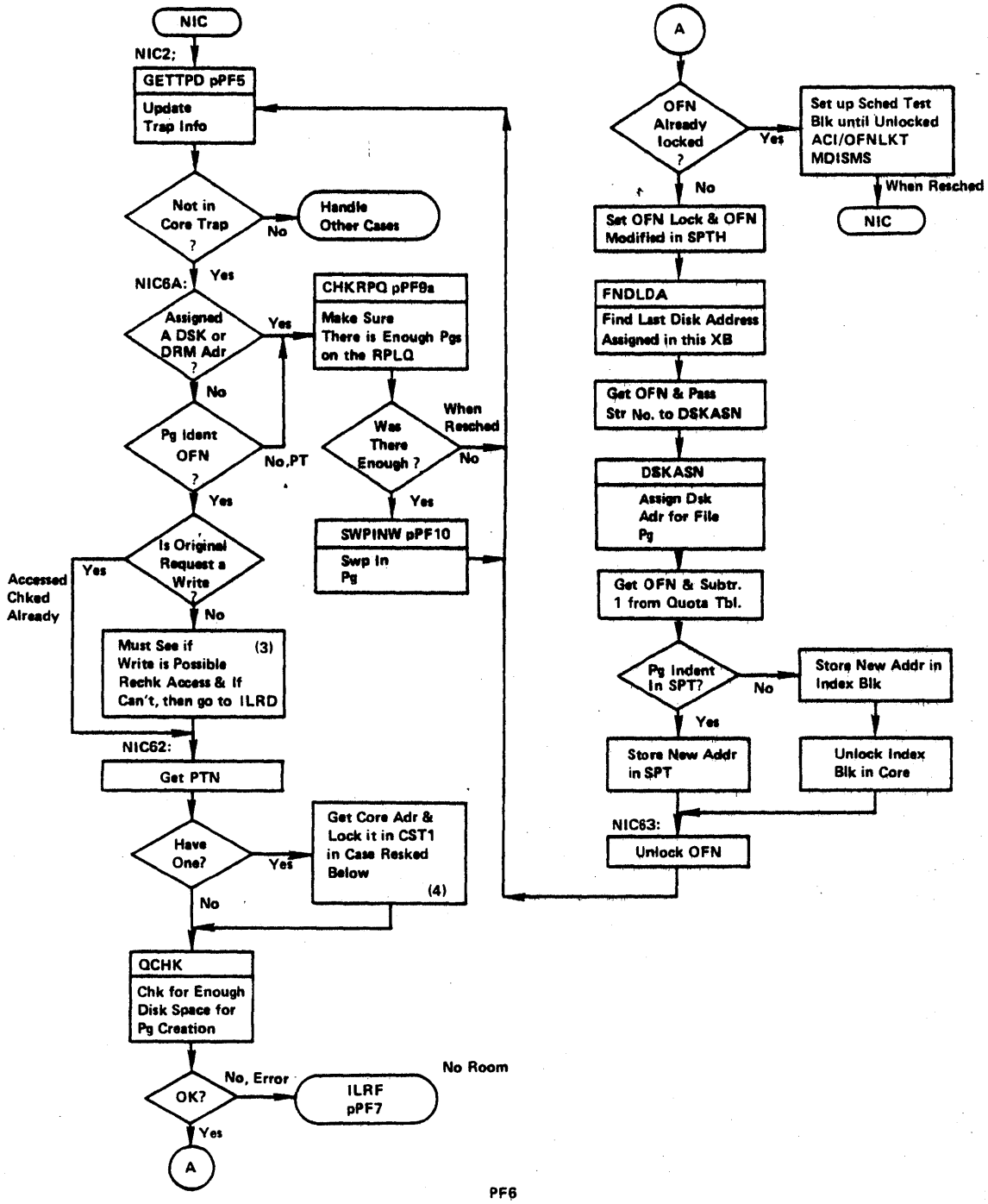

Account for one Pg Trap
PGTACC
Update Interfault Av
Av = (C x Av + Time)/C + 1
Get Subsystem Index
(if any for job) and
account for Pg fault
Get Difference between
Age at last XGC and
Current Age (Adjust for
wrap around if nec.)
Time for Another
XGC (1)
?
No
Yes
Calculate Ratio of Actual
Interfault Avg to Desired
(Desired = 2 x Avg Swp Lat)
Ratio too low
< .5?
Yes
Increase Window Size (CAPT)
(CAPT) + (CAPT)/16 → (CAPT)
No
Ratio too High
≥ 1.5?
No
Yes
Reduce Window Size (CAPT)
(CAPT) – (CAPT)/16 → (CAPT)
See if CAPT within Limits
If > CAPTMX, Set CAPT to CAPTMX
If < CAPTMN, Set CAPT to CAPTMN
Set CAPT to Fixed
Window Size (3 Sec)
A
A
XGC pPF4
Local Garbage Collect
Is Current In Core WS
Near Rerserve WS Amount
Res WS-Cur WS ≤ 4
?
Yes
No
Reduce Reserve WS
and SUMNR by X
X=Res WS-(Cur WS + 4)
NICMGI:
NICCKS pPF3
Check for
Allowable Size
Set Bit for Faulted
Pg in the Fork's WS
Bit Table in the PSB
Ret

PF2

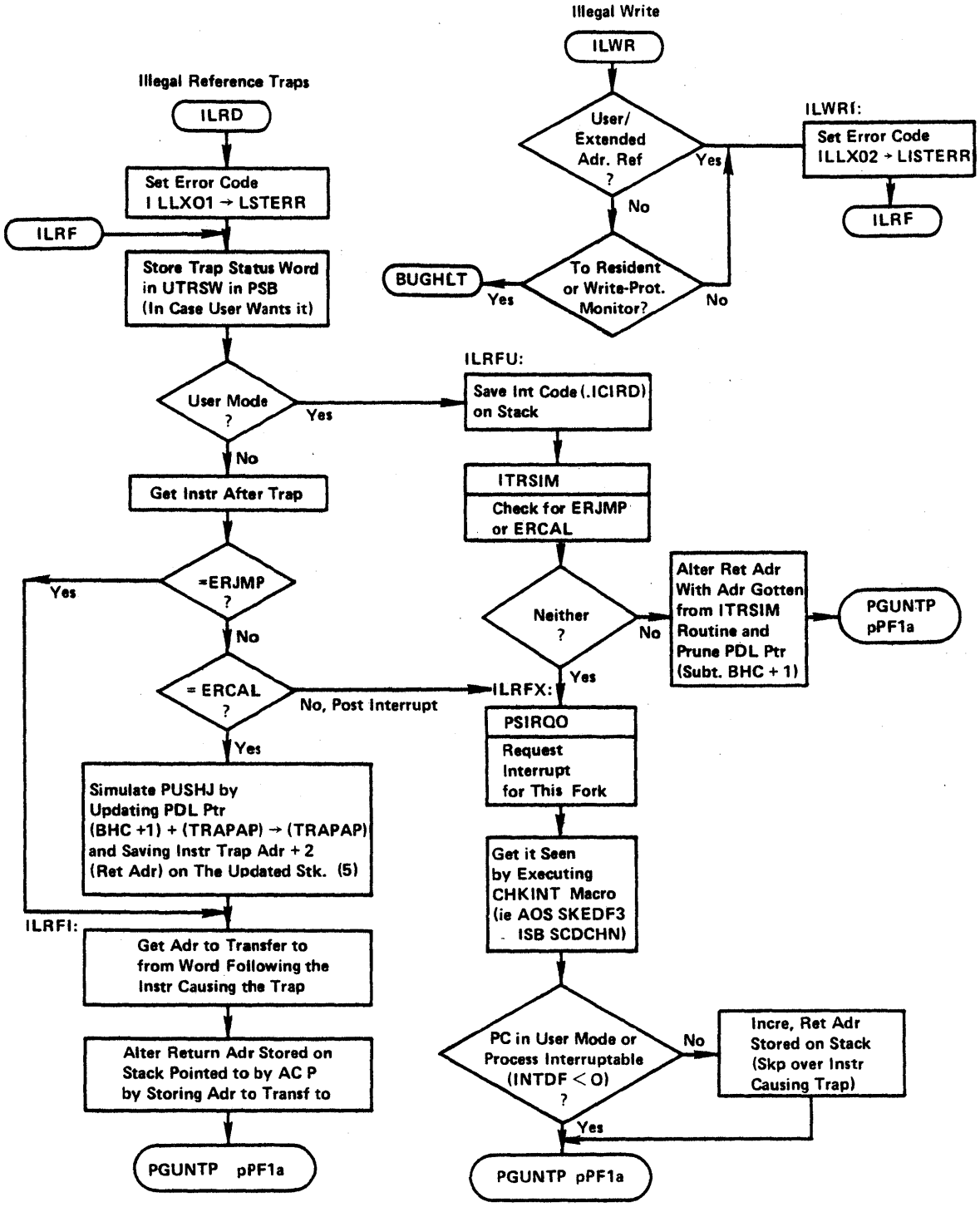

Check Overall Size for Physical Core Limit
NICCK?
Is Fork's Reserve + (GNPBAS) ≥ MAXNR ? (1a)
Yes
No
Is Reserve WS + (GNPBAS) ≥ FNPMAX ?
Yes
No
Res WS + (GNPBAS) > Cur. WS
Yes
RET
No
NIC3C:
Init Temp Window Size to 7(CAPT)/8 for Possible XGC
Reduce Temp Window by CAPT/2
NIC3B:
Cur WS + 1 + (GNPBAS) ≥ MAXNR/FNPMAX
Yes
Doing Outer Level Fault ?
Yes
NIC3:
XGC pPF4
Do Local Garb. Coll.
Window Size=0 ?
No
Yes
No
NIC3E
No
Cur WS + 1 + (GNPBAS) ≥ Smaller Max (SNPMAX) ?
Yes
System Lightly Loaded Load Av < 2 ?
Yes, Let Fork be Larger
Get Cur WS + 1
No
NIC3E:
Set Res WS = Cur WS + 1
Update SUMNR (BALSET TOTAL)
BALSET Now Too Full (SUMNR) > (MAXNR) ?
No
Yes
Set up Sched Test for BALSET Size
NICTST → ACI
RDISMS
Return
PF3


Local Garbage Collection
XGC
Setup to Scan
CSTO Beyond Res Mon
Compute Max No. of Pgs For
To Collect (All of Pgs For
Process in Core)
Calculate Cutoff Age
XGC2:
Get Age Field of Entry
in CSTO
Pg in Use
and
Assigned to
Process
?
No
Yes
If Pg Ref. by Other Fork
Age Invalid, so
Set to Cur Age
Age ≤ Cut off Age
?
No
Yes
Pg Locked
or
Being Written
?
Yes
No
DASWSP
Deassign Pg From Process
Decr. WSP
Release Core No. if
WSP = 0
Mark Pg Unassigned
in CSTOFK
SWPOUT pPF14
Swap it Out
Reached
Max No. of Pgs
To Collect
?
Yes
XGC3
No
XGC1:
Reached
Highest Known
Pg in Core
?
Yes
XGC3
Increment Pg No.
For CSTO SCAN
No
XGC3
PGRCLD
Clear Hdw. Pg Tbl.
Setup to do
Housekeeping for
the Fork's Working Set
XGC5:
For Each Pg in the Fork's WS ,
Clear its bit in the WS Bit Tbl (in PSB)
if the Pg is:
Deleted, Not in Core, or Unassigned
in CSTO (Unless Rd. Compl.)
If the Pg is Rd. Complete, Call AGESN
to Assign the Pg & Set the Age
Return
pF4

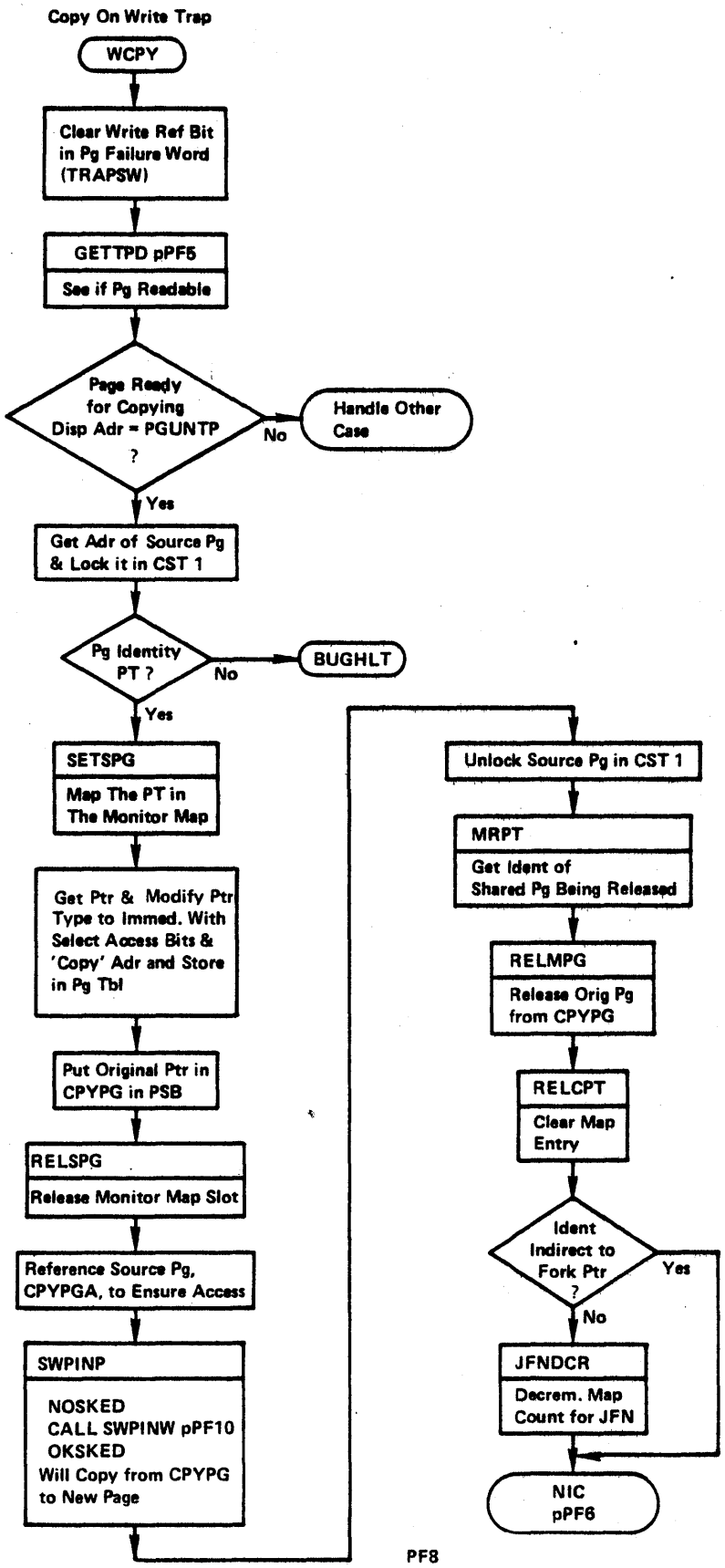

Determine Cause of Trap
GETTPD
Get Pg Fail Word From TRAPSW
FPTA
Get Ident of 1st Ptr
GETTP1:
Get PT Ident
CHKDMO
Chk for Dismounted OFN
Dismounted OFN ?
Yes
No
GETT1F:
Store Err, PMAPX7 in LSTERR
Get Disp Adr ILRD
Return
Get Adrof PT
In Core ? Bits < 12-17 > =0
No
Yes
GETT1A:
Get Disp Adr NIC
GETT1B
Pg Tbl Assigned Age Field ≥ 10 (2) ?
Get Disp Adr TRPO
Return
GETT1C:
Null Ptr ?
Get Disp Adr NPG
Pg Fail Word Indicate This A Write Ref ?
Ptr Word Indicate Copy on Write ?
GETT1D:
Get Disp Adr WCPY
Construct Identity of Pg Pointed to
Ptr Type Indirect ?
Write Legal ?
GETT1E:
Get Disp Adr ILWR
A
Shared Ptr ?
Get Ident of Pg From SPT
CHKDMO
CHK for Dismounted OFN
Dismounted ?
In Core ?
Pg Assigned Age Field ≥ 10? (2)
Get Disp Adr PGUNTP
PF5


NIC
NIC2;
GETTPD pPF5
Update Trap Info
Not in Core Trap ?
No
Handle Other Cases
Yes
NIC6A:
Assigned A DSK or DRM Adr ?
Yes
CHKRPQ pPF9a
Make Sure There is Enough Pgs on the RPLQ
No
Pg Ident OFN ?
No,PT
Was There Enough ?
When Reached
No
Yes
SWPINW pPF10
Swp In Pg
Yes
Is Original Request a Write ?
Yes
Accessed Chked Already
No
Must See if Write is Possible Rechk Access & If Can't, then go to ILRD
(3)
NIC62:
Get PTN
Have One?
Yes
Get Core Adr & Lock it in CST1 in Case Resked Below
(4)
No
QCHK
Chk for Enough Disk Space for Pg Creation
OK?
No, Error
ILRF pPF7
No Room
Yes
A
A
OFN Already locked ?
Yes
Set up Sched Test Blk until Unlocked ACI/OFNLKT MDISMS
When Resched
NIC
No
Set OFN Lock & OFN Modified in SPTH
FNDLDA
Find Last Disk Address Assigned in this XB
Get OFN & Pass Str No. to DSKASN
DSKASN
Assign Dsk Adr for File Pg
Get OFN & Subtr. 1 from Quota Tbl.
Pg Indent In SPT?
No
Store New Addr in Index Blk
Yes
Store New Addr in SPT
Unlock Index Blk in Core
NIC63:
Unlock OFN

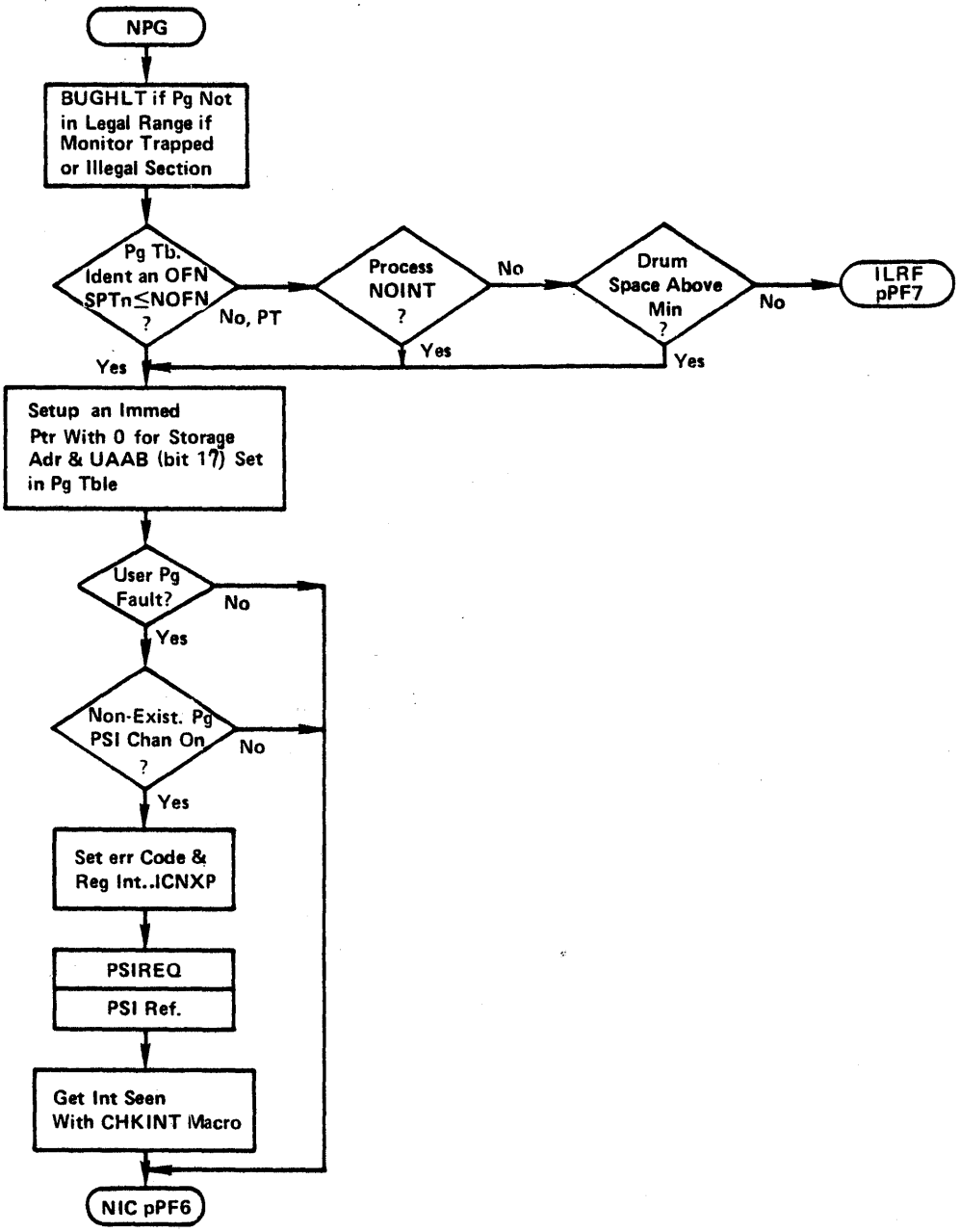
PF6


Illegal Write
ILWR
User/ Extended Adr. Ref ?
Yes
ILWRI:
Set Error Code ILLX02 → LISTERR
ILRF
No
To Resident or Write-Prot. Monitor?
Yes
BUGHLT
No
Illegal Reference Traps
ILRD
Set Error Code I LLXO1 → LSTERR
ILRF
Store Trap Status Word in UTRSW in PSB (In Case User Wants it)
ILRFU:
User Mode ?
Yes
Save Int Code (.ICIRD) on Stack
No
Get Instr After Trap
ITRSIM
Check for ERJMP or ERCAL
=ERJMP ?
Yes
Neither ?
No
Alter Ret Adr With Adr Gotten from ITRSIM Routine and Prune PDL Ptr (Subt. BHC + 1)
PGUNTP pPF1a
No
Yes
= ERCAL ?
No, Post Interrupt
ILRFX:
PSIRQO
Request Interrupt for This Fork
Yes
Simulate PUSHJ by Updating PDL Ptr (BHC +1) + (TRAPAP) → (TRAPAP) and Saving Instr Trap Adr + 2 (Ret Adr) on The Updated Stk. (5)
Get it Seen by Executing CHKINT Macro (ie AOS SKEDF3 ISB SCDCHN)
ILRFI:
Get Adr to Transfer to from Word Following the Instr Causing the Trap
PC in User Mode or Process Interruptable (INTDF < O) ?
No
Incre, Ret Adr Stored on Stack (Skp over Instr Causing Trap)
Alter Return Adr Stored on Stack Pointed to by AC P by Storing Adr to Transf to
Yes
PGUNTP pPF1a
PGUNTP pPF1a

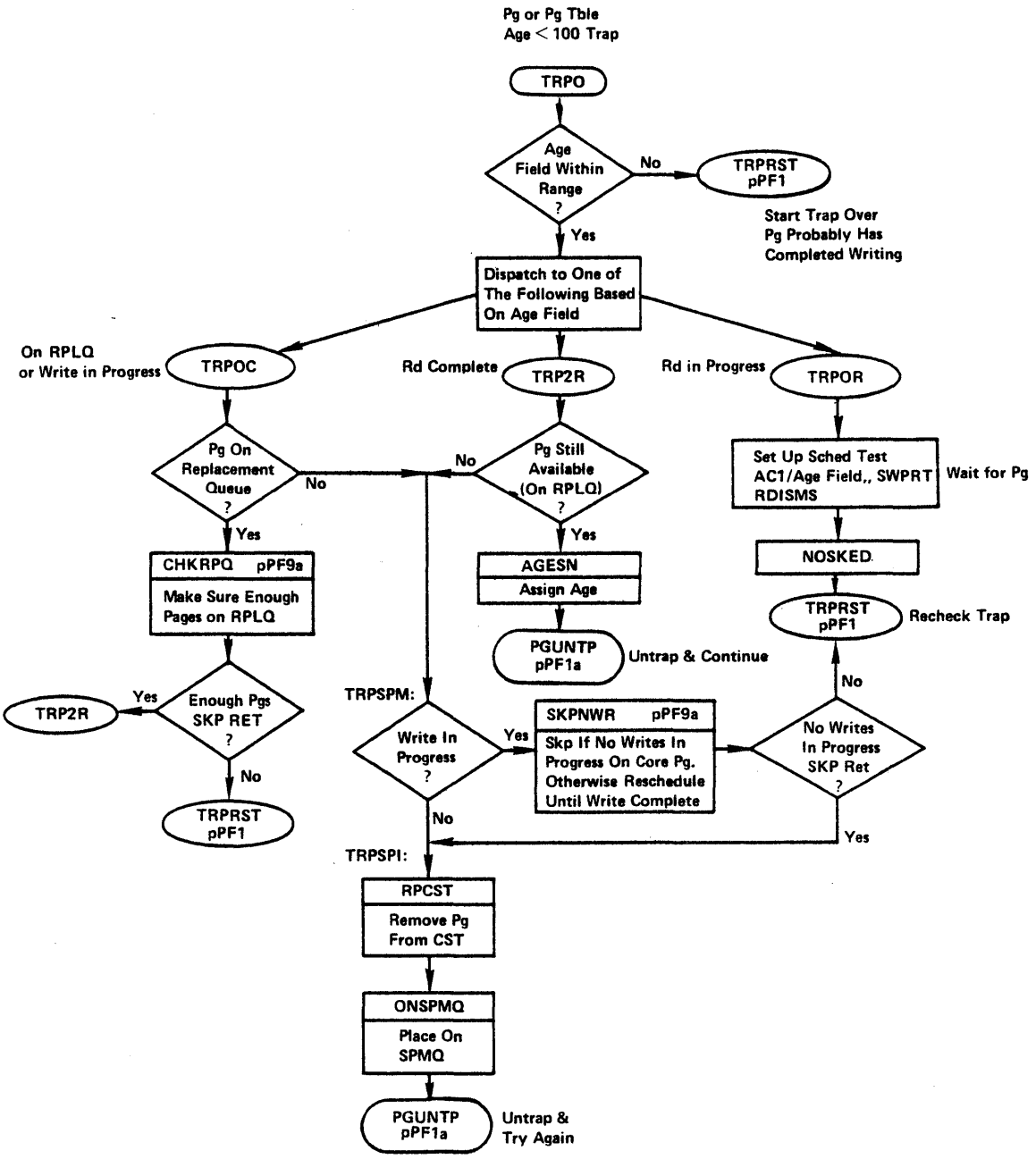
PF7


Copy On Write Trap
WCPY
Clear Write Ref Bit in Pg Failure Word (TRAPSW)
GETTPD pPF5
See if Pg Readable
Page Ready for Copying Disp Adr = PGUNTP ?
No
Handle Other Case
Yes
Get Adr of Source Pg & Lock it in CST 1
Pg Identity PT ?
No
BUGHLT
Yes
SETSPG
Map The PT in The Monitor Map
Get Ptr & Modify Ptr Type to Immed. With Select Access Bits & 'Copy' Adr and Store in Pg Tbl
Put Original Ptr in CPYPG in PSB
RELSPG
Release Monitor Map Slot
Reference Source Pg, CPYPGA, to Ensure Access
SWPINP
NOSKED
CALL SWPINW pPF10
OKSKED
Will Copy from CPYPG to New Page
PF8
Unlock Source Pg in CST 1
MRPT
Get Ident of Shared Pg Being Released
RELMPG
Release Orig Pg from CPYPG
RELCPT
Clear Map Entry
Ident Indirect to Fork Ptr ?
Yes
No
JFNDCR
Decrem. Map Count for JFN
NIC
pPF6

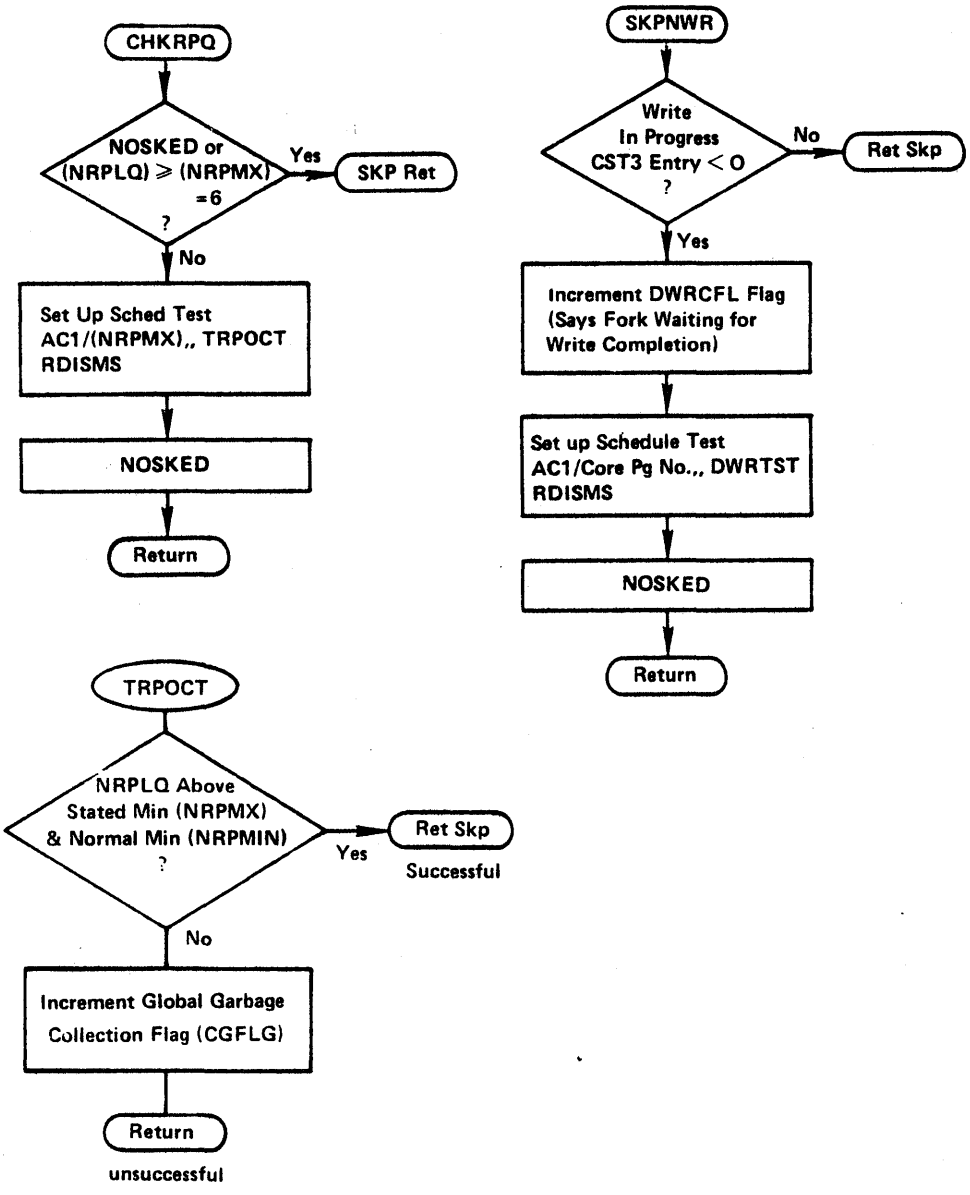
Pg Not in Existance Trap



PF 8a


Pg or Pg Tble
Age < 100 Trap
TRPO
Age
Field Within
Range
?
No
TRPRST
pPF1
Start Trap Over
Pg Probably Has
Completed Writing
Yes
Dispatch to One of
The Following Based
On Age Field
On RPLQ
or Write in Progress
TRPOC
Rd Complete
TRP2R
Rd in Progress
TRPOR
Pg On
Replacement
Queue
?
No
Yes
Pg Still
Available
(On RPLQ)
?
No
Yes
Set Up Sched Test
AC1/Age Field,, SWPRT
RDISMS
Wait for Pg
NOSKED
CHKRPQ pPF9a
Make Sure Enough
Pages on RPLQ
AGESN
Assign Age
TRPRST
pPF1
Recheck Trap
PGUNTP
pPF1a
Untrap & Continue
Yes
TRP2R
Enough Pgs
SKP RET
?
No
TRPRST
pPF1
TRPSPM:
Write In
Progress
?
Yes
SKPNWR pPF9a
Skp If No Writes In
Progress On Core Pg.
Otherwise Reschedule
Until Write Complete
No Writes
In Progress
SKP Ret
?
No
Yes
No
TRPSPI:
RPCST
Remove Pg
From CST
ONSPMQ
Place On
SPMQ
PGUNTP
pPF1a
Untrap &
Try Again

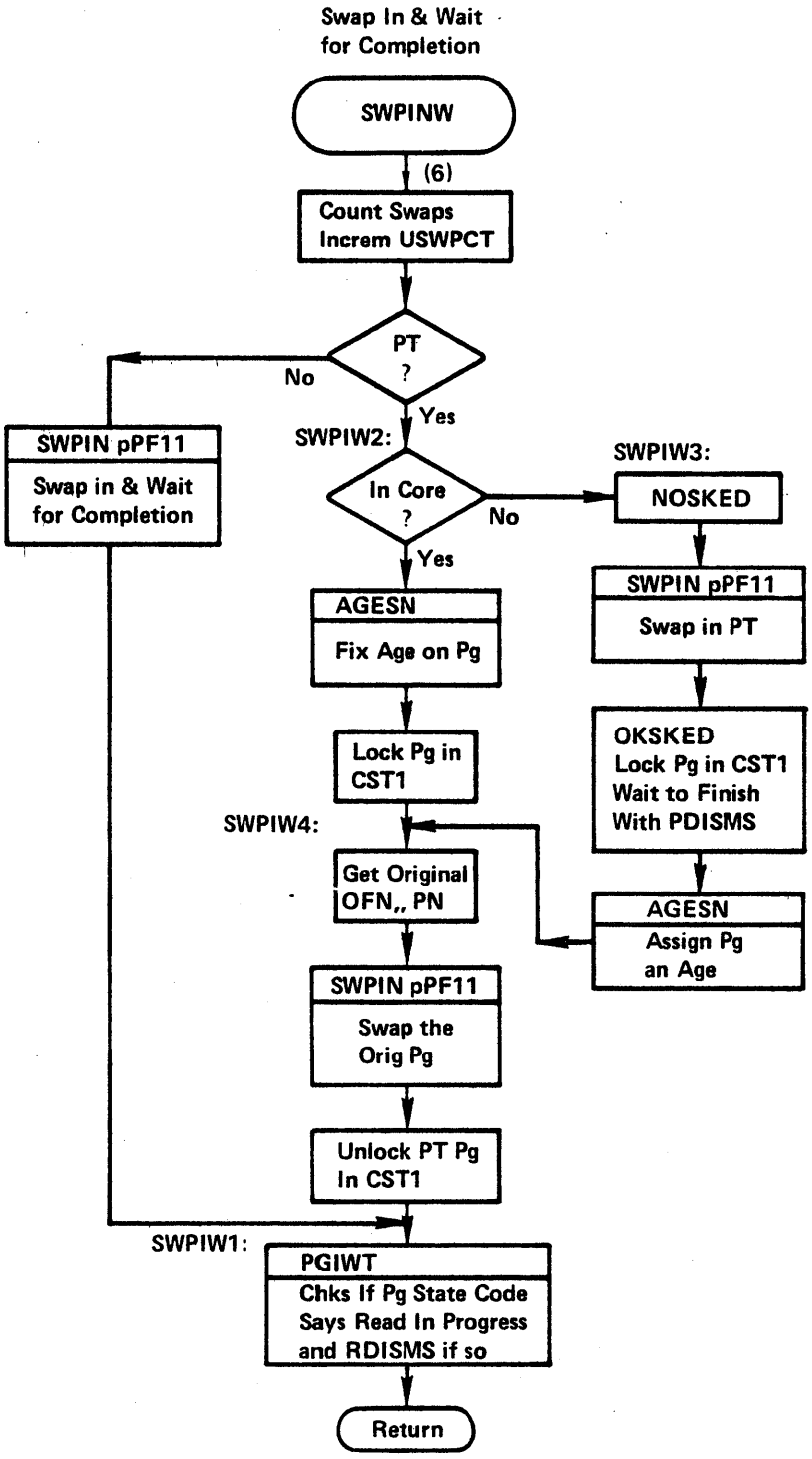
PF9


CHKRPQ
NOSKED or (NRPLQ) ≥ (NRPMX) =6 ?
Yes
SKP Ret
No
Set Up Sched Test AC1/(NRPMX),, TRPOCT RDISMS
NOSKED
Return
SKPNWR
Write In Progress CST3 Entry < O ?
No
Ret Skp
Yes
Increment DWRCFL Flag (Says Fork Waiting for Write Completion)
Set up Schedule Test AC1/Core Pg No.,, DWRTST RDISMS
NOSKED
Return
TRPOCT
NRPLQ Above Stated Min (NRPMX) & Normal Min (NRPMIN) ?
Yes
Ret Skp
Successful
No
Increment Global Garbage Collection Flag (CGFLG)
Return
unsuccessful

PF9a

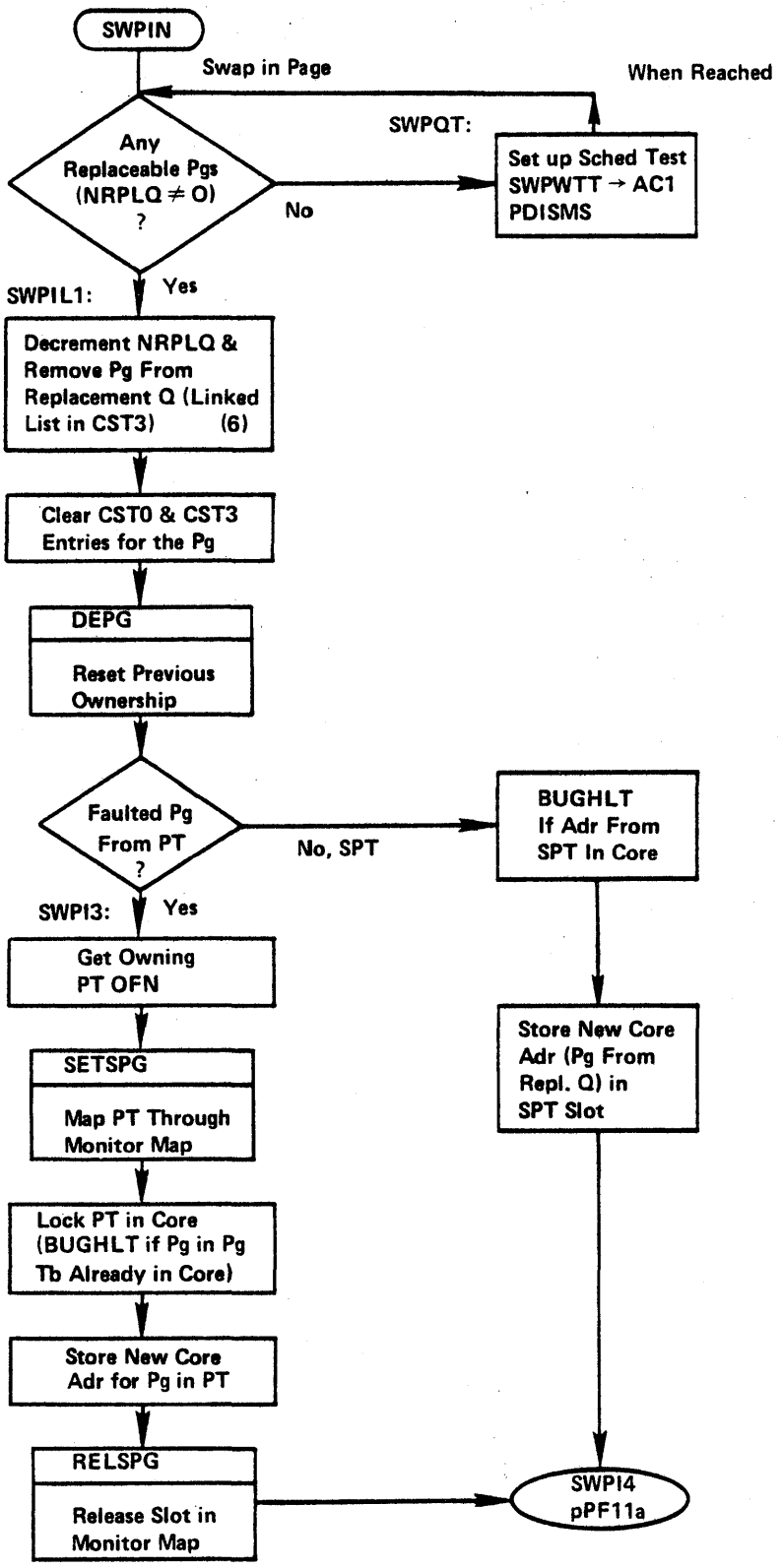
REQUESTING DRUM OR DISK READ

(PAGEM LEVEL)



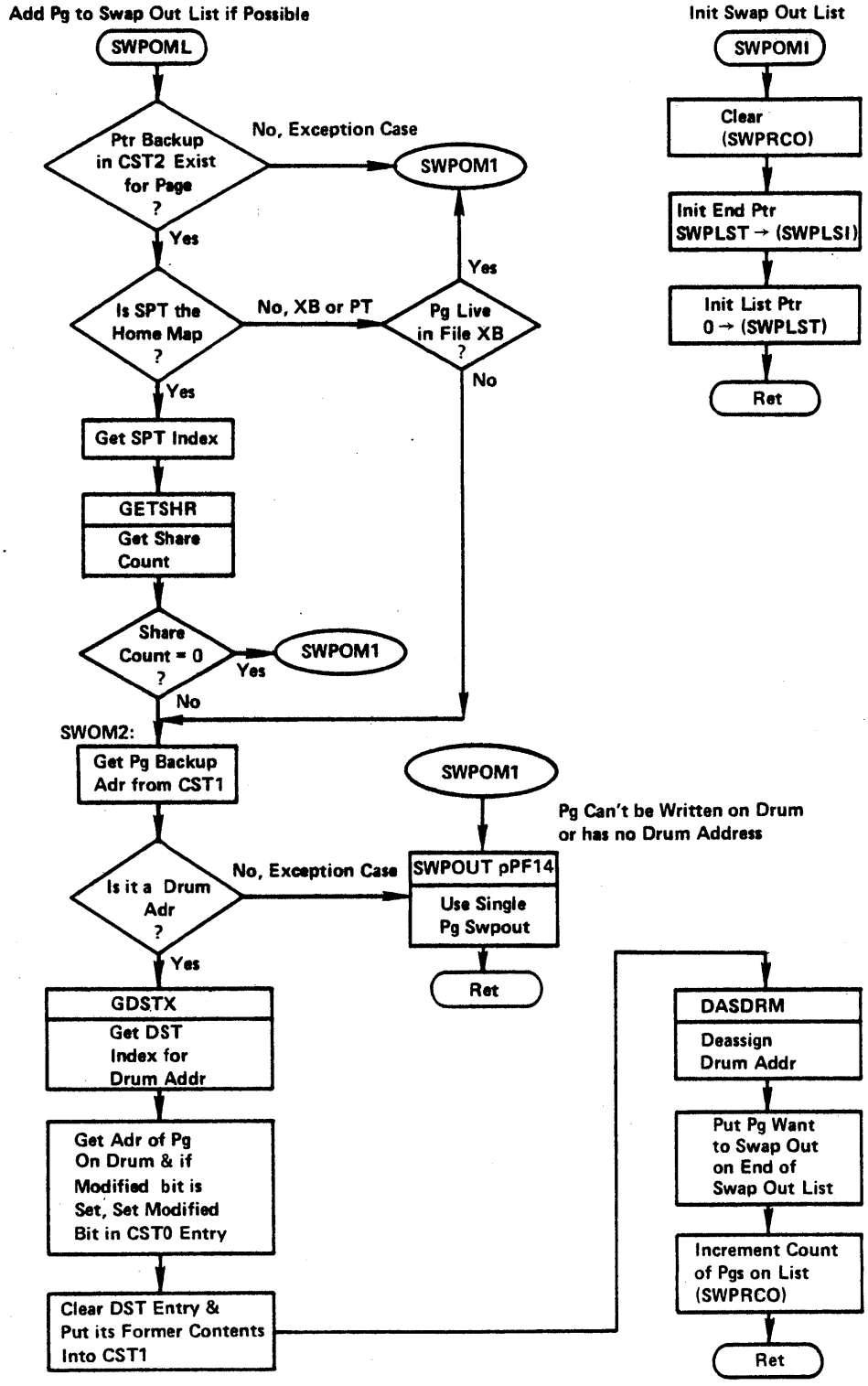
PF10

REQUESTING DRUM OR DISK READ (Continued)
(PAGEM LEVEL)

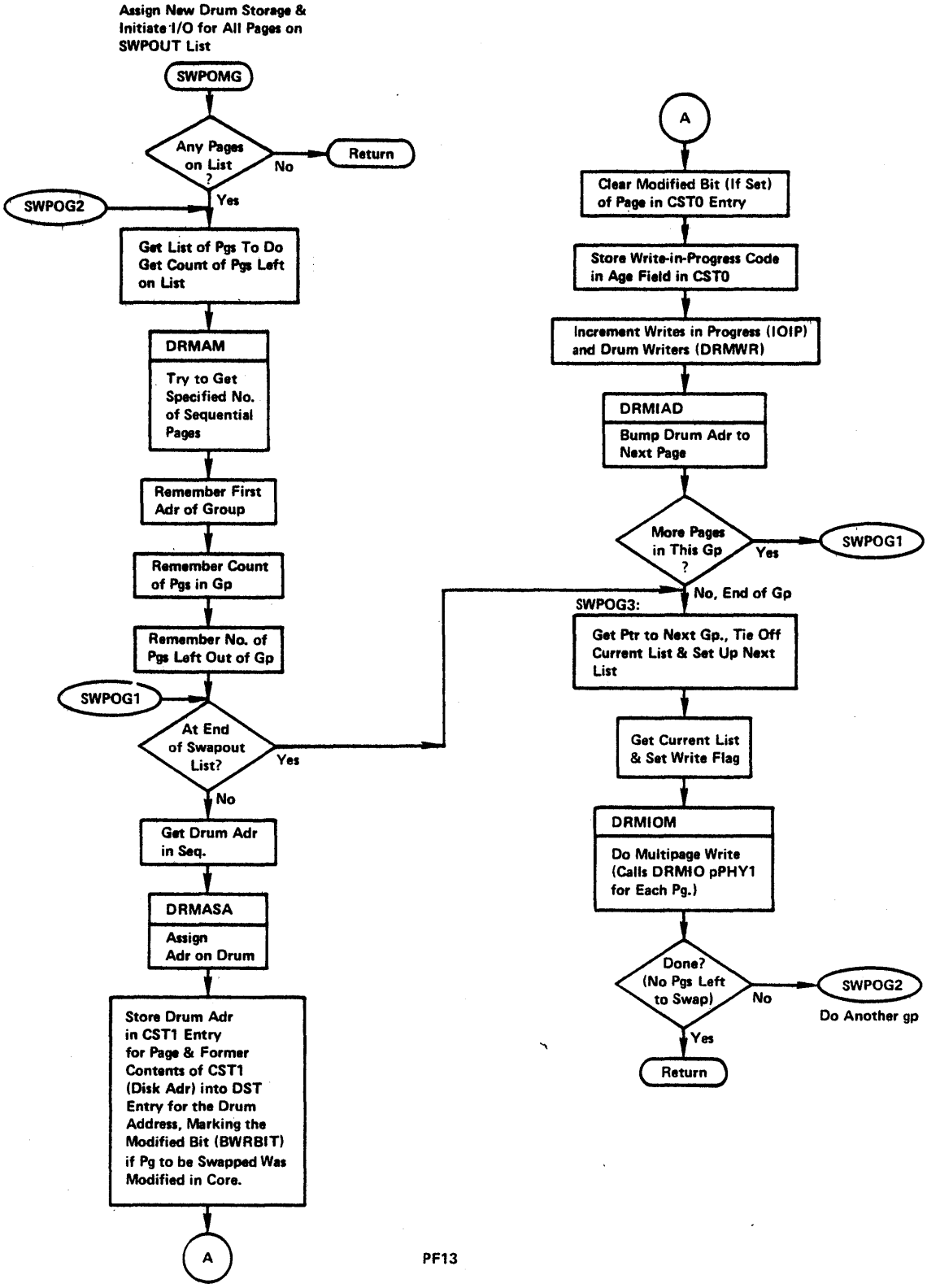


PF11

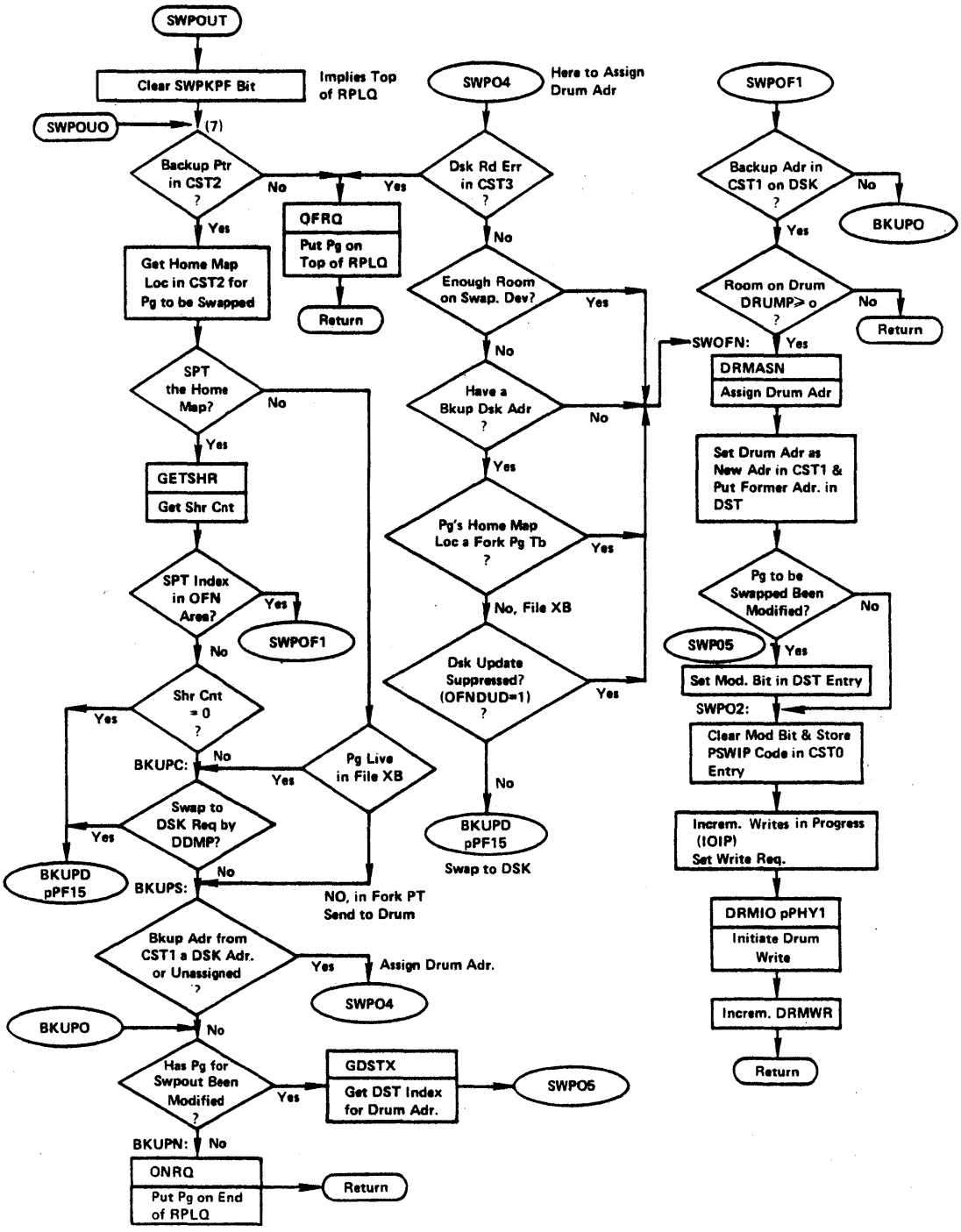
MULTIPLE PAGE SWAP OUT ROUTINE
SWPOMI - Init List
SWPOML - Called to Add Page to Swap Out List if Possible
SWPOMG - To Begin I/O for All Pages on Swap Out List
SWPOUT - Initiate Swap Out of Single Page



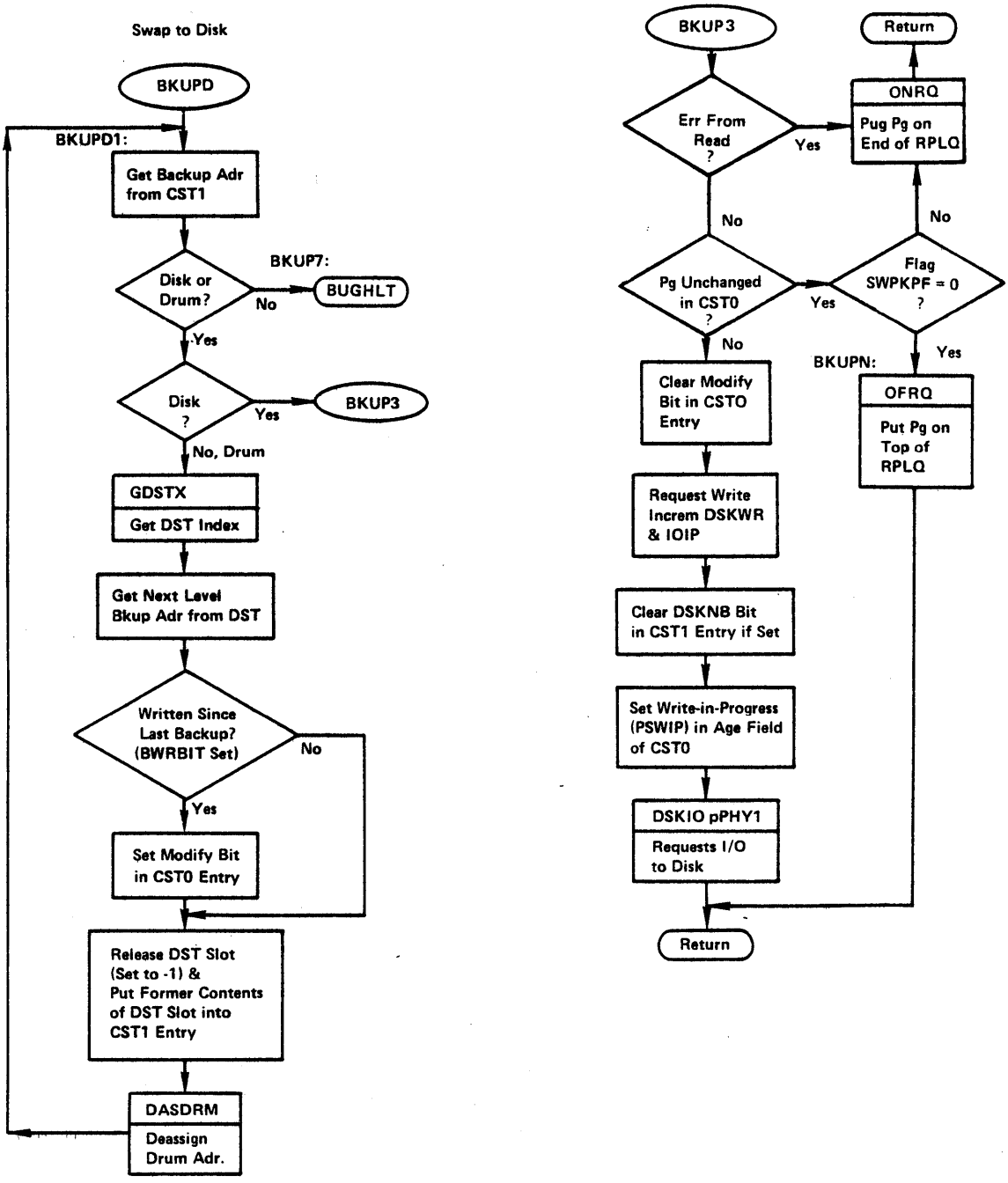
PF12


Assign New Drum Storage & Initiate I/O for All Pages on SWPOUT List
SWPOMG
Any Pages on List ?
No
Return
Yes
SWPOG2
Get List of Pgs To Do Get Count of Pgs Left on List
DRMAM
Try to Get Specified No. of Sequential Pages
Remember First Adr of Group
Remember Count of Pgs in Gp
Remember No. of Pgs Left Out of Gp
SWPOG1
At End of Swapout List?
Yes
No
Get Drum Adr in Seq.
DRMASA
Assign Adr on Drum
Store Drum Adr in CST1 Entry for Page & Former Contents of CST1 (Disk Adr) into DST Entry for the Drum Address, Marking the Modified Bit (BWRBIT) if Pg to be Swapped Was Modified in Core.
A
A
Clear Modified Bit (If Set) of Page in CST0 Entry
Store Write-in-Progress Code in Age Field in CST0
Increment Writes in Progress (IOIP) and Drum Writers (DRMWR)
DRMIAD
Bump Drum Adr to Next Page
More Pages in This Gp ?
Yes
SWPOG1
No, End of Gp
SWPOG3:
Get Ptr to Next Gp., Tie Off Current List & Set Up Next List
Get Current List & Set Write Flag
DRMIOM
Do Multipage Write (Calls DRMIO pPHY1 for Each Pg.)
Done? (No Pgs Left to Swap)
No
SWPOG2
Do Another gp
Yes
Return

PF13


SWPOUT
Clear SWPKPF Bit
Implies Top of RPLQ
SWPOUO
(7)
Backup Ptr in CST2 ?
No
Yes
QFRQ
Put Pg on Top of RPLQ
Return
Yes
Get Home Map Loc in CST2 for Pg to be Swapped
SPT the Home Map?
No
Yes
GETSHR
Get Shr Cnt
SPT Index in OFN Area?
Yes
SWPOF1
No
Shr Cnt = 0 ?
Yes
No
BKUPC:
Yes
Pg Live in File XB
Swap to DSK Req by DDMP?
Yes
BKUPD pPF15
No
BKUPS:
NO, in Fork PT Send to Drum
Bkup Adr from CST1 a DSK Adr. or Unassigned ?
Yes
Assign Drum Adr.
SWPO4
BKUPO
No
Has Pg for Swpout Been Modified ?
Yes
GDSTX
Get DST Index for Drum Adr.
SWPO5
BKUPN:
No
ONRQ
Put Pg on End of RPLQ
Return
SWPO4
Here to Assign Drum Adr
Dsk Rd Err in CST3 ?
No
Enough Room on Swap. Dev?
Yes
No
Have a Bkup Dsk Adr ?
No
Yes
Pg's Home Map Loc a Fork Pg Tb ?
Yes
No, File XB
Dsk Update Suppressed? (OFNDUD=1) ?
Yes
No
BKUPD pPF15
Swap to DSK
SWPOF1
Backup Adr in CST1 on DSK ?
No
BKUPO
Yes
Room on Drum DRUMP≥ o ?
No
Return
Yes
SWOFN:
DRMASN
Assign Drum Adr
Set Drum Adr as New Adr in CST1 & Put Former Adr. in DST
Pg to be Swapped Been Modified?
No
SWP05
Yes
Set Mod. Bit in DST Entry
SWPO2:
Clear Mod Bit & Store PSWIP Code in CST0 Entry
Increm. Writes in Progress (IOIP)
Set Write Req.
DRMIO pPHY1
Initiate Drum Write
Increm. DRMWR
Return

PF14


Swap to Disk
BKUPD
BKUPD1:
Get Backup Adr from CST1
BKUP7:
Disk or Drum?
No
BUGHLT
Yes
Disk ?
Yes
BKUP3
No, Drum
GDSTX
Get DST Index
Get Next Level Bkup Adr from DST
Written Since Last Backup? (BWRBIT Set)
No
Yes
Set Modify Bit in CST0 Entry
Release DST Slot (Set to -1) & Put Former Contents of DST Slot into CST1 Entry
DASDRM
Deassign Drum Adr.
BKUP3
Err From Read ?
Yes
ONRQ
Pug Pg on End of RPLQ
Return
No
Pg Unchanged in CST0 ?
Yes
Flag SWPKPF = 0 ?
No
Yes
BKUPN:
OFRQ
Put Pg on Top of RPLQ
No
Clear Modify Bit in CSTO Entry
Request Write Increm DSKWR & IOIP
Clear DSKNB Bit in CST1 Entry if Set
Set Write-in-Progress (PSWIP) in Age Field of CST0
DSKIO pPHY1
Requests I/O to Disk
Return

PF15

Page Fault Handling Comments

PGTACC

(1) Checks if process has accrued more than or equal to the number of age ticks of GCRATE. Currently, this is set to 50, which implies 2 sec. of process virtual time (i.e., the age stamp is incremented every 40 ms of process run time).

NICCKS

(1) GNPBAS is currently initialized at system startup to zero and is incremented/decremented only when pages are locked/unlocked. It is currently only tested by NICCKS as well.

GETTPD

(2) The age field when used to hold the age stamp, will always have a value of 100 or greater. This checks if any of the lefthand 6 bits of the age field are set.

NIC

(3) Could take the ILRD path, for example, when OPENed file for write, but PMAPed for each of a nonexistent page. A page would have to be created which would then imply a write which was not enabled under PMAP.

(4) If file page faulted does not have its own SPT slot, but has to be mapped (using indirect pointer) via the index blk slot in the OFN area, then the index blk will be locked in core. (So can't be swapped in case of reschedule.)

(5) Note in the predispatch code that AC1 was stored in BHC + 1 and AC, P, which holds a push down list pointer, was saved in TRAPAP.

SWPINW

(6) SWPINW will invoke SWPIN to swap in a page into a page from the RPLQ. However, this same code can also be entered with different flag settings and be used to swap in a page into a page from the special memory queue (SPMQ), a queue used by the memory error handling code.

