

pdp11

RT-11

Software Support Manual

Order No. DEC-11-ORPGA-B-D, DN1

digital

RT-11
Software Support Manual

Order No. DEC-11-ORPGA-B-D, DN1

First Printing, November 1973
Revised: June 1975
January 1976

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this document.

The software described in this document is furnished under a license and may be used or copied only in accordance with the terms of such license.

Digital Equipment Corporation assumes no responsibility for the use or reliability of its software on equipment that is not supplied by DIGITAL.

Copyright © 1973, 1975, 1976 by Digital Equipment Corporation

The postage prepaid READER'S COMMENTS form on the last page of this document requests the user's critical evaluation to assist us in preparing future documentation.

The following are trademarks of Digital Equipment Corporation:

DIGITAL
DEC
PDP
DECUS
UNIBUS
COMPUTER LABS
COMTEX
DDT
DECCOMM

DECsystem-10
DECTape
DIBOL
EDUSYSTEM
FLIP CHIP
FOCAL
INDAC
LAB-8
DECsystem-20

MASSBUS
OMNIBUS
OS/8
PHA
RSTS
RSX
TYPESET-8
TYPESET-10
TYPESET-11

CONTENTS

		<u>Page</u>
CHAPTER 1	RT-11 OVERVIEW	1-1
1.1	INTRODUCTION	1-1
1.2	SYSTEM CONCEPTS AND TERMINOLOGY	1-1
CHAPTER 2	MEMORY LAYOUT	2-1
2.1	FOREGROUND JOB AREA LAYOUT	2-3
2.2	JOB BOUNDARIES IN F/B	2-4
2.3	'FLOATING' USR POSITION	2-6
2.4	MONITOR MEMORY ALLOCATION	2-7
2.5	MEMORY AREAS OF INTEREST	2-9
2.5.1	Monitor Fixed Offsets	2-9
2.5.2	Table Descriptions	2-13
2.5.2.1	\$PNAME (Permanent Name Table)	2-13
2.5.2.2	\$STAT (Device Status Table)	2-13
2.5.2.3	\$ENTRY (Handler Entry Point Table)	2-14
2.5.2.4	\$DVREC (Device Handler Block Table)	2-14
2.5.2.5	\$HSIZE (Handler Size Table)	2-14
2.5.2.6	\$DVSIZ (Device Directory Size Table)	2-15
2.5.2.7	\$UNAM1, \$UNAM2 (User Name Tables)	2-15
2.5.2.8	\$OWNER (Device Ownership Table)	2-16
2.5.2.9	DEVICE Macro	2-16
2.5.3	F/B Impure Area	2-18
2.5.4	Low Memory Bitmap (LOWMAP)	2-21
2.5.4.1	S/J Restrictions	2-22
2.6	USING AUXILIARY TERMINALS AS THE CONSOLE TERMINAL	2-23
2.7	MAKING TTY SET OPTIONS PERMANENT IN F/B MONITOR	2-25
2.7.1	Carriage Width	2-26
2.7.2	Other Options	2-26
CHAPTER 3	FILE STRUCTURES AND FILE FORMATS	3-1
3.1	DEVICE DIRECTORY SEGMENTS	3-1
3.1.1	Directory Header Format	3-1
3.1.2	Directory Entry Format	3-2
3.1.2.1	Status Word	3-3
3.1.2.2	Name and Extension	3-4
3.1.2.3	Total File Length	3-4
3.1.2.4	Job Number and Channel Number	3-4
3.1.2.5	Date	3-5
3.1.2.6	Extra Words	3-7
3.2	SIZE AND NUMBER OF FILES	3-7
3.2.1	Directory Segment Extensions	3-8

		<u>Page</u>
3.3	MAGTAPE AND CASSETTE FILE STRUCTURE	3-11
3.3.1	Magtape File Structure	3-11
3.3.1.1	Bootable Magtape File Structure	3-12.1
3.3.1.2	Moving MT to Other Industry-Compatible Environments	3-13
3.3.1.3	Recovering from Bad Tape Errors	3-13
3.3.2	Cassette File Structure	3-14
3.3.2.1	File Header	3-15
3.4	RT-11 FILE FORMATS	3-16
3.4.1	Object Format (.OBJ)	3-16
3.4.1.1	Global Symbol Directory	3-20
3.4.1.2	ENDGSD Block	3-22
3.4.1.3	TXT Blocks and RLD Blocks	3-22
3.4.1.4	ISD Internal Symbol Directory	3-28
3.4.1.5	ENDMOD Block	3-28
3.4.1.6	Librarian Object Format	3-28
3.4.2	Formatted Binary Format (.LDA)	3-30
3.4.3	Save Image Format (.SAV)	3-31
3.4.4	Relocatable Format (.REL)	3-32
3.4.4.1	Non-Overlay Programs	3-33
3.4.4.2	REL Files With Overlays	3-42
CHAPTER 4	SYSTEM DEVICE	4-1
4.1	DETAILED STRUCTURE OF THE SYSTEM DEVICE	4-1
4.2	CONTENTS OF MONITR.SYS	4-2
4.3	KMON OVERLAYS	4-3
4.4	DETAILED OPERATION OF THE BOOTSTRAP	4-3
4.5	FIXING THE SIZE OF A SYSTEM	4-5
CHAPTER 5	I/O SYSTEM, QUEUES, AND HANDLERS	5-1
5.1	QUEUED I/O IN RT-11	5-1
5.1.1	I/O Queue Elements	5-1
5.1.2	Completion Queue Elements	5-5
5.1.3	Timer Queue Elements	5-7
5.2	DEVICE HANDLERS	5-8
5.2.1	Device Handler Format	5-8
5.2.2	Entry Conditions	5-9
5.2.3	Data Transfer	5-9
5.2.4	Interrupt Handler	5-9
5.3	ADDING A HANDLER TO THE SYSTEM	5-11
5.4	WRITING A SYSTEM DEVICE HANDLER	5-14
5.4.1	The Device Handler	5-14
5.4.2	The Bootstrap	5-15
5.4.3	Building the New System	5-16
5.5	DEVICES WITH SPECIAL DIRECTORIES	5-19
5.5.1	Special Devices	5-19
5.5.1.1	Interfacing to Special Device Handlers	5-19
5.5.1.2	Programmed Requests to Special Devices	5-20

		<u>Page</u>
5.6	ADDING A SET OPTION	5-21
5.7	CONVERTING USER-WRITTEN HANDLERS	5-23
CHAPTER 6	F/B MONITOR DESCRIPTION	6-1
6.1	INTERRUPT MECHANISM AND .INTEN ACTION	6-1
6.2	CONTEXT SWITCH	6-2
6.3	BLOCKING A JOB	6-3
6.4	JOB SCHEDULING AND USE OF .SYNCH REQUEST	6-3
6.5	USR CONTENTION	6-5
6.6	I/O TERMINATION	6-5
CHAPTER 7	RT-11 BATCH	7-1
7.1	CTL FORMAT	7-1
7.2	BATCH RUN-TIME HANDLER	7-2
7.3	BATCH COMPILER	7-4
7.3.1	BATCH Job Initiation	7-4
7.3.2	BATCH Job Termination	7-6
7.3.3	BATCH Compiler Construction	7-6
7.4	BATCH EXAMPLE	7-11
7.5	CTT TEMPORARY FILES	7-22
APPENDIX A	SAMPLE HANDLER LISTINGS	A-1
A.1	RC11/RS64 DEVICE HANDLER	A-2
A.2	RC11/RS64 BOOTSTRAP	A-9
A.3	LP/LS11 DEVICE HANDLER	A-28
A.4	CR11 DEVICE HANDLER	A-34
A.5	TC11 DEVICE HANDLER	A-47
APPENDIX B	FOREGROUND TERMINAL HANDLER	B-1
APPENDIX C	VERSION 1 EMT SUMMARY	C-1
APPENDIX D	FOREGROUND SPOOLER EXAMPLE	D-1

		<u>Page</u>
APPENDIX E	S/J AND F/B MONITOR FLOWCHARTS	E-1
E.1	KMON (KEYBOARD MONITOR) FLOWCHARTS	E-3
E.1.1	KMON Subroutines	E-11
E.1.2	KMON Overlays	E-17
E.2	USR (USER SERVICE ROUTINES) FLOWCHARTS	E-27
E.3	CSI (COMMAND STRING INTERPRETER) FLOWCHARTS	E-45
E.3.1	CSI Subroutines	E-51
E.4	RMON (RESIDENT MONITOR) FLOWCHARTS FOR SINGLE-JOB MONITOR	E-63
E.4.1	EMT Processors	E-67
E.4.2	Clock Interrupt Service	E-83
E.4.3	Console Terminal Interrupt Service	E-85
E.4.4	I/O Routines	E-97
E.5	RMON (RESIDENT MONITOR) FLOWCHARTS FOR FOREGROUND/BACKGROUND MONITOR	E-101
E.5.1	EMT Processors	E-103
E.5.2	Job Arbitration, Error Processing	E-121
E.5.3	Queue Managers (I/O, USR, Completion)	E-131
E.5.4	Clock Interrupt Service	E-137
E.5.5	Console Terminal Interrupt Service	E-139
E.5.6	Resident Device Handlers (TT, Message)	E-147
	Entry Point Index	E-151

FIGURES

<u>Number</u>		<u>Page</u>
2-1	Monitor Memory Layout	2-1
2-2	Foreground Job Area Layout	2-4
2-3	Job Limits	2-5
2-4	Background SYSLOW Examples	2-6
2-5	Memory Allocation	2-8
3-1	Directory Entry Format	3-1
3-2	Directory Segment	3-6
3-3	Object Module Processing	3-17
3-4	Formatted Binary Block	3-18
3-5	GSD Structure	3-20
3-6	TXT Block Format	3-22
3-7	RLD Format	3-24
3-8	Library File Format	3-28
3-9	Library Header Format	3-29
3-10	Entry Point Table Format	3-29
3-11	Library End Trailer	3-30
3-12	Formatted Binary Format	3-31
3-13	REL File Without Overlays	3-33
3-14	REL File With Overlays	3-43
3-15	Overlay Segment Relocation Block	3-44
5-1	I/O Queue Element	5-2
5-2	Completion Queue Element	5-6
5-3	SYNCH Element	5-7
5-4	Timer Queue Element	5-7

TABLES

<u>Number</u>		<u>Page</u>
2-1	Fixed Offsets	2-10
2-2	Impure Area	2-19
2-3	Bitmap Byte Table	2-21
2-4	Default Functions for TTY Options	2-25
2-5	TTCNFG Option Bits	2-27
3-1	Directory Header Words	3-2
3-2	File Types	3-3
3-3	ANSI MT Labels Under RT-11	3-12
3-4	CT File Header Format	3-16
7-1	BATCH Compiler Data Base Description	7-7
C-1	V1 Programmed Requests	C-1

PREFACE

The RT-11 Software Support Manual covers the internal description of the RT-11 software system. Chapter 1 presents an overview of the system and discusses conventions used throughout the manual. Chapters 2 through 6 describe in detail various aspects of the monitor and system structure, including memory layout, monitor tables, file structures, file formats, system device structure, bootstrap operation, I/O queuing system, device handlers and F/B monitor description. Chapter 7 discusses the operation of the BATCH compiler and run-time handler.

The appendixes provide example handler listings, including a foreground terminal handler (Appendix B) and a sample foreground program (Appendix D). Complete flowcharts of both the Single-Job and Foreground/Background Monitors are shown in Appendix E.

The reader should be thoroughly familiar with the RT-11 system. Although the information in this manual is aimed at V02B and V02C users, it should be adequate for Version 2 users also; excluding a few minor alterations (to permit the addition of the new V02B devices), the construction of the monitors has changed very little between the two versions. A comprehensive list of differences between the V02B and V02C and between V2 and V02B systems is included in RT-11 System Release Notes (V02C), (DEC-11-ORNRA-A-D).

It is assumed that the user has read the RT-11 System Reference Manual (DEC-11-ORUGA-B-D) or (DEC-11-ORUGA-C-D) and all other documentation included in the RT-11 kit, and is an experienced PDP-11 programmer. It is recommended that RT-11 monitor source listings be available for reference.

CHAPTER 1

RT-11 OVERVIEW

1.1 INTRODUCTION

RT-11 is a single-user programming and operating system designed for the PDP-11 series of computers. It permits the use of a wide range of peripherals and up to 28K of either solid state or core memory (hereafter referred to as memory).

RT-11 provides two operating environments: Single-Job (S/J) operation, and a powerful Foreground/Background (F/B) capability. Either environment is controlled by a single user from the console terminal keyboard by means of the appropriate monitor--S/J or F/B. The monitors are upwards compatible; features that are used only in a F/B environment are treated as no-ops under the S/J Monitor.

A feature common to both operating environments is the inclusion of a full complement of system development and utility programs to aid the programmer in the development of his own applications.

The normal use and operation of the monitors and system programs is discussed in detail in the RT-11 System Reference Manual. Concepts and applications that are specialized and useful to the more experienced programmer are included in this manual.

1.2 SYSTEM CONCEPTS AND TERMINOLOGY

The basic concepts necessary to use RT-11 effectively are defined in the RT-11 System Reference Manual. The user should be familiar with those concepts before proceeding to use this manual.

Abbreviations used throughout this document are:

<u>TERM</u>	<u>MEANING</u>
KMON	Keyboard Monitor The console terminal interface to RT-11. KMON runs as a background job and allows the user to run programs, assign device names, and generally control the system.
USR	User Service Routines The nonresident (swapping) part of RT-11. The USR performs file-oriented operations.
CSI	Command String Interpreter The CSI is part of the USR. It accepts a string of characters from memory or from the console and performs specified file operations, or syntactically analyzes a command string and constructs a table from the information supplied.
RMON	Resident Monitor RT-11 provides a choice of two Resident Monitors: a Single-Job Monitor and a Foreground/Background Monitor. RMON specifically provides the following services: EMT dispatcher Keyboard (console) interrupt service TT: resident device handler (F/B only) Read/Write processor USR swap routines I/O queuing routines System device handler System I/O tables Message handler (F/B only) Job scheduler (F/B only)
CSW	Channel Status Word Each bit in the CSW contains information relevant to the status of a channel; see Chapter 9 (.SAVESTATUS) of the <u>RT-11 System Reference Manual</u> .

<u>TERM</u>	<u>MEANING</u>
JSW	Job Status Word The JSW contains information in bytes 44 and 45 about the job currently in memory.
F/B	The Foreground/Background version of the monitor
S/J	The Single-Job version of the monitor
B/G	The background job
F/G	The foreground job
<CR>	Carriage Return
<LF>	Line Feed

Various mnemonic names (e.g., BLIMIT, SYSLOW), referred to from within the text and in diagrams and flowcharts, represent the actual symbolic names as they appear in the monitor source listings.

To avoid confusion, underlining is used in most examples to designate computer printout; square brackets, [and], are used to enclose comments. Values for symbolic names used in examples can be found in Table 2 of RT-11 System Release Notes.

CHAPTER 2

MEMORY LAYOUT

RT-11 operates properly in any configuration between 8K and 28K (words) of memory (16K to 28K for the F/B Monitor). No user intervention is required when programs are moved to a different size machine; i.e., programs correctly developed in one environment will work in any size environment (providing there is sufficient memory) with no relinking necessary.

Figure 2-1 shows a general diagram of the memory layout in an RT-11 system.

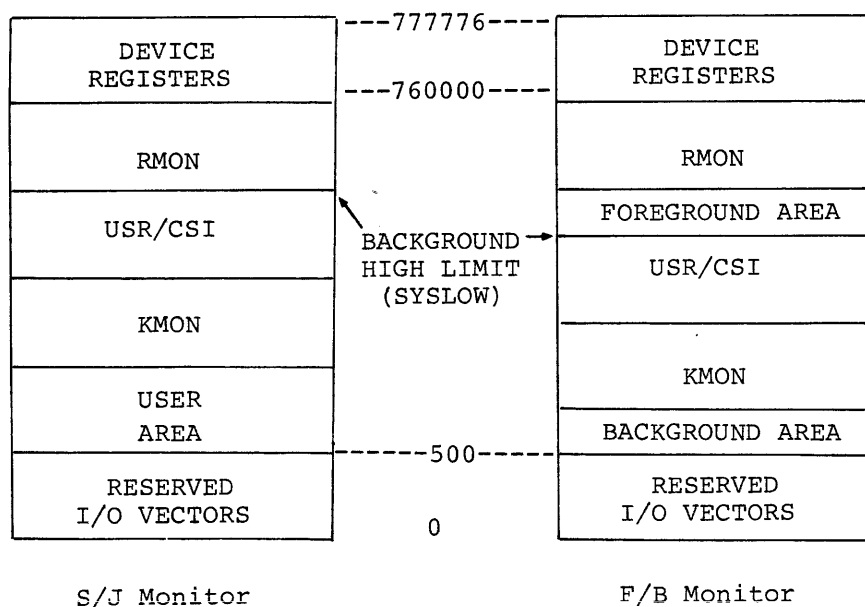


Figure 2-1
Monitor Memory Layout

The memory area diagrammed is arranged as follows:

<u>Memory Area</u>	<u>Use</u>
0-477	Reserved for I/O vectors, RT-11 system communication area.
500-SYSLOW	Space available for user (background) programs. (The high limit of memory for the background is contained in SYSLOW, a location in the monitor data base.)

Space for foreground programs and LOAded handlers is allocated as needed, reducing the amount of space available for a background job.

The areas marked KMON and USR/CSI are the areas that these units normally occupy when they are in memory. The amount of memory that a user program occupies is determined by:

1. The initial size of the program, or
2. The amount of memory the user program requests via a .SETTOP programmed request.

When a user program (background job) is executed (via the KMON commands R, RUN, or GET and START), the top of memory is set to correspond to the size of the program. If the top of user memory never exceeds KMON, both KMON and USR/CSI are resident. If all of memory (up to SYSLOW) is requested (via a .SETTOP), neither the KMON nor the USR is resident and swapping of the USR is required. Programs performing many file-oriented operations gain from having the USR resident, since no time is spent swapping the USR.

The KMON, USR, and RMON modules normally occupy the upper segment of memory. This implies that larger memory configurations automatically have more free memory available.

The area marked DEVICE REGISTERS is the top 4K of memory in any PDP-11 computer. This area is reserved for the status and control registers of peripheral devices.

2.1 FOREGROUND JOB AREA LAYOUT

The foreground job area is located above the KMON/USR, as shown in Figure 2-1, and is allocated by the FRUN command. The actual layout of the job within the foreground area is shown in Figure 2-2. The *impure area* (described in Section 2.5.3) occupies the lowest 207 words of the job area and contains terminal ring buffers, I/O channels, and other job-specific information.

The foreground stack is located immediately above the impure area with a default size of 128 words; this may be changed using the FRUN /S switch. The program may specify a different location for the stack by using an .ASECT into location 42, in which case the /S switch is ignored and the program itself must allocate stack space. Wherever the stack is located, stack overflow will most probably cause program malfunction before penetrating the task area boundary, since either the program itself or the impure area will be corrupted.

NOTE

Users must not use a relocatable symbol as the contents of location 42 when resetting the initial stack pointer via an .ASECT in a foreground job; such a symbol is not relocated when it occurs in an .ASECT in a foreground job. To set the stack to relative location 1000 in a foreground job, use:

```
.ASECT
.=42
.WORD 1000
```

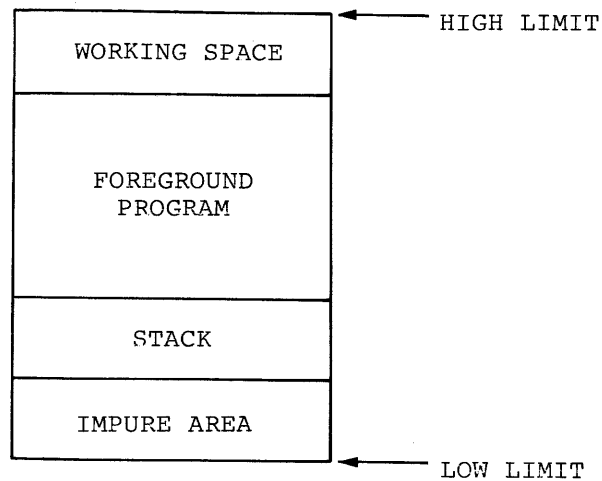


Figure 2-2
Foreground Job Area Layout

The space allocated for the foreground program is sufficient to contain the program code itself, as indicated by location 50 (in block 0 of the file); location 50 is set by the Linker and designates the program's high limit. If the foreground job requires working space, this space must either be reserved from within the program (e.g., using .BLKW) or allocated at run-time using the FRUN /N switch. Space allocated with the /N switch is located above the program as shown in Figure 2-2. Location 50 will point to the top of the program area and a .SETTOP will permit access to any working space.

2.2 JOB BOUNDARIES IN F/B

The actual job boundaries are stored (in RMON) in limit tables for both foreground and background jobs. The FLIMIT table contains high and low boundaries for the foreground, and the BLIMIT table contains boundaries for the background. .SETTOPs are permitted for any job up to its high limit. The SYSLOW pointer mentioned earlier is equivalent to the background BLIMIT high pointer entry. This is shown in Figure 2-3.

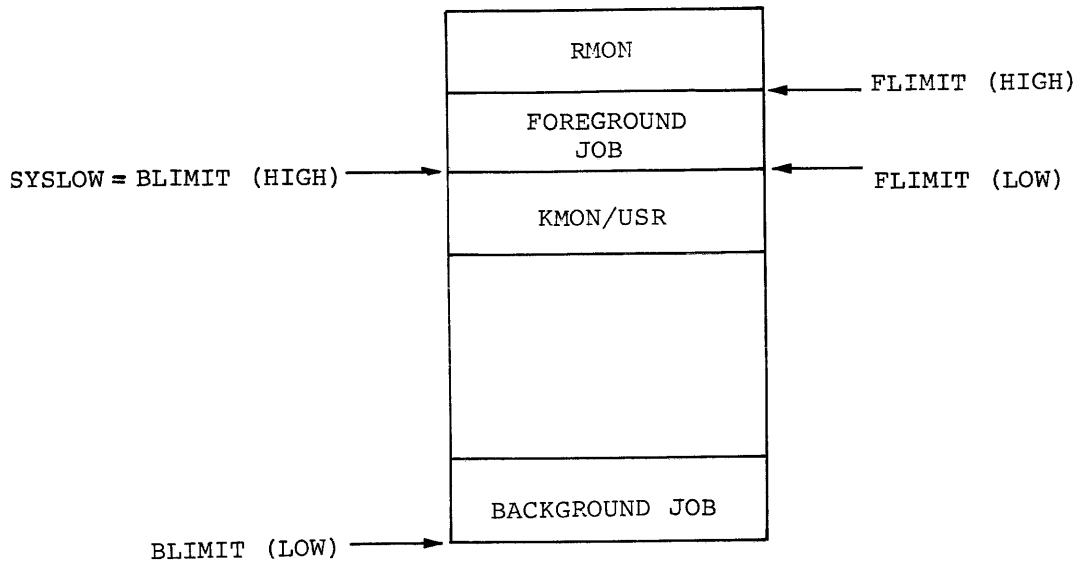


Figure 2-3
Job Limits

The limit pointers for a foreground job are fixed once the job has been loaded into memory. A program that requires working space and uses a .SETTOP will fail if the space is not allocated with the /N switch (a FORTRAN program is a typical case; see Appendix G, Section G.1, of the RT-11 System Reference Manual). The high limit pointer (SYSLOW) for the background, however, is not fixed and will change as space is allocated for LOAded handlers, the text scroller, and foreground jobs. In addition, if the USR is made permanently resident (using the SET USR NOSWAP command), SYSLOW (BLIMIT HIGH) will again change. This is shown in Figure 2-4.

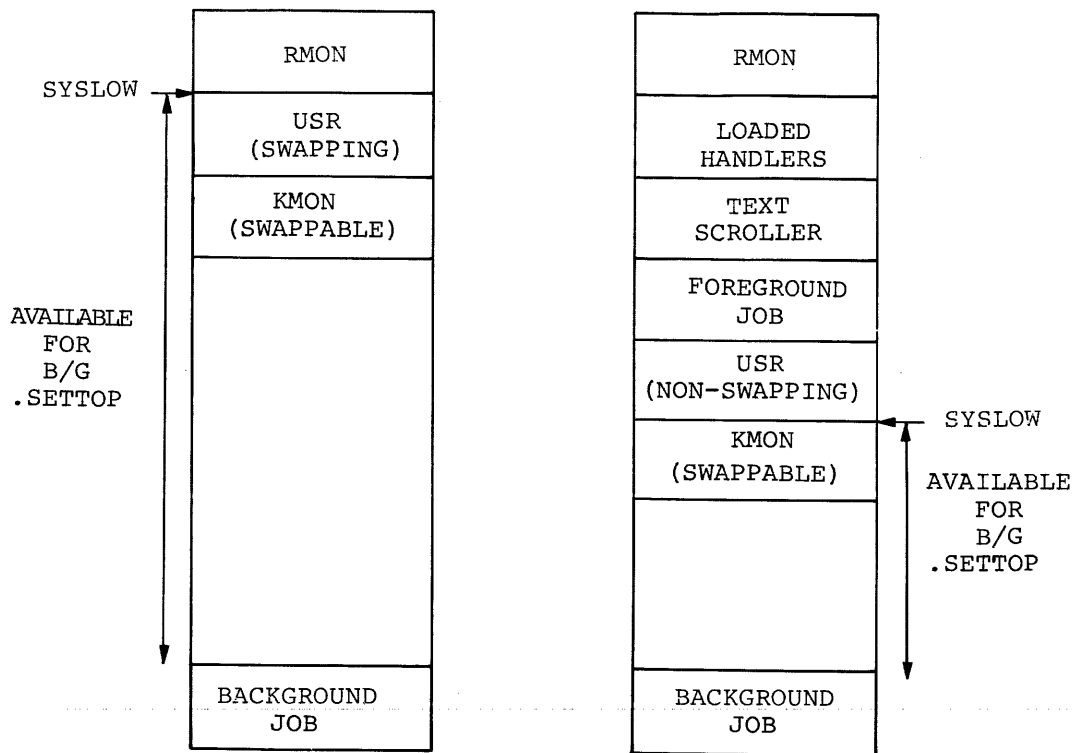


Figure 2-4
Background SYSLOW Examples

2.3 'FLOATING' USR POSITION

The RT-11 USR is normally located in the memory area directly below that pointed to by SYSLOW. For the Version 1 monitor, this was directly below the RMON. For the Version 2 and 2B monitors, the USR position varies as handlers, the scroller, and foreground jobs (in F/B) are loaded into memory; the SYSLOW pointer is corrected for each change in memory configuration. In any case, the SYSLOW position is considered the normal USR swapping position.

It is possible, however, to cause the USR to swap into another location in memory. This is done by setting location 46 (in the system communication area) to the address at which the USR is to swap; if the contents of location 46 are nonzero and even, the monitor loads the USR at the new address. Note, however, that if no swapping is required, the USR is *not* loaded at the address indicated in location 46. Location 46 is cleared by an exit to the Keyboard Monitor (via an .EXIT, .HRESET, .SRESET, or CTRL C).

It is possible to make the USR permanently resident (i.e., non-swapping). Using the SET USR NOSWAP Keyboard Monitor command makes the USR permanently resident at its *normal* position, that is, below the memory area pointed to by SYSLOW.

2.4 MONITOR MEMORY ALLOCATION

RT-11 uses a dynamic memory allocation scheme to provide memory space for LOAded handlers, foreground jobs (F/B Monitor only) and the display text scroller. Memory is allocated in the region above the KMON/USR and below RMON. If there is insufficient memory in this region (initially, after the system is bootstrapped, there is none), memory is taken from the background region by "sliding down" the KMON/USR the required number of words.

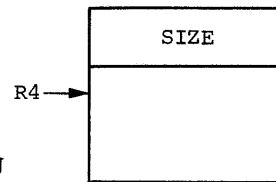
When memory allocated in this manner is released, the memory block is returned to a singly-linked free memory list, the list head of which is in RMON. Any contiguous blocks are concatenated into a single larger block. A block found to be contiguous with the KMON/USR is reclaimed by "sliding up" the KMON/USR, removing the block from the list.

Memory allocation and release is achieved by calls to the GETBLK and PUTBLK routines located in the KMON overlays (the GETBLK and PUTBLK routines are flowcharted in Appendix E). The requested number of words is passed to GETBLK in R0, and the address of the block is returned in R4. An extra word of memory is allocated by GETBLK, which then stores the size of the block in that word. R4 points to the first available word in the block (see Figure 2-5a). When releasing memory, R4 must point to the first available word, the same address returned by GETBLK during allocation (as shown in Figure 2-5b). The block will be linked into the free memory list (shown in Figure 2-5c).

a) Allocating a memory block

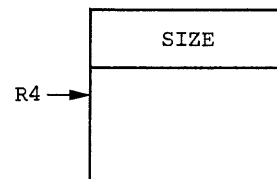
Call sequence:
 $R0 = SIZE$
 JSR PC,GETBLK

(returns with R4 pointing
 to the allocated block)



b) Releasing a memory block

Call sequence:
 $R4 \rightarrow BLOCK$
 JSR PC,PUTBLK



c) Free memory list

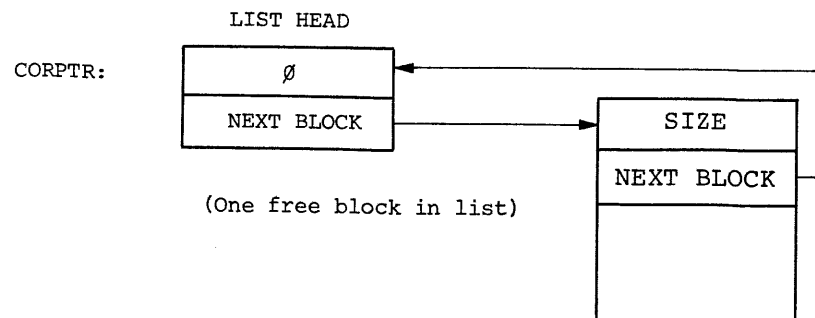


Figure 2-5
 Memory Allocation

When a block of memory of sufficient size is not available, GETBLK must create a hole in memory by sliding down the KMON/USR. This is achieved by a call to KUMOVE, a small routine located physically at the front of the KMON. KUMOVE does the actual work of moving the KMON/USR up in memory. For moves downward, an auxiliary subroutine, MOVEDN, located at the top of the USR, is used.

Whenever a request is made for a block of a certain number of words, the memory allocator searches memory for the first highest block that is large enough to satisfy the request (that is, equal to or larger than the requested number). The goal of the memory allocator is to minimize the amount of free (unused) memory in the foreground region, making the maximum amount of memory available to the background. Contiguous blocks of free memory are merged and reclaimed whenever possible. The search time of the singly-linked list is not a factor, since at any time there will be few nodes (free memory areas) in the list, and the allocator minimizes the number.

2.5 MEMORY AREAS OF INTEREST

This section describes memory areas of particular interest and indicates the contents of those locations. The areas covered are:

1. Monitor Fixed Offsets (F/B & S/J)
2. F/B Impure Area
3. Resident Bitmap (F/B & S/J)
4. Tables

2.5.1 Monitor Fixed Offsets

Certain values are maintained at fixed locations from the start of the Resident Monitor in both F/B and S/J; these quantities (listed in Table 2-1) may be accessed by user programs. The technique used to access these offsets is as follows:

OFFSET = the byte offset to the word desired
RMON = 54

```
MOV @#RMON,Rn          ;ANY GENERAL REGISTER
MOV OFFSET(Rn),Rn
```

Rn now contains the desired quantity. If a byte quantity is desired, a better method is:

```
CLR Rm
MOV @#RMON,Rn
BISB OFFSET(Rn),Rm
```

This ensures that the high-order bits of the register are not set by a MOVB into the register.

Table 2-1
Fixed Offsets

Offset (from Start of RMON) Octal Decimal		Tag	Byte Length	Description
0		-	4	Serves as a link to interrupt entry code.
4		\$CSW	160 ₁₀	Default I/O channels for the background (16 ₁₀ @ 5 words each).
244	164	\$SYSCH	10 ₁₀	Internal I/O channel used for system functions.
256	174	BLKEY	2	Segment number of the directory now in memory. 0 implies no directory is there.
260	176	CHKEY	2	Device index and unit number of the device whose directory is in memory. Bits 1-5 are the device index, bits 8-10 are the unit number.
262	178	\$DATE	2	Current date value. (The format is shown in Chapter 3, section 3.1.2.5.)
264	180	DFLG	2	"Directory operation in progress" flag. Used to inhibit ^C from aborting a job until directory operation is finished.
266	182	\$USRLC	2	Normal location of USR.
270	184	QCOMP	2	Address of I/O completion manager, COMPLT.
272	186	SPUSR	2	Flag word used by MT/CT. If a USR function performed by MT or CT fails, this word is made non-zero.

(continued on next page)

Table 2-1 (Cont.)

Fixed Offsets

Offset (from Start of RMON) Octal Decimal		Tag	Byte Length	Description																						
274	188	SYUNIT	2	High-order byte contains the unit number of the current system device.																						
276	190	SYSVER	1	Monitor version number (2 in Versions 2, 2B, and 2C).																						
277	191	SYSUPD	1	Version release number (1 for V02, 2 for V02B, etc.)																						
300	192	CONFIG	2	System configuration word. A 16-bit series of flags whose meanings are: <table><tr><th>Bit #</th><th>Meaning</th></tr><tr><td>0</td><td>0 → S/J Monitor 1 → F/B Monitor</td></tr><tr><td>2</td><td>1 → VT11 hardware exists</td></tr><tr><td>3</td><td>1 → RT-11 BATCH controls the background</td></tr><tr><td>5</td><td>0 → 60-cycle KW11L clock 1 → 50-cycle clock</td></tr><tr><td>6</td><td>1 → 11/45 FPP present</td></tr><tr><td>7</td><td>0 → No foreground job present 1 → Foreground job is in memory</td></tr><tr><td>8</td><td>1 → User is linked to VT11 scroller</td></tr><tr><td>9</td><td>1 → USR is resident via SET USR</td></tr><tr><td>11</td><td>0 → No PDP-11/03 processor 1 → PDP-11/03 processor</td></tr><tr><td>15</td><td>1 → KW11 clock is present (always set if bit 11 is 1)</td></tr></table> Any bits not currently assigned are reserved by DIGITAL for future use and should not be used arbitrarily by user programs.	Bit #	Meaning	0	0 → S/J Monitor 1 → F/B Monitor	2	1 → VT11 hardware exists	3	1 → RT-11 BATCH controls the background	5	0 → 60-cycle KW11L clock 1 → 50-cycle clock	6	1 → 11/45 FPP present	7	0 → No foreground job present 1 → Foreground job is in memory	8	1 → User is linked to VT11 scroller	9	1 → USR is resident via SET USR	11	0 → No PDP-11/03 processor 1 → PDP-11/03 processor	15	1 → KW11 clock is present (always set if bit 11 is 1)
Bit #	Meaning																									
0	0 → S/J Monitor 1 → F/B Monitor																									
2	1 → VT11 hardware exists																									
3	1 → RT-11 BATCH controls the background																									
5	0 → 60-cycle KW11L clock 1 → 50-cycle clock																									
6	1 → 11/45 FPP present																									
7	0 → No foreground job present 1 → Foreground job is in memory																									
8	1 → User is linked to VT11 scroller																									
9	1 → USR is resident via SET USR																									
11	0 → No PDP-11/03 processor 1 → PDP-11/03 processor																									
15	1 → KW11 clock is present (always set if bit 11 is 1)																									
302	194	SCROLL	2	Address of the VT11 scroller.																						
304	196	TTKS	2	Address of console keyboard status.																						
306	198	TTKB	2	Address of console keyboard buffer.																						

(continued on next page)

Table 2-1 (Cont.)

Fixed Offsets

Offset (from Start of RMON) Octal Decimal		Tag	Byte Length	Description
310	200	TTPS	2	Address of console printer status.
312	202	TTPB	2	Address of console printer buffer. (See Section 2.6, Using Auxiliary Terminals as the Console Terminal.)
314	204	MAXBLK	2	Largest output file permitted with an indefinite length request (initially defined as -1, which implies that no limit is defined).
316	206	E16LST	2	Offset from start of RMON to the dispatch table for EMT's 340-357. (This is used by the BATCH processor.)
320	208	CNTXT	2	Pointer to the impure area for the current executing job.
322	210	JOBNUM	2	
320	208	\$TIME	4	Two words of time of day in the S/J Monitor.
322	210			
324	212	SYNCH	2	Address of monitor routine to handle .SYNCH request.
326	214	LOWMAP	20 ₁₀	Start of low memory protection map. (This map protects vectors at locations 0-476.)
352	234	USRLOC	2	Pointer to current entry point of USR.
354	236	GTVECT	2	Pointer to VT11 vector. The vector is initially positioned at 320.
356	238	ERRCNT	1	Error count byte (for future use by system programs).
357	239	FUTURE	5	Reserved by DIGITAL for future use.

2.5.2 Table Descriptions

The monitor device tables discussed in this section include:

```
$PNAME
$STAT
$ENTRY
$DVREC
$HSIZE
$DVSIZ
$UNAM1,$UNAM2
$OWNER
```

The size of these tables is fixed and is governed by the \$SLOT assignment; the default value is 14₁₀ entries per table. To alter this, it is necessary to first edit a new value of \$SLOT into the monitor source program, then reassemble and relink new monitors.

2.5.2.1 \$PNAME (Permanent Name Table) - \$PNAME is the central table around which all the others are constructed. There is an entry in \$PNAME for each device in the system. Each entry consists of a single word that contains the .RAD50 code for the two-character permanent device name for that device; for example the entry for DECTape is .RAD50 /DT/. The position of devices in this table is non-critical, but their relative position determines the general device index used in various places in the monitor; thus, all other tables must be organized in the same order as \$PNAME (the index into \$PNAME serves as the index into all the other tables for the equivalent device).

2.5.2.2 \$STAT (Device Status Table) - Each device in the system must have a status entry in its corresponding slot in \$STAT. The status word is broken down into two bytes as follows:

Even byte - contains a device identifier. Each unique type of device in the system has an identifying integer. Those defined are:

```
0 = RK05 Disk
1 = TC11 DECTape
2 = Reserved
3 = Line Printer (LP11, LS11, LV11)
4 = Console Terminal (LT33/35, LA30/36,
  VT05, VT50)
5,6 = Reserved
7 = PC11 High-speed Reader
10 = PC11 High-speed Punch
11 = Magtape (TM11, TU10)
12 = RF11 Disk
13 = TA11 Cassette
```

14 = Card Reader (CR11, CM11)
 15 = Reserved
 16 = RJS03/4 Fixed-head Disks
 17 = Reserved
 20 = TJU16 Magtape
 21 = RP11/RP02/RP03 Disk
 22 = RX11/RX01 Diskette

Odd byte - Bit flags with the following meanings:

Bit 15: 1 = Random-access device (disk, DECTape)
 0 = Sequential-access device (line printer, papertape, card reader, magtape, cassette, terminal)
 Bit 14: 1 = Read-only device (card reader, papertape reader)
 Bit 13: 1 = Write-only device (line printer, papertape punch)
 Bit 12: 1 = NonRT-11 directory-structured device (magtape, cassette)
 Bit 11: 1 = Enter handler abort entry every time a job is aborted.
 0 = Handler abort entry taken only if there is an active queue element belonging to aborted job.
 Bit 10: 1 = Handler accepts .SPFUN requests (e.g., MT, CT, DX).
 0 = .SPFUN requests are rejected as illegal.
 Bits 9-8: Reserved

2.5.2.3 \$ENTRY (Handler Entry Point Table) - Whenever a handler is made resident, either by a .FETCH or with the LOAD command, the \$ENTRY slot for that device is made to point to the fourth word of the device handler. The entry is zeroed when the handler is .RELEASEd or UNLOADed.

2.5.2.4 \$DVREC (Device Handler Block Table) - This table (filled in at system bootstrap time) reflects the absolute block position of each of the device handlers on the system device. Since handlers are treated as files under RT-11, their position on the system device is not necessarily fixed. Thus, each time the system is bootstrapped, the handlers are located and \$DVREC is updated with the value of the second block of the handler file. (Because the handlers are linked at 1000, the actual handler code starts in the second block of the file.) A zero entry in the \$DVREC table indicates that no handler for the device in that slot was found on the system device.

2.5.2.5 \$HSIZE (Handler Size Table) - This table contains the size, in bytes, of each device handler. The table is set up at assembly time with the correct values and is used when a .FETCH is executed to provide the size of the specified handler. This size is also returned to the user as one of the values returned in a .DSTAT request.

2.5.2.6 \$DVSIZ (Device Directory Size Table) - Entries in this table are non-zero for file-structured devices only and reflect the number of 256₁₀-word blocks contained on the device. The current devices and their entries are:

<u>Device</u>	<u>Number of 256-Word Blocks</u>	<u>Device</u>	<u>Number of 256-Word Blocks</u>
RK11	11300 ₈	RP02	116300 ₈
TC11	1102 ₈	RJS03	2000 ₈
		RJS04	4000 ₈
RF11	2000 ₈ (1 platter) 4000 ₈ (2 platters) 6000 ₈ (3 platters) 10000 ₈ (4 platters)	RX01	752 ₈

The default for RF11 and RJS03/4 is one platter, or 2000₈ blocks. It is possible to alter the system to indicate the correct number of platters. Instructions are in Chapter 4 of the RT-11 System Generation Manual, (DEC-11-ORGMA-A-D).

2.5.2.7 \$UNAM1, \$UNAM2 (User Name Tables) - These tables are used in conjunction with ASSIGN keyboard functions. The form of the ASSIGN command is:

```
.ASSIGN pnam:unam<CR>
```

where:

```
pnam - a system device name/unit number
unam - a user-assigned device name
```

A typical example is:

```
.ASSIGN DT1:DK
```

The default device name, DK, is now directed to DECTape unit 1. The user-assigned name is stored in an available slot in \$UNAM2, while the device's permanent name/unit is stored in the corresponding slot in \$UNAM1. The system uses a common device name lookup routine that maps any user-assigned name in the \$UNAM2 table into a physical device name to be used in an operation. The total number of ASSIGNS permitted is limited by the value of \$SLOT.

The command:

```
.ASSIGN<CR>
```

zeroes \$UNAM2, thus removing all user assignments.

2.5.2.8 \$OWNER (Device Ownership Table) - This table is used only under F/B to arbitrate device ownership. The table is \$SLOT*2 words in length and is divided into 2-word entries per device. Each 2-word entry is divided into eight 4-bit fields capable of holding a job number. Thus, each device is presumed to have up to eight units, each assigned independently of the others. However, if the device is nonfile-structured, the ownership is assigned to all units.

When a job attempts to access a particular unit of a device, the F/B Monitor checks to be sure the unit being accessed is either public or belongs to the requesting job. If the unit is owned by the other job, a fatal error is generated.

The device is assumed to be public if the 4-bit field is 0. If it is not public, the field contains a code equal to the job number plus one. Since job numbers are always even, the ownership code is odd. Bit 0 of the field being set is then used to indicate that the unit ownership is assigned to a job (1 for the background job and 3 for the foreground job).

2.5.2.9 DEVICE Macro - The DEVICE macro call is used in RMON to allow quick and easy insertion of new devices at assembly time. The form of the macro call is:

```
DEVICE  NAME,SIZ,STAT,ENTRY
```

where:

NAME - two characters of the permanent device name

SIZ - the size of the device's directory in 256-word blocks; 0 means nonfile-structured or special

STAT - the sum of all \$STAT table entries that apply for this device plus the device id (from section 2.5.2.2):

FILST\$ = 100000	Random-access device (disk, DECTape)
RONLY\$ = 40000	Read-only device
WONLY\$ = 20000	Write-only device
SPECL\$ = 10000	Non RT-11 directory-structured device (including MT and CT)
HNDLR\$ = 4000	Handler abort entry
SPFUN\$ = 2000	Special function requests

ENTRY - the 2-character device name with the SYS appended, if this is a system device.

Thus, a sample call is:

```
DEVICE TT,0,4
```

The SIZ entry is 0, since TT is a nonfile-structured device.

The entry for DECTape is:

```
DEVICE DT,1102,1+FILST$,DTSYS
```

The 1+FILST\$ indicates that the device code is 1 and FILST\$ is defined as 100000. The entry for DTSYS is present because DT can be a system device.

In addition to the DEVICE macro, another macro, HSIZE, is defined and sets the handler size for the \$HSIZE table. The format of the HSIZE macro call is:

```
HSIZE HAN,BYT,TYPE
```

where:

HAN - the 2-letter device name

BYT - the handler size in bytes

TYPE - SYS if the device can be a system device; blank otherwise

Chapter 5 shows the use of HSIZE in adding a handler to the RT-11 system. The KMON portion of the monitor source listing should be consulted for greater detail.

2.5.3 F/B Impure Area

An impure area is defined here as that area of memory where the monitor stores all job-dependent data. Thus, the impure area contains all information that the monitor requires to effectively run two independent jobs, both of which are memory-resident. This section details the contents and location of each word (byte) in the impure area.

A table that points to the impure area for a particular job is in the F/B monitor's data base. This table is at \$IMPUR and currently consists of two words: the first is a pointer to the background's impure area (which is permanently resident in RMON at location BKGND), the second is the foreground's pointer. The \$IMPUR table is accessed by using IMPLOC, located at an offset of 422 into RMON. IMPLOC points beyond the end of \$IMPUR to \$IMPUR +4 to facilitate accessing the \$IMPUR table from the top down in order of decreasing priority.

Under RT-11, a background job is always running and will be the KMON if no other background job exists. However, the foreground impure area pointer may be 0 if no foreground job is in memory. When an FRUN command is given, a foreground impure area is created for the job and the \$IMPUR entry for the foreground pointer is updated to point to the impure area.

A foreground program can determine whether the KMON is resident by testing KMONIN, located at an offset of 424 into RMON. KMONIM is non-zero if the KMON is resident and zero if a background job is running. In addition, the file name of the running foreground or background job is located in the job's impure area at offset I.NAME (376). Note that for a background job, KMONIN must first be tested to determine whether the name belongs to an active job since the file descriptor is not cleared when KMON is entered.

Table 2-2 is a detailed breakdown of the contents of the impure area. The offset mentioned is the offset from the start of the impure area itself; thus, the first word in the area has a 0 offset.

Table 2-2
Impure Area

Offset	Mnemonic	Octal Length (Bytes)	Contents
0	I.JSTA	2	Job status.
2	I.QHDR	2	I/O Queue Header.
4	I.CMPE	2	Last entry in completion queue. I/O completion routines are queued for execution. This is the pointer to the last routine to be entered.
6	I.CMPL	2	Completion queue header.
10	I.CHWT	2	Pointer to channel during I/O wait. When a job is waiting for I/O, the address of the channel area in use goes here.
12	I.PCHW	2	Saved channel pointer during execution of a completion routine. The contents of I.PCHW are put in R0 when a completion routine is entered.
14	I.PERR	2	Error byte 52 and 53 saved during completion routines.
16	I.PTTI	2	Previous TT input character.
20	I.TTLC	2	Terminal input ring buffer line count.
22	I.TID	2	Pointer to job ID area.
24	I.JNUM	2	Job number of job that owns this impure area.
26	I.CNUM	2	Number of I/O channels defined. 16 ₁₀ is default, .CDFN can be used to define new ones.
30	I.CSW	2	Pointer to job's channel area.
32	I.IOCT	2	Count of total I/O operations outstanding.
34	I.SCTR	2	Suspension count. Zero means the number of .SPNDs = the number of .RSUMs.
36	I.SPLS	2	Address of the .DEVICE request list.

(continued on next page)

Table 2-2 (Cont.)
Impure Area

Offset	Mnemonic	Octal Length (Bytes)	Contents
40	I.TRAP	2	Address of user trap routine. Set by .TRPSET.
42	I.FPP	2	Address of FPP exception routine. Set by .SFPA.
44	I.SWAP	4	Address and number of extra words to be included in the context switch operation. Set by .CNTXSW request.
50	I.SP	2	Saved stack pointer. When this job is made inactive, the active value of SP is saved here.
52	I.BITM	24	Low memory protection bitmap. This map reflects the user's .PROTECT requests.
(76 through 332 concern the console terminal)			
76	I.IRNG	2	Input ring buffer low limit.
100	I.INPUT	2	Input "PUT" pointer for inter- rupts.
102	I.ICTR	2	Input character counter.
104	I.IGET	2	Input "GET" pointer for .TTYIN.
106	I.ITOP	2	Input ring buffer high limit.
110		144	Input ring buffer.
254	I.OPUT	2	Output "PUT" pointer for interrupts.
256	I.OCTR	2	Output character counter.
260	I.OGET	2	Output "GET" pointer for interrupts.
262	I.OTOP	2	Output ring buffer high limit.
264		50	Output ring buffer.
334	I.QUE	20	Initial I/O queue element.

(continued on next page)

Table 2-2 (Cont.)
Impure Area

Offset	Mnemonic	Octal Length (Bytes)	Contents
354	I.MSG	12	Message channel. Used by .RCVD and .SDAT. This channel is permanently open.
366		10	Job ID area. Contains (<CR><LF>)B(<CR><LF>) or (<CR><LF>)F(<CR><LF>) for terminal prompting. Space has been left for up to a 3-character job name.

2.5.4 Low Memory Bitmap (LOWMAP)

RT-11 maintains a bitmap which reflects the protection status of low memory, locations 0-476. This map is required in order to avoid conflicts in the use of the vectors. In F/B, the .PROTECT request allows a program to gain exclusive control of a vector or a set of vectors. When a vector is protected, the bitmap is updated to indicate which words are protected. If a word in low memory is protected, it will not be destroyed when a new background program is run.

The bitmap is a 20₁₀ byte table which starts 326 bytes from the beginning of the Resident Monitor. Table 2-3 lists the offset from RMON and the corresponding locations represented by that byte:

Table 2-3
Bitmap Byte Table

Offset	Locations (octal)	Offset	Locations (octal)
326	0-16	340	240-256
327	20-36	341	260-277
330	40-56	342	300-316
331	60-76	343	320-336
332	100-116	344	340-356
333	120-136	345	360-376
334	140-156	346	400-416
335	160-176	347	420-436
336	200-216	350	440-456
337	220-236	351	460-476

Each byte in the table reflects the status of 16_{10} words of memory. The first byte in the table controls locations 0-16, the second byte controls locations 20-36, and so on. The bytes are read from left to right. Thus, if locations 0-3 are protected, the first byte of the table contains:

11000000

Note that only individual words are protected, not bytes. Thus, protecting location 0 always implies that the word at location 0 is protected, meaning both locations 0 and 1. If locations 24 and 26 are protected, the second byte of the table contains:

00110000

since the leftmost bit represents location 20 and the rightmost bit represents location 36. To protect locations 300-306, the leftmost 4 bits of byte 342 must be set:

11110000

resulting in a value of 360 for that byte.

2.5.4.1 S/J Restrictions - The S/J Monitor does not support the .PROTECT request. If users wish to protect vectors, the protection must be done in one of two ways:

1. Manually, with PATCH, or
2. Dynamically (from within the user's program)

To protect locations 300-306 dynamically, the following instructions are used:

```
MOV @#54,R0
BISB #11110000,342(R0)
```

Protecting locations with PATCH implies that the vector is permanently protected, even if the system is re-bootstrapped, while the second method provides a temporary measure and does not hold across bootstraps. However, users are cautioned that the second method involves storing directly into the monitor; for this reason it is recommended that S/J users use method 1.

2.6 USING AUXILIARY TERMINALS AS THE CONSOLE TERMINAL

This section describes how RT-11 can be modified to allow a terminal other than the standard console unit 0 to become the console terminal. This procedure is useful in cases where it is desirable to be able to use different console capabilities at different times (for example, at certain times the hard copy output of an LA30 is required, while at other times the speed of a VT05 is desirable). The only information required to make the alteration is:

- 1) the address of the auxiliary terminal's interrupt vectors, and
- 2) the I/O page addresses of the keyboard and printer status register and buffer.

RT-11 is designed so that all console references are done indirectly through centralized pointers. Thus, changing several system locations causes all operations to be transferred to a new terminal.

For this example, assume that the new terminal's interrupt vectors are at 300,302 and 304,306 and that its I/O page addresses are:

TKS at 177500
TKB at 177502
TPS at 177504
TPB at 177506

Also assume that the new terminal is a parallel interface so that no fill characters are required.

.R PATCH <CR>

PATCH Version number

FILE NAME--

*MONITR.SYS/M<CR>

*BASE;ØR<CR>

*6Ø/ VECTIN<LF>

62/ STATIN<LF>

64/ VECTOUT<LF>

66/ STATOUT<CR>

*3ØØ/ nnnnn VECTIN<LF>

3Ø2/ nnnnn STATIN<LF>

3Ø4/ nnnnn VECTOUT<LF>

3Ø6/ nnnnn STATOUT<CR>

*Ø,xx3Ø4/ 17756Ø 1775ØØ<LF>

Ø,xx3Ø6/ 177562 1775Ø2<LF>

Ø,xx31Ø/ 177564 1775Ø4<LF>

Ø,xx312/ 177566 1775Ø6<CR>

*Ø,xx342\ Ø 36Ø<CR>

*E

:

[The current values for the BASE address and for the input/output vectors and status are in Table 2 of RT-11 System Release Notes. They must be copied into the new terminal's vectors.] [nnnnn are arbitrary numbers]

[xx = 16 for S/J, 17 for F/B. Modify monitor's central I/O page pointers]

[Protect new vectors]

The bootstrap must also be changed to relocate the new vector locations when the monitor is first loaded into memory. The bootstrap contains a list of items that must be relocated; the list is located at RELLST in the bootstrap code. The exact position of RELLST varies with each monitor and must be obtained from Table 2 of RT-11 System Release Notes (V02C). The patching procedure is:

.R PATCH <CR>

PATCH Version number

FILE NAME--

*MONITR.SYS/M<CR>

*RELLST+1Ø/ 6Ø 3ØØ<LF>

RELLST+12/ 64 3Ø4<CR>

*E

.R PIP<CR>

*A=MONITR.SYS/U<CR>

*SY:/O<CR>

[Bootstrap must be rewritten. Rebootstrap; system will appear on new terminal.]

It is also possible to write a user program that would perform this procedure dynamically at run-time. Such a program would modify the monitor's protection map and the central I/O page pointers, then set up locations 300-306 and exit. If done dynamically, the monitor file itself is unchanged; thus when the system is bootstrapped, the console terminal reverts to the usual unit.

2.7 MAKING TTY SET OPTIONS PERMANENT IN F/B MONITOR

The F/B Monitor may be configured for different console terminal requirements by use of the TTY options of the SET command. These changes are not permanent and must be made each time the monitor is bootstrapped. By using the patching procedures in this section, the various options required for the installation may be made a permanent part of the F/B Monitor.

Table 2-4 is a description of the TTY options and their default functions in the F/B Monitor as distributed.

Table 2-4
Default Functions for TTY Options

Option	Default	Description
TAB/NOTAB	NOTAB	Hardware tabs converted to spaces.
CRLF/NOCLRF	CRLF	<CR><LF> inserted if WIDTH reached.
FORM/NOFORM	NOFORM	Form Feed converted to Line Feeds.
FB/NOFB	FB	CTRL F/CTRL B cause context switch.
PAGE/NOPAGE	PAGE	CTRL S holds output, CTRL Q continues it.
SCOPE/NOSCOPE	NOSCOPE	VT05, VT50, VT11 is the console terminal (rubout produces backspace, space, backspace).
WIDTH	72(10)	Width of carriage.

The three options enabled are PAGE, CRLF, and FB. The carriage width is set to 72(10) characters (110 octal).

To permanently change these options, the words TTCNFG, TTWIDT and LISTFB in the F/B Monitor must be patched. The exact locations of these words and the BASE address are found in Table 2 of RT-11 System Release Notes (V02C). The numbers used in the following examples are for illustration purposes only and may not be correct for all systems.

2.7.1 Carriage Width

The carriage width is the line width at which the CTRL option generates a carriage return/line feed. This width is changed by patching the word TTWIDT, which for this example is assumed to be located at 21410. See Table 2 of RT-11 System Release Notes (V02C) for the exact locations of BASE and TTWIDT.

```
.R PATCH <CR>
```

PATCH Version number

FILE NAME--

```
*MONITR.SYS/M<CR>
```

[The /M is necessary; set
relocation registers; open
with backslash]

```
*BASE;ØR<CR>
```

```
*Ø,2141Ø\ _____ 11Ø _____ 2Ø4<CR>
```

```
*E
```

```
:
```

In this example, the width is changed from 72₁₀ to 132₁₀ (204₈).

2.7.2 Other Options

Other options are changed by setting or clearing the appropriate bits in TTCNFG. To determine the new value to be inserted in TTCNFG, Table 2-5 is used. For each option, select the permanent value desired. Add together the octal bit patterns for each value selected to determine the new value of TTCNFG.

Table 2-5
TTCNFG Option Bits

Option	Bit Pattern
TAB	000001
CRLF	000002
FORM	000004
FB	000010
PAGE	000200
SCOPE	100000
Any NO option	000000

For example, the monitor default is PAGE, CRLF and FB. Adding together the bit patterns for PAGE, CRLF and FB produces the octal value 212 (= 200 + 10 + 2).

To change this to SCOPE, PAGE, FB, add together the numbers 100000, 200 and 10 to get 100210, the new value of TTCNFG. Using the location of TTCNFG obtained from Table 2 of RT-11 System Release Notes is:

```
.R PATCH <CR>
PATCH Version number
FILE NAME--
*MONITR.SYS/M<CR>
*BASE;ØR<CR>
*Ø,TTCNFG/_____212_____1ØØ21Ø<CR>
*E
:
```

If the FB option is changed, an additional step is necessary. Bit 15 of LISTFB must be changed to reflect the new FB option. Bit 15 must be 0 if the option is FB and must be 1 if the option is NOFB. For example, to change the monitor default to FORM, TAB, NOFB, the value of TTCNFG is 5 (4 + 1 + 0), and bit 15 of LISTFB must be a 1. The patch procedure is:

.R PATCH <CR>

PATCH Version number

FILE NAME--

*MONITR.SYS/M<CR>

[The /M is necessary;
set relocation register;
change TTCNFG;
set bit 15 in LISTFB.]

*BASE;ØR<CR>

*Ø,TTCNFG/ 212 5<CR>

*Ø,LISTFB/ 3316 1Ø3316<CR>

*E

:

After making any of these patches, it is necessary to bootstrap the system to load the new version of the monitor.

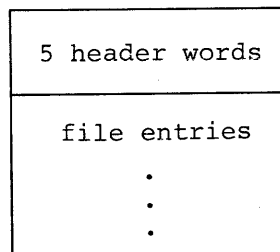
CHAPTER 3

FILE STRUCTURES AND FILE FORMATS

3.1 DEVICE DIRECTORY SEGMENTS

The device directory begins with physical block 6 of any directory-structured device and consists of a series of directory segments that contain the names and lengths of the files on that device. The directory area is variable in length, from 1 to 31 (decimal) directory segments. PIP allows specification of the number of segments when the directory is zeroed. The default value is four directory segments. Each directory segment is made up of two physical blocks; thus, a single directory segment is 512 words in length.

A directory segment has the following format:



3.1.1 Directory Header Format

Each directory segment contains a 5-word header block, leaving 507 (decimal) words for directory entries. The contents of the header words are described in Table 3-1.

Table 3-1
Directory Header Words

Word	Contents
1	The number of segments available for entries. This number is specified in PIP when the device is zeroed and must be in the range $1 \leq N \leq 31_{10}$.
2	Segment number of the next logical directory segment. The directory may, in certain cases, be a linked list. This word is the link word between logically contiguous segments; if equal to 0, there are no more segments in the list. Refer to Section 3.2.1, Directory Segment Extensions, for more details on the link word.
3	The highest segment currently open (each time a new segment is created, this number is incremented). This word is updated only in the first segment and is unused in any but the first segment.
4	The number of extra bytes per directory entry. This number can be specified when the device is zeroed with PIP. Currently, RT-11 does not allow direct manipulation of information in the extra bytes.
5	Block number where files in this segment begin.

3.1.2 Directory Entry Format

The remainder of the segment is filled with directory entries. An entry has the following format:

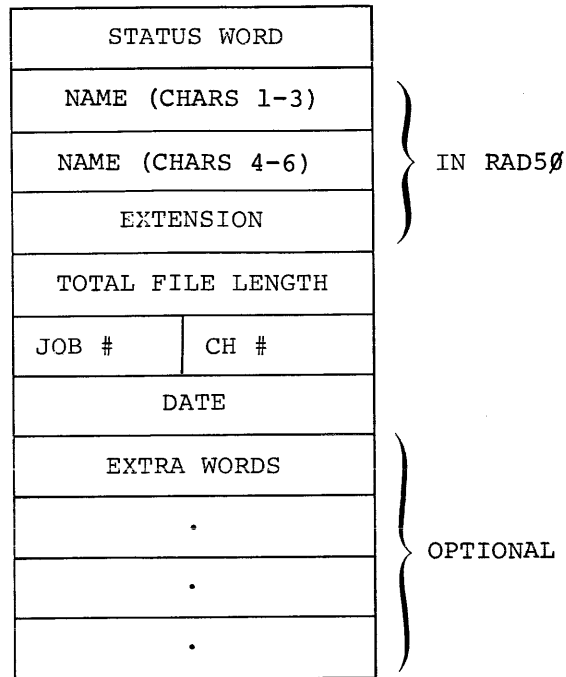


Figure 3-1
Directory Entry Format

3.1.2.1 Status Word - The Status Word is broken down into two bytes of data:

Even byte: Reserved for future use.

Odd byte: Indicates the type of entry. Currently RT-11 recognizes the file types listed in Table 3-2:

Table 3-2
File Types

Value	File Type
1	Tentative File, i.e., one that has been .ENTERed but not .CLOSEd. Files of this type are deleted if not eventually .CLOSED and are listed by PIP as <UNUSED> files.
2	An empty file. The name, extension, and date fields are not used. PIP lists an empty file as <UNUSED> followed by the length of the unused area.

(continued on next page)

Table 3-2 (Cont.)

File Types

Value	File Type
4	A permanent entry. A tentative file that has been .CLOSEd is a permanent file. The name of a permanent file is unique; there can be only one file with a given name and extension. If another exists before the .CLOSE is done, it is deleted by the monitor as part of the .CLOSE operation.
10	End-of-segment marker. RT-11 uses this to determine when the end of the directory segment has been reached during a directory search.

3.1.2.2 Name and Extension - These three words (in .RAD50) represent the symbolic name and extension assigned to a file.

3.1.2.3 Total File Length - The file length consists of the number of blocks currently a part of the file. Attempts to read or write outside the limits of the file result in an End of File error.

3.1.2.4 Job Number and Channel Number - A tentative file is associated with a job in one of two ways:

1. Under the S/J Monitor, the sixth word of the entry holds the channel number on which the file is open. This enables the monitor to locate the correct tentative entry for the channel when the .CLOSE is given. The channel number is loaded into the even byte of the sixth word.
2. In F/B, the channel number is put into the even byte of the sixth word; in addition, the number of the job that is opening the file is put into the odd byte of the word. This is required to uniquely identify the correct tentative file during the .CLOSE and is necessary because both jobs may have files open on their respective channels; the job number differentiates the tentative files.

NOTE

This sixth word is used only when the file is marked as tentative. Once it becomes permanent, the word becomes unused. Its function while permanent is reserved for future use.

3.1.2.5 Date - When a tentative file is created via .ENTER, the system date word is put into the creation date slot for the file. The date word is in the following format:

	15	14			10	9			5	4		0
U N U S E D	MONTH (1-12.)				DAY (1-31.)				YEAR-110(8)			

3.1.2.6 Extra Words - The number of extra words is determined by the number of extra bytes per entry in the header words. Although PIP provides for allocation and listing of extra words, RT-11 provides no direct facilities for manipulating this extra information. Any user program wishing to access these words must perform its own direct operations on the RT-11 directory.

Figure 3-2 shows a typical RT-11 directory segment:

HEADER BLOCK	4	FOUR SEGMENTS AVAILABLE
	0	NO NEXT SEGMENT
	1	HIGHEST OPEN IS #1
	0	NO EXTRA WORDS/ENTRY
	16	FILES START AT BLOCK 16 ₈
FILE ENTRIES	2000	PERMANENT ENTRY
	51646	RAD5Ø FOR "MON"
	35562	RAD5Ø FOR "ITR"
	75273	RAD5Ø FOR "SYS"
	42	FILE IS 34 ₁₀ (42 ₈) BLOCKS LONG
	0	
	0	NO CREATION DATE
	1000	AN EMPTY ENTRY
	0	(THE NAME AND EXTENSION OF AN
	0	EMPTY IS NOT IMPORTANT
	0	
	100	64 _{1Ø} (1ØØ ₈) BLOCKS LONG
	0	
	0	
	2000	PERMANENT
	62570	RAD5Ø FOR "PIP"
	0	
	50553	RAD5Ø FOR "MAC"
	11	FILE IS 9 ₁₀ (11 ₈) BLOCKS LONG
	0	
	0	NO CREATION DATE
	4ØØ	TENTATIVE FILE ON CHANNEL 1
	62570	RAD5Ø FOR "PIP"
	0	
	50553	RAD5Ø FOR "MAC"
	20	
	1	JOB #, CHANNEL #
	0	
	1000	EVERY TENTATIVE MUST BE FOLLOWED BY
	0	AN EMPTY ENTRY
	0	
	0	
	1020	FILE IS 528 _{1Ø} (1Ø2Ø ₈) BLOCKS LONG
	0	
	0	
	4000	END OF DIRECTORY SEGMENT

Figure 3-2
Directory Segment

When the tentative file PIP.MAC is .CLOSEd, the permanent file PIP.MAC is deleted.

To find the starting block of a particular file, first find the directory segment containing the entry for the desired file. Then take the starting block number given in the fifth word of that directory segment and add to it the length of each file in the directory before the desired file. For example, in Figure 3-2, the permanent file PIP.MAC will begin at block number 160 (octal).

3.2 SIZE AND NUMBER OF FILES

The number of files that can be stored on an RT-11 device depends on the number of segments in the device's directory and the number of extra words per entry. The maximum number of directory segments on any RT-11 device is 31_{10} . This theoretically leaves room for a maximum of:

$$31 \times \left[\frac{512-5}{7+N} \right]$$

directory entries, where N equals the number of extra information words per entry. If $N=0$, this indicates that the maximum is 2232_{10} entries.

If files are added sequentially (that is, one immediately after another) without deleting any files, roughly one half the total number of entries will fit on the device before a directory overflow occurs. This results from the way filled directory segments are handled.

When a directory segment becomes full and it is necessary to open a new segment, approximately one half the entries of the filled segment are moved to the newly-opened segment (this process is illustrated in Section 3.2.1); thus, when the final segment is full, all previous segments have approximately one half their total capacity. If this process were not done and a file was deleted from a full segment, the space from the deleted file could not be reclaimed. Every tentative file must be followed by an empty entry (for recovering unused blocks when the file is made permanent). Though only one file is deleted, two entries (tentative and empty) are needed to reclaim the space.

If files are continuously added to a device, the maximum number of entries will be:

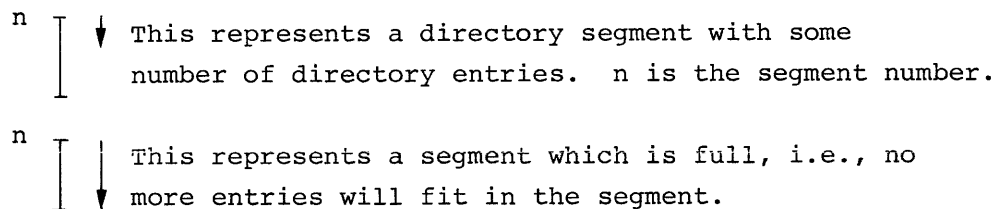
$$(M+1) \left\lceil \frac{507}{2(7+N)} \right\rceil$$

where M equals the number of segments available on the device and N equals the number of extra words.

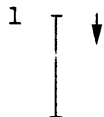
The theoretical total can be realized by compressing the device (using the PIP /S operation) when the directory fills up. PIP packs the directory segments as well as the physical device.

3.2.1 Directory Segment Extensions

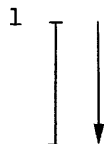
RT-11 allows a maximum of 31 (decimal) directory segments. This section covers the processing of a directory segment. For illustrative purposes, the following symbols are used:



Systems start out with entries entered into segment 1:



As entries are added, segment 1 fills:



When this occurs and an attempt is made to add another entry to the directory, the system must open another directory segment. If another segment is available, the following occurs:

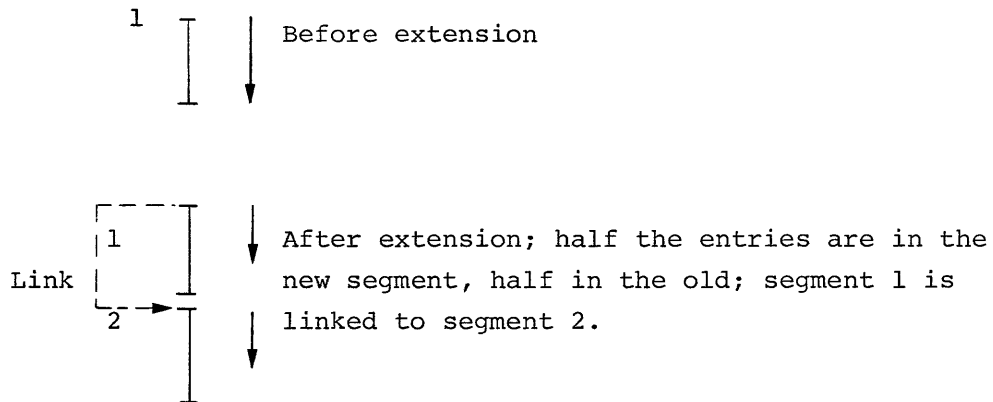
1. one half of the entries from the filled segment are put into the next available segment,
2. the shortened segment is re-written to the disk,
3. the directory segment links are set, and
4. the file is entered in the newly created segment.

NOTE

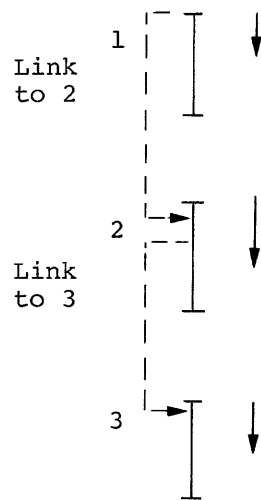
If the last segment becomes full and an attempt is made to enter another file, a fatal error occurs and an error message is generated:

?M-DIR OVFL0?

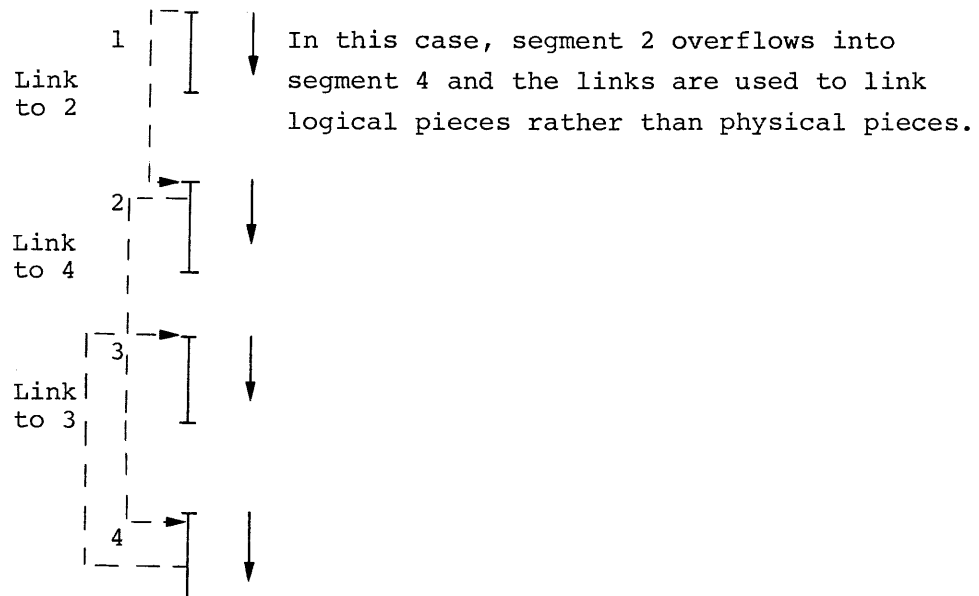
Thus, in the normal case, the segment appears as:



If many more files are entered, they fill up the second segment and overflow into the third segment, if it is available:



In this case, the links between the segments are not strictly necessary, as the segments are contiguous. However, the links do become necessary if a large file is deleted from segment 2 and many small files are entered, since it would then be possible to overflow segment 2 again. If this occurred and a fourth segment existed, the directory would appear:



3.3 MAGTAPE AND CASSETTE FILE STRUCTURE

3.3.1 Magtape File Structure

This section covers the magtape file structure as implemented in RT-11, Versions 2B and 2C. The structure is slightly different from that of Version 2. However, RT-11 V02B and V02C can read magtapes written under Version 2.

RT-11 magtapes use a subset of the VOL1, HDR1, and EOF1 ANSI standard labels. Each magtape file has the format:

```
HDR1*---data---*EOF1*
```

where each asterisk represents a tape mark.

A volume containing a single file has the following format:

```
VOL1 HDR1*---data---*EOF1**
```

A volume containing two files has the following format:

```
VOL1 HDR1*---data---*EOF1*HDR1*---data---*EOF1**
```

A double tape mark following an EOF1 label indicates logical end of tape.

A zeroed magtape has the following format:

```
VOL1**
```

Each label occupies the first 80 bytes of a 256-word physical block, and each byte in the label contains an ASCII character (i.e., if the content of a byte is listed as '1', the byte contains the ASCII code for '1'). Table 3-3 shows the contents of the first 80 bytes in the three labels. Note that VOL1, HDR1, and EOF1 each occupy a full 256-word block, of which only the first 80 bytes are meaningful.

The meanings of the table headings are:

CP - character position in label
Field Name - reference name of field
L - length of field in bytes
Content - content of field

Table 3-3
ANSI MT Labels Under RT-11

Volume-Header Label (VOL1)			
CP	Field Name	L	Content
1-3	Label identifier	3	VOL
4	Label number	1	1
5-10	Volume identifier	6	RT1101
11	Accessibility	1	Blank
12-37	(Reserved)	26	Blanks
38-51	Owner identifier	14	DD%% {used to indicate an {RT-11 MT to RSX-11D
52-79	(Reserved)	28	Blanks
80	Label-Standard Version	1	1
First File Header Label (HDR1)			
CP	Field Name	L	Content
1-3	Label identifier	3	HDR
4	Label number	1	1
5-21	File identifier	17	6-character ASCII file name, followed by '.', followed by 3-character ASCII file extension; left justified, remainder of field is blanks
22-27	File Set identifier	6	RT1101
28-31	File Section Number	4	0001
32-35	File Sequence Number	4	0001
36-39	Generation Number	4	0001
40-41	Generation Vsn Number	2	00
42-47	Creation Date	6	Blank then year*1000+day of year in ASCII (ΔYYDDD); e.g., 2/1/75=Δ75032
48-53	Expiration Date	6	blank then 00000
54	Accessibility	1	blank
55-60	Block Count	6	000000
61-73	System Code	13	RT11 left-justified followed by blanks
74-80	(Reserved)	7	blanks
First End-of-File Label (EOF1)			
Same as HDR1 except that the label identifier (CP 1-3) is <i>EOF</i> , not <i>HDR</i> , and the block count field (CP 55-60) contains the number of blocks in the file as a decimal value encoded in ASCII characters (for example, if the file was 12 blocks long, the block count field would be 00012).			

3.3.1.1 Bootable Magtape File Structure - An RT-11 bootable magtape is a multi-file volume that has the following format:

```
VOL1 BOOT HDRI*---data---*EOF1**
```

where BOOT is a 256-word physical block containing the magtape bootstrap loaders.

The format of the bootable magtape is not standard, because of the BOOT block, but other systems that will skip the BOOT block to HDRI will be able to read RT-11 bootable magtapes if they can read regular RT-11 magtapes.

3.3.1.2 Moving MT to Other Industry-Compatible Environments - RT-11 V02C magtapes may be read by RSX-11D Version 6. RT-11 magtapes should be mounted, under RSX-11D, by using the /OVR switch of the MOUNT command, or by specifying a volume label of "RT1101". RSX-11D Version 6 will not allow the user to write on RT-11 V02B magtapes once they have been mounted. RT-11 V02C can read RSX-11D Version 6 magtapes, but RT-11 users should not attempt to write on tapes created by RSX-11D. Users should note that data structures differ between the two systems and these differences must be handled by the user.

RT-11 V02C magtapes may be read on IBM systems that support ANSI standard label processing. RT-11 V02C magtapes to be read by IBM systems should consist of single file volumes (one file per magtape). Important JCL parameters for reading RT-11 V02C tapes under an IBM OS system are as follows:

(In the DD statement of the Job Control Language)

DISP = OLD

LABEL = (01,AL,,IN)

VOL = (,RETAIN,SER=RT1101)

DSN = RTFILE.MAC

BLKSIZE = 512

DEN = 2 (for 800 bpi 7-track or 9-track tape)

The DSN parameter is the Data Set Name or the RT-11 filename and extension. Files to be moved to other systems should be created with full 6-character filenames and 3-character extensions; filenames less than 6 characters should be enclosed in quotes.

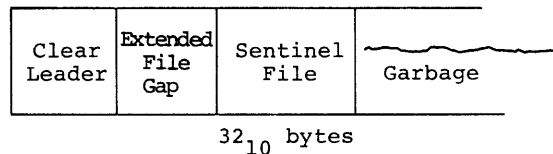
3.3.1.3 Recovering From Bad Tape Errors - When a bad tape error occurs on magtape, the magtape handler will retry the desired function, and, if the error persists, will attempt to save the tape's file structure. It does this on writes, for example, by retrying the write 10 times, using the write with extended file gap to space past the bad tape. If, after retrying, the error still exists, the file will be closed, containing all data written prior to the write on which the error occurred. The user should still be able to write additional files on the tape, since the bad portion of the tape will be within the area of the closed file.

If a bad tape error occurs when writing the file header during ENTER, and retry fails, the handler writes logical end of tape after the previous file on the tape. The remainder of the tape can be accessed only if the last complete file on the tape can be extended (or overwritten by a file of different length) so that the bad tape error does not occur on the file header when a subsequent file is ENTERED.

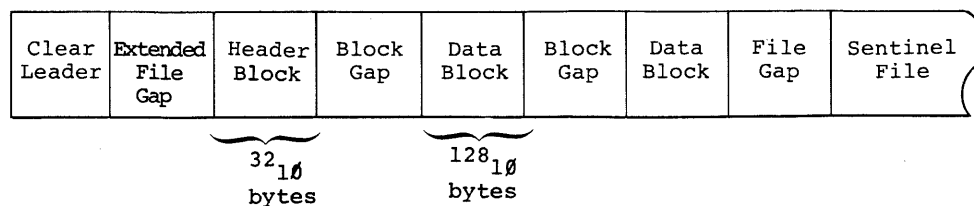
If a bad tape error occurs while writing the end of file label (EOF1) during CLOSE, the handler writes a triple tape mark to signify end of file and logical end of tape. Additional files can be added to the tape only if the last complete file can be extended (or overwritten by a file of different length) so that the bad tape error does not occur at the EOF1 label.

3.3.2 Cassette File Structure

A blank (newly initialized) cassette appears in the format:

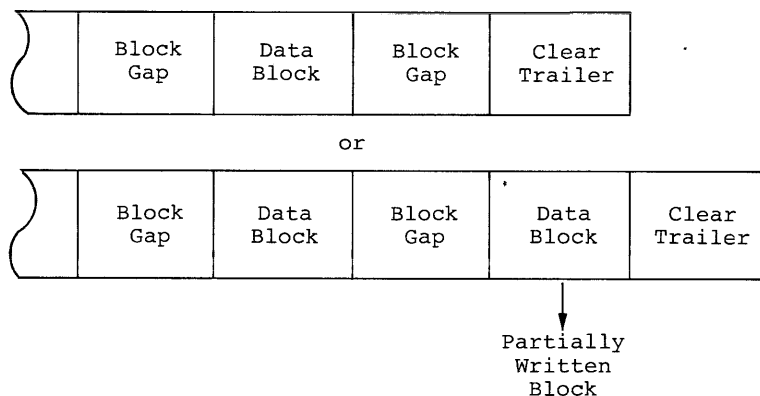


while a cassette with a file on it appears as:



Files normally have data written in 128₁₀-byte blocks. This can be altered by writing cassettes while in *hardware* mode. (In hardware mode, the user program must handle the processing of any headers and sentinel files; in *software* mode the handler automatically does this. Refer to Appendix H of the RT-11 System Reference Manual.)

The preceding diagram shows a file terminated in the usual manner (by a sentinel file). However, the physical end of cassette may occur before the actual end of the file. This format appears as:



In the latter case, for multi-volume processing the partially written block must be the first data block of the next volume.

3.3.2.1 File Header - The File Header is a 32₁₀-byte block that is the first block of any data file on a cassette. If the first byte of the header is null, the header is interpreted as a sentinel file, which is an indication of logical end of cassette. The format of the header is described in Table 3-4.

Table 3-4
CT File Header Format

Byte Number	Contents
0-5	File name in ASCII characters (ASCII is assumed to imply a 7-bit code)
6-8	Extension in ASCII characters
9	Data type (0 for RT-11)
10,11	Block length of 128_{10} (200_8); Note: byte 10=0 (high-order), byte 11= 200_8 (low-order)
12	File sequence number. (0 for a single-volume file or the first volume of a multi-volume file; successive numbers are used for continuations)
13	Level 1; this byte is a 1
14-19	Date of file creation (6 ASCII digits representing day (01-31); month (01-12), and last two digits of the year; 0 or 40_8 in first byte means no date present)
20,21	Zero
22	Record attributes (0 in RT-11 cassettes)
23-28	Reserved for future use
29-31	Reserved for user

3.4 RT-11 FILE FORMATS

3.4.1 Object Format (.OBJ)

An object module is a file containing a program or routine in a binary, relocatable form; object files normally have an .OBJ extension. Object modules are produced by language processors (such as MACRO or FORTRAN) and are processed by the Linker to become a runnable program (in SAV, LDA, or REL format, discussed later). Object files may also be processed by the Librarian to produce library .OBJ files, which are then used by the Linker. Figure 3-3 illustrates this process.

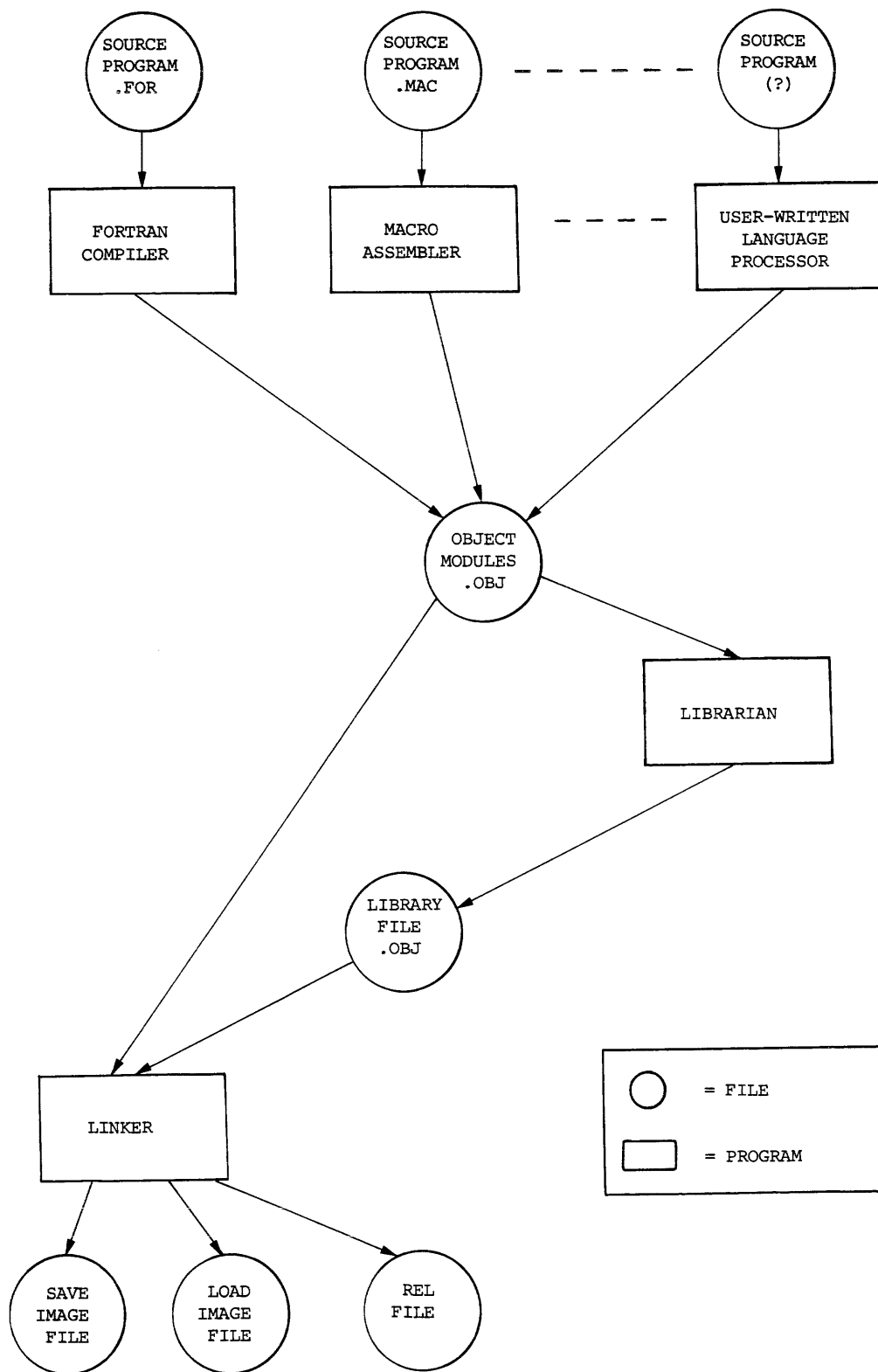


Figure 3-3
Object Module Processing

Many different object modules may be combined to form one file; each object module remains complete and independent. However, object modules combined into a library by the Librarian are no longer independent -- they become part of the library's structure.

Object modules are made up of formatted binary blocks. A formatted binary block is a sequence of 8-bit bytes (stored in an RT-11 file, on paper tape, or by some other means) and is arranged as illustrated in Figure 3-4.

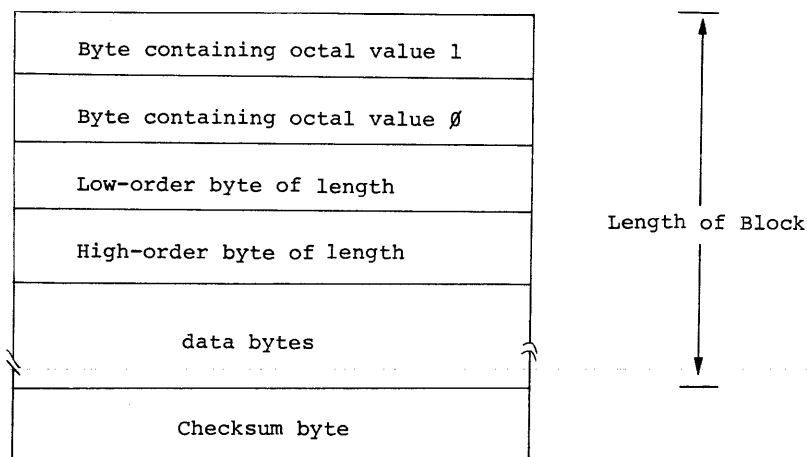


Figure 3-4
Formatted Binary Block

Each formatted binary block has its length stored within it; the length includes all bytes of the block except the checksum byte. The data portion of each formatted binary block contains the actual object module information (described later). The checksum byte is computed such that the sum of all bytes in the formatted binary block, including the checksum byte, is zero when the sum is masked to 8 bits.

Formatted binary blocks are used to hold various kinds of information in an object module; this information is always contained completely in the data portion of the block, surrounded by the formatted binary block structure.

Eight types of data blocks may be present in an object module:

<u>Identification Code</u>	<u>Type of Block</u>	<u>Function</u>	
1	GSD blocks	hold the Global Symbol Directory information	
2	ENDGSD block	signals the end of GSD blocks in a module	
3	TXT blocks	hold the actual binary "text" of the program	
4	RLD blocks	hold Relocation Directory information	
5	ISD blocks	hold Internal Symbol Directory - not supported by RT-11	
6	ENDMOD block	signals end of the object module	
7	Librarian Header Block	17 words holding the status of the library file	Library File Only
10	Librarian End Block	signals the end of the library file	

The structure of object modules produced by a language processor will be described first, followed by details specific only to Library .OBJ files.

The first block of an object module must be a GSD block, and all GSD blocks must appear before the ENDGSD block. The ENDMOD block must be the last block of the module. Except for these three restrictions, blocks may appear in any order within an object module.

When a 16-bit word is stored as part of the data in a block, it is always stored as two consecutive 8-bit bytes, with the low-order byte first.

The first word (data word) of each type of block mentioned above contains the identification code of that block type (1 = GSD block, etc.) with any information present following the identification word.

3.4.1.1 Global Symbol Directory - The object module's global symbol directory contains the following information:

- 0 - Module Name
- 1 - Program Section (CSECT) Definitions
- 2 - Internal Symbol Table Name (not supported by RT-11)
- 3 - Transfer (Start) Address
- 4 - Global Symbol Definitions or References

Each piece of information in the GSD is contained in a *GSD item*, formatted as shown in Figure 3-5:

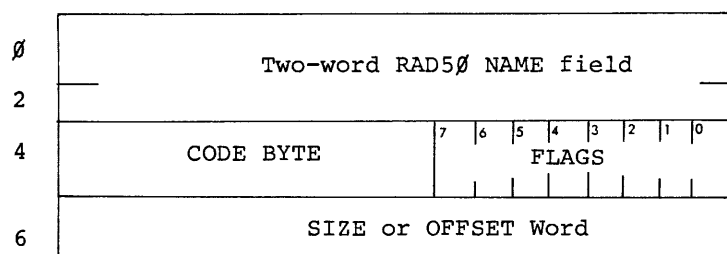


Figure 3-5
GSD Structure

The code byte identifies the information contained in a GSD item according to the codes listed above (0 = Module Name, 1 = Program Section Definition, etc.). The first GSD item of an object module must contain the Module Name information (FLAGS, CODE, and SIZE = 0). There may be no more than five GSD items per GSD block (i.e., per formatted binary block). As many GSD blocks as necessary may be present, but all must appear before the ENDGSD block. GSD blocks need not be contiguous.

Flags are coded as follows:

- Bits 0,1,2,4,7 unused
- Bit 3: 0 = undefined, 1 = defined (used only with Global Symbols)
- Bit 5: 0 = absolute, 1 = relocatable
- Bit 6: 0 = internal, 1 = global

All program sections (CSECTs) defined in a module must be declared in GSD items (code byte = 1). The size word of each program section definition should contain the size in bytes to be reserved for the section. Program sections may be declared more than once, in which case the largest declared size of the section will be used. All global symbols that are defined in a given program section must appear in the GSD items immediately following the definition item of that program section.

A special program section named ". ABS." (where represents a space) is called the absolute section. The absolute section has the special attribute that it is always allocated by the Linker beginning at location 0 of memory. All global symbols that contain absolute (non-relocatable) values should be declared immediately after the GSD item that defines the absolute section. If it is not desired to allocate any memory space to the absolute section, its size word may be specified as zero, even if absolute global symbol definitions occur after it. Flag bit 5 of each absolute global symbol is always set to zero. GSD items that contain the definitions of global symbols (code byte = 4) must immediately follow the program section declaration into which they are to be defined. Flag bit 3 is set to 1 to indicate a symbol definition, bit 5 is set if and only if the symbol is relocatable, and bit 6 is set to indicate that the symbol being defined is a global. In addition, the offset word is set to contain the defined value of the global symbol, relative to the base of the program section in which the global is defined. At link time, the Linker assigned section base is added to get the final value of the global symbol.

Global symbols that are referenced but not defined in the current object module must also appear in GSD items. These *global references* may appear in any GSD item except the very first (which contains the module name). Global references are recognized by code byte 4 with flag bit 3=0, bit 5 is undetermined, and bit 6=1. All global symbols used in the RLD of the object module (described later) must appear in at least one Global Symbol or Program Section GSD item.

If RT-11 is to begin execution of a program within a particular object module of that program, then the information on where to start is given in a Transfer Address (code=3) GSD item. The first even transfer address encountered by the Linker will be passed to RT-11 as the program start address. Whenever the resulting program is run (using R or RUN

for SAV images, FRUN for REL files, or the absolute loader for LDA files), the start address is used to indicate the first executable instruction. If no transfer address is present or if all are odd, the resulting program will not self-start when run. In a Transfer Address GSD item, the name field is used to specify a program section (or global name) and the offset word is used to indicate the offset from the base of that program section (or global) to the starting point of the program. The program section or global name referenced need not be defined in the current object module, but must be defined in some object module included at link time.

NOTE

Program Section and Global names must begin with an alphabetic or numeric character, except for the names `.ABS` and `ABS`.

3.4.1.2 ENDGSD Block - The ENDGSD block contains a single data word, and that is the identification code of the ENDGSD block (2). All GSD blocks in an object module must precede the ENDGSD block.

3.4.1.3 TXT Blocks and RLD Blocks - The first TXT block (3) in an object module (if present) must be preceded by an RLD block (4).

TXT blocks contain the actual binary form of the programs and are formatted as shown in Figure 3-6:

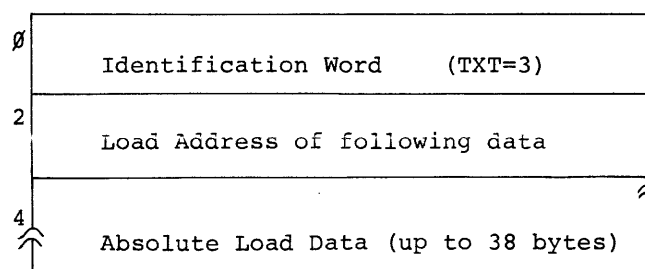


Figure 3-6
TXT Block Format

The load address of a TXT block gives the relative address of the first byte of the absolute load data. The address is relative to the base of the last program section given in a Location Counter Definition RLD command (explained later).

The Absolute Load Data contains the actual bytes that will be loaded into memory when the program is run (except for relocations, described later).

RLD blocks contain variable length RLD commands, used to modify and complete the information contained in TXT blocks. Except for the Location Counter commands, RLD information must appear in an RLD block immediately following the TXT block to be modified.

Available RLD commands are:

1. Internal Relocation
2. Global Relocation
3. Internal Displaced Relocation
4. Global Displaced Relocation
5. Global Additive Relocation
6. Global Additive Displaced Relocation
7. Location Counter Definition
8. Location Counter Modification (not used by RT-11)
9. Set Program Limits

The location counter commands (numbers 7 and 8) are the only two RLD commands that must appear in an RLD block preceding the text blocks modified. The first RLD block must precede the first TXT block and must contain only a location counter definition command (7) in order to declare a program section for loading the first text block. (The location counter modification command (8) is included for compatibility with other systems, but is not used by RT-11.)

The data portion of an RLD block must not be larger than 42_{10} bytes including the identification word (RLD=4) and all RLD commands.

All global names and program section names that appear in RLD commands must appear in GSD items in the same object module. Figure 3-7 shows the format of each RLD command (each part except the first word is optional and may not appear in some commands):

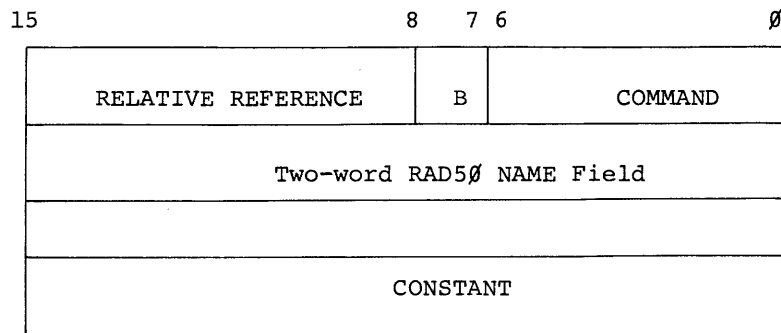


Figure 3-7
RLD Format

An RLD command may be 1, 2, 3, or 4 words long.

The Command Field contains the command code (1 = Internal Relocation, etc.). The Command Field occupies bits 0-6 of the first word of the command. The B field (bit 7) indicates a word command if 0 or a byte command if 1 (only valid for commands 1 through 6). The Relative Reference Field is a pointer into the preceding TXT block and is used with RLD commands that require text locations for modification (commands 1 through 6 and 9). This field specifies the displacement from the beginning of the preceding TXT block to the referenced text data byte (or word). The beginning of the TXT block is the identification word (the first word of the data portion of the block). Thus, the smallest relative reference will normally be 4 (the first byte (word) of the preceding TXT block).

The Name Field is used to hold a Global or Program Section name if the command requires it.

The Constant Field is used to hold a relative address or additive quantity if the command requires it. RLD commands are processed by the Linker as shown in the following situations:

1. Internal Relocation (code 1) - Add the current program section's base to the specified constant and place the result where indicated. This command relocates a direct pointer to an internal relocatable symbol.

Relative Reference	0/1	1
Constant		

Examples:

- a) .WORD LOCAL
- b) MOV #LOCAL,%Ø

2. Global Relocation (code 2) - Place the value of the specified global symbol where indicated. This command generates a direct pointer to an external symbol.

Relative Reference	Ø/1	2
Global Name		

Examples:

- a) .WORD GLOBAL
- b) MOV #GLOBAL,RØ

3. Internal Displaced Relocation (code 3) - Calculate the displacement from the position of the current location plus two to the specified absolute address, and store the result where indicated. This command occurs only when there is a reference to an absolute (non-relocatable) location from a relocatable section.

Relative Reference	Ø/1	3
Constant		

Examples:

- | | | |
|---|---|--|
| <ul style="list-style-type: none"> a) ABS=17755Ø TST ABS b) CLR 17755Ø | } | <p>both addresses cause
internal displaced
relocation to occur</p> |
|---|---|--|

4. Global Displaced Relocation (code 4) - Calculate the displacement from the current location plus two to the specified global address, and store the result where indicated.

Relative Reference	Ø/1	4
Global Name		

Example:

```
.GLOBL GLOBAL
MOV GLOBAL,RØ
```

5. Global Additive Relocation (code 5) - Add the value of the specified global symbol to the specified constant, and store the result where indicated.

Relative Reference	Ø/1	5
Global Name		
Constant		

Example:

```
.GLOBL GLOBAL
CMP #GLOBAL+6,RØ
```

6. Global Additive Displaced Relocation (code 6) - Calculate the displacement from the current location plus two to the address specified by the sum of the global symbol value and the given constant, and place the result where indicated.

Relative Reference	Ø/1	6
Global Name		
Constant		

Example:

```
.GLOBL GLOBAL
CLR GLOBAL+6
```

7. Location Counter Definition (code 7) - This command is used to specify the program section into which the following TXT blocks are to be loaded.

	7
Program Section Name	
Constant	

This command is generated whenever .ASECT or .CSECT is used to initiate or continue a program section. The constant word is effectively ignored by RT-11 and may be used for diagnostic purposes to indicate the relative point at which a program section is being entered.

8. Location Counter Modification (code 10₈) - This command is used to enter the current program section at a different point. This command is effectively ignored by RT-11 and is used for diagnostic purposes only.

	10
Constant	

Examples:

- a) . =100 ;IF WE ARE IN THE ASECT
b) . =.-20 ;IF WE ARE IN A RELOCATABLE SECTION

9. Set Program Limits (code 11₈) - This command (generated by the .LIMIT assembler directive) causes two words in the preceding TXT block to be modified. The first word is to be set to the lowest relocated address of the program. The second word is to be set to the address of the first free location following the relocated code. Note that both words to be modified must appear in the same TXT block.

