CYBER 18/1700 MSOS ANALYSIS

Seminar Number QA4020

Volume 1

Seminars designed to help improve performance and productivity.

APPLICATIONS & SOFTWARE EDUCATION

CONTROL DATA

SEMINAR DIVISION

COURSE NO. QA4020-1

CYBER 18/1700 MSOS ANALYSIS

STUDENT HANDOUT

VOLUME 1

REVISION B

For Training Purposes Only

Control Data Corporation

PROPRIETARY NOTICE

REVISION RECORD

REVISION	DESCRIPTION
A	
(11-22-78)	Manual Release
B	
(02-01-79)	Update Manual
C	
(06-05-80)	Update Manual

Publication No.
QA4020-1

REVISION LETTERS I, O, Q AND X ARE NOT USED

CYBER 18/1700 MSOS ANALYSIS

Address comments concerning this manual to:

CONTROL DATA CORPORATION
National Coordinator
5001 West 80th Street
Bloomington, Minnesota 55437
Attn: Curtis Vicha

or use Comment Sheet in the back of this manual.

CYBER 18 MSOS ANALYSIS

STUDENT HANDOUT

GENERAL CONTENTS

GENERAL DESCRIPTION

COURSE TITLE: CYBER 18 MSOS Analysis

COURSE NUMBER: QA4020

COURSE LENGTH: 5 days

DESCRIPTION:

This course is designed to provide the system programmer with an in-depth study of the MSOS Operating System. The subject matter includes system initialization routines, SYSDAT, system flow, monitor, I/O routines and loader tables. The operating system will be studied at the flowchart and code level.

PREREQUISITES:

Satisfactory completion of Assembly Language (QA3060) and Advanced Coding (QB4030).

OBJECTIVES:

Upon successful completion of this course the student should have achieved the following:

1. Gained a familiarity with the resource material available to the system analyst.
2. Learned the terminology used in the listings and manuals about MSOS.
3. Understanding of the system flow and use of the major system tables.
4. Understand the basic components of a driver and understand the system provided subroutines for drivers (FNR, MAKQ, COMPRQ, ALTDEV).
5. Be able to describe the function of the major request processors (Scheduler, RW, SPACE).
6. Be able to describe Interrupt Processing.
7. Be able to study system listings.
8. Know how to find important system information in a dump of memory and mass memory.

NOTE TO STUDENTS

The purpose of this Student Guide is, first, to be a teaching aid and, secondly, to supply information that is not included in the other manuals given out in the class. Therefore, when a subject is in another manual, a reference will be made to that material and it will not be duplicated here.

We welcome your comments on the Student Guide. We would appreciate examples charts or flow charts that you feel might improve the Student Guide.

Please send all such suggestions to:

Education Services
ASE, MNA02B
Control Data Corporation
P.O. Box O
Minneapolis, Minnesota 55440

RESOURCE DATA

STUDENT MATERIALS:

CYBER 18 MSOS Analysis Student Handout	
VOLUME 1 Student Handout	
VOLUME 2 Listings	
VOLUME 3 Glossary	QA4020-1
CYBER 18 MSOS Analysis Study Dump	QA4020-3
MSOS Version 5 Reference Manual	96769400
MSOS Version 5 Instant	96769430
MSOS Version 5 Diagnostic Handbook	96769450
MSOS Version 5 File Manager Reference Manual	39520600
Software Peripheral Drivers Reference Manual	96769390

COURSE CHART
MSOS ANALYSIS

HOUR	DAY 1	DAY 2	DAY 3	DAY 4	DAY 5
1	INTRODUCTION TO MASS STORAGE OPERATING SYSTEM VERSION 5 ANALYSIS	SYSTEM FLOW ✓	SCHEDULER ✓	DRIVER ✓	LOADER TABLES
2	CYBER 18 HARDWARE	COMMON INTERRUPT HANDLER ✓			DEBUGGING/ TRACING PROCEDURES
3	CYBER 18 SOFTWARE OVERVIEW	DISPATCHER ✓	INTRODUCTION ✓ TO SYSTEM I/O TABLES	MEMORY ALLOCATION	
4			READ/WRITE ✓ REQUEST PROCESSOR		JOB PROCESSOR
5		REQUEST ENTRY/ EXIT ✓	DRIVERS ✓	VOLATILE STORAGE	REVIEW
6				TIMER PACKAGE	
7					

COURSE OUTLINE
MSOS ANALYSIS

I. Introduction to CYBER 18 Mass Storage Operating System Version 5 (MSOS 5) (1 hr.)

A. Overview of MSOS 5

B. Introduction to MSOS 5 Analysis

1. Materials available on MSOS
2. Class introduction
3. Overview of outline

II. CYBER 18 Hardware (1 hr.)

A. CYBER 18 Configurations

1. CYBER 18-10M Computer System
2. CYBER 18-20 Processor
3. CYBER 18 Cartridge Drive

B. Example Configuration

1. CYBER 18 Features (MACRO)
2. Memory word

C. Input/Output Instructions

1. Return from INP or OUT
2. Disk addressing

D. Panel Mode Operation

1. Function Control Register
2. FCR Table

III. Software Overview (4 hrs.)

A. Terms and Concepts

1. Compile, load, execute process
2. Types of programs
3. Background/foreground

B. Priority Structure

1. Priority scheme
2. Interrupts
3. Priority level

C. Queues for CP Usage

1. Interrupt Stack
2. Scheduler's Queue

D. The Libraries

1. Program Library
2. System Library
3. LIBEDT

E. System Initialization

IV. System Flow

A. Common Interrupt Handler (6 hrs.)

1. Interrupt trap
2. Changing priority

B. Dispatcher

1. Scheduler's queue

C. Request entry/exit

1. Indirect request
2. MONI

V. Scheduler (2 hrs.)

A. System Directory Call

B. Pseudo Interrupt

VI. Introduction to System I/O (2 hrs.)

A. System Standard Logical Units

B. Physical Device Tables

1. PHYSTAB
2. LOG 1A
3. LOG 2
4. LOG 1

C. Read/Write Request Format

COURSE OUTLINE (Continued)

VII. Drivers (4 hrs.)

- A. Driver Review
- B. Initiator
- C. Continuator
- D. Error Section
- E. Common Subroutines for all Drivers

VIII. Memory Allocation (2 hrs.)

- A. Core Allocator
- B. Space Driver (SPACDR)
- C. Request for Space
- D. RELEAS Request
- E. SPACE Request Processor
- F. SUBCOR

IX. Volatile Storage (1 hr.)

X. TIMER Package (1 hr.)

- A. TIMER Requests
- B. TIMER
- C. DIAGNOSTIC TIMER

XI. LOADER Tables (1-1/2 hrs.)

- A. LOADER Functions
- B. Background Program Layout
- C. LOADER Blocks
- D. Example Program

COURSE OUTLINE (Continued)

XII.	Debugging/Tracing Procedures	(1-1/2 hrs.)
XIII.	Introduction to Job Processor	(1-1/2 hrs.)
XIV.	MSOS Test and Review	(1-1/2 hrs.)

LESSON GUIDE 1

INTRODUCTION TO MASS STORAGE OPERATING SYSTEM VERSION 5 (MSOS 5)

LESSON PREVIEW:

This lesson is a general introduction to the study of MSOS. The student will be introduced to the resources needed by the systems analysts to maintain the system such as SIMs, PSRs, Data Sheets, etc. The objective and course outline will be gone over so that everyone is aware of the purpose and scope of course.

REFERENCES:

MSOS RM pv/vi

TRAINING AIDS:

Background Material

Visuals VI-1 through VI-12

PROJECTS:

Study questions-1

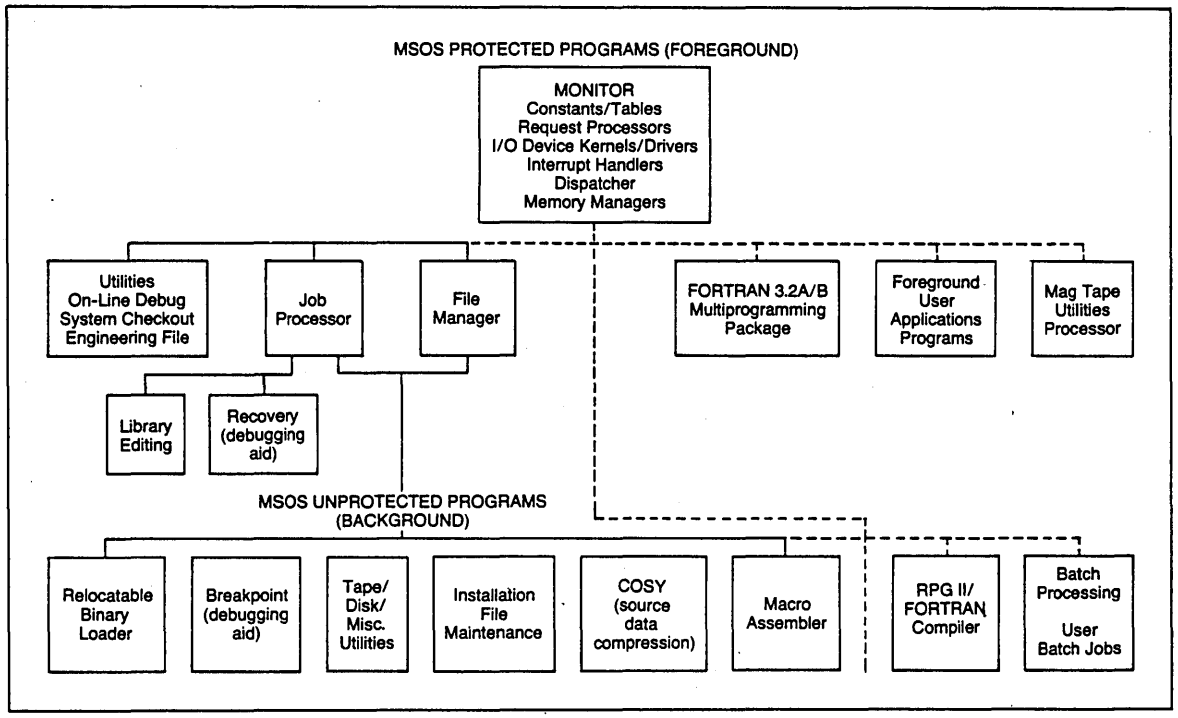
OBJECTIVES:

At the completion of this lesson, the student will be able to:

1. Know where to find information about the Operating System.
2. Know where to go to get assistance.
3. Know how to submit a PSR and how to find out if the problem exists in other places.
4. Gain an understanding of the objectives and scope of this class.

CONTROL DATA® CYBER 18
MASS STORAGE OPERATING SYSTEM VERSION 5 (MSOS 5)

CONTROL DATA CORPORATION



The CONTROL DATA CYBER 18 Mass Storage Operating System Version 5 (MSOS 5) is a multiprogramming system designed to support a variety of applications requiring dedicated system utilization, batch processing, and program checkout features in a real-time environment.

MSOS 5 regulates all multiprogramming on the basis of priority level assigned to a particular operation, whether the operation is program execution or input/output. The system queues for input/output data transmission and program execution by priority level, with no restriction on the number of requests which may be queued at a given time. The program selected for execution is the one with the highest priority level. It remains in execution until completed unless a higher priority level program is scheduled. The lower priority level is then suspended until the higher priority interrupt program is completed.

Sixteen hardware interrupts are used to maximize input/output efficiency and to allow concurrent input/output and computation.

The program protect feature of the hardware is used to segregate central memory into two functional entities—protected memory and unprotected memory.

Protected memory (the foreground) is reserved for executing the operating monitor and any user's real-time high priority application programs. Unprotected memory (the background) is used for execution of batch job processing and program checkout. MSOS 5 can swap (move) the contents of unprotected memory to mass storage, and make the area protected memory for use by foreground programs.

MSOS 5 is extremely modular in design and provides the user considerable flexibility to perform system modification and update.

The MSOS 5 System features the following main capabilities:

- *MONITOR*—The real-time executive for MSOS 5. The monitor is the interface to other programs and systems resources on a priority basis. It is modular and parameters can be set for a variety of hardware and software configurations.

The monitor contains request processors for the following:

I/O—READ/WRITE/FREAD/FWRITE/MOTION

Program Scheduling—SCHDLE/SYSCHD

Time Delays—TIMER/TIMPT1

Memory Allocation—SPACE/RELEAS/PTNCOR

Enable/Disable Scheduling Mass Memory Programs—DISCHD/ENSCHD

Background Requests—STATUS/CORE/EXIT/GTFILE/LOADER

- *JOB PROCESSOR*—Responsible for monitoring background programs running in unprotected memory. Interface is provided for batch stream, unattended jobs or for interactive, operator-controlled jobs. The job processor controls compilers, MACRO assembler, and numerous background utility functions.
- *FILE MANAGER*—General purpose file management package. It creates and maintains both sequential and indexed files. It offers sequential, indexed, and direct methods of record retrieval, as well as variations of

these. The file manager may be used by protected and unprotected programs.

SOFTWARE PRODUCT SET

FORTRAN 3A/B	RPG II
AUTRAN 3	FILE MANAGER
TIMESHARE 3	MAGNETIC TAPE
IMPORT	UTILITIES PROCESSOR
GRAPHICS	I/O DRIVERS

MINIMUM HARDWARE REQUIREMENTS

CDC® CYBER 18 Computer with 16,000 words main memory for MSOS 5

Console device (teletype, CRT)

Input device (paper tape, cards, magnetic tape)

Output device (paper tape, cards, magnetic tape)

Minimum 512,000 words mass storage for MSOS 5

Specifications are subject to change without notice

CONTROL DATA SALES OFFICES ARE LOCATED IN
PRINCIPAL CITIES THROUGHOUT THE WORLD
DATA SYSTEMS MARKETING

BOX 0, MINNEAPOLIS, MINNESOTA 55440
TELEPHONE: (612) 853-5195 TWX: 910-576-2978

201,321 LITHO IN U.S.A. 7/76

CONTROL DATA CORPORATION

PROGRAMMING SYSTEMS REPORT

91024

DATE SUBMITTED MO. DAY YEAR	REFERENCE NO. (SUBMITTER)	INSTALLATION CODE	DATE REC. BY PSR COORD.	CDC PSR NUMBER

FROM

Customer Name _____ Customer Phone (include Area or Country and City Codes) _____

Customer Address _____

Control Data Contact Name _____ Control Data Location (Code) _____ Control Data Contact Signature _____

PE INQUIRY STATEMENT HERE

COMPUTER

Has job run on a previous level?

☐ Yes Level _____

☐ NO

OPERATING SYSTEM (actually used)

Name _____

Version _____ PSR Code-Level _____

Product Number _____

PRODUCT being reported on (give full name, as COBOL 5 under NOS1).

Name _____

Version _____ PSR Code Level _____

Product Number _____

Has CEM service been contracted for this product? YES ☐ NO ☐

SUBMITTER PRIORITY

☐ CRITICAL †

☐ URGENT

☐ SERIOUS

☐ MINOR

☐ INFO

† For critical PSRs be sure to follow procedures noted on the reverse side of this form.

INQUIRY TYPE

☐ 1. Error (Support Materials Enclosed)
☐ 2. Suggestion for Improved Efficiency (SIE)
☐ 3. Request for Software Modification (RSM)

SUPPORT MATERIALS

☐ Card Deck ☐ Dump Tape ☐ Other
☐ Listing ☐ Suggested Fix (Cards)
☐ Data Tape ☐ Typeout

SUGGESTED PROBLEM DESCRIPTION (Limit of 70 Characters Including Embedded Blanks (132 characters for 3000L)) – BE CONCISE

1 ▸ 70

TYPE INQUIRY PER INSTRUCTIONS ON REVERSE SIDE

AA1901 REV. 12/77

MASTER

INSTRUCTIONS FOR FILLING OUT THIS PROGRAMMING SYSTEM REPORT (PSR) FORM — ONLY ONE PROBLEM PER FORM

A PSR form should be used to report any of three types of inquires to CONTROL DATA CORPORATION in regard to standard software products.

Type 1 An error inquiry: the software does not work according to the published reference manual(s).

Type 2 A suggestion for improved efficiency (SIE); suggested change which does not affect the external features of the product; i.e., shorter or faster code.

Type 3 Request for Software Modification (RSM): a request for a change in the way the software product works which will require user adjustments and a change in the reference manual(s).

THE FORM MUST BE TYPED (IF NOT, IT WILL BE REJECTED).

Please complete the form according to the following instructions:

1. Enter date submitted/mailed to Control Data.
2. Enter submitter reference number — any combination of six characters. Use of this field is optional; if not filled in, CDC PSR Coordination will use the metered number on the form as the submitter reference number.
3. Enter four-character customer Installation Code (VIM, FOCUS, ECODU or other user-group code) or, for Control Data installations, an abbreviated Facility Code.
4. Do not type in the DATE REC. BY PSR COORD. or CDC PSR NUMBER blocks.
5. Enter appropriate customer and Control Data name, address and telephone number information. The form should be signed by the local Control Data representative.
6. Enter Computer number, operating system and product identification. For example, computer number could be: CYBER 173, S/N 614.
7. Check the appropriate Submitter Priority of this inquiry (see descriptions below for guidelines concerning priority). There is no commitment that Control Data will assign the same priority to the problem; however, gross disparity between priorities will be questioned.

Critical	(CRITICAL PSR PROCEDURES MUST BE FOLLOWED). Use for system down; frequent (more than 1 per day) system crashes; major projects stalled through software problems, etc. Remember, this is your estimation of problem criticality — to get CDC to handle the problem as critical, it is necessary that established critical PSR procedures be followed, e.g., for CDC CYBER 70/170 the local CDC representative must agree regarding criticality and then must TWX/Telex PSD Field Support (who then get Central Support to accept or reject the critical request). For 3000L systems, contact the 3000L PSR Coordinator.
Urgent	Regular system crashes (more than 1 per week); substantial user difficulties. High probability of serious problems (such as bugs in error recoveries, etc.).
Serious	Problems that definitely need to be fixed at once, but for some reason are below Urgent or Critical. For example, a PSR belongs in this category if the problem can be circumvented, if a local or temporary fix is available, or if it is an urgent problem that only occurs rarely or under unusual circumstances.
Minor	Inconsistencies or irregularities that need to be corrected in the system (Minor refers only to the urgency). Items of inconvenience or of minor or primarily local consequence should preferably be in this category.
Information	Errors in comments, coding techniques, and documentation; nonconformity to standards.

Many problems may seem Critical or Urgent. Therefore the following tests may be helpful in classification of the problem:

- If you will wait for a full test of the corrective code (a corrective code release) rather than implement an uncertified response, the problem is less than Critical and should probably be Serious rather than Urgent.
- If you will continue to tolerate a problem rather than quickly generate a new system after tested corrective code is available, the priority should be Minor or Information.
- If your distribution of PSRs by category places more than 10% to 15% in the critical and urgent categories, you should re-examine your use of these priorities.

8. Check the Type of Inquiry being submitted. (Refer to the first paragraph of this page for determination of proper Type.)
9. Check the type of support materials being submitted with the inquiry. More complete supporting materials will facilitate our isolating the cause of the problem — when feasible/appropriate, please include a system dump tape (core dumps) as part of the materials.
10. Enter a concise description of the problem — which may be used in the PSR index. Since this suggested index entry may be used by others in locating previously reported problems, it is important that the description be accurate and specific. The entry is limited to 70 characters, including embedded blanks (132 characters for 3000L).
11. Please type the inquiry description starting at the top of the form so that we will have the maximum amount of space available for answering the inquiry. A complete description of the problem and related symptoms should be entered to facilitate location and correction thereof. If available/appropriate, we encourage suggested corrective code be submitted as a card deck. (For more than 20 lines of code.) When this is done, we will list the cards and publish them as part of the inquiry.
12. In order to resubmit any PSR for further consideration, please place the following in the Suggested Problem Description area: "This PSR is a resubmittal of PSR ABCXXXX". Please restate the problem and your reason for resubmittal.

Submit all copies of the form to Control Data's local representative, who will sign the PSR and submit it to the appropriate location:

CDC CYBER 70/170
PSR COORDINATION
215 MOFFETT PARK DRIVE
SUNNYVALE, CALIFORNIA 94086

3000L SYSTEMS
PSR COORDINATION, ARH280
4201 LEXINGTON AVENUE NORTH
ST. PAUL, MINNESOTA 55112

SYSTEM 17/1700
PSR COORDINATION
4455 EASTGATE MALL
LA JOLLA, CALIFORNIA 92037

EXAMPLE

CYBER 18
PROGRAMMING SYSTEM
REPORT

AUGUST 31, 1978

SUMMARY

132

SECTION I	-	51 PAGES
SECTION II	-	2 PAGES
SECTION III	-	60 PAGES
SECTION IV	-	0 PAGES
SECTION V	-	6 PAGES

000001

TABLE OF CONTENTS

SUMMARY LEVEL 132

SECTION I

000002

TYPE INQUIRY STARTING HERE

CONTROL DATA CORPORATION

PROGRAMMING SYSTEMS REPORT 17477

DATE SUBMITTED MO DAY YEAR	REFERENCE NO. (SUBMITTOR)	INSTALLATION CODE	DATE REC. BY PSR COORD.	CDC PSR NUMBER
07 26 78	PCL001	PCL	7/27/78	132-1-4 5467

FROM

Charles L. Myers — Customer Name
3333 Michelson Dr., Irvine, CA — Customer Address
92730 (714) 975-5311 — Customer Phone (With Area Code)

Keith Weber — Control Data Contact Name
LJLOPS — Control Data Location (Code)
Keith C. Weber — Control Data Contact Signature

COMPUTER	OPERATING SYSTEM NAME-VERSION-PSR CODE LEVEL	PRODUCT/PROGRAM NAME-VERSION-PSR CODE LEVEL	SUBMITTOR PRIORITY
Cyber -18/10M	MSOS 5.0 126	(Deck Name) DTIMER 5.0	☐ Critical† ☒ Minor ☐ Urgent ☐ Info. ☐ Serious

INQUIRY TYPE
- ☐ 1. Error (Support Materials Enclosed)
- ☐ 2. Suggestion for Improved Efficiency (SIE)
- ☒ 3. Request for Software Modification (RSM)

SUPPORT MATERIALS
- ☐ Card Deck ☐ Dump Tape ☒ Other
- ☐ Listing ☐ Suggested Fix (Cards)
- ☐ Data Tape ☐ Typeout

†For critical PSRs be sure to follow procedures noted on the reverse side of this form.

SUGGESTED PROBLEM DESCRIPTION (Limit of 70 Characters Including Embedded Blanks (132 characters for 3000L)) – BE CONCISE

Module "DTIMER" does not test for end of DGNTAB table properly.

1 ... 70

The DTIMER module terminate processing of "DGNTAB" on any negative value rather than on the specific value $FFFF as specified. This causes a problem when the PHYSDT resides above $8,000.

TYPE INQUIRY PER INSTRUCTIONS ON REVERSE SIDE

AA 1901 Rev 3/75

MASTER 000014

CONTROL DATA CORPORATION

PROGRAMMING SYSTEMS REPORT

84282 108

DATE SUBMITTED MO DAY YEAR	REFERENCE NO (SUBMITTER)	INSTALLATION CODE	DATE REC BY PSR COORD	CDC PSR NUMBER
	215	PLYOPS	7/26/78	5466

132-1-3

FROM

Customer Name

612-553-4342
Customer Phone (include Area or Country and City Codes)

Customer Address

Paul Sitz
Control Data Contact Name

PLY01B
Control Data Location (Code)

Control Data Contact Signature

TYPE INQUIRY STATEMENT HERE

COMPUTER

System 17

Has job run on a previous level?

☐ Yes Level ____

☐ NO

OPERATING SYSTEM (actually used)

Name MSOS

Version 5.0 PSR Code Level 119

Product Number

PRODUCT being reported on (give full name as COBOL 5 under NOS1)

Name ILOAD

Version ____ PSR Code Level 110

Product Number

Has CEM service been contracted for this product? YES ☐ NO ☐

SUBMITTER PRIORITY

☐ CRITICAL †

☐ URGENT

☒ SERIOUS

☐ MINOR

☐ INFO

† For critical PSRs be sure to follow procedures noted on the reverse side of this form

INQUIRY TYPE

☒ 1 Error (Support Materials Enclosed)

☐ 2 Suggestion for Improved Efficiency (SIE)

☐ 3 Request for Software Modification (RSM)

SUPPORT MATERIALS

☐ Card Deck ☐ Dump Tape ☐ Other

☐ Listing ☐ Suggested Fix (Cards)

☐ Data Tape ☐ Typeout

SUGGESTED PROBLEM DESCRIPTION (Limit of 70 Characters including Embedded Blanks (132 characters for 3000L)) - BE CONCISE

1 70

System initializer loader is not compatible with system loader in treatment of Fortran Compiler generated relocatable address in the range of #7F80 to #7FFF.

Suggested code:

```
ILOAD  DCK/ I.H
       DEL/ 1
       NAM  ILOAD              DECK-ID 018   MSOS 5.0
       DEL/ 747.749
*             RELOC. ADDR. GR. OR EQ. #7F80 are ASSUMED TO
*             MEAN BACKWARD RELOCATION -- THE FORTRAN
*             COMPILER GENERATES SUCH RELOCATABLE ADDRESSES
*             ON INDEXED VARIABLES.  LARGER BACKWARD
*             RELOCATION MAY BE OBTAINED BY MAKING THE VALUE
*             AT NXWS3+2 SMALLER
NXWD3  SAM  NXWD4
       SUB  =X#7F80             THIS VALUE IS STRICTLY ARBITRARY
       SAM  NXWD4
       CLR  A                   BIT 15 OF THESE VALUES MUST
*                               BE SET TO CAUSE 16 BIT ADDR. ARITH.
*                               TO GENERATE AN END AROUND CARRY.
```

TYPE INQUIRY PER INSTRUCTIONS ON REVERSE SIDE

AA1901 REV 12/77

MASTER

000013

EXAMPLE

CONTROL DATA CORPORATION

SOFTWARE AVAILABILITY BULLETIN

Distribution List For
CDC CYBER 18/System 17/1700
Bulletin No. 56
July 14, 1977

PSR CORRECTIVE CODE
MSOS 5 AND MSOS 5 PRODUCT SET
(Level 110-118)

A. ABSTRACT

Corrective code for PSRs contained in summaries 110 through 118 is available for the following products in the forms of COSY corrections for program changes and relocatable binary object code for changed programs.

1. MSOS 5 (Product no. A325/A305-01)
2. FORTRAN 3A under MSOS 5 (Product no. A325/A305-02)
3. FORTRAN 3B under MSOS 5 (Product no. A325/A305-03)
4. File Manager 1 under MSOS 5 (Product no. A325/A305-04)
5. Peripheral Drivers 1A under MSOS 5 (Product no. A325/A305-08)
6. Peripheral Drivers 1B under MSOS 5 (Product no. A325/A305-09)
7. Peripheral Drivers 1C under MSOS 5 (Product no. A325/A305-10)
8. RPG II 1 under MSOS 5 (Product no. A325/A305-12)

B. PUBLICATIONS

There are no new manuals published for this release.

C. ORDERING INFORMATION

Licensed software products and their update materials are available only to customers who have entered into a contractual agreement with Control Data for the use of the specific software products.

1. New software products must be covered by a license agreement which lists each product explicitly.
2. Update materials should be ordered by completing the Request for Software Product Update Materials form attached to this bulletin. Information for completing this form is contained in the following paragraphs and in the system summary table at the end of this section. The completed form should be sent to the local sales representative.
   a. Media desired is normally either:
      - M7-556 (7-track magnetic tape at 556 bpi)
      - M7-800 (7-track magnetic tape at 800 bpi)†
      - M9-800 (9-track magnetic tape at 800 bpi)
      - CD (punched cards)
   b. SOFTWARE PRODUCT NUMBER may be selected from column 2 in the system summary table and should correspond to the product number listed in the licensing agreement with Control Data.

† Required for CDC CYBER 18-10M, 18-20, and 18-30 systems when 7-track media is desired.

c. DESCRIPTION is the product name shown in column 1 of the system summary table; the modules listed under the product name should not be entered on the Update Material Request form.

d. UPDATE/RELEASE LEVEL DESIRED is found in column 3 of the system summary table under Nominal Release Level Identifier.

e. UPDATE/RELEASE LEVEL CURRENTLY AT SITE refers to the level of release materials already being used or the latest level previously shipped to your site; this information will enable Software Manufacturing and Distribution to determine exactly what materials are needed to bring the software product up to the desired code/release level.

For example, suppose a site received a software system on 9-track tapes at the initial release level 110 and now wishes to have the latest available materials for the MSOS system and Peripheral Drivers 1B. Based on information in the system summary table, the Request for Software Product Update Materials form should show:

Media Desired	Software Product Number	Description	Update/Release Level Desired	Update/Release Level Currently at Site
M9	A325/A305-01	MSOS 5	118	110
M9	A325/A305-09	Peripheral Drivers 1B	118	110

f. System type refers to the mainframe (for example, 1700, System 17, CDC CYBER 18).

g. COSY corrections are not part of the standard operating system and must be specified, if desired, for each product. If COSY corrections are desired, the customer must have the latest COSY available. The level of the latest COSY can be determined by referring to column 4 of the system summary table.

If COSY corrections are not specified, only the relocatable binary object code will be sent.

MSOS 5.0 SYSTEM SUMMARY TABLE - (LEVEL 110-118)

Product Name	Applicable Product Number† A325 or A305	Nominal Release Level Identifier	Latest COSY Available is at Level	New Features at Level 118	COSY Re-sequenced at Level
MSOS 5	-01	118	110	No	110
FORTRAN 3A	-02	118	102	No	102
FORTRAN 3B	-03	118	102	No	102
File Manager 1	-04	118	110	No	110
Macro Assembler 3	-06	110	110	No	110
Peripheral Drivers 1A	-08	118	110	No	110
Peripheral Drivers 1B	-09	118	110	No	110
Peripheral Drivers 1C	-10	118	110	No	110
Magnetic Tape Utility 2	-11	110	106	No	106
RPG II 1	-12	118	108	No	108
Sort/Merge 1	-13	110	108	No	108

†A325 product numbers refer to the products offered with the optional Central Enhancement and Maintenance Service (CEM Services).

PRODUCT SUPPORT HOTLINE - by J. Michael Birch

Inquiries and problems concerning CYBER18 or 1700 products should be directed to the La Jolla HOTLINE at extension 6328, LJLOPS or by TWX to HOTLINE, LJLOPS.

This service is primarily for the use of PSD field analysts requiring central support for CYBER18 software. However, it may also be used for inquiries regarding status of CYBER18 PSR, status of orders placed with LJLOPS s/w manufacturing and hardware problems not resolvable by normal local CE and Tech support channels. Questions regarding product plans and development schedules will be routed to the LJLOPS Business Office. Schedule and other business problems regarding established accounts should be referred directly to the designated manager.

The HOTLINE is not for customer use. It is intended to provide a single controlled interface for technical inquiries from CDC personnel outside the La Jolla Division. Direct calls to development programmers and others disrupt normal activities and may result in conflicting answers or failure to follow up. Such persons have been asked to redirect their calls to the HOTLINE. Also, the person supposedly an 'expert' on the subject may not be available or the problem may require evaluation by more than one person. The basic procedure is as follows:

HOTLINE PROCEDURE OUTLINE

1. Customer describes problem to PSD Field Analyst.
2. PSD Field Analyst investigates and clarifies problem.
3. PSD Field Analyst TWX's/calls HOTLINE ext 6328 LJLOPS.
4. HOTLINE Coordinator receives TWX/answers phone.
5. HOTLINE Coordinator records inquiry and assigns I.D. no.
6. HOTLINE Coordinator routes inquiry to support Analyst.
7. Support Analyst records problem details.
8. Support Analyst investigates problem and determines response.
9. HOTLINE Coordinator sends response by TWX.
10. HOTLINE Coordinator notes if follow-up required or closes inquiry.

To help the service function smoothly please use the following guidelines:

HOTLINE GUIDELINES

1. HOTLINE is for PSD Field Analysts et al, not customer.
2. Be specific and concise. TWX's are preferred to calls.
3. Undated inquiries should be separately identified.
4. Identify the affected product properly (ITOS, RPGII etc.)
5. Provide your name, facility code, customer site etc., to the Coordinator.

6. Do not ask to speak to specific individuals.
7. Inquiries not responded to within 48 hours will be acknowledged by the Coordinator.
8. Specify the previous inquiry no. if applicable.

We are presently relocating and improving the HOTLINE phone system as well as attempting to improve our procedures and add staff. Your comments and suggestions are welcomed and, together with any complaints re HOTLINE service, may be addressed to J. Michael Birch, Manager, Product Support LJLOPS or George R. Olson, Manager, Systems I&E LJLOPS.

BACKGROUND INFORMATION

Manuals

File Manager Reference Manual	39520600
Cyber 18 Computer Systems	96767850
Installation Handbook (V4)	39520900
Literature Distribution Catalog	

Other

PSR/PSR Summaries

Software Information Memo (SIM)

Programming Systems Information (PSI)
(For CDC personnel only)

NOTE: The January issue has an index of all articles published up to that time.

Feature Abstract Memorandum (FAM)

Software Availability Bulletin (SAB)

Data Sheets

Hot Line - Phone: 714/452-6328 (for CDC Analyst)

TWIX: LJLOPS

Listing of your SYSDAT

Dump of your system (Memory & Disk Tables)

NOTE: The Software Availability Bulletin will tell you which version of the manual goes with your version of the O.S.

Manuals may be ordered from Literature Distribution Services (LDS)
(612/292-2100)

STUDY QUESTIONS-1

1. What is a PSR?

2. Where do you order DATA Sheets?

3. Where would you look for an article a new release of the Operating System?

4. If you were asked to give a presentation on a new piece of hardware where would you look for a summary of its characteristics that would be in a form suitable to hand out to the listeners?

5. If you are not using the latest version of a system, where do you look to find out what version of the manuals apply to your system?

6. What are your objectives in taking this class?

Q1-1

LESSON GUIDE 2

CYBER 18 HARDWARE OVERVIEW

LESSON PREVIEW:

This lesson covers the hardware information necessary to understand the software. The student should be familiar with most of this information, therefore it is included as background information, and for your review. Test your knowledge by going over the study questions.

REFERENCES:

MSOS RM pp. 1-3 thru 1-7, Appendix L
CYBER 18 Computer System Summary, Chapters 1,5,6,7

TRAINING AIDS:

Visuals V2-1 through V2-3

PROJECTS:

Study questions - 2

OBJECTIVES:

1. Student should be able to describe under what conditions an interrupt is responded to by the CPU and what exactly happens when one is responded to.

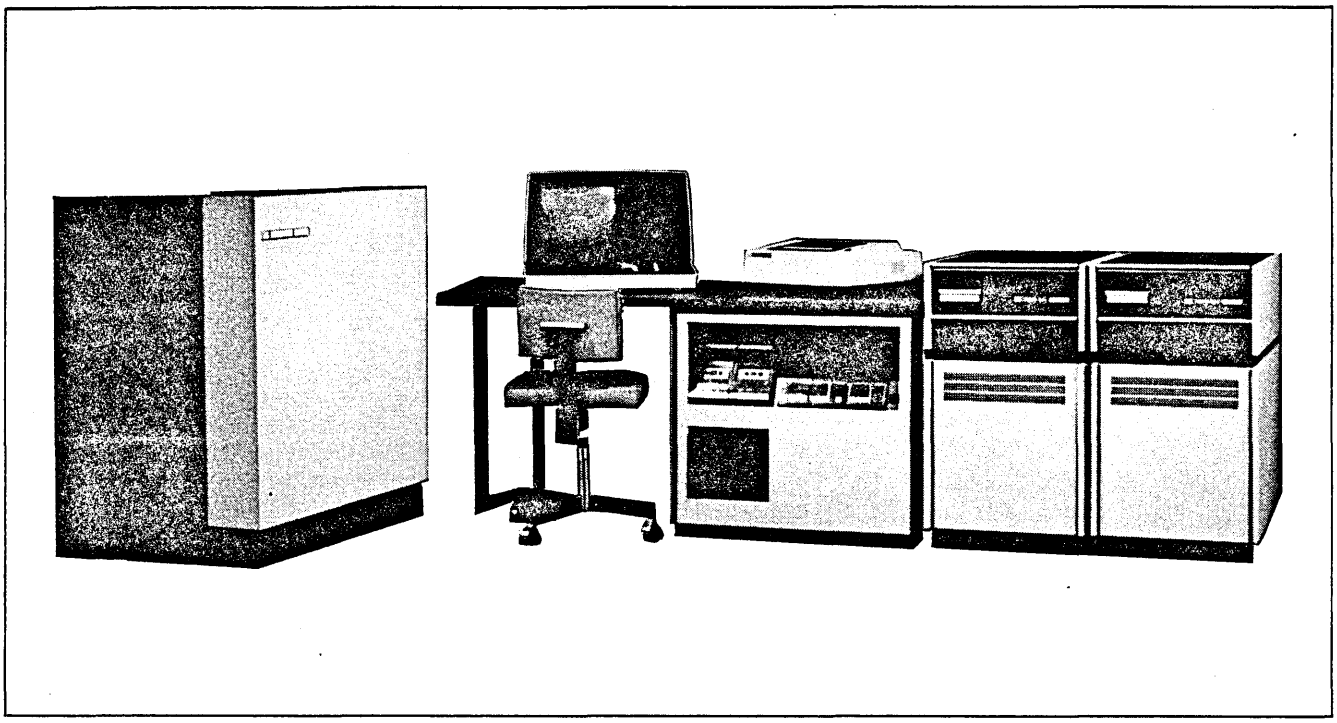
2. The student should be able to describe the elements of a typical configuration.

3. The student should be aware of how the INP/OUT instructions work.

4. Describe the type of information found in the FCR?

CONTROL DATA®
CYBER 18-10M COMPUTER SYSTEM

CONTROL DATA CORPORATION



FEATURES

- General-purpose digital processor, using microprogrammed architecture
- Accommodates 32K through 128K bytes macro main memory
- Main memory effective read/write cycle time of 750 nanoseconds
- Powerful instruction repertoire
- Eight addressing modes for accessing main memory
- Main memory word and region protection
- Main memory parity detection, optional error correction
- Direct memory access
- Integral flexible disk drive for diagnostic loading (Optional use as system peripheral.)
- Automatic program load (deadstart) facility for loader type peripherals
- Integral real-time clock
- Modular design, CPU and controllers on 11" x 14" PC boards for ease of handling
- High reliability and ease of maintenance through state-of-the-art technology and advanced diagnostic capability
- RS232-C compatible I/O interface for console display or TTY
- Priority-oriented interrupt system with sixteen levels of interrupts
- Optional breakpoint controller
- Wide range of peripherals supported
- Cabinet, operator's panel, power distribution, and power supplies included

DESCRIPTION

The CDC® CYBER 18-10M is a general-purpose, 16-bit processor. Execution of macro programs stored in MOS main memory is controlled by micro-level programs stored in micro memory. ROM micro memory is provided for execution of the basic CDC 1700 instruction set and the additional enhanced instructions, including character and field manipulation, indexing, micro memory referencing, autodata transfer, and main memory paging control. Arithmetic is one's complement, signed, fixed-point hardware and/subtract/multiply/divide.

Addressing Modes

The following eight addressing modes are provided for maximum flexibility:

- Absolute
- Indirect
- Relative
- Relative Indirect
- Constant
- Storage
- Storage Indirect
- Field

Instruction Repertoire

CYBER 18-10M incorporates the basic CDC 1700 instruction set and additional enhanced instructions not previously available. This repertoire includes one, two and three word (two 8 bit bytes per word) instructions and is flexible for increased programming efficiency. Instruction groups include the following:

- Transfer
- Logical
- Stop
- Shift
- Interrupt
- Generate Parity
- Character/Field Manipulation
- Execute Micro Code Sequence
- Arithmetic
- Jump
- Decision
- Input/Output
- Memory Paging Control

Some instructions are immediate (literal), resulting in a saving of operand storage and execution time. Multiword instructions, such as indirect addressing, are a means of addressing locations which cannot be accessed directly.

Registers

The 18-10M provides 15 registers, including four general-purpose registers to support the enhanced instruction set, and four special-purpose registers used exclusively for machine control.

Register Functions

A (16 bit) —	Principal arithmetic register; data register during I/O operations
Q (16 bit) —	Auxiliary arithmetic register; peripheral address register during I/O operations
P (15 or 16 — bits)	Program address register
X (16 bits) —	Storage data register
Y (16 bits) —	Address register; hold temporary results during address computation
M (16 bits) —	Interrupt mask register
B (16 bits) —	Breakpoint address register
I (16 bits) —	Indexing, accumulation, and loop control register
1, 2, 3, 4, — (16 bits)	Indexing, accumulation, and loop control registers
LB, UB — (16 bits)	Lower and upper bound registers for unprotected area
MFP — (64 x 9 bits)	Memory page file

Program Protection

CDC CYBER 18-10M offers two modes of protection from damage which may be caused by programs accessing memory outside their own region. Traditional word level protection of the 1700 Series allows individual words to be declared protected by setting a bit in memory associated with that word. A second means of protection uses upper and lower bounds to define an unprotected region. This has the same effect as word protection, except that a large unprotected area can be defined more quickly.

Main Memory System

CDC CYBER 18-10M features high-speed dynamic MOS LSI storage elements. Each word in memory consists of two data bytes, one protect, and one parity bit. Memory is organized as a single bank with two ports — CPU and DMA.

Storage capacity is expandable from 32K to 128K bytes by the simple insertion of individual PC boards. CDC CYBER 18-10M includes no main memory; however, up to two card slots are provided to accept any mixture of 32K and 64K byte MOS memory array boards (Options 1882-16 and 1882-32). The effective memory cycle time at either port is 750 nanoseconds; however, the memory processes simultaneous requests from both ports with an average effective cycle time of 600 nanoseconds.

Interrupt System

CDC CYBER 18-10M firmware emulates 16 levels of vectored interrupt. This system consists of 15 levels of external interrupt and one internal interrupt.

Certain conditions such as an illegal instruction, a memory parity error, or a power failure generate an internal interrupt. External interrupts occur when a computer peripheral device has finished an I/O operation or requires attention. The interrupt system will handle up to 16 interrupts in a flexible and efficient manner.

Real-Time Clock

The real-time clock is an integral part of the CDC CYBER 18-10M, and provides a macro-level interrupt at a programmable interval. The real-time clock appears as a CDC CYBER 18 peripheral to the macro program.

Input/Output Capability

CDC CYBER 18-10M contains nine card slots for peripheral controllers. Three levels of interface are provided for the peripherals: Direct Memory Access (DMA), Auto Data Transfer (ADT), and AQ.

The DMA channel permits direct transfer of data between the peripherals and main memory. The DMA channel supports four devices and permits data transfer rates up to 2,800,000 bytes per second.

ADT provides pseudo DMA transfers of data blocks between main memory and those peripherals designed to accommodate ADT.

At the macro level each transfer appears as DMA; however, each transfer is controlled at the micro level by the emulator in micro memory. Data transfer rates up to 160,000 bytes per second are possible. Three ADT devices are supported.

The AQ channel provides data transfers between CPU registers and peripherals. The transfers are macro-program controlled. CDC CYBER 18-10M supports a maximum of four AQ devices. AQ data transfer rates are software dependent.

One additional I/O interface is included for the operator console device. This interface is both KSR 33/35 TTY compatible and RS232-C compatible.

Program Deadstart

Loading programs into main memory is provided by this feature. Data is input bit-serially from the deadstart program loading device.

Operator's Panel

An operator's panel is also included, and is used to initiate operation of the processor and deadstart device.

PACKAGING

CDC CYBER 18-10M includes a low-profile, free-standing cabinet with integral table top. The processor chassis, with peripheral controllers, power supply module, and power distribution are contained within the cabinet. Individual CPU and peripheral controller PC cards are 11 x 14 inches.

CONFIGURATION

Basic configuration includes a cabinet with operator's panel, a basic processor, a flexible disk drive and controller, an I/O controller to support the operator console, and power supply (no main memory is included).

Minimum system configuration consists of 32K bytes main memory, a load device such as a card reader, and a comment device such as a conversational display terminal.

SOFTWARE

Supporting software includes Mass Storage Operating System (MSOS), Real-Time Operating System (RTOS), and Interactive Terminal Operating System (ITOS). Both MSOS and RTOS are real-time, multiprogramming operating systems, with 16 program priority levels.

RTOS

. . . resides within the CPU memory and has no mass storage requirements. It includes a monitor (subset of MSOS) which occupies less than 1500 words of main memory, exclusive of drivers and optional features.

MSOS

. . . supports applications requiring dedicated system utilization, batch processing, and program checkout features in a real-time environment. Its modular design provides flexibility in system updating or modification.

ITOS

. . . provides an environment in which a terminal user operates with an on-line data base, using interactive application programs. ITOS Release 1 operates in conjunction with MSOS 5.0.

MAINTENANCE FEATURES

Self-test and echo mode tests are included for troubleshooting the basic processor and optional controllers.

The system is also supported by the Operational Diagnostic System (ODS). This maintenance system includes diagnostic software with fault isolation capability, Diagnostic Decision Logic Tables (DDLT's) and detailed repair procedures. These tools produce a highly effective and efficient maintenance system.

OPTIONS AND PERIPHERALS

- Processor Options

1875-1	Breakpoint Controller
1875-2	Breakpoint Panel
1882-16	MOS Memory Expansion, 32K bytes
1882-32	MOS Memory Expansion, 64K bytes
1874-1	Memory Error Correction (ECC)

- Cable Options

1827-950	Line Printer, 50 ft. (15.24 m)
1829-915	Card Reader, 15 ft. (4.57 m)
1843-950	Modem Cable, 50 ft. (15.24 m)

- Peripheral Controller Options

1828-1	Card Reader/Line Printer Controller
1828-2	Card Reader/Line Printer/ Communications/ Line Adapter Controller
1833-4	Cartridge Disk Controller
1843-1	Dual Channel Synchronous/ Asynchronous Communications Line Adapter
1843-2	Eight Channel Communications Line Adapter
1862-1	Paper Tape Reader/Punch Controller

- Peripheral Options

1811-1	Conversational Display Terminal
1811-2	Operator Console
1827-7	Impact Printer, 70 lpm, Matrix
1827-30/31	Line Printer, 300 lpm
1827-60	Line Printer, 600 lpm
1829-30	Card Reader, 300 cpm
1829-60	Card Reader, 600 cpm
1860-1,2,3,4	Tape subsystem, 7 and 9 tracks, 25 ips, 800 bpi NRZI (expandable to 4 tapes)
1860-5,6	Tape subsystem, 9 track, 50 ips, 800 bpi NRZI and 1600 bpi Phase Encode (expandable to 4 tapes)
1865-2	Flexible Disk Drive (second unit)
1866-12	Cartridge Disk Drive, 4.4 million words
1866-14	Cartridge Disk Drive, 8.8 million words
1888-1	Power Transformer, 220 VAC/120 VAC

SPECIFICATIONS

Type: General-purpose 16-bit processor
Organization: Register/file oriented
Hardware Accumulators: 7
Index Registers: 7
Addressing Modes: 8
Arithmetic: One's complement
Priority Interrupt Levels: 16 macro
Macro Memory Type: Dynamic MOS LSI RAM
Macro Memory Size: 32K to 128K bytes
Macro Memory Cycle Time: 750 nsec effective (2 bytes
I/O Ports: 8 (4 DMA, 4 AQ)
Direct Memory Access: Four devices; up to 2,800,000 bytes per second
Auto Data Transfer: Four devices; up to 160,000 bytes per second
AQ Data Transfer: Four devices
Real-Time Clock: Programmable macro interrupt

Physical

Height: 29 in. (73.66 cm)
Width: 61 in. (154.94 cm)
Depth: 31 in. (78.74 cm)
Weight: 475 lbs. (215.460 Kg)

Power

Source: 104 to 127VAC, 1 phase, 3 wire
49.0 to 60.6 HZ (198 to 235VAC, 1 phase, w/Option 1888-1)
Consumption: 2.4KVA

Environmental

Operating Temperature: 50°F to 95°F (10°C to 35°C)
Humidity: 20% to 80% R.H. (noncondensing)
Heat Dissipation: 2064 KCAL/HR (4508 BTU/HR)
Altitude: −1000 to 8000 feet

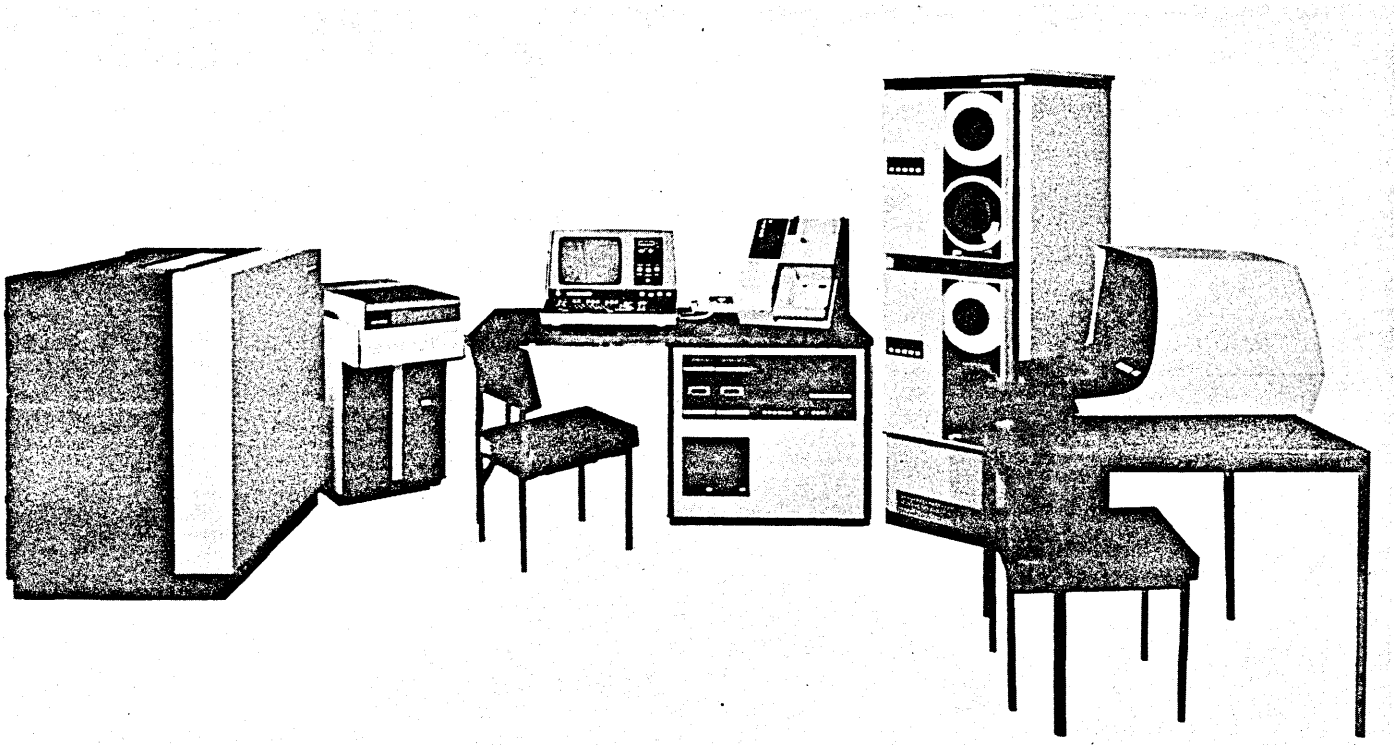
Specifications subject to change without notice

DATA SYSTEMS MARKETING
Box 0
Minneapolis, Minnesota 55440

201,333 LITHO IN U.S.A. 2/77

CONTROL DATA®
CYBER 18-20 PROCESSOR

CONTROL DATA CORPORATION



The CDC® CYBER 18-20 is a general purpose microprogrammable, 16-bit processor. Execution of macro programs stored in MOS main memory is controlled by micro-level programs stored in micro memory. ROM micro memory is provided for execution of the basic CDC 1700 instruction set and the additional enhanced instructions, including character and field manipulation, indexing, micro memory referencing, autodata transfer, and main memory paging control. Read/write micro memory is available for user microprogramming requirements. Arithmetic is one's complement, signed, fixed-point hardware add/subtract/multiply/divide.

FEATURES

- General purpose digital processor, using microprogrammable architecture
- Accommodates 32K through 262K bytes macro main memory
- Main memory effective read/write cycle time of 750 nanoseconds
- Micro instruction cycle time of 168 nanoseconds
- Powerful instruction repertoire
- Eight addressing modes for accessing main memory
- Main memory word and region protection
- Main memory parity detection with optional automatic single-error correction and double-error detection
- Direct memory access
- High-speed I/O data transfer for integral peripheral controllers
- Automatic program load (deadstart) facility for loader type peripherals
- Integral real-time clock
- Modular design, CPU and controllers on 11" x 14" PC boards for ease of handling
- High reliability and easy maintainability through state-of-the-art technology and advanced diagnostic capability
- I/O communications interface for teletypewriter or RS 232-C compatible display terminal
- Priority oriented interrupt system with sixteen levels each of micro and macro interrupts
- Optional breakpoint controller
- Basic processor supports wide range of peripherals
- Cabinet, operator's panel, power distribution, and power supplies included
- Optional read/write micro memory

CONFIGURATION

The basic configuration includes a cabinet with operator's panel, a basic processor, an I/O controller to support the communications console, and power supply (no main memory is included).

It operates in a minimum system configuration of the CYBER 18-20 processor, 32K bytes main memory, a load

device such as a card reader, and a comment device such as a conversational display terminal.

SOFTWARE

Supporting software includes Mass Storage (MSOS) and Real Time (RTOS) Operating Systems. Both MSOS and RTOS are real-time multiprogramming operating systems with 16 program priority levels.

Hardware interrupts are used to maximize input/output efficiency. All I/O requests are processed on a software priority basis. A program protect system is used to maintain system integrity.

RTOS resides within the CPU memory and has no mass storage requirements. Includes a monitor (subset of MSOS) which occupies less than 1500 words of main memory, exclusive of drivers and optional features.

MSOS supports applications requiring dedicated system utilization, batch processing, and program checkout features in a real-time environment. Its modular design provides flexibility in system updating or modification.

PACKAGING

CYBER 18-20 includes a low-profile, free-standing cabinet with integral table top. The processor chassis with peripheral controllers, power supply module, and power distribution are contained within the cabinet. Individual CPU and peripheral controller PC cards are 11 x 14 inches.

MAINTENANCE FEATURES

Self-test and echo mode tests are included for trouble-shooting the basic processor and optional controllers. The system is also supported by controlware diagnostics included in the Operational Diagnostic System (ODS).

Tests are performed while using Diagnostic Decision Logic Tables (DDLT's) and special maintenance procedures that isolate and correct the fault. These features provide maximum efficiency in system maintenance.

OPTIONS AND PERIPHERALS

- Processor Options
  - 1870-1 512 Instruction Micromemory
  - 1870-2 2048 Instruction Micromemory
  - 1874-1 ECC MOS Array, 196K bytes
  - 1875-1 Breakpoint Controller
  - 1875-2 Breakpoint Panel
  - 1882-16 MOS Memory Expansion, 32K bytes
  - 1882-32 MOS Memory Expansion, 65K bytes
- Cable Options
  - 1827-950 Line Printer, 15.24m (50 ft.)
  - 1829-915 Card Reader, 4.57m (15 ft.)
  - 1833-950 Storage Module Driver, 15.24m (50 ft.)
  - 1843-950 Modem Cable, 15.24m (50 ft.)
  - 1843-901 Terminal Adapter
- Peripheral Controller Options
  - 1828-1 Card Reader/Line Printer Controller
  - 1832-4 NRZI Magnetic Tape Controller
  - 1833-1 Storage Module Drive Interface
  - 1833-2 Storage Module Drive Interface (dual CPU)
  - 1833-3 Control Unit for storage module
  - 1833-5 Flexible Disk Drive Controller
  - 1843-1 Dual Channel Synchronous/Asynchronous Communications Line Adapter
- Peripheral Options
  - 1811-1 Conversational Display Terminal
  - 1827-30/31 Line Printer, 300 LPM
  - 1829-30 Card Reader, 300 CPM
  - 1829-60 Card Reader, 600 CPM
  - 1860-72 Tape Transport, 7 track, 25 IPS (up to 4 drives per controller)
  - 1860-92 Tape Transport, 9 track, 25 IPS (up to 4 drives per controller)
  - 1860-200 Tape Drive Installation Kit (upper)
  - 1860-201 Tape Drive Installation Kit (lower)
  - 1865-1 Flexible Disk Drive (unit 0)
  - 1865-2 Flexible Disk Drive (unit 1)
  - 1867-10/11 Storage Module Drive (25 M byte)
  - 1867-20/21 Storage Module Drive (50 M byte)
  - 1887-4 Cabinet
  - 1888-1 Power Transformer, 220 VAC/120 VAC
  - 1888-2 Power Transformer, 120 VAC/220 VAC
  - 1890-1 200 UT Emulation
  - 1890-2 2780 Emulation
  - 1890-3 3780 Emulation
  - 65119-1 Line Printer, 600 LPM

ADDRESSING MODES—CYBER 18-20 provides the following eight addressing modes for maximum flexibility:

- Absolute
- Indirect
- Relative
- Relative Indirect
- Constant
- Storage
- Storage Indirect
- Field

MACRO INSTRUCTION REPERTOIRE

CYBER 18-20 incorporates the basic CDC 1700 instruction set and additional enhanced instructions not previously available. This repertoire includes one, two, and three word instructions and is flexible for increased programming efficiency. Instruction groups include the following:

- Transfer
- Logical
- Stop
- Shift
- Interrupt
- Generate Parity
- Character/Field Manipulation
- Execute Micro Code Sequence
- Arithmetic
- Jump
- Decision
- Input/Output
- Memory Paging Control

Some instructions are immediate (literal), resulting in a saving of operand storage space and execution time. Multi-word instructions, such as indirect addressing, are a means of addressing locations which cannot be accessed directly.

REGISTERS

The CYBER 18-20 processor provides fifteen registers. The seven traditional registers are used in execution of the normal CDC 1700 instruction set; four general-purpose registers have been added to support the enhanced instruction set. Four special-purpose registers are used exclusively for machine control.

REGISTER	*FUNCTION*
A (16 bit)	Principal arithmetic register; data register during I/O operations
Q (16 bit)	Auxiliary arithmetic register; peripheral address register during I/O operations
P (15 or 16 bits)	Program Address Register
X (16 bit)	Storage data register
Y (16 bit)	Address register; holds temporary results during address computation
M (16 bit)	Interrupt mask register
B (16 bit)	Breakpoint address register
I (16 bit)	Indexing, accumulation, and loop control register
B (16 bit)	Breakpoint address register
1, 2, 3, 4, (16 bit)	Indexing, accumulation, and loop control registers
LB, UB (16 bit)	Lower and Upper bound registers for unprotected area
MPF (64 x 9 bits)	Memory page file

PROGRAM PROTECTION

CYBER 18-20 offers two modes of protection from damage which may be caused by programs accessing memory outside their own region. Traditional word level protection of the 1700 Series allows individual words to be declared protected by setting a bit in memory associated with that word. A second means of protection uses upper and lower bounds to define an unprotected region. This has the same effect as word protection, except that a large unprotected area can be defined more quickly.

INTERRUPT SYSTEM

CYBER 18-20 firmware emulates the 16 levels of vectored interrupt featured on the 1700 Series Computers. This system consists of 15 levels of external interrupt and one internal interrupt.

Certain conditions such as an incorrect instruction, a memory parity error, or a power failure will generate an internal interrupt. External interrupts occur when a computer peripheral device has finished an I/O operation or requires attention. The strength of the interrupt scheme is the ability to handle a significant number of interrupts in a flexible and efficient manner.

MAIN MEMORY SYSTEM

CYBER 18-20 features high-speed dynamic MOS LSI storage elements. Each word in memory consists of two data bytes, one protect, and one parity bit. Memory is organized as a single bank with two ports—CPU and DMA.

Storage capacity is expandable from 32K to 262K bytes by the simple insertion of individual PC boards. CYBER 18-20 includes no main memory; however, four card slots are provided to accept any mixture of 32K and 65K byte MOS memory array boards (Options 1882-16 and 1882-32). The effective memory cycle time at either port is 750 nanoseconds; however, the memory processes simultaneous requests from both ports with an average effective cycle time of 600 nanoseconds.

Double-error detection and automatic single-error correction, for up to 196K bytes, is provided as Option 1874-1.

INPUT/OUTPUT CAPABILITY

CYBER 18-20 contains 10 card slots for peripheral controllers. Three levels of interface are provided for the peripherals: Direct Memory Access (DMA), Auto Data Transfer (ADT), and AQ.

The DMA channel permits direct transfer of data between the peripherals and main memory, by-passing the CPU entirely. The DMA channel supports four devices and permits data transfer rates up to 1,400,000 words per second.

ADT provides pseudo DMA transfers of data blocks between main memory and those peripherals designed to accommodate ADT. At the macro level each transfer appears as DMA; however, each transfer is controlled at the micro level by the 1700 emulator in micro memory. Data transfer rates up to 80,000 words per second are possible. Ten ADT devices are supported.

The AQ channel provides data transfers between CPU registers and peripherals. The transfers are macroprogram controlled. CYBER 18-20 supports a maximum of nine AQ devices. AQ data transfer rates are software dependent.

One additional I/O interface is included for the operator input device. This interface is both ASR/KSR 33/35 TTY compatible and RS232-C compatible.

PROGRAM DEADSTART

Loading programs into main memory and read/write micro memory is provided by this feature. Data is input bit-serially from the deadstart program loading device.

REAL-TIME CLOCK

The real-time clock is an integral part of the CYBER 18-20, and provides a macro-level interrupt at a programmable interval. The real-time clock appears as a CYBER 18 peripheral to the macro program.

OPERATOR'S PANEL

CYBER 18-20 includes an operator's panel to initiate operation of the processor and deadstart device.

SPECIFICATIONS

Type: General-purpose, microprogrammable, 16-bit processor

Organization: Register/file oriented

Hardware Accumulators: 7

Index Registers: 7

Addressing Modes: 8

Arithmetic: One's complement; two's complement available with RAM micromemory

Priority Interrupt Levels: 16 micro and 16 macro

Macro Memory Type: Dynamic MOS LSI RAM

Macro Memory Size: 32K to 262K bytes without ECC; 32K to 196K bytes with ECC

Macro Memory Cycle Time: 750 nsec effective (2 bytes)

Micro Instruction Word Length: 32 bits

Micro Memory Type: TTL ROM,TTL RAM available

Micro Memory Size: 1024 instruction ROM; 512 to 4096 instruction RAM available

Micro Memory Cycle Time: 168 nsec with up to 4 parallel operations

Direct Memory Access: Four devices; up to 1,400,000 words per second

Auto Data Transfer: Ten devices; up to 80,000 words per second

AQ Data Transfer: Nine devices

Serial Data Transfer: TTY and RS232-C compatible

Real-Time Clock: Programmable macro interrupt

Physical—

Height: 73.66 cm (29 inches)
Width: 154.94 cm (61 inches)
Depth: 78.74 cm (31 inches)
Weight: 215.460 kg (475 pounds)

Power—

Source: 104 to 127 VAC, 1 phase, 3 wire
49.0 to 60.6 HZ (198 to 235 VAC, 1 phase, w/ Option 1888-1)
Consumption: 2.4 KVA

Environmental—

Operating Temperature: 10°C to 35°C (50°F to 95°F)

Operating Humidity: 20% to 80% RH (non-condensing)

Heat Dissipation: 2064 KCAL/HR (4508 BTU/HR)

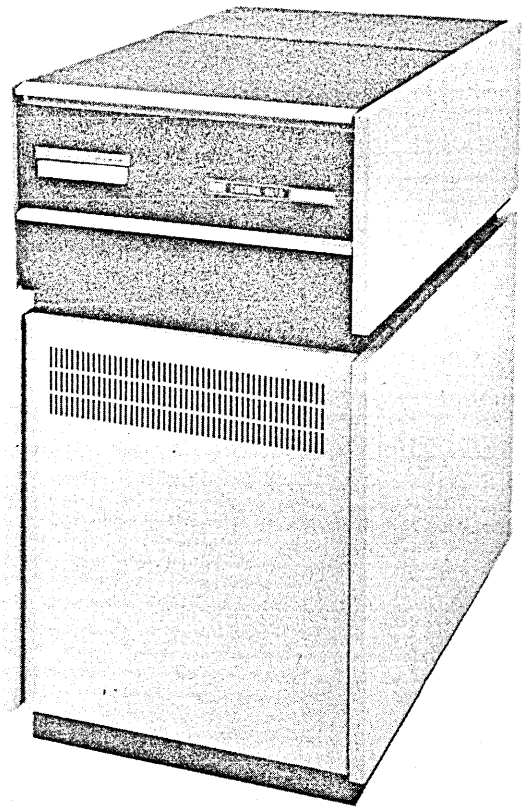
Specifications subject to change without notice.

CONTROL DATA SALES OFFICES ARE LOCATED IN
PRINCIPAL CITIES THROUGHOUT THE WORLD
DATA SYSTEMS MARKETING
BOX 0, MINNEAPOLIS, MINNESOTA 55440
TELEPHONE: (612) 853-5195 TWX: 910-576-2978

201,323 Litho in U.S.A. 7/76

CONTROL DATA® CYBER 18 CARTRIDGE DISK SUBSYSTEM (1833-4 CONTROLLER AND 1866-12/1866-14 DRIVE)

CONTROL DATA CORPORATION



IMPORTANT FEATURES

- Compact modular design
- Up to four drives per CPU I/O port
- Up to 18 million words, on-line
- One fixed and one removable cartridge
- Up to 2.2 million words per cartridge
- Seek overlap for fast data access
- CPU autoload capability
- Self-test features
- Powerful diagnostics

GENERAL DESCRIPTION

The CDC® CYBER 18 1833-4, 1866-12/14 Cartridge Disk Subsystem provides both data and programming mass storage for the central computer system. It consists of a CDC 1833-4 Cartridge Disk Controller and 1866-12 and/or 1866-14 Cartridge Disk Drives. These drives can be intermixed in any combination with up to four drives per controller. Using four 1866-14 double-density drives provides on-line storage of 35 million bytes. One removable cartridge per drive permits unlimited off-line storage.

The drives (1866-12 and 1866-14) can store 4.4 million bytes and 8.8 million bytes respectively. Each drive employs one fixed disk plus one interchangeable cartridge. Information is stored on two oxide-coated surfaces of each disk. Movable head positioning is performed by a closed-loop, proportional servo system which controls a voice-coil linear actuator. The average track-move time is 35 milliseconds.

The 1833-4 Controller consists of a single module which mounts inside of the CPU chassis. The controller interfaces to one direct memory access port and can control a maximum of four disk drives connected in daisy-chain fashion.

OPERATION

This cartridge disk subsystem permits read, write, and data-compare functions to be performed on large amounts of file data. In addition, a special auto-load function permits deadstart loading of disk data from any drive into the CPU main memory. The subsystem accepts multiple seek commands from software and overlaps the seeking operations among drives. Once selected, the data transfer between the disk and CPU memory takes place via a high-speed, direct memory access data path. Data transfer rate is 312,000 8-bit bytes per second. Checkword generation and checking is automatic and provides confidence in data accuracy.

Controls and Indicators—

Operator controls are minimal and conveniently located on the front of the drive unit. Removal and installation of the interchangeable disk cartridge is easily accomplished from the top of the unit.

PACKAGING

The controller mounts inside the CDC CYBER 18 CPU chassis and requires no external power source. A cable connects from

the CPU to the first drive unit, with daisy-chain connection between additional drives. Drive units are compact and mount on a pedestal base. Each drive unit contains its own power supply and cooling facilities. Construction is modular and subassemblies are easily accessible for convenient maintenance and adjustments.

CONFIGURATION

The subsystem includes a 1833-4 Controller, a 20-foot cable between the controller and first drive, and 10-foot cables between adjacent drives. Each drive connects individually to an AC power source. This subsystem operates in a minimum system configuration of a CDC CYBER 18 processor with operator's panel, 32K bytes of main memory, and a comment device such as a display terminal.

SOFTWARE

Supporting software includes the Mass Storage (MSOS) Real-Time (RTOS) and Interactive Terminal (ITOS) Operating Systems. Both MSOS and RTOS are real-time, multiprogramming operating systems with 16 program priority levels.

Hardware interrupts are used to maximize input/output efficiency. All I/O requests are processed on a software priority basis. And a program-protect system is used to maintain system integrity.

RTOS resides within the CPU memory, has no mass storage requirements, and includes a monitor (subset of MSOS) which occupies less than 1500 words of main memory exclusive of drivers and optional features.

MSOS supports applications requiring dedicated system utilization, batch processing, and program checkout features in a real-time environment. Its modular design provides flexibility in system updating or modification.

ITOS provides an environment in which a terminal user operates with an on-line data base, using interactive application programs. ITOS Release 1 operates in conjunction with MSOS 5.0.

MAINTENANCE

The 1833-4/1866 Cartridge Disk Subsystem is supported by a number of maintenance features. Four self-test modes of the controller, initiated by powerful diagnostic software, permit rapid fault detection and isolation. In addition to diagnostic software, Diagnostic Decision Logic Tables (DDLT's) and detailed maintenance procedures make up the total CDC CYBER 18 Operational Diagnostic System (ODS). These features provide maximum efficiency in maintaining the system.