SWPOUT

(7) SWPOUO is called from:

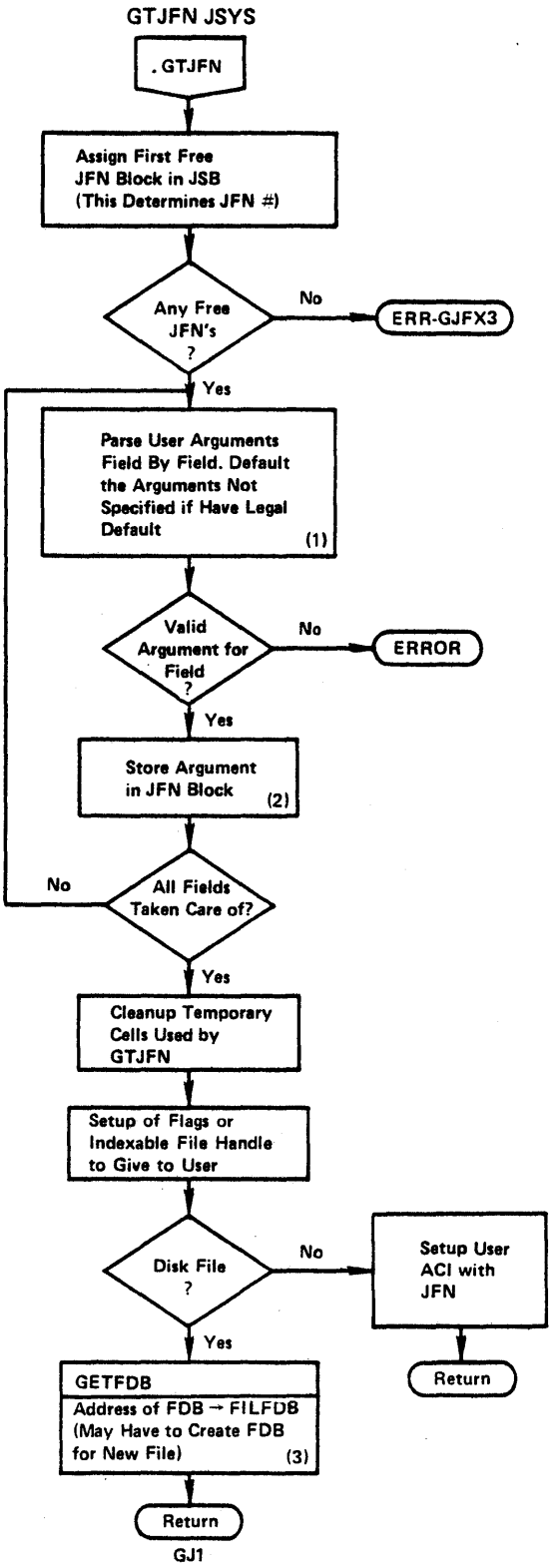
SWPOTO which clears the SWPKPF bit (for top of RPLQ) before calling SWPOUO and

SWPOTK (called from the UPDPGS JSYS) which sets the SWPKPF bit (for end of RPLQ) before calling SWPOUO.

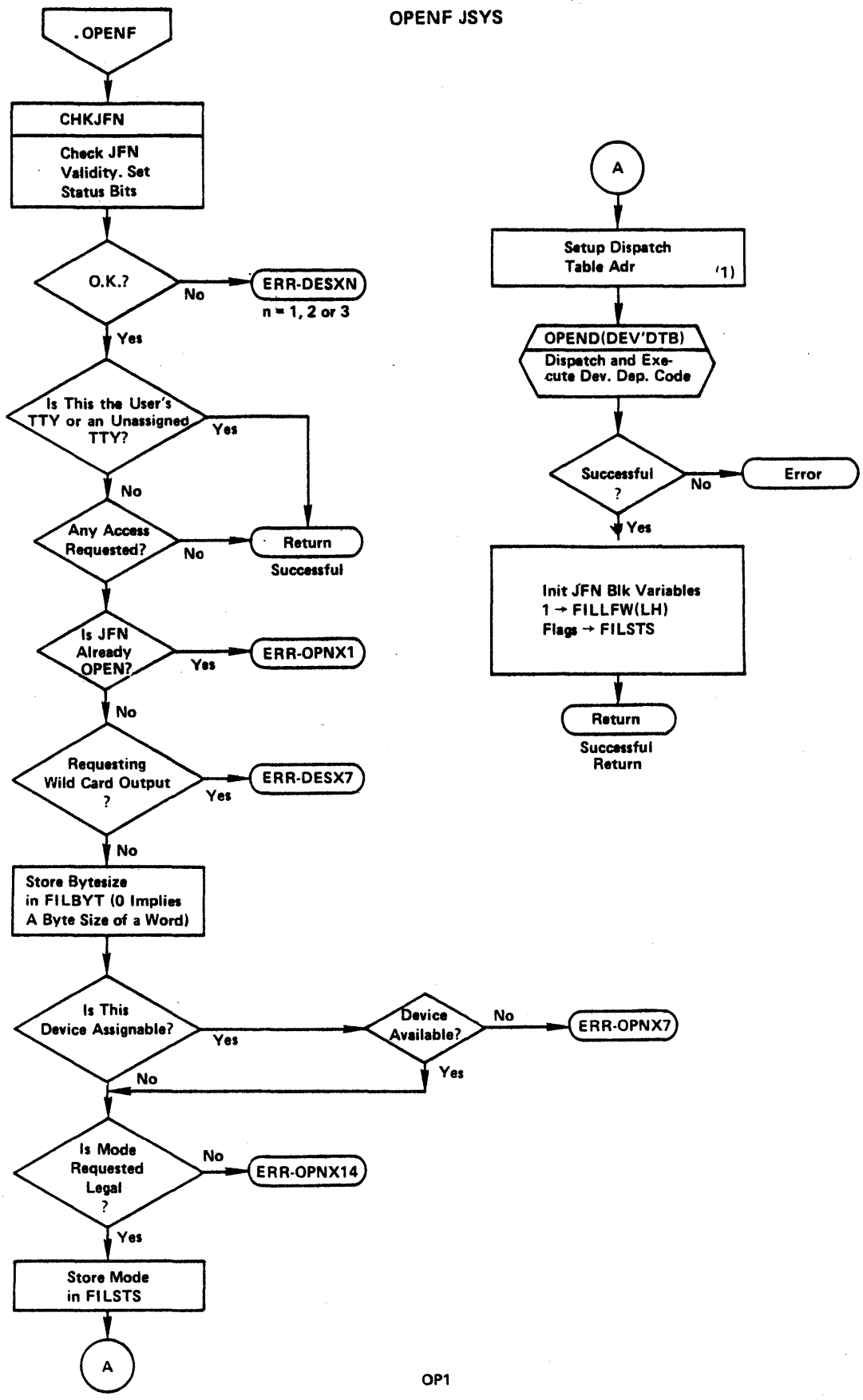
JSYS CALL FLOWCHARTS

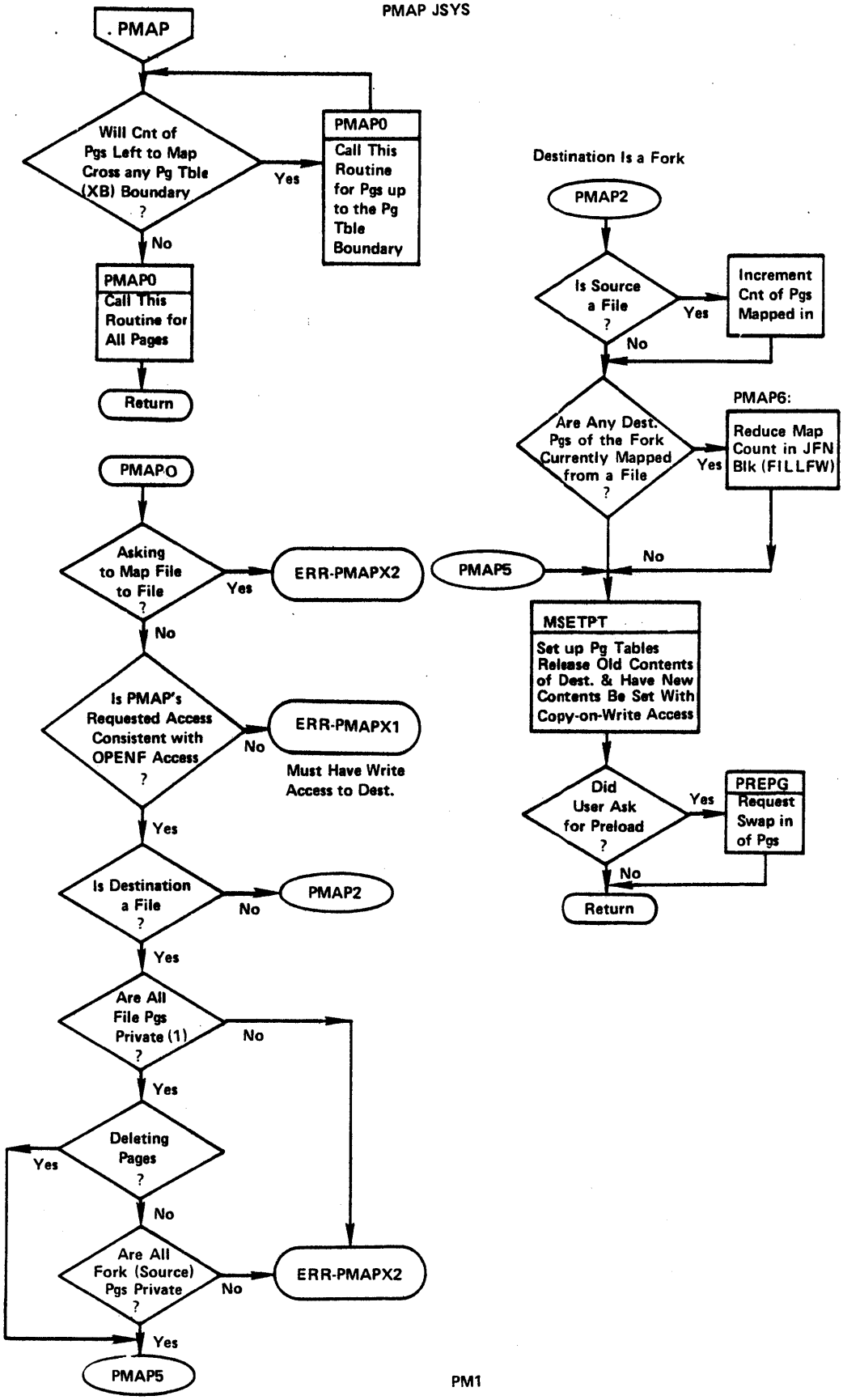
DEVICE INDEPENDENT LEVEL

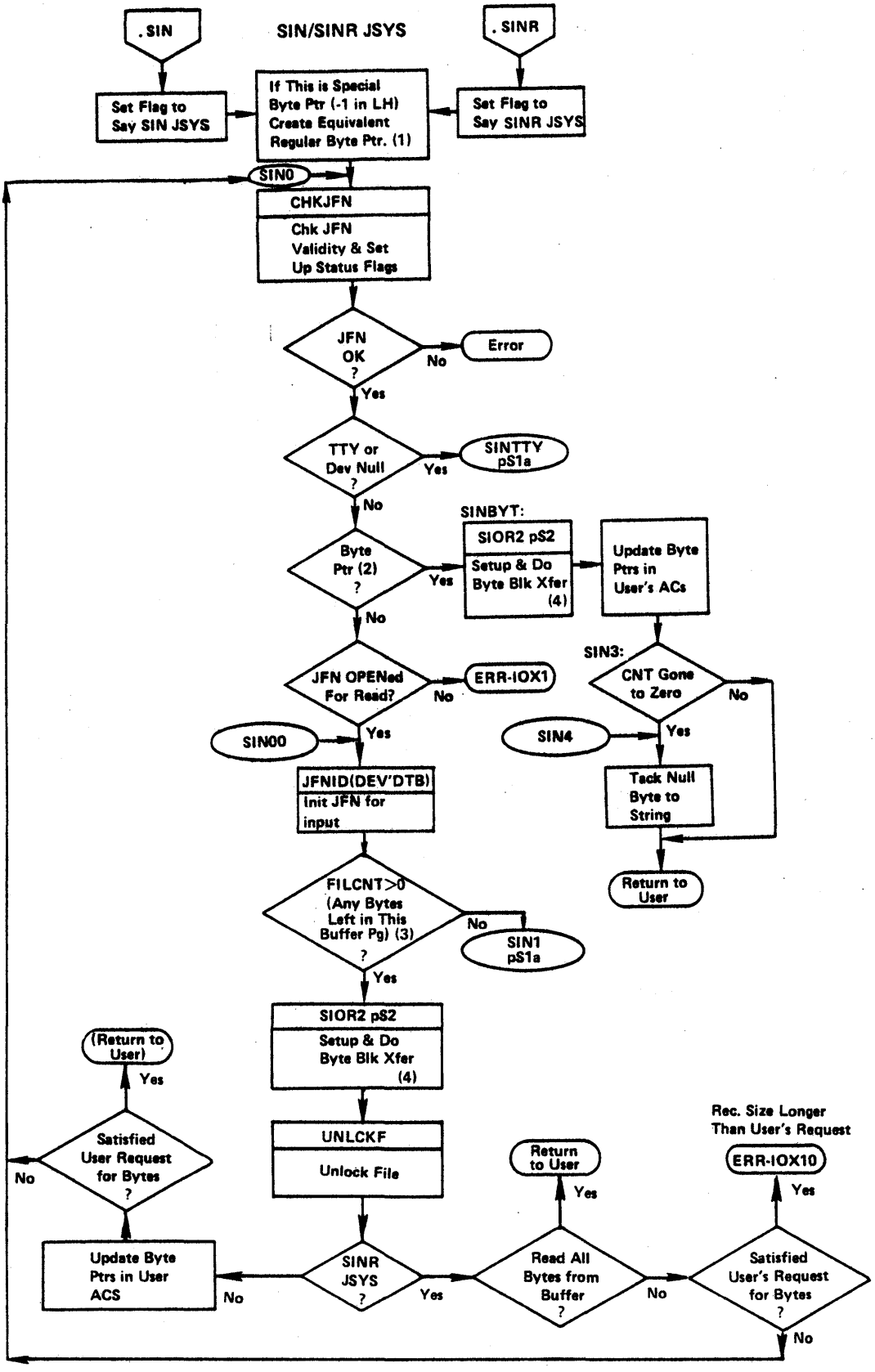
GTJFN -	Get a JFN	GJ1
OPENF -	Open a File	OP1
SIN/SINR-	Sequential Input	S1
	BYTINA - Call Device Dependent Code to Get a Byte	S2
	SIOR2 - String I/O Multiple Byte Transfer	S2
SOUT/SOUTR -	Sequential Output	S3
	BYTOUA - Send Byte to a Service Routine	S4
PMAP -	Map a File or Fork	PM1
UFPGS -	Update File Pages	UD1
CLOSF -	Close a File	CL1


GTJFN JSYS
.GTJFN
Assign First Free JFN Block in JSB (This Determines JFN #)
Any Free JFN's ?
No
ERR-GJFX3
Yes
Parse User Arguments Field By Field. Default the Arguments Not Specified if Have Legal Default
(1)
Valid Argument for Field ?
No
ERROR
Yes
Store Argument in JFN Block
(2)
All Fields Taken Care of?
No
Yes
Cleanup Temporary Cells Used by GTJFN
Setup of Flags or Indexable File Handle to Give to User
Disk File ?
No
Setup User ACI with JFN
Return
Yes
GETFDB
Address of FDB → FILFDB (May Have to Create FDB for New File)
(3)
Return
GJ1

OPENF JSYS

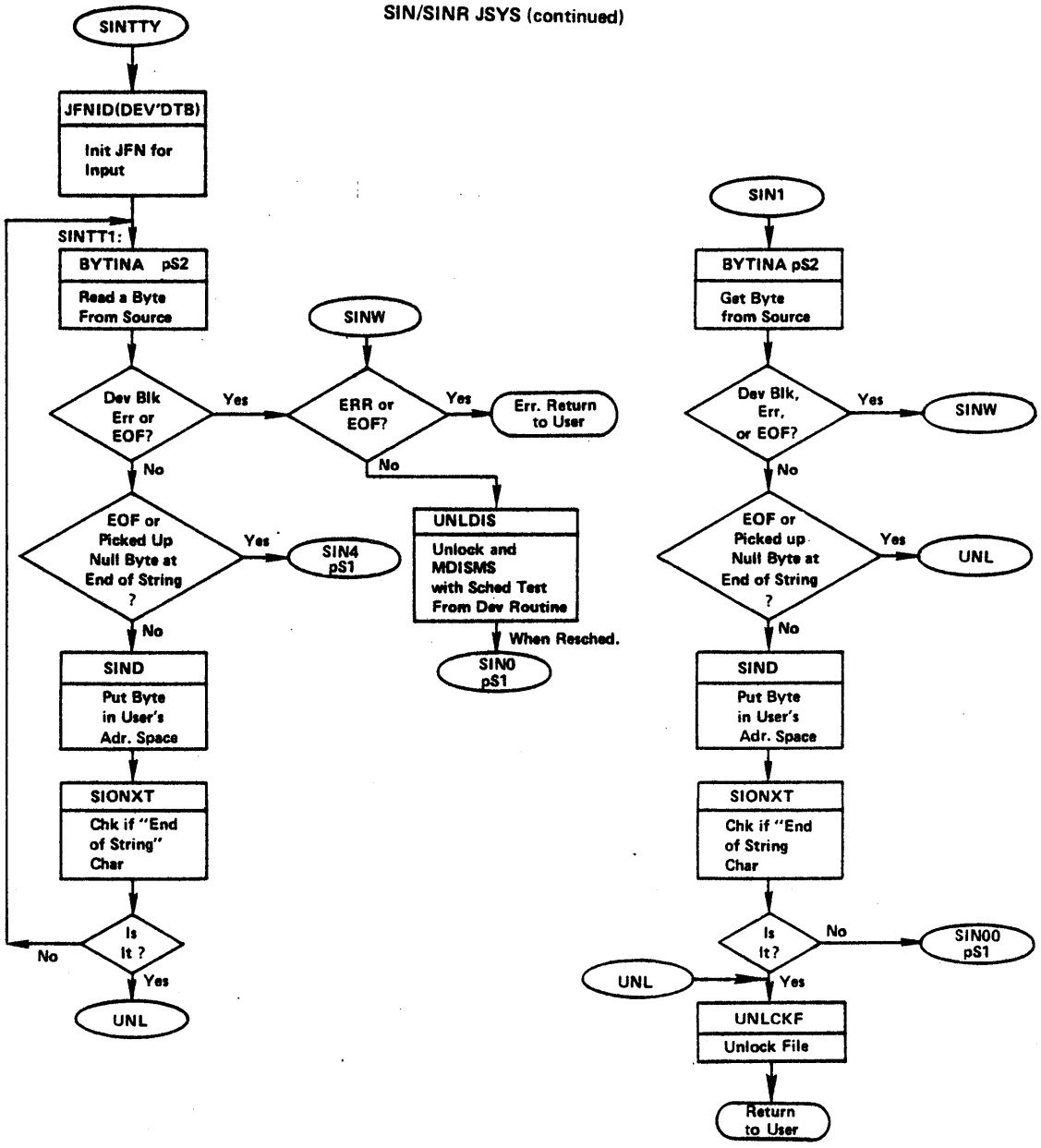

.OPENF
CHKJFN
Check JFN Validity. Set Status Bits
O.K.?
No
ERR-DESXN
n = 1, 2 or 3
Yes
Is This the User's TTY or an Unassigned TTY?
Yes
No
Any Access Requested?
No
Return
Successful
Is JFN Already OPEN?
Yes
ERR-OPNX1
No
Requesting Wild Card Output ?
Yes
ERR-DESX7
No
Store Bytesize in FILBYT (0 Implies A Byte Size of a Word)
Is This Device Assignable?
Yes
Device Available?
No
ERR-OPNX7
Yes
No
Is Mode Requested Legal ?
No
ERR-OPNX14
Yes
Store Mode in FILSTS
A
A
Setup Dispatch Table Adr (1)
OPEND(DEV'DTB)
Dispatch and Execute Dev. Dep. Code
Successful ?
No
Error
Yes
Init JFN Blk Variables
1 → FILLFW(LH)
Flags → FILSTS
Return
Successful Return


. PMAP
Will Cnt of Pgs Left to Map Cross any Pg Tble (XB) Boundary ?
Yes
PMAP0
Call This Routine for Pgs up to the Pg Tble Boundary
No
PMAP0
Call This Routine for All Pages
Return
PMAPO
Asking to Map File to File ?
Yes
ERR-PMAPX2
No
Is PMAP's Requested Access Consistent with OPENF Access ?
No
ERR-PMAPX1
Must Have Write Access to Dest.
Yes
Is Destination a File ?
No
PMAP2
Yes
Are All File Pgs Private (1) ?
No
Yes
Deleting Pages ?
Yes
No
Are All Fork (Source) Pgs Private ?
No
ERR-PMAPX2
Yes
PMAP5
Destination Is a Fork
PMAP2
Is Source a File ?
Yes
Increment Cnt of Pgs Mapped in
No
PMAP6:
Are Any Dest. Pgs of the Fork Currently Mapped from a File ?
Yes
Reduce Map Count in JFN Blk (FILLFW)
No
PMAP5
MSETPT
Set up Pg Tables Release Old Contents of Dest. & Have New Contents Be Set With Copy-on-Write Access
Did User Ask for Preload ?
Yes
PREPG
Request Swap in of Pgs
No
Return

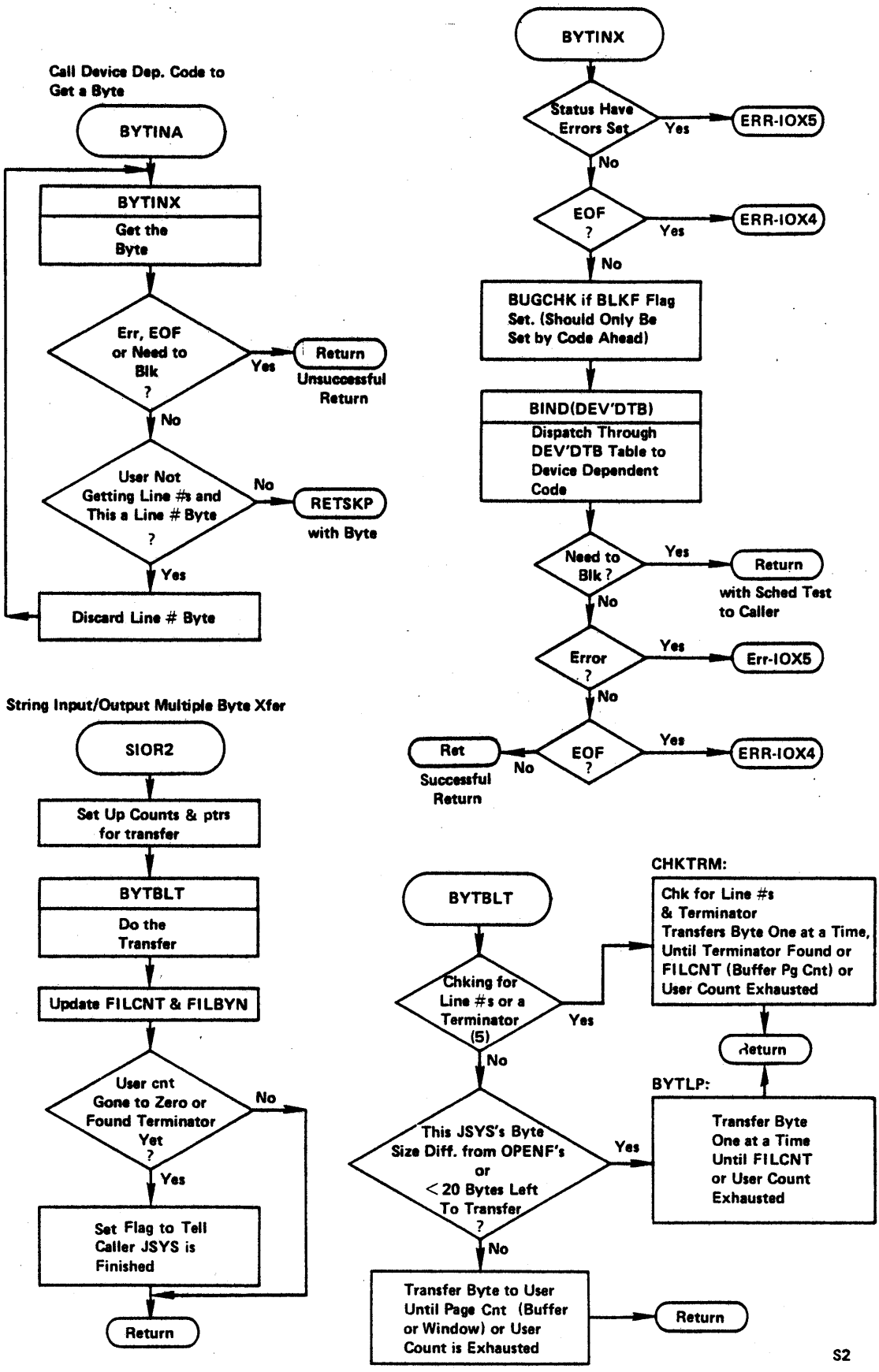

. SIN
SIN/SINR JSYS
. SINR
Set Flag to Say SIN JSYS
If This is Special Byte Ptr (-1 in LH) Create Equivalent Regular Byte Ptr. (1)
Set Flag to Say SINR JSYS
SIN0
CHKJFN
Chk JFN Validity & Set Up Status Flags
JFN OK ?
No
Error
Yes
TTY or Dev Null ?
Yes
SINTTY pS1a
No
SINBYT:
Byte Ptr (2) ?
Yes
SIOR2 pS2
Setup & Do Byte Blk Xfer (4)
Update Byte Ptrs in User's ACs
No
JFN OPENed For Read?
No
ERR-IOX1
Yes
SIN00
SIN3:
CNT Gone to Zero
No
SIN4
Yes
Tack Null Byte to String
Return to User
JFNID(DEV'DTB)
Init JFN for input
FILCNT>0 (Any Bytes Left in This Buffer Pg) (3) ?
No
SIN1 pS1a
Yes
SIOR2 pS2
Setup & Do Byte Blk Xfer (4)
(Return to User)
Yes
Satisfied User Request for Bytes ?
No
UNLCKF
Unlock File
Return to User
Yes
Rec. Size Longer Than User's Request
ERR-IOX10
Yes
Update Byte Ptrs in User ACS
No
SINR JSYS ?
Yes
Read All Bytes from Buffer ?
No
Satisfied User's Request for Bytes ?
No

S1

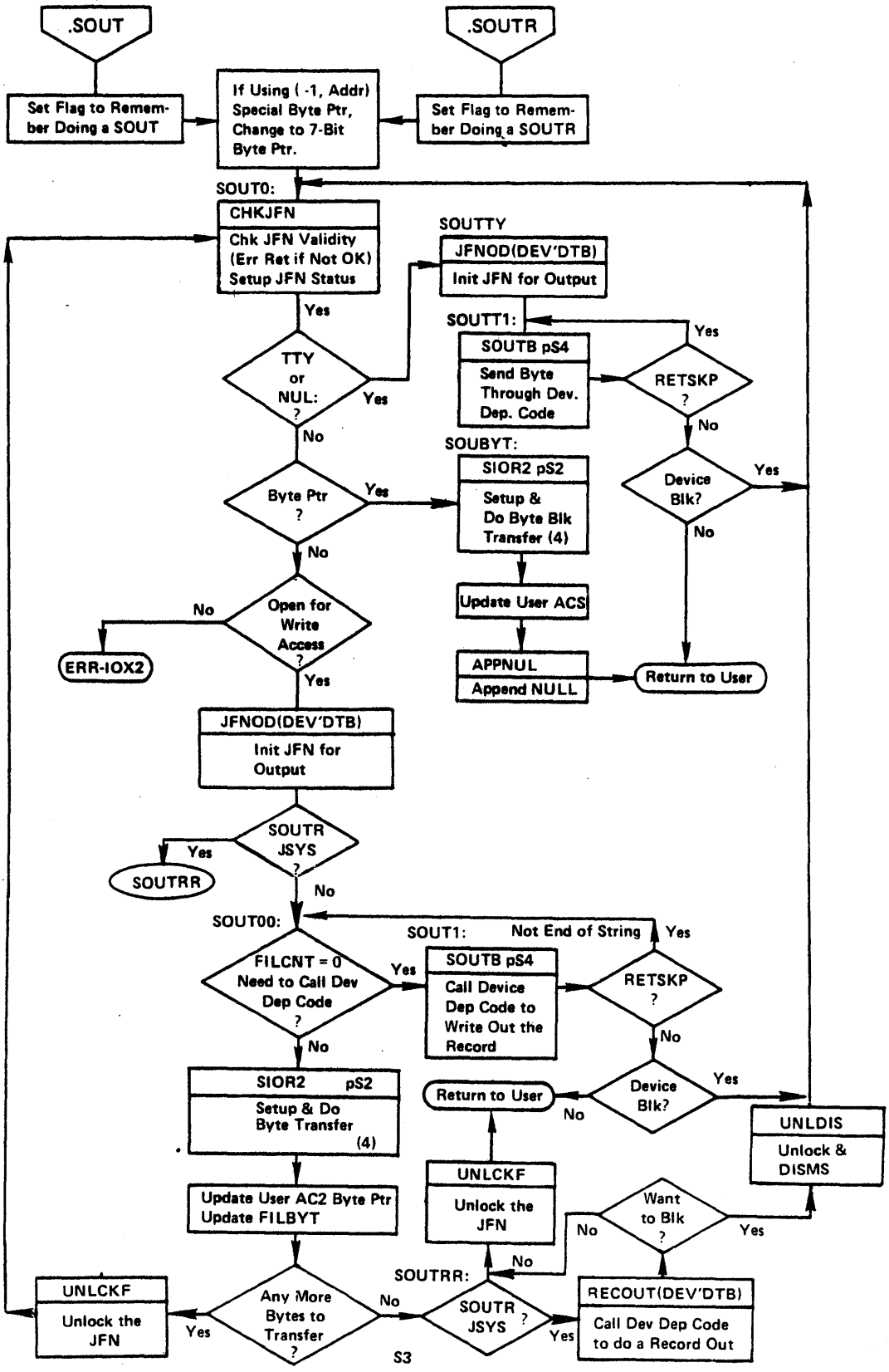
SIN/SINR JSYS (continued)

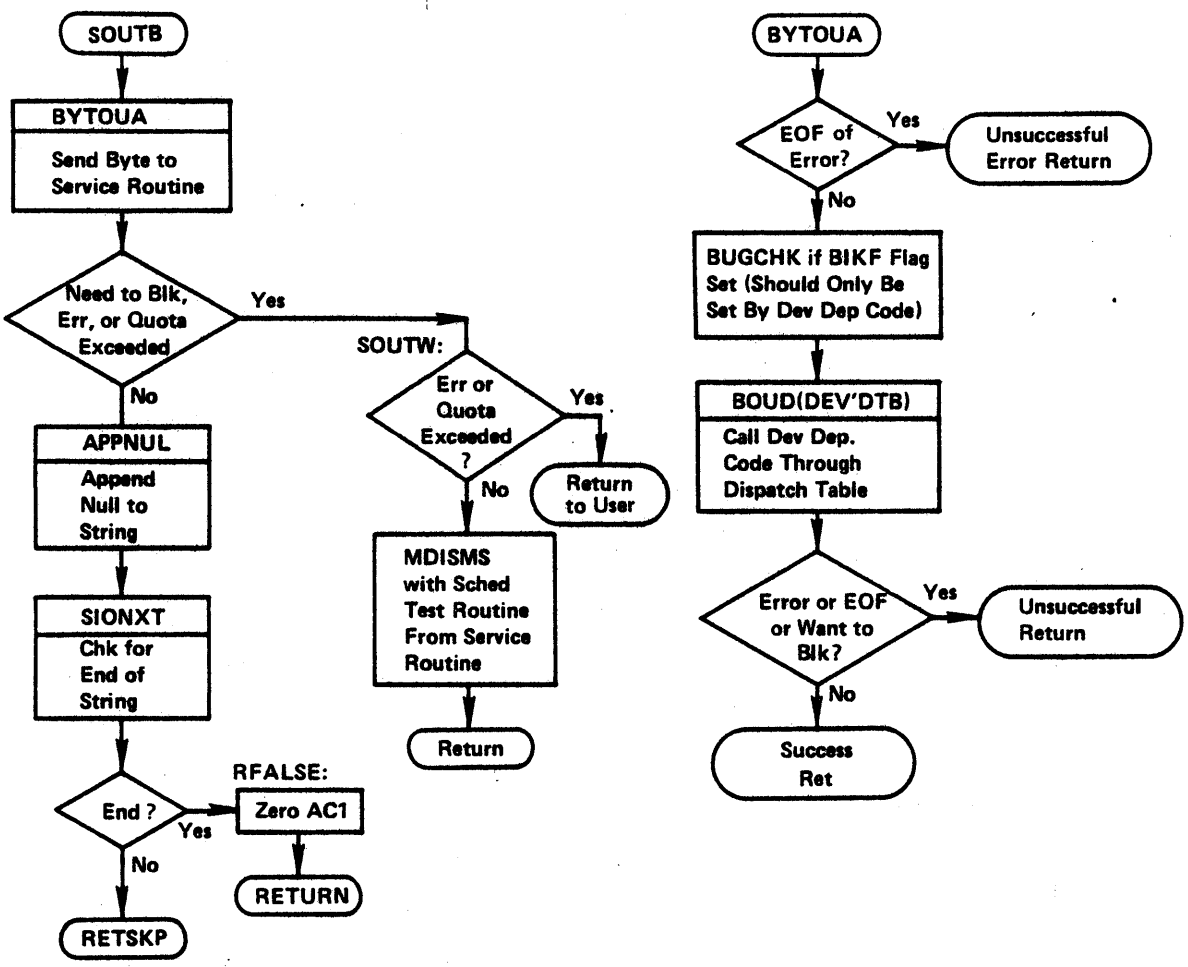


S1a

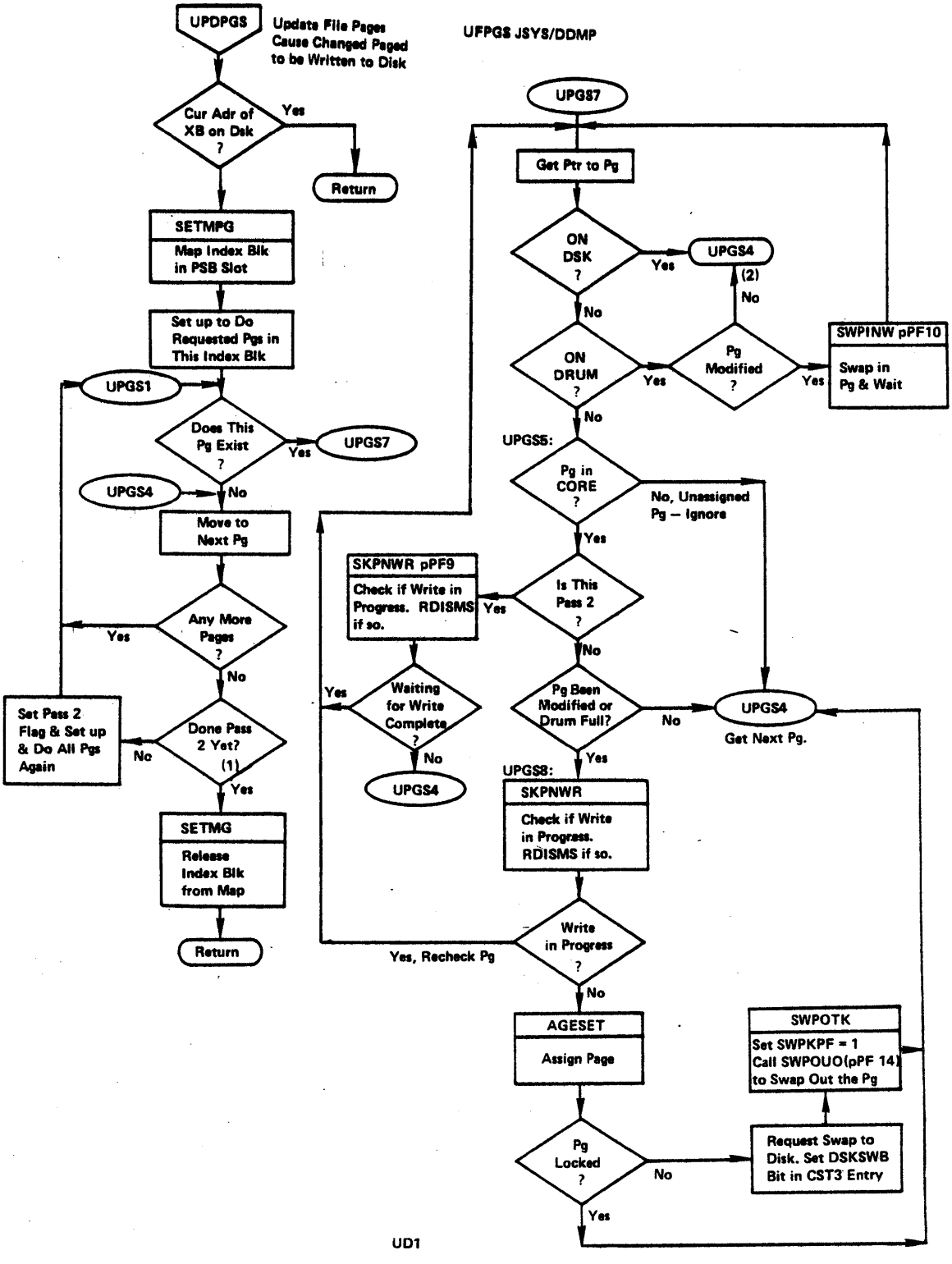

Call Device Dep. Code to Get a Byte
BYTINA
BYTINX
Get the Byte
Err, EOF or Need to Blk ?
Yes
Return Unsuccessful Return
No
User Not Getting Line #s and This a Line # Byte ?
No
RETSKP with Byte
Yes
Discard Line # Byte
BYTINX
Status Have Errors Set
Yes
ERR-IOX5
No
EOF ?
Yes
ERR-IOX4
No
BUGCHK if BLKF Flag Set. (Should Only Be Set by Code Ahead)
BIND(DEV'DTB)
Dispatch Through DEV'DTB Table to Device Dependent Code
Need to Blk ?
Yes
Return with Sched Test to Caller
No
Error ?
Yes
Err-IOX5
No
EOF ?
Yes
ERR-IOX4
No
Ret Successful Return
String Input/Output Multiple Byte Xfer
SIOR2
Set Up Counts & ptrs for transfer
BYTBLT
Do the Transfer
Update FILCNT & FILBYN
User cnt Gone to Zero or Found Terminator Yet ?
No
Yes
Set Flag to Tell Caller JSYS is Finished
Return
BYTBLT
Chking for Line #s or a Terminator (5)
Yes
CHKTRM:
Chk for Line #s & Terminator Transfers Byte One at a Time, Until Terminator Found or FILCNT (Buffer Pg Cnt) or User Count Exhausted
Return
No
BYTLP:
This JSYS's Byte Size Diff. from OPENF's or < 20 Bytes Left To Transfer ?
Yes
Transfer Byte One at a Time Until FILCNT or User Count Exhausted
No
Transfer Byte to User Until Page Cnt (Buffer or Window) or User Count is Exhausted
Return
S2

SOUT/SOUTR JSYS

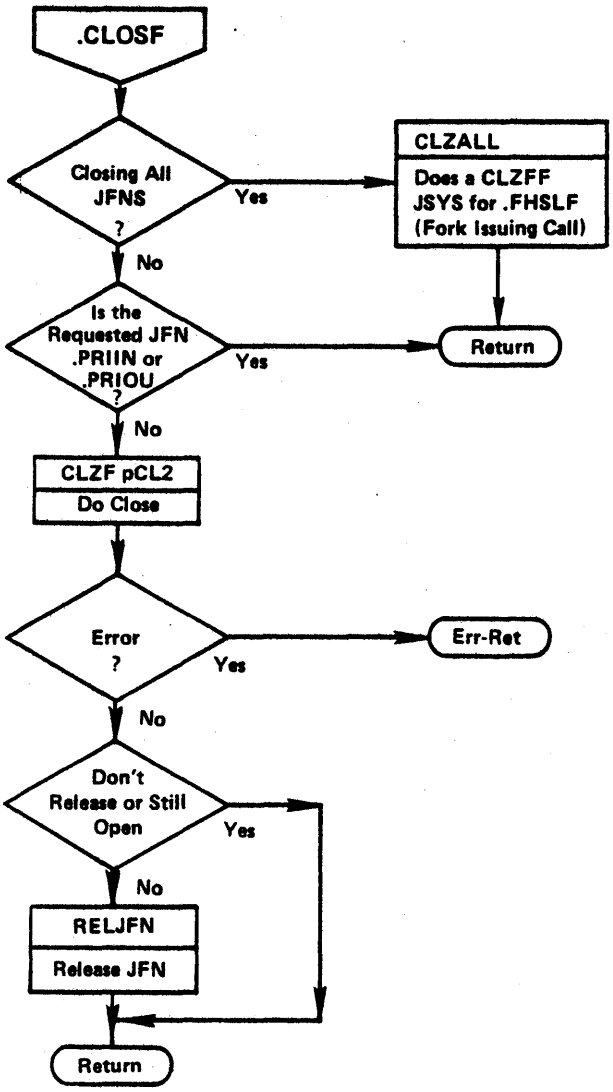



SOUTB
BYTOUA
Send Byte to Service Routine
Need to Blk, Err, or Quota Exceeded
Yes
No
SOUTW:
Err or Quota Exceeded ?
Yes
No
Return to User
MDISMS with Sched Test Routine From Service Routine
Return
APPNUL
Append Null to String
SIONXT
Chk for End of String
End ?
Yes
No
RFALSE:
Zero AC1
RETURN
RETSKP
BYTOUA
EOF of Error?
Yes
Unsuccessful Error Return
No
BUGCHK if BIKF Flag Set (Should Only Be Set By Dev Dep Code)
BOUD(DEV'DTB)
Call Dev Dep. Code Through Dispatch Table
Error or EOF or Want to Blk?
Yes
Unsuccessful Return
No
Success Ret

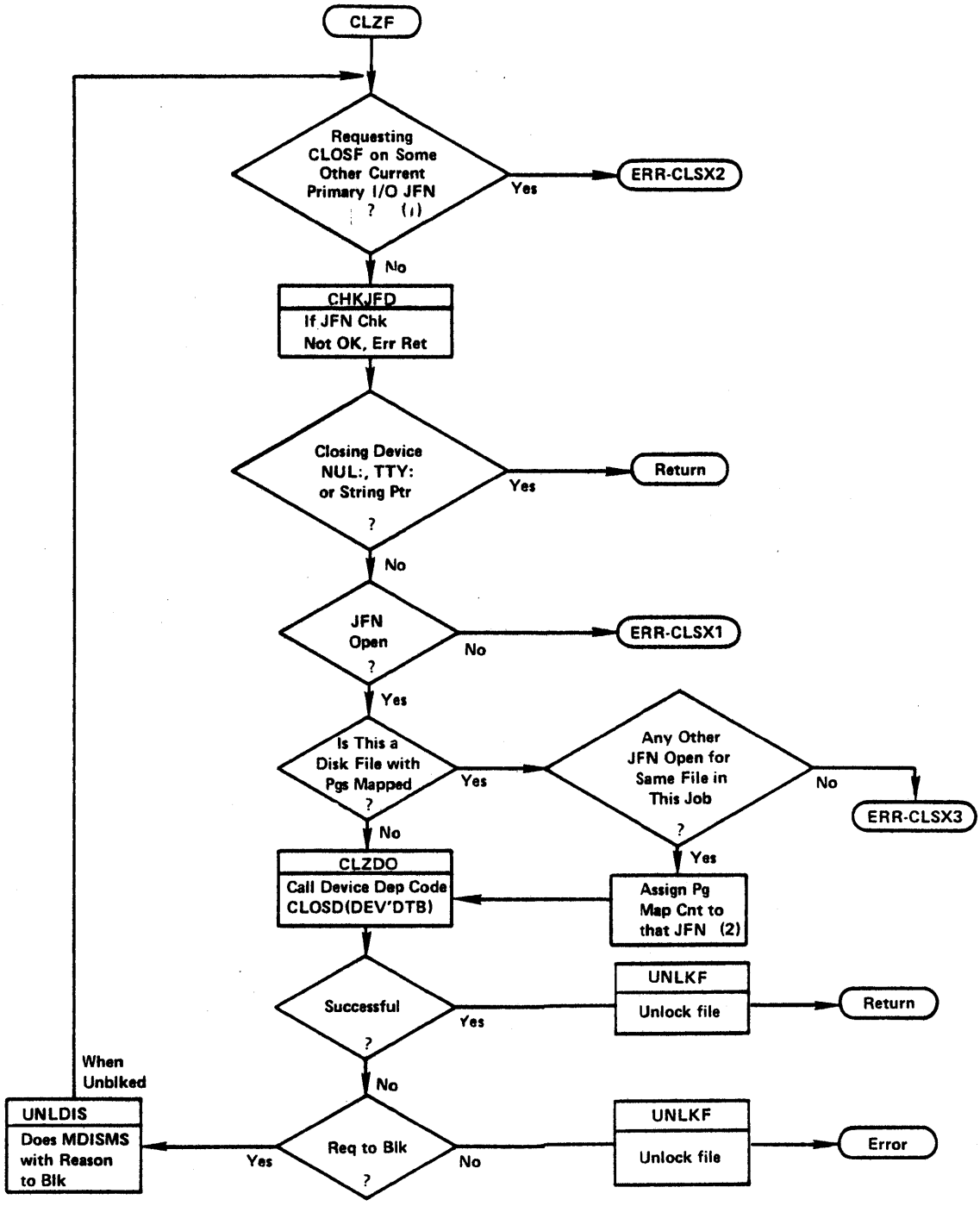
S4


UPDPGS
Updats File Pages Cause Changed Paged to be Written to Disk
Cur Adr of XB on Dsk ?
Yes
Return
SETMPG
Map Index Blk in PSB Slot
Set up to Do Requested Pgs in This Index Blk
UPGS1
Does This Pg Exist ?
Yes
UPGS7
UPGS4
No
Move to Next Pg
Any More Pages ?
Yes
No
Set Pass 2 Flag & Set up & Do All Pgs Again
No
Done Pass 2 Yet? (1)
Yes
SETMG
Release Index Blk from Map
Return
UFPGS JSYS/DDMP
UPGS7
Get Ptr to Pg
ON DSK ?
Yes
UPGS4
(2)
No
No
ON DRUM ?
Yes
Pg Modified ?
Yes
SWPINW pPF10
Swap in Pg & Wait
No
UPGS5:
Pg in CORE ?
No, Unassigned Pg — Ignore
Yes
Is This Pass 2 ?
Yes
SKPNWR pPF9
Check if Write in Progress. RDISMS if so.
No
Pg Been Modified or Drum Full?
No
UPGS4
Get Next Pg.
Yes
Yes
Waiting for Write Complete ?
No
UPGS4
UPGS8:
SKPNWR
Check if Write in Progress. RDISMS if so.
Write in Progress ?
Yes, Recheck Pg
No
AGESET
Assign Page
SWPOTK
Set SWPKPF = 1
Call SWPOUO(pPF 14) to Swap Out the Pg
Pg Locked ?
No
Request Swap to Disk. Set DSKSWB Bit in CST3 Entry
Yes
UD1

CLOSF JSYS



CL1


CLZF
Requesting CLOSF on Some Other Current Primary I/O JFN ? (1)
Yes
ERR-CLSX2
No
CHKJFD
If JFN Chk Not OK, Err Ret
Closing Device NUL:, TTY: or String Ptr ?
Yes
Return
No
JFN Open ?
No
ERR-CLSX1
Yes
Is This a Disk File with Pgs Mapped ?
Yes
Any Other JFN Open for Same File in This Job ?
No
ERR-CLSX3
Yes
No
CLZDO
Call Device Dep Code CLOSD(DEV'DTB)
Assign Pg Map Cnt to that JFN (2)
Successful ?
Yes
UNLKF
Unlock file
Return
No
When Unblked
UNLDIS
Does MDISMS with Reason to Blk
Yes
Req to Blk ?
No
UNLKF
Unlock file
Error

CL2

GTJFN Comments

(1) This code is looking for a file specification of the form:

Dev:Directory Name,type,gen;T(temporary);P(protection);A(account)

One or more fields can be defined by logical names. If any fields are omitted from the specification, the system will default the values as follows:

Device	DSK:
Directory	Connected directory
Name	No default for disk Null for other devices
Generation	Highest existing for input Next highest for output
Protection	As specified for directory or protection of next lower generation
Account	Current user account

(2) The internal GTJFN code uses several locations in the JFN block as temporary cells. These locations have two names in the JFN block table descriptions. The JFN block storage locations set up or used by GTJFN are:

FILLCK*	FILDDN	FILNEN
FILTMP*	FILPRT	FILVER
FILACT*	FILSTS1	FILCOD (LH)
FILOPT*	FILLNM*	
FILDEV	FILDNM	

*Used internally only by the GTJFN JSYS.

(3) The creation process of the FDB simply asks for space in the directory for the FDB.

.OPENF Comments

(1) Cell FILDEV in the JFN blk has the device dispatch table address. For example, for disk, GTJFN sets the dispatch table address to DSKDTB. If spooling to disk, GTJFN sets the dispatch table address to SPLDTB, but the OPENF code changes the dispatch table address to DSKDTB and sets up a file specification in the JFN block.

(1) TOPS-20 allows a user to specify a special byte pointer of -1,, Address which is interpreted as a 7-bit byte size beginning on the word boundary, Address.

(2) A user can do I/O from one place to another in core by specifying byte pointers for both source and destination. This differs from BLT in that the use can transfer on non-word boundaries.

(3) For disk files, FILCNT will be the number of bytes remaining in the window page. For magtape and other devices it will be the number of bytes remaining in the current page of the buffer.

(4) The routine BYTBLT only moves data up to the page boundary of the current buffer page.

(5) If the user has not specified OF%PLN in the OPENF, line numbers are stripped off the beginning of each line. (See SIN JSYS in Monitor Calls manual for definition of terminator.)

.PMAP Comments

(1) A page is private if it is not shared between a file and a fork.

UPDPGS Comments

(1) Routine scans page table twice: first time to request writes on all changed pages. Second time to wait for completion of writes. (This is faster than waiting for each write to complete as it is requested.)

(2) If page has not been modified, a check is made to see if the drum is full and if so, to release this page back to the drum. The map pointer to the page will be changed to its disk address.

CLOSF Comments

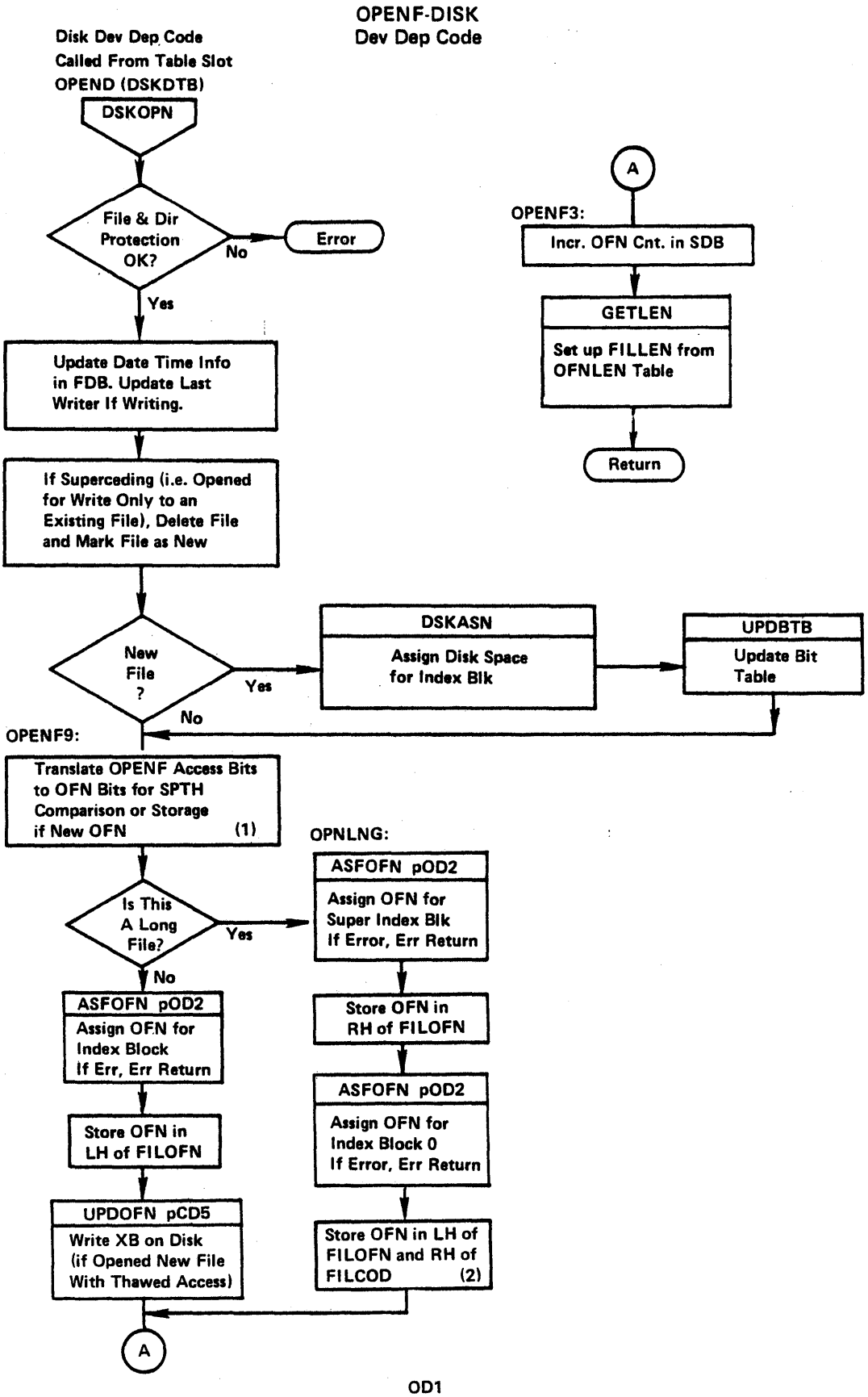
(1) If user has switched primary I/O to some other JFN and attempts to close it, an error results.

(2) The page map count in FILFW reflects the number of pages mapped and a CLOSF can't be done on a file if this count is greater than Ø .