Relative Reference	11
--------------------	----

In addition to the above commands, note that commands numbered 14₈, 15₈, and 16₈ can be generated by MACRO. These commands are identical to commands 4, 5, and 6 respectively, but are used when the *global* is really a program section name.

3.4.1.4 ISD Internal Symbol Directory - Not supported by RT-11.

3.4.1.5 ENDMOD Block - Every object module must end with an ENDMOD block. The ENDMOD block contains a single data word -- the identification code of the ENDMOD block (6).

3.4.1.6 Librarian Object Format - A library .OBJ file contains information additional to that previously defined. The object modules in a library file are preceded by a Library Header Block and Library Directory, and are followed by the Library End Block or trailer. This is illustrated in Figure 3-8.

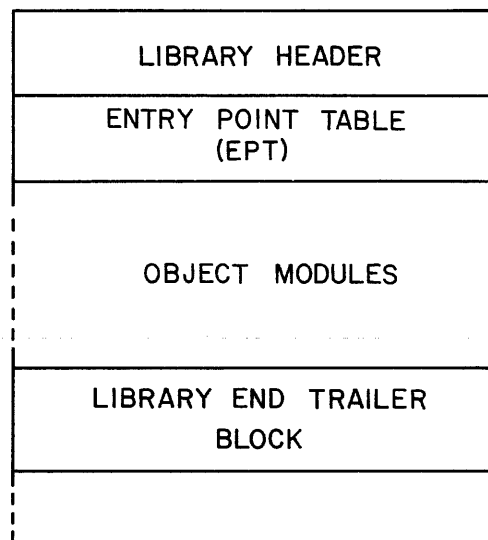


Figure 3-8
Library File Format

Diagrams of each component in the library file structure are included here, but Chapter 7 of the RT-11 System Reference Manual should be consulted for details.

The library header is composed of 17_{10} words describing the status of the file. The contents of the 17 words are shown in Figure 3-9.

I	} FORMATTED BINARY BLOCK HEADER
56 ₈	
7	LIBRARIAN CODE
X	VERSION NUMBER
0	RESERVED
X	MONTH-DAY-YEAR (OR Ø IF NO DATE)
0	} RESERVED
0	
0	
0	
0	
12 ₈	EPT RELATIVE START ADDRESS
X1	EPT ENTRIES ALLOCATED IN BYTES
0	EPT ENTRIES AVAILABLE (NOT USED IN V1)
X2	NEXT INSERT RELATIVE BLOCK NUMBER
X3	NEXT BYTE WITHIN BLOCK
0	NOT USED (MUST BE ZERO)

Figure 3-9
Library Header Format

The Entry Point Table (EPT), Figure 3-10, is composed of four-word entries which contain information related to all object modules in the library file.

0	SYMBOL CHARS 1-3 (RAD50)		BIT 15=1-MODULE NAME 0-CSECT OR ENTRY POINT NAME RELATIVE BYTE MAXIMUM=777 ₈ CSECTS MAXIMUM =177 ₈
2	SYMBOL CHARS 4-6 (RAD50)		
4		ADDRESS OF BLOCK	
6	# OF CSECTS IN OBJECT MODULE	RELATIVE BYTE IN BLOCK	

Figure 3-10
Entry Point Table Format

Object modules follow the Entry Point Table and consist of the types of data blocks already discussed: GSD, ENDGSD, TXT, RLD, and ENDMOD. The information in these blocks is used by the Linker during creation of the load module.

Following all object modules is a specially coded Library End Block (trailer), which signifies the end of the file, shown in Figure 3-11.

	1	FORMATTED BINARY HEADER
	10	FORMATTED BINARY LENGTH
	10	TYPE CODE
	0	NOT USED (MUST BE ZERO)
	357	CHECKSUM BYTE

Figure 3-11
Library End Trailer

3.4.2 Formatted Binary Format (.LDA)

The Linker /L switch produces output files in a paper tape compatible binary format.

Paper tape format, shown in Figure 3-12, is a sequence of formatted binary blocks (as explained in Section 3.4.1 and in Figure 3-4). Each formatted binary block represents the data to be loaded into a specific portion of memory. The data portion of each formatted binary block consists of the absolute load address of the block followed by the absolute data bytes to be loaded into memory beginning at the load address. There may be as many formatted binary blocks as necessary in an LDA file. The last formatted binary block of the file is special; it contains only the program start address in its data portion. If this address is even, the loader passes control to the loaded program at this address. If it is odd, the loader halts upon completion of loading. The final block of the LDA file is recognized by the fact that its length is 6.

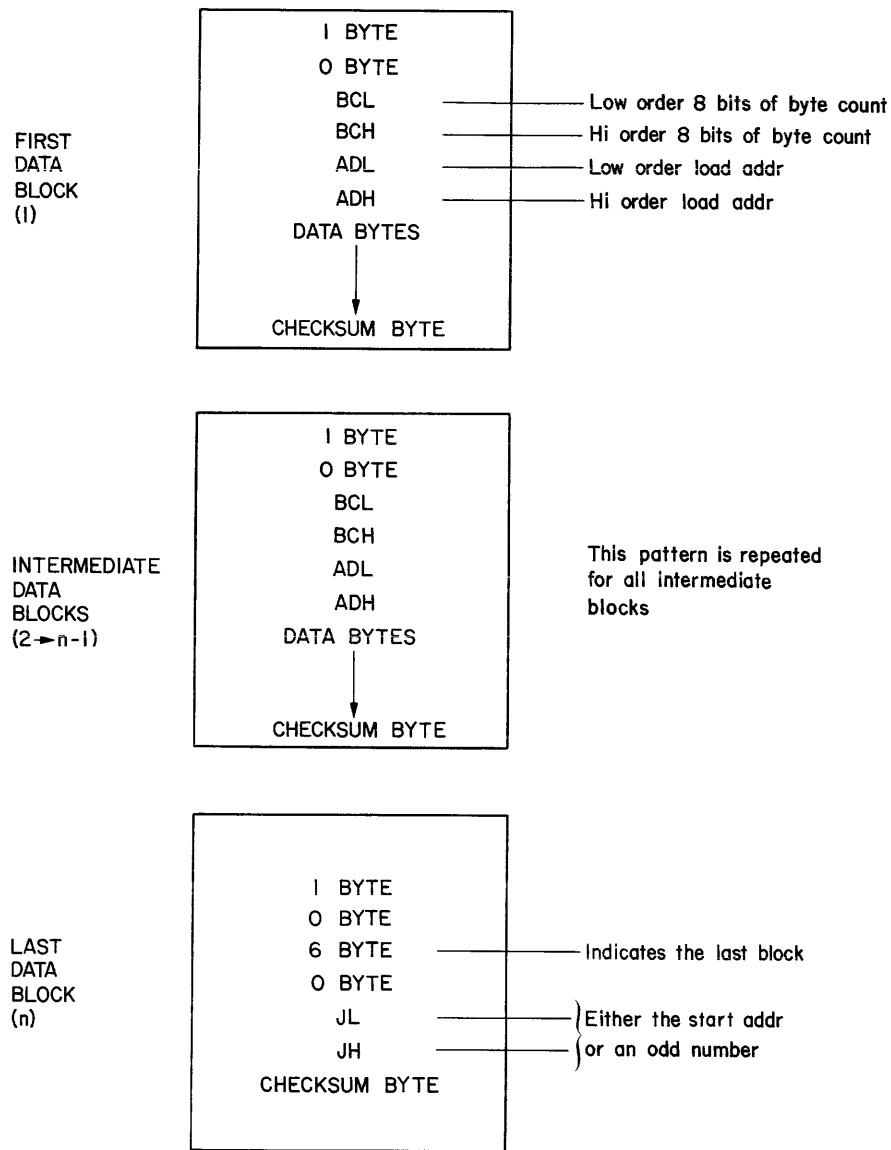


Figure 3-12
Formatted Binary Format

The load module's binary blocks contain only absolute binary load data and absolute load addresses; all global references have been resolved and the appropriate relocation has been performed by the Linker.

3.4.3 Save Image Format (.SAV)

Save image format is used for programs that are to be run in the back-ground. This format is essentially an image of the program as it would appear in memory (block 0 of the file corresponds to memory locations 0-776, block 1 to locations 1000-1776, and so forth).

Locations 360-377 in block 0 of the file are restricted for use by the system. The Linker stores the program memory usage bits in these eight words. Each bit represents one 256-word block of memory and is set if the program occupies that block of memory. This information is used by the R, RUN, and GET commands when loading the program.

When loading a save image program into memory, KMON reads block 0 of the file to extract the memory usage bits. These bits are used to determine whether the program will overlay either the KMON or the USR. If these portions of the monitor will not be overlaid, the entire program is loaded; if the USR and KMON must swap, KMON loads the resident portion of the program, up to the start of KMON. It then puts the portion of the program that overlays KMON/USR into the system swap blocks. When the program starts, the monitor swaps in the virtual portion of the program, overlaying KMON.

When block 0 of a save image file is loaded, each word is checked against the protection bit map (LOWMAP), which is resident in RMON. Locations that are protected in the map, such as location 54 and the system device vectors, are not loaded.

3.4.4 Relocatable Format (.REL)

A foreground job is linked using the Linker /R switch. This causes the Linker to produce output in a linked, relocatable format, with a REL file extension.

The object modules used to create a REL file have been linked and all global references have been resolved. The REL file is not relocated, so it has an effective start address of 0, with relocation information included to be used at FRUN time. The relocation information in the file is used to determine which words in the program must be relocated when the job is installed in memory.

In order to determine if the code to be relocated (as indicated in the relocation information blocks) is to have positive or negative relocation (relative to the start address of the program), the following criteria from the text modification commands is used (R = relative address, G = global address, C = constant):

1. Internal Relocation (.WORD R) - always positive relocation (absolute)
2. Global Relocation (.WORD G) - positive relocation only if the global is not absolute (global)

3. Internal Displaced Relocation - always negative relocation
(MOV 54,R)
4. Global Displaced Relocation - negative relocation only
(MOV G,R) where the global is de-
fined as absolute elsewhere
5. Global Additive Relocation - same as 2 above
(.WORD G + C)
6. Global Additive Displaced - same as 4 above
(MOV G + C,R)
7. Program Counter Commands - not applicable
8. Set Program Limits - always positive relocation
(requires 2 RELs; limit is
two words)

There are two types of REL files to consider, those programs with overlay segments and those without.

3.4.4.1 Non-Overlay Programs - A REL file for a non-overlaid program appears as shown in Figure 3-13:

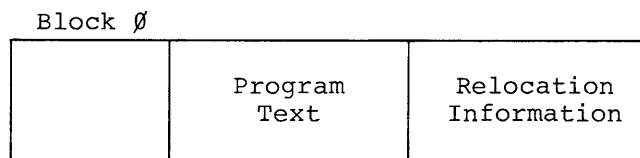


Figure 3-13
REL File Without Overlays

Block 0 (relative to start of the file) contains certain information required by the FRUN processor:

<u>Offset from Begin- ning of Block 0</u>	<u>Contents</u>
52	Size of the program root segment in bytes
54	Size of the overlay region in words; 0 if no overlays
56	REL file identification word, which must contain the RAD50 value of the characters 'REL'
60	Relative block number of relocation in- formation

In addition, the system communication locations (34-50) contain the following information:

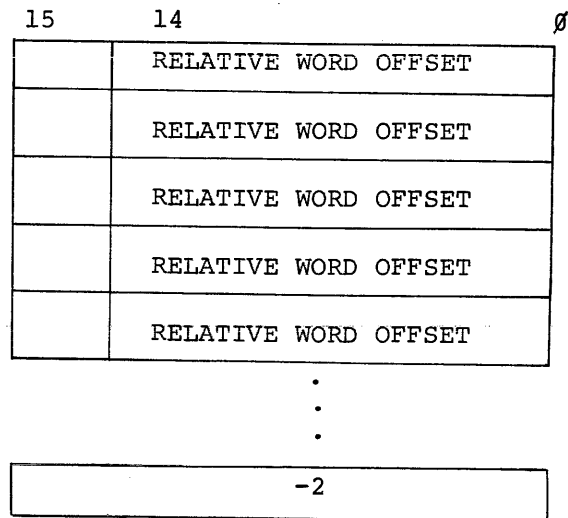
<u>Offset from Beginning of Block 0</u>	<u>Contents</u>
34,36	TRAP vector
40	Start address of program
42	Initial setting of stack pointer
44	Job Status Word
46	USR swap address
50	Highest memory address in user's program

In the case of non-overlaid programs, the FRUN processor performs the following general steps to install a foreground job.

1. Block 0 of the file is read into an internal monitor buffer.
2. The amount of memory required for the job is obtained from location 52 of block 0 of the file, and the space is allocated.
3. The program text is read into the space just allocated for it.
4. The relocation information is read into an internal buffer.
5. The locations indicated in the relocation information area are relocated by adding the relocation quantity, which is the starting address the job occupies in memory.

The relocation information consists of a list of addresses relative to the start of the user's program. This list is scanned, and the appropriate locations in the user's program area are updated with a constant. The job is then ready to be started.

The relocation information is in the following format:



Bits 0-14 represent the relative address to relocate divided by two. This implies that relocation is always done on a word boundary, which is the case. Bit 15 is used to indicate the type of relocation to perform, positive or negative. The relocation constant (which is the load address of the program) is added to or subtracted from the indicated location depending on the sense of bit 15; 0 implies addition, 1 implies subtraction. 177776 terminates the list of relocation information.

Following is an example of a simple, non-overlaid program linked to produce a REL file. A dump of the file follows the program.

```

.TITLE FTFST
.MCALL ..V2...REGDEF,.LOOKUP,.READW,.QSET,.PRINT,.EXIT
..V2..
.REGDEF
ST: .QSET #GLIST,#7
    .LOOKUP #AREA,#0,#PTR
    RCC 1$
    .PRINT #LKFAIL
    .EXIT
1$: .READW #AREA,#0,#RUFF,#256.,#0
    RCC 2$
    .PRINT #RDFAIL
    .EXIT
2$: .PRINT #OK
    .EXIT

GLIST: .BLKW 7*7
AREA: .BLKW 20.
PTR: .RAD50 /PR FILE12/
RUFF: .NLIST
    .REPT 256.
    .WORD 0
    .ENDR
    .LIST
LKFAIL: .ASCIZ /LOOKUP FAILED/
RDFAIL: .ASCIZ /READW FAILED/
OK: .ASCIZ /READW OK/
    .EVEN
    .NLIST
    .REPT <ST+1776-.>/2
    .WORD 0
    .ENDR
    .NLIST
.END ST

```

```

1
2
3
4 000000      ,MCALL  ,TITLE FTEST
5 000000      ,.V2...REGDEF,.LOOKUP,.READW,.QSET,.PRINT,.EXIT
6 000000      ,.V2...
7 000000      ,RFGDEF
8 000000      ST:      .QSET      #QLIST,#7
9 000000      012700  000007      .ITF NR <#7>,      MOV      #7,%0
10 000000      012746  000144'      MOV      #QLIST,-(6.)
11 000000      104353      FMT      #0353
12 000012      .LOOKUP #AREA,#0,#PTR
13 000012  012700  000306'      MOV      #AREA,%0
14 000016  112760  000001  000001      MOV      #1,1(0)
15 000024  105010      CLR      (0)
16 000026  012760  000356'  000002      .ITF NR <#PTR>,      MOV      #PTR,2.(0)
17 000034  005060  000004      CLR      4.(0)
18 000040  104375      FMT      #0375
19 000042  103004      RCC      1$
20 000044      .PRINT #LKFAIL
21 000044  012700  001364'      .ITF NR <#LKFAIL>,      MOV      #LKFAIL,%0
22 000050  104351      FMT      #0351
23 000052      .EXIT
24 000052  104350      FMT      #0350
25 000054      1$:      .READW #AREA,#0,#BUFF,#256.,#0
26 000054  012700  000306'      MOV      #AREA,%0
27 000060  112760  000010  000001      MOV      #8,1(0)
28 000066  105010      CLR      (0)
29 000070  012760  000000  000002      .ITF NR <#0>,      MOV      #0,2.(0)
30 000076  012760  000364'  000004      .ITF NR <#BUFF>,      MOV      #BUFF,4.(0)
31 000104  012760  000400  000006      .ITF NR <#256.>,      MOV      #256.,6.(0)
32 000112  012760  000000  000010      .ITF NR <#0>,      MOV      #0,8.(0)
33 000120  104375      .ITF NR <#X>,      FMT      #0375
34 000122  103004      RCC      2$
35 000124      .PRINT #RDFAIL
36 000124  012700  001402'      .ITF NR <#RDFAIL>,      MOV      #RDFAIL,%0
37 000130  104351      FMT      #0351
38 000132      .EXIT
39 000132  104350      FMT      #0350
40 000134      2$:      .PRINT #OK
41 000134  012700  001417'      .ITF NR <#OK>,      MOV      #OK,%0
42 000140  104351      FMT      #0351

```

```

16 000142          .EXIT          FMT      0350
    000142  104350
17
18 000144          QLTST:  .BLKW   7*7
19 000306          ARFAT:  .BLKW   20.
20 000356  063320  023364  022070  PTR:    .RAD50  /PR FILE12/
26 001364          114      117      117  LKFATL: .ASCIZ  /LOOKUP FATLFD/
    001367          113      125      120
    001372          040      106      101
    001375          111      114      105
    001400          104      000
27 001402          122      105      101  RDFATL: .ASCIZ  /READW FAILED/
    001405          104      127      040
    001410          106      101      111
    001413          114      105      104
    001416          000
28 001417          122      105      101  OK:    .ASCIZ  /READW OK/

```

FTEST RT-11 MACRO VM02-10 25-APR-75 10:37:51 PAGE 1+

```

    001422          104      127      040
    001425          117      113      000
29          .EVEN

```

FTEST RT-11 MACRO VM02-10 25-APR-75 10:37:51 PAGE 1+
SYMBOL TABLE

AREA	000306R	RUFF	000364R	LKFATL	001364R	OK	001417R	PC	=X000007
PTR	000356R	QLTST	000144R	RDFATL	001402R	R0	=X000000	R1	=X000001
R2	=X000002	R3	=X000003	P4	=X000004	R5	=X000005	SP	=X000006
ST	000000R	...V2	= 000001						
. ABS.	000000								
	001776		001						

ERRORS DETECTED: 0
 FREE CORE: 15895. WORDS
 ,LP:/N:ITM/L:MFB=FTEST

```

BLOCK NUMBER 0000
000/ 000000 000000 000000 000000 000000 000000 000000 000000 000000 *.....*
020/ 000000 000000 000000 000000 000000 000000 000000 000000 000000 *.....*
040/ 000000 000000 000000 000000 000000 001776 001776 000000 070524 *.....TQ*
060/ 000003 000000 000000 000000 000000 000000 000000 000000 000000 *.....*
100/ 000000 000000 000000 000000 000000 000000 000000 000000 000000 *.....*
120/ 000000 000000 000000 000000 000000 000000 000000 000000 000000 *.....*
140/ 000000 000000 000000 000000 000000 000000 000000 000000 000000 *.....*
160/ 000000 000000 000000 000000 000000 000000 000000 000000 000000 *.....*
200/ 000000 000000 000000 000000 000000 000000 000000 000000 000000 *.....*
220/ 000000 000000 000000 000000 000000 000000 000000 000000 000000 *.....*
240/ 000000 000000 000000 000000 000000 000000 000000 000000 000000 *.....*
260/ 000000 000000 000000 000000 000000 000000 000000 000000 000000 *.....*
300/ 000000 000000 000000 000000 000000 000000 000000 000000 000000 *.....*
320/ 000000 000000 000000 000000 000000 000000 000000 000000 000000 *.....*
340/ 000000 000000 000000 000000 000000 000000 000000 000000 000000 *.....*
360/ 000300 000000 000000 000000 000000 000000 000000 000000 000000 *.....*
400/ 000000 000000 000000 000000 000000 000000 000000 000000 000000 *.....*
420/ 000000 000000 000000 000000 000000 000000 000000 000000 000000 *.....*
440/ 000000 000000 000000 000000 000000 000000 000000 000000 000000 *.....*
460/ 000000 000000 000000 000000 000000 000000 000000 000000 000000 *.....*
500/ 000000 000000 000000 000000 000000 000000 000000 000000 000000 *.....*
520/ 000000 000000 000000 000000 000000 000000 000000 000000 000000 *.....*
540/ 000000 000000 000000 000000 000000 000000 000000 000000 000000 *.....*
560/ 000000 000000 000000 000000 000000 000000 000000 000000 000000 *.....*
600/ 000000 000000 000000 000000 000000 000000 000000 000000 000000 *.....*
620/ 000000 000000 000000 000000 000000 000000 000000 000000 000000 *.....*
640/ 000000 000000 000000 000000 000000 000000 000000 000000 000000 *.....*
660/ 000000 000000 000000 000000 000000 000000 000000 000000 000000 *.....*
700/ 000000 000000 000000 000000 000000 000000 000000 000000 000000 *.....*
720/ 000000 000000 000000 000000 000000 000000 000000 000000 000000 *.....*
740/ 000000 000000 000000 000000 000000 000000 000000 000000 000000 *.....*
760/ 000000 000000 000000 000000 000000 000000 000000 000000 000000 *.....*

```

In block 0, word 50 shows the highest, non-relocated, memory address in the user program. Word 52 shows the program size in bytes. Word 54 shows the size of the overlay region. The value is non-zero only for programs with overlays. Word 60 contains a 3, indicating that the relocation information begins at block 3 of the file.

```

BLOCK NUMBER 0001
000/ 112700 000007 012746 000144 104353 012700 000306 112740 *...F.N.K...P.*
020/ 000001 000001 105010 012760 000356 000002 005060 000004 *...P.N...Q...*
040/ 104375 103004 012700 001364 104351 104350 012700 000306 *J...T.I.H...F.*
060/ 112760 000010 000001 105010 012760 000000 000002 012740 *P...P...P.*
100/ 000364 000004 012760 000400 000006 012760 000000 000010 *T...P...P...*
120/ 104375 103004 012700 001402 104351 104350 012700 001417 *J...T.H...F.*
140/ 104351 104350 000000 000000 000000 000000 000000 000000 *I.H...P...P...*
160/ 000000 000000 000000 000000 000000 000000 000000 000000 *...P...P...P...*
200/ 000000 000000 000000 000000 000000 000000 000000 000000 *...P...P...P...*
220/ 000000 000000 000000 000000 000000 000000 000000 000000 *...P...P...P...*
240/ 000000 000000 000000 000000 000000 000000 000000 000000 *...P...P...P...*
260/ 000000 000000 000000 000000 000000 000000 000000 000000 *...P...P...P...*
300/ 000000 000000 000000 000000 000000 000000 000000 000000 *...P...P...P...*
320/ 000000 000000 000000 000000 000000 000000 000000 000000 *...P...P...P...*
340/ 000000 000000 000000 000000 000000 000000 000000 000000 *...P...P...P...*
360/ 003364 022070 000000 000000 000000 000000 000000 000000 *T&AS...P...P...*
400/ 000000 000000 000000 000000 000000 000000 000000 000000 *...P...P...P...*
420/ 000000 000000 000000 000000 000000 000000 000000 000000 *...P...P...P...*
440/ 000000 000000 000000 000000 000000 000000 000000 000000 *...P...P...P...*
460/ 000000 000000 000000 000000 000000 000000 000000 000000 *...P...P...P...*
500/ 000000 000000 000000 000000 000000 000000 000000 000000 *...P...P...P...*
520/ 000000 000000 000000 000000 000000 000000 000000 000000 *...P...P...P...*
540/ 000000 000000 000000 000000 000000 000000 000000 000000 *...P...P...P...*
560/ 000000 000000 000000 000000 000000 000000 000000 000000 *...P...P...P...*
600/ 000000 000000 000000 000000 000000 000000 000000 000000 *...P...P...P...*
620/ 000000 000000 000000 000000 000000 000000 000000 000000 *...P...P...P...*
640/ 000000 000000 000000 000000 000000 000000 000000 000000 *...P...P...P...*
660/ 000000 000000 000000 000000 000000 000000 000000 000000 *...P...P...P...*
700/ 000000 000000 000000 000000 000000 000000 000000 000000 *...P...P...P...*
720/ 000000 000000 000000 000000 000000 000000 000000 000000 *...P...P...P...*
740/ 000000 000000 000000 000000 000000 000000 000000 000000 *...P...P...P...*
760/ 000000 000000 000000 000000 000000 000000 000000 000000 *...P...P...P...

```

This block corresponds to locations 0-776 in the assembly listing.

```

BLOCK NUMBER 0002
000/ 000000 000000 000000 000000 000000 000000 000000 000000 *.....*
020/ 000000 000000 000000 000000 000000 000000 000000 000000 *.....*
040/ 000000 000000 000000 000000 000000 000000 000000 000000 *.....*
060/ 000000 000000 000000 000000 000000 000000 000000 000000 *.....*
100/ 000000 000000 000000 000000 000000 000000 000000 000000 *.....*
120/ 000000 000000 000000 000000 000000 000000 000000 000000 *.....*
140/ 000000 000000 000000 000000 000000 000000 000000 000000 *.....*
160/ 000000 000000 000000 000000 000000 000000 000000 000000 *.....*
200/ 000000 000000 000000 000000 000000 000000 000000 000000 *.....*
220/ 000000 000000 000000 000000 000000 000000 000000 000000 *.....*
240/ 000000 000000 000000 000000 000000 000000 000000 000000 *.....*
260/ 000000 000000 000000 000000 000000 000000 000000 000000 *.....*
300/ 000000 000000 000000 000000 000000 000000 000000 000000 *.....*
320/ 000000 000000 000000 000000 000000 000000 000000 000000 *.....*
340/ 000000 000000 000000 000000 000000 000000 000000 000000 *.....*
360/ 000000 000000 047514 045517 050125 043040 044501 042514 *.....LOOKUP FAIL*
400/ 000104 042522 042101 020127 040506 046111 042105 051000 *D.READW FAILD,R*
420/ 040505 053504 047440 000113 000000 000000 000000 000000 *EADW OK.....*
440/ 000000 000000 000000 000000 000000 000000 000000 000000 *.....*
460/ 000000 000000 000000 000000 000000 000000 000000 000000 *.....*
500/ 000000 000000 000000 000000 000000 000000 000000 000000 *.....*
520/ 000000 000000 000000 000000 000000 000000 000000 000000 *.....*
540/ 000000 000000 000000 000000 000000 000000 000000 000000 *.....*
560/ 000000 000000 000000 000000 000000 000000 000000 000000 *.....*
600/ 000000 000000 000000 000000 000000 000000 000000 000000 *.....*
620/ 000000 000000 000000 000000 000000 000000 000000 000000 *.....*
640/ 000000 000000 000000 000000 000000 000000 000000 000000 *.....*
660/ 000000 000000 000000 000000 000000 000000 000000 000000 *.....*
700/ 000000 000000 000000 000000 000000 000000 000000 000000 *.....*
720/ 000000 000000 000000 000000 000000 000000 000000 000000 *.....*
740/ 000000 000000 000000 000000 000000 000000 000000 000000 *.....*
760/ 000000 000000 000000 000000 000000 000000 041056 *.....B*

```

This block corresponds to locations 1000-1776 in the assembly listing.

```

BLOCK NUMBER 0003
000/ 000003 000006 000014 000023 000027 000040 000053 000057 *...+/*
020/ 177776 000000 000000 000000 000000 000000 000000 000000 *...
040/ 000000 000000 000000 000000 000000 000000 000000 000000 *...
060/ 000000 000000 000000 000000 000000 000000 000000 000000 *...
100/ 000000 000000 000000 000000 000000 000000 000000 000000 *...
120/ 000000 000000 000000 000000 000000 000000 000000 000000 *...
140/ 000000 000000 000000 000000 000000 000000 000000 000000 *...
160/ 000000 000000 000000 000000 000000 000000 000000 000000 *...
200/ 000000 000000 000000 000000 000000 000000 000000 000000 *...
220/ 000000 000000 000000 000000 000000 000000 000000 000000 *...
240/ 000000 000000 000000 000000 000000 000000 000000 000000 *...
260/ 000000 000000 000000 000000 000000 000000 000000 000000 *...
300/ 000000 000000 000000 000000 000000 000000 000000 000000 *...
320/ 000000 000000 000000 000000 000000 000000 000000 000000 *...
340/ 000000 000000 000000 000000 000000 000000 000000 000000 *...
360/ 000000 000000 000000 000000 000000 000000 000000 000000 *...
400/ 000000 000000 000000 000000 000000 000000 000000 000000 *...
420/ 000000 000000 000000 000000 000000 000000 000000 000000 *...
440/ 000000 000000 000000 000000 000000 000000 000000 000000 *...
460/ 000000 000000 000000 000000 000000 000000 000000 000000 *...
500/ 000000 000000 000000 000000 000000 000000 000000 000000 *...
520/ 000000 000000 000000 000000 000000 000000 000000 000000 *...
540/ 000000 000000 000000 000000 000000 000000 000000 000000 *...
560/ 000000 000000 000000 000000 000000 000000 000000 000000 *...
600/ 000000 000000 000000 000000 000000 000000 000000 000000 *...
620/ 000000 000000 000000 000000 000000 000000 000000 000000 *...
640/ 000000 000000 000000 000000 000000 000000 000000 000000 *...
660/ 000000 000000 000000 000000 000000 000000 000000 000000 *...
700/ 000000 000000 000000 000000 000000 000000 000000 000000 *...
720/ 000000 000000 000000 000000 000000 000000 000000 000000 *...
740/ 000000 000000 000000 000000 000000 000000 000000 000000 *...
760/ 000000 000000 000000 000000 000000 000000 000000 000000 *...

```

This block shows the root relocation information. The first word of block 3 is a 3; since this is positive, positive relocation is indicated. Locations 6, 14, 30, 46, 56, 100, 126, and 136 must all be positively relocated at FRUN time. (On examination of the assembly listing, those locations marked with a ' need to be relocated.) The 177776 terminates the list.

Had negative relocation been indicated at relative location 6, block 3 would have shown 100003, 6, 14, 23, 27, 40, 53, 57, 177776.

3.4.4.2 REL Files with Overlays - When overlays are included in a program, the file is similar to that of a non-overlaid program. However, the overlay segments must also be relocated. Since overlays are not permanently memory resident but are read in from the file as needed, they require an additional operation. Each overlay segment is relocated (by FRUN) and then rewritten into the file. Then, when the overlay is called in, it will be properly relocated. This process takes

place each time an overlaid file is run with FRUN. The relocation information for overlay files contains both the list of addresses to be modified and the original contents of each location. This allows the file to be FRUN after the first usage.

NOTE

.ASECTs are illegal above 1000₈ and restricted in an overlaid foreground job. Refer to Chapter 6 of the RT-11 System Reference Manual.

A REL file with overlays appears as shown in Figure 3-14:

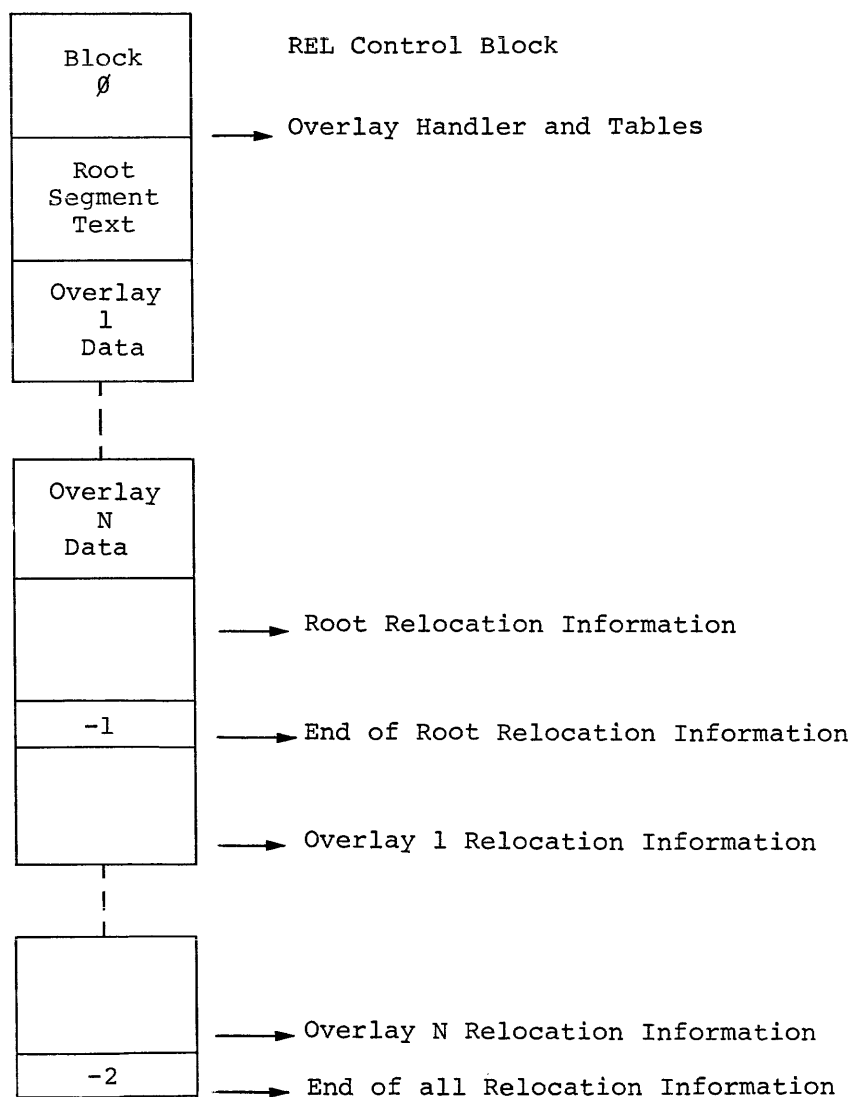


Figure 3-14
REL File With Overlays

In this case, location 54 of block 0 of the REL file contains the size of the overlay region, in words. This is used to allocate space for the job when added to the size of the program base segment in location 52.

After the program base (root) code has been relocated, each existing overlay is read into the program overlay region in memory, relocated via the overlay relocation information, and then written back into the file.

The root relocation information section is terminated with a -1. This -1 is also an indication that an overlay segment relocation block follows. The overlay segment relocation block is shown in Figure 3-15:

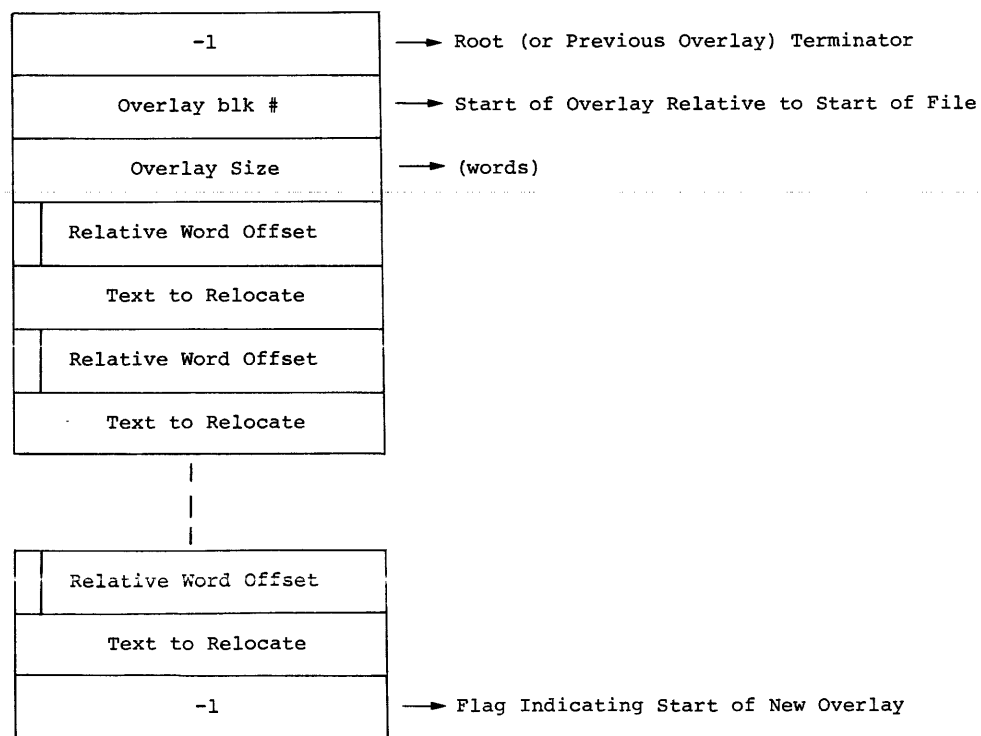


Figure 3-15
Overlay Segment Relocation Block

The displacement is relative to the start of the program and is interpreted as in the nonoverlaid file (i.e., bit 15 indicates the type of relocation, and the displacement is the true displacement divided by two). Encountering -1 indicates that a new overlay region begins here. A -2 indicates the termination of all relocation information. .

CHAPTER 4

SYSTEM DEVICE

4.1 DETAILED STRUCTURE OF THE SYSTEM DEVICE

The RT-11 system device holds all the components of the system and is used by RT-11 to store device handlers and the monitor file. The layout of the system device is:

<u>Block #</u>	<u>Contents</u>
0	Bootstrap
1	Reserved for volume identification information
2	Bootstrap
3 to 5	Reserved for monitor or bootstrap expansion
6 to $(N*2)+5$	Directory segments; N is the number of directory segments
$(N*2)+6$ to end	File storage

All other system components, i.e., the monitor and device handlers, are files on the system device:

<u>File</u>	<u>Contains</u>
MONITR.SYS	The current RT-11 monitor; contains bootstrap, KMON, USR/CSI, RMON, KMON overlays, scratch blocks
SYSMAC.SML	System Macro Library
SYSMAC.S8K	8K System Macro Library
LP.SYS	Line printer handler

<u>File</u>	<u>Contains</u>
DT.SYS	DEctape handler
TT.SYS	Console handler (S/J only)
RK.SYS	RK disk handler
DS.SYS	RJS03/4 fixed-head disk handler
DX.SYS	RX01 flexible disk handler
DP.SYS	RP disk handler
PR.SYS	High-speed reader handler
PP.SYS	High-speed punch handler
CR.SYS	Card reader handler
RF.SYS	RF disk handler
CT.SYS	Cassette handler
MT.SYS	TM11 magtape handler
MM.SYS	TJU16 magtape handler
BA.SYS	BATCH run-time handler

In general, files with the .SYS extension are parts of the monitor system. The bootstrap records the block numbers of the relevant areas in the monitor tables at bootstrap time. Thus, RT-11 is extremely flexible with respect to the interchange and construction of systems.

4.2 CONTENTS OF MONITR.SYS

Following is the block layout of the RT-11 monitor file, MONITR.SYS. Block numbers are relative to the start of the file.

<u>F/B Monitor</u>	<u>Block # (decimal)</u>	<u>Contents</u>
	0-1	Copy of system bootstrap (blocks 0 and 2 of the system device)
	2-17	Swap blocks
	18-24	KMON (includes 2-block KMON overlay area)
	25-32	USR/CSI
	33-47	RMON
	48-57	KMON overlays

<u>S/J Monitor</u>	<u>Block # (decimal)</u>	<u>Contents</u>
	0-1	Copy of system bootstrap
	2-16	Swap blocks
	17-22	KMON (includes 1-block KMON overlay area)
	23-30	USR/CSI
	31-37	RMON
	38-44	KMON overlays

4.3 KMON OVERLAYS

The KMON overlays are one block in size in the S/J Monitor and two blocks in size in the F/B Monitor. The contents of each overlay are described in this list:

<u>Overlay #</u>	<u>S/J</u>	<u>F/B</u>
0	DATE, TIME	DATE, TIME, SAVE, ASSIGN
1	SAVE, ASSIGN	LOAD, UNLOAD, SUSPEND, RESUME, CLOSE, FRUN (Part 1)
2	LOAD, UNLOAD, CLOSE	FRUN (Part 2)
3	GT ON/OFF	GT ON/OFF, SET
4	SET	

4.4 DETAILED OPERATION OF THE BOOTSTRAP

Bootstrapping a system causes a fresh copy of that system to be installed in memory. In the RT-11 boot, certain system device resident tables are also updated. Following is a detailed description of the bootstrap.

<u>Action</u>	<u>Explanation</u>
1. User executes hardware bootstrap	On all system devices except diskette, this causes block 0 of the system device to be read into 0-777. Control then passes to location 0. On diskette, causes logical block 0 to be read into 0-777. Hardware bootstrap reads 64 words from track 1, sector 1. Control passes to location 0, where 64 words from each of sectors 3, 5, and 7 (track 1) are read.
2. Second part of bootstrap is read	The first part of the boot reads the second half into 1000-1777. On diskette, the first part of the boot reads logical block 2 (sectors 9, 11, 13, 15) into 1000-1777.
3. Determine how much memory is available	Boot sets a trap at location 4 and then starts addressing memory. When the trap is taken, illegal memory has been addressed.
4. Look for special devices	Boot sets a trap at location 10 and then tries to address the clock, FPU, and VT11 display processor. Their presence or absence is indicated in the CONFIG word in RMON. (If a PDP-11/03 processor is present, the bootstrap assumes that a clock is present.)
5. Check memory size	If memory is too small to read in the monitor, a message is printed and the boot halts.
6. Read in directory and find MONITR.SYS	The entire directory is searched. If MONITR.SYS is not found, a HALT occurs after the boot prints an error message.
7. Read the monitor into memory	The monitor file, MONITR.SYS, is read into the highest bank of memory.
8. Put pointers to monitor file blocks into RMON	RMON references the monitor swap blocks directly. Thus, the position of the swap blocks varies as the placement of MONITR.SYS varies. The real position of the blocks is updated for each boot operation.
9. Update position-dependent areas in RMON.	MONITR.SYS is initially linked at 8K. However, if more than 8K is available, RT-11 uses it. To do that, certain words must be updated to point to the actual areas of high memory where they will be. Boot contains a list of all words to be updated, located at RELLST in BSTRAP.MAC.

<u>Action</u>	<u>Explanation</u>
10. Update processor-dependent area in RMON	If processor is a PDP-11/03, any PS references in the monitor are changed to use the MFPS and MTPS instructions.
11. LOOKUP the device handlers in system and store their record numbers in \$DVREC	Boot looks at \$PNAME table to find the names of the devices in the system. The extension .SYS is appended. Thus, the PR handler is a file called PR.SYS. The location of the handler is then placed in \$DVREC. If the LOOKUP fails, the device gets a 0 in its \$DVREC entry. That implies that the device handler does not exist.
12. Print bootstrap header	Boot prints monitor identification message "RT-11" followed by monitor type ("FB" or "SJ") followed by version number.
13. Set up locations 0 and 2	Boot puts a "BIC R0,R0" in location zero and an .EXIT EMT in location 2.
14. Turn on KW11-L Clock	The bootstrap turns on the clock, if present in the configuration and processor is not a PDP-11/03.
15. Exit to Keyboard Monitor	

4.5 FIXING THE SIZE OF A SYSTEM

RT-11 is designed to automatically operate from the top of the highest available 4K memory bank. However, it is possible to force the system to operate from a specified area that is not necessarily the highest. For instance, the following series of commands causes RT-11 to run in a 16K environment, even though the configuration actually has 28K of memory:

<u>.R PATCH<CR></u>	[Run RT-11 PATCH program.]
<u>PATCH Version number</u>	00007 =
<u>FILE NAME--</u>	[Specifying MONITR.SYS/M indicates it is a monitor file.
<u>*MONITR.SYS/M<CR></u>	
<u>*BHALT/ 407 0<CR></u>	Change location "BHALT" from a 407 to a 0 (HALT). The correct address of BHALT can be found in Table 2 of RT-11 System Release Notes (V02C).
<u>*E</u>	E causes an exit to the monitor.
<u>.R PIP</u>	
<u>*A=MONITR.SYS/U<CR></u>	
<u>*SY:/O</u>	Now run PIP to update the bootstrap and reboot the system.]

When the bootstrap is performed, the computer halts. The halt allows the user to enter the desired size in the switch register. With this patch installed, the V2 bootstrap uses the top five bits (bits 11-15) of the switch register to determine memory size. If the switch register contains the number 160000 or greater (e.g., if the register is unchanged after booting the system), a normal memory determination is performed. Otherwise, the top five bits are taken to be a number representing the number of 1K word blocks of memory. Each bit has the following value:

<u>Switch Register</u>	<u>Memory Size</u>
4000	1K
10000	2K
20000	4K
40000	8K
100000	16K

A combination of the bits will produce the range of system sizes from 8K through 28K, in 1K increments.

Examples:

1. To boot a system into 24K on a 28K configuration, use the combination:

$$140000 = 100000 (16K) + 40000 (8K)$$

2. To boot the S/J Monitor into 11K, use the combination:

$$54000 = 40000 (8K) + 10000 (2K) + 4000 (1K)$$

When the switch register is set properly, press the CONTINUE switch and the bootstrap will be executed.

If the CONTINUE switch is pressed immediately following the halt without changing the switch settings, a normal memory determination is done. To change the bootstrap back to its original (non-halting) form, execute the same commands as above, but change the 0 at BHALT back to a 407.

This procedure allows the user to 'protect' memory areas, since RT-11 never accesses memory outside the bounds within which it runs.

Another useful procedure, when desiring to always boot a system into a specific memory size or when the console switch register is not available, is to determine the bit combination corresponding to the choice of memory size, as explained above. Then enter the following commands, where xxxxx represents the bit pattern just determined:

```
.R PATCH<CR>
```

[Run RT-11 PATCH program.]

PATCH Version number

FILE NAME--

*MONITR.SYS/M<CR>

*BHALT/ 407 240<LF>

*BHALT+2/ 13702 12702<LF>

*BHALT+4/ 177570 xxxxxx<CR>

*E

.

.

[NOP the branch at BHALT
Change MOV @#SR,R2 to
MOV #VAL,R2. Address of
switch register is replaced
with one of the bit combina-
tions described previously.]

For the patch addresses for other system devices, and for the address of BHALT, consult Table 2 of RT-11 System Release Notes (V02C).

CHAPTER 5

I/O SYSTEM, QUEUES, AND HANDLERS

I/O transfers in RT-11 are handled by the monitor through routines known as device handlers. Device handlers are resident on the system mass storage device and can be called into memory at a location specified by the user (via a .FETCH handler request or KMON LOAD command). Only the device handlers distributed with the system in use (V2 or V02B) may be used; the system will malfunction otherwise.

This chapter describes how to write a new device handler and add it to the system. A summary of differences between Version 1 and Version 2 Device Handler requirements is included for the user who wishes to update old device handlers. Instructions and examples for making a device the system device and for writing a new bootstrap for the device are also included.

5.1 QUEUED I/O IN RT-11

Once a device handler is in memory, any .READ/.WRITE requests for the corresponding devices are interpreted by the monitor and translated into a call to the I/O device handler. To facilitate overlapped I/O and computation, all I/O requests to RT-11 are done through an I/O queue. This section details the structure of the I/O queueing system.

5.1.1 I/O Queue Elements

The RT-11 I/O queue is made up of a linked list of queue elements. A single element has the structure shown in Figure 5-1:

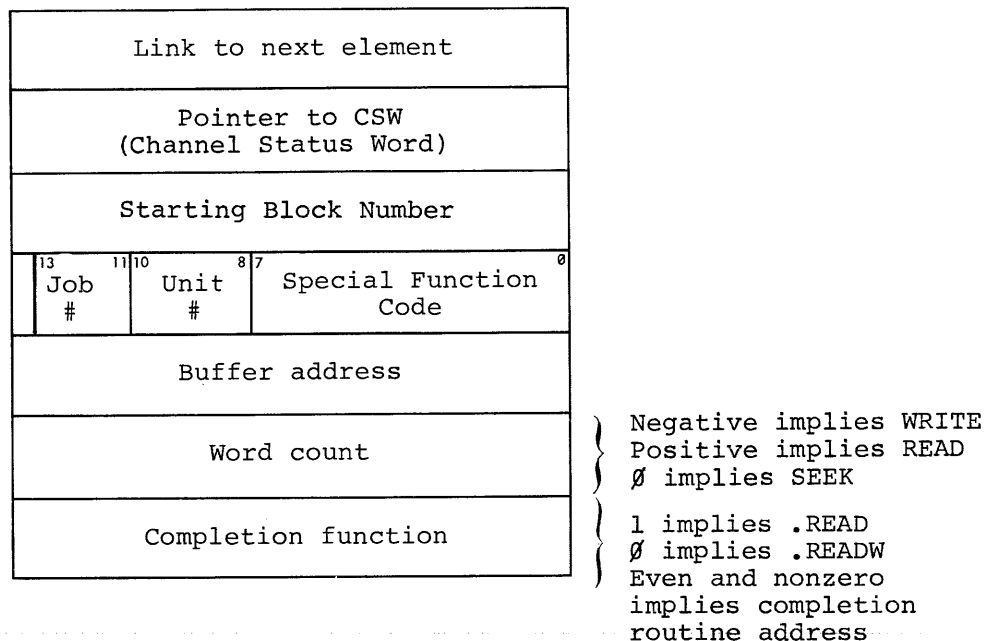
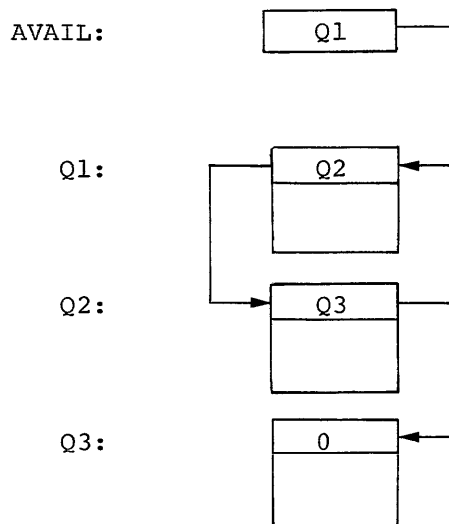


Figure 5-1
I/O Queue Element

RT-11 maintains one queue element in the Resident Monitor. (In F/B, one element per job is maintained in the job's impure area.) This is sufficient for any program that uses wait-mode I/O (.READW/.WRITW). However, for maximum throughput, the .QSET programmed request should be used to create additional queue elements.

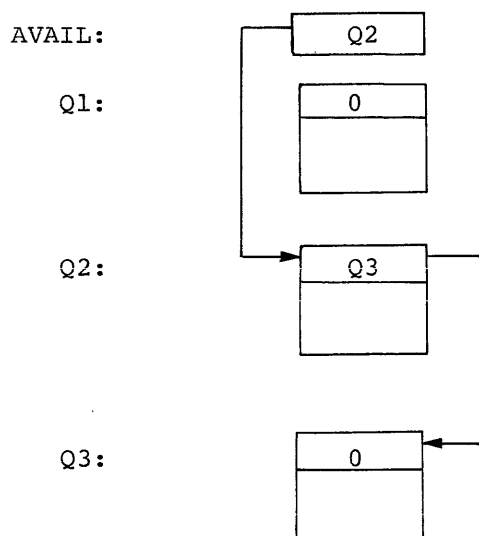
If an I/O operation is requested and a queue element is not available, RT-11 must wait until an element is free to queue the request. This obviously slows up program execution. If asynchronous I/O is desired, extra queue elements should be allocated. It is always sufficient to allocate N new queue elements, where N is the total number of pending requests that can be outstanding at one time in a particular program. This produces a total of N+1 available elements, since the Resident Monitor element is added to the list of available elements.

Diagrammatically, the I/O queue appears as follows:



AVAIL is the list header. It always contains a pointer to an available element. If AVAIL is 0, no elements are currently available.

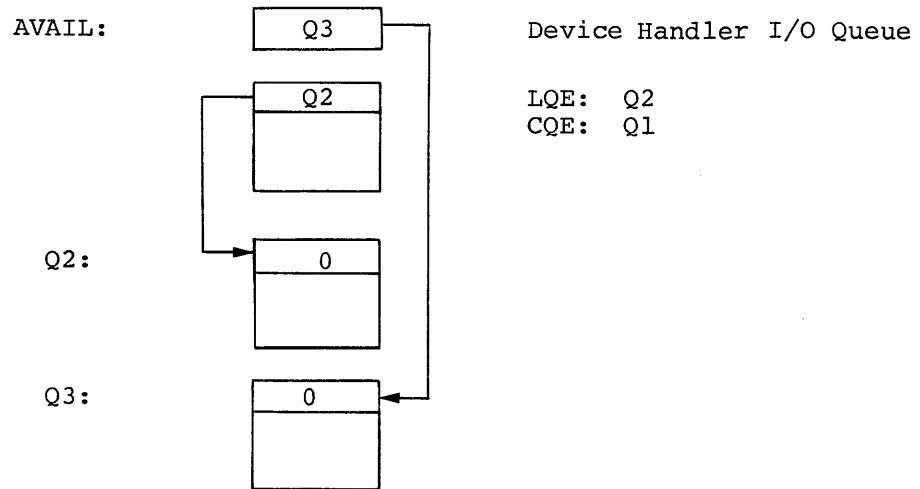
When an I/O request is initiated, an element is allocated (removed from the list of available elements) and is linked into the appropriate device handler's I/O queue. The handler's queue header consists of two pointers: the current queue element (CQE) pointer, pointing to the element at the top of the list, and the last queue element (LQE) pointer, pointing to the last element entered in the queue. The LQE pointer is used by the S/J monitor for fast insertion of new elements into the queue.



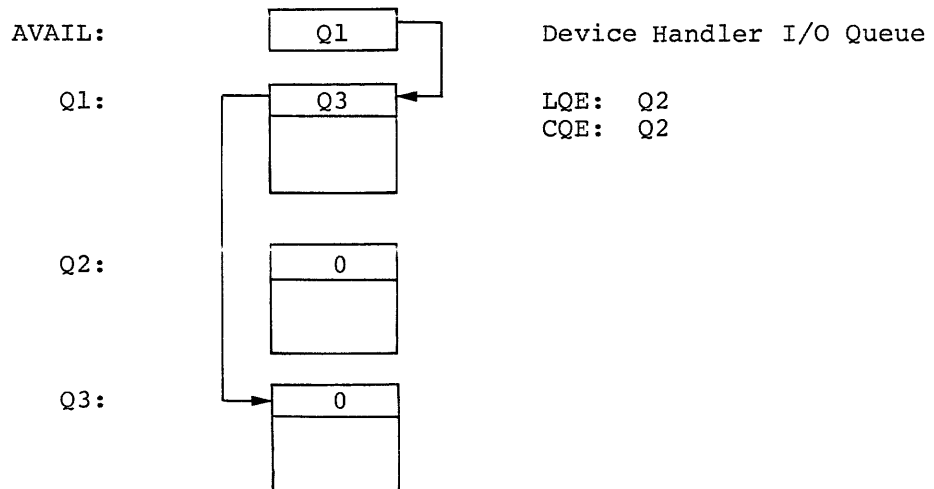
Device Handler I/O Queue

LQE: Q1 (Pointer to last queue element)
CQE: Q1 (Pointer to current queue element)

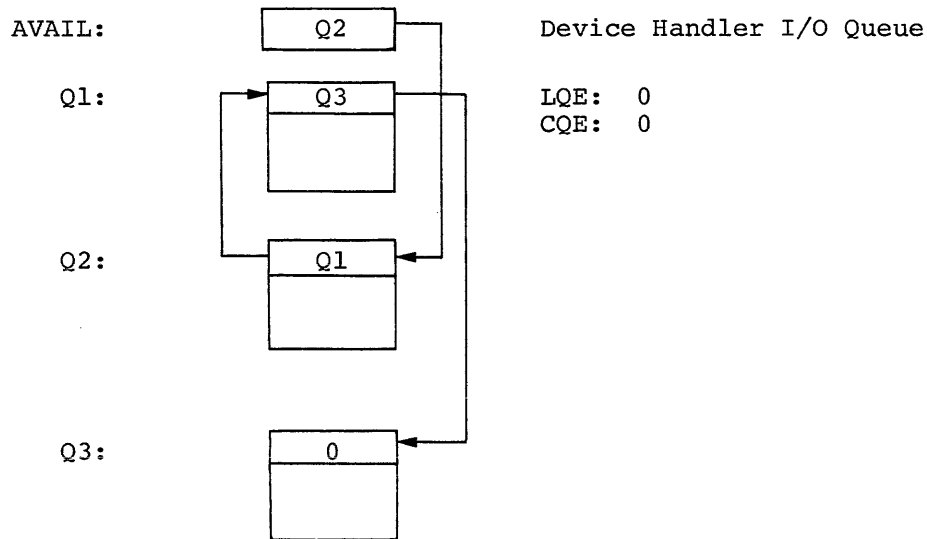
In this case, the device is associated with element Q1. If another request comes in for that same device before the first completes, a waiting queue is built up for that device.



When the I/O transfer in progress completes, Q1 is returned to the list of available elements, and the transfer indicated by Q2 will be initiated:



When Q2 is completed, it too is returned to the list of available elements.



Note that the order of the queue element linkages may be altered.

A distinction between S/J and F/B operation is that F/B maintains two separate queue structures, one for each active job. The queue headers (AVAIL) are words in the user's impure area. The centralized queue manager dispatches transfers in accordance with job priority. Thus, if two requests are queued waiting for a particular device, the foreground request is honored first. At no time, however, will an I/O request already in progress be aborted in favor of a higher priority request; the operation in progress will complete before the next transfer is initiated.

Another difference between S/J and F/B operation is that the F/B scheduler will suspend a job pending the availability of a free queue element and will try to run another job.

5.1.2 Completion Queue Elements

The F/B Monitor maintains, in addition to the queue of I/O transfer requests, a queue of I/O completion requests. When an I/O transfer completes and a completion routine has been specified in the request (i.e., the seventh word of the I/O queue element is even and non-zero), the queue completion logic in the F/B Monitor transfers the request node (element) to the completion queue, placing the channel

status word and channel offset in the node. This has the effect of serializing completion routines, rather than nesting them. Completion routines are called by the completion queue manager on a *first-in/first-out* basis, and the completion routines are entered at priority level 0 rather than at interrupt level.

The .SYNCH request also makes use of the completion queue. When the .SYNCH request is entered, the seven-word area supplied with the request is linked into the head of the completion queue, where it appears to be a request for a completion routine. The .SYNCH request then does an interrupt exit. The code following the .SYNCH request is next called at priority level 0 by the completion queue manager. To prevent the .SYNCH block from being linked into AVAIL (the queue of available elements), the word count is set to -1. The completion queue manager checks the word count before linking a queue element back into the list of available elements, and skips elements with the -1 word count.

Figures 5-2 and 5-3 show the format of the completion queue and .SYNCH elements.

OFFSET	
0	QUEUE LINK
2	
4	
6	
10	CHANNEL STATUS WORD
12	CHANNEL OFFSET
14	COMPLETION ROUTINE ADDRESS

Figure 5-2
Completion Queue Element

OFFSET	
0	QUEUE LINK
2	JOB NUMBER
4	
6	
10	SYNCH I.D.
12	-1
14	SYNCH RETURN ADDRESS

Figure 5-3
.SYNCH Element

5.1.3 Timer Queue Elements

Another queue maintained by the F/B Monitor is the timer queue. This queue is used to implement the .MRKT request, which schedules a completion routine to be entered after a specified period of time. The first two words of the element are the high- and low-order time and the seventh word is the completion routine address. An optional sequence number can be added to the request to distinguish this timer request from others issued by the same job.

The F/B Monitor uses the timer queue internally to implement the .TWAIT request. The .TWAIT request causes the issuing job to be suspended and a timer request is placed in the queue with the .RSUM logic as the completion routine. Refer to Figure 5-4 for the format of the timer queue element.

OFFSET		
0	HIGH-ORDER TIME	C.HOT
2	LOW-ORDER TIME	C.LOT
4	LINK TO NEXT ELEMENT	C.LINK
6	JOB # OF OWNER	C.JNUM
10	OWNER'S SEQUENCE #	C.SEQ
12	0	
14	COMPLETION ADDRESS	C.COMP

Figure 5-4
Timer Queue Element

5.2 DEVICE HANDLERS

This section contains the information necessary to write an RT-11 device handler. It is illustrated with an example, a driver for the RS64 fixed-head disk (with RC11 controller). A source listing is included in Appendix A, Section A.1; portions of this listing are referenced throughout the remainder of this section and in future sections.

The user should refer to the PDP-11 Peripherals Handbook for details regarding the operation of any particular peripheral.

NOTE

All RT-11 handlers must be written in position independent code (PIC). Consult the PDP-11 Processor Handbook for information on writing PIC.

5.2.1 Device Handler Format

The first five words of any device handler are header words. The format is:

<u>Word #</u>	<u>Contents</u>
1	Address of first word of device's interrupt vector.
2	Offset from current PC to interrupt handler.
3	Processor status word to be used when interrupt occurs. Must be 340 (priority 7).
4,5	Zero. These are the queue pointers.

See area C in the example handler (Section A.1).

A word must be provided at the end of the handler. When the handler is .FETCHed, the monitor places a pointer to the monitor common interrupt entry code in the last word of the handler. This requires that the handler size in the monitor's \$HSIZE table be exact or the handler will malfunction. See area M in the example in Section A.1.

The word preceding the interrupt handler entry point must be an unconditional branch to the handler's abort code. The abort code is used by the F/B Monitor to stop I/O for the device. The abort entry point is shown at area G in the example and the abort code is at area K. (See the RT-11 System Reference Manual, Section H.2, for further information.)

5.2.2 Entry Conditions

The device handler is entered directly from the monitor I/O queue manager, at which time it initiates the data transfer. The fifth word of the header contains a pointer into the queue element to be processed. This word (called CQE, for Current Queue Element) points to the third word of the queue element, which is the block number to be read or written. Referring to the example, location RCCQE contains the address of the third word of the queue element to be processed. It is generally advisable to put the pointer into a register, as that greatly facilitates picking up arguments to initiate the transfer. In the example, the entry point is at the location marked by E. Notice that registers need not be saved.

5.2.3 Data Transfer

Most handlers use the interrupt mechanism when transferring data. The handler initiates the transfer and then returns immediately to the monitor with an RTS PC, shown at area F. When the transfer is completed, the device interrupts. When the interrupt routine determines that I/O is complete or that an error has occurred, it jumps to the monitor completion routine in the manner shown at area J in the listing.

If the interrupt mechanism is not used, the data transfer must be completed before returning to the monitor. The handler must loop on a device flag with the interrupt disabled. When I/O is complete, the driver returns to the monitor with a jump to the monitor completion code, similar to that shown at area J in the example.

5.2.4 Interrupt Handler

Once the transfer has been initiated and control has passed back to the monitor, data interrupts will occur.

Information in the header of the handler causes the interrupt to be vectored to the interrupt handling code within the handler. The code at the interrupt location should keep the transfer going, determine when the transfer is complete, and detect errors.

When the transfer is done, control must be passed to the monitor's I/O queue manager, which performs a cleanup operation on the I/O queue.

Restrictions that apply to the interrupt code are:

1. The common interrupt entry into the monitor must be taken. Interrupt routines linked into a program use the .INTEN request described in Chapter 9 of the RT-11 System Reference Manual. Handlers made part of the system have a more efficient method of entry. The last word of the handler is set to point to the monitor common interrupt entry code when the handler is fetched. Upon reception of an interrupt, the handler must execute this code by performing a JSR R5, @\$INPTR, where \$INPTR is the tag commonly used by RT-11 handlers for the pointer word. See areas I and N in the example. The JSR instruction must be followed by the complement of the priority at which the handler will operate. See area I for an easy method to make the assembler compute the complement. On return from the monitor's interrupt entry code, R4 and R5 have been saved and may be used by the handler. Other registers must be saved and restored if they are to be used.
2. A check must be made to determine if the transfer is complete. However, with nonfile-structured devices, such as paper tape, line printer, etc., an interrupt occurs whenever a character has been processed. For these devices, the byte count, which is in the queue element, is used as a character count.

Nonfile-structured input devices should be able to detect an end of file condition, and pass that on to the monitor.

NOTE

The queue element contains a word count, not a byte count. The initial entry to the handler should change the word count to a byte count if the device interrupts at each character. The transfer is complete when the byte count decrements to 0.

Before the conversion to bytes is made, the sign of the word count must be determined since it specifies whether this transfer is a Read or Write. A negative word count implies a Write and should be complemented before converting to bytes.

3. Check for occurrence of an error. If a hardware error occurred, the hard error bit in the channel status word (CSW) should be set, the transfer should be aborted, and the monitor completion code executed. The address of the channel status word is in word 2 of the queue element. The error bit is bit 0 of the CSW. Generally, it is advisable to retry a certain number of times if an error occurs. RT-11 currently retries up to eight times before deciding an error has occurred. (Note that this is true for file-structured devices only.) It is

desirable, in case an error occurs, to do a drive or control reset, where appropriate, to clear the error condition before a retry is initiated. See the area between I and H in the example.

4. If the transfer is not complete and no error has occurred, registers used should be restored, and an RTS PC executed.

To pass an EOF (End of File) to the monitor, the 2000 bit in the CSW should be set. Refer to the sample handler in Appendix A for an example of setting the EOF bit. When EOF is detected on non-file structured devices, the remainder of the input buffer must be zeroed.

5. When the transfer is complete, whether an error occurred or not, the monitor I/O completion code must be entered to terminate activity and/or enter a completion routine. When return is made to the monitor, R4 must point to the fifth word of the handler (RCCQE in the example). See area J in the example for the method of returning to the monitor completion routine.

Handlers should check for special error conditions that can be detected on the initial entry to the handler. For example, trying to write on a read-only device should produce a hard error. It must be emphasized that the user handlers should interface to the system in substantially the same way as the handler in Section A.1. This handler is included as a guide and an example.

5.3 ADDING A HANDLER TO THE SYSTEM

When the handler has been written and debugged, it may be installed in the system by following the procedures in this section. The process consists of inserting information about the handler into the monitor tables listed below.

<u>Table to be Changed</u>	<u>Contents</u>
\$HSIZE	Size of handler (in bytes).
\$DVSIZ	Size of device in 256-word blocks. If nonfile device, entry = 0.
\$PNAME	Permanent name of the device (should be two alphanumeric characters entered in .RAD50 notation, left-justified).
\$STAT	Device status table. Refer to Section 2.5.2.2 for the format of \$STAT table.
LOWMAP	Low memory protection map; refer to Section 2.5.4.

There is no restriction on handler names; any 2-letter combination not currently in use may be chosen for the new handler and the name may be inserted in any unused slot in the \$PNAME table, or in a slot occupied by a nonexistent device (i.e., a device not installed on the user's system). Note that the name must be entered in .RAD50. Since PATCH does not have a .RAD50 interpretation switch, the name must be entered to PATCH in its numerical form. Appendix C of the RT-11 System Reference Manual contains a .RAD50 conversion table; ODT can also be used to perform .RAD50 conversions.

As an example, assume again the handler for the RC11/RS64 disk (the sample handler in Section A.1) is to be inserted in the system. First, the values of the table entries for this device are determined (the addresses used in the example are for illustrative purposes only; consult Table 2 of RT-11 System Release Notes (V02C) for the correct table addresses for the version in use):

\$HSIZE:	316	After assembly, the handler was found to take up 316 bytes. See area 0 in the example listing.
\$DVSIZ:	2000	The disk has 1024 (decimal) 256-word blocks for storage.
\$PNAME:	.RAD50 /RC/ or 70370	The name assigned is RC. The .RAD50 value of RC is 70370.
\$STAT:	100023	The device is file-structured, is a read/write device, and uses the standard RT-11 file structure. The identifier (selected by the user) is 23. Refer to Section 2.5.2.2 for the format of the \$STAT table.
LOWMAP:	14	Protect RC vector 210,212 at byte 336 of LOWMAP (refer to Section 2.5.4.).

Once these values have been decided, the steps for inserting the device handler are:

1. Assemble the handler, using either MACRO or ASEMBL.
2. Link the handler at 1000. The name of the handler should be whatever the \$PNAME entry is, with the .SYS extension appended:

```
.R LINK
*RC.SYS=RC      where RC.OBJ is the handler object
UNDEF GLBLS     module. The default link address is
                  1000.
```

NOTE

If the handler being linked is one that could also be a system device handler, the user can expect one undefined global, \$INTEN.

3. Run PATCH to modify the tables and protect the interrupt vectors.

For this example, assume that the table addresses are found to be:

<u>Table</u>	<u>S/J Address</u>	<u>F/B Address</u>
\$HSIZE	13624	14556
\$DVSIZ	13660	14612
\$PNAME	16470	17630
\$STAT	16524	17664

NOTE

The addresses above are for illustration only. Consult Table 2 of RT-11 System Release Notes (V02C) for current table addresses and for the address of the monitor base location, BASE.

The tables have room for fourteen (decimal) device entries; all are already assigned by the monitor. Assuming that a given configuration never has all supported devices, however, at least one slot should be available to be overlaid. For example, assume the twelfth slot is occupied by a device not installed on the system, and therefore available for change. The octal offset is 26, which, added to the table addresses above, gives the address of the empty slot:

S/J Monitor:

.R PATCH<CR>

PATCH Version number

FILE NAME--

*MONITR.SYS/M<CR>

*BASE;ØR<CR>

*Ø,13652/ 4ØØØ 316<CR>

*Ø,137Ø6/ Ø 2ØØØ<CR>

*Ø,16516/ 625Ø 7037Ø<CR>

*Ø,16552/ 4 1ØØØ23<CR>

*Ø,16336\ 77 <CR>

*E

⋮

[/M is necessary;

Monitor base;

\$HSIZE table;

\$DVSIZ table;

\$PNAME table;

\$STAT table;

Check that vectors in

permanent map are protected;

Exit to monitor]

F/B Monitor

_R PATCH

PATCH Version number

FILE NAME--

*MONITR.SYS/M<CR>

*BASE;ØR<CR>

*Ø,14556/ 4ØØØ 316<CR>

*Ø,14612/ Ø 2ØØØ<CR>

*Ø,17630/ 625Ø 7Ø37Ø<CR>

*Ø,17664/ 4 1ØØØ23<CR>

*Ø,17336 77 <CR>

*E

.

[/M is necessary;

Monitor base;

\$HSIZE table;

\$DVSIZ table;

\$PNAME table;

\$STAT table;

Check that vectors in
permanent map are protected;

Exit to monitor]

At this point, the system should be re-bootstrapped to make the modified monitor resident. The device RC will then be available for use.

5.4 WRITING A SYSTEM DEVICE HANDLER

This section describes the procedures for writing a new system device handler. A system device is the device on which the monitor and handlers are resident. RT-11 currently supports the RK, RF, DP, DS, and DX disks, and DECtape as system devices. The procedures for writing the handler and creating a new monitor are explained, illustrated by the example in Section A.1, the RC11/RS64 handler.

The basic requirements for a system device are random access and read/write capability. These requirements are met by the RC11 disk, which is a multiple platter, fixed-head disk. When writing the driver, the procedures in Section 5.2 should be followed. Because the system handler is linked with the monitor, the additional tagging and global conventions described here must also be followed.

5.4.1 The Device Handler

The following conditions must be observed when writing a system handler. Refer to the example listing in Section A.1.

1. The handler entry point must be tagged xxSYS, where xx is the 2-letter device name. For the RC disk, this is RCSYS. See area D in the listing.
Important: Note that the tag is placed after the third word of the header block.
2. The entry points of all current system devices must be referenced in a global statement. These

currently include RKSYS, RFSYS, DSSYS, DXSYS, DPSYS, DTSYS and RCSYS. See Area A.

3. The entry point tags of all other system devices must be equated to zero. See area B in the listing.
4. A .CSECT SYSHND must be included at the top of the handler code. It is located above area C in the example.
5. The last word of the handler is used for the common interrupt entry address. This should have the tag \$INPTR and should be set to the value \$INTEN. See areas M and N in the example listing. These tags should be global. See area A.
6. The interrupt entry point should have the tag xxINT, or RCINT for this example, and this must be a global. See areas A and H.
7. The handler size must be global, with the symbolic name xxSIZE, or RCSIZE. See area A. This step is not necessary if the monitor sources are available and are being reassembled, since the global will be generated by the HSIZE macro. See Step 3 in Section 5.4.3.

5.4.2 The Bootstrap

This section describes the procedure for modifying the system bootstrap to operate with a new system device. Either the bootstrap source must be acquired, or the listing in Section A.2 may be used. Again, the RC11/RS64 disk is used for an example. The references in this section, however, are to the bootstrap listing found in Section A.2 of Appendix A.

The following changes must be made to the bootstrap to support a new system device:

1. Add a new conditional, \$xxSYS, to the list at point AA. Here xx is the 2-letter device name, and in this case the conditional is \$RCSYS.
2. Add a simple device driver for the device inside a \$xxSYS conditional. This is shown at area CC. Because the RC11 is similar to the other disks, it is possible to share code with the other device drivers, reducing the implementation effort. To do this, the \$RCSYS conditional is added at area BB and the device specific code is at area FF. This code merges with the common code at area GG.

3. The device driver has these characteristics:

- a. The SYSDEV macro must be invoked for the device. The macro arguments are the 2-letter device name and the interrupt vector address. For this example, the arguments are "RC" and "210", shown at area DD on the listing.
- b. The device driver entry point must have the tag READ. See area EE.
- c. When the driver is entered:

RØ = Physical Block Number
R1 = Word Count
R2 = Buffer Address
R3,R4,R5 = are available for use
 by the driver routine

- d. The driver must branch to BIOERR if a fatal I/O error occurs.

5.4.3 Building the New System

This section describes the procedure for building a new monitor using the system device handler and bootstrap just developed. Again, the example used is the RC11/RS64 disk, and the appropriate listings are those in Sections A.1 and A.2.

The procedure is:

1. Assemble the handler, producing an object module with the name xx.OBJ, where xx is the 2-letter device name. In this example, the name is RC.

```
.R MACRO
*RC.OBJ=RC.MAC
```

2. Assemble the bootstrap, defining the conditional \$xxSYS (where xx is again the device name; e.g., \$RCSYS). Define the conditional BF if an F/B bootstrap is desired. Let BF be undefined for an S/J bootstrap. For the S/J bootstrap:

```
.R MACRO<CR>
*RCBTSJ=TT:,DK:BSTRAP<CR>
^$RCSYS=1<CR>
^Z^$RCSYS=1<CR>
^ZERRORS DETECTED: Ø
FREE CORE: 156Ø8. WORDS
```

For the F/B bootstrap:

```
.R MACRO<CR>
*RCBTFB=TT:,DK:BSTRAP<CR>
^$RCSYS=1<CR>
BF=1<CR>
^Z^$RCSYS=1<CR>
BF=1<CR>
^ZERRORS DETECTED: Ø
FREE CORE: 1558Ø. WORDS
```

3. If the monitor sources are available, the DEVICE macro described in Section 2.5.2.9 can be invoked for the new device by editing the macro call into RMONFB.MAC and RMONSJ.MAC and reassembling the monitor. For the RC device, the macro would be:

```
DEVICE RC 2000 100020 RCSYS
```

The HSIZE macro, described in the same section, must also be invoked. For the RC device, the macro would be:

```
HSIZE RC,316,SYS
```

Monitor assembly instructions are in Chapter 5 of the RT-11 System Generation Manual. If this approach is used, the table patching procedure in step 5 is not necessary.

4. Link the monitor with the new bootstrap and device handler.

For S/J:

```
.R LINK
*RCMNSJ.SYS,MAP=RCBTSJ,RT11SJ,RC
```

For F/B:

```
.R LINK
*RCMNFB.SYS,MAP=RCBTFB,RT11FB,RC
```

5. If step 3 was not done and step 4 used the current monitor object modules, then the monitor tables must be patched to enter the device information. The monitor device tables are located using the procedure in Section 5.3. An additional table, the \$ENTRY entry point table, must also be patched. For this example, assume the table addresses are:

<u>Table</u>	<u>S/J Address</u>	<u>F/B Address</u>
\$HSIZE	13674	14602
\$DVSIZ	13730	14636
\$PNAME	16516	17640
\$STAT	16552	17674
\$ENTRY	16612	17612

NOTE

These table addresses are for illustration only. Consult Table 2 of RT-11 System Release Notes (V02C) for the table addresses of the current monitor release and for the address of BASE.

A link map was made during the linking sequence in Step 4. Locate the value of the system handler entry point, xxSYS. For this example, the tag is RCSYS and its value is found to be 56266 for F/B. This value is put in the \$ENTRY table. The other values were determined in Section 5.3:

```
$HSIZE = 316
$DVSIZ = 2000
$PNAME = 70370
$STAT  = 100023
$ENTRY = 56266 (F/B) 45056 (S/J)
```

The patch procedure for the S/J monitor, using the twelfth slot, would then be:

```
.R PATCH<CR>
PATCH Version number
FILE NAME--
*RCMNSJ.SYS/M<CR>           [The /M is necessary;
*BASE;0R<CR>                Monitor base;
*0,13674/ 4000 316<CR>      $HSIZE table;
*0,13730/ 0 2000<CR>        $DVSIZ table;
*0,16516/ 6250 70370<CR>    $PNAME table;
*0,16552/ 4 100023<CR>     $STAT table;
*0,16612/ 0 45056<CR>      $ENTRY table;
*E                            Exit to monitor]
:
```

For the F/B monitor:

```
.R PATCH<CR>
PATCH Version number
FILE NAME--
*RCMNFB.SYS/M<CR>
*BASE;0R<CR>
*0,14602/ 4000 316<CR>      [$HSIZE table;
*0,14636/ 0 2000<CR>        $DVSIZ table;
*0,17640/ 6250 70370<CR>    $PNAME table;
*0,17674/ 4 100023<CR>     $STAT table;
*0,17564/ 0 56266<CR>      $ENTRY table;
*E                            Exit to monitor]
:
```

The new monitor is now complete and may be used by transferring it to an RC disk and renaming it to MONITR.SYS.

5.5 DEVICES WITH SPECIAL DIRECTORIES

The RT-11 monitor can interface to devices having nonstandard (that is, non RT-11) directories. This section discusses the interface to this type of device.

5.5.1 Special Devices

Special devices are file-structured devices that do not use an RT-11 directory format. Examples are magtape and cassette as supported under RT-11. They are identified by setting bit 12 in the device status word. The USR processes directory operations for RT-11 directory-structured devices; for special devices, the handler must process directory operations (LOOKUP, ENTER, CLOSE, DELETE), as well as data transfers.

5.5.1.1 Interfacing to Special Device Handlers - There are three types of processes that a special device handler must perform:

1. Directory operations (.LOOKUP, .ENTER, etc.)
2. Data transfer operations (.READ, .WRITE)
3. Special operations (rewind, backspace, etc.)

The particular process required is passed to the handler in the form of a function code, located in the even byte of the fourth word of the I/O queue element (see Section 5.1.1). The function code may be positive or negative. Positive codes are used for processes of types 1 and 2 above; negative codes indicate device-dependent special functions.

The positive function codes are standard for all devices and include:

<u>Code</u>	<u>Function</u>
Ø	Read/Write
1	Close
2	Delete
3	Lookup
4	Enter

These functions correspond to the programmed requests .READ/.WRITE, .CLOSE, .DELETE, .LOOKUP, and .ENTER, described in Chapter 9 of the RT-11 System Reference Manual. The .RENAME request is not supported for special devices.

A queue element for a special handler will look identical to an element for a standard RT-11 handler when the function is a .READ/.WRITE (negative word count implies a .WRITE). For the remaining positive functions, word 5 of the queue element (the buffer address word discussed in Section 5.1.1) will contain a pointer to the file descriptor block, containing the device name, file name, and file extension in .RAD5Ø format.

Negative function codes are used for device-dependent special functions. Examples of these are backspace and rewind for magtape. Because these functions are characteristic of each device type, no standard definition of negative codes is made; they are defined uniquely for each device.

Software errors (for example, file not found or directory full) occurring in special device handlers during directory operations are returned to the monitor through the procedure described next. A unique error code is chosen for each type of error. This error code is directly returned by placing it in SPUSR (special device USR error), located at a fixed offset (272) into RMON. (Section 2.5.1 discusses monitor fixed offsets.) Hardware errors are returned in the usual manner by setting bit Ø in the channel status word pointed to by the second word of the queue element.

5.5.1.2 Programmed Requests to Special Devices - Programmed requests for directory operations and data transfers to special devices are handled by the standard programmed requests. When a .LOOKUP is done, for example, the monitor checks the device status word for the special device bit. If the device has a special directory structure, the proper function code is inserted into the queue element and the element is directly queued to the handler, by-passing any processing by the RT-11 USR. Device independence is maintained, since .READ, .WRITE, .LOOKUP, .ENTER, .CLOSE, and .DELETE operations are transparent to the user.

Requests for device-dependent special functions having negative function codes, must be issued by using the .SPFUN special function programmed request, described in Chapter 9 of the RT-11 System Reference Manual. Devices which need to use the .SPFUN requests must have a bit set in the device status table (see Section 2.5.2.2).

5.6 ADDING A SET OPTION

The Keyboard Monitor SET command permits certain device handler parameters to be changed from the keyboard. For example, the width of the line printer on a system can be SET with a command such as:

```
SET LP WIDTH=80
```

This is an example of a SET command that requires a numeric argument. Another type of SET command is used to indicate the presence or absence of a particular function. An example of this is a SET command to specify whether an initial form feed should be generated by the LP handler:

SET LP FORM	(generate initial form feed)
SET LP NOFORM	(suppress initial form feed)

In this case, the FORM option may be negated by appending the NO prefix.

The SET command is entirely driven by tables contained in the device handler itself. Making additions to the list of SET options for a device is easy, requiring changes only to the handler, and not to the monitor. This section describes the method of creating or extending the list of SET options for a handler. The example handler used is the LP/LS11 line printer handler, listed in Appendix A in Section A.3. The SET command is described in Chapter 2 of the RT-11 System Reference Manual.

Device handlers have a file name in the form xx.SYS, where xx is the 2-letter device name; e.g., LP.SYS. Handler files are linked in save image format at a base address of 1000, in which a portion of block 0 of the file is used for system parameters. The rest of the block is unused, and block 0 is never FETCHed into memory. The SET command uses the area in block 0 of a handler from 400 to 776 (octal) as the SET command parameter table. The first argument of a SET command must always be the device name; e.g., LP in the previous example command lines. SET looks for a file named xx.SYS (in this case LP.SYS) and reads the first two blocks into the USR buffer area. The first block contains the SET parameter table, and the second block contains handler code to be modified. When the modification is made, the two blocks are written out to the handler file, effectively changing the handler.

The SET parameter table consists of a sequence of 4-word entries. The table is terminated with a zero word; if there are no options available, location 400 must be zero. Each table entry has the form:

.WORD	value	
.RAD50	/option/	[2 words of RAD50]
.BYTE	<routine-400>/2	
.BYTE	mode	

where:

value	is a parameter passed to the routine in register 3.
option	is the name of the SET option; e.g., WIDTH or FORM.
routine	is the name of a routine following the SET table that does the actual handler modification.
mode	indicates the type of SET parameter: a. Numeric argument - byte value of 100 b. NO prefix valid - byte value of 200

The SET command scans the table until it finds an option name matching the input argument (stripped of any NO prefix). For the first example command string, the WIDTH entry would be found (area 2 in the listing in Section A.3). The information in this table entry tells the SET processor that O.WIDTH is the routine to call, that the prefix NO is illegal and that a numeric argument is required. Routine O.WIDTH is located at area 4 on the listing. It uses the numeric argument passed to it to modify the column count constant in the handler. The value passed to it in R3 from the table is the minimum width and is used for error checking.

The following conventions should be observed when adding SET options to a handler:

1. The SET parameter tables must be located in block 0 of the handler file and should start at location 400. This is done by using an .ASECT 400 (area 1 on the listing).
2. Each table entry is four words long, as described previously. The option name may be up to six .RAD50 characters long, and must be left-justified and filled with spaces if necessary. The table terminates with a zero (area 3 on the listing).

3. The routine that does the modification must follow the SET table in block 0 (area 4 on the listing). It is called as a subroutine and terminates with an RTS PC instruction. If the NO prefix was present and valid, the routine is entered at entry point +4. An error is returned by setting the C bit before exit. If a numeric argument is required, it is converted from decimal to octal and passed in R0. The first word of the option table entry is passed in R3.
4. The code in the handler that is modified must be in block 1 of the handler file, i.e., in the first 256 words of the handler. See areas 6 and 7 on the listing for code modified by the WIDTH option.
5. Since an .ASECT 400 was used to start the SET table, the handler must start with an .ASECT 1000. See area 5 on the listing.
6. The SET option should not be used with system device handlers, since the .ASECT will destroy the bootstrap and cause the system to malfunction.

5.7 CONVERTING USER-WRITTEN HANDLERS

User-written device handlers must, in all cases, conform to the standard practices for Version 2 (2B and 2C). General programming information is discussed in Appendix H of the RT-11 System Reference Manual. Points to consider when converting user-written device handlers (written under Version 1 of the RT-11 system) follow; the details of these procedures have already been discussed.

1. The last word of a device handler is used by the monitor, thus the user must be sure to include one extra word at the end of his program when indicating the handler size.
2. The third header word of the handler should be 340, indicating that the interrupt should be taken at level 7.
3. It is not necessary to save/restore registers when the handler is first entered, although to do so is not harmful.
4. When an interrupt occurs, the handler must execute an .INTEN request or its equivalent. On return from .INTEN, R4 and R5 may be used as scratch registers. Device handlers may not do EMT requests without executing a .SYNCH request.
5. The handler must return from an interrupt via an RTS PC.
6. When the transfer is complete, the handler must exit to the monitor to terminate the transfer or enter a completion routine. When return is made to the monitor, R4 should point to the fifth word of the handler.

7. The handler should contain an abort entry point (located at INTERRUPT SERVICE -2) to which control is transferred on forced exit. The abort entry point should contain a BR instruction to code that will perform the necessary operations (stop device action and exit to monitor completion code).

CHAPTER 6

F/B MONITOR DESCRIPTION

The RT-11 Foreground/Background Monitor permits two jobs to simultaneously share memory and other system resources. The foreground job has priority and executes until it is blocked (i.e., execution is suspended pending satisfaction of some condition, such as I/O completion). When the foreground job is blocked, the background job is activated and executes until it finishes or until the foreground blocking condition is removed.

6.1 INTERRUPT MECHANISM AND .INTEN ACTION

All interrupt handlers must be entered at priority level 7 and must execute a .INTEN request on entry. The handler will then be called (as a co-routine of the monitor in system state) at its normal priority level. This is essential to the operation of RT11 for two reasons:

1. As a co-routine of the monitor, the interrupt handler exits to the monitor, which then does job scheduling.
2. Because of the above condition, there is a danger that interrupt processing may be postponed due to a context switch. For example, if a disk interrupts a lower priority device handler and goes to I/O completion, the monitor may switch to the foreground job and delay the lower priority interrupt until the foreground job is again blocked. By requiring the .INTEN request of all interrupt handlers, the monitor can assure that all interrupts are processed before the context switch is made.

The .INTEN request is implemented as a JSR R5 to the first fixed-offset location of RMON, which contains a jump to the interrupt entry code. This code saves R4 (R5 was saved by the JSR) and increments the system state counter. If the interrupt occurred on a job stack, the stack pointer is switched to use the system stack. The priority is lowered

to the handler's requested priority and control returns to the handler via another JSR instruction.

The handler interrupt code now executes in system state, with several results: any further interrupts are handled on the system stack, preventing their loss by a context switch to another job's stack; a context switch or completion routine cannot occur until all pending interrupts are processed; any error occurring in the handler occurs in system state, causing a fatal halt. When the handler exits via an RTS PC instruction, control returns to the monitor, which can now enter the scheduling loop if all interrupts have been processed.

6.2 CONTEXT SWITCH

When passing control from one job to another, the F/B Monitor does a complete context switch, changing the machine environment to that of the new job. The current context is saved on the stack of the current job and is replaced by the context of the new job.

The information saved on the stack includes:

1. The general registers (R0-R5)
2. The system communication area (memory locations 34-52)
3. The FPP registers, if used
4. The list of special locations supplied by the job (via .CNTXSW), if any

In addition, the stack pointer (R6) is saved in the job's impure area at offset I.SP (=50). The switch requires a minimum of 23_{10} words of stack, not including the special swap list.

The following are the minimum calculated times to context switch between jobs. The assumptions are that the F/G job is waiting for I/O completion, the handler completes an I/O request, and there are no user I/O completion routines.

Processor	$\frac{11}{20}$	$\frac{11}{40}$	$\frac{11}{45}$
(core memory)	.66 ms	.36 ms	.28 ms

6.3 BLOCKING A JOB

The F/B Monitor gives priority to the foreground job, which runs until it is blocked by some condition. In this case, the background, if runnable (i.e., not blocked itself), is scheduled. The conditions which may block a job are flagged in the I.JSTA word, which is located in the job's impure area:

<u>Tag</u>	<u>Bit in I.JSTA Word</u>	<u>Condition</u>
TTIWT\$	14	Waiting for terminal input
TTOWT\$	13	Waiting for room in output buffer
CHNWT\$	11	Waiting for channel to complete
SPND\$	10	Suspended
NORUN\$	9	Not loaded
EXIT\$	8	Waiting for all I/O to stop
KSPND\$	6	Suspended from KMON
USRWT\$	4	Waiting for the USR

6.4 JOB SCHEDULING AND USE OF .SYNCH REQUEST

The F/B Monitor uses a scheduling algorithm to share system facilities between two jobs. The goal of the scheduler is to maximize system utilization, with priority given to the foreground job. The scheduler is generalized to use job numbers for scheduling, the higher job number having the higher priority. The background job is assigned job number 0 and the foreground job number 2. Job numbers must be even.

The foreground job runs until it is blocked by some condition (see Section 6.3), at which point the scheduler is initiated. The job list is scanned top down (from highest to lowest priority) for the highest priority job that is runnable. A job is runnable if it is not blocked, or if it is only blocked pending completion and is not suspended. If no jobs are currently runnable, the idle loop is entered.

If the new job is runnable, a context switch is made. The context switch routine tests for the completion pending condition (i.e., I/O is finished and a user completion routine was queued). In this case, a pseudo-interrupt is placed on the job's stack to call the completion queue manager when the scheduler exits to the job.

The scheduler is event driven and is entered from the common interrupt exit path whenever an event has occurred which requires action by the scheduler. The set of such events include:

1. An .EXIT or .CHAIN request
2. A job abort from the console, or an error abort
3. I/O transfer completed
4. Expiration of timed wait
5. A blocking condition encountered:
 - a. .TWAIT request or SUSPEND command
 - b. .TTYIN or .CSI waiting for end of line
 - c. .TTYOUT or .PRINT waiting for room in output buffer
 - d. Attempt to use busy channel
6. A blocking condition removed
7. No queue elements available
8. .SYNCH request (see below).

The .SYNCH request is used in interrupt routines to permit the issuing of other programmed requests. The .SYNCH macro is expanded as a JSR R5 to the .SYNCH code in the F/B resident monitor. The .SYNCH routine uses the associated 7-word block as a queue element for the completion queue.

If the .SYNCH block is not in use, register R5 is incremented to the successful return address and placed in the block as the completion address. The word count is set to -1 to prevent the block from being linked into the AVAIL queue. The block is placed in the completion queue, at its head, and the job associated with the .SYNCH request is flagged to have a completion routine pending. A request for a job switch is entered before the .SYNCH logic exits with an interrupt return.

On exit from the interrupt with a job switch pending, the scheduler is entered and the completion queue manager is called. When control finally returns to the code following the .SYNCH request, it is executing as a completion routine at priority level 0. It can now issue programmed requests without fear of being interrupted. If another interrupt comes in, and it requests a completion routine, the completion routine will be queued pending return of the current

completion routine, since the .SYNCH block is freed before calling the completion routine. Further interrupts will be rejected by the .SYNCH code, unless provision is made for supplying extra .SYNCH blocks.

6.5 USR CONTENTION

The directory operations handled by the USR are not re-entrant, particularly since the directory segment is buffered within the USR. Therefore, to use the USR in F/B, a job must have ownership of the USR. To facilitate this, the F/B monitor maintains a USR queuing mechanism.

Before issuing a USR request, a job must request ownership of the USR. If the USR is in use by another job, even of lower priority, the requesting job is blocked and must wait for the USR. The USRWT\$ flag is set in the I.JSTA word (see Section 6.3) and the job cannot continue until the USR is released and the blocking bit cleared. When the USR is released, the job list is scanned for jobs waiting for the USR, starting with the job having highest priority.

Because of the impact this may have on system performance, CSI requests are handled differently in the F/B system than in the S/J Monitor. If the command string is to come from the console keyboard, the prompting asterisk is printed and then the USR is released, pending completion of command line input. This prevents a job doing a CSI request from locking up the USR and blocking another, perhaps higher priority, job from executing. A job can determine if the USR is available by doing a .TLOCK request (see Chapter 9 of the RT-11 System Reference Manual).

6.6 I/O TERMINATION

Because of the multi-job capabilities of RT-11 F/B, termination of I/O on job exit or abort must be handled differently than in the S/J Monitor. The use of the RESET instruction is unacceptable, and a form of I/O rundown must be used. This is done by the IORSET routine, called when doing an abort or hard exit.

The IORSET routine searches the queue of every resident handler for elements belonging to the aborted job. If a handler is found to be resident and active (i.e., there are elements on its queue), the IORSET routine "holds" the handler from initiating a new transfer by setting bit 15 of the LQE word (entry point) in the handler. The

current transfer may complete, but the hold bit will prevent the queue manager from initiating a new transfer.

While it is held, the handler's queue is examined for the current request. If it belongs to the aborted job, the handler's abort entry point is called to stop the transfer. The queue of pending I/O requests is then examined and any elements belonging to the aborted job are discarded. The hold flag is cleared and a test is made to see if the current transfer completed while the handler was held. If it did, the completion queue manager, COMPLT, is again called to return the completed element and initiate the next transfer. At this point, any elements belonging to the aborted job will have been removed from the queue.

After the device handlers are purged, the internal message handler is examined for waiting messages that were originated by the aborted job. All such messages are discarded. Finally, all mark time requests belonging to the aborted job are cancelled.

CHAPTER 7

RT-11 BATCH

The RT-11 BATCH system is composed of a BATCH compiler and a run-time handler. The BATCH compiler converts BATCH Job Control language into a format comprehensible to the BATCH run-time handler. The compiler creates a control (CTL) file (from the BATCH language statements) which is then scanned by the handler; the CTL format is a versatile programming language in its own right. The result is a BATCH system that is simple to use, and yet easily customized to handle different situations.

7.1 CTL FORMAT

The BATCH run-time handler uses a unique language format that includes many programming features, such as labels, variables, and conditional branches. The directives are explained in detail in Chapter 12 of the RT-11 System Reference Manual.

Each directive consists of a backslash character followed by one or more other characters. For example, to run PIP and generate a listing, the CTL directives \E (execute) and \D (data line) are used:

```
\ER PIP
\DLP:=/L
```

Messages are sent to the console device by using the \@ directive:

```
\@ PLEASE MOUNT DT2
```

Labels and unconditional branches are implemented with the \L (label) and \J (jump) directives:

```
\JEND 1
.
.
\LEND
```

Each BATCH command is sent to the log as it is executed, using the \C (comment) directive:

```
\C
$JOB
```

In this case, every character up to the next backslash is sent to the log.

7.2 BATCH RUN-TIME HANDLER

The BATCH run-time handler (BA.SYS) is constructed as a standard RT-11 device handler. To use the handler, it must be made permanently resident via the monitor LOAD command. The handler links itself into the monitor, intercepting certain EMTs described later.

The linking occurs the first time the BATCH compiler is run after the BA handler is loaded. The compiler does a .READW to the BA handler, which then links itself to the monitor and returns a table of addresses to the BATCH compiler. The linking is achieved by replacing the addresses of monitor EMT routines with corresponding addresses in the BATCH handler. Those EMTs that are diverted include:

<u>EMT</u>	<u>BATCH Handler Routine</u>
.TTYIN	B\$TIN
.TTYOUT	B\$TOT
.EXIT	B\$EXT
.PRINT	B\$PRN

Once the link is established, the BATCH handler cannot be unloaded. The links must first be undone by again running the BATCH compiler and specifying the /U switch. The compiler removes the links and prints a prompting message, after which the UNL BA command can be issued.

With the BA handler linked to the monitor, all console terminal communication is diverted to BA, along with program exits. The BA handler then dispatches the program request to the monitor routine or diverts it to a routine in BA, depending on the values of switches in BATSW1. The switches are:

<u>TAG</u>	<u>BIT</u>	<u>DESCRIPTION</u>
HELP	0	0 = Do not log terminal input (.TTYIN) 1 = Log terminal input
DESTON	1	0 = EMT is going directly to monitor 1 = BA intercepts the EMT
SOURCE	2	0 = Character input by monitor from console terminal 1 = Character input comes from BATCH stream
COMWAT	3	0 = No command 1 = Command is waiting
ACTIVE	4	0 = Console terminal inactive 1 = Console terminal is active; i.e., BA is waiting for input from console terminal
DATA	5	0 = Characters are going to KMON; i.e., KMON is active in B/G 1 = Characters are going to B/G programs
BDESTN	6	0 = Output characters are going to console terminal 1 = Output characters are going to LOG
BGET	7	0 = Normal mode 1 = Get mode (\G); input comes from console terminal until <CR><LF> is encountered
NOTTY	8	0 = Log terminal output 1 = Do not log terminal output (.TTYOUT, .PRINT)
	9-13	Reserved
BSOURC	14	0 = BA directives come from console terminal 1 = BA directives come from CTL file
BEXIT	15	1 = A program has done an .EXIT while DATA switch was set

The BATSW1 word, located six bytes past the handler entry point, determines the state of the system at any given moment. If the word is zero, RT-11 operates normally. When the DESTON bit is set, EMTs are diverted to routines in BA for action, but the specific action taken by those routines is determined by the other switch bits.

For example, if the BDESTN bit is set, output from .TTYOUT and .PRINT is diverted from the console terminal to the log device. If SOURCE is set, the characters for the .TTYIN request are taken from the BATCH stream rather than from the console terminal via the monitor ring buffer. Directives for the BA handler itself may come from either the CTL file or the console terminal, depending on the state of the BSOURC bit.

The state of the background is reflected in the DATA bit. Either the KMON is active (DATA=0) or a program is active (DATA=1). If a program issues an .EXIT request while in DATA mode, the BEXIT state is entered until the BA handler encounters the next KMON directive (\E) in the BATCH stream, causing any unused \D lines to be ignored. A program can be aborted by diverting any of the .TTYIN, .TTYOUT or .PRINT requests to the .EXIT code in the monitor.

7.3 BATCH COMPILER

The obvious function of the BATCH compiler is to convert BATCH Standard Commands into the BA handler directives mentioned in Section 7.1, creating a control (CTL) file. BATCH jobs entered from a card reader or a file-structured device are compiled into a CTL file stored on a file-structured device for execution by the BA handler. However, the BATCH Compiler has other important functions; these are described in this section along with details on the initiation and termination of BATCH jobs.

7.3.1 BATCH Job Initiation

The following sequence of actions is performed by the BATCH Compiler when setting up a job for execution:

1. A check is made to ensure that LOG and BA device handlers are loaded and assigned properly. The LOG handler must be assigned the logical name LOG;; the BATCH Compiler may be run several times during the course of a job to do special tasks for the BA handler, and it will reference LOG:.
2. A nonfile-structured .LOOKUP is done on BA and a .READW is issued. If this is the first time BATCH has been run since BA was loaded, the handler links itself to the monitor (see Section 7.2). BA returns a list of

eleven pointers to important parameters within BA.
These include:

- BA state word (BATSW1)
- CTL file savestatus area (INDATA)
- LOG file savestatus area (ODATA)
- Output (LOG) buffer (OUTBUF)
- Output buffer pointer (BATOPT)
- Output character counter (BATOCT)
- Input character counter (BATICT)
- Monitor EMT dispatch address save areas

3. A command string is collected from the console terminal and is processed by .CSISPC. An input file must be specified.
4. If the input file is a .BAT file to be compiled, a .CTL file is entered. If the LOG: device is file-structured, a fixed-size enter is done and then the file is initialized by writing zeroes in all blocks.
5. A .LOOKUP is done on all input files.
6. The .LOG file is .CLOSED so that a .LOOKUP and .SAVE-STATUS may be done. The savestatus data is placed in the ODATA area in BA.
7. If the input file is a .BAT file, it is now compiled, with output going into the .CTL file.
8. The .CTL file is closed, again so that a .LOOKUP and .SAVESTATUS may be done. The .SAVESTATUS data is transferred to the INDATA area in BA. Buffer pointers and counters in BA are initialized.
9. The BA handler is activated by setting the SOURCE, DESTON, BSOURC and BDESTN bits in the BATSW1 state word in BA. Control passes to BA when the compiler does an .EXIT, assuming an abort is not requested.
10. If an abort is requested (an error occurred during compilation or the /N switch was used), the .LOG file is .REOPENed and all \$ command lines are logged out with any error diagnostics. The BATSW1 word is then cleared before exiting, preventing the execution of the job.