SPECIFICATIONS

Performance—

Recording Density: 220 bpi
Sector Size: 192 18-bit bytes
Sectors Per Track: 29
Tracks Per Surface: 200 plus 4 spares (1866-12)
400 plus 8 spares (1866-14)
Surfaces Per Disk: 2
Head Positioning Time: 7 milliseconds (one-track move)
70 milliseconds (maximum move)
35 milliseconds (average)
Rotational Speed: 2400 rpm
Average Latency: 12.5 milliseconds
Transfer Rate: 312,000 bytes per second

Disk Cartridge—

Diameter: 14 inches (35 cm)
Coating: Magnetic oxide
Configuration: One fixed/one removable

Operator Controls—

Switches/Indicators: Start/Stop
Fault
Spindle Stop

Physical—

Height: 34 inches (86 cm)
Width: 18.5 inches (46 cm)
Depth: 29.75 inches (74 cm)
Weight: 275 pounds (125 kg)

Power Requirements—

Per Drive: 120 volts, 7 amps, 60 Hz, single phase 198-275 volts, 3.5 amps nominal, 50 Hz, single phase

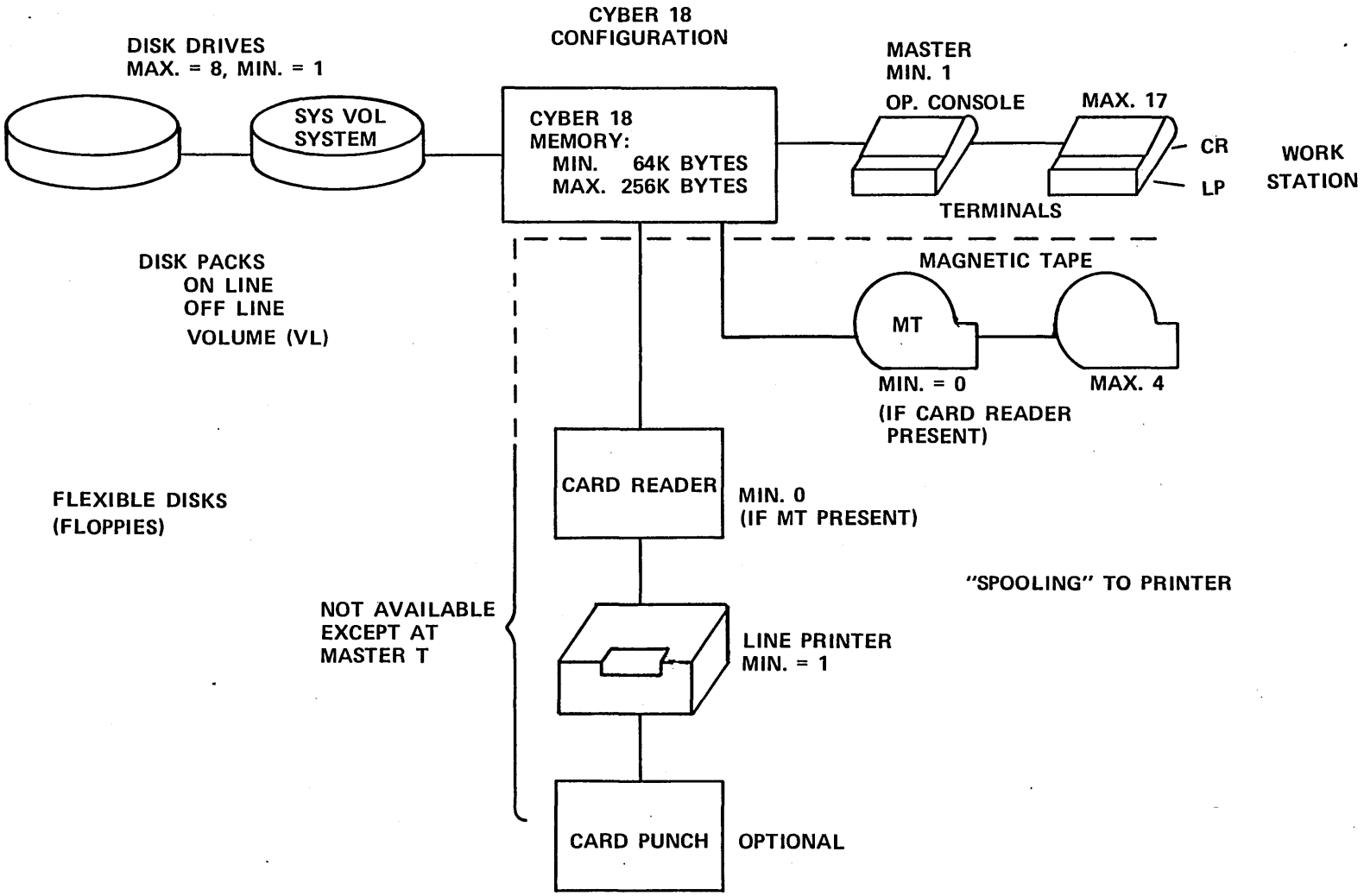
Environmental—

Operating Temperature: 60°F to 90°F (116°C to 32°C)
Operating Humidity: 10 to 80% R.H., noncondensing

Specifications subject to change without notice.

DATA SYSTEMS MARKETING
Box 0
Minneapolis, Minnesota 55440

201,496 LITHO IN U.S.A. 3/77

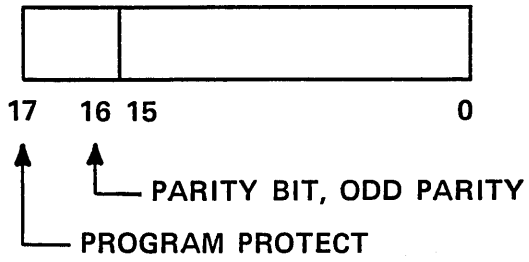

CYBER 18
CONFIGURATION
DISK DRIVES
MAX. = 8, MIN. = 1
SYS VOL
SYSTEM
CYBER 18
MEMORY:
MIN. 64K BYTES
MAX. 256K BYTES
MASTER
MIN. 1
OP. CONSOLE
MAX. 17
CR
LP
WORK
STATION
TERMINALS
DISK PACKS
ON LINE
OFF LINE
VOLUME (VL)
MAGNETIC TAPE
MT
MIN. = 0
MAX. 4
(IF CARD READER
PRESENT)
CARD READER
MIN. 0
(IF MT PRESENT)
FLEXIBLE DISKS
(FLOPPIES)
"SPOOLING" TO PRINTER
NOT AVAILABLE
EXCEPT AT
MASTER T
LINE PRINTER
MIN. = 1
CARD PUNCH
OPTIONAL

CYBER 18 FEATURES
(MACRO)

- 16 BIT OPERAND IN MEMORY
  (2 BITS, 1 FOR PARITY,
  1 FOR PROGRAM PROTECT)
- MEMORY SIZE: 64K TO 262K BYTES
- 7 PROGRAMMABLE REGISTERS
  (A,Q,M,R1,R2,R3,R4)
- ONE'S COMPLEMENT ARITHMETIC
- INTEGER ADD, SUBTRACT, MULTIPLY
  AND DIVIDE
- 16 INTERRUPTS
- CYCLE TIME OF 750 NSEC./WORD

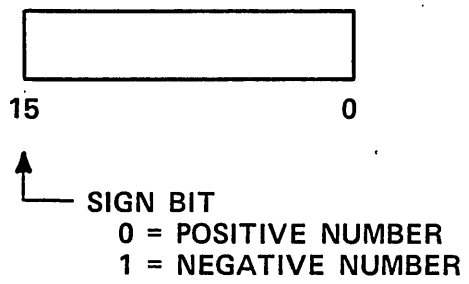
CYBER 18

MEMORY WORD

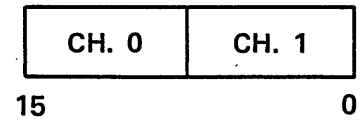


TYPES OF DATA STORED IN MEMORY

* INSTRUCTIONS
  1,2,3 WORDS USED FOR AN INSTRUCTION
  MUST BE IN MEMORY TO BE ABLE TO BE EXECUTED
* NUMBERS
* INTEGERS



* FLOATING POINT
  FP ARE MANIPULATED BY SOFTWARE SUBROUTINES
* CHARACTERS (IN ASCII)



INSTRUCTION EXAMPLE

```
0035                                    LR1*    BUFF1
0035 P0013 0481
0035 P0014 C0F6
0036                                    LR2*    BUFF1,1
0036 P0015 048A
0036 P0016 C0F4
0037                                    LR3*    (ABUF1),2
0037 P0017 04D3
0037 P0018 C0F9
0038                                    LR4+    BUFF1,3
0038 P0019 045C
0038 P001A C000
0038 P001B 000B P
0039                                    LRA     BUFF1,4
0039 P001C 04A6
0039 P001D C000
0039 P001E FFEC
0040                                    LRQ     (ABUF1),A
0040 P001F 04F5
0040 P0020 C000
0040 P0021 FFF0
0041                                    LRI+    (ABUF1),Q
0041 P0022 046F
0041 P0023 C000
0041 P0024 8012 P
```

CHARACTER REPRESENTATION

```
0049 P002F 4142    CHAR        ALF     *,ABCDEFGHIJKLMNOPQRSTUVWXYZ
     P0030 4344                          123456789*
     P0031 4546
     P0032 4748
     P0033 494A
     P0034 4B4C
     P0035 4D4E
     P0036 4F50
     P0037 5152
     P0038 5354
     P0039 5556
     P003A 5758
     P003B 595A
     P0030 3132
     P003D 3334
     P003E 3536
     P003F 3738
     P0040 3920
```

INPUT/OUTPUT INSTRUCTIONS

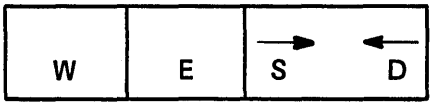
INP

DATA → A
OR
STATUS → A

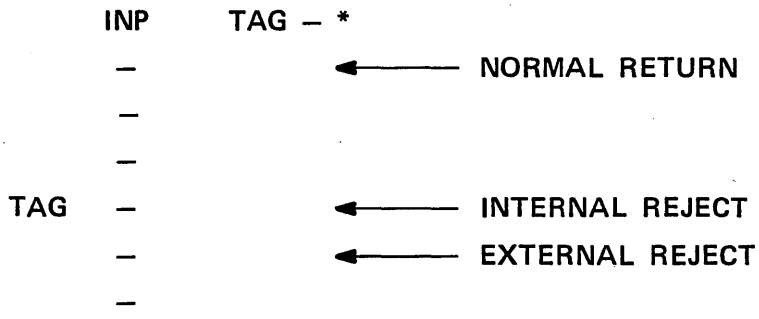
OUT

DATA A →
OR
FUNCTION A →

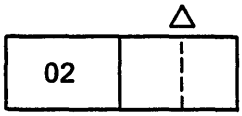
Q CONTAINS THE PERIPHERAL DEVICE'S ADDRESS



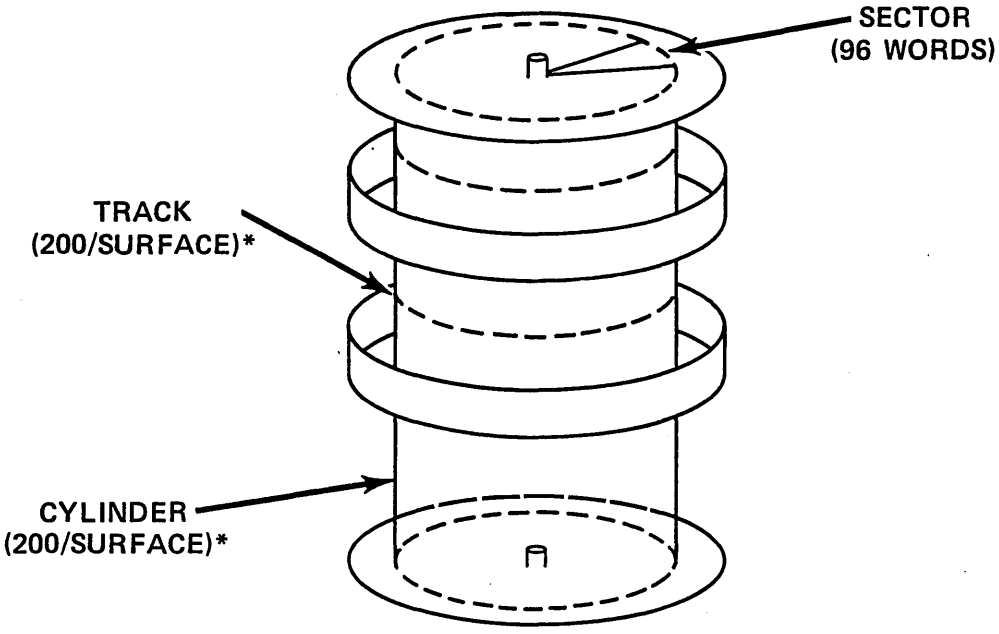
RETURN FROM INP OR OUT



INP MACHINE INSTRUCTION



DISK ADDRESSING



*FOR THE 1866–12

PANEL MODE OPERATION (CYBER 18 ONLY)

ESC BLUE KEY	
J	FUNCTION CONTROL (CHANGE VALUE OF DIGIT)
L	DISPLAY 0
K	DISPLAY 1
H	HALT PROCESSOR
I	START PROCESSOR
@	RETURN TO CONSOLE MODE
G	RUN, DO NOT RETURN TO CONSOLE MODE

FORMAT OF J ENTRY

E.G. J 0 2 G

SET DIGIT 0 TO 2
I.E. SELECT I TO
DISPLAY OR CHANGE

FORMAT OF L & K ENTRY

LG KG	DISPLAY SELECTED REGISTER OR MEMORY LOCATION (∝ P)
LhhhhG	ENTER "hhhh" VALUE TO SELECTED
KhhhhG	REGISTER OR MEMORY

NOTE: IF MORE THAN 4 h'S ENTERED, THE LAST 4 WILL BE TAKEN; IF LESS THAN 4, THE ONES TYPED WILL BE HIGH ORDER BITS

PANEL MODE

FUNCTION CONTROL REGISTER (FCR)

COMMENTS DEVICE HAS 2 MODES

1. CONSOLE MODE
2. PANEL MODE

   [ESC] GO TO PANEL MODE

   (a) OR G GO TO CONSOLE MODE

PURPOSE OF PANEL MODE IS TO GIVE THE OPERATOR A METHOD OF LOOKING AT OR MANIPULATING THE FCR. THE OPERATOR MAY THEN DETERMINE STATUS OF THE CONTROL PROCESSOR, SELECT PROCESSOR FUNCTIONS AND LOOK AT OR CHANGE MEMORY/REGISTERS. MAY BE USED FOR SYSTEM DEBUGGING.

STATUS

HAS OVERFLOW OCCURRED? (∝ SNO, SPE INSTRUCTIONS)

IS A PROTECTED INSTRUCTION BEING EXECUTED?

HAS THE PROTECT FAULT SWITCH BEEN SET?
(∝ SNF, SPF INSTRUCTIONS)

HAS THE PARITY ERROR SWITCH BEEN SET?
(∝ SPE, SNP INSTRUCTIONS)

IS THE INTERRUPT SYSTEM ACTIVE?

IS THE AUTO-START ENABLED?

IS MICRO RUNNING?

IS MACRO RUNNING?

FUNCTIONS

SELECT STEP MODE

SET PROTECT SWITCH

SELECT MULTI-LEVEL INDIRECT ADDRESSING

SELECTIVE STOP (∝ TO SLS INSTRUCTION)

SELECTIVE SKIP (∝ SWS, SWN INSTRUCTIONS)

BREAKPOINT (IF BREAKPOINT BOARD IS PRESENT)

DISPLAY/CHANGE

MEMORY

A

Q

M

X

P

NOTE: CANNOT ACCESS R1-R4

STUDY QUESTIONS - 2

1. What type of thing causes an interrupt?

2. Under what conditions does the CPU respond to an interrupt?

3. What does it mean to take status on a device? What type of information is received?

4. What happens when a parity error is detected?

5. How long will the CPU execute after a power failure?

6. How is a sector addressed on a disk?

7. What is the A/O Channel?

8. What conditions cause a Protect Violation?

9. Where would I find the meaning of the STATUS bits for a particular device?

LESSON GUIDE 3

SOFTWARE OVERVIEW

LESSON PREVIEW:

This lesson will discuss the priority scheme and system methods used to implement the system; i.e. interrupts, MASKT, PRLVL, interrupt stack, scheduler's queue. Terms and concepts basic to the understanding and discussion of the subjects to be covered in later lessons will be reviewed. The details of the libraries, software organization, and core and mass memory will also be discussed.

REFERENCES:

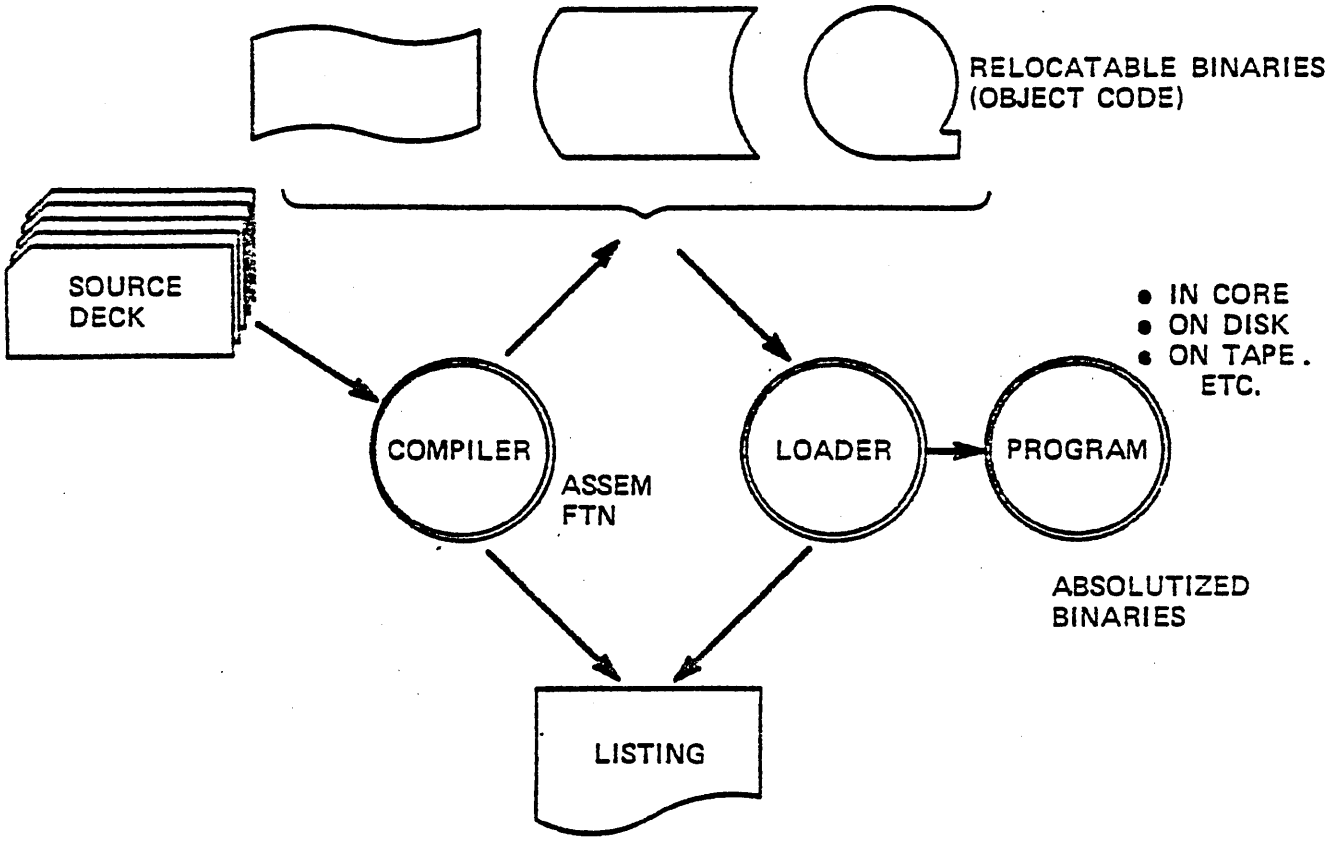
Glossary
Listing of SYSDAT and INSTALL

OBJECTIVES:

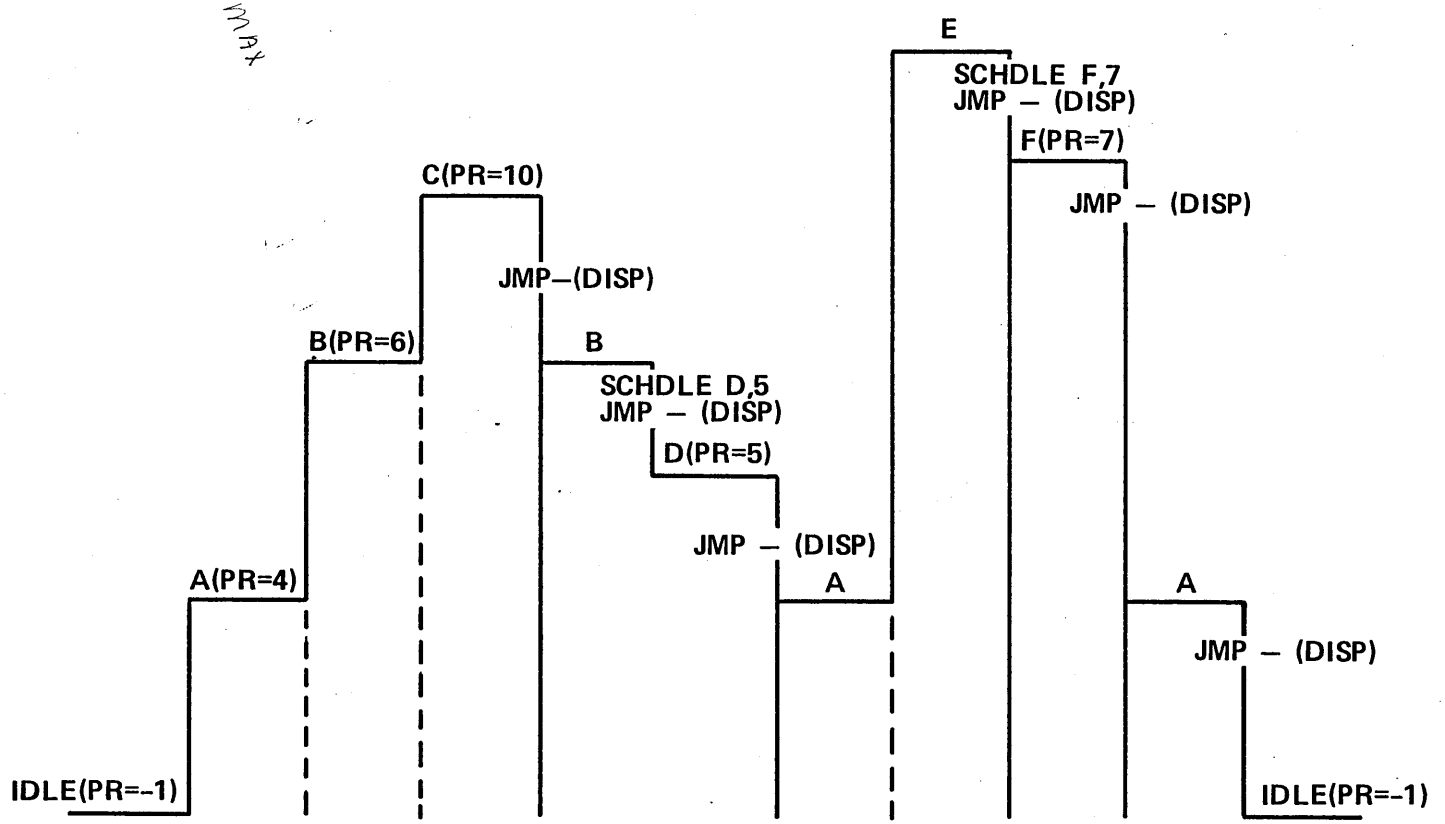
At the completion of this lesson, the student will be able to:

1. Understand the significance of the priority scheme.
2. Discuss the details of maintaining the priority scheme.
3. Explain the purpose of the interrupt stack and scheduler's queue.
4. Discuss the system terms that are necessary to understand the operating system.
5. Describe the flow from a user program to the operating system and back to the user.
6. Obtain information from a dump of core or disk.

COMPILE, LOAD, EXECUTE PROCESS



PRIORITY SCHEME



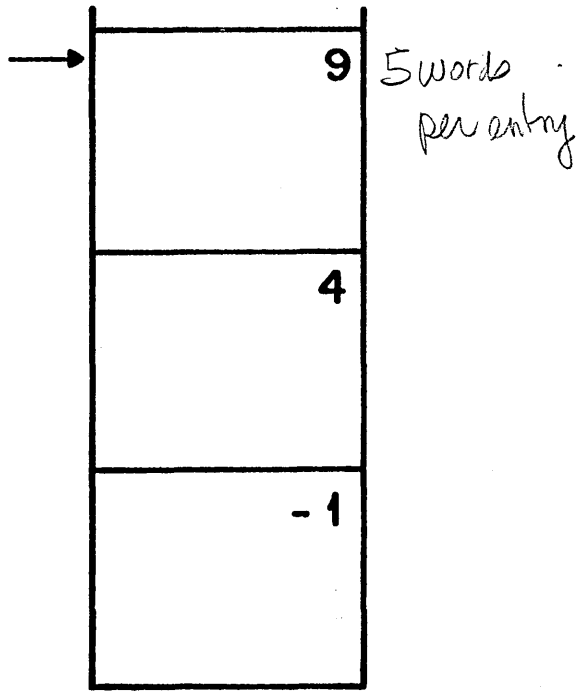
PRIORITY STRUCTURE

- 16 PRIORITY LEVELS

- PRIORITY LEVEL CHANGED IN 'M' REGISTER AND PRLVL

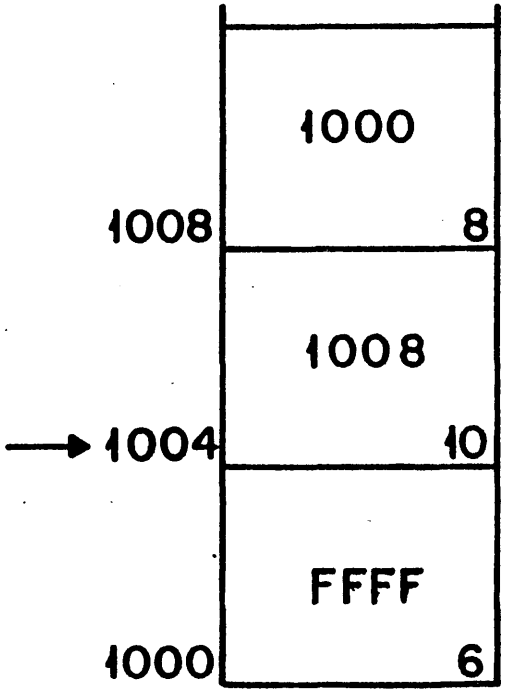
'UNCOMPLETED' PROGRAMS

INTERRUPT STACK



'SCHEDULED' NEW PROGRAMS

SCHEDULER QUEUE



372.7

For interrupts
hardware checks line 0 first

MASK TABLE

INT. Lines

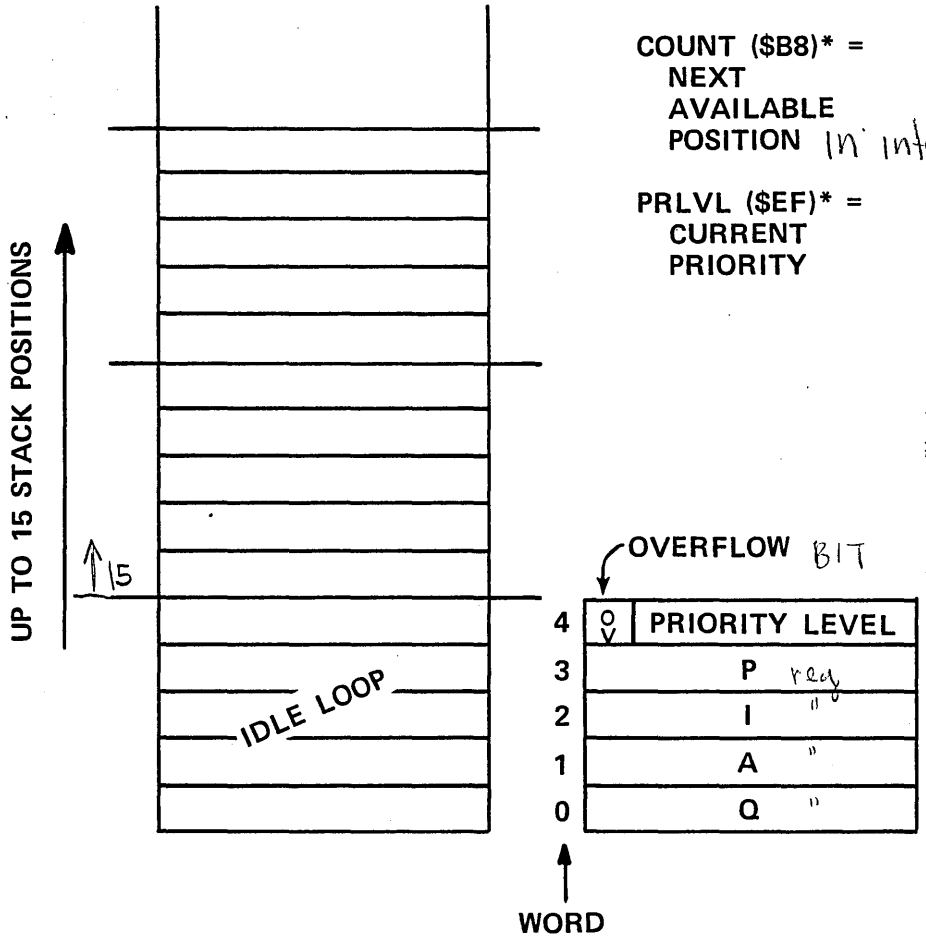
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
0	1	0	0	1	0	0	0	1	0	0	1	1	0	1	1	-1
																0
																1
																2
																3
																4
																5
																6
																7
0	1	0	0	1	0	0	0	1	0	0	1	1	0	1	1	8
0	1	0	0	1	0	0	0	1	0	0	1	0	0	1	1	9
														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

priority levels

line 3 is handled at prlvl 9

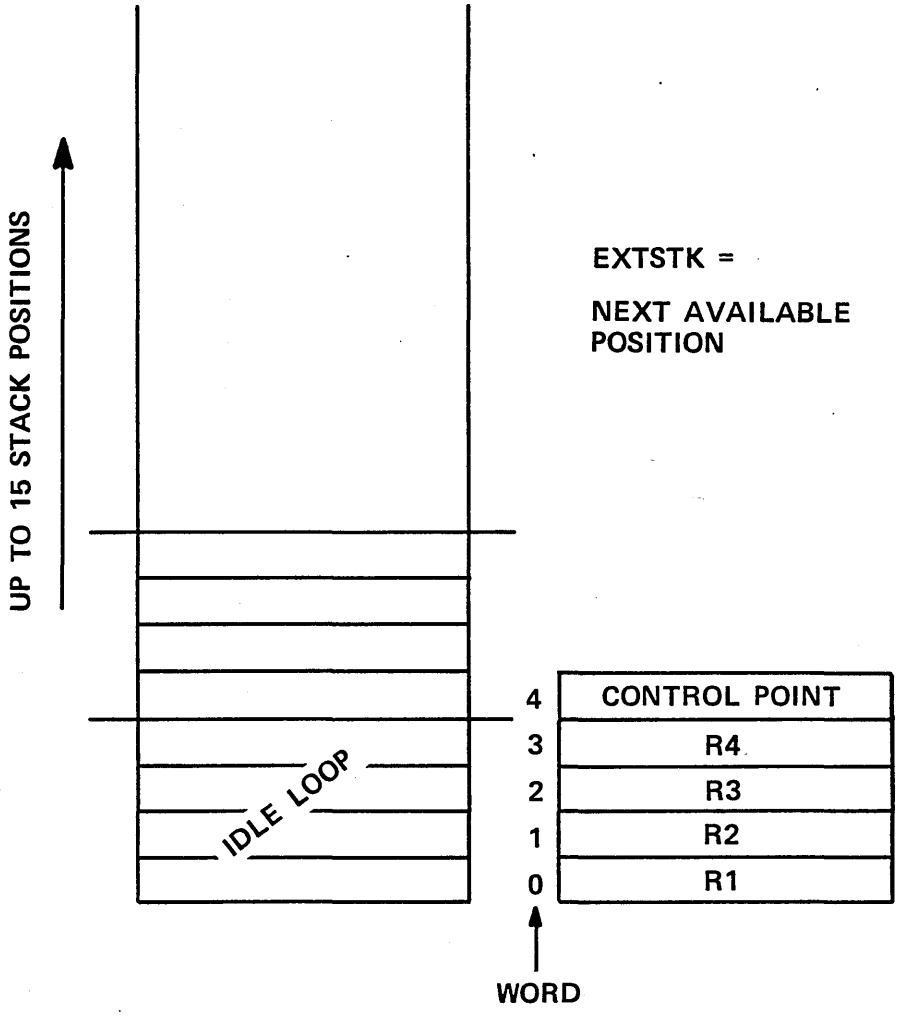
have line on up to level where driver runs.

BASIC INTERRUPT STACK
CYBER 18 HAS TWO!
(AN EXTENDED STACK)

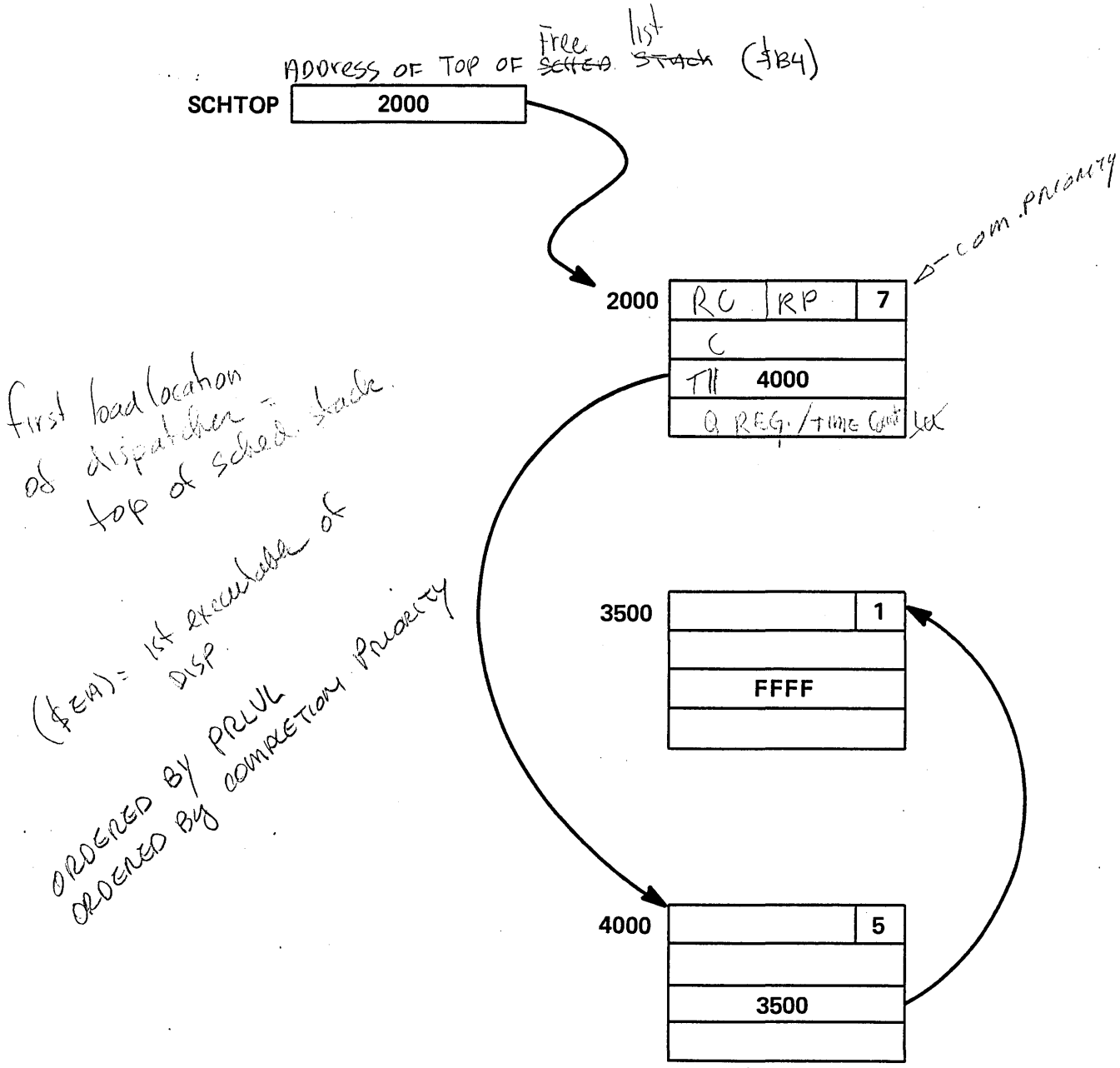


* SYSDAT next

EXTENDED
INTERRUPT STACK
FOR REGISTERS R1-R4



SCHEDULER'S QUEUE — A LIST OF ALL PROGRAMS WAITING TO GO INTO EXECUTION FOR THE FIRST TIME



LIBRARIES

1. PROGRAM LIBRARY

   - BACKGROUND

     *BATCH

2. SYSTEM LIBRARY

   - FOREGROUND

     SYSTEM PROGRAMS
     FILE MANAGER

TWO LIBRARIES

PROGRAM LIBRARY – BACKGROUND

2 TYPES OF ENTRIES

- PROGRAMS IN RELOCATABLE BINARY FORM
- FILES
  DATA
  PROGRAMS IN ABSOLUTIZED BINARY FORM

SYSTEM LIBRARY – FOREGROUND

2 TYPES OF ENTRIES

- CORE RESIDENT
- MASS MEMORY

} ABSOLUTIZED BINARY FORM

PROGRAM LIBRARY

HOW TO ACCESS ENTRIES IN THE LIBRARY

PROGRAMS

1. UNDER *BATCH

   EXAMPLE:

   *JOB, USELIB, CDCIJ, EXECUTE PROGRAM

   *EXLIB

   $6_{7_{8_9}}$

2. LOADER MACRO

FILES

1. GTFILE MACRO
2. DIRECT MASS MEMORY READ

PROGRAM LIBRARY

HOW TO PUT SOMETHING IN THE LIBRARY:

PROGRAMS

```
*JOB,EX2,CDCIJ,  PUT A PROGRAM AS PROGRAM ON THE PROGRAM LIBRARY
*FTN
 OPT LXC
     PROGRAM     WRITE2
     WRITE       (3,100)
100  FORMAT      (*  / / / / THIS IS ANOTHER EXAMPLE ////*)
     END
 MON
*LIBEDT
*K,I8,P8

*L,WRITE2
*DL
*Z
*LIBEDT
*R,WRITE2,
*Z
```

6
7
8
9

FILES

DATA

```
*JOB, DATAF, CDCIJ, PUT DATA FILE IN LIBRARY
*LIBEDT
*K, I10
*N, ABC,,, A
    DATA CARDS
    DATA CARDS
*Z
```

6
7
8
9

PROGRAM IN ABSOLUTE FORM

```
*JOB,EX2,CDCIJ,  PUT A PROGRAM AS A FILE ON THE PROGRAM LIBRARY
*FTN
 OPT LXC
     PROGRAM     WRITE2
     WRITE       (3,100)
100  FORMAT      (*  / / / / THIS IS ANOTHER EXAMPLE ////*)
     END
 MON
*LIBEDT
*K,I8,P8
*P,F
*N,WRITE2,,,B
*DL
*Z
*LIBEDT
*R,WRITE2,F
*Z
```

$6_{7_{8_9}}$

HOW TO FIND OUT WHAT IS IN PROGRAM LIBRARY

***JOB,LISTLB, CDCIJ, LIST PROGRAM LIBRARY**

***LIBEDT**

***DL**

***Z**

$6_{7_{8_9}}$

```
JOB.FTNEX.CDCIJ.   PUT A PROGRAM IN THE PROGRAM LIBRARY
1700 MASS STORAGE OPERATING SYSTEM VERSION 5.0    DATE OF RUN: 08/31/78   SYSTEM ID:  ITUS 1.2 DEMO SYSTE

                FFFFFFFFFFFF     TTTTTTTTTTTTT     NNN       NNN     EEEEEEEEEEEEE     XXX       XXX
                FFFFFFFFFFFF     TTTTTTTTTTTTT     NNN       NNN     EEEEEEEEEEEEE     XXX       XXX
                FFFFFFFFFFFF     TTTTTTTTTTTTT     NNN       NNN     EEEEEEEEEEEEE     XXX       XXX
                FFF                  TTT           NNNNN     NNN     EEE                XXX     XXX
                FFF                  TTT           NNNNNN    NNN     EEE                 XXX   XXX
                FFF                  TTT           NNNNNN    NNN     EEE                  XXX XXX
                FFFFFFFFFFFF         TTT           NNN NNN   NNN     EEEEEEEEEEE           XXXXX
                FFFFFFFFFFFF         TTT           NNN  NNN  NNN     EEEEEEEEEEE            XXX
                FFFFFFFFFFFF         TTT           NNN   NNN NNN     EEEEEEEEEEE           XXXXX
                FFF                  TTT           NNN    NNNNNN     EEE                  XXX XXX
                FFF                  TTT           NNN     NNNNN     EEE                 XXX   XXX
                FFF                  TTT           NNN      NNNN     EEE                XXX     XXX
                FFF                  TTT           NNN       NNN     EEEEEEEEEEEEE     XXX       XXX
                FFF                  TTT           NNN       NNN     EEEEEEEEEEEEE     XXX       XXX
                FFF                  TTT           NNN       NNN     EEEEEEEEEEEEE     XXX       XXX

*FTN

FTN 3.38 (OPT = LXC)              WRITE1        PAGE   1       DATE: 08/31/78    TIME: 1422

      1            PROGRAM    WRITE1
      2            WRITE      (3,100)
      3      100   FORMAT     (* / / / / / /    THIS IS AN EXAMPLE  / / / / / / *)
      4            END

FTN 3.38 (OPT = LXC)              WRITE1        PAGE   2       DATE: 08/31/78    TIME: 1422

PROGRAM LENGTH 0002A (  ...  36)

EXTERNALS
GBSTP  GBGINI

FTN 3.38 (OPT = LXC)              WRITE1        PAGE   3       DATE: 08/31/78    TIME: 1422

***** L I S T   O F   S Y M B O L S *****

EXTERNALS :
---------

  NAME     TYPE       ADDRESS   REFERENCED BY STATEMENT NB :

 GBGINI   INTEGER.FN.  0002
 GBSTP    INTEGER.FN.  0023
```

```
LABELED STATEMENTS :
------------------

  LABEL                ADDRESS  REFERENCED BY STATEMENT NB :

 100                    0008    1,3
 WRITE1                 0000    1




*LIBEDT
  LIB

  IN

  *K,18
  IN

  *L,WRITE1
  IN

  *Z
.*
*WRITE1



 / / / / /   THIS IS AN EXAMPLE  / / / / / /
  STOP
*LIBEDT
  LIB

  IN

  *R,WRITE1
  IN

  *Z
```

```
JOB.EX2.CDCIJ.  PUT A PROGRAM AS A FILE ON THE PROGRAM LIBRARY
1700 MASS STORAGE OPERATING SYSTEM VERSION 5.0     DATE OF RUN: 08/31/78   SYSTEM

  {
FTN
  {
FTN 3.3B (OPT = LXC)                WRITE2        PAGE   1       DATE: 08/31/78

      1             PROGRAM    WRITE2
      2             WRITE      (3,100)
      3       100   FORMAT     (*  / / / / THIS IS ANOTHER EXAMPLE ////*)
      4             END

FTN 3.3B (OPT = LXC)                WRITE2        PAGE   2       DATE: 08/31/78


PROGRAM LENGTH 00020 (    32)

EXTERNALS
QBSTP  QBGIN1
   {
   }
*LIBEDT
 LIB

 IN

 *K,IB,PB
 IN

 *P,F
   WRITE2  7000
   QBGIN1  7020    DECK-ID H04  FTN 3.3 RUNTIME      SUMMARY-117
   QBCLIL  70F6    DECK-ID H05  FTN 3.3 RUNTIME      SUMMARY-102
   QBCMP   7117    DECK-ID H06  FTN 3.3 RUNTIME      SUMMARY-116
   QBRWBU  71F5    DECK-ID H07  FTN 3.3 RUNTIME      SUMMARY-102
   QBERRP  730E    DECK-ID H08  FTN 3.3 RUNTIME      SUMMARY-102
   QBDF1C  73E5    DECK-ID H09  FTN 3.3 RUNTIME      SUMMARY-116
   QBQX    74AE    DECK-ID H10  FTN 3.3 RUNTIME      SUMMARY-116
   QBCUN1  750D    DECK-ID H11  FTN 3.3 RUNTIME      SUMMARY-102
   QBFGET  7596    DECK-ID H12  FTN 3.3 RUNTIME      SUMMARY-116
   QBMAGT  75FE    DECK-ID H13  FTN 3.3 RUNTIME      SUMMARY-117
   TAPCUR  764E    DECK-ID H14  FTN 3.3 RUNTIME      SUMMARY-117
   PSSTOP  76EE    DECK-ID H16  FTN 3.3 RUNTIME      SUMMARY-126
   QBPARD  772A    DECK-ID H17  FTN 3.3 RUNTIME      SUMMARY-126
   PARABN  7798    DECK-ID 612  FTN 3.3 RUNTIME      SUMMARY-111
   QBIFPP  77AA    DECK-ID H01  FTN 3.3 RUNTIME      SUMMARY-106
   QBFS    77E9    DECK-ID H02  FTN 3.3 RUNTIME      SUMMARY-106
   QBTRAN  7A1F    DECK-ID H03  FTN 3.3 RUNTIME      SUMMARY-115
   QBEXP1  825F    DECK-ID H18  FTN 3.3 RUNTIME      SUMMARY-102
   QBEXP6  82DA    DECK-ID H19  FTN 3.3 RUNTIME      SUMMARY-102
   FLOTN   8396    DECK-ID G14  FTN 3.3 RUNTIME      SUMMARY-112
   COMAFP  85D4    DECK-ID G15  FTN 3.3 RUNTIME      SUMMARY-114
   DELDMY  8708    DECK-ID K19  FTN 3.3 RUNTIME      SUMMARY-102
   QBPRMS  870A    DECK-ID S76  RPGII 2.0            SUMMARY-126
 IN
```

```
*N,WRITE2...B
IN

*DL
LIBMAC  SECT.    0C59
ASSEM   SECT.    0C71
ASSIM   SECT.    0C71
PASS1   SECT.    0C77  FILE
PASS2   SECT.    0CA3  FILE
PASS3   SECT.    0CBD  FILE
TABLST  SECT.    0CE2  FILE
XREF    SECT.    0CF7  FILE
MACSKL  SECT.    0D08  FILE
MACROS  SECT.    0F79  FILE
FTN     SECT.    0F8A
EXITF   SECT.    0F8A
PAGCHK  SECT.    0F8A
ASCOPT  SECT.    0F8A
PRGNAM  SECT.    0F8A
PAGNBR  SECT.    0F8A
DATE    SECT.    0F8A
TIME    SECT.    0F8A
FTN3A1  SECT.    0F93  FILE
FTN3A2  SECT.    1004  FILE

                 ≀
                 ≀
                 ≀

BM2EN1  SECT.    2D78  FILE
BMIN1   SECT.    3067  FILE
BMIN2   SECT.    3081  FILE
BMIN3   SECT.    309B  FILE
START   SECT.    2DC2
PRINT   SECT.    2DC2
WRITE2  SECT.    30B5  FILE

FINI
IN

*Z
*LIBEDIT
LIB

IN

*R,WRITE2,F
IN

*Z
```

EXAMPLE
LIST OF PROGRAM LIBRARY

```
#LIBEDT
*DL
LIBMAC  SECT.    0C59
ASSEM   SECT.    0C71
ASSIM   SECT.    0C71
PASS1   SECT.    0C77  FILE
PASS2   SECT.    0CA3  FILE
PASS3   SECT.    0CBD  FILE
TABLSI  SECT.    0CE2  FILE
XREF    SECT.    0CF7  FILE
MACSKL  SECT.    0D08  FILE
MACHUS  SECT.    0F79  FILE
FIN     SECT.    0F8A
EXITF   SECT.    0F8A
PAGCHK  SECT.    0F8A
ASCOPT  SECT.    0F8A
PRGNAM  SECT.    0F8A
PAGNBR  SECT.    0F8A
DATE    SECT.    0F8A
TIME    SECT.    0F8A
FTN3A1  SECT.    0F93  FILE
FTN3A2  SECT.    1004  FILE
```

FTN3A3	SECT.	102C	FILE
FTN3A4	SECT	1057	FILE
FTN3A5	SECT.	1080	FILE
FTN3B1	SECT.	10A3	FILE
FTN3C1	SECT.	1125	FILE
FTN3D1	SECT.	1197	FILE
FTN3E1	SECT.	11F7	FILE
FTN3F1	SECT.	125C	FILE
FTN3ER	SECT.	12A2	FILE
READ	SECT.	12FD	
WRITE	SECT.	12FD	
FREAD	SECT.	12FD	
FWRITE	SECT.	12FD	
SCHEDL	SECT.	12FD	
TIMER	SECT.	12FD	
DISPAT	SECT.	12FD	
DISP	SECT.	12FD	
LINK	SECT	12FD	
ICLOCK	SECT	12FD	
INPINS	SECT.	12FD	
OUTINS	SECT.	12FD	
RELESE	SECT.	12FD	
ICONCT	SECT.	12FD	
OCONCT	SECT.	12FD	
Q8PREP	SECT.	130C	
Q8PKUP	SECT.	130C	
Q9PKUP	SECT.	130C	
Q60F21	SECT.	1311	
Q8GI2F	SECT.	1311	
Q60F2F	SECT.	1311	
RETAD	SECT.	1311	
QSAVE	SECT.	1311	
Q8AB	SECT.	1319	
ABS	SECT.	1319	
SQRT	SECT.	131E	
Q8SG	SECT.	1326	
SIGN	SECT.	1326	
Q60FIX	SECT.	132B	
Q8FX	SECT.	132B	
Q60FLT	SECT.	132B	
Q8FLOT	SECT.	132B	
IFIX	SECT.	132B	
FLOAT	SECT.	132B	
DFIX	SECT.	132B	
Q8DFLT	SECT.	132B	
DFLT	SECT.	132B	
EXP	SECT.	1331	
ALOG	SECT.	1339	
TANH	SECT.	1340	
SIN	SECT.	1347	
COS	SECT.	1347	
ATAN	SECT.	1350	
PARABS	SECT.	1358	
Q81FRM	SECT.	135D	
Q8FS	SECT.	1363	
Q81RAN	SECT.	139C	
Q8GINI	SECT.	13E8	
Q8UNIT	SECT.	13E8	
Q8SKIP	SECT.	13E8	
Q8GEND	SECT.	13F2	

Q8CMP0	SECT.	13F7
Q8CMP1	SECT.	13F7
Q8DEAD	SECT.	13F7
Q8GENS	SECT.	13F7
RECEND	SECT.	13F7
Q8BINB	SECT.	1401
Q8LOCB	SECT.	1401
Q8RWAU	SECT.	1401
Q8INTB	SECT.	1401
Q8BEGB	SECT.	1401
Q8CLRB	SECT.	1401
Q8RINT	SECT.	1401
Q61BUF	SECT.	1401
WRFLG	SECT.	1401
Q8ERRM	SECT.	140C
Q8FERM	SECT.	140C
Q8EREM	SECT.	140C
Q6DFNF	SECT.	1418
Q8DFIN	SECT.	1416
Q8QTOM	SECT.	1421
Q8QTRM	SECT.	1421
Q80X	SECT.	1421
Q8MOVE	SECT.	1421
Q80Y	SECT.	1421
Q80Z	SECT.	1421
Q8QUN1	SECT.	1428
Q8QUN2	SECT.	1428
Q8QUN3	SECT.	1428
Q8FGET	SECT.	142E
Q8FPUT	SECT.	142E
Q8LOCF	SECT.	142E
Q81GP	SECT.	142E
Q8MAGT	SECT.	1436
Q8EOTT	SECT.	1436
Q8UBCK	SECT.	143C
Q8UFLE	SECT.	143C
Q8QWND	SECT.	143C
EOF	SECT.	143C
IOCK	SECT.	1444
Q8PSE	SECT.	1449
Q8PSEN	SECT.	1449
Q8STP	SECT.	1449
Q8STPN	SECT.	1449
Q8COM1	SECT.	1449
Q8PAND	SECT.	1450
Q8EXP1	SECT.	1457
Q8EXP9	SECT.	145E
Q8EXPT	SECT.	145E
Q8EXP2	SECT.	145E
Q8QGET	SECT.	1468
SETBFR	SECT.	1468
ENCODE	SECT.	146C
DECODE	SECT.	146C
COMMON	SECT.	1473
ISAVE	SECT.	1473
IGETCH	SECT.	1477
GETCH	SECT.	1477
IPACK	SECT.	147C
UPDATE	SECT.	1482
DECPL	SECT.	1486

```
INTGR   SECT.   148B
SPACEX  SECT.   1490
HOLRTH  SECT.   1495
QUOTE   SECT.   1495
DCHX    SECT.   149D
HXASC   SECT.   14A5
AFRMOT  SECT.   14AB
RFRMOT  SECT.   14B0
AFRMIN  SECT.   14B5
RFRMIN  SECT.   14BB
ASCHX   SECT.   14C0
HXDC    SECT.   14C6
FLOTIN  SECT.   14CE
FOUT    SECT.   14D4
EOUT    SECT.   14DC
EWRITE  SECT.   14E6
INITLI  SECT.   14EB
RESTRE  SECT.   14EB
FORMTR  SECT.   14F0
CHCNT   SECT.   14FC
Q80FI   SECT.   1502
Q80FL   SECT.   1507
Q80FX   SECT.   150B
HEXASC  SECT.   1511
HEXDEC  SECT.   1516
ASCII   SECT.   1518
DECHEX  SECT.   1520
AFORM   SECT.   1525
RFORM   SECT.   152A
FLOATG  SECT.   152F
FLOT    SECT.   1534
HFLOT   SECT.   1534
IFALT   SECT.   1546
SFALT   SECT.   1546
OPERND  SECT.   1546
NXTOP   SECT.   1546
FPEROR  SECT.   1546
PROCHK  SECT.   1546
SPECOP  SECT.   1546
FLOFOP  SECT.   1546
FIXFOP  SECT.   1546
Q8DXP1  SECT.   1553
Q8DXP9  SECT.    553
DOUT    SECT.   1553
LULIST  SECT.   1557
LISTR   SECT.   1580
OPSORT  SECT.   158C
HGNRD   SECT.   158C
EESORT  SECT.   15AC
COSY    SECT.   15C4
LCOSY   SECT.   1613
CYFT    SECT.   161E
IOUP    SECT.   1626
IOUPV4  SECT.   162F  FILE
DTLF    SECT.   164C
DSKTAP  SECT.   1654  FILE
LIHILD  SECT.   1673
LIBIDO  SECT.   1677  FILE
HELPER  SECT.   1682  FILE
SKED    SECT.   1688

SKFILE  SECT.   16BC  FILE
SILP    SECT.   16E9
SI      SECT.   16F1  FILE
SMDMPI  SECT.   1746
MPSMDI  SECT.   174E  FILE
SMDMPT  SECT.   1751
MPSMDT  SECT.   1759  FILE
TSLOG   SECT.   179D  FILE
ULBUFF  SECT.   17B3  FILE
MNUPKO  SECT.   17E6  FILE
SUBRCM  SECT.   17F8
GETCHR  SECT.   1835
PUTCHR  SECT.   1835
CREATE  SECT.   183A
CLEAR   SECT.   183A
DELETE  SECT.   183A
OPENFL  SECT.   183A
CLOSFL  SECT.   183A
LOKFIL  SECT.   183A
UNLFIL  SECT.   183A
GETFCB  SECT.   183A
UPDFCB  SECT.   183A
RENAME  SECT.   183A
PUTS    SECT.   183A
WRITER  SECT.   183A
READR   SECT.   183A
GETS    SECT.   183A
UPDREC  SECT.   183A
DELREC  SECT.   183A
COMFIL  SECT.   183A
VOLUSE  SECT.   183A
REDUCE  SECT.   183A
USERID  SECT.   1843  FILE
SYMSGF  SECT.   1855  FILE
SYMENU  SECT.   19FD  FILE
PROCED  SECT.   1A0E  FILE
UTIL    SECT.   1A12  FILE
UTBATC  SECT.   1AA4  FILE
UTDISC  SECT.   1B03  FILE
UTHOST  SECT.   1B13  FILE
UTSET   SECT.   1B21  FILE
UTPRIN  SECT.   1B2E  FILE
UTBATS  SECT.   1B4C  FILE
UTDISP  SECT.   1B6D  FILE
UTFLUS  SECT.   1B86  FILE
UTINIT  SECT.   1B99  FILE
UTDEFI  SECT.   1BA9  FILE
UTSTAT  SECT.   1BBD  FILE
UTDELE  SECT.   1BDB  FILE
UTCLEA  SECT.   1BE1  FILE
UTLIST  SECT.   1BE7  FILE
UTRENA  SECT.   1C02  FILE
UTCOMM  SECT.   1C0C  FILE
UTMOUN  SECT.   1C10  FILE
UTDISM  SECT.   1C19  FILE
UTSAVE  SECT.   1C21  FILE
UTPURG  SECT.   1C6C  FILE
UTCOMP  SECT.   1C74  FILE
UTDUMP  SECT.   1C7D  FILE
UTRELO  SECT.   1CBB  FILE
```

UTCOPY	SECT.	1CF2	FILE	Y9ITLP	SECT.	2157
UTLOAD	SECT.	1D2D	FILE	Y9LAHD	SECT.	2157
UTOKLD	SECT.	1D67	FILE	Y9MMOV	SECT.	2157
UTRMLD	SECT.	1DB7	FILE	Y9NSQH	SECT.	2157
EDITOR	SECT.	1E23	FILE	Y9PAGE	SECT.	2157
RMUOPN	SECT.	1EB0	FILE	Y9TBOT	SECT.	2157
RPGMU2	SECT.	1EF6	FILE	Y9INVP	SECT.	2157
RPGMU3	SECT.	1F2B	FILE	Y9DSEC	SECT.	2157
RPGMU4	SECT.	1F7A	FILE	Y9FBSE	SECT.	2157
RMUCLO	SECT.	1FAF	FILE	POSSPB	SECT.	2157
RMUCSE	SECT.	1FE3	FILE	POSSKB	SECT.	2157
RPGSM0	SECT.	2087	FILE	POSSPA	SECT.	2157
RPGSM1	SECT.	20A9	FILE	POSSKA	SECT.	2157
RPGSM2	SECT.	20B1	FILE	R9OCOD	SECT.	2157
RPGSM3	SECT.	20E1	FILE	R9RECP	SECT.	2157
RPGSM4	SECT.	20E9	FILE	R9TANF	SECT.	2157
RPGSM5	SECT.	20EC	FILE	R9TFLG	SECT.	2157
RPGSM6	SECT.	2102	FILE	Y9APFX	SECT.	2157
RPGSM7	SECT.	211D	FILE	Y9ASPC	SECT.	2157
RPGSME	SECT.	212D	FILE	Y9COUN	SECT.	2157
RPGSM9	SECT.	213D	FILE	Y9DTPT	SECT.	2157
RPGII	SECT.	2145		Y9FDPR	SECT.	2157
RPGXX	SECT.	2145		Y9FIPR	SECT.	2157
RPGYY	SECT.	2145		Y9FIPT	SECT.	2157
RPGZZ	SECT.	2145		Y9FLPT	SECT.	2157
CATLOG	SECT.	2149		Y9FPTL	SECT.	2157
CATSEG	SECT.	2149		Y9FPTR	SECT.	2157
RPGFIL	SECT.	214D		Y9FSSA	SECT.	2157
R9CNTR	SECT.	214D		Y9HIND	SECT.	2157
R9JUMP	SECT.	214D		Y9HNUM	SECT.	2157
R9SGTB	SECT.	214D		Y9IBUF	SECT.	2157
R9SGIX	SECT.	214D		Y9KALA	SECT.	2157
R9CKSG	SECT.	214D		Y9KAPF	SECT.	2157
R9MUNO	SECT.	214D		Y9PSFG	SECT.	2157
ATTCHK	SECT.	214D		Y9RECP	SECT.	2157
R9BRAK	SECT.	214D		Y9RPTR	SECT.	2157
R9ROOT	SECT.	2157		Y9TOP1	SECT.	2157
Y9PFCB	SECT.	2157		Y9VFT1	SECT.	2157
Y9FDC1	SECT.	2157		Y9XRPF	SECT.	2157
Y9CMST	SECT.	2157		Y9XPTE	SECT.	2157
Y9MMST	SECT.	2157		DMPTLK	SECT.	2157
Y91NMK	SECT.	2157		Y9TRCE	SECT.	2157
Y9MHLD	SECT.	2157		Y9ERCD	SECT.	2157
Y9DETL	SECT.	2157		Y91REG	SECT.	2157
Y9TOTL	SECT.	2157		R9ERTN	SECT.	2157
Y9LSTR	SECT.	2157		Y9LABL	SECT.	2157
Y9DOTT	SECT.	2157		Y9FTNX	SECT.	2157
Y9TOTT	SECT.	2157		Y9FSTL	SECT.	2157
Y9EOTT	SECT.	2157		R9USER	SECT.	2157
Y9ALSQ	SECT.	2157		R9UNIT	SECT.	2157
Y9CARA	SECT.	2157		R9MODE	SECT.	2157
Y9MARA	SECT.	2157		R9PORT	SECT.	2157
Y9UDAT	SECT.	2157		R9EDT1	SECT.	2157
Y9UDAY	SECT.	2157		R9EDT2	SECT.	2157
Y9UYER	SECT.	2157		R9EDT3	SECT.	2157
Y9UMTH	SECT.	2157		R9EDT4	SECT.	2157
Y9FD1B	SECT.	2157		R9RPRT	SECT.	2169
Y9TB1O	SECT.	2157		R91NTB	SECT.	2169
Y9CMOV	SECT.	2157		R9CLIN	SECT.	2169
Y91NTA	SECT.	2157		R9MTIN	SECT.	2169

R9VIND	SECT.	2169	R9MVBX	SECT.	21CE
R9USND	SECT.	2169	R9MIBX	SECT.	21CE
R9MRIN	SECT.	2169	R9MVB	SECT.	21CE
R9FCTR	SECT.	2169	R9LEY	SECT.	21CE
R9STTS	SECT.	2169	R9SBY	SECT.	21CE
R9MLFU	SECT.	2169	R9MIB	SECT.	21CE
R9REPT	SECT.	2169	R9MVB	SECT.	21D4
R9RYST	SECT.	2169	R9FTNX	SECT.	21D8
R9ACC1	SECT.	2169	CVASEB	SECT.	21E5
YACC1B	SECT.	2169	R9FLDL	SECT.	21EA
YACC10	SECT.	2169	R9999B	SECT.	21F0
R9AC1S	SECT.	2169	R9UNPK	SECT.	21F6
R9AC2S	SECT.	2169	SUBRFL	SECT.	21FC
R9AC3S	SECT.	2169	SUBRED	SECT.	2201
R9AC1N	SECT.	2169	SUBRMV	SECT.	2208
R9AC2N	SECT.	2169	SUBRLM	SECT.	2208
R9AC3N	SECT.	2169	SUBRML	SECT.	2208
R9PPUF	SECT.	2169	SUBRIN	SECT.	220F
R9UPOF	SECT.	2169	SUBRAJ	SECT.	2216
R9INR2	SECT.	2169	CATFIL	SECT.	221E FILE
R9INFF	SECT.	2169	SWITCH	SECT.	2231 FILE
R9INPM	SECT.	2169	RBDPCM	SECT.	2235
R9PONT	SECT.	2169	RBDSEG	SECT.	2235
R9CNCL	SECT.	2169	RBDFIL	SECT.	2239 FILE
YERRSW	SECT.	2169	MOUNT	SECT.	224A FILE
R9FTSW	SECT.	2169	TRACER	SECT.	225B FILE
R9VSWT	SECT.	2169	DSORT	SECT.	225D FILE
R9OVSW	SECT.	2169	SMCMON	SECT.	22EE FILE
R9FTIM	SECT.	2169	SMCEDT	SECT.	2306 FILE
R9FFCB	SECT.	2169	SMCSRT	SECT.	2317 FILE
R9CFIL	SECT.	2169	SMCIMG	SECT.	231D FILE
R9NFCB	SECT.	2169	SMCFMG	SECT.	2322 FILE
R9PRGD	SECT.	2169	VTEST1	SECT.	2326 FILE
R9MRSW	SECT.	2169	VTEST2	SECT.	2347 FILE
R9MRPR	SECT.	2169	VTEST3	SECT.	236B FILE
R9LRSW	SECT.	2169	VTEST4	SECT.	238D FILE
R9RYPS	SECT.	2169	VTEST5	SECT.	238D FILE
R9BLTR	SECT.	2169	VTEST6	SECT.	23F0 FILE
R9FRMR	SECT.	2169	VTIMES	SECT.	2412 FILE
R9ACAX	SECT.	2169	JIM	SECT.	242A FILE
R9ACX1	SECT.	2169	FMERR	SECT.	242B
YACAXN	SECT.	2169	HEXDCZ	SECT.	243B
R9SAVE	SECT.	2178	TIMEI1	SECT.	2442 FILE
R9REST	SECT.	2178	TIMEIT	SECT.	2462
R9FLOW	SECT.	217F	PROG1	SECT.	2470 FILE
B9FLOW	SECT.	217F	PROG3	SECT.	249F FILE
N9FLOW	SECT.	217F	PROG5	SECT.	24DB FILE
STRACE	SECT.	2184	PROG6	SECT.	25AC FILE
SYSMSG	SECT.	2189	TIMIT	SECT.	2518 FILE
R9EXIT	SECT.	2197	PROG2	SECT.	253H FILE
R9FSTL	SECT.	219C	PROG4	SECT.	2591 FILE
R9ELOC	SECT.	21A2	PROG2A	SECT.	25DC FILE
R9TRCS	SECT.	21A5	INV	SECT.	2617
R9TROT	SECT.	21B8	PROG7	SECT.	2618 FILE
R9INDR	SECT.	21BE	PROG4A	SECT.	2677 FILE
R9LEL	SECT.	21C5	DMN2	SECT.	26A9 FILE
R9GTL	SECT.	21C5	TECH1B	SECT.	2505
R9MIW	SECT.	21CA	TOUPUT	SECT.	2517
R9SBYX	SECT.	21CE	TWANLZ	SECT.	2579
R9LBYX	SECT.	21CE	TOPUKG	SECT.	2586

```
TQULST  SECT.  258E
TQCOPY  SECT.  25D6
TQSETI  SECT.  26ED
TQGENL  SECT.  2706
TQTINP  SECT.  2706
TQUBUF  SECT.  2706
TQROUT  SECT.  2708
TQMSGT  SECT.  2708
TQGCHR  SECT.  2708
TQSCHR  SECT.  2708
TQCLXY  SECT.  2708
TQUPXY  SECT.  2708
TQRECD  SECT.  2724
TQIDCD  SECT.  2757
TQAJAX  SECT.  275E
TQUMRK  SECT.  2765
TQSTRG  SECT.  2774
TQPRNT  SECT.  2780
TQEDIT  SECT.  2786
TQGENC  SECT.  27AE
TQDCOD  SECT.  27AE
TQCVHA  SECT.  27AE
TQCVHD  SECT.  27AE
TQCVAH  SECT.  27AE
TQCVDH  SECT.  27AE
TQTDAY  SECT.  2AB2
TQMTRX  SECT.  27B8
TQLDCD  SECT.  27E0
TQZGEN  SECT.  27FD
TQSELC  SECT.  2809
TQMADD  SECT.  284A
TQSLST  SECT.  2869
TQMADR  SECT.  2879
TQABIT  SECT.  2879
TQRTRX  SECT.  2879
TQSETS  SECT.  287E
TQSETC  SECT.  28BD
TQUSER  SECT.  28BD
TQCONT  SECT.  28D6
TQLERT  SECT.  28EB
TQLFMU  SECT.  2901
TQSELF  SECT.  291C
TQHLPF  SECT.  2942
TQHDMP  SECT.  296B
TQSRCH  SECT.  296E
TQSORT  SECT.  2985
LOGOTC  SECT.  2E6C  FILE
TQDATE  SECT.  2AB2
TQTIME  SECT.  2AB2
TQZCON  SECT.  29AE
TQCCP1  SECT.  2A1C  FILE
TQCCP2  SECT.  29D9  FILE
TQCCP1  SECT.  2996
TQPRTA  SECT.  2C47
TQZTEC  SECT.  29D2
TQZTIN  SECT.  27FD
TQZBUF  SECT.  27FD
TQZROU  SECT.  27FD
TQZMSG  SECT.  27FD
TQZGCH  SECT.  27FD
TQZSCH  SECT.  27FD
TQZCXY  SECT.  27FD
TQZUXY  SECT.  27FD
TQSHLD  SECT.  2A7B
TQZINP  SECT.  2A85
TQDMPB  SECT.  2A89
TQZHLF  SECT.  2A8D
TQZSEL  SECT.  2BD5
TQZHLP  SECT.  2BD9
TQPRTZ  SECT.  2BEB
TQPRTS  SECT.  2C14
TQPSL1  SECT.  2C3D
TQPSEL  SECT.  2C42
TQDLCH  SECT.  2A89
ZTEC    SECT.  2C65  FILE
TQUTIL  SECT.  2F9F  FILE
TQPSL1  SECT.  2DD1  FILE
TQSPRD  SECT.  2E62
BENCH1  SECT.  2AB8  FILE
BENCH2  SECT.  2AD6  FILE
COMPL   SECT.  2AED  FILE
TXLIST  SECT.  2B5D  FILE
BIOR    SECT.  2B9E  FILE
TQDSCH  SECT.  2A89
TQDMPR  SECT.  2A89
TQZSDA  SECT.  2F90
LOGTEC  SECT.  3194  FILE
BM1ENT  SECT.  2D2A  FILE
BM2ENT  SECT.  2D76  FILE
BMIN1   SECT.  3067  FILE
BMIN2   SECT.  3081  FILE
BMIN3   SECT.  309B  FILE
START   SECT.  2DC2
PRINT   SECT.  2DC2
WRITE2  SECT.  30B5  FILE

FINI
IN

*Z
```