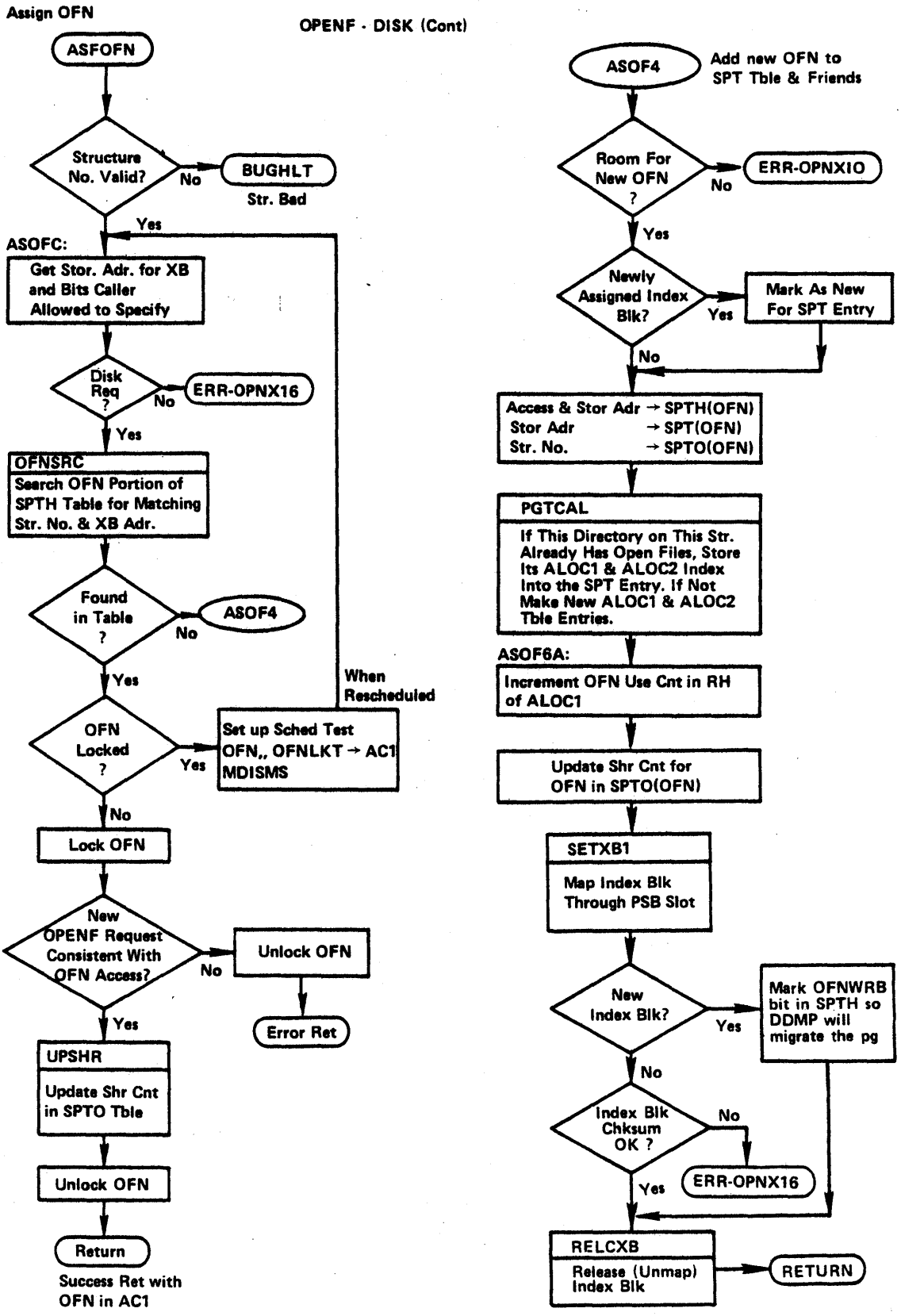
JSYS CALL FLOWCHARTS

DSK DEPENDENT LEVEL

DSKOFN -	Disk Opening of a File	OD1
	ASFOFN - Assign OFN	OD2
	UPDOFN - Update OFN	OD1
DSKSQI/O -	Disk Sequential Input/Output	SD1
	NEWWND - New Window Page (Next Page of File)	SD2
DSKCLZ -	Disk Closing of a File	CD1
	RELOFN - Release OFN	CD2
	DASOFN - Deassign OFN	CD3
	MOVDSK - Move Page Back to Disk	CD3

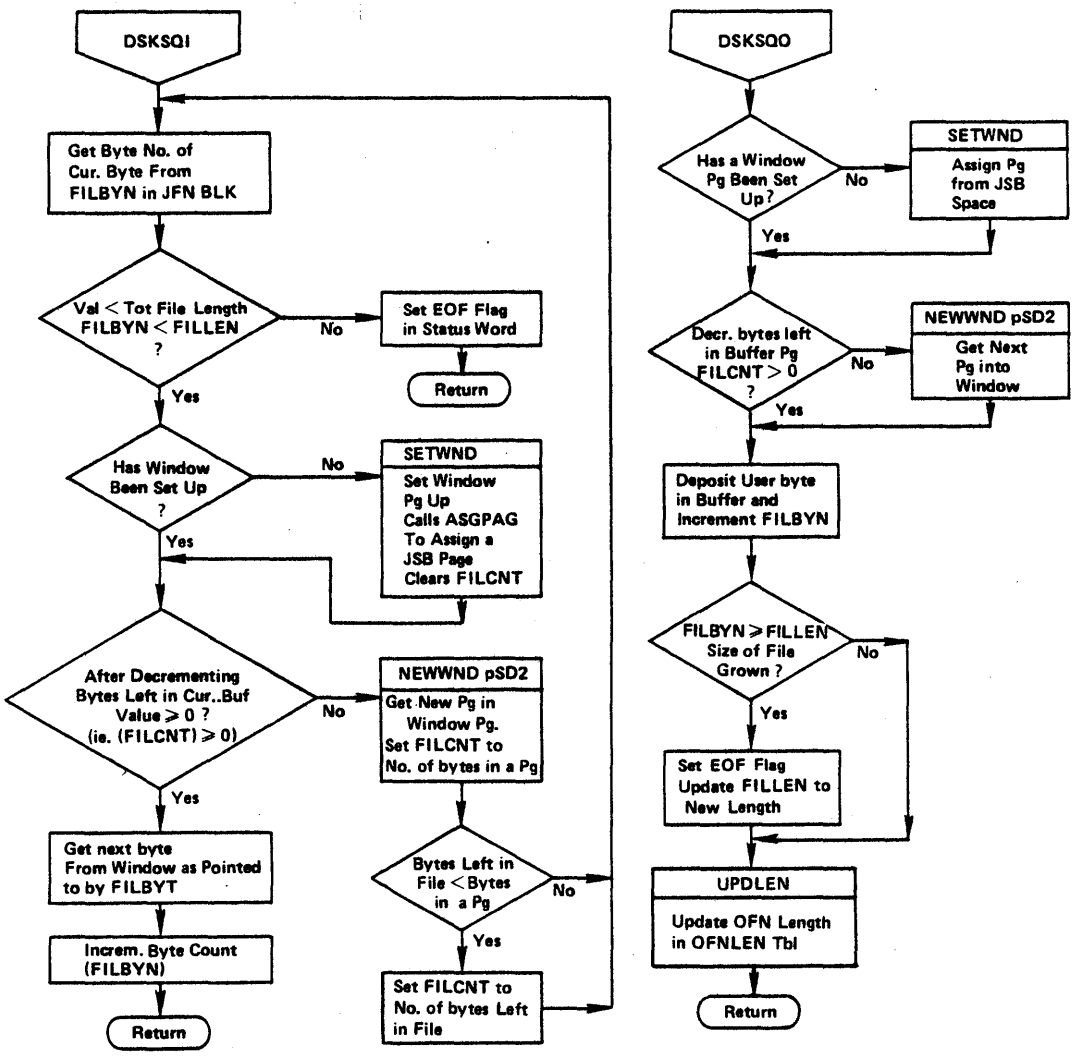

OPENF-DISK
Dev Dep Code
Disk Dev Dep Code
Called From Table Slot
OPEND (DSKDTB)
DSKOPN
File & Dir Protection OK?
No
Error
Yes
Update Date Time Info in FDB. Update Last Writer If Writing.
If Superceding (i.e. Opened for Write Only to an Existing File), Delete File and Mark File as New
New File ?
Yes
No
DSKASN
Assign Disk Space for Index Blk
UPDBTB
Update Bit Table
OPENF9:
Translate OPENF Access Bits to OFN Bits for SPTH Comparison or Storage if New OFN (1)
Is This A Long File?
Yes
No
OPNLNG:
ASFOFN pOD2
Assign OFN for Super Index Blk If Error, Err Return
Store OFN in RH of FILOFN
ASFOFN pOD2
Assign OFN for Index Block 0 If Error, Err Return
Store OFN in LH of FILOFN and RH of FILCOD (2)
ASFOFN pOD2
Assign OFN for Index Block If Err, Err Return
Store OFN in LH of FILOFN
UPDOFN pCD5
Write XB on Disk (if Opened New File With Thawed Access)
A
A
OPENF3:
Incr. OFN Cnt. in SDB
GETLEN
Set up FILLEN from OFNLEN Table
Return

OD1

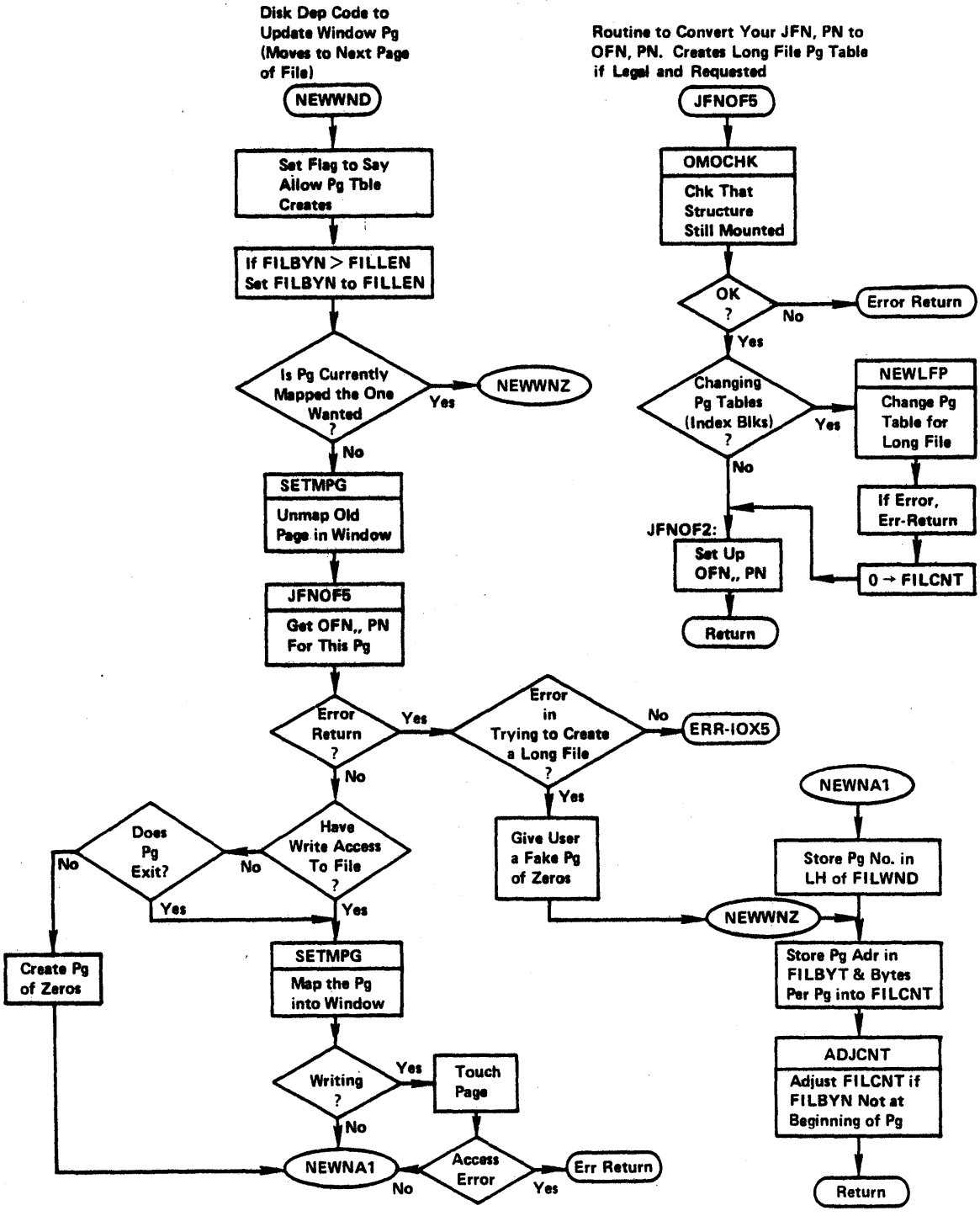

Assign OFN
OPENF - DISK (Cont)
ASFOFN
Structure No. Valid?
No
BUGHLT
Str. Bad
Yes
ASOFC:
Get Stor. Adr. for XB and Bits Caller Allowed to Specify
Disk Req ?
No
ERR-OPNX16
Yes
OFNSRC
Search OFN Portion of SPTH Table for Matching Str. No. & XB Adr.
Found in Table ?
No
ASOF4
Yes
When Rescheduled
OFN Locked ?
Yes
Set up Sched Test OFN,, OFNLKT → AC1 MDISMS
No
Lock OFN
New OPENF Request Consistent With OFN Access?
No
Unlock OFN
Error Ret
Yes
UPSHR
Update Shr Cnt in SPTO Tble
Unlock OFN
Return
Success Ret with OFN in AC1
ASOF4
Add new OFN to SPT Tble & Friends
Room For New OFN ?
No
ERR-OPNXIO
Yes
Newly Assigned Index Blk?
Yes
Mark As New For SPT Entry
No
Access & Stor Adr → SPTH(OFN)
Stor Adr → SPT(OFN)
Str. No. → SPTO(OFN)
PGTCAL
If This Directory on This Str. Already Has Open Files, Store Its ALOC1 & ALOC2 Index Into the SPT Entry. If Not Make New ALOC1 & ALOC2 Tble Entries.
ASOF6A:
Increment OFN Use Cnt in RH of ALOC1
Update Shr Cnt for OFN in SPTO(OFN)
SETXB1
Map Index Blk Through PSB Slot
New Index Blk?
Yes
Mark OFNWRB bit in SPTH so DDMP will migrate the pg
No
Index Blk Chksum OK ?
No
ERR-OPNX16
Yes
RELCXB
Release (Unmap) Index Blk
RETURN

OD2

SEQUENTIAL I/O-DSK
(String & Byte Dev Dep Code)

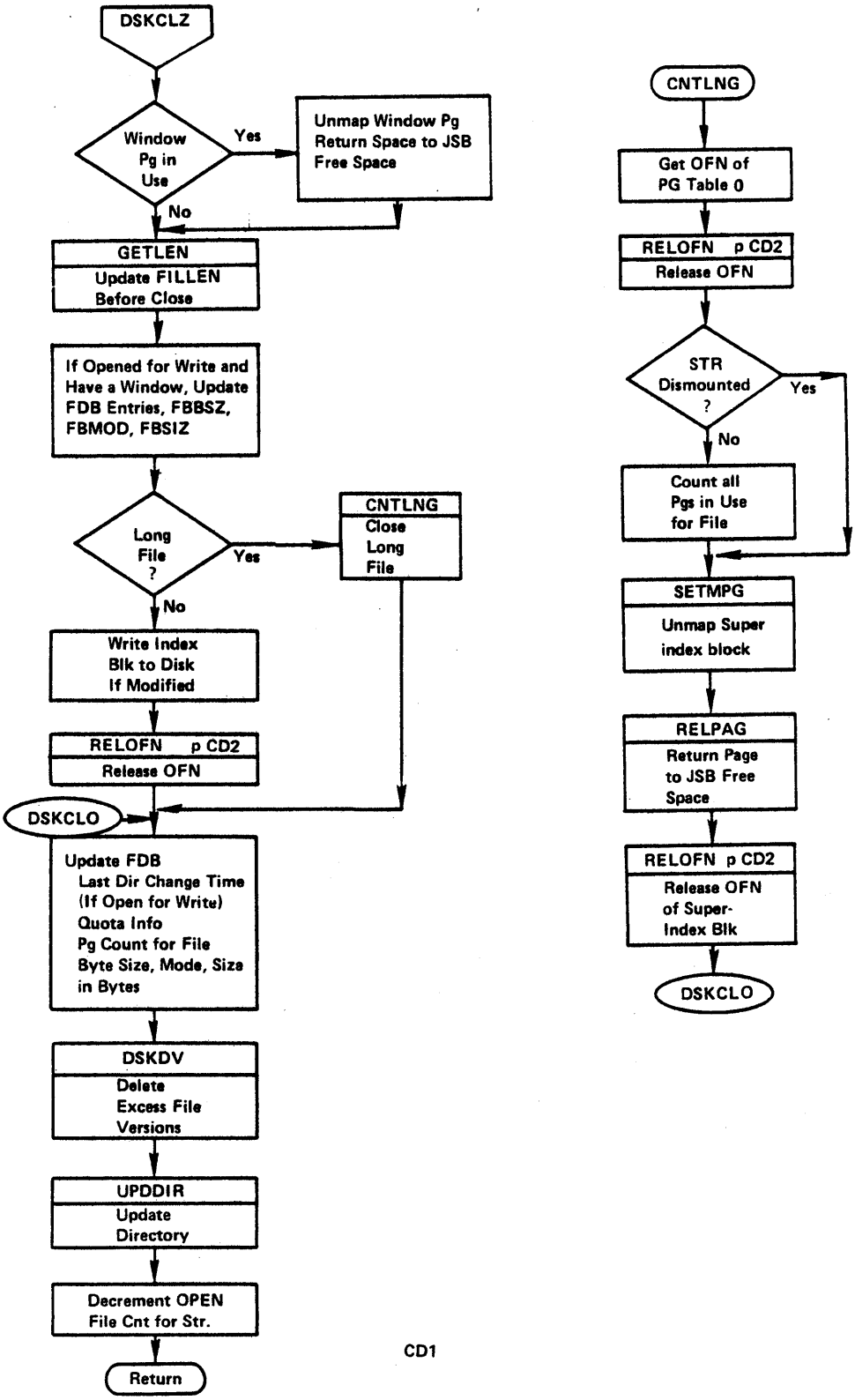


SD1


Disk Dep Code to Update Window Pg (Moves to Next Page of File)
NEWWND
Set Flag to Say Allow Pg Tble Creates
If FILBYN > FILLEN Set FILBYN to FILLEN
Is Pg Currently Mapped the One Wanted ?
Yes
NEWWNZ
No
SETMPG
Unmap Old Page in Window
JFNOF5
Get OFN,, PN For This Pg
Error Return ?
Yes
Error in Trying to Create a Long File ?
No
ERR-IOX5
Yes
Give User a Fake Pg of Zeros
No
Have Write Access To File ?
No
Does Pg Exit?
No
Yes
Yes
Create Pg of Zeros
SETMPG
Map the Pg into Window
Writing ?
Yes
Touch Page
No
NEWNA1
Access Error
Yes
Err Return
No
Routine to Convert Your JFN, PN to OFN, PN. Creates Long File Pg Table if Legal and Requested
JFNOF5
OMOCHK
Chk That Structure Still Mounted
OK ?
No
Error Return
Yes
Changing Pg Tables (Index Blks) ?
Yes
NEWLFP
Change Pg Table for Long File
If Error, Err-Return
No
JFNOF2:
Set Up OFN,, PN
0 → FILCNT
Return
NEWNA1
Store Pg No. in LH of FILWND
NEWWNZ
Store Pg Adr in FILBYT & Bytes Per Pg into FILCNT
ADJCNT
Adjust FILCNT if FILBYN Not at Beginning of Pg
Return

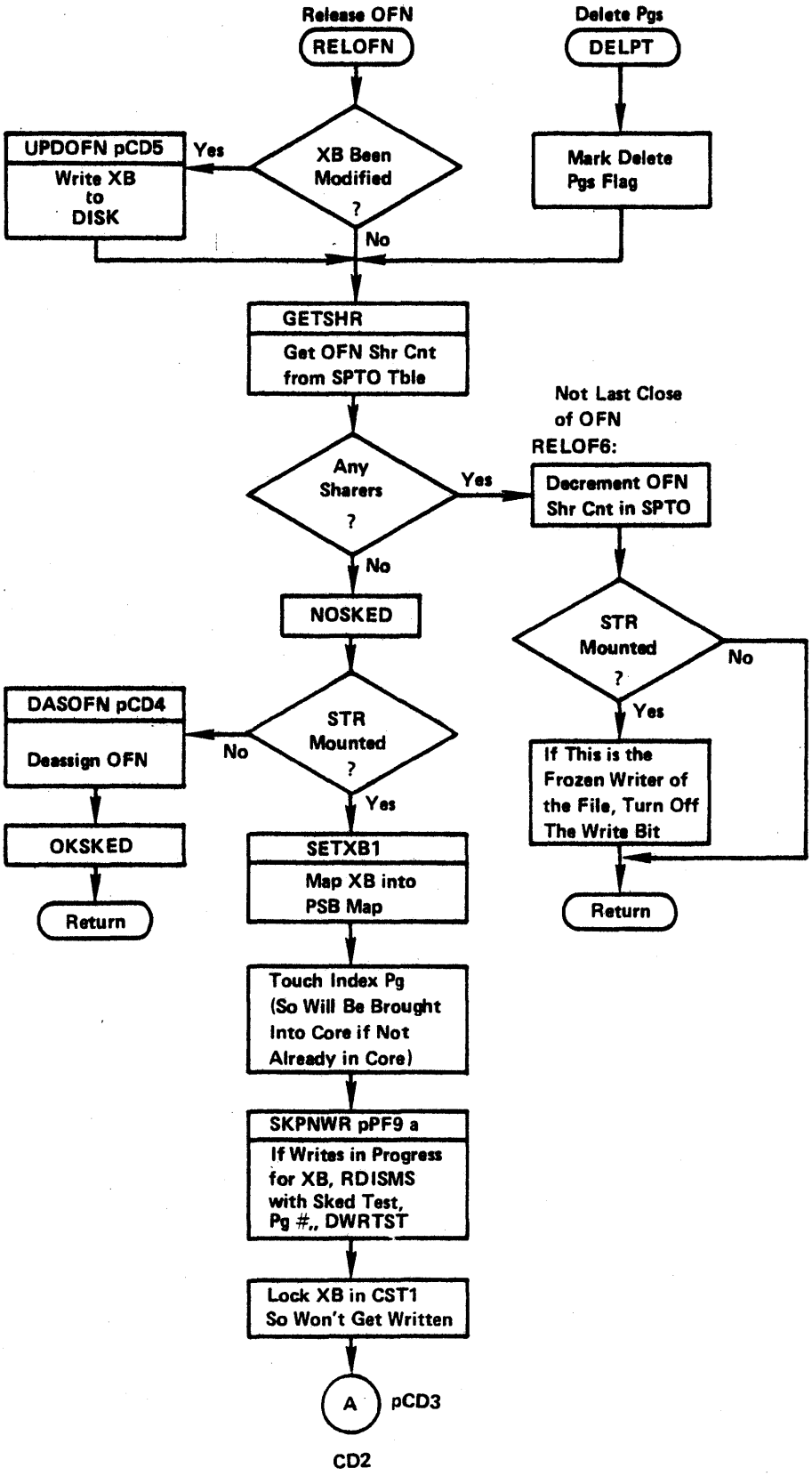
SD2

CLOSF – DSK
Dev. Dep. Code

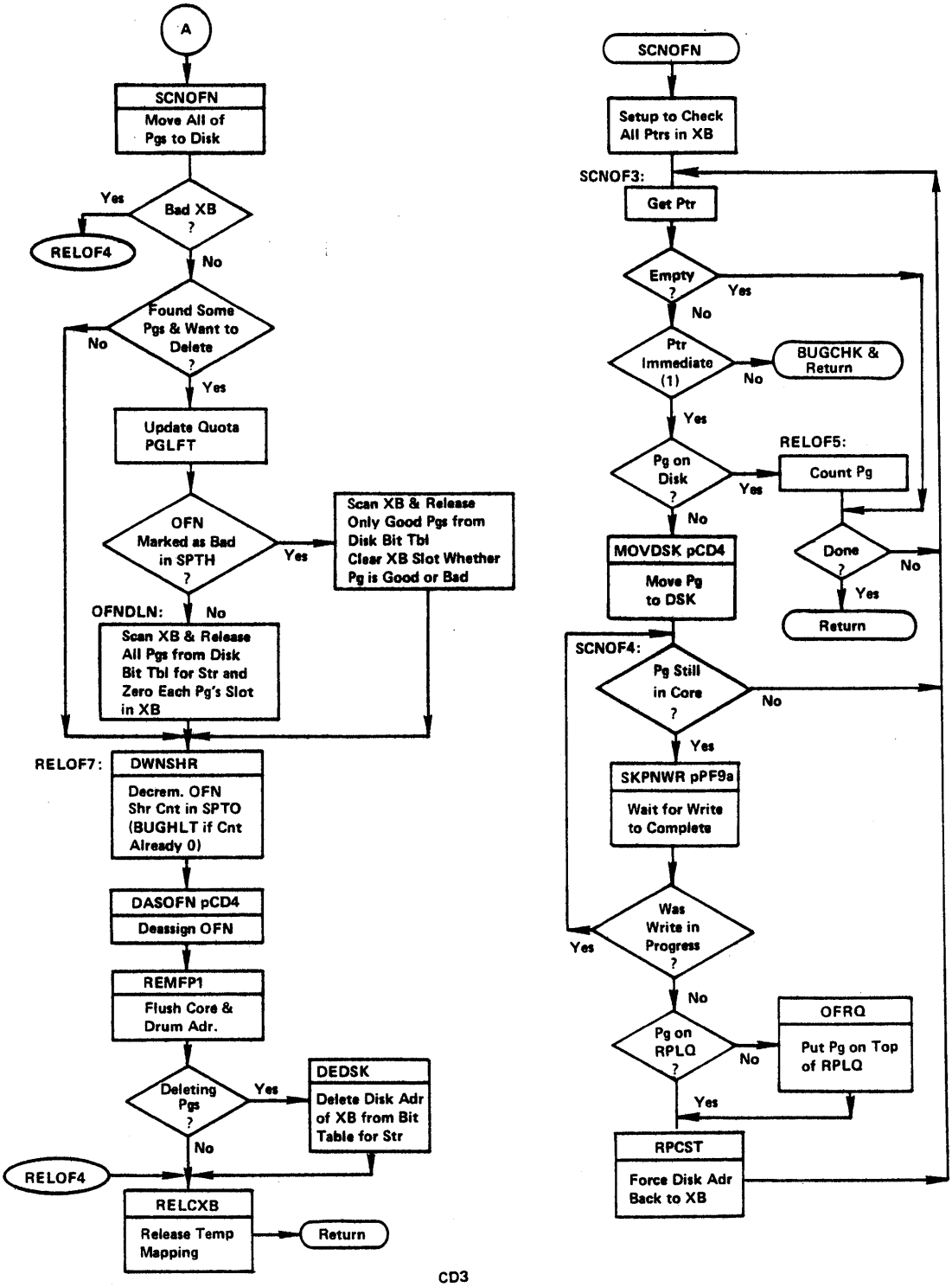


CD1

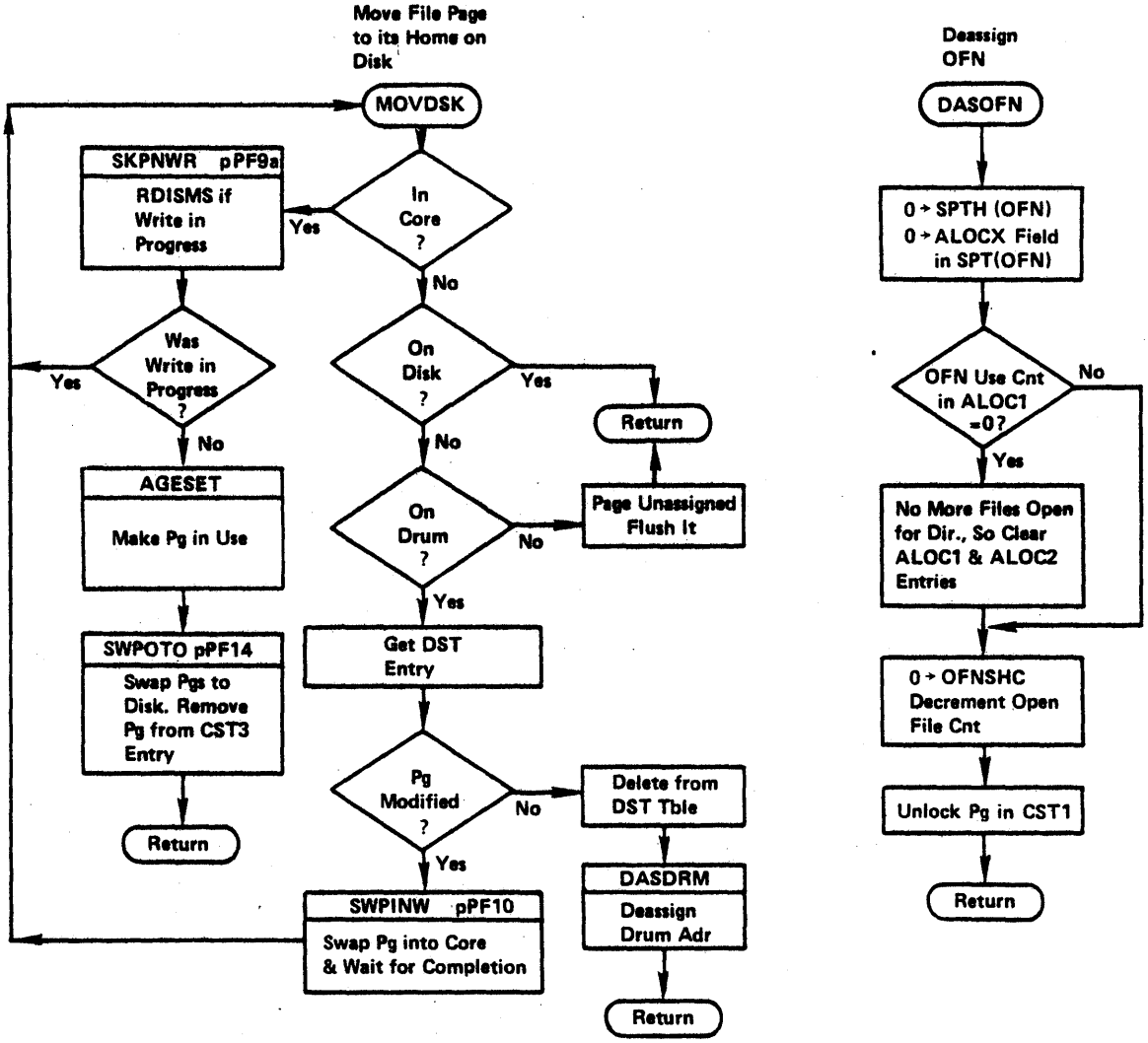
CLOSF-DSK (Continued)

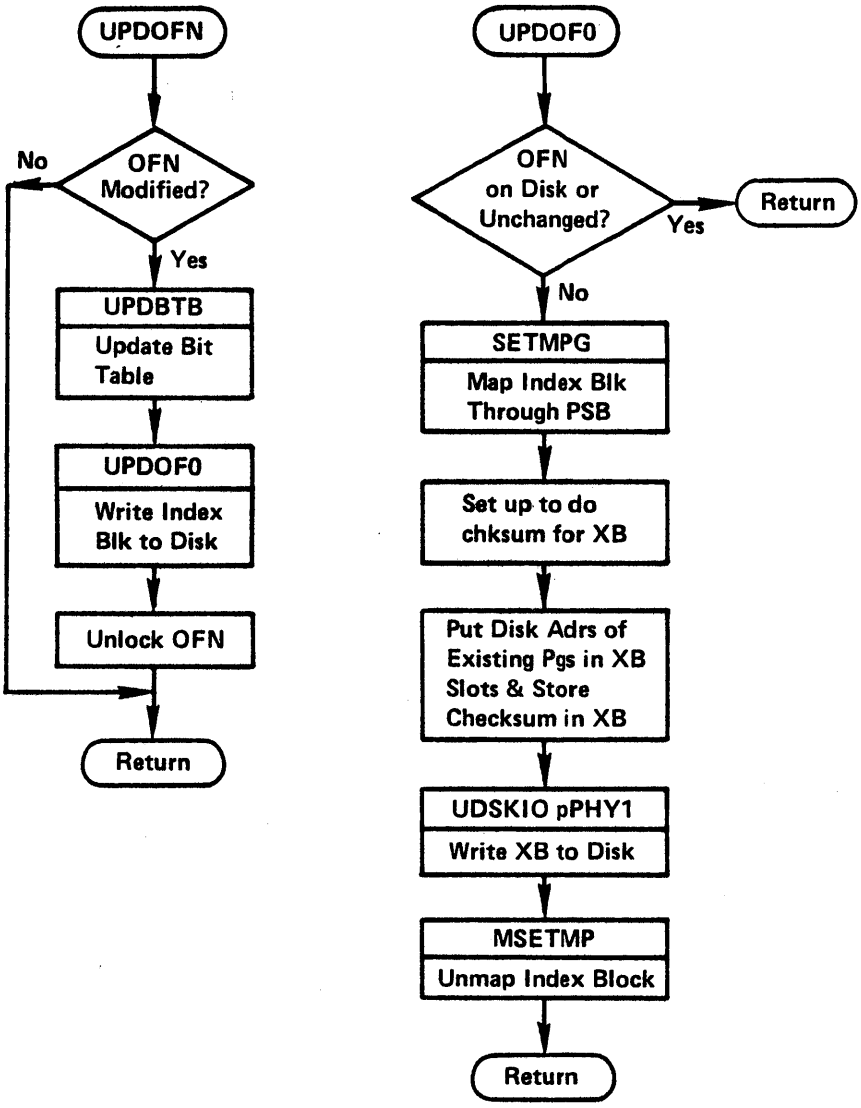


CLOSF-DSK (Continued)



CD3


Move File Page to its Home on Disk
MOVDSK
In Core ?
Yes
SKPNWR pPF9a
RDISMS if Write in Progress
No
On Disk ?
Yes
Return
No
On Drum ?
No
Page Unassigned Flush It
Yes
Get DST Entry
Pg Modified ?
No
Delete from DST Tble
Yes
DASDRM
Deassign Drum Adr
Return
SWPINW pPF10
Swap Pg into Core & Wait for Completion
Was Write in Progress ?
Yes
No
AGESET
Make Pg in Use
SWPOTO pPF14
Swap Pgs to Disk. Remove Pg from CST3 Entry
Return
Deassign OFN
DASOFN
0 → SPTH (OFN)
0 → ALOCX Field in SPT(OFN)
OFN Use Cnt in ALOC1 =0?
No
Yes
No More Files Open for Dir., So Clear ALOC1 & ALOC2 Entries
0 → OFNSHC Decrement Open File Cnt
Unlock Pg in CST1
Return


UPDOFN
No
OFN Modified?
Yes
UPDBTB
Update Bit Table
UPDOF0
Write Index Blk to Disk
Unlock OFN
Return
UPDOF0
OFN on Disk or Unchanged?
Yes
Return
No
SETMPG
Map Index Blk Through PSB
Set up to do chksum for XB
Put Disk Adrs of Existing Pgs in XB Slots & Store Checksum in XB
UDSKIO pPHY1
Write XB to Disk
MSETMP
Unmap Index Block
Return

OPENF-DISK Comments

(1) OFN bits: 0=read, 10=write, 11=thawed, 01=restricted

(2) For a long file, the OFN of index block Ø is remembered in the JFN blk and used as the identity of the file by the ENQ/DEQ facility.

CLOSF-DISK Comments

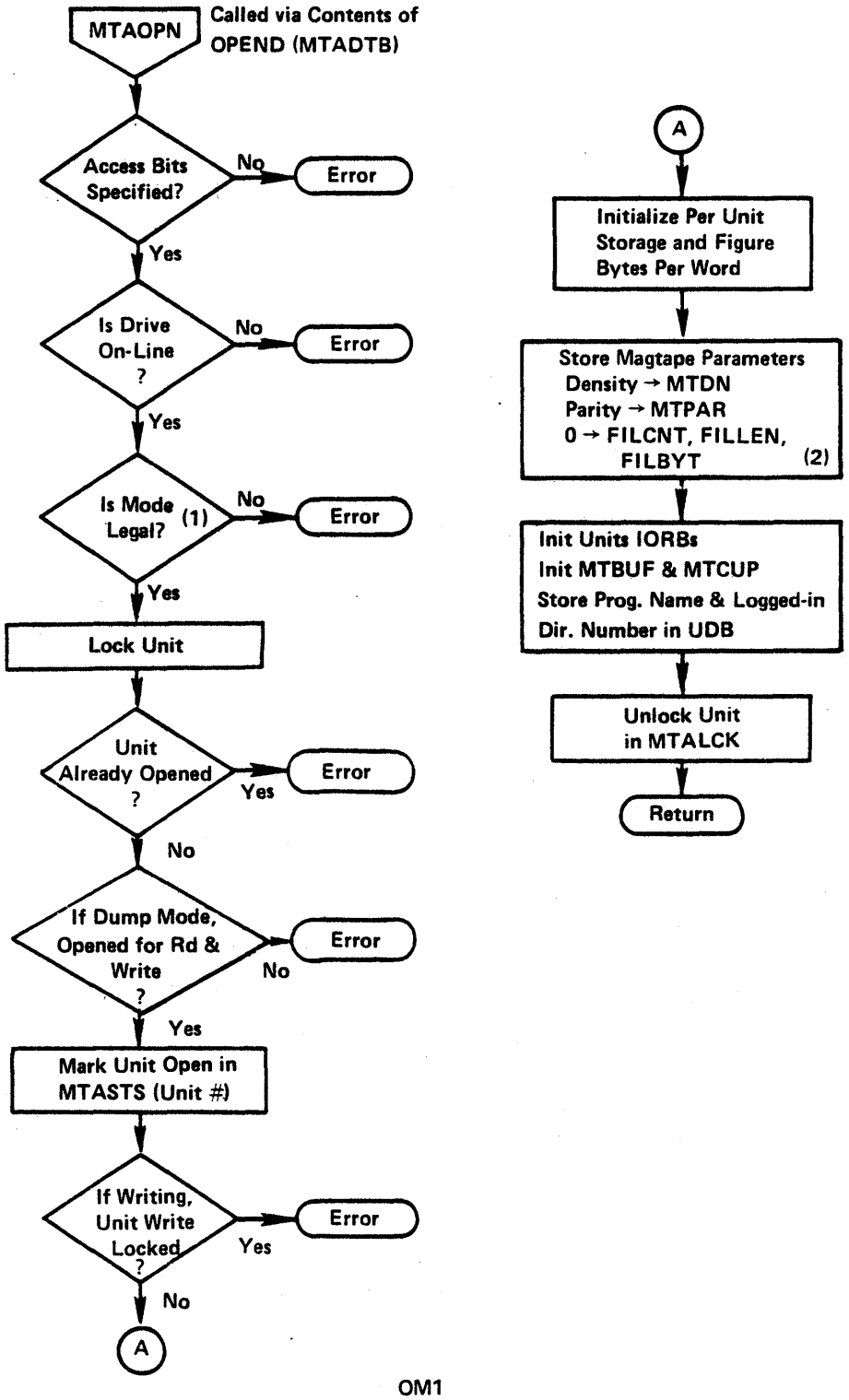
(1) **All storage addresses placed in an index blk have the pointer type field set to immediate.**

JSYS's CALLS

MTA DEPENDENT LEVEL

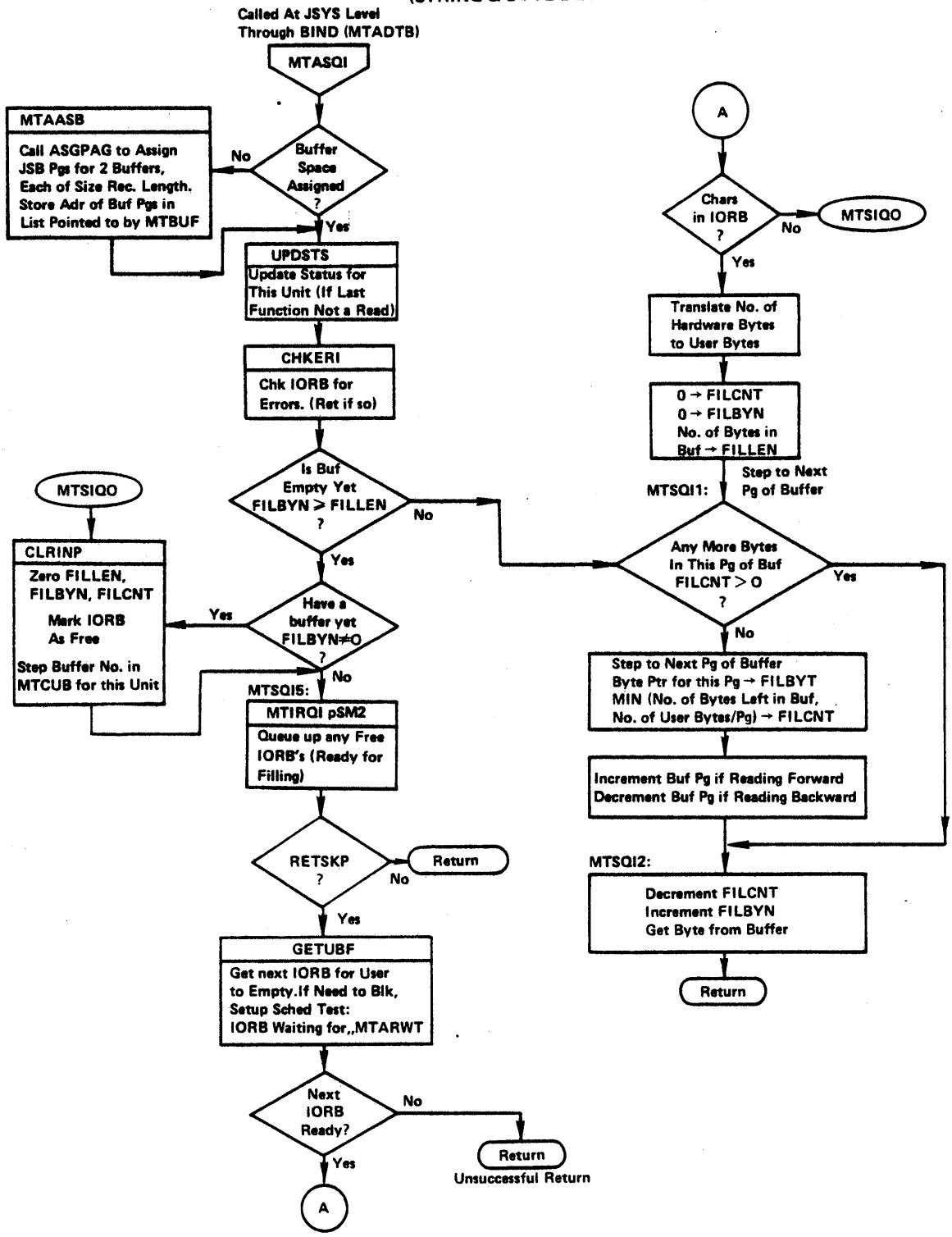
MTAOPN -	Magtape Opening of a File	OM1
MTASQI -	Magtape Sequential Input	SM1
	MTAIRQ - Queue Up Specified IORB	SM2
MTASQO -	Magtape Sequential Output	SM3
MTACLZ -	Magtape Closing of a File	CM1

OPENF – MAGTAPE DEVICE DEPENDENT CODE

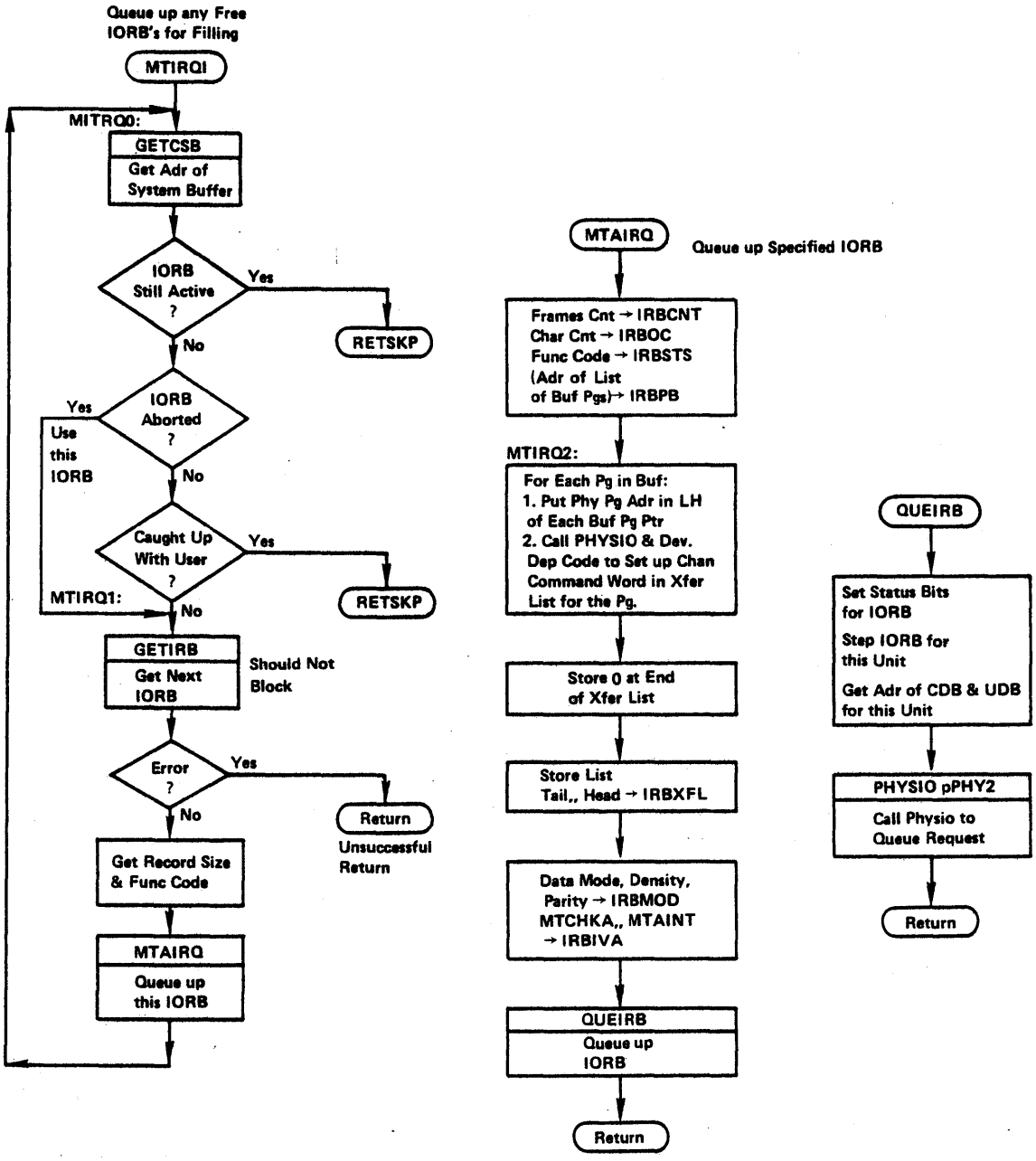


OM1

SEQUENTIAL INPUT – MTA
(STRING & BYTE DEV. DEP. CODE)

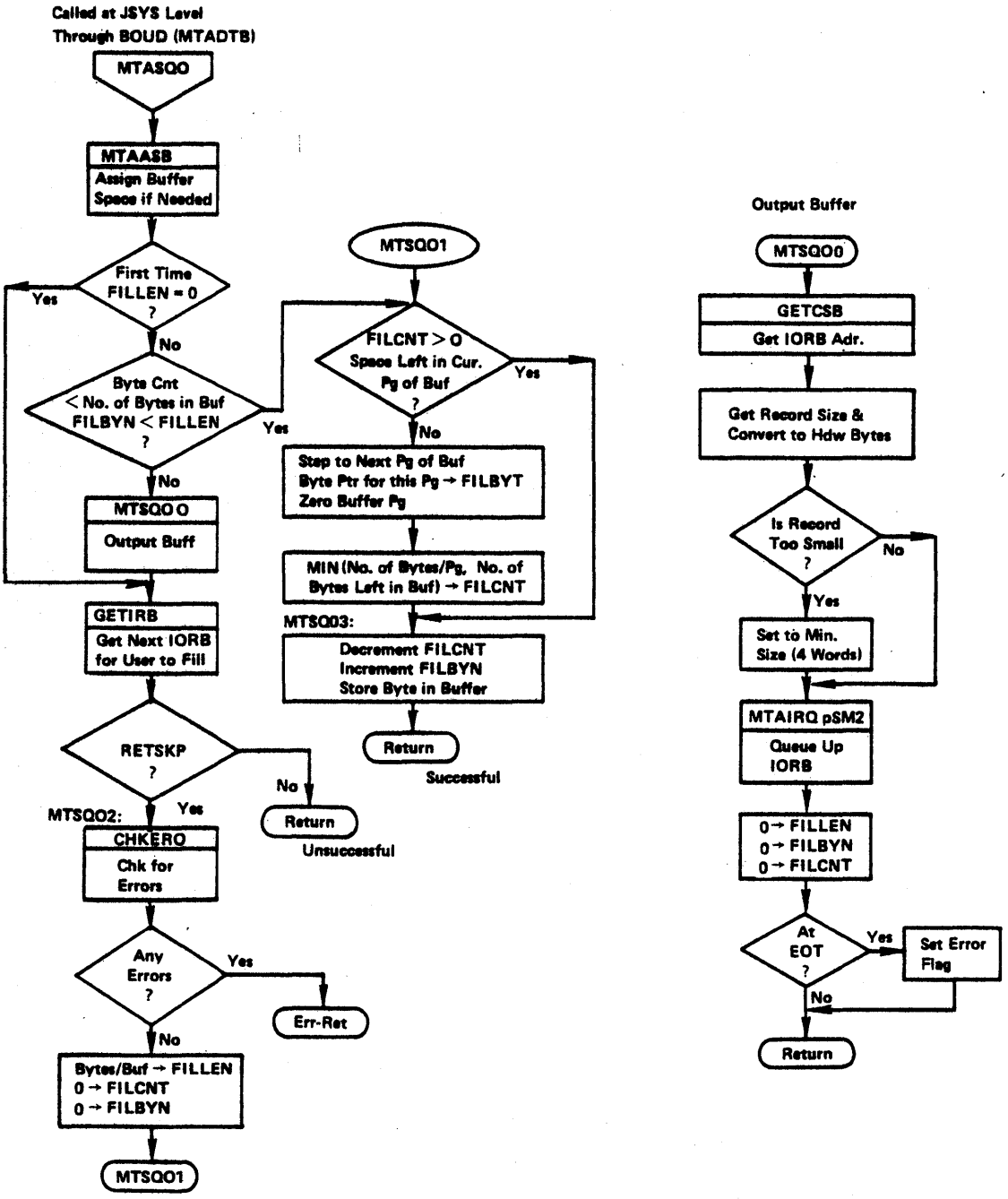


SM1

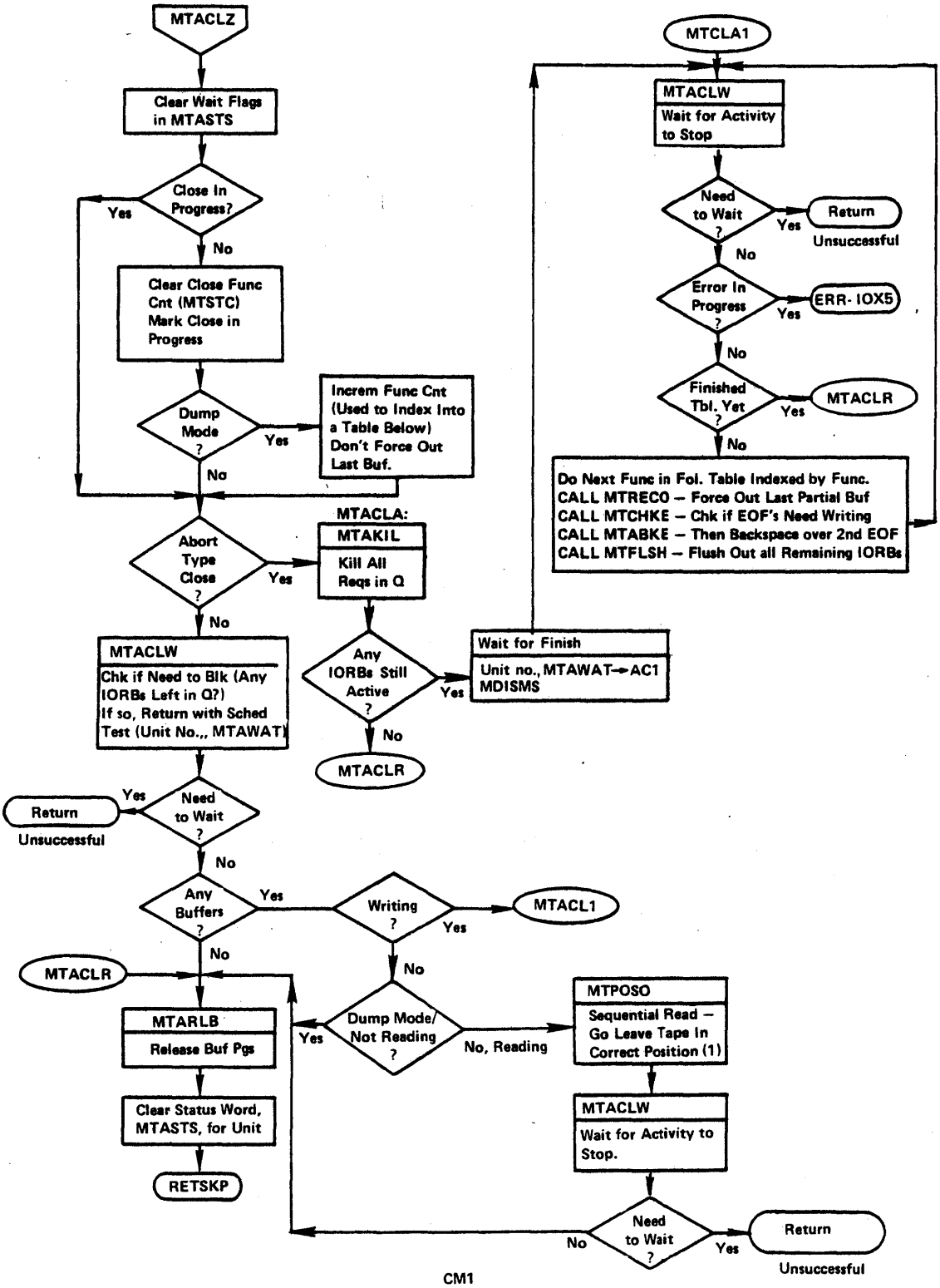

Queue up any Free IORB's for Filling
MTIRQI
MITRQ0:
GETCSB
Get Adr of System Buffer
IORB Still Active ?
Yes
RETSKP
No
IORB Aborted ?
Yes
Use this IORB
No
Caught Up With User ?
Yes
RETSKP
MTIRQ1:
No
GETIRB
Get Next IORB
Should Not Block
Error ?
Yes
Return
Unsuccessful Return
No
Get Record Size & Func Code
MTAIRQ
Queue up this IORB
MTAIRQ
Queue up Specified IORB
Frames Cnt → IRBCNT
Char Cnt → IRBOC
Func Code → IRBSTS
(Adr of List of Buf Pgs)→ IRBPB
MTIRQ2:
For Each Pg in Buf:
1. Put Phy Pg Adr in LH of Each Buf Pg Ptr
2. Call PHYSIO & Dev. Dep Code to Set up Chan Command Word in Xfer List for the Pg.
Store 0 at End of Xfer List
Store List Tail,, Head → IRBXFL
Data Mode, Density, Parity → IRBMOD
MTCHKA,, MTAINT → IRBIVA
QUEIRB
Queue up IORB
Return
QUEIRB
Set Status Bits for IORB
Step IORB for this Unit
Get Adr of CDB & UDB for this Unit
PHYSIO pPHY2
Call Physio to Queue Request
Return

SM2

SEQUENTIAL OUTPUT – MTA
STRING & BYTE DEV. DEP. CODE



CLOSF – MAGTAPE



CM1

OPENF-MAGTAPE Comments

(1) One can open for read and write only in dump mode.

(2) FILCNT/Count of bytes left to use in current page of buffer.
FILLEN/Count of bytes in buffer.
FILBYN/Buffer byte number user is referencing.

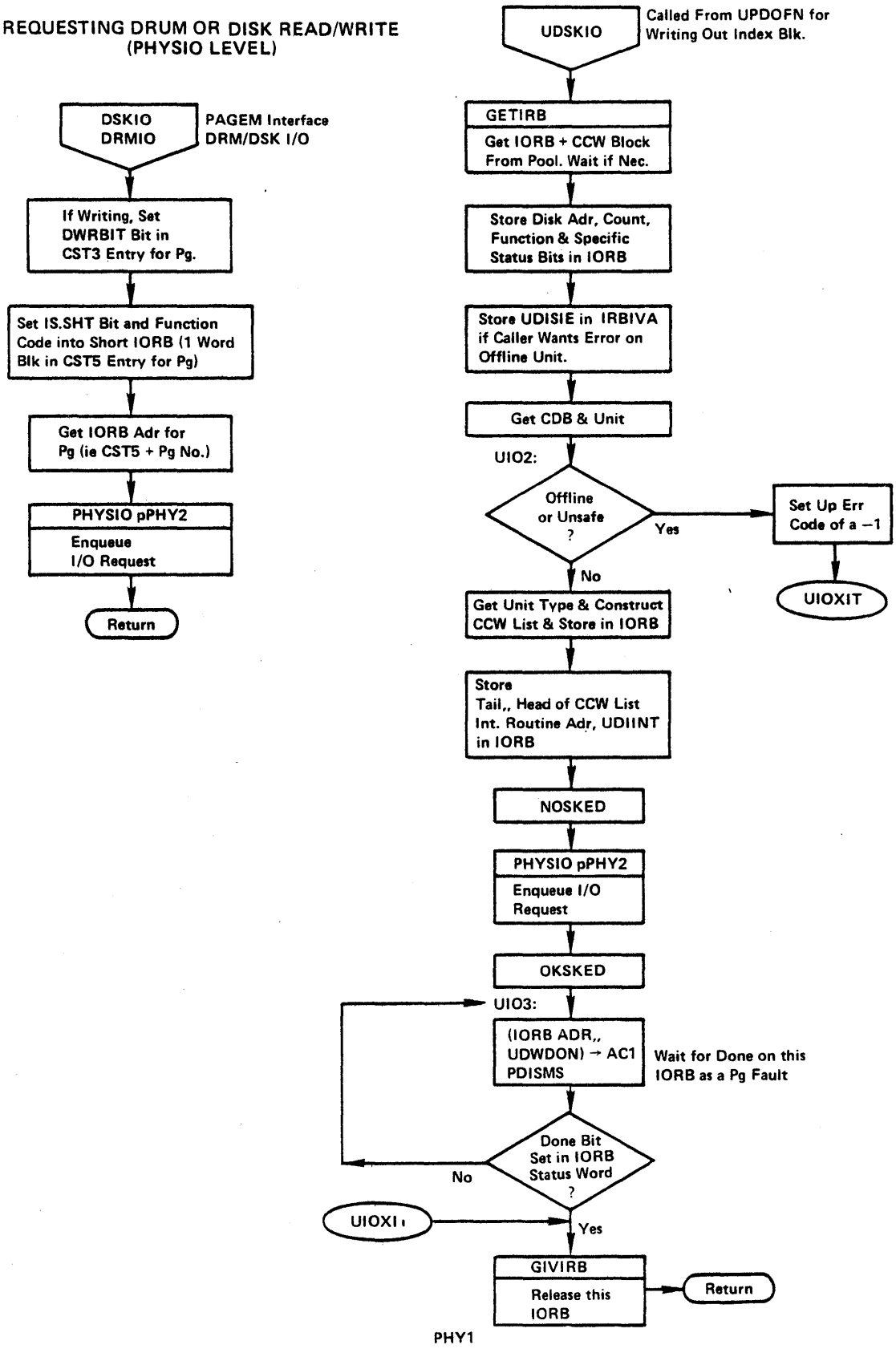
CLOSF - MAGTAPE Comments

(1) Since the monitor reads ahead, backspacing to just after last user record read may be necessary.

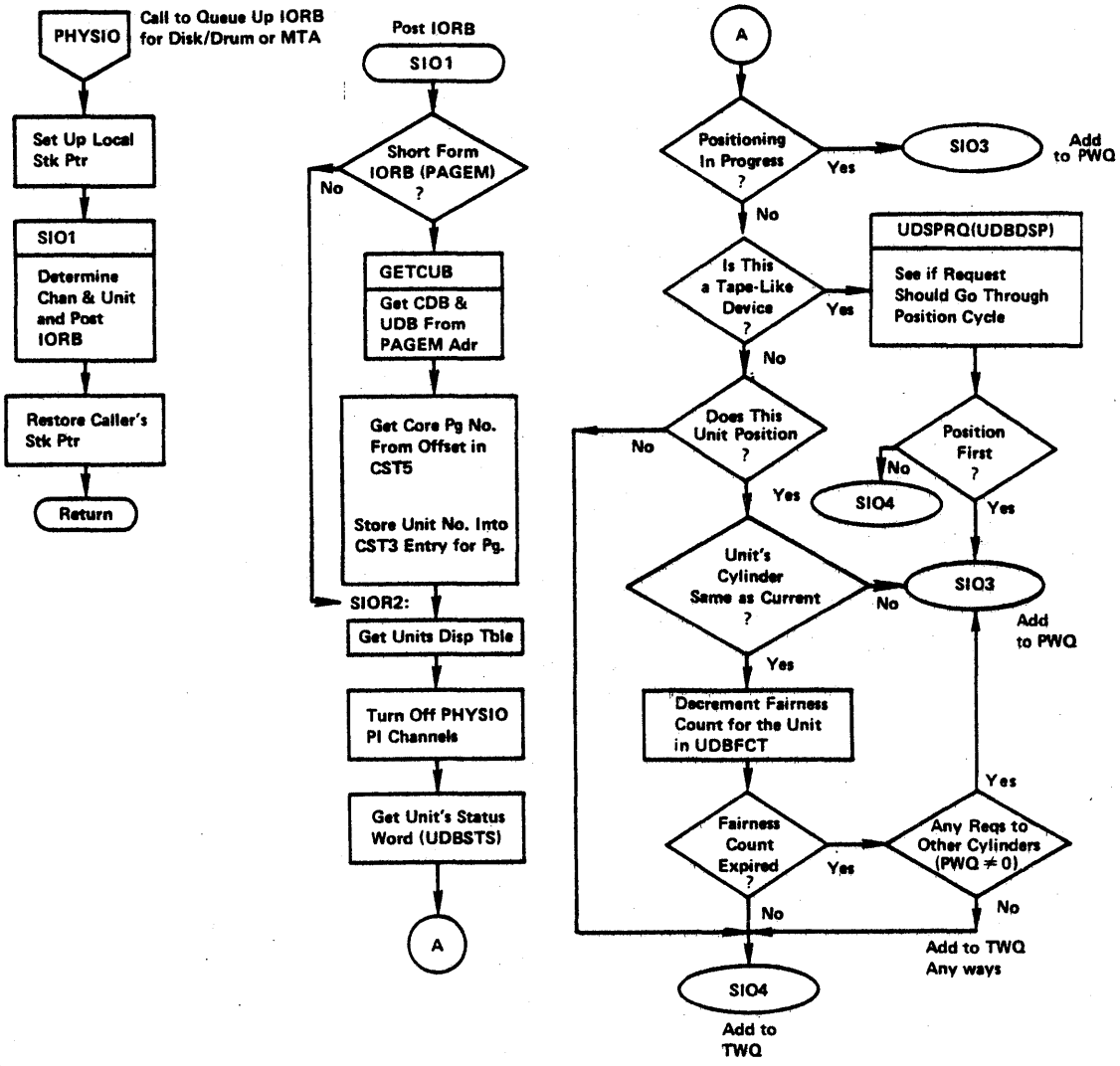
REQUESTING DISK/MTA I/O & INTERRUPT HANDLING FLOWCHARTS (PHYSIO LEVEL)

DRMIO/DSKIO/UDSKIO -	Requesting Drum or Disk Read/Write			PHY1
PHYSIO -	Queue Up IORB Request for Disk, Drum or Magtape			PHY2
	SIO1 -	Post IORB		PHY2
		STRTPS -	Start Unit Positioning	PHY3
		STRTIO -	Start Unit Transferring	PHY3a
PHYINT -	Disk and Magtape Interrupt Handler			PHY4
DONIRB -	Post IORB as Done			PHY5
	SWPDON/UDIINT -		Housekeep for Drum/Disk Done	PHY8
	MTAINT -		Housekeep for Magtape Done	PHY9
SCHSEK -	Schedule "Best" Seek Request			PHY6
SCHXFR -	Schedule "Best" Transfer Request			PHY7

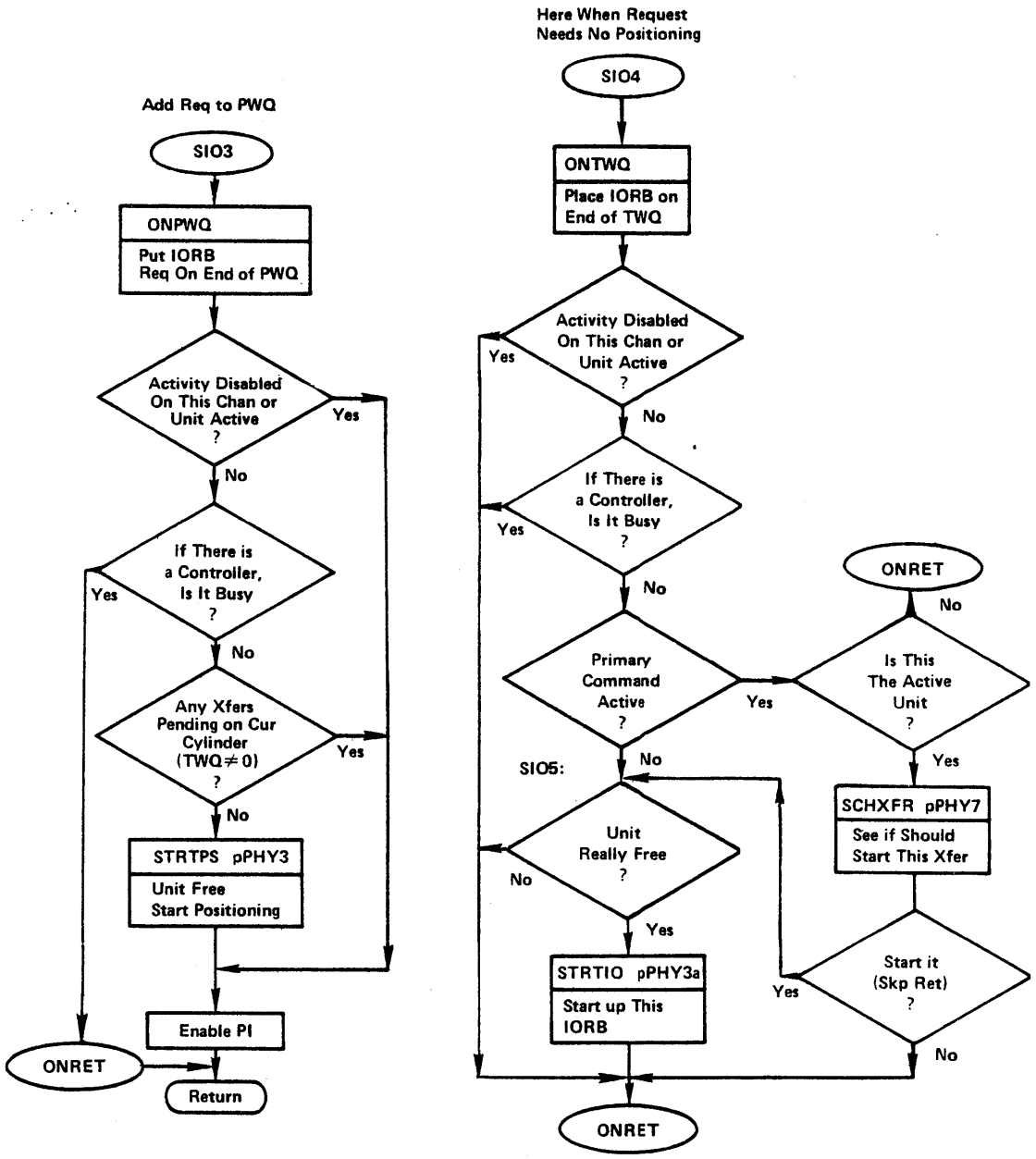
REQUESTING DRUM OR DISK READ/WRITE (PHYSIO LEVEL)