The following switches are used by the BATCH system during job initiation and continuation, and should not be typed by the user:

- /B BATCH continuation of jobs in input stream
- /D Print the physical device name assigned a logical device name in a \$DISMOUNT command
- /M Make a temporary source file
- /R Return from \$CALL
- /S \$CALL subroutine

7.3.2 BATCH Job Termination

Every BATCH job must be terminated with an \$EOJ statement. The \$EOJ statement causes the compiler to insert the CTL directives:

```
\R BATCH  
\D/R
```

The /R switch for the BATCH compiler, which is legal only when entered from a BATCH stream, is used to terminate a BATCH job. This switch causes the compiler to pop the BATCH stack up a level. If the stack was empty, the stream is finished and the compiler cleans up, clears the BATSW1 word in BA, and exits. If the stack is not empty, the /R switch implies a return from a \$CALL. The stack contents are used to restore parameters in the BA handler so that control will return to the calling BATCH stream at the next statement after the \$CALL.

7.3.3 BATCH Compiler Construction

The BATCH Compiler is constructed in two pieces: a data area and a program area. The data area is located in low memory, in a .CSECT named UNPURE. The contents are described in the accompanying table (Table 7-1). The program section, located in the .CSECT named PROGRM, starts at the symbol START. The general register R4 always points to UNPURE and all references to the data base are made as indexed references relative to R4.

Locations in the data base are created with the ENTRLO macro. For example,

```
ENTRLO BOTLCT,0
```

allocates one word in the data base and initializes it to zero. The symbol BOTLCT is an offset into the data base, so that references to BOTLCT are made in the form BOTLCT(R4).

Table 7-1
BATCH Compiler Data Base Description

Tag	Byte Offset	Description
BATSWT	0	BATCH Control Switches ABORT = 100000 ABORT after compile DATDOL = 40000 DATA or DOLLARS set NO = 20000 "NO" prefix on switch CTYOUB = 10000 Output to CTY (\@) LOGOUB = 4000 Output to LOG (\C) DATOUB = 2000 Output to user prog (\D) COMOUB = 1000 Output to monitor (\E) JOB = 400 \$JOB encountered MAKEB = 200 /B switch on command COMMA = 100 Comma terminates command BFORLI = 40 Next link requires FORTRAN library UNIQUE = 20 UNIQUE command option set BANNER = 10 Print BANNER on \$JOB, \$EOJ RT11 = 4 RT11 default on NO '\$' in Column 1 TIME = 2 Print time of day MAKE = 1 Create a source file
BATSW2	2	More BATCH Control Switches ABORT =100000 Second time through ABORT FIRST = 10000 First card processed SBIT = 4000 /S switch on command SEQ = 2000 \$SEQ card processed LSTBIT = 1000 Request temporary listing file COMSWB = 400 Command switches MAKEB = 200 Same as BATSWT STARFD = 100 Asterisk in FD field STAROK = 40 Wild card option is valid BNOEOJ = 20 \$JOB or \$SEQ before \$EOJ LSTDAT = 10 List DATA sections BEOF = 4 EOF encountered on .BAT file XSWT = 2 /X switch set EOJ = 1 \$EOJ encountered
TMPSWT	4	Temporary command switches
COMSWT	6	Current command switches
LINSIZ	10	Input line buffer size
BINLCT	12	Last buffer character count
INSTAT	14	Input buffer status (see OTSTAT)
ICHRPT	16	Input character pointer
BINCTR	20	Input buffer counter

(continued on next page)

Table 7-1 (Cont.)
BATCH Compiler Data Base Description

Tag	Byte Offset	Description
BINARG	22	Input file EMT argument list
BATIBK	24	Input file block number
BATIBP	26	Input buffer address
	30	Input buffer size
	32	Wait I/O
BOTLCT	34	Last output buffer character count
OTSTAT	36	Output buffer status BFREE = 1 0 → Buffer is free BWAIT = 2 In I/O wait BEOF = 4 End of file
OCHRPT	40	Output character pointer
BOTCTR	42	Output character count
BOTARG	44	Output file EMT argument list
BATOBK	46	Output file block number
BATOBP	50	Output buffer address
	52	Output buffer size
	54	Wait I/O
STACK	56	Compiler stack pointer save area These are the arguments passed between BATCH and BA:
BATSW1	60	Pointer to BATSW1 in BA.SYS
INDATA	62	Pointer to INDATA
ODATA	64	Pointer to ODATA
OUTBUF	66	Pointer to BATCH handler output buffer
BATOPT	70	Pointer to output character pointer
BATOCT	72	Pointer to output character counter
BATICCT	74	Pointer to input character counter

(continued on next page)

Table 7-1 (Cont.)
BATCH Compiler Data Base Description

Tag	Byte Offset	Description
		Pointers to EMT intercept pointers:
O\$EXT	76	.EXIT
O\$TIN	100	.TTYIN
O\$TOT	102	.TTYOUT
O\$PRN	104	.PRINT
		CSI Buffer:
SPC0	106	Channel 0
SPC1	120	1
SPC2	132	2
SPC3	144	3
SPC4	154	4
SPC5	164	5
SPC6	174	6
SPC7	204	7
SPC8	214	10
LINIMP	224	Pointer to command line buffer (LINIMM)
LINIMM	226	Command line input buffer
LINIMS	350	Command line buffer save area
LIBLST	470	ASCIZ name of FORTRAN default library plus a line buffer
BATIBF	610	BATCH Compiler input buffers (INBSIZ * 2)
BATOBF	2610	BATCH Compiler output buffers (OTBSIZ * 2)
QSET	4610	Seven I/O queue elements for double/buffering
SOUTMP	4700	Source temporary file descriptor
OBJTMP	4714	Object temporary file descriptor
LOGTYP	4730	LOG device status word (word 0 of .DSTATUS)
ARGARG	4732	EMT argument list for BA handler initializa- tion

(concluded on next page)

Table 7-1 (Cont.)
 BATCH Compiler Data Base Description

Tag	Byte Offset	Description
STKBLK	4744	EMT argument list for READ/WRITE of BATCH stack
DEFCHN	4756	Default channel numbers
DEVSPC	4770	Pointer to device handler space
WDBLK2	4772	Two-word EMT argument block
WDBLK5	5000	Five-word EMT argument block
FTLPC	5012	Contents of PC on BATCH fatal error
AREA0	5014	Pointer to impure area
LSTTMP	5016	Listing temporary file descriptor
SWTMSK	5026	Switch mask for this BATCH directive
FD0	5030	File descriptor 0 for BATCH directive
FD1	5034	1
FD2	5040	2
FD3	5044	3
FD4	5050	4
FD5	5054	5

7.4 BATCH EXAMPLE

The following example demonstrates how the compiler converts BATCH Standard Commands into RT-11 BATCH handler directives. The example consists of a main BATCH stream, EXAMPL.BAT, and a BATCH subroutine file, EDITIT.BAT. EXAMPL creates a program, assembles and runs it. The program, called FILE.MAC, prints a message that is diverted to the log. The listing file from the assembly is printed and then deleted. The BATCH variable S is then tested and, if it is zero, the BATCH subroutine EDITIT is called. The EDITIT stream uses EDIT to edit the file FILE.MAC, changing the message to be printed. After return from EDITIT, the stream branches unconditionally to label L1, repeating the assembly and execution of FILE.MAC. EDITIT increments the variable S before returning, so that the BATCH stream, on encountering the IF statement again, now branches to label L2, skipping the call to EDITIT. \$DIRECTORY and \$DELETE operations are performed before finally exiting from BATCH.

Note the following about the .CTL files created:

1. The \$JOB command produces a comment for the log (the \C directive, but no action directives). Its function is to initialize the BATCH compiler.
2. The \$CREATE command produces directives that run the BATCH compiler, using the file name to be created with a /M switch. This is a special function of the BATCH compiler used to create data files. The compiler will enter the data that follows in the CTL file into the newly created file, until an EOF (CTRL/Z) is encountered. The data is fed to the compiler by the BATCH handler through the .TTYIN programmed request. After the EOF character is encountered, the BATCH compiler closes the new file and exits, returning control to the BATCH handler through the .EXIT request. In this example, the file created is called FILE.MAC.
3. The \$MACRO command has the /RUN switch appended, which forces the compiler to generate a series of assembly, link and execute instructions. A temporary execution file, 000000.SAV, is created from the assembled object module, FILE.OBJ. After execution with the monitor R command, the temporary execution file is deleted with PIP.
4. PIP is used to implement \$PRINT, \$DELETE, \$COPY, and \$DIRECTORY. The compiler translates these commands into the appropriate PIP command strings.

5. The variable S is defined to be zero with the LET statement. This translates into the BATCH handler directive,

`\KS1<null>`

which instructs the BATCH handler to set variable S to the value in the byte following the character 1.

6. Labels are implemented by inserting a \L directive followed by the 6-character label name into the CTL stream where the label was declared. The label is also logged out with the \C directive so that the labels will appear in the log.
7. The unconditional branch, or GOTO command, is implemented with the \J directive immediately followed by the label. Note that the BATCH programmer must indicate whether the branch is forward or reverse. In this case, the branch is a backward reference and a minus sign is prefixed to the label:

`GOTO     -L1`

There is no error checking done by the compiler. If an error is made (e.g., the minus sign is left off the L1), the BATCH handler searches forward in the CTL stream until it finds the label. Since an error was made, the label will not be found. The search (and consequently the BATCH job) terminates when the label stopper (\L\$\$\$\$\$) is encountered at the end of the CTL file.

8. The IF conditional branch is implemented with the \I directive. The \I directive is followed by the name of the variable to be tested, the value to be tested against, and three label fields. Each label field consists of the 6-character label name with a reference character appended. The character 1 indicates the label is a forward reference, a 0 indicates a backward reference. The test value is subtracted from the current value of the variable and the appropriate branch is taken. If no label is specified for a field, it is filled with spaces and causes the BATCH stream to fall through to the next command if that branch is elected.
9. The \$CALL command is very useful and permits a BATCH stream to call another BATCH file as a subroutine, with control returning to the command following the \$CALL. The \$CALL is implemented by simply running the BATCH compiler, passing it the name of the \$CALLED routine with a /S switch appended. Another BATCH compile/execute sequence will follow, but the /S switch will cause the compiler to save certain locations in the BATCH handler in an internal stack in the BA.SYS file. In this example, the \$CALL EDITIT statement causes the file EDITIT.BAT to be compiled and executed.

10. BATCH variables may be used to enter ASCII values into a job stream. In the file EDITIT, the variable A is set equal to the value of the ESC (or ALT MODE) character. The variable A is inserted into a string of EDIT commands in place of the ALT MODE character.
11. The \$EOJ must terminate every BATCH job. The \$EOJ command generates the stopper label, \L\$\$\$\$\$, and then produces directives to run the BATCH compiler again, this time with a /R switch. The compiler, when given a /R switch, checks the BATCH stack. If it is empty, the compiler exits. Otherwise, the stack is popped to restore conditions in the BATCH handler prior to the \$CALL causing the push, and the BATCH stream continues. The \$EOJ finally generates a \E to bring in the KMON and a \F<CR> to terminate the BATCH stream.

EXAMPL.BAT

```
$JOB
$MESSAGE EXAMPLE BATCH STREAM
$CREATE FILE,MAC
      ,MCALL .REGDEF,.PRINT,.EXIT
      .REGDEF

START: .PRINT #MSG
      .EXIT
      .NLIST REY
MSG:    .ASCIZ /THIS MESSAGE COMES FROM THE BATCH STREAM/
      .EVEN
      .LIST REY
      .END START

$EOD
$RT11
      LET S=0

L1:
$MACRO/RUN FILE,LST/LIST FILE,MAC/INPUT FILE/OBJECT
$PRINT FILE,LST
$DELETE FILE,LST
$RT11
      IF(S=0) ,,L2
$CALL EDITIT !CALL EDITIT TO EDIT FILE,MAC
$RT11
      GOTO -L1

L2:
$DIRECTORY FILE.*
$DELETE FILE.*
$EOD
```

EDITIT.BAT

```
$JOB/RT11
$! JOB TO EDIT FILE,MAC
      %S !INCREMENT S TO PREVENT RECURSION
      LET A=33 !A IS ALT MODE

.R EDIT
*ERFILE,MAC'A'R'A'A'
*GMSG:'A'KI .ASCIZ /MODIFIED BY EDITOR RUN BY BATCH/
*'A'EX'A'A'
$EOD
```

EXAMPL.CTL

```
\C
$JOB

SMESSAGE EXAMPLE BATCH STREAM
\EX0 EXAMPLE BATCH STREAM
\C
$CREATE FILE.MAC
\ER BATCH
\DFILE,MAC/M=
    .MCALL .REGDEF,.PRINT,.EXIT
    .REGDEF
START: .PRINT #MSG
    .EXIT
    .NLIST BEY
MSG:   .ASCIZ /THIS MESSAGE COMES FROM THE BATCH STREAM/
    .EVEN
    .LIST BEY
    .END START
\C
$EOD

$RT11
    LET S=0
\KS1 \LL1 \CL1:

$MACRO/RUN FILE.LST/LIST FILE.MAC/INPUT FILE/OBJECT
\ER MACRO
\DFILE,FILE.LST=FILE.MAC
\FID\ER LINK
\0000000=FILE
\ER \0000000
\ER PIP
\0000000.SAV/D
\C
$PRINT FILE.LST
\ER PIP
\DLST:*.*/X=FILE.LST
\FID\C
$DELETE FILE.LST
\ER PIP
\DFILE.LST/D
\C
$RT11
    IF(S=0) ,,L2
\IS 1 1L2 1\L \C
$CALL EDITIT !CALL EDITIT TO EDIT FILE.MAC
\F\ER BATCH
\0EDITIT/S
\C
$RT11
    GOTO -L1
\JL1 0\LL2 \CL2:
```

EXAMPL.CTL (Cont)

```
$DIRECTORY FILE.*
\ER PIP
\DFILE.* /L
\F\D\C
$DELETE FILE.*
\ER PIP
\DFILE.* /D
\C
$EOJ
\L$S$S$S\F\ER BATCH
\O/R
\E\F
```

EDITIT.CTL

```
\C
$JOB/RT11

$! JOB TO EDIT FILE.MAC
      XS      !INCREMENT S TO PREVENT RECURSION
\KS0\C LET A=33      !A IS ALT MODE
\KA1 \ER EDIT
\DEBFILE.MAC\KA2R\KA2\KA2
\DGMSG:\KA2KI .ASCIZ /MODIFIED BY EDIT RUN BY BATCH/
\O\KA2EX\KA2\KA2
\C
$EOJ
\L$S$S$S\F\ER BATCH
\O/R
\E\F
```

EXAMPL.LOG

\$JOB

\$MESSAGE EXAMPLE BATCH STREAM

\$CREATE FILE,MAC

\$EOD

\$RT11

LET S=0

L1 L1:

\$MACRO/RUN FILE,LST/LIST FILE,MAC/INPUT FILE/OBJECT

*ERRORS DETECTED: 0

FREE CORE: 15100, WORDS

*

EXAMPL.LOG (Cont.)

.MAIN. RT-11 MACRO VM02-10 10-APR-75 10:33:45 PAGE 1

```
1      .MCALL .REGDEF,.PRINT,.EXIT
2 000000      .REGDEF
3 000000      START: .PRINT #MSG
4 000006      .EXIT
5      .NLIST REX
6 000010      124 MSG: .ASCIZ /THIS MESSAGE COMES FROM THE BATCH STREAM/
7      .EVEN
8      .LIST REX
9      000000' .END START
```

EXAMPL.LOG (Cont.)

.MAIN. RT=11 MACRO VM02-10 10-APR-75 10:33:45 PAGE 1+
SYMBOL TABLE

MSG	000010R	PC	=X000007	R0	=X000000
R1	=X000001	R2	=X000002	R3	=X000003
R4	=X000004	R5	=X000005	SP	=X000006

START 000000R

.ABS. 000000 000

000062 001

ERRORS DETECTED: 0

FREE CORE: 15100. WORDS

FILE,FILE,LST=FILE,MAC

THIS MESSAGE COMES FROM THE BATCH STREAM

SPRINT FILE,LST

SDELETE FILE,LST

SRT11

IF(S=0) ,,L2

SCALL EDITIT ICALL EDITIT TO EDIT FILE,MAC

SJOB/RT11

S! JOB TO EDIT FILE,MAC

XS

LET A=33

!INCREMENT S TO PREVENT RECURSION

!A IS ALT MODE

*ERFILE,MACSRSS

*

MSG:SKI

.ASCIZ /MODIFIED BY EDITOR RUN BY BATCH/

SEXSS

SEQJ

SSSSSS

SRT11

GOTO -L1

L1:

\$MACRO/RUN FILE,LST/LIST FILE,MAC/INPUT FILE/OBJECT

*ERRORS DETECTED: 0

FREE CORE: 15136. WORDS

*

EXAMPL.LOG (Cont.)

.MAIN. RT-11 MACRO VM02-10 10-APR-75 10:34:08 PAGE 1

```
1          .MCALL .REGDEF,.PRINT,.EXIT
2 000000    .REGDEF
3 000000    START: .PRINT #MSG
4 000006    .EXIT
5          .NLIST REX
6 000010    115 MSG: .ASCIZ /MODIFIED BY EDITOR RUN BY BATCH/
7          .EVEN
8          .LIST REX
9          000000' .END START
```

EXAMPL.LOG (Cont.)

.MAIN. RT-11 MACRO VM02-10 10-APR-75 10:34:08 PAGE 1+
SYMBOL TABLE

MSG	000010R	PC	=X000007	R0	=X000000
R1	=X000001	R2	=X000002	R3	=X000003
R4	=X000004	R5	=X000005	SP	=X000006
START	000000R				
.ABS.	000000	000			
	000050	001			

ERRORS DETECTED: 0
FREE CORE: 15136. WORDS

FILE,FILE,LST=FILE,MAC

MODIFIED BY EDITOR RUN BY BATCH

\$PRINT FILE,LST

\$DELETE FILE,LST

\$RT11
IF(S=0) ,,12
L2:

\$DIRECTORY FILE.*

10-APR-75
FILE .BAK 1 10-APR-75
FILE .MAC 1 10-APR-75
FILE .OBJ 1 10-APR-75
3 FILES, 3 BLOCKS
417 FREE BLOCKS

\$DELETE FILE.*

\$EOJ

7.5 CTT TEMPORARY FILES

In certain cases the BATCH compiler will produce temporary files with the extension CTT and the file name of the BAT file being compiled. These files occur when a multiple input file command string is issued, or when an unexpected \$JOB or \$SEQ statement occurs in a BATCH stream, or when multiple jobs are run from the card reader or a .BAT file.

The CTT file is actually a CTL file used to link together execution of several BATCH jobs. Each CTT file contains the BA directives:

```
\ER BATCH  
\D/B
```

which execute the BATCH compiler, passing it the /B switch.

The CTT file also contains the following information:

1. Current input channel number (range is 3-10₈)
2. Current input file block number
3. The CTL file descriptor block (device, file name and file size)
4. The LOG file descriptor block (device, file name, and file size)
5. The set of input (BAT) file descriptor blocks (device and file name)

When the CTT file is executed, the compiler restores the input channel number and block number and the entire set of file descriptor blocks from the CTT file. If, for example, the input channel number is 4, the second of a string of .BAT files is compiled and executed.

APPENDIX A

SAMPLE HANDLER LISTINGS

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35

TITLE RC11 V01-01 (FIXED HEAD DISK)
RT-11 RC11/RS64 DEVICE HANDLER

IDFC-11-XXXXX-A

IJFG

1OCTOBER 1974

ICOPYRIGHT (C) 1975

IDIGITAL EQUIPMENT CORPORATION
IMAYNARD, MASSACHUSETTS 01754

THIS SOFTWARE IS FURNISHED UNDER A LICENSE FOR USE ONLY
ON A SINGLE COMPUTER SYSTEM AND MAY BE COPIED ONLY WITH
THE INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS
SOFTWARE, OR ANY OTHER COPIES THEREOF, MAY NOT BE PROVIDED
OR OTHERWISE MADE AVAILABLE TO ANY OTHER PERSON EXCEPT
FOR USE ON SUCH SYSTEM AND TO ONE WHO AGREES TO THESE
LICENSE TERMS. TITLE TO AND OWNERSHIP OF THE SOFTWARE
SHALL AT ALL TIMES REMAIN IN DIGITAL.

THE INFORMATION IN THIS DOCUMENT IS SUBJECT
TO CHANGE WITHOUT NOTICE AND SHOULD NOT
BE CONSTRUED AS A COMMITMENT BY DIGITAL
EQUIPMENT CORPORATION. DIGITAL ASSUMES NO
RESPONSIBILITY FOR ANY ERRORS THAT MAY APPEAR
IN THIS DOCUMENT.

DIGITAL ASSUMES NO RESPONSIBILITY FOR THE
PURE OR RELIABILITY OF ITS SOFTWARE ON
EQUIPMENT WHICH IS NOT SUPPLIED BY
DIGITAL.

```

1          ;SRTL HANDLER DEFINITIONS
2          ;MCALL .REGDEF, .LV2..
3
4 000000          ;LV2..
5
6          ;REGISTER DEFINITION
7          ;LIST ME
8 000000          ;REGDEF
9
10          R0=X0
11          R1=X1
12          R2=X2
13          R3=X3
14          R4=X4
15          R5=X5
16          R6=X6
17          R7=X7
18          ;LIST ME
19
20          ;RT-11 MONITOR DEFINED CONSTANTS
21          MONLOW= 54          ;MONITOR BASE POINTER
22          OFFSFT= 270        ;POINTER TO 0 MANAGER
23
24          ;COMPLETION ENTRY
25          HOPRR= 1           ;HARD ERROR BIT
26          RCTRY= 8.         ;RFTRY FOR ERRORS
27
28          ;PRIORITY CONSTANTS
29          PR7 = 340          ;HANDLER ENTERED AT PR7
30          PR5 = 240          ;HANDLER RUNS AT PR5
31
32          ; RC-11 COMMUNICATION CONSTANTS
33          WR= 103            ;ISFT INTERRUPT ENABLE,WRITE & INITIATE FUNC.
34          RD= 104            ;ISFT INTERRUPT ENABLE,READ & INITIATE FUNC.
35          INHCA= 1000        ;INHIB. INCRP. CURRENT ADR. REG (RCCA)
36          ABORT= 400         ;ABORT OPERATION IN PROGRESS (RCCS)
37          RTRYER= 060000     ;RFTRY AFTER ERROR MASK FOR RCCS
38
39          ;BIT 14 = 1 => DATA ERROR
40          ;BIT 13 = 1 => ADDRESS ERROR
41
42          ;RT-11 CONTROL REGISTERS
43          RCLA = 177440       ;LOOK AHEAD REGISTER
44          RCDA = 177442       ;DISK ADDRESS REGISTER
45          RCER = 177444       ;DISK ERROR STATUS REGISTER
46          RCCS = 177446       ;DISK CONTROL AND STATUS REGISTER
47
48          ;RFTRY AFTER ERROR MASK FOR RCCS
49          ;BIT 14 = 1 => DATA ERROR
50          ;BIT 13 = 1 => ADDRESS ERROR
51
52          ;RT-11 CONTROL REGISTERS
53          RCWC = 177450       ;WORD COUNT REGISTER
54          RCCA = 177452       ;CURRENT ADDRESS REGISTER
55          RCMN = 177454       ;MAINTENANCE REGISTER
56          RCDB = 177456       ;DATA BUFFER REGISTER

```

41 000210 RCVEC = 210 INTERRUPT VECTOR ADDRESS

42

43

44

45

46

47

48

49

000000

IRC SYSTEM DEFINITIONS

.GLOBL RCYSYS, RKSYS, RFSYS, DPSYS, DSSYS, DTSYS, DXSYS
 .GLOBL SINPTR, SINTEN, RCINT
 .GLOBL RCISZ

ISIZE IS REQUIRED BY BSTRAP

A

B

RKSYS = 0

RC11 VM1-01 (FIXED HEAD DISK) RT-11 MACRO VM02-09 A-APR-75 12104126 PAGE 2+
 HANDLER DEFINITIONS

50 000000
 51 000000
 52 000000
 53 000000
 54 000000

B

RFSYS = 0
 DPSYS = 0
 DSSYS = 0
 DTSYS = 0
 DXSYS = 0

RC11 VM1-01 (FIXED HEAD DISK) RT-11 MACRO VM02-09 A-APR-75 12104126 PAGE 3
 RC11 DEVICE HANDLER

1
 2
 3
 4 000000
 5
 6 000000
 7
 8
 9 000000 000210
 10 000002 000060
 11 000004 000340
 12
 13 000006
 14 000006 000000
 15 000010 000000
 16
 17
 18 000012 012727 000010
 19 000016 000000
 20 000020 004067 000226
 21 000024 000000
 22 000026 062704 000010
 23 000032 012514
 24 000034 012544
 25 000036 012705 000103

.SRTTL RC11 DEVICE HANDLER
 (MAXIMUM SUPPORT 1 CONTROLLER AND 4 RS64 DISKS)
 (1024 BLOCKS OF 256 WORDS)
 .CSECT SYSHND

RCSTRT:
 ILOAD POINT

C

D

RCYSI
 RCLGFI
 RCCGFI

.WORD RCVEC
 .WORD RCINT
 .WORD PR7
 .WORD 0
 .WORD 0

ADDRESS OF INTERRUPT VECTOR
 OFFSET TO INTERRUPT ROUTINE
 PRIORITY 7

IPINTER TO LAST Q ENTRY
 IPINTER TO CURRENT Q ENTRY

IENTRY POINT

E

RETRY: MOV #RCTRY, (PC)+
 RCNTRY: .WORD 0
 RCNTRY: .ISR R0, RCCOM1
 ZERO: .WORD 0
 ADD #10, R4
 MOV (R5)+, R4
 MOV (R5)+, -(R4)
 MOV #WR, R5

ISPECIFY THE RETRY COUNT
 IRETRY COUNTER
 ISET UP THROUGH COMMON ROUTINE
 INO ZERO FILL INITIATION
 IPINTER TO RCCA
 ISET BUFFER ADR (RCCA)
 ISET WORD COUNT ADR (RCWC)
 IASSUMP WRITE FUNCTION

26 000042 004714
27 000044 001427
28 000046 100402
29 000050 120525
30 000052 004414
31 000054 010544
32 000056 000207

F

181

TST R4
REQ RCHOME
RMT 18
CMPB (R5)+, (R5)+
NEG R4
MOV R5, (R4)
RTS PC

ICHECK WORD COUNT
IND I/O
IWRITE SPECIFIED
ISFT READ FUNCTION CODE
IMAKE WORD COUNT (-) (RCWC)
ISFT PROPER FUNCTION IN RCCS
IGN-AWAY FOR I/O

RC11 VM1-01 (FIXED HEAD DISK) RT-11 MACRO VM02-09 A-APR-75 12:04:26 PAGE 4
RC-11 INTERRUPT ROUTINE

.SATTTL RC-11 INTERRUPT ROUTINE

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36

000060 000430

G

RR RCABRT

ABORT ROUTINE FOR F/B
IMONITOR

000062 004577 000226

H

RCINTI

JSR R5, @SINPTR

INOTIFY MONITOR & SET
IPRIORITY TO LEVEL 5
IRC RUNS AT PRIORITY 5

000066 000100

I

WORD @C4PRS>&PR7

ICHECK FOR ERROR ON RC & RETRY IF APPLICABLE

000070 010705 177714

MOV RCCQE, R5

IGET CURRENT QUEUE ADR

000074 010704 177406

MOV #RCCS, R4

IPPOINT TO DISK CNTRL & STATUS REG

000100 004714

TST R4

IDATA ADDRESS OR WRITE
ICHECK ERROR OR WRITE-LOCK
INON-EXISTENT DISK?

000102 100023

RPL RCFILL

IND.

000104 032714 060000

RIT #RTYER, R4

IONLY DATA & ADR ERRORS MERIT A RETRY

000110 001493

REQ 18

INOPF - GO BACK WITH ERROR

000112 004367 177700

DEC RETRY

IRTRY FATAL ERROR 8 TIMES

000116 001340

RGT RCRETR

IOK DO IT AGAIN.

000120 052755 000001

181

RIS #HERR, @-(R5)

ICAN'T GET BY HARD ERROR

000124 010704

J

RCHOME

MOV PC, R4

ISFT ERROR FLAG +

000126 060704 177662

ADD #RCCQE-, R4

IGN BACK TO MONITOR

000132 010705 000054

MOV #MONLOW, R5

IPYC ADDR OF CURRENT QUEUE ENTRY

000136 000175 000270

JMP @OFFSET(R5)

IEIT TO MONITOR QUEUE COMPLETION

000142 010737 000400

K

RCABRT

MOV #ABORT, @RCCS

IAORT CURRENT OPERATION

000150 000765

BR RCHOME

IEIT TO MONITOR

000152 004067 000100

RCFILL: JSR R0, RCCOMN

IFILL REMAINDER OF DISK BLOCK WRITTEN TOO WITH ZEROS.

000156 001000

WORD INHCA

ICALL COMMON RTN

000160 014505 000002

MOV @-(R5), R5

ISFT TOO INHIBIT INC OF CURRENT ADR (RCCA)
IGET WORD COUNT FROM

Diagram illustrating the structure of the L register, which is connected to the ADD, RTS, and PC registers.

5A	000226	052705	177400	BIS	#177400,R5	WRITE MUST BE LESS THAN
59						1A BLOCK (RCWC TAKES
60						12'S COMPLEMENT NEG. VALUE.)
61	000232	005724		TST	(R4)+	POINT TO RCCS
62	000234	012724	001103	MOV	#WP+INHCA, (R4)+	SET WRITE FUNCTION
63	000240	010524		MOV	R5, (R4)+	SET WORD COUNT (RCWC)
64	000242	010714		MOV	PC, #R4	POINT MEMORY ADDRESS TO A ZERO.
65	000244	062714	177560	ADD	#ZERO--, #R4	INTC (INTO RCCS)
66	000250	000207		RTS	PC	EXIT

```

1          :SRTTL COMMON SUBROUTINE
2          :RCCOMN
3          :COMMON SUBROUTINE USED BY INTERRUPT
4          :AND ENTRY ROUTINES
5
6 000252 012704 177446      :RCCOM1: MOV      #RCCS,R4      :PT TO DSK CNTRL & STATUS REG
7 000256 031014            :RCCOM1: BIT      @R0,@R4      :PTLL TN PROGRESS
8 000260 001402            :      : BEQ      IS          :NO
9 000262 012600            :      : MOV      (SP)+,R0      :POP R0
10 000264 000717           :      : BR       RCHOME      :PTNTS FILL OF BLK WITH 0'S
11 000266 014705 177516    :IS:  : MOV      RCCOF,R5      :PTR TO CURRENT BUFUF ENTRY
12 000272 012546           :      : MOV      (R5)+, -(SP)    :GET BLOCK NUMBER
13 000274 006316           :      : ASL      @SP          :CALCULATE DISK ADDRESS FOR RCDA
14 000276 006316           :      : ASL      @SP          :UNIT, TRACK# + SECTOR ADDRESS)
15 000300 006316           :      : ASL      @SP          :[72*8=256]
16 000302 052014           :      : AIS      (R0)+,@R4      :INHIB CURR. ADR INC (IF NEEDED)
17 000304 024444           :      : CMP      -(R4),-(R4)    :POINT TO RCDA
18 000306 012614           :      : MOV      (SP)+,@R4      :SET DISK ADR FOR TRANSFER
19 000310 004725           :      : TST      (R5)+          :IGNORE UNIT #
20 000312 000200           :      : RTS       R0          :RETURN TO CALLER
21
22
23
24 000314 000000           :      : N-----M          :MONITOR ENTRY ADDR.
25          :      : SINPTR: :WORD  SINTEN
26          :      : RCsize= :RCSTRT-----O          :SIZE OF HANDLER
27          :      :
28          :      :END

```

RC11 V01-01 (FIXED HEAD DISK) RT-11 MACRO VM02-09 A-APR-75 12104126 PAGE 5+
SYMBOL TABLE

ABORT = 000400	DPSYS = 000000 G	DSSYS = 000000 G	DTSYS = 000000 G	DYSYS = 000000 G
HDERR = 000001	YNHCA = 001000	MONLOW = 000004	OFFSPT = 000270	PC = X000007
PR5 = 000240	PR7 = 000340	RCABRT = 000142R	002 RCHA = 177452	RCCOMN = 000256R 002
RCCOM1 = 000252R	002 RCMGE = 000010R	002 RCHS = 177446	RCHA = 177442	RCHB = 177456
RCEP = 177444	002 RCFILL = 000152R	002 RCHOME = 000124R	002 RCINT = 000062RG	002 RCLA = 177440
RCLQE = 000006R	002 RCMN = 177454	RCRETR = 000020R	002 RCSI7E = 000316 G	RCSTRT = 000000R 002
RCSYS = 000006RG	002 RCTRY = 000010	RCVEC = 0000210	RCWC = 177450	RD = 000105
RETRY = 000016R	002 RFSYS = 000000 G	RKSYS = 000000 G	RTRYER = 060000	RD = X000000
R1 = X000001	R2 = X000002	R3 = X000003	R4 = X000004	R5 = X000005
SP = X000006	WR = 000103	ZERO = 000024R	002 SINPTR = 000314RG	002 SINTEN = ***** G
..V2 = 000001				
.ABS. = 000000	000			
	000000			
SYSHND = 000316	002			
ERRORS DETECTED: 0				
FREE CORP: 15627. WORDS				
RC,LP: /NITTM/C=RC				

BOOT V02R-01 RT-11 BOOTSTRAP RT-11 MACRO VM02-09 A-APR-75 11:49:04
TABLE OF CONTENTS

2-	2	MACROS, GLOBALS
3-	1	ASPECT
7-	29	BOOTSTRAP I/O DRIVER - RC11
10-	1	BOOTSTRAP CORE DETERMINATION
11-	1	READ MONITOR, LOOKUP HANDLERS
12-	1	RELOCATION LIST

BOOT V02R-01 RT-11 BOOTSTRAP RT-11 MACRO VM02-09 A-APR-75 11:49:04 PAGE 1

1	000001	SRCSYS=1
2		! TITLE BOOT V02R-01 RT-11 BOOTSTRAP
3		! RT-11 BOOTSTRAP
4		!
5		! DEC-11-ORBITA-D
6		!
7		! COPYRIGHT (C) 1975
8		!
9		! DIGITAL EQUIPMENT CORPORATION
10		! MAYNARD, MASSACHUSETTS 01754
11		!
12		! THIS SOFTWARE IS FURNISHED UNDER A LICENSE FOR USE ONLY
13		! ON A SINGLE COMPUTER SYSTEM AND MAY BE COPIED ONLY WITH
14		! THE INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE,
15		! OR ANY OTHER COPIES THEREOF, MAY NOT BE PROVIDED OR OTHERWISE MADE
16		! AVAILABLE TO ANY OTHER PERSON EXCEPT FOR USE ON SUCH SYSTEM AND TO
17		! ONE WHO AGREES TO THESE LICENSE TERMS. TITLE TO AND OWNERSHIP OF THE
18		! SOFTWARE SHALL AT ALL TIMES REMAIN IN DIGITAL.
19		!
20		! THE INFORMATION IN THIS DOCUMENT IS SUBJECT TO
21		! CHANGE WITHOUT NOTICE AND SHOULD NOT BE CONSTRUED
22		! AS A COMMITMENT BY DIGITAL EQUIPMENT CORPORATION.
23		!
24		! DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE
25		! OR RELIABILITY OF ITS SOFTWARE ON EQUIPMENT
26		! WHICH IS NOT SUPPLIED BY DIGITAL.
27		

```

1
2
3
4
5 000000
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31 000000
32 000000
33 000044
34 177570
35
36
37
38 000000
39 000001
40 000002
41 000003
42 000004
43 000005
44 000006
45 000007
46
47
48

;SMTTL MACROS, GLOBALS

;MCALL '.V1'.
;MCALL '.EXIT, '.LOOKUP, '.PRINT, '.SAVESTATUS

;*****
; CONDITIONAL ASSEMBLY OF ROOT FOR SINGLE USER OR RF SYSTEM
; IF NDF BF BF=0 ;DEFAULT TO SINGLE USER

; GLOBAL REFERENCES TO MONITOR
;GLOBL SDVREC, SENTRY, SINTPR, SKMLC, SMONBL, SPNAME, SSLOT
;GLOBL SSWPRL, SUSRLC, SPNAME
;GLOBL RSTRNG, CORPTR, DKASSG, FILLER, HWPPUS, HWBSPS, KMLOC
;GLOBL KMON, KMONSZ, KWILLS, MAPOFF, GCOMP, RT11SZ
;GLOBL RTLEN, RTSIZE, SWAPSZ, SYENTO, SYINDO, SYNCH, SYASSG
;GLOBL SYSLOW, TTIBUF, TTIBUF, USRLOC, USRSZ, MAXSYH

;GLOBL RELLST

; FOLLOWING ARE GLOBALS FOR EITHER BF OR SU SYSTEM, BUT NOT BOTH

; IF NE BF
;GLOBL MCNTXT, RKND1, RKND2, RKND3, CNTXT, FUDGE1, FUDGE2
;GLOBL MSGENT, RMONSP, SWPTR, SWPTR, TTUSR, TTUSR, .SCRN
; IF
;GLOBL AVAIL, I.CSW, PPRADD, PPRIGN, MONLOC, TRAPLC, TRAPER
;ENDC

PERM = 2000 ;STATUS WORD FOR PERMANENT FILE
ENDBLK = 4000 ;STATUS OF END OF SEGMENT MARK
JSW = 44 ;ADDRESS OF JOB STATUS
SR = 177570 ;CONSOLE SWITCH REGISTER

; REGISTER DEFINITIONS
R0=X0
R1=X1
R2=X2
R3=X3
R4=X4
R5=X5
SP=X6
PC=X7

; MONITOR OFFSET CONSTANTS

```

49	000300	CONFIG	= 300	HARDWARE CONFIGURATION WORD
50	000274	SYUNIT	= 274	SYSTEM UNIT #
51				
52	177546	LKCS	= 177546	CLOCK STATUS REGISTER
53	172000	GT40	= 172000	GT40 LOCATION
54	177560	TKS	= 177560	KEYBOARD STATUS
55	177562	TKR	= 177562	" BUFFER
56	177564	TPS	= 177564	PRINTER STATUS
57	177566	TPR	= 177566	" BUFFER

ROOT VM02B-01 RT-11 BOOTSTRAP RT-11 MACRO VM02-09 A-APR-75 11:49:04 PAGE 3
ASECT

1		.SMTTL	ASECT	
2		.IF NDF SRFSYS		TURN ON SRFSYS IF ALL OTHERS ARE OFF
3		.IF NDF SDTSYS		
4		.IF NDF SDPSYS		
5		.IF NDF SDSSYS		
6		.IF NDF SRCSYS		
7		.IF NDF SDXSYS		
8		SRFSYS = 0		IT MUST BE AN RK SYSTEM
9		.ENDC		
10		.ENDC		
11		.ENDC		
12		.ENDC		
13		.ENDC		
14		.ENDC		
15		.ENDC		
16		.ASECT		
17	000000	. = 0		
18	000000	240		BOOT VALIDATION PATTERN
19	000000	000240		BRANCH TO REAL ROOT
20	000002	000414	RR	ROOT1
21		.IF NDF SDXSYS		
22		. = 34		PUT THE JUMP BOOT IN TRAP VECTOR
23	000034	000034	ROOT11	JMP
24	000034	000167	000460	ROOT
25		.IFF		
26		CSBO = 1		START FUNCTION
27		CSEBUF = 2		EMPTY BUFFER
28		CSRD = 6		READ SECTOR
29		CSUNIT = 20		UNIT 1 SELECTION
30		CSDONE = 40		RY DONE
31		CSTR = 200		RYDS TRANSFER READY
32		CSERR = 100000		RY ERROR
33				
34				
35		RXCS = 177170		RXCS STATUS REGISTER

```

36
37
38      . = 14      READS      INITIALIZE RPT AND IOT VECTORS
39      . WORD      0          ION RPT INTERRUPT TO READS ROUTINE
40      . WORD      WAIT      ION IOT INTERRUPT TO WAIT ROUTINE
41      . BYTE      CSGO+CSRD  IREAD FROM UNIT 0, SETS WEIRD BUT OK PS
42      . BYTE      CSGO+CSRD+CSUNTT IREAD FROM UNIT 1
43      . = 34      I34-52 USEABLE
44      ROOT1: MOV      UNTRD(R0),RDCMD ISET READ FUNCTION FOR CORRECT UNIT
45      RETRY: MOV      @PC,SP      IINIT SP WITH NEXT INSTRUCTION
46      . MOV      #200,R2      IAREA TO READ IN NEXT PART OF ROOT
47      . CLR      R0          ISET TRACK NUMBER
48      . RR      PS          IOUT OF ROOM HERE, GO TO CONTINUATION
49      . = 70      IPAPER TAPE VECTORS
50      2S: MOV      SP,R1      ISET TO BIG WORD COUNT
51      . INC      R0          ISET TO ABSOLUTE TRACK 1
52      . RR      PS          IBRANCH TO CONTINUATION
53      . = 104     IPROGRAMMABLE CLOCK
54      3S: MOV      @PC,R3      IABSOLUTE SECTOR 3 FOR NEXT PART
55      . RPT      ROOT2      ICALL READS SUBROUTINE
56      . RR      ROOT2      IBRANCH TO CONTINUATION
57      . = 120     ILOTS OF UNUSED VECTORS, (DR-110?)

```

BOOT V02R-01 RT-11 BOOTSTRAP RT-11 MACRO VM02-00 A-APR-75 11:49:04 PAGE 3+
 A5ECT

```

58      READS: MOV      #RXCS,R4      IR4 => RX STATUS REGISTER
59      . MOV      R4,R5      IRS WILL POINT TO RX DATA BUFFER
60      . MOV      (PC)+,(R5)+  IINITIATE READ FUNCTION
61      . WORD      0          IGETS FILLED WITH READ COMMAND
62      RDCMD: TOT      ICALL WAIT SUBROUTINE
63      . MOV      R3,@R5      ILOAD SECTOR NUMBER INTO RXDR
64      . TOT      ICALL WAIT SUBROUTINE
65      . MOV      R0,@R5      ILOAD TRACK NUMBER INTO RXDB
66      . TOT      ICALL WAIT SUBROUTINE
67      . MOV      #CSGO+CSEBHF,@R4 ILOAD EMPTY BUFFER FUNCTION INTO RXCB
68      4S: TOT      ICALL WAIT SUBROUTINE
69      . TSTB      @R4          IIS TRANSFER READY UP?
70      . RPL      RTRET      IBRANCH IF NOT, SECTOR MUST BE LOADED
71      . MOVB      @R5,(R2)+  IMOVE DATA BYTE TO MEMORY
72      . DEC      R1          ICHECK BYTE COUNT
73      . RGT      4S          ILOOP AS LONG AS WORD COUNT NOT UP
74      . CLR      R2          IKLUDGE TO SLUFF BUFFER IF SHORT WD CNT
75      . RR      4S          ILOOP
76      WAIT: TST      @R4      IIS TR, ERR, DONE UP? INT ENR CAN'T BE
77      . REQ      WAIT      ILOOP TILL SOMETHING
78      . BMT      RETRY      ISTART AGAIN IF ERROR
79      RTTRET: RTI
80

```

```

81      . = 200
82      ROOT2:  CMPB  (R3)+,(R3)+      ;SECTOR 2 OF RX BOOT
83              RPT   (R3)+,(R3)+      ;BUMP TO SECTOR 5
84              CMPB  (R3)+,(R3)+      ;CALL READS SUBROUTINE
85              RPT   (R3)+,(R3)+      ;BUMP TO SECTOR 7
86              RPT   (R3)+,(R3)+      ;CALL READS SUBROUTINE
87              RIT   #CSUNIT,RDCMD    ;CHECK UNIT ID
88              RNE   IS               ;BRANCH IF BOOTING UNIT 1, R0=1
89              CLR   R0               ;SET TO UNIT 0
90              MOV   R0,(PC)+          ;SAVE UNIT BOOTED FROM FOR LATER
91              .WORD 0                 ;SAVE THE UNIT HERE
92              MOV   #TRWAIT,0#20     ;LETS HANDLE ERRORS DIFFERENTLY
93              JMP   ROOT              ;NOW WE ARE READY TO DO THE REAL BOOT
94
          .ENDC

```

ROOT VM02B-01 RT-11 BOOTSTRAP RT-11 MACRO VM02-09 8-APR-75 11:49:04 PAGE 4
ASECT

```

1      ; FOLLOWING ARE THE BOOTSTRAP I/O DRIVERS FOR EACH VALID
2      ; SYSTEM DEVICE.
3      ; CALLING SEQUENCE:
4      ;   R0 = PHYSICAL BLOCK TO READ/WRITE
5      ;   R1 = WORD COUNT
6      ;   R2 = BUFFER ADDRESS
7      ;   R3,R4,R5 ARE AVAILABLE AND MAY BE DESTROYED BY THE DRIVER
8      ; THE DRIVER MUST GO TO RIGERR IF A FATAL I/O ERROR OCCURS.
9      ; IT MUST ALSO INVOKE THE MACRO SYDEV
10
11      MACRO SYDEV NAME,VECTOR
12      .GLOBAL NAME'INT, NAME'SIZE      ;DEFINE SYSTEM DEVICE INTERRUPT & SIZE
13      SYNAME = 0
14      .IRPC X,<NAME>
15      SYNAME = <SYNAME+'X-100'>+50
16      .ENDR
17      SYVEC = VECTOR                  ;IT VECTORS TO THIS LOCATION
18      . = SYVEC                      ;AT THE VECTORS
19      .WORD NAME'INT,340              ; PUT A VECTOR TO THE SYSTEM HANDLER
20      . = SYSIZE
21      .WORD NAME'SIZE
22      . = 402                        ;PUT HANDLER SIZE WHERE IT CAN BE USED
23      SYRITO = VECTOR / 20           ;AND START THE CODE AT 402
24      SYRITS = %B11000000           ;OFFSET INTO BIT MAP FOR PROTECTION
25      .RPT <VECTOR & 17> / 4        ;COMPUTE ACTUAL BITS
26      SYRITS = SYRITS / 4            ;VECTOR IS A MULTIPLE OF 4
27      .ENDR                          ;SHIFT RIGHT 2 MORE BITS
28      .ENDM SYDEV

```

ROOT VM02-01
ASECT

RT-11 ROOTSTRAP RT-11 MACRO VM02-09 8-APR-75 11:49:04 PAGE 5

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35

```

;IF DF $DSSYS IRS SYSTEM
.SMTTL ROOTSTRAP I/O DRIVER = RS11

; RS11 DISK HANDLER

;IF DF MBUSSC
    SYSDEV DS,150
RSCS2 = 176310
.ENDC
;IF NDF MBUSSC
    SYSDEV DS,204
RSCS2=172050
.ENDC

READ:  MOV     R0,R4                ;COPY BLOCK NUMBER
        MOV     #RSCS2,R5          ;POINT TO REGISTERS
        MOV     (R5),-(SP)         ;SAVE UNIT #
        MOV     #40,R5            ;CONTROLLER CLEAR
        RIT     #2,16(R5)         ;WHAT IS IT?
        RNE     15                ;IT'S AN RS04
        ASL     R4                ;IT'S AN RS03
13:     ASL     R4                ;CONVERT TO TRACK/SECTOR
        RLC     #C7,(SP)          ;STRIP TO UNIT BITS
        MOV     (SP)+,(R5)        ;SET UNIT
        MOV     R4,=(R5)          ;SET BLOCK
        MOV     R2,=(R5)
        MOV     R1,=(R5)
        NEG     R5
        MOV     #71,=(R5)         ;GO, READ, NO INTERRUPT
23:     RIT     #100200,R5        ;WAIT FOR DONE OR ERROR
        BEQ     28
        RMT     RIDERR            ;BOOT ERROR
        RTS     PC

.ENDC

```

```

1      ,IF DF SDPSYS                ;CONDITIONAL FOR RP11 DISK
2      ,SATTL ROOTSTRAP I/O DRIVER = RP11
3
4      ; RP11 DISK DRIVER
5
6      SYSDEFV DP,254                ;DEVICE IS RP. IT VECTORS TO 254
7      RPCS= 176714                  ;RP11 DEVICE CONTROL REG
8      RPNB= 176710                  ;RP03 DEVICE STATUS REG
9      RPDN= 176724                  ;RP03 DISK ADRS REGISTER
10     CS,GO= 000001                  ;GO BIT IN CONTROL & STATUS
11     CS,RD= 000004                  ;READ FUNCTION CODE
12     CS,DRV= 003400                 ;UNIT SELECT BITS
13     DS,ATT= 000377                 ;UNIT ATTN BITS
14
15     READ: MOV R0,R3                ;R3 = BLOCK #
16           JSR R2,DIV                ;GET SECTOR NUMBER
17           ,WORD 10.                 ;BY DIVIDING BY 10
18           MOV R4,=(SP)              ;SAVE SECTOR
19           MOV R5,R3                 ;SET NEW DIVIDEND
20           JSR R2,DIV                ;AND COMPUTE CYL & TRACK
21           ,WORD 20.                 ;BY DIVIDING BY 20
22           SWAB R4                   ;POSITION TRACK IN HIGH BYTE
23           RIS (SP)+,R4              ;AND INSTALL SECTOR
24           MOV #RPDA,R3             ;R3 -> DISK ADRS REG
25           MOV R4,R3                 ;SET TRACK & SECTOR
26           MOV R5,=(R3)              ;AND CYLINDER
27           MOV R2,=(R3)              ;AND BUS ADDRESS
28           MOV R1,=(R3)              ;AND WORD COUNT
29           NEG R3                     ;MAKE NEGATIVE
30           RLC #C<CS,DRV>,=(R3)      ;CLEAR ALL BUT UNIT #
31           RIS #CS,RD+CS,GO,R3 ; AND START READ
32     ;: TSTB R3                      ;WAIT UNTIL TRANSFER COMPLETE
33           RPL 15
34           TST R3                     ;ANY ERRORS?
35           RMT BIOERR                 ;YES
36           MOVB #DS,ATT,#RPNB        ;CLEAR UNIT ATTN FOR BOTH
37           CLRB #RPNB                 ;OLD & ECO'D CONTROLLERS
38           RTS PC                     ;ELSE JUST RETURN
39
40     ; DIVIDE ROUTINE FOR RP HANDLER.
41     ; R5 = R3 / R2, REMAINDER IN R4
42
43     DIV: CLR R5                      ;QUOT. = 0
44           CLR R4                      ;REM. = 0
45           TST R3                      ;IS DIVIDEND 0?
46           BEQ 48                      ;YES = JUST RETURN
47           COM R5                      ;QUOT. = -1 & SET CARRY
48     ;: ROL R3                        ;NORMALIZE

```

49		RCC	18	
50	PSI	ROL	R4	IShift & SUBTRACT
51		CMF	R4, R2	
52		RLO	TS	
53		SUR	R2, R4	
54	SSI	ROL	R5	
55		ASL	R3	
56		RNE	TS	
57		COM	R5	IFIX QUOTIENT

BOOT VM2R-01 RT-11 BOOTSTRAP RT-11 MACRO VM02-09 A-APR-75 11149104 PAGE 6+

58	4SI	TST	(R2)+
59		RTS	R2
60			
61		ENDC	

BOOT VM2R-01 RT-11 BOOTSTRAP RT-11 MACRO VM02-09 A-APR-75 11149104 PAGE 7

1		.IF DF SRKSYS:SRFSYS:SRCSYS	BB
2		.IFDF SRFSYS	CONDITIONAL FOR RF DISK
3		.SATTL BOOTSTRAP I/O DRIVER = RF11	
4			
5			
6			
7			
8			
9			
10			
11			
12			
13			
14			
15			
16			
17			
18			
19			
20			
21			
22			
23			
24			
25			

```

26      .ENDC
27      .IFDF SRCSYS          ;CONDITIONAL FOR RC (RS64) DISK
28
29      .SMTTL BOOTSTRAP I/O DRIVER = RC11
30
31      ; RC11 DISK HANDLER
32      DD      SYSDEV RC,210    ;DEVICE IS RC. IT VECTORS TO 210
33
34      ; RC11 CONTROL REGISTERS
35
36      177440      RCLA= 177440      ;LOOK AHEAD REGISTER
37      177442      RCDA= 177442      ;DISK ADDRESS REGISTER
38      177444      RCER= 177444      ;DISK ERROR STATUS REGISTER
39      177446      RCCS= 177446      ;DISK CONTROL & STATUS REGISTER
40      177450      RCWC= 177450      ;WORD COUNT REGISTER
41      177452      RCCA= 177452      ;CURRENT ADDRESS REGISTER
42      177454      RCMN= 177454      ;MAINTENANCE REGISTER
43      177456      RCDB= 177456      ;DATA BUFFER REGISTER
44
45      000402 012703 177442      EE      READ1      MOV      #RCDA,R3      ;PT TO DISK ADR REGISTER
46      000406 010005              MOV      R0,R5      ;GET BLOCK NUMBER
47      000410 006305              ASL      R5      ;CALCULATE DISK ADR FOR RCDA
48      000412 006305              ASL      R5      ; (UNIT, TRACK # & SECTOR ADR)
49      000414 006305              ASL      R5      ; [32 * 8 = 256]
50      000416 010513              MOV      R5,R3      ;IND PROPER DISK ADR
51      000420 062703 000012      ADD      #12,R3      ;PT TO CURRENT ADR REG + 12
52                                   ; (INTERFACE TO COMMON CODE)
53
54      .ENDC
55      .IFDF SRKSYS
56      .SMTTL BOOTSTRAP I/O DRIVER = RK05
57      ; RK05 DISK HANDLER

```

BOOT V02B-01 RT-11 BOOTSTRAP RT-11 MACRO VM02-09 8-APR-75 11149104 PAGE 7+

BOOTSTRAP I/O DRIVER = RC11

```

58
59      RKDA      SYSDEV RK,220      ;DEVICE IS RK. IT VECTORS TO 220
60      = 177412      ;RK DISK ADDRESS
61
62      READ1      MOV      #14,R3      ;PHYSICAL BLOCK TO RK DISK ADR.
63      RR      PS      ;ENTER BLOCK # COMPUTATION
64      181      ADD      #20,R3      ;CONVERT DISK ADDRESS
65      231      SUB      #14,R0
66      RPL      18
67      ADD      R3,R0      ;R0 HAS DISK ADDRESS
68      581      MOV      #RKDA,R3      ;POINT TO HARDWARE DISK ADDR REGISTER
69      RIC      #17777,R3      ;LEAVE THE UNIT NUMBER
70      BIS      R0,(R3)      ;PUT DISK ADDRESS INTO CONTROLLER

```

```

71
72
73
74 000424 010243
75 000426 010143
76 000430 005413
77 000432 012743 000005
78 000436 105713
79 000440 100376
80 000442 005713
81 000444 100401
82 000446 000207
83
84

```

```

      .ENDC
      ; THIS CODE IS COMMON TO RK05,RC11 AND RF11 HANDLERS
      GG → MOV R2,=(R3) ;BUFFER ADD.
          MOV R1,=(R3) ;WORD COUNT
          NEG (R3) ;(NEGATIVE)
          MOV #5,=(R3) ;START DISK READ
          SSI TSTB (R3) ;WAIT UNTIL COMPLETE
          RPL SS
          TST (R3) ;ANY ERRORS?
          BMT R10ERR ;HARD HALT ON ERROR
          RTS PC
      .ENDC

```

ROOT VM2B-01 RT-11 BOOTSTRAP RT-11 MACRO VM02-09 8-APR-75 11:49:04 PAGE 8
 BOOTSTRAP I/O DRIVER = RC11

```

1      .IF DF SDTSYS
2
3      .SMITL BOOTSTRAP I/O DRIVER = DFCTAPE
4
5      ; DECTAPE BOOTSTRAP HANDLER
6      SYSDEV DT,214 ;DEVICE IS DT. IT VECTORS TO 214.
7      TCCM = 177342 ;COMMAND REGISTER
8      TCOT = 177350 ;DATA REGISTER
9      TCST = 177340 ;STATUS REGISTER
10
11     READ: MOV #TCCM,R4 ;R4 => COMMAND REG
12           MOV #TCOT,R3 ;R3 => DATA REG
13     DTSRCH: MOV R0,R5 ;COPY BLOCK NUMBER
14            SUB #2,R5 ;SEARCH FOR 2 EARLIER
15            MOV #4003,R4 ;REVERSE,RNUM
16     ZSI: BIT #100200,R4 ;WAIT TILL BLOCK FOUND
17          BEQ ZS
18          BMT DTERR
19          CMP R5,R3 ;IS IT THE DESIRED BLOCK
20          BLT DTSRCH ;NO,CONTINUE SEARCHING
21     DTFWRD: MOV #3,R4 ;SEARCH FORWARD (RNUM)
22     4SI: BIT #100200,R4 ;WAIT
23           BEQ 4S
24           BMT DTERR
25           CMP R0,R3 ;DESIRED BLOCK
26           RGT DTFWRD ;NO-SEARCH FORWARD
27           BLT DTSRCH ;NO-SEARCH REVERSE
28           MOV R2,=(R3) ;BUFFER ADDRESS
29           NEG R1
30           MOV R1,=(R3) ;WORD COUNT

```

31		MOV	#5, R4	I READ
32	DT4:	RIT	#100200, R4	I WAIT FOR COMPLETION
33		REQ	DT4	
34		RMT	RIDERR	I READ ERROR
35		CLR	R4	I STOP DT
36		RTS	PC	
37	DTERR:	TST	#TCSY	I WHAT KIND OF ERROR ?
38		RPL	RIDERR	I NOT END ZONE
39		RIT	#4000, R4	I REVERSE?
40		RNE	DTFWRD	I THEN GO SEARCH FORWARD
41		RR	DTSRCH	I ELSE SEARCH REVERSE
42				
43				

.ENDC

BOOT VM02B-01 RT-11 BOOTSTRAP RT-11 MACRO VM02-09 8-APR-75 11:49:04 PAGE 9
 BOOTSTRAP I/O DRIVER = RC11

1			IF OF SDXSYS	IF FLOPPY SYSTEM
2			.SRTTL	BOOTSTRAP I/O DRIVER = FLOPPY
3				
4			SYSDEV	DX, 264
5				IF FLOPPY VECTORS THROUGH 264
6	READ:	ASL	R0	I CONVERT BLOCK TO LOGICAL SECTOR
7		ASL	R0	LSN=BLOCK*4
8		ASL	R1	I MAKE WORD COUNT BYTE COUNT
9	1S:	MOV	R0, -(SP)	I SAVE LSN FOR LATER
10		MOV	R0, R3	I WE NEED 2 COPIES OF LSN FOR MAPPER
11		MOV	R0, R4	
12		CLR	R0	I INIT FOR TRACK QUOTIENT
13		RR	38	I JUMP INTO DIVIDE LOOP
14	2S:	SUB	#23, R3	I PERFORM MAGIC TRACK DISPLACEMENT
15	3S:	INC	R0	I BUMP QUOTIENT, STARTS AT TRACK 1
16		SUB	#26, R4	I TRACK=INTEGER(LSN/26)
17		RPL	28	I LOOP = R4=REM(LSN/26)-26
18		CMF	#-14, R4	I SET C IF SECTOR MAPS TO 1-13
19		ROL	R3	I PERFORM 2:1 INTERLEAVE
20	4S:	SUB	#26, R3	I ADJUST SECTOR INTO RANGE -1,-26
21		RPL	48	I (DIVIDE FOR REMAINDER ONLY)
22		ADD	#27, R3	I NOW PUT SECTOR INTO RANGE 1-26
23		RPT		I CALL READS SUBROUTINE
24		MOV	(SP)+, R0	I GET THE LSN AGAIN
25		INC	R0	I SET UP FOR NEXT LSN
26		TST	R1	I WHATS LEFT IN THE WORD COUNT
27		RGJ	18	I BRANCH TO TRANSFER ANOTHER SECTOR
28		RTS	PC	I RETURN
29	TRWAIT:	TST	R4	I NEW WAIT SUBROUTINE, PRINTS ERRORS
30		REQ	TRWAIT	
31		RPL	RTIRET	I RETURN FROM INTERRUPT

```

32
33          ***** THIS MUST FALL INTO BIOERR *****
34          .ENDC
35
36
37
38 000450 004067 000024      BIOERR: JSR      R0,REPORT      ;SAY THAT WE GOT ERROR
39 000454      015      012      077      .ASCIZ  <15><12>\TA-I/O ERROR\<12>
      000457      102      055      111
      000462      057      117      040
      000465      105      172      122
      000470      117      122      012
      000473      000
40          .EVEN

```

BOOT V02B-01 RT-11 BOOTSTRAP RT-11 MACRO VM02-09 A-APR-75 11:49:04 PAGE 10
 BOOTSTRAP CORE DETERMINATION

```

1          .SMTTL  BOOTSTRAP CORE DETERMINATION
2
3 000474 112037 177566      REPORT: MOVB      (R0)+,#TPR      ;PUT ANOTHER CHARACTER OUT
4 000500 105737 177564      REPORT: TSTB      #TPS      ;WAIT FOR TYPER READY
5 000504 100375              RPL      REPORT      ; ...
6 000506 105710              TSTB      #R0      ;ANYTHING MORE ?
7 000510 001371              RNE      REPORT      ;YES, LOOP
8 000512 000000              RESET              ;STOP ALL DEVICES
9 000514 000000              HALT
10 000516 000776              RR      ;=2      ;KEEP HIM FROM CONTINUING
11
12 000520 012706 010000      BOOT:  MOV      #10000,SP      ;SET STACK POINTER
13 000524 012700 000002      MOV      #2,R0      ;READ IN SECOND PART OF BOOT
14 000530 012701 000400      MOV      #<ROOTSZ-1>+400,R1 ;EVERY BLOCK BUT THE ONE WE ARE IN
15 000534 012702 001000      MOV      #1000,R2      ;INTO LOCATION 1000
16 000540 004767 177636      JSR      PC,READ
17          .IF GT  ;=1000, .ERROR      ;BOOTSTRAP BLOCK 0 TOO BIG
18 000544 012703 000004      MOV      #4,R3      ;POINT TO TRAP LOCATIONS
19 000550 011305              MOV      #R3,R5      ;SAVE TRAP LOC
20 000552 012723 000620      MOV      #NXM,(R3)+      ;SET TRAP FOR NON EXISTENT MEMORY
21 000556 000013              CLR      #R3
22
23          ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
24          ; THIS BOOTSTRAP CAN SIMULATE ANY SIZE PDP-11.
25          ; IF LOCATION 'FIDDLE' IS A HALT, THE CPU WILL STOP DURING THE BOOT.
26          ; ON CONTINUE, THE TOP 5 BYTS OF THE SWITCH REGISTER ARE USED TO
27          ; SET THE TOP OF AVAILABLE CORE AS A MULTIPLE OF 1K.
28          ; IF THE SR IS >= 160000 OR IF FIDDLE IS A BR 18,
29          ; THE BOOTSTRAP WILL DO A NORMAL CORE DETERMINATION.
30          ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
31 000560 000407      FIDDLE: RR      18      ;CHANGE TO HALT FOR FIDDLING

```

32	000562	013702	177570		MOV	#SR,R2	IGET SWITCH VALUE
33	000566	042702	003777		RIC	#3777,R2	ISOLATE TOP 5 BITS (1K INCREMENTS)
34	000572	020227	160000		CMP	R2,#160000	ISHOULD WE DO NORMAL CHECK ?
35	000576	103410			BLO	NXM	INO, USE THE SR VALUE
36	000600	004002		13:	CLR	R2	LOOK FOR TOP OF CORE
37	000602	062702	004000	28:	ADD	#4000,R2	IMOVE TO NEXT 1K BANK
38	000606	020227	160000		CMP	R2,#160000	IREACHED 28K YET ?
39	000612	001402			REQ	NXM	YES, DO A 28K SYSTEM
40	000614	004712			TST	#R2	INO, SEE IF THIS LOCATION EXISTS
41	000616	000771			BR	28	KEEP GOING IF WE DIDN'T TRAP
42	000620	012743	001476	NXM:	MOV	#BCLR,-(R3)	NONMEMORY TRAPS HERE
43	000624	011367	177160		MOV	#R3,10	IBAD INSTRUCTIONS TRAP HERE
44	000630	012701	001604		MOV	#TSLIST,R1	IBITS FOR CLEARING ON ERROR TRAPS
45	000634	010100			MOV	R1,R0	
46	000636	004737	177546		TST	#LKCS	ICHECK PRESENCE OF CLOCK
47	000642	052140			BIS	(R1)+,-(R0)	IADVANCE LIST
48	000644	003737	172000		TST	#GT40	ICHECK FOR DISPLAY
49	000650	052110			BIS	(R1)+,R0	
50	000652	170000			CFCC		ICHECK FOR FPU
51	000654	052110			BIS	(R1)+,R0	
52	000656	010523			MOV	R5,(R3)+	IRESTORE TRAP
53	000660	012723	000340		MOV	#340,(R3)+	ITRAP TO 4 IS PR7, CARRY OFF
54	000664	010523			MOV	R5,(R3)+	IRESTORE 10
55	000666	012723	000341		MOV	#341,(R3)+	ITRAP TO 10 IS PR7, CARRY ON
56	000672	162702	0000000		SUR	#RTSIZE,R2	IR2 NOW POINTS TO WHERE WE WANT THE KMON
57	000676	062702	0000000		ADD	#FILLER,R2	IBUT IT AGAINST THE TOP OF CORE

ROOT V025-01 RT-11 ROOTSTRAP RT-11 MACRO VM02-00
 ROOTSTRAP CORE DETERMINATION

A-APR-75 11149104 PAGE 10+

58	000702	162702			SUR	(PC)+,R2	IRECOVER UNUSED CORE FROM SY:
59		000704					
60		000706					
61	000706	062702	0000000		ADD	#MAXSYH,R2	ISYSTEM HANDLER SIZE PUT HERE
62	000712	020227	010000		CMP	R2,#10000	THIS WAY BECAUSE NO GLOBAL ARITH.
63	000716	103466			BLO	TOOSML	IS IT JUST TOO TINY ?
64	000720	010246			MOV	R2,-(SP)	YES
65	000722	012700	000001		MOV	#1,R0	IPUT LOAD ADDRESS ON STACK
66	000726	006300		OFND:	ASL	R0	INOW READ FIRST DIRECTORY BLOCK
67	000730	062700	000004		ADD	#4,R0	
68	000734	012701	001000		MOV	#1000,R1	IDIRECTORY STARTS AT 6
69	000740	012702	001634		MOV	#BUFFER,R2	
70	000744	004767	177432		JSR	PC,READ	IREAD THE SEGMENT
71	000750	012701	001644		MOV	#BUFFER+10,R1	IPPOINT TO START BLOCK WORD
72	000754	012100			MOV	(R1)+,R0	
73	000756	010102		MONF:	MOV	R1,R2	ISAVE ADDRESS OF STATUS WORD

74	000760	032721	002000		RIT	#PERM,(R1)+		IS IT A PERMANENT FILE?
75	000764	001411			REG	IS		IND. WE ARE TRYING TO FIND THE
76	000766	162721			SUB	(PC)+,(R1)+		FILE MONITR.SYS, AS THAT IS
77	000770	051646			,RAD50	/MON/		THE CURRENT MONITOR.
78	000772	162721			SUB	(PC)+,(R1)+		
79	000774	035562			,RAD50	/ITR/		
80	000776	162711			SUB	(PC)+,(R1)		
81	001000	075273			,RAD50	/SYS/		
82	001002	001002			RNE	IS		LAST WAS NOT .SYS EXTENSION
83	001004	054141			RIS	=(R1),-(R1)		BOTH MUST BE 0
84	001006	001447			REG	MONPND		FOUND THE MONITOR
85	001010	032712	004000	131	RIT	#ENDBLK,(R2)		IS THIS ALL IN SEGMENT?
86	001014	001010			RNE	25		YES. READ NEXT, IF ANY.
87	001016	066200	000010		ADD	10(R2),R0		INCREASE START BLOCK
88	001022	062702	000016		ADD	#16,R2		GET TO NEXT ENTRY
89	001026	066702	000610		ADD	RUFFB+6,R2		
90	001032	010201			MOV	R2,R1		POINT R1 TO NEXT
91	001034	000750			RR	MONP		
92	001036	016700	000574	281	MOV	RUFFB+2,R0		SEE IF NEXT IS AVAILABLE
93	001042	001331			RNF	0FND		YES. CONTINUE
94	001044	004067	177430		JSR	R0,REPORT		HE AIN'T GOT A MONITOR
95	001050	015	012	077	.ASCIZ	<15><12>\?R=NO MONITR.SYS\<12>		
	001053	102	055	116				
	001056	117	040	115				
	001061	117	116	111				
	001064	124	122	056				
	001067	123	131	123				
	001072	012	000					
96					.EVEN			
97	001074	004067	177400	TOOSML1	JSR	R0,REPORT		HE IS IN A TINY MACHINE
98	001100	015	012	077	.ASCIZ	<15><12>\?R=NOT ENOUGH CORE\<12>		
	001103	102	055	116				
	001106	117	124	040				
	001111	105	116	117				
	001114	125	107	110				
	001117	040	103	117				
	001122	122	105	012				
	001125	000						
99					.EVEN			

1			1	:SRTTL	READ MONITOR, LOOKUP HANDLERS
2	001126	011602	MONFND:	MOV	#SP,R2 ;RECALL LOAD LOCATION
3	001130	062700		ADD	#BOOTSZ,R0 ;BUMP R0 OVER BOOT RECORDS
4	001134	010046		MOV	R0,-(SP) ;SAVE SWAP BLOCK POINTER
5	001136	012701	000000G	MOV	#MAXSYH,R1 ;DO GLOBAL ARITHMETIC HERE
6	001142	166701	177536	SUB	SYSIZE,R1 ;R1 = MAXSYH-SYSIZE (BYTES)
7	001146	062701	000000G	ADD	#FILLER,R1 ;ADD AMOUNT OF EXTRA STUFF
8	001152	006201		ASR	R1 ;(WORDS)
9	001154	004401		NEG	R1 ;(TO SUBTRACT)
10	001156	062701	000000G	ADD	#RTLEN,R1 ;LENGTH TO LOAD (WORDS)
11	001162	062700	000000G	ADD	#SWAPSZ,R0 ;POINT TO BLOCK WITH KMON
12	001166	004767	177210	JSR	PC,READ ;READ THE MONITOR INTO PLACE
13	001172	012700	001502	MOV	#RELLST,R0 ;POINT TO LIST OF THINGS TO RELOCATE
14	001176	012601		MOV	(SP)+,R1 ;R1 = SWAP BLOCK NUMBER
15	001200	012604		MOV	(SP)+,R4 ;R4 => KMON IN CORE
16	001202	162704	000000G	SUB	#KMON,R4 ;SUBTRACT LOCATION KMON WAS LINKED TO
17	001206	010164	000000G	MOV	R1,\$SWPBL(R4) ;R4 = BIAS. SET UP SWAP BLOCK #
18	001212	062701	000000G	ADD	#SWAPSZ,R1
19	001216	062701	000000G	ADD	#KMONSZ,R1
20	001222	010164	000000G	MOV	R1,\$MONBL(R4) ;SET UP BLOCK #
21	001226	060430		ADD	R4,#(R0)+ ;RELOCATE A POINTER IN THE ASPECT
22	001230	020027	001524	CMP	R0,#RELLST2 ;DONE YET ?
23	001234	101774		BLO	1\$;NO
24	001236	012005		MOV	(R0)+,R5 ;GET POINTER TO THING IN MONITOR
25	001240	060405		ADD	R4,R5 ;BIAS THE POINTER
26	001242	060415		ADD	R4,R5 ;NOW RELOCATE THE WORD
27	001244	012005		MOV	(R0)+,R5 ;GET NEXT POINTER
28	001246	001374		RNE	2\$
29	001250	011700	000054	MOV	#54,R0 ;POINT TO MONITOR
30					
31				:IF DF	SRKSYSISDXXSYS,\$DPSYSISDSSYS ;THE RK,RX,RP,RJS03/4 CAN BOOT FROM ANY UNIT
32					
33				:IF DF	SRKSYS ;CODE FOR RK
34				MOV	#RKDA,R1 ;GET THE RK UNIT NUMBER
35				ROL	R1
36				ROL	R1
37				ROL	R1
38				ROL	R1
39				RIC	#C7,R1 ;EXTRACT IT
40				:ENDC	;
41				:IF DF	SDSSYS ;CODE FOR RJS03/4
42				MOV	#RSC82,R1 ;UNIT # INTO R1
43				RIC	#C7,R1 ;STRIP TO 3 BITS
44				:ENDC	;
45				:IF DF	SDPSYS ;RP11
46				MOV	#RPCS,R1 ;GET CONTROLLER STATUS REG INTO R1
47				RIC	#C4C8.DRV,R1 ;STRIP TO UNIT NUMBER
48				SWAB	R1 ;UNIT # INTO BITS 2-0

49			.ENDC	IDF SDPSYS
50				
51			.IF DF SDXSYS	IFLOPPY
52			MOV RTUNIT,R1	IGPT BOOTED UNIT (STORED BY ROOT2)
53			.ENDC	IDF SDXSYS
54				
55			ADD R1,DKASSG(R0)	IFIX PERMANENT PSEUDO-ASSIGNMENTS
56			ADD R1,SYASSG(R0)	
57			MOVB R1,SYUNIT+1(R0)	ISSET UNIT NUMBER WE BOOTED

BOOT VM02B-01 RT-11 BOOTSTRAP RT-11 MACRO VM02-09 A-APR-75 11:49:04 PAGE 11+
 READ MONITOR, LOOKUP HANDLERS

58									
59					.ENDC				IDF SRKSYS:SDXSYS:IDPSYS
60					.ENABL LSR				
61									
62	001254	054760	000322	000300	BIS	RCNFG,CONFIG(R0)			ISSET HARDWARE CONFIGURATION
63	001262	004003			CLR	R3			ICOUNT DEVICE SLOTS
64	001264	012701	000000G		MOV	#SENTRY,R1			IPOINT TO SENTRY TABLE IN RMON
65	001270	060401			ADD	R4,R1			
66	001272	004721			TST	(R1)+			IRESIDENT DEVICE ?
67	001274	001414			BEG	58			INO, SKIP IT
68	001276	060441			ADD	R4,-(R1)			IFYES, FIX HANDLER POINTER
69	001300	024127	000000G	070370	CMP	SPNAME(R1),#SYNAME			IS THIS THE SYSTEM DEVICE?
70	001306	001006			BNE	458			INO
71	001310	010360	000000G		MOV	R3,SYINDO(R0)			ISSET SYSTEM INDEX NUMBER
72	001314	060360	000000G		ADD	R3,SYINDO(R0)			I(DOUBLED)
73	001320	011160	000000G		MOV	#R1,SYENTO(R0)			IAND SET UP SYSTEM ENTRY POINTER
74	001324	004721			TST	(R1)+			
75	001326	005203			TNC	R3			IFANY MORE ?
76	001330	022703	000000G		CMP	#SSLOT,R3			
77	001334	001336			BNE	48			
78	001336	062700	000010		ADD	#SYBITO,R0			IFADD IN OFFSET TO SYSTEM VECTOR IN MAP
79	001342	152760	000014	000000G	RISB	#SYBITS,MAPOFF(R0)			IFAND PROTECT IT
80	001350	012701	000000G		MOV	#SPNAME,R1			IPOINT TO PERM NAME TABLE
81	001354	060401			ADD	R4,R1			
82	001356	062704	000000G		ADD	#SDVREC,R4			IPOINT R4 TO SDVREC IN RMON
83	001362	012703	000000G		MOV	#SSLOT,R3			INUMBER TO LOOK UP
84	001366	012167	000222		MOV	(R1)+,FNAME			IFILL IN NAME IN LOOKUP
85	001372				LOOKUP	0,#BLOOK			IFLOOKUP SYINH.SYS
86	001400	103002			BCC	78			IFGO IF THERE
87	001402	004024			CLR	(R4)+			IFCLEAR RECORD NUMBER
88	001404	000406			BR	88			
89	001406				SAVEST	0,#CBLOK			IFSAVE STATUS OF THING
90	001414	016714	000204		MOV	CBLOK+2,#R4			IFSET STARTING RECORD
91	001420	004224			INC	(R4)+			IFFIX IT
92	001422	004303			DEC	R3			

93	001424	001360		BNE	6S	
94						
95	001426	012737	100000 000044	MOV	#100000,#JSW	;NOTHING TO SWAP
96	001434			:PRINT	#BSTRNG	;PRINT BOOT HEADER
97	001442	004000		CLR	R0	
98	001444	012720		MOV	(PC)+,(R0)+	
99	001446	040000		BIC	R0,R0	
100	001450	012720		MOV	(PC)+,(R0)+	
101	001452			.EXIT		
102	001454	032767	000000G 000120	RIT	#KW11LS,RCNFG	;AND IF HE HAS A CLOCK
103	001462	001403		REQ	10S	; WE TURN IT
104	001464	012737	000100 177546	MOV	#100,#LKCS	; ON
105	001472	005000		CLR	R0	
106	001474		1001	.EXIT		
107			.DSABL	LSR		
108						
109	001476	005011		RCLR:	CLR	PR1
110	001500	000002		RTI		;TRAP MEANS THIS CONFIGURATION NYET
						;UNTRAP

BOOT VM02B-01 RT-11 BOOTSTRAP RT-11 MACRO VM02-09 8-APR-75 11:49:04 PAGE 12
RELOCATION LIST

1			.SRTTL	RELOCATION LIST	
2	001502	000004	RELIST:	4	;ILLEGAL MEM AND INST. TRAPS
3	001504	000010		10	
4	001506	000030		30	;EMT
5	001510	000054		54	;ADDRESS OF RMON
6	001512	000060		60	;TTY VECTORS
7	001514	000064		64	
8	001516	000100		100	;CLOCK VECTOR
9	001520	000210		SYVEC	;SYSTEM DEVICE VECTOR
10	001522	000244		244	;LOCATION OF FPU TRAP
11	001524	000000G	RELST2:	USRLOC	;LOCATION OF USR NOW
12	001526	000000G		SUSRLC	;ADDRESS OF 'NORMAL' USR
13	001530	000000G		QCOMP	;QUEUE COMPLETION
14	001532	000000G		SKMLC	;ADDRESS OF KMON
15	001534	000000G		TTIBUF	;TTY RING BUFFER--INPUT
16	001536	000002G		TTIBUF+2	
17	001540	000006G		TTIBUF+6	
18	001542	000010G		TTIBUF+10	
19	001544	000000G		TTIBUF	;TTY RING BUFFER--OUTPUT
20	001546	000004G		TTIBUF+4	
21	001550	000006G		TTIBUF+6	
22	001552	000000G		SYSLW	;LOWEST USED LOCATION
23	001554	000002G		CORPTR+2	;FREE CORE LIST
24	001556	000000G		SINPTR	;POINTER TO SINTEN IN RESIDENT HANDLER
25	001560	000000G		SYNCH	;SYNCHRONIZATION ADDRESS

```

26
27      .IF NE BF
28      MSGENT                ;RELOCATE BR STUFF HERE
29      TTUSR
30      TTUSR
31      FUDGE1
32      FUDGE2
33      BKND1
34      BKND2
35      BKND3
36      CNTXT
37      RCNTXT
38      RMONSP
39      SWIPTR
40      SWOPTR
41      .SCRN
42
43      .IFF
44      001562 000000G        TRAPLC
45      001564 000000G        TRAPER                ;LOCS FOR TRAPS TO 4/10
46      001566 000000G        FPPADD
47      001570 000000G        FPPIGN                ;FPP SERVICE FOR MONITOR
48      001572 000000G        MONLOC                ;WHERE USR WILL SIT
49      001574 000000G        I,CSW                  ;SINGLE USER STUFF HERE
50      001576 000000G        AVAIL                  ;MONITOR FREE 0 POINTER
51      .ENDC
52      001600 000000        0                        ;END OF LIST
53      001602 000000        BCNFG1 0                ;BOOT CONFIGURATION WORD-DO NOT MOVE
54      001604 000000G 000000G 000000G TSLIST1 .WORD  KW11LS,HWDSPS,HWFPPUS ;BITS IN CONFIG WORD
55
56      ; BLOCK IS THE ARGUMENT AREA FOR AN RT-11 LOOKUP.
57      001612 075250        BLOCK1 .RAD50 /SY /

```

BOOT VM02B-01 RT-11 BOOTSTRAP RT-11 MACRO VM02B-09 A-APR-75 11149104 PAGE 12+

RELOCATION LIST

```

58 001614 000000 000000        FNAME1 .WORD  0,0                ;FILENAME GOES HERE
59 001620 075273                .RAD50  /SYS/
60 001622                CBLOCK1 .BLKW  5                ;SAVESTATUS GOES HERE

```

BOOT V02B-01 RT-11 BOOTSTRAP RT-11 MACRO VM02-09 8-APR-75 11:49:04 PAGE 13
RELOCATION LIST

1	001634	BUFFB =
2	000002	ROOTSZ = + 777 / 1000
3	002000	= ROOTSZ * 1000
4	000001	END

BOOT V02B-01 RT-11 BOOTSTRAP RT-11 MACRO VM02-09 8-APR-75 11:49:04 PAGE 13+
SYMBOL TABLE

AVAIL = ***** G	BCLR 001476	RCNFG 001602	BF = 000000	BIDERR 000450
BLOOK 001612	ROOT 000520	ROOTSZ= 000002	ROOT! 000034	BSTRNG= ***** G
BUFFB = 001634	CBLOK 001622	CONFIG= 000300	CORPTR= ***** G	DFND 000726
DKASSG= ***** G	ENDBLK= 004000	FIDDLE 000560	FILLER= ***** G	FNAME 001614
FPPAND= ***** G	FPPIGN= ***** G	GT40 = 172000	HWDSPS= ***** G	HWFPU= ***** G
I.CSW = ***** G	JSW = 000044	KMLOC = ***** G	KMON = ***** G	KMONSZ= ***** G
KW11LS= ***** G	LKCS = 177546	MAPOFF= ***** G	MAXSYH= ***** G	MONF 000756
MONFND 001126	MONLOC= ***** G	NXM 000620	PC =X000007	PERM = 000000
QCOMP = ***** G	RCCA = 177452	RCCS = 177446	RCDA = 177442	RCD8 = 177456
RCER = 177444	RCINT = ***** G	RCLA = 177440	RCMN = 177454	RCSIZE= ***** G
RCWC = 177450	READ 000402	RELLST 001502 G	RELST2 001524	REPORT 000500
REPOR1 000474	RTLEN = ***** G	RTSIZE= ***** G	RT11SZ= ***** G	R0 =X000000
R1 =X000001	R2 =X000002	R3 =X000003	R4 =X000004	R5 =X000005
SP =X000006	SR = 177570	SWAPSZ= ***** G	SYASSG= ***** G	SYBITO= 000010
SYBITS= 000014	SYENTO= ***** G	SYINDO= ***** G	SYNAME= 070370	SYNCH = ***** G
SYSIZE= 000704	SYSLOW= ***** G	SYUNIT= 000274	SYVEC = 000210	TKB = 177562
TKS = 177560	TOOSML 001074	TPR = 177566	TPS = 177564	TRAPER= ***** G
TRAPLC= ***** G	TSLIST 001604	TTIBUF= ***** G	TTIBUF= ***** G	USRLOC= ***** G
USRSZ = ***** G	SDVREC= ***** G	SENTRY= ***** G	SINPTR= ***** G	SKMLOC= ***** G
SMONBL= ***** G	SPNAME= ***** G	SPNAMO= ***** G	SRCYS= 000001	SLOT = ***** G
SSWPBL= ***** G	SUSRLC= ***** G	...V1 = 000001		
. ABS. 002000 000				
000000 001				

ERRORS DETECTED: 0
FREE CORE: 14985. WORDS

RCBTSJ,LP1/NITTM/C=RCSYS,BSTRAP

LP V02-03 25-JUN-74 RT-11 MACRO VM02-10 14-APR-75 10:05:11 PAGE 1

A.3 LP/LS11 DEVICE HANDLER

```
1 .TITLE LP V02-03 25-JUN-74
2
3 ; RT-11 LINE PRINTER (LP/LS11) HANDLER
4 ;
5 ; DEC-11-ORTLA-D
6 ;
7 ; RGR/FP/ARC/EF
8 ;
9 ; MARCH 1973/FEBRUARY 1974
10 ;
11 ; COPYRIGHT (C) 1974,1975
12 ;
13 ; DIGITAL EQUIPMENT CORPORATION
14 ; MAYNARD, MASSACHUSETTS 01754
15 ;
16 ; THIS SOFTWARE IS FURNISHED UNDER A LICENSE FOR USE ONLY
17 ; ON A SINGLE COMPUTER SYSTEM AND MAY BE COPIED ONLY WITH
18 ; THE INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE,
19 ; OR ANY OTHER COPIES THEREOF, MAY NOT BE PROVIDED OR OTHERWISE MADE
20 ; AVAILABLE TO ANY OTHER PERSON EXCEPT FOR USE ON SUCH SYSTEM AND TO
21 ; ONE WHO AGREES TO THESE LICENSE TERMS. TITLE TO AND OWNERSHIP OF THE
22 ; SOFTWARE SHALL AT ALL TIMES REMAIN IN DIGITAL.
23 ;
24 ; THE INFORMATION IN THIS DOCUMENT IS SUBJECT TO
25 ; CHANGE WITHOUT NOTICE AND SHOULD NOT BE CONSTRUED
26 ; AS A COMMITMENT BY DIGITAL EQUIPMENT CORPORATION.
27 ;
28 ; DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE
29 ; OR RELIABILITY OF ITS SOFTWARE OR EQUIPMENT
30 ; WHICH IS NOT SUPPLIED BY DIGITAL.
31 ;
```

```

1
2      000000      R0=%0
3      000001      R1=%1
4      000002      R2=%2
5      000003      R3=%3
6      000004      R4=%4
7      000005      R5=%5
8      000006      SP=%6
9      000007      PC=%7
10
11      ; LINE PRINTER CONTROL REGISTERS
12      177514      LPS      = 177514      ;LINE PRINTER CONTROL REGISTER
13      177516      LPR      = 177516      ;LINE PRINTER DATA BUFFER
14      000200      LPVEC    = 200        ;LINE PRINTER VECTOR ADDR
15
16      ;CONSTANTS FOR MONITOR COMMUNICATION
17      000001      HERR     = 1          ;HARD ERROR BIT
18      000054      MONLOW   = 54         ;BASE ADDR OF MONITOR
19      000270      OFFSET   = 270       ;POINTER TO 0 MANAGER COMP ENTRY
20      000340      PR7      = 340
21      000200      PR4      = 200
22
23      ; ASCII CONSTANTS
24
25      000015      CR        = 15
26      000012      LF        = 12
27      000014      FF        = 14
28      000011      HT        = 11
29
30      000204      COLSTZ    = 132.      ;132 COLS
31
32      .GLOBL COLCNT

```

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52

000000
000400
000400 000036 077100
000402 110454
000406 040025
000410 000240
000412 012620 000000
000416 100033
000420 000240
000422 023752 052760
000426 100040
000430 100500
000432 031066 025700
000436 100045
000440 000040
000442 045570 000000
000446 100052
000450 000000
000452 010067 000462
000456 010067 000532
000462 020003
000464 000207
000466 012703
000470 001403
000472 010367 000504
000476 000207
000500 012703
000502 001471
000504 010367 000350
000510 000207
000512 012703
000514 100441
000516 010367 000332
000522 000207
000524 000003
000526 000240
000530 010367 000400
000534 000207

THE FOLLOWING ARE THE PARAMETERS FOR INTERFACE TO THE MONITOR 'SET' COMMAND

1 .ASPECT
= 400

2 .WORD 30. ;MINIMUM WIDTH
RAD50 /WIDTH /
.WORD <0.WIDTH-400>/2+40000 ;NO 'NO' OPTION, NUMBER REQUIRED

NOP ;NO CR => NOP CROPT
RAD50 /CR /
.WORD <0.CR-400>/2+100000 ;ALLOW 'NO'

NOP ;NO FORM0 => NOP FF0PT
RAD50 /FORM0 /
.WORD <0.FORM0-400>/2+100000 ;ALLOW 'NO'

RMT LPERR-ERR0PT+. ;NO HANG BY GOING TO ERROR
RAD50 /HANG /
.WORD <0.HANG-400>/2+100000 ;ALLOW 'NO'

.WORD 40 ;FOR NO LC, CONVERT LC TO UC
RAD50 /LC /
.WORD <0.LC-400>/2+100000

3 0 ;END OF LIST

4 0.WIDTH:MOV R0,COLCNT ;NEW WIDTH TO 2 CONSTANTS
MOV R0,RSTC+2
CMP R0,R3 ;ERROR IF < 30.
RTS PC

0.CR: MOV (PC)+,R3 ;NO 'NO', SO SET TO NO CR
BFG RSTC-CROPT+.
MOV R3,CROPT ;SET CR OPTION
RTS PC

0.FORM0:MOV (PC)+,R3 ;SET TO NO FORMFFERS ON BLOCK 0
BFG BLK0-FF0PT+.
MOV R3,FF0PT
RTS PC

0.HANG: MOV (PC)+,R3 ;SET TO HANG
BMI RET-ERR0PT+.
MOV R3,ERR0PT
RTS PC

0.LC: CLR R3 ;FOR 'LC', LEAVE LOWER CASE STUFF ALONE
NOP
MOV R3,LC0PT
RTS PC

```

1      001000      5      . = 1000
2      ; LOAD POINT
3      001000 000200      LOADPT: .WORD LPVEC      ; ADDR OF INTERRUPT VECTOR
4      001002 000304      .WORD LPINT-      ; OFFSET TO INTERRUPT SERVICE
5      001004 000200      .WORD PR4      ; PRIORITY 7
6      001006 000000      LPIQF: .WORD 0      ; POINTER TO LAST Q ENTRY
7      001010 000000      LPOQF: .WORD 0      ; POINTER TO CURRENT Q ENTRY
8
9      ; ENTRY POINT
10
11     001012 014704 177772      LP: MOV LPOQF,R4      ; R4 POINTS TO CURRENT Q ENTRY
12     001016 000304 000006      ASI A(R4)      ; WORD COUNT TO BYTE COUNT
13     001022 103115      RCF LPERR      ; A READ REQUEST IS ILLEGAL
14     001024 050737 000100 177514      RIS #100,0#LPS      ; CAUSE AN INTERRUPT, STARTING TRANSFER
15     001032 000207      RTS PC
16
17     ; INTERRUPT SERVICE
18
19     001034 000512      RR IPONE      ; ABORT ENTRY POINT
20
21     001036 000577 000242      LPINT: JSR R5,0INTEN      ; INTO SYSTEM STATE
22     001042 000140      .WORD "C<PR4>&PR7
23     001044 014704 177740      MOV LPOQF,R4      ; R4 -> CURRENT QUEUE ELEMENT
24     001050 000737 177514      TST 0#LPS      ; ERROR CONDITION?
25     001054 100041      ERROPT: RMT RET      ; YES-HANG TILL CORRECTED
26     001056 000724      TST (R4)+      ; IS THIS BLOCK 0?
27     001060 001471      FFOPF: RCF RLK0      ; YES - OUTPUT INITIAL FORM FFED
28     001062 000724      TST (R4)+      ; AND POINT TO ADDR OF NEXT CHAR
29     001064 100737 177514      LPNEXT: TSTB 0#LPS      ; READY FOR ANOTHER CHAR YET?
30     001070 100033      RPL RET      ; NOPE - RETURN FROM INTERRUPT
31     001072 100327      ASLB (PC)+      ; TAB IN PROGRESS?
32     001074 000000      TARFLG: .WORD 0
33     001076 001007      RNF TAR      ; BRANCH IF DOING TAB
34     001100 113405      IGNORE: MOVB 0(R4)+,R5      ; GET NEXT CHAR (IF ANY)
35     001102 040705 177600      RLC #177600,R5      ; 7-BIT
36     001106 000714      TST (R4)      ; ANY MORE CHARS?
37     001110 001464      RCF IPONE      ; NO:FINISHED
38     001112 000524      INC (R4)      ; YES, DECREMENT COUNT (IT WAS NEGATIVE)
39     001114 000524      INC -(R4)      ; BUMP BUFFER POINTER
40     001116 120527 000040      CMPB R5,#40      ; PRINTING CHAR?
41     001122 103417      RLC CHRTST      ; NO-GO TEST FOR SPECIAL CHAR.
42     001124 120705 000140      CMPB #140,R5      ; LOWER CASE?
43     001130 103002      RHTS PCHAR      ; NO
44     001132 160705      SUR (PC)+,R5      ; YES, CONVERT IF DESIRED
45     001134 000040      LCOPT: 40
46     001136 000327      PCHAR: DEC (PC)+      ; ANY ROOM LEFT ON LINE?
47     001140 000204      COLCNT: .WORD COLSTZ      ; # OF PRINTER COLUMNS LEFT
48     001142 000706      RLT IGNORE      ; NO MORE ROOM ON LINE, DON'T PRINT CHAR

```

49 001144 10A327
 50 001146 000001
 51 001150 001423
 52 001152 110537 177516
 53 001156 000742
 54 001160 000207
 55 001162 120527 000011
 56 001166 001420
 57 001170 120527 000012

ASLB (PC)+
 TARCNT: .WORD 1
 REQ RSTTAB
 PC1: MOVB R5,#LPB
 RR LPNEXT
 RET: RTS PC
 CRTST: CMPB R5,#HT
 REQ TARSET
 CMPB R5,#LF

!UPDATE TAR COUNT
 !RSET TAB
 !PRINT THE CHAR
 !TRY FOR NEXT CHAR
 !IS CHAR A TAB?
 !YES-RSET TAB
 !IS IT LF?

LP V02-03 25-JUN-74 RT-11 MACRO VM02-10 14-APR-75 10:05:11 PAGE 4+

5A 001174 001406
 59 001176 120527 000015
 60 001202 000240
 61 001204 120527 000014
 62 001210 001333
 63 001212 012767 000204 177720
 64 001220 012767 000001 177720
 65 001226 000751
 6A 001230 014767 177712 177636
 67 001236 012705 000040
 68 001242 000735
 69
 70 001244 005244
 71 001246 022424
 72 001250 012705 000014
 73 001254 000756
 74
 75 001256 052754 000001
 76
 77
 7A
 79 001262 005037 177514
 80 001266 010704
 81 001270 062704 177520
 82 001274 011705 000054
 83 001300 000175 000270
 84
 85 001304 000000
 86
 87 000306
 8A
 89 000001'

REQ RSTC
 CMPB R5,#CR
 CROPT: NOP
 CMPB R5,#FF
 RNE IGNORE
 RSTC: MOV #COLSIZ,COLCNT
 RSTTAB: MOV #1,TABCNT
 RR PC1
 TARSET: MOV TARCNT,TABFLG
 TAR: MOV #40,R5
 RR PCHAR
 BLK0: TNC =(R4)
 CMP (R4)+,(R4)+
 MOV #FF,R5
 RR RSTC
 LPERR: RIS #HDEPR,0-(R4)
 ! OPERATION COMPLETE
 LPNONE: CLP #1,PS
 MOV PC,R4
 ADD #LPCQE-,R4
 MOV #MONLOW,R5
 JMP #OFFSET(R5)
 INTEN: 0
 LPSIZE = .-LOADPT
 .END

!YES-RFSTORE COLUMN COUNT
 !IS IT CR?
 !IGNORE UNLESS MODIFIED
 !IS IT A FF?
 !NO-CHAR IS NON-PRINTING
 !RF-INITIALIZE COLUMN COUNTER
 !RSET TAB COUNTER
 !PRINT THE CHAR
 !SET UP TAR
 !PRINT SPACES
 !MAKE SURE WE ONLY COME HERE ONCE
 !PRINT TO ADDR OF NEXT CHAR
 !PRINT INITIAL FF
 !SET HARD ERROR BIT

LP V02-03 25-JUN-74 RT-11 MACRO VM02-10 14-APR-75 10:05:11 PAGE 4+

BLK0 001244	CHRTST 001162	COLCNT 001140 G	COLSTZ= 000204	CR = 000015
CROPT 001202	ERROPT 001054	FF = 000014	FFOPT 001060	HDERR = 000001
HT = 000011	YGNORE 001100	TNTEN 001304	LCOPT 001134	LF = 000012
LOADPT 001000	LP 001012	LPR = 177516	IPCQF 001010	LPDNE 001262
LPERR 001256	LPINT 001036	IPLOF 001006	IPNEXT 001064	LPS = 177514
LPSITE= 000306	LPVEC = 000200	MONLOW= 000054	OFFSFT= 000270	N.CR 000466
O.FORM 000500	O.HANG 000512	N.LC 000524	N.WINT 000452	PC = %000007
PCHAR 001136	PC1 001152	PR4 = 000200	PR7 = 000340	RET 001160
RSTC 001212	RSTTAB 001220	R0 = %000000	R1 = %000001	R2 = %000002
R3 = %000003	R4 = %000004	R5 = %000005	SP = %000006	TAR 001236
TARCNT 001146	TARFLG 001074	TARSFT 001230		

. ABS. 001306 000
000000 001

ERRORS DETECTED: 0
FREE CORE: 18070. WORDS

,LP:/NITTM/C=LP

CR.SYS RT-11 MACRO VM02-10 28-APR-75 16:00:38 PAGE 1

```

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28

```

```

.TITLE CR.SYS
RT-11 CARD READER (CP11) HANDLER

DEC-11-OCRHA-D

FCP, ARC, RRP
MARCH 1975

COPYRIGHT (C) 1974, 1975

DIGITAL EQUIPMENT CORPORATION
MAYNARD, MASSACHUSETTS 01754

THIS SOFTWARE IS FURNISHED UNDER A LICENSE FOR USE ONLY
ON A SINGLE COMPUTER SYSTEM AND MAY BE COPIED ONLY WITH
THE INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE,
OR ANY OTHER COPIES THEREOF, MAY NOT BE PROVIDED OR OTHERWISE MADE
AVAILABLE TO ANY OTHER PERSON EXCEPT FOR USE ON SUCH SYSTEM AND TO
ONE WHO AGREES TO THESE LICENSE TERMS. TITLE TO AND OWNERSHIP OF THE
SOFTWARE SHALL AT ALL TIMES REMAIN IN DIGITAL.

THE INFORMATION IN THIS DOCUMENT IS SUBJECT TO
CHANGE WITHOUT NOTICE AND SHOULD NOT BE CONSTRUED
AS A COMMITMENT BY DIGITAL EQUIPMENT CORPORATION.

DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE
OR RELIABILITY OF ITS SOFTWARE ON EQUIPMENT
WHICH IS NOT SUPPLIED BY DIGITAL.