SYSTEM LIBRARY

SYSTEM MACROS INITIATE PROGRAMS IN THE SYSTEM LIBRARY

SCHDLE	**c, p, x, d**	**ALLOCATABLE CORE**
SYSCHD	**c, p**	**PART 1**

HOW TO PUT A PROGRAM IN THE SYSTEM LIBRARY

```
%JOB,SYSLIB,CDCIJ,  PUT A PROGRAM ON THE SYSTEM LIBRARY
%FTN
 OPT LXR
 SOURCE PGM
 THIS PGM MUST BE
 WRITTEN IN A
 SPECIAL FORM TO RUN IN ALLOCATABLE
 CORE
 MON
%LIBEDT
%K,I8,P8
%M,31,,,M,N
%DM
%Z
```

HOW TO DUMP SYSTEM LIBRARY

```
*JOB, LSTLIB, CDCIJ  LIST SYSTEM LIBRARY
*LIBEDT
*DM
*Z
6
7
8
9
```

LIBEDT CONTROL STATEMENTS

MANIPULATE PROGRAM LIBRARY

*L,epn	ADD/REPLACE PROGRAM
*N,n,w_1,w_2,m	ADD/REPLACE FILE
*R,n,F	REMOVE PROGRAM OR FILE
*DL	LIST CONTENTS OF PROGRAM LIBRARY DIRECTORY

MANIPULATE SYSTEM LIBRARY

*A,or,s,n,d,	REPLACE PARTITION PROGRAM
*M,or s,d,M,N	REPLACE SYSTEM LIBRARY ENTRY
*DM	LIST SYSTEM LIBRARY DIRECTORY
*S,or,v,m	SET CORE REQUEST PRIORITY

COPY

*T,i,mi,o,mo,n,f	COPY
*F	TERMINATE *T
*FOK	TRANSFER FOR *T

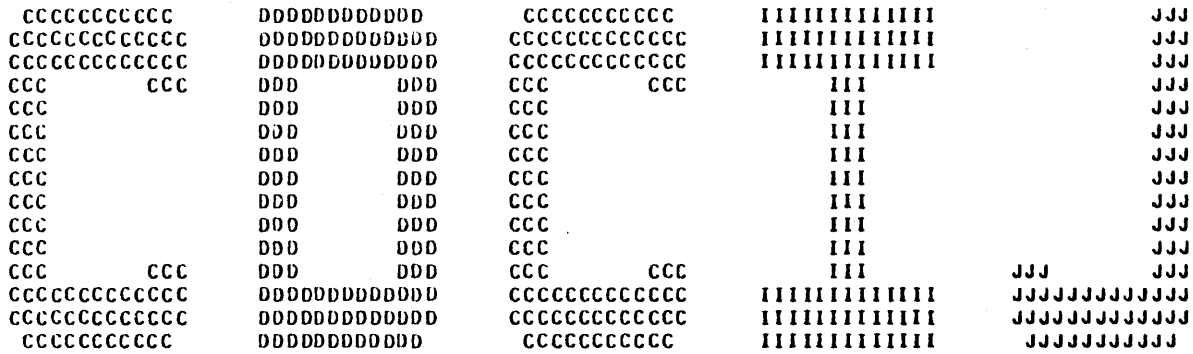
MISCELLANEOUS

*P,n,R/P,Sa	LOAD, COMBINE AND PRODUCE ABSOLUTE RECORD
*K,lu,Plu,Llu	CHANGE STANDARD UNITS
*U	GET CONTROL STATEMENTS FROM COMMENTS DEVICE
*V,lu,m	GET CONTROL STATEMENTS FROM 1U
*Z	TERMINATE LIBEDT

JOB,CDCIJ,**LIST CREPT TABLE

EXAMPLE TO ILLUSTRATE CREPT0 AND CREPT1

1700 MASS STORAGE OPERATING SYSTEM VERSION 5.0 DATE OF RUN: 01/30/79 SYSTEM ID: ARNFAC 18-20 (07/11/78)



3-26.1

*ASSEM

```
0001                        NAM CREPT          LIST CREPT TABLE
0002                    ENT BEGIN
0003 P0000 0800    BEGIN  NOP
0004                      EXT SCHTOP IS IN CREPT1
0005                      EXT LOG1A  IS IN CREPT0
0006                      EXT DUMMY2
0007                *    NOTE-  THIS IS A DO-NOTHING PROGRAM
0008                *           ITS SOLE PURPOSE IS TO LIST THE
0009                *           CORE RESIDENT ENTRY POINT TABLE


0011 P0001 C400 X           LDA+   LOG1A
     P0002 7FFF X
0012 P0003 C400 X           LDA+   DUMMY2
     P0004 7FFF X
0013 P0005 C400 X           LDA+   SCHTOP
     P0006 7FFF X
0014                      END    BEGIN

PGM=  0007 (     7)   COM = 0000 (     0)   DAT = 0000 (     0)
```

*** S Y M B O L T A B L E ***

BEGIN	0002	DUMMY2	0006	I		0000	LOG1A	0005	SCHTOP	0004	

```
*CTO,PLEASE TYPE *E AFTER LOADER ERROR E10
*CTO,THIS WILL CAUSE LOADER TO LINK WITH CREPT
*L,8
    CREPT   6800    LIST CREPT TABLE
*X
  E10
  DUMMY2
  LOG1A
  SCHTOP
```

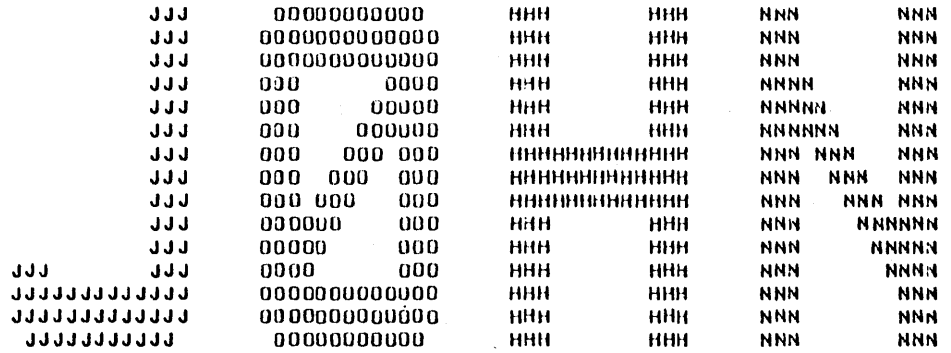
ENTRY POINT TABLE -

TSLSIZ	31CC	P1829	073F	MIB	3EE3	FNR	3B0D
TOD	3E0B	MAKQ	3BB2	LOCF	3F8D	BSMD	4F51
XMALC	2A90	ADEV	3BE0	THDS	7FFF	REQXT	34DB
R9MUX1	0000	R9MUX2	0002	R9MUX3	0003	R9MUX4	0004
R9MUX5	0007	R9MUX6	0001	PBAT00	083C	PBAT01	08BA
READRC	47FC	COMM18	7FFF	JBCNFG	3FA8	RELBYQ	55AF
MRECAD	4972	DATFMT	0000	VOLBLK	0255	DGNTAB	048C
MICSUB	0022	NSTACK	0005	TSTLOC	0A29	TMRTHD	3DDF
VERIFY	00A8	DISMNT	00BD	SECTOR	7F80	TSQPRI	0003
STDINP	000A	UNPTIM	3F7D	N14	0000	QMINTX	7FFF
UNPSRT	6800	TSUSER	0AF4	FUNSHR	4B41	TSLUNT	3125
L1829	02CC	L1860	0489	I18331	4DD8	P83310	0580
FILE4	3F8C	CPREL	33DF	RCTV	34C7	MINTO	1679
FMPA01	09CC	FMPA02	09D7	FMPA03	09EC	BRKPT	0070
MONTO	1676	FMPA06	0A20	FMPA07	0A28	FMPA08	0A28
FMPA09	0A2D	ASAV	385C	FMPL01	5F5A	K65COR	7FFF
FMPL03	6220	FMPL04	61F4	FMPL05	65B4	FMPL06	5FF3
FMPL07	5DDB	FMPL08	5D94	FMPL09	612D	DUMMY1	00CB
DUMMY2	00D2	UNPEND	A7FF	JPSTV4	0046	RELS1A	3F79
WTREAD	30A2	RMUCLO	0007	SLICER	2B7A	TSLMSB	31CA
TSCNAC	2642	TMRLVL	0004	ERRMSG	3F74	SCHTOP	34E5
REAREQ	279E	TSCHAN	3163	STRSEC	0ADA	BSYEFS	4D06
STRLEN	FB3D	TSATCH	28F7	PGLUNT	3122	SYSLVL	3236
MMAMAX	0A16	CLRTCU	519C	CNTHAR	1683	CYLTRK	51E9
GTNXUM	2A17	S1827	0324	IUP	3F80	S18277	032D
LOG1A	0438	VOLA	38C9	CONCU	512D	NABS	37B1
CPFET	3476	MIINP	04EA	SWTCH	3F8F	NMONI	3499
CPMOD	33F8	XMAT	0A01	JBFLV4	0000	DOUT	7FFF
PARAME	3771	EFDATA	4D09	FBASV4	7FFF	MTBFE0	09CB
ECCALG	52C6	RDPTV4	7FFF	FMSAVA	0ABA	JPFLV4	0054
MASCON	5652	COMPV4	3FB6	INPTV4	3F81	MASEXT	55B0
TSMEND	0AC4	JPCHGE	002A	FINDRQ	3B0D	ALCLGH	19F7
SCHSTK	03D8	LSTLOC	0972	MPFLAG	0170	DATSEP	002F
TSIPRC	2BDF	ADNSHP	2A05	TSLOFF	0007	STRBAS	F000
DUMALT	0002	RCOVER	0069	UPLOHM	3827	TSFMCP	333C
N15	0000	TSMMER	3360	CONSLU	0004	CONTCU	4E3D
LRGUSR	0A31	HA	38AB	REL	3AC0	IP1	3847
MAS300	55B0	IDLE	10D7	MASKT	0145	LIBET	3F77
PGMIN	30EA	MMREL	2B55	SPACE	176E	SABS	3789
ECORE	3AC5	RDISP	34E5	NUMCP	0006	TK7RH	7FFF
FMPA10	0A39	FMPA11	0A51	FMPA12	0A5D	FMPA13	0A6B
FMPA14	0A7B	FMPA15	0A92	FMPA16	0A99	FMPA17	0AA3
FMPA18	0AA6	FMPA19	0AAC	FMPL10	65B1	MTBFS0	0359
FMPL12	623A	FMPL13	62A2	FMPL14	657D	BUSY	0973
FMPL16	6092	UCTABL	0CE2	FMPL18	5F2D	FMPL19	6227
MAXCOP	001E	LOBDTB	01A5	UPBDTB	0194	PKEYV4	7FFF
TSPAGE	097C	FMPFLG	7FFF	SKALND	5023	JBCNCL	3F92
UNPIOF	3F7E	LRTABL	0D96	LSIZV4	04B8	PGMINT	310F
TSULBF	272A	SYFAIL	0187	OFNSHP	2A55	IDUMMY	4DB4
RESTOR	0062	COMPRQ	3B82	N11	0000	SYSSEG	0085
CNDRIV	5158	CUCNST	5122	TSMUSR	0A44	NTSUSR	0003
TSEXIT	316F	SYSYER	3738	MI	3E54	MOV	3FC5
USE	7FFF	IDLER	10DC	FILE1	3F89	JPRET	1688
MP1234	3813	NDISP	7FFF	K65T10	7FFF	NXUC	0A1D
K65T12	7FFF	MIBUF	3F75	IPROC	3837	NUMLU	0014
NXUM	0A21	JPRET1	3FA9	CARDRD	375F	JLGDV4	003F
SMDCPA	54CA	U2INIT	7FFF	FMUFCB	4698	RELFIL	7FFF
VOLEND	03D8	OUTPV4	7FFF	LUNAME	0494	ONTIME	0A3B

DENXUC	2A6A	FSTIME	10F8	ABSPAR	44DE	EXPIRE	2643
USKNPG	5200	TSWSEC	2B0B	RQLLIM	0A32	TSCKPM	2BAB
EFSTOR	009A	SMTSIM	031B	JOBPRO	0015	COBOPS	7C00
COMREQ	579A	AMINTX	3F13	EMPSRT	10F5	SYSDAY	3131
SWAPAR	0AF9	PRO	383B	ASC	34B4	RW	36A7
JPT13	0031	SAVLU	3689	SEEK9	5033	CPTBL	098F
CPADD	3447	DAYTO	1677	M1811T	5AE0	ISMD	4DD8
CPCHK	33C6	MMALC	2B10	MIPRO	008C	FLIST	1681
PORTS	07FA	PWFOV	38A8	SPCEV4	7FFF	SETBND	3383
JLLUV4	1684	STLPV4	3A2C	LIKDUM	5210	DEVERR	33E0
SETFNS	5019	FMEOFC	5F82	CPTBLN	0013	MSIZV4	FFFE
TSAREA	096F	TSVFTN	2B8C	JOBENT	000E	CLRINT	4F29
HORMIN	167C	SBATOU	034B	SYSCOP	007E	FMBRRN	4707
N12	0000	PTNREL	7FFF	SYSMON	3037	N5	0000
LSTOUT	0009	N7	0000	N8	0000	N9	0000
AYERTO	1672	LVLSTR	095D	L1827	0308	S1860	030E
L18277	0275	ECXIT	5391	FILE2	3F8A	ENDOV4	263F
MMEF	4D22	DONE	4E26	CPDEF	3395	I1811T	5B1E
ESMD	4EBF	MONI	3499	YERTO	1675	NAMEV4	004D
BUFFE	1670	MINT	3E54	LOOP	3A64	JPSWT	3F88
QSAV	38A6	VINPV4	3F86	CDRDSM	7FFF	XMTEND	0A15
ALTDAI	54ED	TDFUNC	0093	FMSCOM	4428	TIMCPS	003C
ALCORE	38F3	LOADIN	3F90	BGNMON	2640	CLRDAS	5196
LBATIN	0543	RLSECU	5013	PRTCDR	0A2B	TSBGIN	2B7B
NRERLE	0DC3	TOTMIN	167D	INTSTK	01B5	SWPSEC	0A2D
FSLIMT	0FEC	TSINTR	3112	EPORTS	083B	INSTLU	0006
SYUTIL	00AF	LOOO	0120	C18331	4E2A	TIMEC	0005
CHAIN	3160	XMREL	2AE2	AREAC	178B	SECON	167A
PCORE	048A	TRVEC	3F72	DWADD	48F5	UPTOD	3E05
MMAT	0A17	CONTA	167B	SMDGD1	54EF	MSAV	38A7
DATFLG	4E49	TBLADR	10F4	SEKDON	5093	BATINP	000D
STMSV4	1A08	POLCHK	518D	NIPROC	3837	KIBSEC	0003
TSNABL	2640	PARTBL	096E	LIBEDT	0000	EFLIST	00A1
RMUCFT	0006	SMDGDR	54EF	UNLOCK	4B2E	PROTEC	001C
AMONTO	1673	INVINT	04B3	CLFRID	467F	TSPMIN	30ED
HKSPLU	0A30	N13	0000	INIPRT	5977	TSCKMU	23CF
EXTSTK	0205	OVFVOL	38E6	CCP	0974	LOG	4C47
LOG1	044D	LOG2	0462	MAKEQ	3BB2	BEGIN	6800
S1811T	5AAB	LUABS	3771	NFNR	3B0D	JKIN	3F91
JCRDV4	0038	XMMOD	2AEA	NSWP	0A25	AFILV4	005B
ODEBUG	0077	TSIOC1	3262	TSIOC2	3300	SCHLNG	0060
LOADSD	0007	MASERR	5684	CALTHD	095B	MASDRV	54FD
MNTCHK	00B6	TSTASK	2648	BATLST	000E	PLENTH	000D
JPLOAD	0023	XMBLOK	2646	TSOFFM	0A34	TERMLU	0005
DENSWP	2A5C	ONNXUC	29DB	TSATTC	313C	PTYERR	3848
FSCTNE	001E	LMTSIM	033C	EMPSTP	10F6	T1	36A7
T2	36A7	ONNXUM	29E2	T4	36A7	T6	36A7
LSTPRT	0003	AUTOBT	00C4	T9	3555	LTSUSR	0030
MAXSEC	016B	SYSID	0174	UBPROT	0191	LBPROT	0192
ALTERR	0477	TK7WEF	7FFF	TK7DAT	7FFF	NBRLIN	0004
AUTON	083B	N4	0001	N6	0000	N10	0000
PTNALC	7FFF	ACPTBE	0975	CPTBLZ	0072	PROMTR	3E08
BAITOS	0001	RQULIM	0A33	RETIME	0A39	FILOAD	0A3E
QPASWD	0A3F	HKSTAT	0A43	RMUOPN	0000	RMUCSH	0001
RMUCFO	0005	TSPORT	0AC4	TSPEND	0AF4	TSUEND	0B24
HWINIT	7FFF	SIMRSV	7FFF	FMRDEL	5E81	FMMOSU	000A
FMMOIU	000A	FMNRCD	0010	UCTLEN	00B4	MAXLOC	0009
FMMOSF	0006	FMFCBS	0DC4	FMMOIF	0006	FMFCBI	0E1E
FCBSCT	0EC0	FSLEND	1063	MMLUTB	1064	PCTABL	107D
IDLCTR	10E1	TMRTYP	10F1	TMCODE	0008	DMICOD	10F3

BUFF	110C	CGHOST	1670	ADAYTO	1674	HORTO	1678
TODLVL	5016	NSCHED	167E	PSIZV4	04B8	ODBSIZ	0369
CHRSFG	167F	FMASK	1680	Q8STP	1682	T10	176E
T17	176E	N1	0362	N2	0246	S1829	0334
SBATIN	033C	LBATOU	050D	FMPL02	64A2	FMPA04	09FA
FMPA05	0A08	FMPL11	6166	FMPL15	5F4A	FMPL17	5DC8
DATBAS	0000	TSURTN	26E0	TIMSLC	28C3	TSACTV	2641
TSMFLG	2645	ONNSHP	29E9	OFNXUC	2A4E	DENXUM	2A63
XMRSV	2A94	XMRTN	2AE6	TSCLOK	2B73	TSRFTN	289B
TSCLL8	2BA3	TSPMCK	28C5	UNPTBL	2C53	SLICUP	3130
TSLCUP	3133	ATTACH	3139	TSLLSB	31CB	TSXERR	315B
PGMOUT	316C	TSLICE	320D	TSUSCP	3253	EXTREG	336F
CPSET	341C	DISPXX	34E6	T18	3555	T19	35E8
TO	36A7	CKTHRD	3735	RPMASK	372F	T14	3755
T16	3766	CABS	37C9	ALLIN	37E2	PHFAIL	3887
ALVOL	38C9	VOLR	38DE	OFVOL	38E6	REQALC	38F3
RTNCOR	396F	ICORE	39A3	T12	3A8D	SWAPCK	3ACE
LEND	3A5E	SWAPON	3A5D	OUTPUT	3A22	SPACE4	3A56
NOG30A	3A70	SCH	3AE5	NCMPRQ	3B62	ALTDEV	3BE0
CONVER	3CC9	ALTSUB	3D3C	TMINT	3D51	TIMEUP	3D96
T8	3D51	T15	3D51	DTMER	3DE3	DTIMER	3DE3
MIBX	3EE4	RELFLE	3F34	JOBSTR	3EB7	PARBV4	3F7A
SCHERR	3FB1	JOBIND	3F7B	TRANV	3F72	UNPIO	3F7C
SPASH	3F85	FILE3	3F8B	RECOV	3F78	LPTRS	3F8E
BATCLU	3F7F	PRORET	3F87	JBPROE	3F73	TRNVEC	3F76
AUTF9	3F82	AUTFA	3F83	AUTFB	3F84	JBCFGZ	3F98
COUTV4	3FDC	OLDUMP	3FFE	COBOP	4183	FMSHAP	43F5
FMCOMP	443A	FMCOME	442C	FMPPRO	43E9	CKUADR	44ED
MMREAD	4518	MMWRIT	451C	LOKCHK	45DC	REMLOK	464F
INSLOK	465B	FMCHKO	46D3	FMCOVL	46E1	PUTREC	4711
GETNXT	48D6	WRTBAK	4904	COMSEQ	49BE	LOCKFL	4B13
FSHARE	4B3B	DWSUB	4C1B	DWMUL	4C26	CEFDTA	4D07
EFLOCK	4CAB	EFCOVL	4D08	CDUMMY	4DB5	EDUMMY	4DB6
GETLOS	4E28	CSMD	4E2A	E18331	4E8F	EGHOST	4F36
WANTDR	4F50	BSMD4	4F70	ALMERR	4F80	XSMD	4FCE
CLRSKN	5048	SEKCHK	5028	SEKCOM	5092	SEKINP	5091
CKOVRL	51CF	CLRSTS	5190	CLRTDA	518A	DASTAT	51A8
DRICHK	51C8	MSBLSB	51F7	CLRACR	51A2	SMDSTS	51AE
SELFLG	511F	FFILBF	54C5	FILSMD	54C7	SMDCPG	54DE
SMDCPS	54D0	SMDACP	54E9	SMDGCU	54ED	SMDRDR	54F1
LTOFDR	54F6	MPDVCK	54F9	MPDRIV	54FB	ITRMNL	568E
ETRMNL	5808	CTRMNL	5862	ICONSL	5B24		

MASS MEMORY RESIDENT EXAMPLE



*K,P8
*ASSEM

```
0001                        NAM   CONTIN   DECK ID CAL INTERACTIVE PAUSE
0002                        EXT   DUMMY2

0004              **        EXAMPLE OF A MASS MEMORY RESIDENT
0005              *         SYSTEM LIBRARY PROGRAM (*M) USING
0006              *         SYSTEM REQUESTS

0008              *         THE PURPOSE OF THIS PROGRAM IS
0009              *         TO READ CHARACTERS FROM THE STANDARD
0010              *         INPUT UNIT AND RESPOND TO DIFFERENT
0011              *         INPUT CHARACTER PATTERNS.

0013              *         IF THE INPUT CHARACTERS HERE
0014              *         'YES' OR 'NO' THE PROGRAM 'PNOYES'
0015              *         IS SCHEDULED
0016              *         IF 'END' HAS INPUT, THE PROGRAM TERMINATES.
0017              *         IF ANYTHING ELSE IS TYPED IN,
0018              *         IT IS CONSIDERED GARBAGE

0020 P0000 C8FE           NUM   $C8FE        FIRST WORD ADDRESS
0021 P0001 6800           STA   REL+2        CORE RELEASE ADDRESS
     P0002 004C
0022              CONTIN  FWRITE 4,REPLY-*+1,CONMSG-*+5,CONSIZ,,6,6,,I
0022 P0003 54F4
0022 P0004 0D66
0022 P0005 0007
     P0006 0000
0022 P0007 1004
0022 P0008 000D
     P0009 005D
0023 P000A 14EA           JMP-  ($EA)        JUMP TO DISPATCHER
0024              REPLY   FREAD $F9,CHECK-*+1,IBUF-*+5,BUFSIZ,,6,6,I,I
0024 P000B 54F4
0024 P000C 0966
0024 P000D 0007
     P000E 0000
0024 P000F 18F9
0024 P0010 0004
     P0011 004B
0025 P0012 14EA           JMP-  ($EA)        JUMP TO DISPATCHER
0026 P0013 C800   CHECK   LDA   IBUF         FIRST 2 CHARS OF SCREEN ENTRY
     P0014 0043
0027                      CAE   NO           HAS NO KEYED
0027 P0015 0486
0027 P0016 E000
0027 P0017 0044
0028 P0018 1810           JMP*  TRYES        IF NOT TRY YES
0029 P0019 0C00           ENQ   0            0 MEANS NO
0030                      SCHDLE (DUMMY2),6  IF SO SCHEDULE NO AND EXIT
```

```
0030 P001A 54F4
0030 P001B 1206
0030 P001C FFFF X
0031 P001D 0152              SQN     NOTSHD          IF NOT SCHEDULED
0032 P001E 1800              JMP     REL             JUMP TO RELEASE CORE
     P001F 002D
0033                 NOTSHD  FWRITE  4,CONTIN-*+1,NOTMSG-*+5,NOTSIZ,,6,6,,I
0033 P0020 54F4
0033 P0021 0D66
0033 P0022 7FE1
     P0023 0000
0033 P0024 1004
0033 P0025 0007
     P0026 0067
0034 P0027 14EA              JMP-    ($EA)
0035                 TRYES   CAE     YES             HAS YE KEYED
0035 P0028 0486
0035 P0029 E000
0035 P002A 0033
0036 P002B 180D              JMP*    TRYEND          IF NOY TRY END
0037 P002C C800              LDA     IBUF+1          GET NEXT 2 CHARS OF SCREEN ENTRY
     P002D 002B
0038                         CAE     YES+1           HAS S  KEYED
0038 P002E 0486
0038 P002F E000
0038 P0030 002E
0039 P0031 1807              JMP*    TRYEND          IF NOT TRY END
0040 P0032 0C01              ENQ     1               1 MEANS YES
0041                         SCHDLE  (DUMMY2),6      IF SO SCHEDULE YES AND EXIT
0041 P0033 54F4
0041 P0034 1206
0041 P0035 801C X
0042 P0036 1800              JMP     REL             JUMP TO RELEASE CORE
     P0037 0015
0043 P0038 C800     TRYEND   LDA     IBUF            FIRST 2 CHARS OF SCREEN ENTRY
     P0039 001E
0044                         CAE     END             HAS EN KEYED
0044 P003A 0486
0044 P003B E000
0044 P003C 0023
0045 P003D 1812              JMP*    ERROR           IF NOT OUTPUT ERROR AND RETRY
0046 P003E C800              LDA     IBUF+1          GET NEXT 2 CHARS
     P003F 0019
0047                         CAE     END+1           HAS D  KEYED
0047 P0040 0486
0047 P0041 E000
0047 P0042 001E
0048 P0043 180C              JMP*    ERROR           IF NOT OUTPUT ERROR AND RETRY
0049                 PEND    FWRITE  4,REL-*+1,ENDMSG-*+5,ENDSIZ,,6,6,,I
0049 P0044 54F4
0049 P0045 0D66
0049 P0046 0007
     P0047 0000
```

```
0049 P0048 1004
0049 P0049 0008
     P004A 0038
0050 P0048 14EA   POUT     JMP-   (SEA)          JUMP TO DISPATCHER
0051               REL     RELEAS 0,T,0          RELEASE CORE AND JUMP TO DISPATCHER
0051 P004C 54F4
0051 P004D 1801
0051 P004E 0000
0052               ERROR   FWRITE 4,CONTIN-*+1,ERRMSG-*+5,ERRSIZ,,6,6,,I
0052 P004F 54F4
0052 P0050 0D66
0052 P0051 7FB2
     P0052 0000
0052 P0053 1004
0052 P0054 000F
     P0055 001E
0053 P0056 14EA               JMP-   (SEA)          JUMP TO DISPATCHER
0054 P0057 2020   IBUF       ALF    *,          *
     P0058 2020
     P0059 2020
     P005A 2020
0055       0004               EQU    BUFSIZ(*-IBUF)
0056 P0058 4E4F   NO         ALF    *,NO  *
     P005C 2020
0057 P005D 5945   YES        ALF    *,YES *
     P005E 5320
0058 P005F 454E   END        ALF    *,END *
     P0060 4420
0059 P0061 4152   CONMSG     ALF    *,ARE YOU READY TO CONTINUE?*
     P0062 4520
     P0063 594F
     P0064 5520
     P0065 5245
     P0066 4144
     P0067 5920
     P0068 544F
     P0069 2043
     P006A 4F4E
     P006B 5449
     P006C 4E55
     P006D 4540
0060       000D               EQU    CONSIZ(*-CONMSG)
0061 P006E 5245   ERRMSG     ALF    *,REPLY HAS NOT :NO:,:YES:,:END:*
     P006F 504C
     P0070 5920
     P0071 5741
     P0072 5320
     P0073 4E4F
     P0074 5420
     P0075 3A4E
     P0076 4F3A
     P0077 2C3A
     P0078 5945
     P0079 533A
```

```
      P007A 2C3A
      P007B 454E
      P007C 443A
0062        000F              EQU     ERRSIZ(*-ERRMSG)
0063  P007D 594F    ENDMSG    ALF     *,YOU HAVE NOW FINISHED*
      P007E 5520
      P007F 4841
      P0080 5645
      P0081 204E
      P0082 4F57
      P0083 2046
      P0084 494E
      P0085 4953
      P0086 4845
      P0087 4420
0064        0008              EQU     ENDSIZ(*-ENDMSG)
0065  P0088 4E4F    NOTMSG    ALF     *,NOT SCHEDULED*
      P0089 5420
      P008A 5343
      P008B 4845
      P008C 4455
      P008D 4C45
      P008E 4420
0066        0007              EQU     NOTSIZ(*-NOTMSG)
0067                          END

PGM=  008F (   143)   COM = 0000 (     0)   DAT = 0000 (     0)
```

*** S Y M B O L T A B L E ***

BUFSIZ	0055	CHECK	0026	CONMSG	0059	CONSIZ	0060	CONTIN	0022	DUMMY2	0002	END	0058	ENDMSG	0063	ENDSIZ	0064
ERRMSG	0061	ERROR	0052	ERRSIZ	0062	I	0000	IBUF	0054	NO	0056	NOTMSG	0065	NOTSHO	0033	NOTSIZ	0066
PEND	0049	POUT	0050	REL	0051	REPLY	0024	TRYEND	0043	TRYES	0035	YES	0057				

3-26-12

```
*LIBEDT
  LIB

  IN

  *K,IB
  IN

  *M,30,,,M
    CONTIN  0000    DECK ID CAL INTERACTIVE PAUSE
  IN

  *Z
*K,PB
*ASSEM
```

```
0001                            NAM     PNOYES   DECK ID PROCESS NO  AND YES
0002                            EXT     DUMMY1
0003 P0000 C8FE                 NUM     SC8FE        FIRST WORD ADDRESS
0004 P0001 6800                 STA     REL+2        CORE RELEASE ADDRESS
     P0002 001C
0005 P0003 0150                 SGN     PYES         0> NON 0 THEN PROCESS YES
0006                            FWRITE  4,CONSCD-*+1,NOMSG-*+5,NOSIZ,,6,6,,I
0006 P0004 54F4
0006 P0005 0D66
0006 P0006 0007
     P0007 0000
0006 P0008 1004
0006 P0009 000C
     P000A 001A
0007 P0008 14EA                 JMP-    (SEA)        JUMP TO DISPATCHER
0008                    CONSCD  SCHDLE  (DUMMY1),6   RE-SCHEDULE MAIN PROGRAM
0008 P000C 54F4
0008 P000D 1206
0008 P000E FFFF X
0009 P000F 1800                 JMP     REL          JUMP TO RELEASE CORE
     P0010 000C
0010                    PYES    FWRITE  4,OUT-*+1,YESMSG-*+5,YESIZ,,6,6,,I
0010 P0011 54F4
0010 P0012 0D66
0010 P0013 0007
     P0014 0000
0010 P0015 1004
0010 P0016 000F
     P0017 0019
0011 P0018 14EA                 JMP-    (SEA)        JUMP TO DISPATCHER
0012                    OUT     SCHDLE  (DUMMY1),6   RE-SHEDULE MAIN PROGRAM
0012 P0019 54F4
0012 P001A 1206
0012 P001B 800E X
0013                    REL     RELEAS  0,T,0        RELEASE CORE AND JUMP TO DISPATCHER
0013 P001C 54F4
0013 P001D 1801
0013 P001E 0000
0014 P001F 414E NOMSG   ALF     *,ANSWER AGAIN WHEN READY*
     P0020 5357
     P0021 4552
     P0022 2041
     P0023 4741
     P0024 494E
     P0025 2057
     P0026 4845
     P0027 4E20
     P0028 5245
     P0029 4144
     P002A 5920
0015        000C                EQU     NOSIZ(*-NOMSG)
0016 P002B 594F YESMSG  ALF     *,YOU HAVE CONTINUED , WELL DONE*
     P002C 5520
     P002D 4841
```

```
     P002E 5645
     P002F 2043
     P0030 4F4E
     P0031 5449
     P0032 4E55
     P0033 4544
     P0034 202C
     P0035 2057
     P0036 454C
     P0037 4C20
     P0038 444F
     P0039 4E45
0017       000F              EQU     YESIZ(*-YESMSG)
0018                         END

PGM=  003A (    58)   COM = 0000 (     0)   DAT = 0000 (     0)
```

*** S Y M B O L T A B L E ***

CONSCD 0008 DUMMY1 0002 I 0000 NOMSG 0014 NOSIZ 0015 OUT 0012 PYES 0010 REL 0013 YESIZ 0017
YESMSG 0016

```
*LIBEDT
  LIB

  IN

  *K,I8
  IN

  *M,31,,,M
    PNOYES  0000    DECK ID PROCESS NO  AND YES
  IN

  *Z
*CTO,AFTER THE PAUS CONTROL STATEMENT, PLEASE CHANGE
*CTO,STANDARD INPUT UNIT TO 4 (*K,I4)
*CTO,DO A MANUAL INTERRUPT *Z
*CTO,AND SCHEDULE 'CONTIN' BY DOING
*CTO,A MI = S,30,6
*CTO,THANK YOU
*PAUS
```

SYSTEM INITIALIZATION

CONTROL STATEMENTS

BUILD SYSTEM LIBRARY DIRECTORY

*Y	CORE RESIDENT
*YM	MASS MEMORY RESIDENT

PROGRAM LOADING

*L	PART 0 CORE RESIDENT
*LP	PART 1 CORE RESIDENT
*M	PART 0 MASS MEMORY RESIDENT (TO RUN IN ALLOCATABLE CORE)
*MP	PART 1 MASS MEMORY RESIDENT (TO RUN IN PARTITIONED CORE)
*S	PATCH EXTERNALS
*D	DEFINE LABELED COMMON BASE

STANDARD UNIT MANIPULATION

*C	MEMORY MAP LIST UNIT
*I	INPUT UNIT
*0	MASS MEMORY UNIT
*U	READ CONTROL STATEMENTS FROM COMMENT UNIT
*V	READ CONTROL STATEMENTS FROM INPUT UNIT

DISK UTILITY

*G	WRITE ADDRESS TAGS
*H	RUN SURFACE TEST

MISCELLANEOUS

*	COMMENT CARD
* T	TERMINATE

STUDENT PROJECT - 3

Using the core dump answer the following questions:

1. What was the priority of the system when the dump was taken?

2. Is there anything on the Interrupt Stack? If yes, how many programs and what are their priorities? [stack start] if top = start then nothing on stack

3. Are there any programs waiting to go into execution for the first time? 4
SCHEDULE STACK TOP = $303 = start of free space location $84 now has 318 [location] of UDISP has top of schedule stk.

4. What was the contents of the M-Register?
Presume M corresp to PILLVL if not syfal ✓ 489B

5. Have any programs in the System Library been scheduled since Auto Load time? IS completion address filled in yes then ran

6. Are there any dummy entries in the System Library?
Dummy entry has sector address but no length. Usually at end. see 108E-7

7. What were the standard Logical Units at the time of the dump? F9-FD ? std log. units

8. From the INSTALL listing draw the core layout for that system.

How big
what is lastword utimes
Monitor start
allocatable
Sys library

9. Was background in Part 0 or Part 1?

STUDY QUESTIONS - 3

TRUE or FALSE

1. Run-where-loaded programs can run in PART 0. (T) F
2. Run-anywhere programs can run in PART 1. (T) F
3. One can not use two word relative instructions in PART 1. T (F)
4. Partitioned core is reserved for background. T (F)
5. Background can be in either allocatable or partitioned core. (T) F
6. Programs in the System Library may be stored in Relocatable or Absolutized form. T (F)
7. We can compile FORTRAN programs in the foreground. T (F)

MULTIPLE CHOICE

1. In the foreground we can do the following (choose more than one)

   a) COMPILE
   (b) Load
   (c) Debug
   (d) Use the INP/OUT instructions
   e) Add or delete programs from the System Library
   (f) Execute
   (g) Request space in allocatable core

2. Which of the following can be in partitioned core?

   (a) System Library Programs (*L)
   (b) System Library Programs (*LP)
   (c) Data Buffers (PTNCOR)
   d) SYSDAT
   e) SPACE
   (f) Background

3. Auto Load does some of the following, which ones?

   (a) Sets the protect bits for the foreground
   (b) Reads in the Core Resident Programs
   c) Sets up the various areas on the disk
   d) Builds the core image area on disk

Match the following terms with the characteristics that best suit it from the column on the right (more than one may apply to more than one term).

1. Process Program D, B, C E
2. Job A C, D
3. System Library B, D, E, C
4. User A, B, C, D, E
5. Run-anywhere Programs B, ALL
6. ? The D-bit SET A, B, C, E
7. System Program B, C, O, e
8. Program Library A, B
9. Loader A
10. Compiler B A
11. Driver B, C, D, E

a. background
b. foreground
c. Part 1
d. Part 0
e. Priority greater than 2

15
↓
2

6 D BIT = part 0 / part 1 request
ab-add set 0
must set
if not t
set 0

LESSON GUIDE 4

SYSTEM FLOW

LESSON PREVIEW:

This lesson will discuss in detail the common interrupt handler, the dispatcher, and the request entry/exit.

REFERENCES:

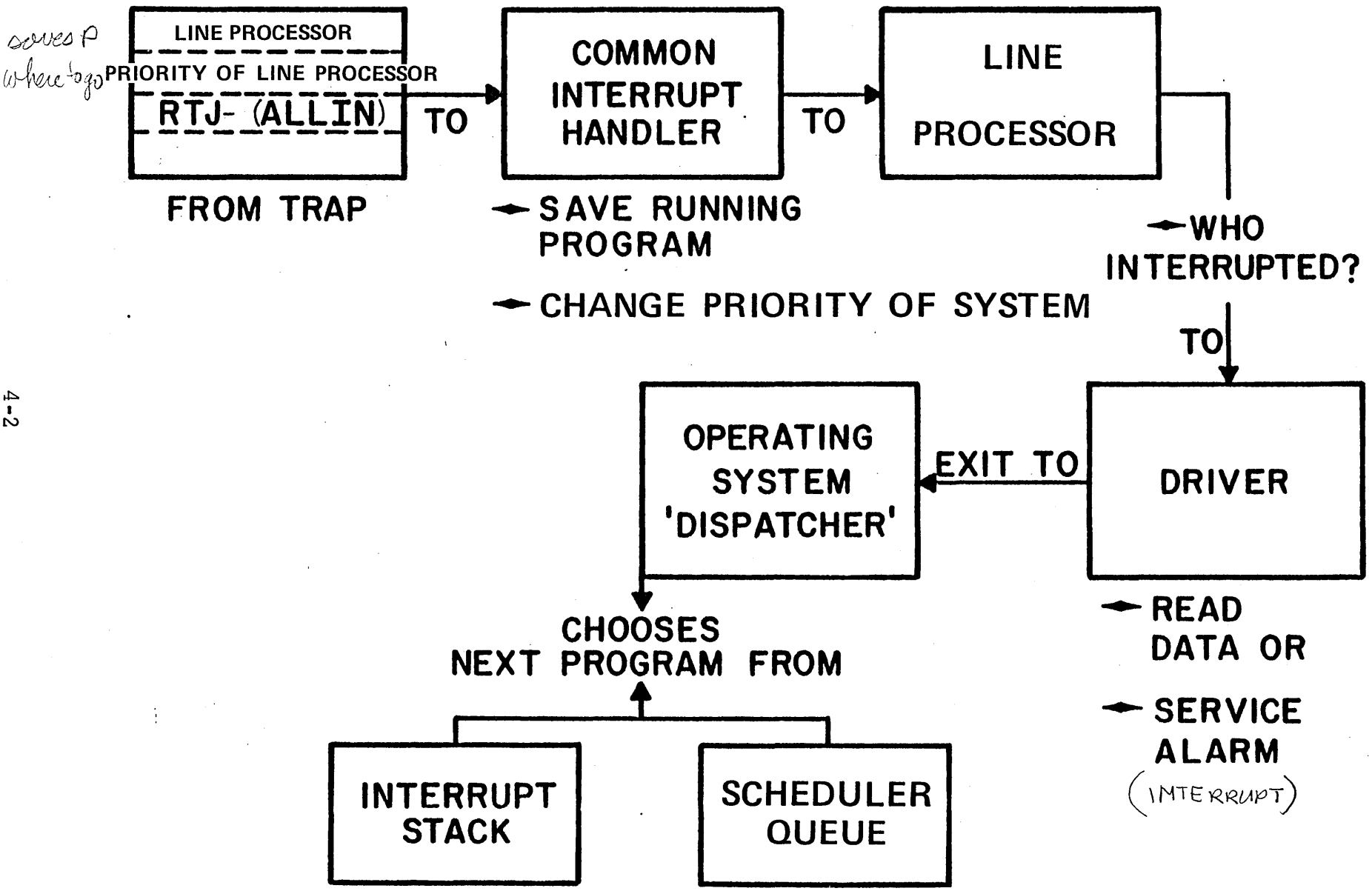
Listings of SYSDAT, NDISP, COMMON, and NMONI.

OBJECTIVES:

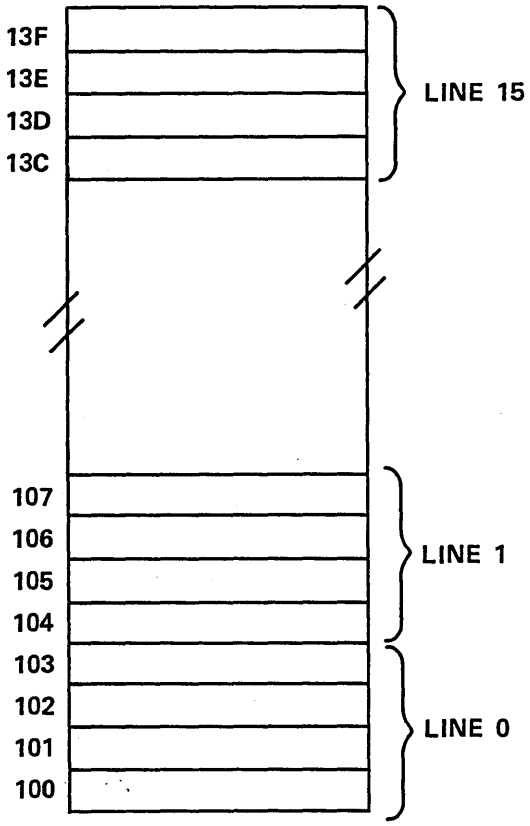
At the completion of this lesson, the student will be able to:

1. Discuss the function and significance to the system of the common interrupt handler, the dispatcher, and MONI.
2. Read system listings.

INTERRUPT FLOW



INTERRUPT TRAP AREA maintained by common interrupt handler.



INTERRUPTS MAY BE NESTED 16 DEEP

FOUR CORE LOCATIONS RESERVED FOR EACH INTERRUPT LINE:

- WORD 4 – ADDRESS OF INTERRUPT PROCESSOR] parameters
- WORD 3 – PRIORITY LEVEL FOR LINE
- WORD 2 – RTJ TO INTERRUPT HANDLER
- WORD 1 – P

WHEN INT occurs:

HARDWARE:

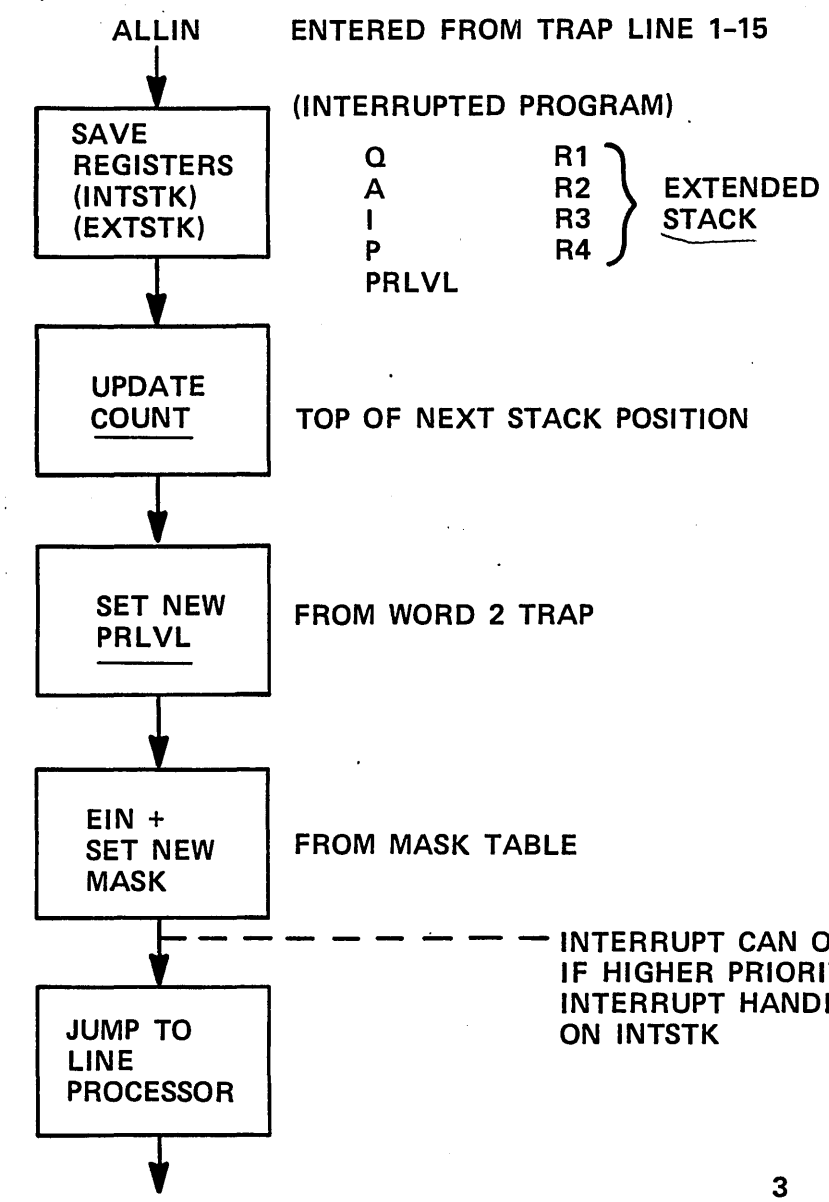
- DISABLES INTERRUPTS
- STORES P OF INTERRUPTED PROGRAM IN WORD 1
- TRANSFERS CONTROL TO WORD 2

INT. HANDLER

SOFTWARE:

- WORD 2 CONTAINS RTJ TO COMMON INTERRUPT HANDLER FOR THE INTERRUPTED line
- INTERRUPT HANDLER SAVES REGISTERS OF INTERRUPTED PROGRAM OVERFLOW AND SETS NEW MASK FROM PRIORITY LEVEL IN WORD 3, ENABLES INTERRUPTS, AND TRANSFERS CONTROL TO INTERRUPT PROCESSOR FOR THAT LINE (FROM ADDRESS IN WORD 4)
- INTERRUPT PROCESSOR OR DRIVER MUST EXIT THROUGH DISPATCH TO RESTORE INTERRUPTED PROGRAM

COMMON INTERRUPT HANDLER



TRAP

3	LINE PROCESSOR*
2	PRIORITY*
1	RTJ - ($FE)
0	P

COMMON INTERRUPT HANDLER

PROGRAM FUNCTION

The functions of the interrupt Handler are to save the machine register by placing them in the interrupt stack, set the mask for the new priority level, enable the interrupts, and transfer control to the primary processor for the interrupt line.

ENTRY INTERFACES

Calling Sequence: The program is called from the interrupt trap locations 100_{16} through $13F_{16}$. Four words are used per line.

OP	ADDRESS	
NUM	0	P-register saved in word one of the interrupt trap location by hardware.
RTJ-	($FE)	Give control to Common Interrupt Handler.
NUM	"level"	Priority level associated with this interrupt. A number between 0 and 15. The larger number corresponds to the higher level.
ADC	"line processor"	Line processor routine to service the interrupt line.

Entry Conditions: Interrupts are inhibited, and the P is saved in the interrupt trap location for this line. This is normally done by hardware but a user may simulate these conditions and generate a psuedo-interrupt. The routine is given control by the return jump following the interrupt trap location.

EXIT INTERFACES

Exit Conditions: The interrupt handler will exit to the line processor with the following conditions.

a. The priority level will be set to the level associated with this interrupt.

b. The mask register, M, will be set to the mask for this priority level.

c. Interrupts will be enabled.

d. The I-register will contain the location associated with the interrupt line; i.e., for interrupt line, L, I will contain 100_{16} + 4*L.

DESCRIPTION

The interrupt handler saves the register A, Q, and I, the priority level of the interrupted program, and the P-register, by placing them in the interrupt stack. The interrupt stack is a push-down, pop-up stack with five words allocated to each entry. A maximum of 16 entries is possible. The registers are saved in the following format.

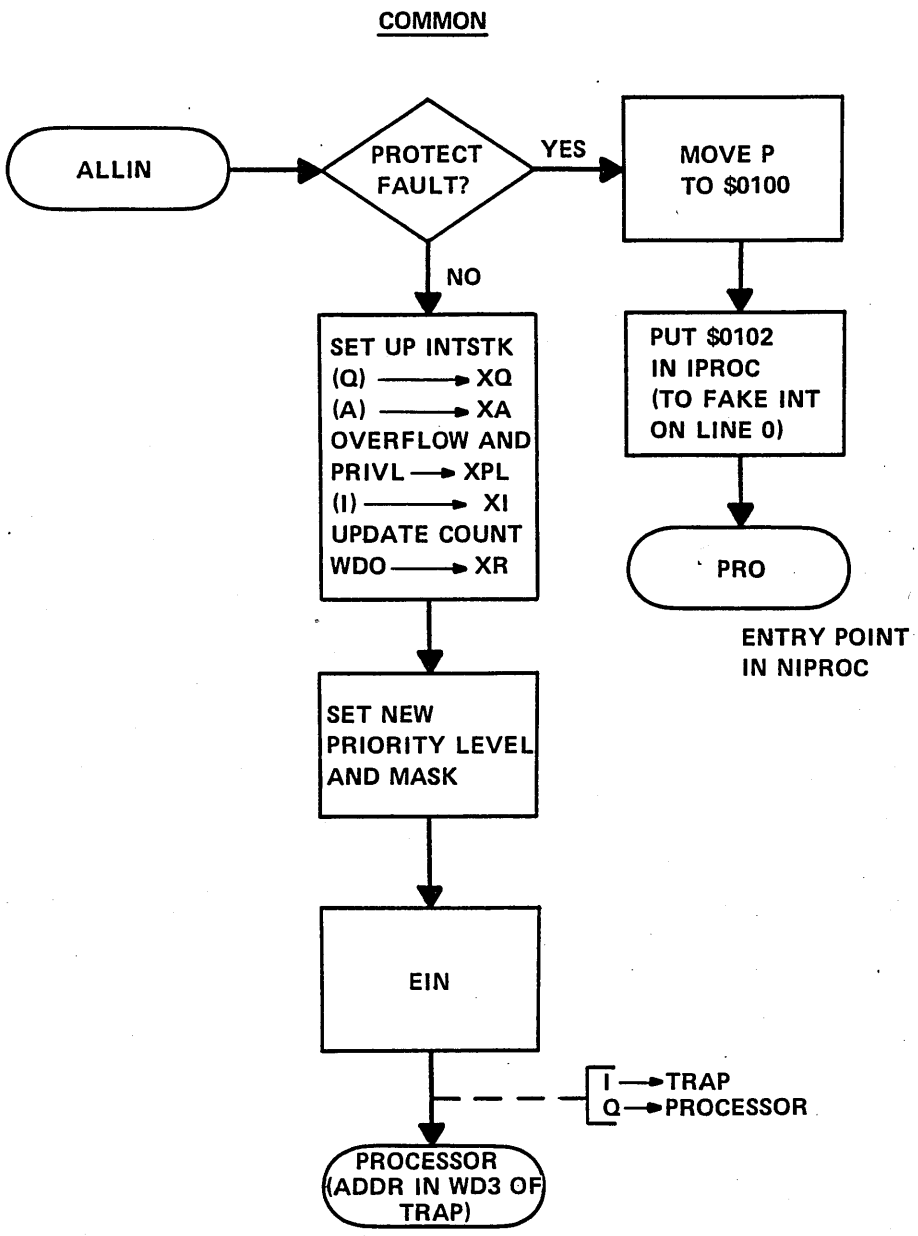
0	Q - saved	Q register saved
1	A - saved	A register saved
2	I - saved	I register saved
3	P - saved	Overflow
4	Priority level	Priority level before the interrupt

After saving Q in the interrupt stack, the address of the current entry to the interrupt stack is held in ! while A, I, and the priority level are saved in the stack. For CYBER 18 systems, registers R1, R2, R3, R4 are saved and the setting of the overflow flip-flop is tested and saved with the priority level. The interrupt stack base counter, COUNT, is incremented by five to point to the next entry in the stack. The return address is retrieved from the interrupt trap location and is saved in the interrupt stack. The address of the trap location is stored in I, the new priority level is stored in $EF and the mask register, M, is set using the corresponding entry in the mask table. Interrupts are enabled and control is transferred to the primary processor routine specified by the third word after the trap location.

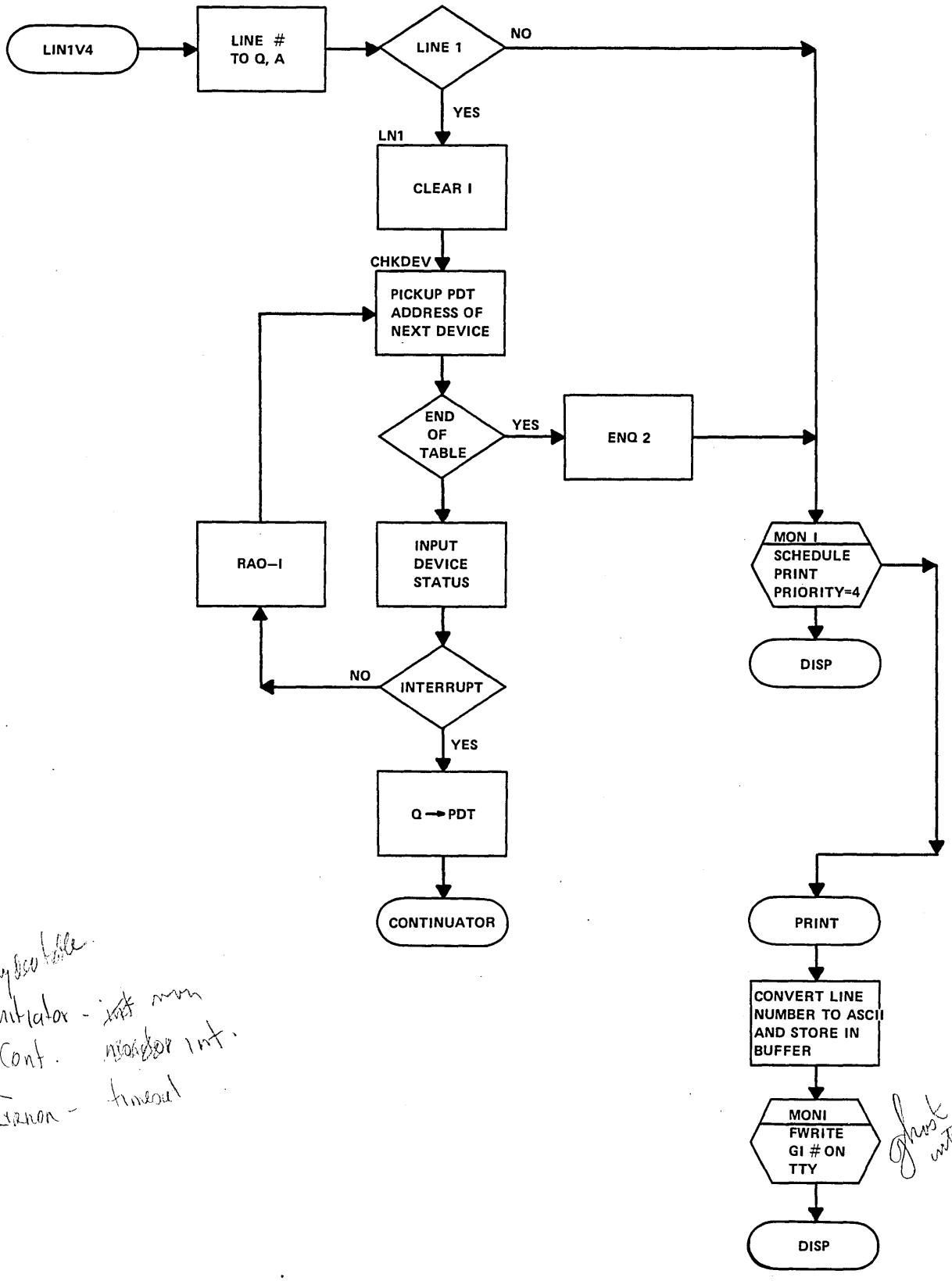
The mask table, MASKT (located in SYSDAT), contains an entry for each priority level. The M-register will always be loaded from the entry in MASKT corresponding to the desired priority level. Those interrupt lines that may not interrupt a program of level n are said to be of a lower or equal priority level and their mask bits must be zero for this level and all levels above. Several lines may have the same priority level.

RESTRICTION

The interrupt has not been acknowledged upon exit. The line processor routine must perform this operation.

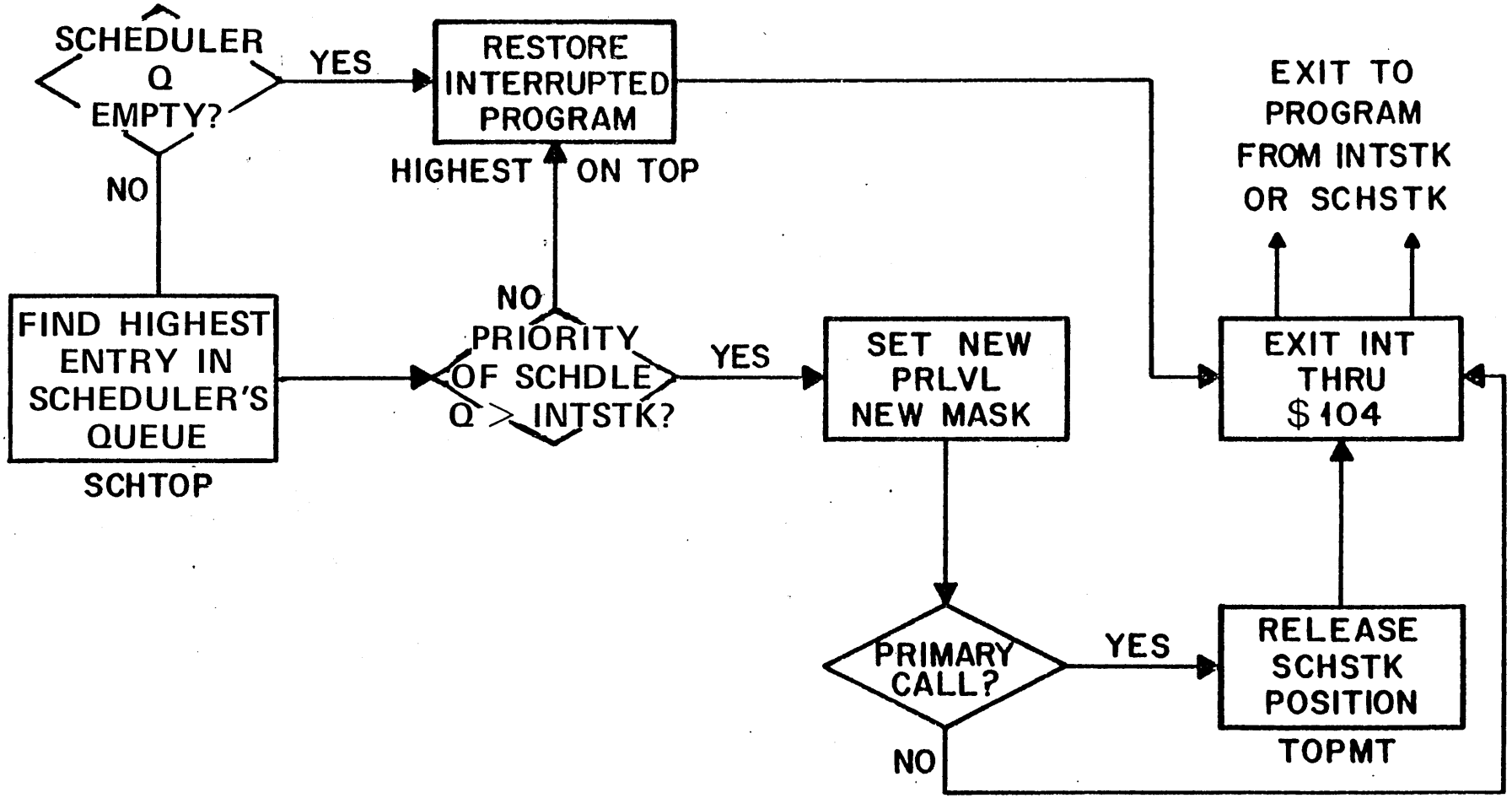

COMMON
ALLIN
PROTECT FAULT?
YES
NO
MOVE P TO $0100
PUT $0102 IN IPROC (TO FAKE INT ON LINE 0)
PRO
ENTRY POINT IN NIPROC
SET UP INTSTK
(Q) → XQ
(A) → XA
OVERFLOW AND PRIVL → XPL
(I) → XI
UPDATE COUNT
WDO → XR
SET NEW PRIORITY LEVEL AND MASK
EIN
I → TRAP
Q → PROCESSOR
PROCESSOR (ADDR IN WD3 OF TRAP)

LINE1V4


LIN1V4
LINE # TO Q, A
LINE 1
NO
YES
LN1
CLEAR I
CHKDEV
PICKUP PDT ADDRESS OF NEXT DEVICE
END OF TABLE
YES
ENQ 2
RAO–I
INPUT DEVICE STATUS
MON I
SCHEDULE PRINT PRIORITY=4
DISP
NO
INTERRUPT
YES
Q → PDT
CONTINUATOR
PRINT
CONVERT LINE NUMBER TO ASCII AND STORE IN BUFFER
MONI
FWRITE GI # ON TTY
DISP

4-8

DISPATCHER


SCHEDULER Q EMPTY?
YES
NO
RESTORE INTERRUPTED PROGRAM
HIGHEST ON TOP
EXIT TO PROGRAM FROM INTSTK OR SCHSTK
FIND HIGHEST ENTRY IN SCHEDULER'S QUEUE
SCHTOP
NO
PRIORITY OF SCHDLE Q > INTSTK?
YES
SET NEW PRLVL NEW MASK
EXIT INT THRU $104
PRIMARY CALL?
YES
RELEASE SCHSTK POSITION
TOPMT
NO

DISPATCHER

INTERNAL SYMBOLS

SCHSTC Routine to initiate a program when taken from the scheduler thread

DISP Start of Program Dispatcher

COMEXT Defined by an EQU and determines the interrupt trap slot location to be used as a common exit

SCHTOP Top of the schedulers thread

DISPATCHER FUNCTION

Whenever a program terminates, it will give control to the Program Dispatcher*. The Program Dispatcher decides which program shall be initiated next. It could be a program previously interrupted and waiting on the interrupt stack, or a program that has been scheduled and is waiting in the scheduler thread. The highest priority program is then initiated by the Program Dispatcher and control given to it.

ENTRY INTERFACES

Entered via a jump to entry point DISP.

EXIT INTERFACES

If control is given to the program that was previously interrupted, the A-, Q-, I-, and M-registers (CYBER 18, R1-R4), and the overflow are restored to their previous condition, as well as priority level. Interrupts are enabled, and control returns to the location at which the interrupt originally occurred.

If control is given to a program on the scheduler thread, A will contain the address of the scheduler thread entry, Q will contain the fourth word of the entry (the original Q in scheduler calls, or an error indication in I/O calls, or coreclock ($E8) in timer calls), priority level and M will contain the mask associated with the priority level, and I and overflow will be in arbitrary configuration. Interrupts are enabled.

INTERNAL DESCRIPTION

After the program is entered, a test is made to determine whether the priority of the highest interrupted program is greater than or equal to the priority of highest program waiting in the scheduler thread. If the interrupted program is to be resumed, the return address is stored in the common exit and I and A are restored. Then, the interrupt stack base is adjusted down by 5 and stored in COUNT, and the priority level restored into the cell containing priority level. The mask associated with this level is transferred into M (which restores M), and then Q is also restored. Control is returned to the interrupted program by an EXI instruction which enables interrupts and jumps to the address in word 0 of that Interrupt Trap Region (Overflow is restored in some systems).

---

* Protected programs may also terminate with a RELEAS request which jumps to the Program Dispatcher.

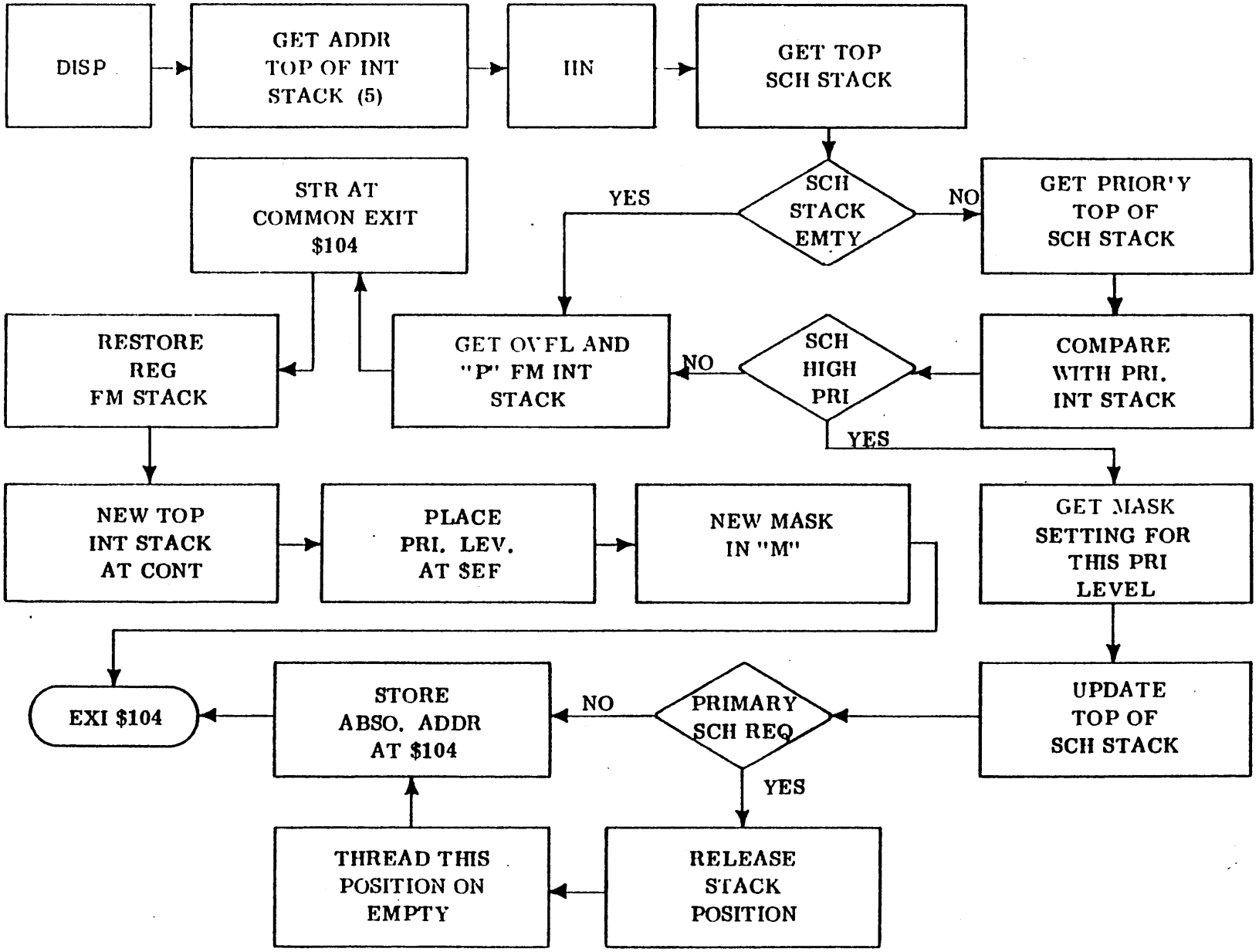
If the program of highest level is on the scheduler thread, the priority specified in the highest thread entry (the address of this entry is in SCHTOP) is placed into the cell containing priority level, and the associated mask placed into M. Then SCHTOP is updated pointing now to the next entry in the thread. If there is no other entry, it contains -0.