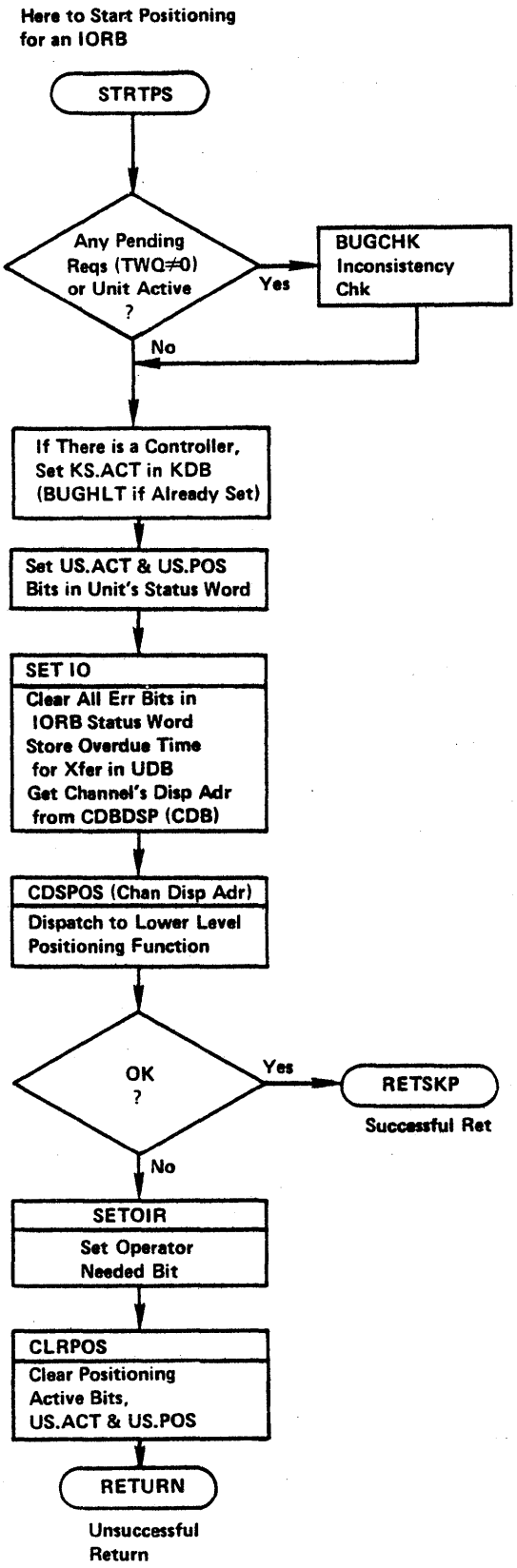
PHY1


PHYSIO
Call to Queue Up IORB for Disk/Drum or MTA
Set Up Local Stk Ptr
SIO1
Determine Chan & Unit and Post IORB
Restore Caller's Stk Ptr
Return
Post IORB
SIO1
Short Form IORB (PAGEM) ?
No
GETCUB
Get CDB & UDB From PAGEM Adr
Get Core Pg No. From Offset in CST5
Store Unit No. Into CST3 Entry for Pg.
SIOR2:
Get Units Disp Tble
Turn Off PHYSIO PI Channels
Get Unit's Status Word (UDBSTS)
A
A
Positioning In Progress ?
Yes
SIO3
Add to PWQ
No
Is This a Tape-Like Device ?
Yes
UDSPRQ(UDBDSP)
See if Request Should Go Through Position Cycle
No
Does This Unit Position ?
No
Yes
Position First ?
No
SIO4
Yes
Unit's Cylinder Same as Current ?
No
SIO3
Add to PWQ
Yes
Decrement Fairness Count for the Unit in UDBFCT
Fairness Count Expired ?
Yes
Any Reqs to Other Cylinders (PWQ ≠ 0)
Yes
No
No
Add to TWQ Any ways
SIO4
Add to TWQ

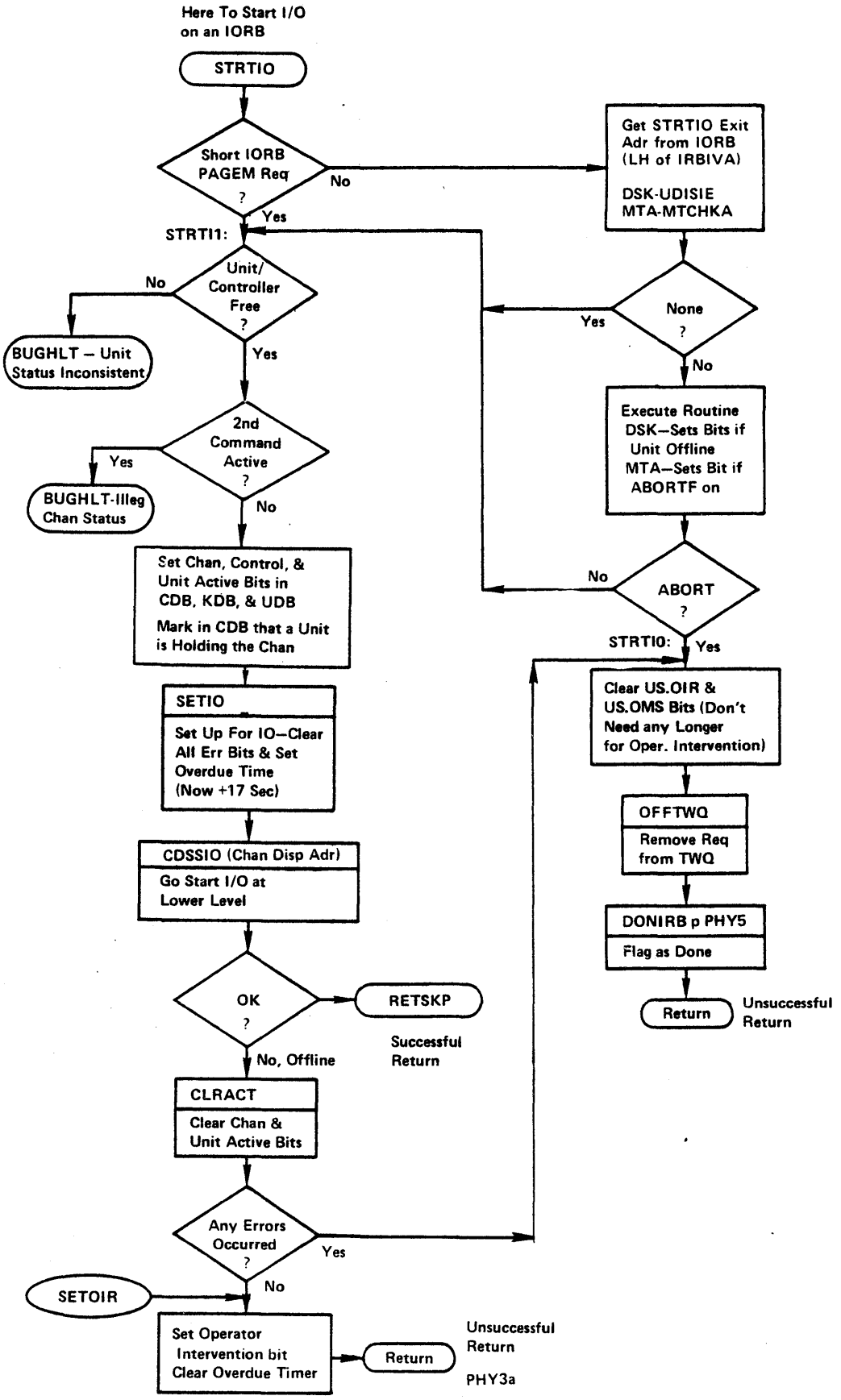
PHY2

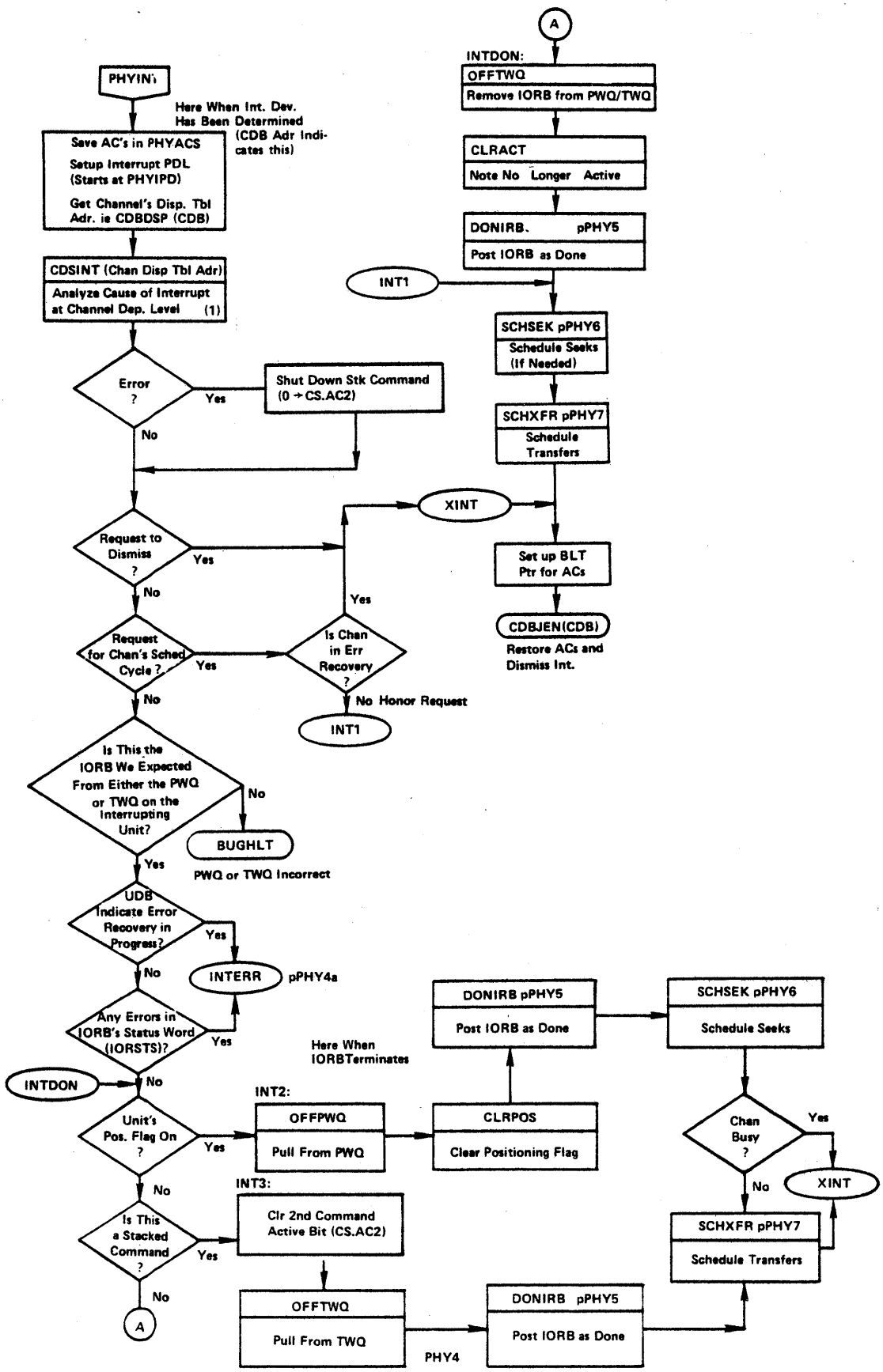

Add Req to PWQ
SIO3
ONPWQ
Put IORB Req On End of PWQ
Activity Disabled On This Chan or Unit Active ?
Yes
No
If There is a Controller, Is It Busy ?
Yes
No
Any Xfers Pending on Cur Cylinder (TWQ≠0) ?
Yes
No
STRTPS pPHY3
Unit Free Start Positioning
Enable PI
ONRET
Return
Here When Request Needs No Positioning
SIO4
ONTWQ
Place IORB on End of TWQ
Activity Disabled On This Chan or Unit Active ?
Yes
No
If There is a Controller, Is It Busy ?
Yes
No
Primary Command Active ?
Yes
No
Is This The Active Unit ?
Yes
No
ONRET
SIO5:
Unit Really Free ?
No
Yes
STRTIO pPHY3a
Start up This IORB
SCHXFR pPHY7
See if Should Start This Xfer
Start it (Skp Ret) ?
Yes
No
ONRET

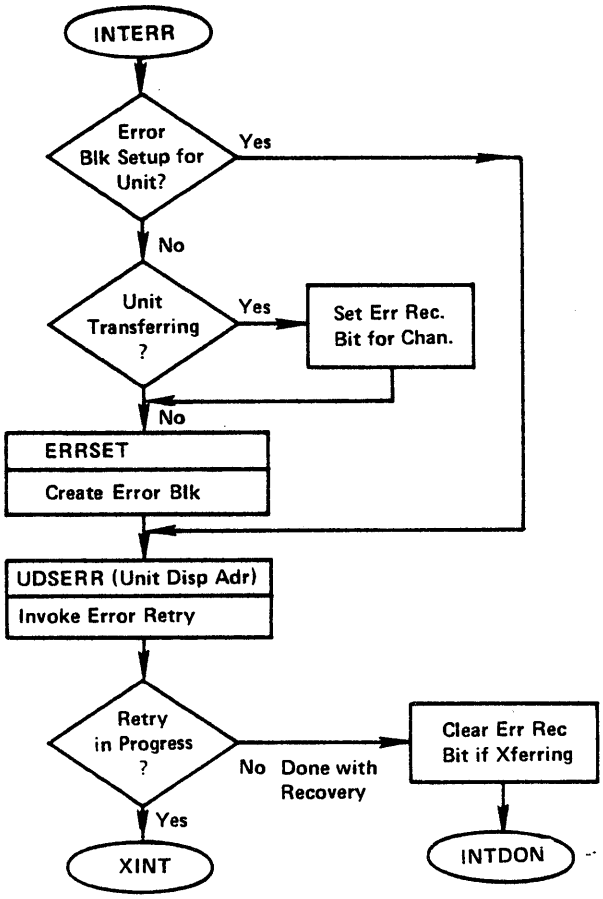
PHY2a


Here to Start Positioning for an IORB
STRTPS
Any Pending Reqs (TWQ≠0) or Unit Active ?
Yes
BUGCHK Inconsistency Chk
No
If There is a Controller, Set KS.ACT in KDB (BUGHLT if Already Set)
Set US.ACT & US.POS Bits in Unit's Status Word
SET IO
Clear All Err Bits in IORB Status Word
Store Overdue Time for Xfer in UDB
Get Channel's Disp Adr from CDBDSP (CDB)
CDSPOS (Chan Disp Adr)
Dispatch to Lower Level Positioning Function
OK ?
Yes
RETSKP
Successful Ret
No
SETOIR
Set Operator Needed Bit
CLRPOS
Clear Positioning Active Bits, US.ACT & US.POS
RETURN
Unsuccessful Return

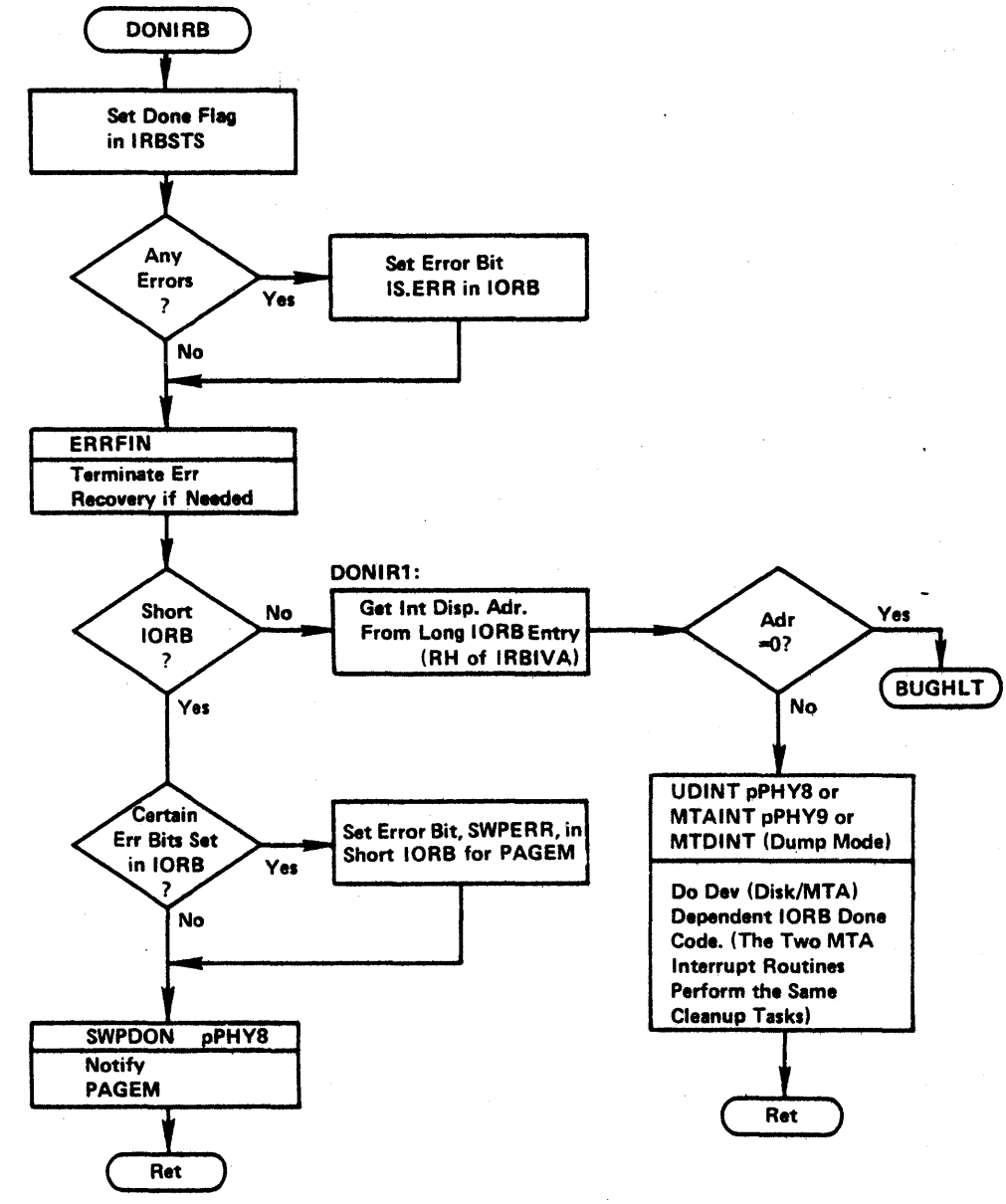
PHY3


Here To Start I/O on an IORB
STRTIO
Short IORB PAGEM Req ?
No
Yes
Get STRTIO Exit Adr from IORB (LH of IRBIVA)
DSK-UDISIE
MTA-MTCHKA
STRTI1:
Unit/ Controller Free ?
No
Yes
BUGHLT – Unit Status Inconsistent
None ?
Yes
No
2nd Command Active ?
Yes
No
BUGHLT-Illeg Chan Status
Execute Routine
DSK–Sets Bits if Unit Offline
MTA–Sets Bit if ABORTF on
Set Chan, Control, & Unit Active Bits in CDB, KDB, & UDB
Mark in CDB that a Unit is Holding the Chan
No
ABORT ?
STRTIO:
Yes
SETIO
Set Up For IO–Clear All Err Bits & Set Overdue Time (Now +17 Sec)
Clear US.OIR & US.OMS Bits (Don't Need any Longer for Oper. Intervention)
OFFTWQ
Remove Req from TWQ
CDSSIO (Chan Disp Adr)
Go Start I/O at Lower Level
DONIRB p PHY5
Flag as Done
OK ?
RETSKP
Successful Return
Return
Unsuccessful Return
No, Offline
CLRACT
Clear Chan & Unit Active Bits
Any Errors Occurred ?
Yes
No
SETOIR
Set Operator Intervention bit
Clear Overdue Timer
Return
Unsuccessful Return
PHY3a

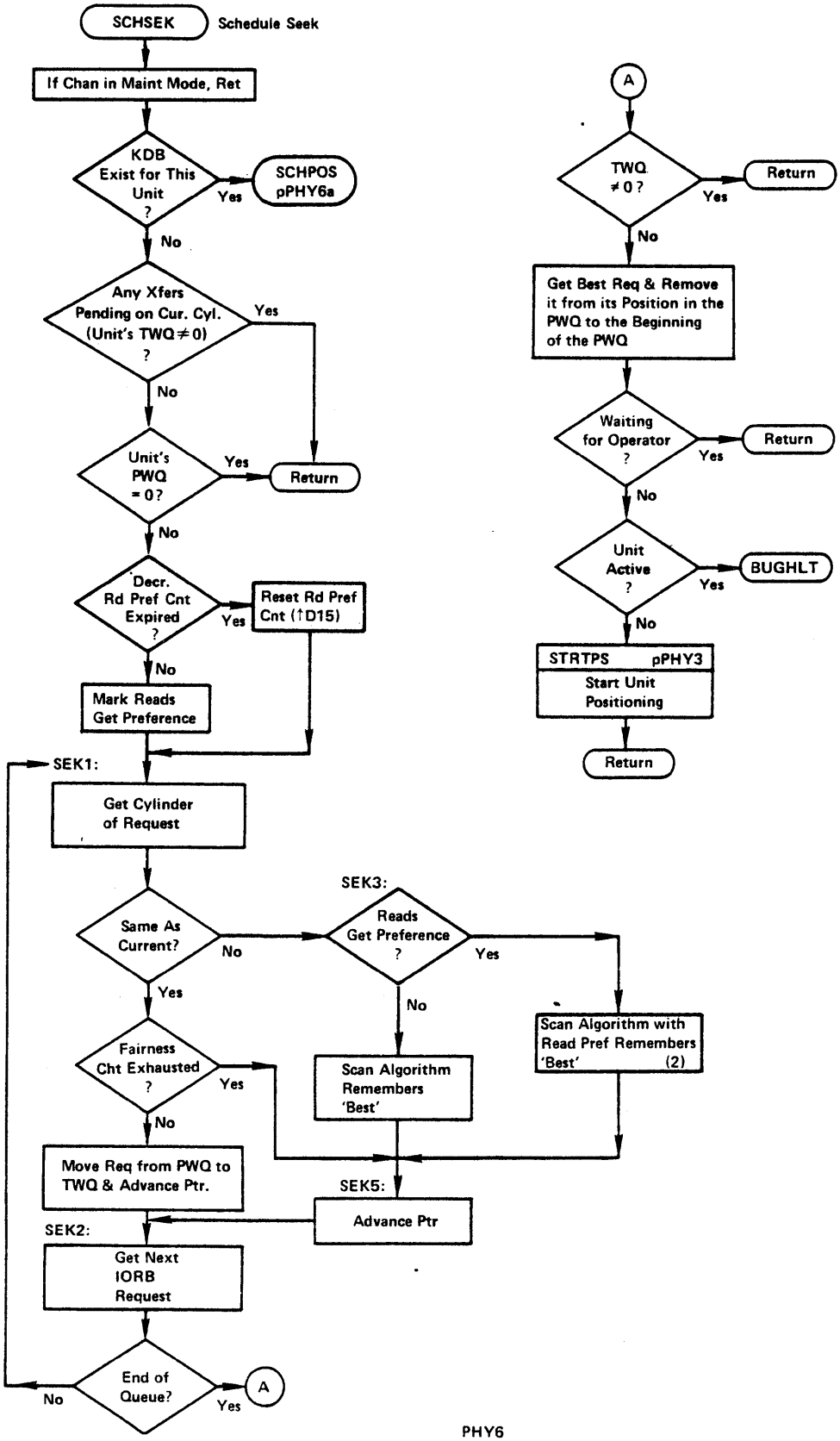

A
INTDON:
OFFTWQ
Remove IORB from PWQ/TWQ
CLRACT
Note No Longer Active
DONIRB. pPHY5
Post IORB as Done
INT1
SCHSEK pPHY6
Schedule Seeks (If Needed)
SCHXFR pPHY7
Schedule Transfers
XINT
Set up BLT Ptr for ACs
CDBJEN(CDB)
Restore ACs and Dismiss Int.
PHYIN
Here When Int. Dev. Has Been Determined (CDB Adr Indicates this)
Save AC's in PHYACS
Setup Interrupt PDL (Starts at PHYIPD)
Get Channel's Disp. Tbl Adr. ie CDBDSP (CDB)
CDSINT (Chan Disp Tbl Adr)
Analyze Cause of Interrupt at Channel Dep. Level (1)
Error ?
Yes
Shut Down Stk Command (0 → CS.AC2)
No
Request to Dismiss ?
Yes
No
Request for Chan's Sched Cycle ?
Yes
Is Chan in Err Recovery ?
Yes
No Honor Request
No
Is This the IORB We Expected From Either the PWQ or TWQ on the Interrupting Unit?
No
BUGHLT
PWQ or TWQ Incorrect
Yes
UDB Indicate Error Recovery in Progress?
Yes
INTERR pPHY4a
No
Any Errors in IORB's Status Word (IORSTS)?
Yes
INTDON
No
Here When IORBTerminates
INT2:
Unit's Pos. Flag On ?
Yes
OFFPWQ
Pull From PWQ
CLRPOS
Clear Positioning Flag
DONIRB pPHY5
Post IORB as Done
SCHSEK pPHY6
Schedule Seeks
Chan Busy ?
Yes
No
XINT
SCHXFR pPHY7
Schedule Transfers
INT3:
No
Is This a Stacked Command ?
Yes
Clr 2nd Command Active Bit (CS.AC2)
No
A
OFFTWQ
Pull From TWQ
PHY4
DONIRB pPHY5
Post IORB as Done


INTERR
Error Blk Setup for Unit?
Yes
No
Unit Transferring ?
Yes
Set Err Rec. Bit for Chan.
No
ERRSET
Create Error Blk
UDSERR (Unit Disp Adr)
Invoke Error Retry
Retry in Progress ?
No Done with Recovery
Clear Err Rec Bit if Xferring
Yes
XINT
INTDON

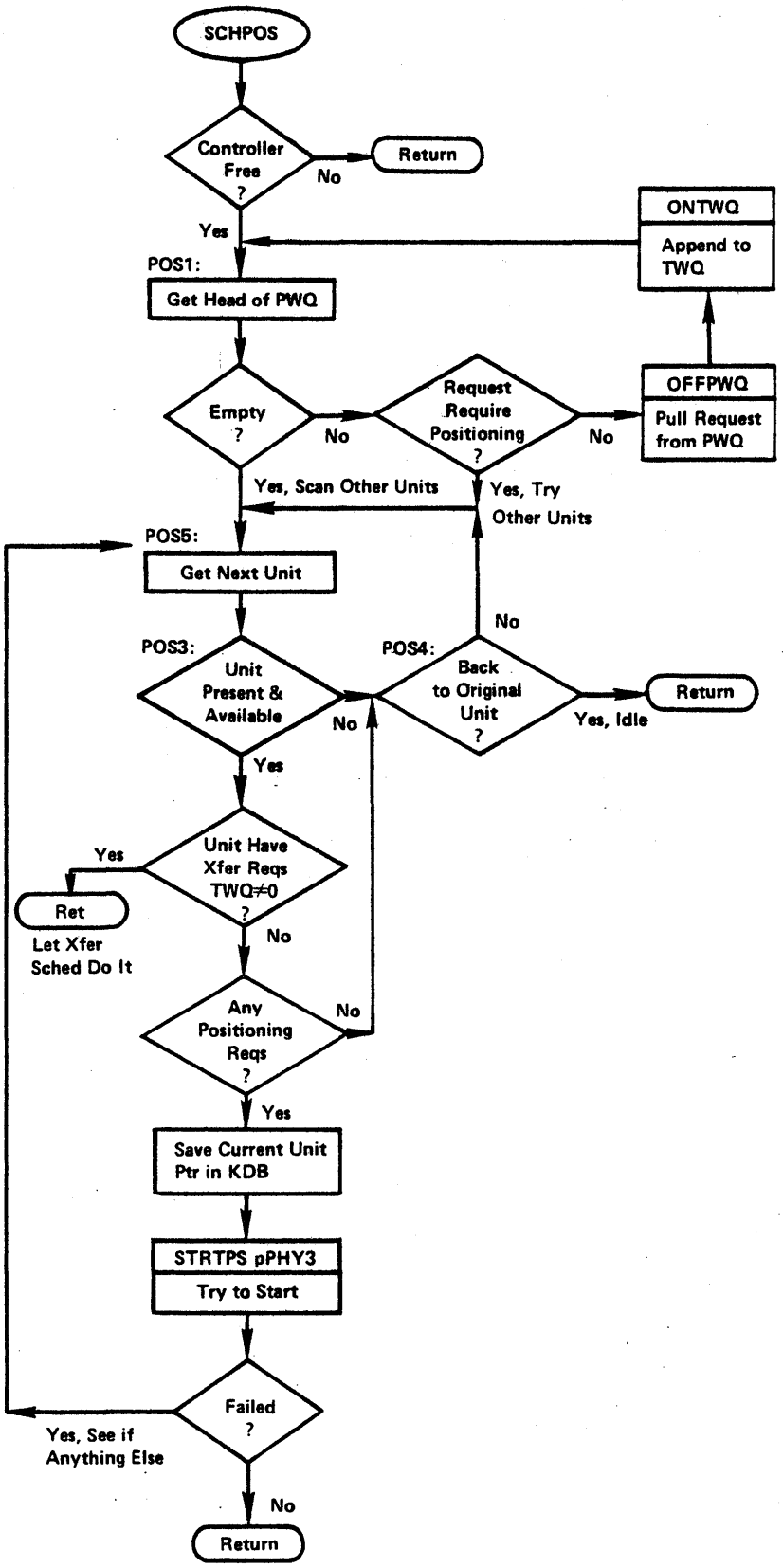
Here to Post an IORB Complete



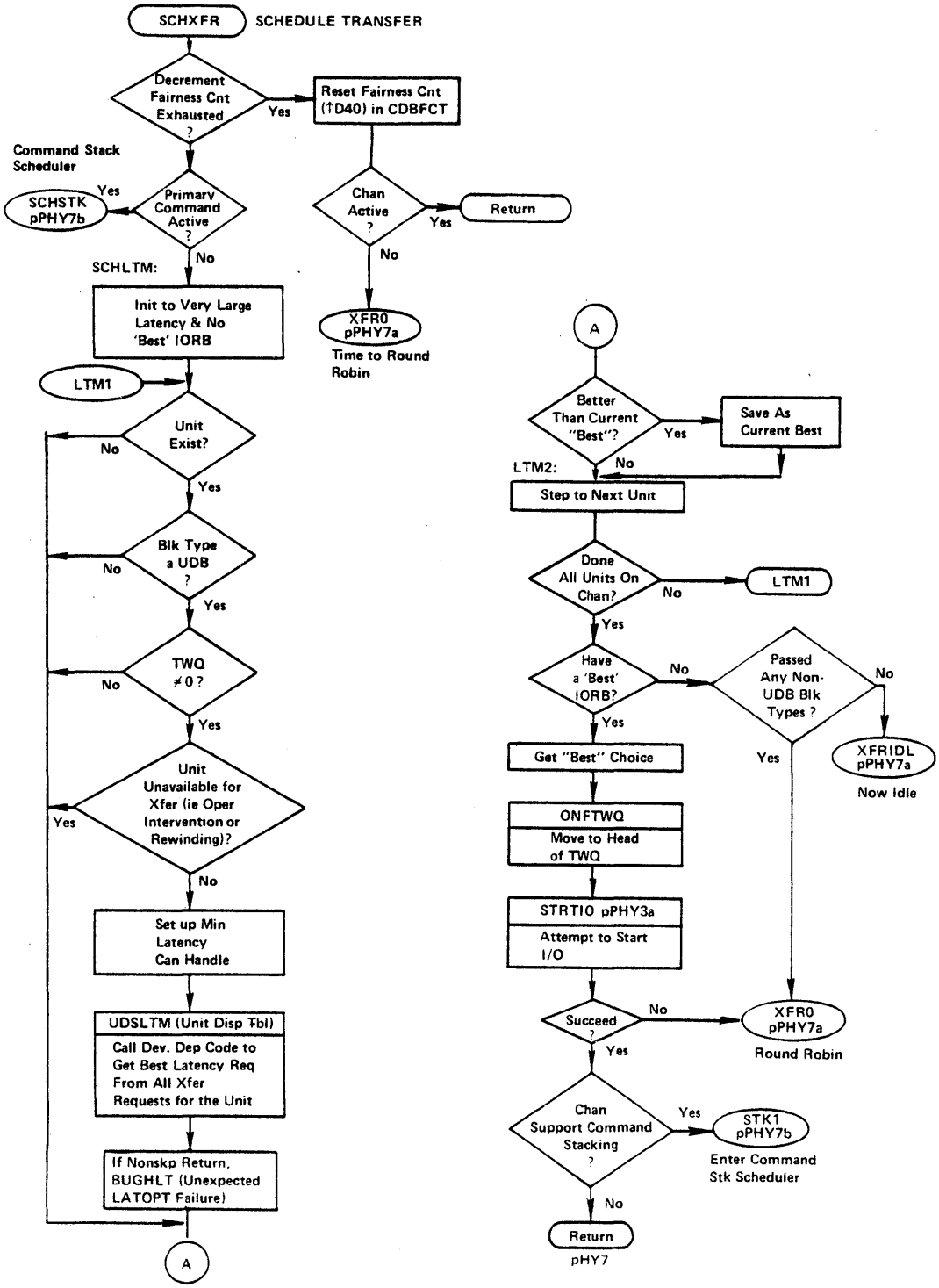
PHY5

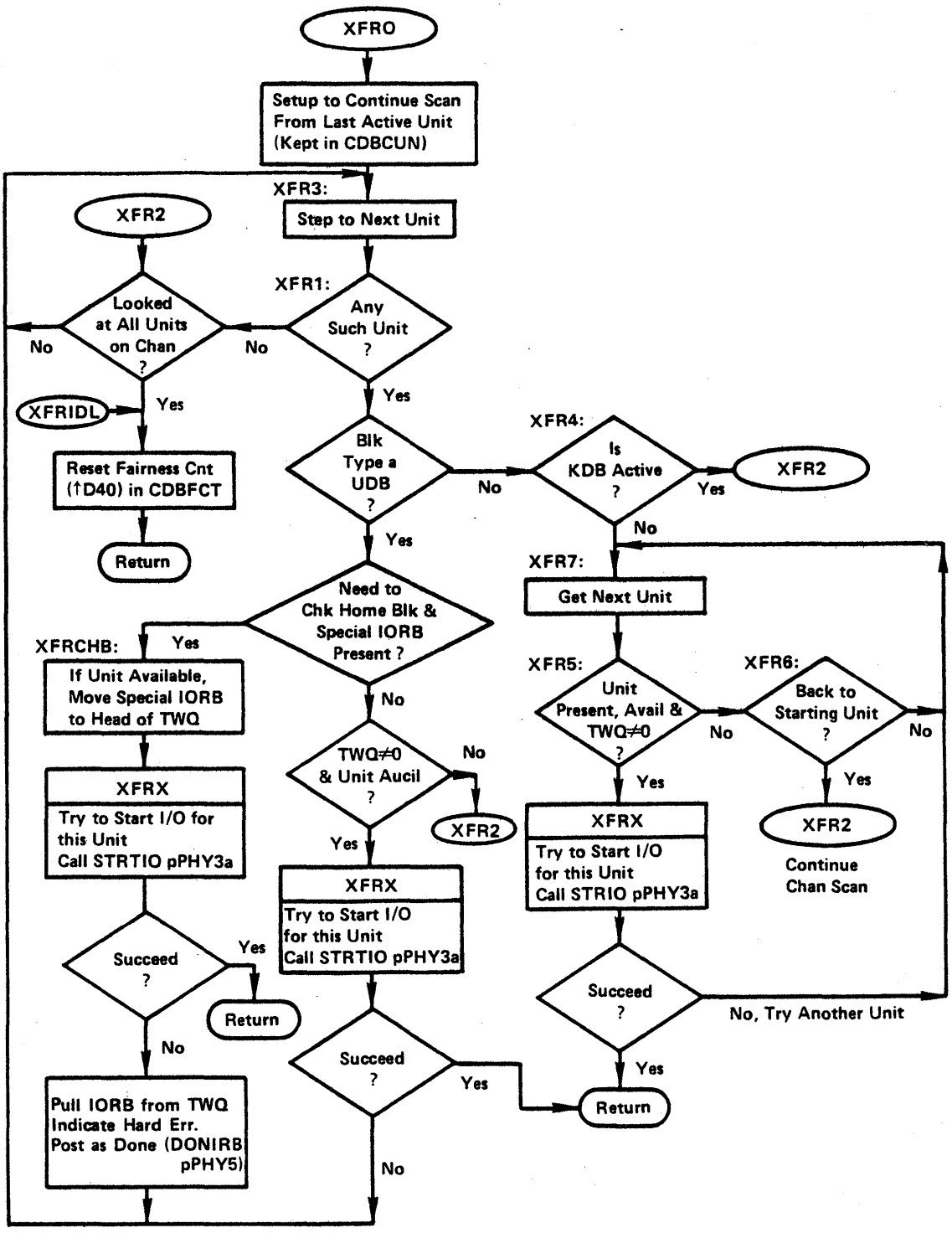

SCHSEK
Schedule Seek
If Chan in Maint Mode, Ret
KDB Exist for This Unit ?
Yes
SCHPOS pPHY6a
No
Any Xfers Pending on Cur. Cyl. (Unit's TWQ≠0) ?
Yes
No
Unit's PWQ = 0?
Yes
Return
No
Decr. Rd Pref Cnt Expired ?
Yes
Reset Rd Pref Cnt (↑D15)
No
Mark Reads Get Preference
SEK1:
Get Cylinder of Request
SEK3:
Same As Current?
No
Reads Get Preference ?
Yes
Yes
No
Fairness Cht Exhausted ?
Yes
Scan Algorithm Remembers 'Best'
Scan Algorithm with Read Pref Remembers 'Best' (2)
No
Move Req from PWQ to TWQ & Advance Ptr.
SEK5:
Advance Ptr
SEK2:
Get Next IORB Request
End of Queue?
No
Yes
A
A
TWQ ≠0?
Yes
Return
No
Get Best Req & Remove it from its Position in the PWQ to the Beginning of the PWQ
Waiting for Operator ?
Yes
Return
No
Unit Active ?
Yes
BUGHLT
No
STRTPS pPHY3
Start Unit Positioning
Return

PHY6

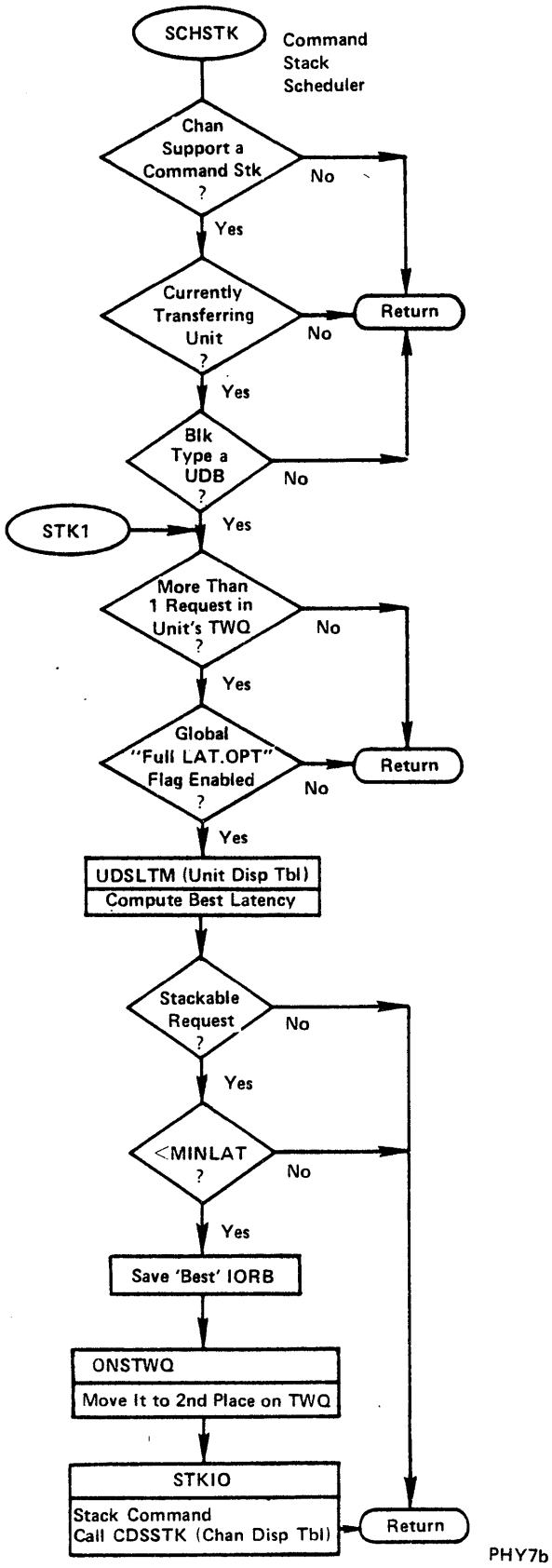

SCHPOS
Controller Free ?
No
Return
Yes
ONTWQ
Append to TWQ
POS1:
Get Head of PWQ
Empty ?
No
Request Require Positioning ?
No
OFFPWQ
Pull Request from PWQ
Yes, Scan Other Units
Yes, Try Other Units
POS5:
Get Next Unit
No
POS3:
Unit Present & Available
No
POS4:
Back to Original Unit ?
Return
Yes, Idle
Yes
Yes
Unit Have Xfer Reqs TWQ≠0 ?
Ret
Let Xfer Sched Do It
No
Any Positioning Reqs ?
No
Yes
Save Current Unit Ptr in KDB
STRTPS pPHY3
Try to Start
Failed ?
Yes, See if Anything Else
No
Return

PHY6a


SCHXFR
SCHEDULE TRANSFER
Decrement Fairness Cnt Exhausted ?
Yes
Reset Fairness Cnt (↑D40) in CDBFCT
Command Stack Scheduler
Primary Command Active ?
Yes
SCHSTK pPHY7b
No
Chan Active ?
Yes
Return
No
XFR0 pPHY7a
Time to Round Robin
SCHLTM:
Init to Very Large Latency & No 'Best' IORB
LTM1
Unit Exist?
No
Yes
Blk Type a UDB ?
No
Yes
TWQ ≠0 ?
No
Yes
Unit Unavailable for Xfer (ie Oper Intervention or Rewinding)?
Yes
No
Set up Min Latency Can Handle
UDSLTM (Unit Disp Tbl)
Call Dev. Dep Code to Get Best Latency Req From All Xfer Requests for the Unit
If Nonskp Return, BUGHLT (Unexpected LATOPT Failure)
A
A
Better Than Current "Best"?
Yes
Save As Current Best
No
LTM2:
Step to Next Unit
Done All Units On Chan?
No
LTM1
Yes
Have a 'Best' IORB?
No
Passed Any Non-UDB Blk Types ?
No
XFRIDL pPHY7a
Now Idle
Yes
Yes
Get "Best" Choice
ONFTWQ
Move to Head of TWQ
STRTIO pPHY3a
Attempt to Start I/O
Succeed ?
No
XFR0 pPHY7a
Round Robin
Yes
Chan Support Command Stacking ?
Yes
STK1 pPHY7b
Enter Command Stk Scheduler
No
Return
pHY7

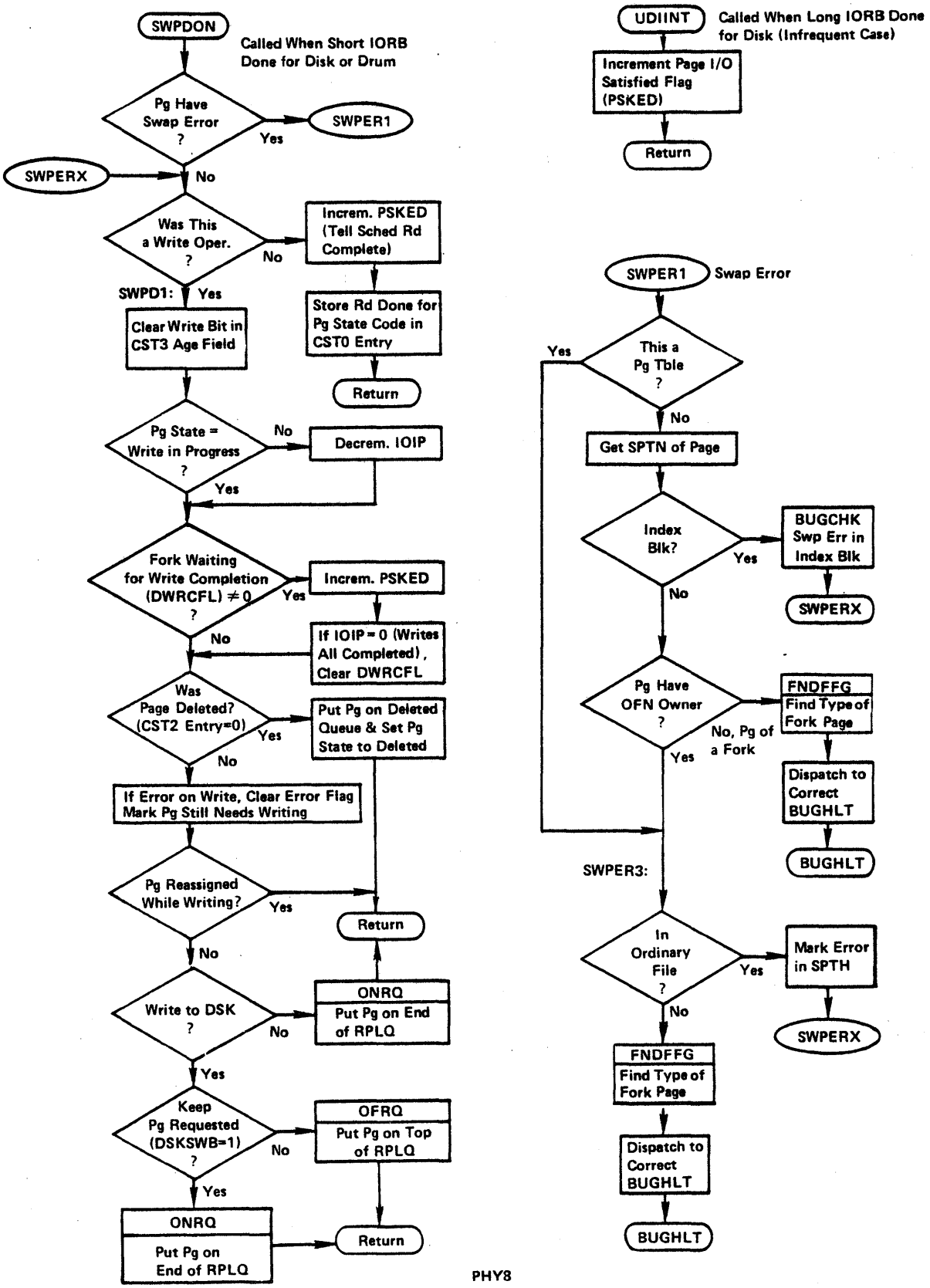

XFRO
Setup to Continue Scan From Last Active Unit (Kept in CDBCUN)
XFR3:
Step to Next Unit
XFR2
XFR1:
Any Such Unit ?
No
Looked at All Units on Chan ?
No
Yes
XFRIDL
Yes
Reset Fairness Cnt (↑D40) in CDBFCT
Return
Blk Type a UDB ?
No
XFR4:
Is KDB Active ?
Yes
XFR2
No
Yes
XFR7:
Get Next Unit
Need to Chk Home Blk & Special IORB Present ?
XFRCHB:
Yes
If Unit Available, Move Special IORB to Head of TWQ
XFRX
Try to Start I/O for this Unit
Call STRTIO pPHY3a
Succeed ?
Yes
Return
No
Pull IORB from TWQ
Indicate Hard Err.
Post as Done (DONIRB pPHY5)
No
TWQ≠0 & Unit Aucil ?
No
XFR2
Yes
XFRX
Try to Start I/O for this Unit
Call STRTIO pPHY3a
Succeed ?
Yes
Return
No
XFR5:
Unit Present, Avail & TWQ≠0 ?
No
XFR6:
Back to Starting Unit ?
No
Yes
XFRX
Try to Start I/O for this Unit
Call STRIO pPHY3a
Yes
XFR2
Continue Chan Scan
Succeed ?
No, Try Another Unit
Yes
Return

PHY7a

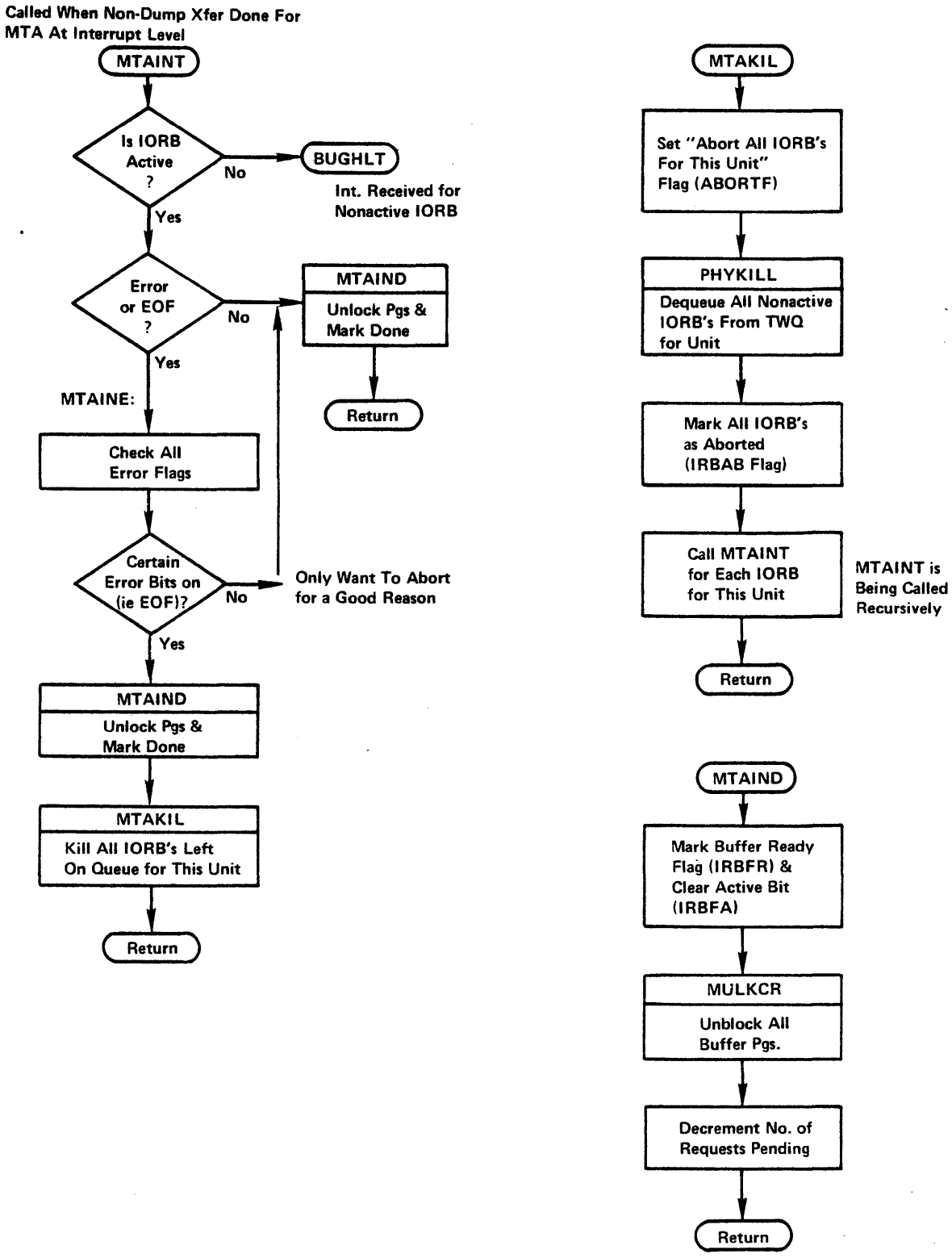

SCHSTK
Command Stack Scheduler
Chan Support a Command Stk ?
No
Yes
Currently Transferring Unit ?
No
Return
Yes
Blk Type a UDB ?
No
STK1
Yes
More Than 1 Request in Unit's TWQ ?
No
Yes
Global "Full LAT.OPT" Flag Enabled ?
No
Return
Yes
UDSLTM (Unit Disp Tbl)
Compute Best Latency
Stackable Request ?
No
Yes
<MINLAT ?
No
Yes
Save 'Best' IORB
ONSTWQ
Move It to 2nd Place on TWQ
STKIO
Stack Command
Call CDSSTK (Chan Disp Tbl)
Return

PHY7b

"INTERRUPT DONE" DSK/DRUM DEPENDENT CODE



."INTERRUPT DONE" MAGTAPE DEPENDENT CODE



PHY9

Requesting DISK/MTA I/O Comments

SIO1

(1) The algorithm for queuing up a MTA request is:

If the request requires positioning, append the request to the PWQ.

If the request requires no positioning (i.e., Read/Write Forward or Read Reverse) append the request to the TWQ only if the PWQ is empty. Otherwise, append it to the PWQ.

DSK/MTA Interrupt Handling Comments

PHYINT

(1) The channel dependent routine (RH2INT for RH2Øs) is called to analyze the interrupt. Lower level routines called by RH2INT (i.e., Unit dependent routines) return an argument in AC, P4, to PHYINT to indicate whether to dismiss the interrupt (P4 = Ø), to schedule another channel cycle right away (P4 < Ø) or to housekeep the current request (P4 > Ø) before scheduling another channel cycle. The channel dependent routine also records error information so that PHYINT can see if error recovery is in progress or should be started.

The request to dismiss (P4 = Ø) is invoked for example when the done flag is on and the channel is not occupied. The request for an immediate channel cycle (P4 < Ø) is made when a positioning done interrupt has occurred and there is no transfer in progress. Transfer Done requests will require further housekeeping (P4 > Ø) by PHYINT before scheduling another channel cycle.

SCHSEK

(2) The scan algorithm with read preference in effect performs as follows:

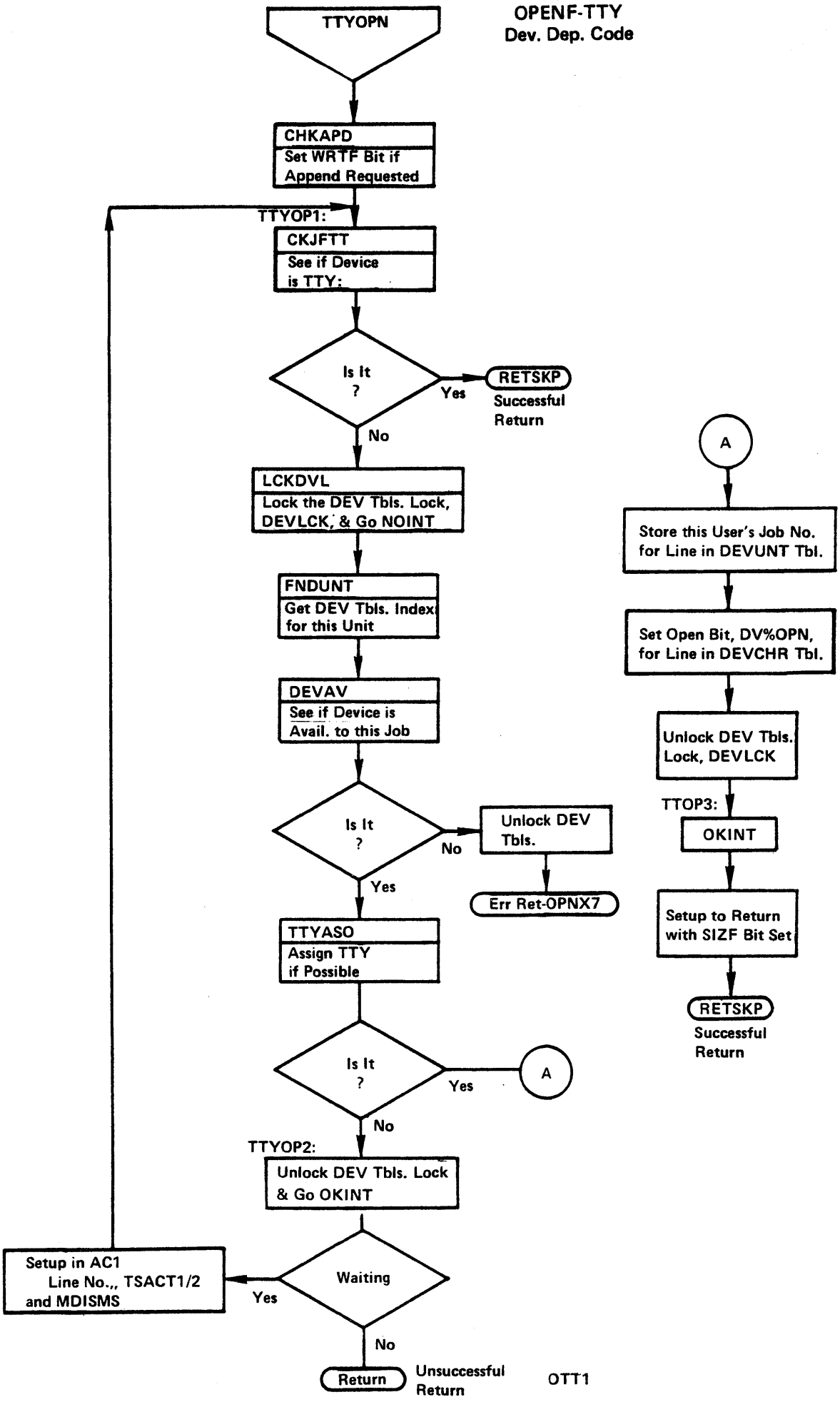
Take the next higher-numbered cylinder read request from the current cylinder. If none, take the next higher-numbered cylinder (write) request from the current cylinder.

If none, take the lowest numbered cylinder read request from the current cylinder. If none, take the lowest numbered cylinder (write) request from the current cylinder.