```

CR.SYS RT-11 MACRO VM02-10 28-APR-75 16:00:38 PAGE 2
MISCELLANEOUS EQUATES

```

1
2
3
4
5
6
7
8
9
10
11
12

```

```

.SRTTL MISCELLANEOUS EQUATES

R0=X0
R1=X1
R2=X2
R3=X3
R4=X4
R5=X5
SP=X6
PC=X7

CARD READER CONTROL

```

13	000230	CRVECT=230	I INTERRUPT VECTOR
14	177160	CRST=177160	I CARD READER STATUS REGISTER
15	177162	CRB1=177162	I DATA BUFFER 1
16	177164	CRB2=177164	I DATA BUFFER 2
17			
18		I CONSTANTS FOR MONITOR COMMUNICATIONS	
19	000001	HDERR=1	I HARD ERROR
20	000054	MONLOW=54	I BASE ADDRESS OF MONITOR
21	000270	OFFSET=270	I OFFSET TO HANDLER RETURN
22	177776	PS=177776	I PROGRAM STATUS WORD
23	000340	PR7=340	I PRIORITY 7
24	000300	PR4=300	I PRIORITY 4
25			
26		I ASCII CONSTANTS	
27	000015	CR=15	I CARRIAGE RETURN
28	000012	LF=12	I LINE FEED
29	000040	SPACE=40	I SPACE
30	000041	EOF=41	I END OF FILE
31			
32		I CARD READER CONTROL AND STATUS BITS	
33	000001	READ=1	I READ
34	000002	EJECT=2	I EJECT CARD
35	000100	INTER=100	I INTERRUPT ENABLE
36	000200	COLD=200	I COLUMN DONE
37	000400	READY=400	I READY
38	001000	BUSY=1000	I BUSY
39	000000	ONLYIN=2000	I ONLY IN
40	000000	DATLAT=4000	I DATA LATE
41	010000	MOTIN=10000	I MOTION CHECK (CM11 ONLY)
42	020000	HOPCK=20000	I HOPPER CHECK (CM11 ONLY)
43	040000	CADDN=40000	I CARD DONE
44	100000	ERR=100000	I ERROR
45			

```

1
2
3
4
5
6
7      000000
8      000400
9
10
11
12      000400 000404
13      000402 010634 022600
14      000406 100025
15
16
17
18
19
20      000410 000403
21      000412 077731 050500
22      000416 100032
23
24
25
26
27      000420 001151
28      000422 031066 025700
29      000426 100037
30
31
32
33
34      000430 000032
35      000432 012434 017500
36      000436 040044
37
38
39
40      000440 000022
41      000442 035111 026210
42      000446 100072
43
44      000450 000000

```

.SRCTL CONFIGURATION SECTION

THE FOLLOWING CODE IS EXECUTED WHEN A "SET CR" CONSOLE COMMAND IS GIVEN.

.ASECT
=400 CONFIGURATION AREA

SET CR [NO] CRIF
APPEND/DO NOT APPEND CARRIAGE RETURN/LINE FEED TO EACH CARD IMAGE

RR .+1XCRLF=XCRLF
RAD50 /CRLF/
WORD <CRLF=400>/2+100000

SET CR [NO] TRIM
TRIM/DO NOT TRIM TRAILING BLANKS FROM CARD IMAGES

RR .+1XTRIM=XTRIM
RAD50 /TRIM/
WORD <TRIM=400>/2+100000

SET CR [NO] HANG
HANG/RETURN HARD ERROR IF READER NOT READY AT START OF TRANSFER

RNF .+1ERROR=XHANG
RAD50 /HANG/
WORD <HANG=400>/2+100000

SET CR CODE 1=1 026 10291
SET TRANSLATION TO 026 (029) MODE

WORD 026.
RAD50 /CODE/
WORD <CODE=400>/2+40000

SET CR [NO] IMAGE
TRANSMIT EACH COLUMN AS 12 BITS (ONE WORD/COLUMN)

WORD NOTMAG=IMBASE
RAD50 /IMAGE/
WORD <IMAGE=400>/2+100000

WORD 0 END-OF-OPTIONS FLAG

			.SRTTL	CONFIGURATION SUBROUTINES	
1					
2	000452	012703	CRI F:	MOV	(PC)+,R3 ;NOP IF POSITIVE
3	000454	000240		NOP	
4	000456	010367	000246'	MOV	R3,XCRIF ;ENTRY POINT FOR NO
5	000462	000207		RTS	PC
6					
7	000464	012703	TRIM:	MOV	(PC)+,R3 ;NOP IF POSITIVE
8	000466	000240		NOP	
9	000470	010367	000236'	MOV	R3,XTRIM ;ENTRY POINT FOR NO
10	000474	000207		RTS	PC
11					
12	000476	012703	HANG:	MOV	(PC)+,R3 ;NOP IF POSITIVE
13	000500	000240		NOP	
14	000502	010367	000036'	MOV	R3,XHANG ;ENTRY POINT FOR NO
15	000506	000207		RTS	PC
16					
17	000510	010701	CODE:	MOV	PC,R1 ;R1 -> CONVERSION TABLE FOR 024
18	000512	062701	000122	ADD	#SFT024=.,01
19	000516	160300		SUB	R3,R0 ;026 REQUESTED?
20	000520	100407		BMT	CODEXT ;NOPE = ERROR (NOTE THAT C IS SET)
21	000522	001407		REQ	SETCND ;YES, IT'S 026 - GO DO IT
22	000524	062701	000050	ADD	#SFT029-SET026,R1 ;R1 -> CONVERSION TABLE FOR 029
23	000530	022700	000003	CMP	#3,R0 ;WAS IT 029?
24	000534	001402		REQ	SETCND ;YES
25	000536	000261		SEC	;ELSE AN ERROR - INDICATE SUCH
26	000540	000207		CODEXT: RTS	;AND RETURN TO KMON
27	000542	010703		SETCND: MOV	PC,R3 ;R3 -> CHARACTER TABLE
28	000544	062703	177716'	ADD	#CHRTBL=.,R3
29	000550	000500		SCODF: CLR	R0 ;PICK UP NEXT OFFSET TO BE MODIFIED
30	000552	152100		RISH	(R1)+,R0 ;FROM APPROPRIATE TABLE
31	000554	001771		REQ	CODEXT ;ALL DONE (NOTE: C IS CLEAR)
32	000556	060300		ADD	R3,R0 ;POINT TO BYTE TO MODIFY
33	000560	112110		MOVB	(R1)+,@R0 ;AND PLUS IN NEW VALUE
34	000562	000772		RR	SCODF ;CONTINUE
35					
36					
37	000564	062703	000010	IMAGE: ADD	#YAIMAG-NOTMAG,R3 ;POINT TO "YES" TABLE
38	000570	060703		ADD	PC,R3 ;ENTRY FOR "NO"
39	000572	012367	000136'	IMBASE: MOV	(R3)+,XIM1 ;AND PATCH HANDLER
40	000576	012367	000154'	MOV	(R3)+,XIM2
41	000602	012367	000156'	MOV	(R3)+,XIM2+2
42	000606	012367	000164'	MOV	(R3)+,XIM3+2
43	000612	000207		RTS	PC
44					
45	000614	001402		NOTMAG: REQ	;+NEXTCHR=XIM1
46	000616	116537	000306	MOVB	CHRTBL=XIM2(R5),@(PC)+
47	000622	000001		:WORD	1
48	000624	000402		YATMAG: RR	;+NEXTCHR=XIM1
49	000626	017737	177162	MOV	@RR1,@(PC)+
50	000632	000002		:WORD	2

A-38

```

1          .SRITL 024, 029 CONVERSION TABLES
2
3          ; 024 CONVERSION TABLE
4          ; MODIFIES CHARACTER TABLE TO ACCEPT 026 KEYPUNCH CODES
5
6 000634      012      137      SET026: .BYTF 012,137      TRACK ARROW      (8-2)
7 000636      013      075      .BYTF 013,075      EQUAL      (8-3)
8 000640      015      136      .BYTF 015,136      UP ARROW      (8-5)
9 000642      016      047      .BYTF 016,047      APOSTROPHE      (8-6)
10 000644      017      134      .BYTF 017,134      BACKSLASH      (8-7)
11 000646      052      073      .BYTF 052,073      SEMICOLON      (0-8-2)
12 000650      054      050      .BYTF 054,050      LEFT PAREN      (0-8-4)
13 000652      055      042      .BYTF 055,042      QUOTES      (0-8-5)
14 000654      056      043      .BYTF 056,043      L.R. STGN      (0-8-6)
15 000656      057      045      .BYTF 057,045      PERCENT      (0-8-7)
16 000660      112      072      .BYTF 112,072      COLON      (11-A-2)
17 000662      115      133      .BYTF 115,133      L BRACKET      (11-A-5)
18 000664      116      076      .BYTF 116,076      GREATER THAN      (11-A-6)
19 000666      117      046      .BYTF 117,046      AMPERSAND      (11-A-7)
20 000670      200      053      .BYTF 200,053      PLUS      (12)
21 000672      212      077      .BYTF 212,077      QUESTION      (12-A-2)
22 000674      214      051      .BYTF 214,051      RIGHT PAREN      (12-A-4)
23 000676      215      135      .BYTF 215,135      R BRACKET      (12-A-5)
24 000700      216      074      .BYTF 216,074      LESS THAN      (12-A-6)
25 000702      000000      .WORD 0      *** END OF TABLE **
26
27          ; 029 CONVERSION TABLE
28          ; MODIFIES CHARACTER TABLE TO ACCEPT 029 KEYPUNCH CODES
29
30 000704      012      072      SET029: .BYTF 012,072      COLON      (8-2)
31 000706      013      043      .BYTF 013,043      L.R. STGN      (8-3)
32 000710      015      047      .BYTF 015,047      APOSTROPHE      (8-5)
33 000712      016      075      .BYTF 016,075      EQUAL      (8-6)
34 000714      017      042      .BYTF 017,042      QUOTES      (8-7)
35 000716      052      134      .BYTF 052,134      BACKSLASH      (0-8-2)
36 000720      054      045      .BYTF 054,045      PERCENT      (0-8-4)
37 000722      055      137      .BYTF 055,137      TRACK ARROW      (0-8-5)
38 000724      056      076      .BYTF 056,076      GREATER THAN      (0-8-6)
39 000726      057      077      .BYTF 057,077      QUESTION      (0-8-7)
40 000730      112      135      .BYTF 112,135      R BRACKET      (11-A-2)
41 000732      115      051      .BYTF 115,051      RIGHT PAREN      (11-A-5)
42 000734      116      073      .BYTF 116,073      SEMICOLON      (11-A-6)
43 000736      117      136      .BYTF 117,136      UP ARROW      (11-A-7)
44 000740      200      046      .BYTF 200,046      AMPERSAND      (12)
45 000742      212      133      .BYTF 212,133      L BRACKET      (12-A-2)
46 000744      214      074      .BYTF 214,074      LESS THAN      (12-A-4)
47 000746      215      050      .BYTF 215,050      LEFT PAREN      (12-A-5)
48 000750      216      053      .BYTF 216,053      PLUS      (12-A-6)
49 000752      000000      .WORD 0      *** END OF TABLE **
50
51          .JTF GF <CHRTBI-10ADPT+2165-1000,          .ERROR 1TABLE NOT IN BLOCK 1

```

```

1          .SRTTL  HANDLER PROPER
2
3          000000"
4          .CSECT  CR11
5
6          ; LOAD POINT
7          LOADPT: .WORD  CRVECT      ; INTERRUPT VECTOR
8                  .WORD  CRINT=,     ; OFFSET TO INTERRUPT SERVICE
9                  .WORD  300         ; IPS
10         CRIGF: .WORD  0            ; LAST QUEUE ENTRY
11         CRQGF: .WORD  0            ; CURRENT QUEUE ENTRY
12
13         ; ENTRY POINT
14
15         CRHAND: MOV     CRQGF,R5      ; POINT TO 0 ELEMENT
16                 MOV     CRPTR,R4      ; POINT INTO CARD IMAGE
17                 ASI     6(R5)         ; CONVERT WORD COUNT TO BYTE COUNT
18                 PLBS     LEPROR      ; NULL REQUEST OR WRITE IS LOGIC FRP
19                 BIT     #READY+BUSY,0#CRST ; IS READER READY?
20                 XHANG: NOP            ; * PATCH HERE TO ISSUE HARD ERROR
21                 TST     (R5)+         ; BLOCK 0 REQUEST ?
22                 RFD     READR         ; YES, GO INITIATE REQUEST
23                 TST     (R5)+         ; NO, POINT TO BUFFER PTRS
24                 RR      CONT         ; GO SEE IF ANY STUFF IS LEFT IN OLD CARD
25
26                 RR      ABORT         ; ABORT
27
28         ; INTERRUPT ENTRY POINT
29
30         CRINT: .ISR     R5,0#INPTR    ; ENTER SYSTEM STATE
31                .WORD  2C<PR6>8PR7
32                DEC     COICNT         ; COUNT DOWN INTERRUPTS THIS CARD
33                MOV     #CRST,R5      ; GET STATUS
34                RMT     FRPOP         ; WHOOPS -- ERROR
35                TSTR     R5           ; CHECK FOR COLUMN DONE
36                RPI     CAPD         ; BRANCH IF NOT COLUMN DONE
37                MOV     #CR2,R5       ; GET COMPRESSED CHAR
38                RFD     INPOLT        ; IT'S BLANK
39                MOV     #CR1,-(SP)     ; GET EXPANDED CHAR
40                CMP     CRPTR,BUFFPT   ; FIRST COLUMN?
41                RNE     TSTPIN        ; NOPE
42                MOV     #SP,CHAR12     ; ELSE SAVE FOR EOF CHECK
43                RLC     #177003,0SP    ; CHECK ONLY ROWS 1-7
44                MOV     #SP,-(SP)      ; CHECK FOR INVALID PUNCHES
45                NEG     #SP           ; BY CHECKING FOR 2 OR MORE PUNCHES
46                RLC     (SP)+,(SP)+    ; IN COLUMNS 1-7
47                RFD     NXTCHR         ; IT'S OKAY
48                MOV     #377,R5       ; ELSE FORCE TRANSLATION INTO 134
49                MOV     CRPTR,(PC)+    ; REMEMBER POSITION OF NON-BLANK

```

49	000150	000000		ENDPTR: .WORD	0	
50	000152	060705		INCLT: ADD	PC,R5	MAKE A PIC POINTER TO TRANS TABLE
51	000154	114537	000306	YIM2: MOVB	CHRTBL=(R5),@(PC)+	PUT TRANSLATED CHAR IN LINE
52	000160	000000		CHRPTR: .WORD	0	
53	000162	060767	000001 177770	YIM3: ADD	#1,CHRPTR	PUSH BUFFER POINTER BY 1 OR 2
54	000170	052737	000100 177160	INTRET: R1	#INTFB,#CRST	ENABLE ON LINE INTR IF NECESSARY
55	000176	000207		RTS	PC	
56						
57						

CR,SYS RT-11 MACRO VM02-17 28-APR-75 16:00:38 PAGE 4+
HANDLER PROPFER

58						
59						
60	000200	032705	040000	CARD: RIT	#CARDN,R5	CARD DONE?
61	000204	001440		REQ	FRP1	NOPE -- SPURIOUS INTERRUPT - IGNORE
62	000206	012704		MOV	(PC)+,R4	R4 -> CHRRUF
63	000210	000000		RUEPTR: .WORD	0	POINTER TO CHRRUF
64	000212	016705	177572	MOV	CRCPF,R5	POINT TO 0 ELEMENT
65	000216	022525		CMP	(R5)+,(R5)+	PUSH OVER BUFFER POINTER
66	000220	022727	007417	CMP	#7017,(PC)+	END OF FILE?
67	000224	000000		CHAR12: .WORD	0	12-BIT FOR FIRST CARD COL.
68	000226	001471		REQ	ENNETL	YES
69	000230	010146		MOV	R1,=(SP)	SAVE R1
70	000232	016701	177722	MOV	CHRPTR,R1	POINT TO LAST END OF 80 CHARS
71	000236	000240		YTRIM: NOP		* PATCH HERE TO SUPPRESS TRIMMING
72	000240	016701	177704	MOV	ENDPTR,R1	IDA, GIVE IT TO HIM
73	000244	000201		INC	R1	
74	000246			IXTRIM:		
75	000246	000240		YCOLF: NOP		* PATCH HERE TO SUPPRESS CR/LF
76	000250	112721	000015	MOVB	#CR,(R1)+	JA, GIVE IT HIM
77	000254	112721	000012	MOVB	#LF,(R1)+	
78	000260	010167	177664	IXCRIF: MOV	R1,ENDPTR	THIS IS NOW THE END
79	000264	012601		MOV	(SP)+,R1	RESTORE R1
80	000266	000404		RR	CONT	ENTER FILLING LOOP
81	000270	112435		FIIBUF: MOVB	(R4)+,@(R5)+	PUT A BYTE IN HIS BUFFER
82	000272	000515		DEC	#R5	IS HE FULL ?
83	000274	001457		REQ	DETMON	YEP
84	000276	000245		INC	=(R5)	PUSH BUFFER POINTER
85	000300	026704	177644	CONT: CMP	ENDPTR,R4	END OF OUR CARD ?
86	000304	101371		RHT	FIIBUF	NOT YET
87	000306	032737	001400 177160	FRP1: RIT	#READY+BUSY,#CRST	OKAY TO INIT READ?
88	000314	001325		RNF	INTRET	NOPE - GO HANG UNTIL READY
89	000316	010704		READR: MOV	PC,R4	POINT TO CHRRUF
90	000320	062704	000542	ADD	#CHRRUF=.,04	

```

91 000324 010467 177630      MOV      R4,CHRPTR      ;START FILLING FROM NEW CARD
92 000330 010467 177614      MOV      R4,ENDPTR      ;WHICH IS AS YET EMPTY
93 000334 010467 177650      MOV      R4,BUFPTR      ;ESTABLISH BUFFER POINTER
94 000340 005067 177660      CLR      CHA12        ;CLEAR EOF FLAG
95 000344 012727 000120      MOV      #0,,(PC)+      ;SET COLUMN COUNT
96 000350 000000              COICNT:  WORD      0          ;COUNT OF COLUMNS REMAINING IN CARD
97 000352 012737 000101 177160      MOV      #READ+INTER,##CRST ;START A CARD GOING
98 000360 000207              RTS      PC          ;BYE
99
100
101
102 000362 016705 177422      IERROR:  MOV      CRCOE,R5      ;POINT TO QUEUE ELEMENT
103 000366 052755 000001      RIS      #HERR,0-(R5)    ;YOU CAN'T WRITE ON A READER
104 000372 000416              RR              ABORT
105
106 000374 032705 004000      ERROR:  RIT      #DATL,R5      ;DATA RATE IS ONLY NOT DURABLE
107 000400 001370              RNF      IERROR      ;ISSUE HARD ERROR IF SO
108 000402 005767 177742      TST      COICNT          ;DONE WITH DATA COLUMNS?
109 000406 100337              RPI      FRP1        ;NONE -- MUST RE PICK CHECK, ETC.,
110
111 000410 000673              RR      CAPD          ; START A NEW READ TO CORRECT CONDITION.
112
113
114
115
116
117
118
119
120
121
122
123
124 000434 010467 177520      PETMON:  MOV      R4,CHRPTR      ;SAVE POSITION IN CARD
125 000440 005037 177140      CLR      ##CRST        ;NO INTERRUPTS
126 000444 010704              MOV      PC,R4        ;THE USUAL MONITOR RETURN
127 000446 062704 177342      ADD      #CRCOE-,,R4
128 000452 013705 000054      MOV      ##MONLOW,R5
129 000456 000175 000270      JMP      @OFFSET(R5)

```

CR.SYS RT=11 MACRO VM02-10 28-APR-75 16:00:38 PAGE 6+
HANDLER PROPER

```

115 000412 012504      ENDFIL:  MOV      (R5)+,R4      ;POINT INTO HIS BUFFER
116 000414 105024      CLDBUF:  CLRB      (R4)+      ;CLEAR IT ALL
117 000416 005315      DEC      @R5
118 000420 001375      RNF      CLDBUF
119 000422 052775 020000 177770      RIS      #20000,0-10(R5) ;SET EOF BIT IN CHANNEL
120 000430 005067 177514      ABORT:  CLR      ENDPTR      ;FORCE A READ NEXT TIME
121
122
123
124 000434 010467 177520      PETMON:  MOV      R4,CHRPTR      ;SAVE POSITION IN CARD
125 000440 005037 177140      CLR      ##CRST        ;NO INTERRUPTS
126 000444 010704              MOV      PC,R4        ;THE USUAL MONITOR RETURN
127 000446 062704 177342      ADD      #CRCOE-,,R4
128 000452 013705 000054      MOV      ##MONLOW,R5
129 000456 000175 000270      JMP      @OFFSET(R5)

```

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35

000462
000400
000462'

```
.SRITL CHARACTER TABLE

; THE FOLLOWING MACRO TAKES AS ARGUMENTS THE ASCII TRANSLATION
; DESIRED AND THE LIST OF PUNCH COMBINATIONS FOR THAT CHARACTER.

.MACRO C SLIST
  T=0
  .IRP X,<SLIST>
  .IF NE X'
  .IF IE X'=-7
    T=T+X'
  .IFF
    II=10
  .REPT X'=-A.
    II=II+II
  .ENDR
  T=T+II
  .ENDC
  .IFF
    T=T+40
  .ENDC
  .ENDR
  .=CHPTBL+T
  .BYTE $CHAR
  $CHAR = $CHAR + 1
.ENDM C

; THE FOLLOWING TABLE TRANSLATES 029 KEYPUNCH CODES TO ASCII.
CHPTBL:
.REPT 256.
  .BYTE 130 ;DEC STANDARD ERROR CHARACTER
.ENDR

.=CHPTBL
```

36	000000		SCMAP = 0	START AT OCTAL 000
37				
38	000462	C	<12,0,0,2,1>	INULI
39	000754	C	<12,0,1>	CTRL-A
40	000704	C	<12,0,2>	CTRL-B
41	000705	C	<12,0,3>	CTRL-C
42	000706	C	<9,7>	CTRL-D
43	000512	C	<0,9,8,5>	CTRL-E
44	000560	C	<0,9,8,6>	CTRL-F
45	000561	C	<0,9,8,7>	CTRL-G
46	000562	C	<11,0,6>	CTRL-H
47	000611	C	<12,0,5>	CTRL-I
48	000710	C	<0,9,5>	CTRL-J
49	000550	C	<12,0,8,3>	CTRL-K
50	000716	C	<12,0,8,4>	CTRL-L
51	000717	C	<12,0,8,5>	CTRL-M
52	000720	C	<12,0,8,6>	CTRL-N
53	000721	C	<12,0,8,7>	CTRL-O
54	000722	C	<12,11,9,8,1>	CTRL-P
55	001014	C	<11,0,1>	CTRL-Q
56	000604	C	<11,0,2>	CTRL-R
57	000605	C	<11,0,3>	CTRL-S

A-43

CR.SYS RT-11 MACRO VM02-10 28-APR-75 16:00:3A PAGE 7+
CHARACTER TABLE

58	000606	C	<9,8,4>	CTRL-T
59	000517	C	<9,8,5>	CTRL-U
60	000520	C	<9,2>	CTRL-V
61	000505	C	<0,9,6>	CTRL-W
62	000551	C	<11,0,8>	CTRL-X
63	000613	C	<11,0,8,1>	CTRL-Y
64	000614	C	<9,8,7>	CTRL-Z
65	000522	C	<0,9,7>	ALTMODE (FSCAPE)
66	000552	C	<11,0,8,4>	CTRL-[
67	000617	C	<11,0,8,5>	CTRL-]
68	000620	C	<11,0,8,6>	CTRL-^
69	000621	C	<11,0,8,7>	CTRL-`
70	000622	C	<>	SPACE
71	000463	C	<12,8,7>	!
72	000702	C	<8,7>	"
73	000502	C	<8,3>	#
74	000476	C	<11,8,3>	\$
75	000576	C	<0,8,4>	%
76	000537	C	<12>	&
77	000663	C	<8,5>	'
78	000500	C	<12,8,5>	(
79	000700	C	<11,8,5>	)
80	000600	C	<11,8,4>	*

81	000577	C	<12,A,4>	I+
82	000701	C	<0,8,3>	I,
83	000536	C	<11>	I-
84	000563	C	<12,A,3>	I.
85	000676	C	<0,1>	I/
86	000524	C	<0>	I0
87	000523	C	<1>	I1
88	000444	C	<2>	I2
89	000445	C	<3>	I3
90	000466	C	<4>	I4
91	000467	C	<5>	I5
92	000470	C	<6>	I6
93	000471	C	<7>	I7
94	000472	C	<8>	I8
95	000473	C	<9>	I9
96	000503	C	<8,2>	I:
97	000475	C	<11,A,4>	I;
98	000601	C	<12,A,4>	I<
99	000677	C	<8,6>	I#
100	000501	C	<0,8,6>	I>
101	000541	C	<0,8,7>	I?
102	000542	C	<8,4>	I@
103	000477	C	<12,1>	IA
104	000664	C	<12,2>	IA
105	000665	C	<12,3>	IC
106	000666	C	<12,4>	ID
107	000667	C	<12,5>	IE
108	000670	C	<12,6>	IF
109	000671	C	<12,7>	IG
110	000672	C	<12,8>	IH
111	000673	C	<12,9>	II
112	000703	C	<11,1>	IJ
113	000564	C	<11,2>	IK
114	000565	C	<11,3>	IL

CR.SYS RT-11 MACRO VM02-10 28-APR-75 16:00:38 PAGE 7+

CHARACTER TABLE

115	000566	C	<11,4>	IM
116	000567	C	<11,5>	IN
117	000570	C	<11,6>	IO
118	000571	C	<11,7>	IP
119	000572	C	<11,8>	IQ
120	000573	C	<11,9>	IR
121	000603	C	<0,2>	IS
122	000525	C	<0,3>	IT

123 000526
 124 000527
 125 000530
 126 000531
 127 000532
 128 000533
 129 000543
 130 000675
 131 000535
 132 000575
 133 000602
 134 000540
 135 000474
 136 000724
 137 000725
 138 000726
 139 000727
 140 000730
 141 000731
 142 000732
 143 000733
 144 000743
 145 000764
 146 000765
 147 000766
 148 000767
 149 000770
 150 000771
 151 000772
 152 000773
 153 001003
 154 000625
 155 000626
 156 000627
 157 000630
 158 000631
 159 000632
 160 000633
 161 000643
 162 000723
 163 000763
 164 000623
 165 000624
 166
 167 001062
 168
 169 001062
 170 001324 000000
 171

U	<0,4>	!U
U	<0,5>	!V
U	<0,6>	!W
U	<0,7>	!X
U	<0,8>	!Y
U	<0,9>	!Z
U	<1,0,2>	!["
U	<0,8,2>	!\
U	<11,8,2>	!1
U	<11,8,7>	!2
U	<0,8,5>	!.
U	<8,1>	!ACCENT GRAVE
U	<12,0,1>	!LC A
U	<12,0,2>	!LC B
U	<12,0,3>	!LC C
U	<12,0,4>	!LC D
U	<12,0,5>	!LC E
U	<12,0,6>	!LC F
U	<12,0,7>	!LC G
U	<12,0,8>	!LC H
U	<12,0,9>	!LC I
U	<12,11,1>	!LC J
U	<12,11,2>	!LC K
U	<12,11,3>	!LC L
U	<12,11,4>	!LC M
U	<12,11,5>	!LC N
U	<12,11,6>	!LC O
U	<12,11,7>	!LC P
U	<12,11,8>	!LC Q
U	<12,11,9>	!LC R
U	<11,0,2>	!LC S
U	<11,0,3>	!LC T
U	<11,0,4>	!LC U
U	<11,0,5>	!LC V
U	<11,0,6>	!LC W
U	<11,0,7>	!LC X
U	<11,0,8>	!LC Y
U	<11,0,9>	!LC Z
U	<12,0>	!OPEN BRACE
U	<12,11>	!VERTICAL BAR
U	<11,0>	!CLOSE BAR
U	<11,0,1>	!TITLE
U	<12,0,7>	!DEL

. = CTRL + 256.

CHOBIF: RIKW 21.
 SINPTR: WORD 0

!PLUGGED TO POINT TO COMMON ENTRY

CR.SYS RT-11 MACRO VM02-10 28-APR-75 16:00:38 PAGE 7+
 CHARACTER TABLE

172 001326 CRSITE=-LOADPT
 173
 174 000001' .END

CR.SYS RT-11 MACRO VM02-10 28-APR-75 16:00:38 PAGE 7+
 SYMBOL TABLE

ABORT 000430R 002 RUFPTR 000210R 002 BUSY = 001000
 CHAR12 000224R 002 CHRBUF 001062R 002 CHRPTR 000160R
 CODE 000510 CODEXT 000540 COLCNT 000350R
 CR = 000015 CRP1 = 177162 CRP2 = 177164
 CRINT 000052R 002 CRIF 000452 CRIOF 000006R
 CRVECT 000230 DATLAT= 004000 EJECT = 000002
 FOF = 000041 FRR = 100000 FRRDP 000374R
 HANG 000476 HOFRR = 000001 HOPCK = 020000
 INCOLT 000152R 002 INTER = 000100 INTRFT 000170R
 LOADPT 000000R 002 IXCRIE 000260R 002 IYTRIM 000246R
 NOYTAG 000614 NXTCHR 000144R 002 OFFSFT= 000270
 PRA = 000300 PR7 = 000340 P8 = 177776
 READY = 000400 RETMON 000434R 002 P9 = %000000
 R3 = %000003 R4 = %000004 P5 = %000005
 SET026 000634 SET029 000704 SP = %000006
 TRIM 000464 TSTPIN 000124R 002 U = 000020
 XIM1 000136R 002 YIM2 000154R 002 YIM3 000162R
 XCHAR = 000200 YINPTR 001324R 002
 . ABS. 000754 000
 000000 001
 CR11 001326 002
 ERRORS DETECTED: 0
 FREE CORE: 17698. WORDS
 ,LP:/N:TTM/C=CP

CARD 000200R 002 CARDN = 040000
 CHRTPL 000462R 002 CLRBUF 000414R 002
 COLD = 000200 CONT 000300R 002
 CRIOF 000010R 002 CRHAND 000012R 002
 CRSITE= 001326 CRST = 177160
 ENDFIL 000412R 002 ENDPTR 000150R 002
 FRP1 000306R 002 FILBUF 000270R 002
 IMAGE 000564 IMAGE 000572
 IERROR 000362R 002 IF = 000012
 MONLOW= 000054 MOTIN = 010000
 ONLIN = 002000 PC = %000007
 READ = 000001 READP 000316R 002
 P1 = %000001 P2 = %000002
 SCODE 000550 SETCOD 000542
 SPACF = 000040 T = 000227
 YCRLE 000246R 002 YHANG 000036R 002
 YTRIM 000236R 002 YATMAG 000624

```
1      .TITLE  DT VM2-07 12-APR-74
2
3      ; RT-11 DECTAPE (TC11) HANDLER
4      ;
5      ; DEC-11-OPTDA-D
6      ;
7      ; PB/EF
8      ;
9      ; AUGUST, 1974
10     ;
11     ; COPYRIGHT (C) 1974,1975
12     ;
13     ; DIGITAL EQUIPMENT CORPORATION
14     ; MAYNARD, MASSACHUSETTS 01754
15     ;
16     ; THIS SOFTWARE IS FURNISHED UNDER A LICENSE FOR USE ONLY
17     ; ON A SINGLE COMPUTER SYSTEM AND MAY BE COPIED ONLY WITH
18     ; THE INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE,
19     ; OR ANY OTHER COPIES THEREOF, MAY NOT BE PROVIDED OR OTHERWISE MADE
20     ; AVAILABLE TO ANY OTHER PERSON EXCEPT FOR USE ON SUCH SYSTEM AND TO
21     ; ONE WHO AGREES TO THESE LICENSE TERMS. TITLE TO AND OWNERSHIP OF THE
22     ; SOFTWARE SHALL AT ALL TIMES REMAIN IN DIGITAL.
23     ;
24     ; THE INFORMATION IN THIS DOCUMENT IS SUBJECT TO
25     ; CHANGE WITHOUT NOTICE AND SHOULD NOT BE CONSTRUED
26     ; AS A COMMITMENT BY DIGITAL EQUIPMENT CORPORATION.
27     ;
28     ; DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE
29     ; OR RELIABILITY OF ITS SOFTWARE ON EQUIPMENT
30     ; WHICH IS NOT SUPPLIED BY DIGITAL.
31     ;
```

1	000000	.CRECT RYSHND	
2			
3		.ENARL ISR	
4			
5	000000	R0=%0	
6	000001	R1=%1	
7	000002	R2=%2	
8	000003	R3=%3	
9	000004	R4=%4	
10	000005	R5=%5	
11	000006	SP=%6	
12	000007	PC=%7	
13	177776	PS=177776	
14			
15		.DECTAPE CONTROL REGISTERS	
16	177350	TCNT = 177350	DATA REGISTER
17	177340	TCST = 177340	CONTROL AND STATUS REGISTER
18	177342	TCOM = 177342	COMMAND REGISTER
19	177344	TCWC = 177344	WORD COUNT REGISTER
20	177346	TCRA = 177346	BUS ADDRESS REGISTER
21	000214	TCVEC = 214	TC11 INTERRUPT VECTOR
22			
23		.CONSTANTS FOR MONITOR COMMUNICATION	
24	000001	HDERR = 1	HARD ERROR BIT
25	000050	MONLOW = 50	MONITOR BASE POINTER
26	000270	OFFSET = 270	POINTER TO Q MANAGER COMP ENTRY
27			
28		.GLOBAL DTSYS, PKSYS, PFSYS, DPSYS, DSSYS, DXSYS	
29		.GLOBAL SINTPR, SINTEN, DTINT	
30			
31	000000	PKSYS = 0	PK IS NOT RESIDENT
32	000000	PFSYS = 0	NEITHER IS PF
33	000000	DXSYS = 0	
34	000000	DPSYS = 0	
35	000000	DSSYS = 0	
36	000340	PR7 = 340	ENTERS AT LEVEL 7

```

1
2 000000
3
4 000000 000214
5 000002 000032
6 000004 000340
7 000006
8 000006 000000
9 000010 000000
10
11
12 000012 012727 000010
13 000016 000000
14 000020 013746 177776
15 000024 000767 000004
16 000030 000207
17
18
19
20 000032 000510
21 000034 004577 000262
22 000040 000004
23 000042 010006
24 000044 014700 177740
25 000050 012704 177342
26 000054 012046
27 000056 012005
28 000060 042705 174377
29 000064 032714 100100
30 000070 100452
31 000072 001502
32
33 000074 032714 000002
34 000100 001463
35 000102 023767 177350 000206
36 000110 001422
37 000112 003407
38 000114 052705 004000
39 000120 162716 000002
40
41
42 000124 000402
43 000126 052705 010000
44
45 000132 052705 000103
46 000136 012667 000154
47 000142 010514
48 000144 012600

      REG:
      ; LOAD POINT
      ;WORD TCVER
      ;WORD DTINT-
      ;WORD PR7
      ;ADDRESS OF INTERRUPT VECTOR
      ;OFFSET TO INTERRUPT SERVICE
      ;PRIORITY 7

      DTSYS:
      DTICF: ;WORD 0
      DTICF: ;WORD 0
      ;POINTER TO LAST Q ENTRY
      ;POINTER TO CURRENT Q ENTRY

      ; ENTRY POINT
      MOV #8, (PC)+
      DTTRY: ;WORD 0
      MOV #PS, -(SP)
      JSR PC, DTINT
      RTS PC
      ;INIT THE RETRY COUNT
      ;RETRY COUNTER
      ;FAKE AN INTERRUPT
      ;BACK TO MONITOR

      ; INTERRUPT SERVICE

      RR      DSTOP
      DTINT: JSR #5, SINPTR
      ;WORD PC<300>&PR7
      MOV R0, -(SP)
      MOV DTICF, R0
      MOV #TCM, R4
      MOV (R0)+, -(SP)
      MOV (R0)+, R5
      RLC #PC<3400>, R5
      RIT #100100, (R4)
      RMT DTICF
      RR      RETRY
      ;ABORT CALL FROM RF SYSTEM
      ;NOW JSR TO COMMON CODE
      ;RUN AT LEVEL 6

      RIT #2, (R4)
      RR      DTDONE
      CMP #TCM, RWANT
      RR      RLKEND
      DIRECT: RLF FORWARD
      REVERSE: RIS #4000, R5
      SUR #2, (SP)
      ;R0 POINTS TO Q ELEMENT
      ;R4 POINTS TO CONTROL REGISTER
      ;DESIRED BLOCK # ONTO STACK
      ;UNIT # INTO R5
      ;ISOLATE UNIT NUMBER
      ;ERROR HIT ON?
      ;YES
      ;IF INTERRUPT IS OFF, WE ARE INITIATING
      ;A REQUEST
      ;SEARCHING?
      ;NO-A READ OR WRITE JUST COMPLETED
      ;COMPARE ACTUAL BLOCK TO DESIRED BLOCK
      ;FOUND IT
      ;SEARCH IN THE FORWARD DIRECTION
      ;SET REVERSE BIT
      ;SEARCH FOR TWO BLOCKS BEFORE ONE
      ;ACTUALLY DESIRED (TO ALLOW
      ;SPACE FOR THE TURN-AROUND
      ;DON'T SET DELAY INHIBIT
      ;TAPE IS ALREADY MOVING FORWARD
      ;SO INHIBIT HARDWARE DELAY
      ;INTERRUPT ENABLE, RNIM, AND GO
      ;REMEMBER THE BLOCK WE ARE LOOKING FOR
      ;TELI CONTROLLER TO GO
      ;RESTORE R0

```

49	000146	000207		RIS	PC	BACK INTO MONITOR
50						
51	000150	032714	004000	ENDZR:	RIT	#4000,(R4)
52	000154	001757			REN	REVERSE
53	000156	032714	004000	RLVEND:	RIT	#4000,(R4)
54	000162	001363			RNF	FORWARD
55						IND-WE HAVE TO TURN AROUND
56						
57						

I INITIATE READ/WRITE REQUEST

DT VM2-07 12-APR-74 RT-11 MACRO VM02-10 28-APR-75 16:04:24 PAGE 3+

58	000164	052705	010115		RIS	#10115,R5	I ASSUME WRITE
59	000170	012037	177346		MOV	(R0)+,0#TCRA	ICORE ADDRESS
60	000174	011016			MOV	(R0),(SP)	WORD COUNT (OVER BLOCK #)
61	000176	100404			RMT	IS	IWRITE WAS A GOOD GUESS
62	000200	001423			REN	RTDONE	IF ZERO, SEEK
63	000202	005416			NEG	(SP)	IF READ-NEGATE WORD COUNT
64	000204	042705	000010		RIF	#10,R5	IF SET READ FUNCTION
65	000210	012637	177344	IS:	MOV	(SP)+,0#TCWC	IF SET WORD COUNT
66	000214	000752			RR	RETRNI	
67							
68							
69	000216	032737	104000	177340	DTERR:	RIT	#104000,0#TCST
70	000224	100003			RPI	NOTE7	IEND7 ERROR?
71	000226	032714	000002		RIT	#2,(R4)	INT END7
72	000232	001346			RNF	ENDZR	IWERE WE SEARCHING?
73	000234	005367	177556		NOTE7:	DEF	IF YES-REVERSE TAPE
74	000240	003017			RGT	RETRY	IMORE TRIES LEFT?
75	000242	052770	000001	177772	RIS	#HDERR,0-6(R0)	IF YES
76							IND-SET HARD ERROR BIT
77							
78							
79	000250	005726			RTDONE:	TST	(SP)+
80	000252	012600			MOV	(SP)+,R0	IF POP BLOCK
81	000254	112737	000011	177342	DTSTOP:	MOV	#11,0#TCCM
82	000262	010704			MOV	PC,R4	IF STORE R0
83	000264	062704	177524		ADD	#DTCOE-,R0	IF STOP SELECTED DRIVE
84	000270	013705	000054		MOV	#MONLOW,R5	
85	000274	000175	000270		JMP	#OFFSET(R5)	IF AND OF ONE IN R4
86							
87							
88							
89	000300	105737	177340		RETRY:	TSTB	#*TCST
90	000304	100710			RMT	FORW1	IF TAPE UP TO SPEED?
91	000306	062767	000004	000002	ADD	#4,BWANT	IF YES-AVOTO STOPPING TAPE
92	000314	022716			RMP	(PC)+,(SP)	IND-IT TAKES 4 BLOCKS TO START AND STOP
							IF MAKE AN ATTEMPT TO START IN THE

93 000316 000000
 94 000320 000674
 95
 96 000322 000000
 97
 98 000324
 99
 100 000001

RWANT: 0
 RR DIRECT
 RIGHT DIRECTION BASED ON LAST BLOCK
 DEFSTRED
 \$INPTR: .WORD \$INTFN
 DT\$ITE = .-REG
 \$SIZE OF DT HANDLER
 .END

DT V02-07 12-APR-74 RT-11 MACRO VM02-10 28-APR-75 16:04:24 PAGE 3+
 SYMBOL TABLE

REG	000000R	002	RLKFND	000156R	002	RWANT	000316R	002	DIRECT	000112R	002	DP\$YS	= 000000 G	
NS\$YS	= 000000 G		DTGCF	000010R	002	DTDONE	000250R	002	DTFRP	000216R	002	DTINT	000034RG	002
DTLOF	000006R	002	DT\$ITE	000324		DTSTOP	000254R	002	DT\$YS	000006RG	002	DTTRY	000016R	002
DX\$YS	= 000000 G		FNDZR	000150R	002	FORWAR	000132R	002	FORW1	000126R	002	WDFRP	= 000001	
MONLOW	000054		NOTE7	000234R	002	OFFSFT	000270		PC	= %000007		PR7	= 000340	
PS	= 177776		PETRN1	000142R	002	PETRY	000300R	002	REVERS	000114R	002	RFSYS	= 000000 G	
RK\$YS	= 000000 G		R0	= %000000		R1	= %000001		R2	= %000002		R3	= %000003	
R4	= %000004		R5	= %000005		RP	= %000006		TCRA	= 177346		TCRM	= 177342	
TCNT	= 177350		TCST	= 177340		TCVEC	= 000214		TCWC	= 177344		\$INPTR	000322RG	002
\$INTFN	= ***** G													
.ABS.	000000	000												
	000000	001												
SYSHND	000324	002												
ERRORS DETECTED:	0													
FREE CORR:	14078. WORDS													
.LP:/N:TTM/C=DT														

APPENDIX B

FOREGROUND TERMINAL HANDLER

The following listing is a terminal handler for the foreground. The user can write his own handler using this code as an example, or use the copy provided in the software kit. Instructions for its use are found on the second and third pages of the listing.

```

1
2
3      .TITLE KB.MAC V01-01
4      ; RT-11 V2 DEVICE INDEPENDENT TERMINAL HANDLER, KB.
5      ;
6      ; DEC-11-ORPRA-D
7      ;
8      ; COPYRIGHT (C) 1975
9      ;
10     ; DIGITAL EQUIPMENT CORPORATION
11     ; MAYNARD, MASSACHUSETTS 01754
12     ;
13     ; THIS SOFTWARE IS FURNISHED UNDER A LICENSE FOR USE ONLY
14     ; ON A SINGLE COMPUTER SYSTEM AND MAY BE COPIED ONLY WITH
15     ; THE INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE,
16     ; OR ANY OTHER COPIES THEREOF, MAY NOT BE PROVIDED OR OTHERWISE MADE
17     ; AVAILABLE TO ANY OTHER PERSON EXCEPT FOR USE ON SUCH SYSTEM AND TO
18     ; ONE WHO AGREES TO THESE LICENSE TERMS. TITLE TO AND OWNERSHIP OF THE
19     ; SOFTWARE SHALL AT ALL TIMES REMAIN IN DIGITAL.
20     ;
21     ; THE INFORMATION IN THIS DOCUMENT IS SUBJECT TO
22     ; CHANGE WITHOUT NOTICE AND SHOULD NOT BE CONSTRUED
23     ; AS A COMMITMENT BY DIGITAL EQUIPMENT CORPORATION.
24     ;
25     ; DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE
26     ; OR RELIABILITY OF ITS SOFTWARE ON EQUIPMENT
27     ; WHICH IS NOT SUPPLIED BY DIGITAL.
28     ;
29     ;      MARCH 1975
30     ;      RGB

```

```

1
2
3      ;RT-11 V2 DEVICE INDEPENDENT TERMINAL HANDLER, KB. KB
4      ;CAN BE USED BY EITHER THE FOREGROUND OR BACKGROUND (BUT NOT
5      ;BOTH SIMULTANEOUSLY) TO READ AND WRITE TO ANY DL-11A OR KL-11A
6      ;CONTROLLED TERMINAL.
7      ;
8      ;THIS HANDLER HAS THE FOLLOWING CHARACTERISTICS:
9      ;      1)CARRIAGE RETURN CAUSES THE REMAINDER
10     ;      OF THE INPUT BUFFER FOR THE CALLING READ REQUEST TO BE
11     ;      ZERO-FILLED,AND THE READ IS COMPLETED. THUS,THE HANDLER

```

```

11      TRANSFERS ONE LINE AT A TIME, NO MATTER HOW LONG THE
12      INPUT BUFFER IS FOR THE READ REQUEST; THE UNUSED PORTION
13      OF THE BUFFER IS ZERO-FILLED. CARRIAGE RETURN ECHOES
14      CARRIAGE RETURN, /LINE=FEED, AND INSERTS CR AND LF CHARACTERS
15      IN THE BUFFER IF THERE IS ROOM, ELSE ONLY CR IS PLACED IN THE BUFFER.
16      2)FORM FEED ECHOES 7 LINE FEEDS, AND INSERTS A FF CHARACTER IN
17      BUFFER.
18      3)RUBOUT ECHOES "\" AND DELETES THE LAST CHARACTER IN THE BUFFER.
19      IF THERE ARE NO CHARACTERS IN THE BUFFER, RUBOUT DOES NOT ECHO
20      AND IS IGNORED.
21      4)TAB ECHOES ENOUGH SPACES TO POSITION THE PRINT HEAD AT THE
22      NEXT TAB STOP, AND INSERTS A TAB CHARACTER IN THE BUFFER.
23      5)CTRL U ECHOES "U" AND ERASES THE CURRENT LINE.
24      6)CTRL Z ECHOES "Z" AND CAUSES THE HANDLER TO REPORT END-OF-FILE.
25      THE CTRL Z CHARACTER IS NOT INSERTED IN THE BUFFER.
26      7)THE LOW-SPEED READER WILL RUN IF IT IS TURNED ON WHILE A READ
27      REQUEST IS PENDING TO THE HANDLER. IF THE TAPE BEING READ HAS
28      MANY TABS, HOWEVER, THE TIME NECESSARY TO ECHO THE TABS WILL
29      CAUSE CHARACTERS FOLLOWING THE TABS TO BE LOST. TO DISABLE THE
30      ECHOING OF TABS, THE "SET" COMMAND CAN BE USED AS FOLLOWS:
31      "SET KR LSR" WILL DISABLE TAB ECHOING, ALLOWING A TAPE
32      TO BE READ WITHOUT CHARACTER LOSS.
33      "SET KR NO LSR" WILL ENABLE TAB ECHOING, FOR NORMAL KEYBOARD
34      INPUT. THIS IS THE DEFAULT.
35      8)WHEN THE HANDLER RECEIVES A READ REQUEST, A "P" CHARACTER IS
36      PRINTED IN THE LEFT MARGIN OF THE TERMINAL TO SIGNIFY THAT THE
37      HANDLER IS READY FOR INPUT. THIS CHARACTER CAN BE CHANGED, OR THE
38      PROMPT FEATURE CAN BE REMOVED, BY RE-ASSIGNING THE SYMBOL
39      "PROMPT" TO THE ASCII VALUE OF THE
40      DESIRED CHARACTER. SETTING PROMPT TO "0" WILL CAUSE NO CHARACTER
41      TO BE PRINTED.
42      9)IF NO READ REQUEST IS ACTIVE, THE HANDLER WILL NOT ACCEPT INPUT,
43      AND THE KEYBOARD WILL NOT ECHO. IF IT DOES ECHO, THE HANDLER IS
44      ACCEPTING INPUT.
45
46      THIS HANDLER CONTAINS CONDITIONAL CODE TO SUPPORT TERMINALS THAT
47      REQUIRE FILLER CHARACTERS AFTER A PARTICULAR CHARACTER. TO ENABLE THE
48      FILLER FUNCTION, DEFINE THE SYMBOL "FILCHR" EQUAL TO THE ASCII
49      VALUE FOR THE CHARACTER TO BE FILLED AFTER, AND THE SYMBOL "FILCNT"
50      TO BE THE OCTAL NUMBER OF NULLS TO BE ISSUED AFTER EACH OCCURRENCE
51      OF THE CHARACTER DEFINED BY "FILCHR". FOR EXAMPLE, TO PROVIDE
52      12 FILLER CHARACTERS AFTER A CARRIAGE RETURN, SET "FILCHR=15" AND
53      "FILCNT=14".

```

```

1  )THE HANDLER IS INSTALLED VIA THE FOLLOWING PROCEDURE:
2  )
3  ) 1)ASSEMBLE IT AS FOLLOWS:
4  )   DEFINE FILLER CONDITIONALS IF NECESSARY
5  )   .R MACRO
6  )   *KB=KB
7  )
8  ) 2)LINK IT AS FOLLOWS:
9  )   .R LINK
10  )   *KB,SYS=KB
11  )
12  ) 3)INSTALL IT AS DEVICE "KB",AS DESCRIBED IN SECTION XXXXXX
13  )   OF THE RT-11 V2 SOFTWARE SUPPORT MANUAL. REMEMBER THAT
14  )   THE VECTORS FOR THE TERMINAL MUST BE PROTECTED IN THE BIT MAP
15  )   AS DESCRIBED IN THAT SECTION.
16  )   THE VALUES FOR THE VARIOUS TABLE ENTRIES SHOULD BE
17  )       HSIZE=VALUE OF SYMBOL "KRSTZ" ON LAST LINE OF LISTING
18  )       DVSZ=0 (NON-FILE STRUCTURED DEVICE)
19  )       PNAME=42420 (RAD40 FOR "KB ")
20  )       STAT= HIGH ORDER BYTE=0,LOW ORDER BYTE=ANY DEVICE NUMBER
21  )       AVAILABLE. NOTE THAT IT CANNOT BE 4. A VALUE >50
22  )       IS RECOMMENDED.
23  ) 4)ONCE INSTALLED, KB: WILL BE AVAILABLE WHEN THE SYSTEM IS REBOOTED.
24  )
25  )THE HANDLER ITSELF IS ACTIVATED WITH READ AND WRITE REQUESTS,AS ARE ALL
26  )RT-11 DEVICE HANDLERS. WHEN USING SYSTEM PROGRAMS WHICH OPERATE ON
27  )LARGE BUFFERS,SEVERAL LINES MAY ACCUMULATE IN THE BUFFER BEFORE
28  )THEY APPEAR ON THE TERMINAL,AND THEN ALL AT ONCE. TO AVOID THIS PROBLEM,
29  )EACH OUTPUT BUFFER CAN BE ZERO-FILLED AND SENT TO THE TERMINAL TO PRINT
30  )EACH LINE-THE HANDLER WILL IGNORE NULLS ON OUTPUT.
31  )IN FORTRAN,EACH LINE CAN BE FORCED IN OR OUT BY USING A REWIND
32  )FOLLOWING EACH READ OR WRITE TO THE DEVICE. FOR EXAMPLE:
33  )
34  )   LOGICAL*1 INPUT(80)
35  )   CALL ASSIGN (7,"KB/C")
36  )   WRITE (7,1)
37  )   REWIND 7
38  )   WRITE (7,2)
39  )   REWIND 7
40  )   READ (7,3) INPUT
41  )   REWIND 7
42  )
43  )
44  )
45  )
46  )   FORMAT . . . .
47  )   FORMAT . . .
48  )   FORMAT . . .
49  )

```

THE HANDLER CAN BE "RE-CONFIGURED" FOR VARIOUS VECTOR AND REGISTER ADDRESSES BY CHANGING THE ASSIGNMENTS OF THE SYMBOLS "KBVEC" AND "KBOSR" ON THE FOLLOWING PAGE, EDITING THESE TWO ISUFFICES TO CHANGE ALL FLOATING ADDRESSES.

B-5

```

1
2
3 000000      ,MCALL 'RPGDEF','.V2'...INTEN
4 000000      ,REGDEF
5              ,.V2..
6
7      000300      IVECTOR AND DEVICE REGISTER ADDRESSES-EDIT THESE TWO TO RECONFIGURE
8      174500      KBVEC=300      IKEYBOARD VECTOR,
9                      KBCSR=174500      IKEYBOARD CONTROL REGISTER
10
11      000304      IOTHER DEVICE ADDRESSES
12      174502      TPVEC=KBVEC+4      IPRINTER VECTOR
13      174504      KBRUF=KBCSR+2      IKEYBOARD BUFFER REGISTER
14      174506      TPCSR=KBCSR+4      IPRINTER CONTROL REGISTER
15                      TPBUF=KBCSR+6      IPRINTER BUFFER
16
17      000270      ICONSTANTS
18                      OFFSET=270      IOFFSET TO MONITOR COMPLETION ENTRY
19
20      020000      EOP=20000      IEOF BIT IN CSW
21      000340      PR7=340      IPSW VALUE FOR PRIORITY 7
22      000200      PR4=200      IPSW VALUE FOR PRIORITY 4
23
24      000011      HT=11      ITAB
25      000012      LP=12      ILYNE FEED
26      000014      FF=14      IFORM FEED
27      000015      CR=15      ICARRIAGE RETURN=15
28      000025      CTRLU=25      ICTRL/U
29      000032      CTRLZ=32      ICTRL/Z
30      000040      SPACE=40      ISPACE
31      000177      DELETE=177      IRIUBOUT
32
33      000024      EBLENGTH=20.      ILENGTH OF ECHO BUFFER
34
35      000076      PROMPT='>'      IPROMPT CHARACTER
36
37      ISET LSR CODE
38      ITHE FOLLOWING IS THE HANDLER INTERFACE TO THE MONITOR SET_COMMAND.
39      IFOR DETAILS OF INTERFACING TO THE SPT COMMAND,SEE THE RT-11 V2 SOFTWARE
40      ISUPPORT MANUAL
41      ,ASECT
42      ,=400
43      HT
44      ,RAD50 /LSR /      IFOR NOLSR,SET LSROPT TO "HT"
45      ,WORD 40PLSR=400>/2+100000
46      0
47      OPLSR: MOV (PC)+,R3      IFOR LSR,SPT LSROPT TO 377
48      377
49      MOV R3,LSROPT      IMODIFY OPTION VARIABLE IN HANDLER
50      RTS PC      IRETURN TO SET PROCESSOR
51
52      ,CSECT

```

```

1
2
3 000000 000304
4 000002 000116
5 000004 000340
6 000006 000000
7 000010 000000
8
9
10
11
12
13
14
15
16
17
18 000012 010700
19 000014 062700 000246
20 000020 010037 000300
21 000024 012737 000340 000302
22 000032 004067 000702
23 000036 016705 177746
24 000042 022525
25 000044 011567 000672
26 000050 012567 000670
27 000054 006315
28 000056 011567 000654
29 000062 001466
30 000064 103420
31 000066 011504
32 000070 014305
33 000072 105025
34 000074 004304
35 000076 001375
36 000100 104267 000634
37 000104 004167 000446
38 000110 076 377
39 000112 000167 000440

;THIS IS THE HANDLER HEADER AREA,USED BY PETCH AND THE
;QUEUE MANAGER TO STORE VARIABLES CRITICAL TO HANDLER OPERATION.
KBSTRT: .WORD TPVEC ;PRINTER VECTOR ADDRESS
        .WORD TPINT-. ;OFFSET TO PRINTER INTERRUPT SERVICE
        .WORD PR7
KBLQE: .WORD 0 ;LAST QUEUE ENTRY
KBCQE: .WORD 0 ;CURRENT QUEUE ENTRY

;FOLLOWING IS THE TRANSFER INITIATION CODE.
;THE FIRST WORD OF THIS ROUTINE IS THE ENTRY POINT FOR ALL
;TRANSFER REQUESTS. THE KEYBOARD VECTOR IS SET UP (PETCH ONLY SETS UP THE
;PRINTER VECTOR),AND THE PARAMETERS FOR THE TRANSFER ARE ESTABLISHED.
;IF THE REQUEST IS A WRITE, CONTROL TRANSFERS TO THE PRINTER ROUTINE TO
;OUTPUT THE FIRST CHARACTER FROM THE USER BUFFER. IF IT IS A READ,
;THE ENTIRE USER BUFFER IS ZEROED,A FLAG (READFL) IS SET TO
;SHOW THAT A READ IS IN PROGRESS,AND A PROMPT CHARACTER IS ECHOED
;ON THE TERMINAL BEFORE THE KEYBOARD INTERRUPT IS ENABLED.

MOV PC,R0
ADD #KBINT-.,R0 ;CALCULATE ABSOLUTE ADDRESS OF KEYBOARD INTERRUPT SERVICE
MOV R0,#KBVEC ;SET UP KEYBOARD VECTOR
MOV #PR7,#KBVPC+2
RETRY: CLR READFL ;INIT READ FLAG AND TAB COUNT
        MOV KBCQE,R5 ;POINT TO CURRENT Q ELEMENT
        CMP (R5)+,(R5)+ ;ADD 4 TO R5
        MOV (R5),URPTR ;SET UP POINTER TO USER BUFFER
        MOV (R5)+,UBPTR1 ;AND SAVE ORIGINAL POINTER FOR LATER
        ASL (R5) ;MAKE WORD COUNT INTO BYTE COUNT
        MOV (R5),BYTCNT ;AND SAVE IT
        REQ DONE ;WORD COUNT OF 0 IS SEEK,WHICH IS NOP IN THIS HANDLER
        RCR TPINT2 ;IF NEGATIVE,WRITE TO PRINTER
        MOV (R5),R4 ;BYTE COUNT TO R4
        MOV -(R5),R5 ;USER BUFFER POINTER IN R5
        JS: CLR R5 ;ZERO USER BUFFER BEFORE STARTING TRANSFER
        DEC R4
        BNF 33 ;BRANCH IF NOT DONE
        INCB READFL ;SET "READ IN PROGRESS" FLAG
        JSR R1,ECHO ;PROMPT INPUT WITH ">"
        .BYTE PROMPT,377
        JMP KBTN ;ENABLE KEYBOARD INTERRUPT AND RETURN

```

```

1      ;THIS IS THE ABORT ENTRY POINT-THE HANDLER IS ENTERED AT THIS ADDRESS
2      ;IF THE MONITOR RECIEVES A REQUEST TO ABORT ANY IO TRANSFER IN PROGRESS
3      000116 000446      RR      ABORT
4
5      ;THIS IS THE TERMINAL OUTPUT INTERRUPT SERVICE. AFTER ENTERING SYSTEM STATE,
6      ;IT DETERMINES IF THERE ARE ANY CHARACTERS IN THE ECHO BUFFER TO BE
7      ;PRINTED. IF NOT,IT THEN DETERMINES WHETHER A WRITE REQUEST IS IN PROGRESS
8      ;OR NOT. IF SO,THE NEXT CHARACTER IN THE USER BUFFER IS PRINTED.
9      ;IF NOT,THE INTERRUPT IS DISMISSED.
10     ;IF THERE ARE CHARACTERS IN THE ECHO BUFFER,THE FIRST CHARACTER IN THE
11     ;LIST IS FETCHED INTO R4,THE LIST IN THE ECHO BUFFER IS "SLID UP"
12     ;BY ONE CHARACTER,AND THE CHARACTER IN R4 IS THEN PRINTED.
13     ;IF THE FILLER CONDITIONAL CODE IS INCLUDED AT ASSEMBLY TIME,
14     ;THE CHARACTER IN R4 IS COMPARED AGAINST THE CHARACTER TO BE FILLED AFTER.
15     ;IF THE SAME,A COUNT OF NECESSARY FILLS IS STUFFED IN "FILLCN" AND THE
16     ;CHARACTER IS PRINTED. THE INTERRUPT SERVICE THEN CHECKS THE NUMBER
17     ;OF FILLS NEEDED AS THE FIRST ITEM,AND PRINTS NULLS IF ANY ARE LEFT
18     000120 004577 000622      TPNT1: JSR      R5,0SINPTR      ;ENTER SYSTEM STATE
19     000124 000140      ;.WORD      *C<PR4>BPR7
20     000126 003737 000200 176504      TPNT2: RIT      #200,0#TPCSR      ;IS THE PRINTER READY
21     000134 001436      REG      RTSPC      ;YES-THEN WAIT FOR INTERRUPT TO PRINT ANYTHING
22
23     ;IFDF      FILLCHR      ;CONDITIONAL CODE FOR FILLER
24     TSTB      FILLCN1      ;ANY FILLS NEED TO BE OUTPUT?
25     RLE      38      ;BRANCH IF NOT
26     DECB      FILLCN1      ;YES-DECREASE NUMBER BY ONE
27     CLR      R4      ;NULL IS FILLER
28     BR      TPNT3      ;GO PRINT IT
29     .ENDC
30     000136 010705      38i      MOV      PC,R5      ;CALC ABSOLUTE ADDRESS
31     000140 062705 000550      ADD      @RRSTRY-1,R5      ;OF ECHO BUFFER
32     000144 111504      MOV      (R5),R4      ;GET CHAR TO ECHO FROM ECHO BUFFER
33     000146 001410      REG      18      ;BRANCH IF BUFFER EMPTY
34     000150 012746 000024      MOV      @ELENGTH-1(SP)      ;NUMBER OF CHARS IN ECHO BUFFER ON STACK
35     000154 114525 000001      28i      MOV      1(R5),(R5)+      ;SLIDE ECHO LIST UP
36     000160 004316      DEC      (SP)      ;DECREASE COUNT OF CHARS TO SLIDE
37     000162 003374      BGT      #8      ;BRANCH IF NOT FINISHED
38     000164 004726      TST      (SP)+      ;DONE-CLEAN UP STACK
39     000166 000412      BR      TPNT1      ;AND PRINT CHAR
40
41     000170 104767 000544      18i      TSTB      READFL      ;ARE WE READING OR WRITING?
42     000174 001016      RNE      RTSPC      ;BRANCH IF READING
43     000176 117704 000540      TPNT1: MOV      @URPTR,R4      ;GET CHAR FROM USER BUFFER INTO R4
44     000202 004267 000534      INC      UBPTR      ;BUMP BUFFER POINTER
45     000206 004267 000524      INC      RYCNT      ;AND DECREASE TRANSFER COUNT
46     000212 003012      RGT      DONE      ;BRANCH IF TRANSFER COMPLETE
47     000214 104704      TPNT1: TSTB      R4      ;DON'T PRINT NULLS

```

```

48 000216 001743          REG      TPOUT2          IBRANCH IF NULL
49          IFDF          FILCHR          ICONDITONAL CODE FOR FILLER
50          CMPB          R4,FILCR1        IDGES THIS CHAR NEED TO BE FILLED AFTER?
51          ANF          TPOUT3          IBRANCH IF NOT
52          MOVB          #FILCNT,FILCN1    IYFS-SET UP COUNT OF FILLS NEEDED
53          ENDC
54 000220 012737 000100 176504 TPOUT3: MOV      #100,#TPCSR          IENABLE PRINTER INTERRUPT
55 000226 110437 176506          MOVB      R4,#TPBUF          IPRINT CHARACTER
56 000232 000207          RTSPC: RTS      PC          IRETURN TO MONITOR
57

```

KB,MAC V01-01 RT-11 MACRO VM02-09 8-APR-75 12:33:51 PAGE 64

```

58          IREQUEST TERMINATION AND ABORT CODE
59          ITHIS ROUTINE IS ENTERED WHEN THE I/O TRANSFER IS
60          ICOMPLETED OR ABORTED. THE DEVICE INTERRUPTS ARE DISABLED, AND
61          ISTANDARD MONITOR COMPLETION ENTRY CODE IS EXECUTED.
62 000234 005037 176504          ABORT: CLR      #TPCSR          IDISABLE OUTPUT INTERRUPTS
63 000240 005037 176500          DONE: CLR      #KBCSR          IDISABLE INPUT INTERRUPTS
64 000244 010704          MOV      PC,R4          ISTANDARD MONITOR
65 000246 062704 177542          ADD      #KACGE-.,R4          ICOMPLETION ENTRY
66 000252 013705 000054          MOV      #4,R5          ICODE
67 000256 000175 000270          JMP      #OFFSET(R5)

```

KB,MAC V01-01 RT-11 MACRO VM02-09 8-APR-75 12:33:51 PAGE 7

```

1          IKEYBOARD INTERRUPT SERVICE
2          ITHIS IS THE KEYBOARD INTERRUPT SERVICE ROUTINE. AFTER ENTERING
3          ISYSTEM STATE, IT GETS THE TYPED CHARACTER INTO R4, THEN
4          IPROCEEDS DOWN A CHAIN OF CHECKS FOR THE SPECIAL CASE CHARACTERS
5          I(RUBOUT, CTRL U, CTRL Z, CR, FF). IF IT IS ONE OF THE SPECIAL
6          ICHARACTERS, THE ROUTINE "ECHO" IS CALLED TO ECHO APPROPRIATE
7          ICHARACTERS ON THE TERMINAL, THEN APPROPRIATE ACTION FOR THE SPECIAL CASE
8          IIS TAKEN. IF A NORMAL CHARACTER IS TYPED, IT IS ECHOED AND PLACED
9          IIN THE USER BUFFER BEFORE THE INTERRUPT IS DISMISSED.
10
11 000262 004577 000460          KBINT: JSR      R5,#SINPTR          IENTER SYSTEM STATE
12 000266 000140          WORD      C4PR4>SPR7
13 000270 113704 176502          MOVB      #KBRUF,R4          IGET CHAR
14 000274 042704 177600          BIC      #177600,R4          ISTRIP TO SEVEN BITS
15 000300 120427 000177          CMPB      R4,#DELETE          IIS THIS CHARACTER A RUBOUT?
16 000304 001020          ANF          IIS
17 000306 026767 000430 000430          CMP      UBPTR,UBPTR1          IANY CHARS LEFT TO RUB OUT?

```

18	000314	001520			REQ	KBTN	INO=IGNORE RUBOUT
19	000316	004367	000420		DEC	UBPTR	IBACK UP POINTER INTO USER BUFFER
20	000322	004167	000240		JSR	R1,ECHO	
21	000326	134	377		.BYTE	'\,377	
22	000330	104267	000405		INCB	TARCNT	IBUMP TAR COUNTER FOR "\"
23	000334	104077	000402		CLRB	UBPTR	IZERO RUBBED OUT CHAR
24	000340	004267	000372		INC	RYTCNT	IAND INCREASE TRANSFER COUNT TO REFLECT LOST CHAR
25	000344	000504			RR	KBTN	IRE=ENABLE INTERRUPTS AND EXIT
26	000346	120477	000014	1141	CMPB	R4,0FF	IS THIS CHAR A FORM FEED?
27	000352	001006			RNE	AS	IBRANCH IF NOT
28	000354	004167	000206		JSR	R1,ECHO	IYES=ECHO 7 LINE FEEDS
29	000360	012	012	012	.BYTE	LF,LF,LF,LF,LF,LF,LF,LF,377	
	000363	012	012	012			
	000366	012	377				
30	000370	120427	000015	651	CMPB	R4,#CR	IS THIS CHAR A CR?
31	000374	001017			RNE	78	IBRANCH IF NOT
32	000376	004167	000164		JSR	R1,ECHO	IYES=ECHO CR,LF
33	000402	015	012	000	.BYTE	CR,LF,0,377	
	000405	377					
34	000406	110477	000330		MOVB	R4,UBPTR	IPUT CR IN USER BUFFER
35	000412	004267	000324		INC	UBPTR	IBUMP USER BUFFER POINTER
36	000416	004367	000314		DEC	RYTCNT	IROOM IN BUFFER FOR LF TOO?
37	000422	001706			REQ	DONE	IDON'T INSERT IT IF NOT
38	000424	110777	000012	000310	MOVB	#LF,UBPTR	ELSE ADD LF TO BUFFER
39	000432	000702			RR	DONE	

KB,MAC V01-01 RT-11 MACRO VM02-09 8-APR-75 12:33:51 PAGE 8

1	000434	120427	000025	751	CMPB	R4,#CTRLU	IS CHAR CTRL U?
2	000440	001007			RNE	83	IBRANCH IF NOT
3	000442	004167	000120		JSR	R1,ECHO	IECHO "U"
4	000446	136	135	015	.BYTE	'U,CR,LF,0,377	
	000451	012	000	377			
5	000454	000167	177352		JMP	RETRY	IAND RESTART READ
6							
7	000460	120427	000032	851	CMPB	R4,#CTRLZ	IS CHAR CTRL Z?
8	000464	001013			RNE	93	IBRANCH IF NOT
9	000466	004167	000074		JSR	R1,ECHO	IECHO "Z"
10	000472	136	132	015	.BYTE	'Z,CR,LF,0,377	
	000475	012	000	377			
11	000500	016705	177304		MOV	KBCQE,R5	IPPOINT R5 TO Q ELEMENT
12	000504	052775	020000	177776	RIS	#EOF,0-2(R5)	IAND SET EOF FLAG IN CSW
13	000512	000652			RR	DONE	ISTOP TRANSFER
14							
15	000514	120427	000040	951	CMPB	R4,#40	IS THIS A PRINTING CHAR?
16	000520	002402			RLT	213	IBRANCH IF NOT
17	000522	104267	000213		INCB	TARCNT	IY8-INCREASE TAR POSITION

18	000526	110467	000004	2101	MOVB	R4,208	ISRT UP TO ECHO CHAR
19	000532	004167	000030		JSR	R1,ECHO	
20	000536	000	377	2001	.BYTE	0,377	
21	000540	110477	000176		MOVB	R4,0UBPTR	INPUT CHAR IN USER BUFFER
22	000544	004267	000172		TNC	UBPTR	BUMP BUFFER POINTER
23	000550	004367	000162		DEC	RYCNT	ANY MORE TO TRANSFER
24	000554	001631			REQ	DONE	BRANCH IF NOT
25	000556	012737	000101 176500	KBTN1	MOV	#101,0#KBCAR	ENABLE KEYBOARD INTERRUPT
26	000564	000207			RTS	PC	RETURN TO MONITOR

KB,MAC V01-01 RT-11 MACRO VM02-09 A-APR-75 1P133151 PAGE 9

1							
2							
3						ISUBROUTINE ECHO	
4						THIS SUBROUTINE SERVES TO PLACE THE SPECIFIED CHARACTERS IN THE	
5						ECHO BUFFER, AND START THE PRINTER IN CASE IT IS IDLE.	
6						THE CALLING SEQUENCE IS	
7						JSR R1,ECHO	
8						.BYTE CHAR1,CHAR2,CHAR3,...,CHARN,377	
9						ON ENTRY, R4 CONTAINS THE CHAR TYPED AT THE KEYBOARD.	
10						NOTE THAT THERE MUST BE AN EVEN NUMBER OF BYTES IN THE ARGUMENT LIST	
11						AND THEREFORE THE NUMBER OF CHARACTERS EXCLUDING THE 377	
12						MUST BE ODD.	
13						WHEN ENTERED, ECHO SCANS THE ECHO BUFFER TO FIND THE END OF THE	
14						ECHO LIST, WHICH IS MARKED BY A NULL BYTE. WHEN THE END OF THE LIST	
15						IS FOUND, IT IS DETERMINED IF THERE ARE AT LEAST 8 FREE SLOTS IN THE LIST	
16						TO ACCOMMODATE A POSSIBLE LINE FEED OR FORM FEED. IF NOT, THE	
17						CHARACTER JUST TYPED IS IGNORED. IF SO, THE CHARACTERS FROM THE	
18						ARGUMENT LIST FOLLOWING THE CALL ARE INSERTED IN THE BUFFER.	
19						THE PRINTER IS STARTED IF IT IS IDLE, AND THE ROUTINE RETURNS.	
20						NOTE THAT TAB IS A SPECIAL CASE: IF R4 CONTAINS A TAB CHARACTER	
21						WHEN THIS ROUTINE IS ENTERED, THE ARGUMENT LIST IS NOT USED. RATHER,	
22						AN APPROPRIATE NUMBER OF SPACES TO MOVE THE PRINT HEAD TO THE	
23						NEXT TAB STOP ARE PLACED IN THE ECHO BUFFER, AND THE ROUTINE RETURNS	
24	000566	010705			ECHO:	LEA R1, PC, R5	ICALL ABSOLUTE ADDRESS
25	000570	062705	000120		ADD	#R1, R5	OF ECHO BUFFER
26	000574	010567	000134		MOV	R5, TEMP	SAVE ADDRESS OF ECHO BUFFER
27	000600	062767	000023 000126		ADD	#ELENGTH-1, TEMP	TEMP POINTS TO END OF ECHO BUFFER
28	000606	105725		45:	TSB	(R5)+	IS THIS END OF ECHO LIST?
29	000610	001376			RNE	48	BRANCH IF NOT
30	000612	004305			DEC	R5	YES=R5 POINTS TO FIRST FREE SLOT IN ECHO LIST
31	000614	160567	000114		SUB	R5, TEMP	FIND NUMBER OF FREE SLOTS IN ECHO LIST
32	000620	024727	000110 000010		CMR	TEMP, #8	IS THERE ENOUGH ROOM TO ECHO TAB OR FF?
33	000626	004002			RGY	38	BRANCH IF YES
34	000630	012601			MOV	(R5)+, R1	IGNORE THIS CHAR THEN
35	000632	000751			RR	KBIN	DISMISS INTERRUPT

KB.MAC V01-01 RT-11 MACRO VM02-00 8-APR-75 12133151 PAGE 10+
SYMBOL TABLE

ABORT	000234R	RYTCNT	000736R	CR	= 000015	CTRLU	= 000025	CTRLZ	= 000032
DELET	= 000177	DONE	000240R	FBLENG	= 000024	ECHO	000566R	FOF	= 020000
FF	= 000014	HT	= 000011	KBAUF	= 176502	KBCQE	000010R	KBC8R	= 176500
KBTN	000556R	KBTNT	000262R	KBLGF	000006R	KBSIZE	= 000750	KBSTRT	000000R
KBVEC	= 000300	LF	= 000012	LSROPT	000640R	OFFSET	= 000270	OPL8R	000412
PC	=X000007	PROMPT	= 000076	PR4	= 000200	PR7	= 000340	RBSTRT	000710R
READFL	000740R	RETRY	000032R	RTSPC	000232R	R0	=X000000	R1	=X000001
R2	=X000002	R3	=X000003	R4	=X000004	R5	=X000005	SP	=X000006
SPACE	= 000040	TARCNT	000741R	TEMP	000734R	TPAUF	= 176506	TPC8R	= 176504
TPINT	000120R	TPOUT	000176R	TPOUT1	000214R	TPOUT2	000126R	TPOUT3	000220R
TPVEC	= 000304	UBPTR	000742R	UBPTR1	000744R	SINPTR	000746R	...V2	= 000001

. ABS. 000424 000
000750 001
ERRORS DETECTED: 0
FREE CORE: 15460. WORDS

KB,LP1/NITM/C=KR

APPENDIX C

VERSION 1 EMT SUMMARY

Although Version 1 programmed requests are supported by Versions 2, 2B, and 2C of RT-11, it is strongly recommended that the Version 1 formats not be used. For purposes of compatibility, however, this section provides a brief review of the V1 format. The V2/V2B/V2C format is covered in detail in Chapter 9 of the RT-11 System Reference Manual.

In brief, the major distinctions between V1 and V2/V2B formats are:

1. V1 format has arguments pushed on the stack and in R0. V2/V2B/V2C requests generally accept a set of arguments, or an argument in R0.
2. V1 channel numbers are restricted to 16_{10} . Also, the channel number in V1 is not a legal assembler argument; it is merely an integer in the range 0 to 15_{10} .
3. V1 requests are non-reentrant because the channel number and function code are embedded within the EMT instruction.

Table C-1 lists all the Version 1 macro calls. Those in the left column have the same format as the corresponding Version 2/2B/2C request; those in the right column have a different format, shown after the table. The operations performed by the requests are the same in both versions.

Table C-1
V1 Programmed Requests

V1 - Format Same as V2/V2B	V1 - Format Different from V2/V2B/V2C
.CSIGEN .CSISPC .DATE .DSTAT .EXIT .FETCH	.CLOSE .DELETE .ENTER .LOOKUP .READ .READC

(continued on next page)

V1 - Format Same as V2/V2B	V1 - Format Different from V2/V2B/V2C
.HRESET .LOCK .PRINT .QSET .RCTRLO .RELEAS .SETTOP .SRESET .TTINR .TTOUTR .TTYIN .TTYOUT .UNLOCK	.READW .RENAME .REOPEN .SAVESTATUS .WAIT .WRITE .WRITC .WRITW

```
.blk      A block number specifying the relative block in a file
          where an I/O transfer is to begin.

.buff     A buffer address specifying a memory location into
          which or from which an I/O transfer is to be performed.

.cblk     The address of the five words of user memory where the
          channel status will be stored.

.chan     A channel number in the range 0-17 (octal).

.crt      The entry point of a completion routine.

.dblk     The address of the 4-word RAD50 file description
          (dev:file.ext).

.length   The number of blocks allocated to the file being opened.

.wcnt     A word count specifying the number of words to be trans-
          ferred to or from the buffer during an I/O operation.

.CLOSE .chan

.DELETE .chan,.dblk

.ENTRY .chan,.dblk,.length

.LOOKUP .chan,.dblk

.READ     {
.READC    {      .chan,.buff,.wcnt,.crt,.blk [ .crt is required
.READW    }      only for .READC ]
}

.RENAME .chan,.dblk

.REOPEN .chan,.cblk

.SAVESTATUS .chan,.cblk
```

```
.WAIT .chan
```

```
.WRITE }  
.WRITC } .chan,.buff,.wcnt,.crtm,.blk [ .crtm is required  
.WRITW } [ only for .WRITC ]
```

The system macro library (SYSMAC.SML) can be used with Versions 2 and 2B to generate Version 1 programmed requests.

Under Version 2, the ..V2.. macro is capable of handling V1 expansions. ..V2.. normally expands as:

```
.MCALL ...CM1,...CM2,...CM3,...CM4  
...V2=1
```

This causes Version 2 expansions in all cases. To allow expansion of all V1 requests in their V1 format (and all new Version 2 requests in V2 format) the ..V2.. macro should not be called, but the utility macros must still be defined:

```
.MCALL ...CM1,...CM2,...CM3,...CM4
```

Omitting both ..V2.. and the utility macros causes all old V1 requests to be expanded in V1 format; no V2 requests can be used.

Under Version 2B, the ..V1.. macro call enables expansion of all macros in Version 1 format. ..V1.. expands as:

```
...V1=1
```

To enable expansion of all Version 1 macros in V1 format and all new Version 2 macros in V2 format, these statements must be included:

```
.MCALL ..V1,...CM1,...CM2,...CM3,...CM4  
..V1..
```

A listing of SYSMAC.SML is provided in the RT-11 System Reference Manual.

APPENDIX D
FOREGROUND SPOOLER EXAMPLE

The following program is an example of a line printer spooler for the foreground. Instructions for its use follow.

1. Create the program using the Editor and store it on the system device under the name LSPPOOL.MAC.
2. Next assemble it under MACRO and then link it to create the REL format output file:

```
.R MACRO
*LSPPOOL=LSPPOOL
```

```
.R LINK
*LSPPOOL=LSPPOOL/R
```

3. Load the necessary handlers (in this case, LP and RF) and run the program. All files on device RF with the extension .LST are listed on the line printer and then deleted from RF:

```
.LOA LP,RF<CR>
```

```
.FRU LSPPOOL<CR>
```

```
F>  
DEVICE TO SPOOL?
```

```
B>
```

```
.
```

[Control must be redirected
to the foreground via ^F.]

```
F>  
RF:*.LST<CR>
```

This program assumes device DK: and extension .LPT unless otherwise indicated.

```

1          .TITLE  LSPDOL - LINE PRINTER SPOOLER
2          .SBTTL  A USEFUL FOREGROUND PROGRAM
3
4          ; THIS PROGRAM FOR THE FOREGROUND IS A LINE PRINTER SPOOLER.
5          ; IT SEARCHES A SPECIFIED DEVICE FOR FILES WITH A PARTICULAR
6          ; EXTENSION (THE DEFAULT IS LPT) AND PRINTS THEM, DELETING
7          ; AFTER PRINTING. IF NONE ARE FOUND, IT WILL GO TO SLEEP FOR
8          ; HALF A MINUTE, PERMITTING THE BACKGROUND TO RUN.
9          ;
10         ; TO RUN LSPDOL, FIRST LOAD LP HANDLER AND INPUT DEVICE HANDLER
11         ; IF IT IS NOT THE SYSTEM DEVICE TYPE.
12         ;
13         ; F.G.,
14         ;
15         ; .LOA LP,RF
16         ; .FRU LSPDOL
17         ;
18         ; LSPDOL WILL TYPE: "DEVICE TO SPOOL?"
19         ; TYPE INPUT DEVICE AND FILE DESCRIPTION, F.G.:
20         ;
21         ; RF:*.LST
22
23
24         .MCALL  ..V2...REGDEF
25         .MCALL  .READW,.WRITW,.LOOKUP,.DELETE,.CSISPC,.TTYIN
26         .MCALL  .PRINT,.TTYOUT,.SRESET,.PCTRLD,.CLOSE,.EXIT
27         .MCALL  .DSTATUS,.TWAIT
28
29         ..V2..
30         .REGDEF
31
32         000046      USRSPW  =      46      ;USR SWAP LOCATION POINTER
33         000052      FRPBYT  =      52      ;ERROR CODE
34         000015      CR      =      15      ;CARRIAGE RETURN
35         000012      LF      =      12      ;LINE FEED
36
37         000000 012737 001260' 000046  START:  MOV      #BUFF,#USRSPW  ;MAKE USR SWAP OVER BUFF
38         000006 010627              MOV      SP,(PC)+    ;SAVE STACK POINTER FOR RESET
39         000010 000000      STKSAV: .WORD      0
40         000012              .DSTATUS #TOR,#LP          ;MUST BE IN MEMORY.
41         000024 103403              PCS      15        ;ILLEGAL DEVICE
42         000026 005767 000750      TST      TOR+4      ;TEST ENTRY POINT
43         000032 001011              RNE      BEGIN      ;BR TO BEGIN IF LOADED
44         000034              1$:      .PRINT  #MSG0      ;LP NOT IN MEMORY!

```

```

45 000042          .EXIT          ;BACK TO USER FOR A LOAD LP
46
47          ; COME HERE ON BAD COMMAND STRING
48 000044  RADCOM: .PRINT  #MSG2      ;PRINT ERROR MESSAGE
49 000052  016706  177732      MOV    STKSAV,SP      ;RESET STACK, FALL THRU TO BEGIN
50
51 000056  BEGIN:  .CLOSE  #0          ;WE WILL USE CH 0, SO CLEAN IT UP
52 000070          .CTRLN          ;RESET CTRL/O FLAG SO
53 000072          .PRINT  #MSG1      ;PROMPTING MSG WILL PRINT.
54 000100  012702  001142'    MOV    #CSIRLK,R2      ;PRINT TO COMMAND STRING BUFFER
55 000104  010201          MOV    R2,R1      ;COPY THE POINTER AND INPUT COMMAND
56 000106          1$:  .TTYTN          ;A CHARACTER AT A TIME.
57 000112  022700  000015      CMP    #CR,R0      ;CARRIAGE RETURN?

```

LSPool - LINE PRINTER SPOOLER RT-11 MACRO VM02-11 26-NOV-75 00:07:21 PAGE 1+

A USEFUL FOREGROUND PROGRAM

```

58 000116 001773      REQ    1$          ;YES, IGNORE IT.
59 000120 110022      MOVB    R0,(R2)+      ;MOVE IT INTO BUFFER AND
60 000122 122700 000012  CMPB    #LF,R0      ;TEST FOR END OF LINE.
61 000126 001367      BNF     1$          ;NO, GET ANOTHER CHARACTER.
62 000130 105042      CLRB    -(R2)        ;YES, CLEAR OUT THE LINE FEED
63 000132          .CSISPC R1,#DEEXT,R1      ;PROCESS THE COMMAND
64 000144 012600      MOV     (SP)+,R0      ;TEST # OF SWITCHES;C BIT UNCHANGED.
65 000146 001336      BNE     RADCOM        ;NO SWITCHES ALLOWED
66 000150 103735      RCS     RADCOM        ;SYNTAX ERROR
67 000152          .LOOKUP #IOB,#1,#LP
68 000204 016700 000776  MOV     CSIBLK+36.,R0      ;GET FILE EXTENSION TO PRINT.
69 000210 001002      BNE     3$          ;BRANCH IF USER SPECIFIED,
70 000212 012700      MOV     (PC)+,R0      ;ELSE USE LPT EXTENSION
71 000214 046624      .RAD50 /LPT/
72 000216 010067 000244 3$:  MOV     R0,LPT...      ;SAVE THE EXTENSION FOR LATER.
73 000222 016700 000752  MOV     CSIBLK+30.,R0      ;GET THE INPUT DEVICE NAME
74 000226 001002      BNE     4$          ;BRANCH IF USER SPECIFIED,
75 000230 012700      MOV     (PC)+,R0      ;ELSE USE THE
76 000232 015326      .RAD50 /DK0/        ;DEFAULT DEVICE.
77 000234 010011 4$:  MOV     R0,R1      ;SET DEVICE NAME IN FILE
78 000236 005061 000002  CLR     2(R1)      ;DESCRIPTOR BLOCK, CLEARING
79          ;OUT ANY FILE NAME.
80 000242          .DSTATUS #TOR,R1      ;INPUT DEVICE HANDLER MUST BE RESIDENT
81 000252 103403      RCS     5$          ;ILLEGAL DEVICE
82 000254 005767 000522  TST     TOR+4      ;TEST ENTRY POINT
83 000260 001004      BNE     6$          ;BRANCH IF O.K.,
84 000262          5$:  .PRINT  #MSG3      ;ELSE PRINT MESSAGE
85 000270 000072      BR      BEGIN

```

```

86 000272
87 000320 103067
88
89 000322 012702 001260
90 000326 012703 000001
91 000332 004503
92 000334 062703 000004
93
94
95 000340
96 000402 103436
97 000404 010205
98 000406 062705 000012
99 000412 032715 004000
100 000416 001416
101 000420 014203 000002
102 000424 001342
103 000426
104 000452 000723
105 000454 032725 002000
106 000460 001404
107 000462 026527 000004
108 000466 000000
109 000470 001407
110 000472 062705 000014
111 000476 000745
112 000500
113 000506 000711
114

```

```

4S: .LOOKUP #IOB,#0,P1      ;OPEN CHANNEL TO READ DIRECTORY
      RCS      RADERR
      .ENARL LSR
FINDLP: MOV      #BUFF,R2      ;R2 -> BUFFER
      MOV      #1,R3          ;INIT R3
1S:    ASL      R3             ;MULTIPLY BY 2
      ADD      #4,R3          ;AND ADD 4 TO GET BLOCK NUMBER
                                ;OF DIRECTORY SEGMENT (STARTING
                                ;AT BLOCK 6 OF DEVICE).
      .READW   #IOB,#0,R2,#1000,R3 ;READ A DIRECTORY SEGMENT
      RCS      RADERR          ;ERROR READING DIRECTORY!!
      MOV      R2,R5          ;COPY POINTER
      ADD      #12,R5         ;MOVE PAST DIRECTORY HEADER
2S:    RIT      #0000,R5      ;TEST FOR END OF SEGMENT
      BEQ      3S             ;BRANCH IF MORE TO GO
      MOV      2(R2),R3       ;GET LINK TO NEXT SEGMENT
      BNE      1S             ;BRANCH IF ANOTHER SEGMENT EXISTS,
                                ;ELSE WAIT A WHILE
      .TWAIT   #IOB,#TIMBLK
      RR        FINDLP        ;WE'RE AWAKE, LOOK FOR A FILE
3S:    RIT      #2000,(R5)+    ;TEMPORARY FILE?
      BEQ      4S             ;YES, SKIP IT
      CMP      4(R5),(PC)+    ;DOES THE EXTENSION MATCH?
      LPT...: .WORD 0
      BEQ      COPIER         ;YES, GO PRINT IT.
4S:    ADD      #14,R5         ;NO, ADVANCE TO NEXT ENTRY
      RR        2S            ;AND GO LOOK AT IT.
RADERR: .PRINT  #MSG4         ;PRINT A MESSAGE
      RR        1S           ;THEN TRY NEXT SEGMENT
      .DSARL LSR

```

LSPPOOL - LINE PRINTER SPOOLER RT-11 MACRO VM02-11 26-NOV-75 00:07:21 PAGE 1+

A USEFUL FOREGROUND PROGRAM

```

115
116
117
118 000510 012561 000002
119 000514 012561 000004
120 000520 012561 000006
121 000524
122 000554 103642
123 000556 005005
124 000560
125 000624 103424
126 000626 010204

```

; THIS ROUTINE PRINTS THE SPOOLED FILE JUST FOUND AND THEN DELETES IT.

```

COPIER: MOV      (R5)+,2(R1)    ;COPY FILE NAME AND EXTENSION
      MOV      (R5)+,4(R1)    ;INTO FILE DESCRIPTOR BLOCK
      MOV      (R5)+,6(R1)    ;FOR A LOOKUP.
      .LOOKUP   #IOB,#2,R1    ;LOOKUP THE FILE ON CHANNEL 2.
      RCS      FINDLP        ;SOMETHING FUNNY HAPPENED
      CLR      R5             ;O.K., COPY IT, R5 IS BLOCK #
1S:    .READW   #IOB,#2,R2,#1000,R5 ;READ 1000 WORDS
      RCS      2S            ;ERROR ON READ
      MOV      R0,R4          ;COPY ACTUAL WORD COUNT TRANSFERRED

```

```

127 000630      .WRITW  #IOB,#1,R2,R4,R5 ;AND WRITE IT TO CHANNEL 1
128 000672 000725 TST      (R5)+      ;BUMP BLOCK # RY 2
129 000674 000731 RR        1$          ;CONTINUE UNTIL EOF.
130 000676 100737 000052 2$: TSTB     #ERRBYT ;WAS ERROR JUST AN EOF?
131 000702 001412 REQ        4$          ;YES.
132 000704      .CLOSE  #2            ;NO, CLOSE THE FILE AND
133 000716      .PRINT  #ERRIN        ;REPORT AN INPUT ERROR
134 000724 000167 177372 3$: JMP      FINDLP ;THEN FIND ANOTHER FILE.
135 000730 4$:      .CLOSE  #2            ;ON EOF, CLOSE THE FILE
136 000742      .DELETE #IOB,#3,R1    ;MUST DELETE USING AN INACTIVE CHANNEL
137 000772 000754 RR          3$          ;THEN CONTINUE
138
139 000774 044636 LP:      .RAD50 /LP0/    ;SPOOLER OUTPUT DEVICE
140 000776 TOR:      .BLKW  5           ;EMT ARGUMENT BLOCK
141
142 001010 MSG0:      .ASCII /NO LP/
143 001016 MSG1:      .ASCII /DEVICE TO SPOOL?/
144 001037 MSG2:      .ASCII /TRY AGAIN/
145 001051 MSG3:      .ASCII /DEVICE?/
146 001061 MSG4:      .ASCII /ERROR READING DIRECTORY/
147 001111 ERPIN:    .ASCII /INPUT ERROR/
148      .EVEN
149
150 001126 000700 001604      .LTST  BIN
151 001132 044624 TIMBLK: .WORD  0,60,*15.
152 001134 000700 000000 000000 DEFEXT: .RAD50 /LPT/
153 001142      .WORD  0,0,0          ;CSI DEFAULT EXTENSIONS
154 001260 CSTBLK: .BLKW  39.
155      .RUFF: .BLKR  10000          ;CSI WORK AREA
156      .END      START             ;BUFFER AREA

```

LSPPOOL = LINE PRINTER SPOOLER RT-11 MACRO VM02-11 26-NOV-75 00:07:21 PAGE 1+

```

SYMBOL TABLE
RADCOM 000044R RADERR 000500R BEGIN 000056R RUFF 001260R COPIER 000510R
CR = 000015 CSJBLK 001142R DEFEXT 001132R ERRBYT= 000052 ERRIN 001111R
FINDLP 000322R TOR 000776R LF = 000012 LP 000774R LPT... 000466R
MSG0 001010R MSG1 001016R MSG2 001037R MSG3 001051R MSG4 001061R
PC =X000007 R0 =X000000 R1 =X000001 R2 =X000002 R3 =X000003
R4 =X000004 R5 =X000005 SP =X000006 START 000000R STKSAV 000010R
TIMBLK 001126R USRSPW= 000046 ...VP = 000001
. ABS. 000000 000
011260 001
ERRORS DETECTED: 0
FREE CORP: 15035. WORDS
,LP:/N:ITM/C=LSPPOOL,MAC

```

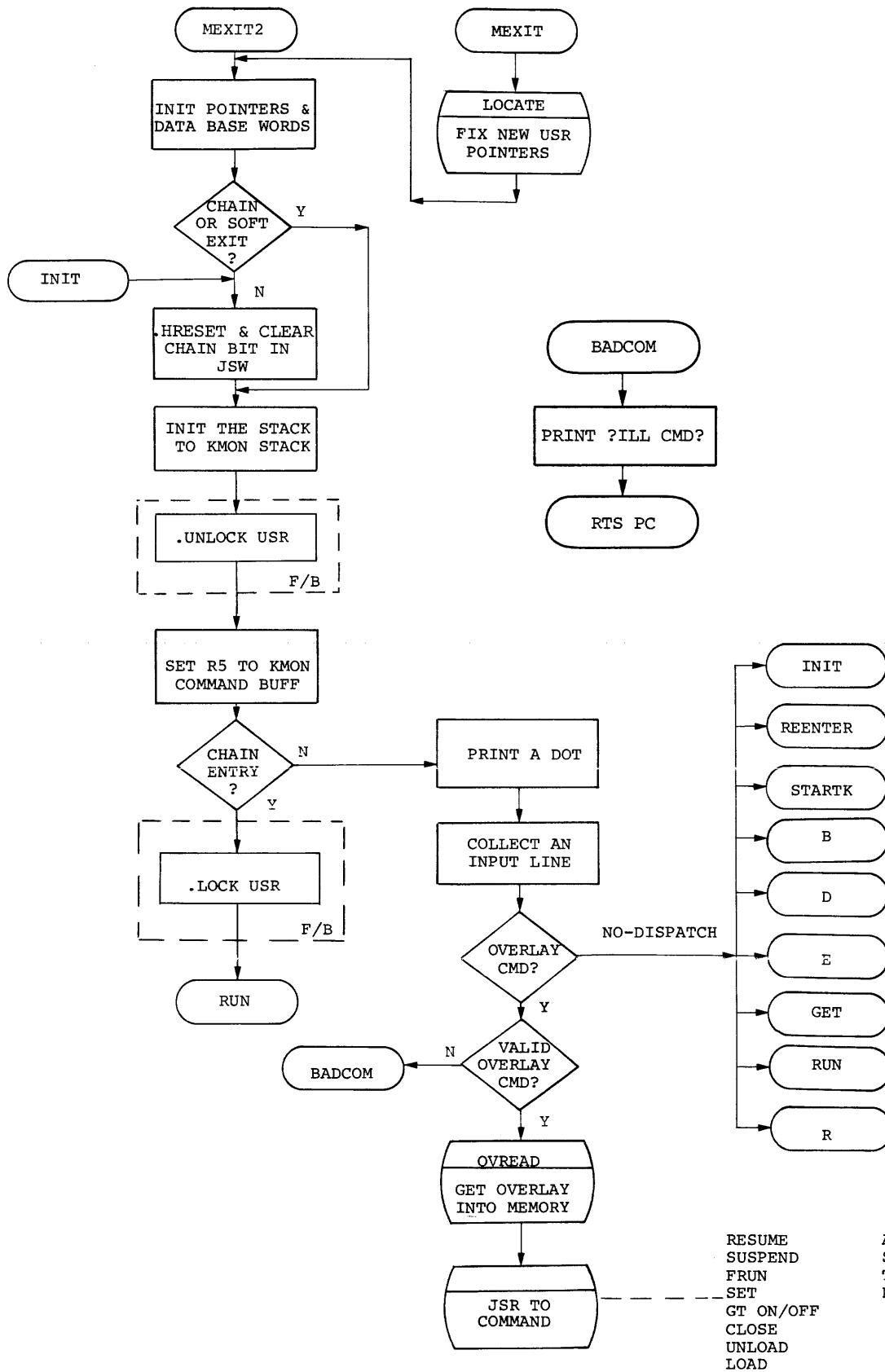

APPENDIX E
S/J AND F/B MONITOR FLOWCHARTS

The following flowcharts are of the Single-Job and Foreground/
Background Monitors. It is recommended that the reader have source
listings available for reference. Steps inside  are per-
formed only in the F/B or S/J Monitor, as noted.

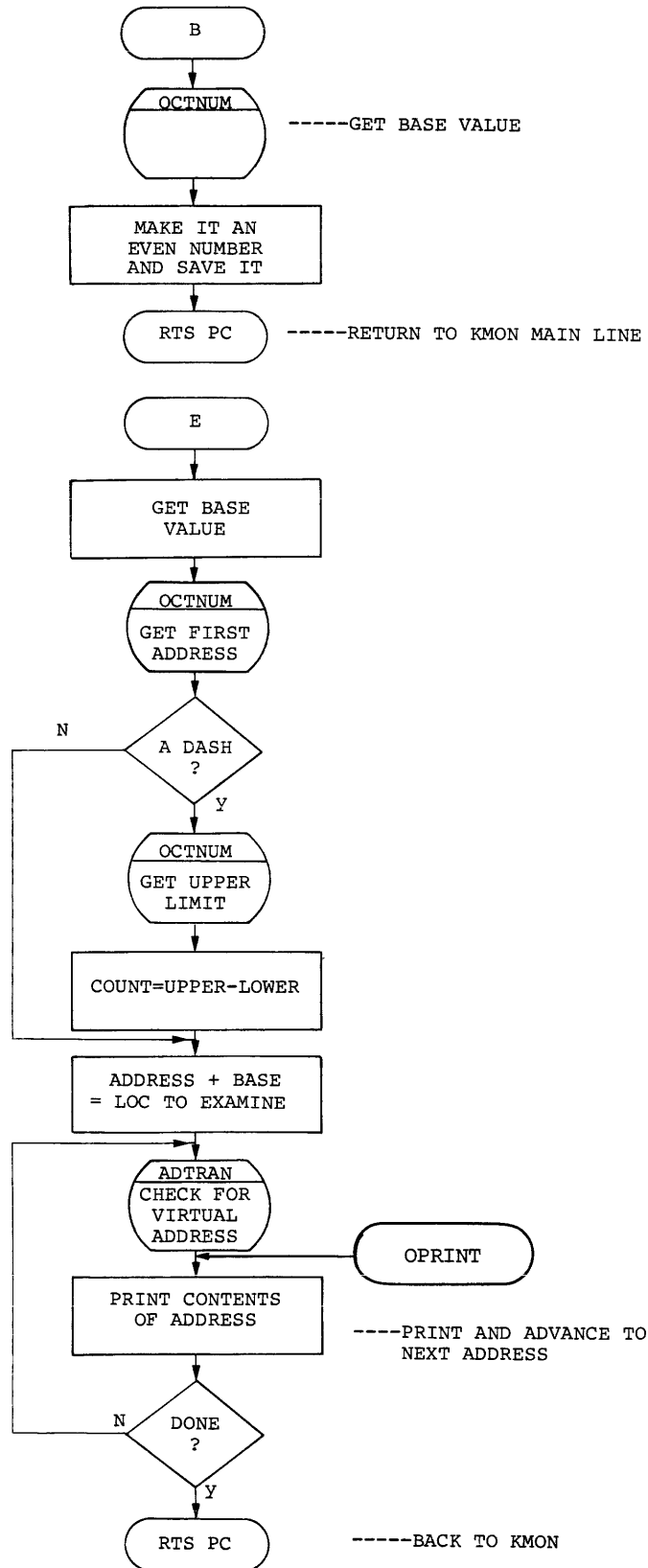
An index of all entry points appears at the end of the appendix.