Next, a test is made whether the scheduler thread entry was a primary scheduler's request (i.e., not resulting from a completed I/O call or an expired timer call) and is therefore, in the scheduler's stack.

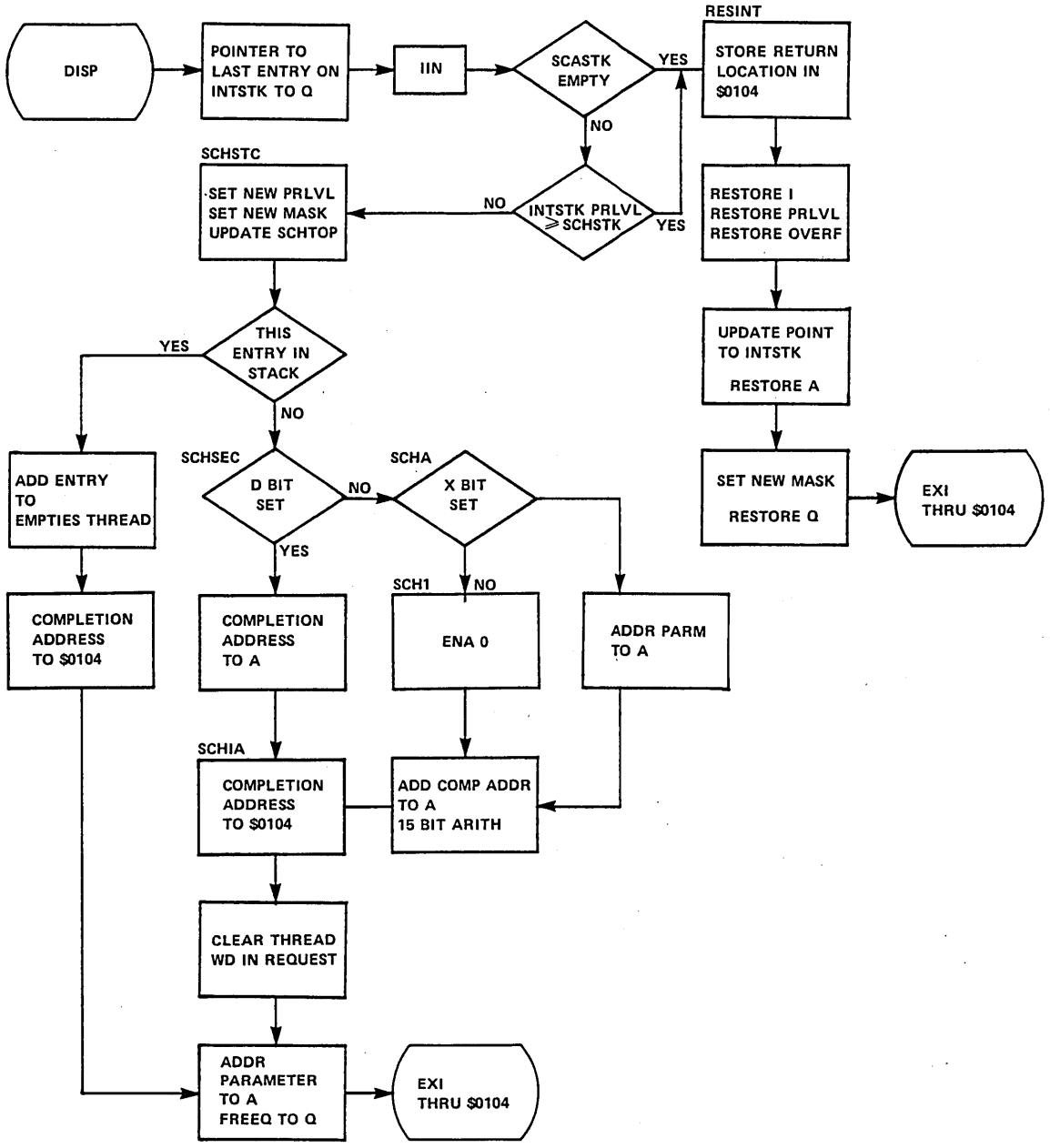
If yes, the scheduler stack position is added to the thread of "empties" and the address to which control is to be given is stored in the common exit. Then the address of the entry is put into A and the fourth word of the call into Q. Control is transferred with an EXI instruction which enables interrupts and transfers control to the address in location $104.

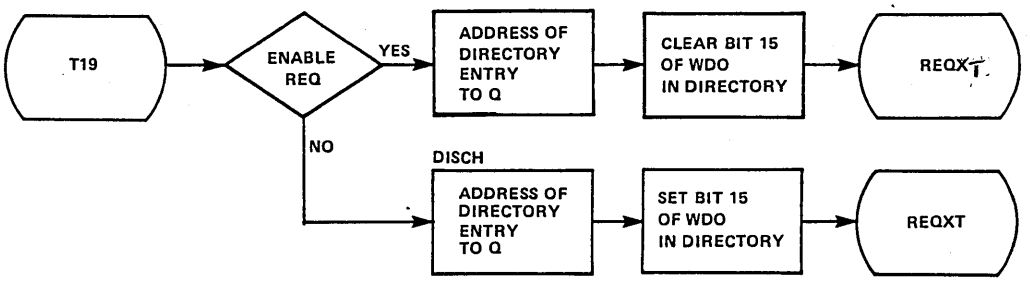
If the scheduler thread entry resulted from an I/O or Timer call (that is, it was a secondary scheduler request), the specified completion location may be relative. If it is, the absolute address is determined and the address stored in the common exit. Then the third word of the entry (containing the thread) is set to 0 as an indication that the call is completed and could be made again. A and Q are loaded and Control is transferred as above.

DISPATCHER

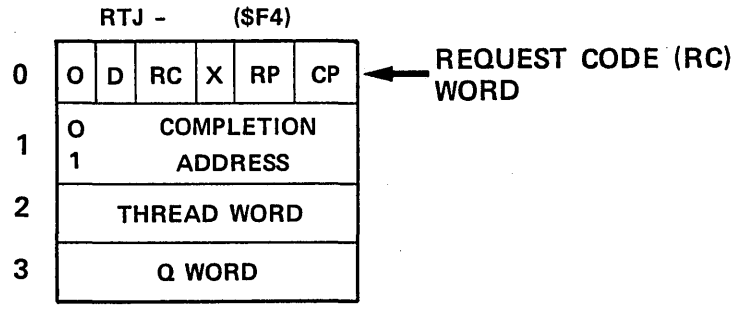

DISP
GET ADDR TOP OF INT STACK (5)
IIN
GET TOP SCH STACK
SCH STACK EMTY
YES
NO
GET PRIOR'Y TOP OF SCH STACK
STR AT COMMON EXIT $104
RESTORE REG FM STACK
GET OVFL AND "P" FM INT STACK
NO
SCH HIGH PRI
COMPARE WITH PRI. INT STACK
YES
NEW TOP INT STACK AT CONT
PLACE PRI. LEV. AT $EF
NEW MASK IN "M"
GET MASK SETTING FOR THIS PRI LEVEL
EXI $104
STORE ABSO. ADDR AT $104
NO
PRIMARY SCH REQ
UPDATE TOP OF SCH STACK
YES
THREAD THIS POSITION ON EMPTY
RELEASE STACK POSITION

NDISP – DISPATCHER SECTION





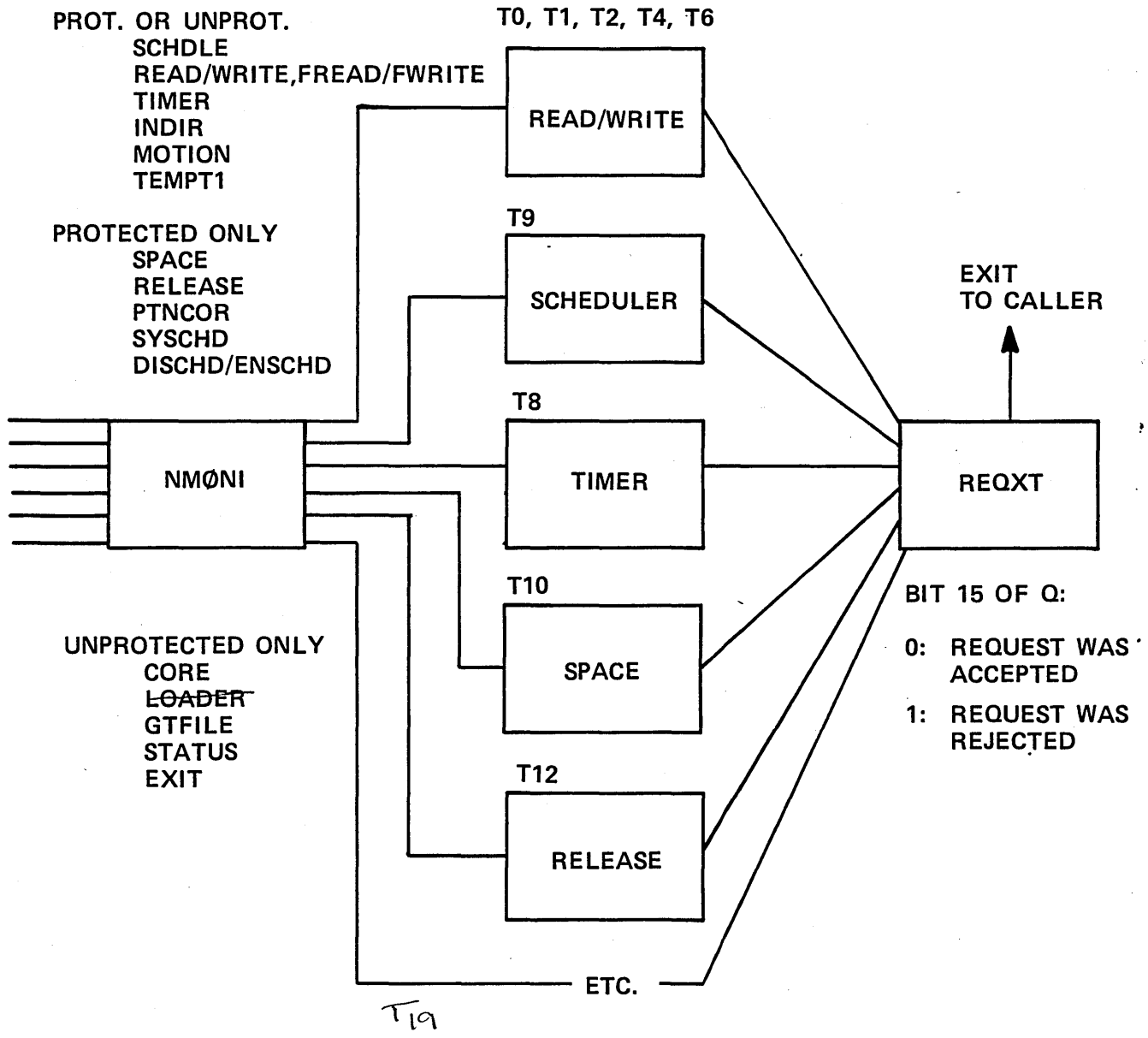
GENERAL FORMAT OF A SYSTEM REQUEST

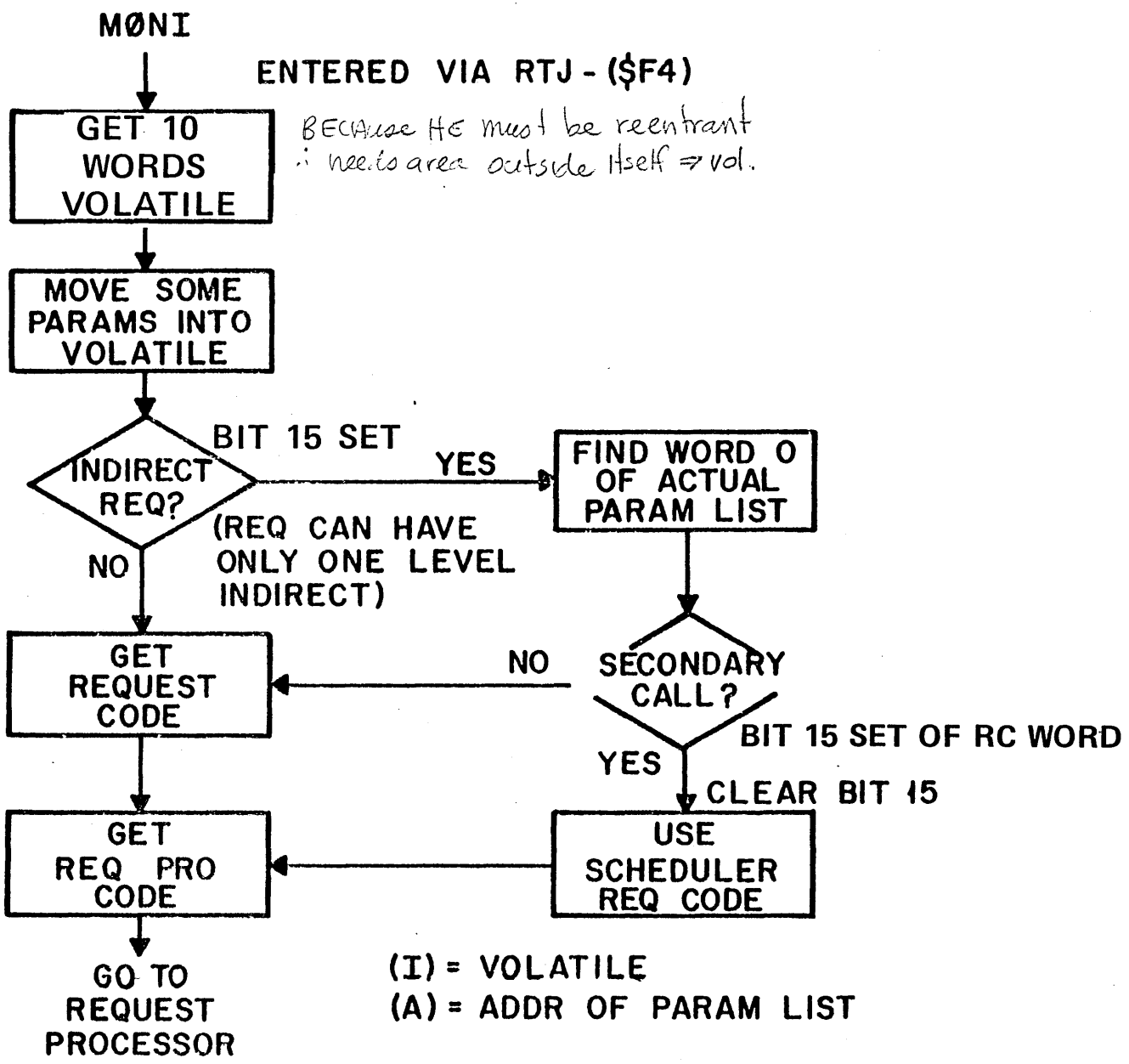


FOR EXAMPLE:

```
0022 P003B 54F4    SCHPRT   RTJ-   ($F4)        SCHEDULE PRINT AT PRIORITY 0
0023 P003C 1200             NUM    $1200        BIFORE EXIT, TO DROP PRIORITY
0024 P003D 0040 P           ADC    PRINT        BACK TO 0.
0025 P003E 54F4             RTJ-   ($F4)
0026 P003F 0A00             NUM    $A00         EXIT REQUEST
0027 P0040 54F4    PRINT    RTJ-   ($F4)
0028 P0041 0401             NUM    $0401        PRINT,CP=1 RP=0
0029 P0042 0049 P           ADC    COMPPR       COMPLETION ADDRESS
0030 P0043 0000             NUM    0,$1009,35 35 WORDS ON THE TTY
     P0044 1009
     P0045 0023
0031 P0046 0002 P           ADC    BUF          FWA BUFFER
```

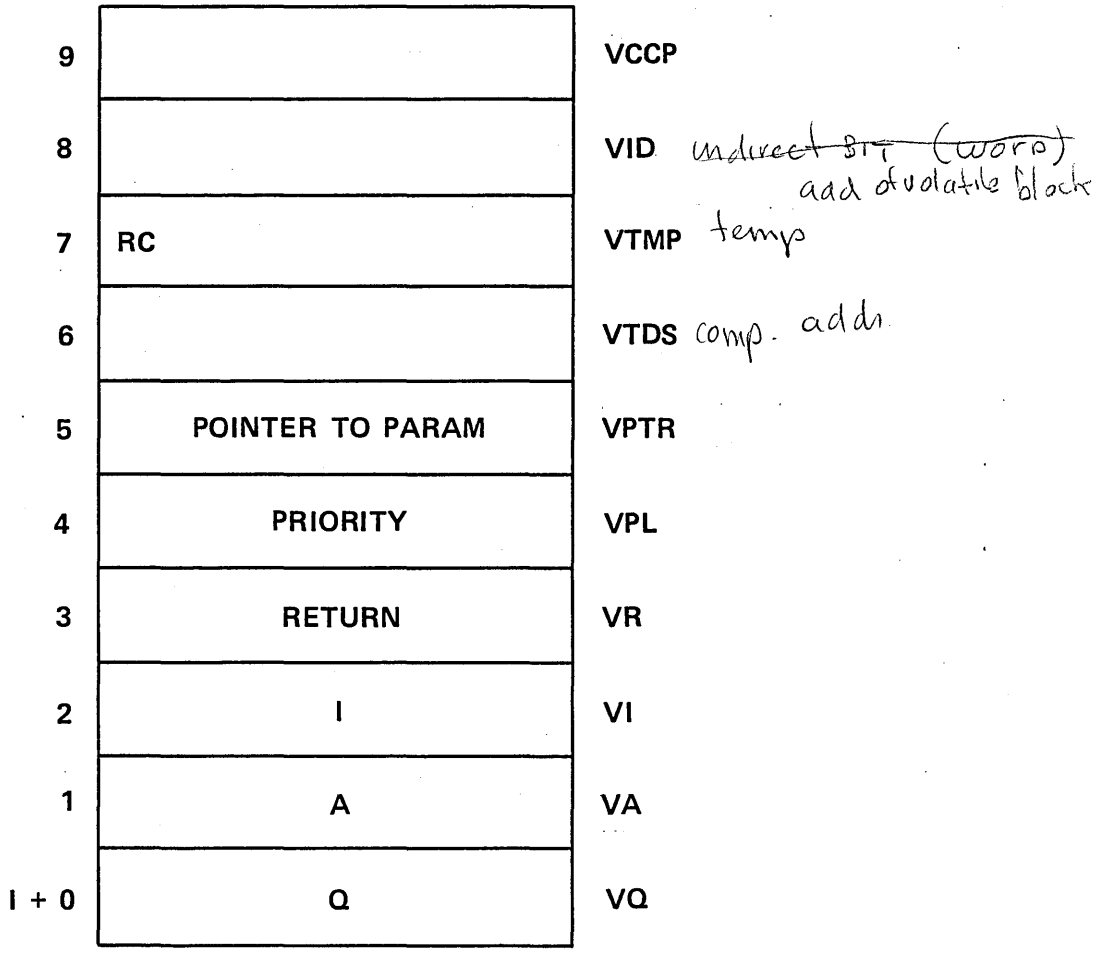
Routine EESORT




MØNI
ENTERED VIA RTJ - ($F4)
GET 10 WORDS VOLATILE
BECAUSE HE must be reentrant
∴ needs area outside itself ⇒ vol.
MOVE SOME PARAMS INTO VOLATILE
INDIRECT REQ?
BIT 15 SET
YES
(REQ CAN HAVE ONLY ONE LEVEL INDIRECT)
NO
FIND WORD 0 OF ACTUAL PARAM LIST
SECONDARY CALL?
NO
BIT 15 SET OF RC WORD
YES
CLEAR BIT 15
GET REQUEST CODE
USE SCHEDULER REQ CODE
GET REQ PRO CODE
GO TO REQUEST PROCESSOR
(I) = VOLATILE
(A) = ADDR OF PARAM LIST

MONI GETS VOLATILE

STORAGE FOR EACH REQUEST



MONITOR ENTRY AND EXIT FOR REQUESTS

INTERNAL SYMBOL DEFINITIONS

VR	Relative location in volatile containing the user program's return address. (Equals 3.)
VPTR	Relative location in volatile containing the pointer to the user program's parameter list (5).
ZERO	A location in the communication region containing a zero. ($22).
ONEBIT	The first location of a table constructed so that entry n contains 2^n. This is normally $23.
RCSCHD	The code for the scheduler request (9).
LPMSK	The first location of a table constructed such that entry n contains 2^n-1. This is normally location 2.
VTMP	Relative location in volatile containing the request code (7).
AMONI	A location in the communication region containing the location of this program. This is normally location $F4.
V	The number of words of volatile allocated per request (10).
AREQXT	A location in the communication region containing REQXT (request exit). This is normally $B9.
MONI	The subroutine entry point to the Request Entry Processor.
RCTV	The Request Code transfer vector containing the names T_0 through T_{30}.
REQXT	Common Exit for monitor requests.

PROGRAM FUNCTION

User programs generate requests for various functions such as I/O, core allocation, and scheduling. All of these requests are processed by the Request Entry Processor. Its function is to reserve volatile storage, save the registers A, Q, P, and I in volatile storage, and give control to one of the request processor routines $T_0 \ldots T_{30}$, depending upon the request code, RC, in the user's calling sequence.

ENTRY INTERFACES

Entered from protected programs as a result of a monitor call. Entered from unprotected programs via IPROC.

EXIT INTERFACES

The Request Entry Processor gives control to the request processors, T_0 through T_{30}, with specific information in the registers. Each request processor upon entry can assume the following:

REGISTER	CONTENTS
A	A_{14-0} is the location of the parameter list. If $A_{15} = 0$, then the reference to the parameters in the call was direct. Otherwise, $A_{15} = 1$, and the reference was indirect (an INDIR request).
Q	Absolute address of the request processor being executed.
I	I contains the location of a ten (10) word block of volatile storage.

LOCATION	MNEMONIC	
(I) + 0	VQ	The user's Q-register is saved here.
(I) + 1	VA	The user's A-register is saved here.
(I) + 2	VI	The user's I-register is saved here.
(I) + 3	VR	The return address of the user. If this was an indirect call, then the return address has been incremented by one (1) to give the correct return address. Otherwise, this was a direct call and the return address must be adjusted by the request processor.
(I) + 4	VPL	Not set by the Request Entry Processor. Intended to hold the request priority level.
(I) + 5	VPTR	The location of the user's parameter list. This is in the accumulator A. See discussion of A above.

LOCATION	MNEMONIC	(Continued)
(I) + 6	VTDS	Not set by the Request Entry Processor. It is intended to contain the top of the stack for the desired logical unit.
(I) + 7	VTMP	A temporary storage cell containing the request code, RC.
(I) + 8	VID	
(I) + 9	VCCP	Control Point Number (ITOS)

RETURN TO REQUESTER

Control will be returned to the next instruction with the registers A, Q, and I restored. Overflow will not be saved. Interrupts will be enabled and the priority level will be the same as upon entry.

INTERNAL DESCRIPTION

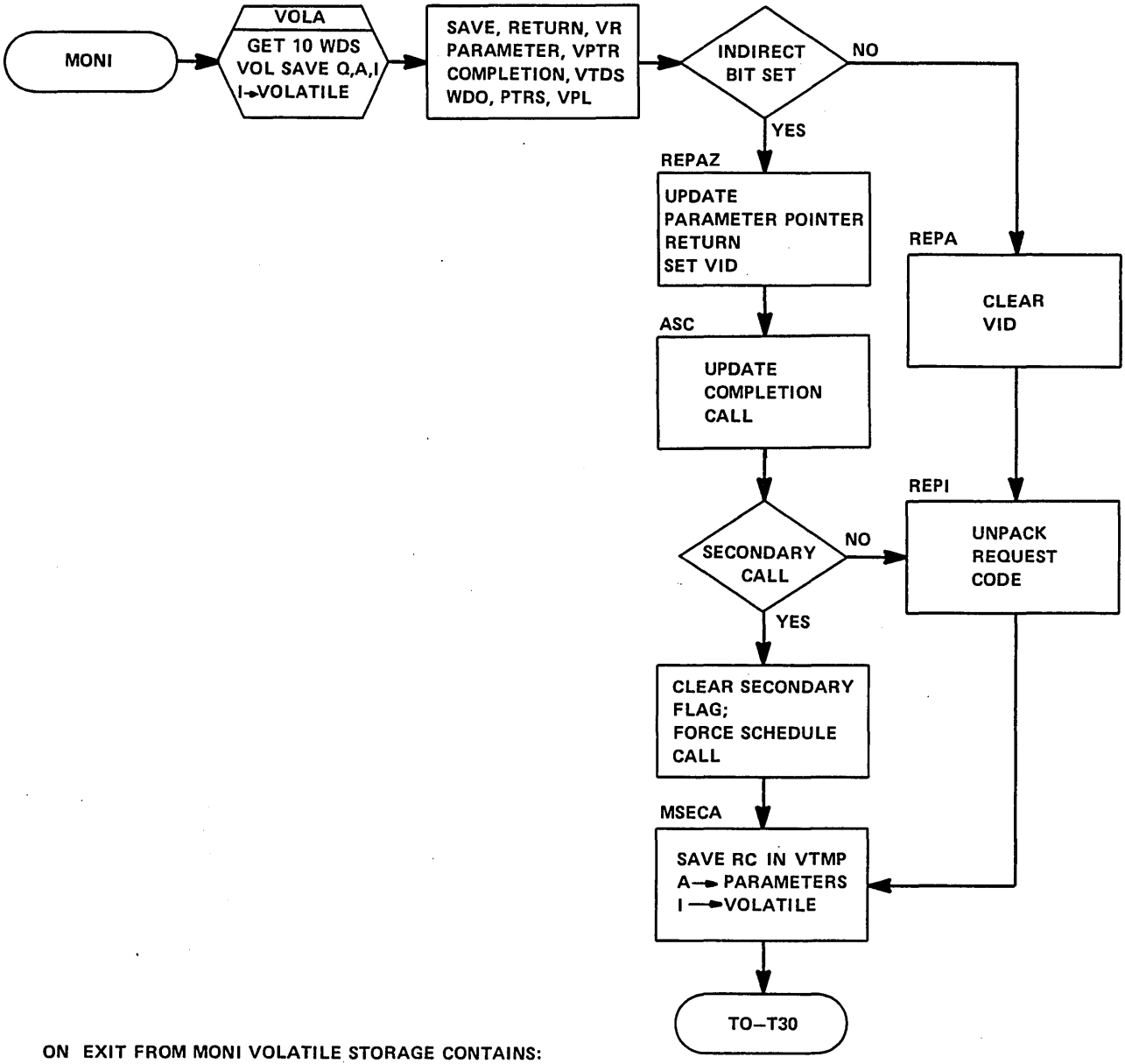
The Request Entry Processor handles all monitor requests made by the user program. The user enters the Request Entry Processor via an indirect return jump to MONI. The Request Entry Processor inhibits all interrupts, saves the user's registers Q, A, I, and return address in an area unique to this request, and then enables interrupts. The Request Entry Processor is re-entrant beyond this point, and works only with the data area unique to this request. The I-register is used to hold the address of this unique area which is called volatile storage. The location of the parameter list is then stored in volatile. If this request has an indirect reference to the parameter list, the return address to the program is adjusted to return control to the next sequential instruction. If this indirect call was made as the result of the completion of an I/O operation, the registers are adjusted to make this look like a scheduler call since the request code in the user's request parameter list may not be altered. Control is then given to the request processor specified by the request code.

RESTRICTIONS

The I-register must be conserved throughout the request processor called since it contains the address of volatile storage. Each request processor must be re-entrant since it runs at the requestor's level. When each request processor finishes, it must return the volatile core storage by jumping to REQXT.

Label	Op	Address	
	JMP-	(AREQXT)	Address of request exit. REQXT is contained in AREQXT.

NOTE: The "MINI MONITOR REQUEST ENTRY" is identical in every way with this module with a single exception: it is equipped to handle only 13 requests.



ON EXIT FROM MONI VOLATILE STORAGE CONTAINS:

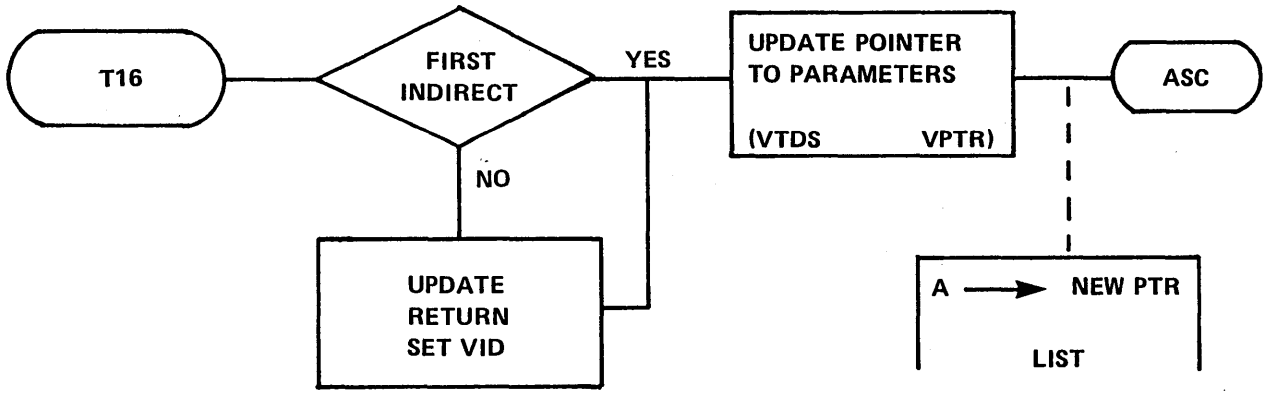
VQ	CALLER'S Q
VA	CALLER'S A
VI	CALLER'S I
VR	RETURN (1ST WD AFTER CALL)
VPL	WD 0 OF PTRS
VPTR	ADDRESS OF PARAMETERS
VTDS	WD 1 OF PTRS
VTMP	REQUEST CODE
VID	INDIRECT FLAG
VCCP	

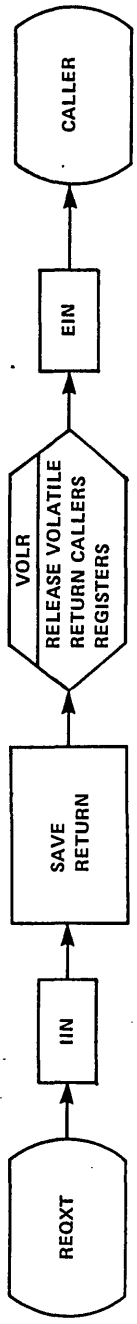
IF SECONDARY SCHEDULE; CALL RC≠9

(1) POINTS TO VOLATILE; (A) POINTS TO PTR LIST

T16

ENTRY POINT	T16	ENTERED ONLY FROM MONI ON PART 1 INDIRECT REQUESTS.
EXTERNAL	ASC	ENTRY POINT IN MONI.




REQXT
IIN
SAVE RETURN
VOLR
RELEASE VOLATILE RETURN CALLERS REGISTERS
EIN
CALLER

STUDY QUESTIONS - 4

1. How can you tell if the scheduler stack is full? IF TOP OF FREE LIST (4B4) IS FULL (FFFF)

2. Define primary and secondary scheduler request.
1° requests lic in scheduler stack. When completed are returned to free list
2° somewhere else & don't get added to Free List

3. How can the DISP tell if a scheduler request was primary or secondary?
IF ADDR is with boundaries of sched. stack then = 1°

4. What are the functions of the Common Interrupt Handler?
Save state of Machine int. on stack
Set new priority level of system
JUMP TO correct processor.

5. What interrupts are recognized on LINEO and what action takes place for each?
Parity/Protect Parityerror: if = find cell with parity error
monitor assumes DSA Problem. 'DSA'

6. Why does a Request Processor need to exit through REQXT?
Needs Volatile
Needs to release Volatile ata "common exit point)"

Protect Fault: Was background active Y - passes control to BKG protect processo
BKG probably making Moni reqst
N - Something wrong => SysFail

Line0 protect fault = memory protect fault.

line 0 int also can be powerfail. System saves its own status, saves pwr 0
then when power comes back it becomes at 0

LESSON GUIDE 5

SCHEDULER

LESSON PREVIEW:

The basic functions of the scheduler will be discussed.

INSERTS ITEMS ON SCHEDULER STACK.

Software interrupt if new thing is ↑ priority

can sched. ordinals addresses

REFERENCES:

Listing of scheduler, T9.

ordinal really = addr. in sys. Library

ordinal can only be on sche. stack 1x

TRAINING AIDS:

became thread word = busy

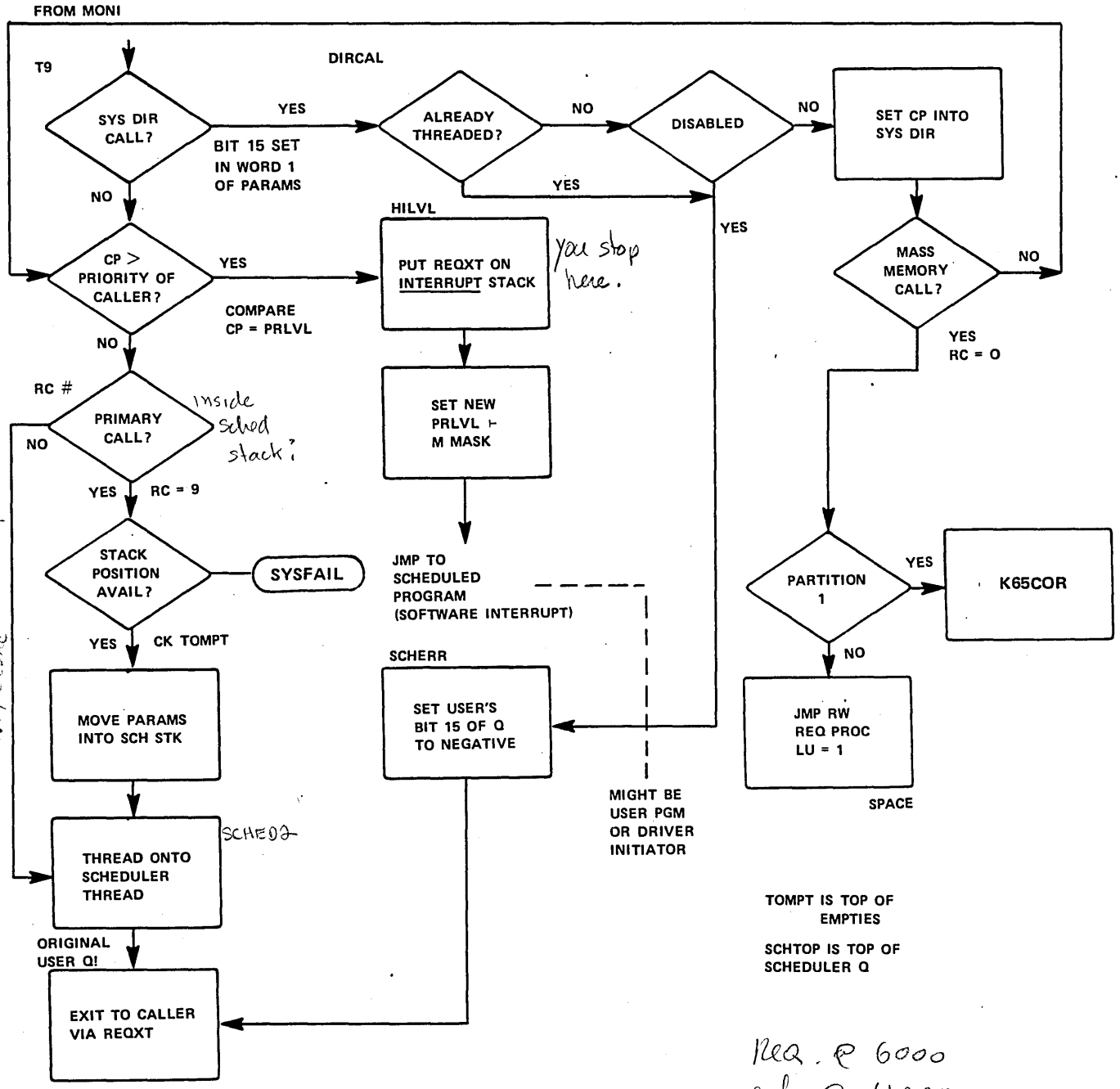
once brought into core can be sched again

PROJECTS:

OBJECTIVES:

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

1. Determine the events generated by SCHDLE.
2. Discuss the significance of the scheduler in MSOS.



request / store. app to this?

If sched. req. accepted Q = +
rejected Q = −

Req. @ 6000
sch @ 4000
Δ = -2000
comp 5FFF (2/7 = 5)

Q → 17 6000
sct 15 E000
+5FFF
1 3FFF
4000

SCHEDULER

SYMBOLS

T10	Entry point of SPACE request
SCHTOP	Location in NDISP containing location of top entry in schedule stack

FUNCTION

In a given system, numerous requets for the execution of programs at specific priority levels may be generated. Specifically, these requests are generated when:

a. an I/O request has been completed,

b. a specified time interval has elapsed,

c. core has been allocated,

d. System Director SCHDLE request has been executed.

These requests are called Secondary Scheduler Requests. Requests may also be made by any program directly. They are called Scheduler Requests or Primary Scheduler Requests.

It is the function of the Scheduler Request Processor to:

a. cause the immediate execution of a requested program if it is of a higher priority level than the requesting (current) program, or

b. thread the request by priority and within a priority by first-in-first-out, if its priority is the current priority.

If the requested program is mass memory resident, the Scheduler Request Processor will cause allocation of core for this program and transfer of the program from mass memory. After the program has been transferred, a Scheduler Request is made, which results in a. or b. above.

Whenever a program terminates, the Program Dispatcher will select the next program to be run, either from the top of the scheduler thread or the interrupt stack.

ENTRY INTERFACES

Progrm is entered from the Request Entry Processor. The calling (requesting) program must have interrupts enabled.

EXIT INTERFACES

The program exits either to the requested program (completion address), if the level is higher than the current one, or to the request exit.

In the first case, the priority level, I and the return address leading to the request exit are saved in the proper positions of the interrupt stack and its base adjusted. A, Q, and I are saved in volatile, which is not released until the requested program terminates. I contains the base of volatile storage, when control is given to the requested program.

Interrupts are enabled and the requested priority level and mask set.

In the second case the request has been threaded. Control goes to REQXT to restore the registers for the requestor and enable interrupts.

INTERNAL DESCRIPTION

All Scheduler Requests are identified by the request entry processor, which also allocates a sufficient amount of volatile storage for reentrancy purposes. Then control is given to the Scheduler Request Processor (Symbol T9). Interrupts are enabled and I contains the base address of the allocated volatile storage. Volatile is organized in the following manner:

(I) + 0 contains Q
(I) + 1 contains A
(I) + 2 contains Priority Level of Request
(I) + 3 contains Return Address
(I) + 4 contains I
(I) + 5 contains Pointer to Request Parameter List
(I) + 6 contains First Word of Request (Temp.)
(I) + 7 contains Second Word of Request (Temp.)
etc.

First, the return address is adjusted by two locations unless the call was indirect, in which case it had already been adjusted by the Request Entry Processor. Then word 1 and 2 of the call are stored in volatile temporarily. If the call is a directory call control is given to DIRCAL. If not a directory call, a test is made to see if the requested program is of higher level than the current one, in which case control transfers to HILVL.

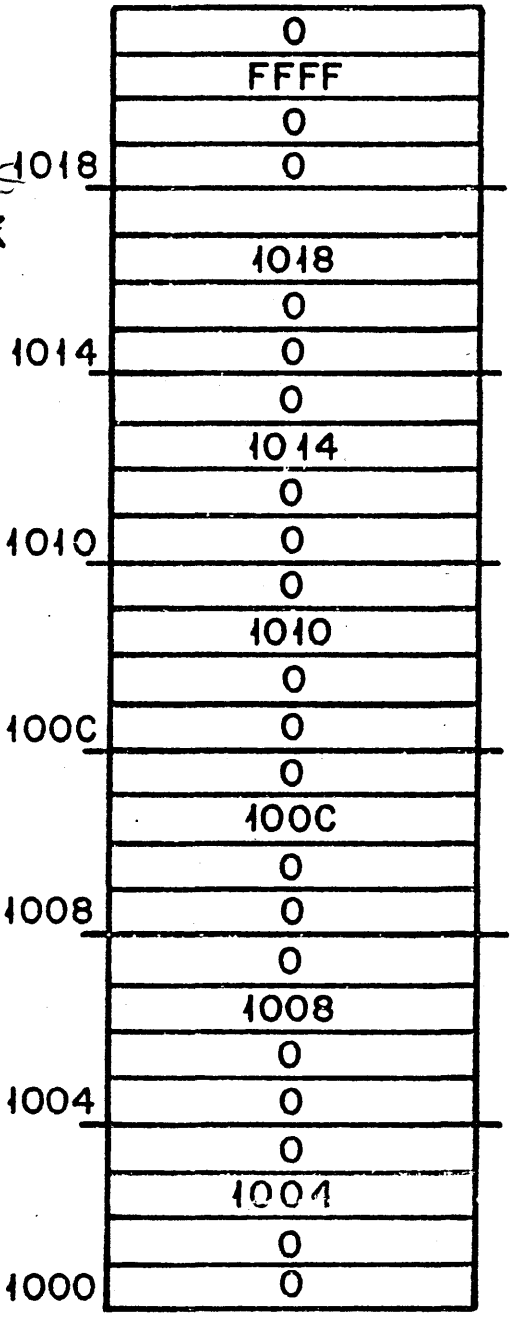
Otherwise, a test for a primary call (SCHDLE request) is made and only then, if it is not a directory call, not of a higher level and not a secondary call, is a position in the Scheduler Stack obtained and the request transferred from volatile (I) + 6 and (I) + 7 into the stack.

The current priority level and I are saved in the interrupt stack and the interrupt stack base address count is incremented by (5). The request exit is stored as the return address since upon return from the program volatile must be restored as well as A and Q. Then the requested priority level and the associated mask are set and control is given to the new "GO TO" Address.

SCHEDULER ~~STACK~~ QUEUE
AFTER AUTOLOAD

SCHTOP = FFFF

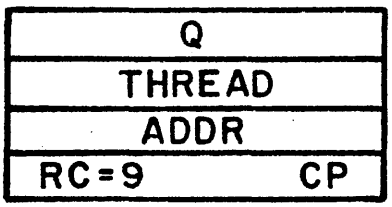
TOMPT = 1000

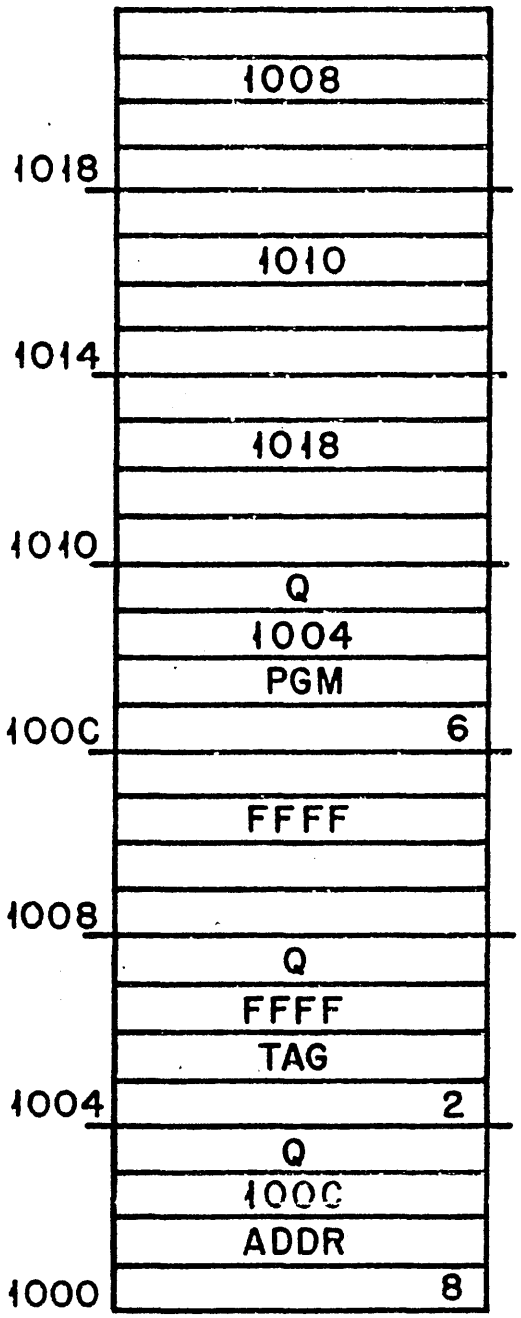


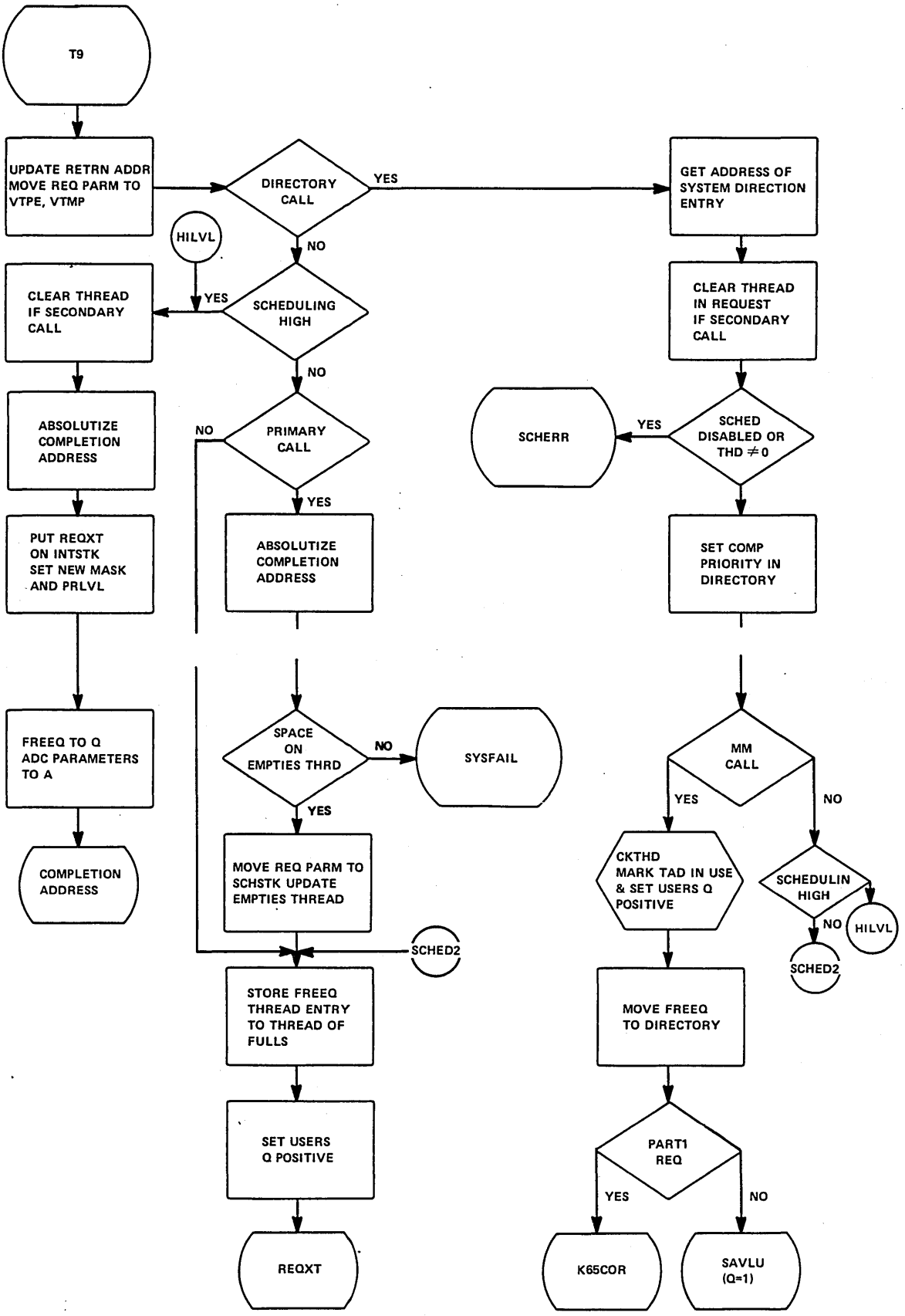
Q
THREAD
ADDR
RC=9 CP

SCHTOP = 1000

TOMPT = 1014






T9
UPDATE RETRN ADDR
MOVE REQ PARM TO
VTPE, VTMP
DIRECTORY
CALL
YES
GET ADDRESS OF
SYSTEM DIRECTION
ENTRY
HILVL
NO
CLEAR THREAD
IF SECONDARY
CALL
YES
SCHEDULING
HIGH
CLEAR THREAD
IN REQUEST
IF SECONDARY
CALL
NO
ABSOLUTIZE
COMPLETION
ADDRESS
NO
PRIMARY
CALL
SCHERR
YES
SCHED
DISABLED OR
THD ≠ 0
YES
PUT REQXT
ON INTSTK
SET NEW MASK
AND PRLVL
ABSOLUTIZE
COMPLETION
ADDRESS
SET COMP
PRIORITY IN
DIRECTORY
FREEQ TO Q
ADC PARAMETERS
TO A
SPACE
ON
EMPTIES THRD
NO
SYSFAIL
MM
CALL
YES
NO
COMPLETION
ADDRESS
YES
CKTHD
MARK TAD IN USE
& SET USERS Q
POSITIVE
SCHEDULIN
HIGH
NO
HILVL
SCHED2
MOVE REQ PARM TO
SCHSTK UPDATE
EMPTIES THREAD
SCHED2
STORE FREEQ
THREAD ENTRY
TO THREAD OF
FULLS
MOVE FREEQ
TO DIRECTORY
SET USERS
Q POSITIVE
PART1
REQ
YES
NO
REQXT
K65COR
SAVLU
(Q=1)

LESSON GUIDE 6

INTRODUCTION TO SYSTEM I/O

LESSON PREVIEW:

This lesson covers the Physical Device Table, LOG1, LOG1A, and LOG2 tables. The T/W Request Processor is also discussed. Emphasis will be placed on dump analysis as a method of determining the state of a given peripheral device.

REFERENCES:

Chapters 1 and 2 of Software Peripheral Drivers RM
Listing of SYSDAT and RW

TRAINING AIDS:

PROJECTS:

1. Student Project - 6
2. Study questions - 6

OBJECTIVES:

At the completion of this lesson, the student will be able to:

1. Understand the function and purpose of Physical Device Table, LOG2, LOG1A, LOG1.
2. Find the Physical Device Table in a dump for a particular logical unit.
3. Interrupt the information in the dump concerning I/O.
4. Understand and discuss the major functions of the RW processor.

SYSTEM STANDARD LOGICAL UNITS

LUN 1	SPACE DRIVER
LUN 2	DUMMY
LUN 3	DUMMY
LUN 4	COMMENT DEVICE
LUN 6	MT
LUN 7	PSEUDO TAPE MAG Tape Simulator
LUN 8	LIBRARY UNIT
LUN 9	PRINTER (LIST)
LUN 10	CR (STANDARD INPUT)
LUN 11	STANDARD OUTPUT
LUN 12	FORTRAN LIST UNIT

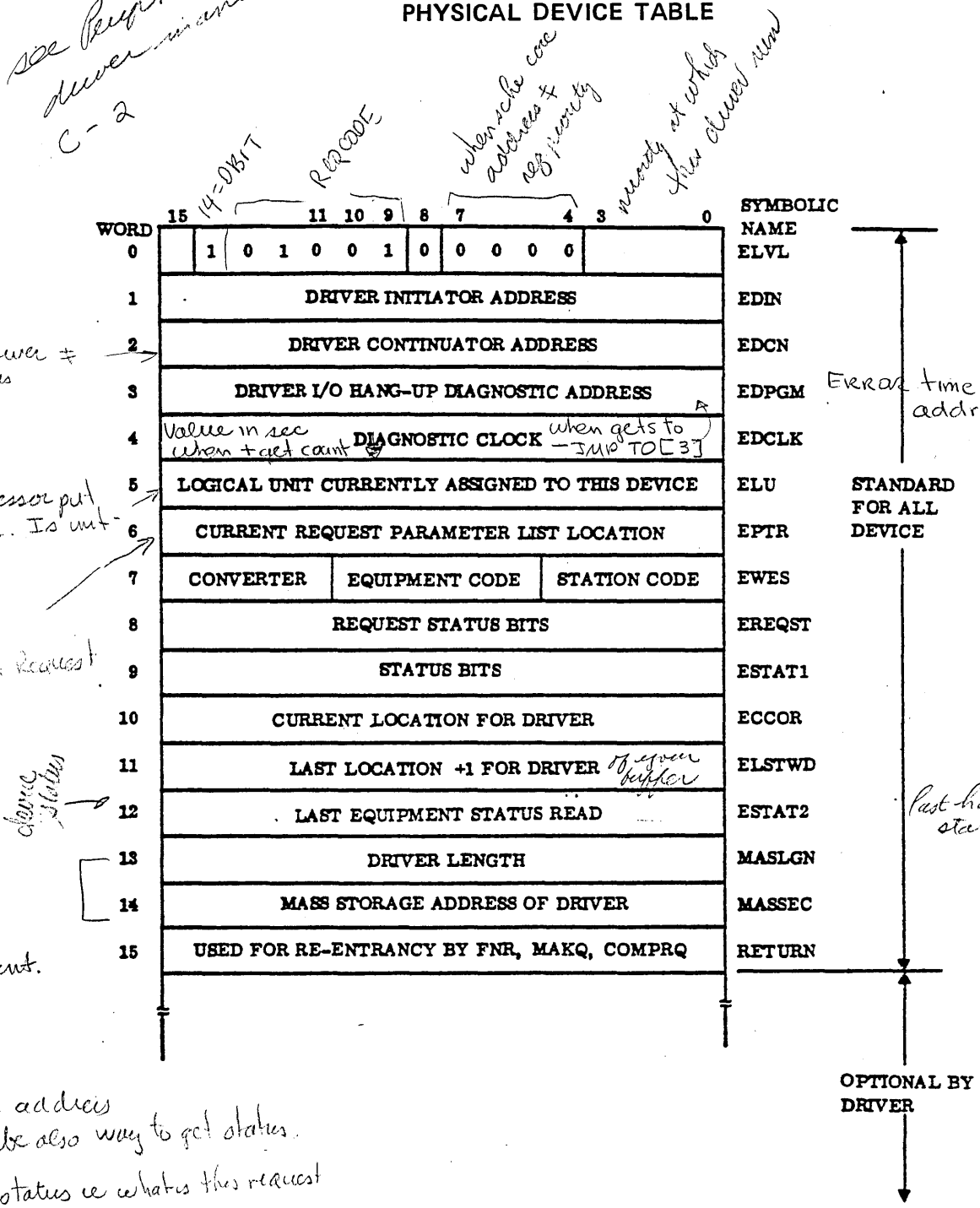
PHYSICAL DEVICE TABLE

Each device has a physical equipment table that contains the interfacing information specified by the user to the device. It contains the entry adresses to the driver responsible for operating the device, the station address that tells the driver which device to use, and the information which allows the driver to fulfill the current request. The table contains at least 16 words for a device. Words 0 through 15 have a standard function for all devices. Additional words are added for use by the output message buffer package and special use by drivers. Drivers written in Kernal form have an additional eight specified words (words 16 through 23). Additional words for these kernal drivers begin at word 24.

The physical device tables are included in SYSDAT (the system and parameters program).

see Peripheral Driver manual C-2

PHYSICAL DEVICE TABLE



pseudo driver # have this

Error time out address.

RW processor put this here. Is initially 0

7) filled in by Find Next Request FNR

device status

Past hardware states.

used only if driver is mass resident.

7) device address maybe also way to get status.

8). request status ie what is this request currently doing?

LØG 1A

LARGEST LEGAL LUN
PHYSTB ADDR FOR LUN 1
PHYSTB ADDR FOR LUN 2
PHYSTB ADDR FOR LUN 3
≷
PHYSTB ADDR FOR LUN N

LØG 1

LARGEST LEGAL LUN	
	ALTERNATE FOR LUN 1
	ALTERNATE FOR LUN 2
	ALTERNATE FOR LUN 3
	≷
	ALTERNATE FOR LUN N

1 2 3 4

bit 15
14 shared bit
13 "down" bit
0 = up
1 = down

if down system tries alternate

bit 14 allows driver to be sched on IX

bit 12 set if down message written

LØG 2

LARGEST LEGAL LUN
TOP OF THREAD LUN 1
TOP OF THREAD LUN 2
TOP OF THREAD LUN 3
≷
TOP OF THREAD LUN N

Scheduler thread for each LU

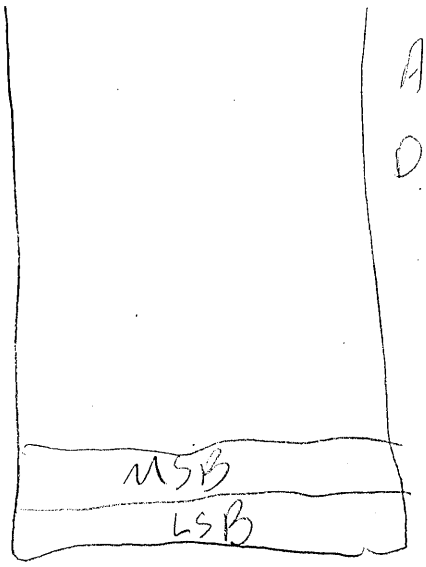
request gets added to this stack

LOGICAL UNIT TABLES

READ/WRITE REQUEST FORMAT

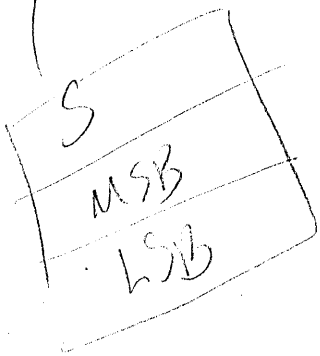
RTJ - ($F4)

0	D	RC = 1, 2, 4, 6	X	RP	CP	
0 1	COMPL ADDR					
THREAD 0						
ERR Q	OMI	OOI	AIO	LUN		
0 1	NO. WORDS (n)					
0 1	S FWA ($)					



FOR EXAMPLE:

```
0013 P002F 54F4   START   RTJ-   ($F4)      INITIATE FREAD
0014 P0030 0201           NUM    $0201       READ, CP=1 RP=0
0015 P0031 0038 P         ADC    COMPRD      COMPLETION ADDRESS
0016 P0032 0000           NUM    0,$100C     THREAD,LUN CR=12,ASCII
     P0033 100C
0017 P0034 0028           NUM    40          ONE CARD TO READ
0018 P0035 0002 P         ADC    BUF         FWA BUFFER AREA
```



PROCESSOR FOR READ, WRITE FORMAT READ, FORMAT WRITE

ENTRY INTERFACES

The Request Processors (T0, T1, T4 and T6) are entered from the Request Entry Processor with the A, Q and I and Volatile set up as shown below.

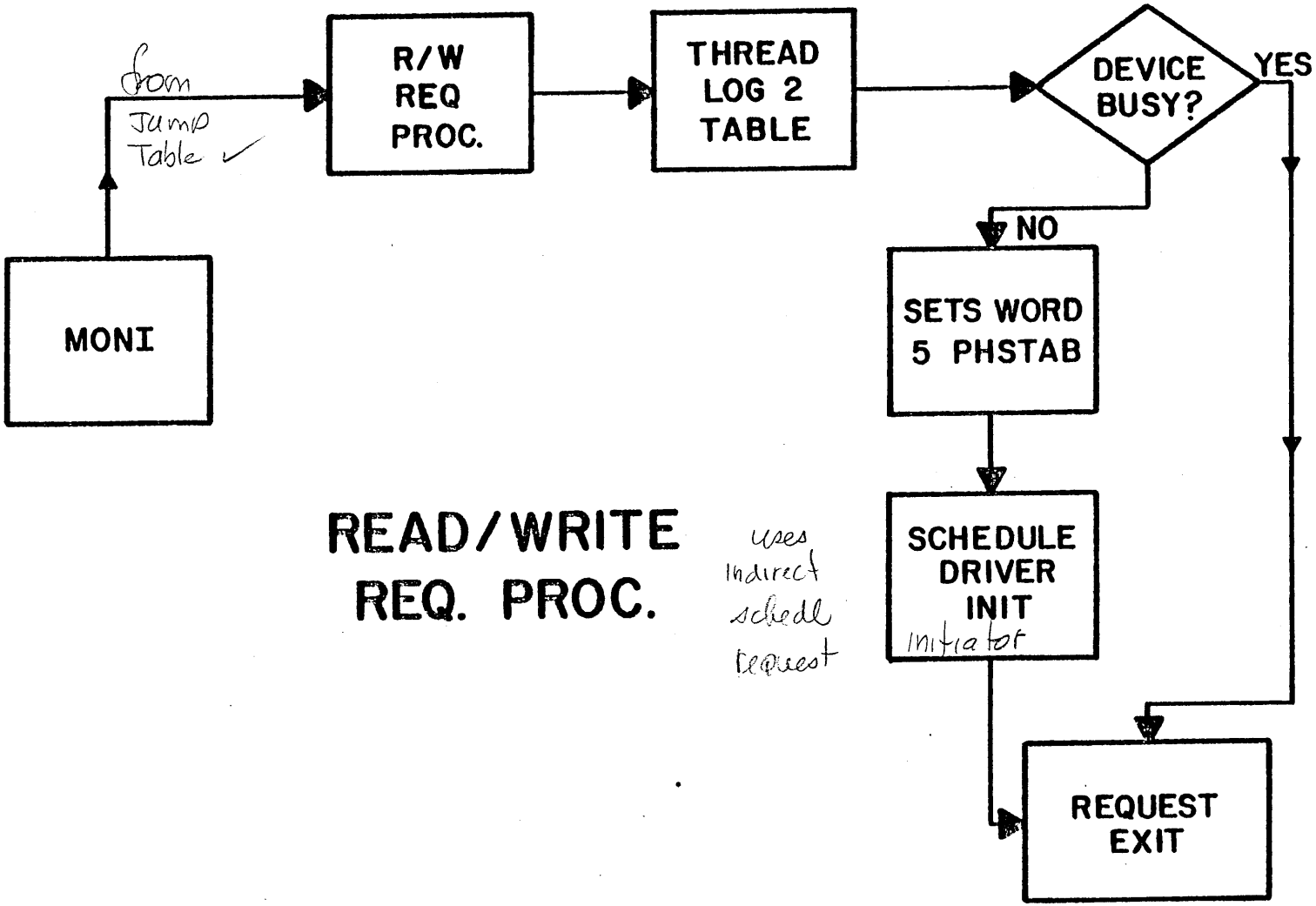
REGISTER	CONTENTS
A	A_{14}-0 is the location of the parameter list. If A_{15}=0, then the reference to the parameters in the call was direct. Otherwise, A_{15} = 1, and the reference was indirect.
Q	Absolute address of the request processor being executed.
I	I contains the location of an 9-word block of volatile.

VOLATILE STORAGE	MNEMONIC	
(I) + 0	VQ	Q saved by Request Entry Processor.
(I) + 1	VA	A saved by Request Entry Processor.
(I) + 2	VPL	Used to hold request priority level.
(I) + 3	VR	P-register saved by Request Entry Processor. If indirect all, P is already incremented by 1 for proper return address.
(I) + 4	VI	The I-register saved by REP.
(I) + 5	VPTR	Used to hold the user's parameter list location, also in A above.
(I) + 6	VTPE	Used to hold the preceding thread location.
(I) + 7	VTMP	A temporary used to hold logical unit number.
(I) + 8	VID	

EXIT INTERFACES

Exit to the Driver:

The driver will be scheduled if the device associated with this logical unit is not busy. The Q register upon entry to the driver Initiator will contain the location of the physical device table entry for the device.



READ/WRITE REQ. PROC.

uses Indirect schedl request

Exit to the User:

The request processor returns control to the REQXT where the volatile storage is released and control is returned to the caller.

Upon return to the user, the registers A, I, and Q_{14-0} will be restored. If $Q_{15=1}$, the thread location in the parameter list is not zero, implying that this request is already on some other thread. In this case, no action will be taken on this call. This action is apparent only to protected callers.

Scheduling of the Completion Address, C

Control will be returned to the Completion Address C at level CP when the I/O requested has finished or if the device is down and no alternate exists. Q will contain word 3 of the parameter list. The high order bits of Q will contain the error code V.

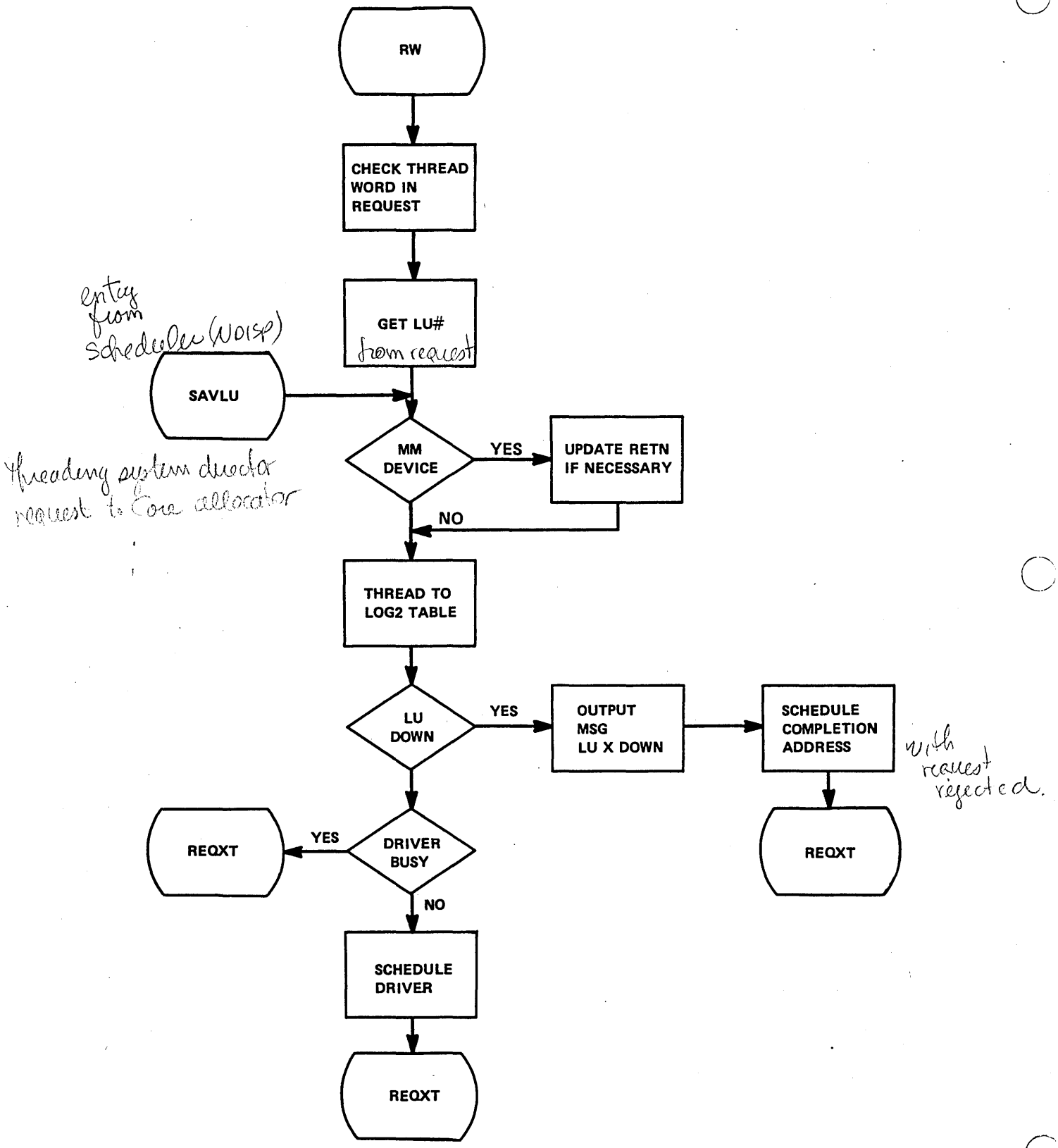
INTERNAL DESCRIPTION

Requests are threaded onto the logical unit according to Request Priority. If the associated device is not assigned to a logical unit and is operational, the driver for the device is called; or, if the device has failed and has no alternate, the completion address is scheduled with an error code indicating failure returned to the completion address. Subroutine ALTSUB, in the Alternate Device Handler, is used to obtain the alternate logical unit if required.