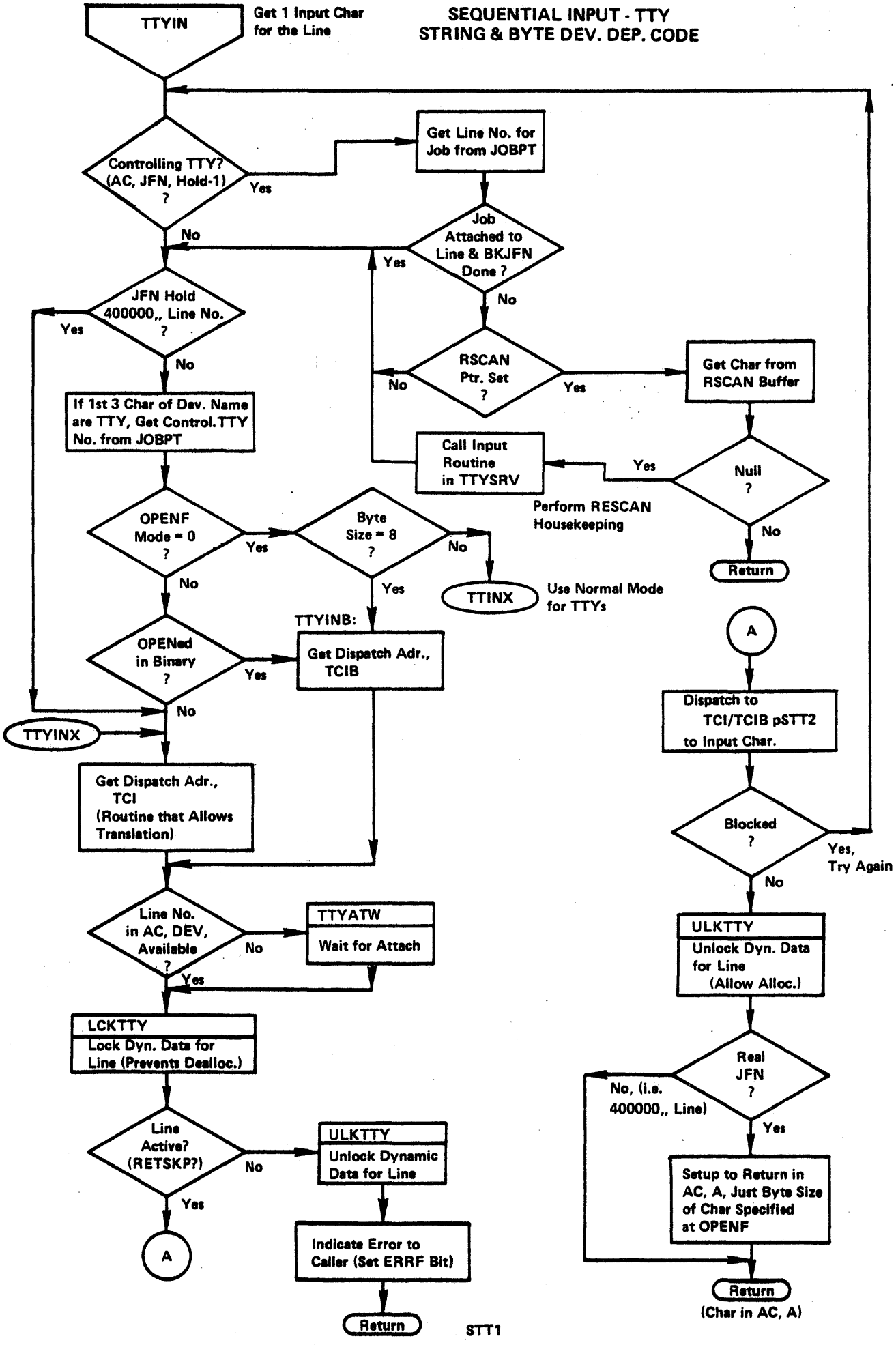
JSYS CALL FLOWCHARTS

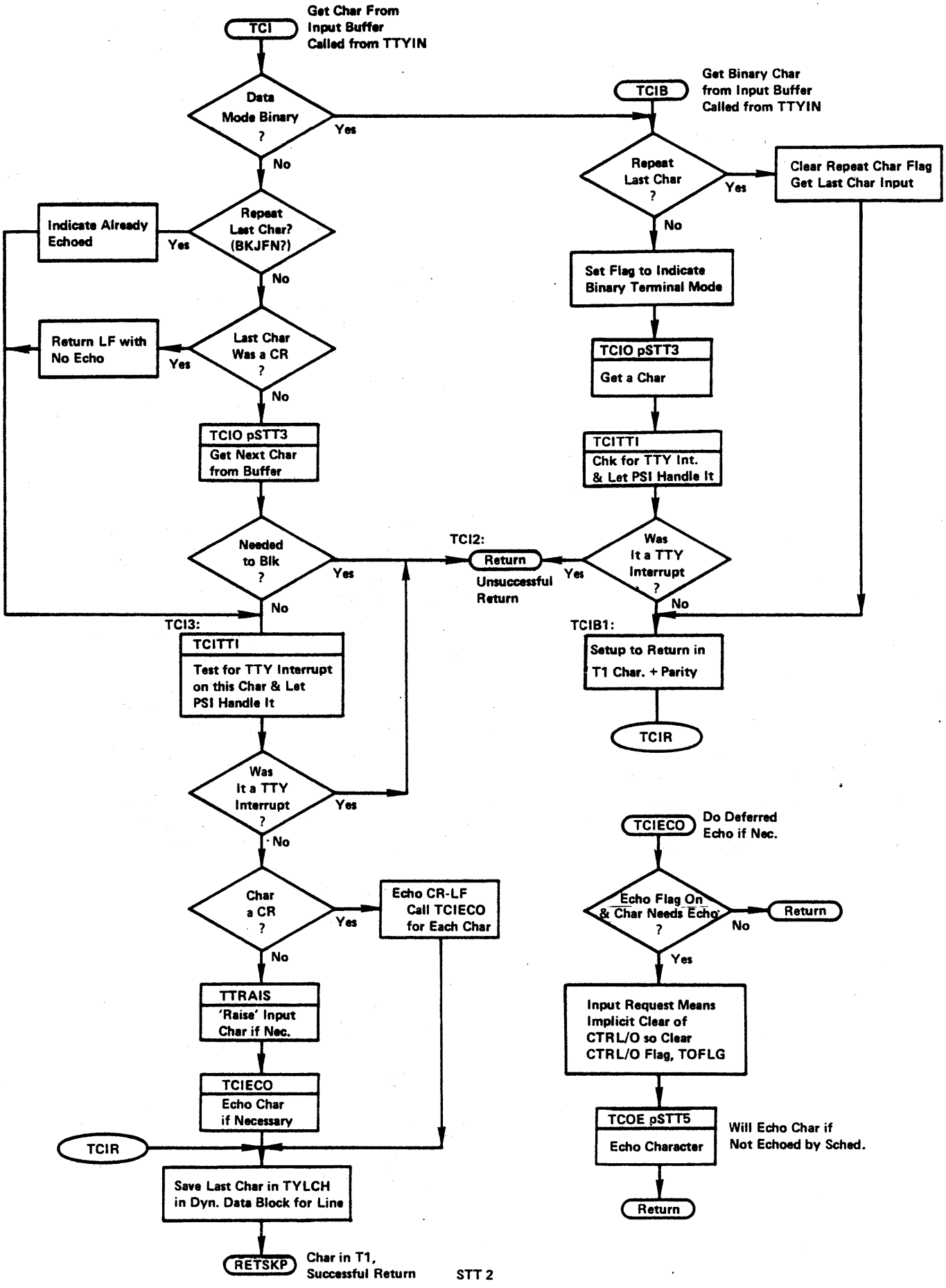
TTY DEPENDENT LEVEL

TTYOPN - Teletype Opening of a File	OTT1
TTYIN - Teletype Sequential Input	STT1
TCI/TCIB - Get Character from Line's Input Buffer	STT2
TCIO - Get a Character	STT3
TCOE - Echo Character	STT5
TTYOUT - Teletype Sequential Output	STT4
TCO/TCOB - 1st Level: Output a Single Character - Translate According to Fork's Specification	STT5
TCOY - 2nd Level: Do Links & Formats for a Particular Device	STT6
TCOUT - 3rd Level: Do Buffering and Output 1 Character	STT7
TTSND - Send Character to Line	STT8
TTYCLS - Teletype Closing of a File	CLTT1

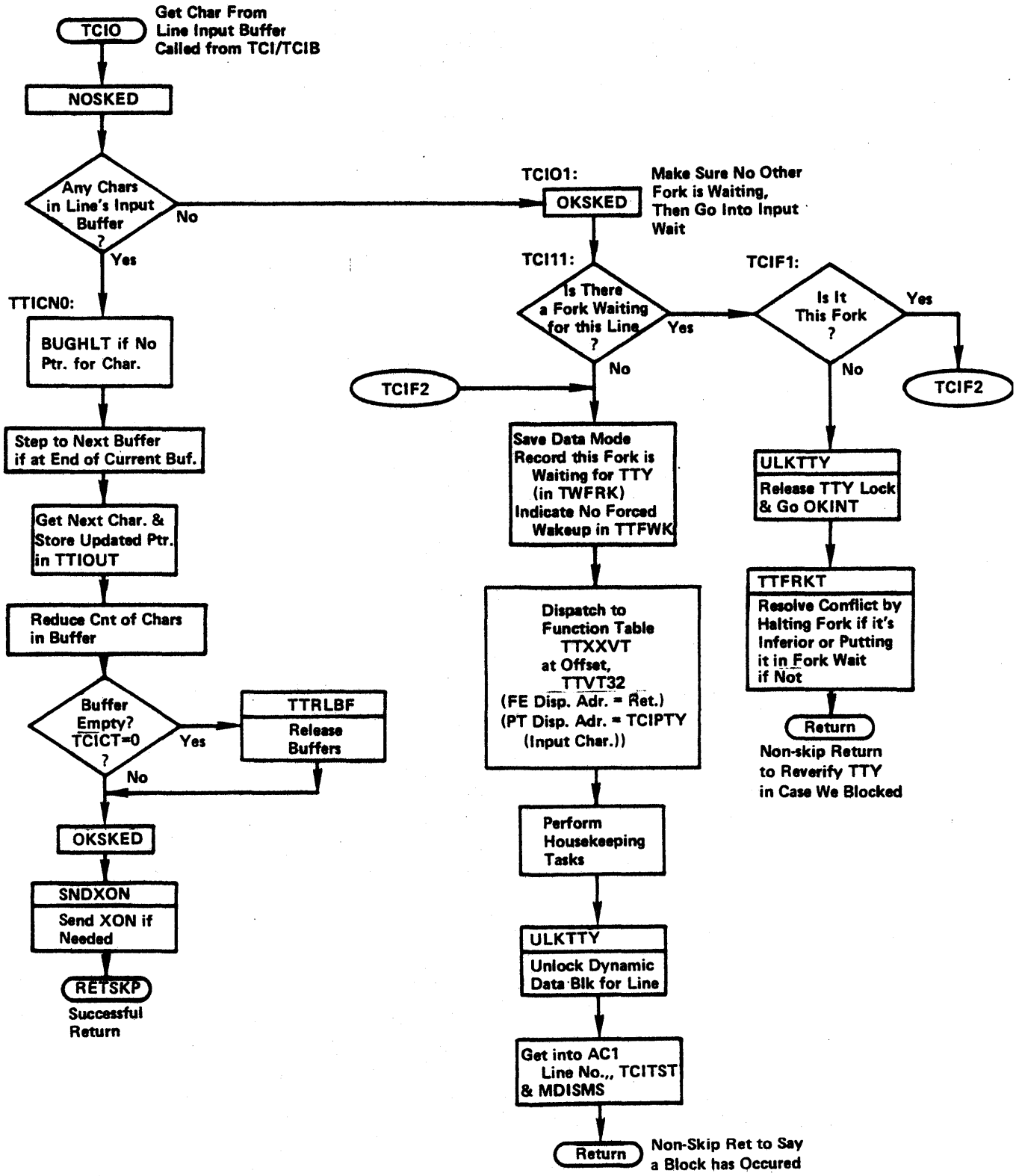

TTYOPN
OPENF-TTY
Dev. Dep. Code
CHKAPD
Set WRTF Bit if
Append Requested
TTYOP1:
CKJFTT
See if Device
is TTY:
Is It
?
Yes
RETSKP
Successful
Return
No
LCKDVL
Lock the DEV Tbls. Lock,
DEVLCK, & Go NOINT
FNDUNT
Get DEV Tbls. Index
for this Unit
DEVAV
See if Device is
Avail. to this Job
Is It
?
No
Unlock DEV
Tbls.
Err Ret-OPNX7
Yes
TTYASO
Assign TTY
if Possible
Is It
?
Yes
A
No
TTYOP2:
Unlock DEV Tbls. Lock
& Go OKINT
Setup in AC1
Line No.., TSACT1/2
and MDISMS
Waiting
Yes
No
Return
Unsuccessful
Return
A
Store this User's Job No.
for Line in DEVUNT Tbl.
Set Open Bit, DV%OPN,
for Line in DEVCHR Tbl.
Unlock DEV Tbls.
Lock, DEVLCK
TTOP3:
OKINT
Setup to Return
with SIZF Bit Set
RETSKP
Successful
Return

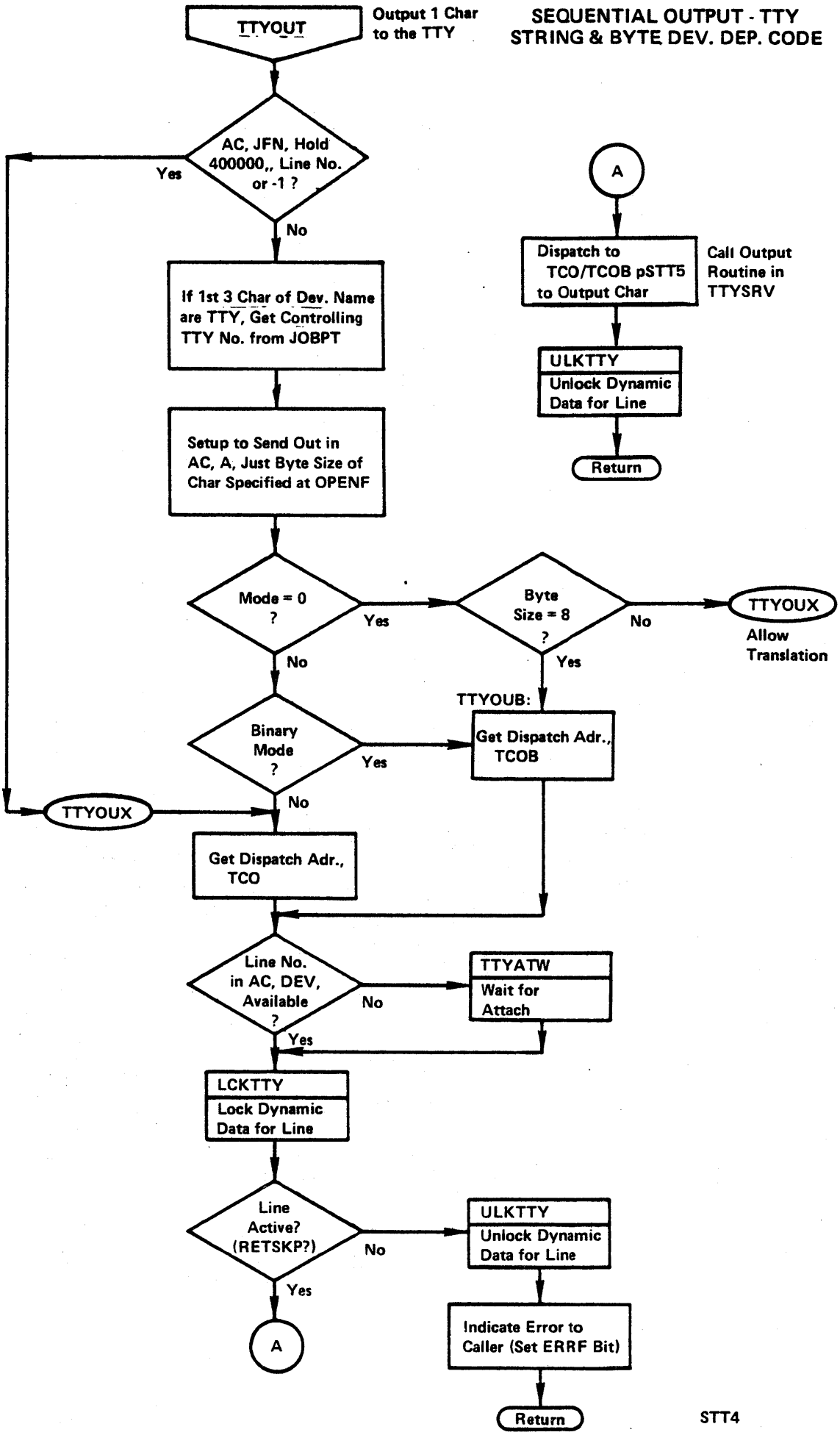
OTT1


TTYIN
Get 1 Input Char for the Line
SEQUENTIAL INPUT - TTY
STRING & BYTE DEV. DEP. CODE
Controlling TTY? (AC, JFN, Hold-1) ?
Yes
Get Line No. for Job from JOBPT
No
Job Attached to Line & BKJFN Done ?
Yes
No
JFN Hold 400000,, Line No. ?
Yes
No
RSCAN Ptr. Set ?
No
Yes
Get Char from RSCAN Buffer
If 1st 3 Char of Dev. Name are TTY, Get Control.TTY No. from JOBPT
Call Input Routine in TTYSRV
Yes
Null ?
Perform RESCAN Housekeeping
No
Return
OPENF Mode = 0 ?
Yes
Byte Size = 8 ?
No
TTINX
Use Normal Mode for TTYs
No
Yes
TTYINB:
OPENed in Binary ?
Yes
Get Dispatch Adr., TCIB
No
TTYINX
Get Dispatch Adr., TCI (Routine that Allows Translation)
A
Dispatch to TCI/TCIB pSTT2 to Input Char.
Blocked ?
Yes, Try Again
No
Line No. in AC, DEV, Available ?
No
TTYATW
Wait for Attach
Yes
ULKTTY
Unlock Dyn. Data for Line (Allow Alloc.)
LCKTTY
Lock Dyn. Data for Line (Prevents Dealloc.)
Real JFN ?
No, (i.e. 400000,, Line)
Yes
Line Active? (RETSKP?)
No
ULKTTY
Unlock Dynamic Data for Line
Setup to Return in AC, A, Just Byte Size of Char Specified at OPENF
Yes
A
Indicate Error to Caller (Set ERRF Bit)
Return
(Char in AC, A)
Return
STT1

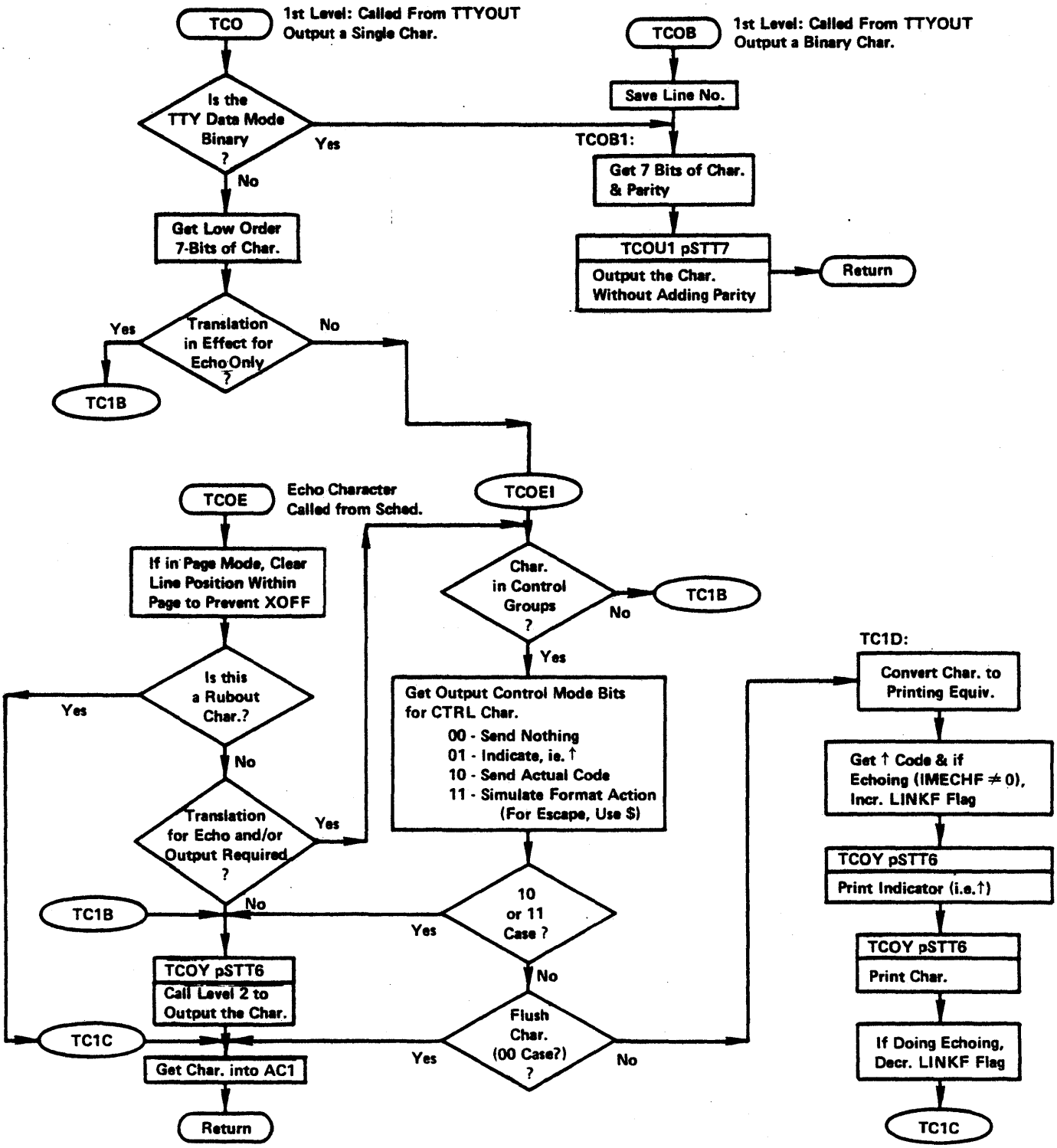

TCI
Get Char From Input Buffer Called from TTYIN
Data Mode Binary ?
Yes
No
TCIB
Get Binary Char from Input Buffer Called from TTYIN
Repeat Last Char ?
Yes
Clear Repeat Char Flag Get Last Char Input
No
Set Flag to Indicate Binary Terminal Mode
TCIO pSTT3
Get a Char
TCITTI
Chk for TTY Int. & Let PSI Handle It
Was It a TTY Interrupt ?
Yes
No
TCIB1:
Setup to Return in T1 Char. + Parity
TCIR
Repeat Last Char? (BKJFN?)
Yes
Indicate Already Echoed
No
Last Char Was a CR ?
Yes
Return LF with No Echo
No
TCIO pSTT3
Get Next Char from Buffer
Needed to Blk ?
Yes
TCI2:
Return
Unsuccessful Return
No
TCI3:
TCITTI
Test for TTY Interrupt on this Char & Let PSI Handle It
Was It a TTY Interrupt ?
Yes
No
Char a CR ?
Yes
Echo CR-LF Call TCIECO for Each Char
No
TTRAIS
'Raise' Input Char if Nec.
TCIECO
Echo Char if Necessary
TCIR
Save Last Char in TYLCH in Dyn. Data Block for Line
RETSKP
Char in T1, Successful Return
TCIECO
Do Deferred Echo if Nec.
Echo Flag On & Char Needs Echo ?
Yes
No
Return
Input Request Means Implicit Clear of CTRL/O so Clear CTRL/O Flag, TOFLG
TCOE pSTT5
Echo Character
Will Echo Char if Not Echoed by Sched.
Return

STT 2

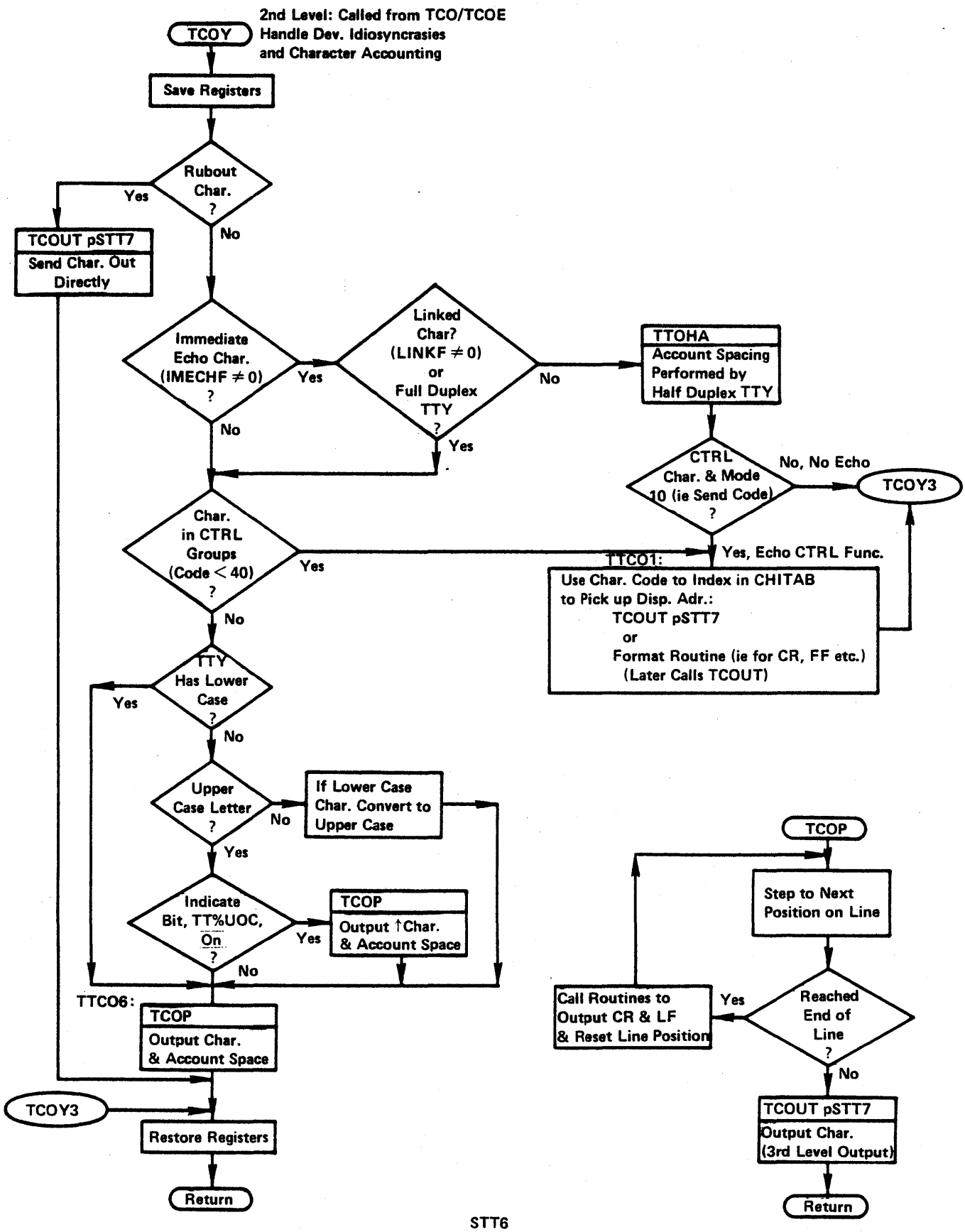

TCIO
Get Char From Line Input Buffer Called from TCI/TCIB
NOSKED
Any Chars in Line's Input Buffer ?
No
Yes
TTICN0:
BUGHLT if No Ptr. for Char.
Step to Next Buffer if at End of Current Buf.
Get Next Char. & Store Updated Ptr. in TTIOUT
Reduce Cnt of Chars in Buffer
Buffer Empty? TCICT=0 ?
Yes
TTRLBF
Release Buffers
No
OKSKED
SNDXON
Send XON if Needed
RETSKP
Successful Return
TCIO1:
OKSKED
Make Sure No Other Fork is Waiting, Then Go Into Input Wait
TCI11:
Is There a Fork Waiting for this Line ?
Yes
No
TCIF1:
Is It This Fork ?
Yes
No
TCIF2
TCIF2
Save Data Mode Record this Fork is Waiting for TTY (in TWFRK) Indicate No Forced Wakeup in TTFWK
ULKTTY
Release TTY Lock & Go OKINT
TTFRKT
Resolve Conflict by Halting Fork if it's Inferior or Putting it in Fork Wait if Not
Return
Non-skip Return to Reverify TTY in Case We Blocked
Dispatch to Function Table TTXXVT at Offset, TTVT32
(FE Disp. Adr. = Ret.)
(PT Disp. Adr. = TCIPTY (Input Char.))
Perform Housekeeping Tasks
ULKTTY
Unlock Dynamic Data Blk for Line
Get into AC1 Line No.,, TCITST & MDISMS
Return
Non-Skip Ret to Say a Block has Occured


TTYOUT
Output 1 Char to the TTY
SEQUENTIAL OUTPUT - TTY STRING & BYTE DEV. DEP. CODE
AC, JFN, Hold 400000,, Line No. or -1 ?
Yes
No
If 1st 3 Char of Dev. Name are TTY, Get Controlling TTY No. from JOBPT
Setup to Send Out in AC, A, Just Byte Size of Char Specified at OPENF
Mode = 0 ?
Yes
No
Byte Size = 8 ?
No
Yes
TTYOUX
Allow Translation
TTYOUB:
Binary Mode ?
Yes
No
Get Dispatch Adr., TCOB
TTYOUX
Get Dispatch Adr., TCO
Line No. in AC, DEV, Available ?
No
Yes
TTYATW
Wait for Attach
LCKTTY
Lock Dynamic Data for Line
Line Active? (RETSKP?)
No
Yes
ULKTTY
Unlock Dynamic Data for Line
Indicate Error to Caller (Set ERRF Bit)
Return
A
A
Dispatch to TCO/TCOB pSTT5 to Output Char
Call Output Routine in TTYSRV
ULKTTY
Unlock Dynamic Data for Line
Return
STT4

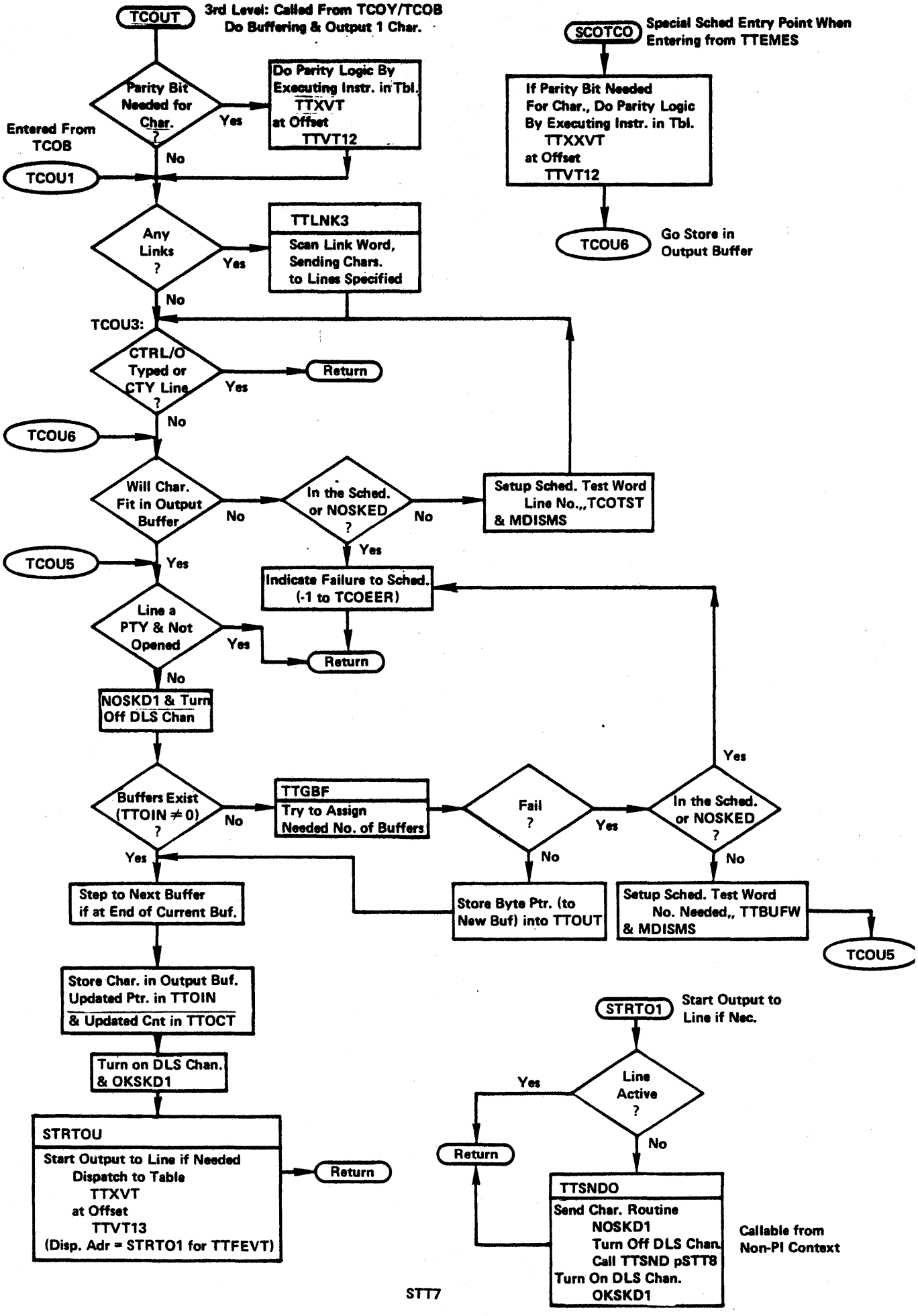
TCO - 1st Level - Translate According to Program's Desires
TCOY - 2nd Level - Do Links & Format for a Particular Device
TCOUT - 3rd Level - Do Buffering, etc.



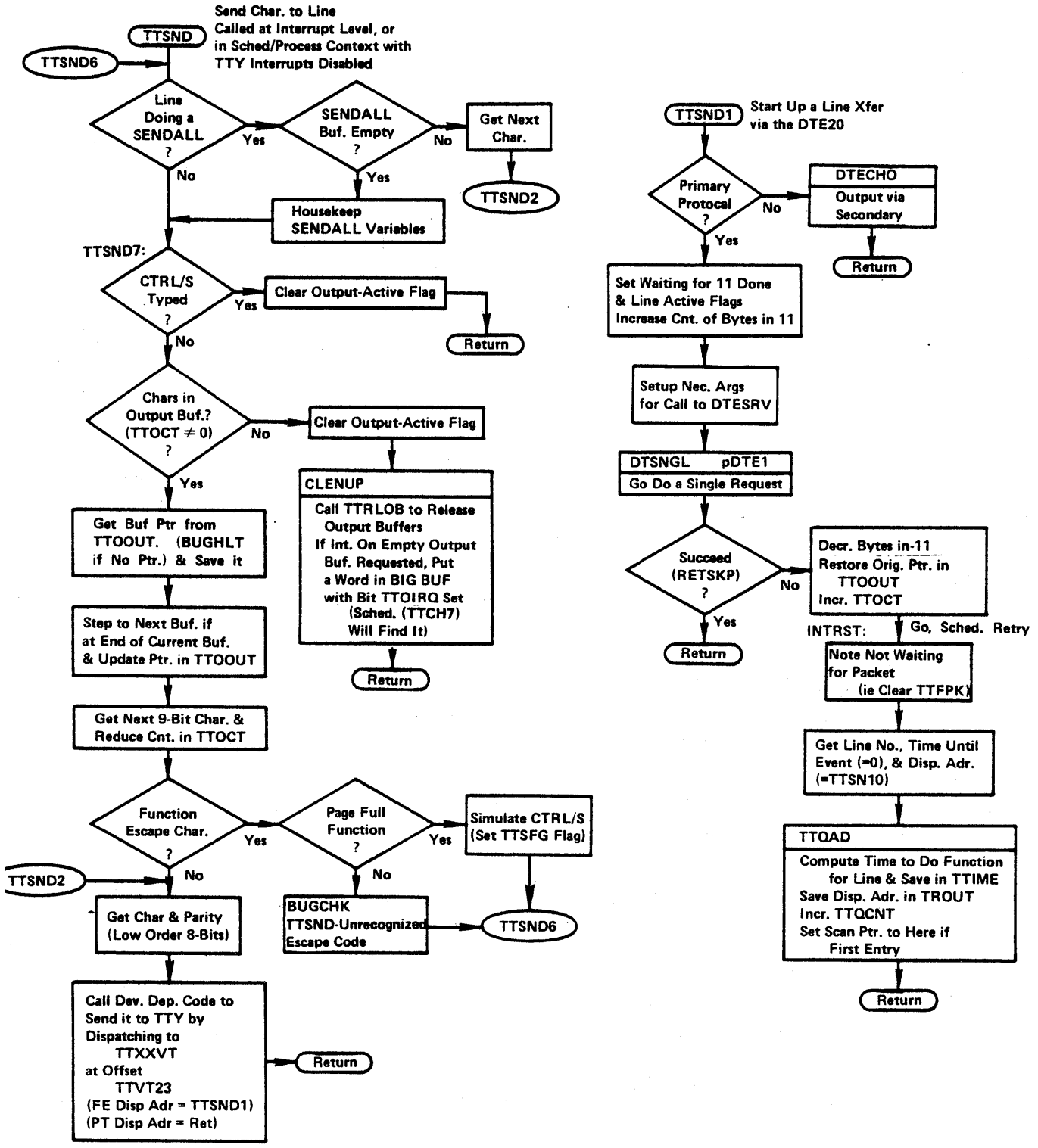
STT5


2nd Level: Called from TCO/TCOE
Handle Dev. Idiosyncrasies
and Character Accounting
TCOY
Save Registers
Rubout Char. ?
Yes
No
TCOUT pSTT7
Send Char. Out Directly
Immediate Echo Char. (IMECHF ≠ 0) ?
Yes
No
Linked Char? (LINKF ≠ 0) or Full Duplex TTY ?
Yes
No
TTOHA
Account Spacing Performed by Half Duplex TTY
CTRL Char. & Mode 10 (ie Send Code) ?
No, No Echo
TCOY3
Char. in CTRL Groups (Code < 40) ?
Yes
No
TTCO1:
Yes, Echo CTRL Func.
Use Char. Code to Index in CHITAB
to Pick up Disp. Adr.:
TCOUT pSTT7
or
Format Routine (ie for CR, FF etc.)
(Later Calls TCOUT)
TTY Has Lower Case ?
Yes
No
Upper Case Letter ?
No
If Lower Case Char. Convert to Upper Case
Yes
Indicate Bit, TT%UOC, On ?
Yes
TCOP
Output ↑Char. & Account Space
No
TTCO6:
TCOP
Output Char. & Account Space
TCOY3
Restore Registers
Return
TCOP
Step to Next Position on Line
Reached End of Line ?
Yes
Call Routines to Output CR & LF & Reset Line Position
No
TCOUT pSTT7
Output Char. (3rd Level Output)
Return

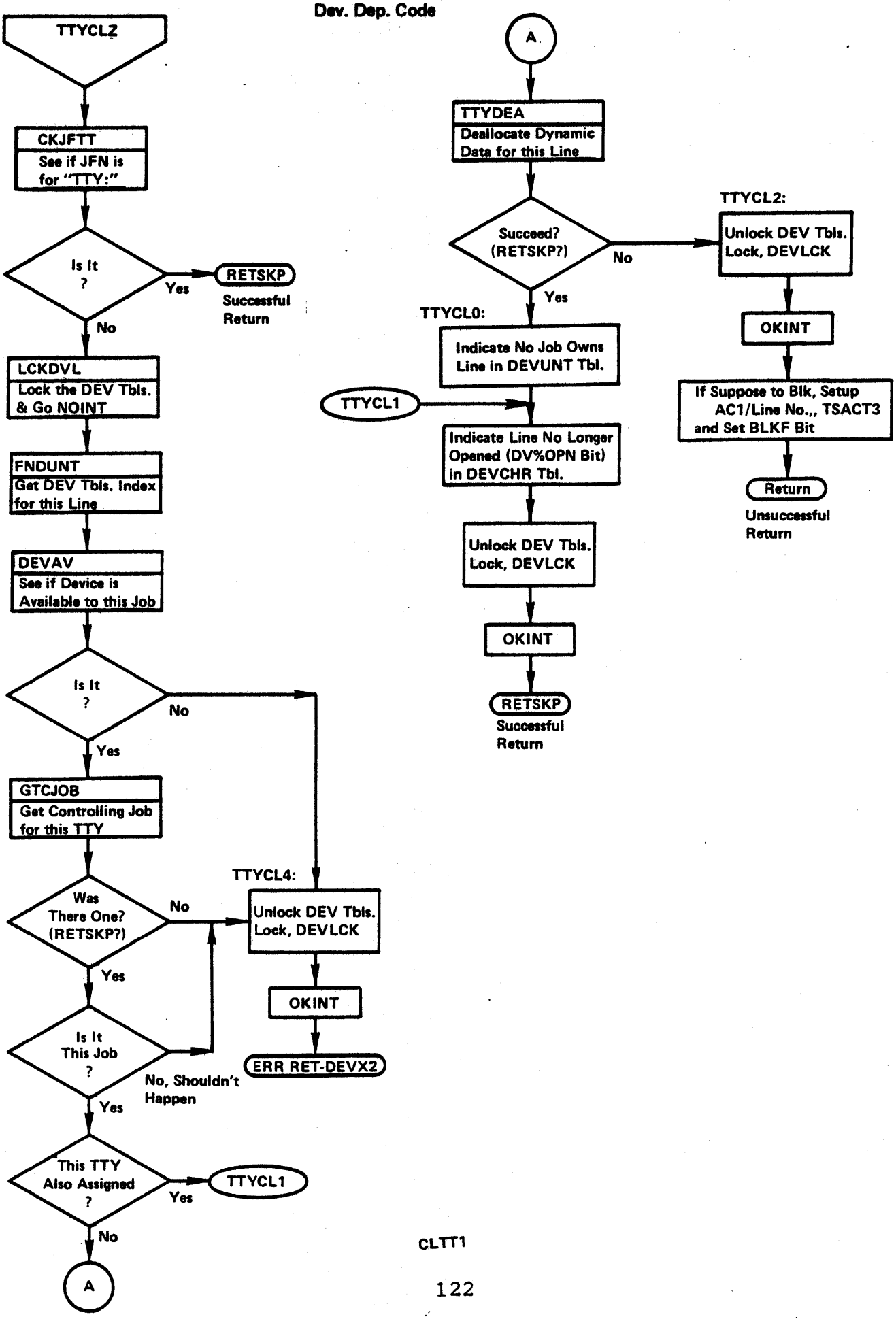
STT6


TCOUT
3rd Level: Called From TCOY/TCOB
Do Buffering & Output 1 Char.
Parity Bit Needed for Char. ?
Yes
Do Parity Logic By Executing Instr. in Tbl. TTXVT at Offset TTVT12
Entered From TCOB
No
TCOU1
Any Links ?
Yes
TTLNK3
Scan Link Word, Sending Chars. to Lines Specified
No
TCOU3:
CTRL/O Typed or CTY Line ?
Yes
Return
No
TCOU6
Will Char. Fit in Output Buffer
No
In the Sched. or NOSKED ?
No
Setup Sched. Test Word Line No.,,TCOTST & MDISMS
Yes
TCOU5
Yes
Indicate Failure to Sched. (-1 to TCOEER)
Line a PTY & Not Opened
Yes
Return
No
NOSKD1 & Turn Off DLS Chan
Buffers Exist (TTOIN ≠ 0) ?
No
TTGBF
Try to Assign Needed No. of Buffers
Fail ?
Yes
In the Sched. or NOSKED ?
Yes
No
No
Yes
Step to Next Buffer if at End of Current Buf.
Store Byte Ptr. (to New Buf) into TTOUT
Setup Sched. Test Word No. Needed,, TTBUFW & MDISMS
TCOU5
Store Char. in Output Buf. Updated Ptr. in TTOIN & Updated Cnt in TTOCT
Turn on DLS Chan. & OKSKD1
STRTOU
Start Output to Line if Needed
Dispatch to Table TTXVT at Offset TTVT13
(Disp. Adr = STRTO1 for TTFEVT)
Return
SCOTCO
Special Sched Entry Point When Entering from TTEMES
If Parity Bit Needed For Char., Do Parity Logic By Executing Instr. in Tbl. TTXXVT at Offset TTVT12
TCOU6
Go Store in Output Buffer
STRTO1
Start Output to Line if Nec.
Line Active ?
Yes
No
Return
TTSNDO
Send Char. Routine NOSKD1
Turn Off DLS Chan.
Call TTSND pSTT8
Turn On DLS Chan. OKSKD1
Callable from Non-PI Context

STT7


Send Char. to Line
Called at Interrupt Level, or
in Sched/Process Context with
TTY Interrupts Disabled
TTSND
TTSND6
Line Doing a SENDALL ?
Yes
SENDALL Buf. Empty ?
No
Get Next Char.
TTSND2
Yes
No
Housekeep SENDALL Variables
TTSND7:
CTRL/S Typed ?
Yes
Clear Output-Active Flag
Return
No
Chars in Output Buf.? (TTOCT ≠ 0) ?
No
Clear Output-Active Flag
CLENUP
Call TTRLOB to Release Output Buffers
If Int. On Empty Output Buf. Requested, Put a Word in BIG BUF with Bit TTOIRQ Set (Sched. (TTCH7) Will Find It)
Return
Yes
Get Buf Ptr from TTOOUT. (BUGHLT if No Ptr.) & Save it
Step to Next Buf. if at End of Current Buf. & Update Ptr. in TTOOUT
Get Next 9-Bit Char. & Reduce Cnt. in TTOCT
Function Escape Char. ?
Yes
Page Full Function ?
Yes
Simulate CTRL/S (Set TTSFG Flag)
TTSND2
No
No
Get Char & Parity (Low Order 8-Bits)
BUGCHK TTSND-Unrecognized Escape Code
TTSND6
Call Dev. Dep. Code to Send it to TTY by Dispatching to TTXXVT at Offset TTVT23 (FE Disp Adr = TTSND1) (PT Disp Adr = Ret)
Return
TTSND1
Start Up a Line Xfer via the DTE20
Primary Protocal ?
No
DTECHO
Output via Secondary
Return
Yes
Set Waiting for 11 Done & Line Active Flags Increase Cnt. of Bytes in 11
Setup Nec. Args for Call to DTESRV
DTSNGL pDTE1
Go Do a Single Request
Succeed (RETSKP) ?
No
Decr. Bytes in-11
Restore Orig. Ptr. in TTOOUT
Incr. TTOCT
INTRST:
Go, Sched. Retry
Note Not Waiting for Packet (ie Clear TTFPK)
Get Line No., Time Until Event (=0), & Disp. Adr. (=TTSN10)
TTQAD
Compute Time to Do Function for Line & Save in TTIME
Save Disp. Adr. in TROUT
Incr. TTQCNT
Set Scan Ptr. to Here if First Entry
Return
Yes
Return

STT8

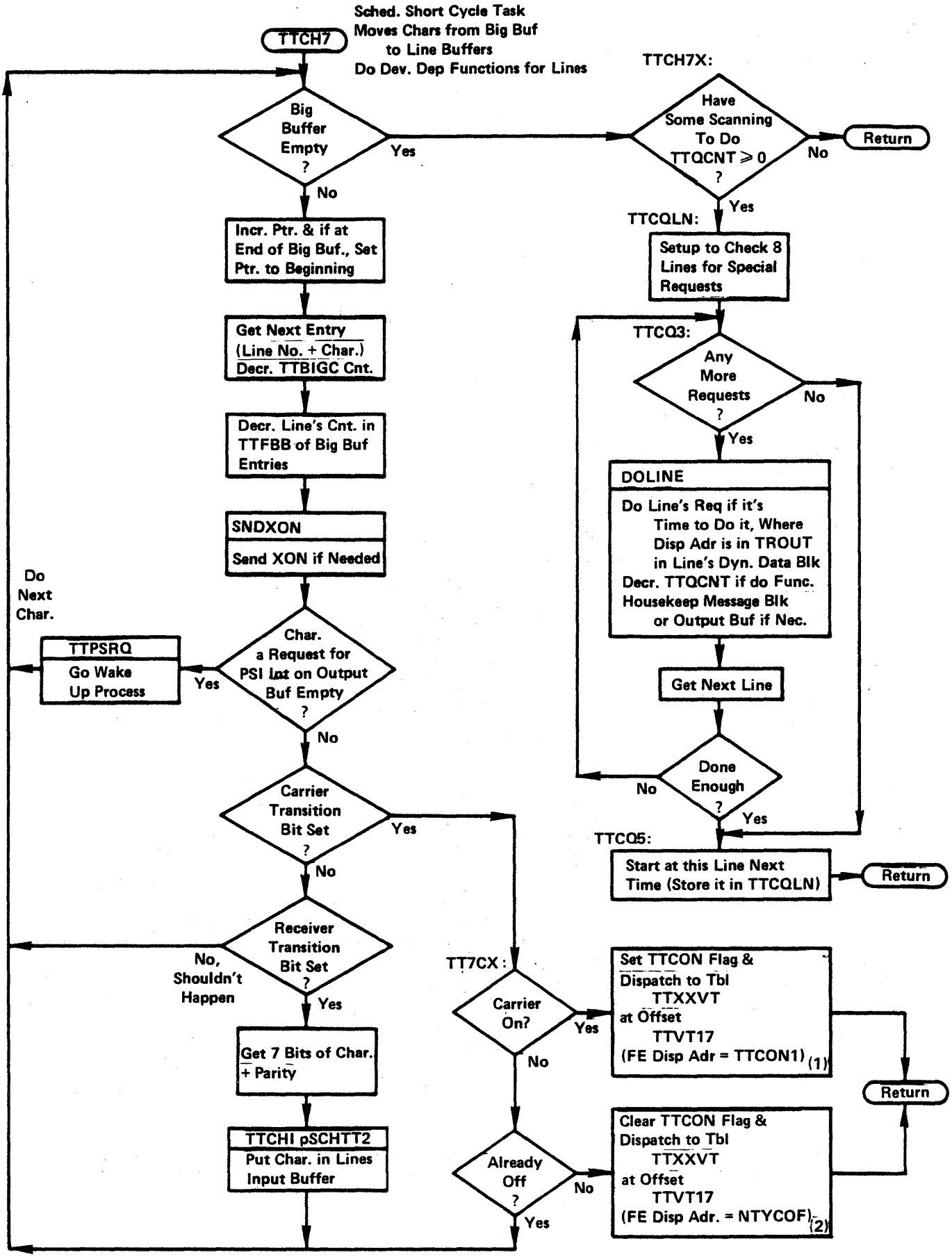

TTYCLZ
CKJFTT
See if JFN is for "TTY:"
Is It ?
Yes
RETSKP
Successful Return
No
LCKDVL
Lock the DEV Tbls. & Go NOINT
FNDUNT
Get DEV Tbls. Index for this Line
DEVAV
See if Device is Available to this Job
Is It ?
No
Yes
GTCJOB
Get Controlling Job for this TTY
TTYCL4:
Was There One? (RETSKP?)
No
Unlock DEV Tbls. Lock, DEVLCK
Yes
OKINT
Is It This Job ?
ERR RET-DEVX2
No, Shouldn't Happen
Yes
This TTY Also Assigned ?
TTYCL1
Yes
No
A
A
TTYDEA
Deallocate Dynamic Data for this Line
Succeed? (RETSKP?)
No
TTYCL2:
Unlock DEV Tbls. Lock, DEVLCK
OKINT
If Suppose to Blk, Setup AC1/Line No.,, TSACT3 and Set BLKF Bit
Return
Unsuccessful Return
Yes
TTYCL0:
Indicate No Job Owns Line in DEVUNT Tbl.
TTYCL1
Indicate Line No Longer Opened (DV%OPN Bit) in DEVCHR Tbl.
Unlock DEV Tbls. Lock, DEVLCK
OKINT
RETSKP
Successful Return

SCHEDULER TTY INPUT ANALYSIS & STORAGE

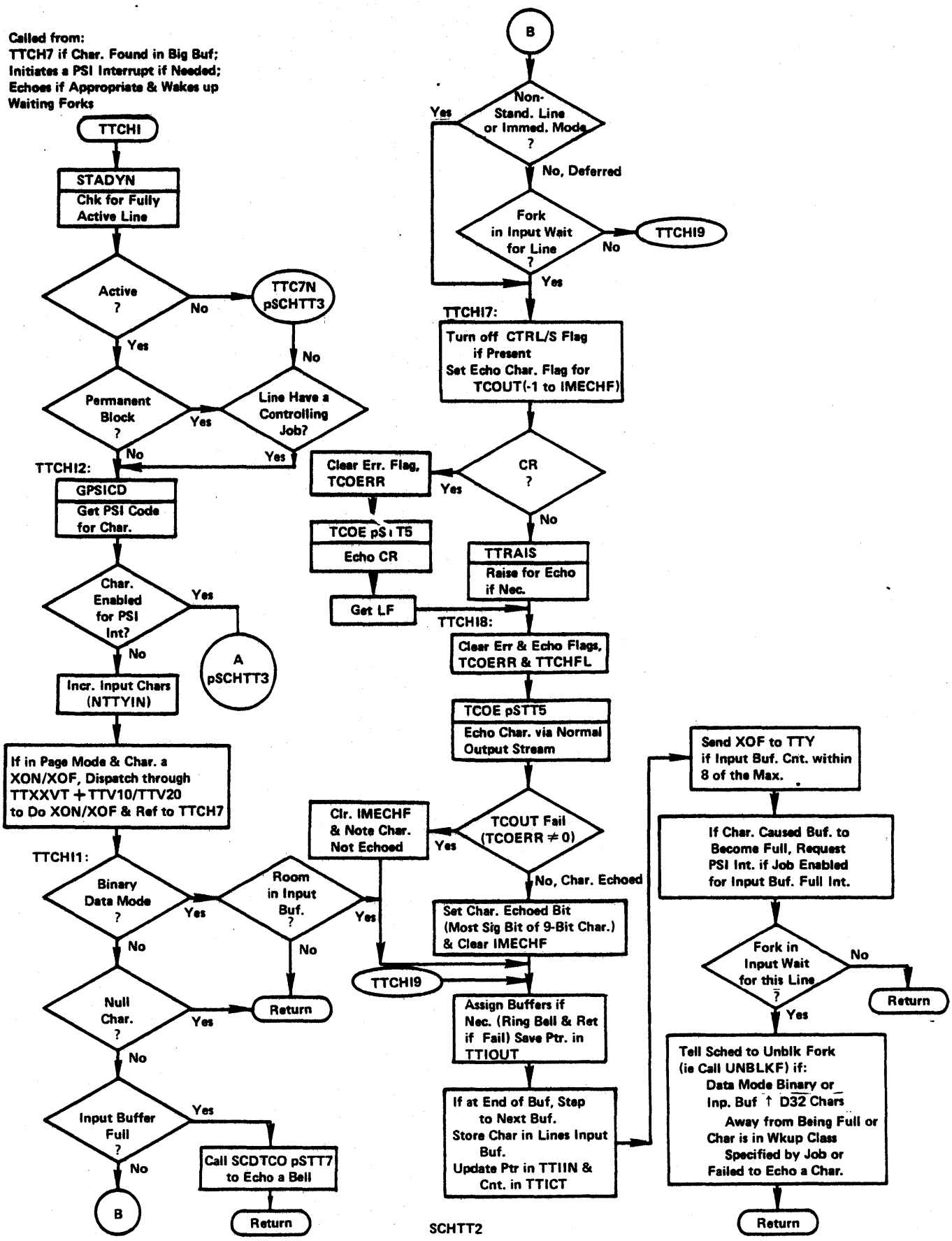
TTCH7 - Moves Characters from the Big Buffer to Line Buffers — SCHTT1

TTCHI - Initiates a PSI Interrupt if Needed, Echoes if Appropriate & Wakes Up Waiting Forks — SCHTT2

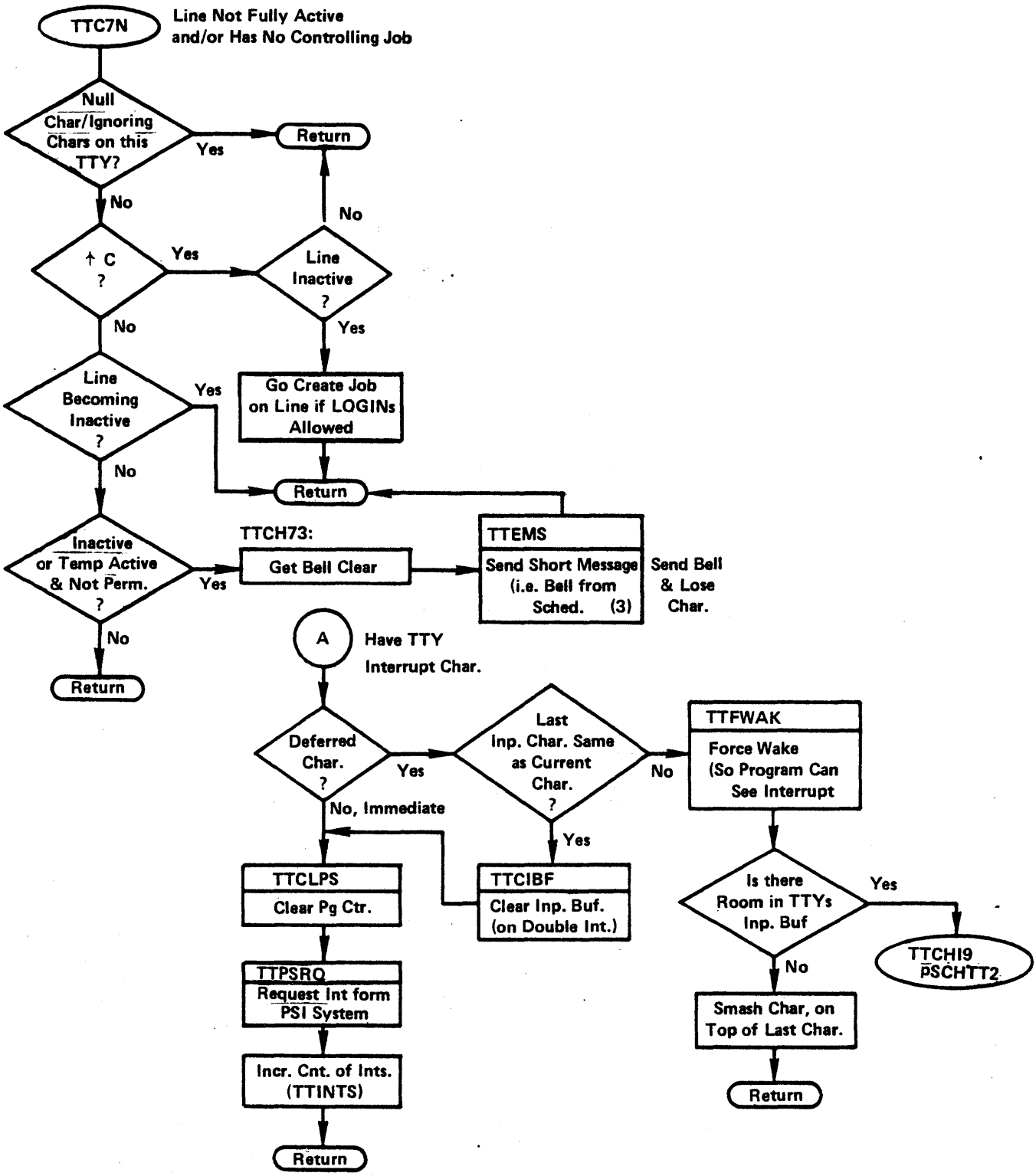
SCHEDULER TTY INPUT ANALYSIS & STORAGE



SCHTT1


Called from:
TTCH7 if Char. Found in Big Buf;
Initiates a PSI Interrupt if Needed;
Echoes if Appropriate & Wakes up
Waiting Forks
TTCHI
STADYN
Chk for Fully Active Line
Active ?
No
TTC7N pSCHTT3
Yes
Permanent Block ?
Yes
Line Have a Controlling Job?
No
Yes
No
TTCHI2:
GPSICD
Get PSI Code for Char.
Char. Enabled for PSI Int?
Yes
A pSCHTT3
No
Incr. Input Chars (NTTYIN)
If in Page Mode & Char. a XON/XOF, Dispatch through TTXXVT + TTV10/TTV20 to Do XON/XOF & Ref to TTCH7
TTCHI1:
Binary Data Mode ?
Yes
Room in Input Buf. ?
Yes
No
No
Null Char. ?
Yes
Return
No
Input Buffer Full ?
Yes
Call SCDTCO pSTT7 to Echo a Bell
No
Return
B
B
Non-Stand. Line or Immed. Mode ?
Yes
No, Deferred
Fork in Input Wait for Line ?
No
TTCHI9
Yes
TTCHI7:
Turn off CTRL/S Flag if Present
Set Echo Char. Flag for TCOUT(-1 to IMECHF)
CR ?
Yes
Clear Err. Flag, TCOERR
No
TCOE pSTT5
Echo CR
TTRAIS
Raise for Echo if Nec.
Get LF
TTCHI8:
Clear Err & Echo Flags, TCOERR & TTCHFL
TCOE pSTT5
Echo Char. via Normal Output Stream
TCOUT Fail (TCOERR ≠ 0)
Yes
Clr. IMECHF & Note Char. Not Echoed
TTCHI9
No, Char. Echoed
Set Char. Echoed Bit (Most Sig Bit of 9-Bit Char.) & Clear IMECHF
Assign Buffers if Nec. (Ring Bell & Ret if Fail) Save Ptr. in TTIOUT
If at End of Buf, Step to Next Buf.
Store Char in Lines Input Buf.
Update Ptr in TTIIN & Cnt. in TTICT
Send XOF to TTY if Input Buf. Cnt. within 8 of the Max.
If Char. Caused Buf. to Become Full, Request PSI Int. if Job Enabled for Input Buf. Full Int.
Fork in Input Wait for this Line ?
No
Return
Yes
Tell Sched to Unblk Fork (ie Call UNBLKF) if:
Data Mode Binary or
Inp. Buf ↑ D32 Chars Away from Being Full or
Char is in Wkup Class Specified by Job or
Failed to Echo a Char.
Return

SCHTT2


TTC7N
Line Not Fully Active and/or Has No Controlling Job
Null Char/Ignoring Chars on this TTY?
Yes
Return
No
↑ C ?
Yes
Line Inactive ?
No
Yes
No
Line Becoming Inactive ?
Yes
Go Create Job on Line if LOGINs Allowed
No
Return
Inactive or Temp Active & Not Perm. ?
Yes
TTCH73:
Get Bell Clear
TTEMS
Send Short Message (i.e. Bell from Sched. (3)
Send Bell & Lose Char.
No
Return
A
Have TTY Interrupt Char.
Deferred Char. ?
Yes
Last Inp. Char. Same as Current Char. ?
No
TTFWAK
Force Wake (So Program Can See Interrupt
No, Immediate
Yes
TTCLPS
Clear Pg Ctr.
TTCIBF
Clear Inp. Buf. (on Double Int.)
Is there Room in TTYs Inp. Buf
Yes
TTCHI9
PSCHTT2
No
TTPSRQ
Request Int form PSI System
Smash Char, on Top of Last Char.
Incr. Cnt. of Ints. (TTINTS)
Return
Return

SCHTT 3

Scheduler TTY Input Comments

TTCH7

(1) The carrier-on routine for the FE device is TTCON1. If the line is in use or a job is being created, it just returns. Otherwise, it creates a job by the CTRL/C mechanism (i.e., putting a request in Scheduler's Request Queue, SCDRQB) before returning.

(2) The carrier-off routine for the FE device is NTYCOF. It flushes outputs and issues an interrupt via the PSI system if process has enabled for carrier-off interrupt. It then issues a monitor-internal interrupt via routine, PSIR4, which causes the top fork to go to JOBCOF in MEXEC to cause the job to be detached.

TTC7N

(3) TTEMES is called at Scheduler Level to send a short message to a line. If the line is active, it appends characters to the line's output buffer. If the line is not active, it creates a message-length dynamic block for the line and puts the characters into this block.

TTEMES calls SCDTCO (pSTT7) to output each character via TCOUT to the buffer or message block.