E.1 KMON (KEYBOARD MONITOR) FLOWCHARTS

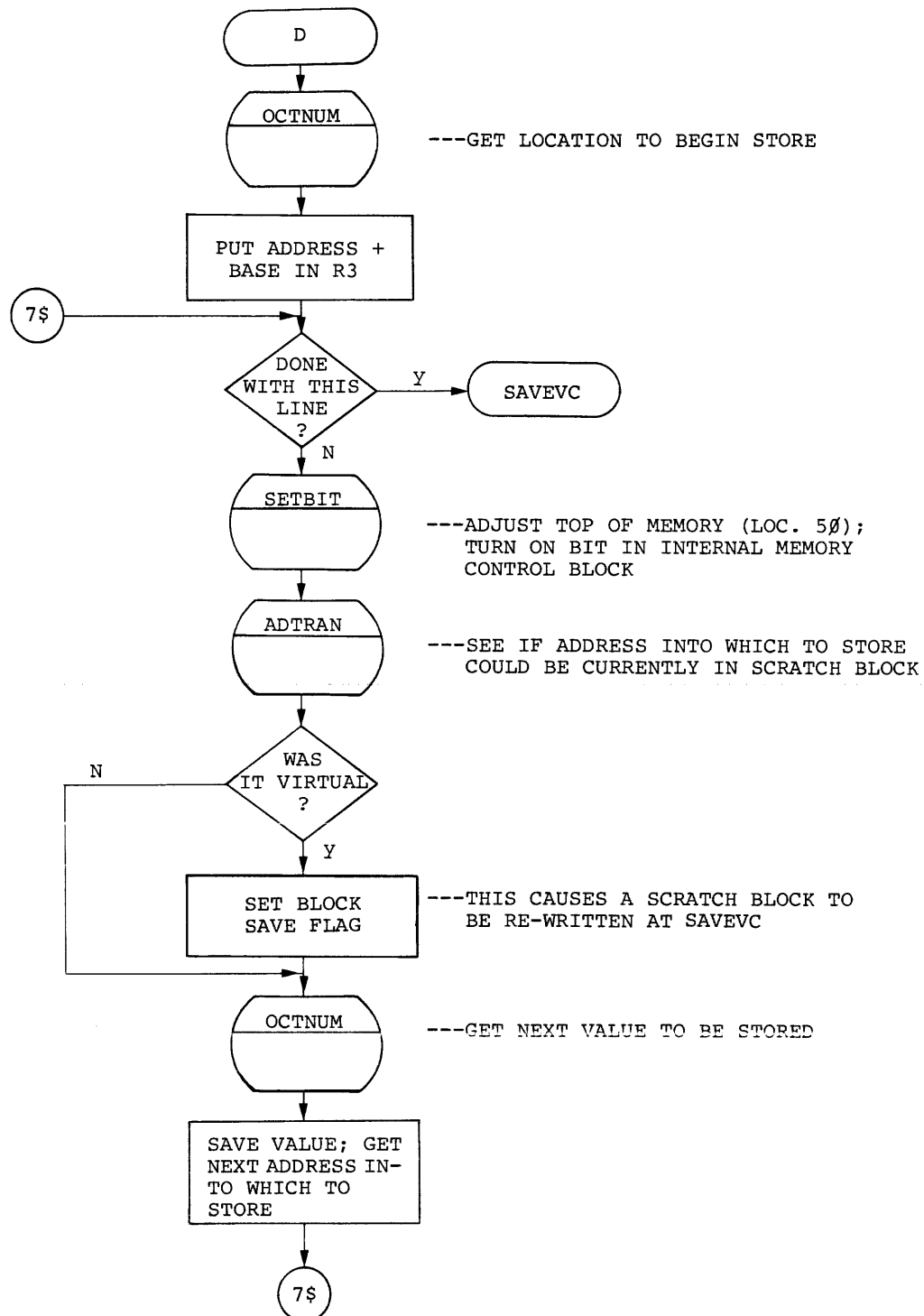
KMON



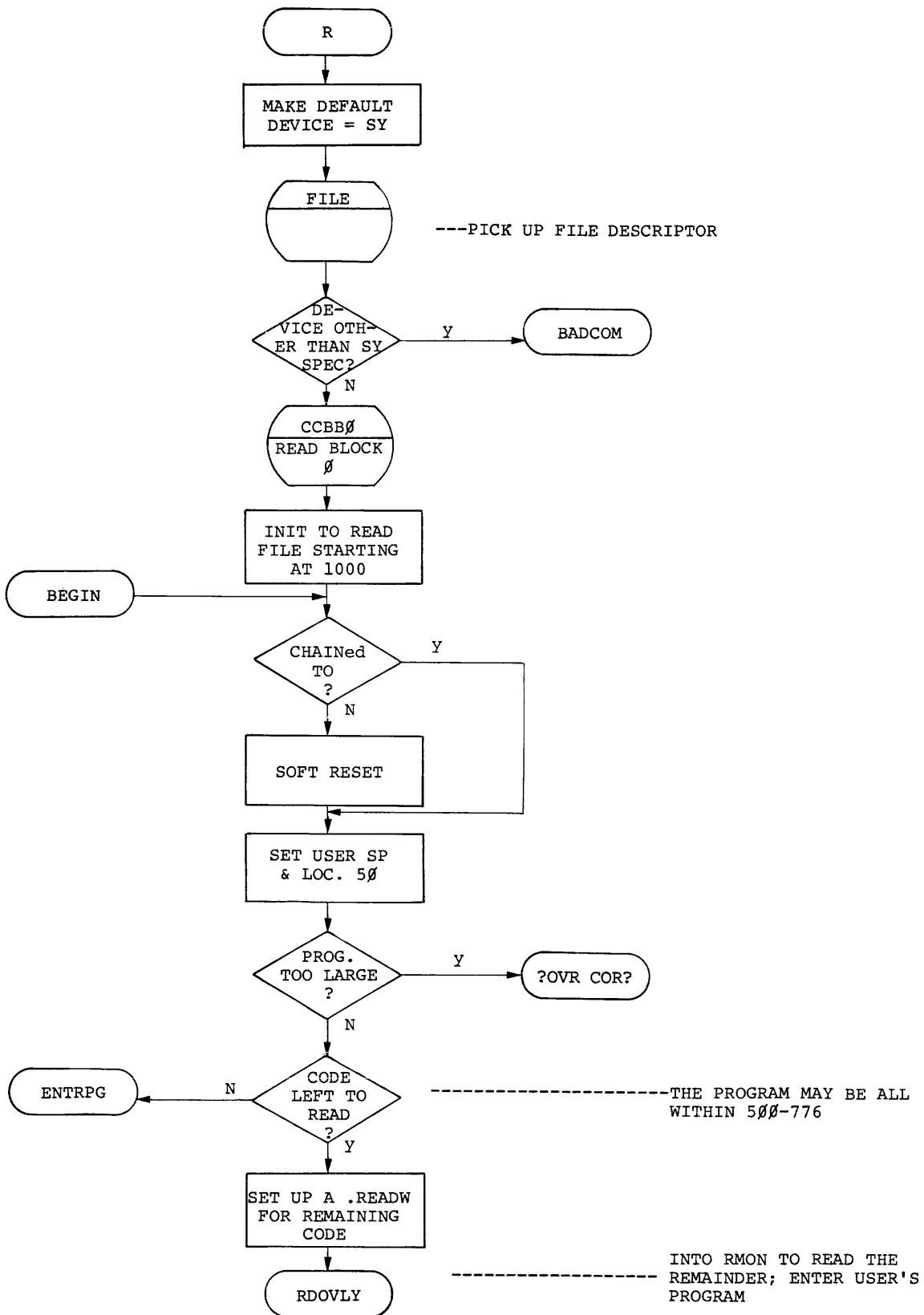
BASE/EXAMINE



DEPOSIT

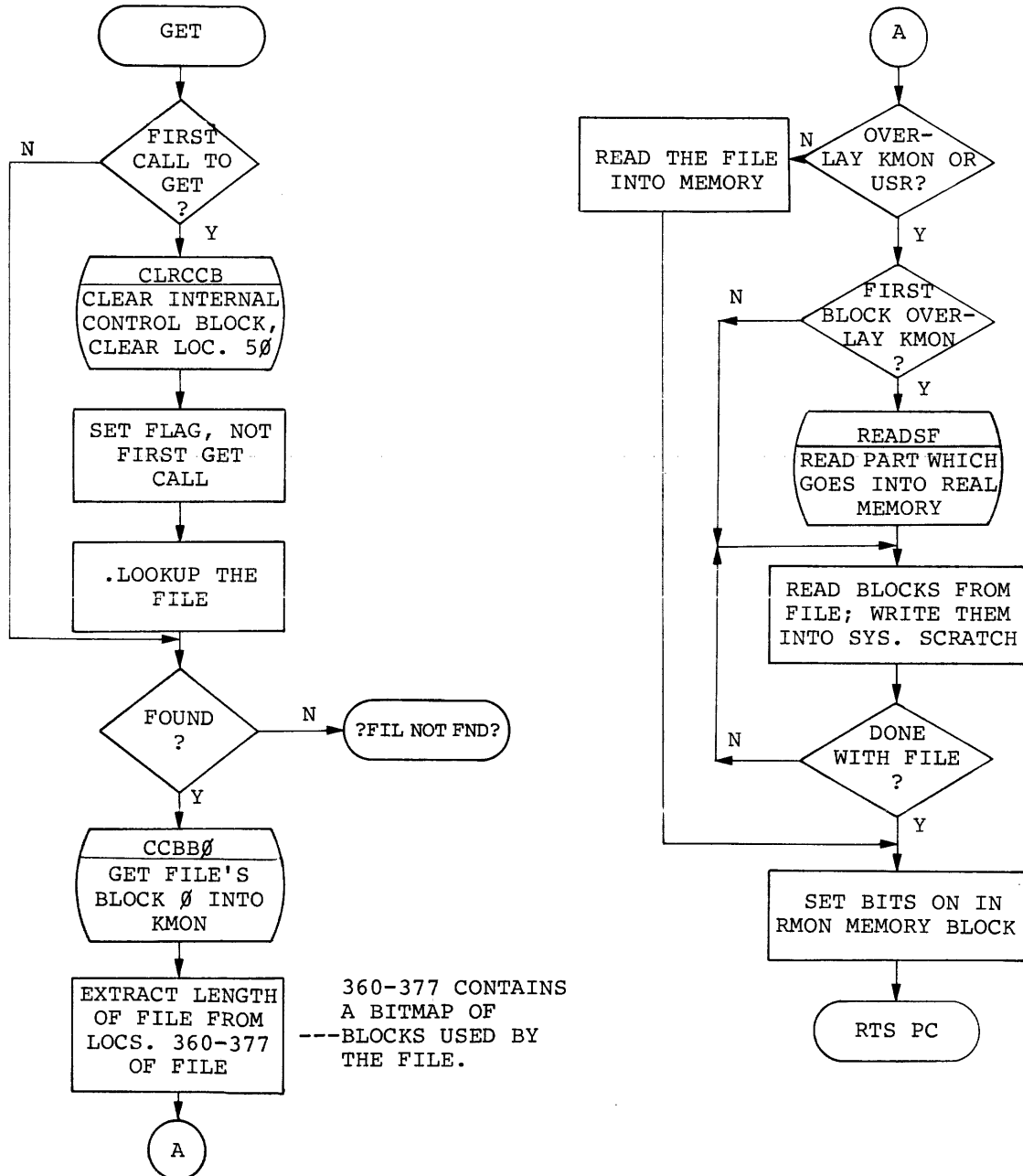


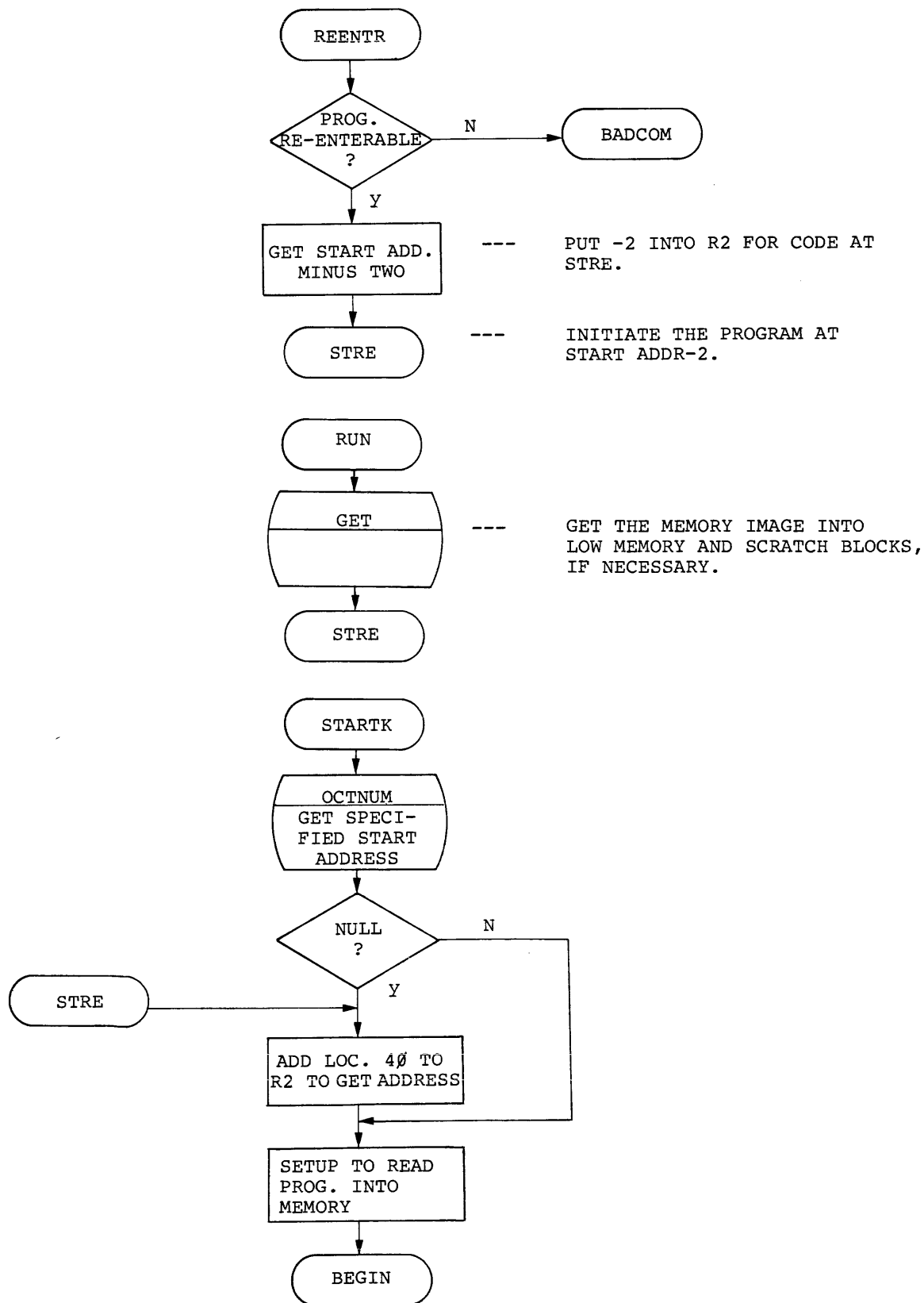
SAVEVC - Entered to rewrite the current virtual block back into the system scratch area. It also acts as the exit point for Deposit; The RTS PC will return control to KMON.



GET

GET - Used to load a .SAV image into memory. If parts of the file overlay KMON/USR, those parts are placed into system scratch blocks.

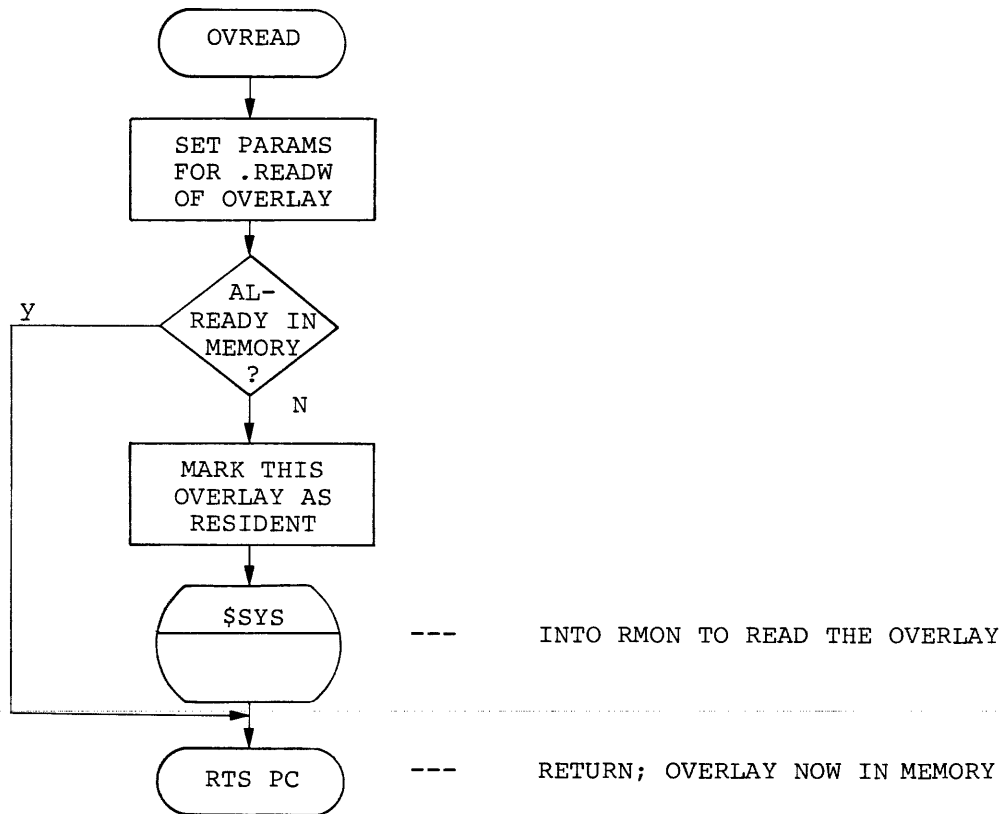




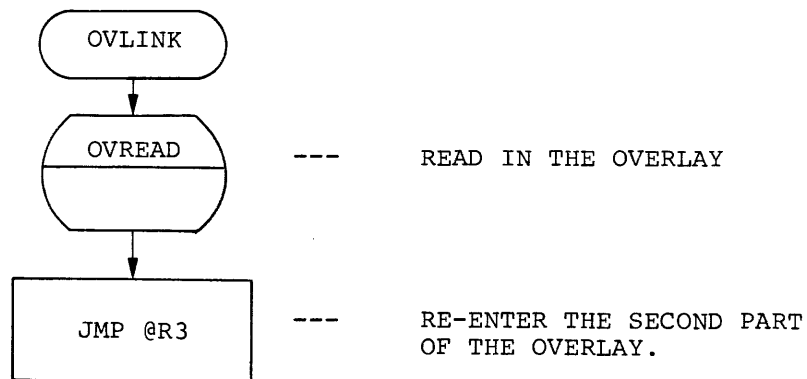
E.1.1 KMON Subroutines

OVREAD/OVLINK

OVREAD - Used to read overlay command processors into memory.

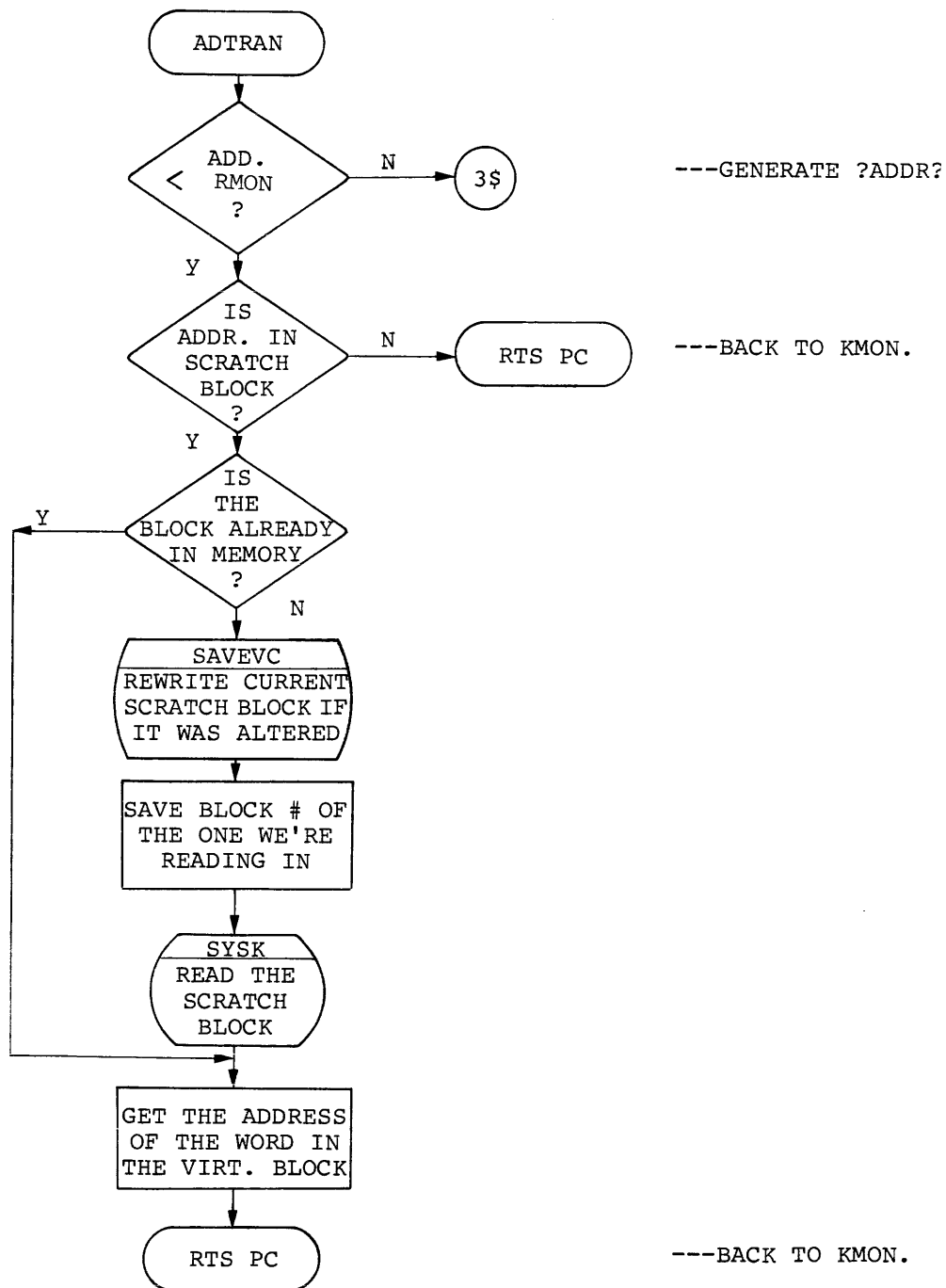


OVLINK - Called from overlay processors to allow linking from one overlay to the other.



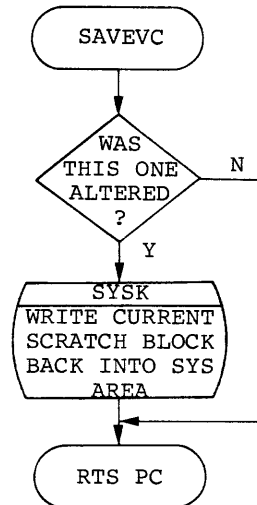
ADTRAN

ADTRAN - Used to determine if a user-typed address is a) legal (i.e., address of RMON), b) in scratch blocks on system device.

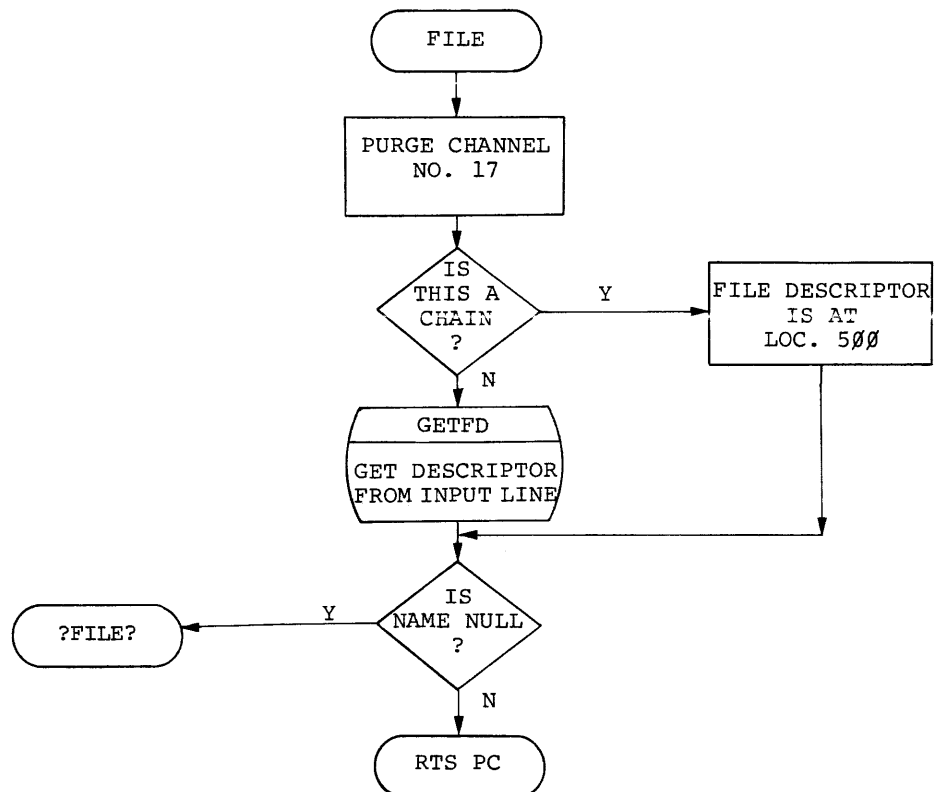


SAVEVC/FILE

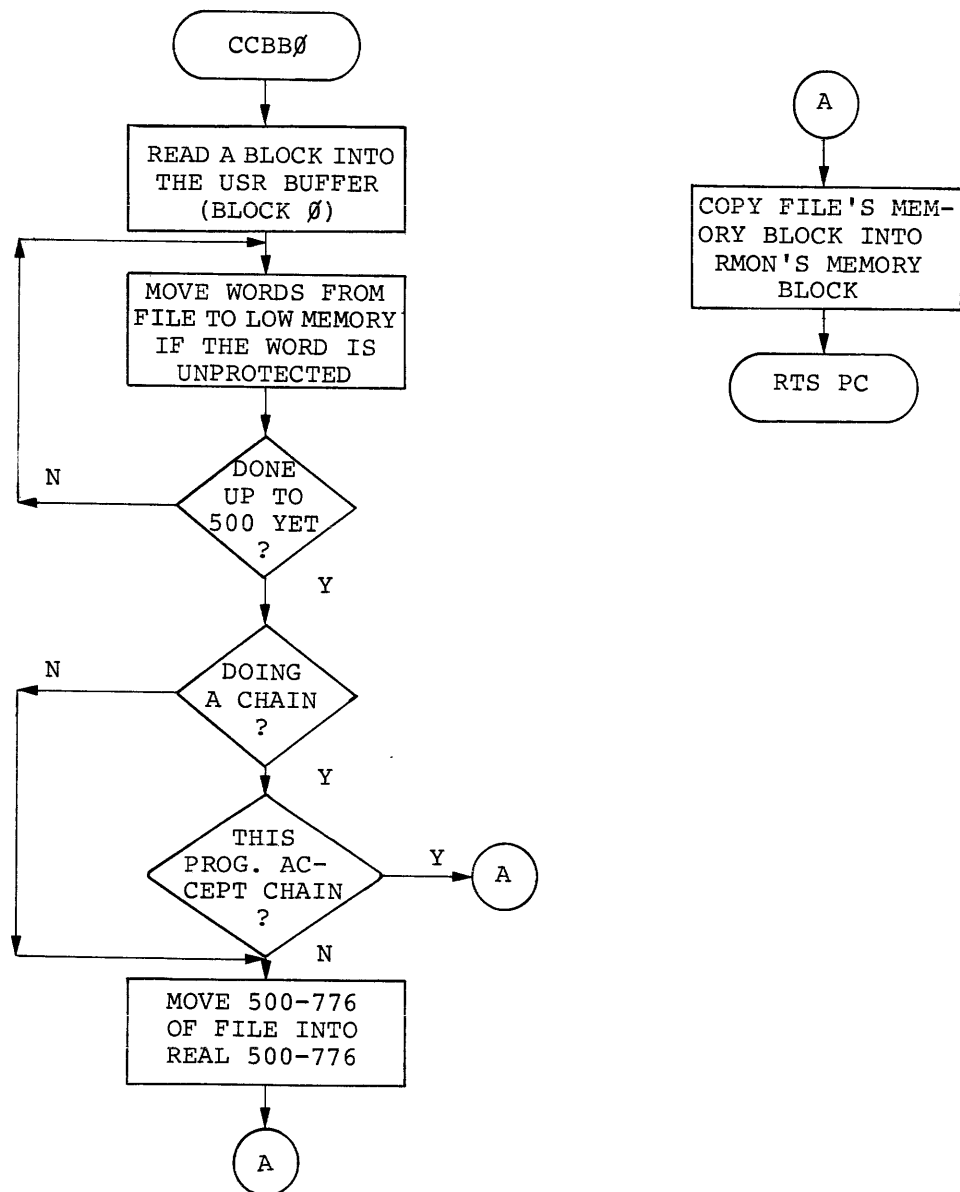
SAVEVC - Rewrites a block of memory back to the system scratch area if the block's contents were altered with a Deposit.



FILE - Called to pick up the .RAD50 representation of DEV:FILE.EXT. It will assume a default extension of .SAV.

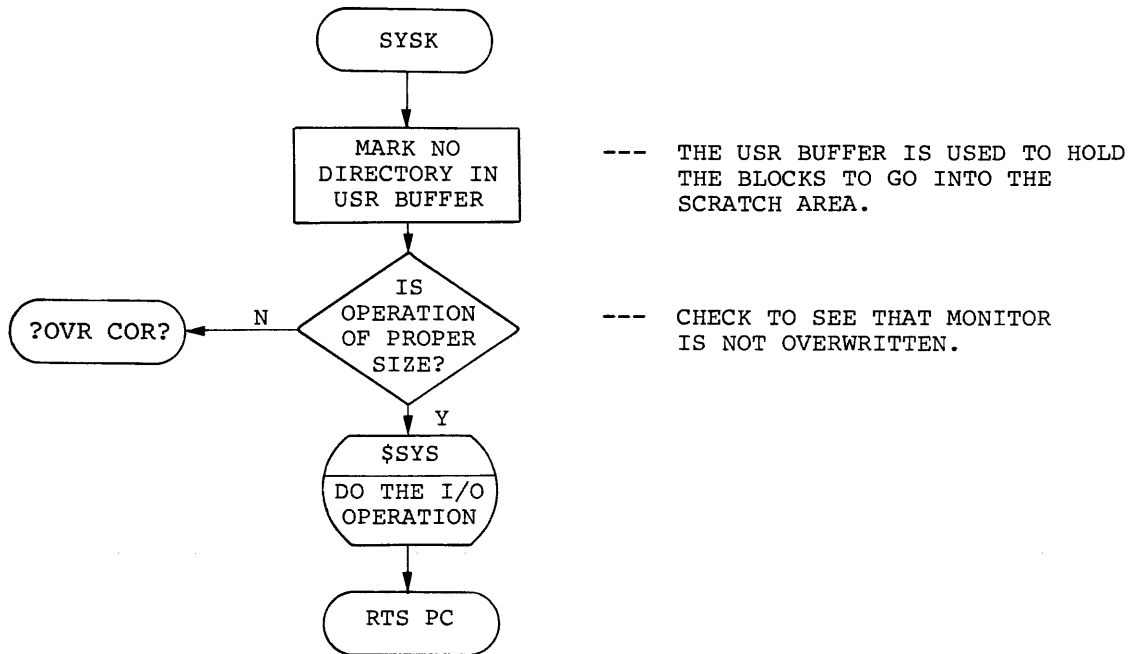


CCBBØ - The CCBBØ routine reads the first block of a .SAV file into the USR buffer, then moves selected locations from that block into the corresponding physical memory locations. The words moved are those marked with Ø's in the RMON bitmap. This procedure protects the system from having its vectors overlaid. If a chain is being done to a program which does not accept a CHAIN, 500-776 will be loaded with the contents of the file.



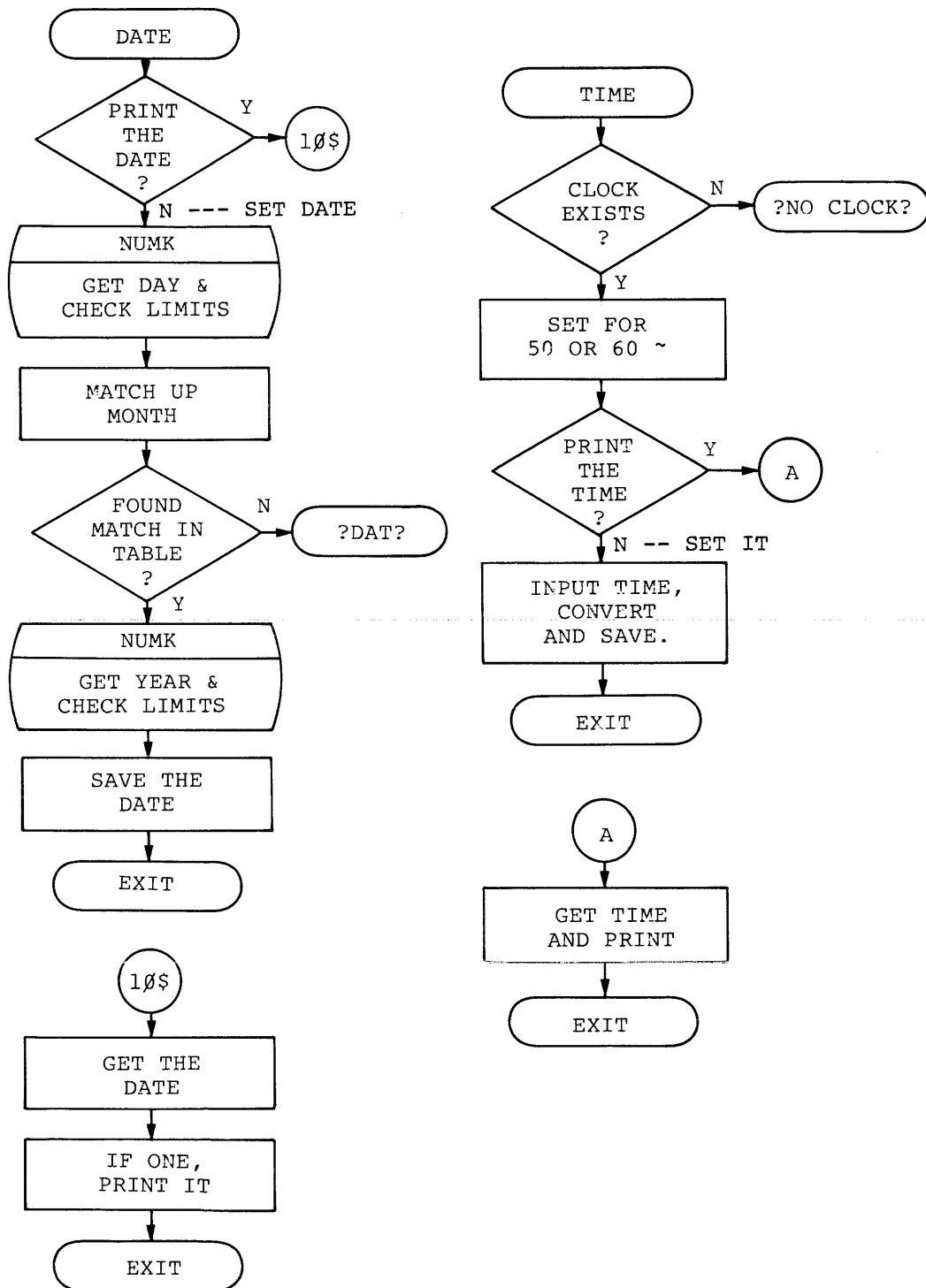
SYSK

SYSK - Used to read/write blocks into and out of the system scratch area.

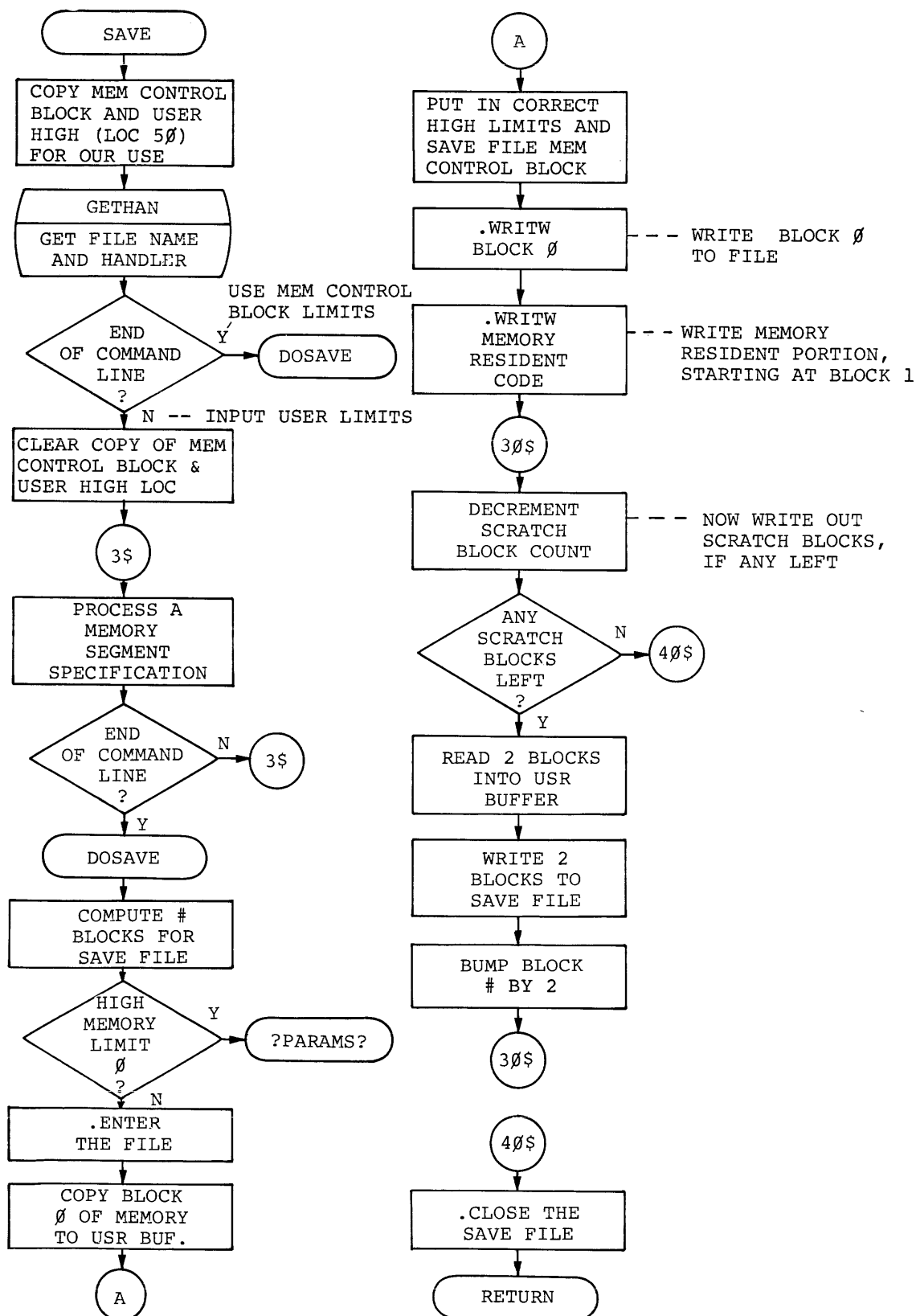


E.1.2 KMON Overlays

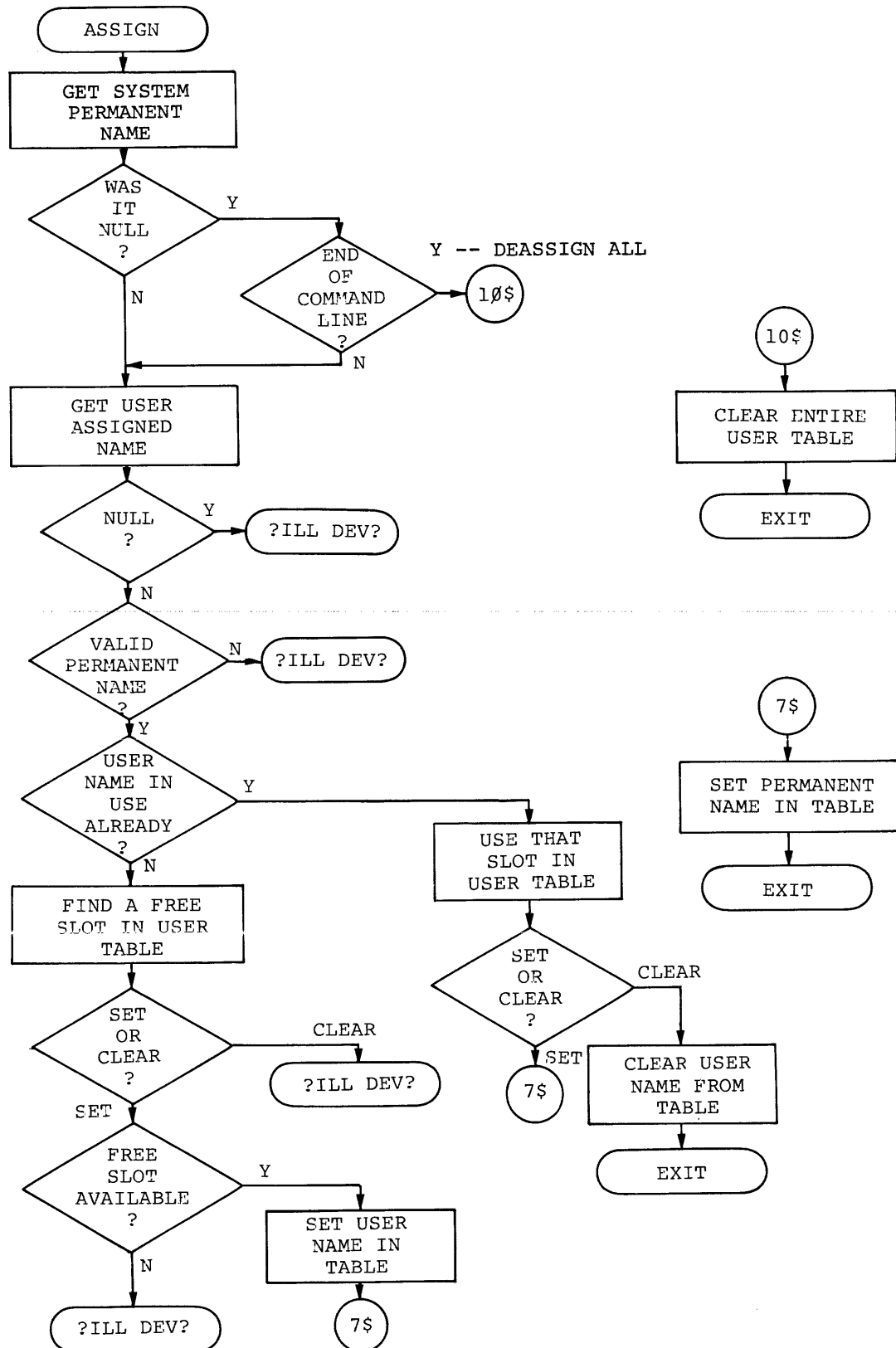
DATE/TIME

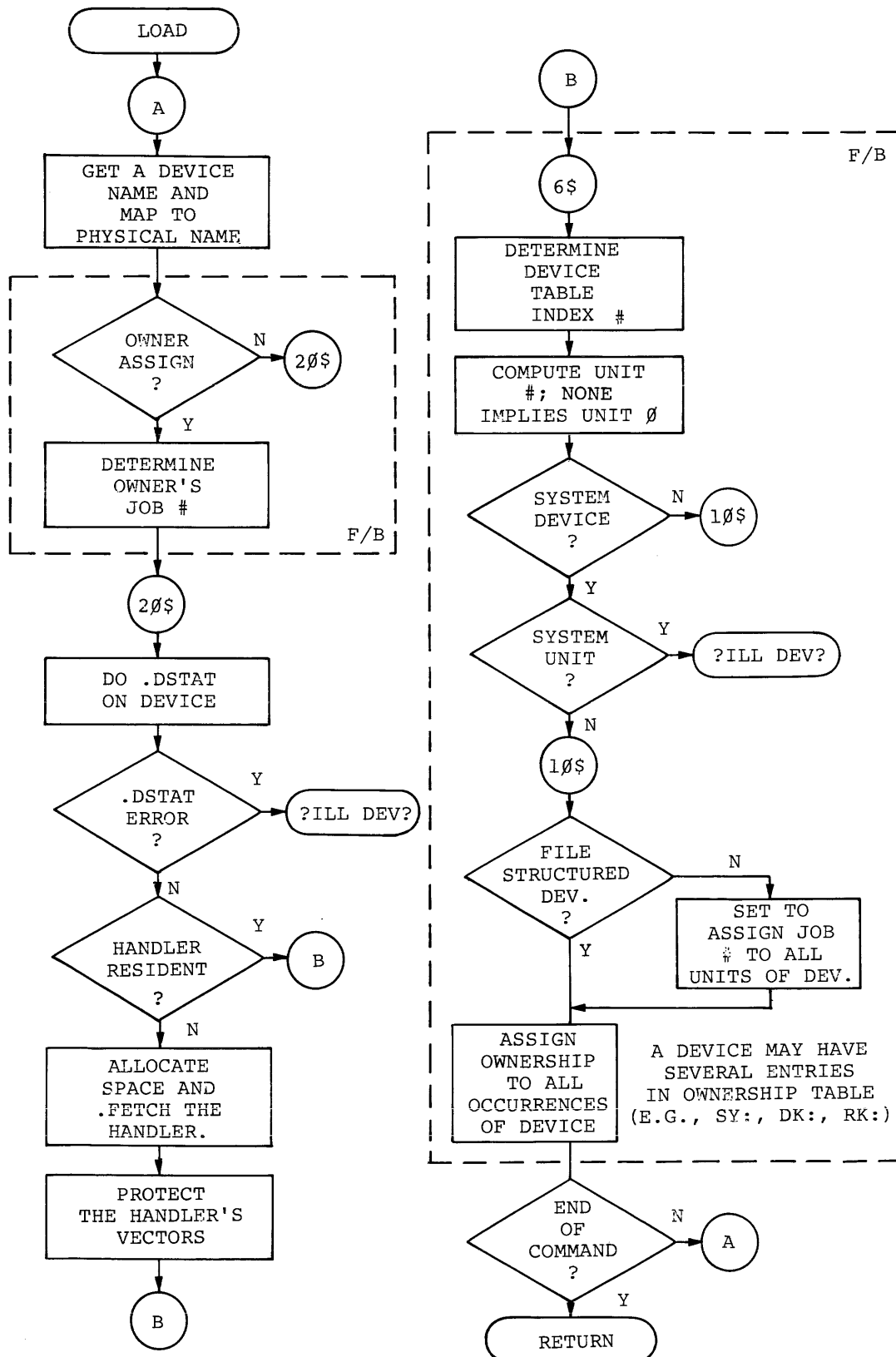


SAVE

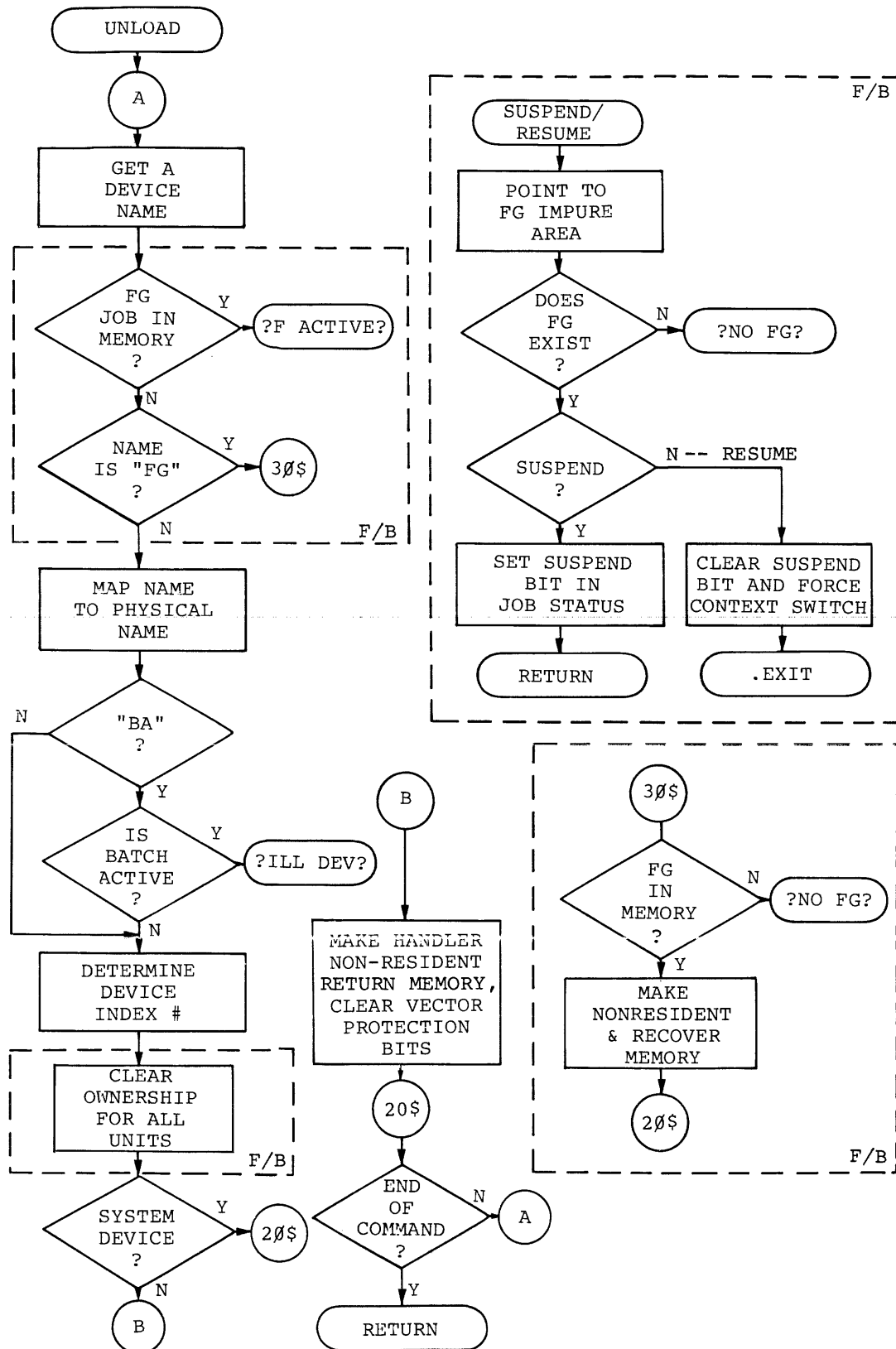


ASSIGN

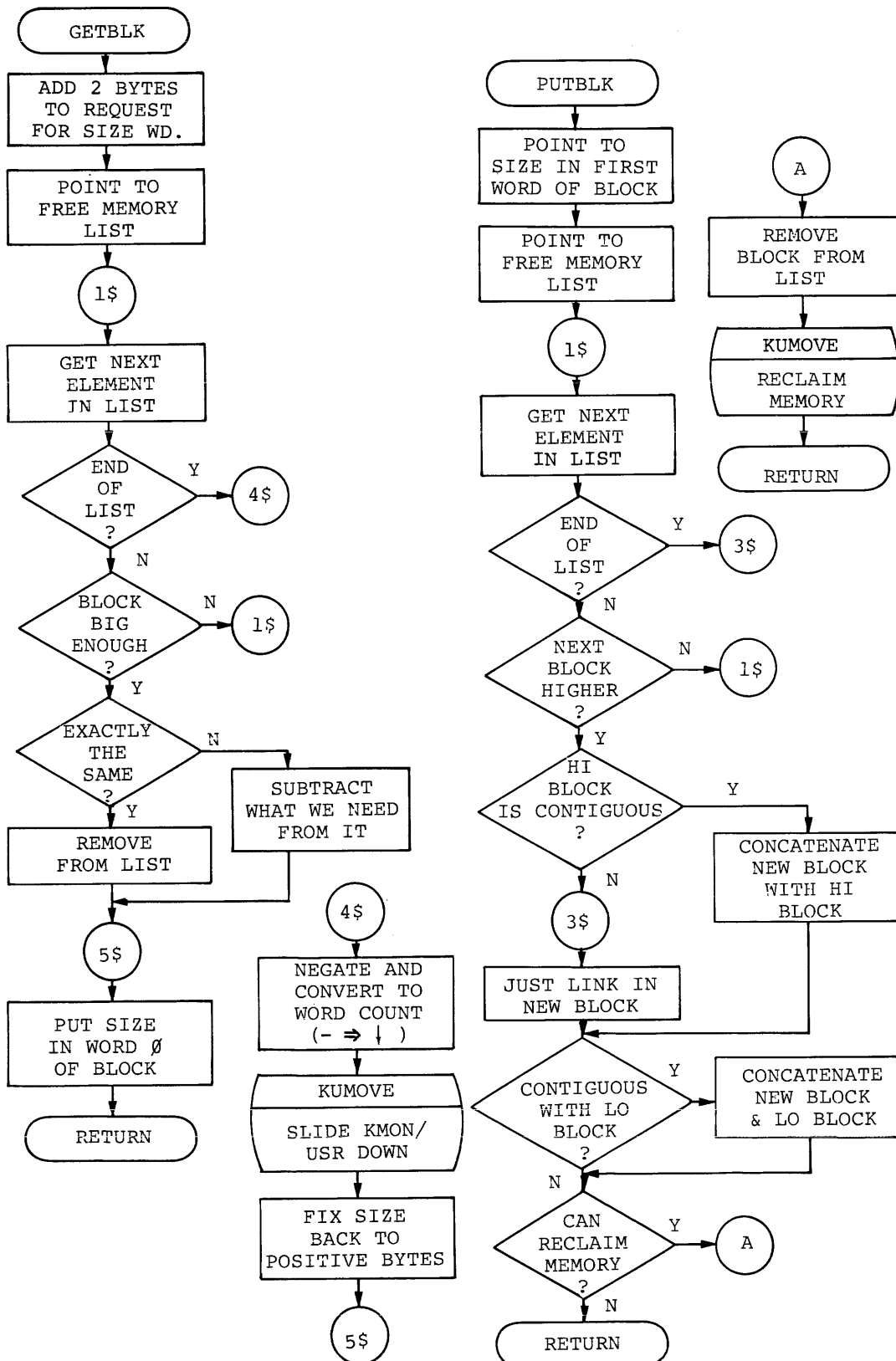




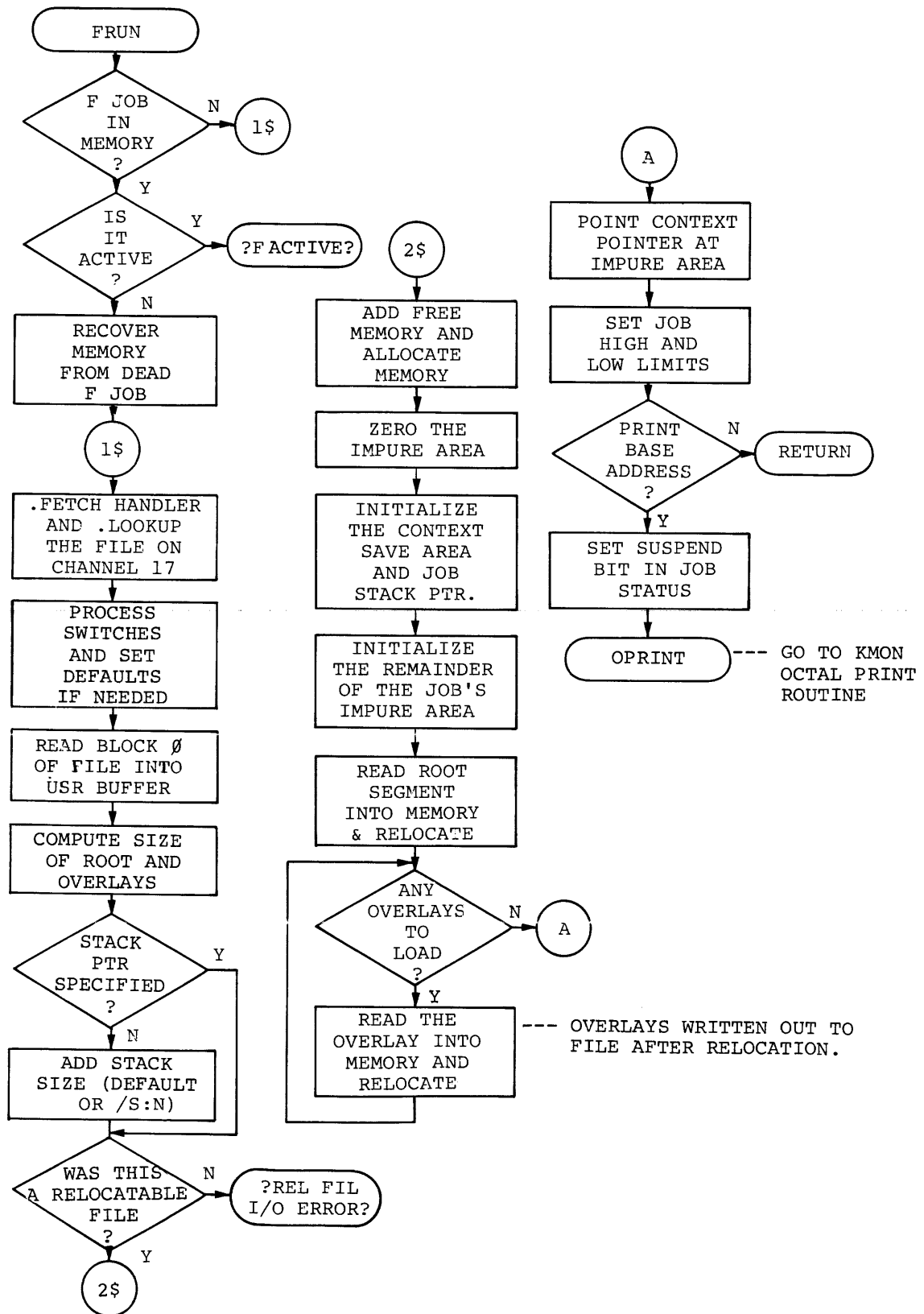
UNLOAD/SUSPEND/RESUME



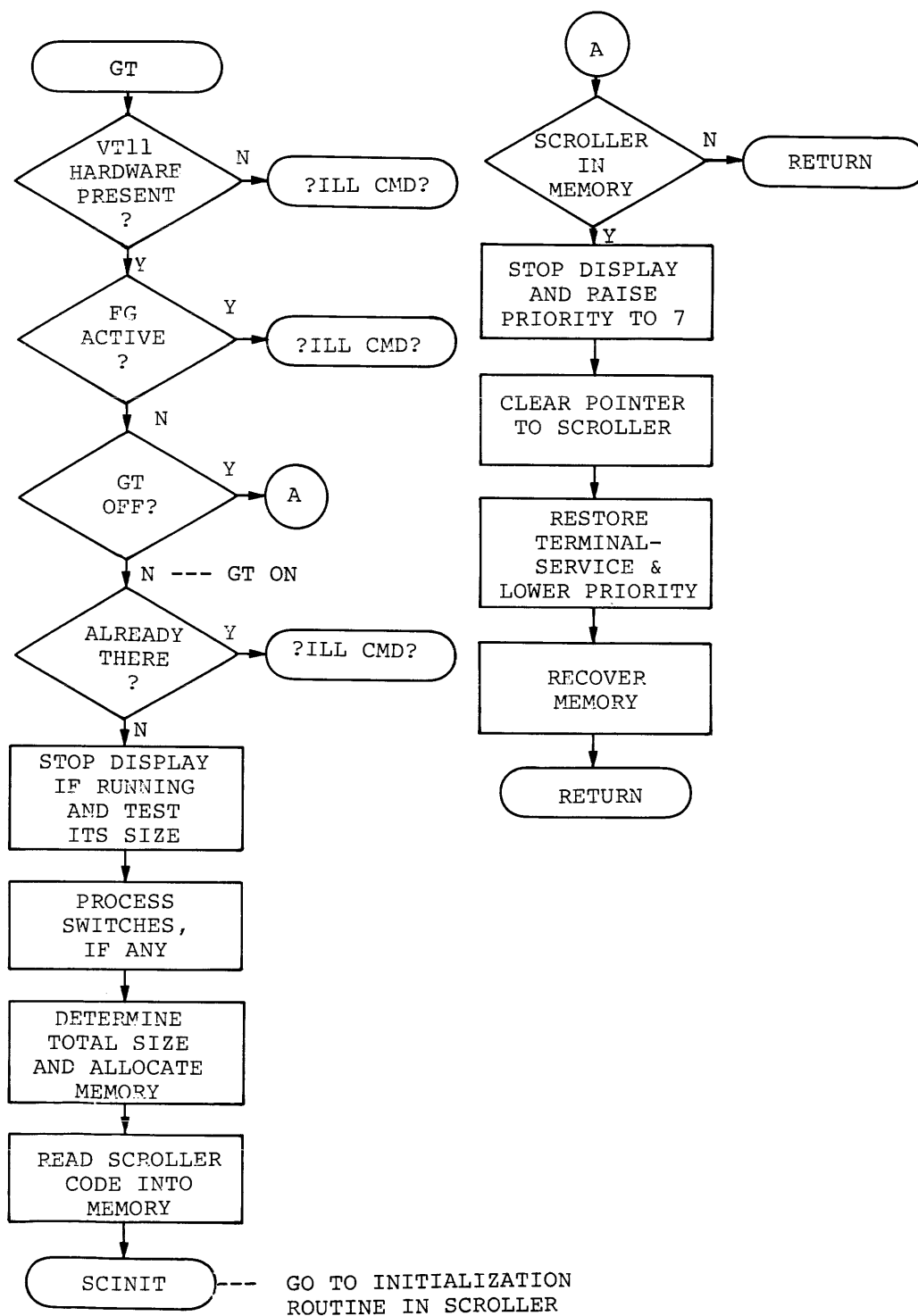
GET/PUT A BLOCK OF MEMORY



FRUN



GT ON/OFF



E.2 USR (USER SERVICE ROUTINES) FLOWCHARTS

USRBUF/FATAL/CDFN

The first 2 blocks of the USR are used by the USR for directory operations. They are also used by the KMON at various points for a 2-block general purpose buffer. There is, however, executable code in the buffer that can be executed every time a fresh copy of the USR is read from the system device. The functions included in the buffer are:

1. USR Relocation

This code is executed whenever the USR is newly read into memory. It serves to make certain pointers into RMON absolute.

2. Fatal error processor and fatal error messages (S/J only)

3. CDFN (channel define) EMT (S/J only)

The CDFN EMT call forces a new copy of the USR into memory to guarantee the presence of the EMT processor.

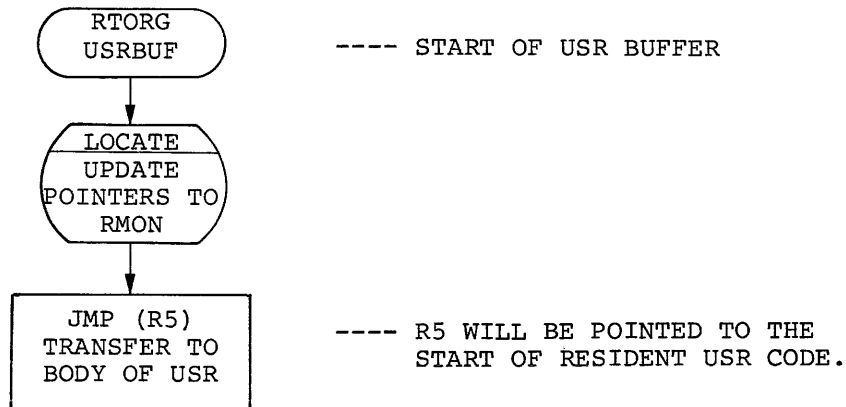
The flows for these functions follow.

NOTE

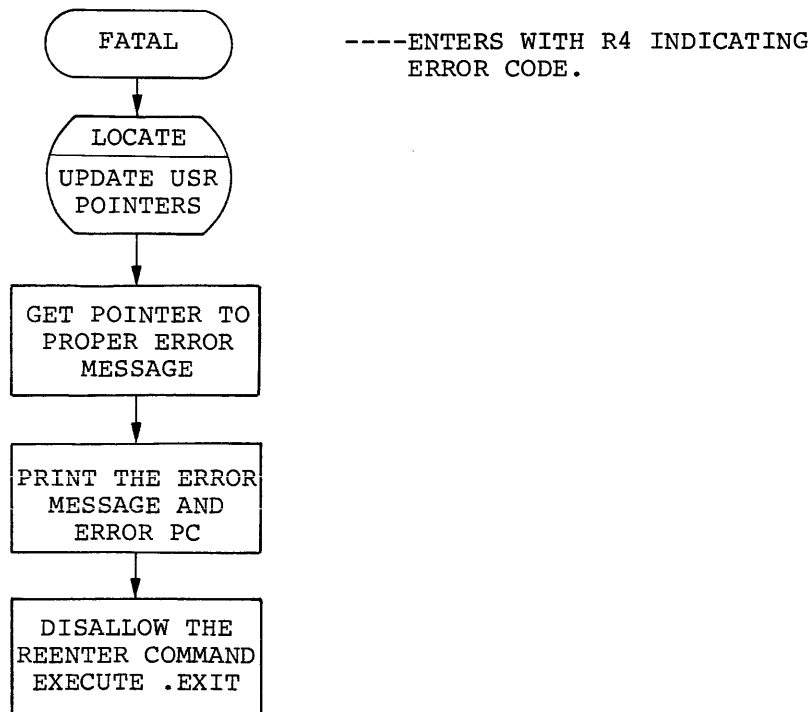
Fatal error handler and CDFN processor are RMON functions in the F/B Monitor. The only code in the buffer in the F/B system will be the USR relocation code.

USRBUF/FATAL/CDFN (CONT.)

USRBUF is the initial entry point for USR calls when the USR has just been read into memory. LOCATE sets up pointers into RMON.

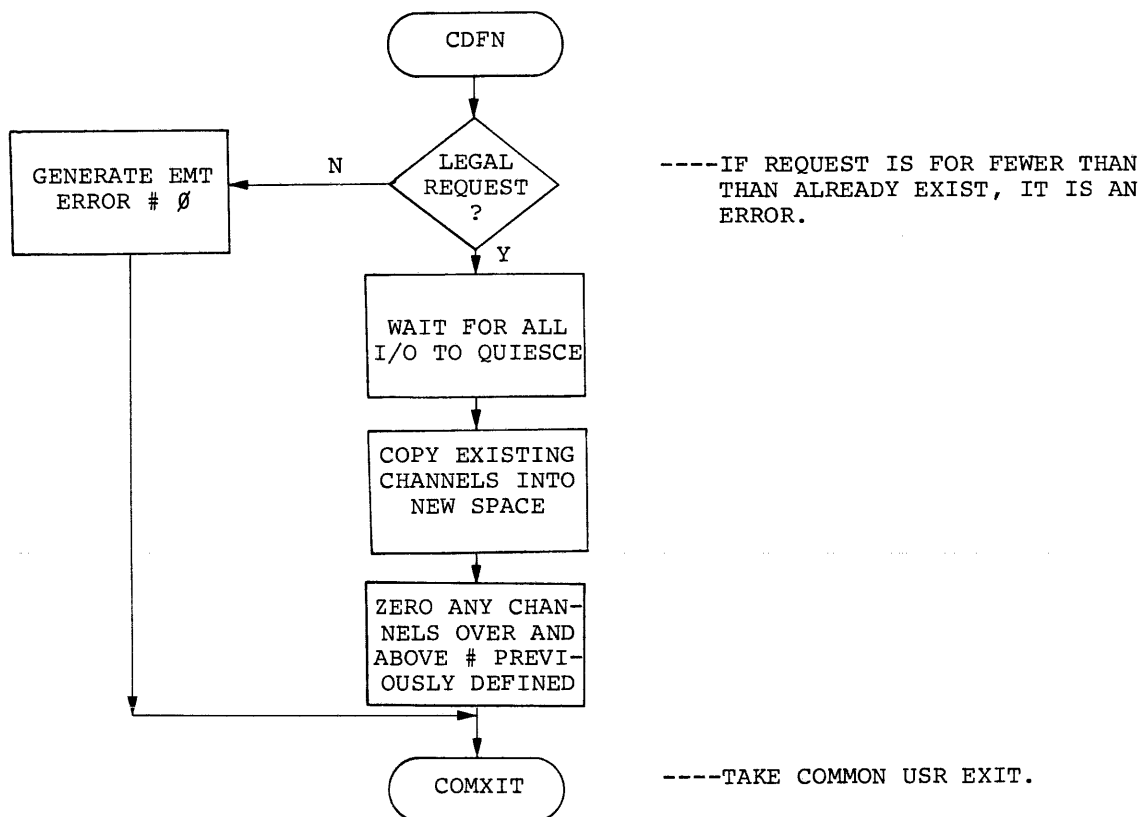


The LOCATE routine is called to update the list of pointers at RELIST. The list is initially a list of address differences (i.e., VALUE-\$RMON where VALUE is the desired location and \$RMON is the address of the start of RMON). LOCATE then makes all the differences into absolute addresses. Any errors which would generate a ?M-error use the FATAL error processor code to generate the message in the S/J system. This is a resident function in F/B.



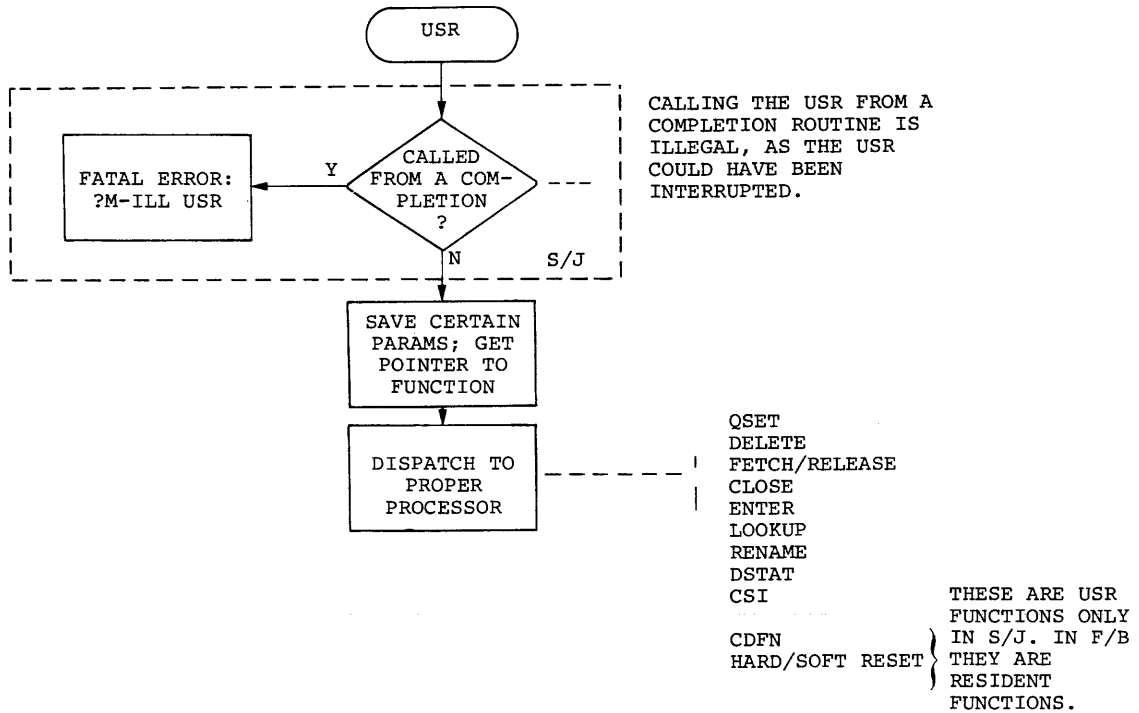
USRBUF/FATAL/CDFN (CONT.)

CDFN - A resident function in the F/B system.

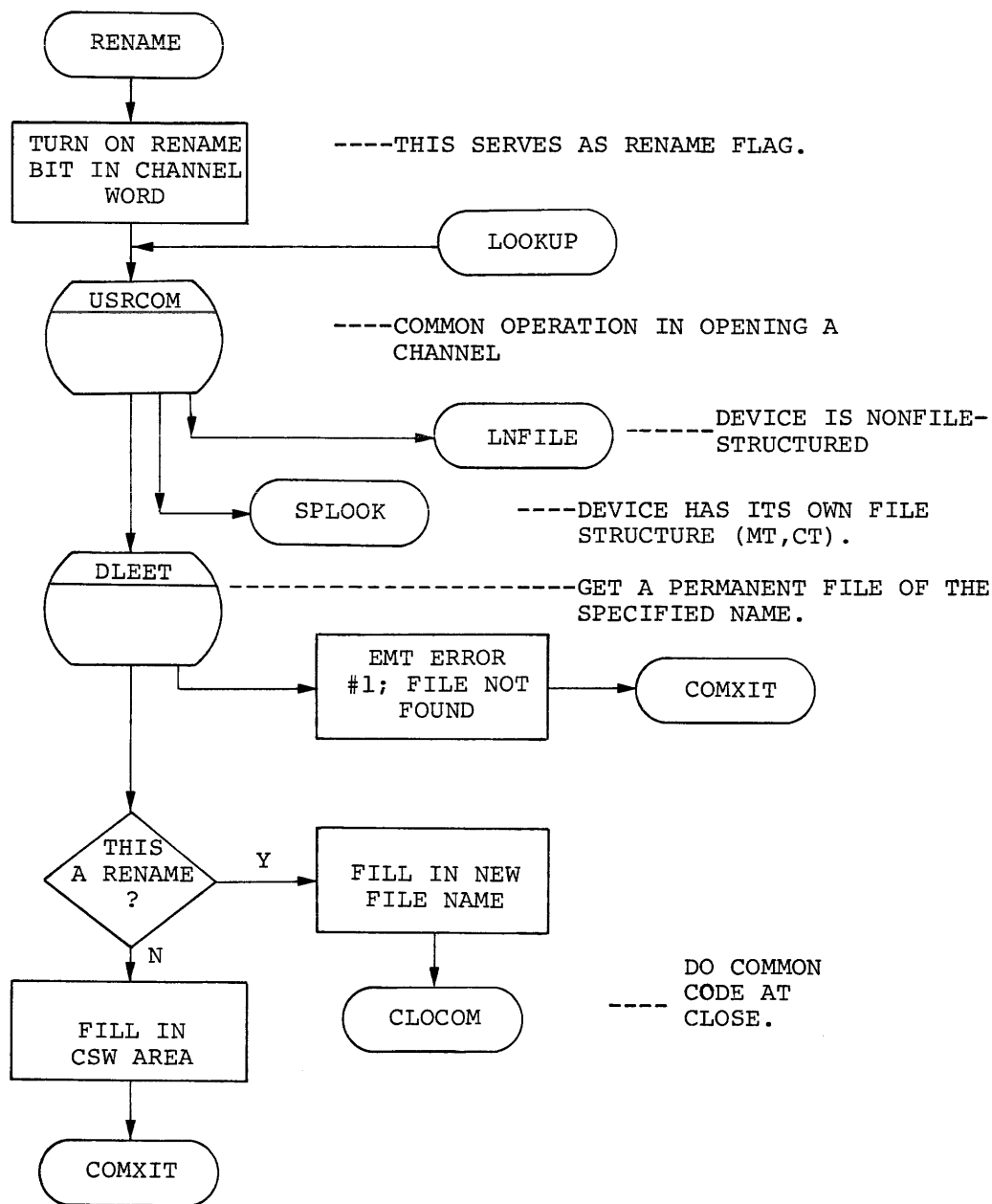


The following flowcharts detail the code contained in the main body of the USR. On entry to the USR, R2 contains an index representing the function to be performed. This is used to dispatch control to the proper processor.

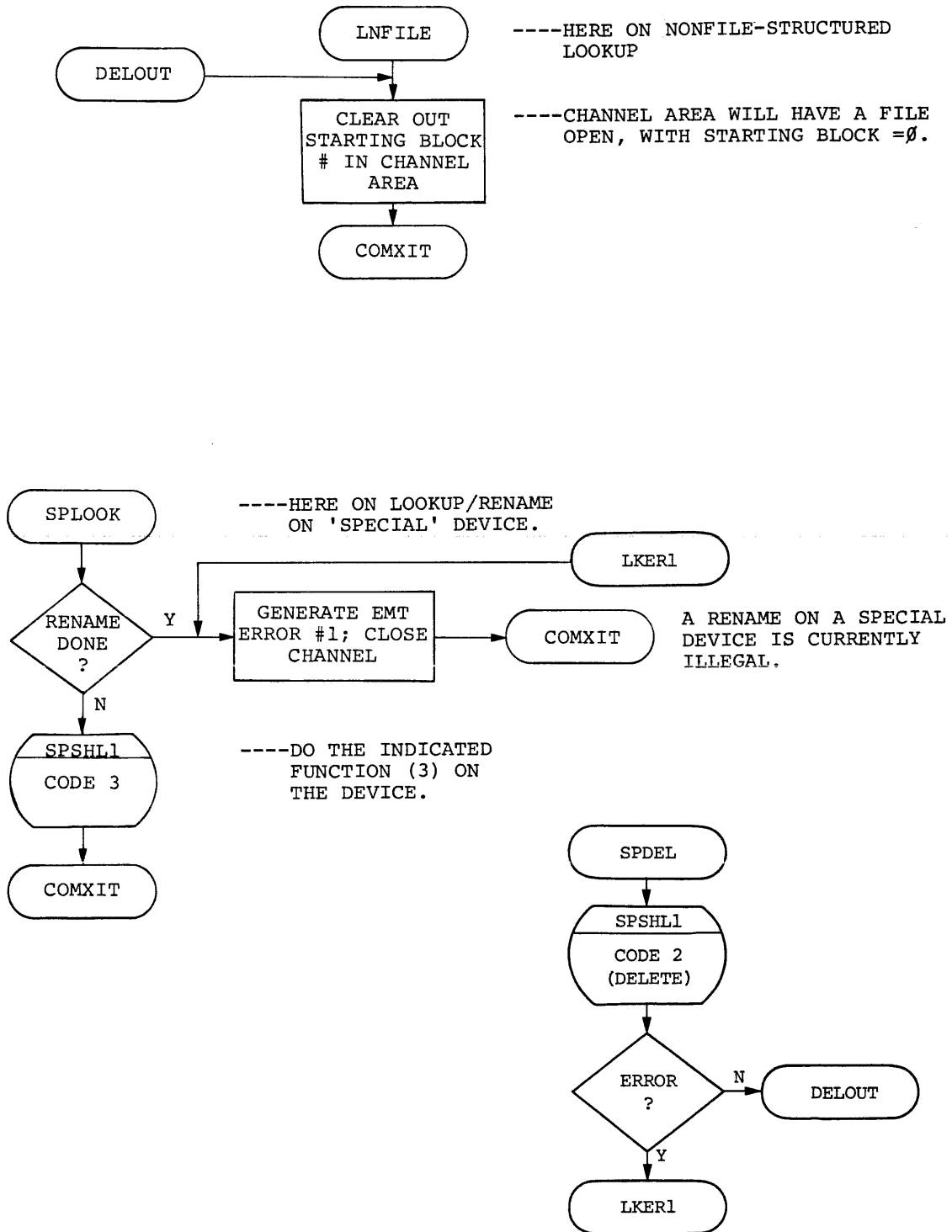
USR CODE



LOOKUP/RENAME

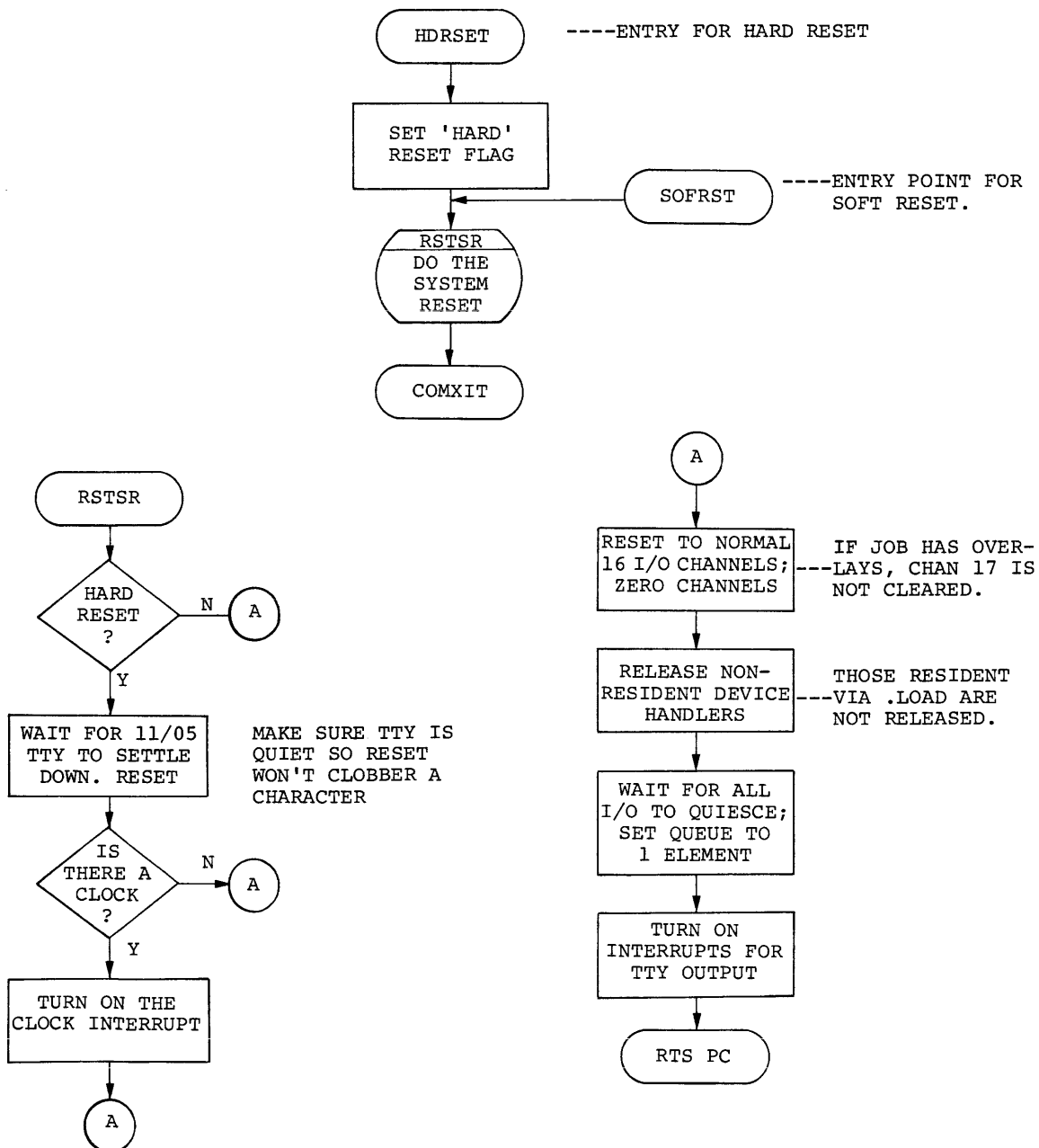


LOOKUP/RENAME (CONT.)

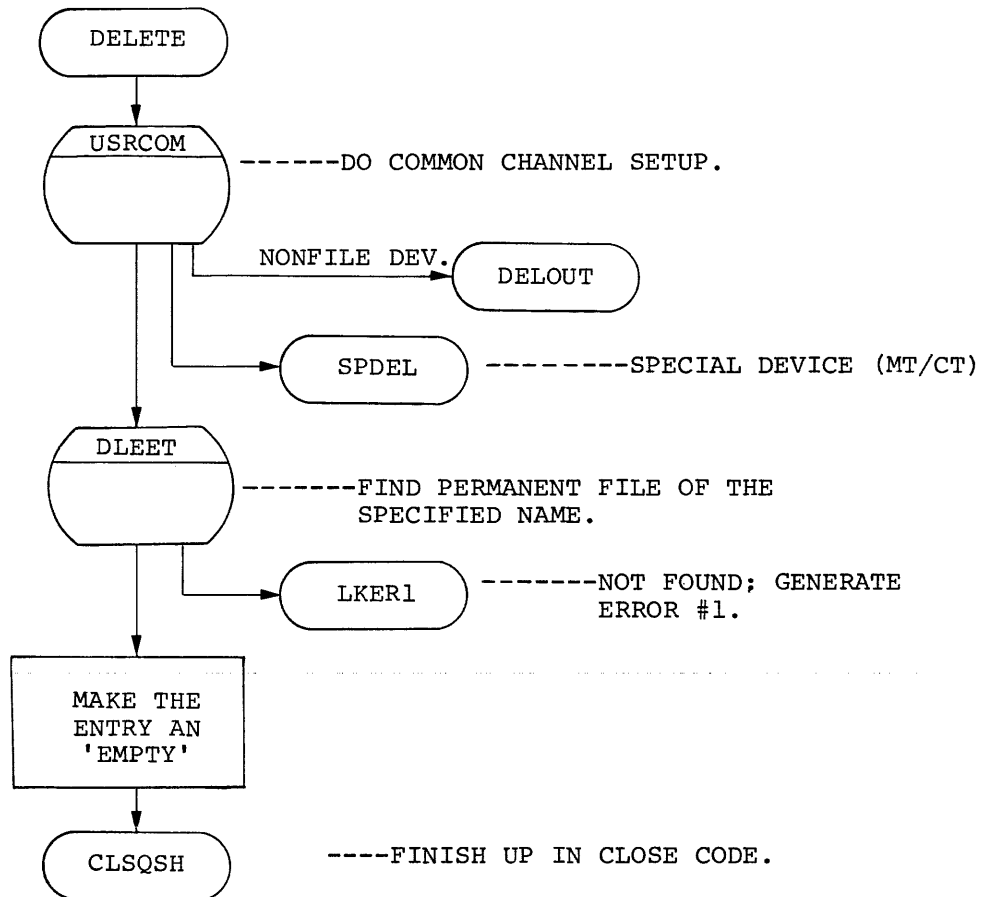


HARD/SOFT RESET

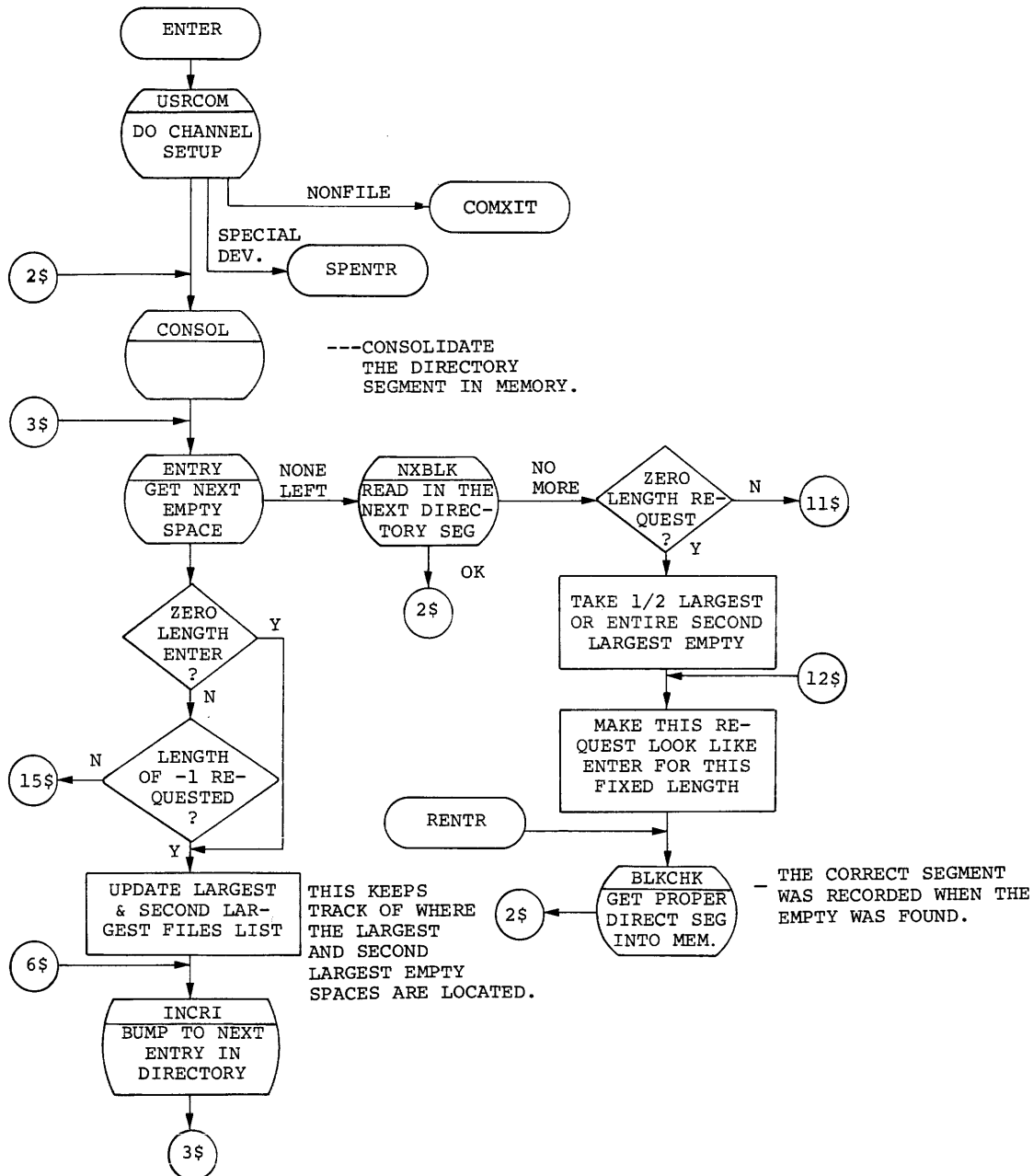
These are resident functions in F/B; USR functions in S/J.



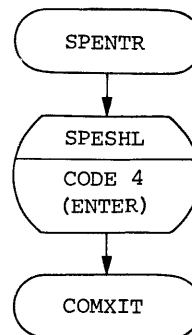
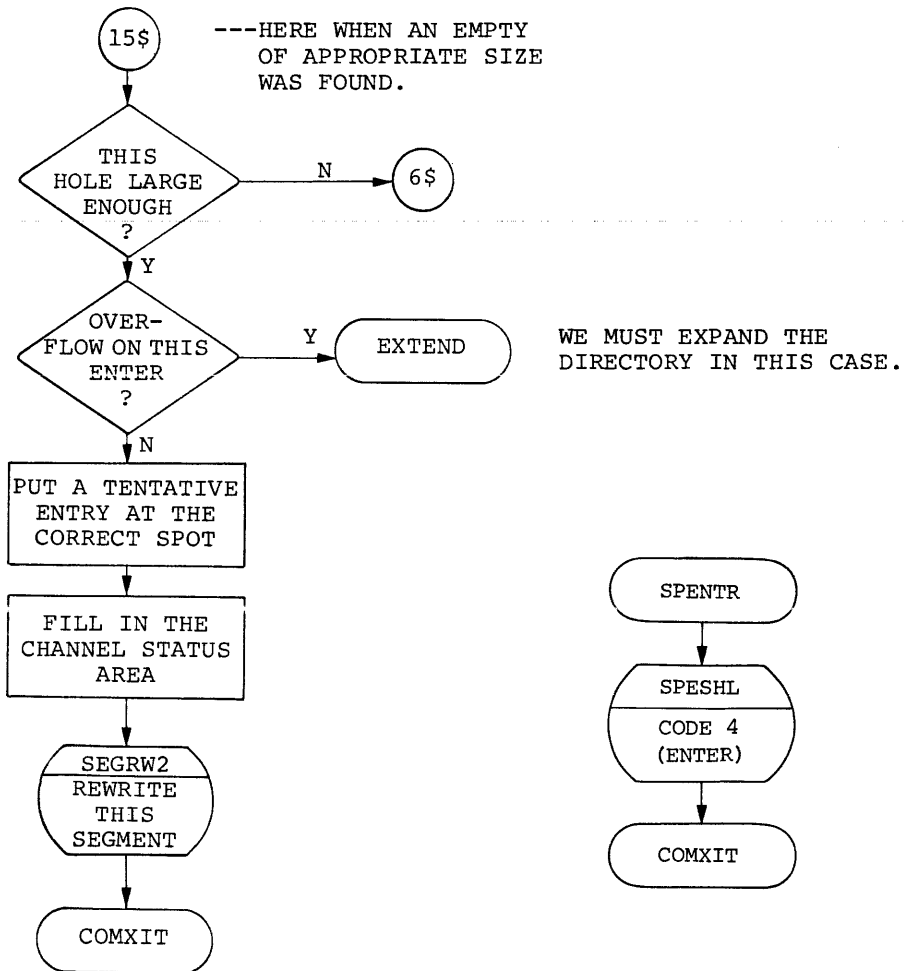
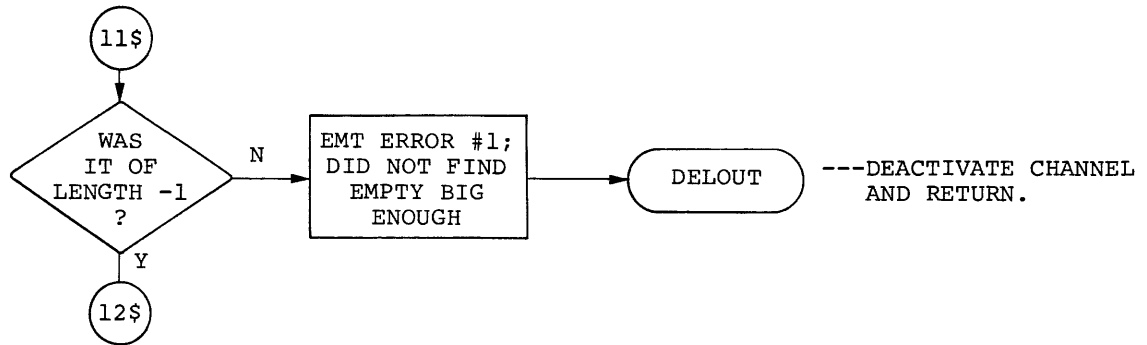
DELETE



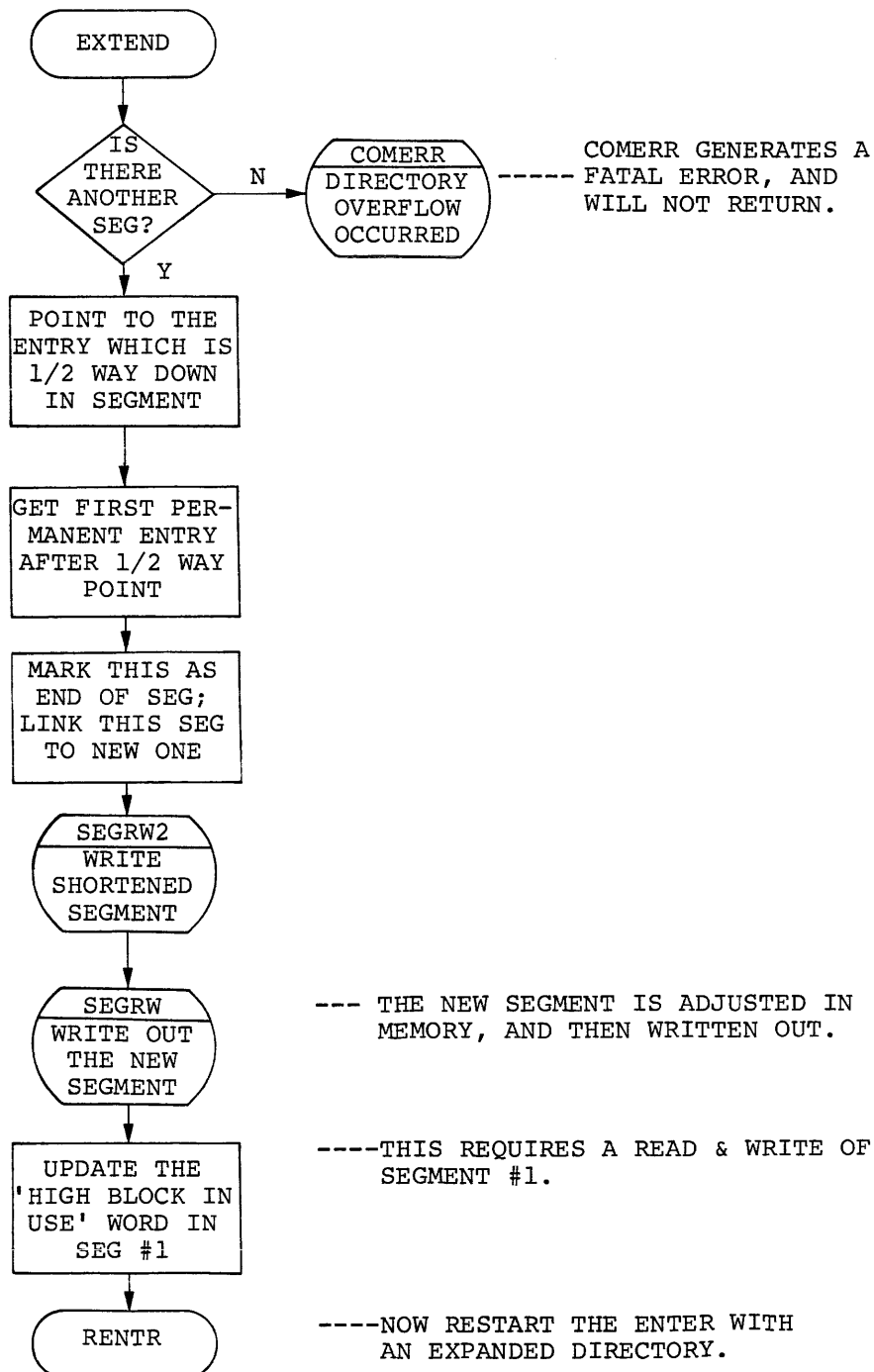
ENTER



ENTER (CONT.)