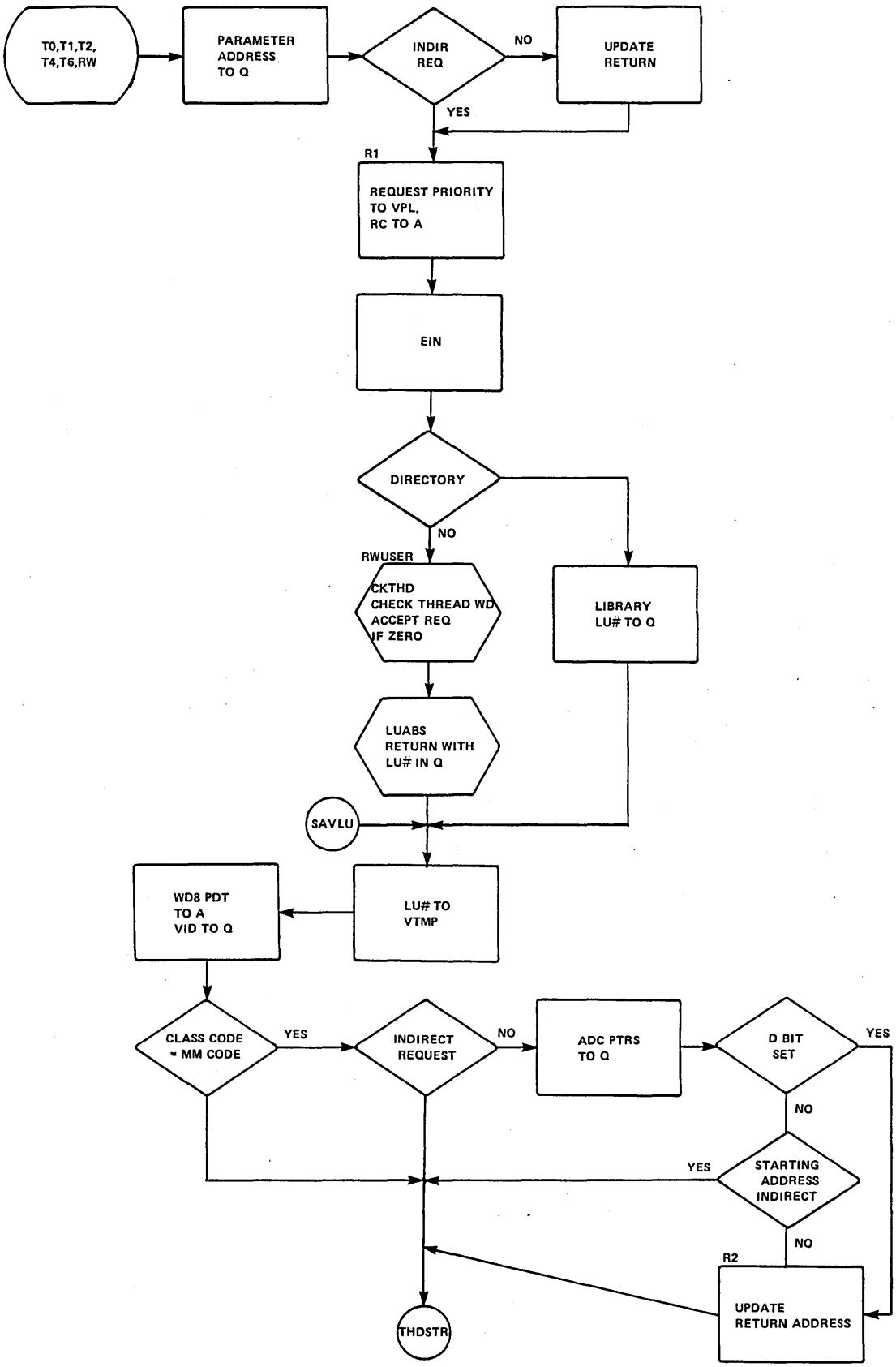
NOTE: THE *MINI* RW PROCESSOR* module is identical to this module. If the *MINI ERROR PROCESSOR* module is used, ALTSUB simply returns to the caller.

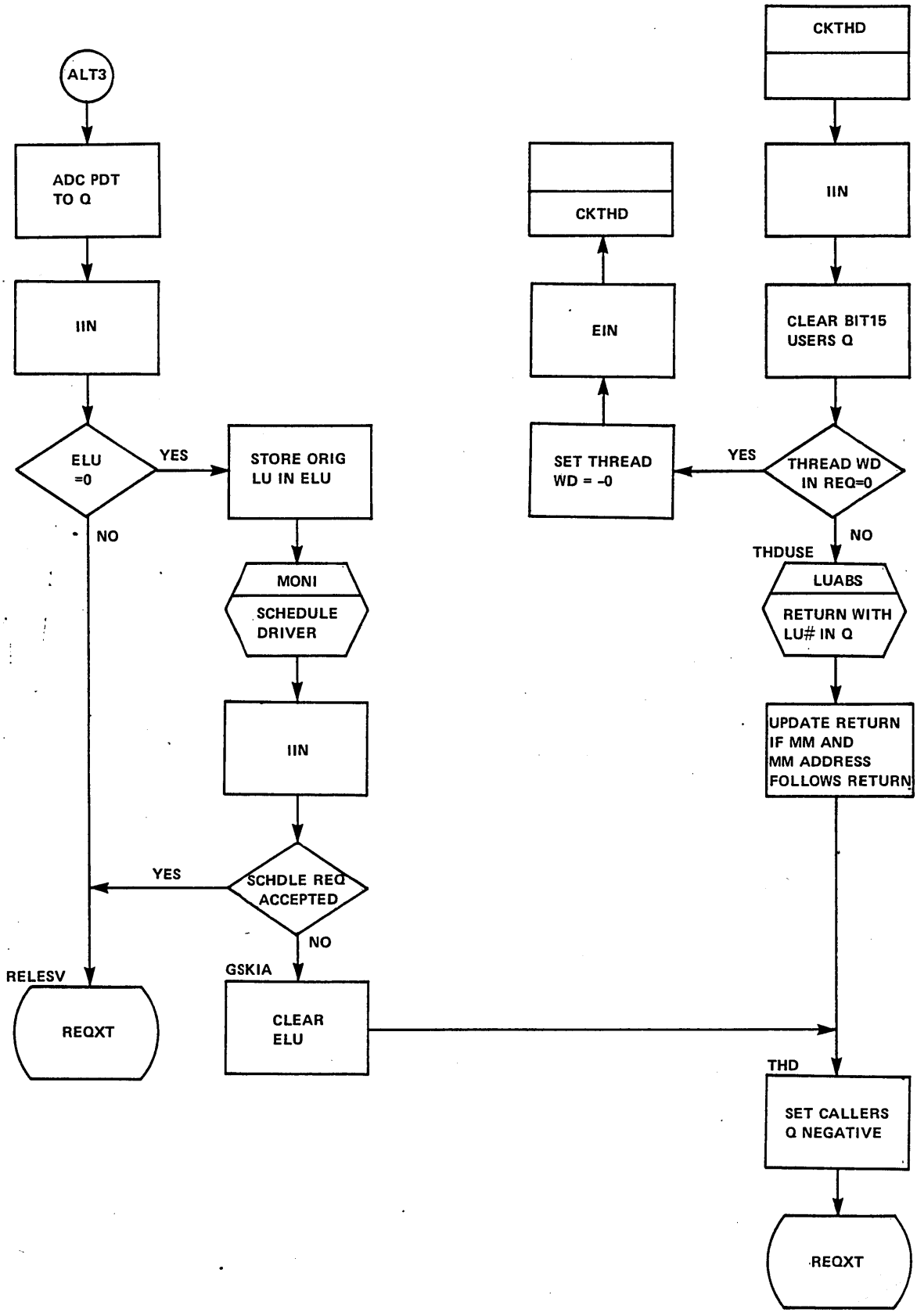
REQUEST CODE ZERO

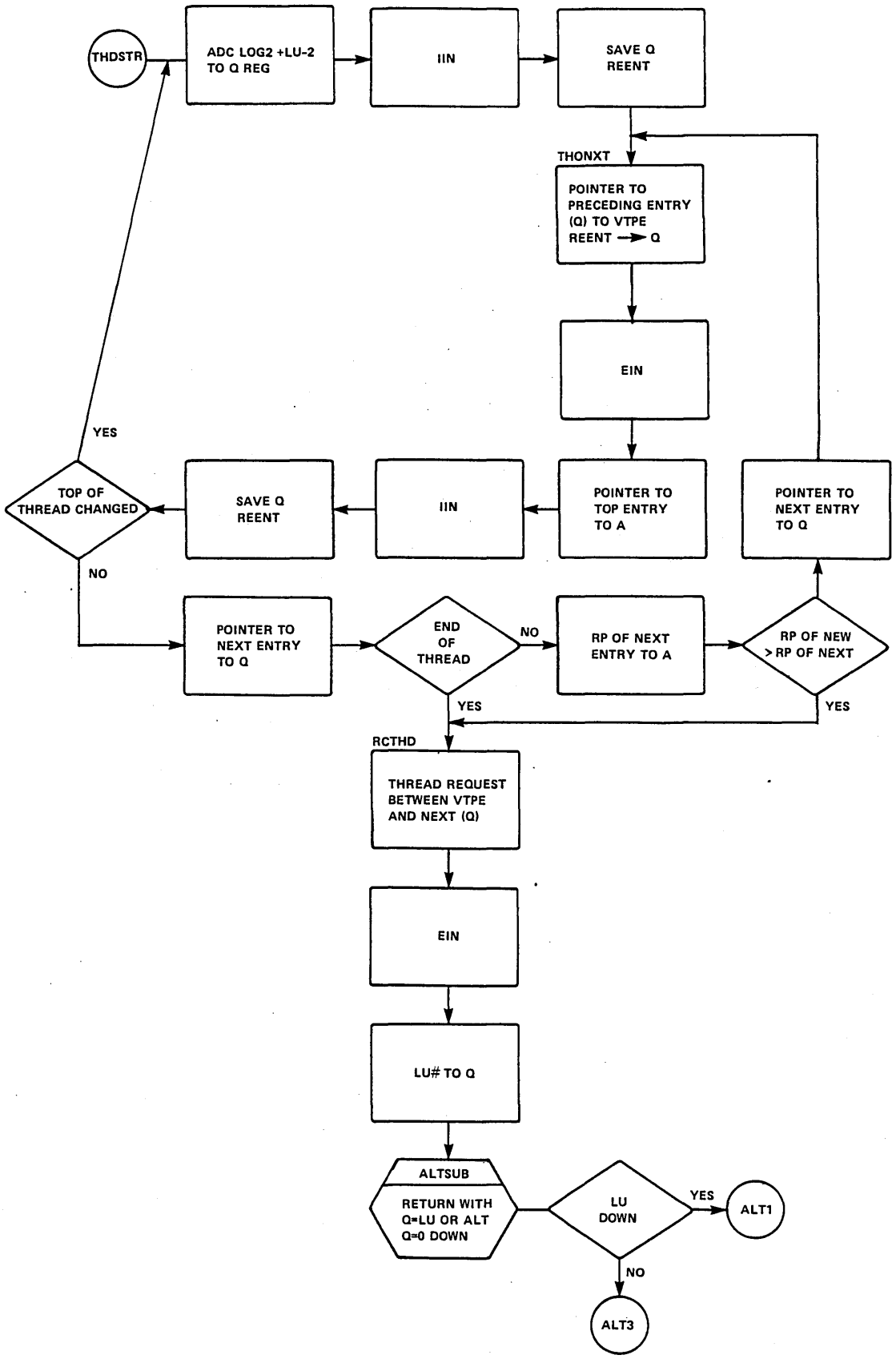
The zero request code is used to cause mass storage reads which result from SCHDLE requests. For example, if a mass storage resident program is scheduled, the SCHDLE request processor passes the system directory entry to the SPACE processor for allocation of space. The SPACE processor then passes the system directory entry to this processor to effect a transfer of the program from mass storage. The apparent request code carried in the system directory entry is zero.

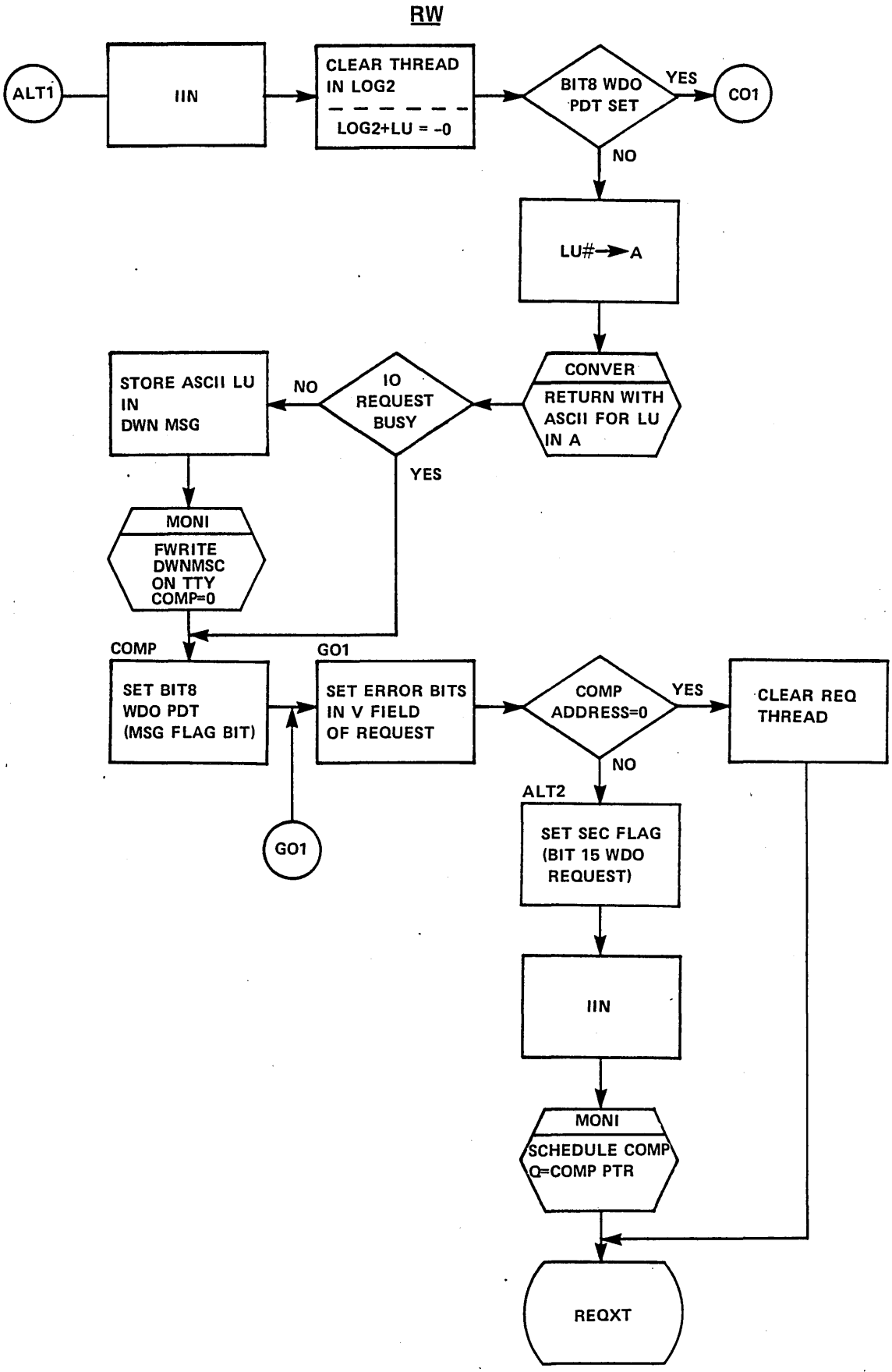

RW
CHECK THREAD WORD IN REQUEST
GET LU# from request
entry from Scheduler (WDISP)
SAVLU
MM DEVICE
YES
UPDATE RETN IF NECESSARY
NO
Threading system directs request to Core allocator
THREAD TO LOG2 TABLE
LU DOWN
YES
OUTPUT MSG LU X DOWN
SCHEDULE COMPLETION ADDRESS
with request rejected.
REQXT
DRIVER BUSY
YES
REQXT
NO
SCHEDULE DRIVER
REQXT

RW


T0,T1,T2,
T4,T6,RW
PARAMETER
ADDRESS
TO Q
INDIR
REQ
NO
UPDATE
RETURN
YES
R1
REQUEST PRIORITY
TO VPL,
RC TO A
EIN
DIRECTORY
NO
RWUSER
CKTHD
CHECK THREAD WD
ACCEPT REQ
IF ZERO
LIBRARY
LU# TO Q
LUABS
RETURN WITH
LU# IN Q
SAVLU
WD8 PDT
TO A
VID TO Q
LU# TO
VTMP
CLASS CODE
= MM CODE
YES
INDIRECT
REQUEST
NO
ADC PTRS
TO Q
D BIT
SET
YES
NO
YES
STARTING
ADDRESS
INDIRECT
NO
R2
UPDATE
RETURN ADDRESS
THDSTR


ALT3
ADC PDT TO Q
IIN
ELU =0
YES
NO
STORE ORIG LU IN ELU
MONI
SCHEDULE DRIVER
IIN
SCHDLE REQ ACCEPTED
YES
NO
RELESV
REQXT
GSKIA
CLEAR ELU
CKTHD
IIN
CLEAR BIT15 USERS Q
THREAD WD IN REQ=0
YES
NO
THDUSE
LUABS
RETURN WITH LU# IN Q
UPDATE RETURN IF MM AND MM ADDRESS FOLLOWS RETURN
THD
SET CALLERS Q NEGATIVE
REQXT
SET THREAD WD = -0
EIN
CKTHD


THDSTR
ADC LOG2 +LU-2 TO Q REG
IIN
SAVE Q REENT
THONXT
POINTER TO PRECEDING ENTRY (Q) TO VTPE REENT → Q
EIN
POINTER TO TOP ENTRY TO A
IIN
SAVE Q REENT
TOP OF THREAD CHANGED
YES
NO
POINTER TO NEXT ENTRY TO Q
POINTER TO NEXT ENTRY TO Q
END OF THREAD
NO
RP OF NEXT ENTRY TO A
RP OF NEW > RP OF NEXT
YES
YES
RCTHD
THREAD REQUEST BETWEEN VTPE AND NEXT (Q)
EIN
LU# TO Q
ALTSUB
RETURN WITH Q=LU OR ALT Q=0 DOWN
LU DOWN
YES
ALT1
NO
ALT3


RW
ALT1
IIN
CLEAR THREAD
IN LOG2
LOG2+LU = -0
BIT8 WDO
PDT SET
YES
CO1
NO
LU#→A
CONVER
RETURN WITH
ASCII FOR LU
IN A
IO
REQUEST
BUSY
NO
STORE ASCII LU
IN
DWN MSG
YES
MONI
FWRITE
DWNMSC
ON TTY
COMP=0
COMP
GO1
SET BIT8
WDO PDT
(MSG FLAG BIT)
SET ERROR BITS
IN V FIELD
OF REQUEST
COMP
ADDRESS=0
YES
CLEAR REQ
THREAD
NO
ALT2
GO1
SET SEC FLAG
(BIT 15 WDO
REQUEST)
IIN
MONI
SCHEDULE COMP
Q=COMP PTR
REQXT

STUDENT PROJECT-6

phyptab => word 12

1. From the dump, find the last status taken on the line printer, what does it tell you? $0019 = EOP, Ready for data, Ready.
2. Was there any I/O in progress when the dump was taken? check log 2 if not FFFF then I/O I&B active
3. What is the maximum logical unit number in the system? $1B
4. Are there any shared devices? Which ones are they? Lots will mus with this? Dummy, TTY, MAG TAPE, LINE printer
5. Are any devices marked down? No
   look for a 6 or if LU down message written sets bit 12 :. 6 or 7
6. What is the Alternate device for the comments device? 13
7. Is the driver for the line printer core resident in this system? If not, was it in core when the dump was taken? phystab for line printer => lenght word = 0 then not in mm
8. What is the address for initiator portion of the system disk driver? phyptab. $9417

try to check MMEXEC Program.

STUDY QUESTIONS - 6

1. **How can you tell if a driver is busy?** ELU Logical unit word. also tells what LU it is processing for that req. Set by RW processor.
2. **How does a driver get put into execution?** SCHEDULED BY RW request processor
3. **What is the function of the LOG1 table?** LISTS ALTERnates for a device. Tells IF LU down/up/SHARED.
4. **How are requests ordered in the LOG2 thread?** REQUEST PRIORITY [WITHIN RP FIFO]
5. **What happens if a driver is working on a priority 0 request and a request of priority 10 is put in the LOG 2 thread?** DRIVER FINISHES REQUEST at priority 0 then goes to work on priority 10 REQUEST.
6. **How can a driver be busy if he can be executing and your program can be making requests to the system?** WHILE driver is active it may wait for data transfer to occur. Driver may go to DISPATCHER TO allow OTHER programs TO run.
7. **How does a driver know when the operation is complete?** IT GETS END OF OPERATION INTERRUPT (FLAG)
8. **At what prioirity does the RW processor run?** AT SAME PRIORITY AS your PROGRAM
9. **How many times is MONI entered due to a RW request and why?** 1
10. **Does RW set any words in the PHYSTB? If so, under what conditions?** SETS ELU TO indicate a request IS ACTIVE on that device
11. **If RW does not schedule the driver, how will the driver ever get into execution to find our request on LOG2?** Another request to the particular device will try again to schedule the driver. You may/may not loose original request
12. **How does the RW find the driver's address so that he can schedule it?** LOG1A => PHYSTAB word 1 which contains driver initiator address.

When driver is busy can never insert anything on top of thread because top of thread is request being handled.

1). CALLS monitor only 1ce from 3 different places.
1) Schedule the drive
2) Schedule completion Address of program sched if request gets aborted
3). ~~Print~~ Lu down message. schedule LU down message

(Allocatable (#PRILVLS) (16))

Mass Memory driver has address of executive who takes care of scheduling the driver.

LESSON GUIDE 7

DRIVERS

LESSON PREVIEW:

The lesson will introduce the general structure of a driver under MSOS. The class will study the subroutines provided by the system for all drivers and will examine an example driver.

REFERENCES:

Software Peripheral Drivers RM Chapter 1 & 2/MSOS 5 pp. 2-8 and 2-9, Appendix C Listing of a driver

TRAINING AIDS:

PROJECTS:

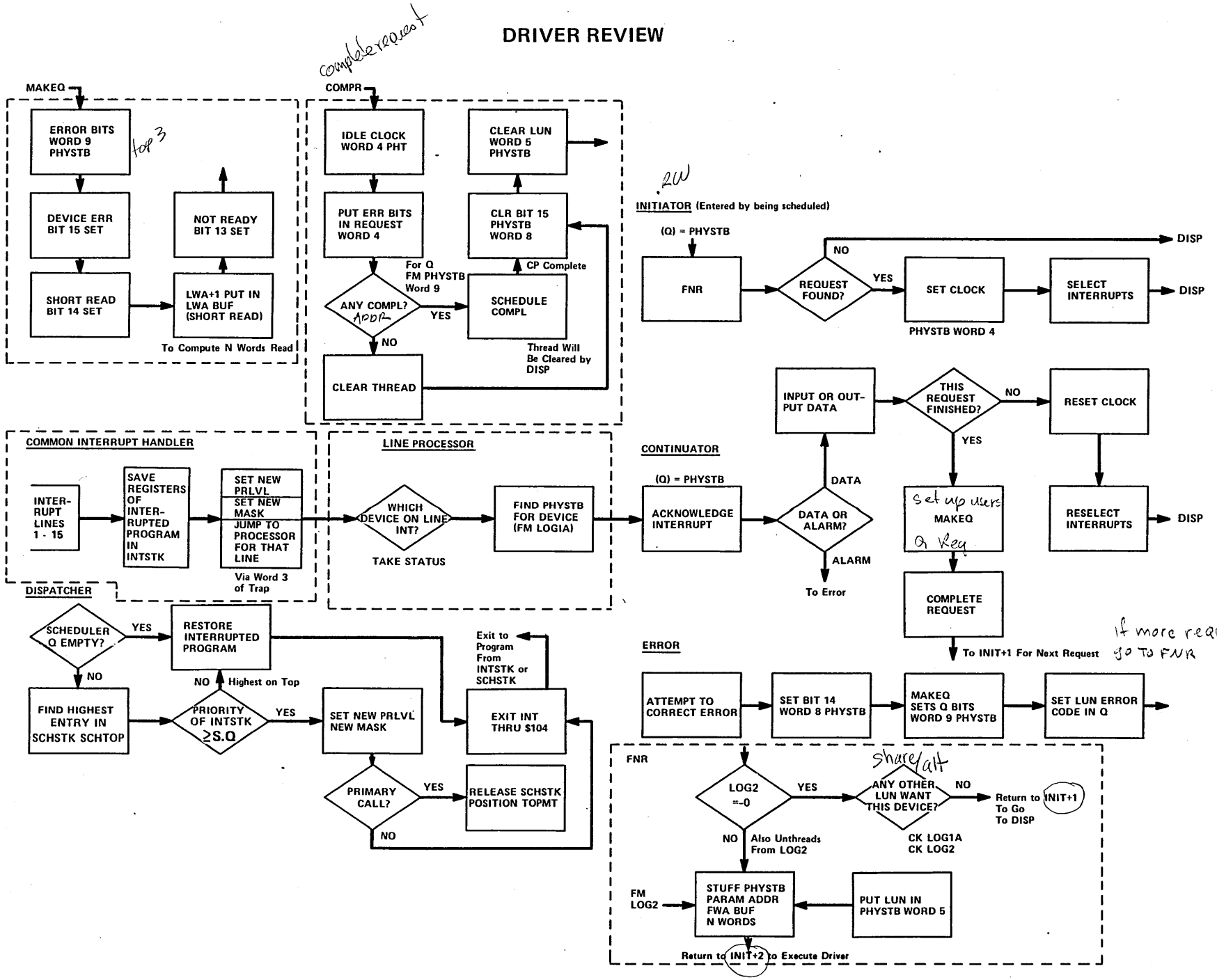
Study Questions-7

OBJECTIVES:

At the completion of this lesson, the student will be able to:

1. Discuss in detail the structure of a driver under MSOS.
2. Discuss in detail the functions of FNR, COMPRQ, MAKEQ, ALTDEV.
3. Trace the flow of events as a result of an I/O request.

DRIVER REVIEW



DRIVER INITIATOR

(ENTERED FROM SCHEDULE REQUEST BY RW)

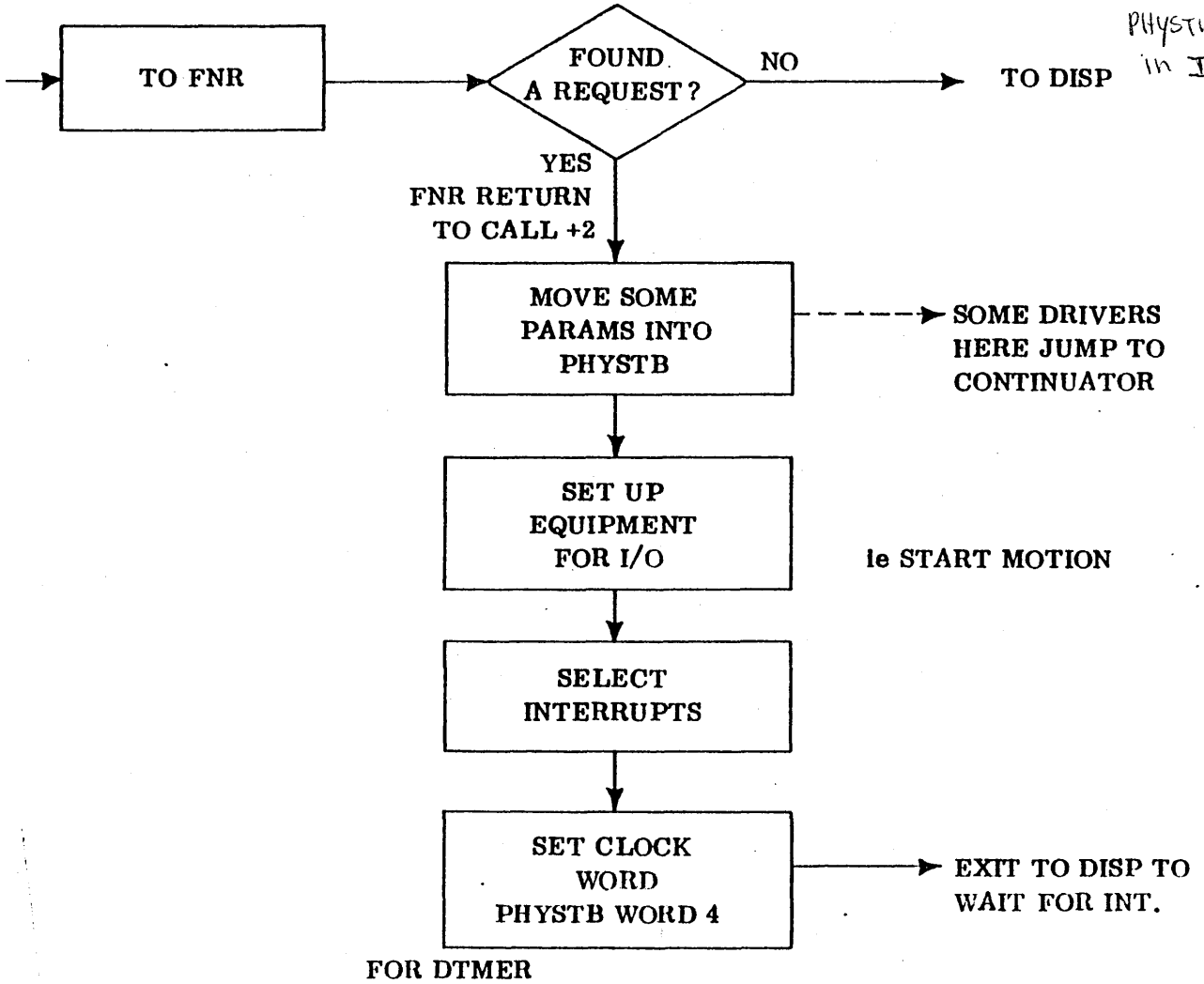
when FNR is being called from.

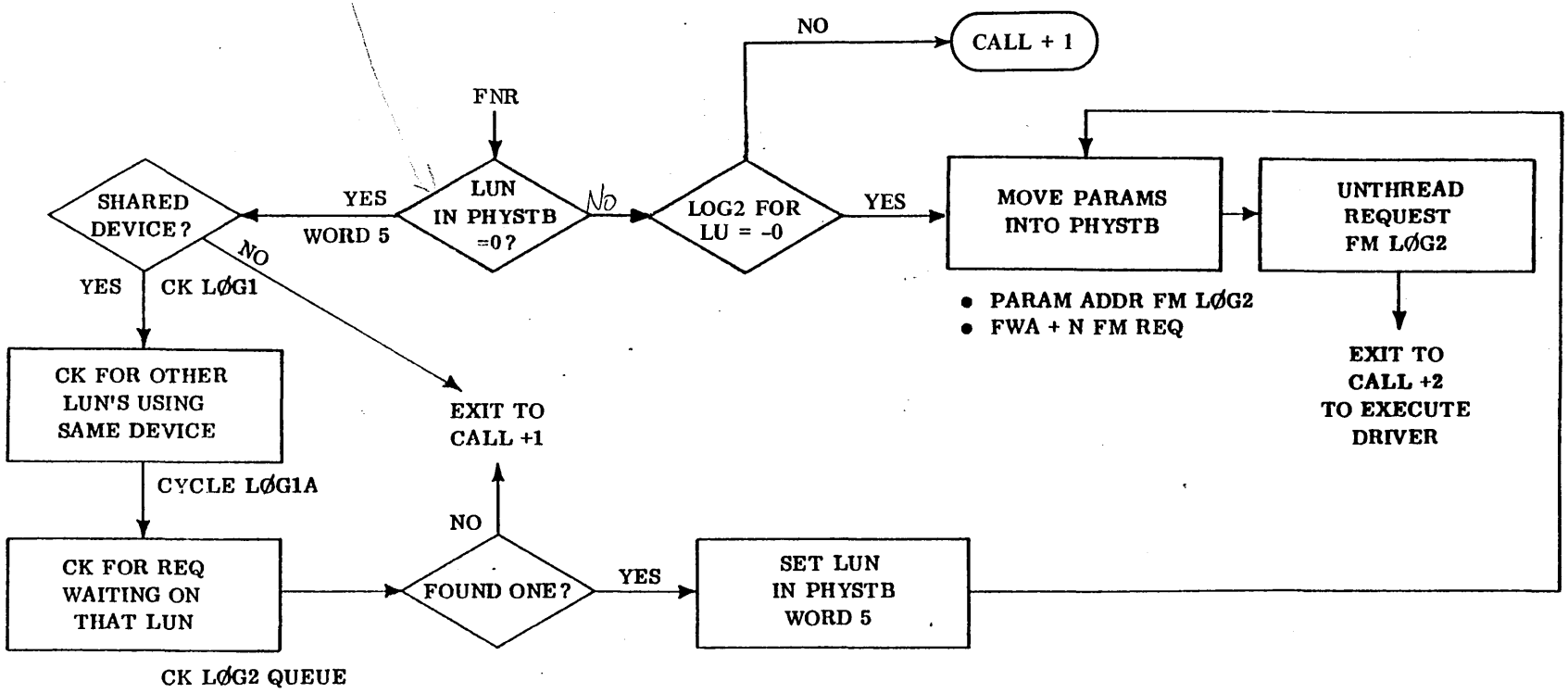
(CALL)

STQ - I (Q) = PHYSTB ADDR
RTJ - (AFNR)
JMP - (ADISP)

first part core res. driver (almost) always looks like this

most drivers assume PHYSTAB ADDRESS is in I




FNR
LUN IN PHYSTB =0?
YES
WORD 5
No
LOG2 FOR LU = -0
NO
CALL + 1
YES
MOVE PARAMS INTO PHYSTB
UNTHREAD REQUEST FM LØG2
• PARAM ADDR FM LØG2
• FWA + N FM REQ
EXIT TO CALL +2 TO EXECUTE DRIVER
SHARED DEVICE?
YES
CK LØG1
NO
EXIT TO CALL +1
CK FOR OTHER LUN'S USING SAME DEVICE
CYCLE LØG1A
CK FOR REQ WAITING ON THAT LUN
CK LØG2 QUEUE
NO
FOUND ONE?
YES
SET LUN IN PHYSTB WORD 5

FIND NEXT REQUEST FOR DRIVER (FNR)

FUNCTION

The function of this subroutine is to find the request which should be processed next by a driver for a device. It performs as much of the Physical Device Table set-up associated with each new request (or part of a request) as is common to all I/O drivers.

ENTRY INTERFACES

Entered via a return jump to entry point FNR with the physical device table slot address in I.

EXIT INTERFACES

If there are no more requests for action by a device, the subroutine returns to the driver at the location following the Return Jump which called the subroutine. call+1

If more action is required, the subroutine returns to the driver at the second location past the Return Jump with the following conditions: call+2

The I-register is undisturbed.

The A, Q, and Overflow registers are not restored.

The physical device table slot is set with the information specified in the description of the table in the ERS. ?

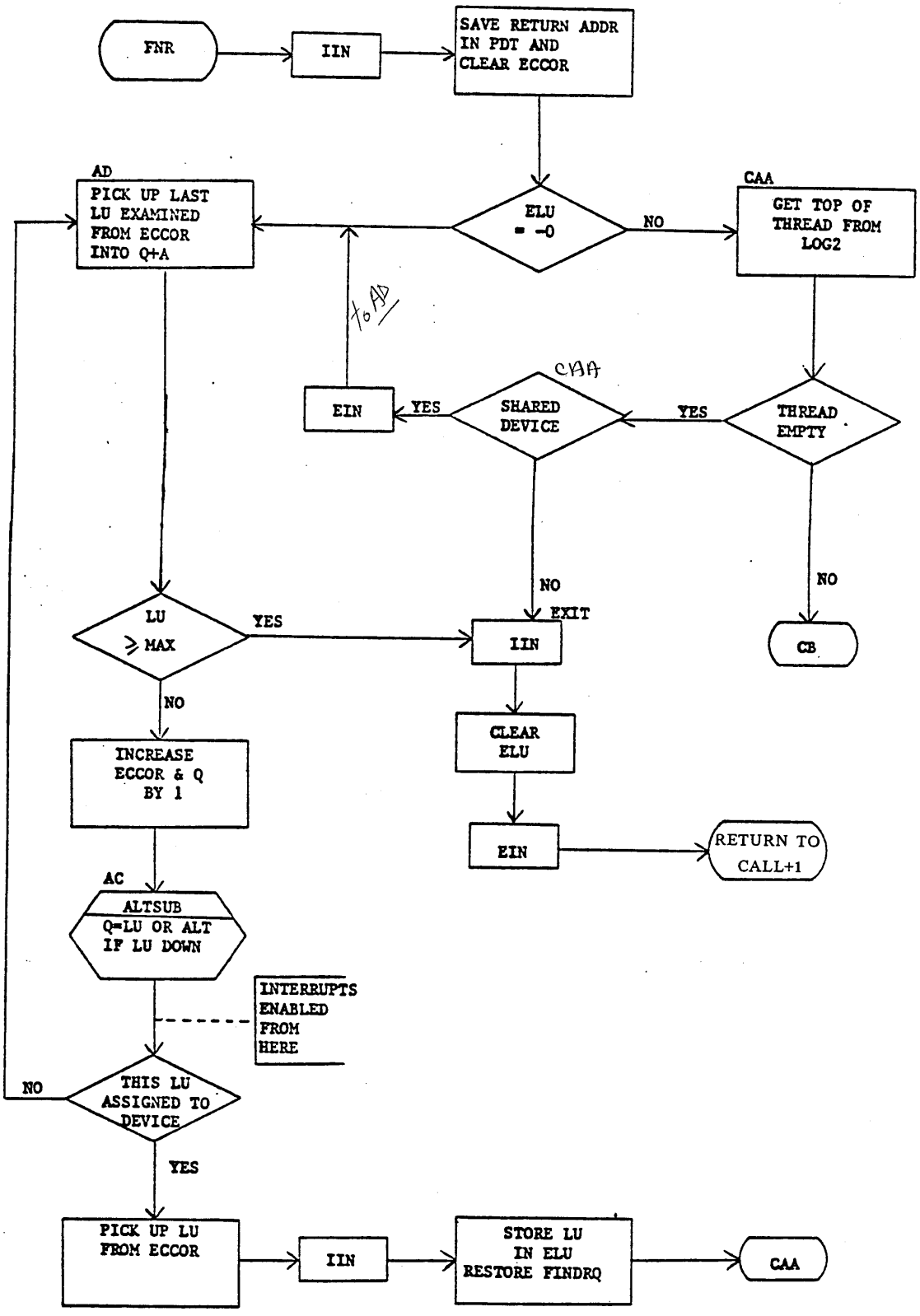
INTERNAL DESCRIPTION

The top request on the logical unit thread is removed from the thread and its parameter list address and absolutized I/O List first and last +1 addresses are placed in the assigned physical device table. Program control is then returned to the driver. At some later time, when the driver has completed the last I/O action required by that request and has received an interrupt (if applicable) indicating completion of the last action, the driver calls the Complete Request for Driver subroutine, thereby completing the processing of the request.

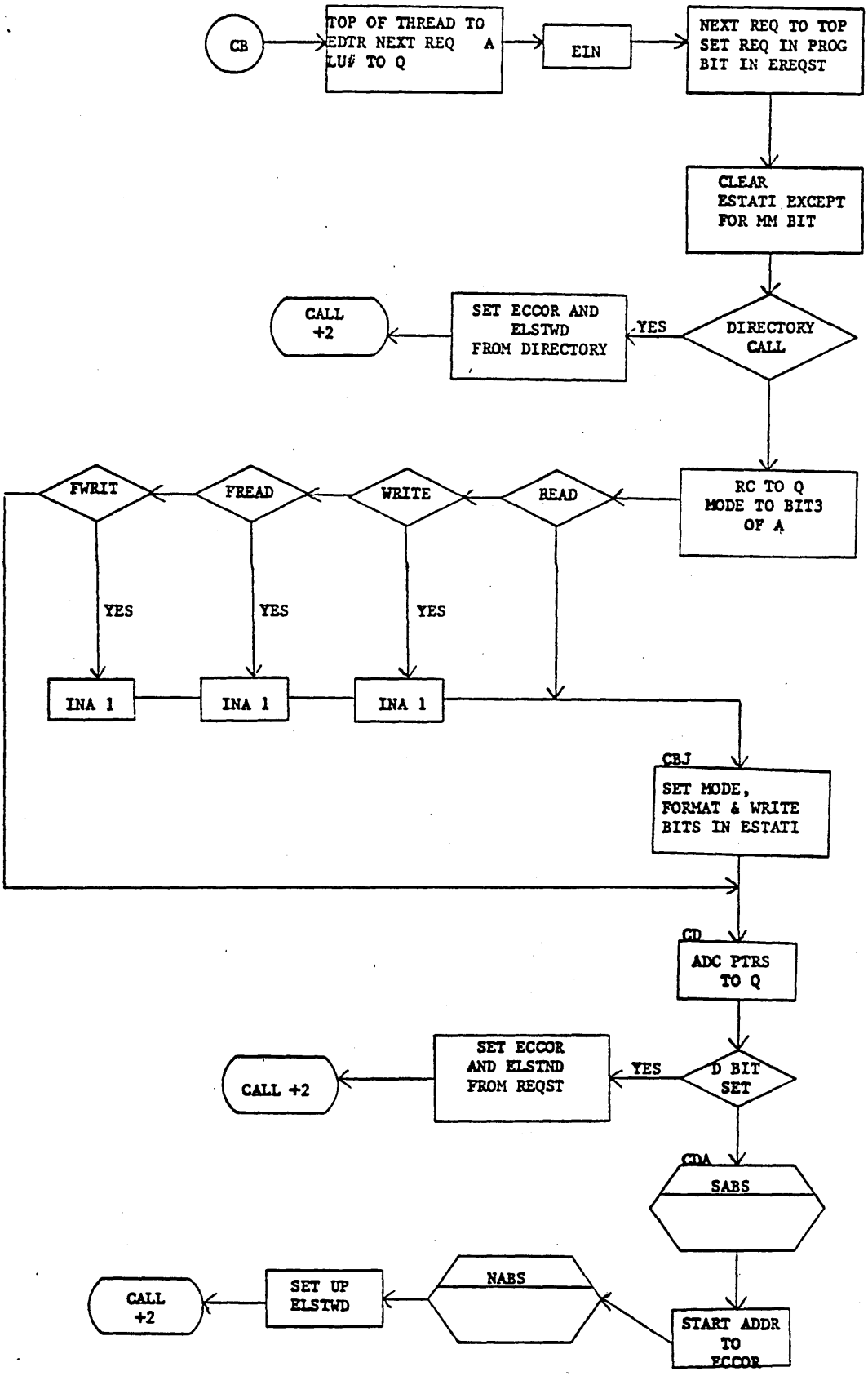
If the device is shared by several logical units, the Complete Request for Driver subroutine sets the logical unit word in the physical device table to FFFF. Upon finding that a device is assigned to the logical unit $FFFF_{16}$ the Find Next Request for Driver subroutine searches the Logical Unit Table for the highest priority (i.e., lowest number) Logical Unit which requires the available device.

This provides sharing of devices by several user routines. However, no request, once started, is interrupted; only upon completion of each request is a higher priority requirement executed.

FNR


FNR
IIN
SAVE RETURN ADDR IN PDT AND CLEAR ECCOR
AD
PICK UP LAST LU EXAMINED FROM ECCOR INTO Q+A
ELU = -0
NO
CAA
GET TOP OF THREAD FROM LOG2
to AD
EIN
YES
SHARED DEVICE
CAA
YES
THREAD EMPTY
NO
EXIT
NO
LU ≥ MAX
YES
IIN
CB
NO
INCREASE ECCOR & Q BY 1
CLEAR ELU
EIN
RETURN TO CALL+1
AC
ALTSUB
Q=LU OR ALT IF LU DOWN
INTERRUPTS ENABLED FROM HERE
NO
THIS LU ASSIGNED TO DEVICE
YES
PICK UP LU FROM ECCOR
IIN
STORE LU IN ELU RESTORE FINDRQ
CAA

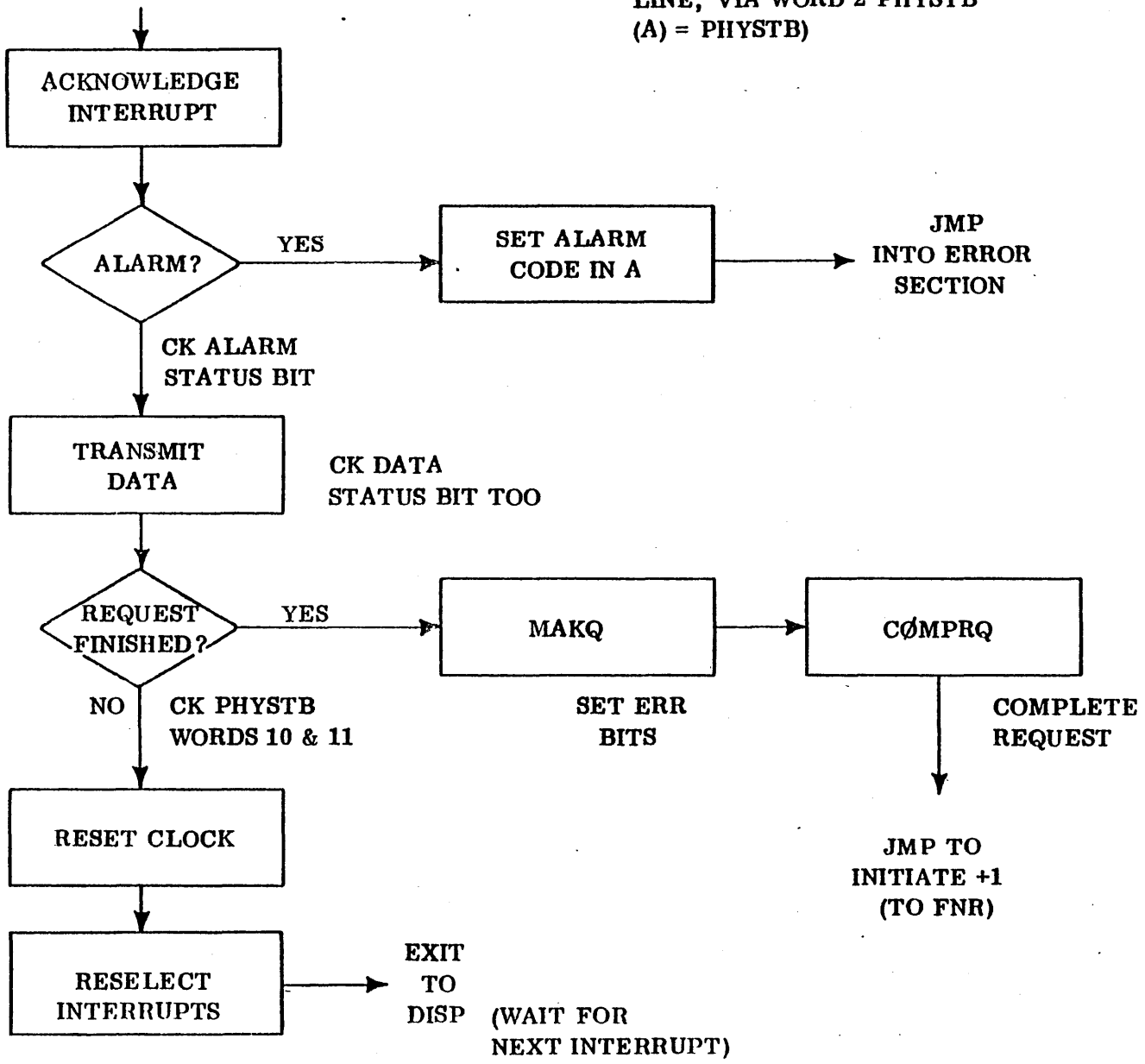
FNR


CB
TOP OF THREAD TO EDTR NEXT REQ A LU# TO Q
EIN
NEXT REQ TO TOP SET REQ IN PROG BIT IN EREQST
CLEAR ESTATI EXCEPT FOR MM BIT
DIRECTORY CALL
YES
SET ECCOR AND ELSTWD FROM DIRECTORY
CALL +2
RC TO Q MODE TO BIT3 OF A
READ
WRITE
FREAD
FWRIT
YES
YES
YES
INA 1
INA 1
INA 1
CBJ
SET MODE, FORMAT & WRITE BITS IN ESTATI
CD
ADC PTRS TO Q
D BIT SET
YES
SET ECCOR AND ELSTND FROM REQST
CALL +2
CDA
SABS
START ADDR TO ECCOR
NABS
SET UP ELSTWD
CALL +2

acknowledges int's
only brought into execution
because of an interrupt.

CONTINUATOR

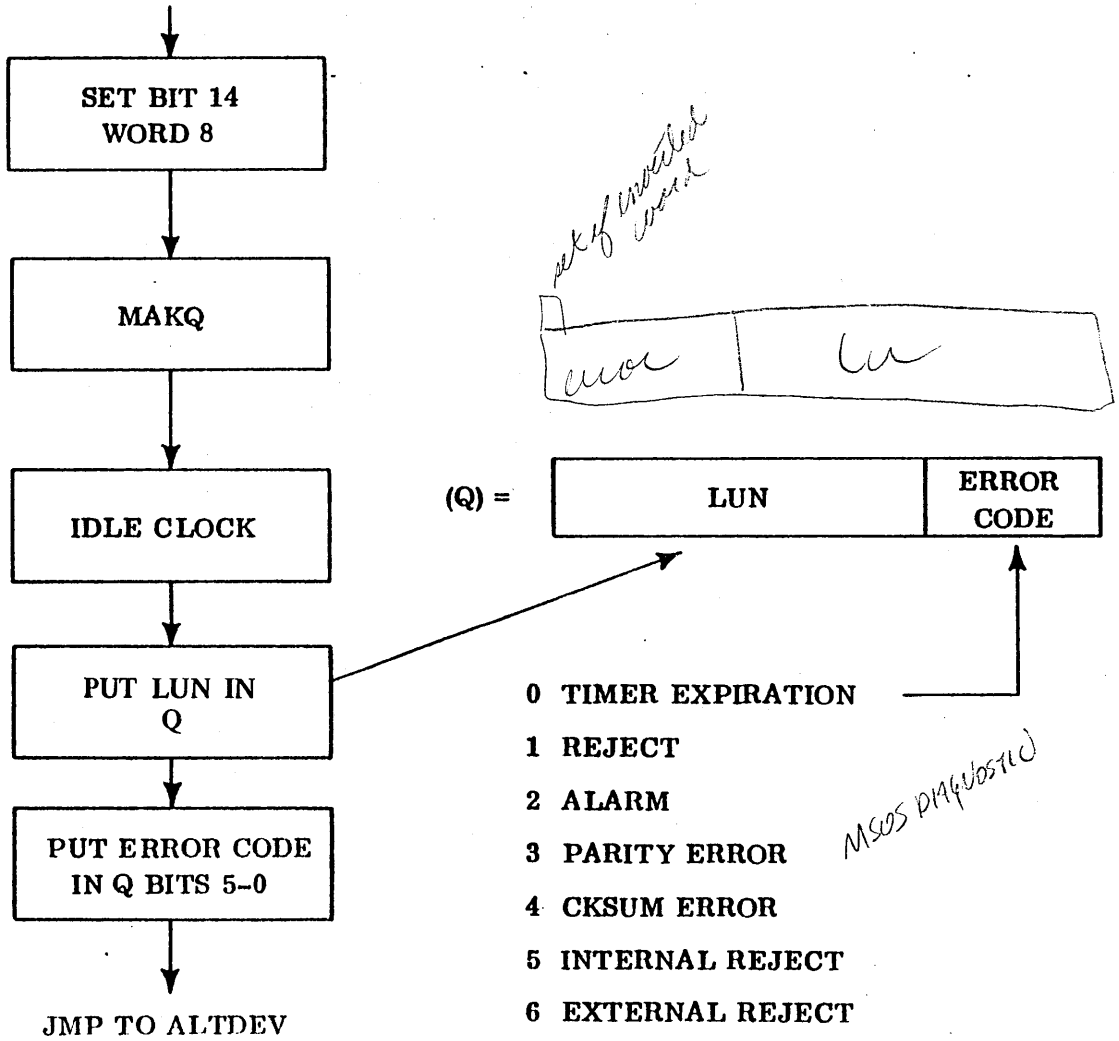
(ENTERED FROM INT. PROC. FOR LINE, VIA WORD 2 PHYSTB
(A) = PHYSTB)

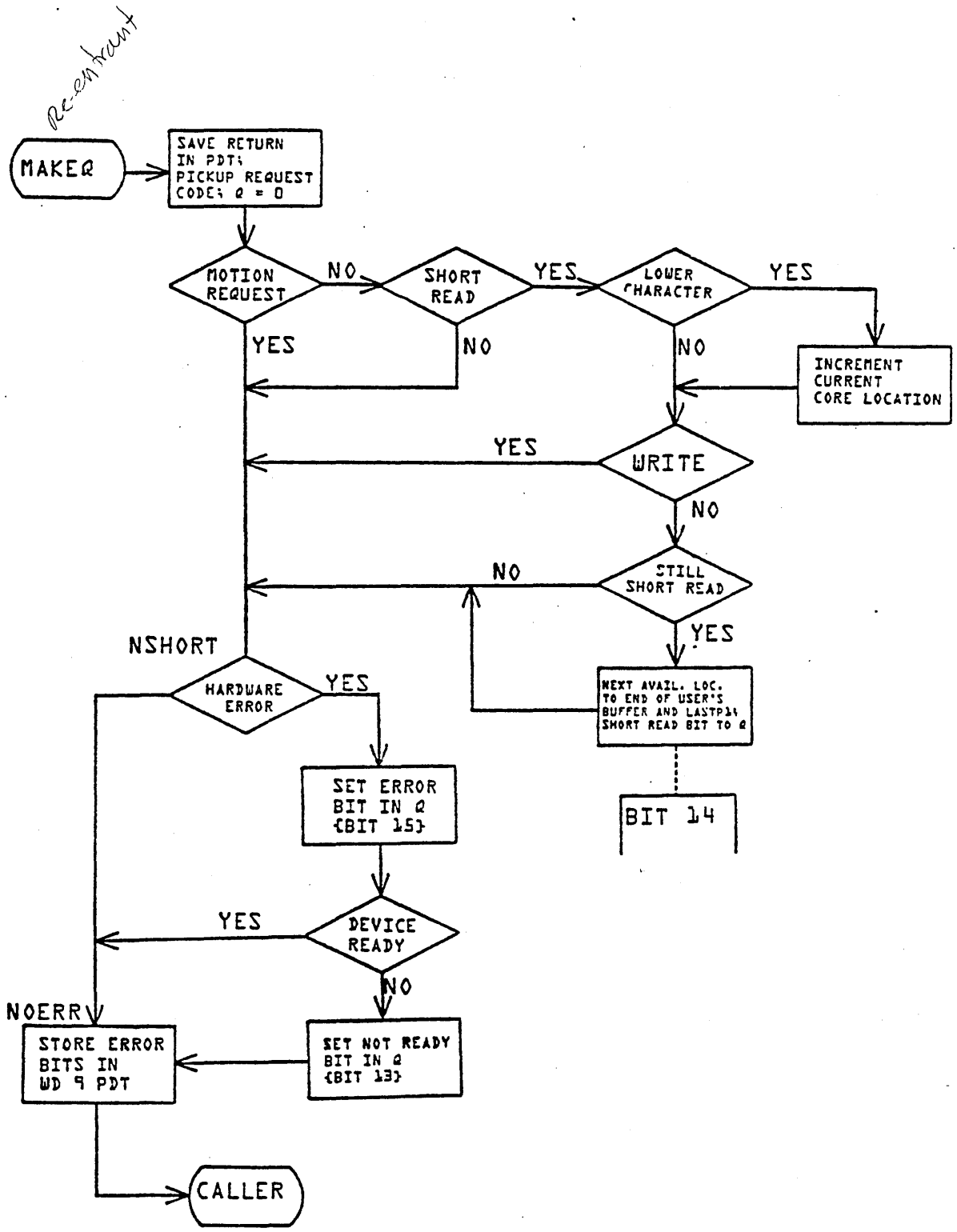


ERROR SECTION

ENTERED FROM CONTINUATOR OR -

DTMER (VIA WORD 3 PHYSTB)




Reentrant
MAKEQ
SAVE RETURN IN PDT; PICKUP REQUEST CODE; Q = 0
MOTION REQUEST
NO
SHORT READ
YES
LOWER CHARACTER
YES
YES
NO
NO
INCREMENT CURRENT CORE LOCATION
YES
WRITE
NO
NO
STILL SHORT READ
YES
NSHORT
HARDWARE ERROR
YES
NEXT AVAIL. LOC. TO END OF USER'S BUFFER AND LASTP1; SHORT READ BIT TO Q
BIT 14
SET ERROR BIT IN Q (BIT 15)
YES
DEVICE READY
NO
NOERR
STORE ERROR BITS IN WD 9 PDT
SET NOT READY BIT IN Q (BIT 13)
CALLER

COMPLETE REQUEST FOR DRIVER ROUTINE

EXTERNAL SYMBOLS

COMPRQ Entry point

FUNCTION

The functions of this subroutine are to initiate completion requests to the Scheduler for threaded I/O requests and to perform other housekeeping details upon completion of an I/O action by an I/O device driver.

ENTRY INTERFACES

COMPRQ is entered via a return jump with the physical device table address for the device in I.

EXIT INTERFACES

The contents of the I register are not disturbed. The contents of the A, Q, and Overflow registers are destroyed. Interrupts are enabled.

INTERNAL DESCRIPTION

The routine is entered from an I/O device driver via a Return Jump to COMPRQ. Interrupts are immediately inhibited.

The Diagnostic Clock cell in the Physical Device Table is set idle.

For Logical Units which do not share devices, the completion address, if not zero, is scheduled with the error field from the Physical Device Table replacing the V field of the I/O request parameter list. The request parameter list, which contains a request code designating it an I/O call, is flagged as a secondary scheduler call by setting bit 15 of the first word (field I) to "one". The scheduler later resets it to "zero". The device is not released from its logical unit assignment.

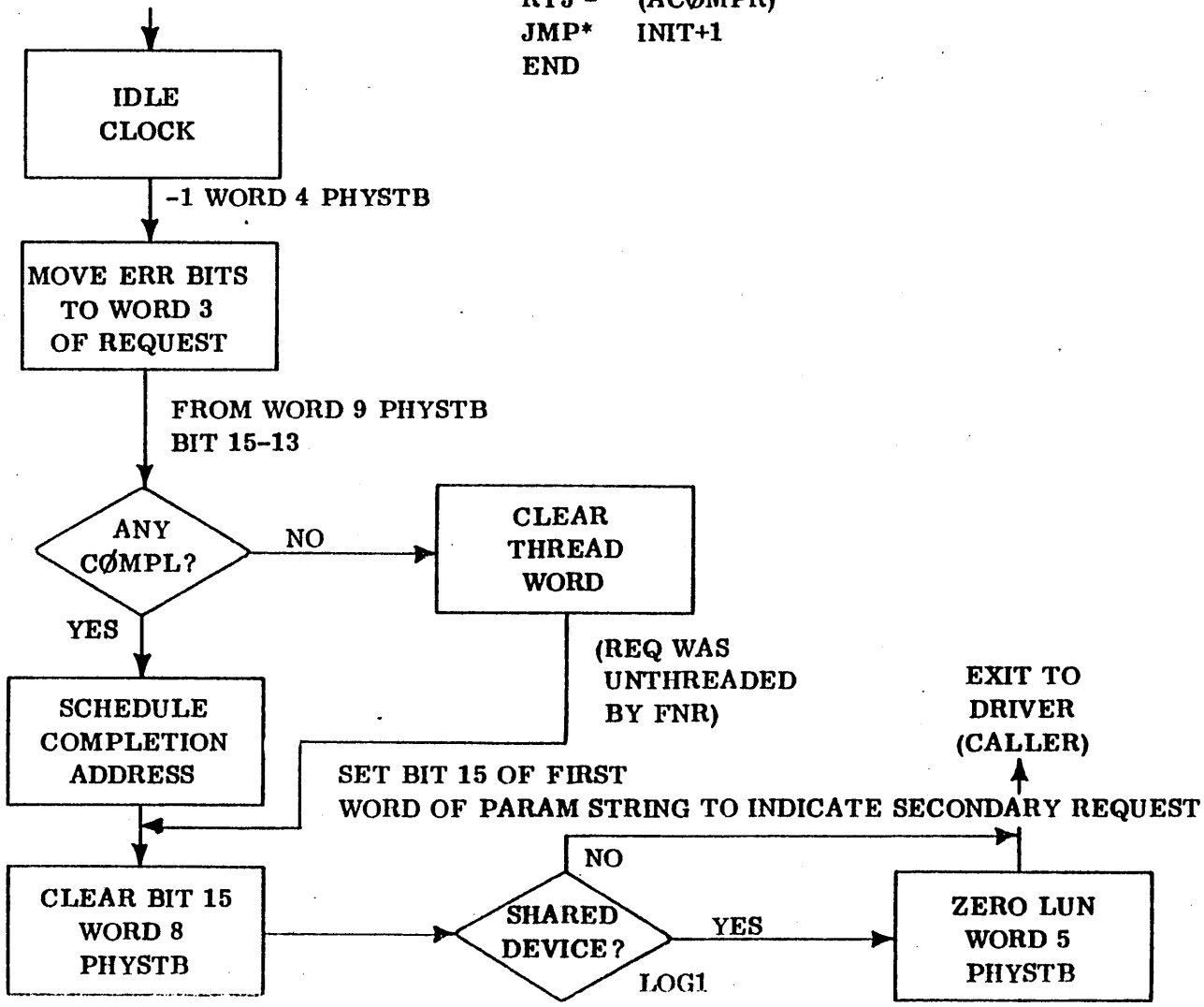
For Logical Units which share devices, completed threaded requests are treated like requests to ordinary Logical Units. The device is then assigned to a pseudo Logical Unit, $FFFF_{16}$ assignment.

The subroutine exits to the location following the Return Jump which called it.