REQUESTING DTE OUTPUT & DTE INTERRUPT HANDLING FLOWCHARTS

(DTE PROTOCOL HANDLER)

DTSNGL - Queue Up a Single Character Output Request DTE1

DTEQ - DTE Request Queues for RSX2ØF Protocol DTE1

DTESKD - Start a To -11 Operation DTE2

SKVER1 - Process RSX2ØF Packet DTE2

INTDTE - DTE Interrupt Handler DTE3

DN11X - To -11 Done DTE4

DEQDTE - Dequeue Completed Request, Post it, and Schedule Next One DTE4

TTYINT - Complete a TTY Output Request DTE5

DNSNGL - Post Single Character Done DTE4

DINGME - 1Ø Received a Doorbell Interrupt DTE6

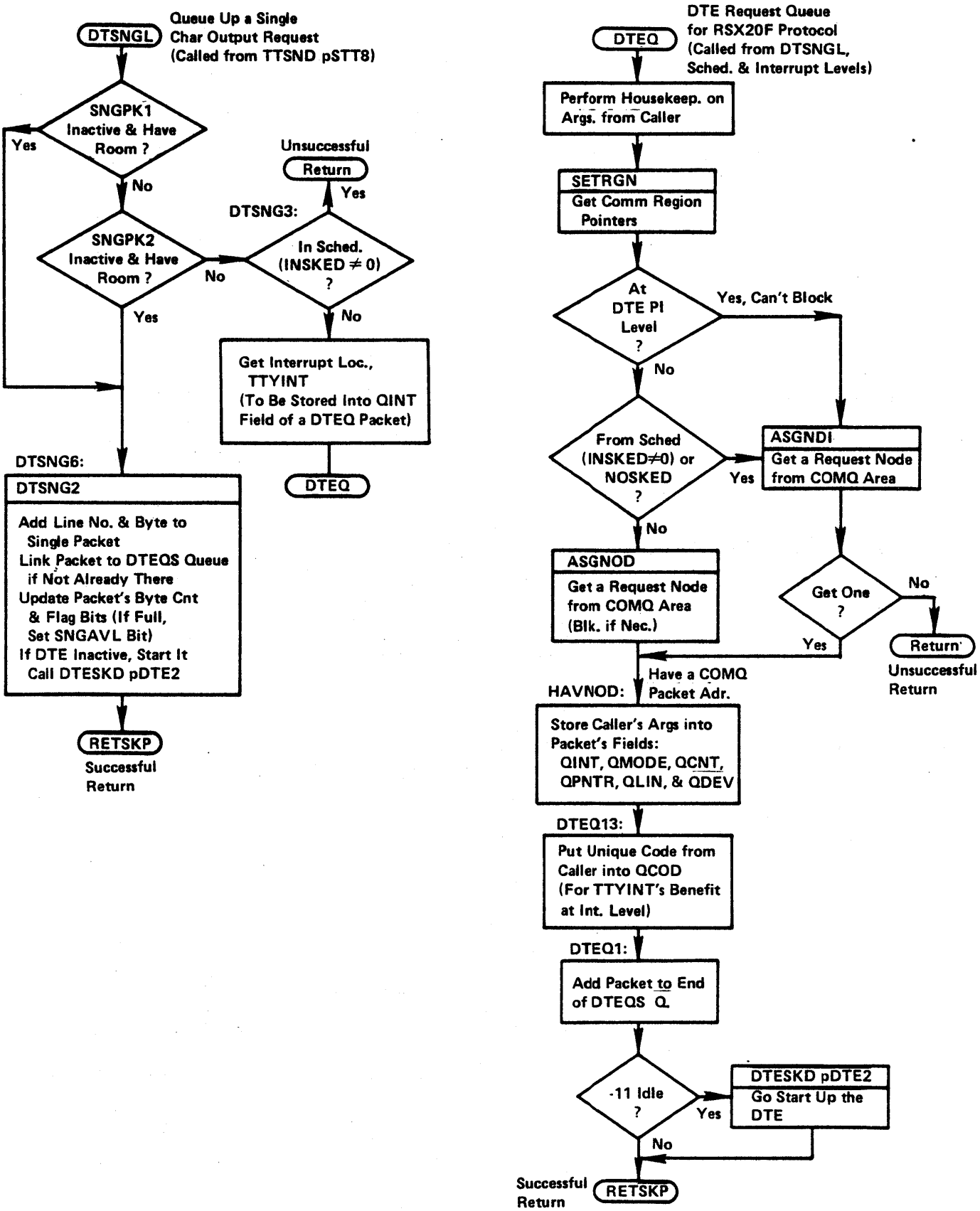
DOFRGM - Start a To -1Ø Transfer DTE7

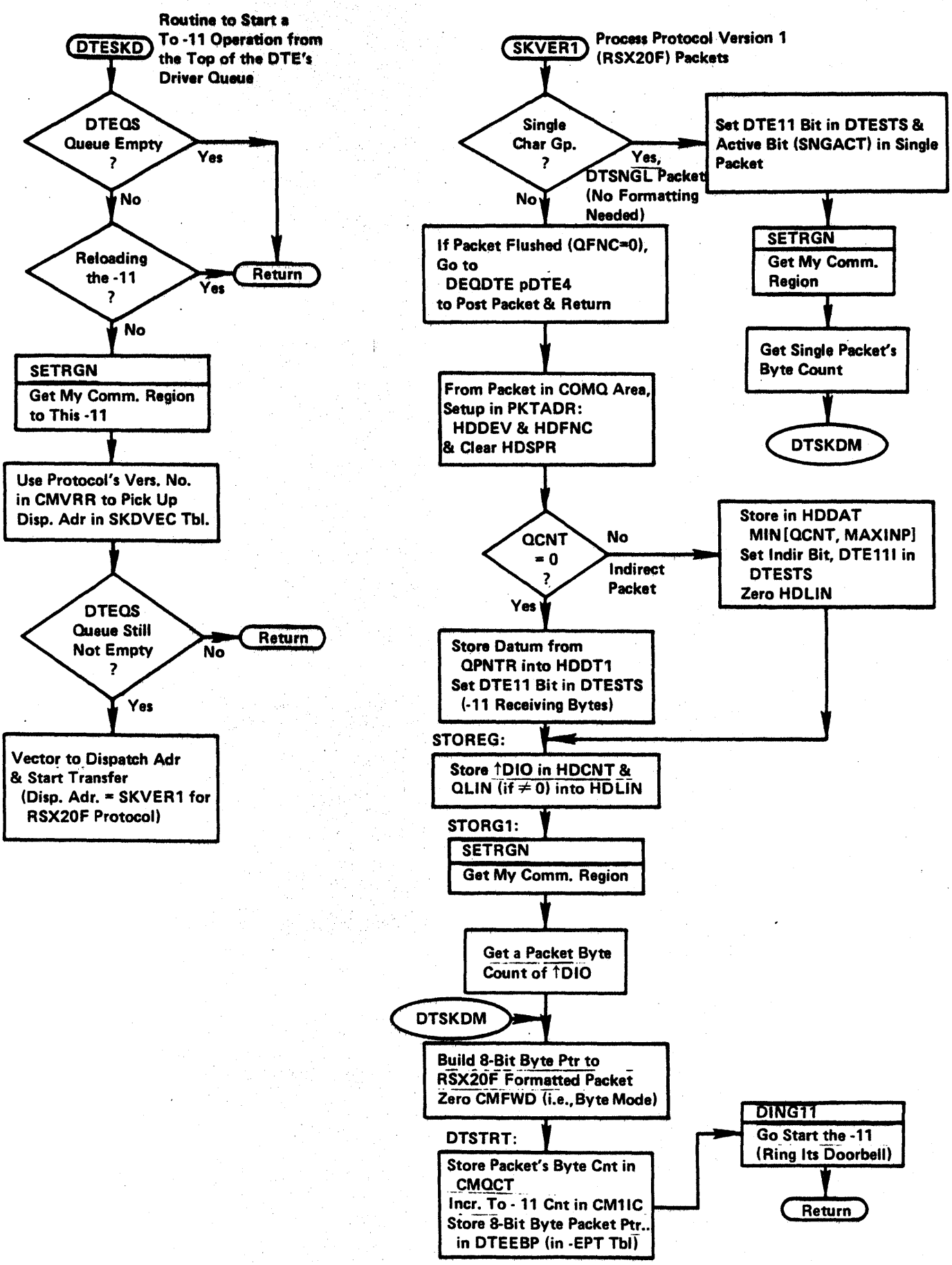
DN1ORS - To -1Ø Done DTE8

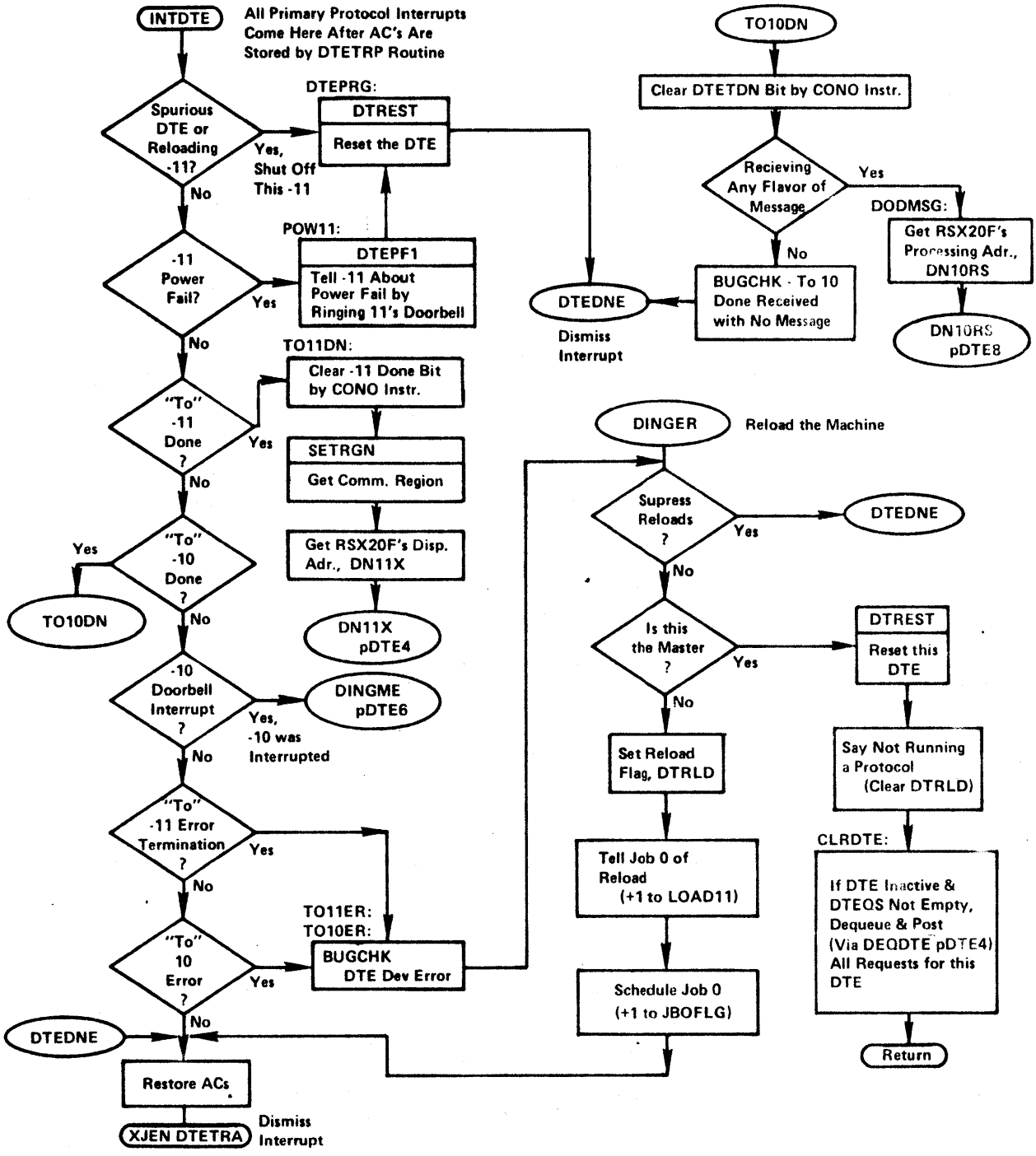
TAKLC2 - Process To -1Ø Done for RSX2ØF Protocol DTE9

BIGST2 - Store Character into the Big Buffer DTE1Ø

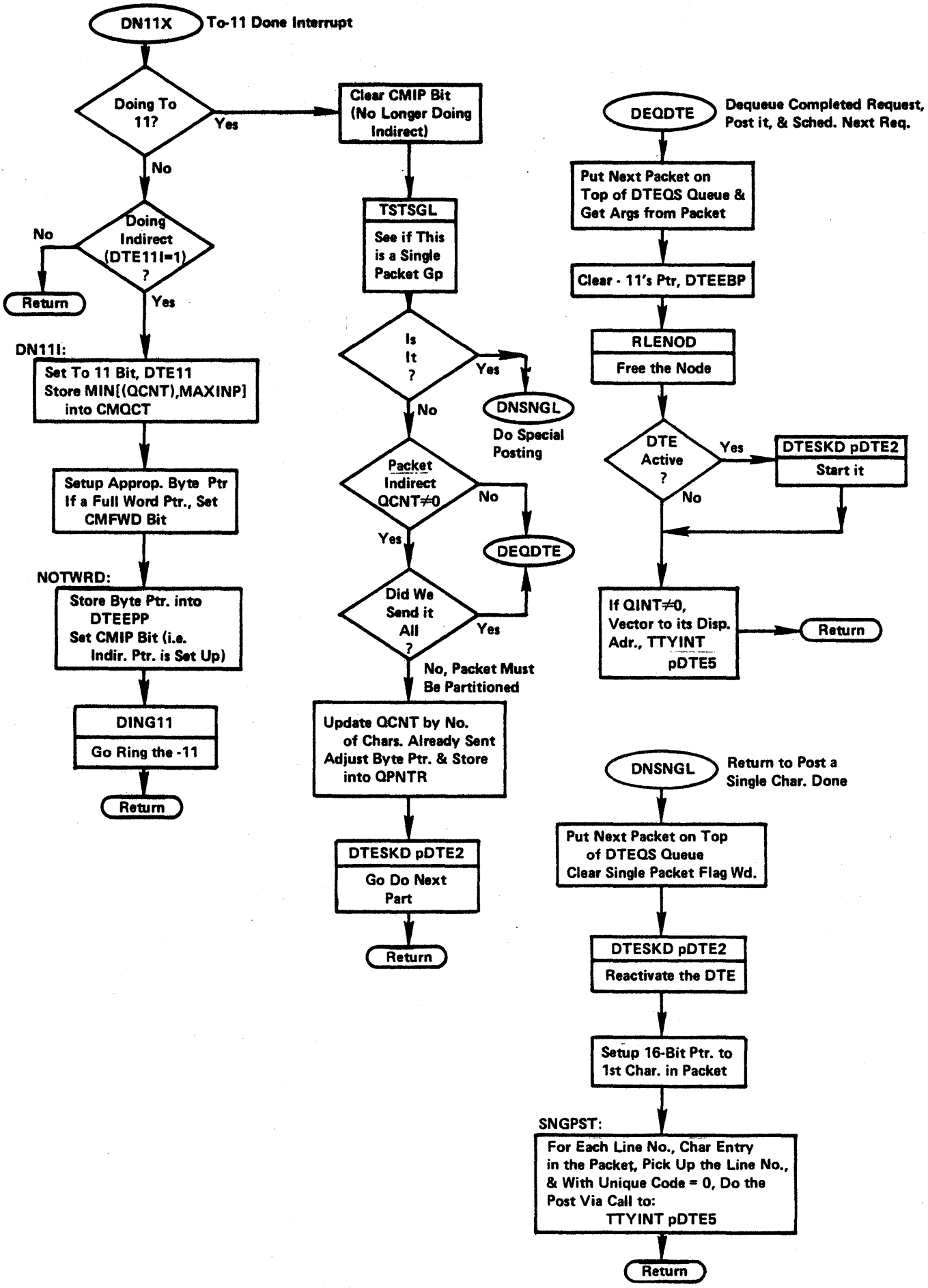
REQUESTING DTE OUTPUT
Called From Interrupt, JSYS & Scheduler Levels

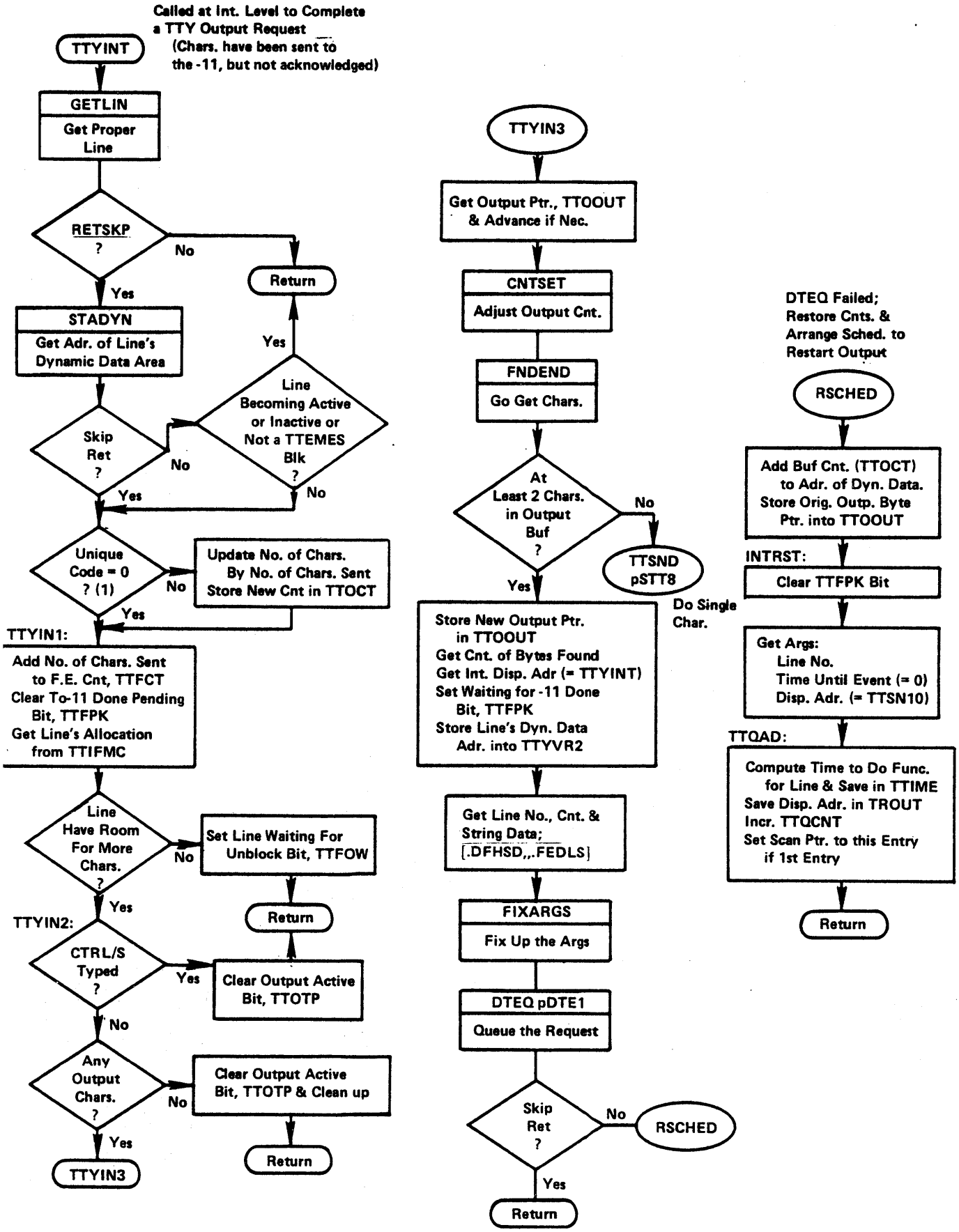



DTESKD
Routine to Start a To -11 Operation from the Top of the DTE's Driver Queue
DTEQS Queue Empty ?
Yes
No
Reloading the -11 ?
Yes
Return
No
SETRGN
Get My Comm. Region to This -11
Use Protocol's Vers. No. in CMVRR to Pick Up Disp. Adr in SKDVEC Tbl.
DTEQS Queue Still Not Empty ?
No
Return
Yes
Vector to Dispatch Adr & Start Transfer (Disp. Adr. = SKVER1 for RSX20F Protocol)
SKVER1
Process Protocol Version 1 (RSX20F) Packets
Single Char Gp. ?
Yes, DTSNGL Packet (No Formatting Needed)
Set DTE11 Bit in DTESTS & Active Bit (SNGACT) in Single Packet
No
If Packet Flushed (QFNC=0), Go to DEQDTE pDTE4 to Post Packet & Return
SETRGN
Get My Comm. Region
Get Single Packet's Byte Count
DTSKDM
From Packet in COMQ Area, Setup in PKTADR: HDDEV & HDFNC & Clear HDSPR
QCNT = 0 ?
No
Indirect Packet
Store in HDDAT MIN[QCNT, MAXINP] Set Indir Bit, DTE111 in DTESTS Zero HDLIN
Yes
Store Datum from QPNTR into HDDT1 Set DTE11 Bit in DTESTS (-11 Receiving Bytes)
STOREG:
Store ↑DIO in HDCNT & QLIN (if ≠ 0) into HDLIN
STORG1:
SETRGN
Get My Comm. Region
Get a Packet Byte Count of ↑DIO
DTSKDM
Build 8-Bit Byte Ptr to RSX20F Formatted Packet Zero CMFWD (i.e., Byte Mode)
DTSTRT:
Store Packet's Byte Cnt in CMQCT Incr. To - 11 Cnt in CM1IC Store 8-Bit Byte Packet Ptr.. in DTEEBP (in -EPT Tbl)
DING11
Go Start the -11 (Ring Its Doorbell)
Return

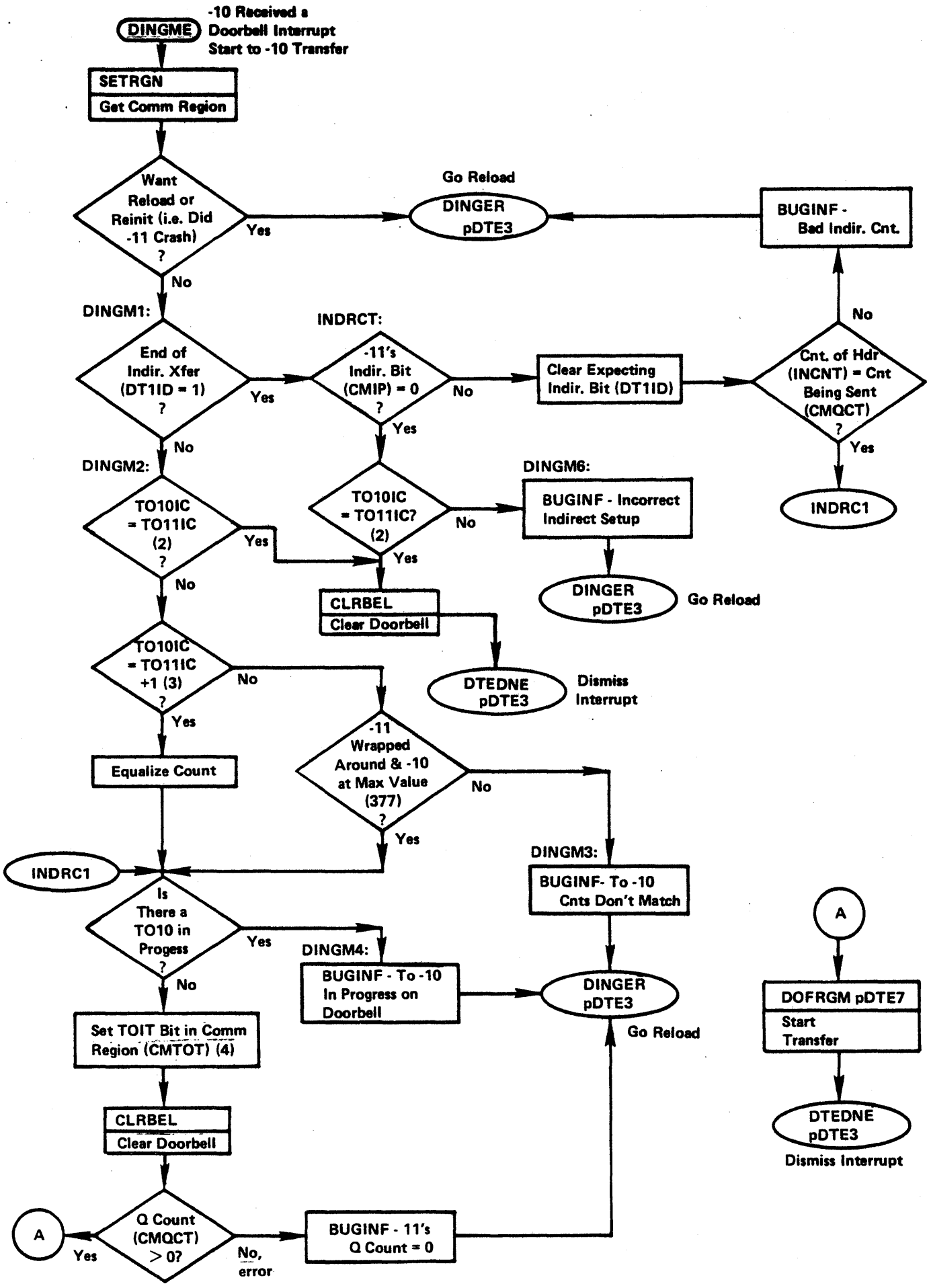

INTDTE
All Primary Protocol Interrupts Come Here After AC's Are Stored by DTETRP Routine
DTEPRG:
DTREST
Reset the DTE
Spurious DTE or Reloading -11?
Yes, Shut Off This -11
No
POW11:
DTEPF1
Tell -11 About Power Fail by Ringing 11's Doorbell
-11 Power Fail?
Yes
No
TO11DN:
Clear -11 Done Bit by CONO Instr.
"To" -11 Done ?
Yes
No
SETRGN
Get Comm. Region
Get RSX20F's Disp. Adr., DN11X
DN11X pDTE4
Yes
"To" -10 Done ?
TO10DN
No
-10 Doorbell Interrupt ?
Yes, -10 was Interrupted
DINGME pDTE6
No
"To" -11 Error Termination ?
Yes
No
TO11ER:
TO10ER:
"To" 10 Error ?
Yes
BUGCHK DTE Dev Error
No
DTEDNE
Restore ACs
XJEN DTETRA
Dismiss Interrupt
TO10DN
Clear DTETDN Bit by CONO Instr.
Recieving Any Flavor of Message
Yes
DODMSG:
Get RSX20F's Processing Adr., DN10RS
DN10RS pDTE8
No
BUGCHK - To 10 Done Received with No Message
DTEDNE
Dismiss Interrupt
DINGER
Reload the Machine
Supress Reloads ?
Yes
DTEDNE
No
Is this the Master ?
Yes
DTREST
Reset this DTE
No
Set Reload Flag, DTRLD
Tell Job 0 of Reload (+1 to LOAD11)
Schedule Job 0 (+1 to JBOFLG)
Say Not Running a Protocol (Clear DTRLD)
CLRDTE:
If DTE Inactive & DTEOS Not Empty, Dequeue & Post (Via DEQDTE pDTE4) All Requests for this DTE
Return

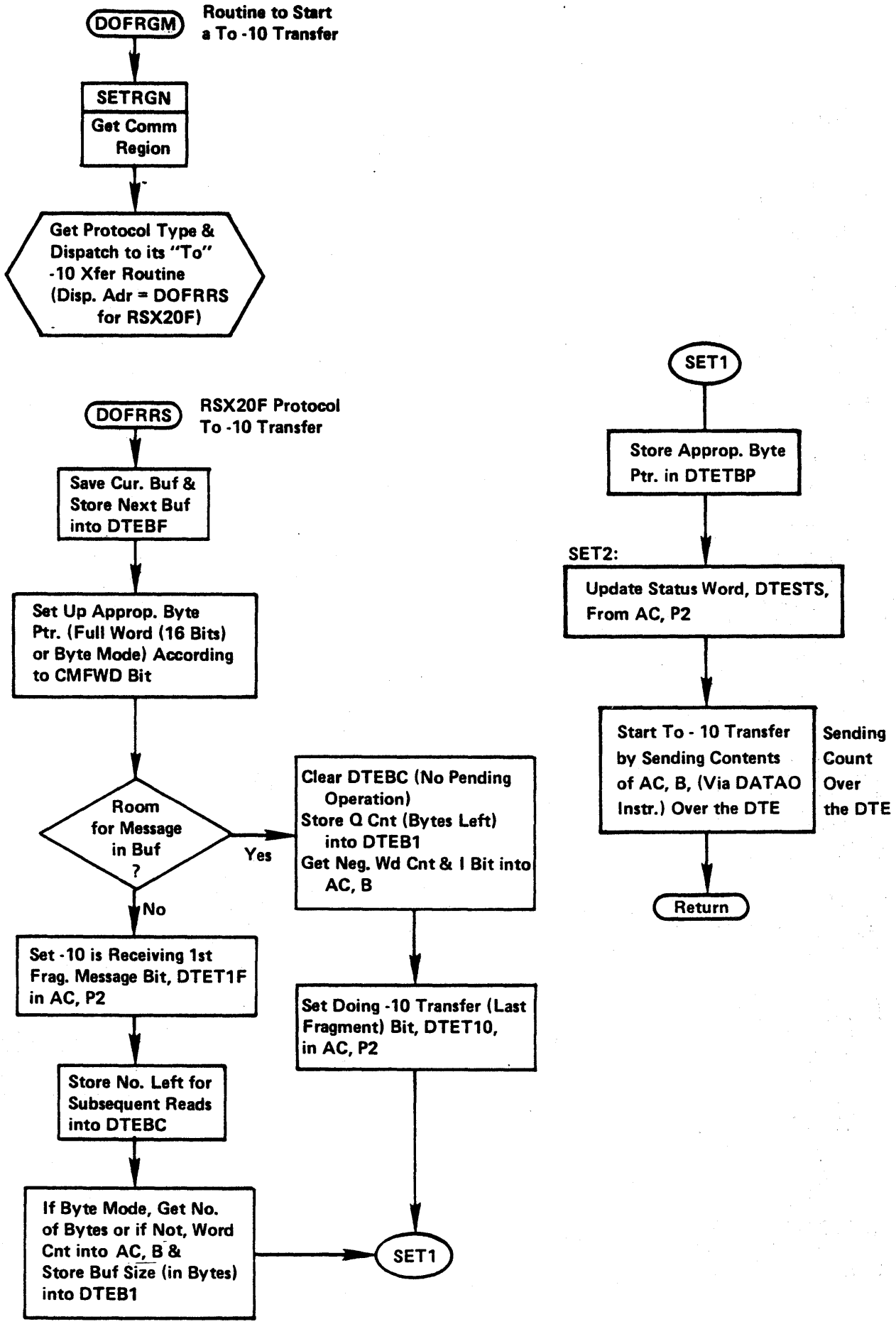
DTE3

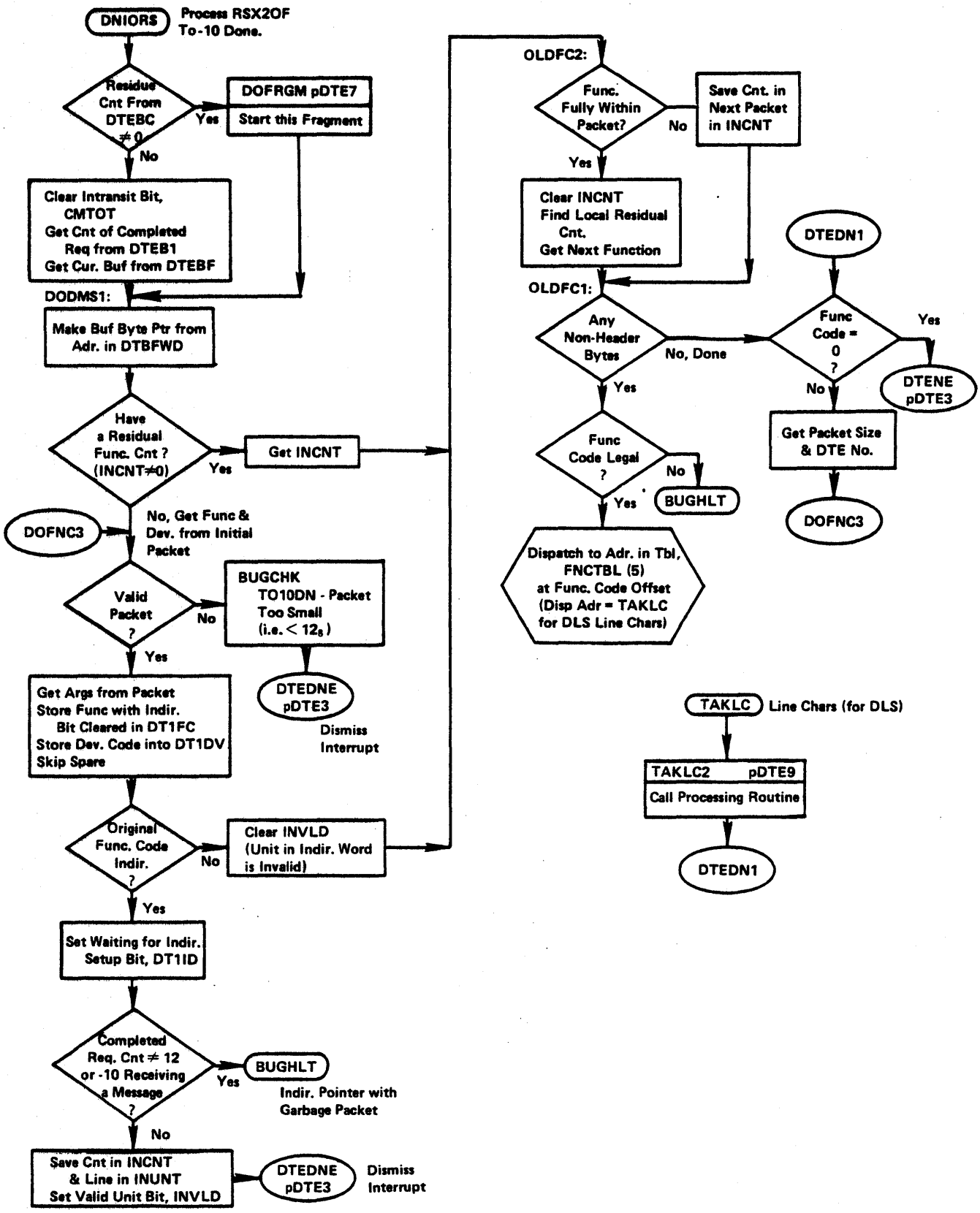

DN11X
To-11 Done Interrupt
Doing To 11?
Yes
No
Clear CMIP Bit (No Longer Doing Indirect)
Doing Indirect (DTE11I=1) ?
No
Return
Yes
DN11I:
Set To 11 Bit, DTE11
Store MIN[(QCNT),MAXINP] into CMQCT
Setup Approp. Byte Ptr If a Full Word Ptr., Set CMFWD Bit
NOTWRD:
Store Byte Ptr. into DTEEPP
Set CMIP Bit (i.e. Indir. Ptr. is Set Up)
DING11
Go Ring the -11
Return
TSTSGL
See if This is a Single Packet Gp
Is It ?
Yes
DNSNGL
Do Special Posting
No
Packet Indirect QCNT≠0
No
DEQDTE
Yes
Did We Send it All ?
Yes
No, Packet Must Be Partitioned
Update QCNT by No. of Chars. Already Sent Adjust Byte Ptr. & Store into QPNTR
DTESKD pDTE2
Go Do Next Part
Return
DEQDTE
Dequeue Completed Request, Post it, & Sched. Next Req.
Put Next Packet on Top of DTEQS Queue & Get Args from Packet
Clear - 11's Ptr, DTEEBP
RLENOD
Free the Node
DTE Active ?
Yes
DTESKD pDTE2
Start it
No
If QINT≠0, Vector to its Disp. Adr., TTYINT pDTE5
Return
DNSNGL
Return to Post a Single Char. Done
Put Next Packet on Top of DTEQS Queue Clear Single Packet Flag Wd.
DTESKD pDTE2
Reactivate the DTE
Setup 16-Bit Ptr. to 1st Char. in Packet
SNGPST:
For Each Line No., Char Entry in the Packet, Pick Up the Line No., & With Unique Code = 0, Do the Post Via Call to: TTYINT pDTE5
Return


Called at Int. Level to Complete a TTY Output Request (Chars. have been sent to the -11, but not acknowledged)
TTYINT
GETLIN
Get Proper Line
RETSKP ?
No
Return
Yes
STADYN
Get Adr. of Line's Dynamic Data Area
Yes
Line Becoming Active or Inactive or Not a TTEMES Blk ?
Skip Ret ?
No
No
Yes
Unique Code = 0 ? (1)
No
Update No. of Chars. By No. of Chars. Sent Store New Cnt in TTOCT
Yes
TTYIN1:
Add No. of Chars. Sent to F.E. Cnt, TTFCT
Clear To-11 Done Pending Bit, TTFPK
Get Line's Allocation from TTIFMC
Line Have Room For More Chars. ?
No
Set Line Waiting For Unblock Bit, TTFOW
Yes
Return
TTYIN2:
CTRL/S Typed ?
Yes
Clear Output Active Bit, TTOTP
No
Any Output Chars. ?
No
Clear Output Active Bit, TTOTP & Clean up
Yes
TTYIN3
Return
TTYIN3
Get Output Ptr., TTOOUT & Advance if Nec.
CNTSET
Adjust Output Cnt.
FNDEND
Go Get Chars.
At Least 2 Chars. in Output Buf ?
No
TTSND pSTT8
Do Single Char.
Yes
Store New Output Ptr. in TTOOUT
Get Cnt. of Bytes Found
Get Int. Disp. Adr (= TTYINT)
Set Waiting for -11 Done Bit, TTFPK
Store Line's Dyn. Data Adr. into TTYVR2
Get Line No., Cnt. & String Data; [.DFHSD,,.FEDLS]
FIXARGS
Fix Up the Args
DTEQ pDTE1
Queue the Request
Skip Ret ?
No
RSCHED
Yes
Return
DTEQ Failed; Restore Cnts. & Arrange Sched. to Restart Output
RSCHED
Add Buf Cnt. (TTOCT) to Adr. of Dyn. Data.
Store Orig. Outp. Byte Ptr. into TTOOUT
INTRST:
Clear TTFPK Bit
Get Args:
Line No.
Time Until Event (= 0)
Disp. Adr. (= TTSN10)
TTQAD:
Compute Time to Do Func. for Line & Save in TTIME
Save Disp. Adr. in TROUT
Incr. TTQCNT
Set Scan Ptr. to this Entry if 1st Entry
Return

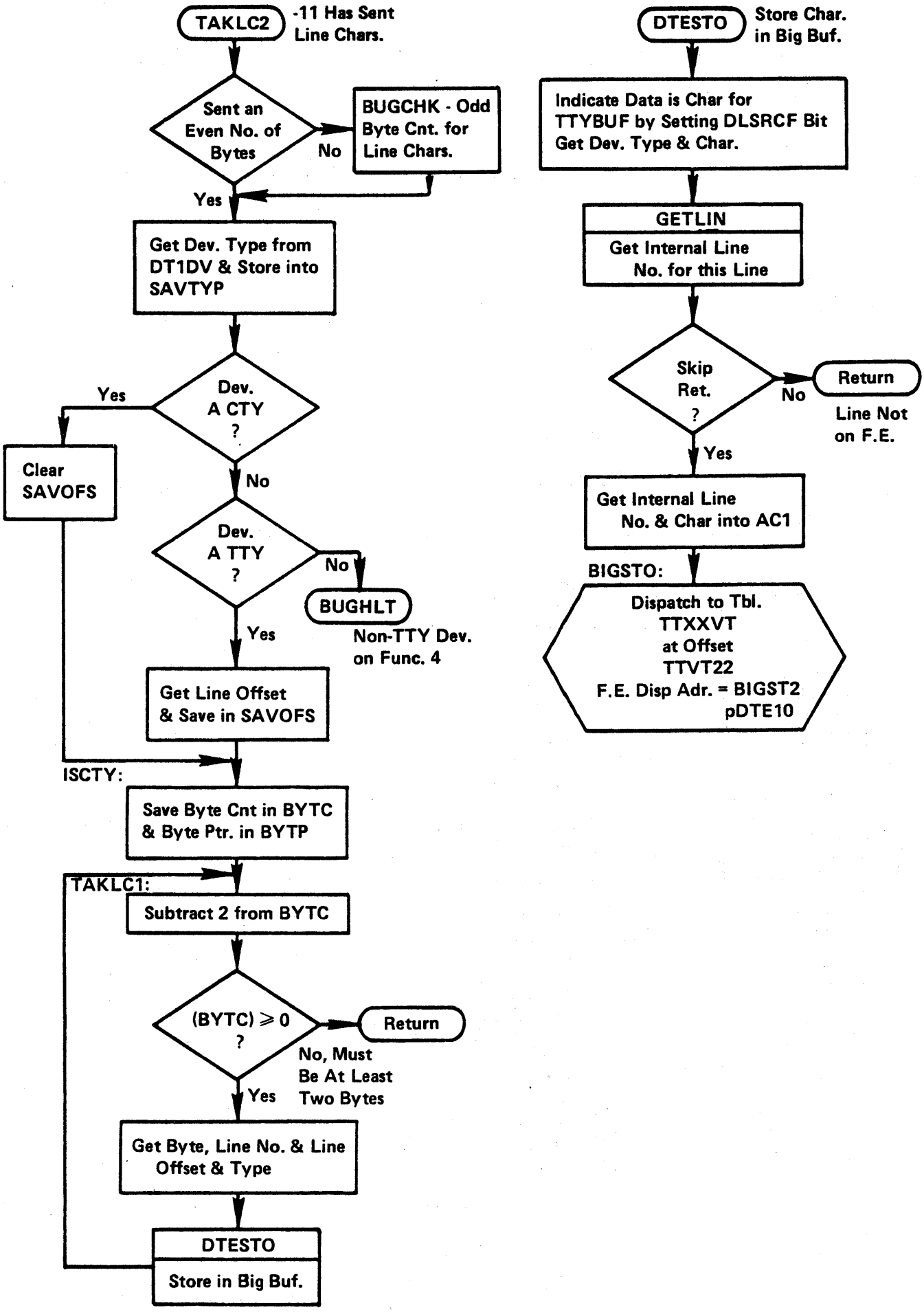
DTE5

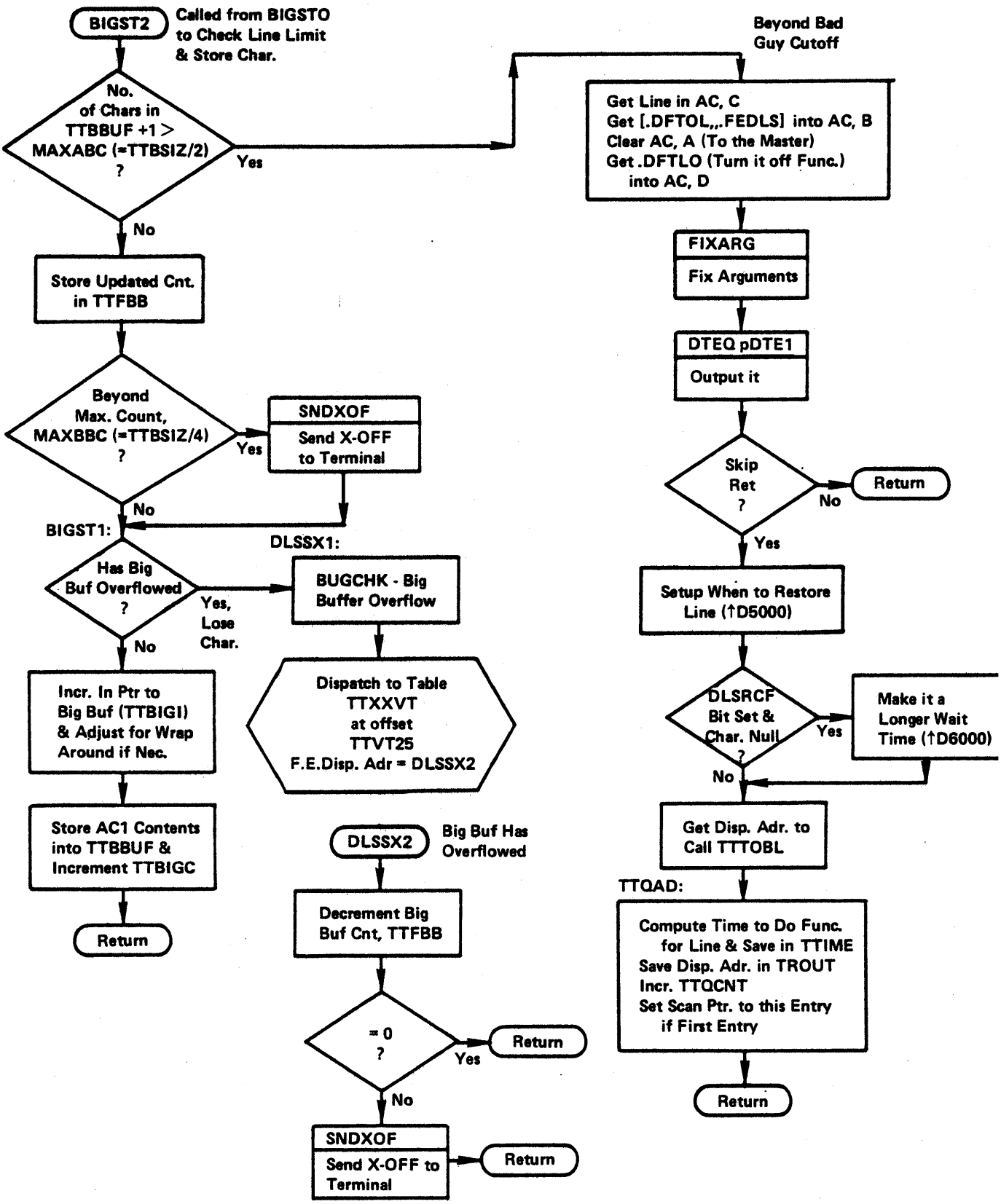

DINGME
-10 Received a Doorbell Interrupt Start to -10 Transfer
SETRGN
Get Comm Region
Want Reload or Reinit (i.e. Did -11 Crash) ?
Yes
Go Reload
DINGER pDTE3
BUGINF - Bad Indir. Cnt.
No
DINGM1:
End of Indir. Xfer (DT1ID = 1) ?
Yes
INDRCT:
-11's Indir. Bit (CMIP) = 0 ?
No
Clear Expecting Indir. Bit (DT1ID)
Cnt. of Hdr (INCNT) = Cnt Being Sent (CMQCT) ?
No
Yes
INDRC1
No
DINGM2:
TO10IC = TO11IC (2) ?
Yes
No
TO10IC = TO11IC? (2)
No
DINGM6:
BUGINF - Incorrect Indirect Setup
DINGER pDTE3
Go Reload
Yes
CLRBEL
Clear Doorbell
DTEDNE pDTE3
Dismiss Interrupt
TO10IC = TO11IC +1 (3) ?
No
Yes
Equalize Count
-11 Wrapped Around & -10 at Max Value (377) ?
No
Yes
INDRC1
DINGM3:
BUGINF- To -10 Cnts Don't Match
Is There a TO10 in Progess ?
Yes
DINGM4:
BUGINF - To -10 In Progress on Doorbell
DINGER pDTE3
Go Reload
No
Set TOIT Bit in Comm Region (CMTOT) (4)
CLRBEL
Clear Doorbell
A
Q Count (CMQCT) > 0?
Yes
No, error
BUGINF - 11's Q Count = 0
A
DOFRGM pDTE7
Start Transfer
DTEDNE pDTE3
Dismiss Interrupt


DOFRGM
Routine to Start a To -10 Transfer
SETRGN
Get Comm Region
Get Protocol Type & Dispatch to its "To" -10 Xfer Routine (Disp. Adr = DOFRRS for RSX20F)
DOFRRS
RSX20F Protocol To -10 Transfer
Save Cur. Buf & Store Next Buf into DTEBF
Set Up Approp. Byte Ptr. (Full Word (16 Bits) or Byte Mode) According to CMFWD Bit
Room for Message in Buf ?
Yes
No
Clear DTEBC (No Pending Operation)
Store Q Cnt (Bytes Left) into DTEB1
Get Neg. Wd Cnt & I Bit into AC, B
Set -10 is Receiving 1st Frag. Message Bit, DTET1F in AC, P2
Store No. Left for Subsequent Reads into DTEBC
Set Doing -10 Transfer (Last Fragment) Bit, DTET10, in AC, P2
If Byte Mode, Get No. of Bytes or if Not, Word Cnt into AC, B & Store Buf Size (in Bytes) into DTEB1
SET1
SET1
Store Approp. Byte Ptr. in DTETBP
SET2:
Update Status Word, DTESTS, From AC, P2
Start To - 10 Transfer by Sending Contents of AC, B, (Via DATAO Instr.) Over the DTE
Sending Count Over the DTE
Return


DNIORS
Process RSX2OF To-10 Done.
Residue Cnt From DTEBC ≠0
Yes
DOFRGM pDTE7
Start this Fragment
No
Clear Intransit Bit, CMTOT
Get Cnt of Completed Req from DTEB1
Get Cur. Buf from DTEBF
DODMS1:
Make Buf Byte Ptr from Adr. in DTBFWD
Have a Residual Func. Cnt ? (INCNT≠0)
Yes
Get INCNT
DOFNC3
No, Get Func & Dev. from Initial Packet
Valid Packet ?
No
BUGCHK
TO10DN - Packet Too Small (i.e. < 12₈)
DTEDNE pDTE3
Dismiss Interrupt
Yes
Get Args from Packet
Store Func with Indir. Bit Cleared in DT1FC
Store Dev. Code into DT1DV
Skip Spare
Original Func. Code Indir. ?
No
Clear INVLD (Unit in Indir. Word is Invalid)
Yes
Set Waiting for Indir. Setup Bit, DT1ID
Completed Req. Cnt ≠ 12 or -10 Receiving a Message ?
Yes
BUGHLT
Indir. Pointer with Garbage Packet
No
Save Cnt in INCNT & Line in INUNT
Set Valid Unit Bit, INVLD
DTEDNE pDTE3
Dismiss Interrupt
OLDFC2:
Func. Fully Within Packet?
No
Save Cnt. in Next Packet in INCNT
Yes
Clear INCNT
Find Local Residual Cnt.
Get Next Function
OLDFC1:
Any Non-Header Bytes
No, Done
Yes
Func Code = 0 ?
Yes
DTENE pDTE3
No
Get Packet Size & DTE No.
DOFNC3
DTEDN1
Func Code Legal ?
No
BUGHLT
Yes
Dispatch to Adr. in Tbl, FNCTBL (5) at Func. Code Offset (Disp Adr = TAKLC for DLS Line Chars)
TAKLC
Line Chars (for DLS)
TAKLC2 pDTE9
Call Processing Routine
DTEDN1

DTE8


TAKLC2
-11 Has Sent Line Chars.
Sent an Even No. of Bytes
No
BUGCHK - Odd Byte Cnt. for Line Chars.
Yes
Get Dev. Type from DT1DV & Store into SAVTYP
Dev. A CTY ?
Yes
Clear SAVOFS
No
Dev. A TTY ?
No
BUGHLT
Non-TTY Dev. on Func. 4
Yes
Get Line Offset & Save in SAVOFS
ISCTY:
Save Byte Cnt in BYTC & Byte Ptr. in BYTP
TAKLC1:
Subtract 2 from BYTC
(BYTC) ⩾ 0 ?
Return
No, Must Be At Least Two Bytes
Yes
Get Byte, Line No. & Line Offset & Type
DTESTO
Store in Big Buf.
DTESTO
Store Char. in Big Buf.
Indicate Data is Char for TTYBUF by Setting DLSRCF Bit Get Dev. Type & Char.
GETLIN
Get Internal Line No. for this Line
Skip Ret. ?
No
Return
Line Not on F.E.
Yes
Get Internal Line No. & Char into AC1
BIGSTO:
Dispatch to Tbl. TTXXVT at Offset TTVT22
F.E. Disp Adr. = BIGST2
pDTE10


BIGST2
Called from BIGSTO to Check Line Limit & Store Char.
No. of Chars in TTBBUF +1 > MAXABC (=TTBSIZ/2) ?
Yes
No
Store Updated Cnt. in TTFBB
Beyond Max. Count, MAXBBC (=TTBSIZ/4) ?
Yes
SNDXOF
Send X-OFF to Terminal
No
BIGST1:
Has Big Buf Overflowed ?
Yes, Lose Char.
No
DLSSX1:
BUGCHK - Big Buffer Overflow
Dispatch to Table TTXXVT at offset TTVT25 F.E.Disp. Adr = DLSSX2
Incr. In Ptr to Big Buf (TTBIGI) & Adjust for Wrap Around if Nec.
Store AC1 Contents into TTBBUF & Increment TTBIGC
Return
DLSSX2
Big Buf Has Overflowed
Decrement Big Buf Cnt, TTFBB
= 0 ?
Yes
Return
No
SNDXOF
Send X-OFF to Terminal
Return
Beyond Bad Guy Cutoff
Get Line in AC, C
Get [.DFTOL,,.FEDLS] into AC, B
Clear AC, A (To the Master)
Get .DFTLO (Turn it off Func.) into AC, D
FIXARG
Fix Arguments
DTEQ pDTE1
Output it
Skip Ret ?
No
Return
Yes
Setup When to Restore Line (↑D5000)
DLSRCF Bit Set & Char. Null ?
Yes
Make it a Longer Wait Time (↑D6000)
No
Get Disp. Adr. to Call TTTOBL
TTQAD:
Compute Time to Do Func. for Line & Save in TTIME
Save Disp. Adr. in TROUT
Incr. TTQCNT
Set Scan Ptr. to this Entry if First Entry
Return

DTE Interrupt Handling Comments

TTYINT

(1) The Unique Code argument of form (Ø, count) tells TTYINT the number of characters that have been sent to the -11 in some call to DTEQ that specified TTYINT as its return address.

Count = Ø implies this was a single character (DTSNGL was called) and buffer counts have already been updated.

Count ≠ Ø implies this was multiple characters and the count must be updated.

DINGME

(2) TO1ØIC and TO11IC are wrap-around counters of Indirect Transfers where TO1ØIC is maintained by the -11 and TO11IC is maintained by the -1Ø. If the two wrap-around counters are equal, it means the transfer finished correctly.

(3) If the difference between the wrap-around counters is greater than 1, the -11 has tried to send a direct transfer before the last indirect transfer finished or a doorbell has been lost in a previous transaction.

(4) Receiver sets TOIT equal to 1 in Sender's section of Receiver's communication region after Sender sets @ or increments Q count and rings the doorbell; Receiver clears TOIT upon getting To-Receiver Done (This assures that the Receiver doesn't lose an interrupt).

(5) The function table has dispatches for such features as:

- F.E. telling about the CTY
- String data for the CDR
- Line characters (for DLS)
- -11 Sending error information
- -11 wants or is sending time of day
- Line dialed up, hung up or line buffer empty
- Set line speed or allocation
- Take -11 reload information
- Acknowledge all devices and units
- Take KLINIK data.

Job Startup

INTRODUCTION

A job is started by putting a request in table SCDRQB, the scheduler special request table. An entry in SCDRQB is one word with the following format: data in the left half and dispatch address in the right half. The dispatch address for job startup is JOBSRT. All jobs are started this way, including Job 0.

A request to start Job 0 is put in SCDRQB by the system startup code. Requests for later jobs will be added to SCDRQB at the TTYSRV interrupt level when the user types CTRL/C on an inactive line.

If the job being started is Job 0, some special system initialization routines are called.

OVERHEAD CYCLE JOB STARTUP TASKS

Table SCDRQB is checked in the overhead cycle for requests. Currently, SCDRQB is used only for job startup requests; routine JOBSRT is called in the overhead cycle to start a job.

JOBSRT checks the availability of system resources such as the number of free SPT slots, the amount of drum space available, the number of free job and fork slots. If there are not enough system resources to start the job, the message "FULL" is typed on the terminal and the job is not started.

If enough resources are available to start the job, the necessary slots are assigned in the SPT table and the job and fork numbers are assigned. The fork is added to WTLST with a null wait test of [0,,JSKP]. JSKP is a routine which will always give the skip return, indicating that the fork's wait is satisfied and the fork can be moved to the GOLST. The new job and new fork flags are set in table FKINT, indexed by fork number. JOBSRT then returns and the overhead cycle continues.

When routine WTCHK is called by the overhead cycle, the fork will be moved to the GOLST. WTCHK is called by the 100 ms. clock when there is no fork to schedule. Note that when Job 0 is being started, there will be no other forks on the system; therefore, WTCHK will be called to move the fork to the GOLST.

When SKDJOB chooses the fork to run, it will see both the new job and new fork flags and will set the PC for the fork to PIRQ; when the fork starts, it will begin execution there.

JOB STARTUP TASKS IN PROCESS CONTEXT

When the process is chosen to run, it executes code in EXEC mode to complete job startup. When Job 0 is started, special routines are called to finish system startup. The job startup routines and their functions are described below.

1. PIRQ (fork startup code)

   The PSB and JSB (if appropriate) are initialized and EXEC0 is stored as the PC. The software interrupt is dismissed. When the fork resumes execution, the PC is EXEC0.

2. EXEC0

   If this is the first job on the system (Job 0), call the following routines:

   1. -SEBINI - initialize SYSERR data base
   2. -FSIINI - mount the public structure. This includes:
      1. Call FSIDIA if the file system is being initialized (FSIDIA is the dialogue to define PS:)
      2. Read the home block

3. Create SDB and STRTAB entries for PS:

3. -SWPINI - initialize the swapper data base tables

4. -CHKBAT - read the BAT blocks and perform quick consistency check for the PS: structure.

5. -GETSWM - get swappable monitor, using VBOOT

6. -RESLCK - lock down some resident free space

After getting the swappable monitor, the Job Ø initialization code joins normal job startup at GOTSWM.

3. GOTSWM

1. -The fork structure (tables FKPTRS and SYSFK) in the JSB is initialized for the job

2. -The software interrupt channels the monitor uses are reserved

3. -JBFINI is called to initialize JSB locations

If this is Job Ø initialization (SYSIFG equals zero), the following tasks are completed (after which Job Ø initialization rejoins normal job initialization at SYSINE):

1. -FILINI - file system initialization; if flag MI%RFS is on in STARTF, the file system is built

2. -PIDINI - IPCF data base initialization

3. -SLNINI - initialize system logical names
4. -GETNAM - read MONNAM.TXT to get system banner
5. -Initialize the time zone
6. -Initialize accounting flags

4. SYSINE

1. -Initialize job's PID and ENQ/DEQ quota
2. -If CRJOB JSYS, set controlling job and TTY locations
3. -TTYASN - assign controlling terminal number
4. -TTCKSP - set terminal speed if necessary

If this is a special job, log it in as operator and dispatch to the appropriate address from table SPECJT, indexed by job number. Currently the only special job is Job 0; it dispatches to RUNDD, whose functions are described below.

If this is a normal job, set up terminal information (such as setting terminal type to standard). Get a JFN on the EXEC, GET it and start it.

5. RUNDD (Job 0 only)

1. -Set CTY as controlling TTY
2. -TTSPIN - initialize terminal speeds to null
3. -PROINI - start primary protocol

4. -DTRMDS - tell console front end not to answer data sets

5. -Try to get time and date from the console front end; if it does not know, ask the operator

6. -Run SETSPD to set line speeds, system logical names, and other system parameters

7. -If regular startup (DBUGSW equal Ø or 1), tell all users that system is restarting

8. -LOGSST - log system restart in SYSERR file

9. -Run CHECKD, if necessary

1Ø. -SERINI - initialize SYSERR logging fork; starts fork at SEBRUN in exec mode. This fork opens the SYSERR log file and MDISMSs with wait test [Ø,,SEBTST]. When there is a request is queued for SYSERR, its wait condition is satisfied and it processes the request

11. -USGINI - start accounting; opens USAGE and CHECKPOINT files

12. -Run SETSPD to copy DUMP.EXE to DUMP.CPY and move any queued SYSERR blocks from the dump to ERROR.SYS

13. -Start DDMP in this fork and send message to CTY saying DDMP is running

14. -IMPBEG - start NCP fork, if any (for ARPA systems)

15. -NSPINI - initialize DECnet fork and data base

16. -Create fork and start SYSJOB in it

17. -Go to CHKR, the background task that runs every 1Ø seconds.

NOTE

Job 0 (the DDMP fork) runs only in exec mode, using the monitor address space.

System Startup

STARTUP VECTORS

The system can be started at one of several addresses, depending on the kind of startup. The system startup code begins in STG with a set of startup vectors; the first vector is loaded at 140. The startup vectors and their functions are as follows:

EVDDT=140/ JRST DDTX

Starts the monitor in EDDT; does not do a reset first. Can be used for debugging when a restart without resetting the machine is desired.

141/ JRST SYSDDT

Resets the machine and starts the monitor in EDDT. Can be used for debugging when a reset and restart is desired or when a restart alone does not work.

EVDDTX=142/ JRST DDTX

Copy of 140 in case 140 is clobbered. (Provided for historical reasons.)

EVSLOD=143/ JRST SYSLOD

Initializes the file system. Used for system installation.

144/ XPCW RLODPC

Reloads vector for front end.

EVRST=145/ JRST SYSRST

Meant for use with power fail; not really supported.

EVLGO=146/ JRST SYSGO

Provided for historical reasons.

EVGO=147/ JRST SYSGO1

Normal restart.

Of these startup vectors, only 140, 141, 143 and 147 currently provide really useful and separate system startups. Functionally, they provide:

1. Enter EDDT -- Both 140 and 141 enter EDDT

   Startup vector 141 does a reset of the machine first by calling PIRST (to reset the PI system) and IORST (to set up EBR, set up UBR, and clear paging and cache).

2. Initialize file system (vector 143)

   Sets flag MI%RST in STARTF to note the file system is being initialized and then joins regular system startup code at SYSLOD+1. This flag triggers the file system initialization in the Job 0 code.

3. Normal startup (vector 147)

   Sets STARTF flag to indicate normal startup and begins system startup at SYSLOD+1.

SYSTEM STARTUP CODE

The system startup code which begins at SYSLOD+1 calls routines to initialize the monitor data base, puts a job request in the scheduler request table and goes to the scheduler. The scheduler schedules the Job 0 fork, which does the remainder of system startup. The routines called to initialize the monitor data base (before the Job 0 fork is started), and their functions are:

1. SYSLOD code (in STG)

    1. -Execute PIRST and IORST (see above).
    2. -Initialize DTEs
    3. -Clear resident storage area
    4. -Read APR serial number
    5. -Initialize bit table data base
    6. -Read APR serial number
    7. -Set extended addressing flag to indicate if machine has extended addressing
    8. -Set up BUGCHK, BUGHLT, and BUGINF (in case a breakpoint was set)
    9. -Note if EDDT is to be flushed
    1Ø. -Initialize SWPCOR

2. RESFPI (in FREE)

    Initialize resident free pool.

3. PAGRST (in APRSRV)

    Set up EPT, scheduler's UPT and set up for TOPS-2Ø paging. This includes:

    1. -Store SPT and CST base addresses in AC block 6 (microcode/software interface AC block)
    2. -Initialize CSTDAT and CSTMSK in AC block 6
    3. -Initialize EPT, using template IEPTØ
    4. -Initialize scheduler's UPT, using template IUPTØ

5. -Store KIPFS as page fail dispatch address in scheduler's UPT

6. -Set up CONOPG for KL paging, (but do not turn on KL paging)

4. PGRINI (in PAGEM)

Initialize pager data base and turn on KL paging.

1. -Initialize DST, SPT, SPTH, and CST tables

2. -Assign SPT slots for running fork's PSB, JSB and for the bit table and MMAP

3. -Initialize MMAP

4. -Initialize FPTABL (PAGEMs section dispatch table); makes all sections illegal

5. -Set up section pointer(s) for monitor in MSECTB; set up dispatch addresses in FPTABL for all legal sections in this monitor

6. -Initialize MMSPTN (MMAPs SPT slot)

7. -Set up MMAP entries for resident monitor (virtual and physical addresses are the same)

8. -Set up MMAP entries for the bit table (indirect pointers through the bit table's index block)

9. -Set up MMAP entries for JSB area and PSB area (indirect pointers through the JSB and PSB maps)

10. -Set up ARPANET section (if this is an ARPA machine)

11. -Set scheduler context

12. -Turn on TOPS-20 paging (call PGRON)

13. -Construct RPLQ

14. -Set up core management constants

5. UNBINI *2020 only*

Initialize UNIBUS related data base.

6. PHYINI (in PHYSIO)

Initialize the PHYSIO data base. A CDB is set up for each channel and a UDB for each disk and magtape unit. The IORB free list is built.

7. TTINIT (in TTYSRV)

Initialize TTY data base.

1. -Initialize TTY buffers

2. -Initialize TTY tables that are indexed by line

3. -Establish line type for each line

4. -Assign internal line numbers for all lines

5. -Initialize BIGBUF data base

8. SCDIN (in SCHED)

Initialize scheduler data base.

1. -Initialize scheduler flags

2. -Set up OKSKED locations (RSKCHK)

3. -Set up JSYS trap queue

4. -Initialize free fork list

5. -Initialize free job list

6. -Turn off Job Ø alarm (until Job Ø fork is started)

7. -Initialize balance set queue

9. PIINIT (in APRSRV)

Initialize priority interrupt system.

1. -Set up XPCW instructions in the standard interrupt locations for levels 4-7

2. -Set up for power fail

3. -Initialize DTE (set up XPCW instructions for each DTE vectored interrupt location and set up the DTE PI channel for vectored interrupts)

4. -Set up interval timer (set up XPCW instruction for interval timer, set the interval, and set its PI assignment)

1Ø. SCDRQ7 (in SCHED)

Put a request for a new job in the scheduler request table; this starts the first Job Ø fork.

Enter the scheduler at SCHEDØ; that is, start the overhead cycle. The overhead cycle services requests in the scheduler request table and this starts the first Job Ø fork. The rest of system initialization happens in process context (Job Ø context). See the job startup description for the rest of system startup. (Note that the system has not yet read in the swappable monitor, has not asked for the time and date, etc.)