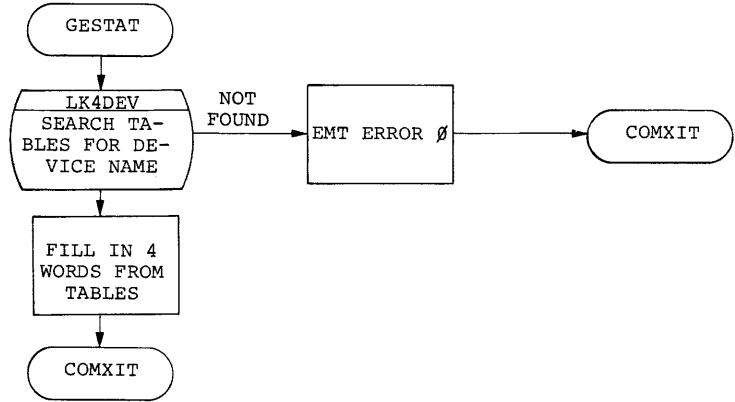


EXTEND

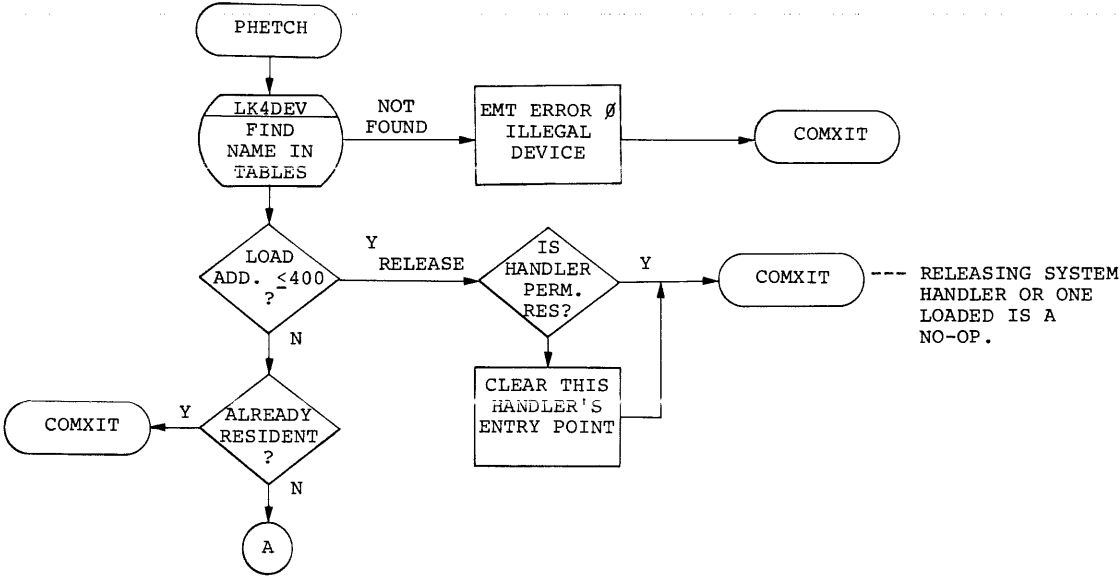


DSTAT/FETCH/RELEASE

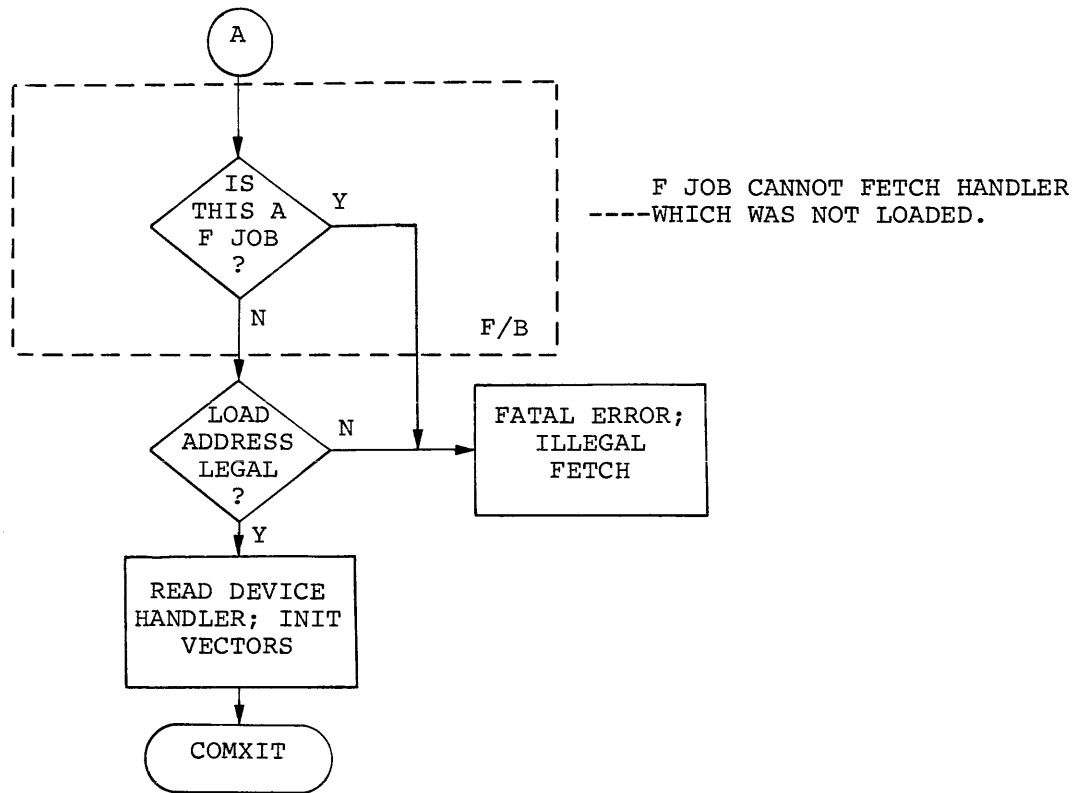
DSTAT- GET DEVICE STATUS



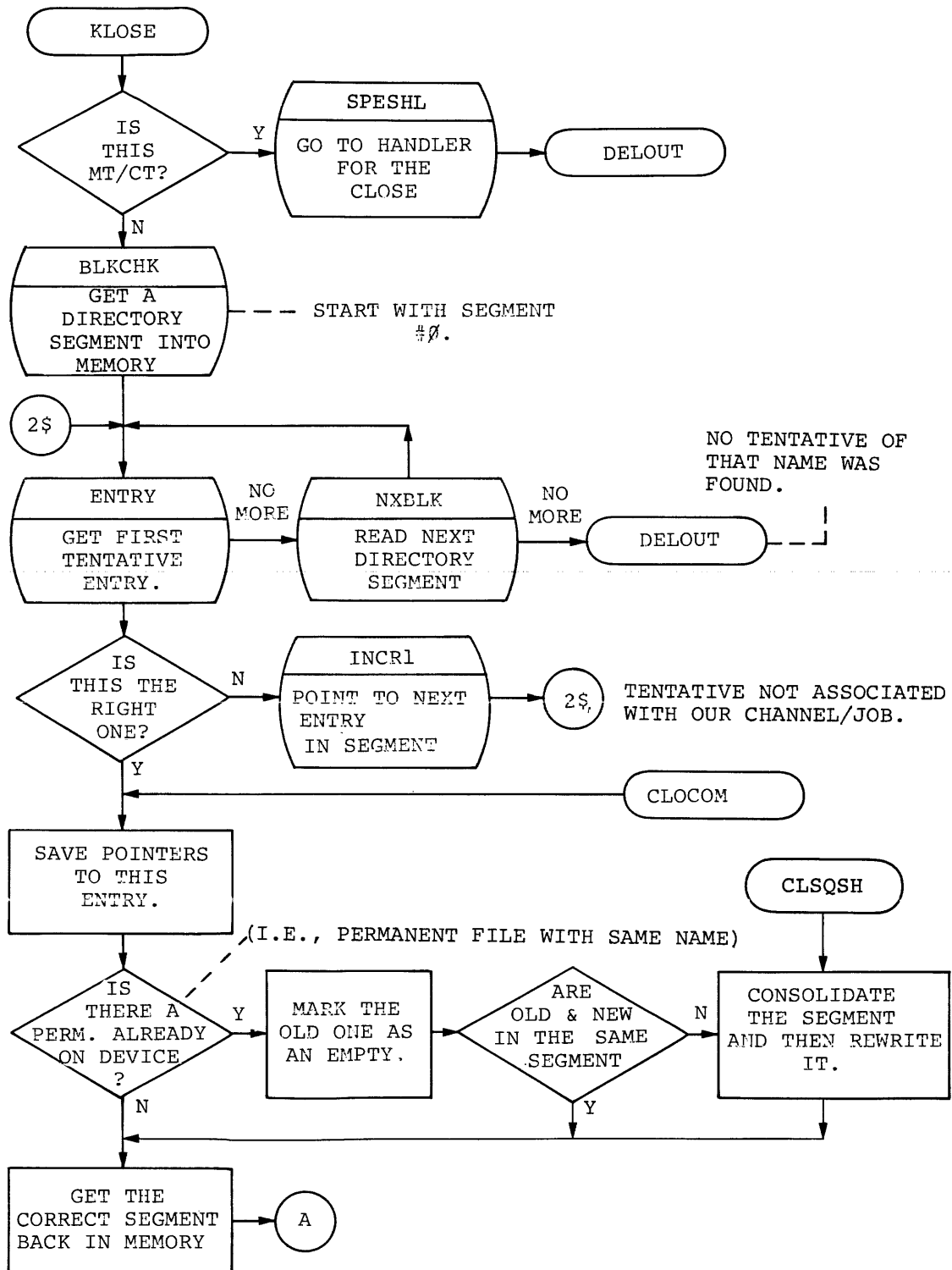
FETCH/RELEASE

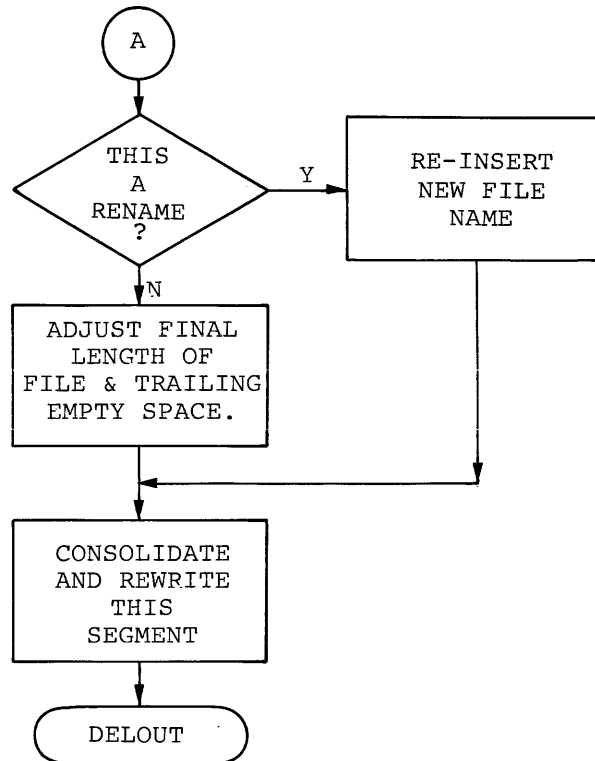


DSTAT/FETCH/RELEASE (CONT.)

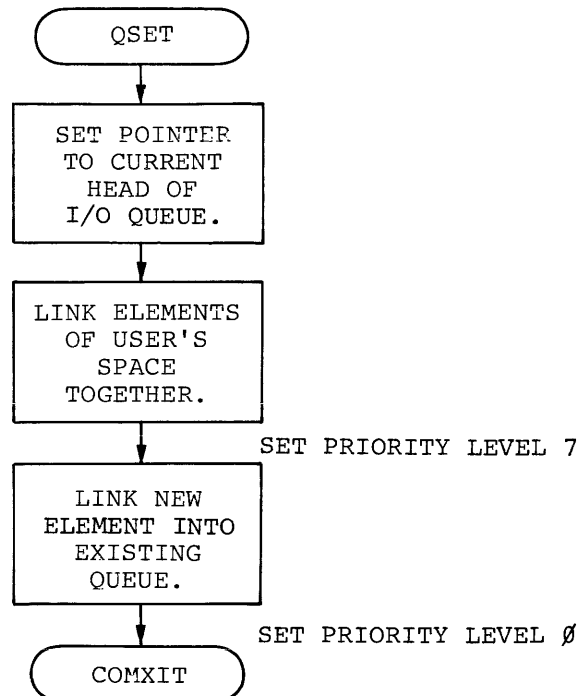


CLOSE



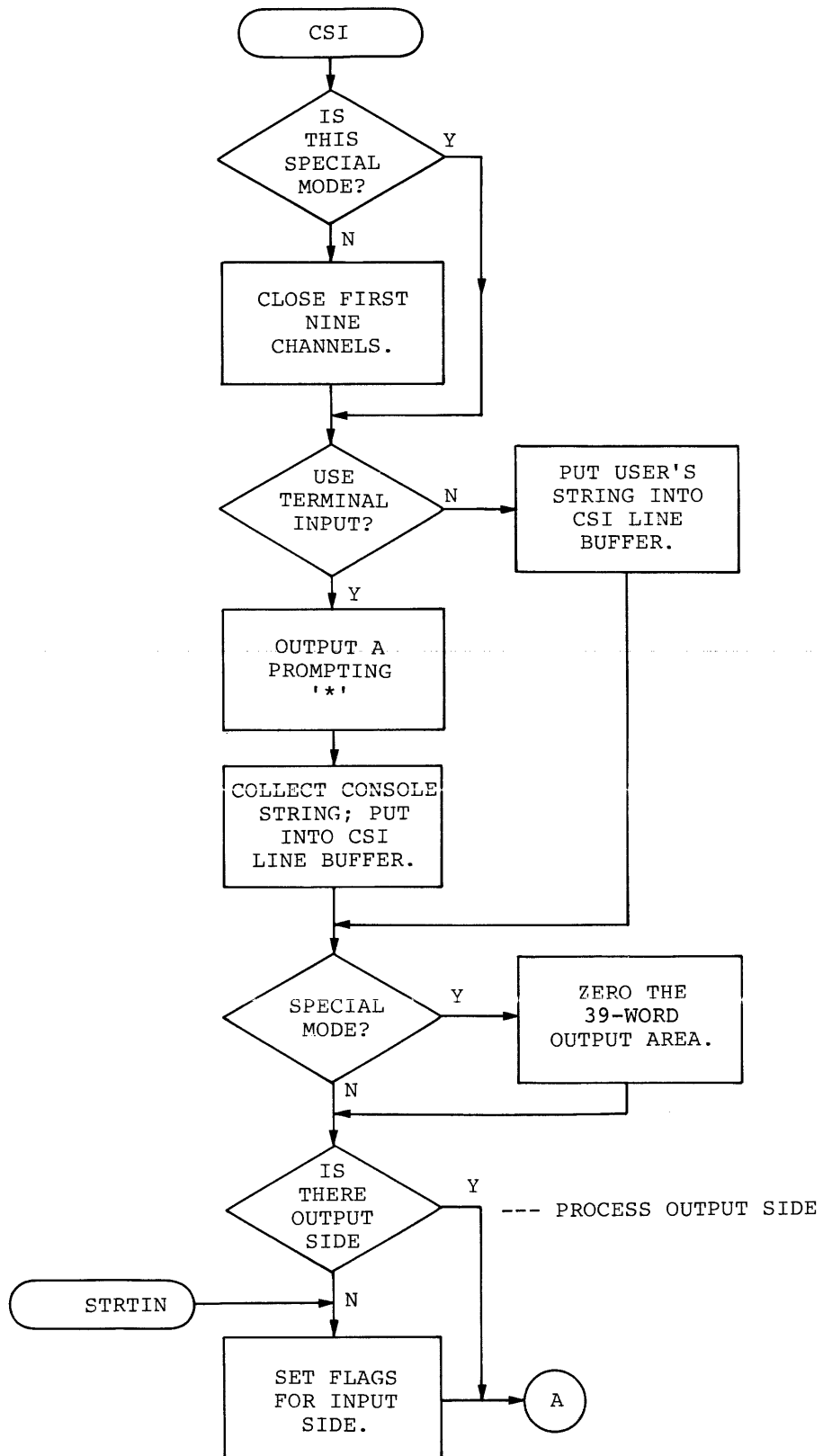


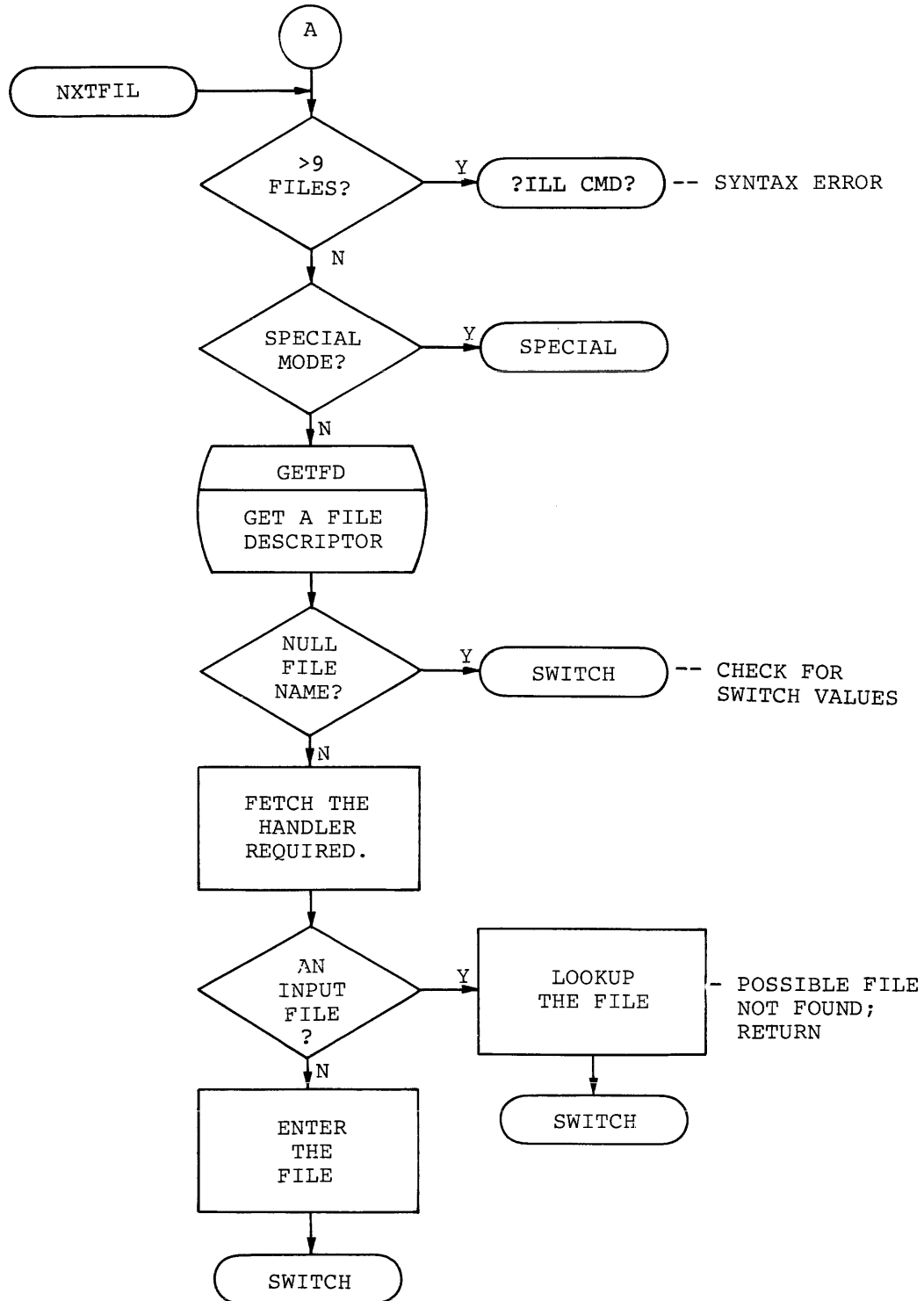
QUEUE EXTEND (QSET)



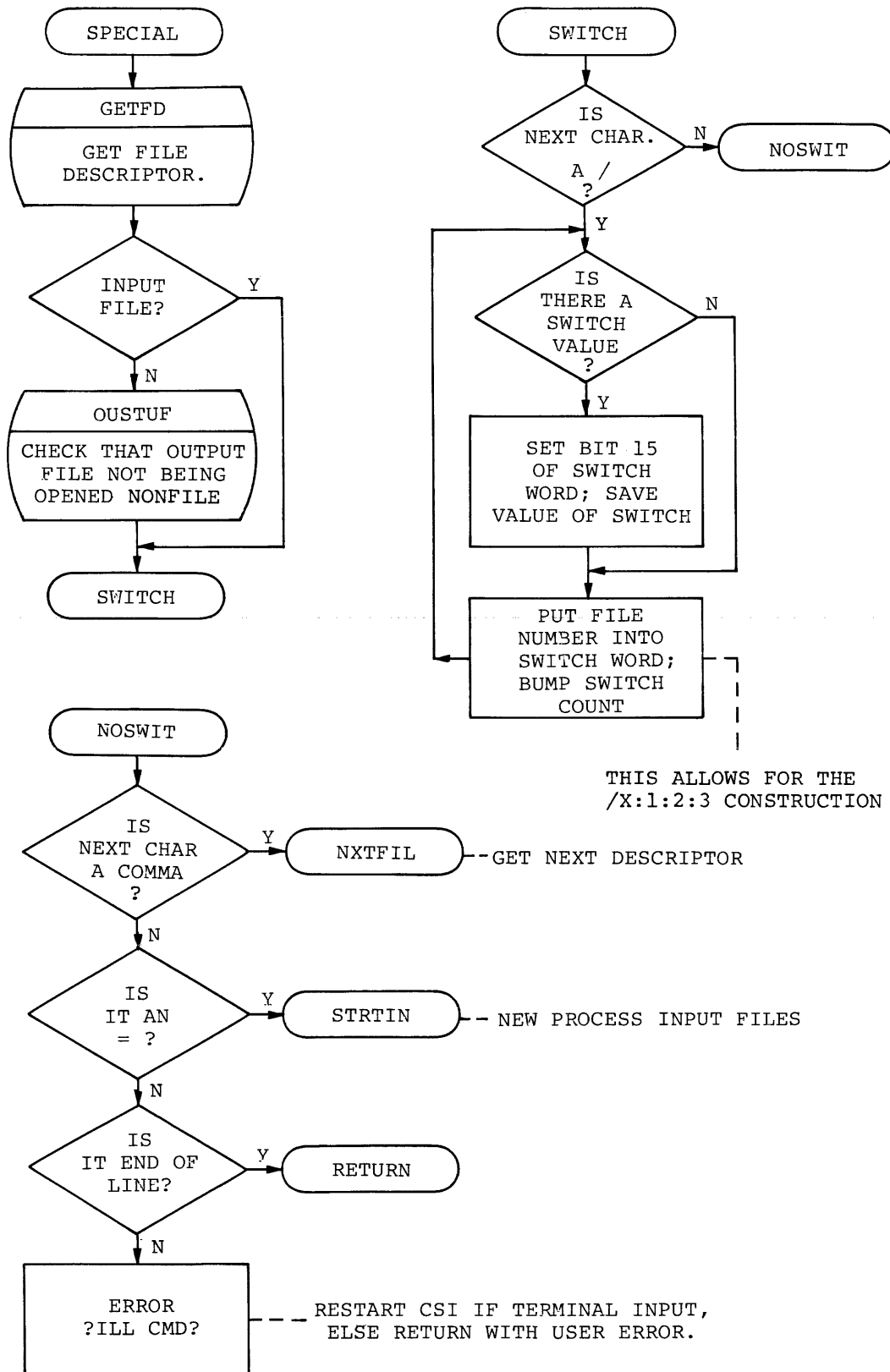
E.3 CSI (COMMAND STRING INTERPRETER) FLOWCHARTS

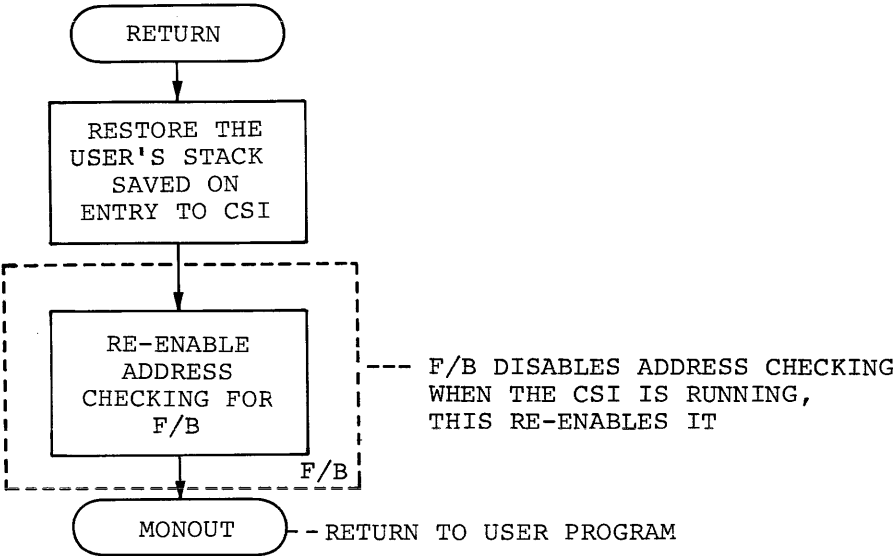
CSI CODE





CSI CODE (CONT.)



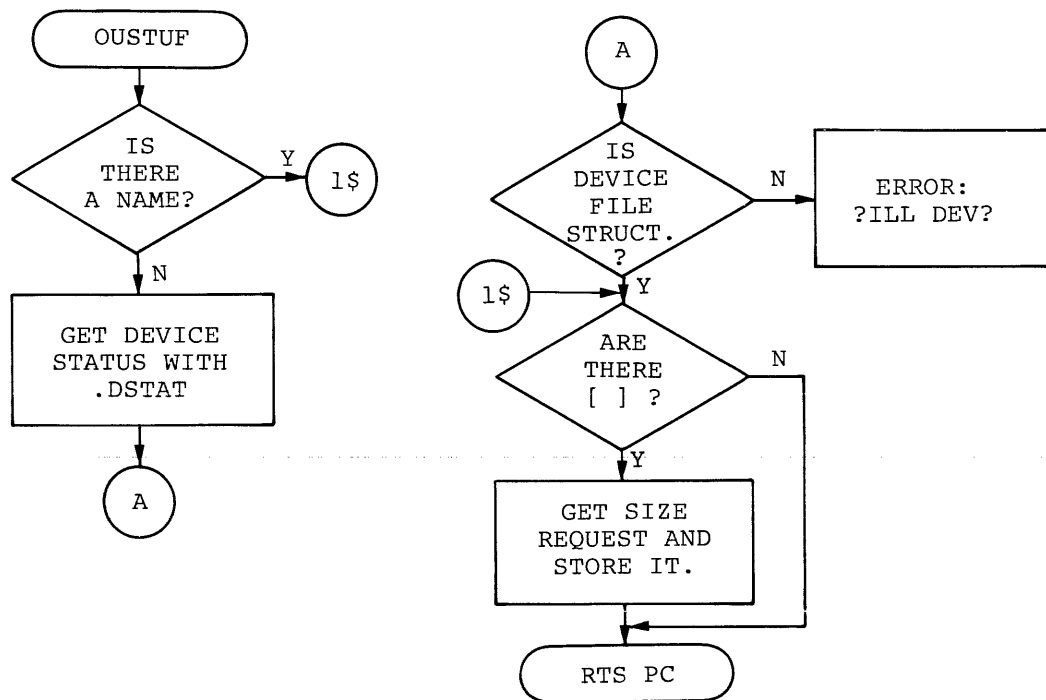


E.3.1 CSI Subroutines

These subroutines are used by the CSI, and, in certain cases by the KMON.

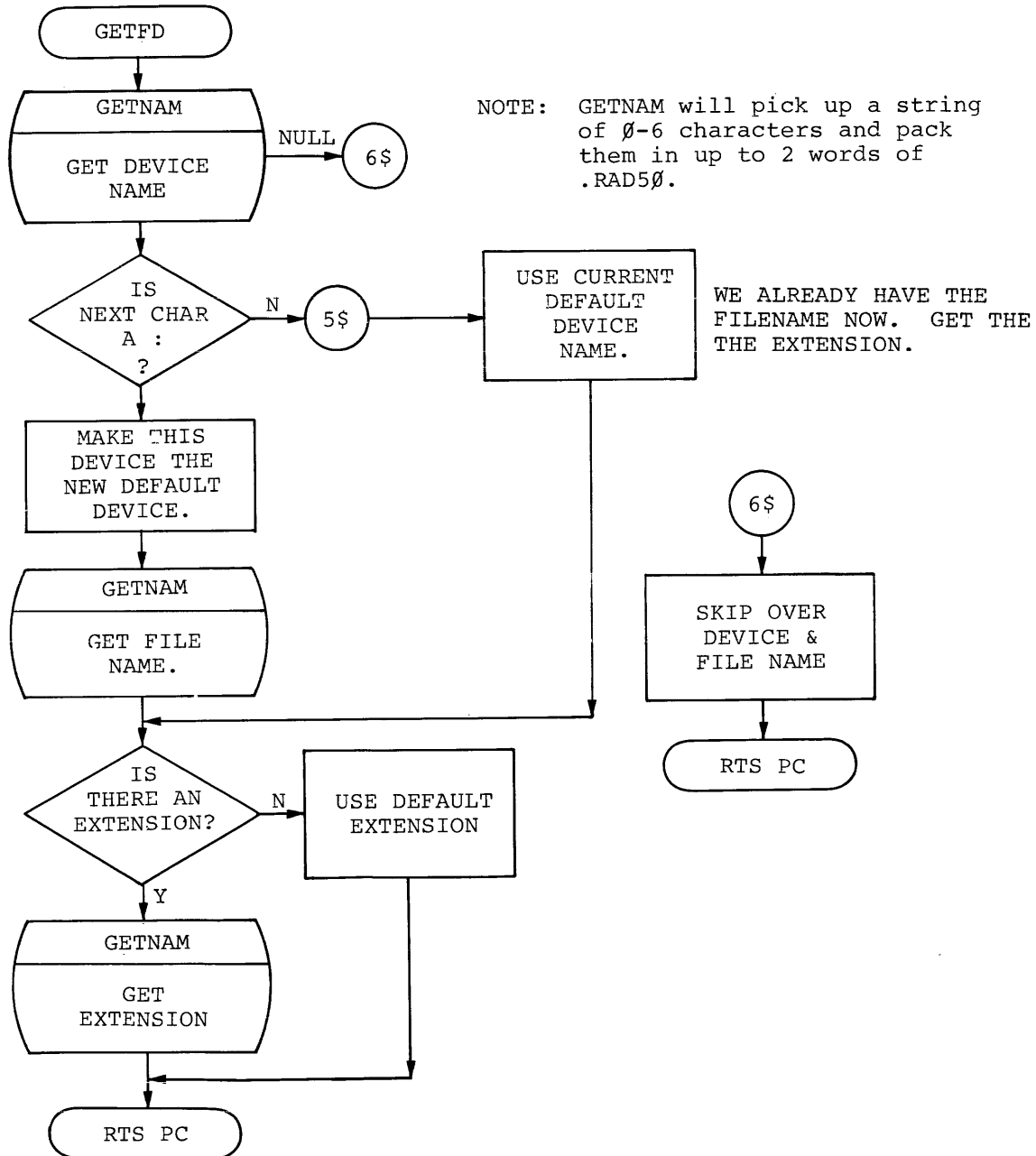
OUSTUF

OUSTUF - This routine verifies that an output descriptor has a file name. If not, a syntax error is generated. It also will scan off the size in [] if it was specified.



GETFD/GETNAM

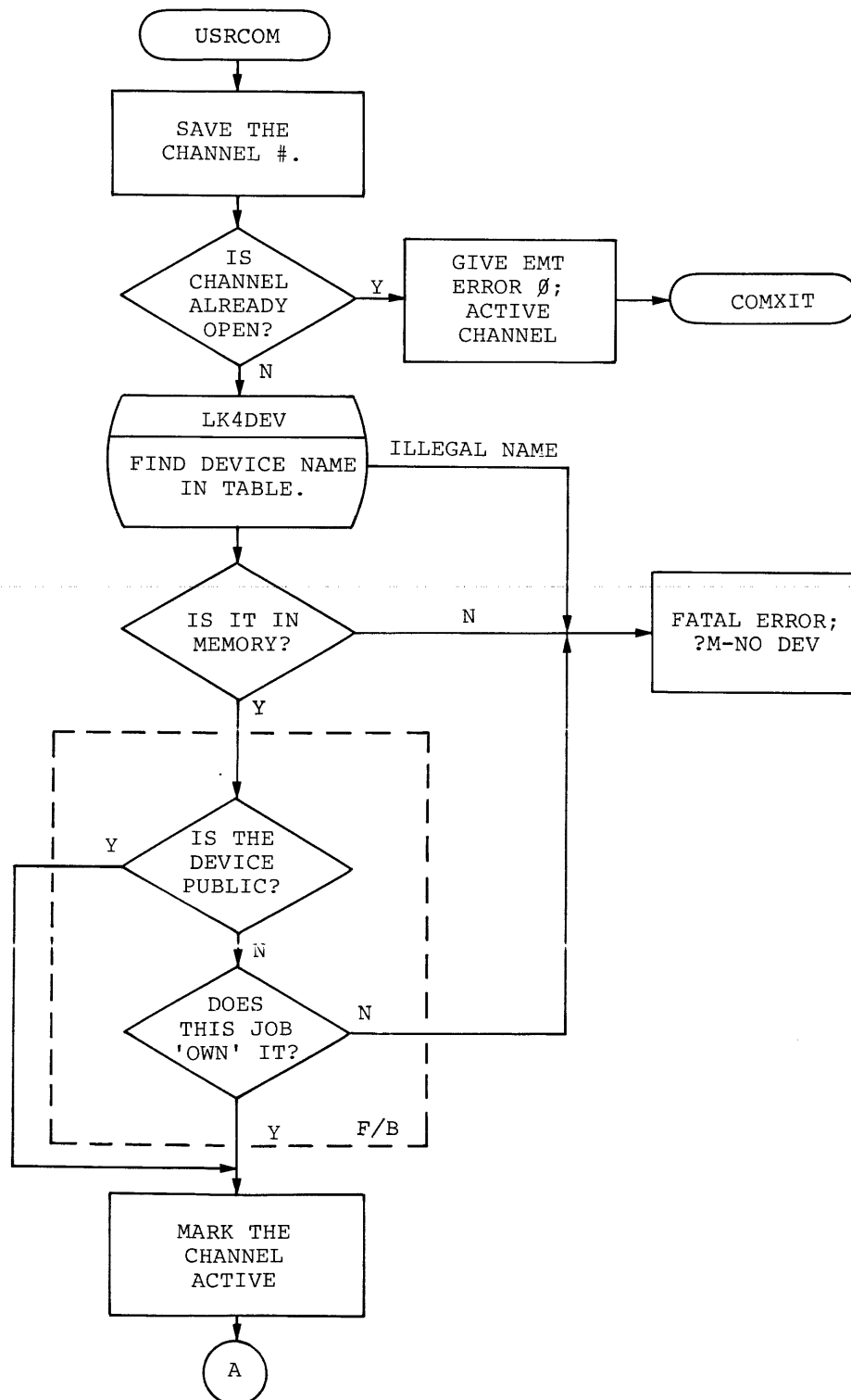
GETFD - Picks up a file descriptor (DEV:FILE.EXT) from an input string and packs it in 4 words of .RAD5Ø.

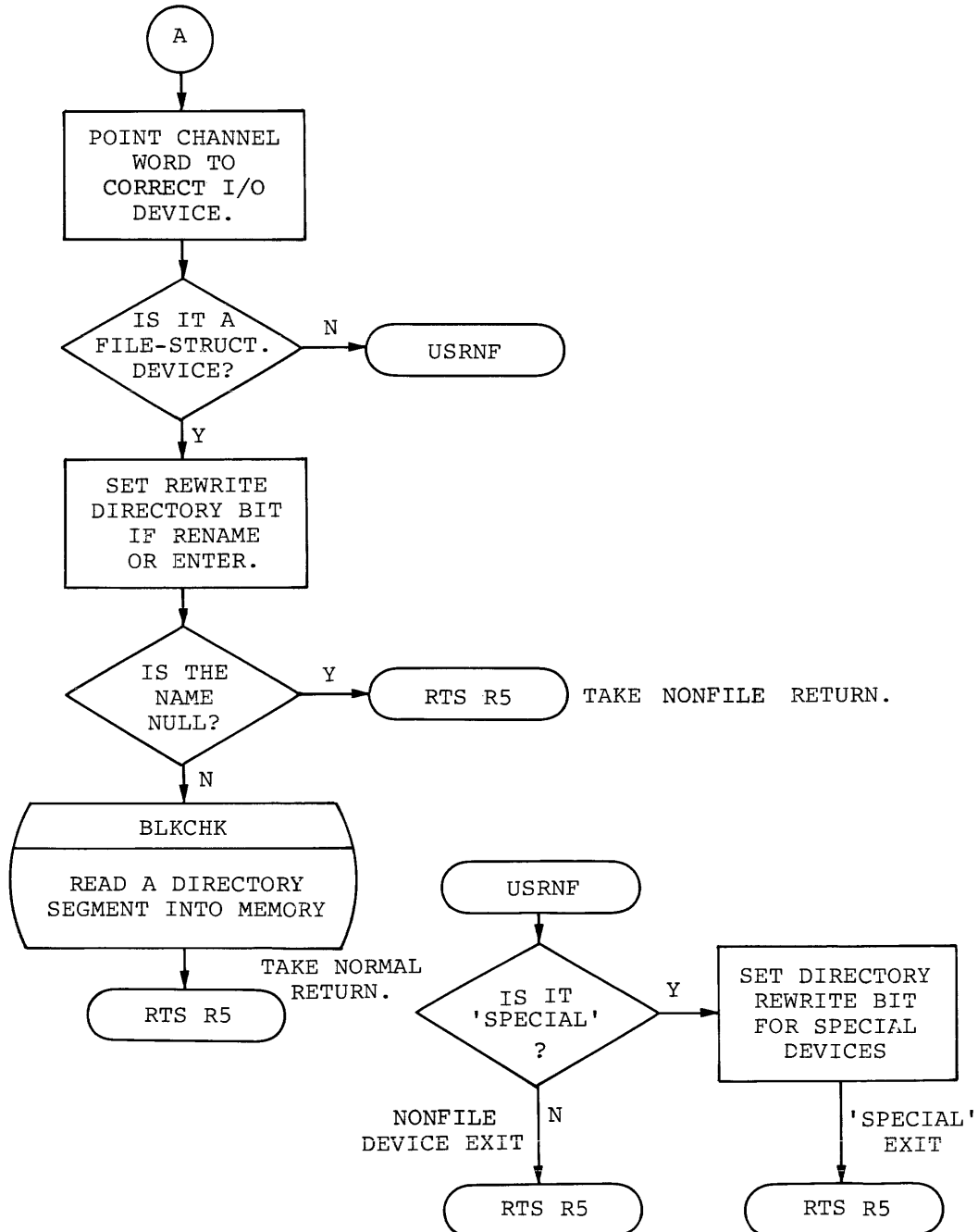


GETNAM - Converts a string of 0-6 alphanumeric characters to a 2-word RAD5Ø group. The two words are zero filled when necessary. See code at GETNAM in the source listing if greater detail is necessary.

USRCOM

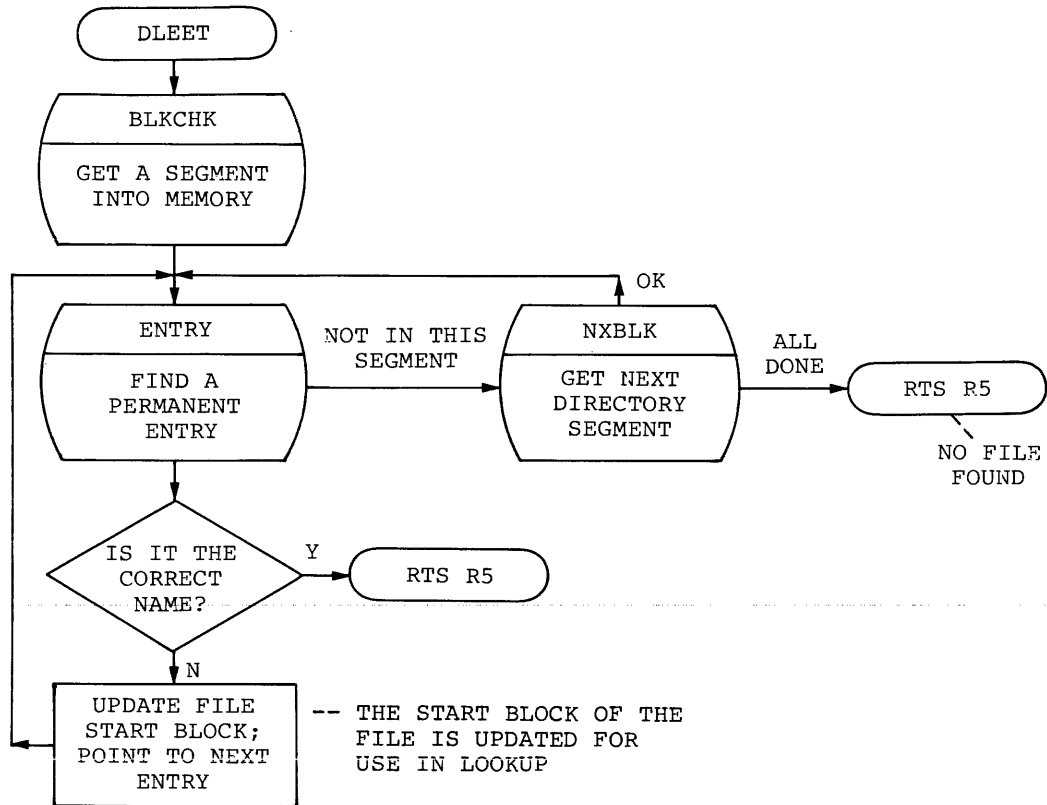
USRCOM - This routine is used to prepare a channel for I/O operations.



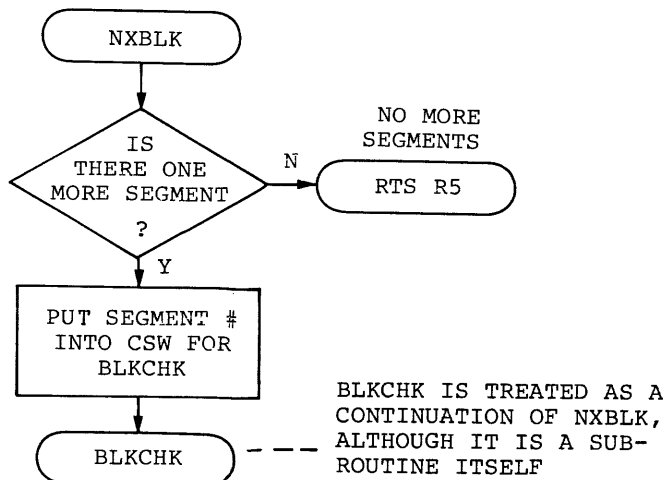


DLEET/NXBLK

DLEET - This routine scans a device directory to find a file of a specified name.

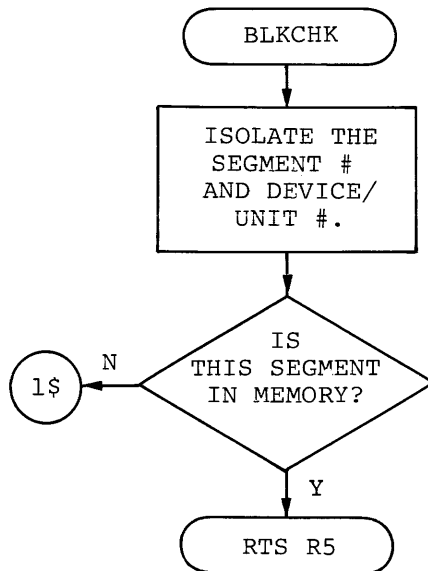


NXBLK - Gets the next in the series of directory segments, if one exists.

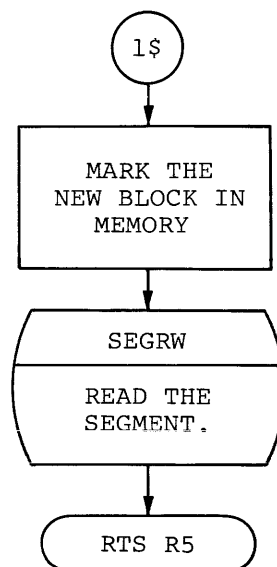


BLKCHK

BLKCHK - This routine isolates the segment number contained in bits 8-12 of the CSW, and checks to see if that segment is in memory at the current time. If not, it is read in.



Note that not only must the segment numbers agree, but also the device and unit numbers must be the same.



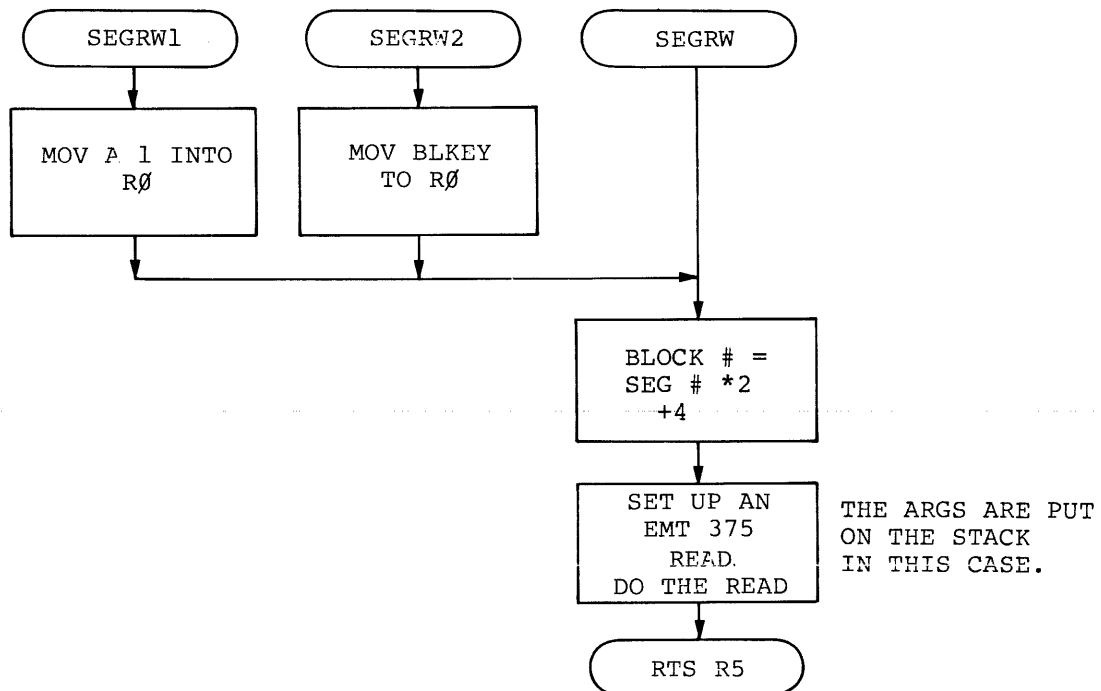
SEGRW

SEGRW - Segment Read/Write. This routine read/writes selected directory segments. There are three entry points:

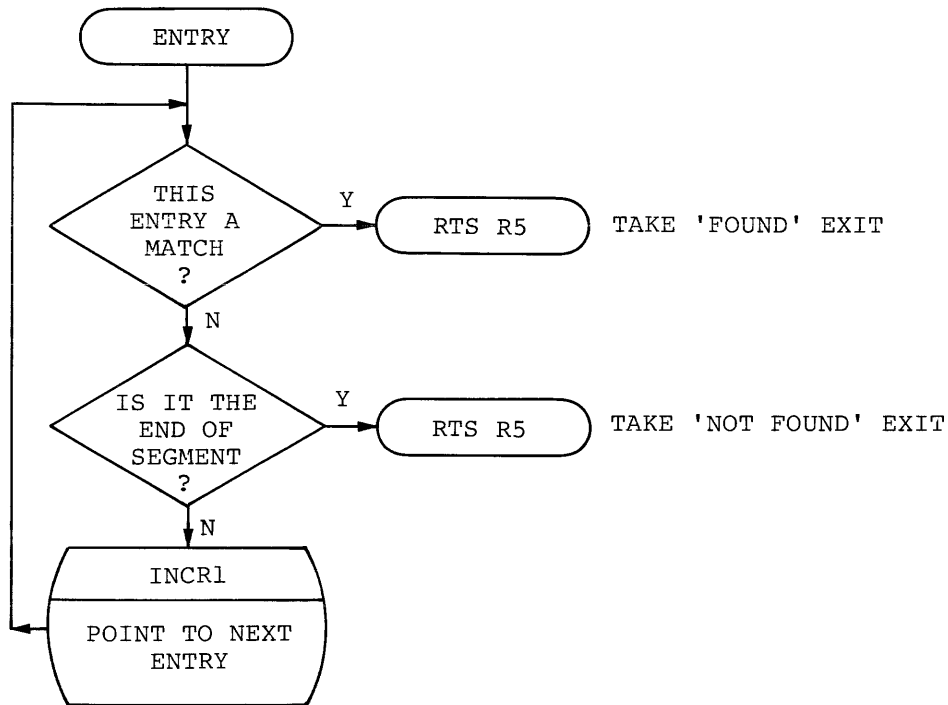
SEGRW1: Use segment #1

SEGRW2: Use the segment currently in memory (BLKEY)

SEGRW: Use the number in R0 as the segment #.



ENTRY - This routine uses R1 as a pointer into a directory segment to find a specified file type (Permanent, Tentative, Empty) or the end of segment mark.



INCR1 - This routine bumps R1 to the next entry in a directory segment.

COMERR - This routine generates a fatal error from the USR. The call is:

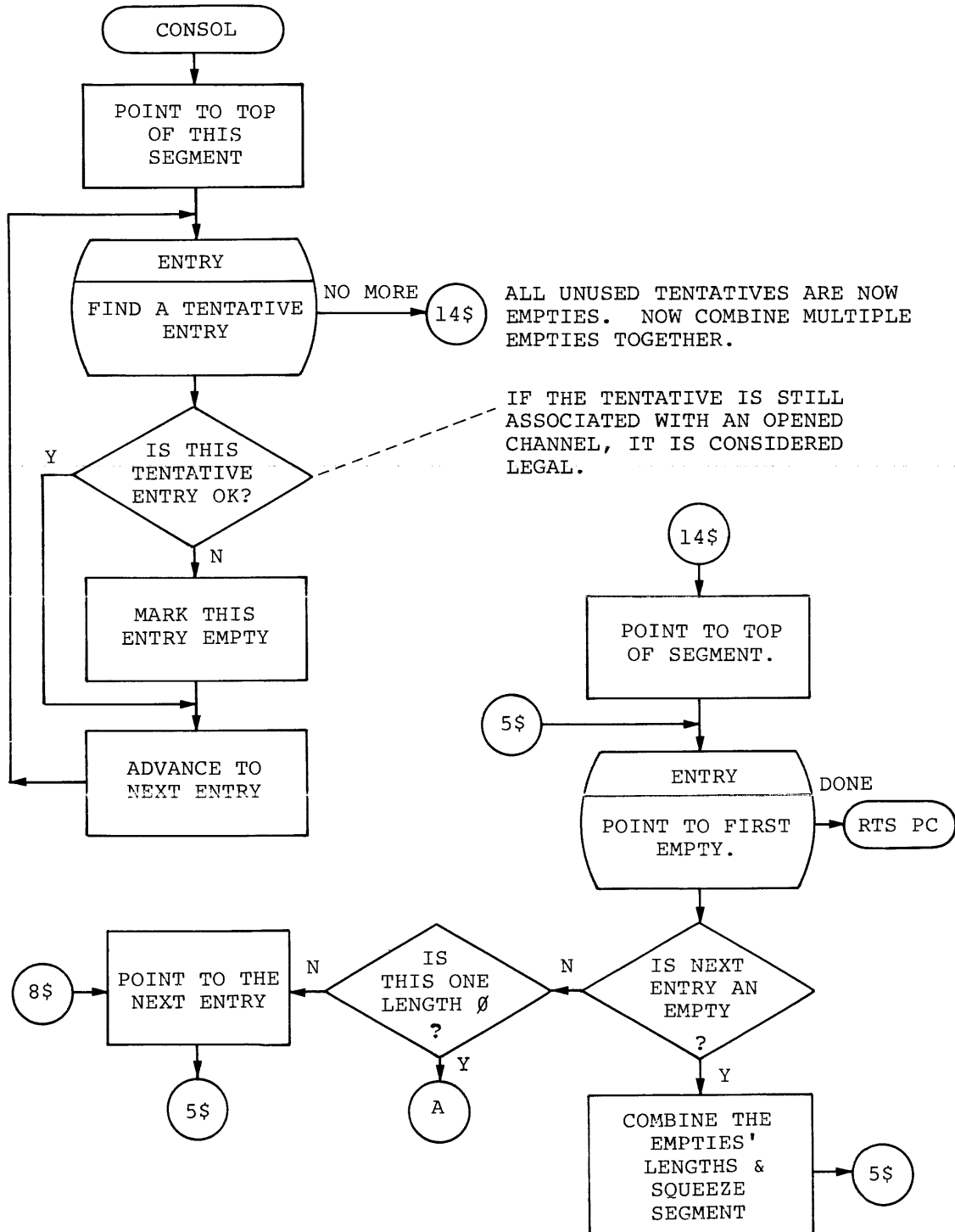
```
JSR R5,COMERR
code
```

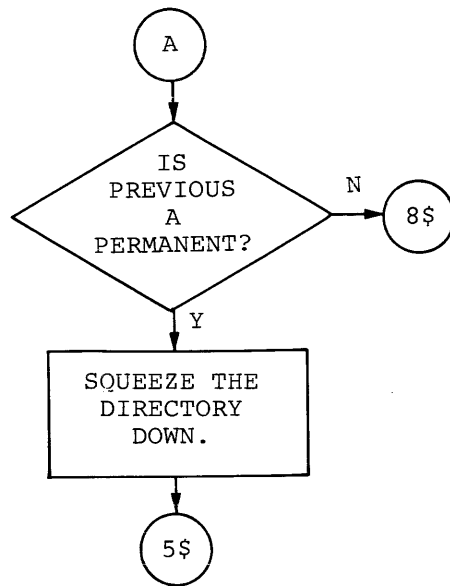
Code is used to indicate which error is to be generated. If .SERR is in effect, control passes to COMXIT, which returns to RMON.

SPESHL - This routine is used to effect file operations on MT/CT. This is done by passing a READ request to the Q manager. The even byte of the completion function will contain a 377. The queue manager detects this, and modifies the I/O queue element to indicate that the handler should perform a USR function.

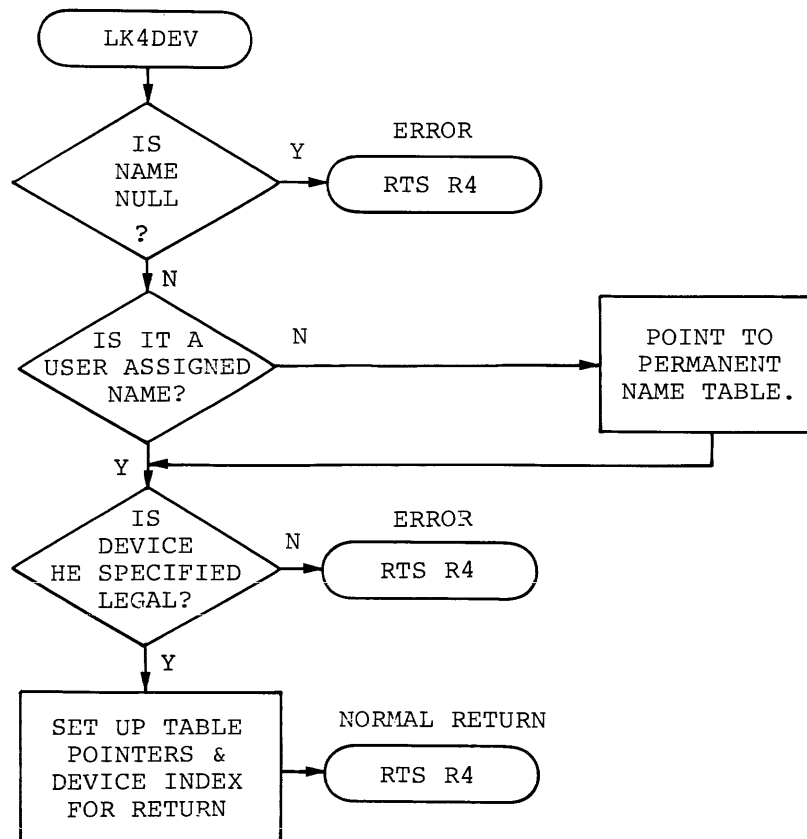
CONSOL

CONSOL - This routine is used to compact a directory segment. It combines consecutive empties into one, and makes empties out of tentative files which are not associated with an active channel.





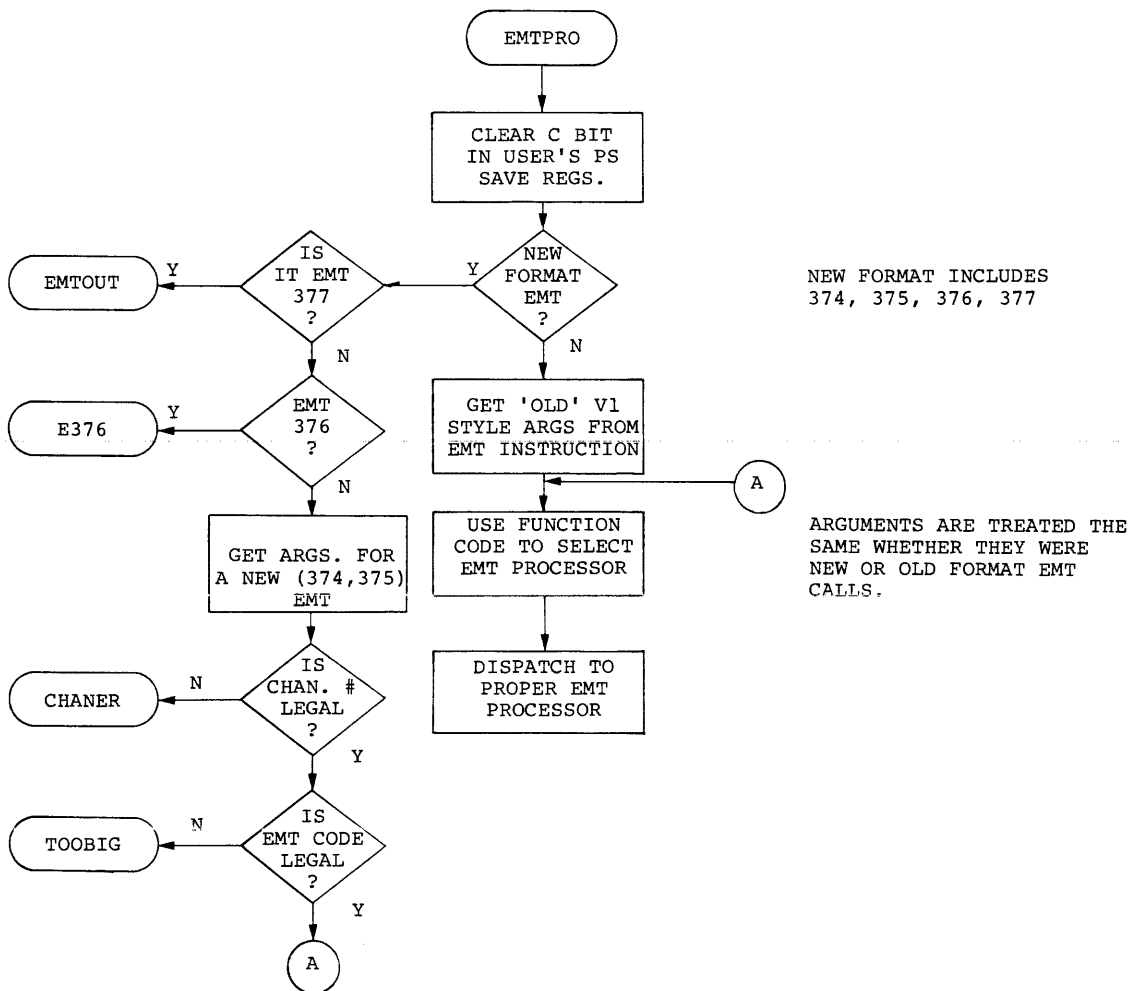
LK4DEV - This routine looks up a specified device name in the system tables. It first attempts to find the name in the user assigned name table; failing that, the permanent name table is searched.



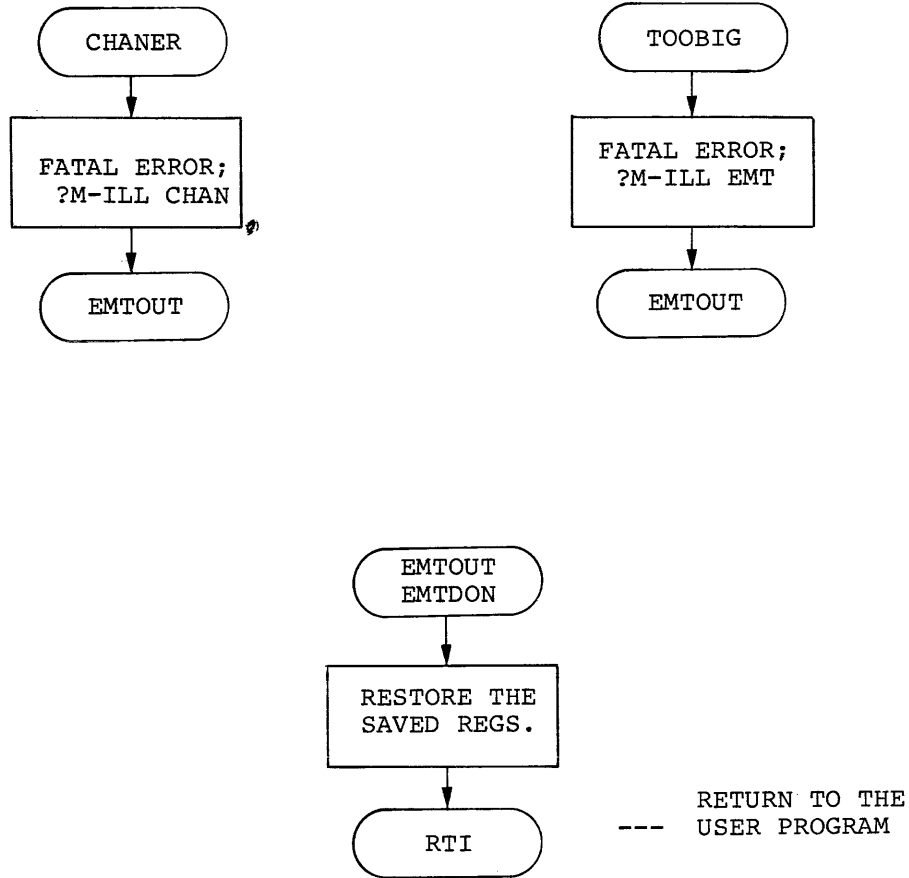
E.4 RMON (RESIDENT MONITOR) FLOWCHARTS FOR SINGLE-JOB MONITOR

EMT DISPATCHER

The code of the EMT dispatcher is entered when an EMT instruction is executed. The EMT instruction is decoded and control passes to the appropriate code for processing.



EMT DISPATCHER (CONT.)



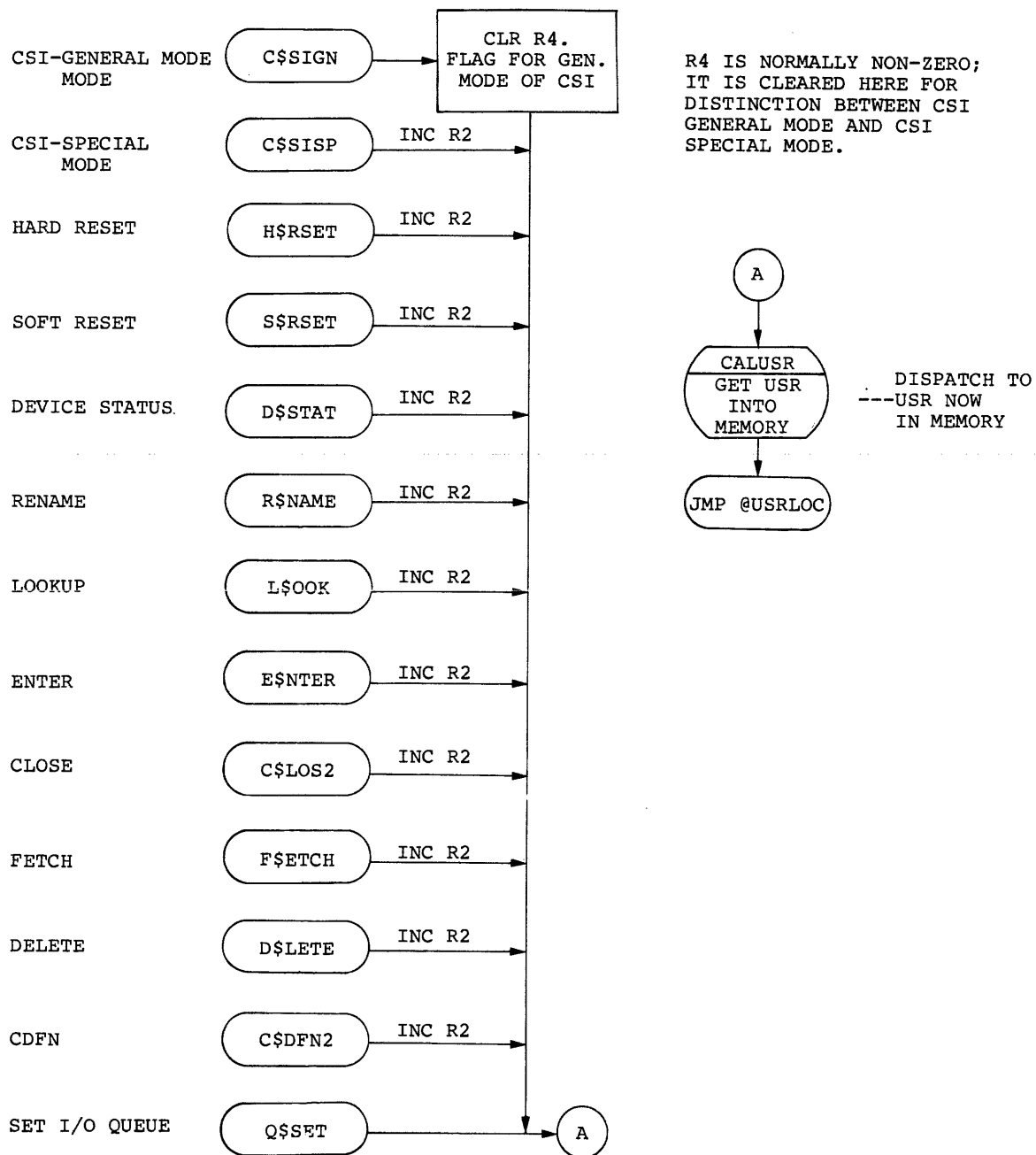
The following EMT requests are no-ops in the S/J Monitor:

Mark Time	.MRKT
Cancel Mark Time	.CMKT
Timed Wait	.TWAIT
Send Data	.SDAT
Receive Data	.RCVD
Channel Status	.CSTAT
Protect Vectors	.PROTECT
Channel Copy	.CHCOPY
Special Device	.DEVICE

Executing these requests in S/J will cause an immediate successful returns with no action taken.

USR DISPATCHER TABLE FOR EMT'S 340-357

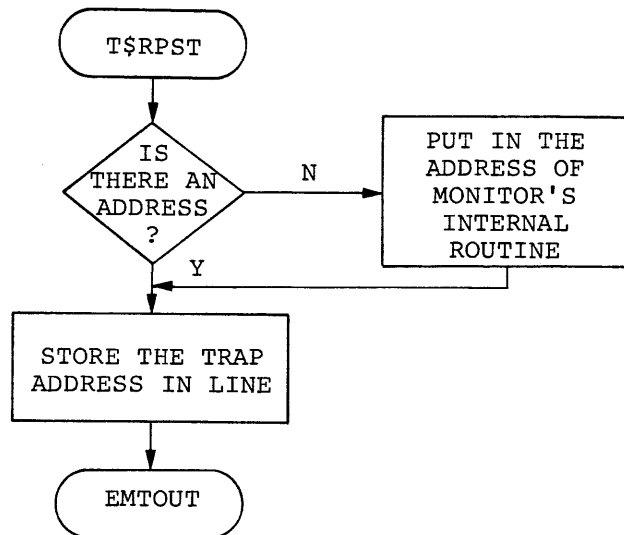
The USR Dispatch code handles dispatching those EMT's which require the USR. At each entry point, an INC R2 is performed. Thus, R2 acts as a function identifier once the USR is entered.



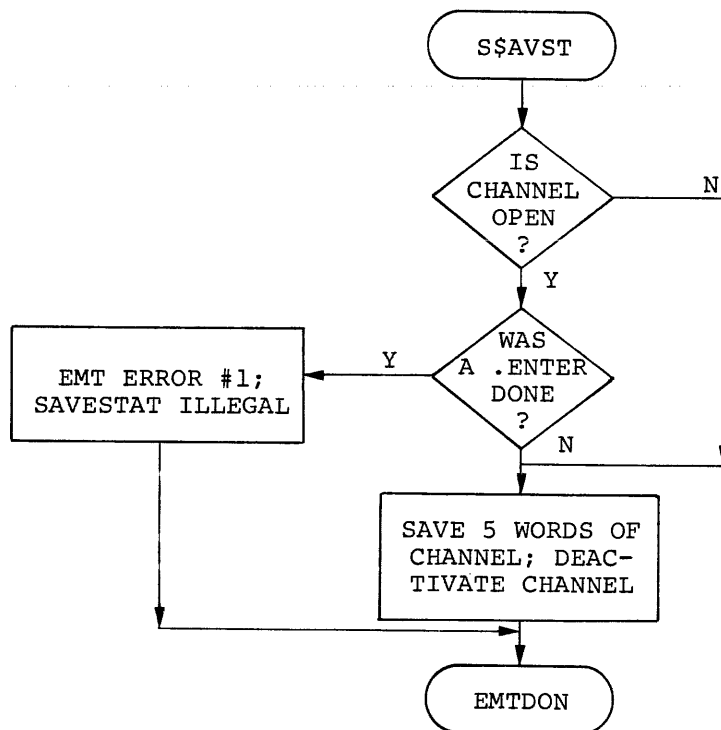
E.4.1 EMT Processors

SET TRAP/SAVE STATUS

SET TRAP ADDRESS

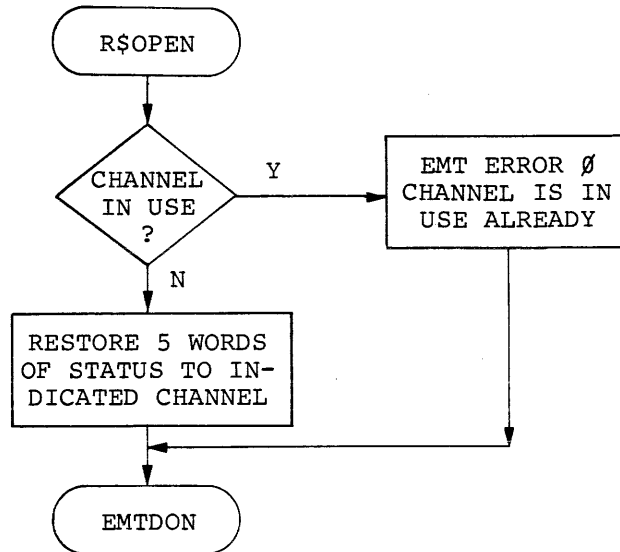


SAVESTATUS

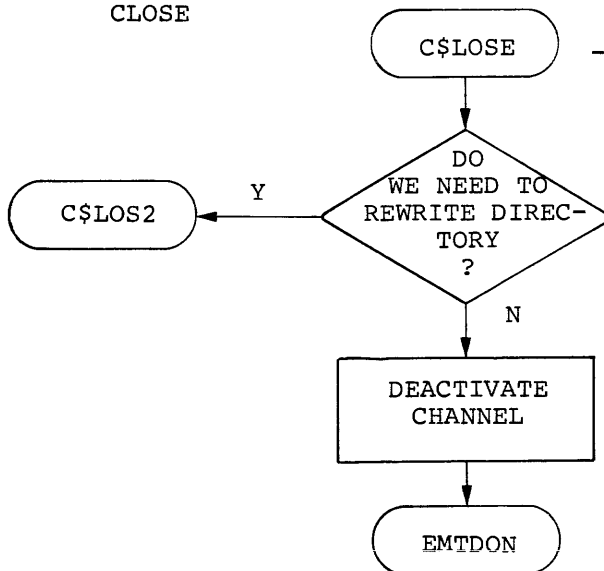


REOPEN/CLOSE/RDOVLY

REOPEN

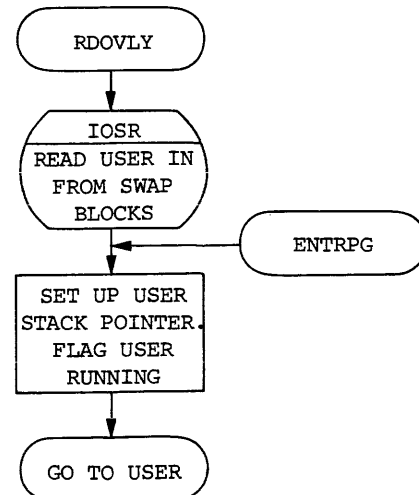


CLOSE

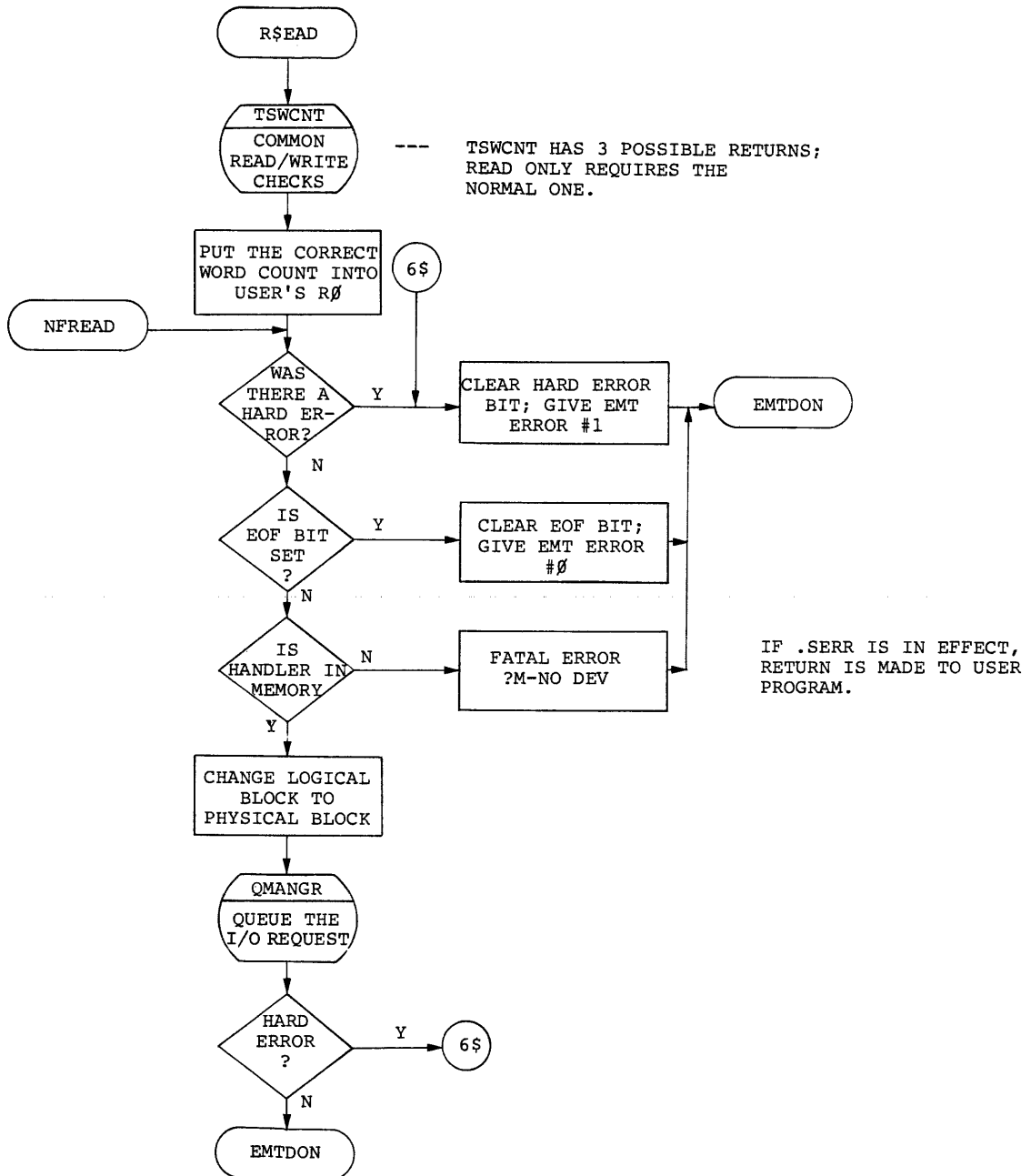


--- IF A LOOKUP WERE DONE, THE
USR IS NOT REQUIRED.

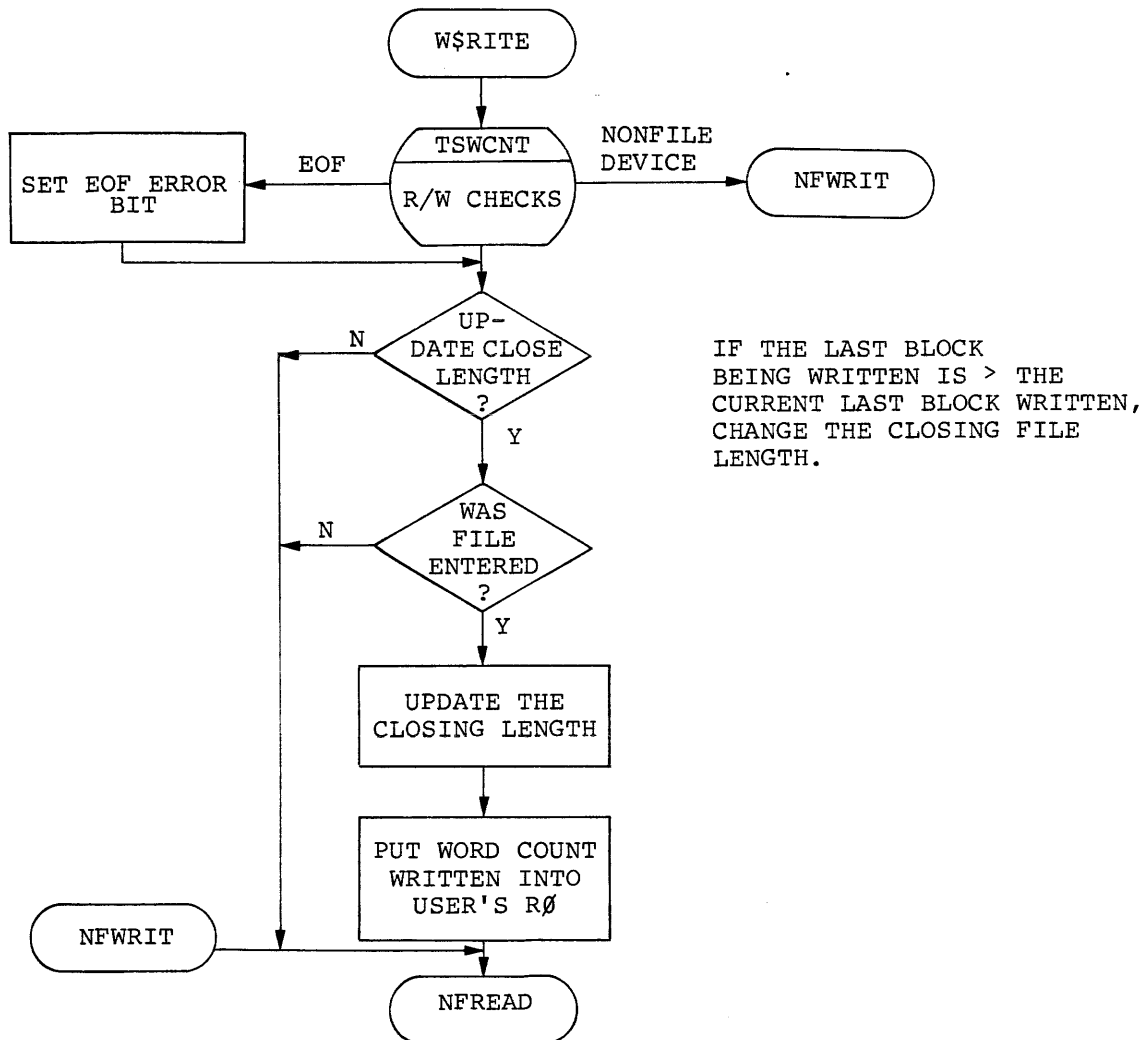
RDOVLY



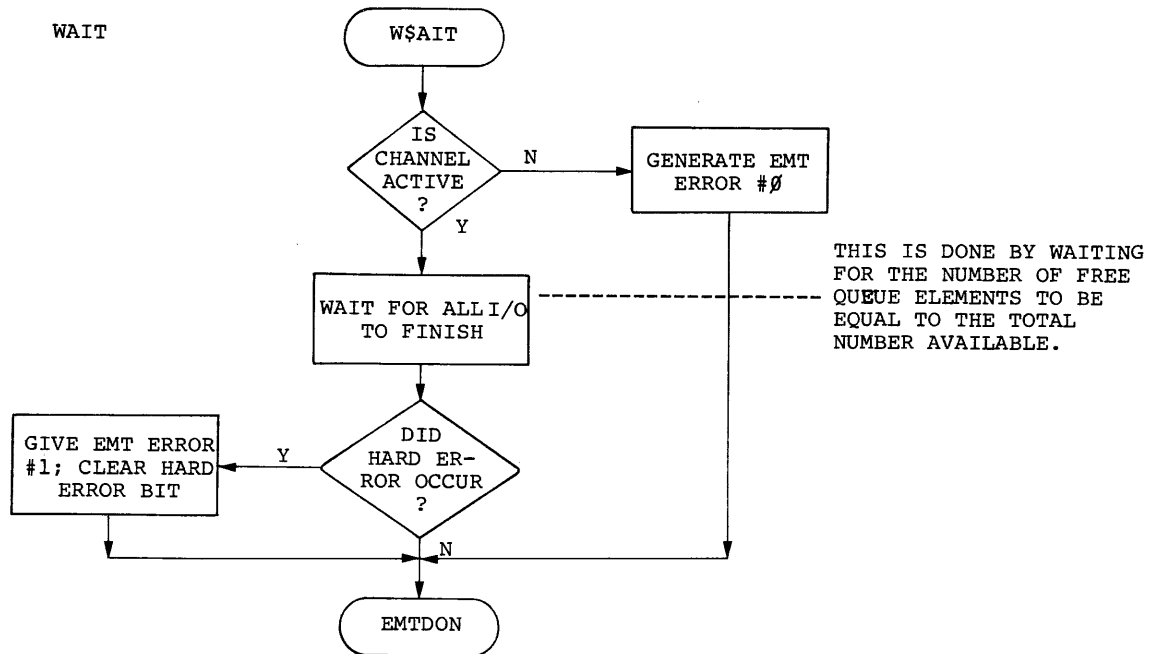
READ



WRITE

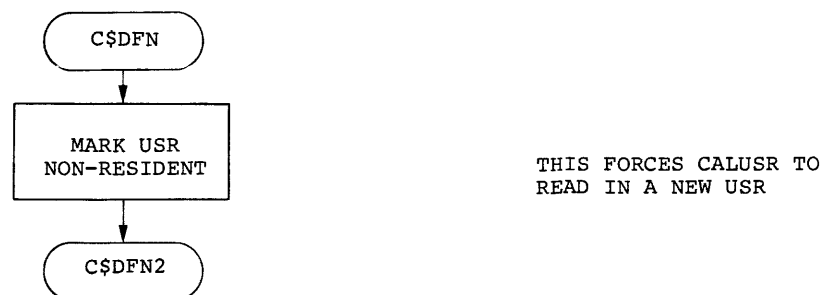


WAIT/CDFN



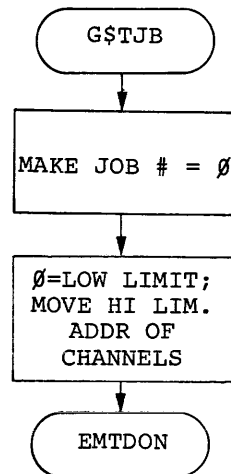
CDFN

Channel Define - the resident portion of CDFN causes a fresh copy of the USR to be read in, then enters the USR.

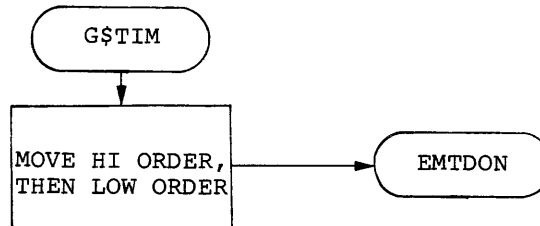


GET JOB PARAMETERS
GET TIME OF DAY
SET FPP EXCEPTION

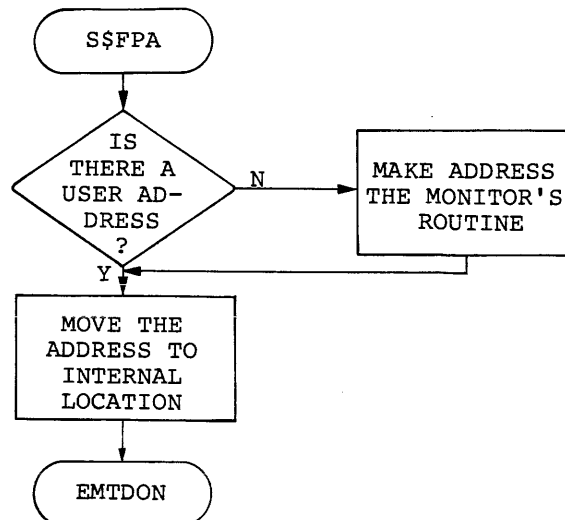
GET JOB PARAMETERS



GET TIME OF DAY

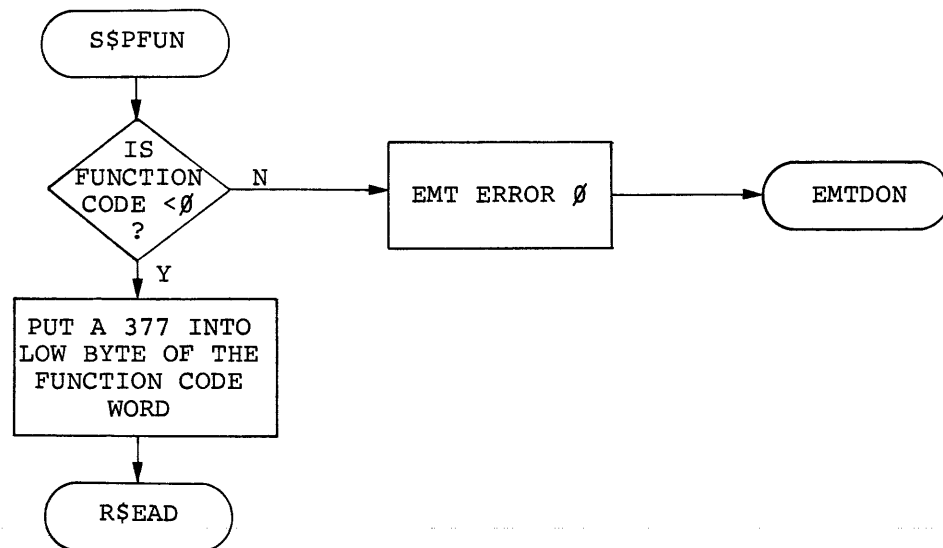


SET FPP EXCEPTION



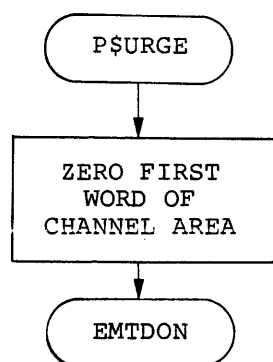
SPECIAL FUNCTIONS/PURGE
SOFT/HARD ERRORS

SPECIAL FUNCTIONS (MAGTAPE/CASSETTE)

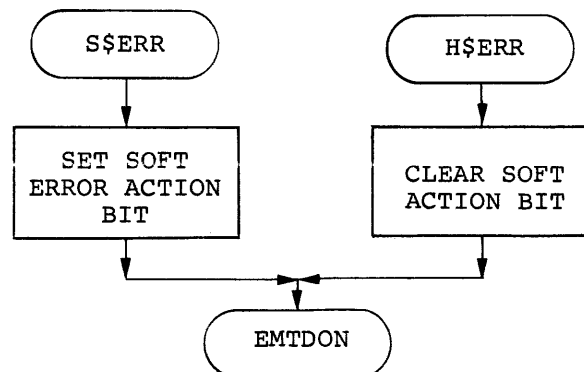


SPECIAL FUNCTIONS/PURGE
SOFT/HARD ERRORS

PURGE

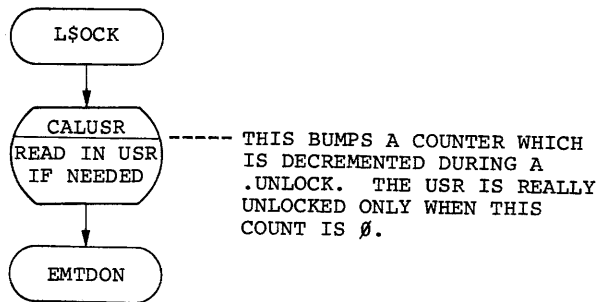


SOFT/HARD ERRORS

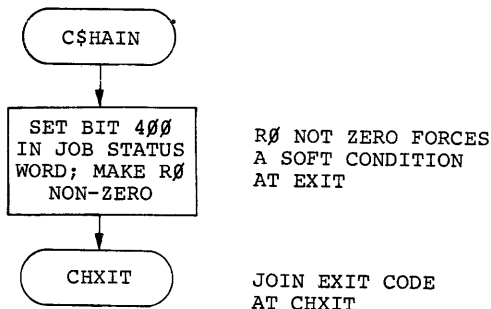


LOCK USR/CHAIN/UNLOCK USR

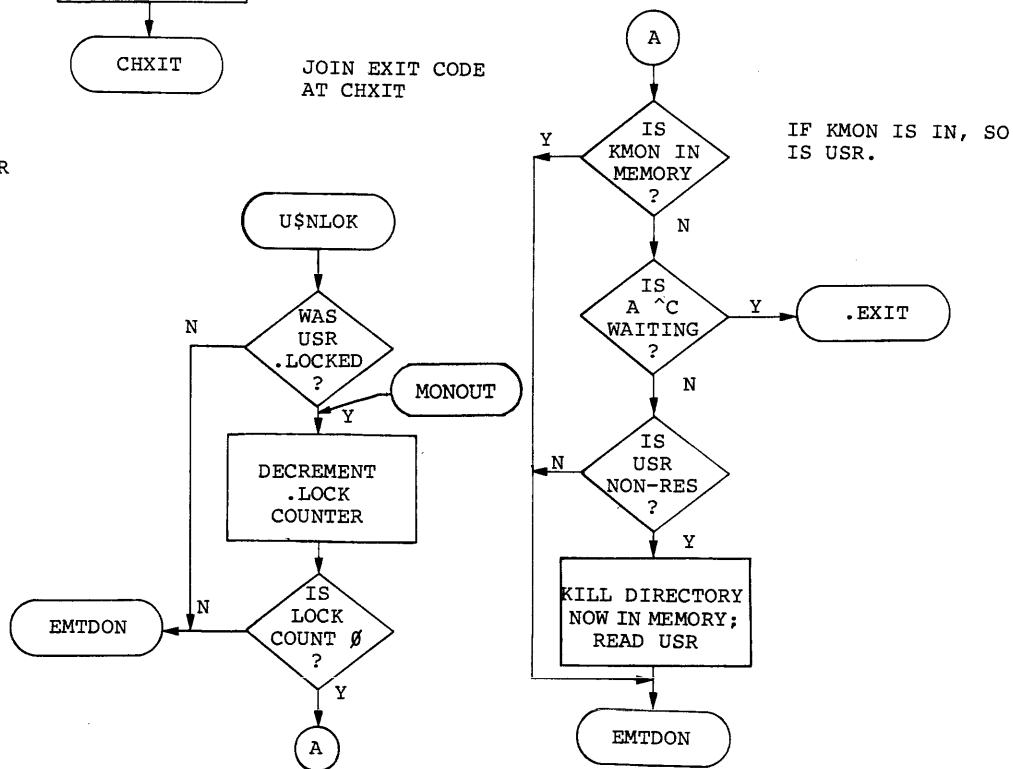
LOCK USR



CHAIN

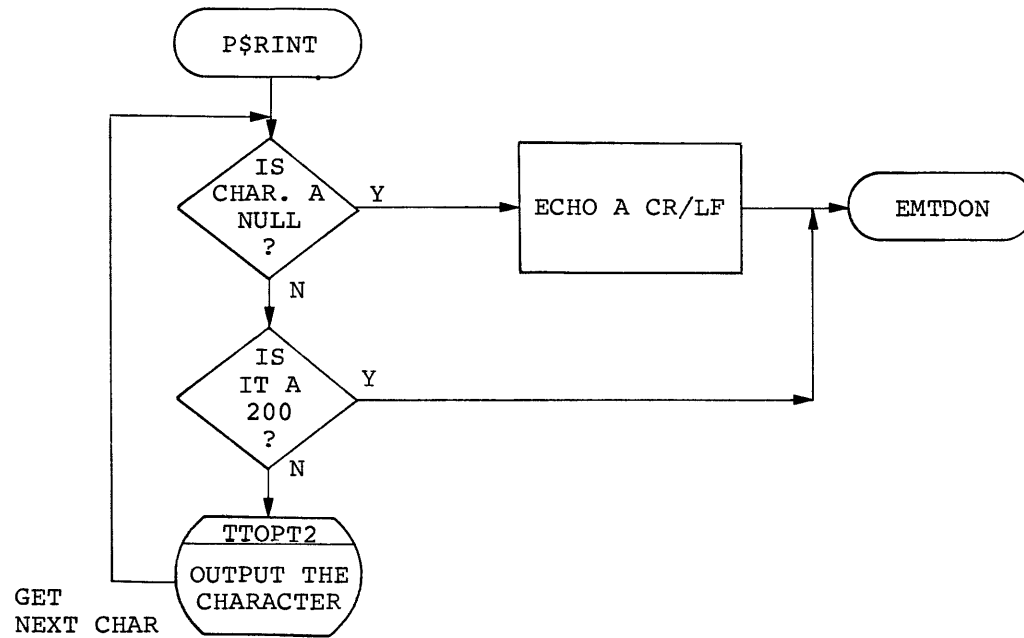


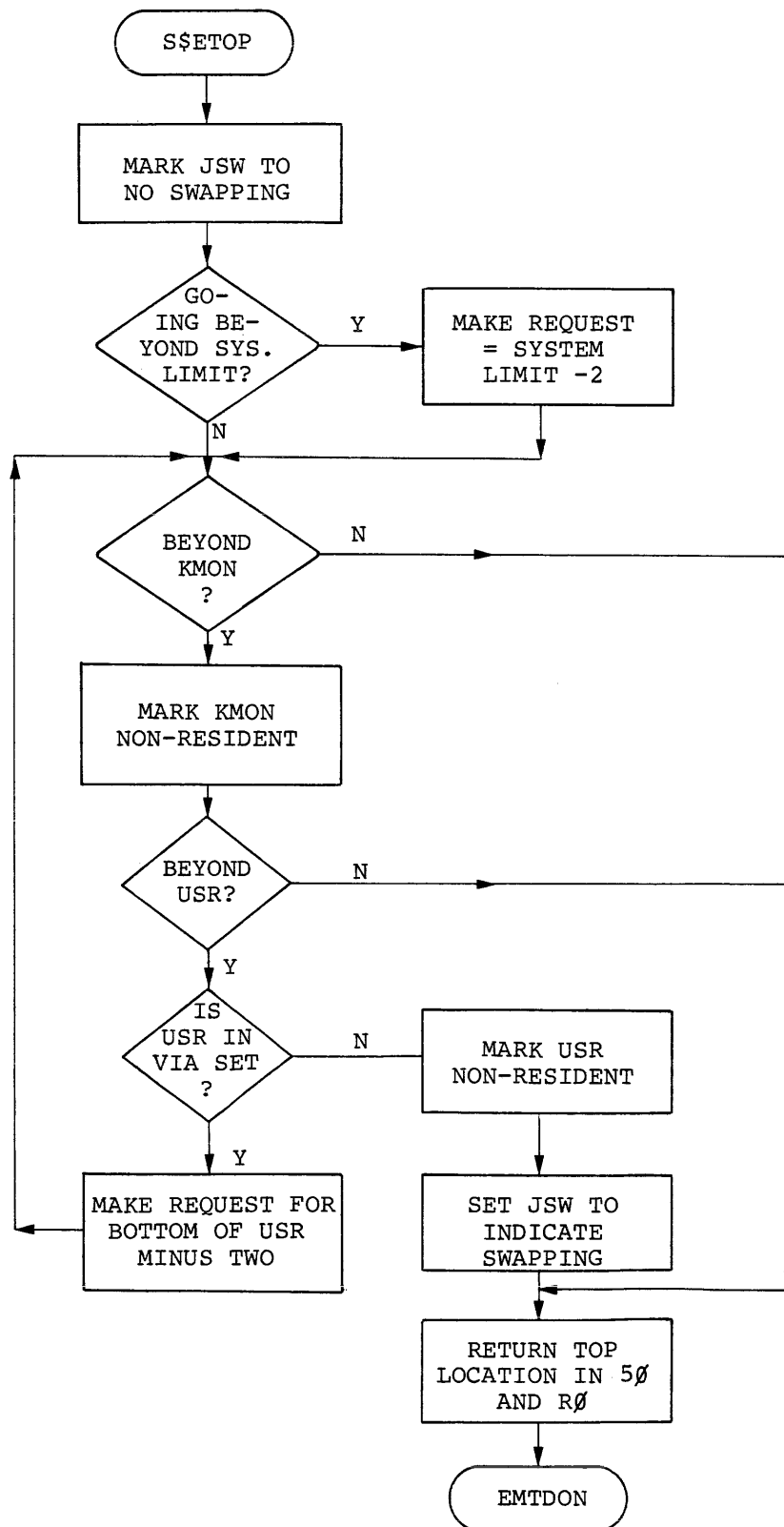
UNLOCK USR



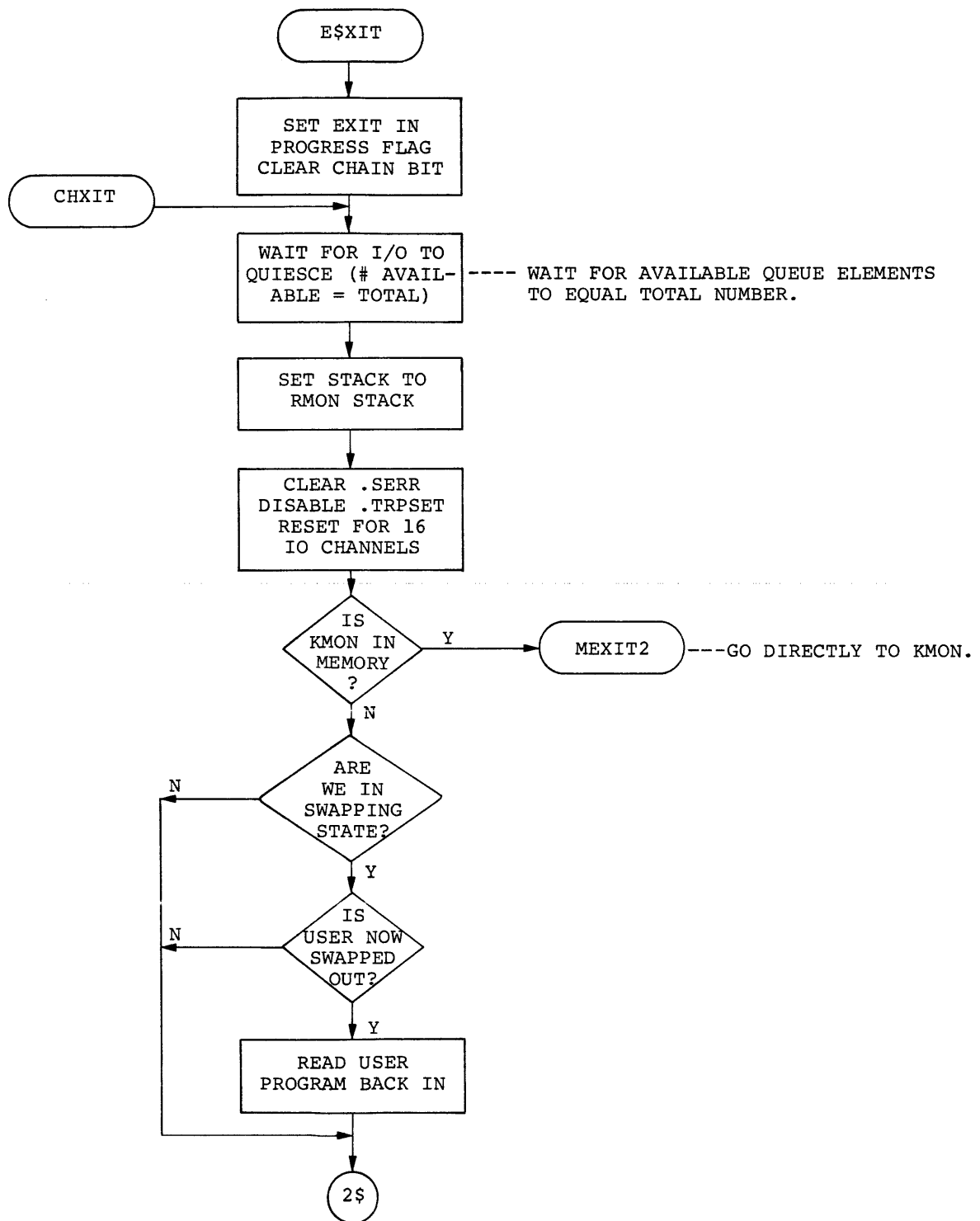
PRINT

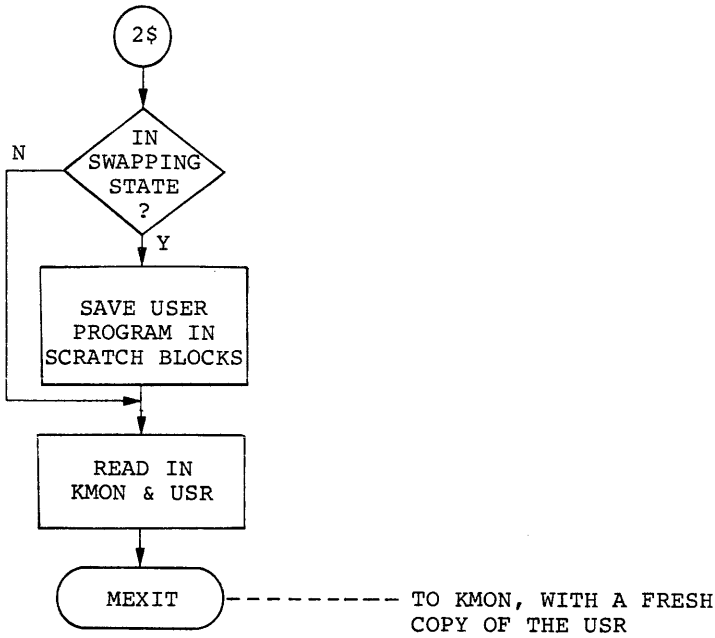
PRINT - Causes a line to be output to the console terminal.