COMPLETE REQUEST

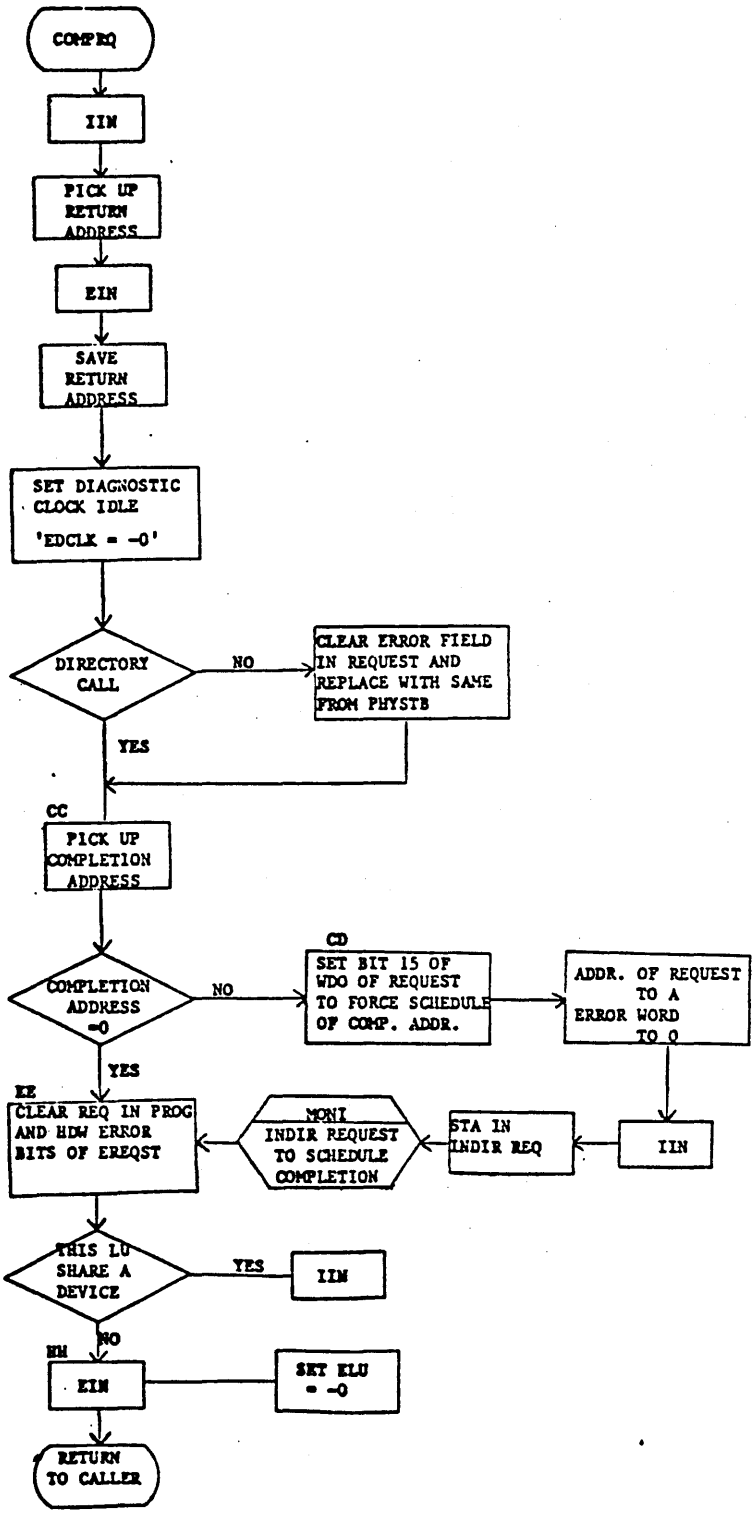
SUBROUTINE USED BY ALL DRIVERS TO COMPLETE REQUEST

```
RTJ -   (ACØMPR)
JMP*    INIT+1
END
```

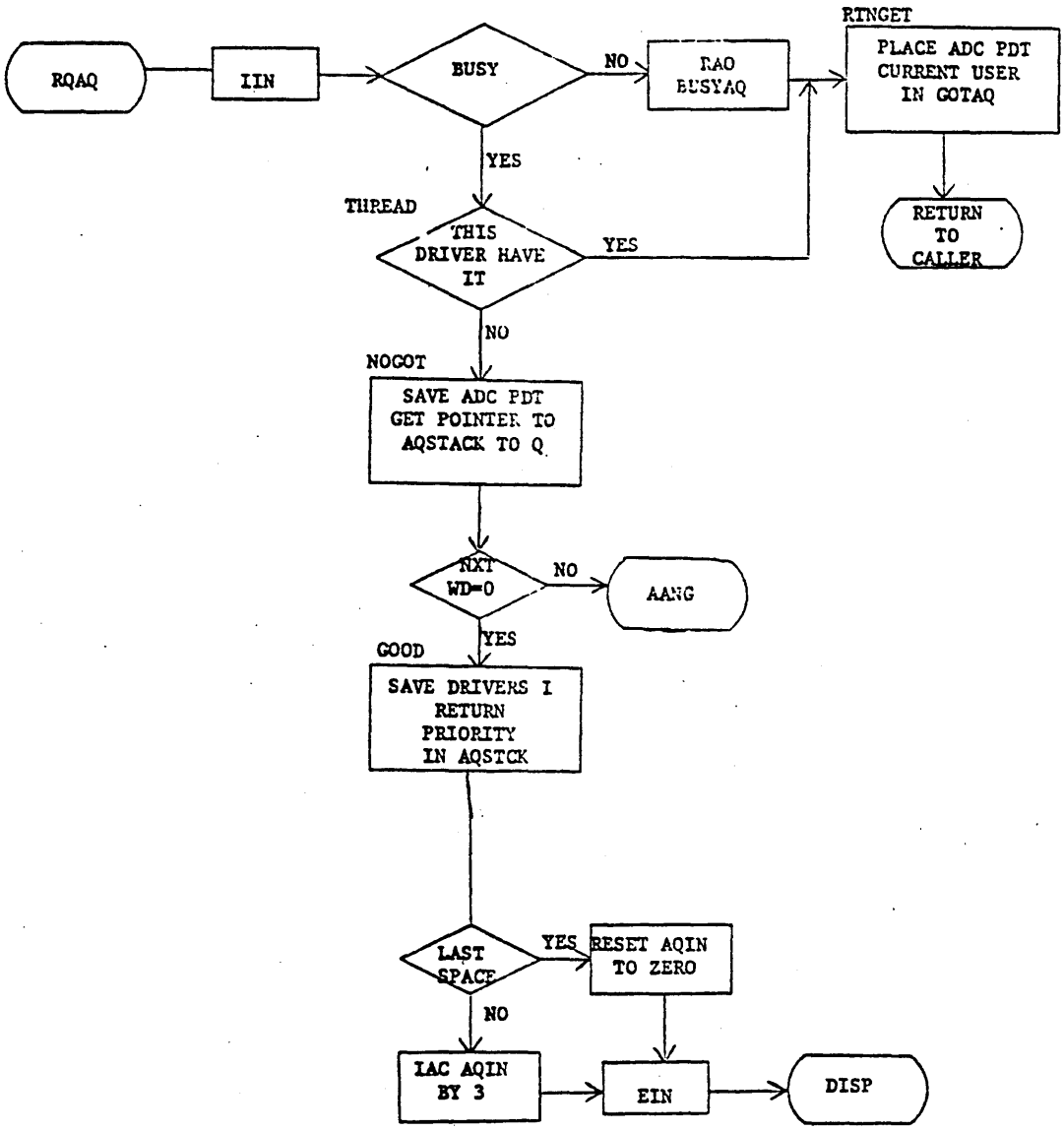


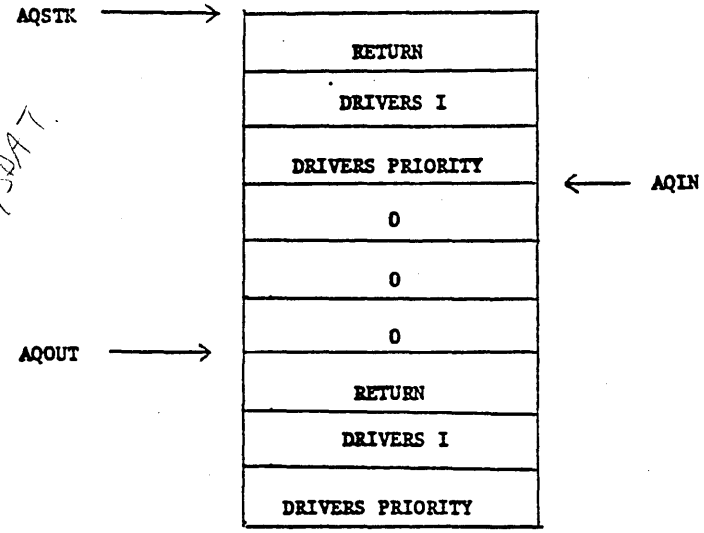
OP. COMPLETE BIT

COMPLETE REQUEST

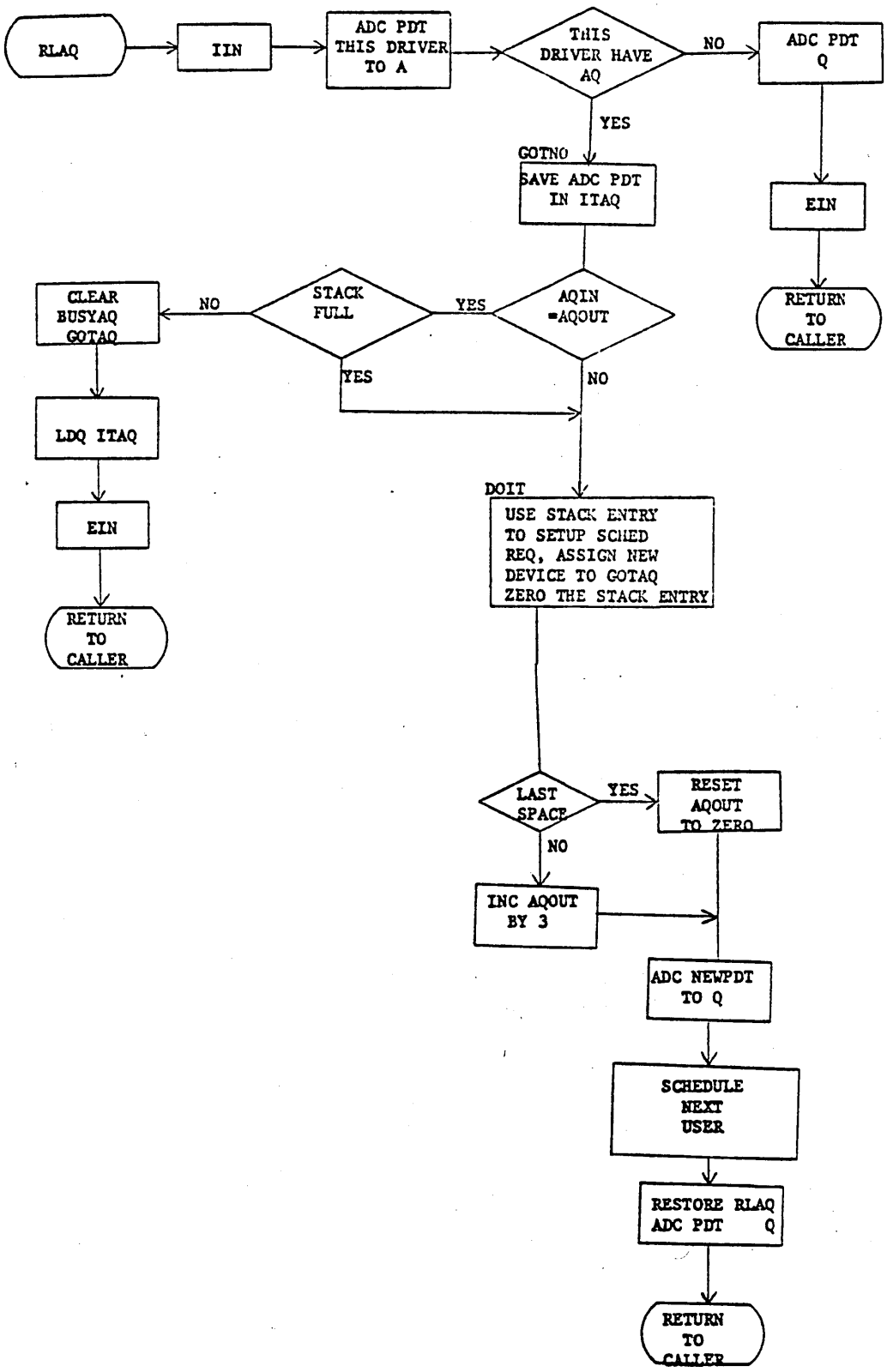


ALAQ–1

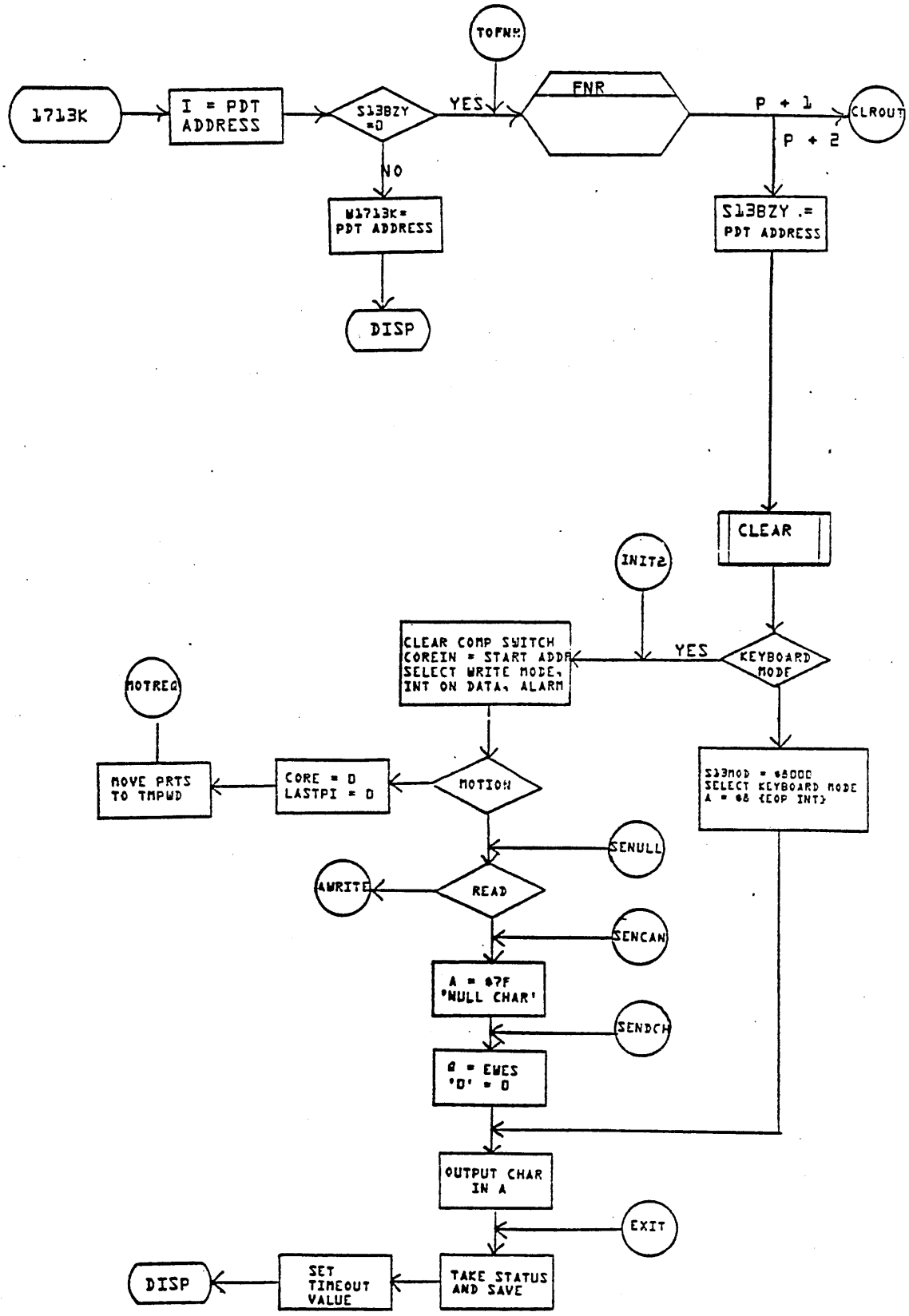

RQAQ
IIN
BUSY
NO
RAO BUSYAQ
RTNGET
PLACE ADC PDT CURRENT USER IN GOTAQ
YES
THREAD
THIS DRIVER HAVE IT
YES
RETURN TO CALLER
NO
NOGOT
SAVE ADC PDT GET POINTER TO AQSTACK TO Q
NXT WD=0
NO
AANG
YES
GOOD
SAVE DRIVERS I RETURN PRIORITY IN AQSTCK
LAST SPACE
YES
RESET AQIN TO ZERO
NO
IAC AQIN BY 3
EIN
DISP


AQSTK
RETURN
DRIVERS I
DRIVERS PRIORITY
AQIN
0
0
0
AQOUT
RETURN
DRIVERS I
DRIVERS PRIORITY

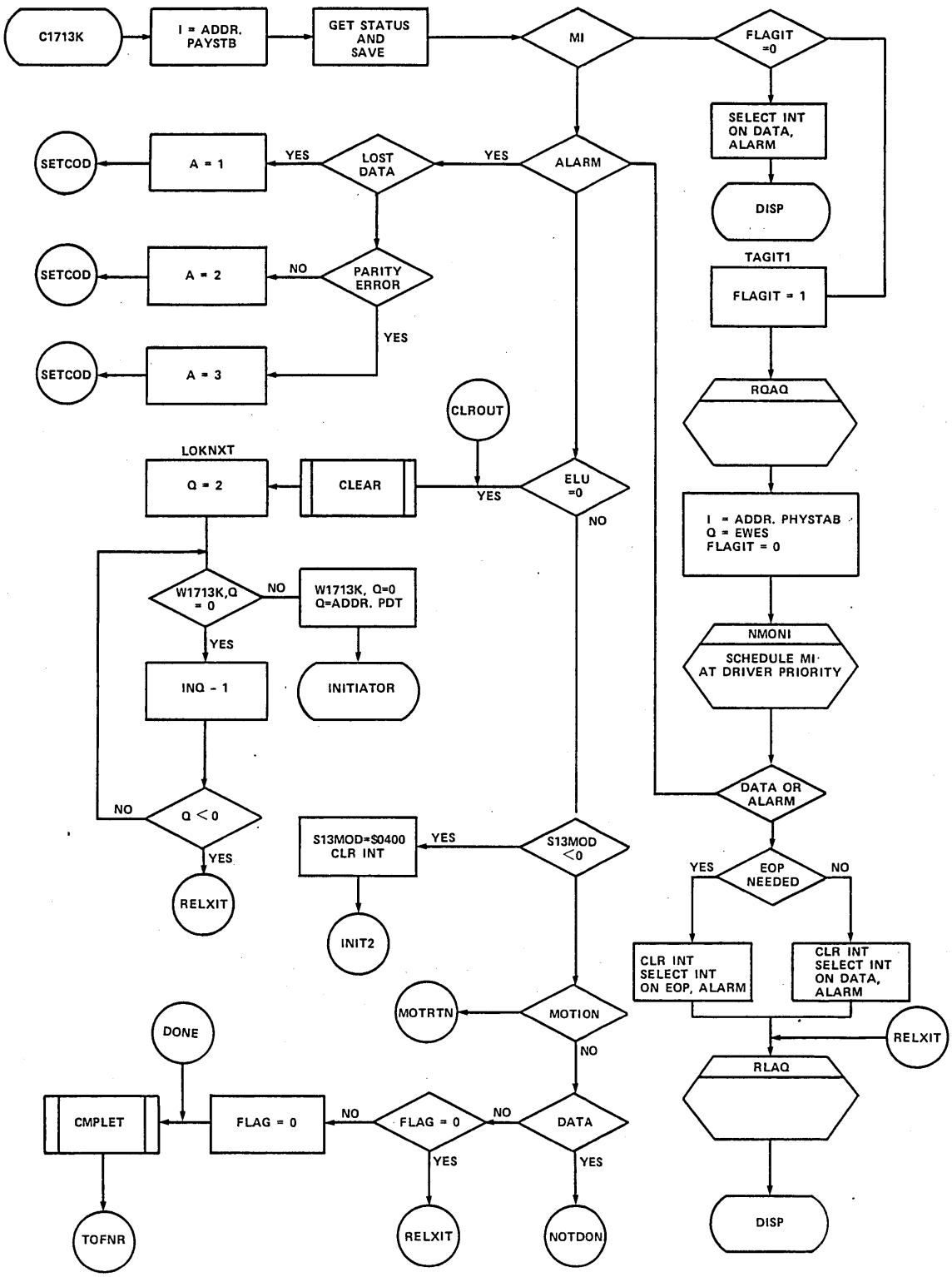
ALAQ–2

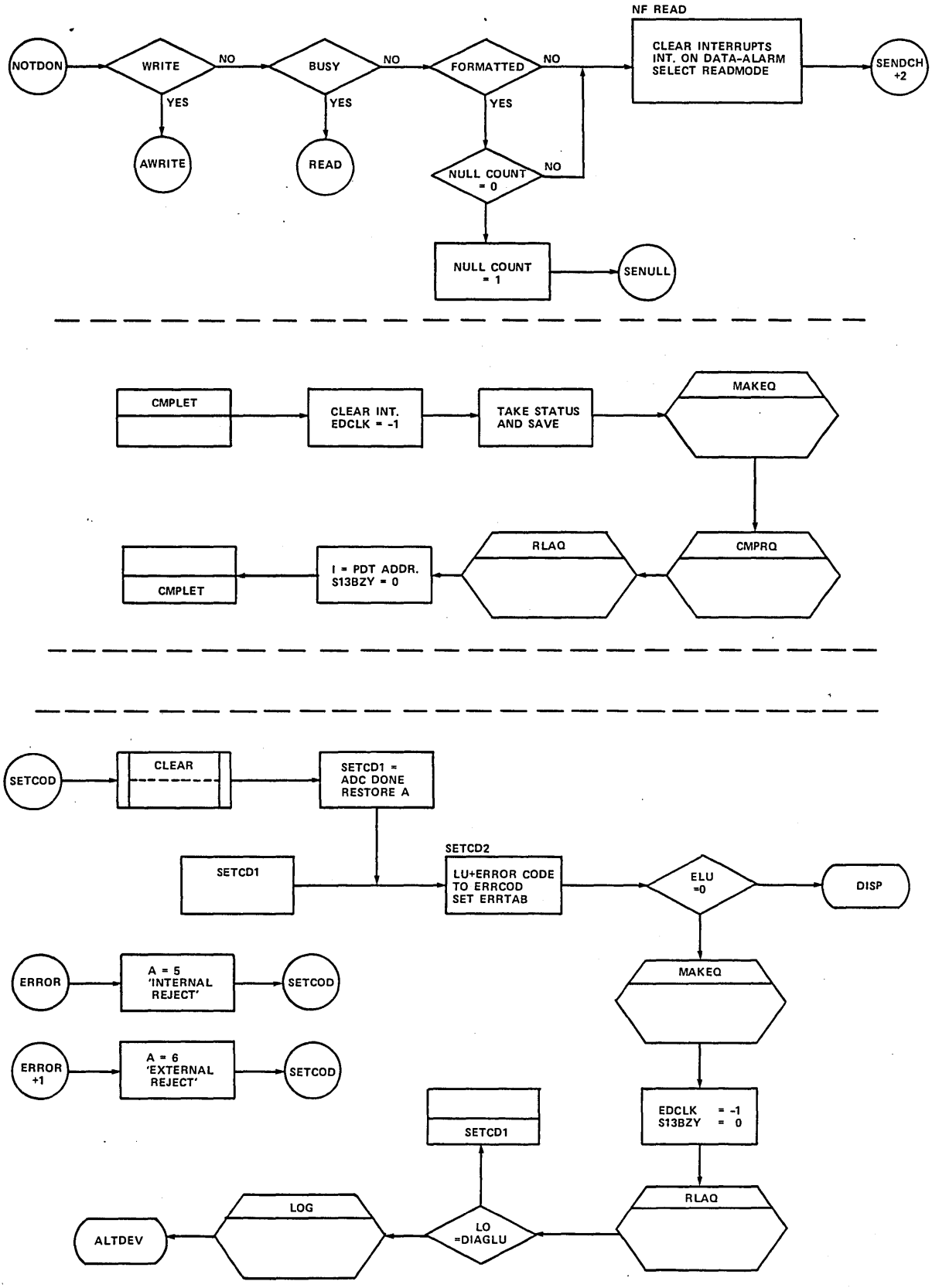

RLAQ
IIN
ADC PDT THIS DRIVER TO A
THIS DRIVER HAVE AQ
NO
ADC PDT Q
YES
GOTNO
SAVE ADC PDT IN ITAQ
EIN
RETURN TO CALLER
AQIN =AQOUT
STACK FULL
NO
YES
CLEAR BUSYAQ GOTAQ
LDQ ITAQ
EIN
RETURN TO CALLER
YES
NO
DOIT
USE STACK ENTRY TO SETUP SCHED REQ, ASSIGN NEW DEVICE TO GOTAQ ZERO THE STACK ENTRY
LAST SPACE
YES
RESET AQOUT TO ZERO
NO
INC AQOUT BY 3
ADC NEWPDT TO Q
SCHEDULE NEXT USER
RESTORE RLAQ ADC PDT Q
RETURN TO CALLER

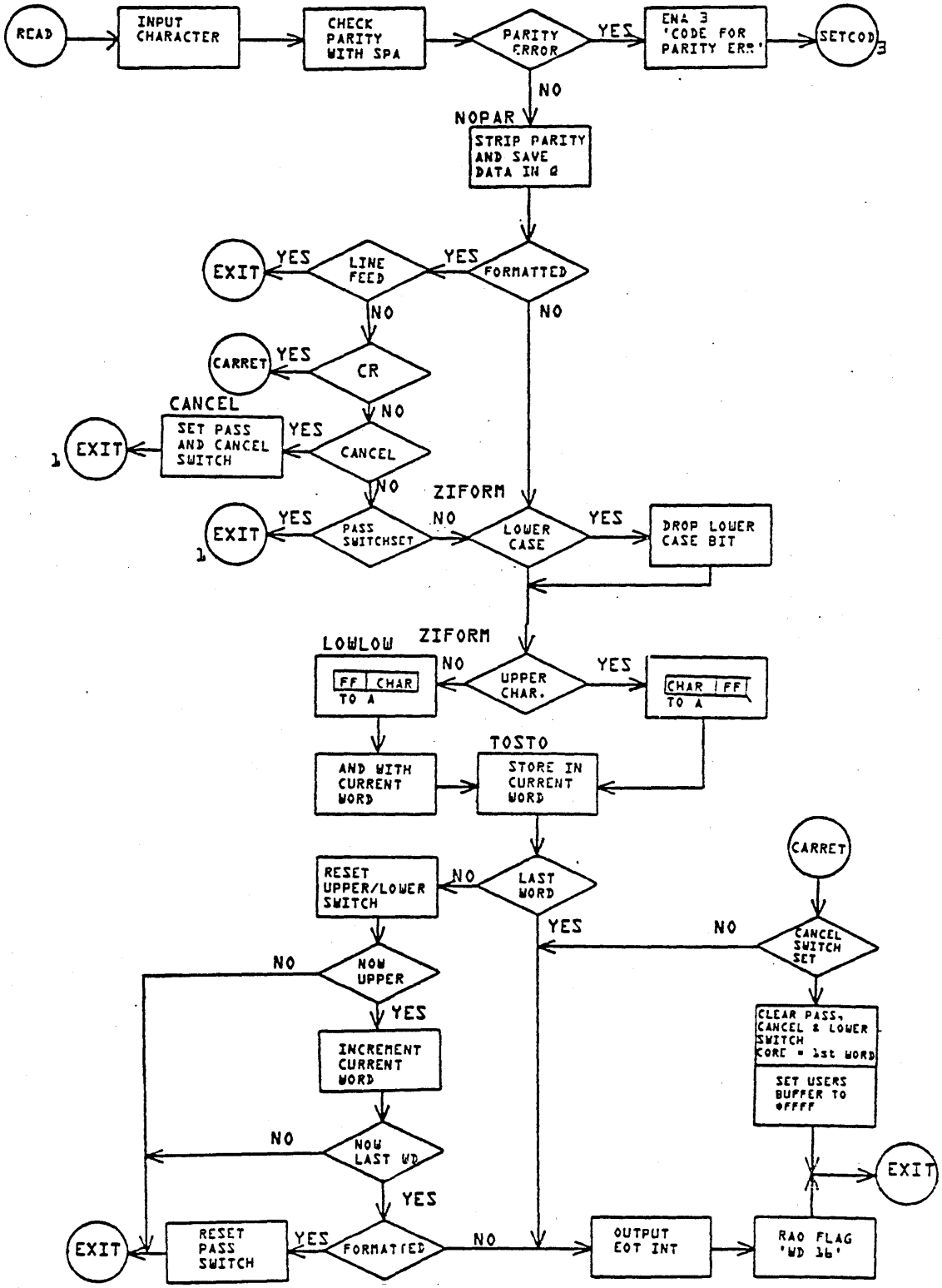
D1713K–1

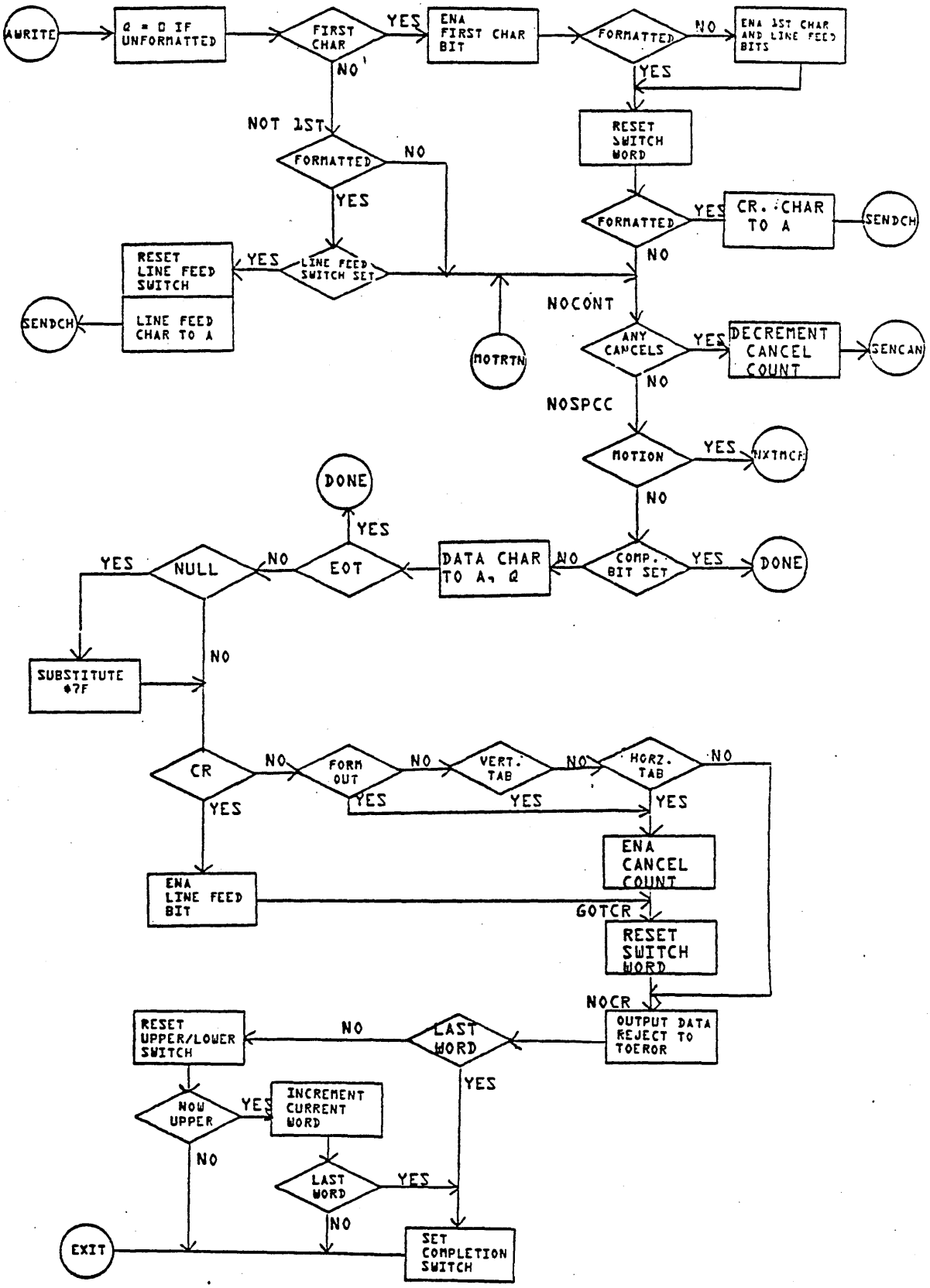

TOFNR
1713K
I = PDT ADDRESS
S13BZY =0
YES
FNR
P + 1
CLROUT
P + 2
NO
W1713K= PDT ADDRESS
S13BZY .= PDT ADDRESS
DISP
CLEAR
INIT2
KEYBOARD MODE
YES
CLEAR COMP SWITCH
COREIN = START ADDR
SELECT WRITE MODE,
INT ON DATA, ALARM
MOTREQ
MOVE PRTS TO TMPWD
CORE = 0
LASTPI = 0
MOTION
S13MOD = *8000
SELECT KEYBOARD MODE
A = 08 (EOP INT)
SENULL
AWRITE
READ
SENCAN
A = 07F 'NULL CHAR'
SENDCH
A = EWES 'D' = 0
OUTPUT CHAR IN A
EXIT
DISP
SET TIMEOUT VALUE
TAKE STATUS AND SAVE

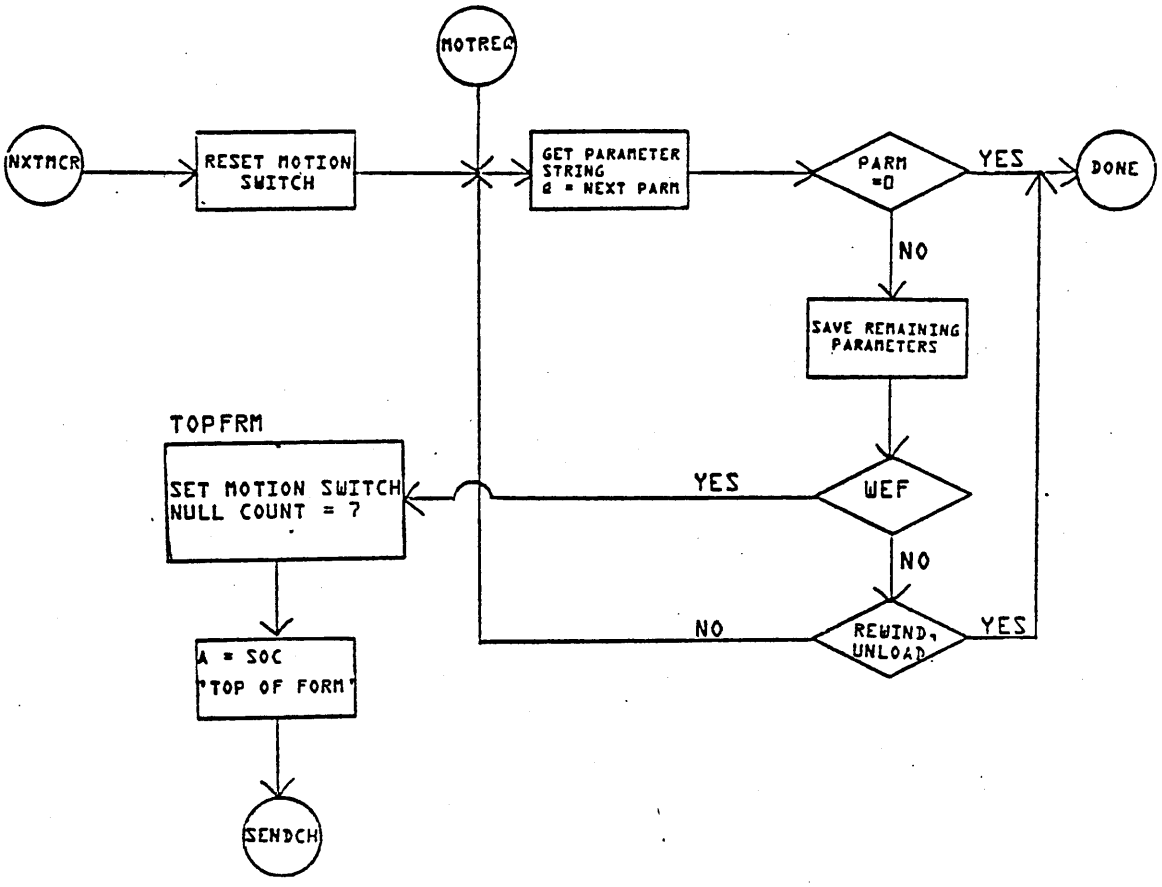
D1713K–2

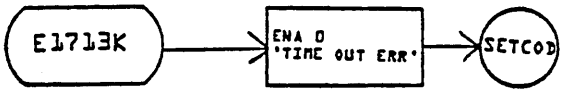

C1713K
I = ADDR. PAYSTB
GET STATUS AND SAVE
MI
FLAGIT =0
SELECT INT ON DATA, ALARM
DISP
TAGIT1
FLAGIT = 1
ALARM
YES
LOST DATA
YES
A = 1
SETCOD
NO
PARITY ERROR
A = 2
SETCOD
YES
A = 3
SETCOD
CLROUT
ELU =0
YES
NO
CLEAR
LOKNXT
Q = 2
W1713K,Q = 0
NO
W1713K, Q=0 Q=ADDR. PDT
INITIATOR
YES
INQ - 1
Q < 0
NO
YES
RELXIT
S13MOD=$0400 CLR INT
YES
S13MOD <0
INIT2
MOTRTN
MOTION
NO
DONE
CMPLET
FLAG = 0
NO
FLAG = 0
NO
DATA
YES
YES
TOFNR
RELXIT
NOTDON
RQAQ
I = ADDR. PHYSTAB Q = EWES FLAGIT = 0
NMONI
SCHEDULE MI AT DRIVER PRIORITY
DATA OR ALARM
YES
EOP NEEDED
NO
CLR INT SELECT INT ON EOP, ALARM
CLR INT SELECT INT ON DATA, ALARM
RELXIT
RLAQ
DISP

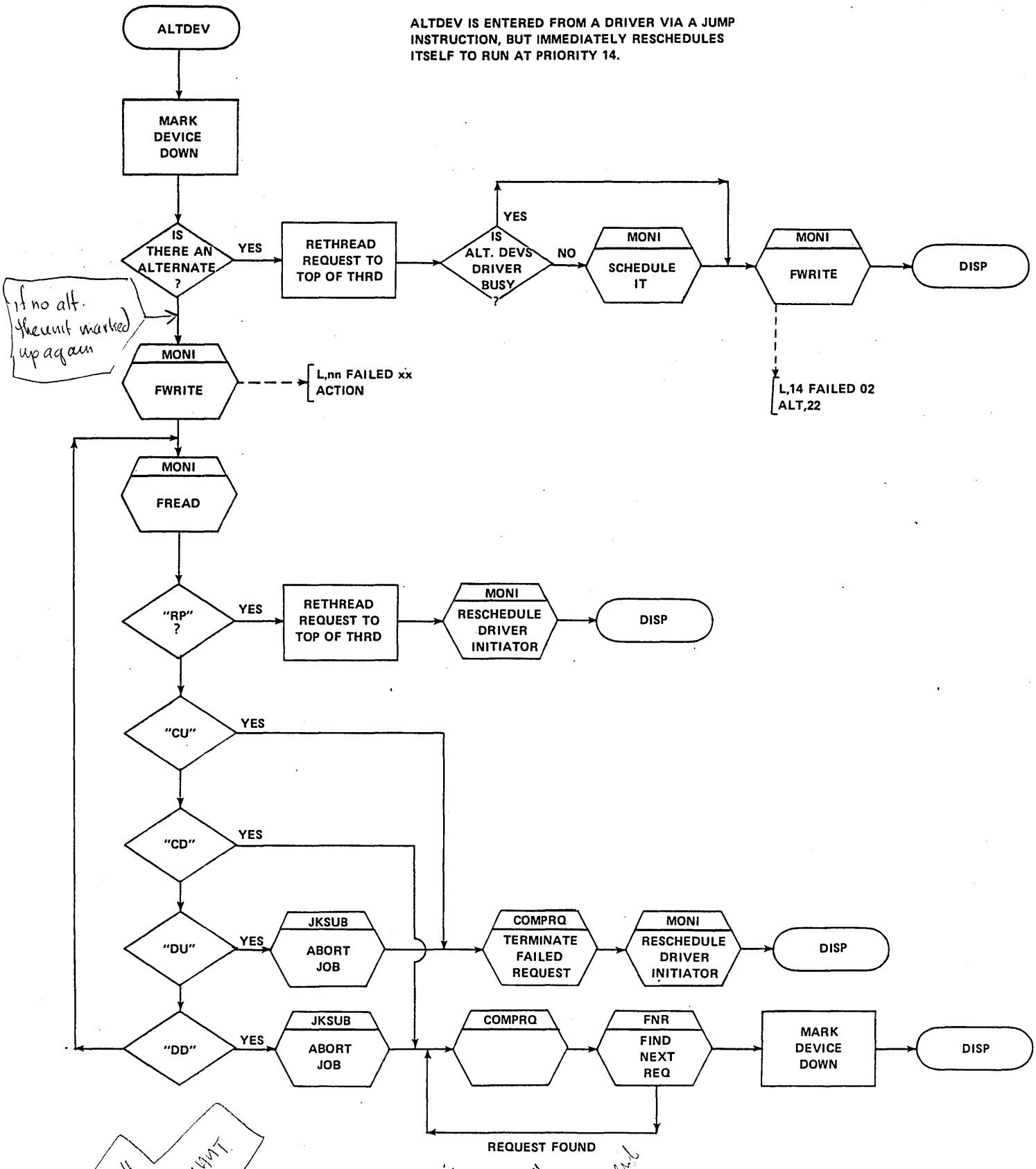

NOTDON
WRITE
NO
YES
AWRITE
BUSY
NO
YES
READ
FORMATTED
NO
YES
NULL COUNT = 0
NO
NULL COUNT = 1
SENULL
NF READ
CLEAR INTERRUPTS INT. ON DATA-ALARM SELECT READMODE
SENDCH +2
CMPLET
CLEAR INT. EDCLK = -1
TAKE STATUS AND SAVE
MAKEQ
CMPRQ
RLAQ
I = PDT ADDR. S13BZY = 0
CMPLET
SETCOD
CLEAR
SETCD1 = ADC DONE RESTORE A
SETCD1
SETCD2
LU+ERROR CODE TO ERRCOD SET ERRTAB
ELU =0
DISP
MAKEQ
ERROR
A = 5 'INTERNAL REJECT'
SETCOD
ERROR +1
A = 6 'EXTERNAL REJECT'
SETCOD
EDCLK = -1 S13BZY = 0
SETCD1
RLAQ
LO =DIAGLU
LOG
ALTDEV


READ
INPUT CHARACTER
CHECK PARITY WITH SPA
PARITY ERROR
YES
ENA 3 'CODE FOR PARITY ERR'
SETCOD
3
NO
NOPAR
STRIP PARITY AND SAVE DATA IN Q
FORMATTED
YES
LINE FEED
YES
EXIT
NO
CARRET
YES
CR
NO
CANCEL
SET PASS AND CANCEL SWITCH
YES
CANCEL
NO
1
EXIT
ZIFORM
PASS SWITCHSET
YES
1
EXIT
NO
LOWER CASE
YES
DROP LOWER CASE BIT
LOWLOW
ZIFORM
FF CHAR TO A
NO
UPPER CHAR.
YES
CHAR FF TO A
AND WITH CURRENT WORD
TOSTO
STORE IN CURRENT WORD
RESET UPPER/LOWER SWITCH
NO
LAST WORD
YES
CARRET
NO
CANCEL SWITCH SET
NO
NOW UPPER
YES
INCREMENT CURRENT WORD
CLEAR PASS, CANCEL & LOWER SWITCH
CORE = 1st WORD
SET USERS BUFFER TO #FFFF
NO
NOW LAST WD
YES
EXIT
EXIT
RESET PASS SWITCH
YES
FORMATTED
NO
OUTPUT EOT INT
RAO FLAG 'WD 16'


AWRITE
Q = 0 IF UNFORMATTED
FIRST CHAR
YES
NO
ENA FIRST CHAR BIT
FORMATTED
NO
ENA 1ST CHAR AND LINE FEED BITS
YES
NOT 1ST
FORMATTED
NO
YES
RESET SWITCH WORD
FORMATTED
YES
CR. CHAR TO A
SENDCH
NO
RESET LINE FEED SWITCH
YES
LINE FEED SWITCH SET
SENDCH
LINE FEED CHAR TO A
MOTRTN
NOCONT
ANY CANCELS
YES
DECREMENT CANCEL COUNT
SENCAN
NO
NOSPCC
MOTION
YES
NXTMCE
NO
DONE
YES
NULL
NO
EOT
DATA CHAR TO A, Q
NO
COMP. BIT SET
YES
DONE
YES
SUBSTITUTE 07F
NO
CR
NO
FORM OUT
NO
VERT. TAB
NO
HORZ. TAB
NO
YES
YES
YES
YES
ENA CANCEL COUNT
ENA LINE FEED BIT
GOTCR
RESET SWITCH WORD
NOCR
OUTPUT DATA REJECT TO TOEROR
RESET UPPER/LOWER SWITCH
NO
LAST WORD
YES
NOW UPPER
YES
INCREMENT CURRENT WORD
NO
LAST WORD
YES
NO
EXIT
SET COMPLETION SWITCH


MOTREQ
NXTMCR
RESET MOTION SWITCH
GET PARAMETER STRING
Q = NEXT PARM
PARM =0
YES
DONE
NO
SAVE REMAINING PARAMETERS
TOPFRM
SET MOTION SWITCH
NULL COUNT = 7
YES
WEF
NO
NO
REWIND, UNLOAD
YES
A = SOC
'TOP OF FORM'
SENDCH


E1713K
ENA D
'TIME OUT ERR'
SETCOD


ALTDEV
ALTDEV IS ENTERED FROM A DRIVER VIA A JUMP INSTRUCTION, BUT IMMEDIATELY RESCHEDULES ITSELF TO RUN AT PRIORITY 14.
MARK DEVICE DOWN
IS THERE AN ALTERNATE ?
YES
RETHREAD REQUEST TO TOP OF THRD
IS ALT. DEVS DRIVER BUSY ?
YES
NO
MONI
SCHEDULE IT
MONI
FWRITE
DISP
L,14 FAILED 02
ALT,22
if no alt. the unit marked up again
MONI
FWRITE
L,nn FAILED xx
ACTION
MONI
FREAD
"RP" ?
YES
RETHREAD REQUEST TO TOP OF THRD
MONI
RESCHEDULE DRIVER INITIATOR
DISP
"CU"
YES
"CD"
YES
"DU"
YES
JKSUB
ABORT JOB
COMPRQ
TERMINATE FAILED REQUEST
MONI
RESCHEDULE DRIVER INITIATOR
DISP
"DD"
YES
JKSUB
ABORT JOB
COMPRQ
FNR
FIND NEXT REQ
MARK DEVICE DOWN
DISP
REQUEST FOUND
Priority 14 NOT RECURSIVE
NOTE: If unit is marked down then driver + rescheduled

STUDY QUESTIONS - 7

1. What system routines are common to all drivers? FNR, MAKEQ, ~~INIT~~, ~~CONTIN~~ ALTDEV (ADEV) COMPREQ COMPRQ
2. What is the function of the INIT portion of a driver? UNTHREAD THE REQUEST. IF NONE - DRIVER GOES TO DISPATCHER
3. When is the requestors thread word threaded to the LOG2 and when is it unthreaded, when is it cleared? THREADED BY RW, UNTHREADED BY FNR, CLEARED BY COMPLETE REQUEST
4. Who passes control to each of three divisions of the driver?
5. What are the first three instructions of every driver, why?
6. Who clears bit 15 of word 8 in PHYSTB? 15 indicates if request is active CLEARED BY COMPLETE REQUEST
7. How does MAKEQ know if error has occurred? BIT 14 word 8 INDICATES AN ERROR OCCURED.
8. Who schedules the completion address? RW handles initiation of the request. Complete request handles the scheduling of completion address.
9. What is the function of the continuator portion of the driver? Acknowledges the interrupt(s)

4) 3 divisions OF DRIVER — who passes control

INITIATOR	SCHEDULER (scheduled by R/W)
continuator	interrupt line processor
Error handler	DIAGNOSTIC TIMER

5) STQ-I SAVES PHYSTAB ADDRESS)
RTJ-FNR EXPECTS TO FIND PHYSTAB ADDR IN I
JMP-DISP NO more requests
FNR returns to call+1 if no more
call+2 if found a request

drivers ("scheduler stack") = log 2 thread

LESSON GUIDE 8

MEMORY ALLOCATION

LESSON PREVIEW:

This session will give a detailed presentation of the two dynamic memory allocation schemes under MSOS. The drivers and swapping schemes associated with each will also be discussed.

REFERENCES:

Pages 2-15 and 2-16 of MSOS 5 RM
Listings of SPACE, RW, DCORE, ALCORE, and SYSDAT

TRAINING AIDS:

Visuals V8-1 through V8-6.

PROJECTS:

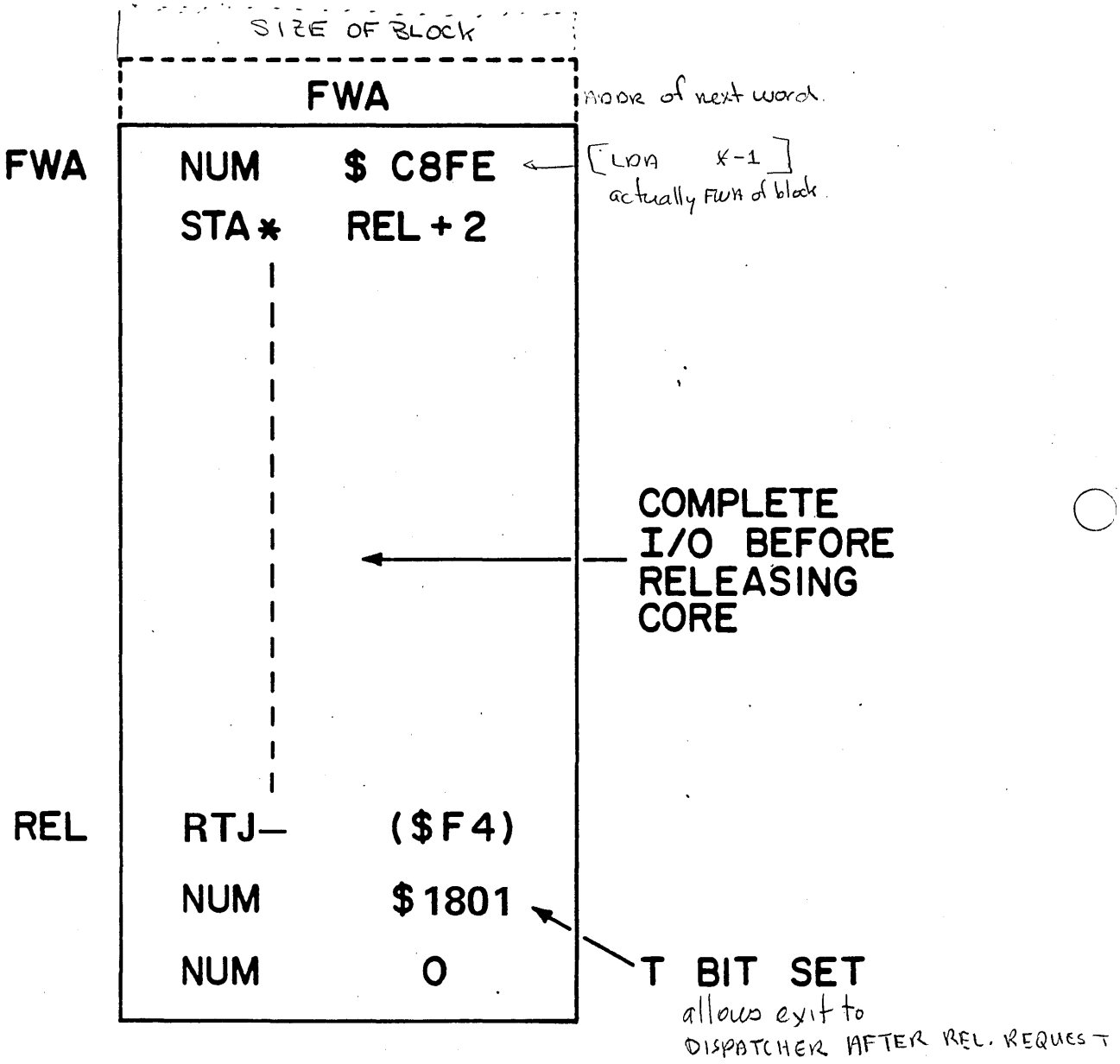
OBJECTIVES:

At the completion of this lesson, the student will be able to:

1. Describe the allocation algorithm for allocatable core and Partitioned core.
2. Establish the parameters for the Space Allocator and PTNCOR Allocator (LVLSTR, PARTBL)
3. Understand the significance of the RP parameter in the System Directory.

if you request a has place of allocatable
when you ask for allocatable you must release it.

MASS MEMORY PROGRAM IN ALLOCATABLE CORE



SPACE, PTNCOR, RELEAS, SWAPPING AND RESTART

GENERAL BACKGROUND

Many modules are nonresidents, i.e., they are not kept in core. Therefore, when they are operated, it is necessary to read them in from the library. There is an area reserved for this purpose, the size of which varies from system to system. Each nonresident program, prior to operation, must be assigned space in this area and read into it. Similarily, when a nonresident program completes its function, it must cause the area allocated to it to be restored to the block of empty space available for allocation to other nonresident programs. The SPACE, PTNCOR and RELEAS requests deal with these operations.

If it is necessary to allocate space in the nonresident area and insufficient space is available, it may be possible to preempt that area of core used for job processing. The procedure involved is called swapping.

For purposes of allocating core space in as simple a manner as possible, the area to be allocated is treated as an I/O device. This pseudo device is operated by a pseudo controller (the core allocator) which is operated via a driver (SPACDR). The SPACE and RELEAS requests take the place of READ and WRITE requests in this situation. In order for this operation to work smoothly, the pseudo device is always considered to be logical unit #1. This is true for all systems. The modules to be discussed in this lesson are:

CORE ALLOCATOR
SPACDR
SPACE REQUEST PROCESSOR
SUBCOR

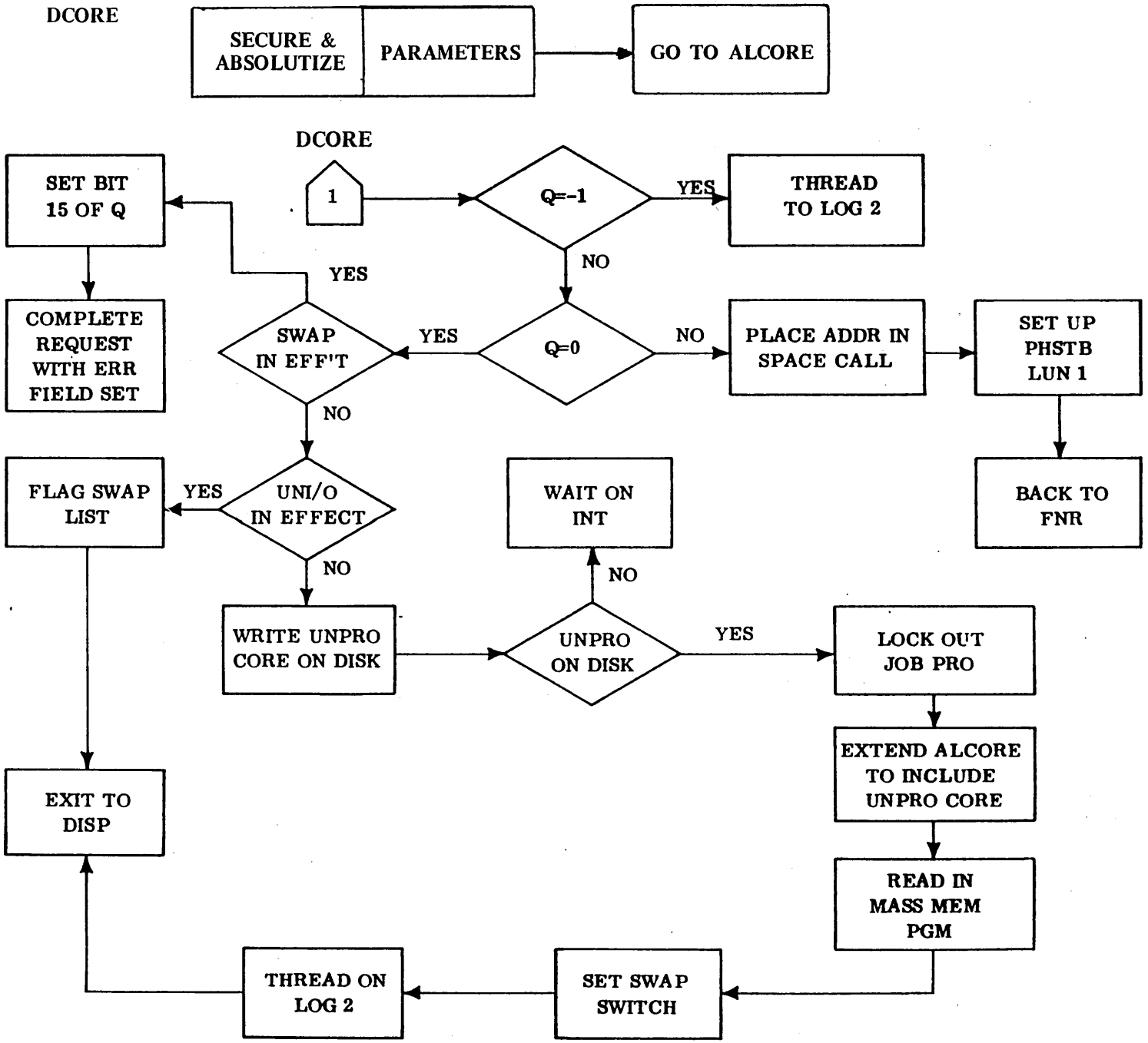
CORE ALLOCATOR

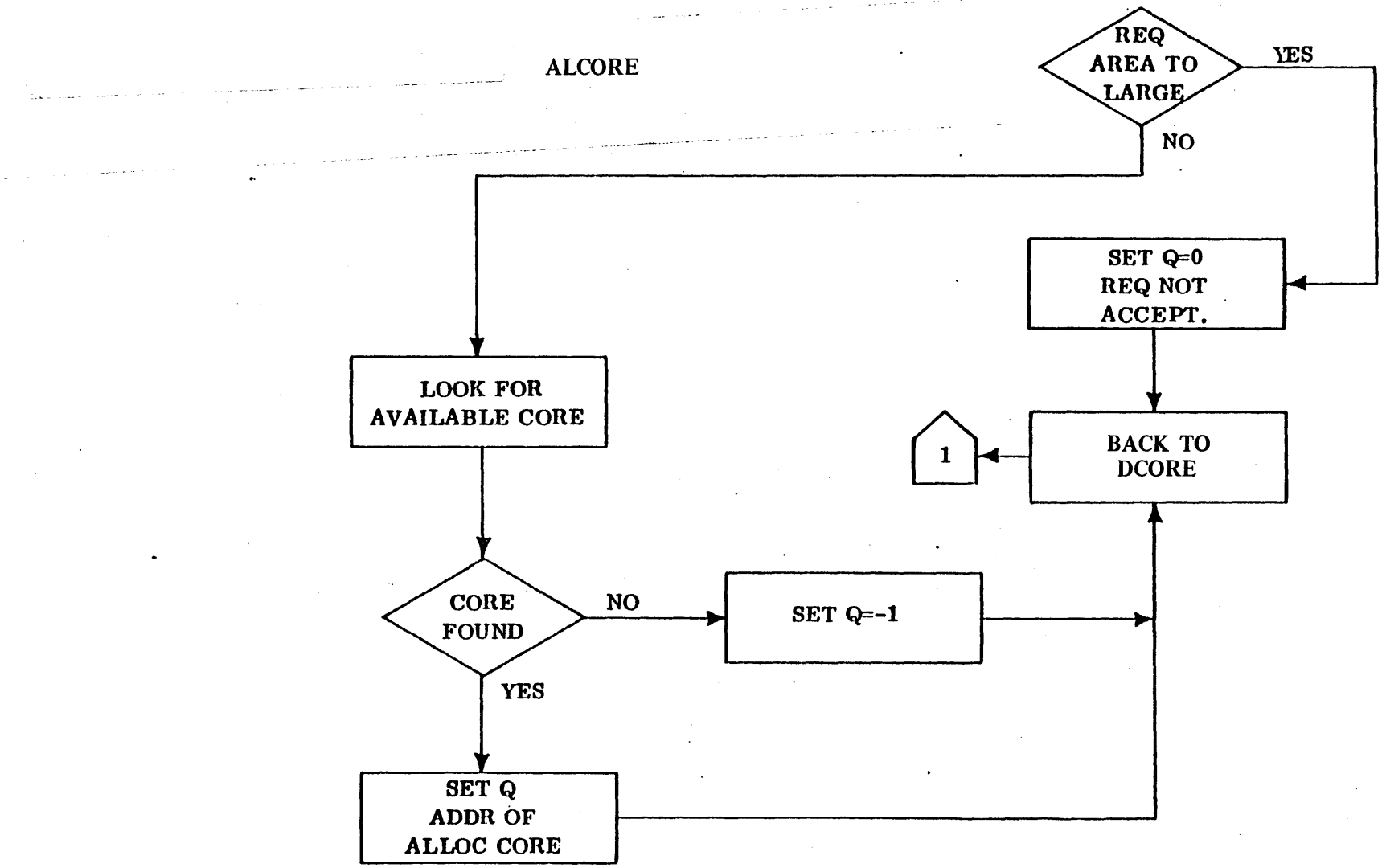
EXTERNAL SYMBOLS

LVLSTR	Level start table
LEND	Level end
CALTHD	Core allocator thread

INTERNAL SYMBOLS

MINSIZ	Minimum allocatable area (assembled as 2)
MAXNO	Largest single precision positive number


DCORE
SECURE & ABSOLUTIZE
PARAMETERS
GO TO ALCORE
DCORE
1
Q=-1
YES
THREAD TO LOG 2
NO
SET BIT 15 OF Q
YES
COMPLETE REQUEST WITH ERR FIELD SET
SWAP IN EFF'T
YES
Q=0
NO
PLACE ADDR IN SPACE CALL
SET UP PHSTB LUN 1
NO
BACK TO FNR
FLAG SWAP LIST
YES
UNI/O IN EFFECT
WAIT ON INT
NO
NO
WRITE UNPRO CORE ON DISK
UNPRO ON DISK
YES
LOCK OUT JOB PRO
EXTEND ALCORE TO INCLUDE UNPRO CORE
EXIT TO DISP
READ IN MASS MEM PGM
THREAD ON LOG 2
SET SWAP SWITCH


ALCORE
REQ AREA TO LARGE
YES
NO
SET Q=0 REQ NOT ACCEPT.
LOOK FOR AVAILABLE CORE
1
BACK TO DCORE
CORE FOUND
NO
SET Q=-1
YES
SET Q ADDR OF ALLOC CORE

Scheduling a mass memory resident system directory program causes the following operations to be executed.

1. Space is assigned in the allocatable core area.

2. The program is read into core from mass memory.

3. The starting address of the program, i.e., the start of the assigned core area, is scheduled at the requested priority.

All mass memory resident system directory programs that are to be run in allocatable core must be written to be "run anywhere" (using relative addressing, etc.) since the program amy be assigned different core areas on successive operations. The mass memory programs that are to run in partitioned core must be absolutized relative to a particular partition and then run at that address only.

FUNCTION OF THE PROGRAM

The Core Allocator module allocates core to programs which are mass memory resident. It also allocates core to programs which require additional temporary working area at execution time.

The Core Allocator is required in the monitor on all systems which have a mass memory in allocatable core.

The Core Allocator accept returned areas of core and, if possible, combines the returned area with adjacent areas.

Requests for core allocation are stacked by request priority and core is allocated on a priority basis; i.e., the higher priority programs have access to more of the allocatable core.

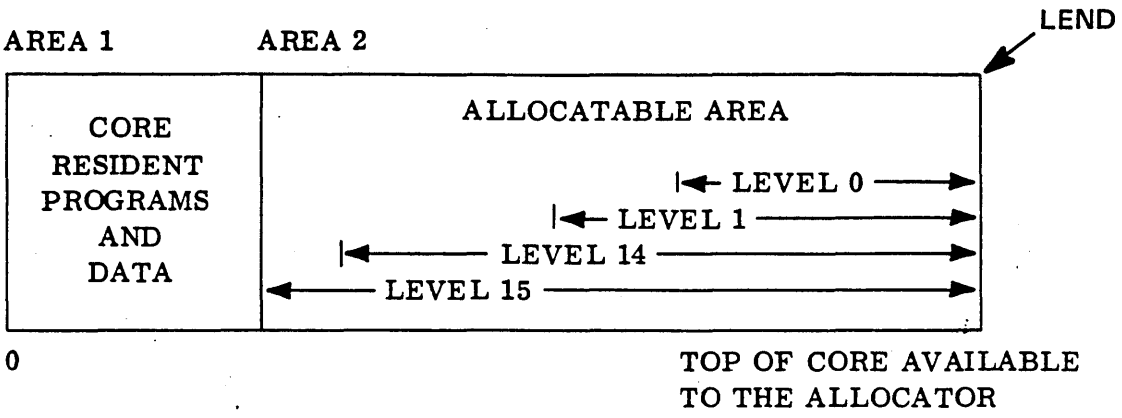
COMPREHENSIVE PROGRAM DESCRIPTION

The Core Allocator threads together all the pieces of available core memory. Initially there is one piece of core which is the entire area. As allocations are made, the available area gets broken up into many pieces. As pieces are returned, they are regrouped into as few pieces as possible. The thread of available pieces is arranged in ascending address order.

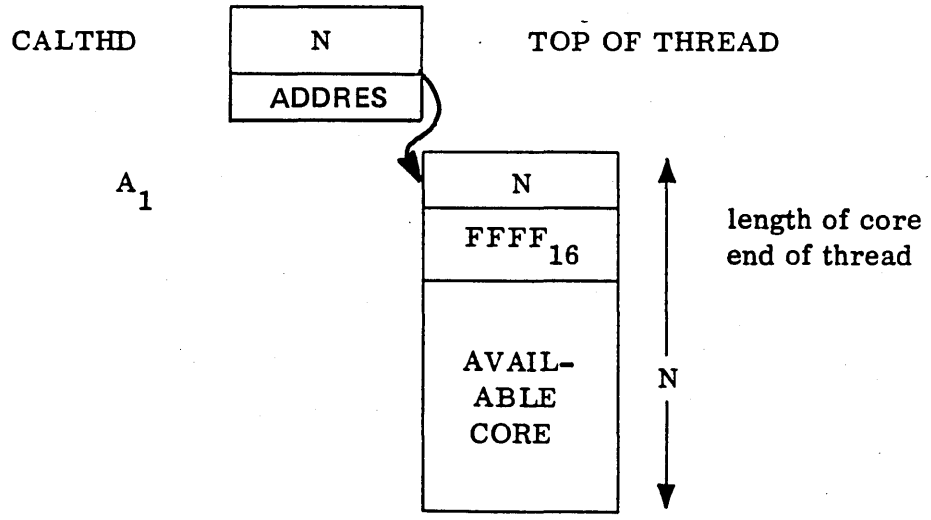
ORGANIZATION OF CORE

Part 0 is divided into two areas:
Area 1) the core resident programs constants; Area 2) the allocatable area.

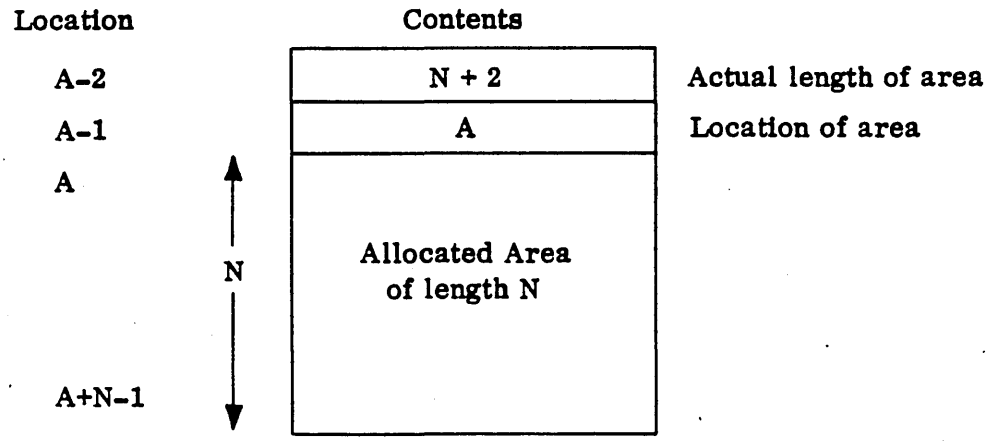
PART 0



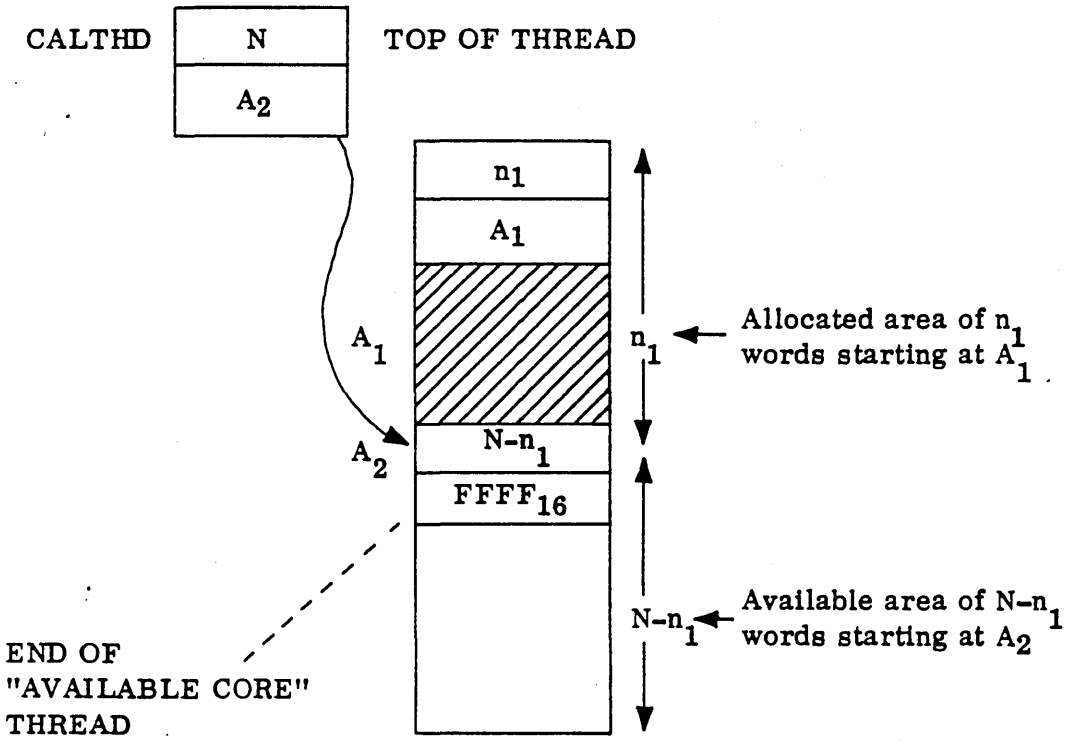
CORE MEMORY IS INITIALIZED AS FOLLOWS:



INDIVIDUAL PIECES OF ALLOCATED CORE ARE ORGANIZED AS:



After an allocation has been made, core memory appears as shown below:



Area 2 is allocated by the core allocator according to the request priority in the parameter list. A fixed amount of the available core is available to each priority level. Higher priority levels have access to more of the core than lower priorities. This has the effect of guaranteeing that many low priority programs cannot use an area set aside for a high priority program. An area can always be available to a higher level by restricting the area available to lower levels. The core allocator also selects the core from the smallest available piece. This has the effect of minimizing the number of pieces of core that are twoo small to be usable. The technique uses the small leftover pieces first while leaving the big pieces for future requests.

The core allocator stores two control words into the allocated core area. The first word, located at "A-2" always contains the requested length N, plus 2, and represents the actual length of the allocated area. The second word, located at "A-1", always contains the address of the area, A.

CORE ALLOCATION LOGIC

The subroutine, REQALC, (request allocation) actually does the analysis to select the available area of memory. The logic is discussed below. REQALC is called by the Core Allocator Driver with the parameters, requested length and level.

If the requested length is larger than the area available to the requested level, then REQALC immediately returns with a zero parameter to the driver.

Otherwise, a search of all available core is made to select that piece which has the following properties:

1. The piece must contain N+2 words available to the requested level.

2. The remaining piece (after N+2 words are allocated) is smaller than the corresponding piece of all other allocatable areas.

If no such piece is found, then the parameter, -1, is returned to the Core Allocator Driver. Otherwise, the optimal piece is broken into two or three parts, and the thread of available core is strung through the leftover piece. The leftover pieces are restricted to being larger than MINSIZ so that they can contain the thread information.

CORE RETURN LOGIC

The subroutine RTNCOR does the analysis to combine the return piece of core with the already available pieces. RTNCOR is entered from the RELEAS request processor (SPACDR).

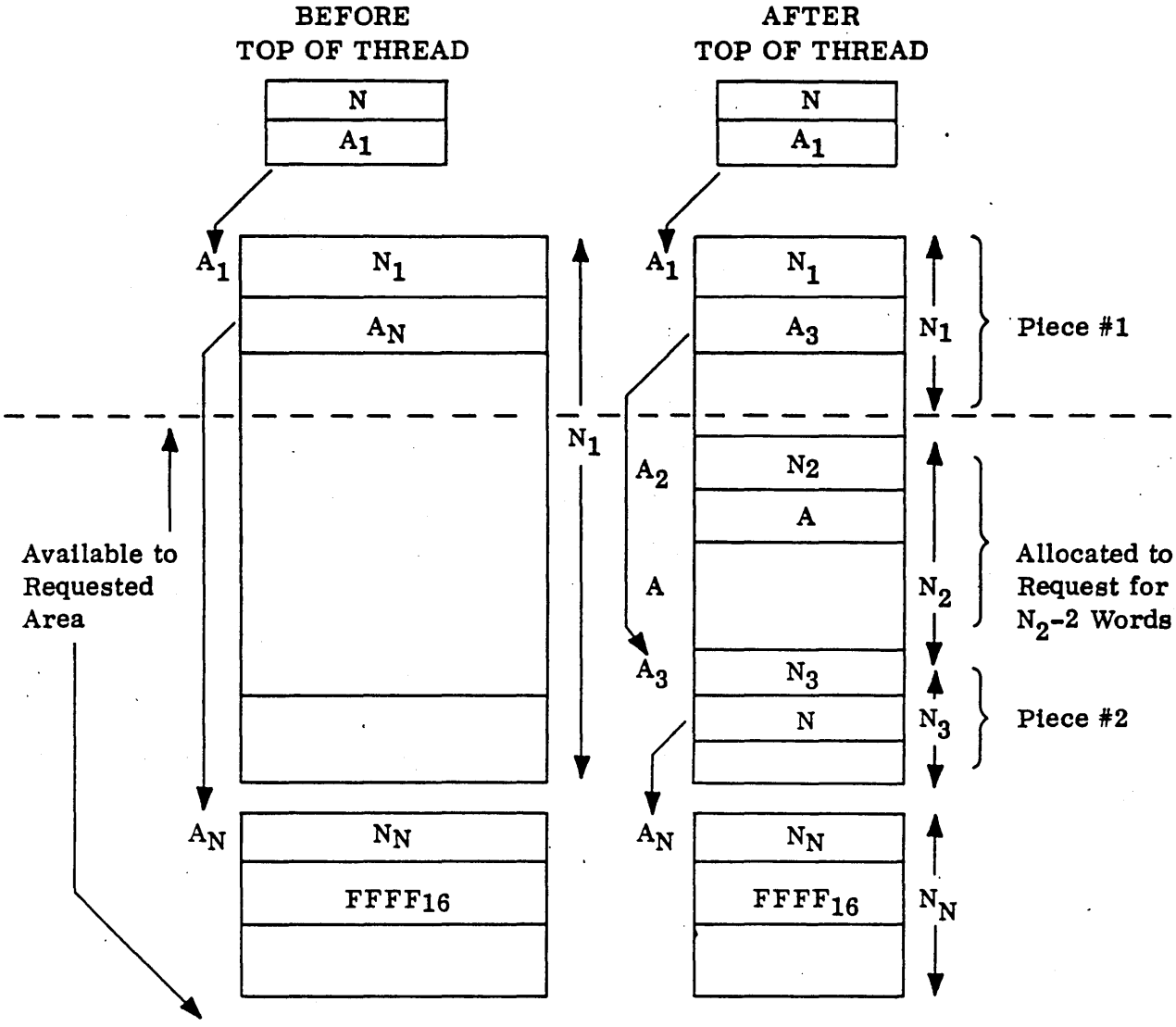
A search is made to find the first piece of available core which is below the returned piece. The returned piece is threaded into its proper position (the available core thread is ordered by ascending core location).