Appendix I

PART A

Alphabetical List of BUGHLTs,

BUGCHKs and BUGINFs

```
BUG(HLT,ABKSKD,<ADDRESS BREAK FROM SCHEDULER CONTEXT>)
BUG(HLT,ADDONF,<ADDOBJ-LLLKUP FAILED>)
BUG(HLT,APRNX1,<NXM DETECTED BY APR>,<A>)
BUG(HLT,APRNX1,<NXM DETECTED BY APR>)
BUG(HLT,APRNX2,<NXM DETECTED BY APR>) ;YES
BUG(HLT,ASGSW2,<SWPOMG-CAN'T ASSIGN RESERVED DRUM ADDRESS>)
BUG(HLT,ASOFNF,<DELFIL: ASOFN GAVE FAIL RETURN FOR LONG FILE XB>)
BUG(HLT,ASTJFN,<GETFDB: CALLED FOR JFN WITH OUTPUT STARS>)
BUG(HLT,BADBTB,<NIC- ILLEGAL REFERENCE TO BIT TABLE>)
BUG(HLT,BADDAC,<INSACT - NULL ACCOUNT STRING SEEN>)
BUG(HLT,BADREC,<FILINI - Reconstruction of ROOT-DIRECTORY failed>)
BUG(HLT,BADROT,<FILIN2: ROOT-DIRECTORY IS INVALID>)
BUG(HLT,BADTTY,<TRANSFER TO NONEXISTENT TTY CODE>)
BUG(HLT,BADTYP,<BAD LABEL FIELD DESC>)
BUG(HLT,BADXT1,<INDEX TABLE MISSING AND CAN NOT BE CREATED>)
BUG(HLT,BADXTB,<FILIN2: Could not initialize index table>)
BUG(HLT,BKUPDF,<BKUPD - BAD CST1 ENTRY OR INCONSISTENT CST>)
BUG(HLT,BOOTCR,<GETSWM - NOT ENOUGH CORE FOR SWPMON>)
BUG(HLT,BOOTER,<GETSWM - ERROR LOADING SWPMON>)
BUG(HLT,BOOTLK,<GSMDSK - FAILED TO LOCK NEEDED PAGES>)
BUG(HLT,BOOTMP,<GSMDSK - CANNOT MAP BOOTSTRAP PAGES>)
BUG(HLT,BTBCR1,<FILINI - NO BIT TABLE FILE AND UNABLE TO CREATE ONE>)
BUG(HLT,BTBCRT,<FILINI - COULD NOT INITIALIZE BIT TABLE FOR PUBLIC STRUCTURE>)
BUG(HLT,CDILVT,ILLEGAL DEVICE TYPE)
BUG(HLT,CKDFRK,<JOB 0 CFORK FAILED>)
BUG(HLT,CLRACE,UNABLE TO CLEAR REGISTER ACCESS ERROR)
BUG(HLT,CST2I1,<PAGE TABLE CORE POINTER AND CST2 FAIL TO CORRESPOND>)
BUG(HLT,CST2I2,<MVPT-CST2 INCONSISTENT>)
BUG(HLT,CST2I3,<PAGE TABLE CORE POINTER AND CST2 FAIL TO CORRESPOND>)
BUG(HLT,DELNDF,<DELNOD-LLLKUP FAILED>)
BUG(HLT,DGUTPG,<DIAG - LOCKED PAGE LIST PAGE LOCKED AT DIAG UNLOCK>)
BUG(HLT,DGZTPA,<DIAG - LOCKED PAGE LIST PAGE WAS ZERO>)
BUG(HLT,DNOPT0,<DSKCLZ-JFNOFN FAILED FOR PAGE 0>)
BUG(HLT,DRMFL1,<ASFSB-UNEXPECTED DRUM FULL>)
BUG(HLT,DRMFUL,<DRUM COMPLETELY FULL>)
BUG(HLT,DRMIBT,<DRMASN-BIT TABLE INCONSISTENT>)
BUG(HLT,DRMNFR,<DRMAM-CAN'T FIND PAGE WHEN DRMFRE NON-0>)
BUG(HLT,DRUMP1,<DRMIO - DRUMP ON BUT NO DRUM CODE IN SYSTEM>)
BUG(HLT,DST2SM,<SWPINI-DST TOO SMALL>)
BUG(HLT,DTECAR,<DTESRV- CARRIER FUNCTION WITH NO LINE NUMBER PRESENT>)
BUG(HLT,DTEDEV,<LINEAL -ILLEGAL DEVICE>)
BUG(HLT,DTEIDP,<DTESRV- INDIRECT POINTER WITH GARBAGE PACKET>)
BUG(HLT,DTEIFR,<DTESRV-ILLEGAL FUNCTION REQUEST FROM 11>)
BUG(HLT,DTEMCC,<DOFRGM-MCB DISAGREES WITH COUNT>) ;NO
BUG(HLT,DTETTY,<TAKLC-NON-TTY DEVICE ON FUNCTION CODE 4>)
BUG(HLT,DTEUIF,<DTESRV-UNIMPLEMENTED FUNCTION FROM 11>)
BUG(HLT,DUPCOR,<No core for DUP11>)
BUG(HLT,DUPCOR,<No core for DUP11>)
BUG(HLT,DUPUBA,<no Unibus Address>)
BUG(HLT,DUPUBA,<no Unibus Address>)
BUG(HLT,DZCLRB,<UNABLE TO RESET DZ11>)
BUG(HLT,EXPAFK,<EXPALL: JOB 0 CFORK FAILED>)
BUG(HLT,FATAPE,<FATAL ADDRESS PARITY ERROR>,<A>)
BUG(HLT,FATCDP,<FATAL CACHE DIRECTORY PARITY ERROR>,<A>)
BUG(HLT,FATMER,<FATAL MEMORY ERROR>)
BUG(HLT,FILBTB,<UNABLE TO WRITE BIT TABLE FILE>)
BUG(HLT,FILIRD,<FILINW: COULD NOT INITIALIZE THE ROOT DIRECTORY>)
BUG(HLT,FILMAP,<FILIN2: COULD NOT MAP IN ROOT-DIRECTORY>)
BUG(HLT,FILRID,<FILINW: INDEX TABLE ALREADY SET UP FOR ROOT DIR>)
BUG(HLT,FRKNPT,<FKHPTN - FORK HAS NO PAGE TABLE>)
```

```
BUG(HLT,FRKPTE,<BADCPG-FATAL ERROR IN FORK PT PAGE>)
BUG(HLT,FRKSLF,<SUSFK - GIVEN SELF AS ARG>)
BUG(HLT,GLFNF,<GLREM - FORK NOT FOUND>)
BUG(HLT,GTFDB2,<NEWLFP: GETFDB FAILURE FOR OPEN FILE.>)
BUG(HLT,GTFDB3,<DSKREN-GETFDB FAILURE FOR OPEN FILE>)
BUG(HLT,GTFDB6,<CRDIØA: CANNOT DO GETFDB ON ROOT-DIRECTORY >)
BUG(HLT,HSYFRK,<HSYS-JOB Ø CFORK FAILED>)
BUG(HLT,IBCPYW,<COPY-WRITE POINTER IN INDEX BLOCK>)
BUG(HLT,IBOFNF,<FILINI: ASOFN FAILURE FOR ROOT DIRECTORY IB>)
BUG(HLT,IDXNOS,<FILINI - COULD NOT ASSIGN FREE SPACE FOR IDXTAB>)
BUG(HLT,ILAGE,<BAD AGE FIELD IN CSTØ>)
BUG(HLT,ILBOOT,<GETSWM-ILLEGAL VALUE OF BOOTFL>)
BUG(HLT,ILCHS1,<PHYSIO - ILLEGAL CHANNEL STATUS AT SIO>)
BUG(HLT,ILCHS2,<PHYSIO - ILLEGAL CHANNEL STATE AT STKIO>)
BUG(HLT,ILCNST,<PHYSIO - ILLEGAL CALL TO CONSTW>)
BUG(HLT,ILCNSP,<PHYSIO - ILLEGAL CALL TO CONSPW>)
BUG(HLT,ILCST1,<ILLEGAL ADDRESS IN CST1 ENTRY, CAN'T RESTART>)
BUG(HLT,ILDEST,<ILLEGAL DESTINATION IDENTIFIER TO SETMPG OR SETPT>)
BUG(HLT,ILDRA2,<DRMIAD-ILLEGAL DRUM ADDRESS>)
BUG(HLT,ILFPTE,<ILLFPT: ILLEGAL SECTION NUMBER REFERENCED>)
BUG(HLT,ILGDA1,<GDSTX - BAD ADDRESS>)
BUG(HLT,ILGDA2,<GDSTX - BAD ADDRESS>)
BUG(HLT,ILIRBL,<PHYSIO - IORB LINK NOT NULL AT ONFPWQ>)
BUG(HLT,ILLIND,<ILLEGAL INDIRECT>)
BUG(HLT,ILMADR,<ILLEGAL ADDRESS REFERENCE IN MONITOR>)
BUG(HLT,ILOFN1,<MSCANP-ILLEG IDENT>)
BUG(HLT,ILOKSK,<OKSKED WHEN NOT NOSKED>)
BUG(HLT,ILPAGN,<MRKMPG-INVALID PAGE NUMBER>)
BUG(HLT,ILPAG1,<SWPOTØ-INVALID PAGE>)
BUG(HLT,ILPDAR,<PHYSIO - ILLEGAL DISK ADDRESS IN PAGEM REQUEST>)
BUG(HLT,ILPLK1,<MLKPG-ILLEGAL ARGS>)
BUG(HLT,ILPPT1,<UPDOFN-BAD POINTER IN PAGE TABLE>)
BUG(HLT,ILPPT2,<UPDPGS-BAD POINTER IN PAGE TABLE>)
BUG(HLT,ILPPT3,<BAD POINTER IN PAGE TABLE>)
BUG(HLT,ILPTN1,<MRPACS-ILLEG PTN>)
BUG(HLT,ILRBLT,<PHYSIO - IORB LINK NOT NULL AT ONF/STWQ>)
BUG(HLT,ILRFPD,<PDL-OV IN ILLEGAL PAGE REFERENCE>)
BUG(HLT,ILSPTI,<ILLEGAL SPT INDEX GIVEN TO SETMXB>)
BUG(HLT,ILSPTH,<SETPT-SPTH INCONSISTENT WITH XB>)
BUG(HLT,ILSRC,<ILLEGAL SOURCE IDENTIFIER GIVEN TO SETPT>)
BUG(HLT,ILSTP3,<VERLUK: IMPOSSIBLE SKIP RETURN FROM EXTLUU>)
BUG(HLT,ILSWPA,<SWPIN - ILLEGAL SWAP ADDRESS>)
BUG(HLT,ILTWQ,<PHYINT - TWQ OR PWQ INCORRECT>) ;NO.
BUG(HLT,ILTWQP,<PHYSIO - PWQ OR TWQ TAIL POINTER INCORRECT>)
BUG(HLT,ILULK1,<MULKPG - TRIED TO UNLOCK PAGE NOT LOCKED>)
BUG(HLT,ILULK2,<TRIED TO UNLOCK PAGE NOT LOCKED>)
BUG(HLT,ILULK3,<MULKMP - ILLEGAL MONITOR ADDRESS>)
BUG(HLT,ILULK4,<MULKCR - ILLEGAL CORE PAGE NUMBER>)
BUG(HLT,ILUST1,<PHYSIO - UNIT STATUS INCONSISTENT AT SIO>)
BUG(HLT,ILUST5,<PHYSIO - ILLEGAL UNIT OR CHANNEL STATE AT STKIO>)
BUG(HLT,ILUST4,<PHYSIO - CONTROLLER ACTIVE AT SPS>)
BUG(HLT,ILUST3,<PHYSIO - SCHSEK - IMPOSSIBLE UNIT STATUS>)
BUG(HLT,ILWRT2,<ATTEMPTED WRITE REF TO PROTECTED MONITOR>)
BUG(HLT,ILXBP,<SETPT-BAD POINTER IN XB>)
BUG(HLT,IMPAFB,<IMPCQ: ATTEMPT TO UNLOCK BUFFER ON FREELIST>)
BUG(HLT,IMPALF,<IMPLKB: ATTEMPT TO LOCK BUFFER ON FREELIST>)
BUG(HLT,IMPAUF,<IMPEIN: BUFFER ON FREELIST USED FOR INPUT>)
BUG(HLT,IMPCCF,<CAN'T CREATE IMP FORK>)
BUG(HLT,IMPNBC,<PKMSG: NEGATIVE RESIDUAL BYTE COUNT>)
BUG(HLT,IMPNII,<NO IMP INPUT BUFFERS>)
```

```
BUG(HLT,IMPRMI,<IMP - REGULAR MESSAGE ON IRREG QUEUE>)
BUG(HLT,IMPUBF,<IMULKB: ATTEMPT TO UNLOCK BUFFER ON FREELIST>)
BUG(HLT,IMPUFB,<IMIP1: ATTEMPT TO UNLOCK BUFFER ON FREELIST>)
BUG(HLT,IMPUUO,<IMPOSSIBLE MUUO>)
BUG(HLT,INVDTE,<DTEQ- INVALID DTE SPECIFIED>)
BUG(HLT,IONXM,<I/O NXM ON UNIBUS DEVICE>)
BUG(HLT,IOPGF,<IO PAGE FAIL>,<Q1>)
BUG(HLT,IPCOVL,<PIDINI: PIDS AND FREE POOL OVERLAP, IPCF WON'T WORK!>)
BUG(HLT,JØNRUN,<JOB Ø NOT RUN FOR TOO LONG, PROBABLE SWAPPING HANGUP>)
BUG(HLT,JSBNIC,<SETPPG-JSB NOT IN CORE>)
BUG(HLT,JTENQE,<JTENQ WITH BAD NSKED>)
BUG(HLT,KMCIII,<KMC11 illegal input interrupt>,<T1,T2>)
BUG(HLT,KMCIII,<KMC11 illegal input interrupt>,<T1,T2>)
BUG(HLT,KPALVH,<KEEP ALIVE CEASED>)
BUG(HLT,LCKDIR,<ATTEMPT TO LOCK DIRECTORY TWICE FOR SAME FORK>)
BUG(HLT,LUUMNØ,<LUUO IN MONITOR CONTEXT>)
BUG(HLT,LUUMON,<.LBCHK: ILLEGAL LUUO FROM MONITOR CONTEXT>)
BUG(HLT,MAP41F,<MAPF41 FAILED TO SKIP>)
BUG(HLT,MAPBT1,<OFN FOR BIT TABLE IS ZERO>)
BUG(HLT,MDDJFN,<GETFDB: CALLED FOR NON-MDD DEVICE>)
BUG(HLT,MNTLNG,<MNTBTB - BIT TABLE IS A LONG FILE>)
BUG(HLT,MONPDL,<OVERFLOW OR PDL OVERFLOW TRAP IN MONITOR>)
BUG(HLT,MPEUTP,<PFCDPE-UNKNOWN TRAP ON TEST REFERENCE>)
BUG(HLT,MTARIN,<MTAINT: INTERRUPT RECEIVED FOR NONACTIVE IORB>)
BUG(HLT,MTFCNX,<MTLFCN: FUNCTION CODE TOO LARGE>)
BUG(HLT,NCDWA,<KSINI: NO CARDREADER UBA WINDOW>)
BUG(HLT,NETBAU,<ASNTBF: ATTEMPT TO ASSIGN A BUFFER ALREADY IN USE>)
BUG(HLT,NETBAF,<RLNTBF: ATTEMPT TO RELEASE BUFFER ALREADY ON FREE LIST>)
BUG(HLT,NETIEF,<NETOPN: EXTDEC FAILURE AFTER PREVIOUS NON-FAILURE.>)
BUG(HLT,NETNNI,<NETINI: NNTBFS NOT INTEGRAL MULTIPLE OF MAXWPM>)
BUG(HLT,NETRBL,<ASNTBF: REQUEST FOR BUFFER LARGER THAN MAXWPM>)
BUG(HLT,NETRBG,<RLNTBF: ATTEMPT TO RELEASE BUFFER AT GARBAGE LOCATION>)
BUG(HLT,NETWNS,<WATNOT: WAS CALLED FROM SCHEDULER LEVEL.>)
BUG(HLT,NEWBAK,<FILRFS - NEWIB FAILURE FOR BACKUP ROOT-DIR>)
BUG(HLT,NEWROT,<FILRFS - NEWIB FAILURE FOR ROOT-DIRECTORY>)
BUG(HLT,NLWA,<L2INI:  No lineprinter window available>)
BUG(HLT,NOACB,<MENTR - NO MORE AC BLOCKS>)
BUG(HLT,NOADXB,<RELOFN-NO DSK ADR FOR XB>)
BUG(HLT,NOBTBN,<FILINI - UNABLE TO GET SIZE OF BOOTSTRAP.BIN FILE>)
BUG(HLT,NOCTY,<UNABLE TO ALLOCATE DATA FOR CTY>)
BUG(HLT,NOFEFS,<FILINI - UNABLE TO GET SIZE OF FRONT END FILE SYSTEM>)
BUG(HLT,NOFNDU,<FNDUNT-CAN'T FIND DEVICE FOR JFN>)
BUG(HLT,NOIORB,<SETIRB - MISSING IORB>)
BUG(HLT,NOLEN,<UPDLEN: NO LENGTH INFO FOR OFN>)
BUG(HLT,NOPGTØ,<OPNLNG: NO PAGE TABLE Ø IN LONG FILE.>)
BUG(HLT,NORSXF,<FAILED TO GET SPACE FOR MASTER DTE>)
BUG(HLT,NOSEB2,<PGMPE-NO SYSERR BUFFER AVAILABLE>)
BUG(HLT,NOTOFN,<UPDOFØ-ARG NOT OFN>)
BUG(HLT,NOUBWA,<RH2NCH: NO UNIBUS WINDOW FOR RH11>)
BUG(HLT,NOXADR,<EXTENDED ADDRESSING CONFUSION>)
BUG(HLT,NSKDIS,<DISMISS WHILE NOSKED OR WITH NON-RES TEST ADDRESS>)
BUG(HLT,NSPFRK,<NSPINI-CFORK FAILED>)
BUG(HLT,NSPUDF,<UNSUPPORTED NETWORK FUNCTION>)
BUG(HLT,NULQTA,QCHK - NO QUOTA INFO SETUP)
BUG(HLT,OFFSPE,<OFFSPQ- PAGE NOT ON SPMQ>)
BUG(HLT,OPOPAC,<MRETN - TRIED TO OVER-POP AC STACK>)
BUG(HLT,OVFLOW,<ASOFN - ALLOCATION TABLE OVERFLOW>)
BUG(HLT,PAGLCK,<DESPT-PAGE LOCKED>)
BUG(HLT,PAGNIC,<GETCPP-PAGE NOT IN CORE>)
BUG(HLT,PFCDP,<MEMORY PARITY ERROR>)
```

```
BUG(HLT,PGNDEL,<REMFPB-PAGE NOT COMPLETELY DELETED>)
BUG(HLT,PH2WUI,<WRONG UNIT INTERRUPTED>)
BUG(HLT,PHYCH1,<PHYSIO - HOME BLOCK CHECK IORB ALREADY ON TWQ>)
BUG(HLT,PHYICA,<PHYINI - ILLEGAL ARGUMENT TO CORE ALLOC>)
BUG(HLT,PHYLTF,<PHYSIO - SCHLTM - UNEXPECTED LATOPT FAILURE>)
BUG(HLT,PHYPØE,<PHYALZ - PAGE Ø STORAGE EXHAUSTED>)
BUG(HLT,PIITRP,<INSTRUCTION TRAP WHILE PI IN PROGRESS OR IN SCHEDULER>)
BUG(HLT,PISKED,<ENTERED SCHEDULER WITH PI IN PROGRESS>)
BUG(HLT,PITRAP,<PAGER TRAP WHILE PI IN PROGRESS>)
BUG(HLT,PRONX2,<NXM DETECTED BY PROCESSOR>)
BUG(HLT,PSBNIC,<SETPPG-PSB NOT IN CORE>)
BUG(HLT,PSISTK,<PSI STORAGE STACK OVERFLOW>)
BUG(HLT,PTAIC,<SWPIN - PT PAGE ALREADY IN CORE>)
BUG(HLT,PTDEL,<DESPT-PT NOT DELETED>)
BUG(HLT,PTMPE,<PAGE TABLE PARITY ERROR>)
BUG(HLT,PTNIC1,<SWPIN - PAGE TABLE NOT IN CORE>)
BUG(HLT,PTNONØ,<SETPTØ - PREVIOUS CONTENTS NON-Ø>)
BUG(HLT,PTOVRN,<UPDPGS-COUNT TOO LARGE>)
BUG(HLT,PVTRP,<PROPRIETARY VIOLATION TRAP>)
BUG(HLT,PWRFL,<FATAL POWER FAILURE>) ;CRASH AND RELOAD
BUG(HLT,PYILUN,<PHYSIO - ILLEGAL UNIT NUMBER>)
BUG(HLT,RH11CC,<PHYH11 - ILLEGAL CHANNEL COMMAND WORD>)
BUG(HLT,RH1ICF,<PHYH11 - INVALID CHANNEL FUNCTION>)
BUG(HLT,RH2ICF,<PHYRH2 - INVALID CHANNEL FUNCTION>)
BUG(HLT,RP4FEX,<PHYP4 - ILLEGAL FUNCTION>)
BUG(HLT,RP4IF2,<PHYP4 - ILLEGAL FUNCTION AT STKIO>)
BUG(HLT,RP4IFC,<PHYP4 - ILLEGAL FUNCTION AT CNV>) ;YES TO EITHER
BUG(HLT,RP4ILF,<PHYP4 - ILLEGAL FUNCTION ON INTERRUPT>)
BUG(HLT,RP4LTF,<PHYP4 - FAILED TO FIND TWQ ENTRY AT RP4LTM>)
BUG(HLT,RP4PNF,<PHYP4 - DISK PHYSICAL PARAMETERS NOT FOUND>)
BUG(HLT,RP4UNF,<PHYP4 - UNIT TYPE NOT FOUND:>,T1)
BUG(HLT,RPGERR,<BADCPG-FATAL ERROR IN RESIDENT PAGE>)
BUG(HLT,RSMFAI,<RESSMM-FAILED TO ASSIGN SWAP MON PAGE>)
BUG(HLT,SECEX1,<SETMPG-ATTEMPT TO MAP NON-EX SECTION>)
BUG(HLT,SECG37,<ILSCN-SECTION NUMBER GREATER THAN 37>)
BUG(HLT,SECGT1,<PGRT3 - SECTION NUMBER GREATER THAN MAXSEC>)
BUG(HLT,SECNX,<CREATING PAGE TABLE FOR NON-Ø SECTION>)
BUG(HLT,SERFRK,<SERINI-CANNOT CREATE SYSERR FORK>)
BUG(HLT,SHRNØØ,<DESPT-SHARE COUNT NON-ZERO>)
BUG(HLT,SHROFN,<UPSHR-OFN SHARE COUNT OVERFLOW>) ;YES
BUG(HLT,SHROFD,<DWNSHR-OFN SHARE COUNT UNDERFLOW>)
BUG(HLT,SKDCL1,<CALL TO SCHEDULER WHEN ALREADY IN SCHEDULER>)
BUG(HLT,SKDCL2,<CALL TO SCHEDULER WHEN ALREADY IN SCHEDULER>)
BUG(HLT,SKDMPE,<MPE IN SCHEDULER OR PI CONTEXT>)
BUG(HLT,SKDPF1,<PAGE FAIL IN SCHED CONTEXT>) ;NO
BUG(HLT,SKDTRP,<INSTRUCTION TRAP WHILE IN SCHEDULER>)
BUG(HLT,SMNOFR,<NO FREE SPACE FOR SM1Ø VECTORS>)
BUG(HLT,SPTFL1,<SPT COMPLETELY FULL>)
BUG(HLT,SPTFL2,<SPT COMPLETELY FULL>)
BUG(HLT,SPTPIC,<SWPIN - SPT PAGE ALREADY IN CORE>)
BUG(HLT,SPTSHR,<UPSHR-SPT SHARE COUNT OVERFLOW>)
BUG(HLT,STKOVF,<MONITOR STACK OVERFLOW>)
BUG(HLT,STRBAD,<ASOFN-ILLEGAL STRUCTURE NUMBER>)
BUG(HLT,STZERO,<FILINI: STRTAB ENTRY FOR PS IS Ø>)
BUG(HLT,SWPMNE,<SWAP ERROR IN SWAPPABLE MONITOR>)
BUG(HLT,SWPPSB,<SWAP ERROR IN PSB PAGE>)
BUG(HLT,SWPPTP,<SWAP ERROR IN UNKNOWN PT PAGE>)
BUG(HLT,SWPPT,<SWAP ERROR IN UNKNOWN PT>)
BUG(HLT,SWPUPT,<SWAP ERROR IN UPT, OR PSB>)
BUG(HLT,TTBAD1,<BAD DEVICE DESIGNATOR FOR TERMINAL AT ATACH2>)
```

```
BUG(HLT,TTDAS1,<HLTJB: UNABLE TO DEASSIGN CONTROLLING TERMINAL>)
BUG(HLT,TTICNØ,<TCI - NO BUFFER POINTER BUT COUNT NON-Ø>)
BUG(HLT,TTNAC8,<CAN'T ASSIGN TERMINAL AT DEVINI>)
BUG(HLT,TTNAC5,<CTY NOT ACTIVE AT FSIINI>)
BUG(HLT,TTNAC4,<CTY NOT ACTIVE AT FSIPBI>)
BUG(HLT,TTNAC3,<CTY NOT ACTIVE AT FSIPBO>)
BUG(HLT,TTOCNØ,<TTSTO - NO BUFFER BUT COUNT NON-Ø>)
BUG(HLT,TTONOB,<TTY OUTPUT - NO BUFFER BUT COUNT NON-Ø>)
BUG(HLT,TWQNUL,<PHYSIO - PWQ OR TWQ WAS NULL AT A SEEK OR TRANSFER COMPLETION>)
BUG(HLT,UBANXM,<I/O NMX FROM UNIBUS DEVICE>,<UPTPFW,UPTPFO>)
BUG(HLT,UIONIR,<UDSKIO - NO IORB FOR NOSKED FORK>)
BUG(HLT,UNPGF1,<MEMPAR-PARITY ERROR DURING MEM SCAN>)
BUG(HLT,UNPGF2,<UNKNOWN PAGE FAILURE TYPE>)
BUG(HLT,UNTRAP,<UNKNOWN TRAP INSTRUCTION>)
BUG(HLT,UNXMPE,<PFCDPE-UNEXPECTED PARITY ERROR TRAP>)
BUG(HLT,UXXCKP,<COULDN'T CREATE CHECKPOINT FILE>)
BUG(HLT,UXXCRE,<CANNOT CREATE USAGE FILE>)
BUG(HLT,UXXILL,<USGMES: ILLEGAL FUNCTION CODE>)
BUG(HLT,UXXMAP,<USGMAP: CALL TO JFNOFN FAILED>)
BUG(HLT,UXXOPN,<UNABLE TO OPEN USAGE FILE>)
BUG(HLT,WRTLNG,<WRTBTB - BIT TABLE IS A LONG FILE>)
BUG(HLT,XSCORE,<CST TO SMALL FOR PHYSICAL CORE PRESENT>)
```

```
BUG(CHK,ASAASG,<DSKASA - ASSIGNING ALREADY ASSIGNED DISK ADDRESS>,<T1,T2>)
BUG(CHK,ASGBAD,<DSKASA - ASSIGNING BAD DISK ADDRESS>,<T3,T2>)
BUG(CHK,ASGBPG,<INIBTB-FAILED TO ASSIGN BAD PAGE(S)>,<T1,T2>)
BUG(CHK,ASGREQ,<ILLEGAL POOL NUMBER GIVEN TO ASGRES>)
BUG(CHK,ASGREP,<ILLEGAL PRIORITY GIVEN TO ASGRES>)
BUG(CHK,ASGSWB,<SWPINI-CAN'T ASSIGN BAD ADDRESS>)
BUG(CHK,BADBAT,<BAT BLOCKS UNREADABLE>)
BUG(CHK,BADBAK,<FILIN2 - BACKUP COPY OF ROOT DIRECTORY IS NOT GOOD>)
BUG(CHK,BADDIS,<TAPE: INCONSISTENT STATE CODE>)
BUG(CHK,BADHDR,<bad DDCMP header>,<T1,T2>)
BUG(CHK,BADHDR,<bad DDCMP header>,<T1,T2>)
BUG(CHK,BADIDX,<IDXINI: PARTIALLY UNSUCCESSFUL INDEX TABLE REBUILD>)
BUG(CHK,BADTAB,<VERACT - SPURIOUS HASH TABLE ENCOUNTERED>)
BUG(CHK,BADXT2,<INDEX TABLE MISSING AND WAS CREATED>)
BUG(CHK,BLKF1,<BYTINA: BLKF SET BEFORE CALLING SERVICE ROUTINE>)
BUG(CHK,BLKF2,<BYTOUA: BLKF SET BEFORE CALL TO SERVICE ROUTINE>)
BUG(CHK,BLKF3,<CLZDO: BLKF SET BEFORE CALL TO SERVICE ROUTINE>)
BUG(CHK,BLKF4,<.GDSTS: BLKF SET BEFORE CALL TO DEVICE ROUTINE>)
BUG(CHK,BLKF5,<.MTOPR: BLKF SET BEFORE CALL TO DEVICE ROUTINE>)
BUG(CHK,BLKF6,<.SDSTS: BLKF SET BEFORE CALL TO DEVICE ROUTINE>)
BUG(CHK,CDBDIN,<CD11 LOST INTERRUPT ENABLE>,T4)
BUG(CHK,CKLBLK,<CKLERR: CLOSE AND ABORT BLOCKED>)
BUG(CHK,CLZABF,<CLZFFW: SERVICE ROUTINE BLOCKED ON AN ABORT CLOSE>)
BUG(CHK,CPYUF1,<CACCT: IMPOSSIBLE FAILURE OF CPYFU1.>)
BUG(CHK,CRDBAK,<CRDIR3: COULD NOT MAKE BACKUP COPY OF ROOT-DIRECTORY>)
BUG(CHK,CRDBK1,<CRDIR4:COULD NOT MAKE BACKUP COPY OF ROOT-DIRECTORY>)
BUG(CHK,CRDNOM,<CRDIR-FAILED TO MAKE MAIL.TXT FILE>)
BUG(CHK,CRDOLD,<CRGDGB: OLD FORMAT CRDIR IS ILLEGAL>)
BUG(CHK,CRDSDF,<CRDIR1: SETDIR FAILED ON NEW DIRECTORY>)
BUG(CHK,CRSPAG,<VERACT - ACCOUNT DATA BLOCK CROSSES A PAGE BOUNDARY>)
BUG(CHK,DEABAD,<DSKDEA - DEASSIGNING BAD DISK ADDRESS>,<T3,T2>)
BUG(CHK,DEAUNA,<DEDSK-DEASSIGNING UNASSIGNED DISK ADDRESS>,<T1,T2>)
BUG(CHK,DEQMDF,<DEQUE: INTERNAL MONITOR DEQ FAILED>)
BUG(CHK,DEVUCF,<DEVAV - UNEXPECTED CHKDES FAILURE>)
BUG(CHK,DIRACT,<ACTBAD: ILLEGAL FORMAT FOR DIRECTORY ACCOUNT BLOCK IN DIRECTORY:>,<A,B>)
BUG(CHK,DIRB2S,<RLDFB1: DIRECTORY FREE BLOCK TOO SMALL IN DIRECTORY:>,<A,B>)
BUG(CHK,DIRB2L,<RLDFB2: DIRECTORY FREE BLOCK TOO LARGE IN DIRECTORY:>,<A,B>)
BUG(CHK,DIRBAD,<SETDI4: SMASHED DIRECTORY NUMBER:>,<A,SETDNM>)
BUG(CHK,DIRBAF,<RLDFB5: BLOCK ALREADY ON DIRECTORY FREE LIST IN DIRECTORY:>,<A,B>)
BUG(CHK,DIRBCB,<RLDFB3: DIRECTORY FREE BLOCK CROSSES PAGE BOUNDARY IN DIRECTORY:>,<A,B>)
BUG(CHK,DIRBLK,<BLKSCN: ILLEGAL BLOCK TYPE IN DIRECTORY:>,<A>)
BUG(CHK,DIRDNL,<ULKDIR-DIRECTORY NOT LOCKED, DIRECTORY NUMBER:>,<T1,T2>)
BUG(CHK,DIREXT,<EXTBAD: ILLEGAL FORMAT FOR DIRECTORY EXTENSION BLOCK IN DIRECTORY:>,<A,B>)
BUG(CHK,DIRFDB,<ILLEGAL FORMAT FOR FDB IN DIRECTORY:>,<A,B>)
BUG(CHK,DIRFKP,<SETDIR-DIR PAGE 0 BELONGS TO FORK IN DIRECTORY:>,<B,SETDNM>)
BUG(CHK,DIRFRE,<FREBAD: ILLEGAL FORMAT FOR DIRECTORY FREE BLOCK IN DIRECTORY:>,<A,B>)
BUG(CHK,DIRIFB,<RLDFB4: ILLEGAL BLOCK TYPE ON DIRECTORY FREE LIST IN DIRECTORY:>,<A,B>)
BUG(CHK,DIRNAM,<NAMBAD: ILLEGAL FORMAT FOR DIRECTORY NAME BLOCK IN DIRECTORY:>,<A,B>)
BUG(CHK,DIRPG0,<DR0CHK: ILLEGAL FORMAT FOR DIRECTORY PAGE 0 IN DIRECTORY:>,<A,B>)
BUG(CHK,DIRPG1,<DRHCHK: DIRECTORY HEADER BLOCK IS BAD IN DIRECTORY:>,<A,B>)
BUG(CHK,DIRRHB,<RLDFB6: ATTEMPTING TO RETURN A HEADER BLOCK IN DIRECTORY:>,<A,B>)
BUG(CHK,DIRSY1,<DELDL8: DIRECTORY SYMBOL TABLE FOULED UP FOR DIRECTORY:>,<A,B>)
BUG(CHK,DIRSY2,<MDDNAM: SYMBOL TABLE FOULED UP IN DIRECTORY:>,<A,B>)
BUG(CHK,DIRSY3,<LOOKUP: SYMBOL SEARCH FOULED UP IN DIRECTORY:>,<C,B>)
BUG(CHK,DIRSY4,<NAMCM4: DIRECTORY SYMBOL TABLE FOULED UP IN DIRECTORY:>,<A,B>)
BUG(CHK,DIRSY5,<SYMBAD: ILLEGAL FORMAT FOR DIRECTORY SYMBOL TABLE IN DIRECTORY:>,<A,B>)
BUG(CHK,DIRSY6,<RBLDST: PREMATURELY RAN OUT OF ROOM IN SYMBOL TABLE IN DIRECTORY:>,<A,B>)
BUG(CHK,DIRULK,<ULKMD2: ATTEMPT TO UNLOCK ILLEGALLY FORMATTED DIR, DIR NUMBER:>,<T1,T2>)
BUG(CHK,DIRUNS,<UNSBAD: ILLEGAL FORMAT FOR DIRECTORY USER NAME BLOCK IN DIRECTORY:>,<A,B>)
BUG(CHK,DMPRLF,<DMPREL-FAILED TO RELEASE PAGE>)
```

```
BUG(CHK,DSKBT1,<DSK BIT TABLE FOULED, CAN'T FIND FREE PAGE ON TRACK WITH NON-0 COUNT>,<T2,
BUG(CHK,DSKBT3,<DISK BIT TABLE ALREADY LOCKED AT LCKBTB>,<T1>)
BUG(CHK,DTEDAT,<TAKTOD- ILLEGAL FORMAT FOR TIME/DATE>)
BUG(CHK,DTEERR,<DTESRV-DTE DEVICE ERROR>,<A,F>)
BUG(CHK,DTEODD,<TAKLC-ODD BYTE COUNT FOR LINE CHARACTERS>)
BUG(CHK,DTEP2S,<T010DN-PACKET TOO SMALL>)
BUG(CHK,DTEPGF,<DTE TRANSFER PAGE FAIL>,<A>)
BUG(CHK,DTETIP,<DTETDN-T010 DONE RECEIVED WITH NO TRANSFER IN PROGRESS>)
BUG(CHK,DVCHRX,<DVCHR1 - UNEXPECTED CHKDES FAILURE WITHIN .DVCHR>)
BUG(CHK,DX2DIE,<PHYX2 - DX20 HALTED>,<T1>)
BUG(CHK,DX2FGS,<PHYX2 - FAIL TO GET SENSE BYTES>) ;PUT OUT BUGCHK
BUG(CHK,DX2FUS,<PHYX2 - FAIL TO UPDATE SENSE BYTES>)
BUG(CHK,DX2HLT,<PHYX2 - DX20 HALTED>,<T1>)
BUG(CHK,DX2IDM,<PHYX2 - ILLEGAL DATA MODE AT DONE INT>,<T2>)
BUG(CHK,DX2IEC,<PHYX2 - ILLEGAL ERROR CLASS CODE>,<T1>)
BUG(CHK,DX2IFS,<PHYX2 - ILLEGAL FUNCTION AT START IO>,<Q1>)
BUG(CHK,DX2MCF,<PHYX2 - DX20 MICROCODE CHECK FAILURE>)
BUG(CHK,DX2NRT,<DX2ERR - IS.NRT SET ON SUCCESSFUL RETRY>)
BUG(CHK,DX2NUD,<PHYX2 - CHANNEL DONE INTERRUPT BUT NO UNIT ACTIVE>)
BUG(CHK,DX2NUE,<PHYX2 - NO ACTIVE UDB AND DX20 COMPOSITE ERROR SET>,<T4,T1>)
BUG(CHK,DX2RFU,<PHYX2 - ERROR RECOVERY CONFUSED>)
BUG(CHK,DX2UPE,<PHYX2 - FAIL TO UPDATE SENSE BYTES DURING INITIALIZATION>)
BUG(CHK,DZLINT,<DZ11 LOST INTERRUPT ENABLE>,<T2>)
BUG(CHK,DZNENB,<DZSND1 - TRANSMIT NOT ENABLED ON INTERRUPT>)
BUG(CHK,DZOVER,<DZ11 SILO OVERRUN>,T1)
BUG(CHK,EFACF3,<EFACT: FAILED TO WRITE INTO FACT FILE>)
BUG(CHK,EFACF1,<EFACT: CLOSF FAILED TO CLOSE FACT FILE.>)
BUG(CHK,ENQMLF,<ENQUE: INTERNAL ENQ OF A MONITOR LOCK FAILED>)
BUG(CHK,EXPRCD,<EXPALL: RCDIR FAILURE>)
BUG(CHK,FEBAD,<FEHSD-WRONG FE>) ;NO
BUG(CHK,FEBFOV,<FEHSD-BUFFER OVERFLOW>,<A,C>) ;NO
BUG(CHK,FEOCPB,<FEFSYS - FAILED TO BACKUP ROOT-DIRECTORY>,<T1>)
BUG(CHK,FEUSTS,<FESSTS-UNKNOWN STATUS>)
BUG(CHK,FILBAK,<FILCRD: COULD NOT CREATE BACKUP OF ROOT-DIR>)
BUG(CHK,FILBOT,<COULD NOT CREATE BOOTSTRAP.BIN FILE>)
BUG(CHK,FILCCD,<Could not create directory>)
BUG(CHK,FILFEF,<Could not create Front End File System>)
BUG(CHK,FILHOM,<UNABLE TO REWRITE HOME BLOCKS IN WRTBTB>)
BUG(CHK,FILJB1,<FILCRD: No room to create standard system directories>)
BUG(CHK,FIXBAD,<Could not re-write Home Blocks to point to FE Filesystem>)
BUG(CHK,FIXBDB,<COULD NOT RE-WRITE HOME BLOCKS TO POINT TO BOOTSTRAP.BIN>)
BUG(CHK,FKWSP1,<LOADBS-UNREASONABLE FKWSP>,<T1,T2,T3>)
BUG(CHK,FLKINT,<FLOCK-CALLED WHILE NOINT>)
BUG(CHK,FLKNS,<FUNLK-LOCK NOT SET>)
BUG(CHK,FLKTIM,<FLOCK-TIMEOUT>)
BUG(CHK,FRKBAL,<AGESET-FORK NOT IN BALSET>)
BUG(CHK,FRKNDL,<FORK NOT PROPERLY DELETED>)
BUG(CHK,GTFDB1,<DSKINS: GETFDB FAILURE.>)
BUG(CHK,HARDCE,<HARD CACHE ERRORS--CACHE DESELECTED>)
BUG(CHK,HSHERR,<VERACT - HASH VALUE OUT OF RANGE>)
BUG(CHK,IDFOD1,<AT MENTR - INTDF OVERLY DECREMENTED>)
BUG(CHK,IDFOD2,<AT MRETN - INTDF OVERLY DECREMENTED>)>
BUG(CHK,ILDRA1,<DASDRM-ILLEGAL OR UNASSIGNED DRUM ADDRESS>)
BUG(CHK,ILIBPT,<BAD POINTER TYPE IN INDEX BLOCK>)
BUG(CHK,ILJRFN,<JFKRFH - BAD JRFN, IGNORED>)
BUG(CHK,ILLDMS,<BADDMS: ILLEGAL DMS JSYS FROM MONITOR CONTEXT>)
BUG(CHK,ILLTAB,<TABLK2: TABLE NOT IN PROPER FORMAT>)
BUG(CHK,ILLUUO,<KIBADU: ILLEGAL UUO FROM MONITOR CONTEXT>,<KIMUFL,KIMUPC,KIMUEF>)
BUG(CHK,ILPID1,<CREPID: ATTEMPT TO CREATE ILLEGAL PID>)
BUG(CHK,ILPID2,<DELPID: VALIDATED PID TURNED ILLEGAL>)
```

```
BUG(CHK,ILPSEC,<ILLEGAL SECTION NUMBER>,<TRAPPC,TRAPSW>)
BUG(CHK,ILUST2,<PHYSIO - UNIT STATUS INCONSISTENT AT SPS>)
BUG(CHK,IMPHNW,<LHOSTN DISAGREES WITH THE IMP>)
BUG(CHK,IMPIFH,<IMPGC-IMPOSSIBLE FAILURE OF IMPHFL>)
BUG(CHK,IMPLTF,<IMPLT FULL>)
BUG(CHK,IMPMSL,<PKMSG - MSG TOO LARGE>)
BUG(CHK,IMPNMA,<PKBY1: NO MSG ALLOCATION>,T2)
BUG(CHK,IMPREM,<UPBRB: RECEIVED EXCESSIVE MESSAGES>,T2)
BUG(CHK,IMPTMB,<NVTXG1: TOO MANY BREAKS OUTSTANDING>)
BUG(CHK,IMPUX0,<IMP JB0 FORK - UNEXPECTED INTERRUPT>)
BUG(CHK,IPCFKH,<CHKPDD: COULD NOT FIND LOCAL FORK HANDLE>)
BUG(CHK,IPCFRK,<PIDINB: CANNOT CREATE FORKS FOR IPCF>)
BUG(CHK,IPCJB0,<PIDINI: NOT IN CONTEXT OF JOB 0>)
BUG(CHK,IPCMCN,<MESREC: MESSAGE COUNT WENT NEGATIVE>)
BUG(CHK,IPCSOD,<GETMES: SENDER'S COUNT OVERLY DECREMENTED>)
BUG(CHK,KLIOVF,<DTESRV-KLINIK DATA BASE TOO LARGE>,<C>)
BUG(CHK,KMCNTI,<KMC11 not taking input>)
BUG(CHK,KMCNTI,<KMC11 not taking input>)
BUG(CHK,LNGDIR,<LONG DIRECTORY FILE IN DIRECTORY:>,<T3>)
BUG(CHK,LNMILI,<LNMLUK: ILLEGAL VALUE OF LOGICAL NAME TABLE INDEX>)
BUG(CHK,LP2IEN,<LINEPRINTER LOST INTERRUPT ENABLE>,U)
BUG(CHK,MPIDXO,<MAPIDX - No OFN for Index Table File>)
BUG(CHK,MSGCLB,<DDCMP transmit message clobbered>)
BUG(CHK,MSGCLB,<DDCMP transmit message clobbered>)
BUG(CHK,MSGPTR,<Bad msg pointer>)
BUG(CHK,MSGPTR,<Bad msg pointer>)
BUG(CHK,MTANOI,<GETUBF: NO QUEUED IORB'S FOR INPUT>)
BUG(CHK,MTANOQ,<IRBDN1: IRBDON CALLED FOR NON-QUEUED UP IORB>)
BUG(CHK,MTANOA,<IRBDN2: IRBDON CALLED FOR AN ACTIVE IORB>)
BUG(CHK,MTAORN,<MTDIR0: MAGTAPE IORB OVERRUN>)
BUG(CHK,NETDET,<NVTDET: COULD NOT CLOSE NVT>,<T1>)
BUG(CHK,NOALCM,<ALCMES: CANNOT SEND MESSAGE TO ALLOCATOR>)
BUG(CHK,NOBAT1,<FAILED TO WRITE PRIMARY BAT BLOCK>,<T1,T2>)
BUG(CHK,NOBAT2,<FAILED TO WRITE SECONDARY BAT BLOCK>,<T1,T2>)
BUG(CHK,NOBTB,<FILINI - UNABLE TO OPEN BIT TABLE FILE>)
BUG(CHK,NODIR1,<SPLMES: DIRST FAILED ON EXISTING DIRECTORY NAME>)
BUG(CHK,NOFRSP,<ttspst- COULD NOT GET A FREE BLOCK>)
BUG(CHK,NOINTR,<ITRAP AND PREVIOUS CONTEXT WAS NOINT>)
BUG(CHK,NOMHDR,<ILLEGAL MESSAGE WITH NO HEADER>)
BUG(CHK,NOPID,<PIDKFL: PID DISAPPEARED>)
BUG(CHK,NOSERF,<CAN'T GTJFN ERROR REPORT FILE>)
BUG(CHK,NOSKTR,<ITRAP FROM NOSKED CONTEXT>)
BUG(CHK,NOSLNM,<SLNINI: CANNOT CREATE SYSTEM LOGICAL NAME>)
BUG(CHK,NOSPLM,<RELJFN: COULD NOT SEND SPOOL MESSAGE TO QUASAR>)
BUG(CHK,NOUTF1,<SPLOPN: NOUT OF DIRECTORY NUMBER FAILED>)
BUG(CHK,NOUTF2,<SPLMES: NOUT OF GENERATION NUMBER FAILED>)
BUG(CHK,NPWQPD,<PHYSIO - NULL PWQ AT POSITION DONE>)
BUG(CHK,NRFTCL,<PHYSIO - NO REQUESTS FOUND FOR CYLINDER SEEKED>)
BUG(CHK,NSKDT2,<PGRTRP-BAD INTDF>)
BUG(CHK,NSKDT2,<PGRTRP-BAD NSKED OR INTDF>)
BUG(CHK,NSPRTH,<NSPTSK- INVALID ROUTING HEADER>,<T1,T2>)
BUG(CHK,NWJTBE,<NO FREE JTB BLOCKS>)
BUG(CHK,P11PAR,<PHYH11 -- CONTROL WRITE PARITY ERR>,<T1,T2>)
BUG(CHK,P1NED1,<PHYH11 - RH11 NON EX DISK READING REGISTER>,<T1,T2>)
BUG(CHK,P2RAE2,<PHYH11 - REGISTER ACCESS ERR WRITING REG>,<T1,T2,T3>)
BUG(CHK,P2RAE1,<PHYH2 - RH20 REGISTER ACCESS ERROR READING REGISTER>,<T1,T2,T3>)
BUG(CHK,P2RAE2,<PHYH2 - REGISTER ACCESS ERR WRITING REG>,<T1,T2,T3,T4>)
BUG(CHK,P2RAE3,<PHYH2 - REGISTER ACC ERR ON DONE OR ATN INTERRUPT>,<T1,T2,T3>)
BUG(CHK,PH1IHM,<PHYH11 - ILLEGAL HDW MODE - WORD MODE ASSUMED>)
BUG(CHK,PH1PIE,<PHYH11 - RH11 LOST INTERRUPT ENABLE>)
```

```
BUG(CHK,PH2IHM,<PHYH2 - ILLEGAL HDW MODE - WORD MODE ASSUMED>)
BUG(CHK,PH2PIM,<PHYH2 - RH2Ø LOST PI ASSIGNMENT>,<T2>)
BUG(CHK,PHYNIR,<PHYSIO - NULL INTERRUPT ROUTINE AT OPERATION DONE>)
BUG(CHK,PI1ERR,<UNEXPECTED UNVECTORED INTERRUPT ON CHANNEL 1>)
BUG(CHK,PI2ERR,<UNEXPECTED UNVECTORED INTERRUPT ON CHANNEL 2>)
BUG(CHK,PI4ERR,<UNEXPECTED UNVECTORED INTERRUPT ON CHANNEL 4>)
BUG(CHK,PI5ERR,<UNEXPECTED UNVECTORED INTERRUPT ON CHANNEL 5>)
BUG(CHK,PI6ERR,<UNEXPECTED UNVECTORED INTERRUPT ON CHANNEL 6>)
BUG(CHK,PIDFLF,<CREPID: FREE PID LIST FOULED UP>)
BUG(CHK,PIDOD1,<MUTCHO: PID COUNT OVERLY DECREMENTED>)
BUG(CHK,PIDOD2,<DELPID: OVERLY DECREMENTED PID COUNT>)
BUG(CHK,PM2SIO,<PHYM2 - ILLEGAL FUNCTION AT START IO>)
BUG(CHK,PSINSK,<PSI FROM NOSKED CONTEXT>)
BUG(CHK,PWRRES,<POWER RESTART>) ;GIVE CHANCE TO LOOK AROUND
BUG(CHK,RELBAD,<RELFRE-BAD BLOCK BEING RELEASED>)
BUG(CHK,RELRNG,<RELFRE: BLOCK OUT OF RANGE>)
BUG(CHK,RESBAD,<RELRES: ILLEGAL ADDRESS PASSED TO RELRES>)
BUG(CHK,RESBAZ,<RELRES: FREE BLOCK RETURNED MORE THAN ONCE>)
BUG(CHK,RESBND,<RELRES: RELEASING SPACE BEYOND END OF RESIDENT FREE POOL>)
BUG(CHK,RFILPF,<REFILL ERROR PAGE FAIL>)
BUG(CHK,RP4SSC,<PHYP4 - STUCK SECTOR COUNTER>,<T1,T2>)
BUG(CHK,SEBISS,<SEBCPY-INSUFFICIENT STRING STORAGE IN BLOCK>)
BUG(CHK,SEBUDT,<SEBCPY-UNKNOWN DATA TYPE>,<T1,T4>)
BUG(CHK,SERFOF,<CAN'T OPENF ERROR REPORT FILE>)
BUG(CHK,SERGOF,<SETOFI-CANNOT GTJFN/OPEN SYSERR FILE>)
BUG(CHK,SNPIC,<SNPFN3: INSTRUCTION BEING REPLACED HAS CHANGED>)
BUG(CHK,SNPLKF,<SNPFNØ: CANNOT LOCK DOWN PAGE INTO MONITOR>)
BUG(CHK,SNPODB,<SNPF4C: COUNT OF INSERTED BREAK POINTS OVERLY DECREMENTED>)
BUG(CHK,SNPUNL,<SNPF5A: CANNOT UNLOCK SNOOP PAGE>)
BUG(CHK,SPWRFL,<SPURIOUS POWER FAIL INDICATION>)
BUG(CHK,SRQOVF,<SCDRQ-SCHED REQUEST QUEUE OVERFLOW>)
BUG(CHK,SUMNR1,<AJBALS-SUMNR INCORRECT>)
BUG(CHK,SUMNR2,<SUMNR INCORRECT>)
BUG(CHK,SWPASF,<CHKBAT-FAILED TO ASSIGN BAD SWAPPING ADDRESS>,<C,CKBDRA>)
BUG(CHK,SWPFPE,<SWAP ERROR IN SENSITIVE FILE PAGE>)
BUG(CHK,SWPIBE,<SWAP ERROR IN INDEX BLOCK>)
BUG(CHK,SWPJSB,<SWAP ERROR IN JSB PAGE>)
BUG(CHK,SYSERF,<LOGSST-NO SYSERR STORAGE FOR RESTART ENTRY>)
BUG(CHK,TM2CCI,<PHYM2 - TMØ2 SSC OR SLA WONT CLEAR>)
BUG(CHK,TM2HER,<TM2ERR - IS.HER SET ON SUCCESSFUL RETRY>)
BUG(CHK,TM2IDM,<PHYM2 - ILLEGAL DATA MODE AT DONE INT>)
BUG(CHK,TM2IF2,<PHYM2 - ILLEGAL FUNCTION ON COMMAND DONE>)
BUG(CHK,TM2NUD,<PHYM2 - CHANNEL DONE INTERRUPT BUT NO UNIT ACTIVE>)
BUG(CHK,TM2RFU,<PHYM2 - ERROR RECOVERY CONFUSED>,<T1,Q1,T3>)
BUG(CHK,TRPSIE,<NO MONITOR FOR TRAPPED FORK>)
BUG(CHK,TTILEC,<TTSND-UNRECOGNIZED ESCAPE CODE>,<2,3>)
BUG(CHK,TTNAC1,<LINE NOT ACTIVE AT PTYOPN>)
BUG(CHK,TTNAC7,<DEALLOCATING INACTIVE LINE>,T2)
BUG(CHK,TTYBBO,<TTYSRV-BIG BUFFER OVERFLOW>)
BUG(CHK,TTYNTB,<RAN OUT OF TTY BUFFERS>)
BUG(CHK,ULKBAD,<UNLOCKING TTY WHEN COUNT IS ZERO>,T2)
BUG(CHK,ULKSTZ,<OVERLY DECREMENTED STRUCTURE LOCK>)
BUG(CHK,UNBFNF,<UNBLK1 - FORK NOT FOUND>)
BUG(CHK,UNPIRX,<UNPIR-NO PSI IN PROGRESS>)
BUG(CHK,UXXCL1,<UNABLE TO CREATE NEW USAGE FILE>)
BUG(CHK,UXXCL2,<UNABLE TO OPEN NEW USAGE FILE>)
BUG(CHK,UXXCL3,<UNABLE TO CLOSE USAGE FILE>)
BUG(CHK,UXXFAI,<USAGE JSYS FAILURE>)
BUG(CHK,UXXWER,<WRITE ERROR IN USAGE FILE>,<T1>)
BUG(CHK,WRTBT4,<ASOFN ON BIT TABLE FILE FAILED>,<T2>)
```

```
BUG(CHK,WRTCPB,<WRTBTB - FAILED TO BACKUP ROOT-DIRECTORY>,<T1>)
BUG(CHK,WSPNEG,<SOSWSP-WSP NEGATIVE>)
BUG(CHK,XBWERR,<UPDOFN-DSK WRITE ERROR ON XB>)
```

```
BUG(INF,CLZDIN,<NETCLZ-COULD NOT SEND DI>)
BUG(INF,DELBDD,<DELDIR: BAD DIRECTORY DELETED. REBUILD BIT TABLE>)
BUG(INF,DLDEF,<LOGICAL NAME DEFINE FAILED FOR FE CTY>)
BUG(INF,DN20ST,<DTESRV- DN20 STOPPED>,<B>)
BUG(INF,DT11DN,<DTECHK- 10 LOST TO11DN INTERRUPT>)
BUG(INF,DTECDM,<DTESRV- TO -10 COUNTS DON'T MATCH>,<A>)
BUG(INF,DTEDIN,<DTESRV- TO -10 IN PROGRESS ON DOORBELL>,<A>)
BUG(INF,DTEDME,<DTESRV- ZERO Q COUNT>,<A>)
BUG(INF,DTELDB,<DTECHK- 11 LOST DOORBELL>)
BUG(INF,DTELPI,<DTECHK- DTE LOST PI ASSIGNMENT>)
BUG(INF,DTEPNR,<DTESRV- INCORRECT INDIRECT SETUP>,<A>)
BUG(INF,DX2IDX,<PHYX2 - ILLEGAL RETRY BYTE POINTER>)
BUG(INF,DX2IRF,<PHYX2 - ILLEGAL FUNCTION DURING RETRY>)
BUG(INF,DX2N2S,<PHYX2 - MORE TU70S THAN TABLE SPACE, EXCESS IGNORED>)
BUG(INF,DX2UNA,<PHYX2 - ATTENTION INTERRUPT AND UDB NOT ACTIVE>)
BUG(INF,ILLSTR,<NSPTSK-ILLEGAL INIT MESSAGE>,<Q1>)
BUG(INF,IMINX1,<UNUSUAL ANI INTERRUPT, CONI ANI IS>,<T1>)
BUG(INF,IMINX2,<IMIERR CALLED, CONI ANI IS>,<T1>)
BUG(INF,IMPABF,<ASNTBF FAILED>)
BUG(INF,IMPBSC,<MESSAGE HAS BAD SIZE OR COUNT>,<T1,T2>)
BUG(INF,IMPCTH,<IMPNCL TOO HIGH>)
BUG(INF,IMPCUL,<RECD CTL MSG FOR UNKNOWN LINK>,<T1,T2,T3>)
BUG(INF,IMPHIF,<HSTINI FAILED TO FIND HOST NAME FILE>)
BUG(INF,IMPIFC,<ILL FMT CTL MSG>,<T2,T3>)
BUG(INF,IMPLAE,<IMPOPL: LINK ALREADY EXISTS>,T2)
BUG(INF,IMPLEO,<CAN'T FIND LT ENTRY FOR OUTPUT MESSAGE>,<T1,T2>)
BUG(INF,IMPMSO,<MESSAGE STUCK IN OUTPUT QUEUE>,T2)
BUG(INF,IMPMUL,<RECEIVED MSG FOR UNKNOWN LINK>,<T1,T2>)
BUG(INF,IMPNEA,<NVT RECEIVED BYTES EXCEEDING ALLOCATION>)
BUG(INF,IMPOFL,<MESSAGE BUFFER OVERFLOW>,<T1,T2,T3,T4>)
BUG(INF,IMPREA,<RECD EXCESS ALL>,T2)
BUG(INF,IMPRNO,<RFNM OVERDUE>,T2)
BUG(INF,IMPRNE,<RECD NCP ERR>,<T1,T2>)
BUG(INF,IMPXBO,<IRREG MSG BUFFER OVERFLOW>)
BUG(INF,IMPXUT,<RECEIVED IRREG MSG WITH UNKNOWN LINK OR TYPE>,<T1,T2,T3>)
BUG(INF,INDCNT,<DTESRV- BAD INDIRECT COUNT>)
BUG(INF,NCPFUN,<NCP FSM RECIEVED FUNNY INPUT>,<T1,T2,UNIT>)
BUG(INF,OVRDTA,<PHYSIO - OVERDUE TRANSFER ABORTED>,<T1,T3,T2>)
BUG(INF,P1NED3,<PHYH11 - NON EX DISK ON DONE OR ATN INTERRUPT>,<T1,T2>)
BUG(INF,PH2DNA,<PHYH2 - DONE INTERRUPT AND CHANNEL NOT ACTIVE>,<T2>)
BUG(INF,PHYCH2,<PHYSIO - HOME BLOCK CHECK IORB TIMED OUT>)
BUG(INF,PHYCH3,<PHYSIO - HOME BLOCK CHECK IORB TIMED OUT BUT WAS NOT ON TWQ>)
BUG(INF,PHYICE,<PHYINI - FAILED TO ASSIGN RESIDENT STG>)
BUG(INF,SBSERF,<SBSERR-COULD NOT GET ERROR BLOCK>)
BUG(INF,TM2IDX,<PHYM2 - ILLEGAL RETRY BYTE POINTER>)
BUG(INF,TM2IRF,<PHYM2 - ILLEGAL FUNCTION DURING RETRY>)
BUG(INF,TM2N2S,<PHYM2 - MORE DRIVES THAN TABLE SPACE, EXCESS IGNORED>)
BUG(INF,TM2UNA,<PHYM2 - DONE INTERRUPT AND UDB NOT ACTIVE>,<Q1,P3,T1>)
BUG(INF,TTYPI2,<SCANNER LOST PI ASSIGNMENT, COULD NOT RESTORE>)
BUG(INF,TTYPI1,<SCANNER LOST PI ASSIGNMENT, SUCCESSFULLY RESTORED>)
BUG(INF,USGHOL,<LOST PAGE(S) IN USAGE FILE>)
BUG(INF,UXXFIT,<CHECKPOINT FILE NOT IN CORRECT FORMAT FOR THIS SYSTEM, REBUILDING...>)
```