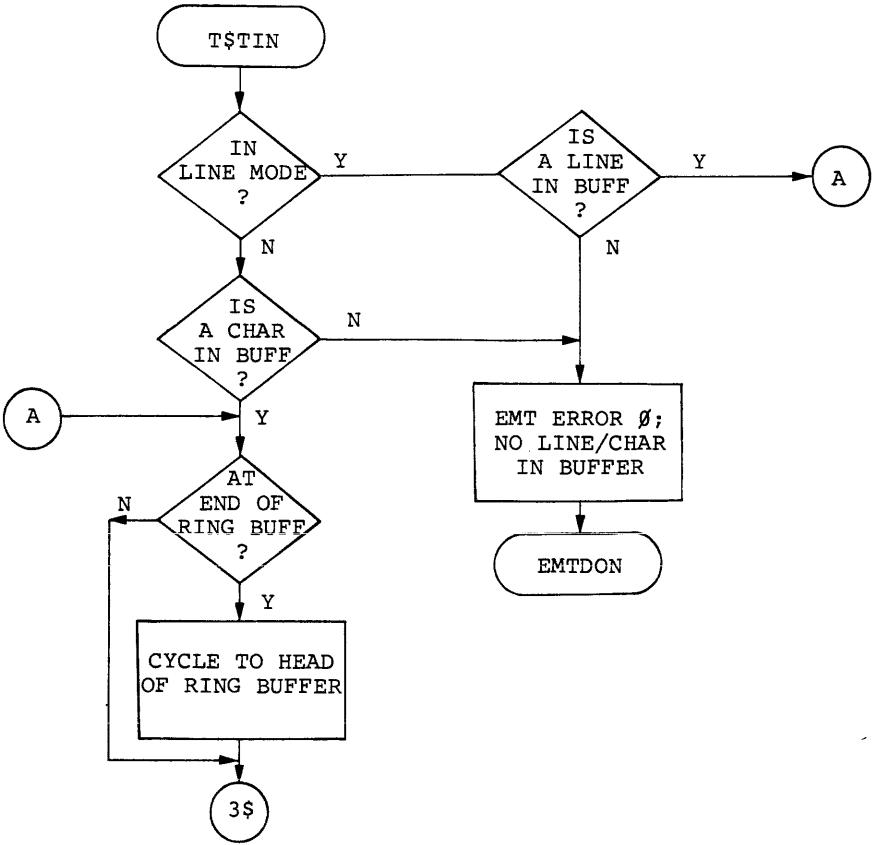


EXIT

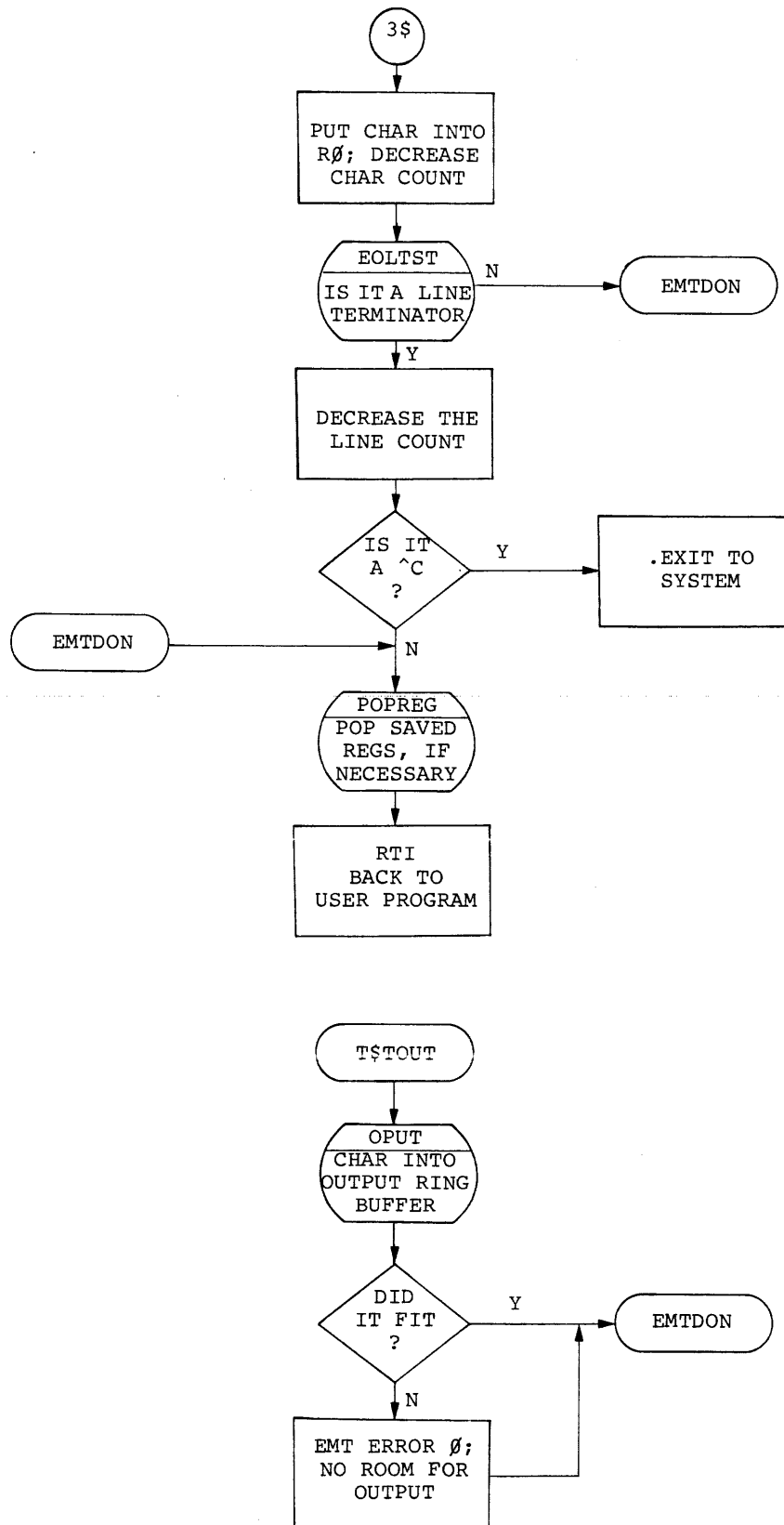




TTYIN



TTYIN (CONT.)/TTYOUT

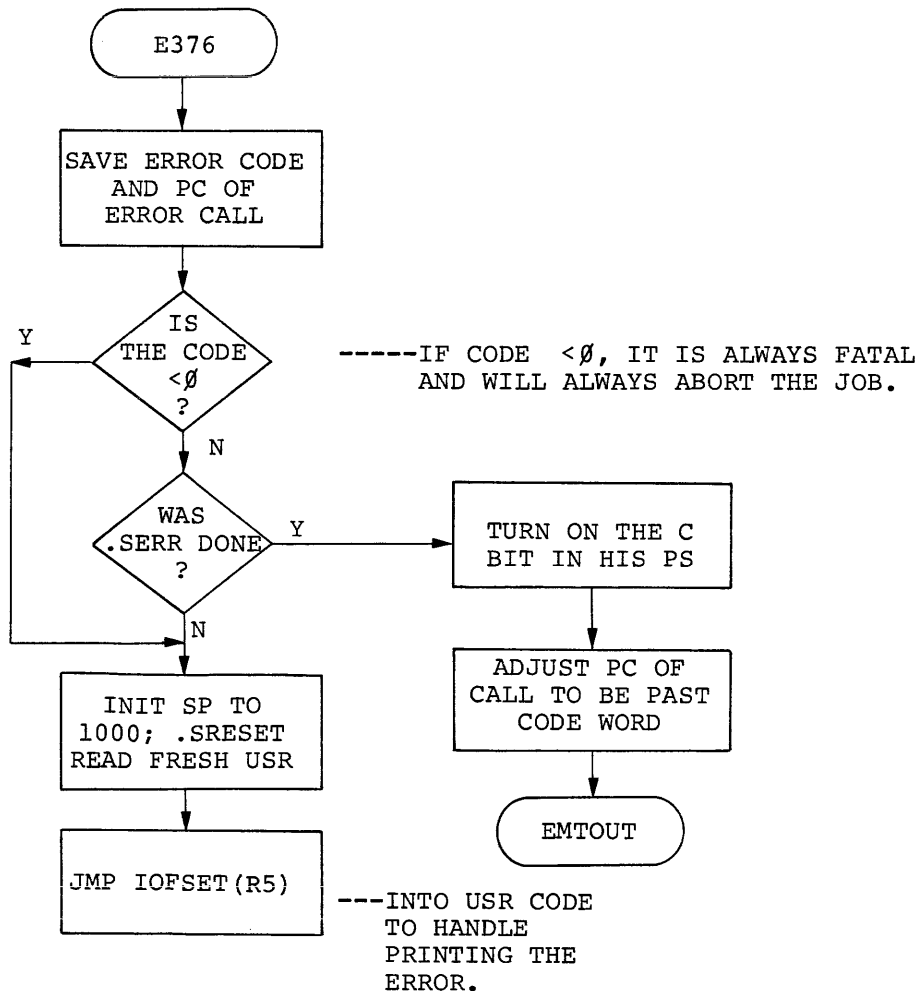


FATAL ERROR PROCESSING

EMT 376 is reserved for reporting fatal monitor errors. When a fatal error condition is encountered, a call of the form:

EMT 376
code

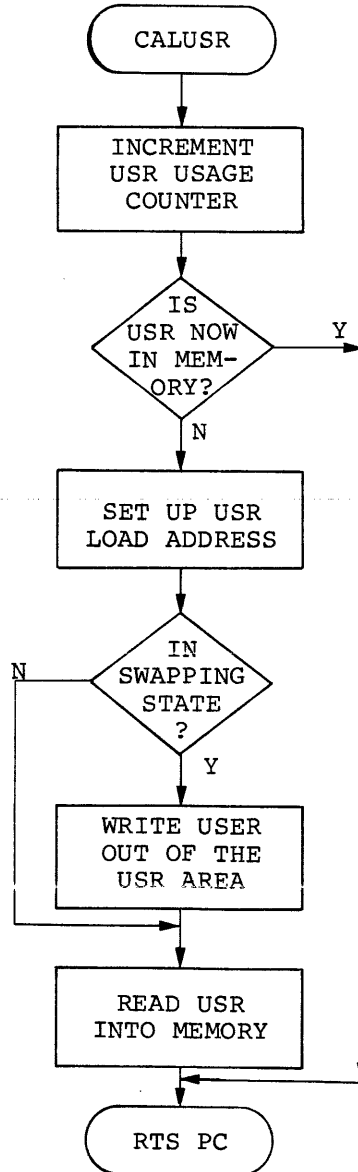
is executed. This indicates to RT-11 that a fatal error has occurred. The normal response is to print a ?M-error message and then abort the job. However, if a .SERR request has been done, no message will occur and control will pass to the user's program. The error bit (C bit) will be set and byte 52 will contain the negative of the error code.



CALUSR

CALUSR is used to ensure that the USR is in memory for a USR type request. It will handle the situation where the user program must be written to scratch blocks before the USR is read in. Entry is made at CLUSR2 when an error has occurred and the error processing code in the USR buffer is required.

EITHER 'NORMAL' VALUE
OR WHAT THE USER HAS
PUT IN LOCATION 46.



E.4.2 Clock Interrupt Service

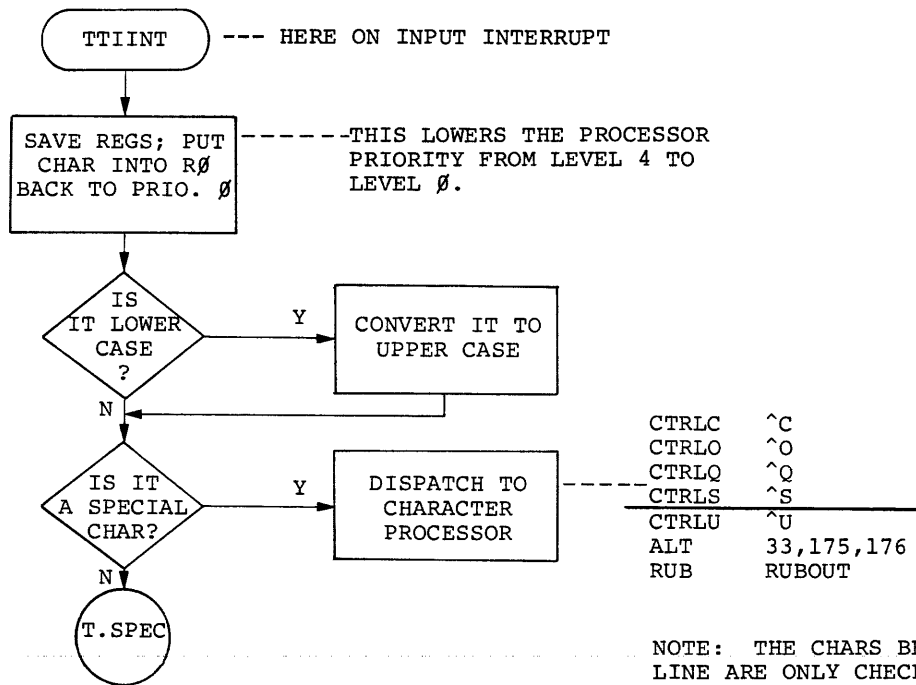
The interrupt service for the clock is primitive. The clock vector is set up such that the interrupt routine is always entered with the C bit = 1. At the interrupt routine, the code is:

```
ADC $TIME+2
ADC $TIME
RTI
```

Since the C bit is 1, \$TIME+2 is incremented by the ADC. When the low order word goes from 177777 to 0, the C bit remains on and \$TIME is then incremented. No 24 hour wrap around is provided.

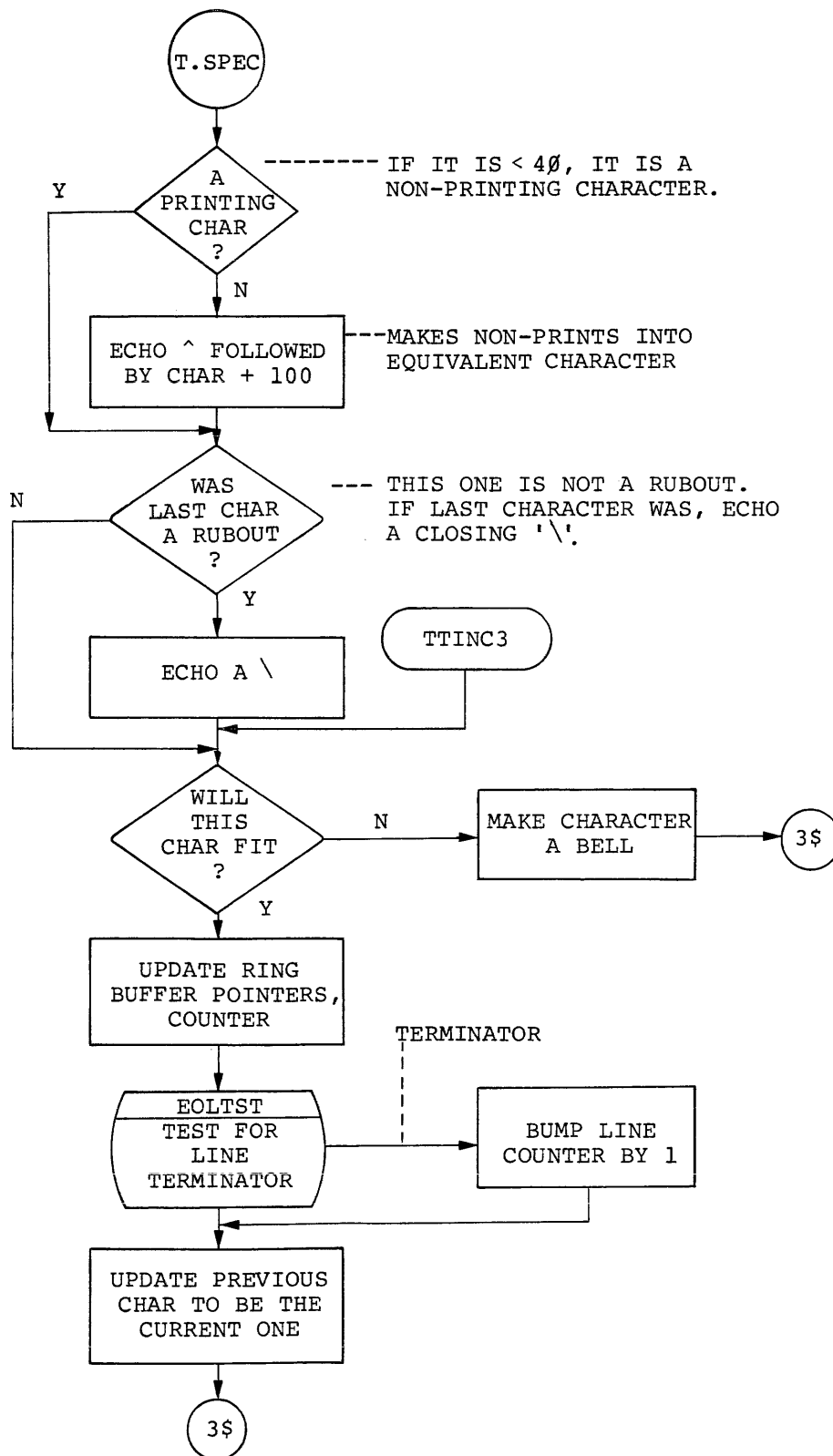
E.4.3 Console Terminal Interrupt Service

TT INPUT INTERRUPT SERVICE

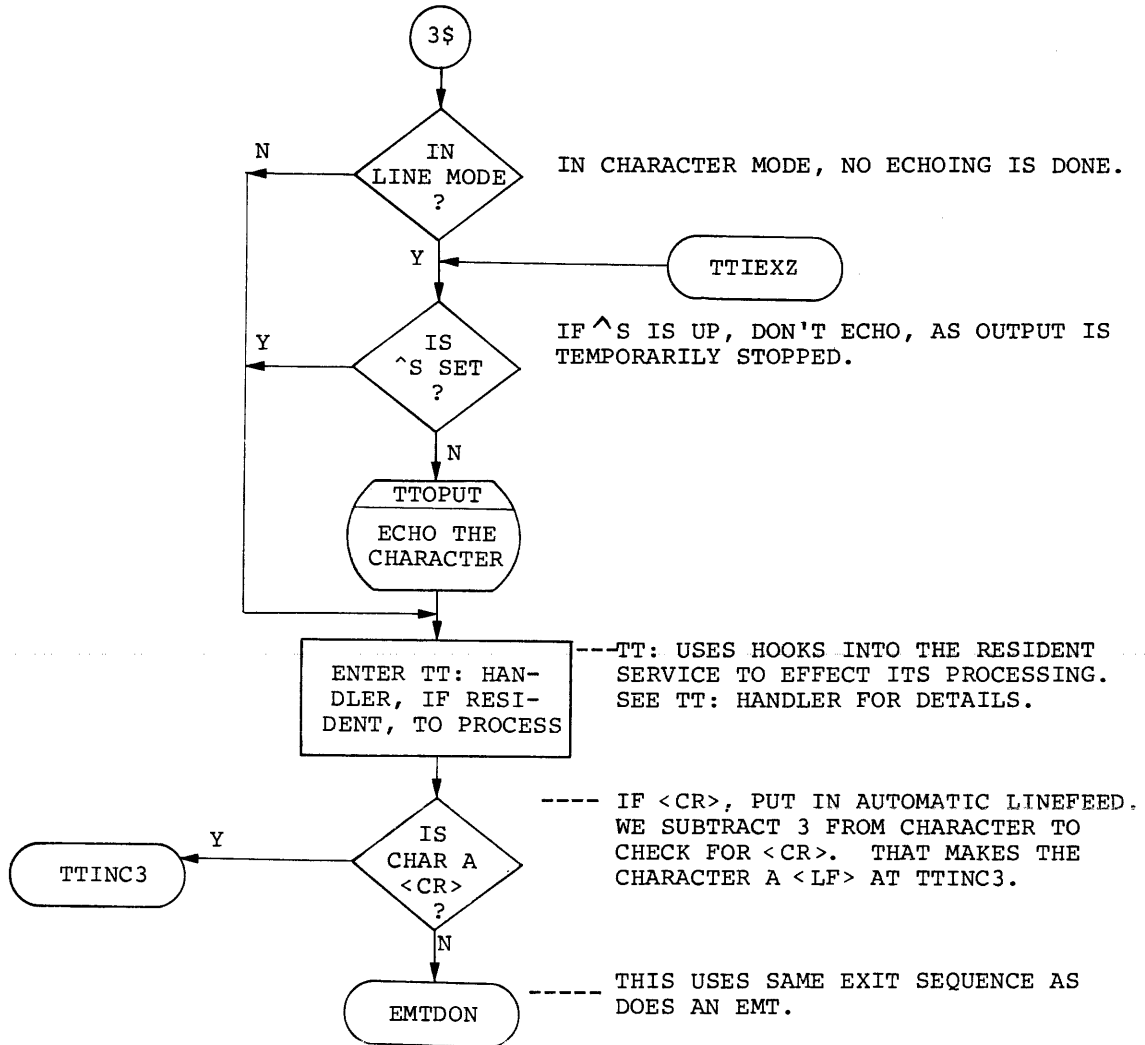


NOTE: THE CHARS BELOW THE LINE ARE ONLY CHECKED WHEN TT IS IN LINE MODE. IN CHARACTER MODE THEY ARE NOT ACTED UPON.

TT INPUT INTERRUPT SERVICE (CONT.)



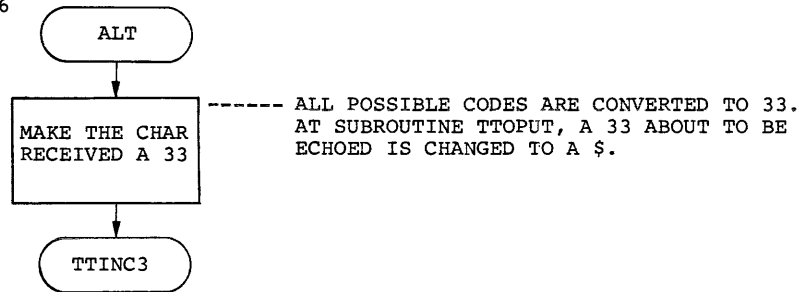
TT INPUT INTERRUPT SERVICE (CONT.)



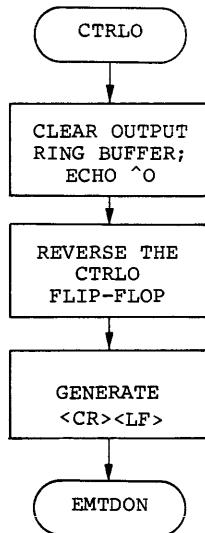
ALTMODE/CONTROL O, S, Q

These routines are entered when any of the corresponding special characters are struck.

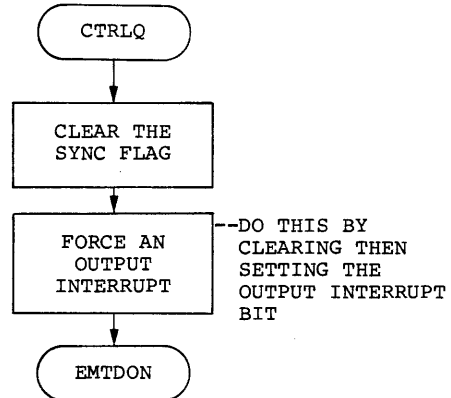
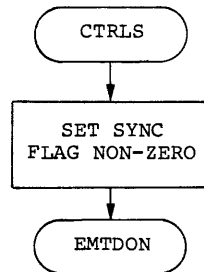
ALTMODE - 33,175,176



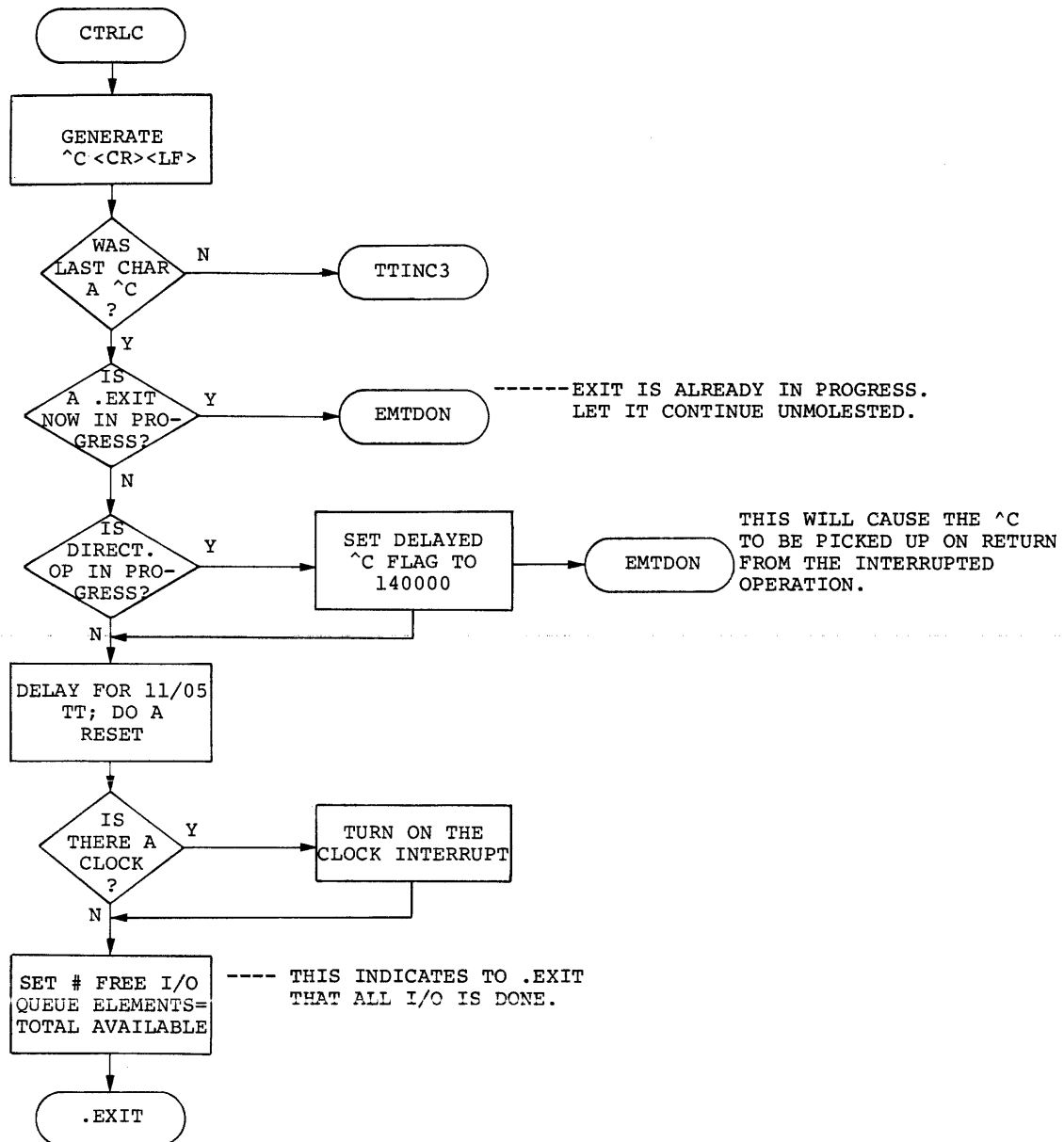
CONTROL O



CONTROL S, CONTROL Q

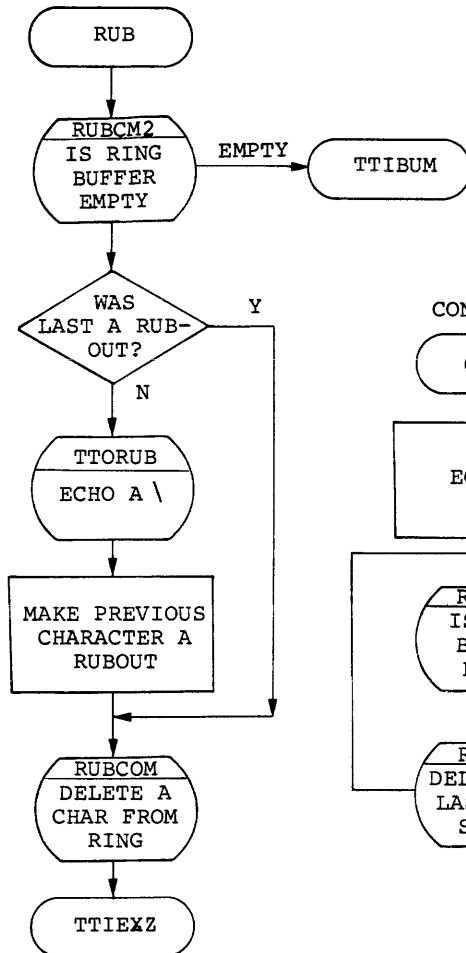


CONTROL C

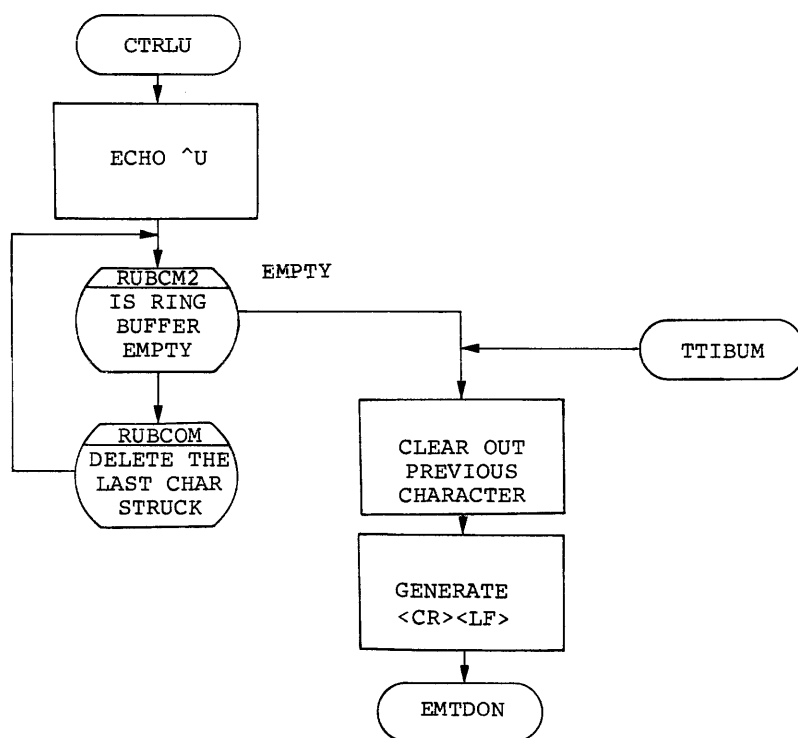


RUBOUT/CONTROL U

RUBOUT

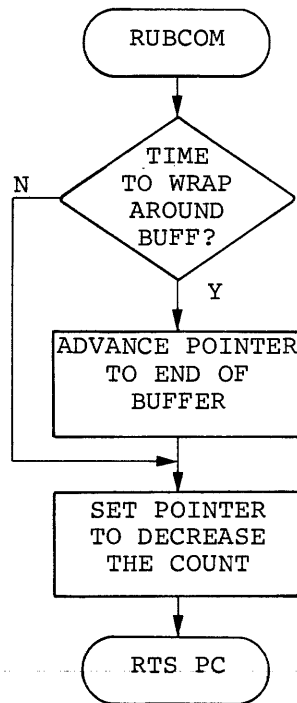


CONTROL U

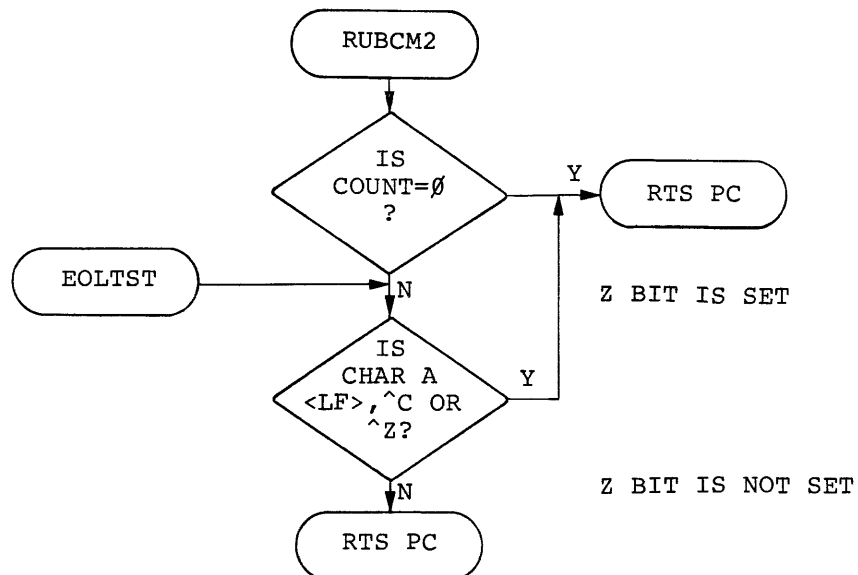


RUBCON/RUBCM2

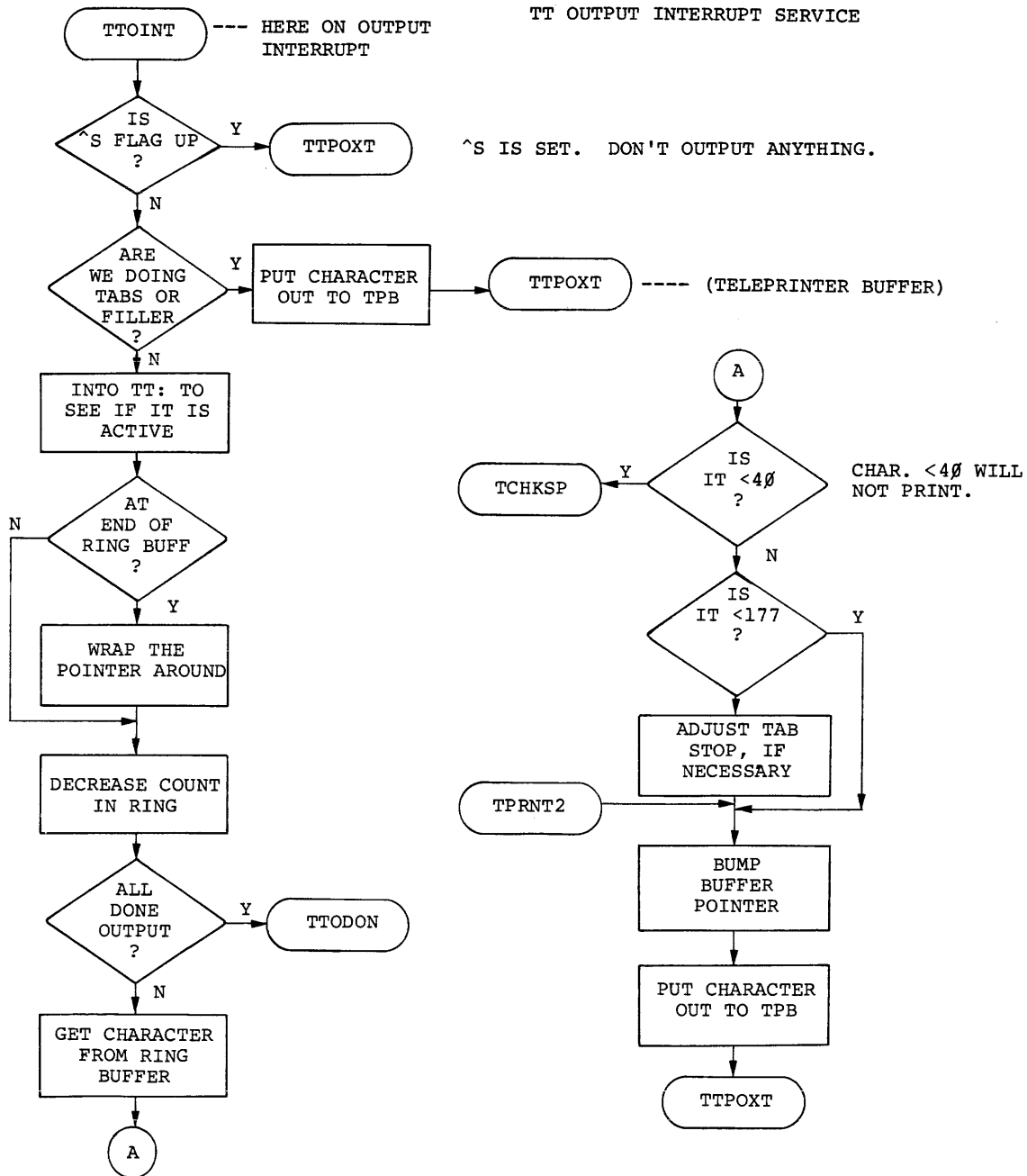
RUBCOM will update the input ring buffer pointers when a character is to be deleted.



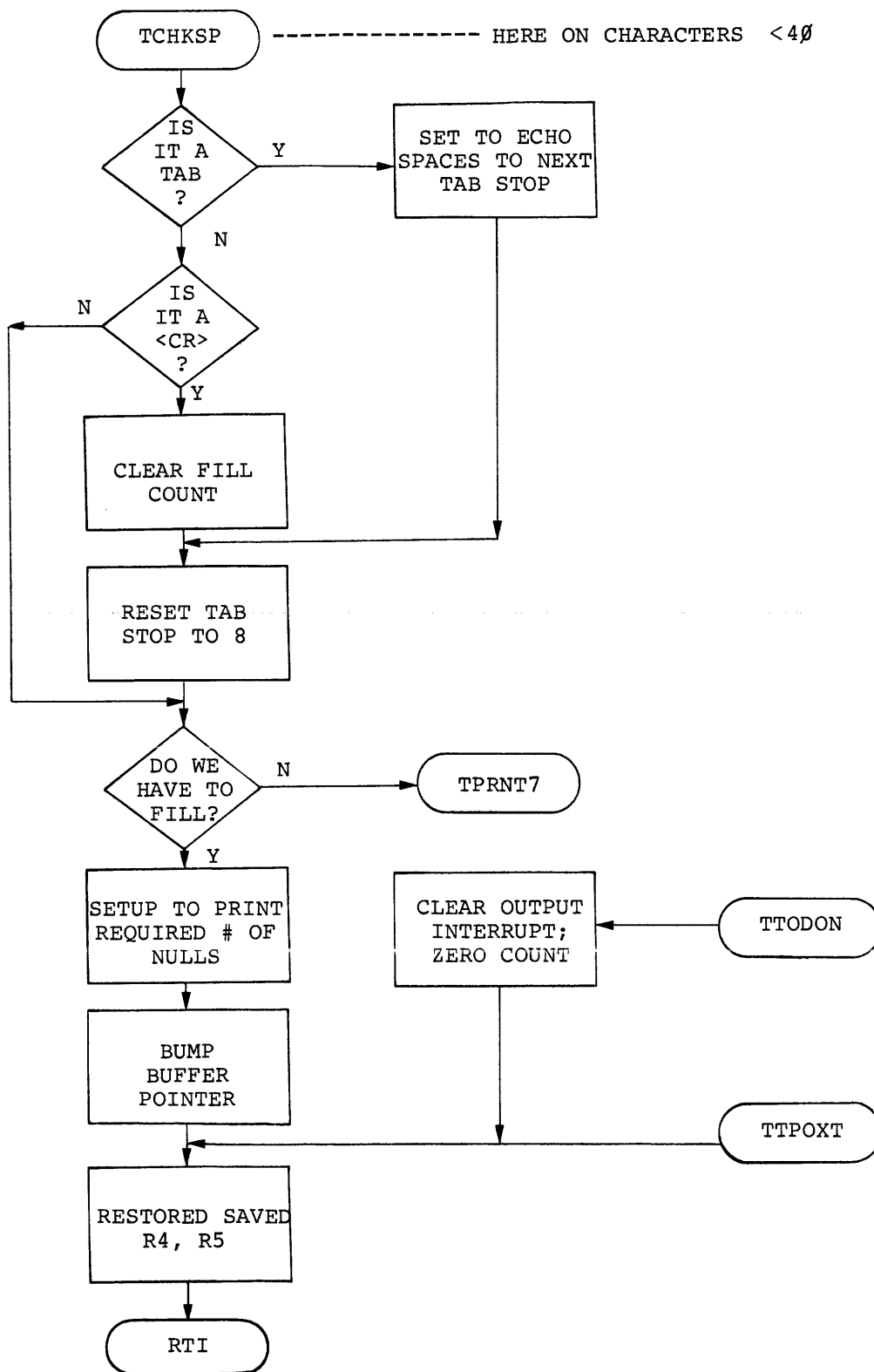
RUBCM2 checks to see if the ring buffer is empty. The buffer is empty if either the count = 0 or if the character to be deleted is a line terminator. This routine falls into routine EOLTST. The zero condition is returned if the buffer is empty.



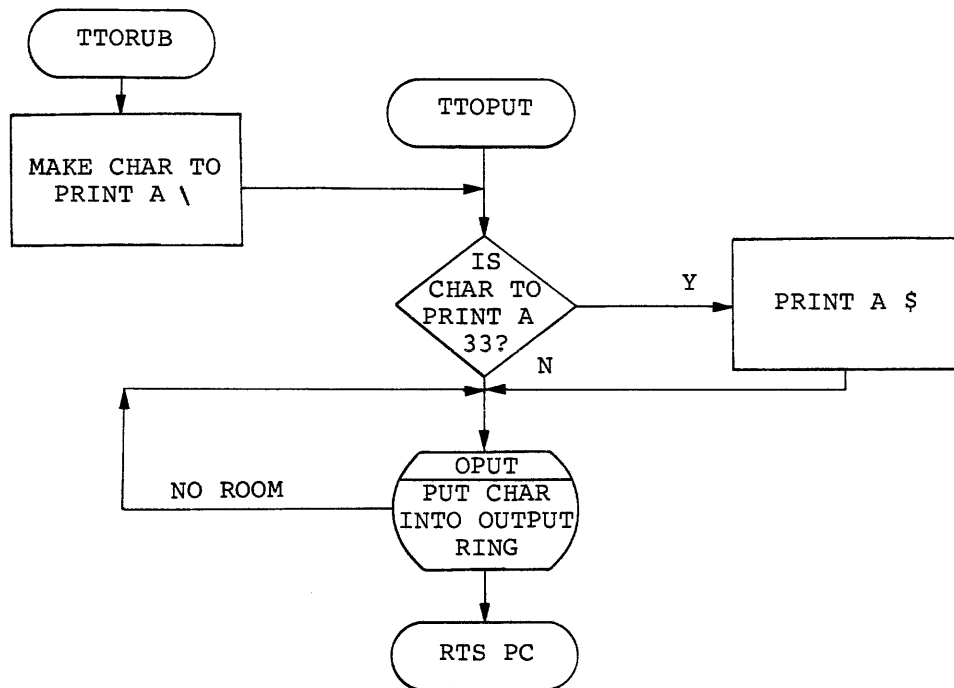
TT OUTPUT INTERRUPT SERVICE



TT OUTPUT INTERRUPT SERVICE (CONT.)

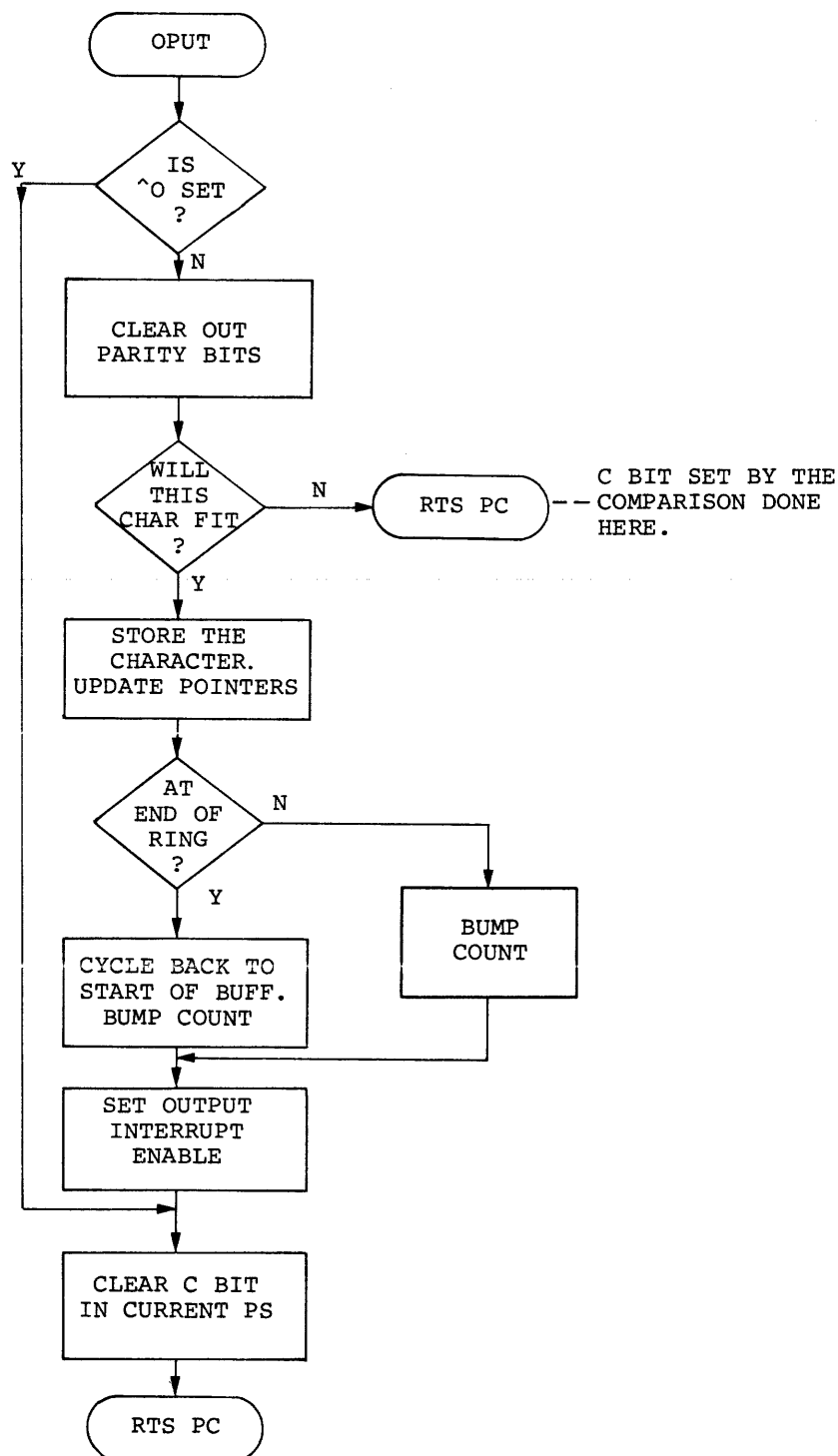


TTORUB and TTOPUT handle the printing of ALTMODE and RUBOUT. They print a \$ for ALTMODE and \ for RUBOUT.



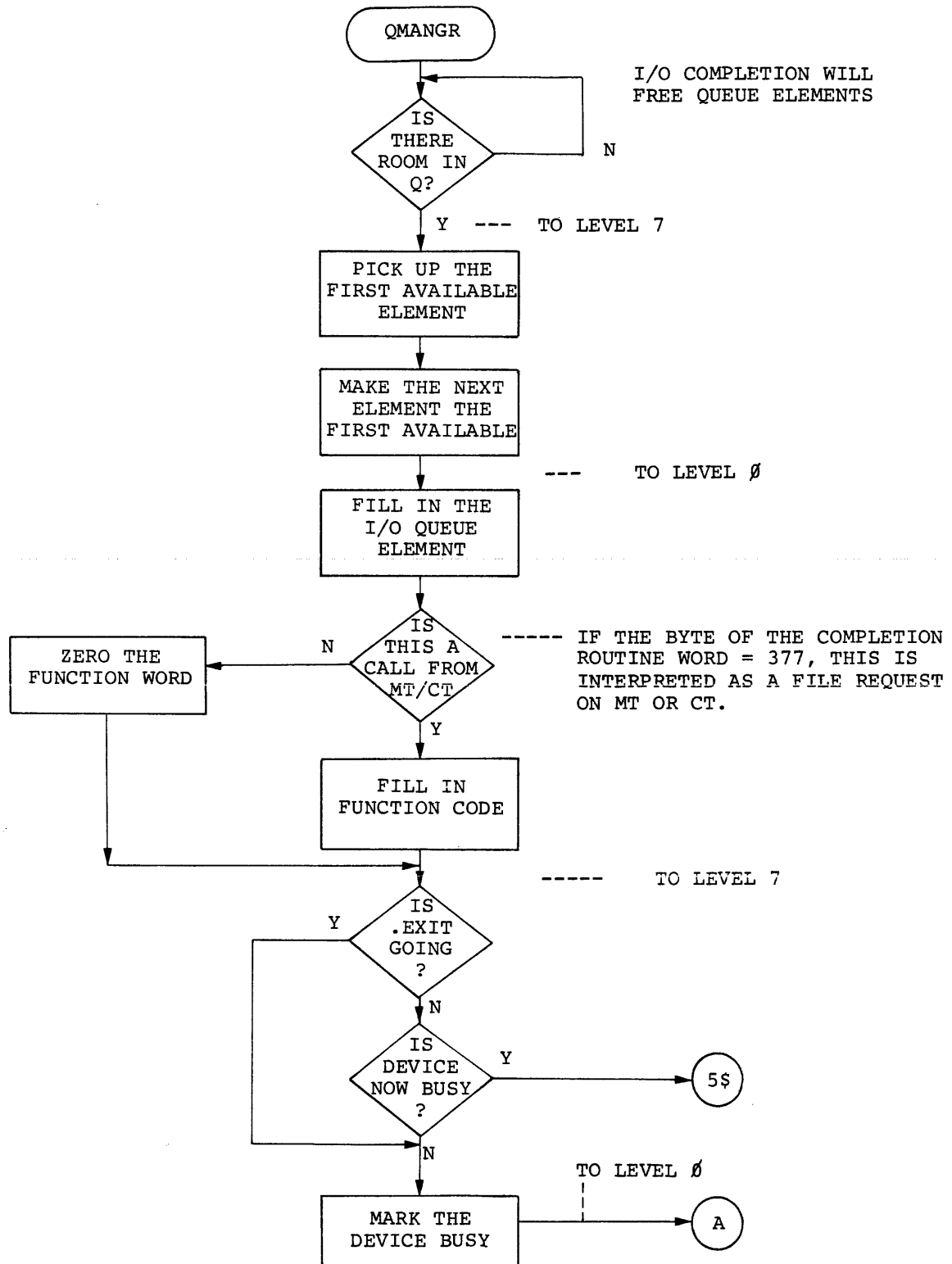
OPUT

OPUT actually puts the output character into the ring buffer. It updates the ring pointers and sets the interrupt enable bit. If the buffer is full, it returns with the C bit set.

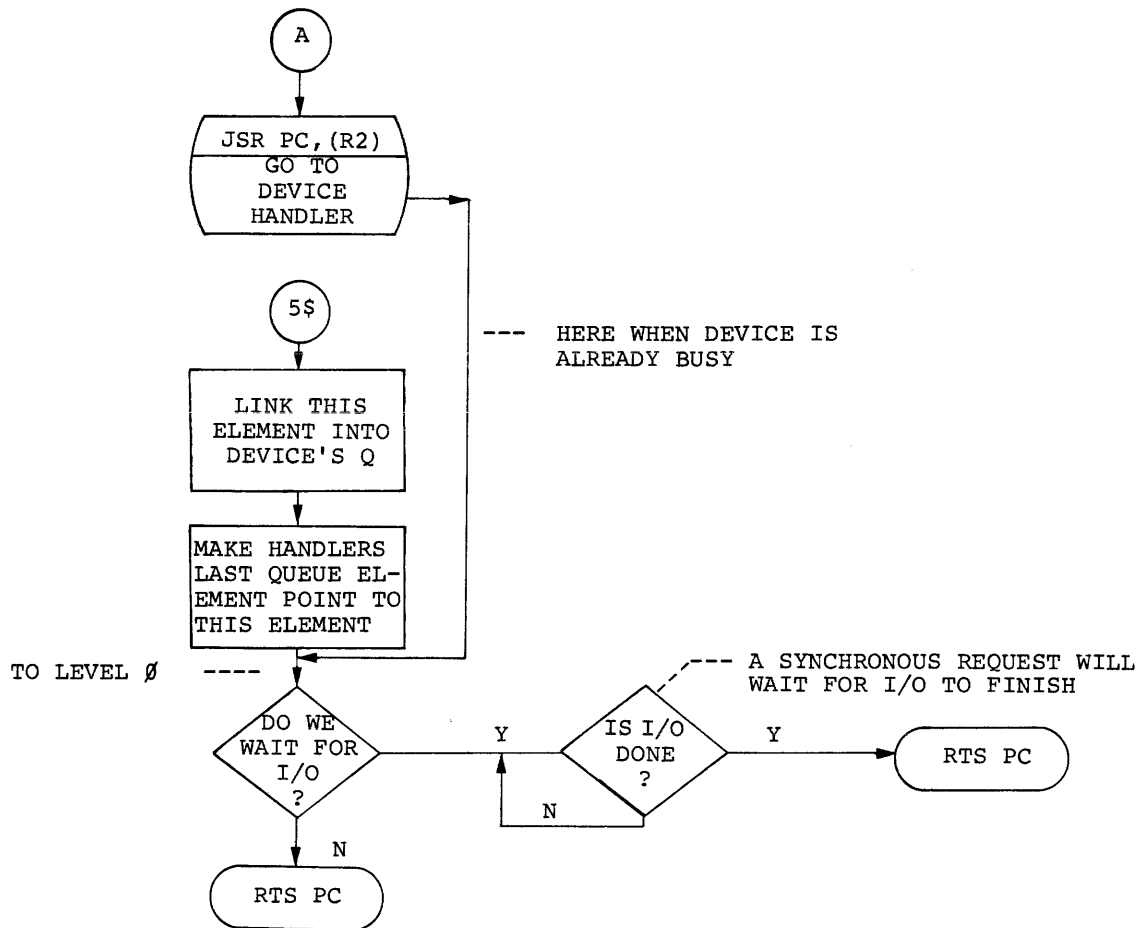


E.4.4 I/O Routines

I/O QUEUE MANAGEMENT ROUTINES

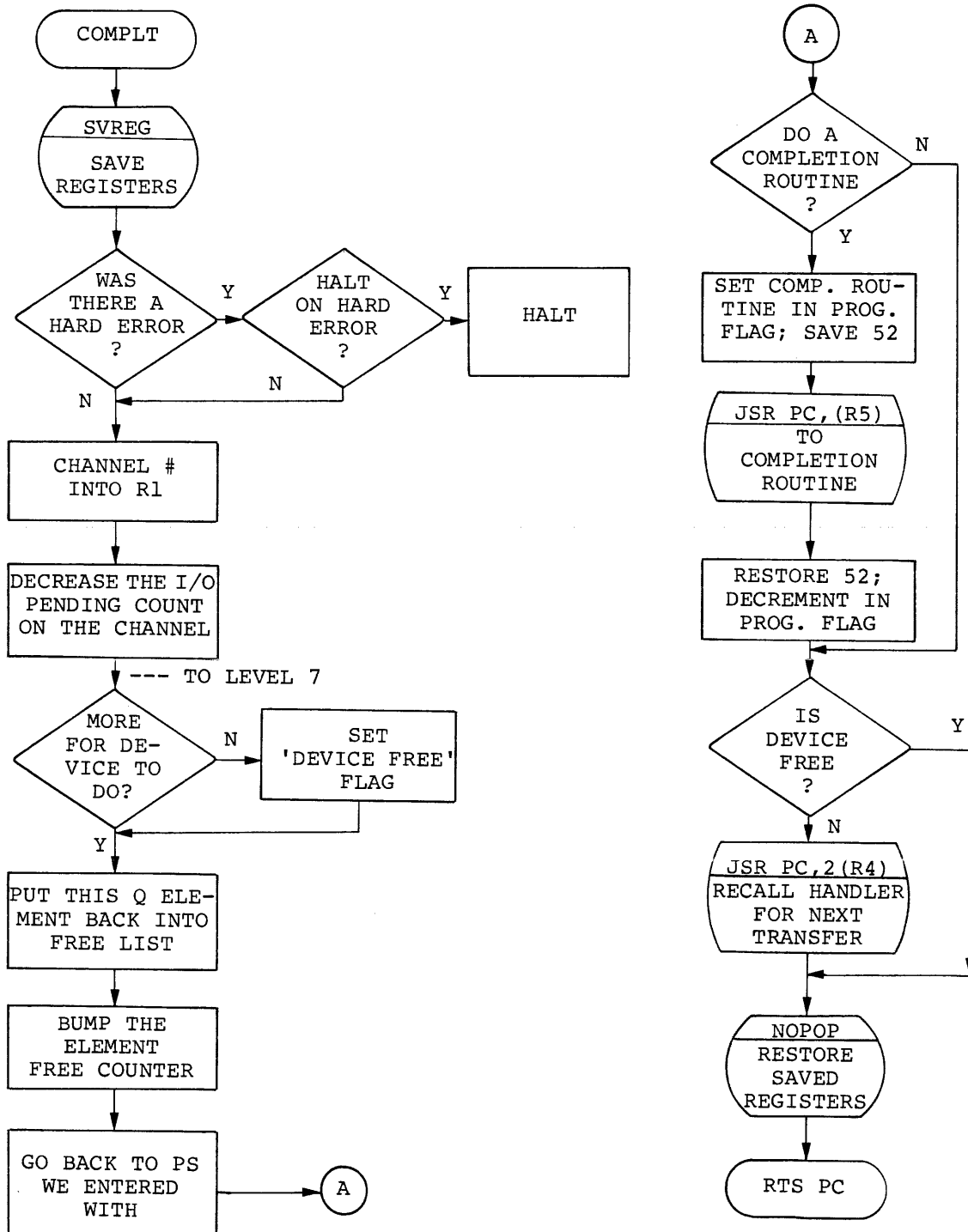


I/O QUEUE MANAGEMENT ROUTINES (CONT.)



I/O QUEUE COMPLETION

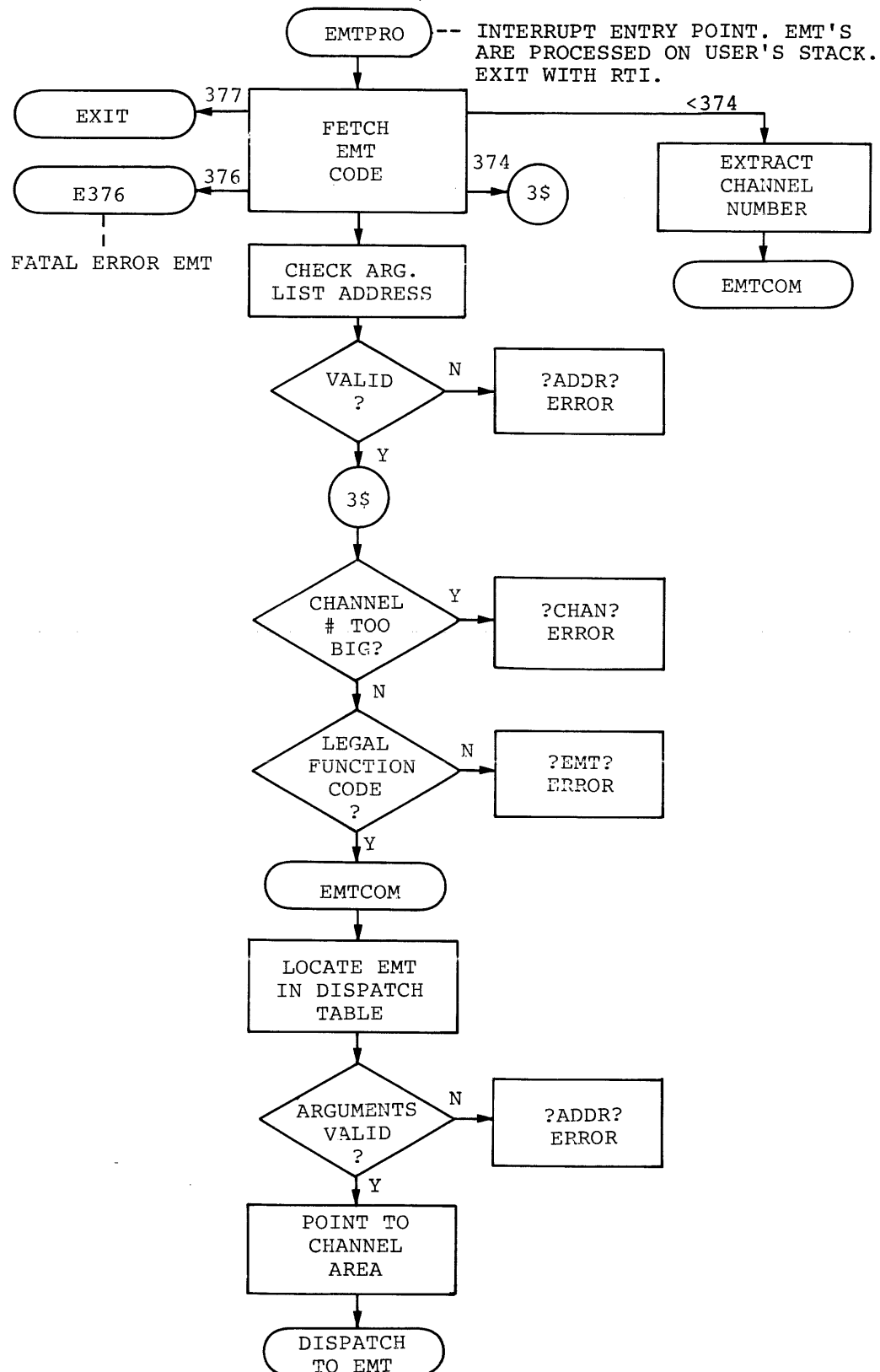
COMPLT is entered when an I/O transfer finishes.

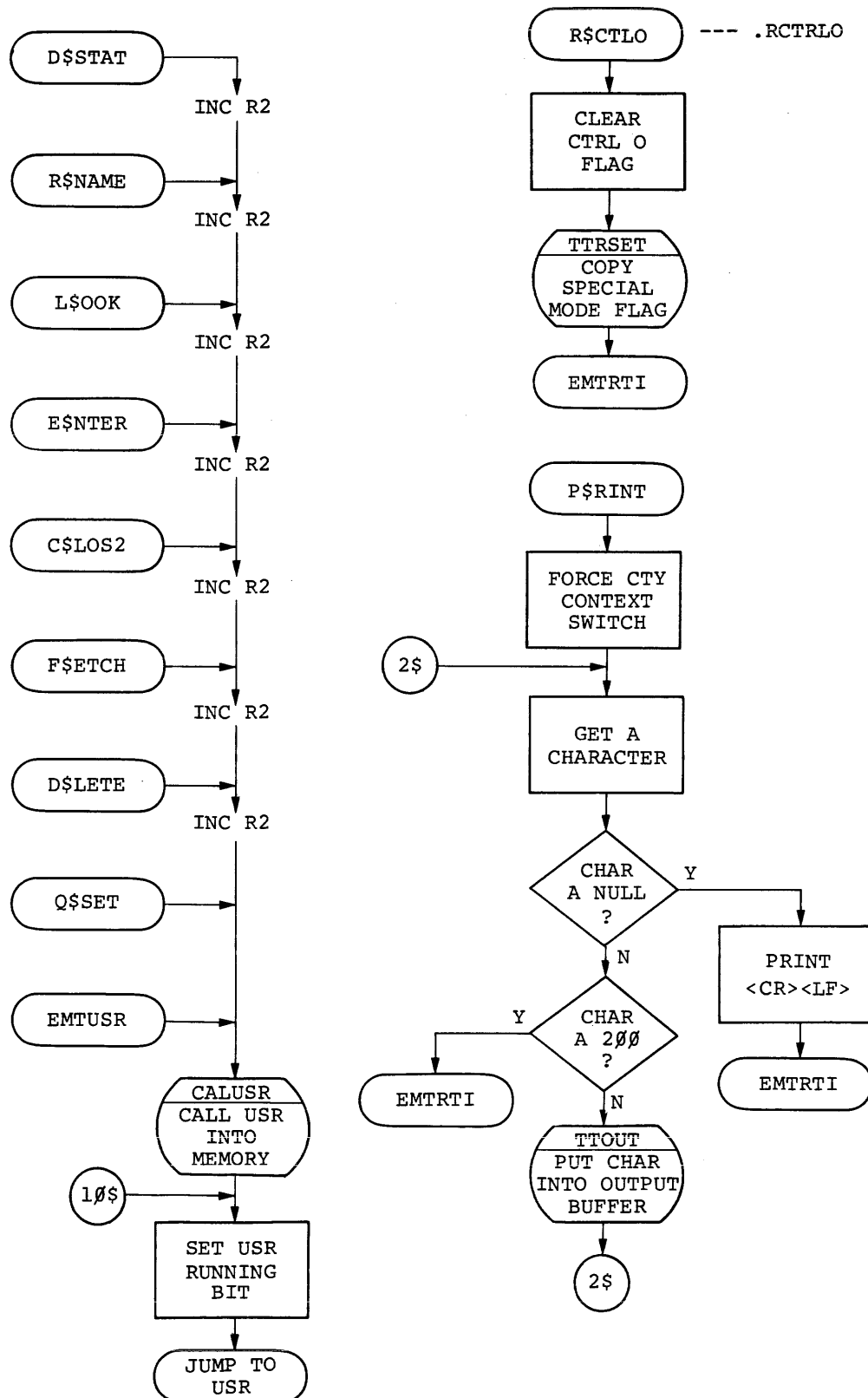


E.5 RMON (RESIDENT MONITOR) FLOWCHARTS FOR
FOREGROUND/BACKGROUND MONITOR

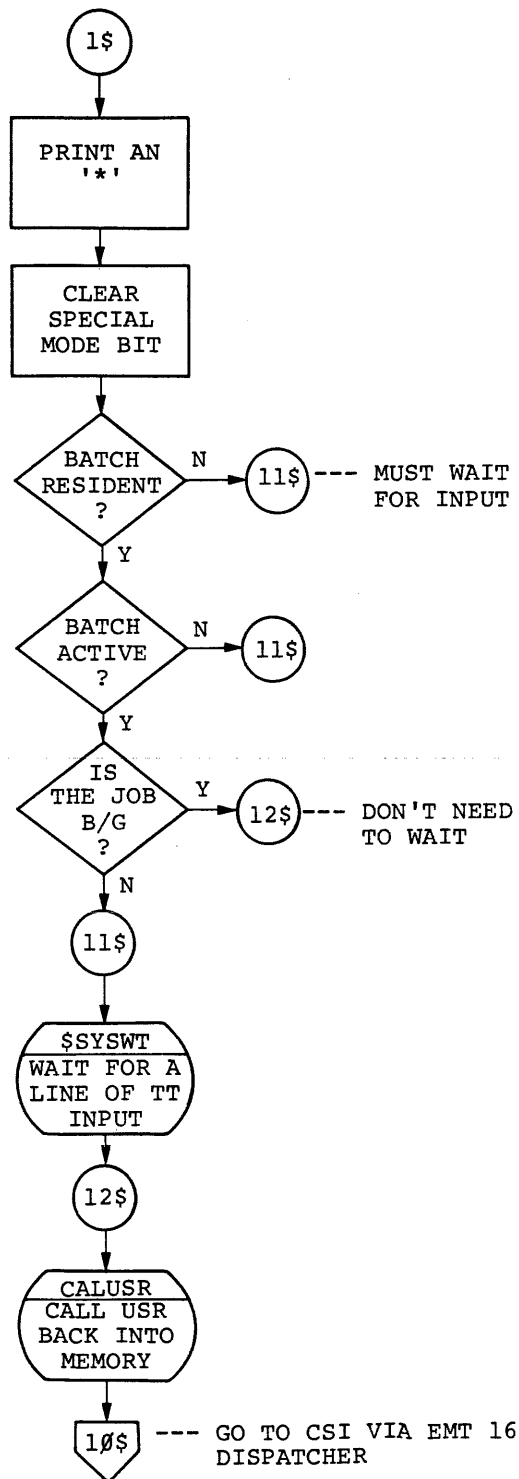
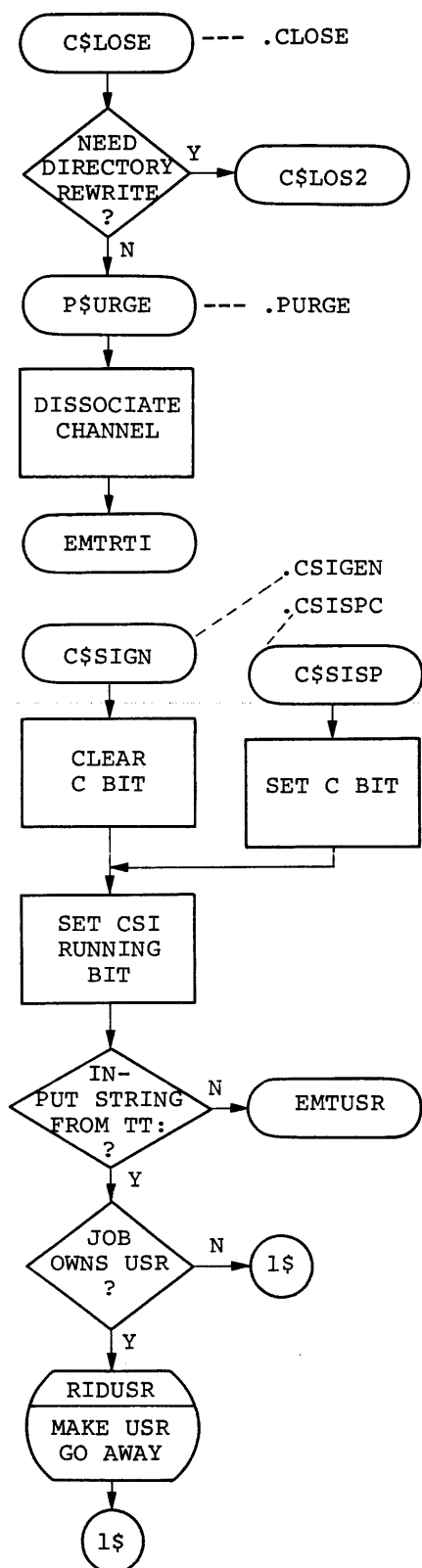
E.5.1 EMT Processors

EMT DISPATCHER

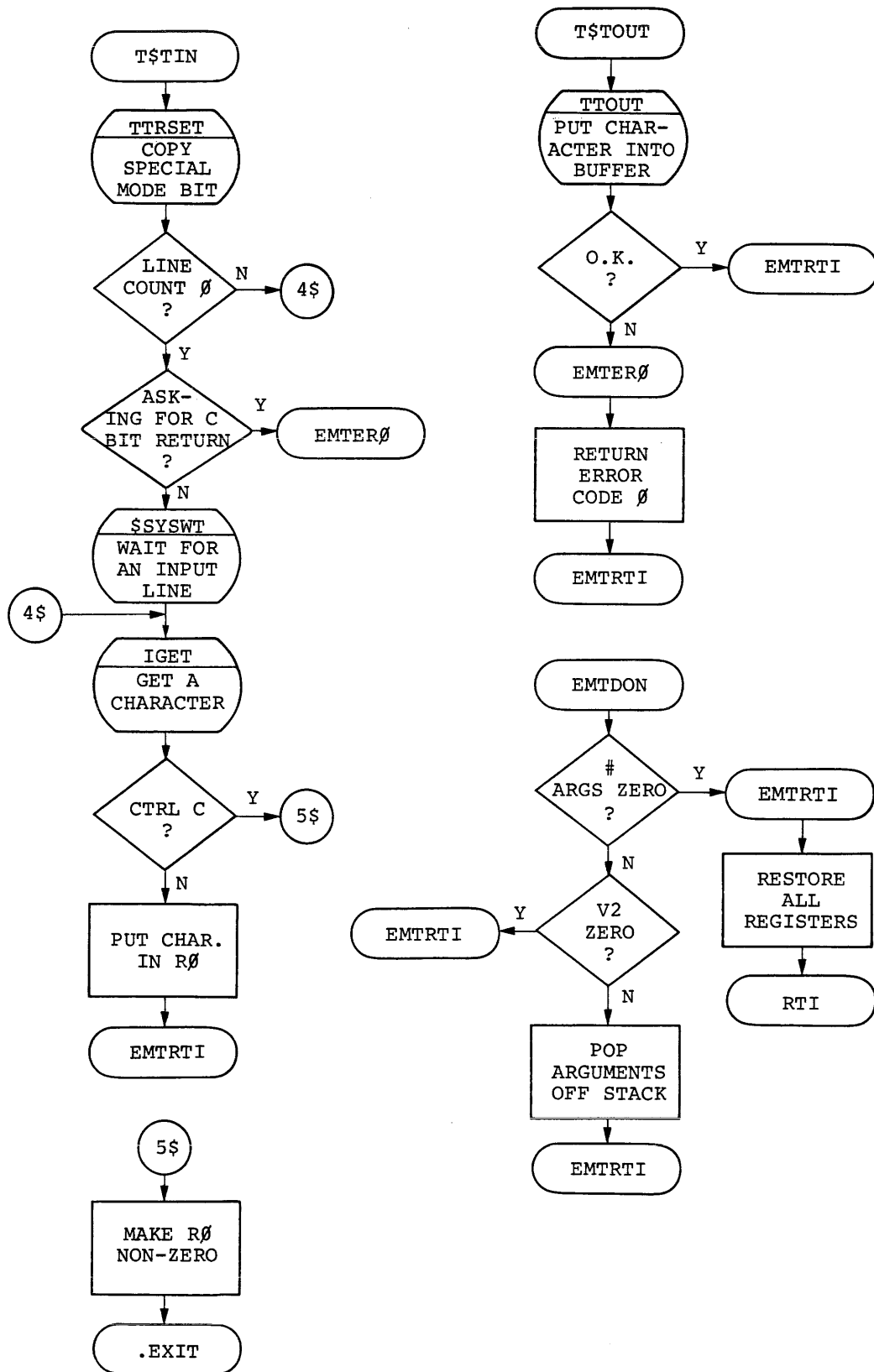




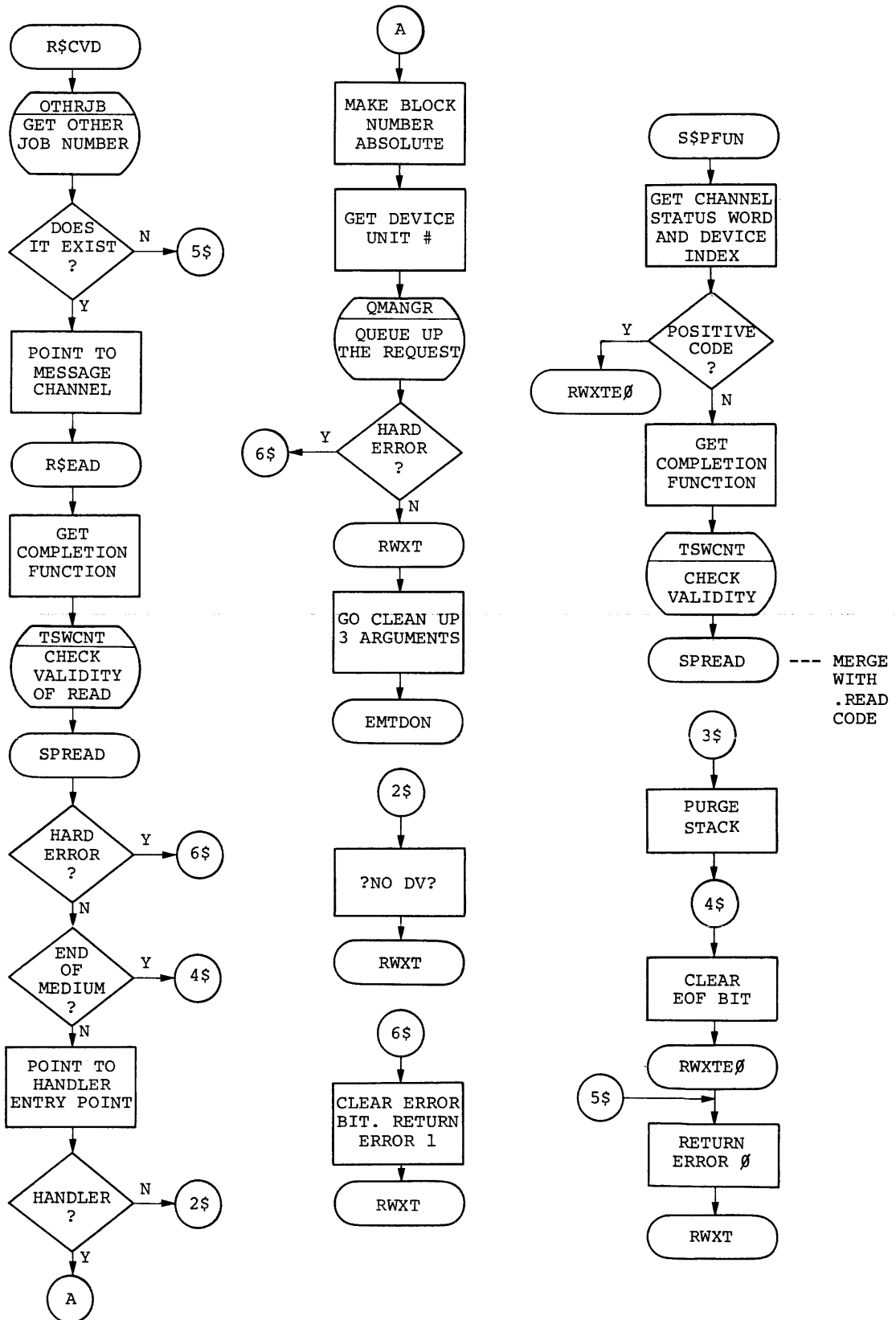
.CLOSE, .PURGE, .CSISPC, .CSIGEN



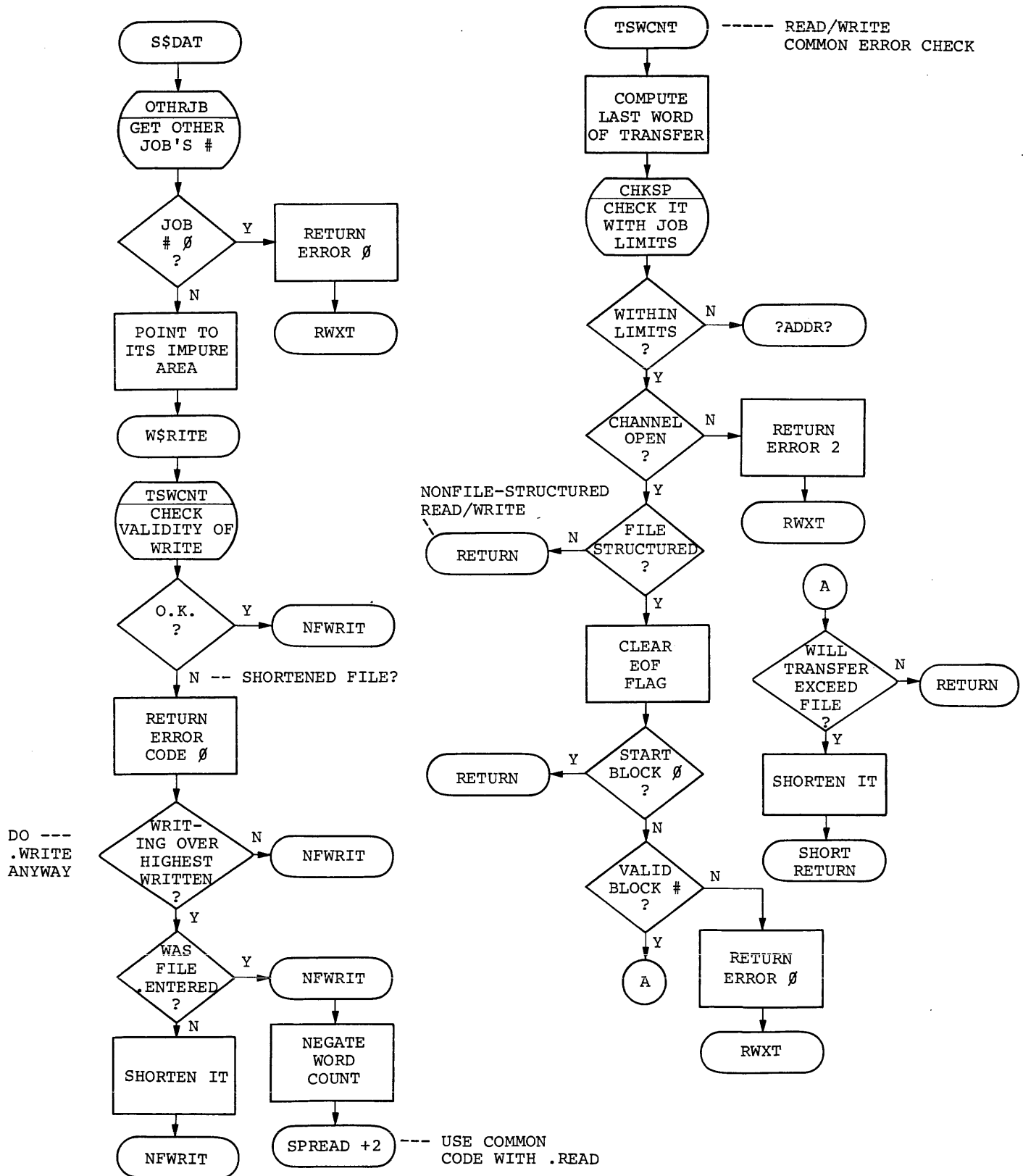
.TTYIN, .TTYOUT, EMT RETURN



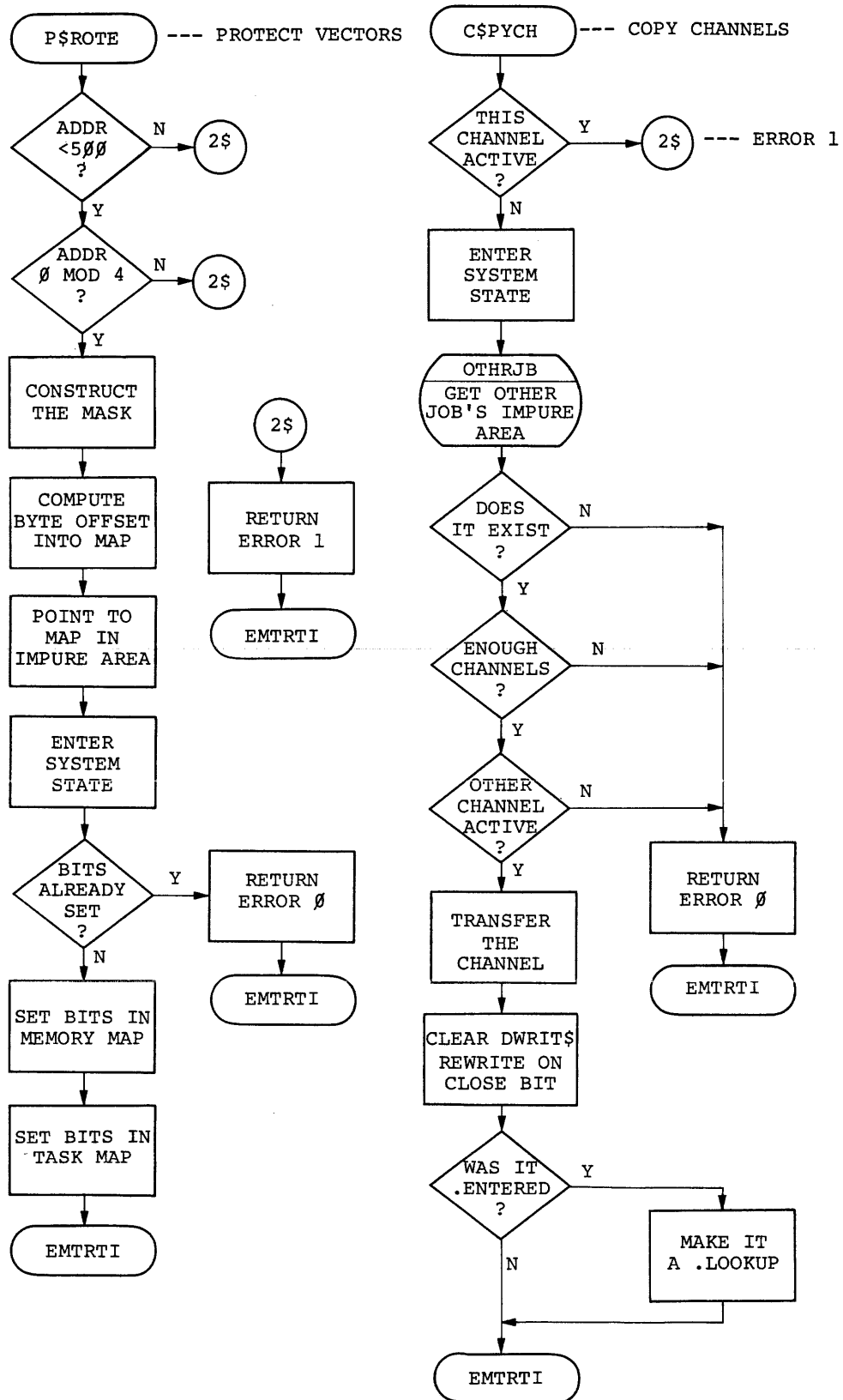
.READ, .RCVD, .SPFUN



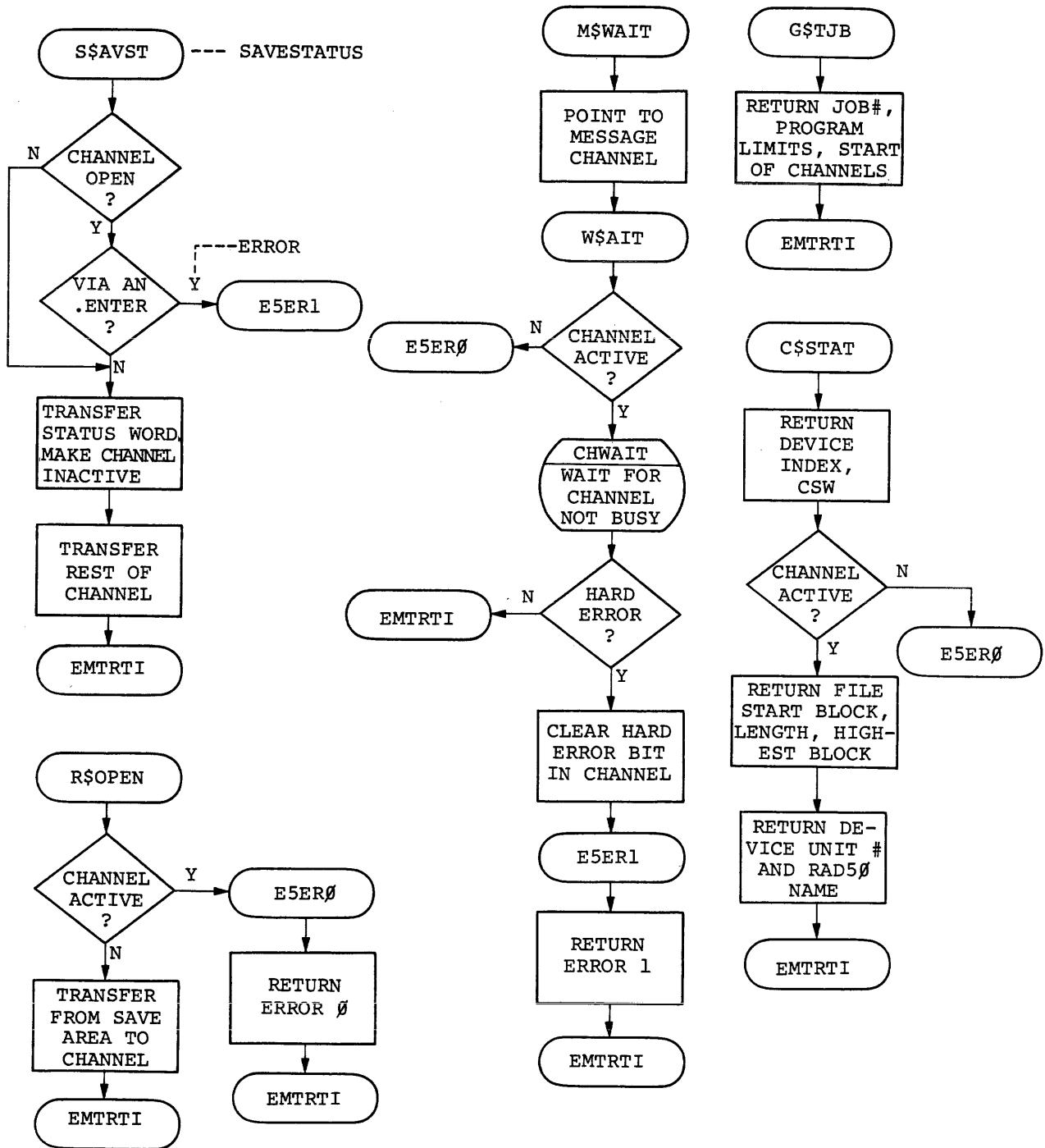
.SDAT, .WRITE



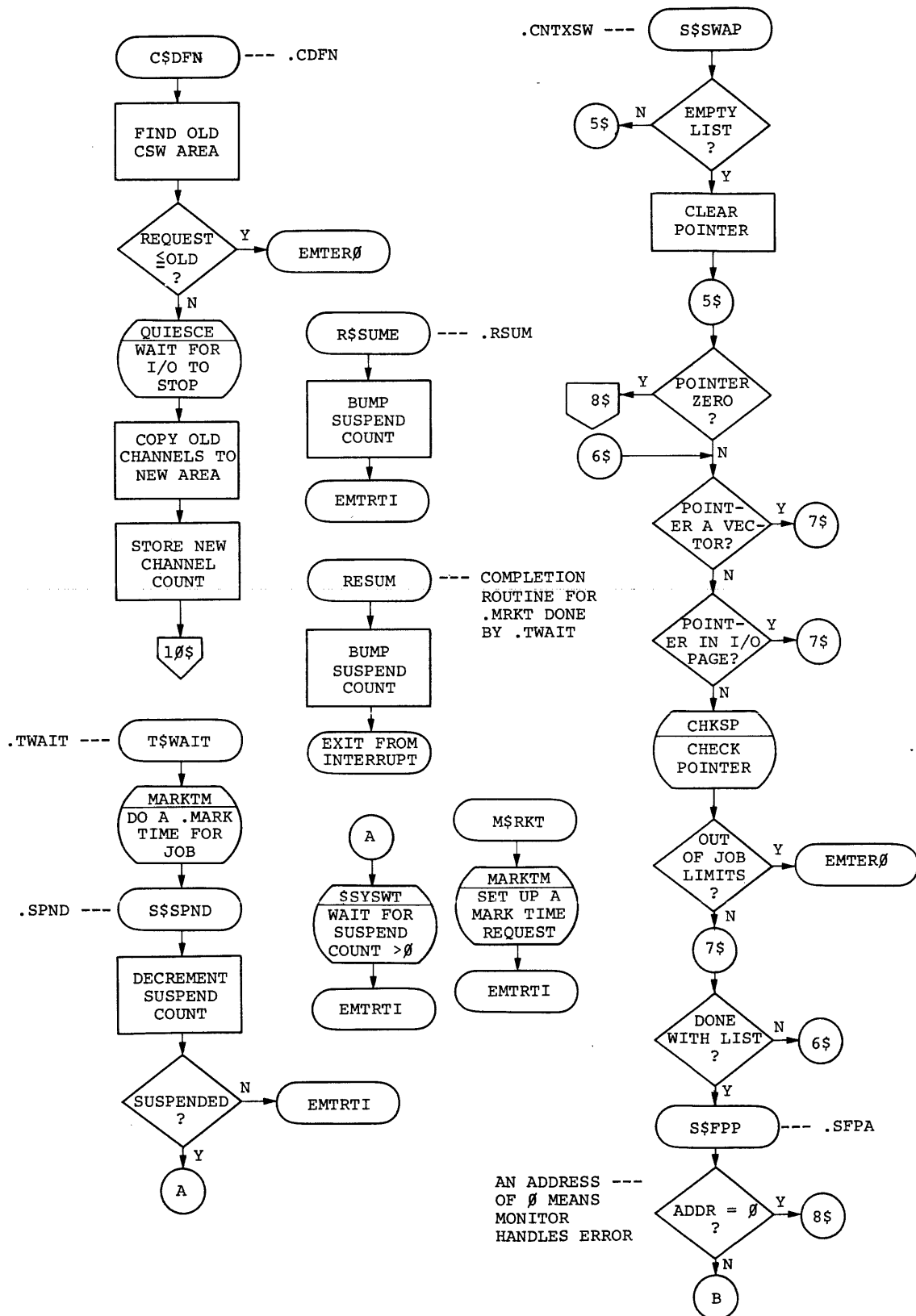
.PROTECT, .CHCOPY



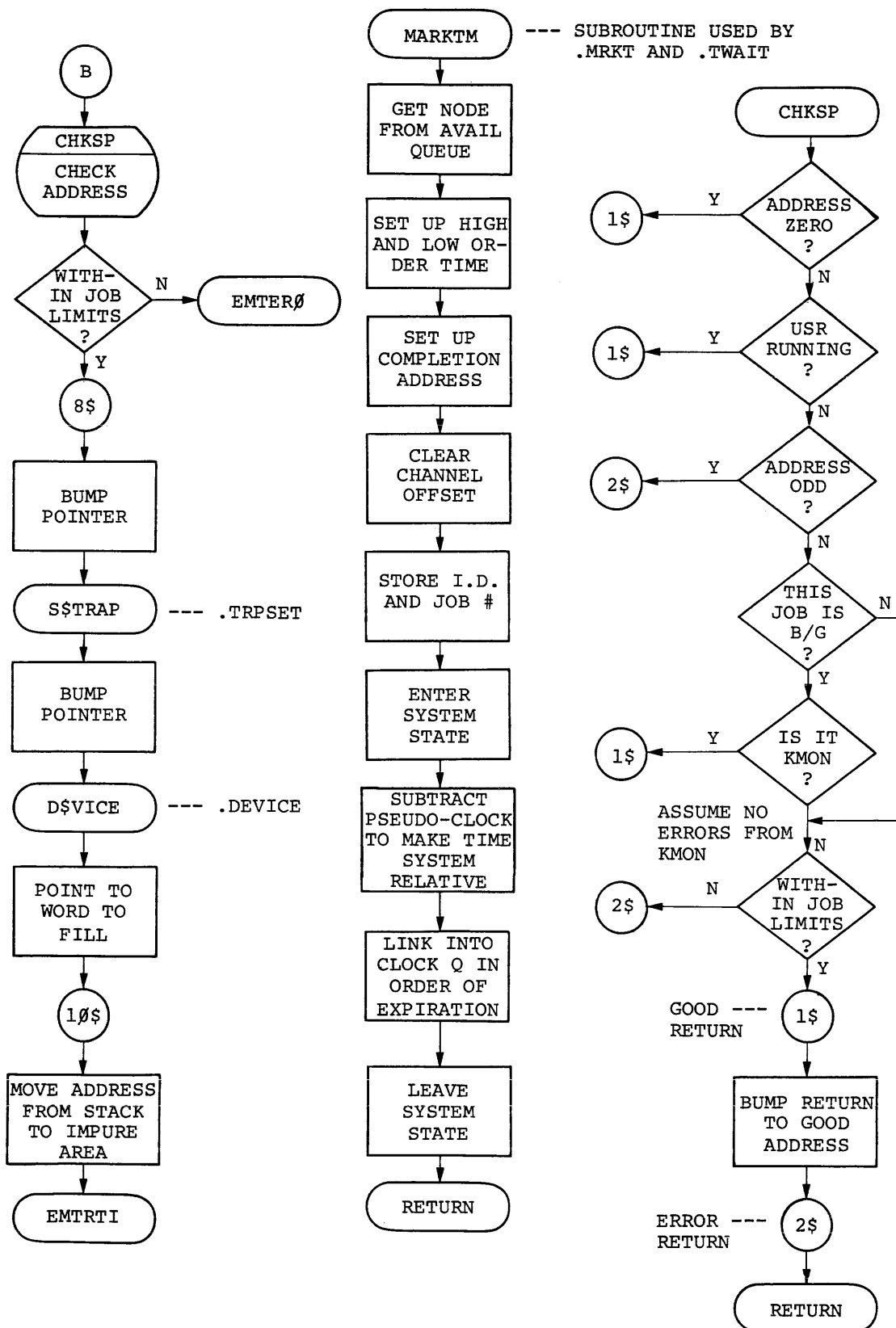
.SAVESTATUS, .REOPEN,
.M\$WAIT, .WAIT,
.GTJB, .CSTATUS



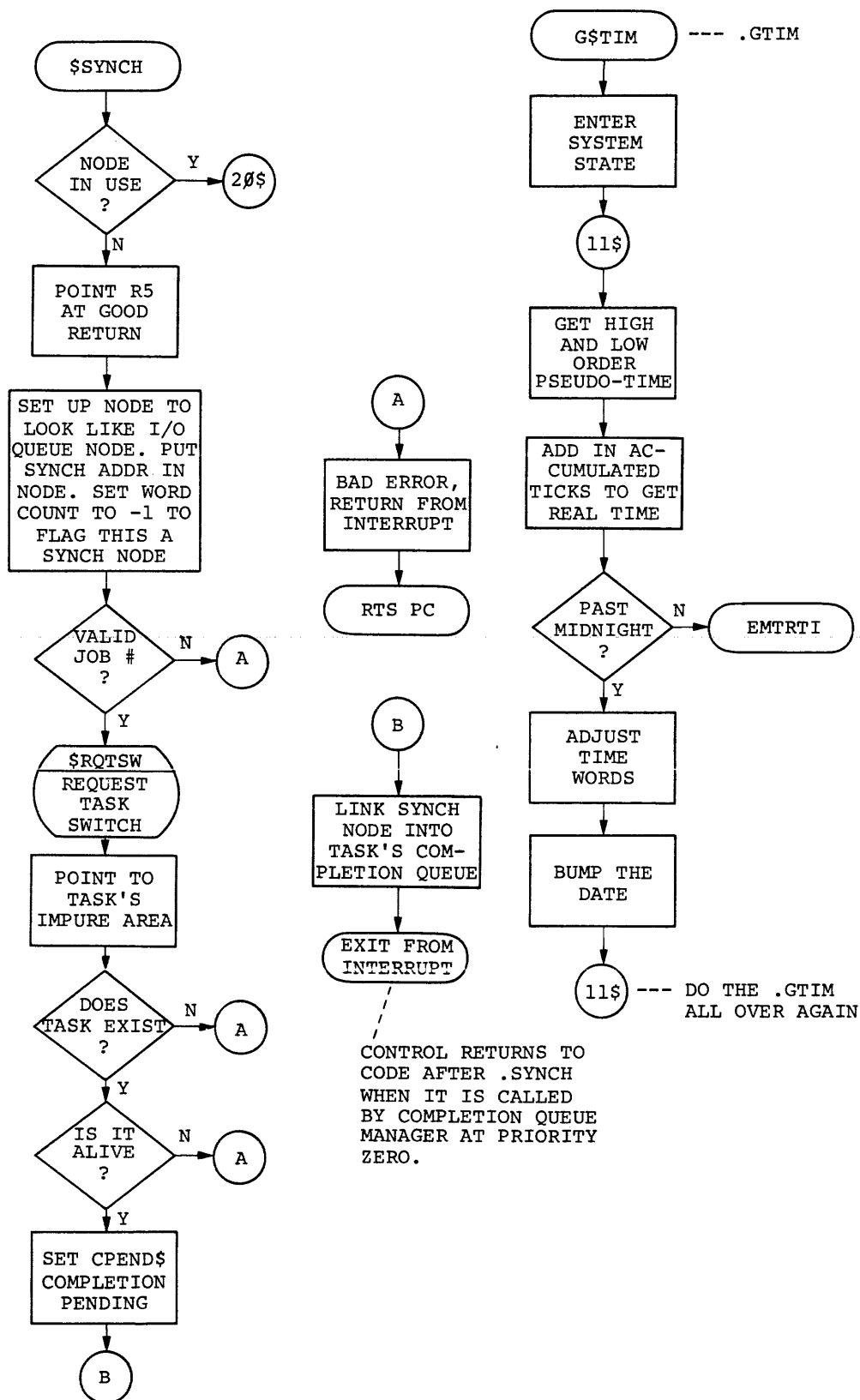
```
.CDFN, .TWAIT, .SPND, .RSUM,  
.CNTXSW, .SFPA, .TRPSET, .DEVICE
```



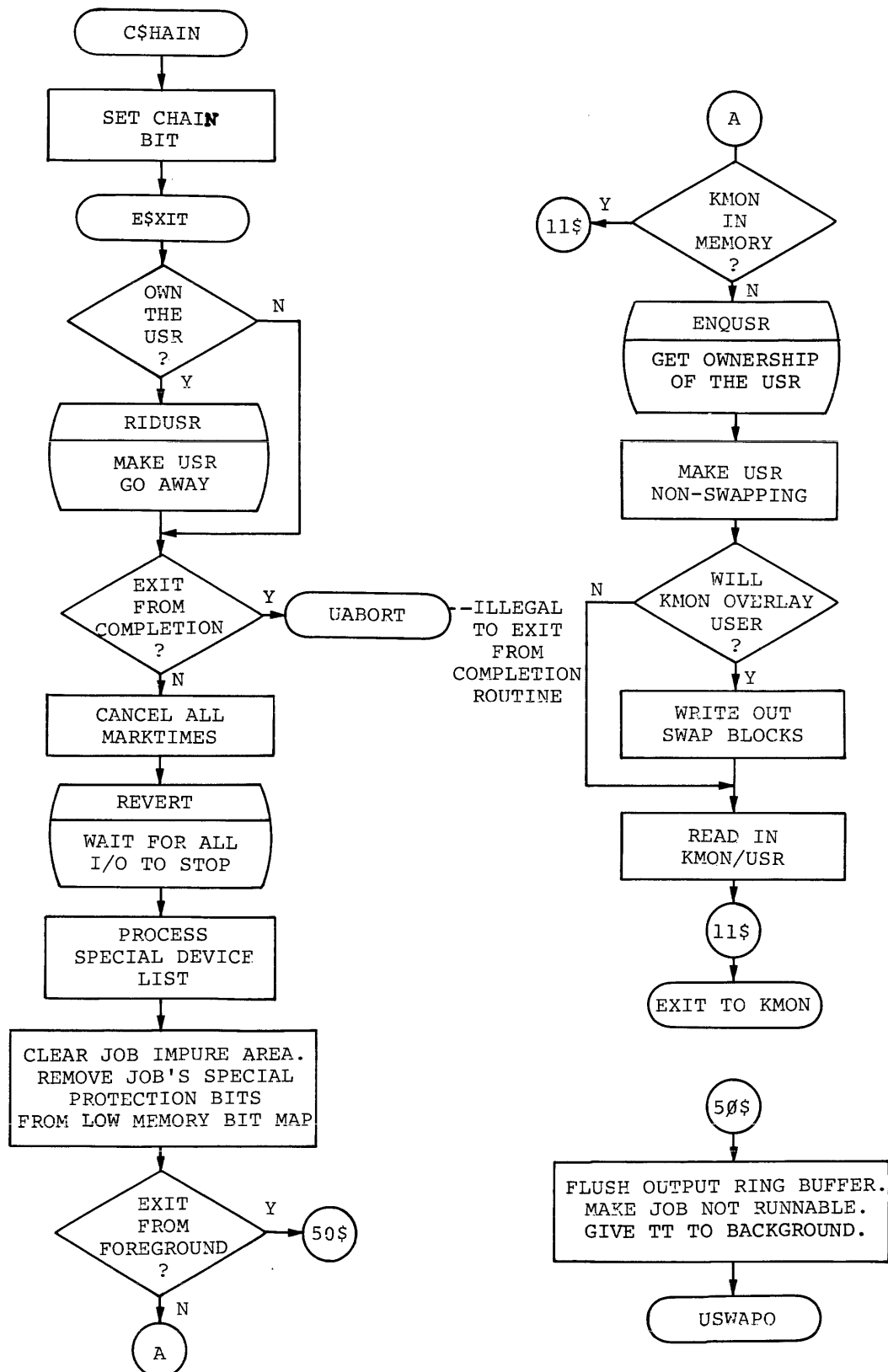
.TRPSET, .DEVICE (CONT.)