A check is made to see if the returned piece touches its lower and/or upper neighbor. If so, the adjacent pieces are combined into one piece and the thread is updated.

TABLES

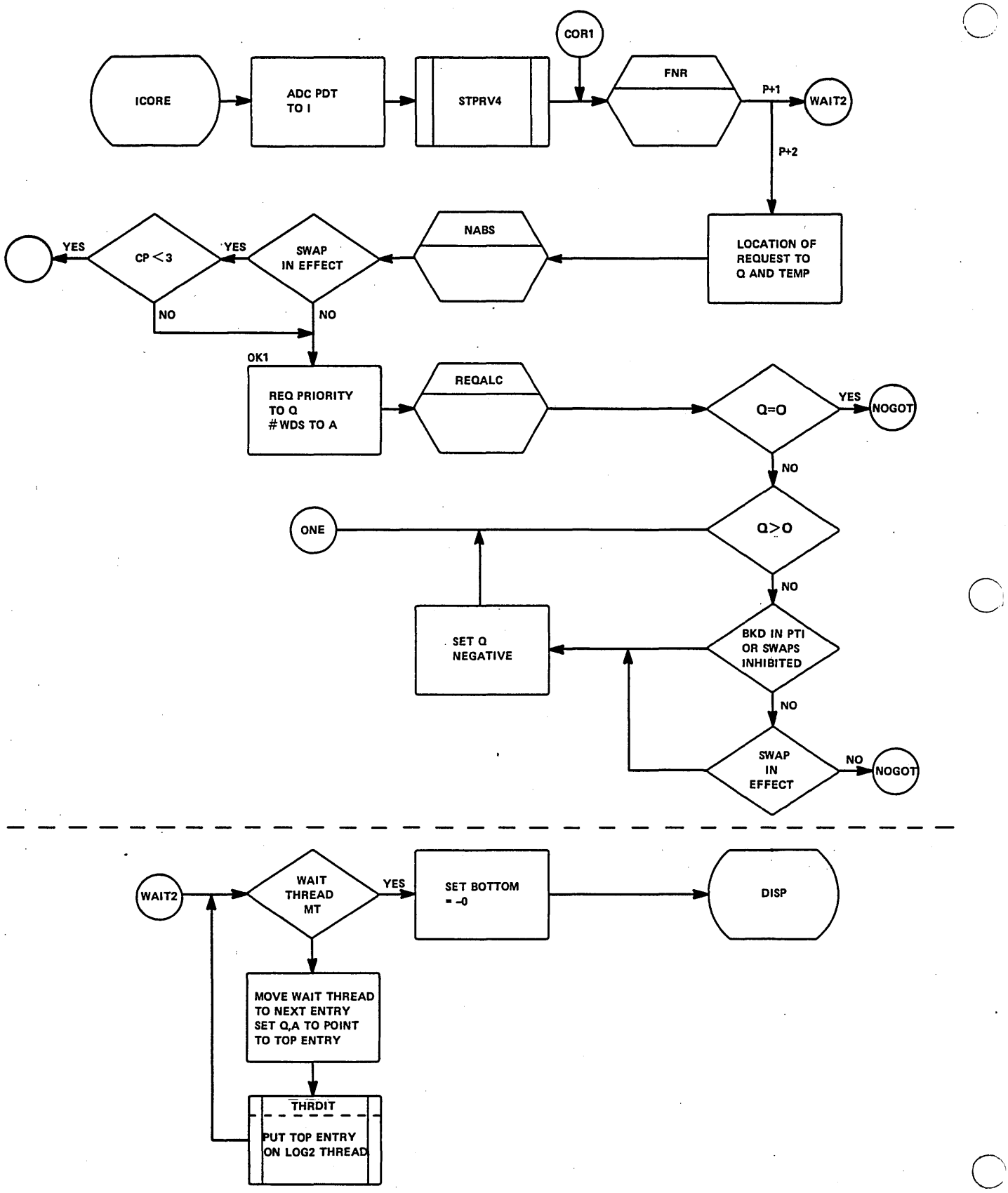
LVLSTR — This table contains 17 cells and is located in the system table module. The first 16 cells are indexed by priority level. Each entry contains the core address of the first cell allocatable to programs with request priorities of the level represented by the index. The last cell contains the address of the last cell in the area which is controlled by the core allocator.

CORE ALLOCATION PIECES

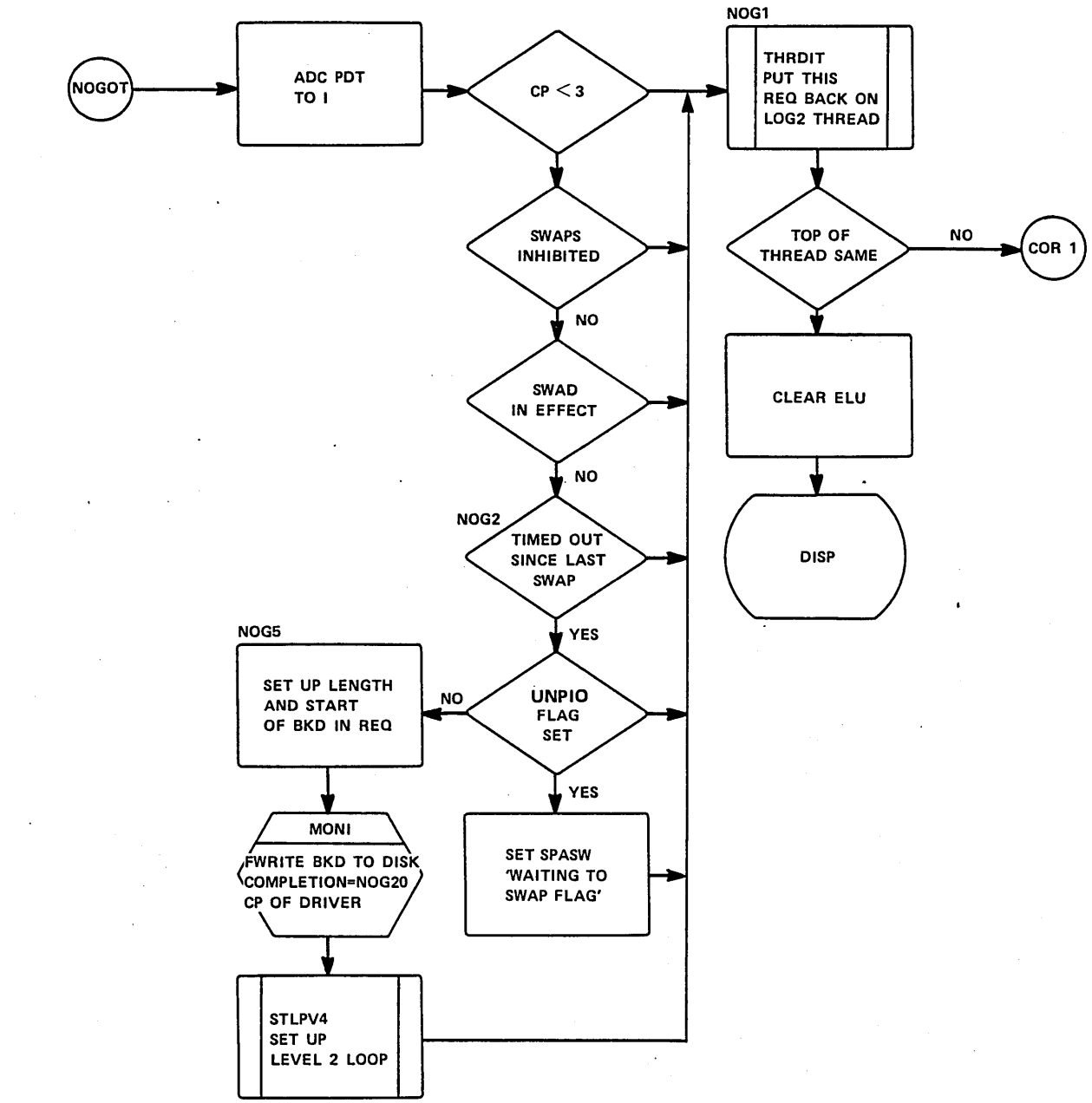


NOTE: PIECE #1 LIES BELOW THE AREA AVAILABLE TO THE LEVEL AND PIECE #2 REMAINS AFTER THE REQUESTED PIECE HAS BEEN REMOVED.

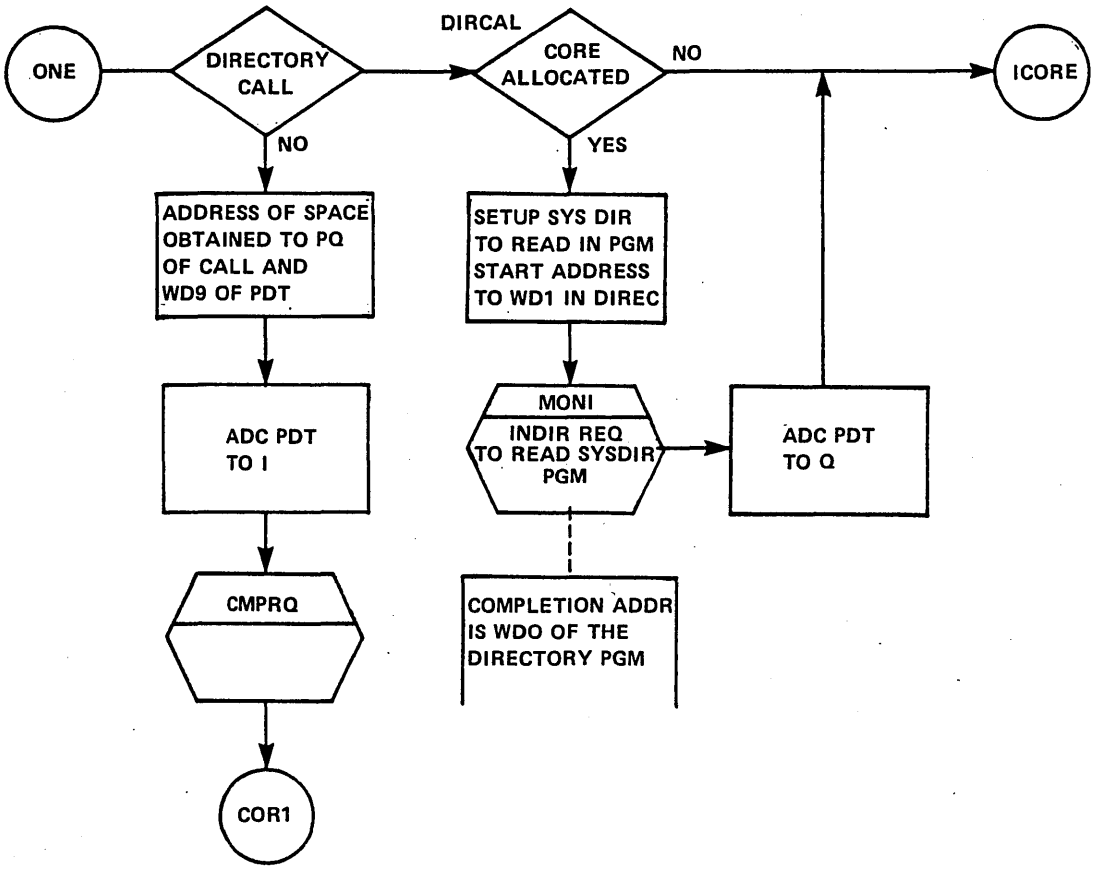
DCORE

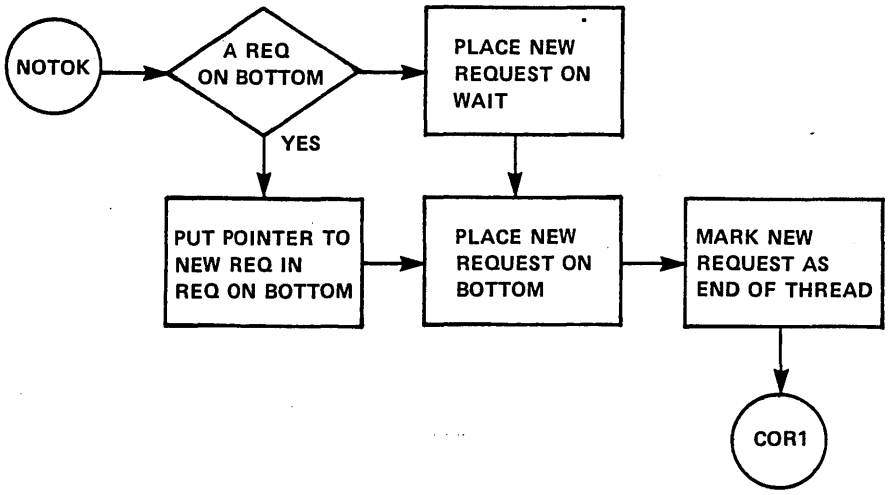

ICORE
ADC PDT TO I
STPRV4
COR1
FNR
P+1
WAIT2
P+2
LOCATION OF REQUEST TO Q AND TEMP
NABS
SWAP IN EFFECT
YES
CP < 3
YES
NO
NO
OK1
REQ PRIORITY TO Q #WDS TO A
REQALC
Q=O
YES
NOGOT
NO
Q>O
ONE
NO
BKD IN PTI OR SWAPS INHIBITED
SET Q NEGATIVE
NO
SWAP IN EFFECT
NO
NOGOT
WAIT2
WAIT THREAD MT
YES
SET BOTTOM = -0
DISP
MOVE WAIT THREAD TO NEXT ENTRY SET Q,A TO POINT TO TOP ENTRY
THRDIT
PUT TOP ENTRY ON LOG2 THREAD

DCORE

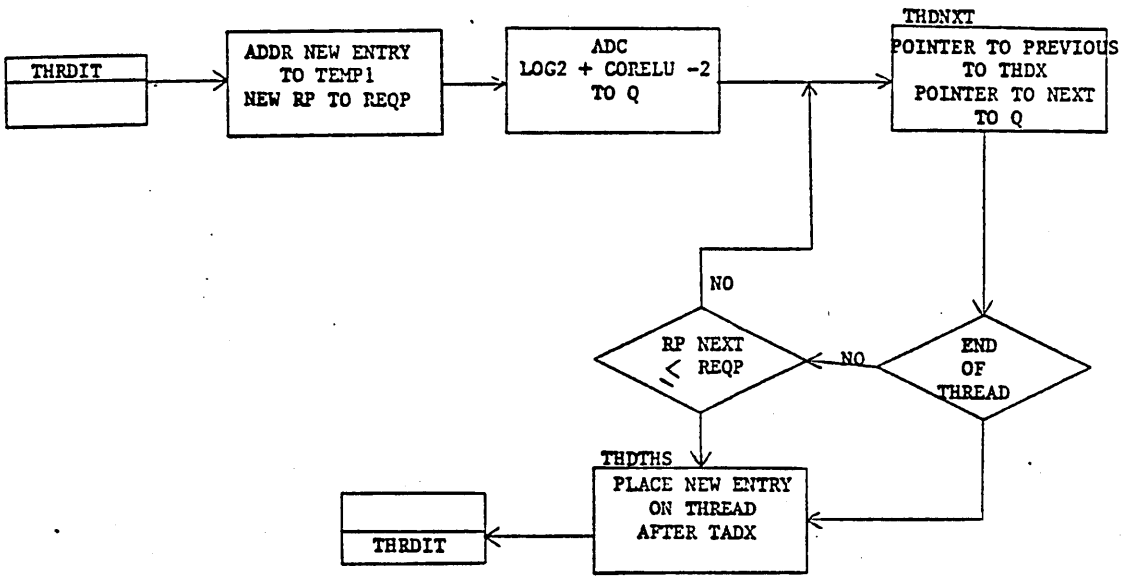

NOGOT
ADC PDT TO I
CP < 3
NOG1
THRDIT PUT THIS REQ BACK ON LOG2 THREAD
SWAPS INHIBITED
TOP OF THREAD SAME
NO
COR 1
NO
SWAD IN EFFECT
CLEAR ELU
NO
NOG2
TIMED OUT SINCE LAST SWAP
DISP
YES
NOG5
SET UP LENGTH AND START OF BKD IN REQ
NO
UNPIO FLAG SET
YES
MONI
FWRITE BKD TO DISK COMPLETION=NOG20 CP OF DRIVER
SET SPASW 'WAITING TO SWAP FLAG'
STLPV4 SET UP LEVEL 2 LOOP
STLPV4
CLEAR WAITING TO SWAP FLAG
LOOP FG = 0
YES
SET LOOPFG ≠ 0
MONI
SCHEDULE LOOPEN CP=2
SET SWAPON FLAG
STLPV4

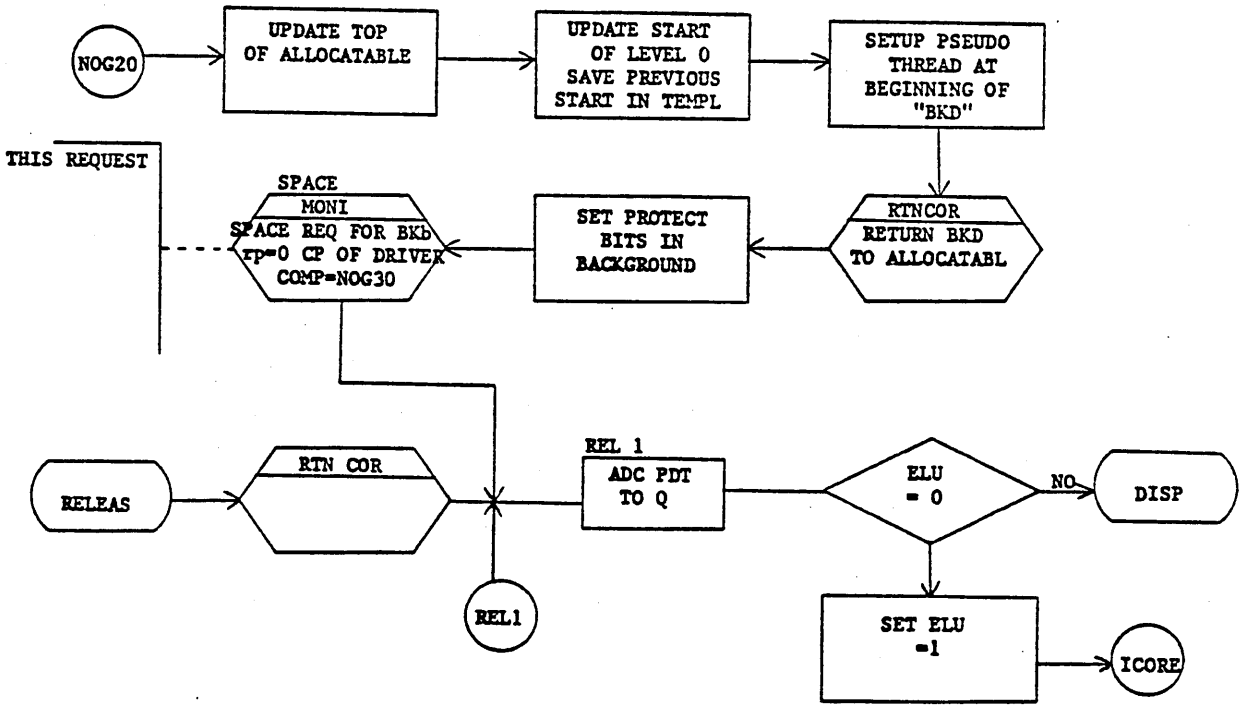
DCORE


ONE
DIRECTORY CALL
DIRCAL
CORE ALLOCATED
NO
ICORE
NO
YES
ADDRESS OF SPACE OBTAINED TO PQ OF CALL AND WD9 OF PDT
SETUP SYS DIR TO READ IN PGM START ADDRESS TO WD1 IN DIREC
ADC PDT TO I
MONI
INDIR REQ TO READ SYSDIR PGM
ADC PDT TO Q
CMPRQ
COMPLETION ADDR IS WDO OF THE DIRECTORY PGM
COR1


NOTOK
A REQ ON BOTTOM
PLACE NEW REQUEST ON WAIT
YES
PUT POINTER TO NEW REQ IN REQ ON BOTTOM
PLACE NEW REQUEST ON BOTTOM
MARK NEW REQUEST AS END OF THREAD
COR1

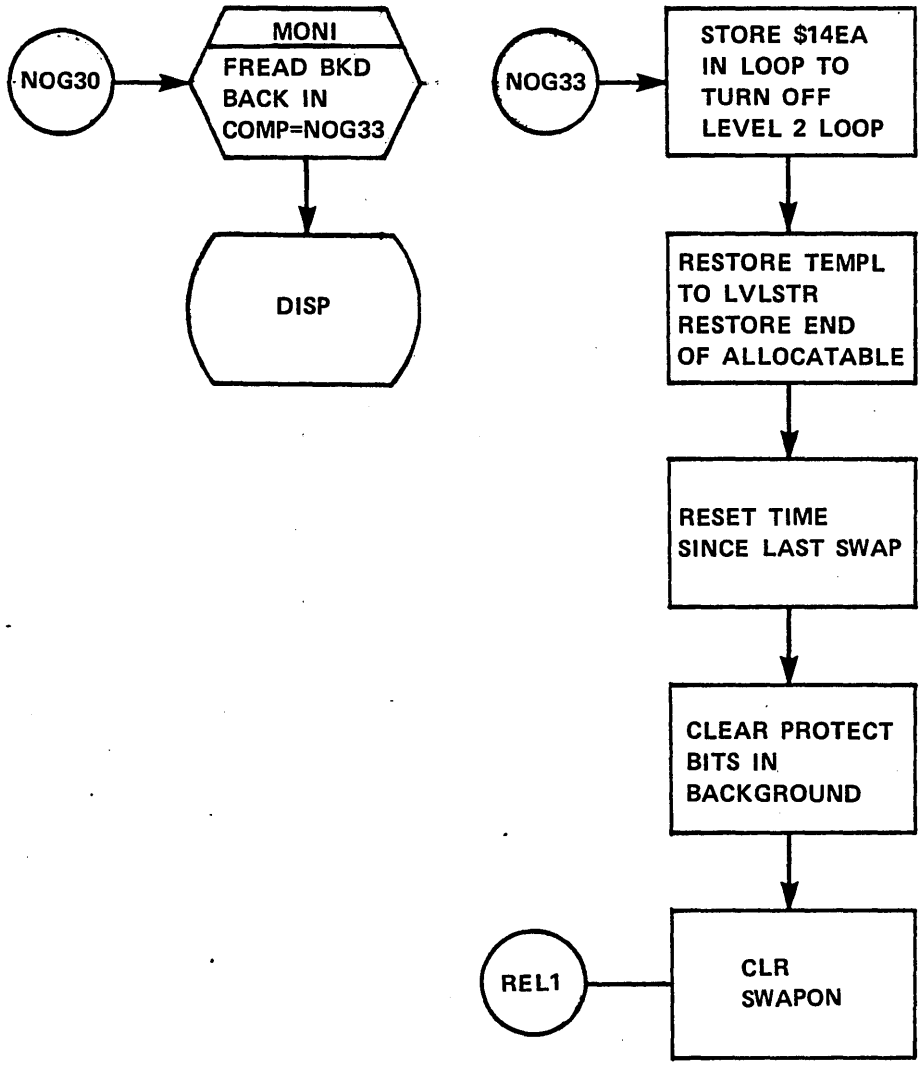
DCORE

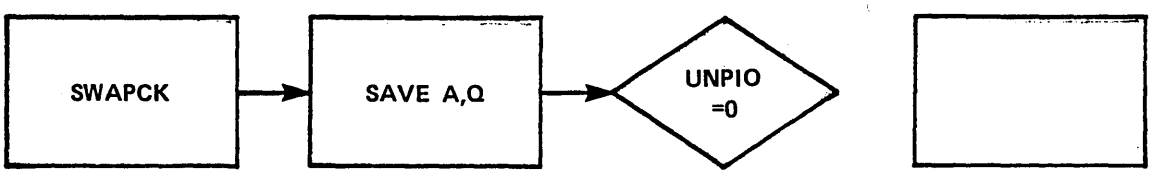

THRDIT
ADDR NEW ENTRY TO TEMP1 NEW RP TO REQP
ADC LOG2 + CORELU -2 TO Q
THDNXT
POINTER TO PREVIOUS TO THDX POINTER TO NEXT TO Q
NO
RP NEXT ≤ REQP
NO
END OF THREAD
THDTHS
PLACE NEW ENTRY ON THREAD AFTER TADX
THRDIT


NOG20
UPDATE TOP OF ALLOCATABLE
UPDATE START OF LEVEL 0 SAVE PREVIOUS START IN TEMPL
SETUP PSEUDO THREAD AT BEGINNING OF "BKD"
THIS REQUEST
SPACE MONI
SPACE REQ FOR BKb rp=0 CP OF DRIVER COMP=NOG30
SET PROTECT BITS IN BACKGROUND
RTNCOR RETURN BKD TO ALLOCATABL
RELEAS
RTN COR
REL 1
ADC PDT TO Q
ELU = 0
NO
DISP
REL1
SET ELU =1
ICORE

DCORE

ENTERED ON COMPLETION OF SPACE REQUEST TO RESTORE BACKGROUND





EXTERNAL SYMBOLS

LEND	Address of last location in the area controlled by the core allocator.
LOGIA	Logical unit table containing PHYSTB addresses for each logical unit.
CALTHD	Core allocator thread.
RTNCOR	Entry to core allocator for releasing space.
CORE	PHYSTB entry for the core allocator.
LVLSTR	Level start table.
SWAPAR	Mass storage address of area where unprotected core contents are saved during swap. Filled by the initializer.
UNPIO	Count of number of unprotected I/O calls pending.
SPASW	A switch in TRANV used to inform the protect processor that a swap is desired.
LOG2	Logical unit table containing thread tops for all logical units.
REQALC	Entry to the core allocator for allocation of space.
AREAC	Start address of block controlled by the core allocator.

INTERNAL SYMBOLS

INTVAL	Number of seconds between swaps. When no timer package used, this should be set to -1 (assembled as 1).
PRI	Priority level of core allocator (assembled as 7).

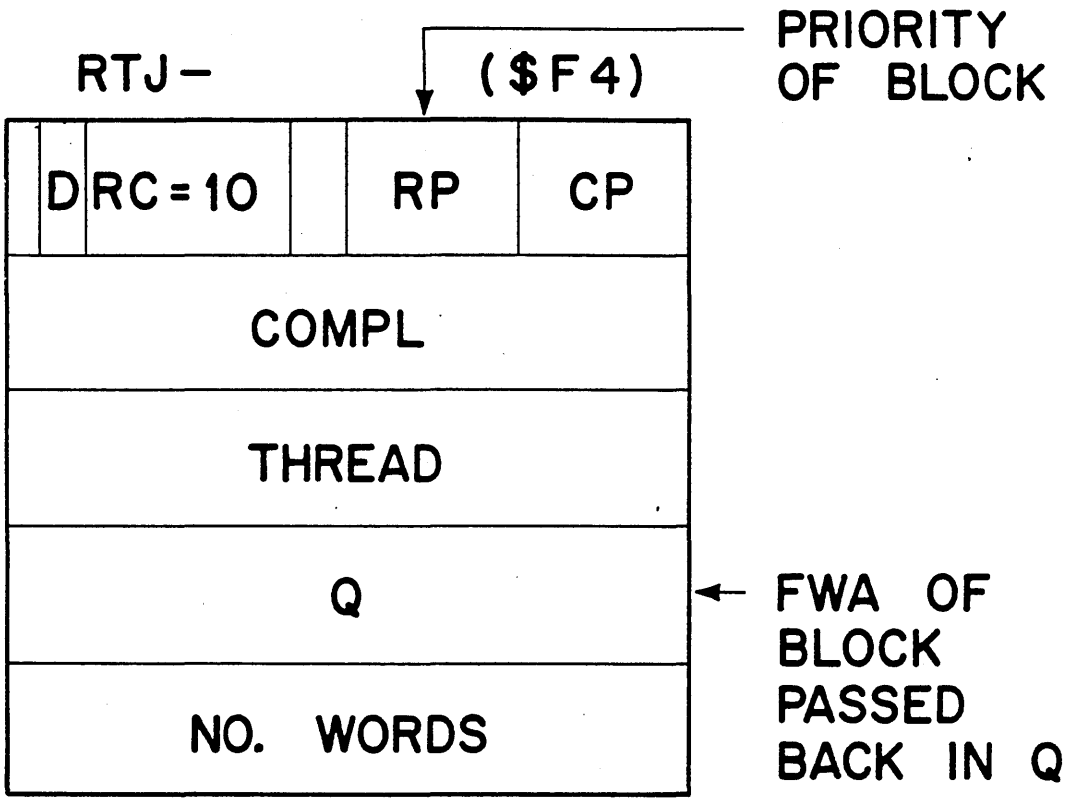
SWAPCK ENTRY POINTS

SWAPCK is the entry point to a subroutine used by the job processer and library edit programs to count down the UNPIO unprotected I/O counter and restart the space driver if it is waiting to swap and UNPIO is zero.

FUNCTION OF THE PROGRAM

SPACDR serves as the driver for the core allocator and as the request processor for RELEAS requests. In this capacity it makes all decisions in the area of swapping and stacking calls for space.

SPACE REQUEST

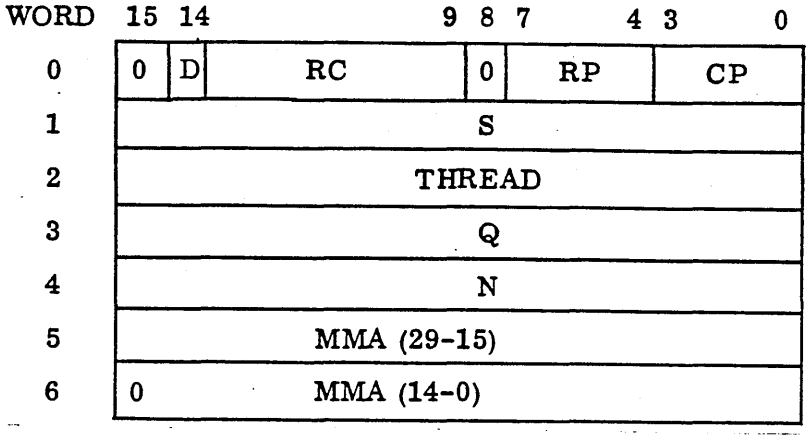


REQUESTS FOR SPACE

Requests for space comes from two sources; namely, schedule calls for nonresident system directory programs and SPACE requests.

SYSTEM DIRECTORY FORMAT

The scheduler gives control to SPACDR when a system directory request for a mass memory resident program is made. SPACDR determines the starting address of the program, based upon the areas of core that are currently available and enters this address in word 1, S, of the System Directory entry. The format for the system directory is shown below:



7 words per entry in the Directory for Mass Memory Resident Programs

RC — is the request code for the System Directory and is zero.

RP — is the request priority used in the allocation of core memory. RP is a number from 0 to 15. (Set by the LIBEDT *S statement). RP=) to 3 is reserved for use by the Job Processor.

CP — is the completion priority at which the mass memory resident program will be scheduled after the read is complete. CP is set for the Scheduler and is obtained from the requesting program's scheduler call.

S — is the starting Core address of the program and also the first location of the allocated core. This is set by the core allocator.

THREAD — is the thread location used to point to the next entry on a threaded list. This directory entry will be placed on the following threads:

THREAD NAME	POSITION DETERMINED BY	WHEN
Core Allocator	RP	after scheduling
Mass Memory I/O Driver	RP	after allocation
Scheduler	CP	after Mass Memory Read

The thread location is set non-zero by the Core Allocator Request Processor and is cleared to zero on completion.

Q is the parameter passed from the requesting program to the requested program.

N is the length in words of this program on mass memory.

MMA is a double-length word containing the mass memory address of this program. The first word contains the most significant 15 bits. The second word contains the least significant 15 bits.

SPACE REQUESTS

The user program may make a Monitor request for allocating core. The core area will be allocated to the requesting program and must be returned by the requesting program before it will be reassigned to another program. The list of parameters is as follows:

	15	14 ... 9	8	7 ... 4	3 ... 0
PARAM=0	0	RC	X	RP	CP
1	C				
2	THREAD				
3	Q				
4	N				

RC is the space request code and is equal to 10.

X is a relative/absolute indicator, modifying C.

RP is the request priority, the relative priority of this request used to determine the position on the core allocator thread and also to determine area of core allowable. RP is a number from 4 to 15.

CP is the completion priority, the level at which control will be returned to C.

C specifies the completion address. Control will return to C after the allocation has been made, or if allocation is impossible.

THREAD is the thread location used to point to the next entry on a threaded list. This monitor request will be placed on the following threads:

THREAD NAME	POSITION DETERMINED BY	WHEN
Core Allocator	RP	after request
Scheduler	CP	after allocation

The thread must initially be zero, and is reset to zero on completion.

Q contains the address of the area allocated and is in the Q register when control is given to the completion address, C. If allocation is impossible Q will be set negative.

N is the number of words requested.

INTERNAL DESCRIPTION OF ALLOCATION

The Space Driver SPACDR is operated by a SCHDLE request from the request processor (just like any other driver). It uses subroutine FNR for new requests and uses the Core Allocator Subroutine, CORALC, to obtain the space required. If sufficient space is available then COMPRQ is used to complete the request. Q will be set to the address of the allocated area when the completion address for the space request is scheduled via COMPRQ. If it is impossible for sufficient space to be available and swapping is in effect then the completion address will be scheduled with Q set negative denoting an error. Errors of this type due to system directory calls cause the system directory call to be ignored but cannot be detected by the caller as no completion address is available.

If sufficient space is not available then an attempt is made to swap, the request is rethreaded and the driver is set "not busy." If core is released before swapping is effected, then the space driver will be reentered and the request will be completed if sufficient space is available. Otherwise the request will be processed after the core swap area is released. For swapping to be executed the following conditions must all be true.

1. The completion priority is greater than 2. This is necessary since programs of level 2 and below are not operated after a swap since they might involve job processing.

2. A swap is not already in effect.

3. A suitable time interval, since the last swap has passed.

4. No unprotected I/O is in progress.

If any of these conditions are not fulfilled, the request is put back on the core request thread just before SPACDR exits to the dispatcher.

Additionally, in the case of condition 4, SPASW is set non-zero so that the protect processor will schedule SPACDR whenever UNPIO-0 and the allocator is not busy.

If the above conditions for swap are fulfilled, then the following operations occur:

1. A write is started which transfers the contents of unprotected core to a designated area on mass storage. This area is set up at system initialization.

2. A loop is scheduled at level 2 to lock out all programs at that level and below.

3. The LVLSTR table and LEND are updated to reflect the additional space available for allocation.

4. SWAPON is set to one, to indicate a swap has occurred.

At the completion of these operations the space driver is marked "not busy" and the request that caused the swap is rethreaded to the top of the LOG2 request thread. When the swap transfer to mass storage is completed, the space driver resumes as follows:

1. The core allocator is entered to release the space just made available.

2. The area is protected.

3. A space request for the swapped area is added to the wait list for threading on the allocator thread at completion of SPACDR processing.

4. A new attempt is made to allocate the space to the call which caused the swap.

When enough space is released so that the area is again available for job processing (the SPACE request made above is completed) the above procedures are reserved and the job is resumed as if no swap occurred.

NOTE: For swapping to combine the allocatable "unprotected" areas, the space request processor must be the last resident module.

The priority level of the space driver is determined by the completion priority set in Word 0 of the CORE physical device table. It is usually set to seven (7). When a swap occurs the space driver must set all the protect bits in the unprotected core area. To do this requires 6.6 microseconds per location. Thus, for an "unprotected" area of size 10K the driver level will be busy in this loop for approximately 66 milliseconds when a swap is requested or released.

The space driver rethreads a request back on to the allocator thread if it is not possible to allocate enough space for the request at that time. No attempt is made to process lower priority requests even though they may require less space. The exception to this rule is if the request to be rethreaded has a completion priority of less than three (3). These requests are put on a wait thread temporarily and then an attempt is made to allocate space to the next request on the allocator thread. When any other requests have been processed requests on the wait thread are returned to the allocator thread.

On completion of job processing, routine JOBEND in the Manual Interrupt Processor is entered to cause a core swap. This is done by making a special Space request that can only be satisfied at the given request priority by a core swap. The special area so allocated is released when the job processor is requested. This area occupies only four cells for the allocator thread at the end of the "unprotected area".

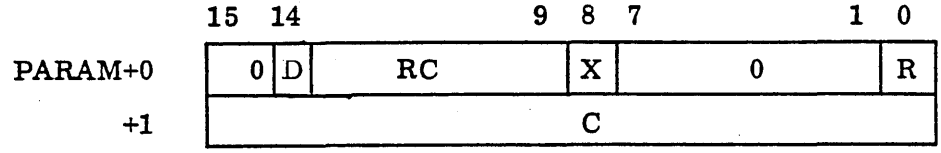
Unnecessary swapping is thus avoided when the job processor is not in use. Excessive swapping on temporary overloads during job processing can be avoided by setting the minimum interval between swaps, INTVAL appropriately. Table LVLSTR must be set up very carefully noting that programs that are not independent cannot be assigned to the same request priority; i.e., they must have separate allocatable areas in which to run. It is not sufficient to provide a total allocatable area at one request priority sufficient for only two dependent programs, since one of the programs could be assigned to the middle of this area, leaving insufficient area for the other program.

RELEAS REQUEST

MONITOR REQUEST FOR RETURNING CORE

All programs that have been allocated core memory, must return the allocated core to the Core Allocator, when they are finished. This includes all mass memory resident programs.

The calling sequence is shown below.



RC . is the request code twelve (12) for returning core.

X is an absolute/relative indicator.

R is the return control indicator. If R=0, control is given to the dispatcher after core is returned. This is the value of R to be used when a program returns the core in which it resides. Since the core will be reallocated, the program residing in it may be destroyed. Thus, control is not returned to the program but to the Dispatcher instead. Otherwise R-1 control is given to the user at the next instruction.

C specifies the area being returned.

If $C_{15} = 0$, X is ignored and $C_{14} - 0$ is the absolute core address of the area being returned. (Absolute direct)

If $C_{15} = 1$ and X = 0, the $C_{14} - 0$ is the location that contains the absolute core address of the area being returned. (Absolute indirect)

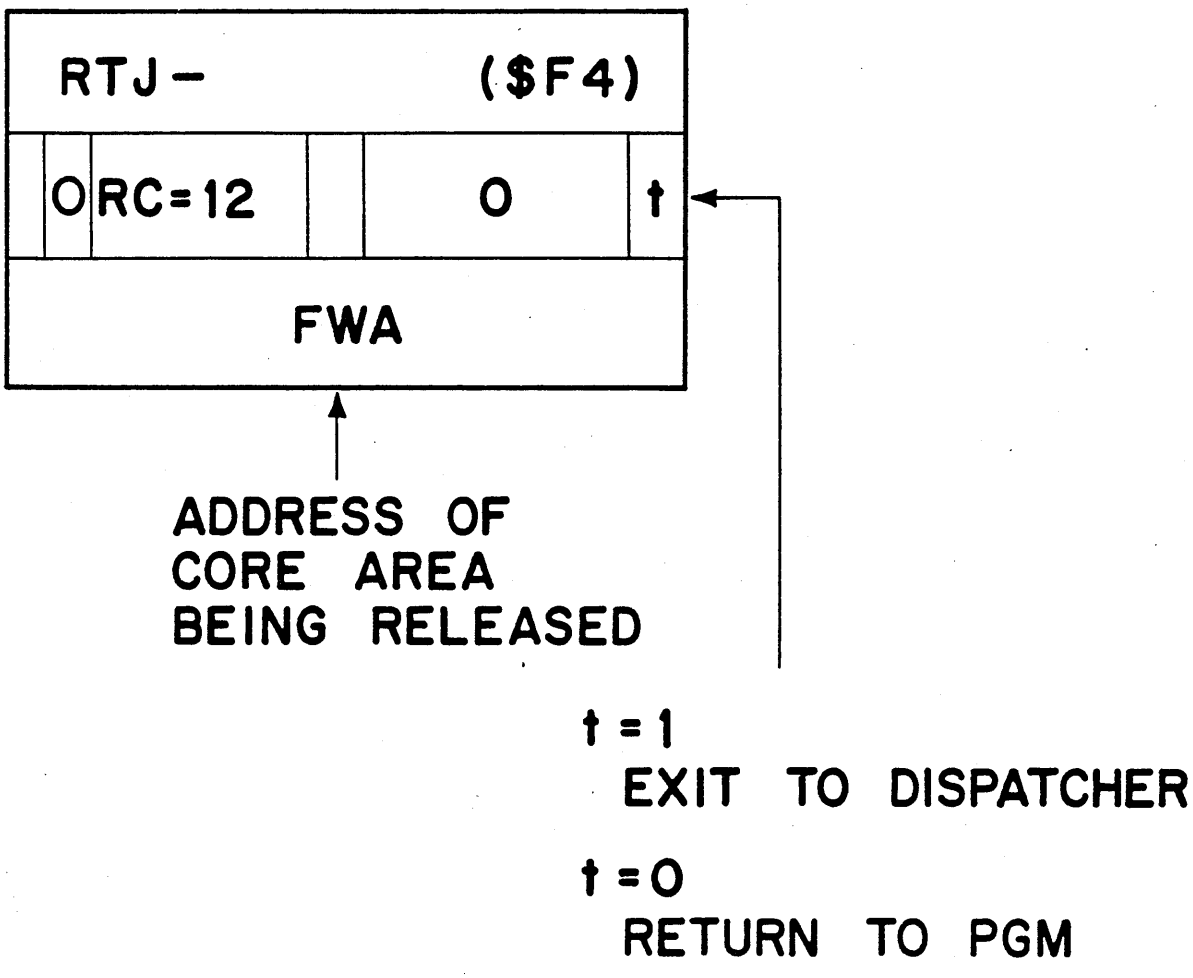
If $C_{15} = 1$ and X = 0, then $C_{14} - 0$ is a 15-bit relative address which when added to the address of the parameter list gives the core address of the area being returned. (Relative, direct)

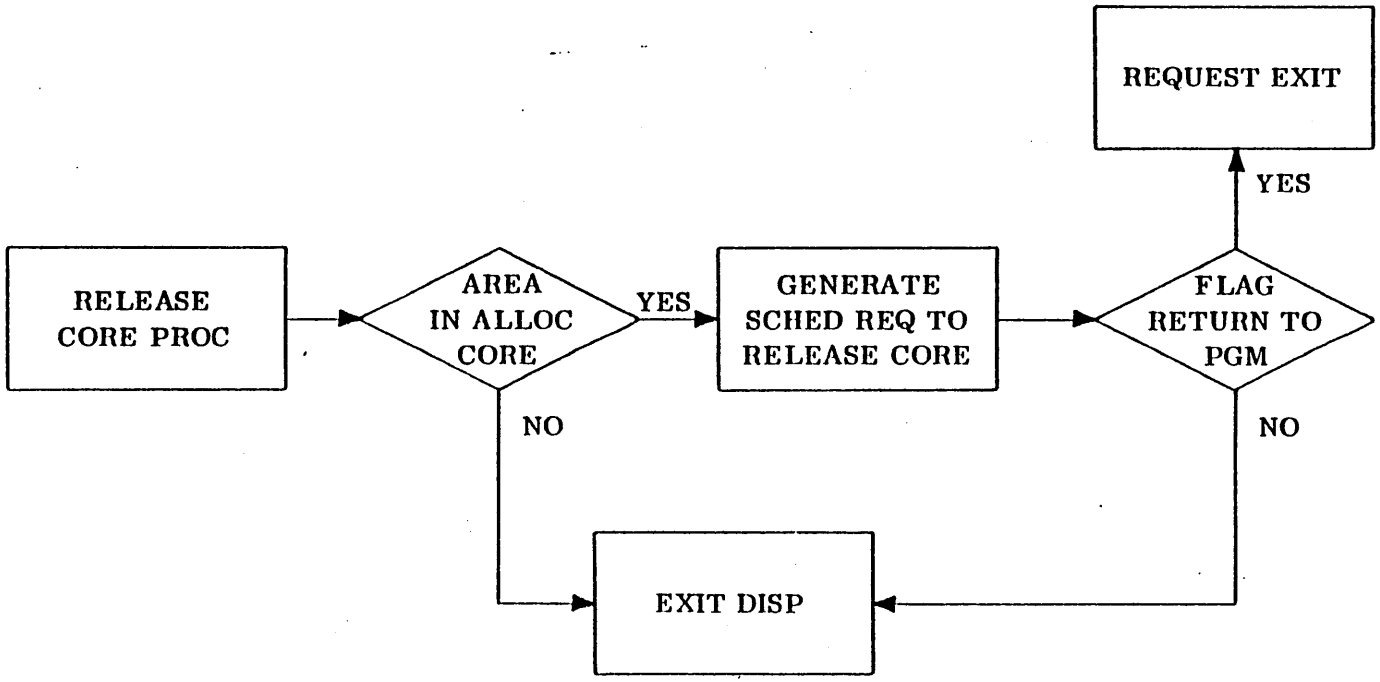
Note that relative indirect is <u>not</u> allowed.

Notes on returning core:

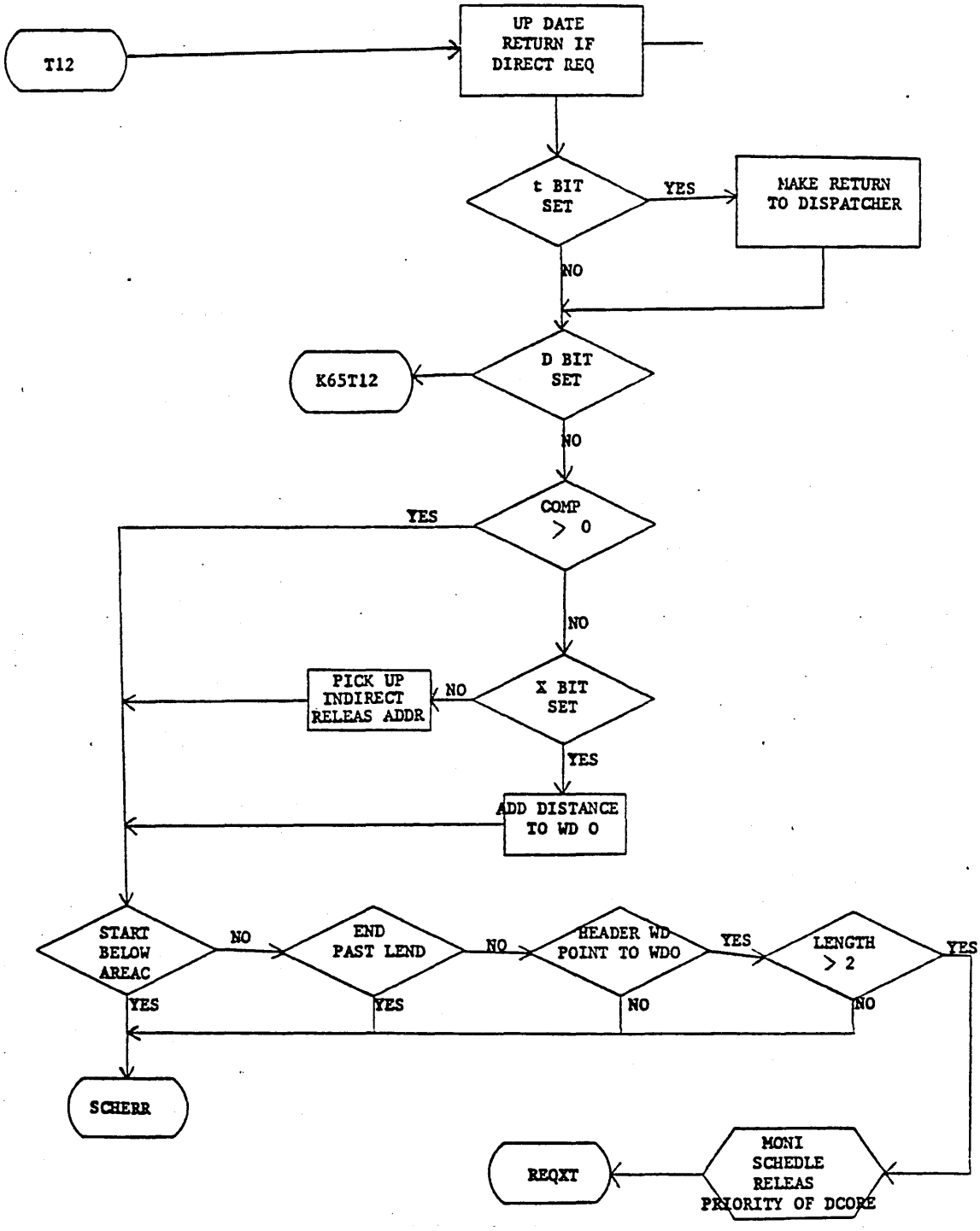
User programs must return each piece of core which they have been allocated. Otherwise the piece of core will remain allocated indefinitely. Each piece must be returned once only.

RELEAS REQUEST




RELEASE CORE PROC
AREA IN ALLOC CORE
YES
GENERATE SCHED REQ TO RELEASE CORE
FLAG RETURN TO PGM
YES
REQUEST EXIT
NO
NO
EXIT DISP

RELEASE REQUEST PROCESSOR ENTERED AT PRIORITY OF CALLER



A check is made to determine if the area of core being returned belongs to the allocatable area. If the area of core being returned is outside the allocatable area, then the request is ignored and control does not come back to the user, but instead goes to the Dispatcher. Using this feature all programs, whether mass memory or core resident, can be written identically. At the end of a program, the RELEAS request is made with R, the return indicator, set to zero, and C specifying the start of the program. For core resident programs no core is returned and control goes to the dispatcher. For mass memory resident programs, the core is returned and control is given to the dispatcher. The coding for both core resident and mass memory resident routines is the same.

SPACE REQUEST PROCESSOR

The SPACE Request Processor is entered in the same manner as the R/W Processor. Its purpose is to set necessary parameters (logical unit number, etc.) so that the R/W Processor can complete processing of the request. In addition, this processor contains the block of core controlled by the Core Allocator and the restart program.

EXTERNAL SYMBOLS

CKTHRD	Routine in R/W Processor which checks for non-zero thread.
SAVLU	Location in R/W Processor to which the SPACE Request Processor exits.
RPMASK	Mask for request priority.
IDLE	The level -1 idle loop.

INTERNAL SYMBOLS

AVCORE	Size of the allocatable core area.

RESTART ROUTINE

Since this program is operated once immediately after AUTO LOAD, it is located in the block to be controlled by the core allocator.

It is entered via the following procedure when the system is on mass storage.

1. MASTER CLEAR the machine.
2. Depress the AUTO LOAD button on the mass storage device.
3. Depress the RUN switch. This causes the machine to execute a program which reads the resident portion of the system from mass storage. When this is done, the program jumps to the address specified in location 1, which is the address of the restart program.

The restart program performs the following operation before jumping to the idle loop.

1. Protects all locations which must be protected and unprotects all others.
2. Enables the timer interrupt and initiates the diagnostic timer if present.
3. Requests that the protect switch be activated.

The 1573 LINE SYNCH.

Timing Generator (timer) is assumed to be interfaced via a 1750 Data and Control Terminal (DCT) that is assigned to Equipment No. 8. It is started by an output with $A=A000_{16}$ and $Q=0400_{16}$. If this output results in a reject, the following message will be printed on the output comment device:

TIMER RJ

This message will occur if the Timer is not present or if the 400hZ power supply is switched off or the equipment code assigned to the DCT is not 8.

The message SET PROGRAM PROTECT is then typed to request that the operator set the protect switch to ON.

This module can be used to replace SPACDR and Core Allocator with the savings of approximately 350 cells.

Certain restrictions are attendant on the use of SUBCOR.

1. No swapping is available.

2. RELEAS requests must be given in an order precisely in reverse of the allocations.

3. A request for space which exceeds the limits of allocatable core will never be given. If one is attempted, SUBCOR will hang in a 1 cell loop.

LESSON GUIDE 9

VOLATILE STORAGE

LESSON PREVIEW:

Volatile storage assignment will be discussed.

REFERENCES:

Listings of SYDAT, ALVOL, and OFVOL.

TRAINING AIDS:

PROJECTS:

OBJECTIVES:

At the completion of this lesson, the student will be able to:

1. Discuss volatile storage assignments.
2. Understand the function of VOLBLK.
3. Trace events in ALVOL and OFVOL.

VOLATILE STORAGE ASSIGNMENT

Volatile storage (VOLBLK) is the storage area located in SYSDAT that is reserved for the allocation of small blocks of data storage for reentrant routines.

Volatile storage is available only to protected programs. At least three locations must be requested and all system interrupts disabled prior to entry at VOLA and VOLR.

The volatile storage area acquired must be released at the same priority level at which it was acquired. The requesting program and any accompanying program sequence must not go to the dispatcher prior to the release of the volatile storage area.

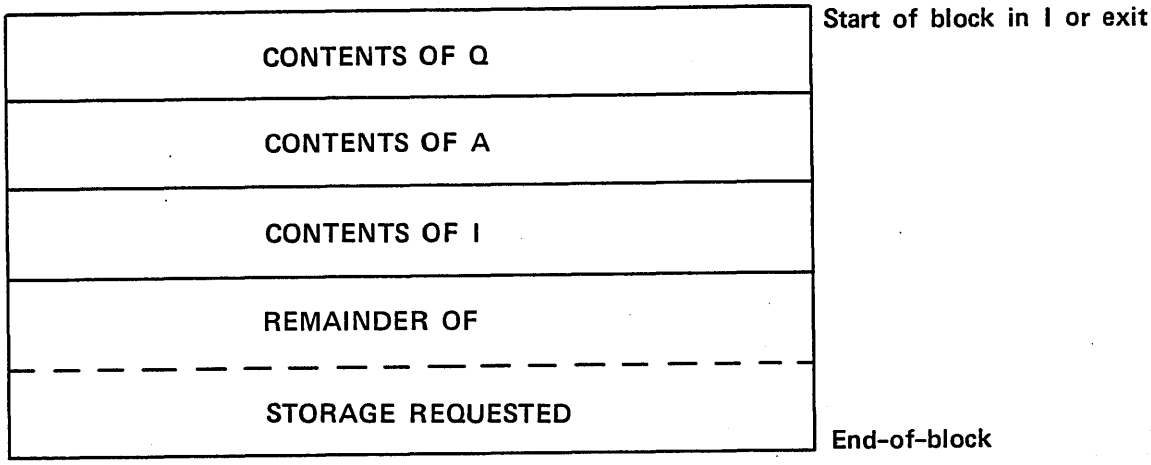
A request for more volatile storage than is available constitutes a catastrophic condition. The volatile storage assignment program enters OVFVOL with the following in the A and Q registers:

- A Amount of overflow in words
- Q Base address of the interrupt stack

OVFVOL clears the M register and writes OV on the comment device. No further action can be taken and the system hangs ($18FF_{16}$ instruction). The OV error is caused by incorrect set-up or use of the system.

A block of storage is assigned with the entry point VOLA and released with the entry point VOLR. Both entry points are entered by an RTJ with interrupts inhibited.

On the entry to VOLA, the block size is contained in the word following the RTJ. VOLA assigns specified locations and fills the first three locations of the block with the contents of Q, A, and I as follows:



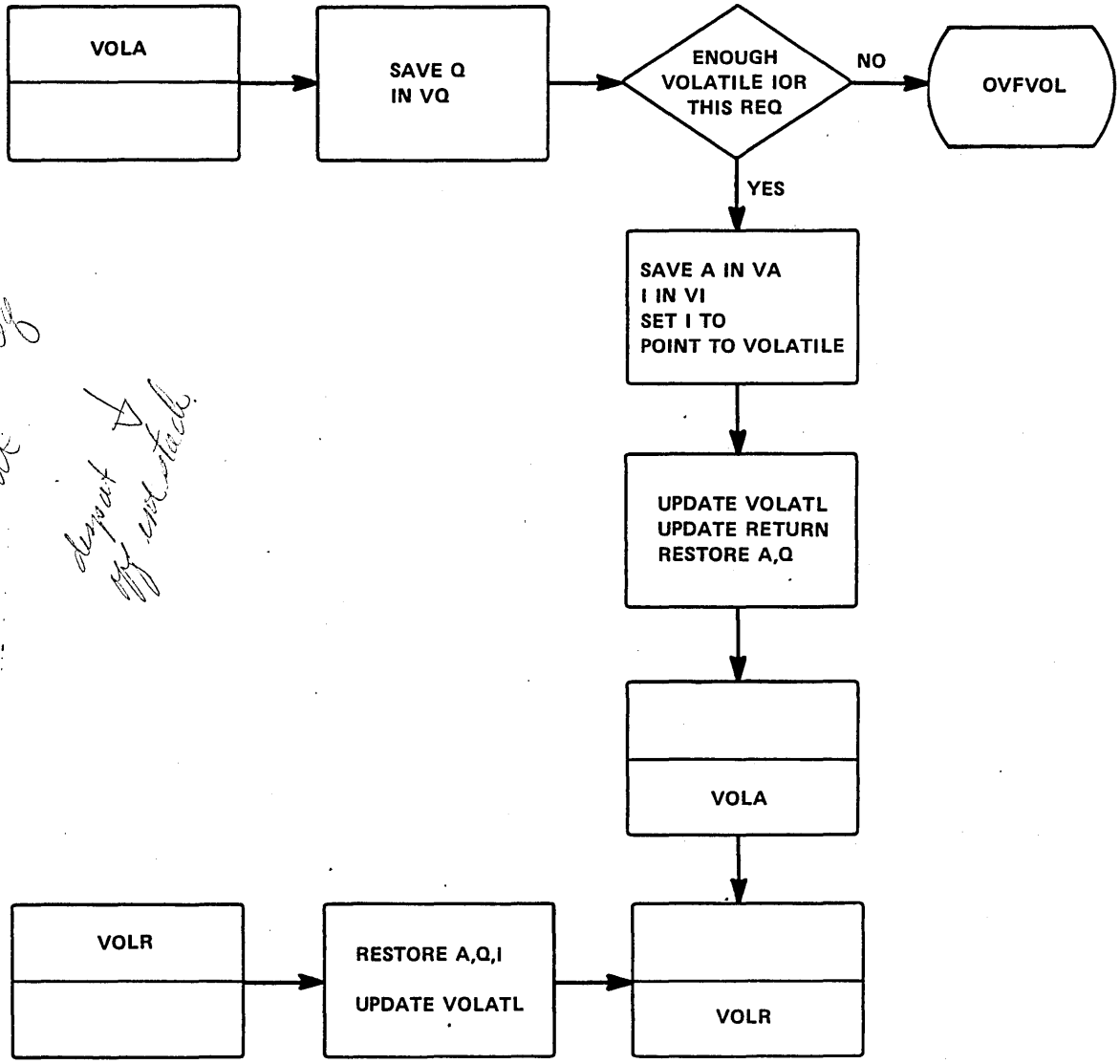
On exit from VOLA, the I register contains the address of the start of the assigned block.

On return from VOLA, a block of eight volatile storage locations has been assigned and words O through 2 have been filled. The program stores word 3 and later uses the remaining words.

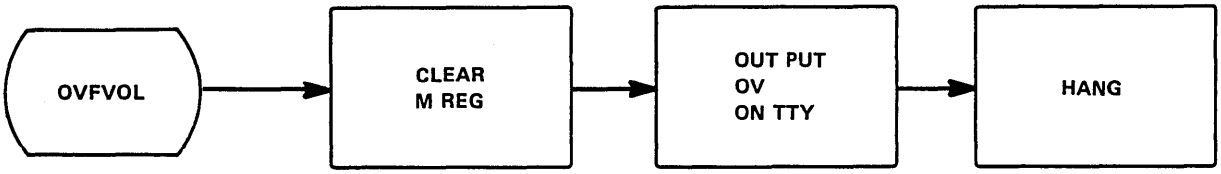
Location	15 4	3	2	1	0
LOC + 0	ORIGINAL CONTENTS OF Q	0	0	1	0
1	ORIGINAL CONTENTS OF A	0	0	0	1
2	ORIGINAL CONTENT OF I	0	0	1	1
3	RETURN ADDRESS (SAVED BY REQUESTING PROGRAM)				
4	TEMPORARY STORAGE				
.					
.					
7					

The I register contains the core location represented by LOC. The contents of A and Q are the same as an entry to VOLA. On entry to VOLR, I must contain LOC. On return from VOLR, the eight locations of volatile storage have been released. The contents of the A, Q and I registers are replaced with the contents of the first three locations of the released block.

ALVOL


VOLA
SAVE Q
IN VQ
ENOUGH
VOLATILE IOR
THIS REQ
NO
OVFVOL
YES
SAVE A IN VA
I IN VI
SET I TO
POINT TO VOLATILE
UPDATE VOLATL
UPDATE RETURN
RESTORE A,Q
VOLA
VOLR
RESTORE A,Q,I
UPDATE VOLATL
VOLR

OFVOL


OVFVOL
CLEAR
M REG
OUT PUT
OV
ON TTY
HANG

LESSON GUIDE 10

TIMER PACKAGE

LESSON PREVIEW:

The TIMER requests and DIAGNOSTIC TIMER of the TIMER Package will be discussed.

REFERENCES:

Listings of TMINT and TIMER

TRAINING AIDS:

PROJECTS:

OBJECTIVES:

To study all the functions and programs of the TIMER package.

TIMER REQUEST

RTJ - ($F4)

	RC=8		TYPE OF UNITS	CP
COMPL				
NUMBER OF UNITS				

TIME IS MEASURED IN UNITS (COUNTS)

0 - BASIC COUNT (60/CPS)

1 - 1/10 SEC (6 X BASIC)

2 - SEC (10 X 1 CT)

3 - MIN (60 X 2 CT)

TIMED INTERRUPT 1/60 SEC TO 32,768 MIN

TIMER PACKAGE

COMPONENTS

The TIMER package is made up of two modules:

TIMER PACKAGE
DIAGNOSTIC TIMER

The former processes TIMER requests, timer interrupts and delay expiration. The latter processes I/O hangups.

EXTERNAL SYMBOLS USED BY TIMER PACKAGE

SCHERR Used to exit if the schedule stack is full

TIMACK Acknowledge code for time interrupts

EXTERNAL SYMBOLS USED BY DIAGNOSTIC TIMER

The starting address label for each PHYSTB entry, to be interrogated by this module, is declared as an external symbol.

TIMER REQUEST PROCESSING

Entry Interface

Entered from the monitor entry for requests via a jump. "I" contains location of volatile, and "A" contains location of the request.

Exit Interfaces

Exit is made to SCHERR if no schedule stack space remains open. Exit is made to request exit after the request has been added to an appropriate stack.

Internal Operation

On entry, the request processor translates the completion address and attempts to fill an empty schedule stack entry with a SCHDLE request at the level specified in the TIMER request. If no empty exists, exit is made to SCHERR.

The newly filled schedule stack entry is then threaded to one of 4 lists, depending on the "U" parameter. The caller's delay time is added to the stack entry as the "Q" parameter. Exit is then made to the request exit.

TIME INTERRUPT AND EXPIRATION PROCESSING

After the interrupt is acknowledged, each of the counters for the 4 lists are examined to see if one count for that list has expired. If not, the respective count is decremented and exit is made to the dispatcher. If the count is expired, it is reset and the threaded list corresponding to that counter is examined. The delay in each member of the list is decremented. Those delays which are decremented to zero cause SCHDLE requests which result in operation of the concerned program. When this process is complete, the next counter is decremented, etc.

If the acknowledge of the time interrupt is rejected, the program will exit to the dispatcher.

DIAGNOSTIC TIMER OPERATION

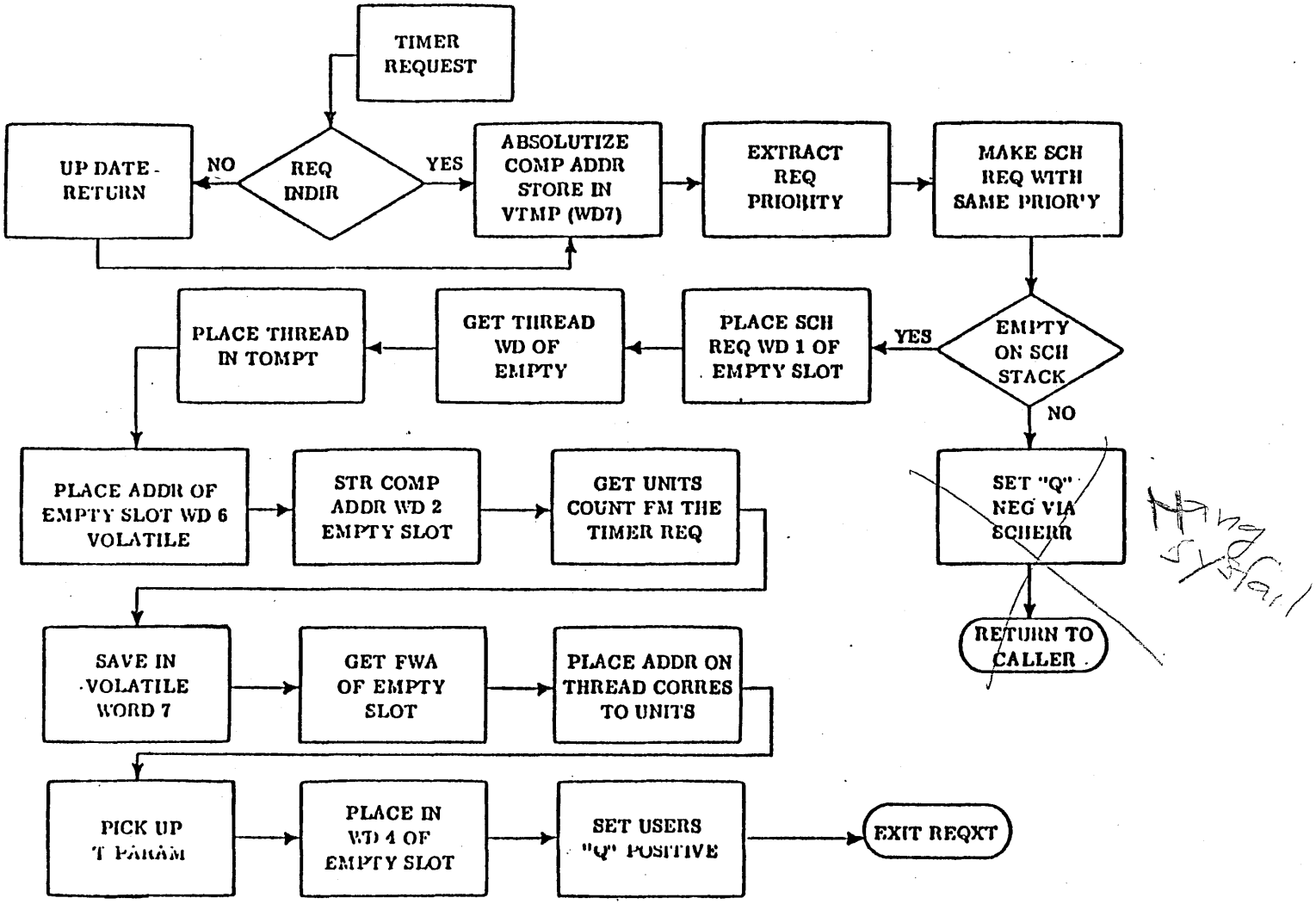
This module is operated periodically as the result of a TIMER request generated by itself. The first TIMER request is made in the startup routine at AUTO LOAD time. On entry, this module decrements the clock cell (in PHYSTB) of each non-idle device in the table DGNTAB. If the clock cell becomes minus, the device is assumed to be hung up and the error entry to the driver is scheduled. When this process is complete for each device, the module makes a TIMER request, to cause its next execution, and exits to the dispatcher.

INTERNAL SYMBOLS USED BY THE TIMER PACKAGE

These symbols are defined via EQU pseudo operation and can be easily deduced from the listing.

INTERNAL SYMBOLS USED BY THE DIAGNOSTIC TIMER

EDCLK	Index to diagnostic clock in each PHYSTB entry
EDPGM	Index to location of error routine in each PHYSTB entry
SECOND	Number of timer pulses per second
DELAY	Number of seconds between successive operation of the diagnostic timer
DTVAL	Priority level at which the diagnostic timer operates. (Assembly value is 13).
NUMPU	Number of physical devices.