Appendix I

PART B

List of BUGHLTs, BUGCHKs and BUGINFs by Module

ANNBIG.MAC.2

ANNLGE.MAC.1

ANNMED.MAC.2

ANNSML.MAC.2

ANPBIG.MAC.6

ANPLGE.MAC.4

ANPMED.MAC.7

ANPSML.MAC.5

APRSRV.MAC.157

```
BUG(HLT,LUUMNØ,<LUUO IN MONITOR CONTEXT>)
BUG(HLT,FATCDP,<FATAL CACHE DIRECTORY PARITY ERROR>,<A>)
BUG(HLT,FATAPE,<FATAL ADDRESS PARITY ERROR>,<A>)
BUG(HLT,APRNX1,<NXM DETECTED BY APR>,<A>)
BUG(HLT,APRNX1,<NXM DETECTED BY APR>)
BUG(HLT,STKOVF,<MONITOR STACK OVERFLOW>)
BUG(HLT,PWRFL,<FATAL POWER FAILURE>) ;CRASH AND RELOAD
BUG(HLT,IOPGF,<IO PAGE FAIL>,<Q1>)
BUG(HLT,KPALVH,<KEEP ALIVE CEASED>)
BUG(HLT,FATMER,<FATAL MEMORY ERROR>)
BUG(HLT,UNPGF1,<MEMPAR-PARITY ERROR DURING MEM SCAN>)
BUG(HLT,SMNOFR,<NO FREE SPACE FOR SM1Ø VECTORS>)
BUG(HLT,LUUMON,<.LBCHK: ILLEGAL LUUO FROM MONITOR CONTEXT>)
BUG(HLT,IMPUUO,<IMPOSSIBLE MUUO>)
BUG(HLT,SKDPF1,<PAGE FAIL IN SCHED CONTEXT>) ;NO
BUG(HLT,UNTRAP,<UNKNOWN TRAP INSTRUCTION>)
BUG(HLT,MONPDL,<OVERFLOW OR PDL OVERFLOW TRAP IN MONITOR>)
BUG(HLT,PTNIC1,<SWPIN - PAGE TABLE NOT IN CORE>)
BUG(HLT,UNPGF2,<UNKNOWN PAGE FAILURE TYPE>)
BUG(HLT,IONXM,<I/O NXM ON UNIBUS DEVICE>)
BUG(HLT,PVTRP,<PROPRIETARY VIOLATION TRAP>)
BUG(HLT,PTMPE,<PAGE TABLE PARITY ERROR>)
BUG(HLT,SKDMPE,<MPE IN SCHEDULER OR PI CONTEXT>)
BUG(HLT,APRNX2,<NXM DETECTED BY APR>) ;YES
BUG(HLT,NOSEB2,<PGMPE-NO SYSERR BUFFER AVAILABLE>)
BUG(HLT,UNXMPE,<PFCDPE-UNEXPECTED PARITY ERROR TRAP>)
BUG(HLT,MPEUTP,<PFCDPE-UNKNOWN TRAP ON TEST REFERENCE>)
BUG(HLT,PRONX2,<NXM DETECTED BY PROCESSOR>)
BUG(HLT,PFCDP,<MEMORY PARITY ERROR>)

BUG(CHK,PI1ERR,<UNEXPECTED UNVECTORED INTERRUPT ON CHANNEL 1>)
BUG(CHK,PI2ERR,<UNEXPECTED UNVECTORED INTERRUPT ON CHANNEL 2>)
BUG(CHK,PI4ERR,<UNEXPECTED UNVECTORED INTERRUPT ON CHANNEL 4>)
```

```
BUG(CHK,SPWRFL,<SPURIOUS POWER FAIL INDICATION>)
BUG(CHK,PWRRES,<POWER RESTART>) ;GIVE CHANCE TO LOOK AROUND
;BUG(CHK,MPEDEV,<MEMORY PARITY ERROR DETECTED BY APR OR DEVICE>)
BUG(CHK,ILLUUO,<KIBADU: ILLEGAL UUO FROM MONITOR CONTEXT>,<KIMUFL,KIMUPC,KIMUEF>)
BUG(CHK,ILLDMS,<BADDMS: ILLEGAL DMS JSYS FROM MONITOR CONTEXT>)
BUG(CHK,RFILPF,<REFILL ERROR PAGE FAIL>)
BUG(CHK,ILPSEC,<ILLEGAL SECTION NUMBER>,<TRAPPC,TRAPSW>)
BUG(CHK,HARDCE,<HARD CACHE ERRORS--CACHE DESELECTED>)

BUG(INF,SBSERF,<SBSERR-COULD NOT GET ERROR BLOCK>)

ARPAF.MAC.3


BOOT.MAC.39


CDKLDV.MAC.8


CDKSDV.MAC.57

BUG(HLT,NCDWA,<KSINI: NO CARDREADER UBA WINDOW>)

BUG(CHK,CDBDIN,<CD11 LOST INTERRUPT ENABLE>,T4)

CDRSRV.MAC.3

BUG(HLT,CDILVT,ILLEGAL DEVICE TYPE)

COMND.MAC.12


DATIME.MAC.54


DDT.MAC.16


DDTBLT.MAC.3


DDTU.MAC.4


DEVICE.MAC.96

BUG(HLT,TTNAC8,<CAN'T ASSIGN TERMINAL AT DEVINI>)
BUG(HLT,NOFNDU,<FNDUNT-CAN'T FIND DEVICE FOR JFN>)

BUG(CHK,DEVUCF,<DEVAV - UNEXPECTED CHKDES FAILURE>)

DIAG.MAC.8

BUG(HLT,DGZTPA,<DIAG - LOCKED PAGE LIST PAGE WAS ZERO>)
BUG(HLT,DGUTPG,<DIAG - LOCKED PAGE LIST PAGE LOCKED AT DIAG UNLOCK>)

DIRECT.MAC.4
```

```
BUG(HLT,BADDAC,<INSACT - NULL ACCOUNT STRING SEEN>)
BUG(HLT,LCKDIR,<ATTEMPT TO LOCK DIRECTORY TWICE FOR SAME FORK>)

BUG(CHK,DIRSY1,<DELDL8: DIRECTORY SYMBOL TABLE FOULED UP FOR DIRECTORY:>,<A,B>)
BUG(CHK,DIRBAD,<SETDI4: SMASHED DIRECTORY NUMBER:>,<A,SETDNM>)
BUG(CHK,DIRFKP,<SETDIR-DIR PAGE 0 BELONGS TO FORK IN DIRECTORY:>,<B,SETDNM>)
BUG(CHK,DIRULK,<ULKMD2: ATTEMPT TO UNLOCK ILLEGALLY FORMATTED DIR, DIR NUMBER:>,<T1,T2>)
BUG(CHK,DIRDNL,<ULKDIR-DIRECTORY NOT LOCKED, DIRECTORY NUMBER:>,<T1,T2>)
BUG(CHK,LNGDIR,<LONG DIRECTORY FILE IN DIRECTORY:>,<T3>)
BUG(CHK,DIRSY2,<MDDNAM: SYMBOL TABLE FOULED UP IN DIRECTORY:>,<A,B>)
BUG(CHK,DIRSY3,<LOOKUP: SYMBOL SEARCH FOULED UP IN DIRECTORY:>,<C,B>)
BUG(CHK,DIRSY4,<NAMCM4: DIRECTORY SYMBOL TABLE FOULED UP IN DIRECTORY:>,<A,B>)
BUG(CHK,MPIDXO,<MAPIDX - No OFN for Index Table File>)
BUG(CHK,DIRPG0,<DR0CHK: ILLEGAL FORMAT FOR DIRECTORY PAGE 0 IN DIRECTORY:>,<A,B>)
BUG(CHK,DIRPG1,<DRHCHK: DIRECTORY HEADER BLOCK IS BAD IN DIRECTORY:>,<A,B>)
BUG(CHK,DIRSY5,<SYMBAD: ILLEGAL FORMAT FOR DIRECTORY SYMBOL TABLE IN DIRECTORY:>,<A,B>)
BUG(CHK,DIRFDB,<ILLEGAL FORMAT FOR FDB IN DIRECTORY:>,<A,B>)
BUG(CHK,DIRNAM,<NAMBAD: ILLEGAL FORMAT FOR DIRECTORY NAME BLOCK IN DIRECTORY:>,<A,B>)
BUG(CHK,DIREXT,<EXTBAD: ILLEGAL FORMAT FOR DIRECTORY EXTENSION BLOCK IN DIRECTORY:>,<A,B>)
BUG(CHK,DIRACT,<ACTBAD: ILLEGAL FORMAT FOR DIRECTORY ACCOUNT BLOCK IN DIRECTORY:>,<A,B>)
BUG(CHK,DIRFRE,<FREBAD: ILLEGAL FORMAT FOR DIRECTORY FREE BLOCK IN DIRECTORY:>,<A,B>)
BUG(CHK,DIRUNS,<UNSBAD: ILLEGAL FORMAT FOR DIRECTORY USER NAME BLOCK IN DIRECTORY:>,<A,B>)
BUG(CHK,DIRSY6,<RBLDST: PREMATURELY RAN OUT OF ROOM IN SYMBOL TABLE IN DIRECTORY:>,<A,B>)
BUG(CHK,DIRBLK,<BLKSCN: ILLEGAL BLOCK TYPE IN DIRECTORY:>,<A>)
BUG(CHK,DIRB2S,<RLDFB1: DIRECTORY FREE BLOCK TOO SMALL IN DIRECTORY:>,<A,B>)
BUG(CHK,DIRB2L,<RLDFB2: DIRECTORY FREE BLOCK TOO LARGE IN DIRECTORY:>,<A,B>)
BUG(CHK,DIRBCB,<RLDFB3: DIRECTORY FREE BLOCK CROSSES PAGE BOUNDARY IN DIRECTORY:>,<A,B>)
BUG(CHK,DIRIFB,<RLDFB4: ILLEGAL BLOCK TYPE ON DIRECTORY FREE LIST IN DIRECTORY:>,<A,B>)
BUG(CHK,DIRBAF,<RLDFB5: BLOCK ALREADY ON DIRECTORY FREE LIST IN DIRECTORY:>,<A,B>)
BUG(CHK,DIRRHB,<RLDFB6: ATTEMPTING TO RETURN A HEADER BLOCK IN DIRECTORY:>,<A,B>)

DISC.MAC.7

BUG(HLT,NOPGT0,<OPNLNG: NO PAGE TABLE 0 IN LONG FILE.>)
BUG(HLT,GTFDB2,<NEWLFP: GETFDB FAILURE FOR OPEN FILE.>)
BUG(HLT,DNOPT0,<DSKCLZ-JFNOFN FAILED FOR PAGE 0>)
BUG(HLT,ASOFNF,<DELFIL: ASOFN GAVE FAIL RETURN FOR LONG FILE XB>)
BUG(HLT,NOLEN,<UPDLEN: NO LENGTH INFO FOR OFN>)
BUG(HLT,GTFDB3,<DSKREN-GETFDB FAILURE FOR OPEN FILE>)

BUG(CHK,GTFDB1,<DSKINS: GETFDB FAILURE.>)
BUG(CHK,NOUTF1,<SPLOPN: NOUT OF DIRECTORY NUMBER FAILED>)

DSKALC.MAC.11

BUG(HLT,WRTLNG,<WRTBTB - BIT TABLE IS A LONG FILE>)
BUG(HLT,MNTLNG,<MNTBTB - BIT TABLE IS A LONG FILE>)
BUG(HLT,MAPBT1,<OFN FOR BIT TABLE IS ZERO>)
BUG(HLT,TTNAC5,<CTY NOT ACTIVE AT FSIINI>)
BUG(HLT,TTNAC4,<CTY NOT ACTIVE AT FSIPBI>)
BUG(HLT,TTNAC3,<CTY NOT ACTIVE AT FSIPBO>)

BUG(CHK,DSKBT1,<DSK BIT TABLE FOULED, CAN'T FIND FREE PAGE ON TRACK WITH NON-0 COUNT>,<T2,T3>)
BUG(CHK,ASAASG,<DSKASA - ASSIGNING ALREADY ASSIGNED DISK ADDRESS>,<T1,T2>)
BUG(CHK,ASGBAD,<DSKASA - ASSIGNING BAD DISK ADDRESS>,<T3,T2>)
BUG(CHK,DEAUNA,<DEDSK-DEASSIGNING UNASSIGNED DISK ADDRESS>,<T1,T2>)
BUG(CHK,DEABAD,<DSKDEA - DEASSIGNING BAD DISK ADDRESS>,<T3,T2>)
BUG(CHK,ASGBPG,<INIBTB-FAILED TO ASSIGN BAD PAGE(S)>,<T1,T2>)
BUG(CHK,WRTCPB,<WRTBTB - FAILED TO BACKUP ROOT-DIRECTORY>,<T1>)
BUG(CHK,WRTBT4,<ASOFN ON BIT TABLE FILE FAILED>,<T2>)
```

```
BUG(CHK,DSKBT3,<DISK BIT TABLE ALREADY LOCKED AT LCKBTB>,<T1>)
;BUG(CHK,NO2PRT,<FEFSYS-NO DUAL-PORTED DISK. USING LOGICAL 0>)
BUG(CHK,FEOCPB,<FEFSYS - FAILED TO BACKUP ROOT-DIRECTORY>,<T1>)
BUG(CHK,BADBAT,<BAT BLOCKS UNREADABLE>)
BUG(CHK,SWPASF,<CHKBAT-FAILED TO ASSIGN BAD SWAPPING ADDRESS>,<C,CKBDRA>)
BUG(CHK,NOBAT1,<FAILED TO WRITE PRIMARY BAT BLOCK>,<T1,T2>)
BUG(CHK,NOBAT2,<FAILED TO WRITE SECONDARY BAT BLOCK>,<T1,T2>)

DTESM.MAC.18


DTESRV.MAC.24

BUG(HLT,NORSXF,<FAILED TO GET SPACE FOR MASTER DTE>)
BUG(HLT,DTEIDP,<DTESRV- INDIRECT POINTER WITH GARBAGE PACKET>)
BUG(HLT,DTEIFR,<DTESRV-ILLEGAL FUNCTION REQUEST FROM 11>)
BUG(HLT,DTEUIF,<DTESRV-UNIMPLEMENTED FUNCTION FROM 11>)
BUG(HLT,DTETTY,<TAKLC-NON-TTY DEVICE ON FUNCTION CODE 4>)
BUG(HLT,DTECAR,<DTESRV- CARRIER FUNCTION WITH NO LINE NUMBER PRESENT>)
BUG(HLT,DTEDEV,<LINEAL -ILLEGAL DEVICE>)
BUG(HLT,DTEMCC,<DOFRGM-MCB DISAGREES WITH COUNT>) ;NO
BUG(HLT,INVDTE,<DTEQ- INVALID DTE SPECIFIED>)

BUG(CHK,DTETIP,<DTETDN-TO10 DONE RECEIVED WITH NO TRANSFER IN PROGRESS>)
BUG(CHK,DTEP2S,<TO10DN-PACKET TOO SMALL>)
BUG(CHK,DTEODD,<TAKLC-ODD BYTE COUNT FOR LINE CHARACTERS>)
BUG(CHK,DTEDAT,<TAKTOD- ILLEGAL FORMAT FOR TIME/DATE>)
BUG(CHK,KLIOVF,<DTESRV-KLINIK DATA BASE TOO LARGE>,<C>)
BUG(CHK,DTEERR,<DTESRV-DTE DEVICE ERROR>,<A,F>)
BUG(CHK,DTEPGF,<DTE TRANSFER PAGE FAIL>,<A>)

BUG(INF,INDCNT,<DTESRV- BAD INDIRECT COUNT>)
BUG(INF,DTECDM,<DTESRV- TO -10 COUNTS DON'T MATCH>,<A>)
BUG(INF,DTEDIN,<DTESRV- TO -10 IN PROGRESS ON DOORBELL>,<A>)
BUG(INF,DTEDME,<DTESRV- ZERO Q COUNT>,<A>)
BUG(INF,DTEPNR,<DTESRV- INCORRECT INDIRECT SETUP>,<A>)
BUG(INF,DTELPI,<DTECHK- DTE LOST PI ASSIGNMENT>)
BUG(INF,DTELDB,<DTECHK- 11 LOST DOORBELL>)
BUG(INF,DT11DN,<DTECHK- 10 LOST TO11DN INTERRUPT>)
BUG(INF,DN20ST,<DTESRV- DN20 STOPPED>,<B>)

DUPSRV.MAC.26

BUG(HLT,DUPUBA,<no Unibus Address>)
BUG(HLT,DUPCOR,<No core for DUP11>)
BUG(HLT,KMCIII,<KMC11 illegal input interrupt>,<T1,T2>)

BUG(CHK,KMCNTI,<KMC11 not taking input>)
BUG(CHK,BADHDR,<bad DDCMP header>,<T1,T2>)
BUG(CHK,MSGCLB,<DDCMP transmit message clobbered>)
BUG(CHK,MSGPTR,<Bad msg pointer>)

EDDT.MAC.8


ENQ.MAC.77

BUG(CHK,ENQMLF,<ENQUE: INTERNAL ENQ OF A MONITOR LOCK FAILED>)
BUG(CHK,DEQMDF,<DEQUE: INTERNAL MONITOR DEQ FAILED>)
```

EXPRE.MAC.6

FDDT.MAC.4

FESRV.MAC.3

```
BUG(CHK,FEUSTS,<FESSTS-UNKNOWN STATUS>)
BUG(CHK,FEBAD,<FEHSD-WRONG FE>) ;NO
BUG(CHK,FEBFOV,<FEHSD-BUFFER OVERFLOW>,<A,C>) ;NO
```

FILDDT.MAC.6

FILINI.MAC.7

```
BUG(HLT,BADREC,<FILINI - Reconstruction of ROOT-DIRECTORY failed>)
BUG(HLT,IBOFNF,<FILINI: ASOFN FAILURE FOR ROOT DIRECTORY IB>)
BUG(HLT,BTBCR1,<FILINI - NO BIT TABLE FILE AND UNABLE TO CREATE ONE>)
BUG(HLT,BADXT1,<INDEX TABLE MISSING AND CAN NOT BE CREATED>)
BUG(HLT,BTBCRT,<FILINI - COULD NOT INITIALIZE BIT TABLE FOR PUBLIC STRUCTURE>)
BUG(HLT,NEWROT,<FILRFS - NEWIB FAILURE FOR ROOT-DIRECTORY>)
BUG(HLT,NEWBAK,<FILRFS - NEWIB FAILURE FOR BACKUP ROOT-DIR>)
BUG(HLT,IDXNOS,<FILINI - COULD NOT ASSIGN FREE SPACE FOR IDXTAB>)
BUG(HLT,FILRID,<FILINW: INDEX TABLE ALREADY SET UP FOR ROOT DIR>)
BUG(HLT,FILIRD,<FILINW: COULD NOT INITIALIZE THE ROOT DIRECTORY>)
BUG(HLT,FILBTB,<UNABLE TO WRITE BIT TABLE FILE>)
BUG(HLT,NOFEFS,<FILINI - UNABLE TO GET SIZE OF FRONT END FILE SYSTEM>)
BUG(HLT,NOBTBN,<FILINI - UNABLE TO GET SIZE OF BOOTSTRAP.BIN FILE>)
BUG(HLT,FILMAP,<FILIN2: COULD NOT MAP IN ROOT-DIRECTORY>)
BUG(HLT,BADROT,<FILIN2: ROOT-DIRECTORY IS INVALID>)
BUG(HLT,STZERO,<FILINI: STRTAB ENTRY FOR PS IS 0>)
BUG(HLT,BADXTB,<FILIN2: Could not initialize index table>)

BUG(CHK,NOBTB,<FILINI - UNABLE TO OPEN BIT TABLE FILE>)
BUG(CHK,BADXT2,<INDEX TABLE MISSING AND WAS CREATED>)
BUG(CHK,FILHOM,<UNABLE TO REWRITE HOME BLOCKS IN WRTBTB>)
BUG(CHK,FILFEF,<Could not create Front End File System>)
BUG(CHK,FILBOT,<COULD NOT CREATE BOOTSTRAP.BIN FILE>)
BUG(CHK,BADBAK,<FILIN2 - BACKUP COPY OF ROOT DIRECTORY IS NOT GOOD>)
BUG(CHK,FIXBAD,<Could not re-write Home Blocks to point to FE Filesystem>)
BUG(CHK,FIXBDB,<COULD NOT RE-WRITE HOME BLOCKS TO POINT TO BOOTSTRAP.BIN>)
BUG(CHK,FILJB1,<FILCRD: No room to create standard system directories>)
BUG(CHK,FILCCD,<Could not create directory>)
BUG(CHK,FILBAK,<FILCRD: COULD NOT CREATE BACKUP OF ROOT-DIR>)
BUG(CHK,BADIDX,<IDXINI: PARTIALLY UNSUCCESSFUL INDEX TABLE REBUILD>)
```

FILMSC.MAC.4

```
BUG(CHK,TTNAC1,<LINE NOT ACTIVE AT PTYOPN>)
```

FORK.MAC.8

```
BUG(HLT,FRKSLF,<SUSFK - GIVEN SELF AS ARG>)
BUG(HLT,MAP41F,<MAPF41 FAILED TO SKIP>)
BUG(HLT,FRKNPT,<FKHPTN - FORK HAS NO PAGE TABLE>)

BUG(CHK,ILJRFN,<JFKRFH - BAD JRFN, IGNORED>)
BUG(CHK,FLKINT,<FLOCK-CALLED WHILE NOINT>)
```

```
BUG(CHK,FLKTIM,<FLOCK-TIMEOUT>)
BUG(CHK,FLKNS,<FUNLK-LOCK NOT SET>)
;BUG(CHK,NOXRFH,<DASFKH - ATTEMPT TO DEASSIGN NONEXISTANT RFH, IGNORED>)
BUG(CHK,NWJTBE,<NO FREE JTB BLOCKS>)

FREE.MAC.8

BUG(CHK,RELRNG,<RELFRE: BLOCK OUT OF RANGE>)
BUG(CHK,RELBAD,<RELFRE-BAD BLOCK BEING RELEASED>)
BUG(CHK,ASGREQ,<ILLEGAL POOL NUMBER GIVEN TO ASGRES>)
BUG(CHK,ASGREP,<ILLEGAL PRIORITY GIVEN TO ASGRES>)
BUG(CHK,RESBAD,<RELRES: ILLEGAL ADDRESS PASSED TO RELRES>)
BUG(CHK,RESBND,<RELRES: RELEASING SPACE BEYOND END OF RESIDENT FREE POOL>)
BUG(CHK,RESBAZ,<RELRES: FREE BLOCK RETURNED MORE THAN ONCE>)

FUTILI.MAC.91

BUG(CHK,ULKSTZ,<OVERLY DECREMENTED STRUCTURE LOCK>)

GLOBS.MAC.93


GTJFN.MAC.3

BUG(CHK,NOSPLM,<RELJFN: COULD NOT SEND SPOOL MESSAGE TO QUASAR>)

IMPANX.MAC.10

BUG(HLT,IMPNII,<NO IMP INPUT BUFFERS>)
BUG(HLT,IMPAUF,<IMPEIN: BUFFER ON FREELIST USED FOR INPUT>)

BUG(CHK,IMPHNW,<LHOSTN DISAGREES WITH THE IMP>)

BUG(INF,IMINX1,<UNUSUAL ANI INTERRUPT, CONI ANI IS>,<T1>)
BUG(INF,IMPOFL,<MESSAGE BUFFER OVERFLOW>,<T1,T2,T3,T4>)
BUG(INF,IMINX2,<IMIERR CALLED, CONI ANI IS>,<T1>)

IMPDV.MAC.5

BUG(HLT,IMPCCF,<CAN'T CREATE IMP FORK>)
BUG(HLT,IMPUFB,<IMIP1: ATTEMPT TO UNLOCK BUFFER ON FREELIST>)
BUG(HLT,IMPRMI,<IMP - REGULAR MESSAGE ON IRREG QUEUE>)
BUG(HLT,IMPNBC,<PKMSG: NEGATIVE RESIDUAL BYTE COUNT>)
BUG(HLT,IMPALF,<IMPLKB: ATTEMPT TO LOCK BUFFER ON FREELIST>)
BUG(HLT,IMPAFB,<IMPCQ: ATTEMPT TO UNLOCK BUFFER ON FREELIST>)
BUG(HLT,IMPUBF,<IMULKB: ATTEMPT TO UNLOCK BUFFER ON FREELIST>)

BUG(CHK,IMPUX0,<IMP JB0 FORK - UNEXPECTED INTERRUPT>)
BUG(CHK,IMPLTF,<IMPLT FULL>)
BUG(CHK,IMPIFH,<IMPGC-IMPOSSIBLE FAILURE OF IMPHFL>)
BUG(CHK,IMPREM,<UPBRB: RECEIVED EXCESSIVE MESSAGES>,T2)
BUG(CHK,IMPMSL,<PKMSG - MSG TOO LARGE>)
BUG(CHK,IMPNMA,<PKBY1: NO MSG ALLOCATION>,T2)

BUG(INF,IMPHIF,<HSTINI FAILED TO FIND HOST NAME FILE>)
BUG(INF,IMPMUL,<RECEIVED MSG FOR UNKNOWN LINK>,<T1,T2>)
BUG(INF,IMPRNO,<RFNM OVERDUE>,T2)
BUG(INF,IMPMSO,<MESSAGE STUCK IN OUTPUT QUEUE>,T2)
BUG(INF,IMPXBO,<IRREG MSG BUFFER OVERFLOW>)
BUG(INF,IMPXUT,<RECEIVED IRREG MSG WITH UNKNOWN LINK OR TYPE>,<T1,T2,T3>)
```

```
BUG(INF,IMPCTH,<IMPNCL TOO HIGH>)
BUG(INF,IMPIFC,<ILL FMT CTL MSG>,<T2,T3>)
BUG(INF,IMPREA,<RECD EXCESS ALL>,T2)
BUG(INF,IMPRNE,<RECD NCP ERR>,<T1,T2>)
BUG(INF,IMPCUL,<RECD CTL MSG FOR UNKNOWN LINK>,<T1,T2,T3>)
BUG(INF,IMPLAE,<IMPOPL: LINK ALREADY EXISTS>,T2)
BUG(INF,IMPBSC,<MESSAGE HAS BAD SIZE OR COUNT>,<T1,T2>)
BUG(INF,IMPLEO,<CAN'T FIND LT ENTRY FOR OUTPUT MESSAGE>,<T1,T2>)
BUG(INF,IMPABF,<ASNTBF FAILED>)

IO.MAC.7

BUG(CHK,BLKF1,<BYTINA: BLKF SET BEFORE CALLING SERVICE ROUTINE>)
BUG(CHK,BLKF2,<BYTOUA: BLKF SET BEFORE CALL TO SERVICE ROUTINE>)
BUG(CHK,DMPRLF,<DMPREL-FAILED TO RELEASE PAGE>)

IPCF.MAC.133

BUG(HLT,IPCOVL,<PIDINI: PIDS AND FREE POOL OVERLAP, IPCF WON'T WORK!>)

BUG(CHK,IPCMCN,<MESREC: MESSAGE COUNT WENT NEGATIVE>)
BUG(CHK,PIDOD1,<MUTCHO: PID COUNT OVERLY DECREMENTED>)
BUG(CHK,IPCFKH,<CHKPDD: COULD NOT FIND LOCAL FORK HANDLE>)
BUG(CHK,PIDFLF,<CREPID: FREE PID LIST FOULED UP>)
BUG(CHK,ILPID1,<CREPID: ATTEMPT TO CREATE ILLEGAL PID>)
BUG(CHK,PIDOD2,<DELPID: OVERLY DECREMENTED PID COUNT>)
BUG(CHK,ILPID2,<DELPID: VALIDATED PID TURNED ILLEGAL>)
BUG(CHK,IPCSOD,<GETMES: SENDER'S COUNT OVERLY DECREMENTED>)
BUG(CHK,NOPID,<PIDKFL: PID DISAPPEARED>)
BUG(CHK,NODIR1,<SPLMES: DIRST FAILED ON EXISTING DIRECTORY NAME>)
BUG(CHK,NOUTF2,<SPLMES: NOUT OF GENERATION NUMBER FAILED>)
BUG(CHK,NOALCM,<ALCMES: CANNOT SEND MESSAGE TO ALLOCATOR>)
BUG(CHK,IPCJBØ,<PIDINI: NOT IN CONTEXT OF JOB Ø>)
BUG(CHK,IPCFRK,<PIDINB: CANNOT CREATE FORKS FOR IPCF>)

JSYSA.MAC.25

BUG(CHK,CPYUF1,<CACCT: IMPOSSIBLE FAILURE OF CPYFU1.>)
BUG(CHK,EFACF3,<EFACT: FAILED TO WRITE INTO FACT FILE>)
BUG(CHK,EFACF1,<EFACT: CLOSF FAILED TO CLOSE FACT FILE.>)
BUG(CHK,SNPLKF,<SNPFNØ: CANNOT LOCK DOWN PAGE INTO MONITOR>)
BUG(CHK,SNPIC,<SNPFN3: INSTRUCTION BEING REPLACED HAS CHANGED>)
BUG(CHK,SNPODB,<SNPF4C: COUNT OF INSERTED BREAK POINTS OVERLY DECREMENTED>)
BUG(CHK,SNPUNL,<SNPF5A: CANNOT UNLOCK SNOOP PAGE>)
BUG(CHK,HSHERR,<VERACT - HASH VALUE OUT OF RANGE>)
BUG(CHK,CRSPAG,<VERACT - ACCOUNT DATA BLOCK CROSSES A PAGE BOUNDARY>)
BUG(CHK,BADTAB,<VERACT - SPURIOUS HASH TABLE ENCOUNTERED>)

JSYSF.MAC.341

BUG(HLT,GTFDB6,<CRDIØA: CANNOT DO GETFDB ON ROOT-DIRECTORY >)

BUG(CHK,CLZABF,<CLZFFW: SERVICE ROUTINE BLOCKED ON AN ABORT CLOSE>)
BUG(CHK,BLKF3,<CLZDO: BLKF SET BEFORE CALL TO SERVICE ROUTINE>)
BUG(CHK,CRDSDF,<CRDIR1: SETDIR FAILED ON NEW DIRECTORY>)
BUG(CHK,CRDBAK,<CRDIR3: COULD NOT MAKE BACKUP COPY OF ROOT-DIRECTORY>)
BUG(CHK,CRDOLD,<CRGDGB: OLD FORMAT CRDIR IS ILLEGAL>)
BUG(CHK,CRDNOM,<CRDIR-FAILED TO MAKE MAIL.TXT FILE>)
BUG(CHK,CRDBK1,<CRDIR4:COULD NOT MAKE BACKUP COPY OF ROOT-DIRECTORY>)
BUG(CHK,DVCHRX,<DVCHR1 - UNEXPECTED CHKDES FAILURE WITHIN .DVCHR>)
```

```
BUG(CHK,BLKF4,<.GDSTS: BLKF SET BEFORE CALL TO DEVICE ROUTINE>)
BUG(CHK,BLKF5,<.MTOPR: BLKF SET BEFORE CALL TO DEVICE ROUTINE>)
BUG(CHK,BLKF6,<.SDSTS: BLKF SET BEFORE CALL TO DEVICE ROUTINE>)

BUG(INF,DELBDD,<DELDIR: BAD DIRECTORY DELETED. REBUILD BIT TABLE>)

KDPSRV.MAC.29

BUG(HLT,DUPUBA,<no Unibus Address>)
BUG(HLT,DUPCOR,<No core for DUP11>)
BUG(HLT,KMCIII,<KMC11 illegal input interrupt>,<T1,T2>)

BUG(CHK,KMCNTI,<KMC11 not taking input>)
BUG(CHK,BADHDR,<bad DDCMP header>,<T1,T2>)
BUG(CHK,MSGCLB,<DDCMP transmit message clobbered>)
BUG(CHK,MSGPTR,<Bad msg pointer>)

KLBPRE.MAC.1


KLPRE.MAC.1


KSPRE.MAC.2


LDINIT.MAC.73


LINEPR.MAC.19


LOGNAM.MAC.66

BUG(CHK,NOSLNM,<SLNINI: CANNOT CREATE SYSTEM LOGICAL NAME>)
BUG(CHK,LNMILI,<LNMLUK: ILLEGAL VALUE OF LOGICAL NAME TABLE INDEX>)
BUG(CHK,ILLTAB,<TABLK2: TABLE NOT IN PROPER FORMAT>)

LOOKUP.MAC.1

BUG(HLT,ASTJFN,<GETFDB: CALLED FOR JFN WITH OUTPUT STARS>)
BUG(HLT,MDDJFN,<GETFDB: CALLED FOR NON-MDD DEVICE>)
BUG(HLT,ILSTP3,<VERLUK: IMPOSSIBLE SKIP RETURN FROM EXTLUU>)

LPFEDM.MAC.6


LPFEDV.MAC.1


LPKSDV.MAC.75

BUG(HLT,NLWA,<L2INI:  No lineprinter window available>)

BUG(CHK,LP2IEN,<LINEPRINTER LOST INTERRUPT ENABLE>,U)

MAGTAP.MAC.10

BUG(HLT,MTARIN,<MTAINT: INTERRUPT RECEIVED FOR NONACTIVE IORB>)
```

```
BUG(CHK,MTANOI,<GETUBF: NO QUEUED IORB'S FOR INPUT>)
BUG(CHK,MTANOQ,<IRBDN1: IRBDON CALLED FOR NON-QUEUED UP IORB>)
BUG(CHK,MTANOA,<IRBDN2: IRBDON CALLED FOR AN ACTIVE IORB>)
BUG(CHK,MTAORN,<MTDIR0: MAGTAPE IORB OVERRUN>)

MDDT.MAC.7


MEXEC.MAC.36

BUG(HLT,CKDFRK,<JOB 0 CFORK FAILED>)
BUG(HLT,ILBOOT,<GETSWM-ILLEGAL VALUE OF BOOTFL>)
BUG(HLT,BOOTCR,<GETSWM - NOT ENOUGH CORE FOR SWPMON>)
BUG(HLT,BOOTMP,<GSMDSK - CANNOT MAP BOOTSTRAP PAGES>)
BUG(HLT,BOOTLK,<GSMDSK - FAILED TO LOCK NEEDED PAGES>)
BUG(HLT,BOOTER,<GETSWM - ERROR LOADING SWPMON>)
BUG(HLT,HSYFRK,<HSYS-JOB 0 CFORK FAILED>)
BUG(HLT,EXPAFK,<EXPALL: JOB 0 CFORK FAILED>)
BUG(HLT,TTBAD1,<BAD DEVICE DESIGNATOR FOR TERMINAL AT ATACH2>)
BUG(HLT,UXXCRE,<CANNOT CREATE USAGE FILE>)
BUG(HLT,UXXOPN,<UNABLE TO OPEN USAGE FILE>)
BUG(HLT,UXXCKP,<COULDN'T CREATE CHECKPOINT FILE>)
BUG(HLT,UXXMAP,<USGMAP: CALL TO JFNOFN FAILED>)
BUG(HLT,UXXILL,<USGMES: ILLEGAL FUNCTION CODE>)

BUG(CHK,NOSERF,<CAN'T GTJFN ERROR REPORT FILE>)
BUG(CHK,SERFOF,<CAN'T OPENF ERROR REPORT FILE>)
BUG(CHK,SYSERF,<LOGSST-NO SYSERR STORAGE FOR RESTART ENTRY>)
BUG(CHK,EXPRCD,<EXPALL: RCDIR FAILURE>)
BUG(CHK,UXXFAI,<USAGE JSYS FAILURE>)
BUG(CHK,UXXWER,<WRITE ERROR IN USAGE FILE>,<T1>)
BUG(CHK,UXXCL1,<UNABLE TO CREATE NEW USAGE FILE>)
BUG(CHK,UXXCL2,<UNABLE TO OPEN NEW USAGE FILE>)
BUG(CHK,UXXCL3,<UNABLE TO CLOSE USAGE FILE>)

BUG(INF,USGHOL,<LOST PAGE(S) IN USAGE FILE>)
BUG(INF,UXXFIT,<CHECKPOINT FILE NOT IN CORRECT FORMAT FOR THIS SYSTEM, REBUILDING...>)

MFLIN.MAC.25


MFLOUT.MAC.26


MR.MAC.19


MSTR.MAC.4


N20MED.MAC.1


N20SML.MAC.1


N60BIG.MAC.1


N60MAX.MAC.1
```

NAMAM0.MAC.2

NAMAN.MAC.1

NAMBCH.MAC.4

NAMBIG.MAC.4

NAMDEV.MAC.3

NAMKST.MAC.1

NAMMED.MAC.4

NAMMIN.MAC.4

NAMSML.MAC.3

NCOMND.MAC.1

NETWRK.MAC.6

```
BUG(HLT,NETNNI,<NETINI: NNTBFS NOT INTEGRAL MULTIPLE OF MAXWPM>)
BUG(HLT,NETIEF,<NETOPN: EXTDEC FAILURE AFTER PREVIOUS NON-FAILURE.>)
BUG(HLT,NETWNS,<WATNOT: WAS CALLED FROM SCHEDULER LEVEL.>)
BUG(HLT,NETRBL,<ASNTBF: REQUEST FOR BUFFER LARGER THAN MAXWPM>)
BUG(HLT,NETBAU,<ASNTBF: ATTEMPT TO ASSIGN A BUFFER ALREADY IN USE>)
BUG(HLT,NETRBG,<RLNTBF: ATTEMPT TO RELEASE BUFFER AT GARBAGE LOCATION>)
BUG(HLT,NETBAF,<RLNTBF: ATTEMPT TO RELEASE BUFFER ALREADY ON FREE LIST>)

BUG(CHK,NETDET,<NVTDET: COULD NOT CLOSE NVT>,<T1>)

BUG(INF,NCPFUN,<NCP FSM RECIEVED FUNNY INPUT>,<T1,T2,UNIT>)
```

NSPSRV.MAC.65

```
BUG(HLT,NSPFRK,<NSPINI-CFORK FAILED>)
BUG(HLT,DELNDF,<DELNOD-LLLKUP FAILED>)
BUG(HLT,ADDONF,<ADDOBJ-LLLKUP FAILED>)

BUG(CHK,NSPRTH,<NSPTSK- INVALID ROUTING HEADER>,<T1,T2>)
BUG(CHK,NOMHDR,<ILLEGAL MESSAGE WITH NO HEADER>)

BUG(INF,CLZDIN,<NETCLZ-COULD NOT SEND DI>)
BUG(INF,ILLSTR,<NSPTSK-ILLEGAL INIT MESSAGE>,<Q1>)
```

P20MDD.MAC.3

P20MED.MAC.5

P20SMD.MAC.3

P20SML.MAC.7

P60BIG.MAC.3

P60MAX.MAC.6

PAGEM.MAC.696

```
BUG(HLT,RSMFAI,<RESSMM-FAILED TO ASSIGN SWAP MON PAGE>)
BUG(HLT,ILCST1,<ILLEGAL ADDRESS IN CST1 ENTRY, CAN'T RESTART>)
BUG(HLT,NOTOFN,<UPDOF0-ARG NOT OFN>)
BUG(HLT,ILPPT1,<UPDOFN-BAD POINTER IN PAGE TABLE>)
BUG(HLT,PTOVRN,<UPDPGS-COUNT TOO LARGE>)
BUG(HLT,ILPPT2,<UPDPGS-BAD POINTER IN PAGE TABLE>)
BUG(HLT,STRBAD,<ASOFN-ILLEGAL STRUCTURE NUMBER>)
BUG(HLT,OVFLOW,<ASOFN - ALLOCATION TABLE OVERFLOW>)
BUG(HLT,NOADXB,<RELOFN-NO DSK ADR FOR XB>)
BUG(HLT,SPTFL1,<SPT COMPLETELY FULL>)
BUG(HLT,SHRNO0,<DESPT-SHARE COUNT NON-ZERO>)
BUG(HLT,PTDEL,<DESPT-PT NOT DELETED>)
BUG(HLT,PAGLCK,<DESPT-PAGE LOCKED>)
BUG(HLT,DRMFL1,<ASFSB-UNEXPECTED DRUM FULL>)
BUG(HLT,ILOFN1,<MSCANP-ILLEG IDENT>)
BUG(HLT,ILPTN1,<MRPACS-ILLEG PTN>)
BUG(HLT,SECEX1,<SETMPG-ATTEMPT TO MAP NON-EX SECTION>)
BUG(HLT,XSCORE,<CST TO SMALL FOR PHYSICAL CORE PRESENT>)
BUG(HLT,ILSRC,<ILLEGAL SOURCE IDENTIFIER GIVEN TO SETPT>)
BUG(HLT,ILXBP,<SETPT-BAD POINTER IN XB>)
BUG(HLT,PTNON0,<SETPT0 - PREVIOUS CONTENTS NON-0>)
BUG(HLT,ILSPTI,<ILLEGAL SPT INDEX GIVEN TO SETMXB>)
BUG(HLT,ILDEST,<ILLEGAL DESTINATION IDENTIFIER TO SETMPG OR SETPT>)
BUG(HLT,SPTFL2,<SPT COMPLETELY FULL>)
BUG(HLT,CST2I1,<PAGE TABLE CORE POINTER AND CST2 FAIL TO CORRESPOND>)
BUG(HLT,ILSPTH,<SETPT-SPTH INCONSISTENT WITH XB>)
BUG(HLT,CST2I2,<MVPT-CST2 INCONSISTENT>)
BUG(HLT,CST2I3,<PAGE TABLE CORE POINTER AND CST2 FAIL TO CORRESPOND>)
BUG(HLT,SHROFN,<UPSHR-OFN SHARE COUNT OVERFLOW>) ;YES
BUG(HLT,SPTSHR,<UPSHR-SPT SHARE COUNT OVERFLOW>)
BUG(HLT,SHROFD,<DWNSHR-OFN SHARE COUNT UNDERFLOW>)
BUG(HLT,PGNDEL,<REMFPB-PAGE NOT COMPLETELY DELETED>)
BUG(HLT,RPGERR,<BADCPG-FATAL ERROR IN RESIDENT PAGE>)
BUG(HLT,FRKPTE,<BADCPG-FATAL ERROR IN FORK PT PAGE>)
BUG(HLT,ILFPTE,<ILLFPT: ILLEGAL SECTION NUMBER REFERENCED>)
BUG(HLT,ILPAGN,<MRKMPG-INVALID PAGE NUMBER>)
BUG(HLT,ILPLK1,<MLKPG-ILLEGAL ARGS>)
BUG(HLT,ILULK1,<MULKPG - TRIED TO UNLOCK PAGE NOT LOCKED>)
BUG(HLT,ILULK2,<TRIED TO UNLOCK PAGE NOT LOCKED>)
BUG(HLT,ILULK3,<MULKMP - ILLEGAL MONITOR ADDRESS>)
BUG(HLT,ILULK4,<MULKCR - ILLEGAL CORE PAGE NUMBER>)
BUG(HLT,PAGNIC,<GETCPP-PAGE NOT IN CORE>)
```

```
BUG(HLT,PSBNIC,<SETPPG-PSB NOT IN CORE>)
BUG(HLT,JSBNIC,<SETPPG-JSB NOT IN CORE>)
BUG(HLT,ASGSW2,<SWPOMG-CAN'T ASSIGN RESERVED DRUM ADDRESS>)
BUG(HLT,ILPAG1,<SWPOT0-INVALID PAGE>)
BUG(HLT,DRMFUL,<DRUM COMPLETELY FULL>)
BUG(HLT,BKUPDF,<BKUPD - BAD CST1 ENTRY OR INCONSISTENT CST>)
BUG(HLT,SECGT1,<PGRT3 - SECTION NUMBER GREATER THAN MAXSEC>)
BUG(HLT,PITRAP,<PAGER TRAP WHILE PI IN PROGRESS>)
BUG(HLT,UBANXM,<I/O NMX FROM UNIBUS DEVICE>,<UPTPFW,UPTPFO>)
BUG(HLT,ABKSKD,<ADDRESS BREAK FROM SCHEDULER CONTEXT>)
BUG(HLT,ILLIND,<ILLEGAL INDIRECT>)
BUG(HLT,SECG37,<ILSCN-SECTION NUMBER GREATER THAN 37>)
BUG(HLT,SECNX,<CREATING PAGE TABLE FOR NON-0 SECTION>)
BUG(HLT,ILAGE,<BAD AGE FIELD IN CST0>)
BUG(HLT,ILRFPD,<PDL-OV IN ILLEGAL PAGE REFERENCE>)
BUG(HLT,ILWRT2,<ATTEMPTED WRITE REF TO PROTECTED MONITOR>)
BUG(HLT,ILMADR,<ILLEGAL ADDRESS REFERENCE IN MONITOR>)
BUG(HLT,ILPPT3,<BAD POINTER IN PAGE TABLE>)
BUG(HLT,IBCPYW,<COPY-WRITE POINTER IN INDEX BLOCK>)
BUG(HLT,BADBTB,<NIC- ILLEGAL REFERENCE TO BIT TABLE>)
BUG(HLT,NULQTA,QCHK - NO QUOTA INFO SETUP)
BUG(HLT,SPTPIC,<SWPIN - SPT PAGE ALREADY IN CORE>)
BUG(HLT,PTAIC,<SWPIN - PT PAGE ALREADY IN CORE>)
BUG(HLT,ILSWPA,<SWPIN - ILLEGAL SWAP ADDRESS>)
BUG(HLT,SWPMNE,<SWAP ERROR IN SWAPPABLE MONITOR>)
BUG(HLT,SWPPSB,<SWAP ERROR IN PSB PAGE>)
BUG(HLT,SWPPTP,<SWAP ERROR IN UNKNOWN PT PAGE>)
BUG(HLT,SWPPT,<SWAP ERROR IN UNKNOWN PT>)
BUG(HLT,SWPUPT,<SWAP ERROR IN UPT, OR PSB>)
BUG(HLT,OFFSPE,<OFFSPQ- PAGE NOT ON SPMQ>)

BUG(CHK,XBWERR,<UPDOFN-DSK WRITE ERROR ON XB>)
BUG(CHK,ILIBPT,<BAD POINTER TYPE IN INDEX BLOCK>)
BUG(CHK,WSPNEG,<SOSWSP-WSP NEGATIVE>)
BUG(CHK,NSKDT2,<PGRTRP-BAD INTDF>)
BUG(CHK,NSKDT2,<PGRTRP-BAD NSKED OR INTDF>)
BUG(CHK,FRKBAL,<AGESET-FORK NOT IN BALSET>)
BUG(CHK,SWPFPE,<SWAP ERROR IN SENSITIVE FILE PAGE>)
BUG(CHK,SWPIBE,<SWAP ERROR IN INDEX BLOCK>)
BUG(CHK,SWPJSB,<SWAP ERROR IN JSB PAGE>)

PARAM0.MAC.2

PARAMS.MAC.9

PARAN.MAC.3

PARBCH.MAC.7

PARBIG.MAC.10

PARDEV.MAC.5

PARKST.MAC.12
```

PARMED.MAC.4

PARMIN.MAC.8

PARSML.MAC.8

PHYH11.MAC.53

```
BUG(HLT,NOUBWA,<RH2NCH: NO UNIBUS WINDOW FOR RH11>)
BUG(HLT,CLRACE,UNABLE TO CLEAR REGISTER ACCESS ERROR)
BUG(HLT,RH1ICF,<PHYH11 - INVALID CHANNEL FUNCTION>)
BUG(HLT,RH11CC,<PHYH11 - ILLEGAL CHANNEL COMMAND WORD>)

BUG(CHK,PH1PIE,<PHYH11 - RH11 LOST INTERRUPT ENABLE>)
BUG(CHK,P1NED1,<PHYH11 - RH11 NON EX DISK READING REGISTER>,<T1,T2>)
BUG(CHK,P11PAR,<PHYH11 -- CONTROL WRITE PARITY ERR>,<T1,T2>)
BUG(CHK,P2RAE2,<PHYH11 - REGISTER ACCESS ERR WRITING REG>,<T1,T2,T3>)
BUG(CHK,PH1IHM,<PHYH11 - ILLEGAL HDW MODE - WORD MODE ASSUMED>)

BUG(INF,P1NED3,<PHYH11 - NON EX DISK ON DONE OR ATN INTERRUPT>,<T1,T2>)
```

PHYH2.MAC.71

```
BUG(HLT,RH2ICF,<PHYRH2 - INVALID CHANNEL FUNCTION>)
BUG(HLT,PH2WUI,<WRONG UNIT INTERRUPTED>)

BUG(CHK,PH2PIM,<PHYH2 - RH2Ø LOST PI ASSIGNMENT>,<T2>)
BUG(CHK,P2RAE1,<PHYH2 - RH2Ø REGISTER ACCESS ERROR READING REGISTER>,<T1,T2,T3>)
BUG(CHK,P2RAE2,<PHYH2 - REGISTER ACCESS ERR WRITING REG>,<T1,T2,T3,T4>)
BUG(CHK,PH2IHM,<PHYH2 - ILLEGAL HDW MODE - WORD MODE ASSUMED>)
BUG(CHK,P2RAE3,<PHYH2 - REGISTER ACC ERR ON DONE OR ATN INTERRUPT>,<T1,T2,T3>)

BUG(INF,PH2DNA,<PHYH2 - DONE INTERRUPT AND CHANNEL NOT ACTIVE>,<T2>)
```

PHYM2.MAC.2Ø

```
BUG(CHK,PM2SIO,<PHYM2 - ILLEGAL FUNCTION AT START IO>)
BUG(CHK,TM2NUD,<PHYM2 - CHANNEL DONE INTERRUPT BUT NO UNIT ACTIVE>)
BUG(CHK,TM2IDM,<PHYM2 - ILLEGAL DATA MODE AT DONE INT>)
BUG(CHK,TM2IF2,<PHYM2 - ILLEGAL FUNCTION ON COMMAND DONE>)
BUG(CHK,TM2HER,<TM2ERR - IS.HER SET ON SUCCESSFUL RETRY>)
BUG(CHK,TM2RFU,<PHYM2 - ERROR RECOVERY CONFUSED>,<T1,Q1,T3>)
BUG(CHK,TM2CCI,<PHYM2 - TMØ2 SSC OR SLA WONT CLEAR>)

BUG(INF,TM2N2S,<PHYM2 - MORE DRIVES THAN TABLE SPACE, EXCESS IGNORED>)
BUG(INF,TM2UNA,<PHYM2 - DONE INTERRUPT AND UDB NOT ACTIVE>,<Q1,P3,T1>)
BUG(INF,TM2IDX,<PHYM2 - ILLEGAL RETRY BYTE POINTER>)
BUG(INF,TM2IRF,<PHYM2 - ILLEGAL FUNCTION DURING RETRY>)
```

PHYP4.MAC.8Ø

```
BUG(HLT,RP4UNF,<PHYP4 - UNIT TYPE NOT FOUND:>,T1)
BUG(HLT,RP4PNF,<PHYP4 - DISK PHYSICAL PARAMETERS NOT FOUND>)
BUG(HLT,RP4FEX,<PHYP4 - ILLEGAL FUNCTION>)
BUG(HLT,RP4IF2,<PHYP4 - ILLEGAL FUNCTION AT STKIO>)
```

```
BUG(HLT,RP4IFC,<PHYP4 - ILLEGAL FUNCTION AT CNV>) ;YES TO EITHER
BUG(HLT,RP4LTF,<PHYP4 - FAILED TO FIND TWQ ENTRY AT RP4LTM>)
BUG(HLT,RP4ILF,<PHYP4 - ILLEGAL FUNCTION ON INTERRUPT>)

BUG(CHK,RP4SSC,<PHYP4 - STUCK SECTOR COUNTER>,<T1,T2>)

PHYPAR.MAC.4


PHYSIO.MAC.172

BUG(HLT,PHYP0E,<PHYALZ - PAGE 0 STORAGE EXHAUSTED>)
BUG(HLT,PHYICA,<PHYINI - ILLEGAL ARGUMENT TO CORE ALLOC>)
BUG(HLT,ILTWQ,<PHYINT - TWQ OR PWQ INCORRECT>) ;NO.
BUG(HLT,ILPDAR,<PHYSIO - ILLEGAL DISK ADDRESS IN PAGEM REQUEST>)
BUG(HLT,PYILUN,<PHYSIO - ILLEGAL UNIT NUMBER>)
BUG(HLT,NOIORB,<SETIRB - MISSING IORB>)
BUG(HLT,ILRBLT,<PHYSIO - IORB LINK NOT NULL AT ONF/STWQ>)
BUG(HLT,ILTWQP,<PHYSIO - PWQ OR TWQ TAIL POINTER INCORRECT>)
BUG(HLT,ILIRBL,<PHYSIO - IORB LINK NOT NULL AT ONFPWQ>)
BUG(HLT,ILCNST,<PHYSIO - ILLEGAL CALL TO CONSTW>)
BUG(HLT,ILCNSP,<PHYSIO - ILLEGAL CALL TO CONSPW>)
BUG(HLT,TWQNUL,<PHYSIO - PWQ OR TWQ WAS NULL AT A SEEK OR TRANSFER COMPLETION>)
BUG(HLT,ILUST1,<PHYSIO - UNIT STATUS INCONSISTENT AT SIO>)
BUG(HLT,ILCHS1,<PHYSIO - ILLEGAL CHANNEL STATUS AT SIO>)
BUG(HLT,ILUST5,<PHYSIO - ILLEGAL UNIT OR CHANNEL STATE AT STKIO>)
BUG(HLT,ILCHS2,<PHYSIO - ILLEGAL CHANNEL STATE AT STKIO>)
BUG(HLT,ILUST4,<PHYSIO - CONTROLLER ACTIVE AT SPS>)
BUG(HLT,ILUST3,<PHYSIO - SCHSEK - IMPOSSIBLE UNIT STATUS>)
BUG(HLT,PHYCH1,<PHYSIO - HOME BLOCK CHECK IORB ALREADY ON TWQ>)
BUG(HLT,PHYLTF,<PHYSIO - SCHLTM - UNEXPECTED LATOPT FAILURE>)
BUG(HLT,UIONIR,<UDSKIO - NO IORB FOR NOSKED FORK>)

BUG(CHK,PHYNIR,<PHYSIO - NULL INTERRUPT ROUTINE AT OPERATION DONE>)
BUG(CHK,NRFTCL,<PHYSIO - NO REQUESTS FOUND FOR CYLINDER SEEKED>)
BUG(CHK,NPWQPD,<PHYSIO - NULL PWQ AT POSITION DONE>)
BUG(CHK,ILUST2,<PHYSIO - UNIT STATUS INCONSISTENT AT SPS>)

BUG(INF,PHYICE,<PHYINI - FAILED TO ASSIGN RESIDENT STG>)
BUG(INF,PHYCH2,<PHYSIO - HOME BLOCK CHECK IORB TIMED OUT>)
BUG(INF,PHYCH3,<PHYSIO - HOME BLOCK CHECK IORB TIMED OUT BUT WAS NOT ON TWQ>)
BUG(INF,OVRDTA,<PHYSIO - OVERDUE TRANSFER ABORTED>,<T1,T3,T2>)

PHYX2.MAC.14

BUG(CHK,DX2FGS,<PHYX2 - FAIL TO GET SENSE BYTES>) ;PUT OUT BUGCHK
BUG(CHK,DX2HLT,<PHYX2 - DX20 HALTED>,<T1>)
BUG(CHK,DX2FUS,<PHYX2 - FAIL TO UPDATE SENSE BYTES>)
BUG(CHK,DX2DIE,<PHYX2 - DX20 HALTED>,<T1>)
BUG(CHK,DX2IEC,<PHYX2 - ILLEGAL ERROR CLASS CODE>,<T1>)
BUG(CHK,DX2MCF,<PHYX2 - DX20 MICROCODE CHECK FAILURE>)
BUG(CHK,DX2UPE,<PHYX2 - FAIL TO UPDATE SENSE BYTES DURING INITIALIZATION>)
BUG(CHK,DX2IFS,<PHYX2 - ILLEGAL FUNCTION AT START IO>,<Q1>)
BUG(CHK,DX2NUD,<PHYX2 - CHANNEL DONE INTERRUPT BUT NO UNIT ACTIVE>)
BUG(CHK,DX2NUE,<PHYX2 - NO ACTIVE UDB AND DX20 COMPOSITE ERROR SET>,<T4,T1>)
BUG(CHK,DX2IDM,<PHYX2 - ILLEGAL DATA MODE AT DONE INT>,<T2>)
BUG(CHK,DX2NRT,<DX2ERR - IS.NRT SET ON SUCCESSFUL RETRY>)
BUG(CHK,DX2RFU,<PHYX2 - ERROR RECOVERY CONFUSED>)
```

```
BUG(INF,DX2N2S,<PHYX2 - MORE TU7ØS THAN TABLE SPACE, EXCESS IGNORED>)
BUG(INF,DX2UNA,<PHYX2 - ATTENTION INTERRUPT AND UDB NOT ACTIVE>)
BUG(INF,DX2IDX,<PHYX2 - ILLEGAL RETRY BYTE POINTER>)
BUG(INF,DX2IRF,<PHYX2 - ILLEGAL FUNCTION DURING RETRY>)

PMT.MAC.1


POSTLD.MAC.1Ø


PROKL.MAC.9


PROKS.MAC.3Ø


PROLGØ.MAC.8


PROLOG.MAC.52


PSM.MAC.3


SCHED.MAC.1Ø

BUG(HLT,NSKDIS,<DISMISS WHILE NOSKED OR WITH NON-RES TEST ADDRESS>)
BUG(HLT,SKDCL1,<CALL TO SCHEDULER WHEN ALREADY IN SCHEDULER>)
BUG(HLT,SKDCL2,<CALL TO SCHEDULER WHEN ALREADY IN SCHEDULER>)
BUG(HLT,ILOKSK,<OKSKED WHEN NOT NOSKED>)
BUG(HLT,SKDTRP,<INSTRUCTION TRAP WHILE IN SCHEDULER>)
BUG(HLT,PIITRP,<INSTRUCTION TRAP WHILE PI IN PROGRESS OR IN SCHEDULER>)
BUG(HLT,PISKED,<ENTERED SCHEDULER WITH PI IN PROGRESS>)
BUG(HLT,JØNRUN,<JOB Ø NOT RUN FOR TOO LONG, PROBABLE SWAPPING HANGUP>)
BUG(HLT,GLFNF,<GLREM - FORK NOT FOUND>)
BUG(HLT,TTDAS1,<HLTJB: UNABLE TO DEASSIGN CONTROLLING TERMINAL>)
BUG(HLT,PSISTK,<PSI STORAGE STACK OVERFLOW>)
BUG(HLT,JTENQE,<JTENQ WITH BAD NSKED>)
BUG(HLT,NOACB,<MENTR - NO MORE AC BLOCKS>)
BUG(HLT,OPOPAC,<MRETN - TRIED TO OVER-POP AC STACK>)

BUG(CHK,NOINTR,<ITRAP AND PREVIOUS CONTEXT WAS NOINT>)
BUG(CHK,NOSKTR,<ITRAP FROM NOSKED CONTEXT>)
BUG(CHK,SRQOVF,<SCDRQ-SCHED REQUEST QUEUE OVERFLOW>)
BUG(CHK,SUMNR1,<AJBALS-SUMNR INCORRECT>)
BUG(CHK,SUMNR2,<SUMNR INCORRECT>)
BUG(CHK,FKWSP1,<LOADBS-UNREASONABLE FKWSP>,<T1,T2,T3>)
BUG(CHK,UNBFNF,<UNBLK1 - FORK NOT FOUND>)
BUG(CHK,FRKNDL,<FORK NOT PROPERLY DELETED>)
BUG(CHK,UNPIRX,<UNPIR-NO PSI IN PROGRESS>)
BUG(CHK,PSINSK,<PSI FROM NOSKED CONTEXT>)
BUG(CHK,TRPSIE,<NO MONITOR FOR TRAPPED FORK>)
BUG(CHK,IDFOD1,<AT MENTR - INTDF OVERLY DECREMENTED>)
BUG(CHK,IDFOD2,<AT MRETN - INTDF OVERLY DECREMENTED>)>

SERCOD.MAC.16
```

SMPRE.MAC.6

SMPRM0.MAC.1

SMPRMS.MAC.14

STG.MAC.53

```
BUG(HLT,NSPUDF,<UNSUPPORTED NETWORK FUNCTION>)
BUG(HLT,NOXADR,<EXTENDED ADDRESSING CONFUSION>)
BUG(HLT,DRUMP1,<DRMIO - DRUMP ON BUT NO DRUM CODE IN SYSTEM>)

BUG(CHK,PI5ERR,<UNEXPECTED UNVECTORED INTERRUPT ON CHANNEL 5>)
BUG(CHK,PI6ERR,<UNEXPECTED UNVECTORED INTERRUPT ON CHANNEL 6>)
```

SWPALC.MAC.35

```
BUG(HLT,DRMIBT,<DRMASN-BIT TABLE INCONSISTENT>)
BUG(HLT,DRMNFR,<DRMAM-CAN'T FIND PAGE WHEN DRMFRE NON-0>)
BUG(HLT,ILGDA1,<GDSTX - BAD ADDRESS>)
BUG(HLT,ILGDA2,<GDSTX - BAD ADDRESS>)
BUG(HLT,ILDRA2,<DRMIAD-ILLEGAL DRUM ADDRESS>)
BUG(HLT,DST2SM,<SWPINI-DST TOO SMALL>)

BUG(CHK,ILDRA1,<DASDRM-ILLEGAL OR UNASSIGNED DRUM ADDRESS>)
BUG(CHK,ASGSWB,<SWPINI-CAN'T ASSIGN BAD ADDRESS>)
```

SYSERR.MAC.1

```
BUG(HLT,SERFRK,<SERINI-CANNOT CREATE SYSERR FORK>)

BUG(CHK,SEBUDT,<SEBCPY-UNKNOWN DATA TYPE>,<T1,T4>)
BUG(CHK,SEBISS,<SEBCPY-INSUFFICIENT STRING STORAGE IN BLOCK>)
BUG(CHK,SERGOF,<SETOFI-CANNOT GTJFN/OPEN SYSERR FILE>)
```

TAPE.MAC.80

```
BUG(HLT,MTFCNX,<MTLFCN: FUNCTION CODE TOO LARGE>)
BUG(HLT,BADTYP,<BAD LABEL FIELD DESC>)

BUG(CHK,BADDIS,<TAPE: INCONSISTENT STATE CODE>)
BUG(CHK,CKLBLK,<CKLERR: CLOSE AND ABORT BLOCKED>)
```

TIMER.MAC.6

TTDCDV.MAC.17

```
BUG(INF,TTYPI2,<SCANNER LOST PI ASSIGNMENT, COULD NOT RESTORE>)
BUG(INF,TTYPI1,<SCANNER LOST PI ASSIGNMENT, SUCCESSFULLY RESTORED>)
```

TTDZDV.MAC.154

```
BUG(HLT,DZCLRB,<UNABLE TO RESET DZ11>)
```

```
BUG(CHK,DZNENB,<DZSND1 - TRANSMIT NOT ENABLED ON INTERRUPT>)
BUG(CHK,DZLINT,<DZ11 LOST INTERRUPT ENABLE>,<T2>)
BUG(CHK,DZOVER,<DZ11 SILO OVERRUN>,T1)

TTFEDV.MAC.9

BUG(CHK,NOFRSP,<ttspst- COULD NOT GET A FREE BLOCK>)

BUG(INF,DLDEF,<LOGICAL NAME DEFINE FAILED FOR FE CTY>)

TTMCDV.MAC.16


TTNTDV.MAC.7

BUG(CHK,IMPTMB,<NVTXG1: TOO MANY BREAKS OUTSTANDING>)

BUG(INF,IMPNEA,<NVT RECEIVED BYTES EXCEEDING ALLOCATION>)

TTPTDV.MAC.1


TTYSRV.MAC.43

BUG(HLT,NOCTY,<UNABLE TO ALLOCATE DATA FOR CTY>)
BUG(HLT,TTOCNØ,<TTSTO - NO BUFFER BUT COUNT NON-Ø>)
BUG(HLT,TTICNØ,<TCI - NO BUFFER POINTER BUT COUNT NON-Ø>)
BUG(HLT,TTONOB,<TTY OUTPUT - NO BUFFER BUT COUNT NON-Ø>)
BUG(HLT,BADTTY,<TRANSFER TO NONEXISTENT TTY CODE>)

BUG(CHK,ULKBAD,<UNLOCKING TTY WHEN COUNT IS ZERO>,T2)
BUG(CHK,TTNAC7,<DEALLOCATING INACTIVE LINE>,T2)
BUG(CHK,TTYNTB,<RAN OUT OF TTY BUFFERS>)
BUG(CHK,TTYBBO,<TTYSRV-BIG BUFFER OVERFLOW>)
BUG(CHK,TTILEC,<TTSND-UNRECOGNIZED ESCAPE CODE>,<2,3>)

VERSIO.MAC.184
```