TIMER REQUEST
UP DATE - RETURN
NO
REQ INDIR
YES
ABSOLUTIZE COMP ADDR STORE IN VTMP (WD7)
EXTRACT REQ PRIORITY
MAKE SCH REQ WITH SAME PRIOR'Y
EMPTY ON SCH STACK
YES
PLACE SCH REQ WD 1 OF EMPTY SLOT
GET THREAD WD OF EMPTY
PLACE THREAD IN TOMPT
NO
SET "Q" NEG VIA SCHERR
RETURN TO CALLER
PLACE ADDR OF EMPTY SLOT WD 6 VOLATILE
STR COMP ADDR WD 2 EMPTY SLOT
GET UNITS COUNT FM THE TIMER REQ
SAVE IN VOLATILE WORD 7
GET FWA OF EMPTY SLOT
PLACE ADDR ON THREAD CORRES TO UNITS
PICK UP T PARAM
PLACE IN WD 4 OF EMPTY SLOT
SET USERS "Q" POSITIVE
EXIT REQXT

LESSON GUIDE 11

LOADER TABLES

LESSON PREVIEW:

This lesson is designed to exhibit the detailed LOADER functions. In addition, the student will be introduced to a relocatable program format.

REFERENCES:

Chapter 12 of MSOS 5 RM

TRAINING AIDS:

PROJECTS:

1. Student Project - 11
2. Study Questions - 11

OBJECTIVES:

At the completion of this lesson, the student will be able to:

1. Understand the LOADER'S FUNCTIONS.

2. Interpret object code.

MAJOR LOADER FUNCTIONS

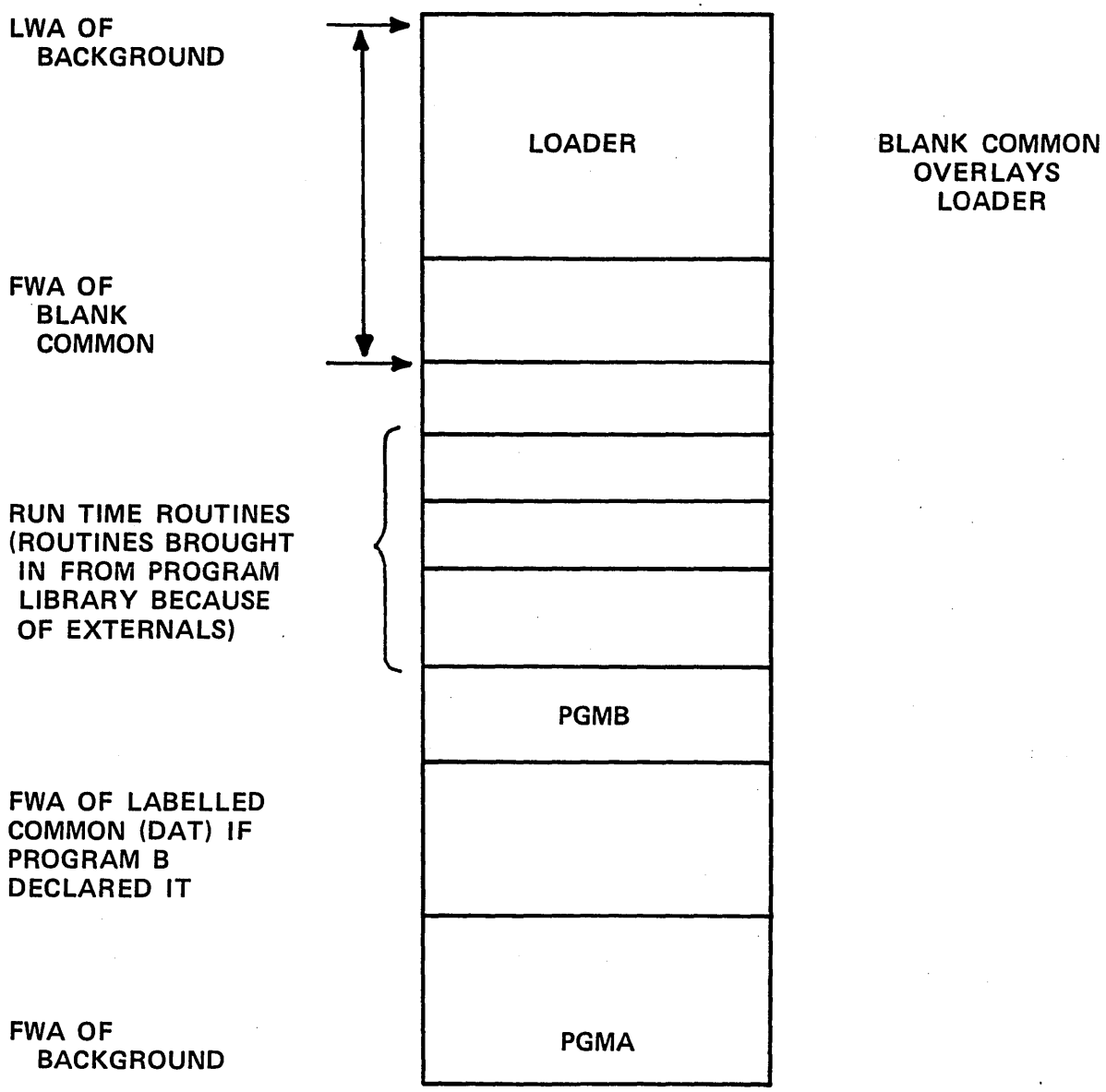
- LOCATES THE PROGRAM TO BE LOADED
- MAKES RELOCATABLE ADDRESSES ABSOLUTE
  - PROGRAM RELOCATABLE
  - BLANK COMMON RELOCATABLE
  - LABEL COMMON RELOCATABLE
- LINKS EXTERNALS
- RECORDS LOAD MAP
- RECORDS ENTRY POINT TABLE
- TRANSFERS CONTROL

MSOS LOADERS

- **BACKGROUND LOADER (*L)** RELOCATABLE & ABSOLUTIZES IT, THEN Transfers control to it
- **LIBEDT LOADER (*P)** RELOCATABLE Programs ONLY
- **SYSTEM INITIALIZER (*L,*LP,*M,*MP)**

loader runs in protected memory
when done IP unprotects the locations
required by loader.

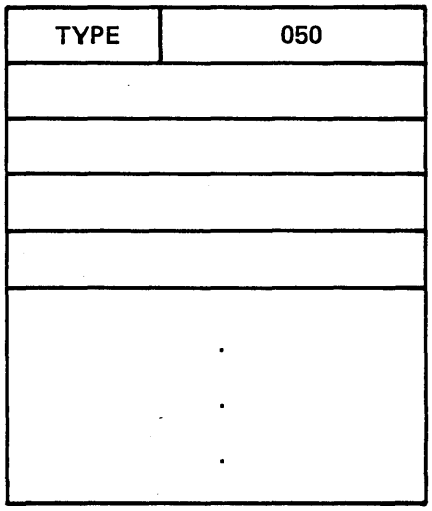
BACKGROUND PROGRAM LAYOUT



RBD command word + 4 words data

LOADER BLOCKS
GENERAL FORMAT

HEADER



MAX. 60 WORDS

TYPE 3 BITS

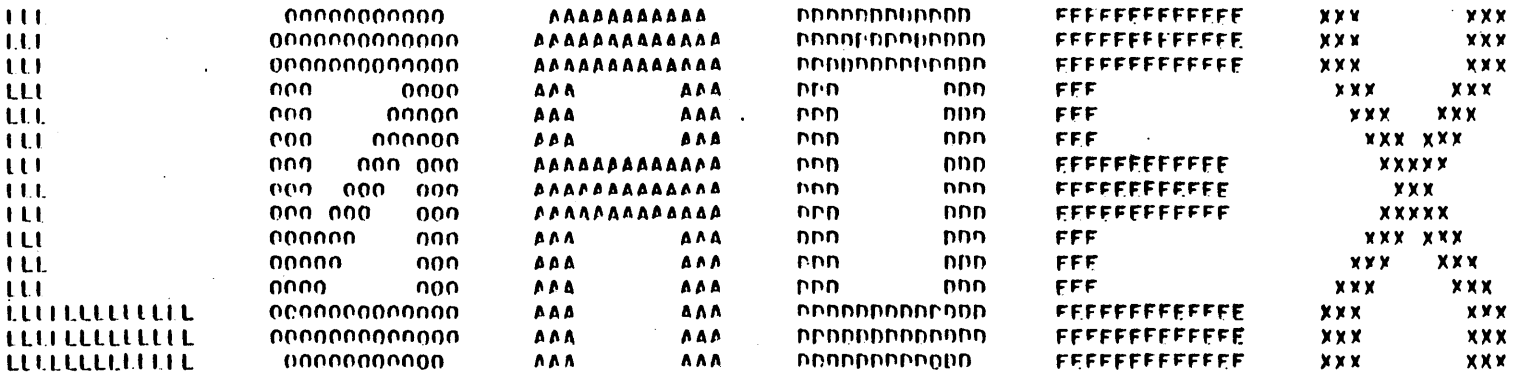
NAM	001	NAME BLOCK
RBD	010	COMMAND SEQUENCE
BZS	011	ZERO STORAGE
ENT	100	ENTRY POINT
EXT	101	EXTERNAL NAME
XFR	110	TRANSFER

only 1
Relocatable Binary Data

only 1 Points to first executable statement in program.

*T TERMINAL LOAD FROM THIS MEDIA

MP → Sub
Sub
Sub
need → PROG
to point here.




```
*K,PB,I 0
*ASSEM
```

```
          DUM                 PAGE    1              DATE: 10/29/78

0001                          NAM  DUM            DUMMY PGM - FIRST PGM WIPED OUT BECAUSE OF 6789
0002 P0000 2020    MSG        ALF  *.  A DO NOTHING PROGRAM*
     P0001 4120
     P0002 444F
     P0003 204F
     P0004 4654
     P0005 4849
     P0006 4F47
     P0007 2050
     P0008 524F
     P0009 4752
     P000A 414D
0003                          END

PGM=  000P (     11)   COM = 0000 (      0)   DAT = 0000 (      0)
```

```
0001                        NAM  MAIN          LOADER BLOCK EXAMPLE
0002                        ENT   MAIN          TELL ASSEM AND LOADER THE SYM MAY BE
0003                *                          REFERENCED FROM OUTSIDE THE PROGRAM
0004                        EXT   CLEAR         TELL ASSEM THAT CLEAR IS DEFINED ELSEWHERE
0005                *
0006                ***     THIS IS THE MAIN PROGRAM
0007                *
0008 P0000 0013   MAIN    END 19              SET UP PARAMETERS IN A AND Q
0009 P0001 C000           LDA   =XL
     P0002 0016 P
0010 P0003 5400 X         PTJ   CLEAR         CALL THE SUBROUTINE
     P0004 7FFF X
0011                *
0012 P0005 0009           END   9             SET UP PARAMETERS IN A AND Q AGAIN
0013 P0006 C000           LDA   =XM
     P0007 002A P
0014 P0008 5400 X         PTJ+ CLEAR          CALL IT AGAIN WITH NEW PARAMETERS
     P0009 0004 X
0015                *
0016                *
0017                *       USE COM AND DAT
0018                *
0019                *
0020        0000 C          COM   FX(10)
0021        0000 D          DAT   FX2(5)
0022                *
0023 P000A 0009           END   9
0024 P000B C000           LDA   =XFX
     P000C 0000 C
0025 P000D 5400 X         PTJ+ CLEAR
     P000E 0009 X
0026                *
0027                *       ANOTHER EXAMPLE OF A CALL TO SUBROUTINE
0028                *
0029                        EXT   CAT
0030                *
0031 P000F C400           LDA+ FX
     P0010 0000 C
0032 P0011 6400           STA+ FX2
     P0012 0000 D
0033 P0013 5400 X         PTJ+ CAT
     P0014 7FFF X
0034 P0015 14FA           JMP- ($FA)
0035                *
0036 P0016 0014           BSS   L(20)*M(10)   THESE ARE THE AREAS THE SUB WILL CLEAR
     P002A 000A
0037                *
0038                        END MAIN

PGM=  0034 (    52)   COM = 000A (    10)   DAT = 0005 (     5)
```


b.

```
CLEAR                PAGE    1              DATE: 10/29/78

0001                          NAM    CLEAR        THE SUBROUTINE TO CLEAR A BUFFER
0002                          ENT    CLEAR        MAKE SYMBOL CLEAR KNOWN TO LOADER
0003                    *
0004                    ***     HERE IS THE SUBROUTINE
0005                    *
0006 P0000 0000    CLEAR   NUM 0                THE RTJ HARDWARE WILL STORE THE RETURN ADDR HERE
0007 P0001 6400            STA* DOIT+1          PUT EWA OF ARRAY IN STA INSTRUCTION
     P0002 0005 P
0008 P0003 0A00             ENA 0                INIT
0009 P0004 6600    DOIT    STA* **,0            PROC  NOTE A TWO WORD INSTRUCTION
     P0005 0000 P
0010 P0006 0DFF             INO -1
0011 P0007 0172            SOM   EXIT           TEST
0012 P0008 1400            JMP* DOIT            TEST
     P0009 0004 P
0013 P000A 1CE5    EXIT    JMP* (CLEAR)         RETURN TO MAIN
0014                       END

PGM=  000B (     11)    COM = 0000 (      0)    DAT = 0000 (      0)
```

```
CAT                  PAGE    1              DATE: 10/29/78

0001                          NAM  CAT            AN EXAMPLE OF A SUBROUTINE
0002                           ENT CAT
0003 P0000 0000    CAT         NUM 0
0004                          EWRITE       9,COMPL,MSG,LENGTH,A,0,1,,,1
0004 P0001 54E4
0004 P0002 4C01
0004 P0003 0009 P
     P0004 0000
0004 P0005 1009
0004 P0006 0008
     P0007 000A P
0005 P0008 14EA            JMP- (SEA)
0006 P0009 1CE6    COMPL   JMP* (CAT)
0007 P000A 204A    MSG      ALF *, JUST AN EXAMPLE*
     P000B 5553
     P000C 5420
     P000D 414E
     P000E 2045
     P000F 5841
     P0010 4D50
     P0011 4C45
0008          000B          EQU LENGTH(*-MSG)
0009                        END

PGM=  0012 (     18)    COM = 0000 (      0)    DAT = 0000 (      0)
```


C,

```
#L.P
     DUM       7000     DUMMY PGM - FIRST PGM WIPED OUT BECAUSE OF ZTP
     MAIN      7010     LOADER BLOCK EXAMPLE
     CLEAR     7046     THE SUBROUTINE TO CLEAR A BUFFER
     CAT       704E     AN EXAMPLE OF A SUBROUTINE
#Y
```

```
ENTRY POINT TABLE -
    ###CON   AFE6
    ###DAT   700P
    CLEAR    7044     MAIN     7010     CAT      704E
```

```
 JUST AN EXAMPLE
0001     3200
```

← FROM ODEBUG

↑ contains sector address of SCRATCH

DUMP of CORE

```
7000    2020  4120  444F  204E  4F54  4A49  4F47  2050
7008    524F  4752  4140  0000  18F9  C000  0030  1A01
7010    0C13  C000  7026  5400  7044  0C09  C000  703A
7018    5400  7044  0C09  C000  AFE6  5400  7044  C400
7020    AFE6  6400  700P  5400  704F  14FA  0000  0000
7028    0000  0000  0000  0000  0000  0000  0000  0000
7030    0000  0000  0000  0000  0000  0000  0000  0000
7038    0000  0000  0000  0000  0000  0000  0000  0000
7040    0000  0000  0000  0000  701E  6400  7049  0A00
7048    6600  AFE6  0DFF  0172  1400  7048  1CF5  7025
7050    54F4  4C01  705B  0000  1009  000B  7059  14FA
7058    1CF6  204A  5553  5420  414F  2045  5841  4D50
```


d.

Mass Memmory Dump

Loader block general format next 4 pp

relocation bytes

FW manual

```
AFE6    0000  0000  0000  0000  0000  0000  0000  0000
AFFF    0000  0000
     0000  32CD  ← sector #
     2A54  0000  0000  0000  0000  0000  0000  0000
     0000  0000  0000  0000  0000  0000  0000  0000
     0000  0000  0000  0000  0000  0000  0000  0000
     0000  0000  0000  0000  0000  0000  0000  0000
     0000  0000  0000  0000  0000  0000  0000  0000
     0000  0000  0000  0000  0000  0000  0000  0000
     0000  0000  0000  0000  0000  0000  0000  0000
     0000  0000  0000  0000  0000  0000  0000  0000
     0000  0000  0000  0000  0000  0000  0000  0000
     0000  0000  0000  0000  0000  0000  0000  0000
     0000  0000  0000  0000  0000  0000  0000  0000
     0000  0000  0000  0000  0000  0000  0000  0000
     0000  32CE
     4050  1000  0000  2020  4120  444F  0000  204E
     4F54  4849  4F47  0008  2050  524F  4752  414D
     0000  0000  0000  0000  0000  0000  0000  0000
     0000  0000  0000  0000  0000  0000  0000  0000
     0000  0000  0000  0000  0000  0000  0000  0000
     0000  0000  0000  0000  0000  0000  0000  0000
     0000  0000  0000  0000  0000  0000  0000  0000
     0000  0000  0000  0000  0000  0000  0000  0000
     0000  0000  0000  0000  0000  0000  0000  0000
     0000  0000  0000  0000  0000  0000  0000  0000
     0000  0000  0000  0000  0000  0000  0000  0000
     0000  0000  0000  0000  0000  0000  0000  0000
     0000  32CF
     C050  2020  2020  2020  0000  0000  0000  0000
     0000  0000  0000  0000  0000  0000  0000  0000
     0000  0000  0000  0000  0000  0000  0000  0000
     0000  0000  0000  0000  0000  0000  0000  0000
     0000  0000  0000  0000  0000  0000  0000  0000
     0000  0000  0000  0000  0000  0000  0000  0000
     0000  0000  0000  0000  0000  0000  0000  0000
     0000  0000  0000  0000  0000  0000  0000  0000
     0000  0000  0000  0000  0000  0000  0000  0000
     0000  0000  0000  0000  0000  0000  0000  0000
     0000  0000  0000  0000  0000  0000  0000  0000
     0000  0000  0000  0000  0000  0000  0000  0000
     0000  0000  0000  0000  0000  0000  0000  0000
     0000  32D0
     2050  000A  0005  0034  4D41  494F  2020  0000
     0000  0000  2020  4C4F  4144  4552  2042  4C4F
     434B  2045  5841  4D50  4C45  2020  2020  2020
```

0000 0000 0000 1000 ↓ last one on the card

XFER CARD SYMBOLIC NAME

```
2020  2020  2020  2020  2020  2020  2020  2020
2020  2020  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  32D1
4050  1001  0000  0C13  C000  0016  0000  5400
7FFF  0CC9  C000  1010  002A  5400  0004  0C09
0201  C000  0000  5400  0009  0203  C400  0000
6400  0000  0080  5400  7FFF  14FA  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  32D2
8050  4D41  494F  2020  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  32D3
A050  4341  5420  2020  0014  434C  4541  5220
000F  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  32D4
C050  4D41  494F  2020  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
0000  0000  0000  0000  0000  0000  0000  0000
```

e,

```
0000 0000 0000 0000 0000 0000 0000 0000
0000 32D5
2050 0000 0000 0000 434C 4541 5220 0000
0000 0000 2020 544F 4520 5355 4252 4F55
5449 4E45 2054 4F20 434C 4541 5220 4120
4255 4646 4552 2020 2020 2020 2020 2020
2020 2020 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 32D6
4050 1001 0000 0000 6400 0005 0010 0A00
6600 0000 0DFF 001A 0172 1400 0004 1CF5
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 32D7
8050 434C 4541 5220 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 32D8
C050 2020 2020 2020 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 32D9
2050 0000 0000 0012 4341 5420 2020 0000
0000 0000 2020 414E 2045 5841 4D50 4C45
2045 4620 4120 5355 4252 4F55 5449 4E45
2020 2020 2020 2020 2020 2020 2020 2020
2020 2020 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
```

```
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 32DA
4050 1000 0000 0000 54E4 4C01 1000 0000
0000 1000 0000 1000 0000 14E8 1CF6 2048
0000 5553 5420 414E 2045 0080 5841 4D50
4C45 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 32DB
8050 4341 5420 2020 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 32DC
C050 2020 2020 2020 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 32DD
2050 FFFF 7160 0FCF 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
0000 0000 0000 0000 0000 0000 0000 0000
```

STUDENT PROJECT - 11

Using the dump, answer the following questions:

1. Draw the core layout after the programs are loaded.
2. How many programs are included in the relocatable binary file?
3. How many RBD blocks were needed for the first program?
4. What are the names of the externals referenced in the program?
5. What is the transfer address and where is it in the program?

STUDY QUESTIONS - 11

1. What is an unsatisfied external?
2. Where does the background loader search for externals and in what order?

   Where does the LIBEDT loader search for externals?
3. How do you detect a LOADER error?
4. Where are the LOADER BLOCKS created?
5. Can the LOADER be called from foreground?

LESSON GUIDE 12

DEBUGGING/TRACING PROCEDURES

LESSON PREVIEW:

This lesson will outline the CYBER 18 Debugging/Tracing procedures

REFERENCES:

Chapter 10 of MSOS 5 RM

TRAINING AIDS:

PROJECTS:

OBJECTIVES:

At the completion of this lesson, the student should be able to analyze a system dump for effective debugging.

PROCEDURE	REASON
1) Do not "Master Clear."	This would clear all registers and interrupts that are currently true.
2) Set the step/run switch to step.	This will halt the main frame but it will not destroy the registers.
3) Save contents of the "P" register.	This will contain the address of the next instruction to be executed.
4) Display the M-Register (MASK) and save the contents.	This will show what interrupt lines are enabled and disabled.
5) Sweep memory location EF_{16} (Current Priority Level) and save the contents.	This is the current software priority unless some program is storing into this location.
6) Sweep memory location $B8_{16}$ (top of Interrupt Stack) and save contents.	This will contain the address of the "top of the Interrupt Stack." This is a push-down pop-up pointer.
7) Using the listing of SYSBUF, verify that location ES_{16} contains an address that falls within the BSS black labeled INTSTK. If the interrupt stack overflowed go to step 8; otherwise go to step 12.	This check will verify that the "M" register setting and software priority levels are in parallel if the system is still operational. It is possible for the whole system to be wiped out.
8) The current priority level from step 5 should now be used as an index to the MASKT table found in the SYSBUF listing. The word found should be the same as the M-register from step 4. If it is the same go to step 9; if not, go to step 10.	This step will help to determine if the monitor is possibly wiped out, still in control but partially destroyed, or if there are priority problems. EXAMPLE: $EF_{16} = 6$ $FFFF MASKT $FFFF $FE0F $FEEF $FFFF $E373 $0DFF $0777 $0747 Same O.K. M-Register=0777

A = lowest you want to dump
B = highest

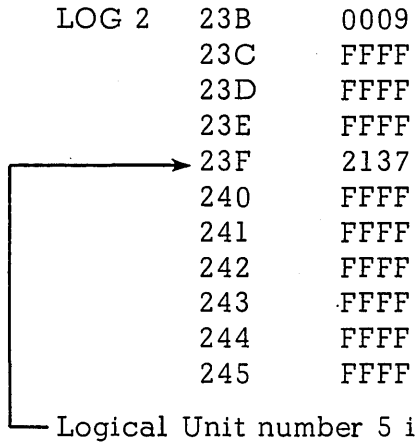
PROCEDURE	REASON
9) The SYSBUF listing is needed. The problem is almost certain to be in the Interrupt trap Region and the MASKT table. The priority level for each line number is declared in the third memory location for a four word group starting at location $100 and ending at location $13F. Using these words as indices to the MASKT table, verify that the bit number corresponding to the line number is a "1" for all priorities lower and a "0" for all equal and above. Correct any error and test again. FINISH.	In this example line 1 interrupt was enabled at its running priority, thus allowing a priority 10 interrupts to interrupt a priority 10 program which is not correct. "A" should be $0005. Line 0 100 XXXX MASKT $FFFF 101 XXXX +1 $FFFF 102 000F +2 $FFFF 103 XXXX +3 $FFFF 104 XXXX +4 $FFFF 105 XXXX +5 $FFFF 106 000A +6 $0777 107 XXXX +7 $0848 108 XXXX +8 $0747 109 XXXX +9 $0047 10A 000D +A $0007 10B XXXX +B $0005 +C $0005 +D $0001 +E $0001 +F $0001 $0000
10) The address of the word found in step 9. Once that address is calculated, sweep the contents of that memory location to verify that it is the same as the M-Register setting. If it is not, go to step 11, otherwise go to step 19.	P008B NUM $777 201 +8B 28C and location 28C = 744 OK 28C = 744 Error
11) There is not much to go on at this point as it is apparent that the executive system is no longer in control. The MASKT table is either partially wiped out or completely changed and some module has executed an illegal instruction. An attempt to find the problem could be made by going to step 19 but do not count on too much.	This is bad because core has been changed and illegal instructions have been executed. The interregister instructions where the "M" register is the destination register has been executed. Chances are control has been transferred to some address that contained constants which were executed as instructions. One could run a spot comparison of memory versus what should be in memory to centralize the changed area. This may or may not supply a clue as to the source of the problem.

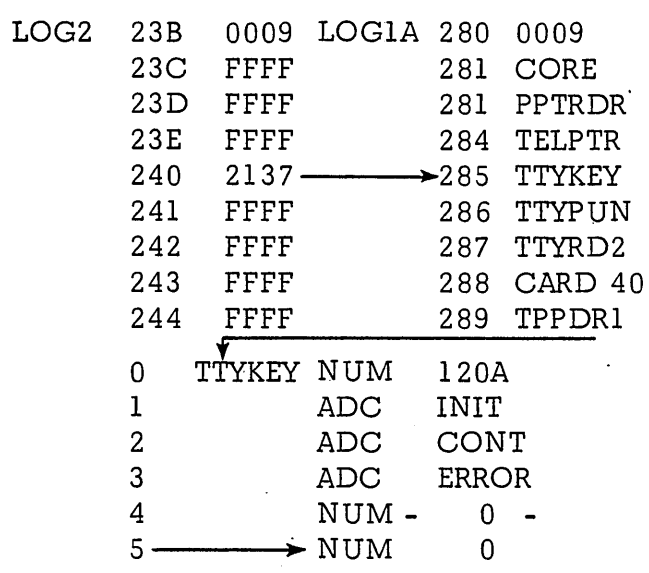
PROCEDURE	REASON
12) Repeat the procedure specified in step 8 only go to step 13 if they are the same; otherwise go to step 14.	This will point out such things as the state of the tables and whether the monitor is in Control.
13) There is really nothing to do here but the system appears to be in good shape for debugging. Go to step 19.	The tables appear to be intact and the monitor still appears to be in control. The problem should be found without much trouble.
14) Repeat the procedure specified in step 10 only go to step 15 if the M-register compares to core, otherwise go to step 16.	This should supply enough information as to whether or not the monitor is in control. Regardless of the circumstances an attempt to trace the problem or problems will be attempted.
15) Try to find out what program is wiping out the MASKT table. If no logical path is available go to step 19.	All in all things look pretty good. The executive system appears to be in control but some program is storing in the area occupied by the MASKT table. It will be in protected core-so all that is needed is to find it.
16) Compare the contents of the M-register with that of memory. If the MASKT table is in core correctly go to step 17, otherwise, go to step 18.	This will let the analyst know who is in control.
17) Correct the program that is currently in execution as it appears that this program has executed an interregister instruction where the M-register was the destination register. If the solution to the problem is not apparent go to step 19.	The monitor appears to be in control but the M-register has changed.
18) It may be extremely difficult to find the source of the problem as it appears that the tables are wiped out, monitor is not in control and an illegal interregister instruction where M is the destination register has been executed. Spot checking core may help but the system is in pretty bad shape. If nothing else works go to step 19.	Control was probably transferred to some address containing data rather than executable instructions where the data was treated as instructions.

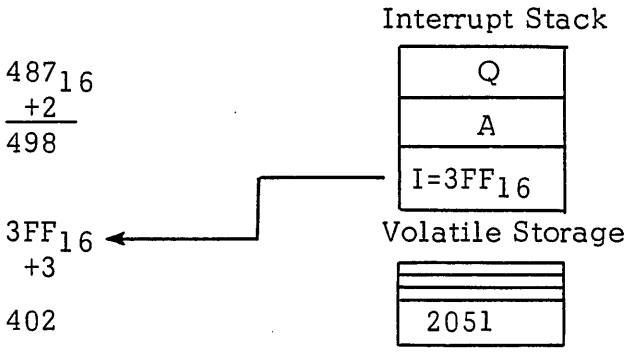
PROCEDURE	REASON
19) Attempt to find out if it is possible that an interrupt is being processed. When the current priority level is the same as a priority level specified in the interrupt trap region (step 9), chances are good that an interrupt is being processed. If it is go to step 20, otherwise go to step 24.	There are several possible trouble spots when processing interrupts but most of them are quite easily detected and they are not usually too difficult to correct. New drivers and physical equipment tables should be looked at quite carefully.
20) Determine what line number interrupt is being processed by using the memory map (SI Listing). Find the program called COMMON (Common Interrupt Handler). The address where common was loaded will contain the address of some location in the interrupt trap region. This address should pinpoint the interrupt line currently being processed.	Most of the interrupt lines use the "Common Interrupt Handler" to preserve the state of the computer and do the housekeeping required to change from one priority to another. When common is bypassed for any line number, the Interrupt Routine used by that line should be interrogated. When the line number is known, the analyst can check the "Interrupt Response Routine" for that line to find out just what devices operate under that line.
21) The absolute address of the "Interrupt Response Routine" will be in the last word of the four word groupings by the line number in the interrupt trap region. This address should point to some address in the SYSBUF. Go to that address (Listings only needed) and acquire the addresses of the Physical Equipment tables of all devices on this line.	This could point to some error conditions such as having the interrupt cabled into the wrong line number or show where the linkage from the interrupt trap region to the Driver for the device is broken. The "Interrupt Response Routine" should contain the "Physical Equipment tables" addresses for all devices processed by the line number.
22) Verify that all of the Initiator priority levels in the "Physical Equipment tables" and the continuator priority level specified in the interrupt trap region are the same. If they are the same, go to step 24, otherwise go to next step.	INTERRUPT TRAP NAME MEMORY CONTENTS Line 01 104 105 54FE priority cont. 106 000A 107 LYNE01 address in response routine

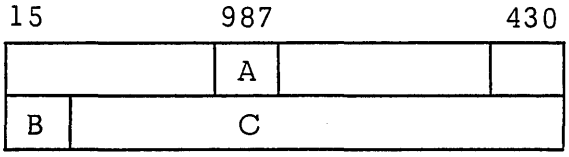
PROCEDURE	REASON
22) (Continued)	INTERRUPT TRAP NAME MEMORY CONTENTS LYNE01 ADC TTY Physical Equipment ADC CRDRD ADC PTREAD ADDR of Table TTY NUM $230A CRDRD NUM 120B Wrong priority should be A (12...)
23) Correct either the priority level in the interrupt trap and/or the priority or priorities in the Physical Equipment tables. Be sure to check the MASKT table if any levels are changed in the interrupt trap and correct accordingly. FINISHED.	The error conditions should be known at this point. Make all the necessary corrections, SYSBUF if required and rebuild system. Attempt run again. This is a common trouble spot when addressing new drivers or changing the priorities of the standard devices.
24) Check to see if the program being executed might possibly be a mass memory resident program or the result of a mass memory program being executed. If it is not mass memory resident go to step 31, otherwise proceed to next procedure.	There are several possible trouble spots that could be caused by Mass Memory programs especially when they have not been tested in a real time environment. For Part O programs, they have address constants preventing the program from being run-anywhere, or releasing allocated core without giving up control etc.
25) The next step is to Dump the Mass Storage Systems Directory. The address to start the dump is the contents of memory location EB_{16} plus the contents of memory location $E7_{16}$. The last address is the contents of location $E6_{16}$.	This should point to where the programs were last loaded. EXAMPLE LOC 14B 000, 332F 000, 0123, 0780, 0000, 0001 152 0010, 221B, 0000, 0157, 0181, 0000, 0020 159 0020, 2003, 0000, 0138, 04C0, 0000, 00025 →Length →Beginning address where loads last time →Current operating priority level

PROCEDURE	REASON
26) Is the program counter pointing to any of the areas in allocatable partition core. If it does not go to step 30, otherwise continue.	Word 1 of the System Directory for each entry will contain the address where control was transferred after the core was allocated. This could show what program is currently being executed.
27) Verify that the program is operating at the priority level assigned. This can be verified by checking Bits 0-3 in the word 0 for the Directory entry currently being checked. If it is okay go to step 29.	The priority level should be checked to the current priority level. If the priorities are the same, there is probably a bug in the program; otherwise the error should be quite simple to trace and correct.
28) Get a listing of the program currently in operation. Check all I/O and Space and PTNCOR Requests, priorities specified for the completion addresses, whether the completion address in any point included in the Mass Storage program being checked. Except for some very special cases the completion priorities should be the same as the priority level in the System Directory and the current running priority. Correct discrepancies and restart.	This is again a common error spot. A program could be initiated at a high priority level, say seven. As the program is being executed, it initiates an I/O request with a completion priority of five. Now the program is running at two different priority levels which could cause some problems.
29) For programs running in allocatable core, check the program for such things as address constants, mode of addressing, or other possible bugs. Correct and reassemble FINISHED.	It appears that the program was not written as a run-anywhere program. When addressing any location in the main program or subprograms the mode must be relative; when addressing permanent core resident programs the mode must be absolute and address constants are taboo unless ADC*
30) It is difficult to say where we are at this time. Tracing through the history of paths taken by the monitor may offer some clue. Possible trouble spots are monitor calls where the mode of addressing is specified incorrectly. The loader has no way of checking these error conditions. Proceed to next step.	EXAMPLES: Relative (Incorrect) RJT - ($F4) NUM - $1305 ADC - PARA Absolute (Correct) RTJ - ($F4) NUM - $1205 ADC - PARA

PROCEDURE	REASON
30) Continued	Sysdir (Correct) (for part O) RTJ - ($F4) NUM - $1205 ADC - PARA Example number 1 is incorrect as the monitor will send control to the address following the return jump and the contents of the next location.
31) The address pointing to volatile storage will be needed to trace the history of the monitor events. The pointer to the next block of Volatile Storage can be found in Memory Location FO_{16} - Save this for future use.	Whenever a request is made to the monitor, the Request Entry Processor will request a temporary storage area called Volatile Storage. This temporary storage area may contain valuable information such as where the call (request) was initiated and where the parameters used by the monitor could be found. This information may point directly to the trouble spot.
32) If there is a possibility of I/O hang-up go to the next step but if it looks as though the problem is definitely software go to step number 37.	It may be possible to determine at this time that there is definitely some problem either Monitor Request or modes of addressing. If that is the case, there is no reason to check for possible I/O hang-up.
33) Find the LOG 2 table in SYSBUF. This table should be dumped to verify that there are no requests waiting to use a particular logical unit. This table is the "top of thread" waiting list for each logical unit. If they are all flagged as empty ($FFFF_{16}$) proceed to step 35, otherwise, continue to next step.	EXAMPLE:  Logical Unit number 5 is threaded, therefore, the device should be marked as busy.

PROCEDURE	REASON
34) The LOG 2 table, as all Logical Unit tables, is ordered by logical unit number. Using the example in step 33, get an address from entry 5 in the LOGIA table. This will contain the address of the Physical Equipment table. Now verify that word 5 in the Physical Equipment table is other than 0. If it is 0 then correct the driver. After the driver goes to "Complete Request" it must again go to FNR before giving up control. It appears that this was not what the driver did.	EXAMPLE: 
35) I/O hang-up possibilities still have not been eliminated. Every Physical Equipment table in SYSBUF will have to be checked verifying that none of the devices are presently busy. If none are busy proceed to step 37, otherwise correct and continue:	Using the drawing on step 34, the addresses of the Physical Equipment tables will be found in the LOG1A. Example CORE, PPTRDR, PPTPCH, and etc., are all absolute addresses of Physical Equipment tables. Each of these addresses +5 will be the busy word and should be zeroes for all devices unless I/O is in process.
36) If the system had the timer but the device was not timed and could be, add the device to the Diagnostic Timer Table. If there was no timer a routine should be written to check for I/O hang-up. Anyway it appears that an interrupt was lost.	When a controller sends an interrupt to the computer which is not retained until the driver acknowledges it or the driver output/input a character and expects an interrupt back when the controller is ready for the next operation but the interrupt does not come back, I/O hang-up is assumed. The device may never be operated again. The diagnostic timer will prevent this but not all devices are timed.

PROCEDURE	REASON
37) The following steps can be used to find out what paths the monitor has taken. First, find out if the last request to start another program (scheduler's call) was requesting that program. This can be verified by checking the last entry in the interrupt stack. If the contents of LOC $B8_{16}$+3 equal an address that contains the same value as location $B9_{16}$ then the last request made was higher. If that was the case go to next step, otherwise go to step 39.	Whenever a program is scheduled up (higher priority than the requesting program), the requesting program is temporarily halted (pseudointerrupt) and the requested program is placed into execution immediately. EXAMPLE: 1) Interrupt Stack Pointer LOC $B8_{16}$ contains 487_{16}. 2) Address of the Request Exit Processor can be found in LOC $B9_{16}$: It contains $107C_{16}$. (LOC $B8_{16}$) Q Value = 487_{16} (Q + 3) A Value = $107C_{16}$
38) Either the program requested was not debugged completely or the absolute/relative indicator (parameter X) in the requesting program was incorrect (most logical). If an error in the requested program is suspected, debug it, otherwise find out from where the request was originated. This will be an extension of step 37. The contents of LOC $B8_{16}$+2 will contain the starting address of the volatile storage used to process this request. That address +3 will contain the return address for the requesting program. With this information the parameters could be verified and corrected if in error. Correct and Restart.	What probably happened was that the address where control was sent was specified as an absolute address when it should have been relative or vice versa. It also may have been a System Directory Call and the program was not on the Directory or vice versa. EXAMPLE: Interrupt Stack Pointer LOC $B8_{16}$ contains 487_{16}. 

PROCEDURE	REASON
38) Continued	204E 204F 2050 2051 54F4 1206 3215 Return Parameter List  A = 0 the C is absolute 1 the C is relative B = 1 Directory
39) The next problem is to find out whether the request was a primary scheduler call (request to start another program) or a Secondary Scheduler call (start execution because of I/O being completed.) This can be checked by comparing the contents of an address calculated by getting the contents of Memory LOC $B4_{16}$+1 to the contents of memory location 104_{16} (Common Exit) if they are the same, the call was a primary scheduler call: otherwise, it was a secondary call. Go to step 40 for a primary call. Go to step 41 for a secondary call.	Top of the Scheduler stack empty list is LOC $B4_{16}$. It contains $4F6_{16}$. 4F6 +1 = 4F7 4F6 4F6 1206 4F7 3215 Common Exit used by monitor is Location 104_{16}. 104 3215 This example shows that the call was a primary call since LOC $4F7_{16}$ LOC contains the same value as LOC 104_{16}.
40) An attempt should be made at this time to find out from where the request was initiated. If there were no requests for volatile storage which would wipe out the history which leads back to the requestor, the procedure will be quite simple: otherwise, take a few stabs in the dark or reinitialize the System. Memory location FO_{16} will contain the pointer to volatile storage.	Pointer to Volatile Storage is location FO_{16}. It contains $3FF_{16}$. Volatile Storage 3FF 400 401 402 2651 204E 54F4 204F 1306 2050 1136 2051 Return

PROCEDURE	REASON
40) Continued The contents of that address +3 could point to the return address following the call. Parameters should be checked for the same error conditions listed for step 38. Correct and Restart.	
41) When tracing down the original requestor for I/O or CORE Allocation there are two paths which may be followed. If there is a listing of Complete Request for Drivers available, look for the Label CE in the assembly listing. Using the memory map along with the listing figure out the absolute address where CE can be found. (It will contain a negative number.) Go to the address specified by bits 0 - 14. This should be checked out. When a listing of Complete Request for Drivers is not available use the same procedure listed under step 40. The address found in location 402 may point to the address following the label defined as CE. Once the parameter list has been found, again check the relative/absolute indicators.	Example - 1: MEMORY MAP COMPLETE REQUEST LISTING PARAME 1329 P0040 NOP VOLA 1387 P0041 RTJ - (AMONI) COMPRQ 13A8 P0042CE 0 0 13A8 +42 $13EA_{16}$ → 13EA A04F Excluding Bit 15 is address $204F_{16}$ RTJ AMONI 204F RC X Example - 2: $F0_{16}$ contains $3FF_{16}$ Volatile 3FF 400 401 402 13EB This address minus 1 should equal CE. For Example 1 from this point.

LESSON GUIDE 13

JOB PROCESSOR

LESSON PREVIEW:

This lesson will introduce the JOB Processor and its related routines.

REFERENCES:

Listings of MINT, JOBPROC, MIPROC, JOBENT, and PARAME

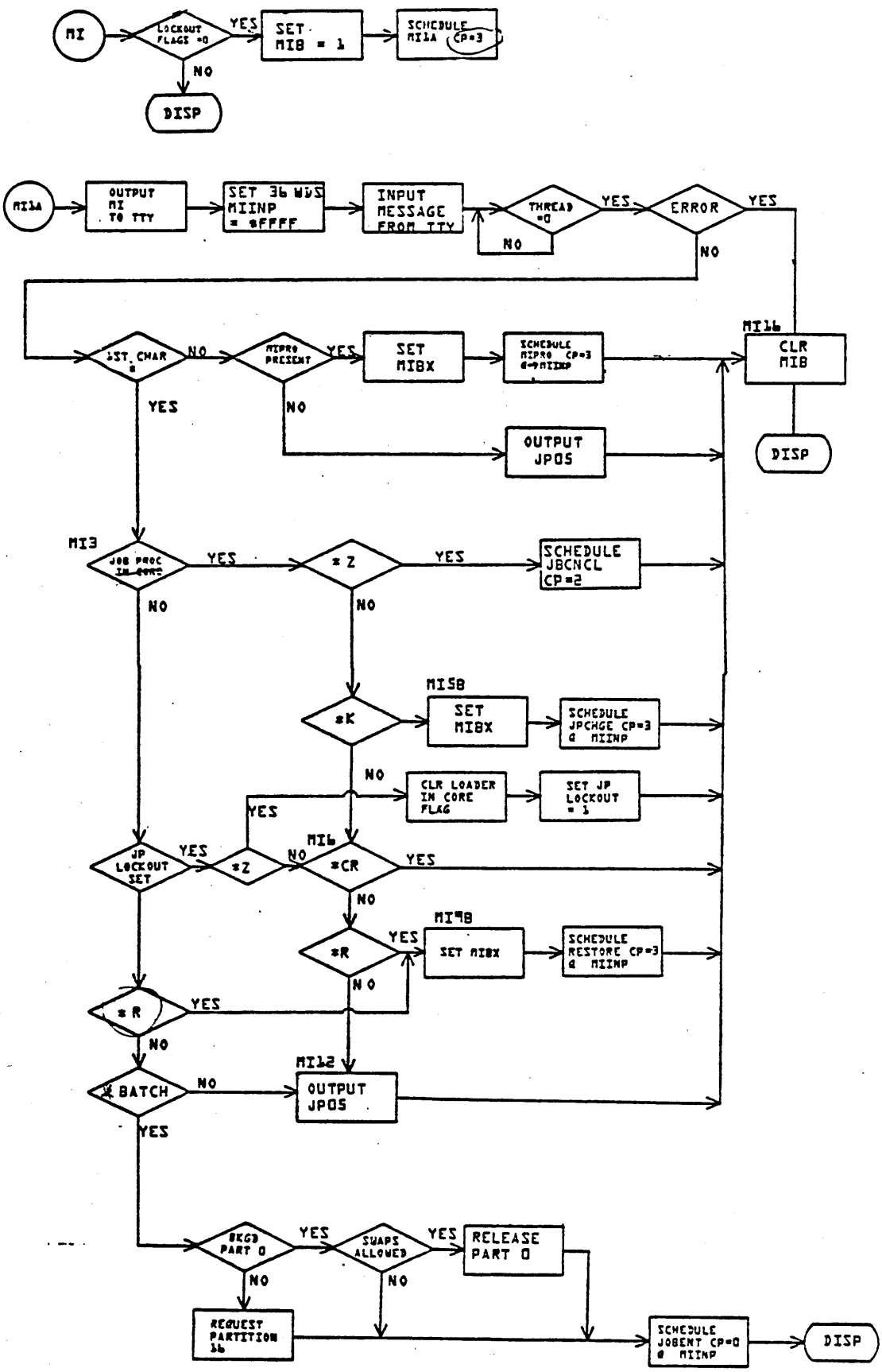
TRAINING AIDS:

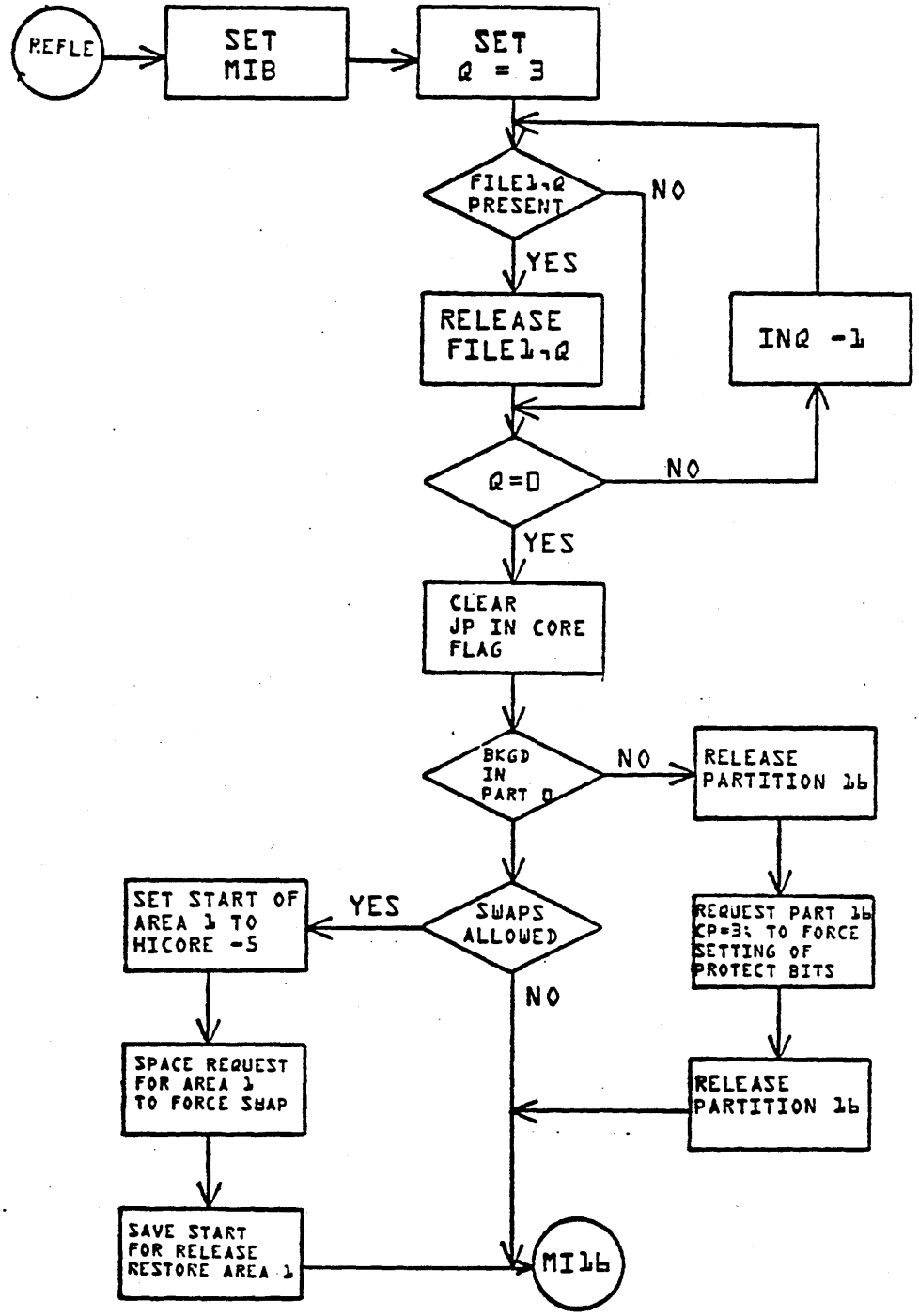
PROJECTS:

OBJECTIVES:

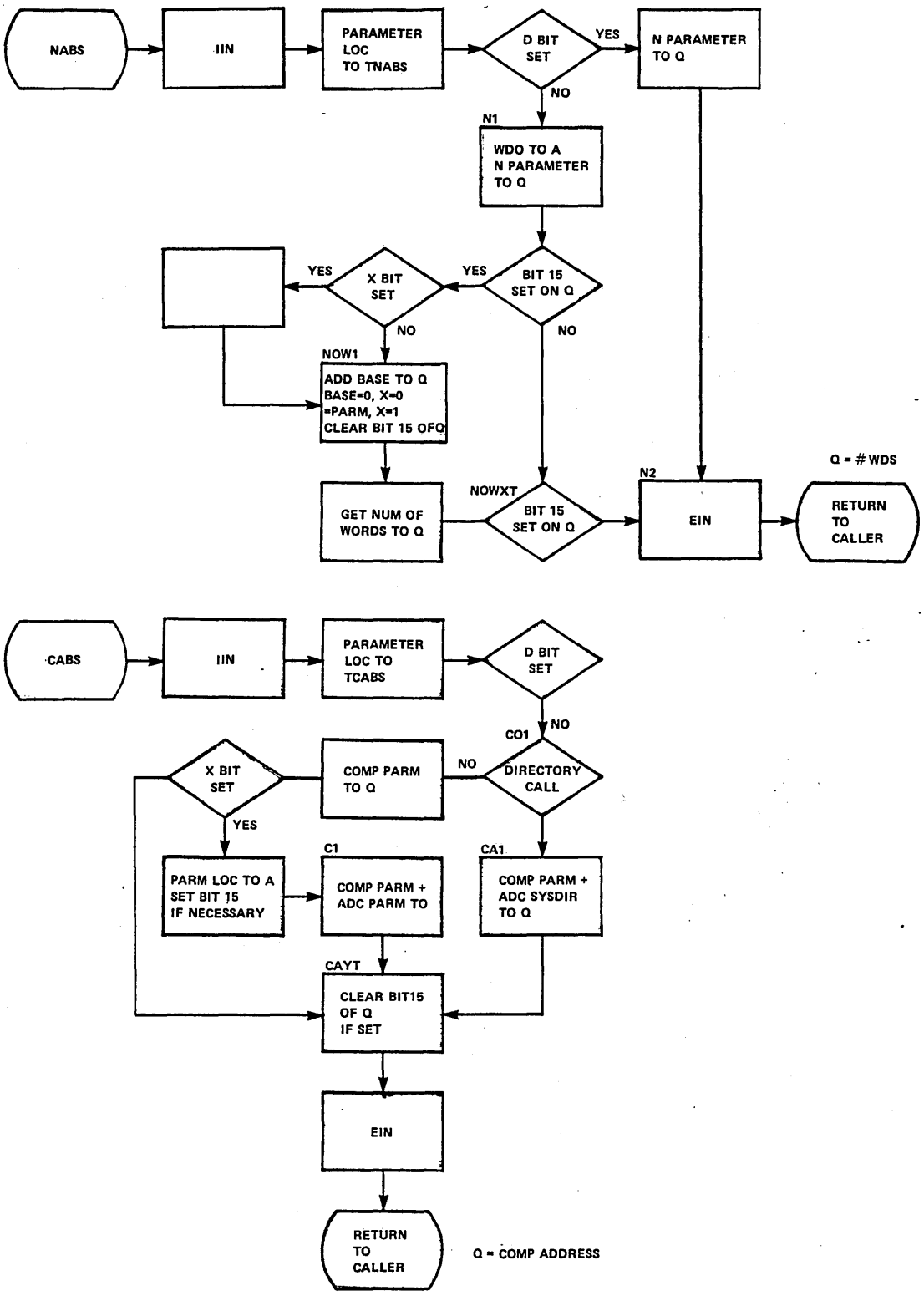
At the completion of this lesson, the student will be able to discuss the Job Processor.

MINT


MI
LOCKOUT FLAG=0
YES
NO
SET MIB = 1
SCHEDULE MI1A CP=3
DISP
MI1A
OUTPUT MI TO TTY
SET 36 WDS MIINP = #FFFF
INPUT MESSAGE FROM TTY
THREAD =0
NO
YES
ERROR
YES
NO
1ST CHAR #
NO
MIPRO PRESENT
YES
NO
SET MIBX
SCHEDULE MIPRO CP=3 @→MIINP
MI1B
CLR MIB
DISP
OUTPUT JPOS
YES
MI3
JOB PROC IN CORE
YES
NO
#Z
YES
NO
SCHEDULE JBCNCL CP=2
MI5B
#K
SET MIBX
SCHEDULE JPCHGE CP=3 @ MIINP
NO
CLR LOADER IN CORE FLAG
SET JP LOCKOUT = 1
YES
MI6
JP LOCKOUT SET
YES
#Z
NO
#CR
YES
NO
MI7B
#R
YES
SET MIBX
SCHEDULE RESTORE CP=3 @ MIINP
NO
#R
YES
NO
MI12
#BATCH
NO
OUTPUT JPOS
YES
REQD PART 0
YES
NO
SWAPS ALLOWED
YES
NO
RELEASE PART 0
REQUEST PARTITION 16
SCHEDULE JOBENT CP=0 @ MIINP
DISP


REFLE
SET MIB
SET Q = 3
FILE1,Q PRESENT
NO
YES
RELEASE FILE1,Q
INQ -1
Q=0
NO
YES
CLEAR JP IN CORE FLAG
BKGD IN PART 0
NO
RELEASE PARTITION 16
SET START OF AREA 1 TO HICORE -5
YES
SWAPS ALLOWED
REQUEST PART 16 CP=3; TO FORCE SETTING OF PROTECT BITS
NO
SPACE REQUEST FOR AREA 1 TO FORCE SWAP
RELEASE PARTITION 16
SAVE START FOR RELEASE RESTORE AREA 1
MI16

PARAME


NABS
IIN
PARAMETER LOC TO TNABS
D BIT SET
YES
N PARAMETER TO Q
NO
N1
WDO TO A N PARAMETER TO Q
YES
X BIT SET
YES
BIT 15 SET ON Q
NO
NO
NOW1
ADD BASE TO Q BASE=0, X=0 =PARM, X=1 CLEAR BIT 15 OFQ
GET NUM OF WORDS TO Q
NOWXT
BIT 15 SET ON Q
N2
EIN
RETURN TO CALLER
Q = # WDS
CABS
IIN
PARAMETER LOC TO TCABS
D BIT SET
NO
CO1
X BIT SET
COMP PARM TO Q
NO
DIRECTORY CALL
YES
C1
CA1
PARM LOC TO A SET BIT 15 IF NECESSARY
COMP PARM + ADC PARM TO
COMP PARM + ADC SYSDIR TO Q
CAYT
CLEAR BIT15 OF Q IF SET
EIN
RETURN TO CALLER
Q = COMP ADDRESS

REVIEW QUESTIONS MSOS

1. The driver's execute at what priority levels? Higher than 7 check int trap region of this line Physlab also
2. At what priority is a mass memory program executed? WHATEVER IT IS SCHED. AT.
3. What happens if an unprotected program executes a release request and a mass memory program is residing in that area? Unprotec. Program ≠ execute release request ∴ probprotect fault JPO xxx
4. How does the R/W request processor determine if a driver is busy? ELM word in Physlab ≠ 0
5. When a driver completes one request, does it jump to the Dispatches? Goes to next request
6. What determines ~~the~~ area in allocatable ~~/~~ partition core that is allocated to a mass memory program that is in the System Directory? Request Priority
7. What is the advantage of having a core resident program in the system directory? If an ordinal (can schedule without knowing its address) you can schedule it by ordinal name & you can change (replace it)
8. If a priority 3 program makes a schedule request for a priority 5 program, who goes on what queue? 3 goes on interrupt stack & priority 5 program runs.
9. What control statement to the system initializer determines the core resident programs? Part 0 = XL
   Part 1 = XLP
10. What determines the priority at which a timer request is threaded into the scheduler stack? Random timer has own que - counts down time
11. How can a program cause itself to run at a lower priority? Schedule itself at a lower priority & JMP to DISPATCHER
12. What program determines the program to be put into execution after a program completes execution? DISPATCHER SELECTS next highest priority program
13. How is the initiator portion of a driver put into execution? SCHEDULED
14. Who releases volatile that MONI obtains? REQUEST EXIT
15. What control statement determine the programs that are to be placed in the system directory? Who receives these instructions? *Y MMAP SYSTEM initializer
    *Y core
16. Who transfers control to the line processor for a line after an interrupt has been generated? Common interrupt handler except for internal interrupt hand which handles line 0
17. How does a completion address for a READ/WRITE request get scheduled? Complete request schedules it
18. How does the diagnostic timer routine know which drivers he is to check? DIAGNOSTIC TIMER TABLE in Sysdat (you make entries)
19. How does the error portion of a driver get into execution? ~~If clock times out~~ diagnostic timer ⇒ control to error address
20. Under what conditions can a swap take place? ~~If you need more allocate~~ If swap ≠ in effect : If priority > 2 : time delay ok since last swap
21. How does the diagnostic timer routine know that a driver is in execution? clock = FFFF / -1 if not in execution / knows in execution if this word is +
22. If DCORE is busy when a space request is made, what happens to the request? ~~lost~~ rescheduled put on allocators thread
    allocator = driver

23. What is the function of the LOGIA table? PHYSTAB ADDRESS

24. What does the "Q" register contain when control is returned from a request processor, if the request was accepted? A POSITIVE # (what you sent to it with bit 15 = SET)

25. How can a programmer determine when his I/O request has been completed?
complete request will zero the thread word or will schedule the completion address

26. If the printer, which is usually logical unit 9 has been declared down, where does the operating system look for an alternate? LOG 1 table

27. Where can you look to determine the number of devices attached to interrupt line 1?
Line 1 table

28. When should a program be written so that it is reentrant? when it istands possibility of being shared at multiple prority levels.

29. Under what condition or conditions will a scheduler request be rejected?
stack full if you have DISABLED SCHEDULED / IF ORDINAL ALREADY SCH.

30. Under what condition or conditions will a READ/WRITE request be rejected?
IF your thread ≠0 / if device is down and there ≠ alternates

31. What determines where control will be transferred after a release request has been executed? T bit set dispatcher
T bit ≠set next word of code. return to caller

32. How does MAKEQ determine whether an I/O request has been completed with errors?
checks error status in phystab – bit 14

33. What bits are set by MAKEQ when an error condition arises?
15/14/13 & ESTAT 1 word.

34. What steps are followed by the scheduler request processor when a program is scheduled at a higher priority than the currently running program?
current guest to int stack / scheduled program starts.

35. How does the system determine the number of logical units?
1st word log 1, log 1A, log 2 tables

36. Who puts information such as S, S+n, and operation in progress in the physical device table, and what programs use this information?
S=Shared bit set up in Systad Phystab

37. In what program do you find the code for the scheduler?
DISPATCHER [NDISP]

38. How many processors are used when a schedule request is made for a mass memory program? schedule / RW / core allocator / RW / MM device / RW when transfer complete

39. Where is Request Exit located?
Monitor [END OF MONITOR Request]

40. Who sets the new mask after an interrupt is generated and how does he determine what that setting should be? Common Int Handler

finally schedule the program with same priority

41. When are interrupts re-enable after the hardware has disabled them because of an interrupt?
after stack current user / set up new priority / but before int enabled

42. Who idles the clock word in a driver when I/O is complete with no errors?
complete request COMRQST

43. Who idles the clock if an I/O error did occur?

44. Who zeroes the threadword in a READ/WRITE request, if no completion address was scheduled? If one was scheduled?
if none sched COMPLETE REQUEST
if sched) no zeroing of thread word they just schedule the comp. address
[maybe no wait

45. What type of coding must be used by mass memory programs that run in partition core? That run in allocatable core?
absolute / Run anywhere around

46. How does a core resident program pass control to a mass memory program? Can their entry point be declared external? Schedules it? No it assumes first word is executable :- not needed.

47. What type of addressing do mass memory programs use when referencing core resident programs? Why? absolute. Core resident programs don't move (:- always in same spot. Also don't know where you'll be running.

48. What is the function of the mask table (MASKT) and who sets it up initially?
tells which Int. lines enabled/inhibited. Set up in Sydat.

49. What program must be customized by the user before building his system?
Sydat

50. What program must be the last core resident program in part 0 loaded and why?
SPACE because allocatable core will appear next page all else allocs.

51. When are the request priorities set for mass memory programs?
1st job after build. * This is a LIBED command.

52. What routine is used to add the assembler and compiler to the program library?
LIBERT

53. What control statement is used when replacing a mass memory program to link the core resident entry points used by the mass memory program? LOAD *M
*M replaces

54. What is the purpose of the table of presents? Where is the table located?
used to unprotect ENT points so can be used by unprotected pgm SYSDAT

55. What is a source program?
Set of instructions to be compiled

56. What is an object program?
Program produced by assembler

57. What is an absolutized program?
program that has absolute addresses so it can be executed.

58. What is the difference between run anywhere and run-where-loaded programs?
Run anywhere = no program relocatable symbols
Run where loaded = may have "

59. What determines the drivers to be placed in an operating system?
# of devices & types - what physical devices you have in system.

60. Who initially sets the clock word in the physical device table?
Diagnostic Timer word & it's set by initiator portion of driver

61. What is the purpose of the MAKEQ routine?
Makes up the U bits in Q

62. What is the function of the complete request routine?
Housekeeping: clears diagnostic clock, schedules. checks Q for on return

63. What is the function of the first 2 words of Physical Device Table?
Scheduler call - & where driver is used for enhanced request

64. How does a driver know when he has transferred the desired number of words?
When ECCON = Lastword + 1

65. When a 'short read' takes place, how can the programmer determine how many words were actually transferred? last word of buffer contains next avail. word of buffer. Sets bit 14

66. Who notifies the operator a device is down and what responses may the operator make?
ADEV alt dev. handler => comment device RP restore
CU
DU
DD

67. What two ways may a program be placed into execution?
Schedule & interrupt

1. Which program(s) place entries in the interrupt stack:

   a. Dispatcher
   b. Read/Write Request Processor
   c. Common Interrupt Handler and Internal Interrupt Processor
   (d.) Common Interrupt Handler and Scheduler

2. Which program(s) remove entries from the interrupt stack:

   a. Dispatcher
   b. Read/Write Request Processor
   c. Common Interrupt Handler and Internal Interrupt Processor
   (d.) Common Interrupt Handler and Scheduler

3. Requests threaded to the Scheduler's thread but not in the Scheduler Stack are:

   a. Primary Scheduler calls
   b. Timer calls
   (c.) Secondary Scheduler calls
   d. all of the above

4. Requests threaded in the Scheduler's Stack and to the Scheduler's Thread are:

   (a.) Primary Scheduler calls
   b. Timer calls
   c. Secondary Scheduler calls
   d. all of the above

5. Requests threaded in the Scheduler's Stack but not to the Scheduler's Thread are:

   a. Primary Scheduler calls
   (b.) Timer calls
   c. Secondary Scheduler calls
   d. all of the above

6. How many entry points are there to the monitor:

   (a.) one RTI (MON)
   b. two
   c. three
   d. sixty-nine

7. All hardware interrupts enter via:

   a. the Interrupt Stack
   b. Request Entry Processor
   c. Dispatcher
   (d.) the Interrupt Trap Region

8. The address of the dispatcher can always be found in memory location:

a. $F4
b. $FE
c. $BB
d. $EA

9. How many threads may the standard Timer Package have pointing into the Scheduler Stack area:

a. four
b. three
c. two
d. one

Note: Use the following example to answer questions 10-19.

ASSEMBLY	MACHINE CODE
RTJ - ($F4)	54F4
NUM $60	OC46
ADC (COMPL)	8007
ADC 0	0000
NUM $18FB	18FB
ADC (LENGTH)	D213
ADC BUFF	5800

10. What type of request is this:

a. READ
b. WRITE
c. FORMAT READ
d. FORMAT WRITE

11. What is the Request Priority:

a. seven
b. four
c. five
d. three

12. Number of words to be transfered:

a. can be found in location $D213.
b. is illegal.
c. is $D213 words.
d. can be found in location $5213.

13. The completion address:

a. is relative
b. is absolute
c. is an index into the Program Directory
d. is an index into the System Directory
e. non of the above.

14. The logical unit number:

a. is four
b. can be found in location $FB
c. is eight
d. is ASCII

15. How many parameter words are required in a R/W request for a non-mass storage device:

a. two
b. four
c. six
d. ten

16. The address of the Request Entry Processor can be found in location:

a. $18FB
b. $FB
c. $05
d. $F4 monitor = Request Entry Processor

17. Which answer is true:

a. this call is an indirect monitor call This is a FWrite
b. this call is a FORMAT READ REQUEST
c. this call is a direct monitor call
d. the mode of addressing is relative X bit must be set

18. What will be the software priority when the completion address is entered:

a. four
b. five
c. six
d. seven

19. The (a) field in the logical unit is set to:

a. zero
b. one
c. two
d. eight

20. One parameter may be passed when making a scheduler's call. How is this accomplished:

a. through the Q-register
b. through the parameter list
c. through the communications region
d. through the I-register

21. Which type of request is used to request allocatable core:

a. CORE
b. SPACE
c. RELEASE
d. GET FILE

22. Which request is not available to unprotected programs:

a. CORE
b. SPACE
c. TIMER
d. SCHEDULER

23. Which requests are available to protected programs only:

a. SCHEDULER, TIMER, SPACE, RELEASE
b. CORE, LOADER, GET FILE, STATUS, EXIT
c. READ, WRITE, FORMAT READ, FORMAT WRITE
d. SPACE, RELEASE

24. The entry point to the scheduler is:

a. TC
b. T4
c. T9 scheduler is a request code of 9
d. T12

25. Which request is not re-entrant:

a. SCHEDULER all others are handled by monitor
b. STATUS status is handles by Job Processor.
c. TIMER
d. RELEASE

26. What determines the AREA of allocatable core for a SPACE request:

a. running priority
b. Completion Priority
c. Request Priority
d. logical unit

27. How many parameter words are required for a TIMER call:

a. two
b. three
c. four
d. six

28. If a TIMER call was just placed on the thread with the "u" field equal to one and the "t" field equal to 15, how long will it be before it will be removed:

a. between 1.5 and 1.6 seconds
b. between 1.4 and 1.5 minutes
c. between 1.5 and 1.6 minutes
d. between 1.4 and 1.5 hours

29. What will cause a Scheduler's Request to be rejected by the Scheduler:

a. incorrect request code
b. illegal address to transfer control to
c. the thread word in the System Directory non zero
d. all of the above

30. How can a program determine the first address of core allocated following a SPACE request:

a. in the word following the thread location in a SPACE request
b. in the Q-register at the Completion Address
c. mass storage programs operating in Allocatable Core can get it with this coding at its entry; NUM$C8FE
d. all of the above

31. What does the monitor do to flag requests that have been rejected:

a. set the Q-register to -zero
b. store a -zero in the thread
c. sets bit #15 of the Q-register to a one
d. exits to the dispatcher

32. How many parameter words are required for a SPACE request:

a. four
b. five
c. six
d. two

33. Which of the following tables is associated with threading:

a. LOG2
b. Interrupt Trap
c. Interrupt Stack
d. Volatile Storage

34. Which of the following is push down-pop up stack:

a. LOG2
b. Interrupt Trap
c. Volatile Storage
d. driver

35. In what area can the pointer to the System Directory be found:

a. MONI
b. Scheduler
c. Program Directory
d. Communications Area

36. How many words are required for each entry in the Scheduler Stack:

a. three
b. four
c. five
d. variable

37. How many words are required for each entry in the Interrupt Stack:

a. three
b. four
c. five
d. variable

38. How many words are required for each entry in Volatile Storage:

a. nine
b. four
c. five
d. variable but at least three

39. How many possible standard threads are there to the Scheduler Stack area:

a. two
b. four
c. six
d. eight

40. Which table is used to make a device down:

a. Physical Device Table
b. LOG1
c. LOG2
d. LOG1A

41. Which table contains the addresses of all Physical Device Tables:

a. LOG1
b. LOG2
c. LOG1A
d. BUFFER

42. The waiting list to use a logical unit is the:

a. LOG1 table
b. LOG2 table
c. LOG1A table
d. WAIT table

43. Which table is used to prevent unprotected programs from using certain devices:

a. LOG1 table
b. LOG1A table
c. BUFFER table
d. Physical Device Table

44. Which table allows unprotected programs the use of certain reentrant protected routines:

a. Physical Device Tables
b. Table of Presets
c. Entry Point Table
d. Program Directory

45. Devices are marked as busy by the:

a. Physical Device Tables
b. LOG2 table
c. LOG1 table
d. LOG1A table

46. The current running priority is saved in the:

a. Physical Device Table
b. Dispatcher
c. Interrupt Trap Region
d. Communications Region PRLVL