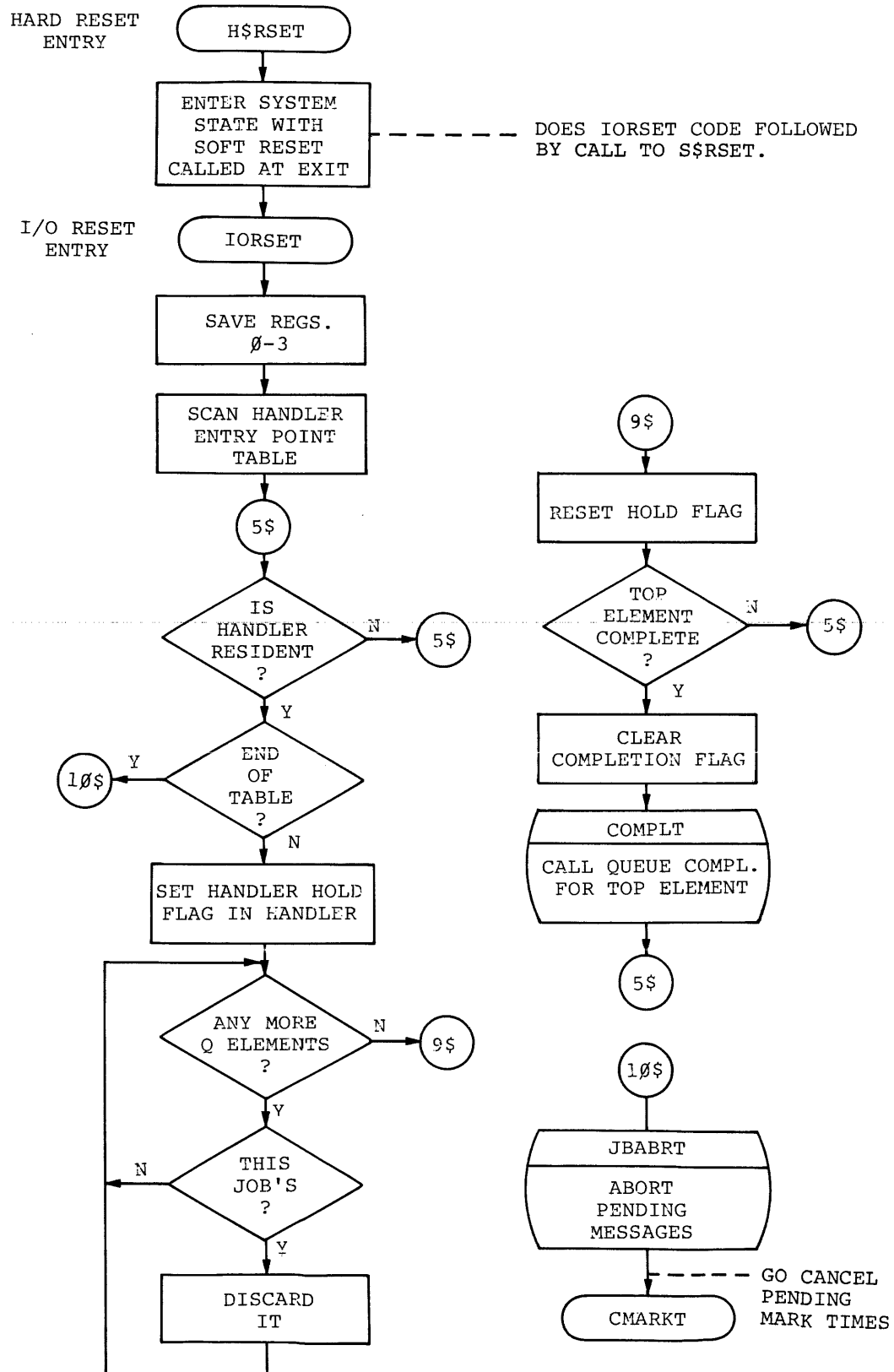
.SYNCH, .GTIM



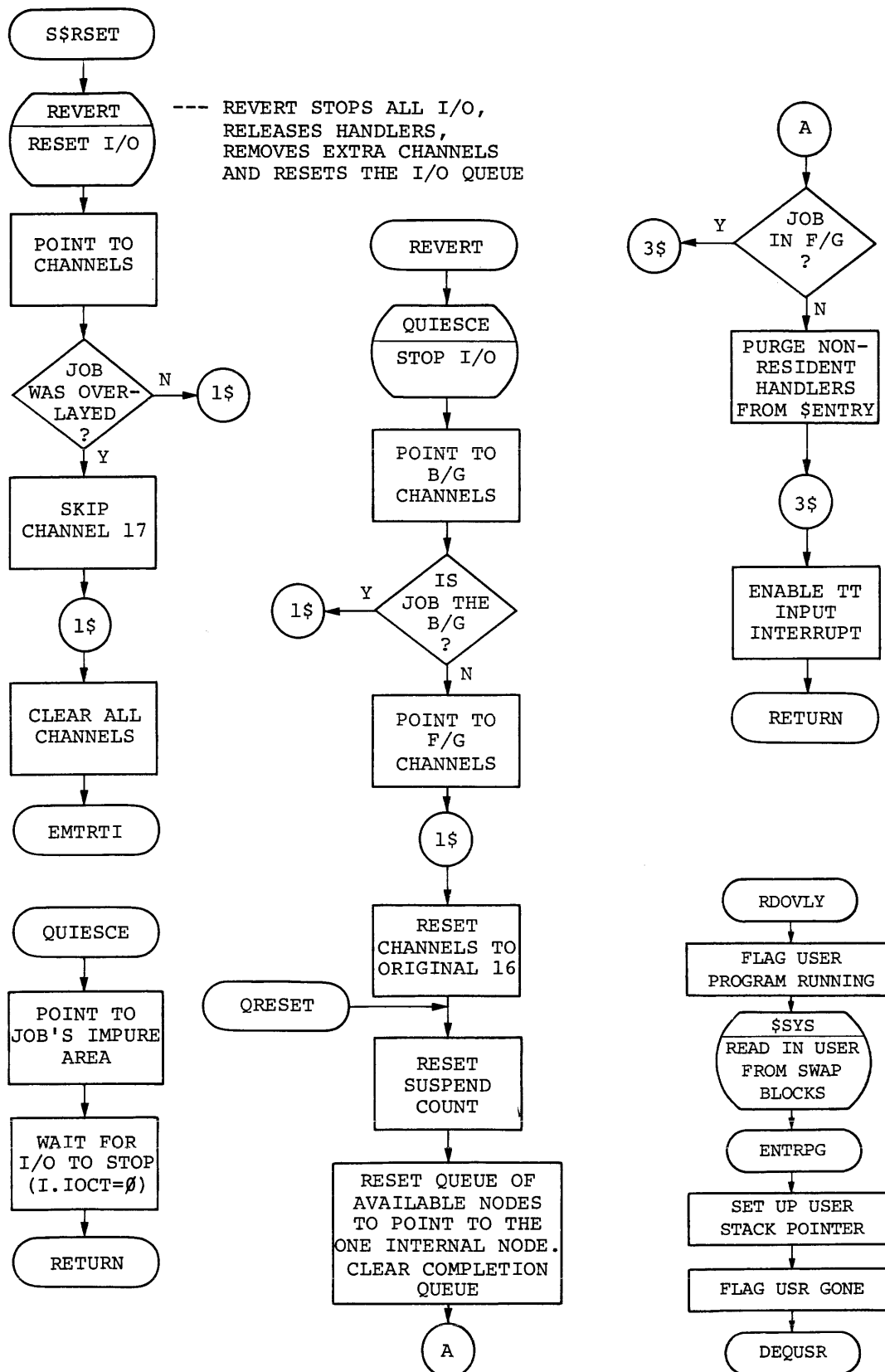
.EXIT
.CHAIN



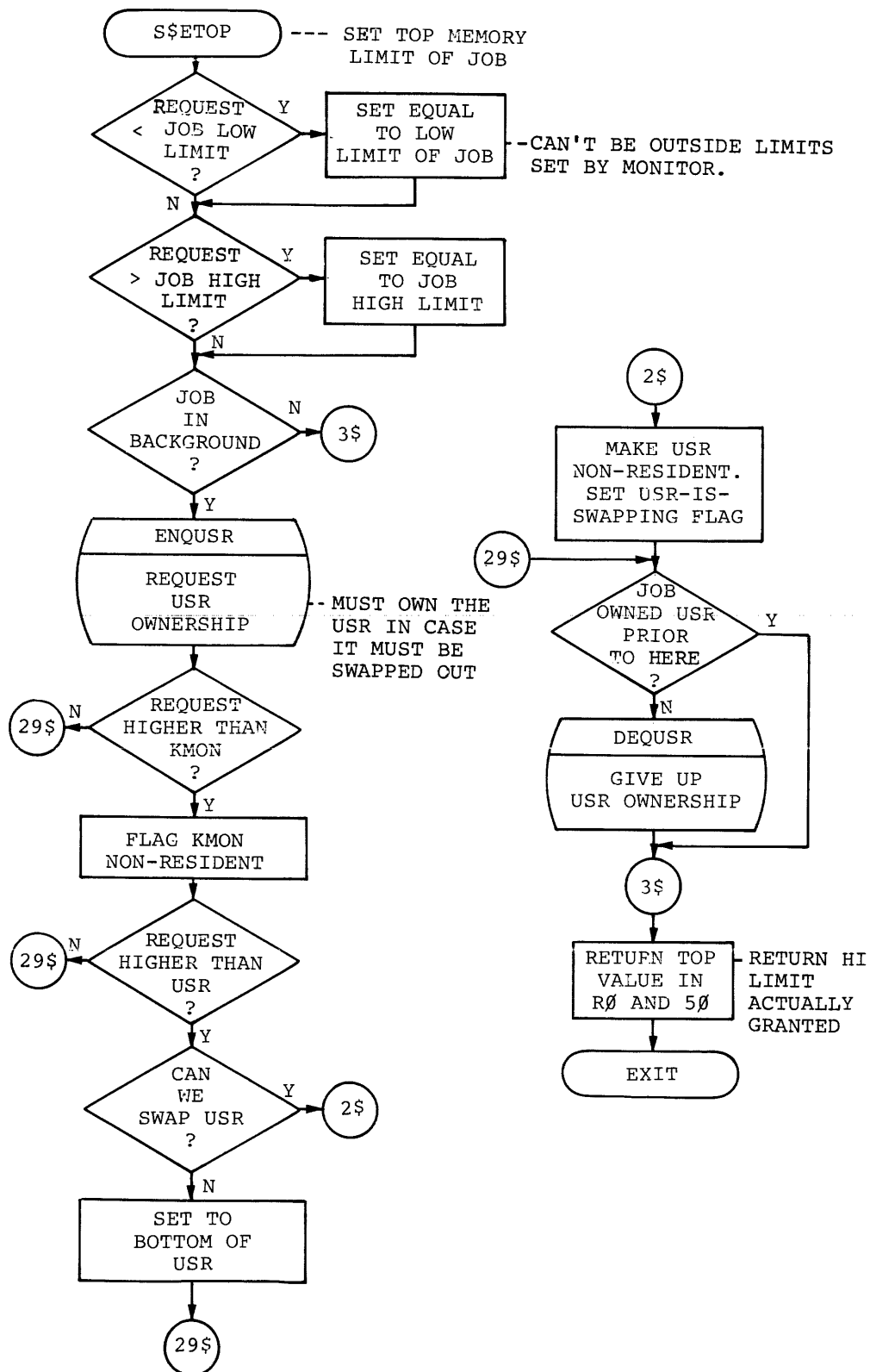
HARD AND SOFT RESET



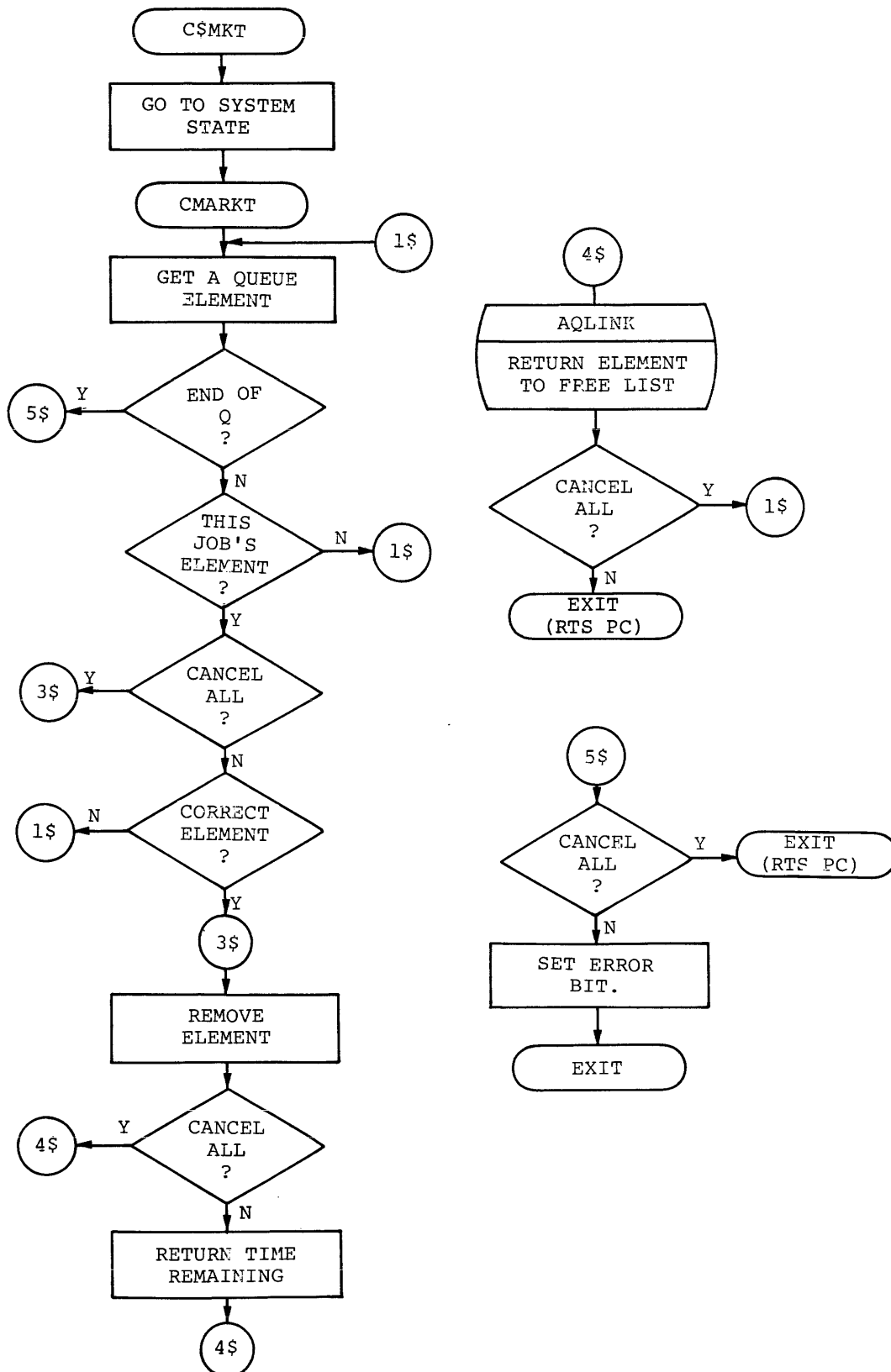
HARD AND SOFT RESET (CONT.)/RDOVLY



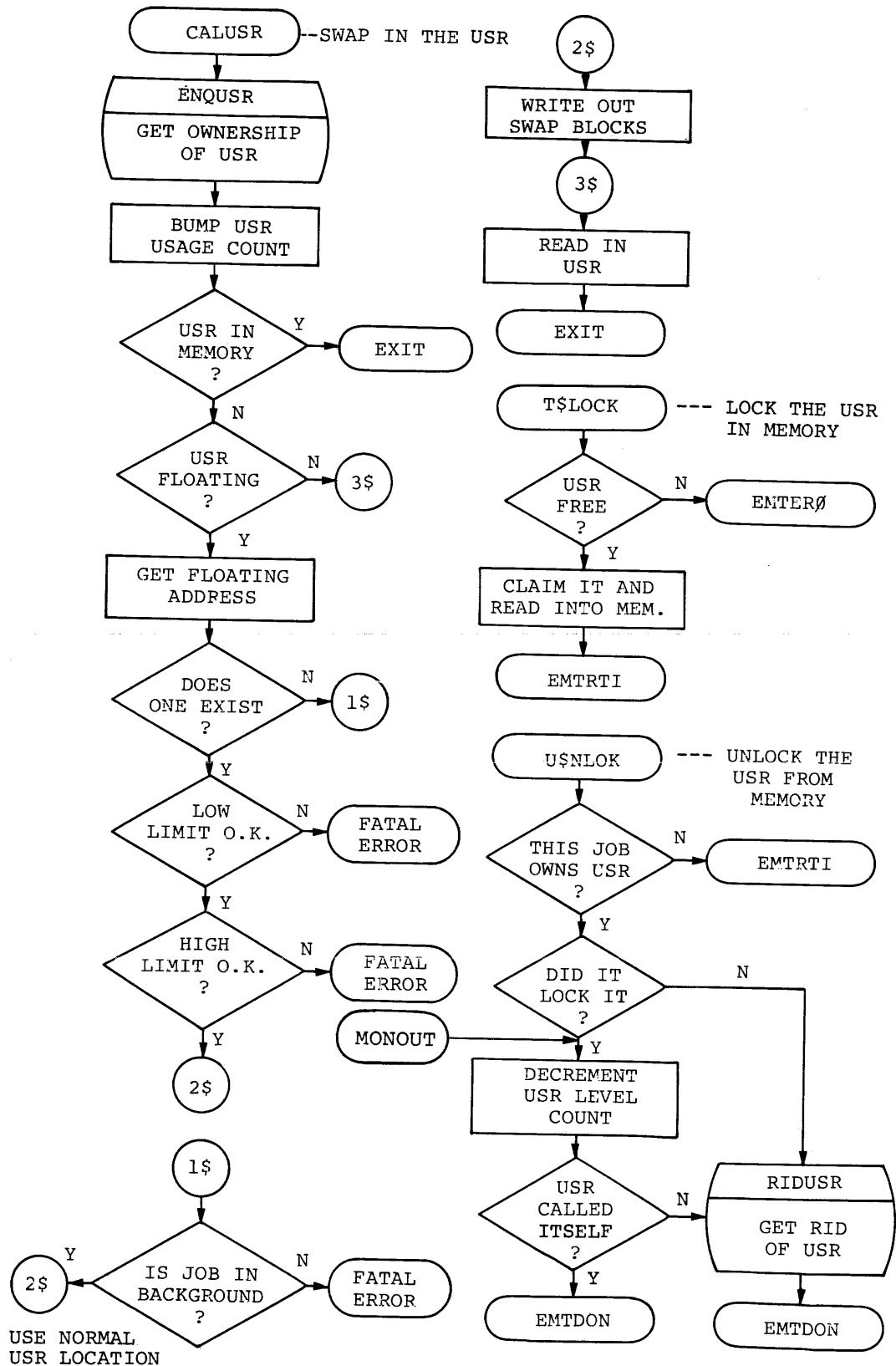
.SETTOP



CANCEL MARK TIME

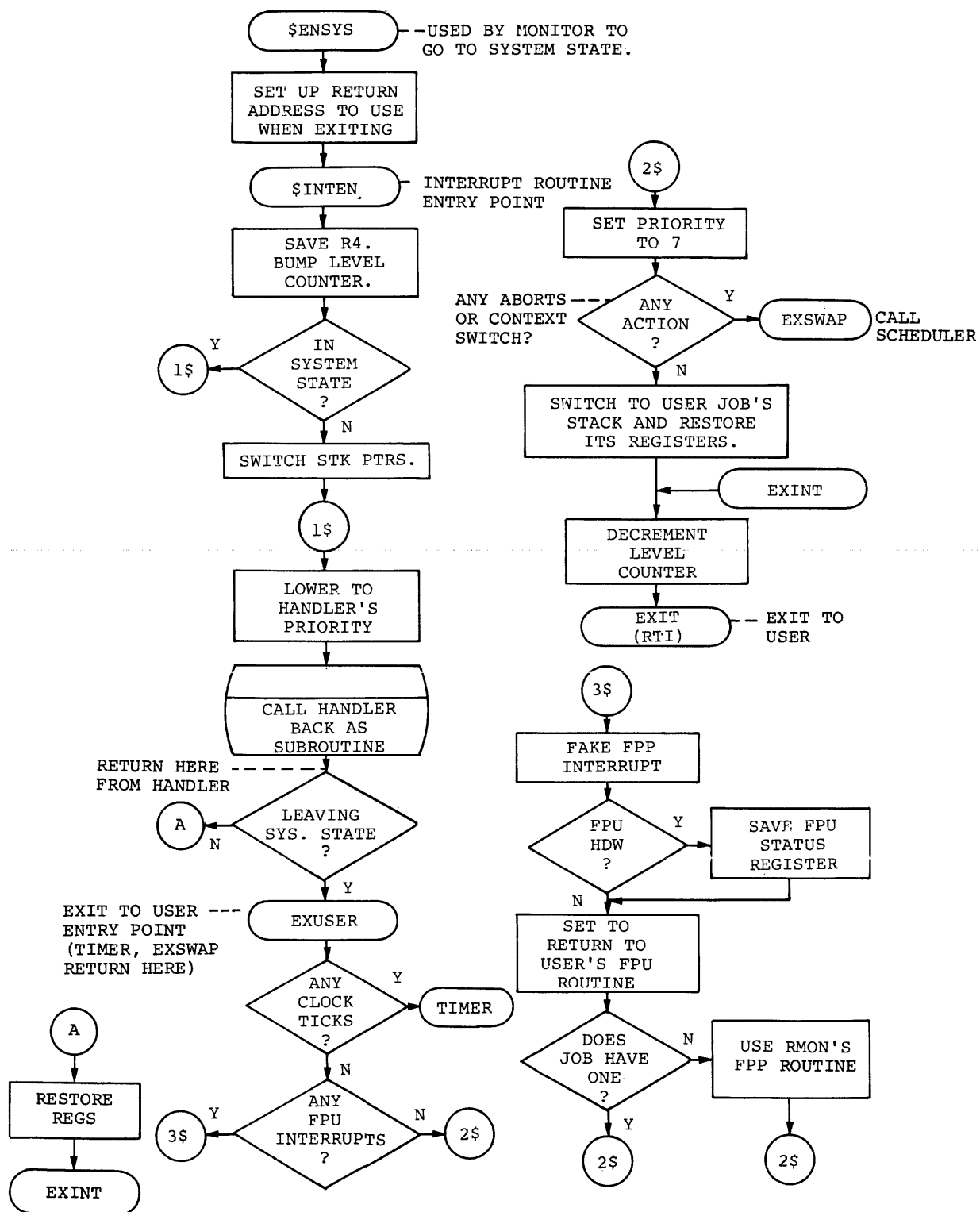


SWAP IN USR, LOCK/UNLOCK USR

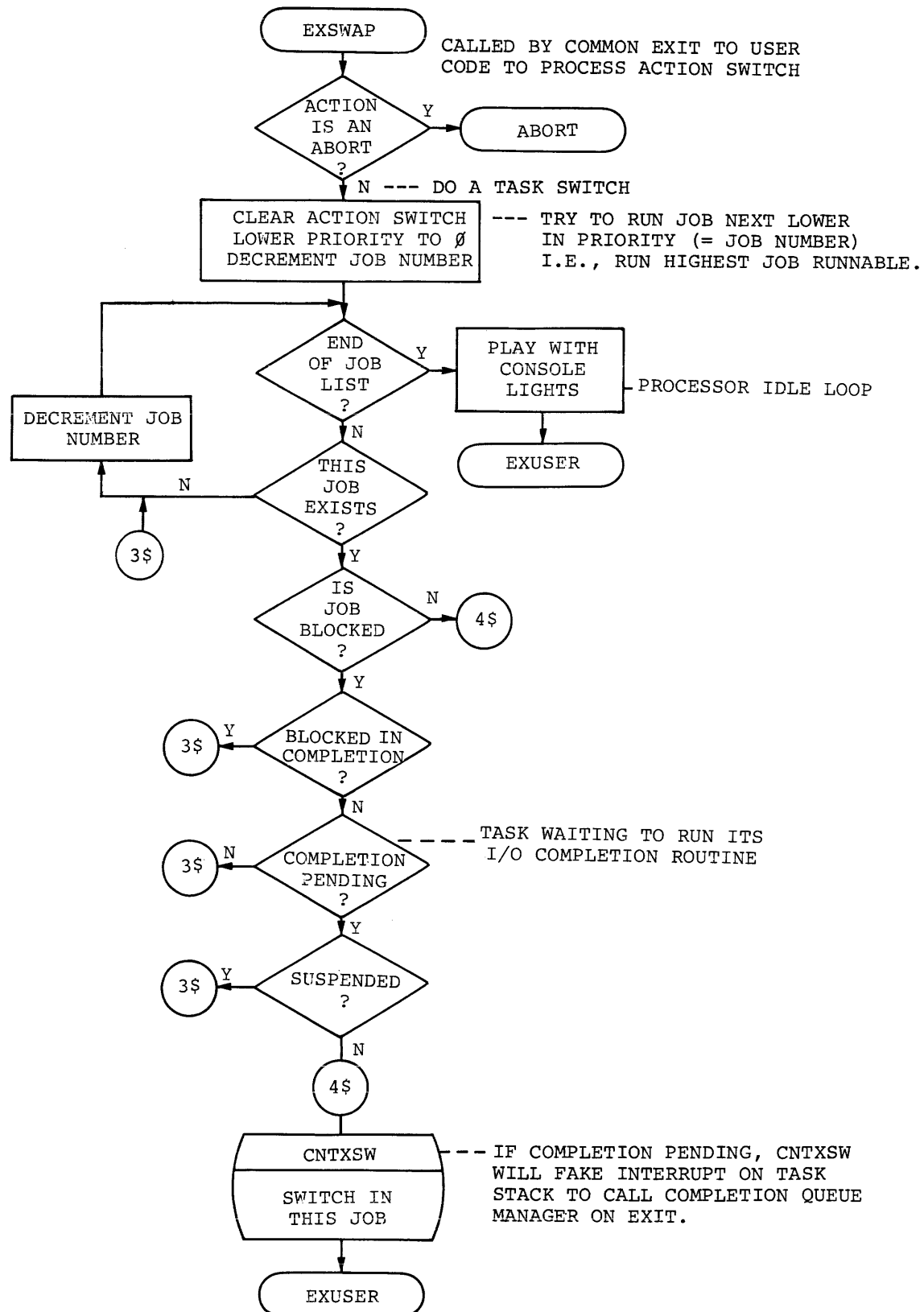


E.5.2 Job Arbitration, Error Processing

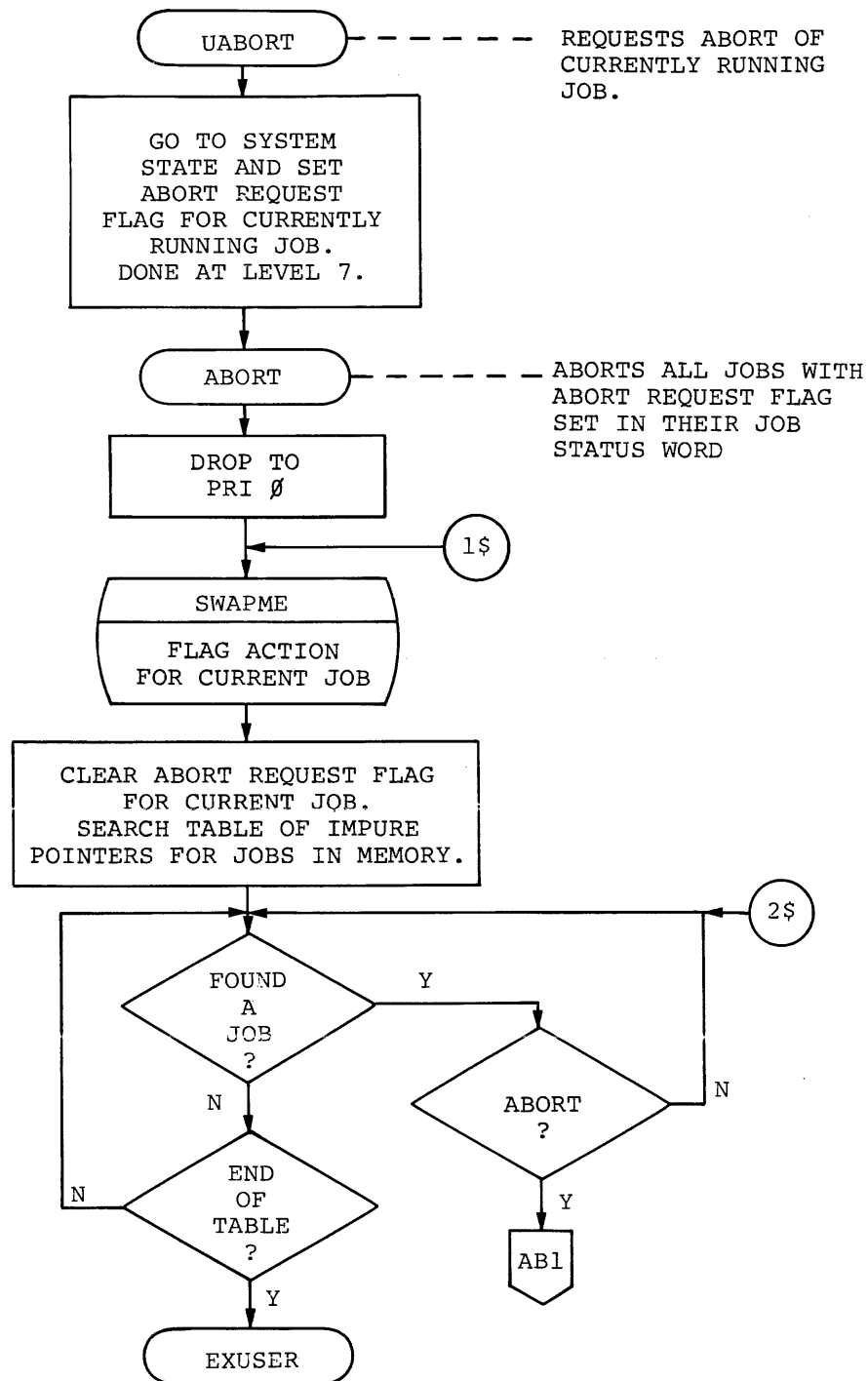
COMMON INTERRUPT ENTRY AND EXIT



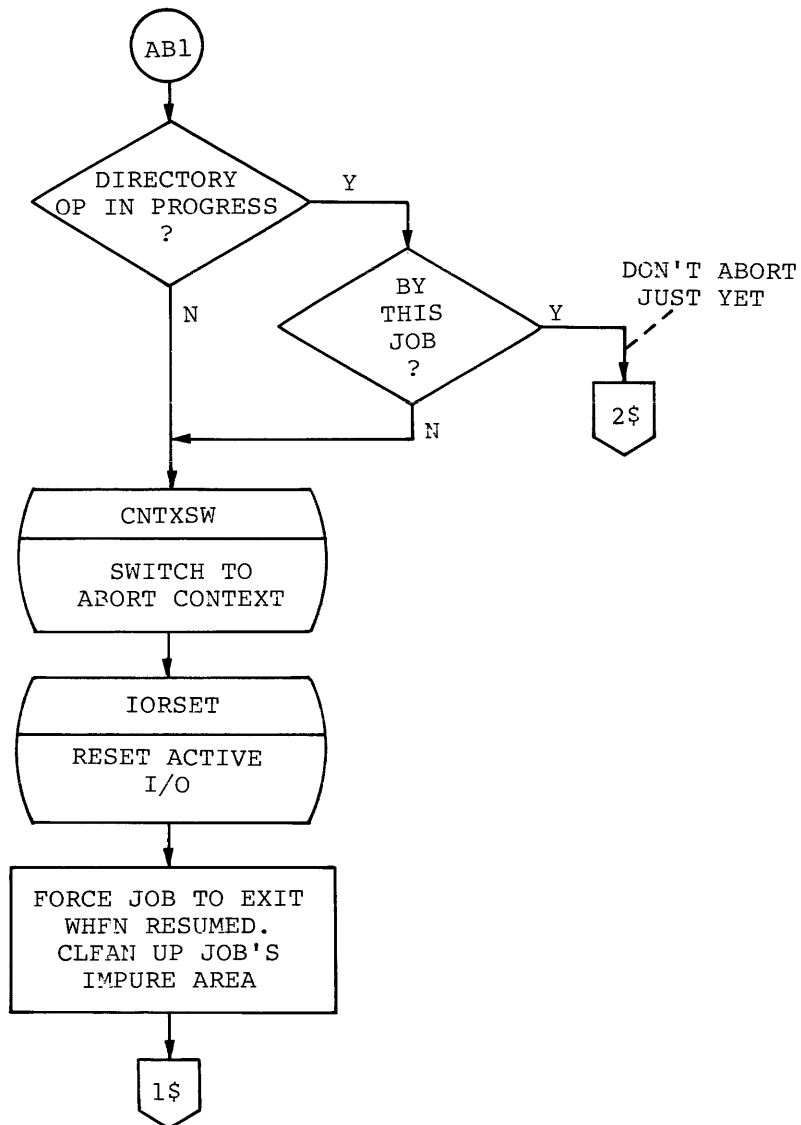
SCHEDULER



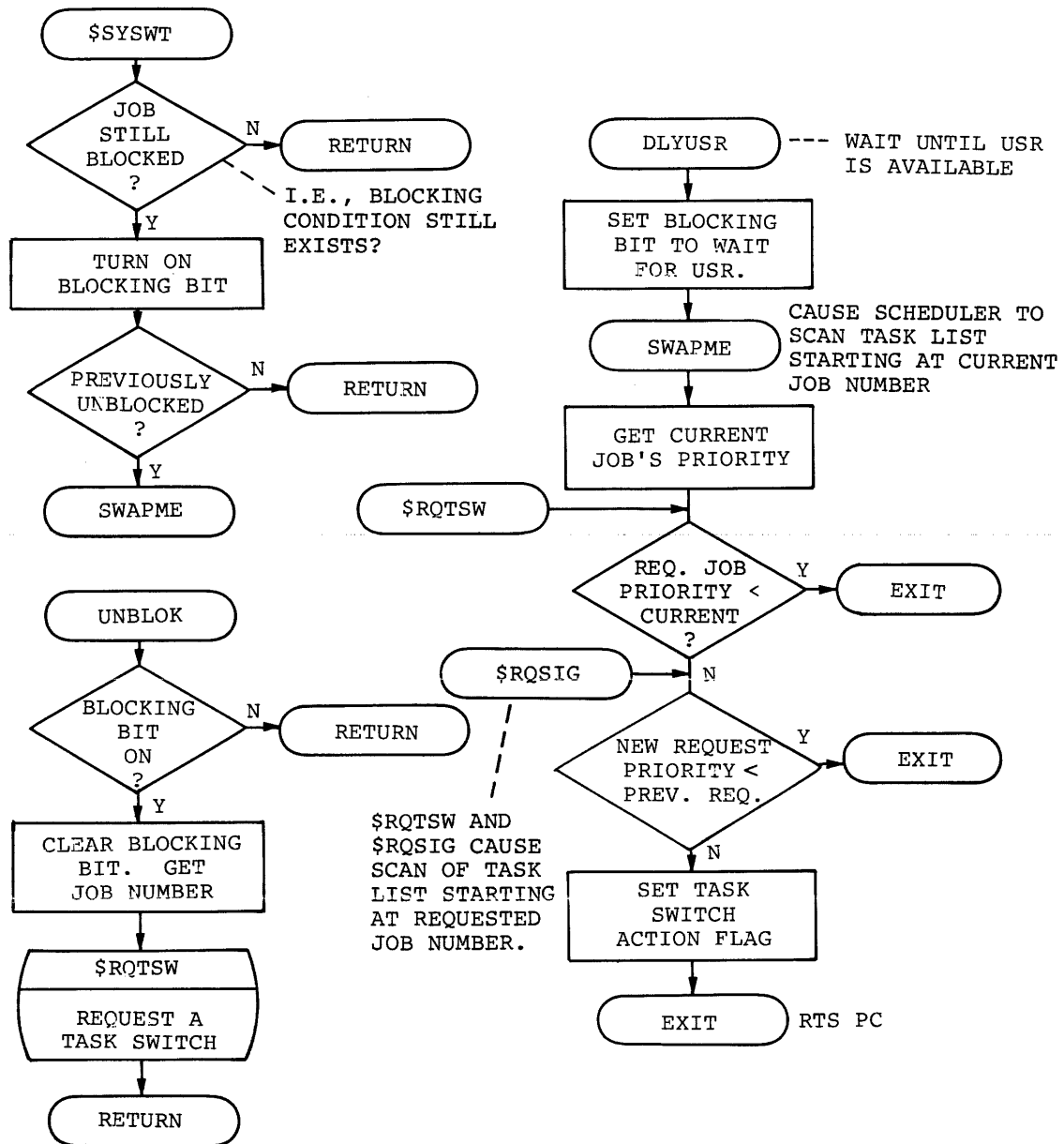
JOB ABORT



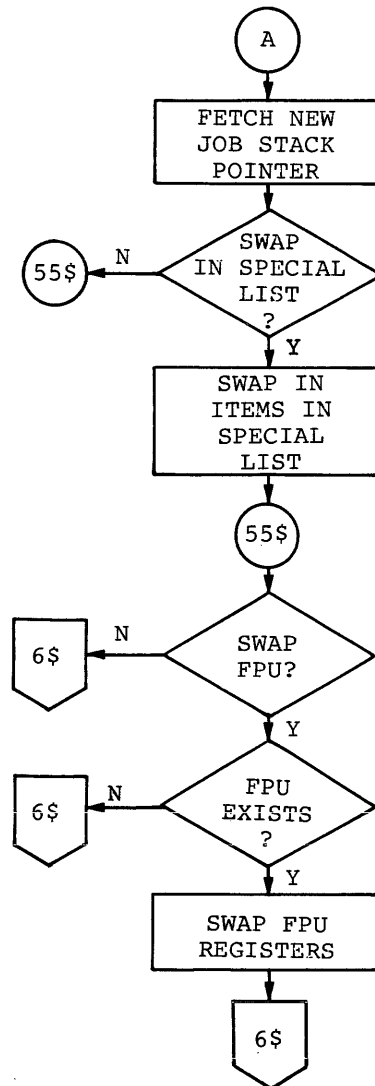
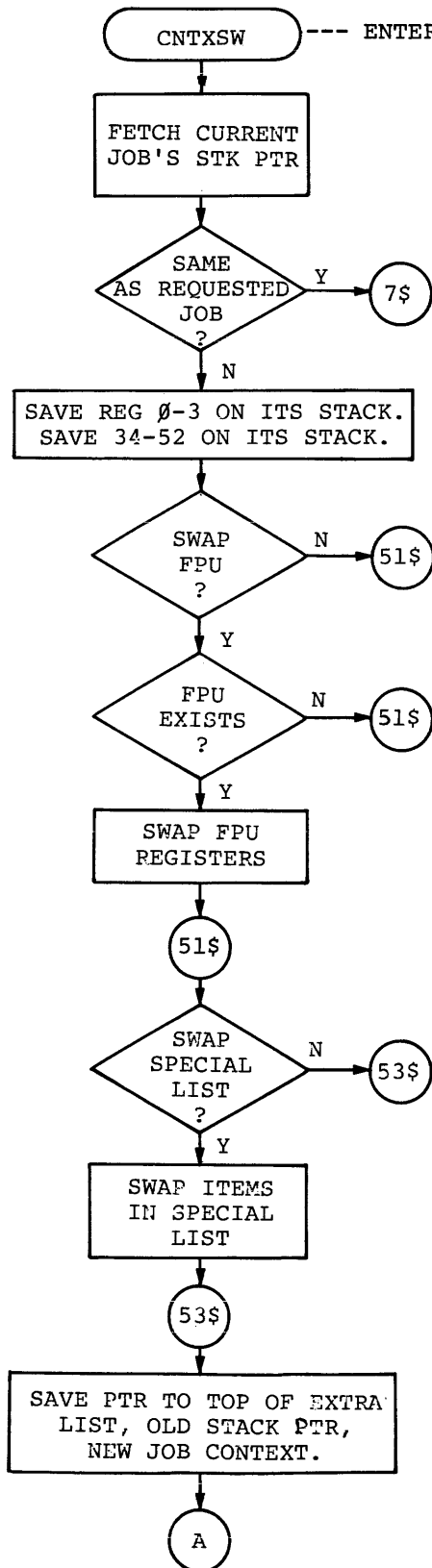
JOB ABORT (CONT.)



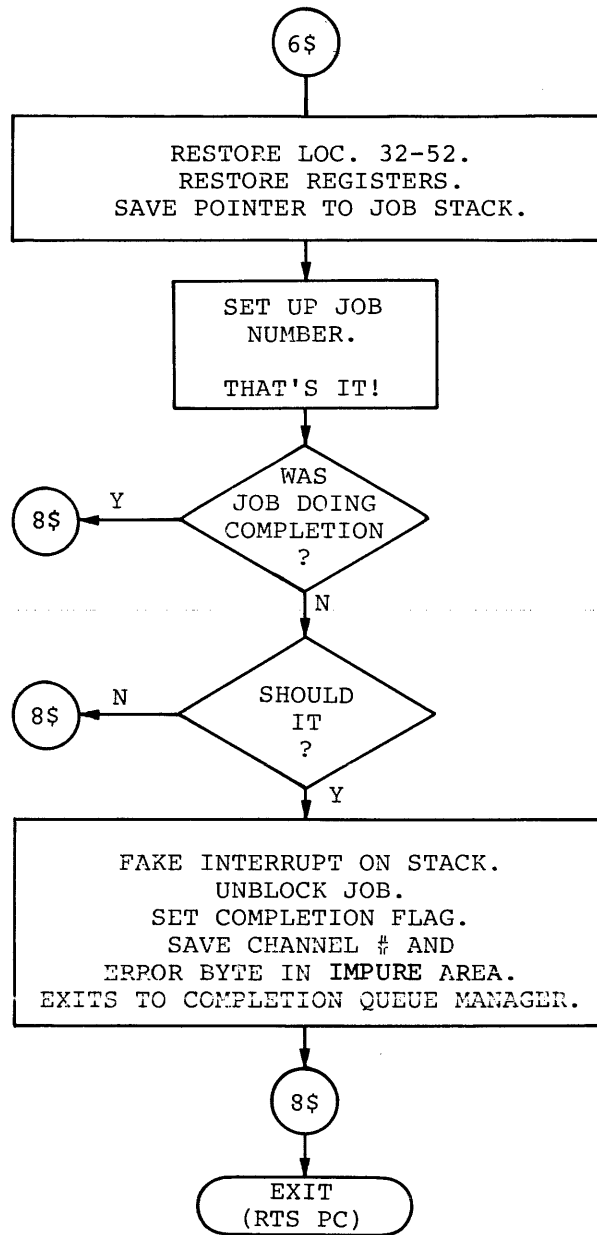
BLOCK A TASK/UNBLOCK A TASK
REQUEST TASK SWITCH



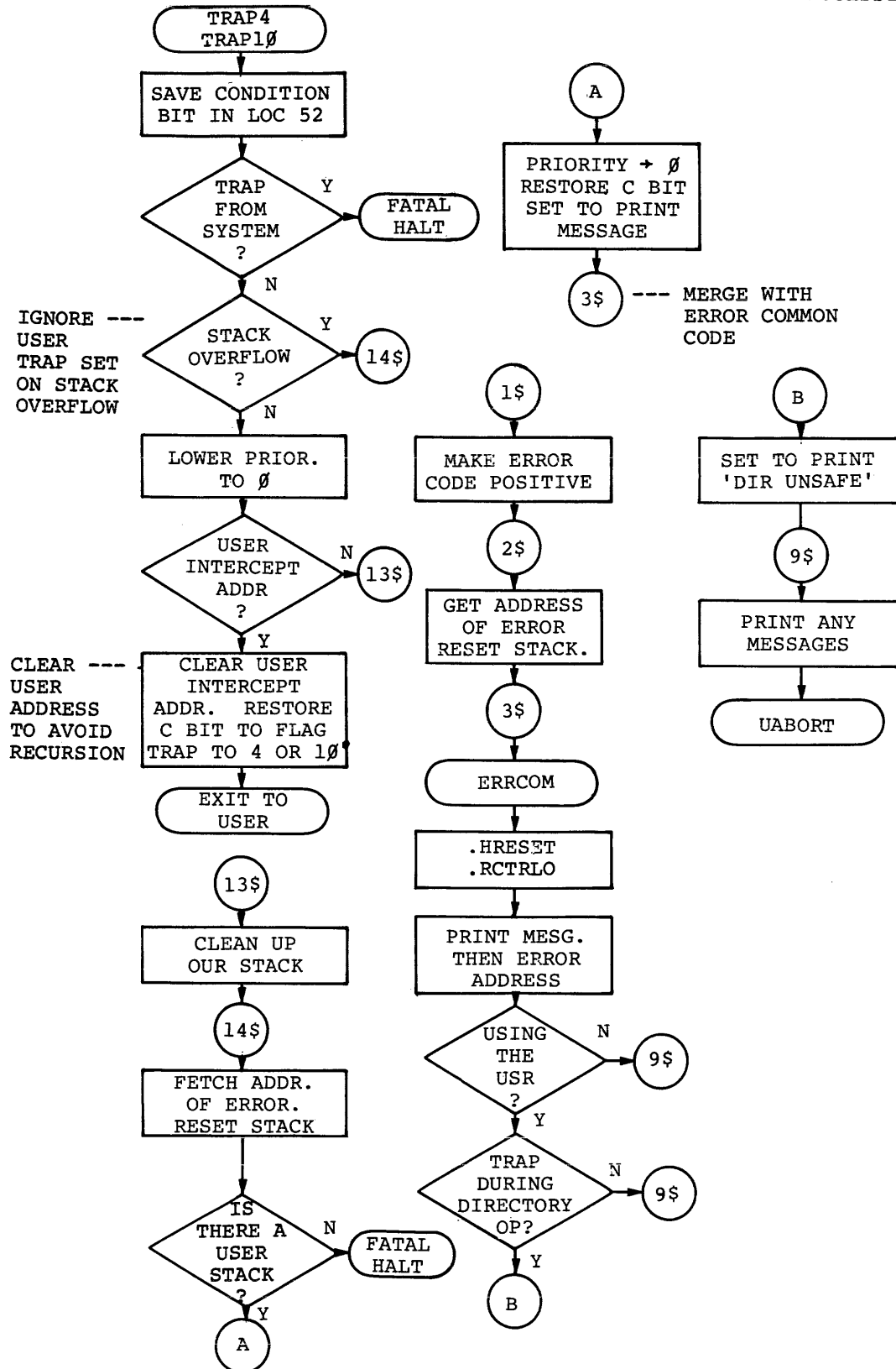
CHANGE CURRENT CONTEXT



CHANGE CURRENT CONTEXT (CONT.)

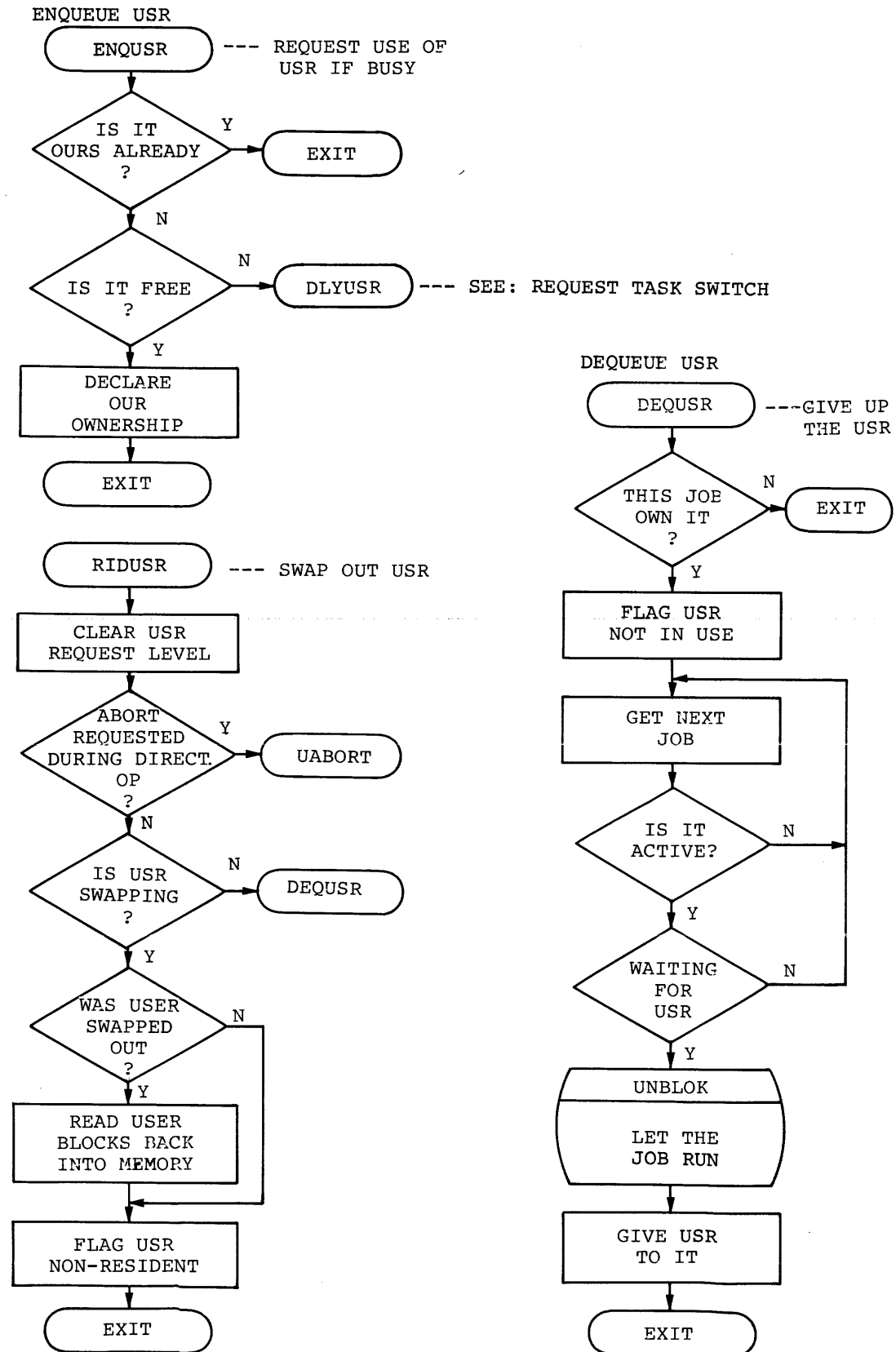


ERROR PROCESSING

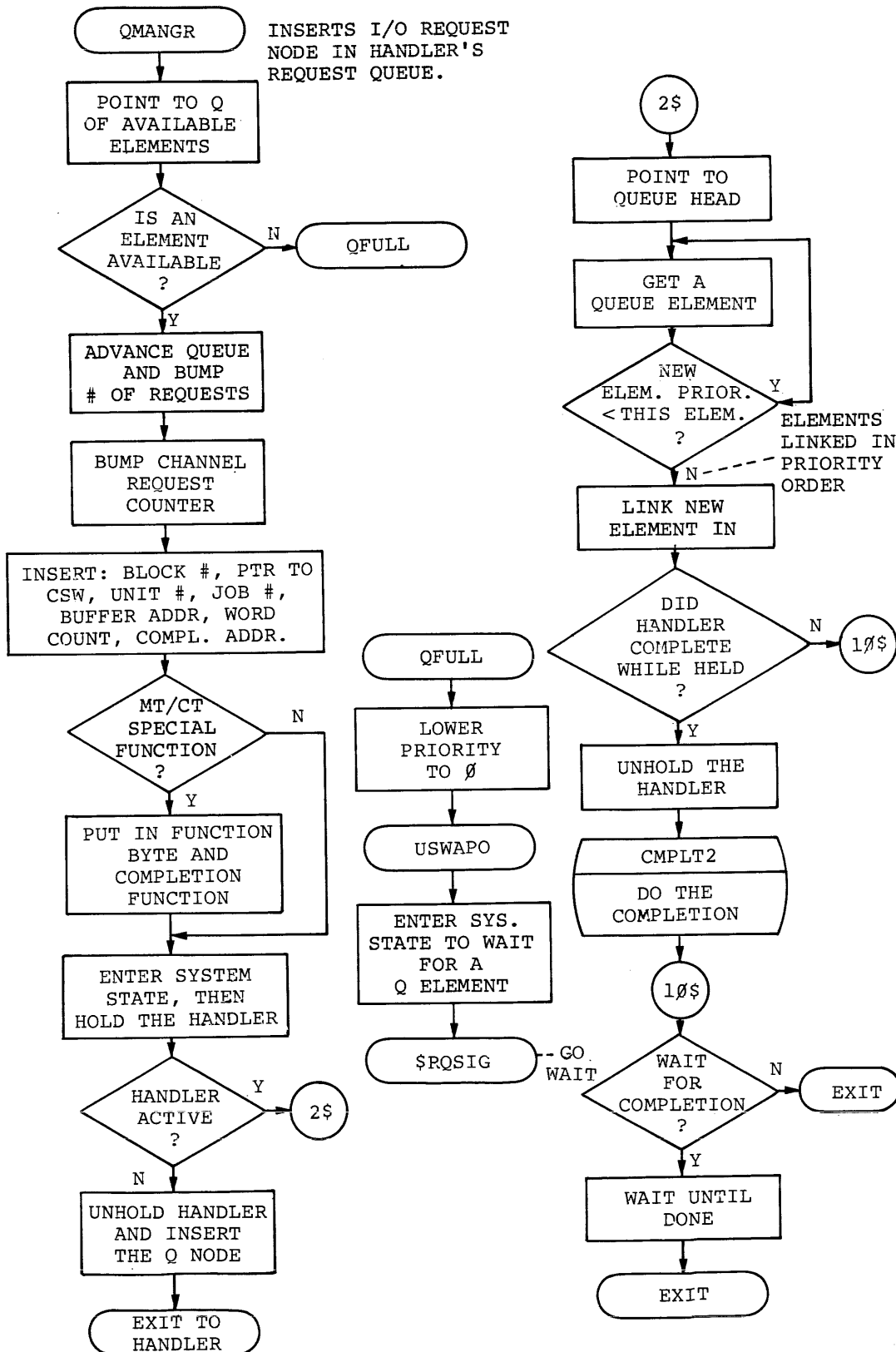


E.5.3 Queue Managers (I/O, USR, Completion)

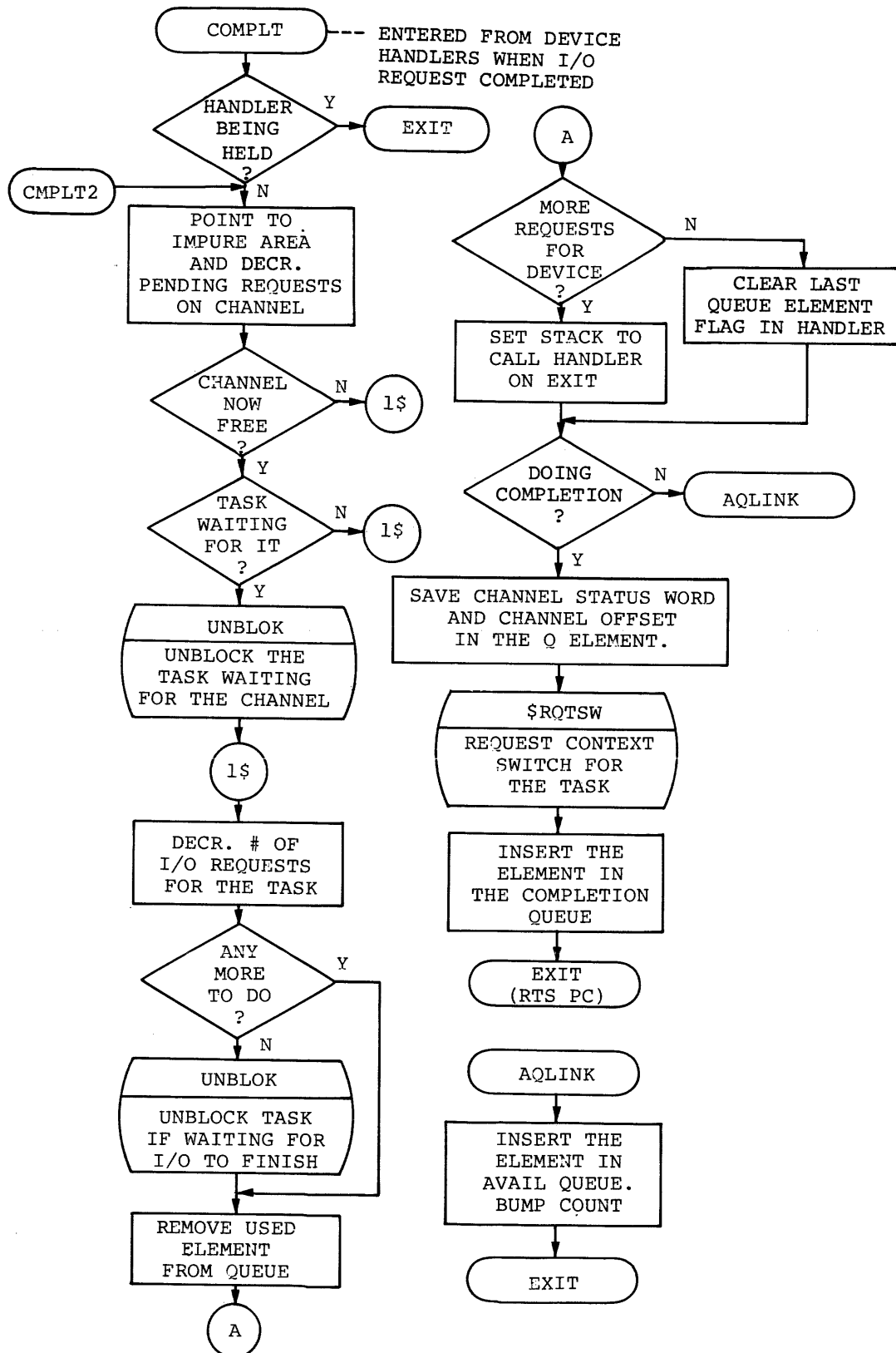
ENQUEUE/DEQUEUE USR



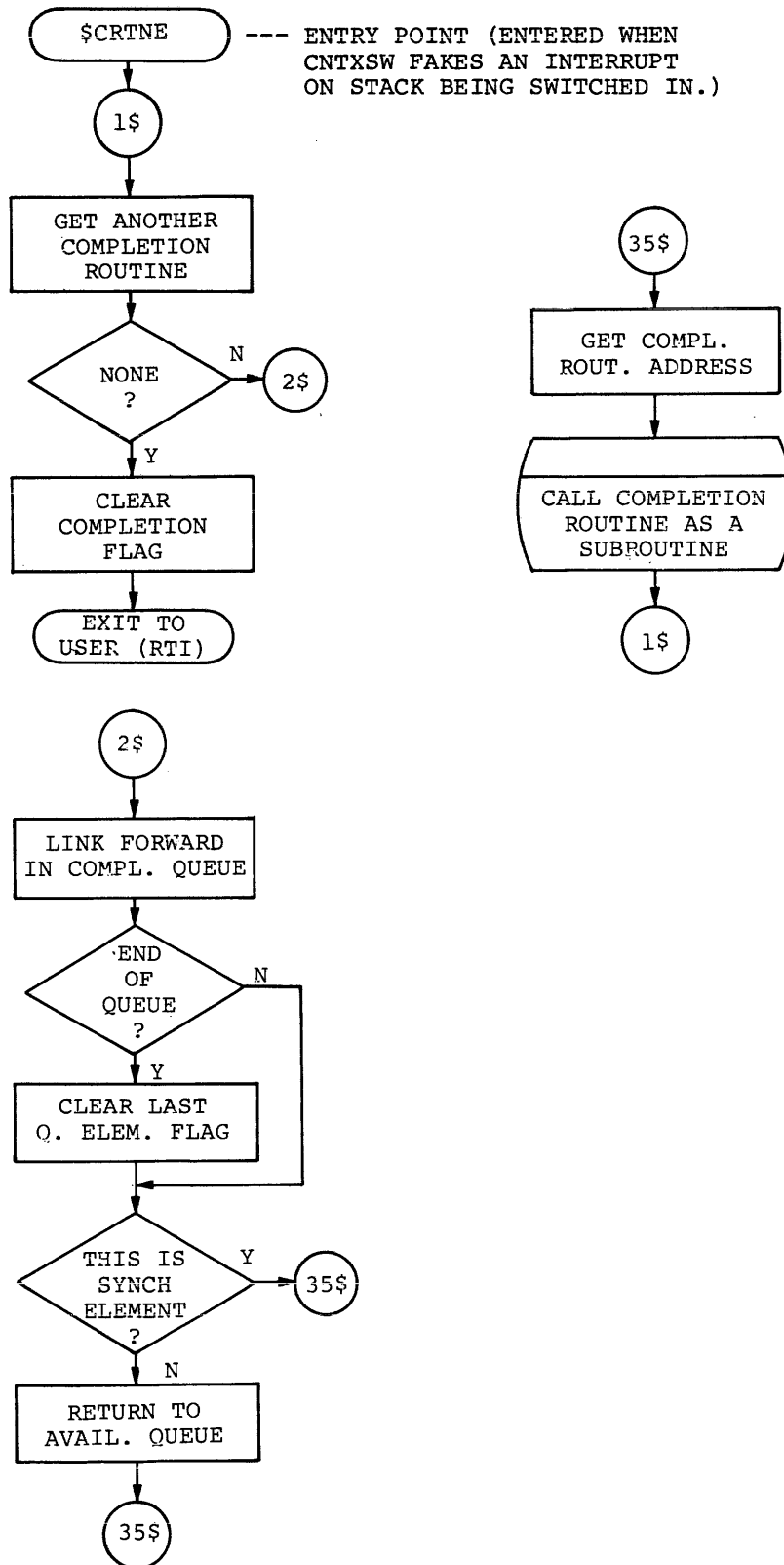
I/O QUEUE MANAGER



QUEUE COMPLETION

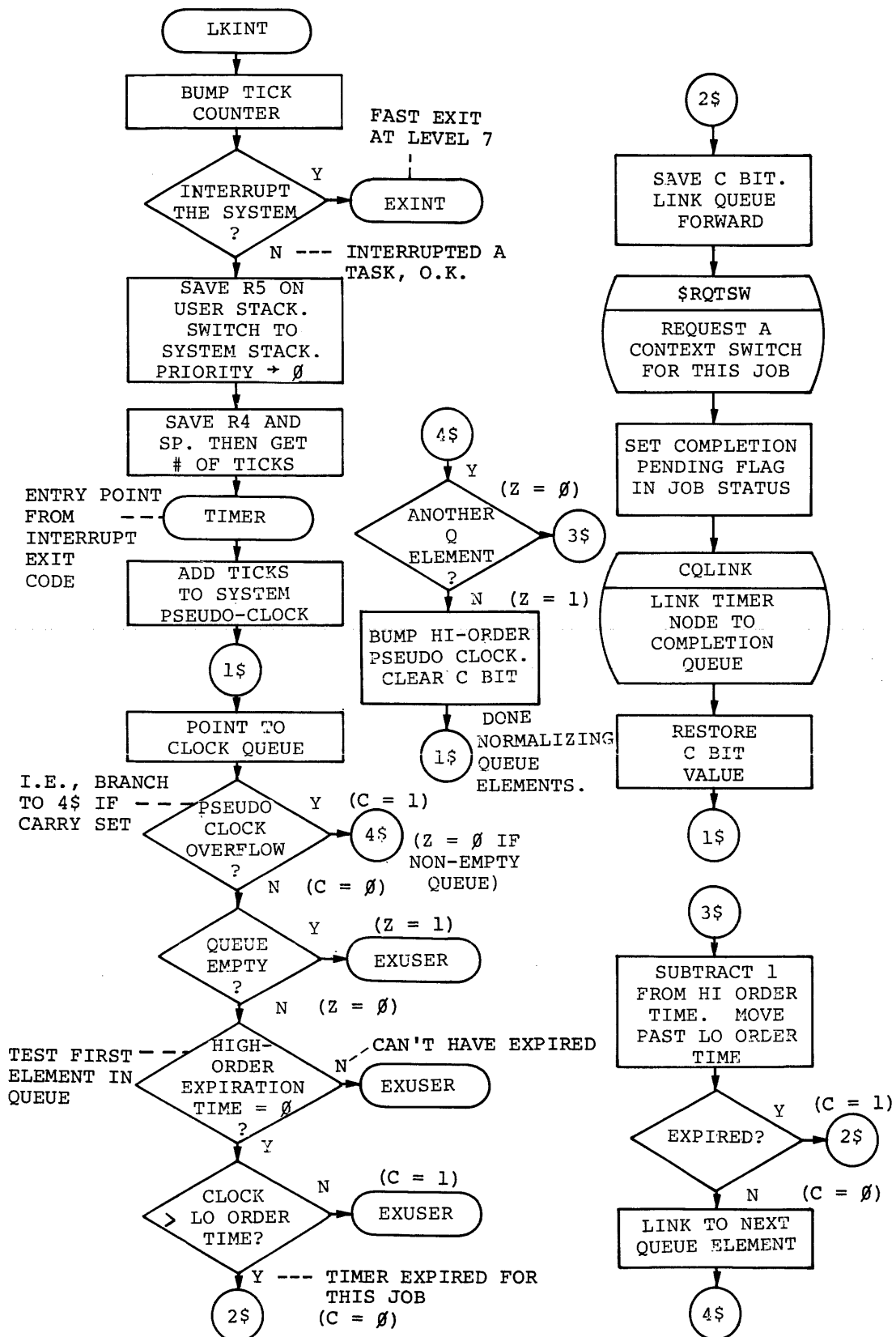


COMPLETION QUEUE MANAGER



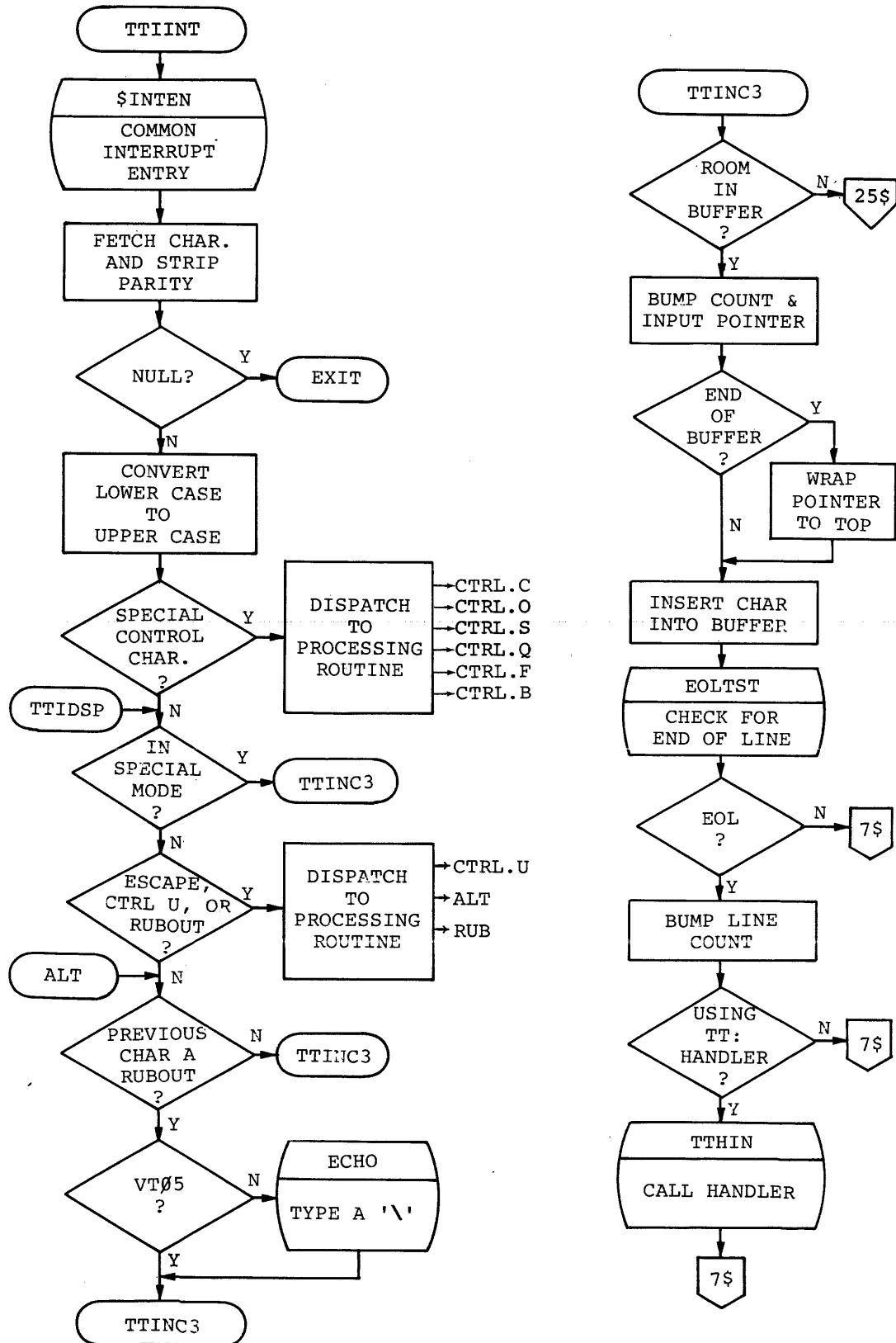
E.5.4 Clock Interrupt Service

CLOCK INTERRUPT HANDLER

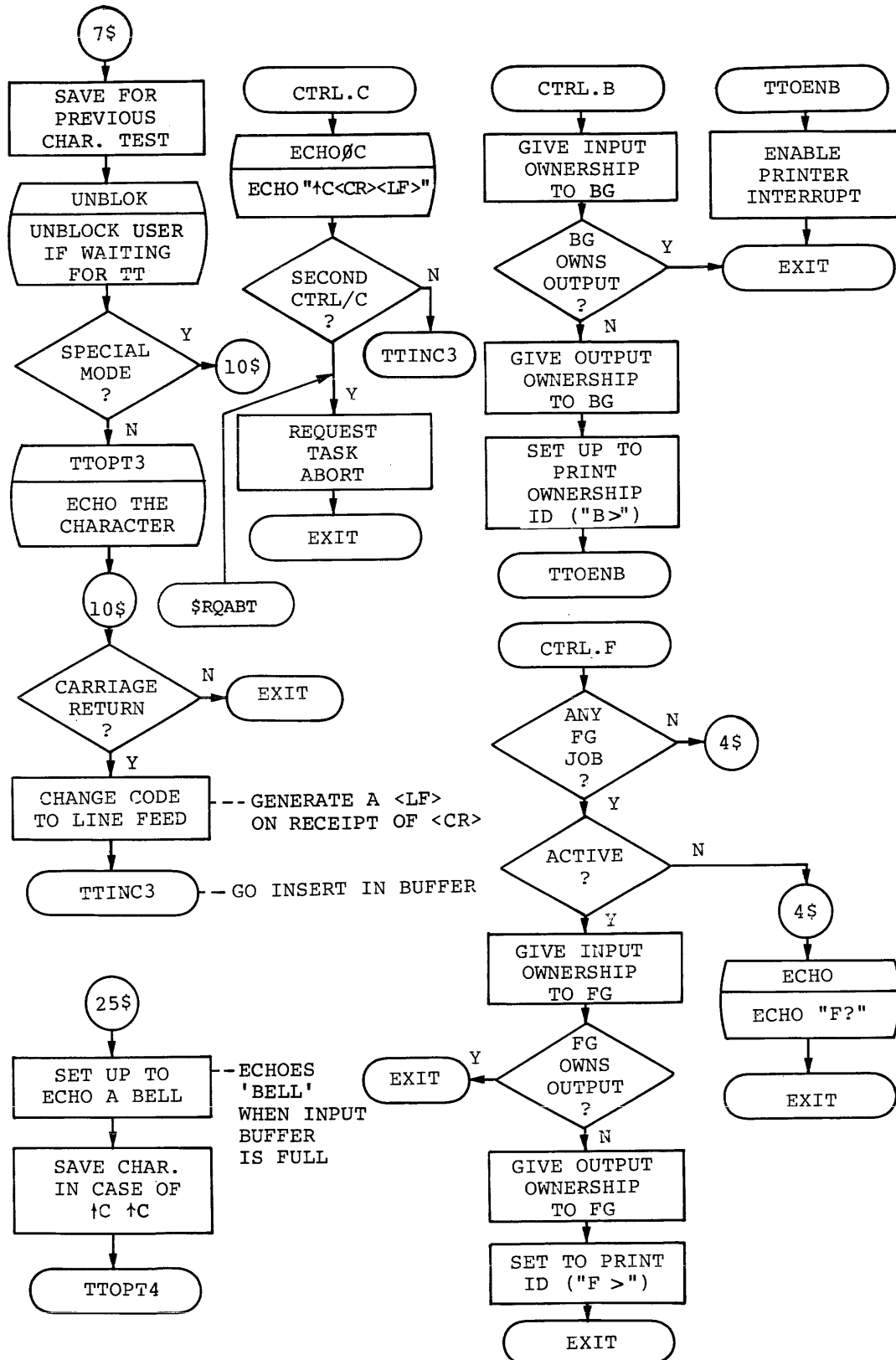


E.5.5 Console Terminal Interrupt Service .

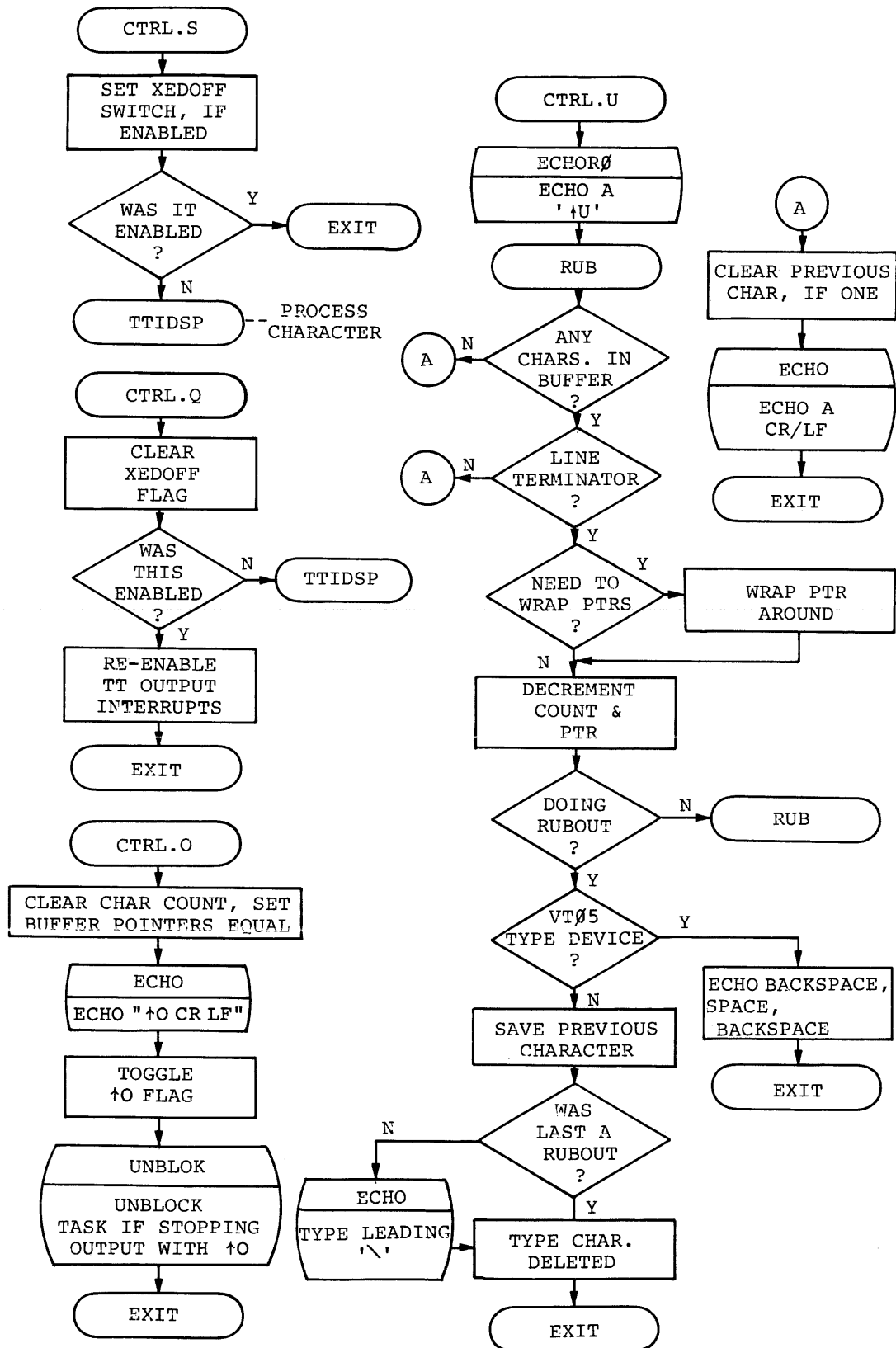
TT INPUT INTERRUPT ROUTINE



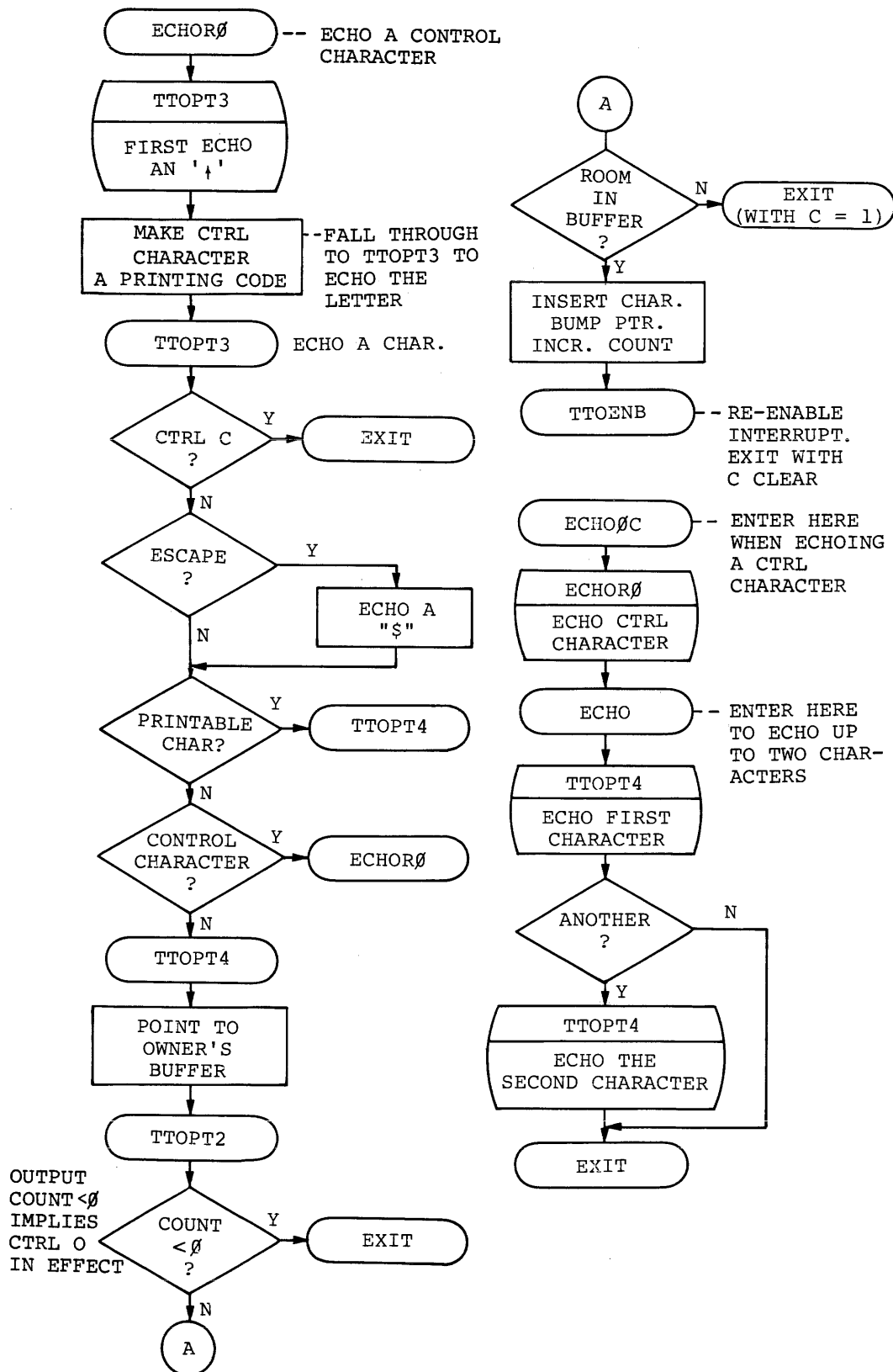
TT INPUT INTERRUPT ROUTINE (CONT.)



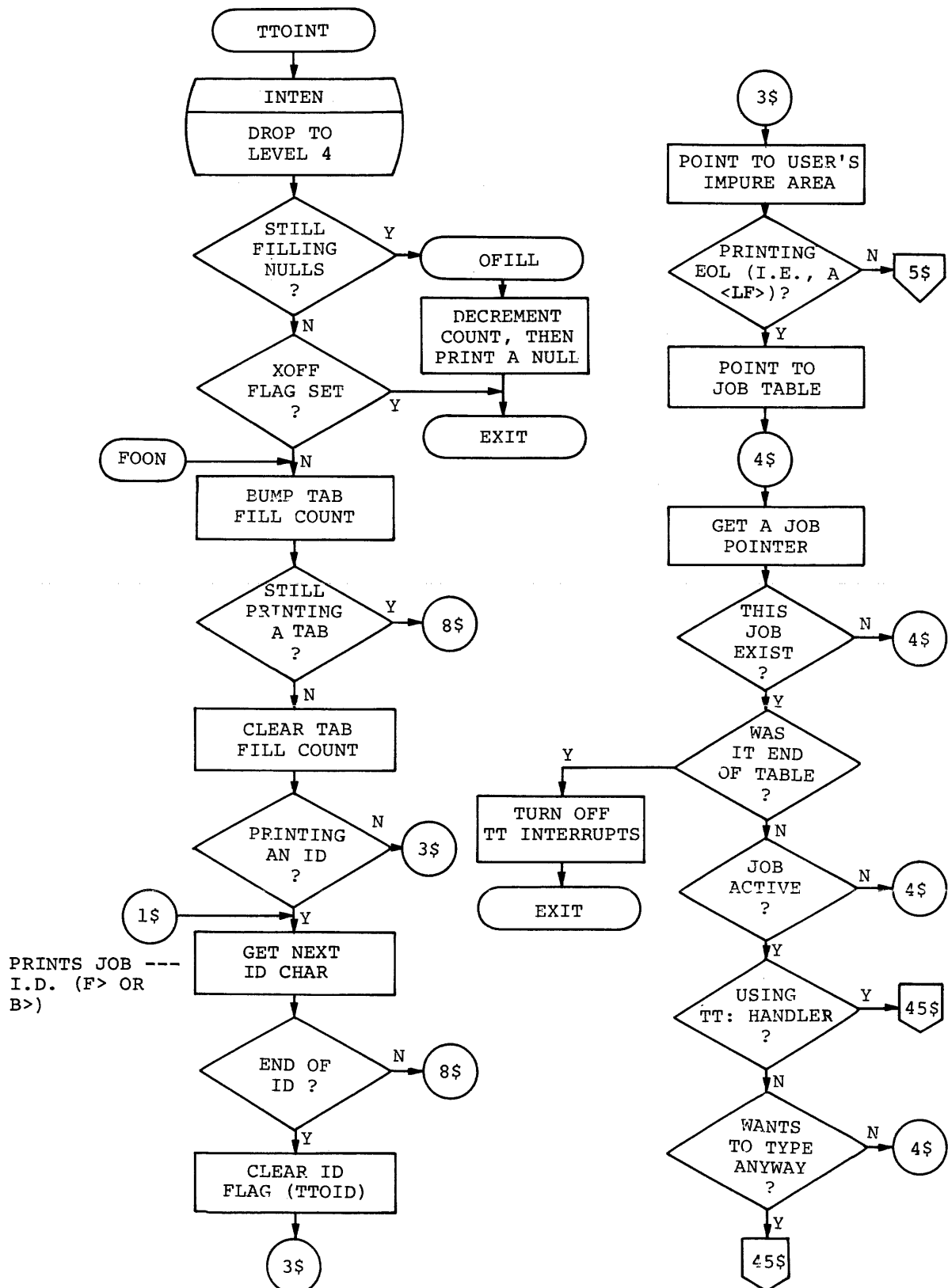
TT INPUT INTERRUPT ROUTINE (CONT.)



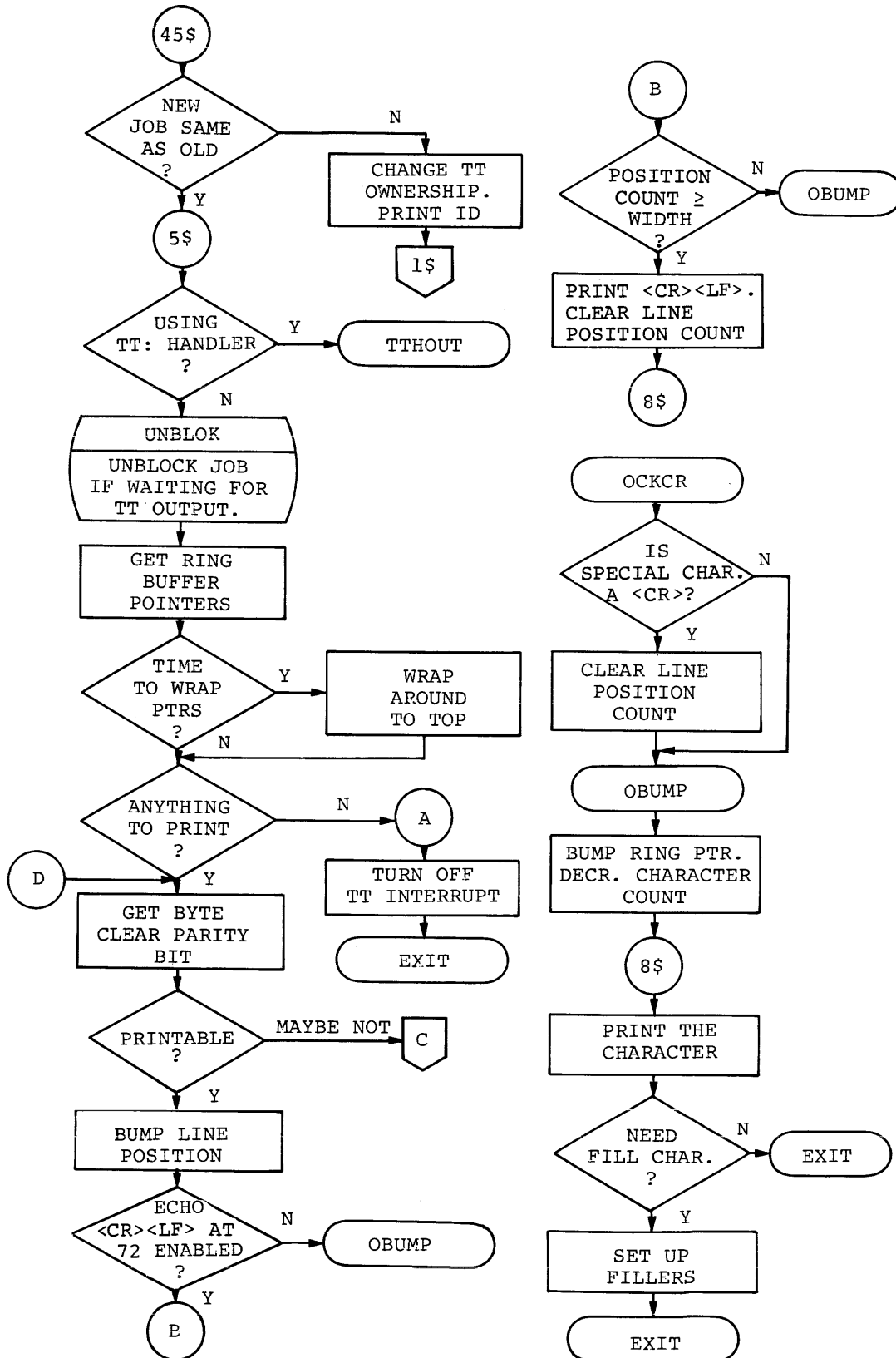
TT ECHO SERVICE



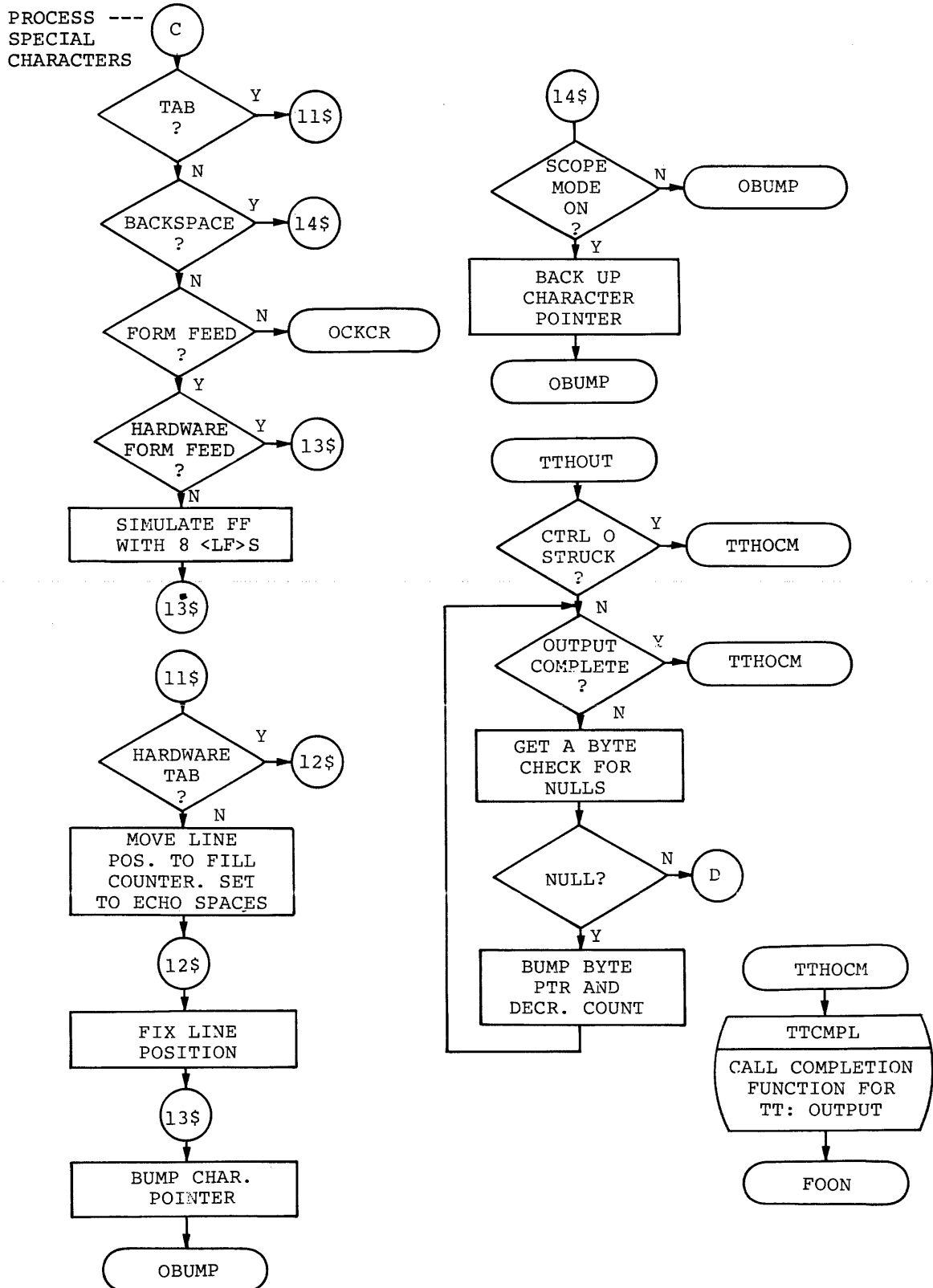
TT OUTPUT INTERRUPT ROUTINE



TT OUTPUT INTERRUPT ROUTINE (CONT.)

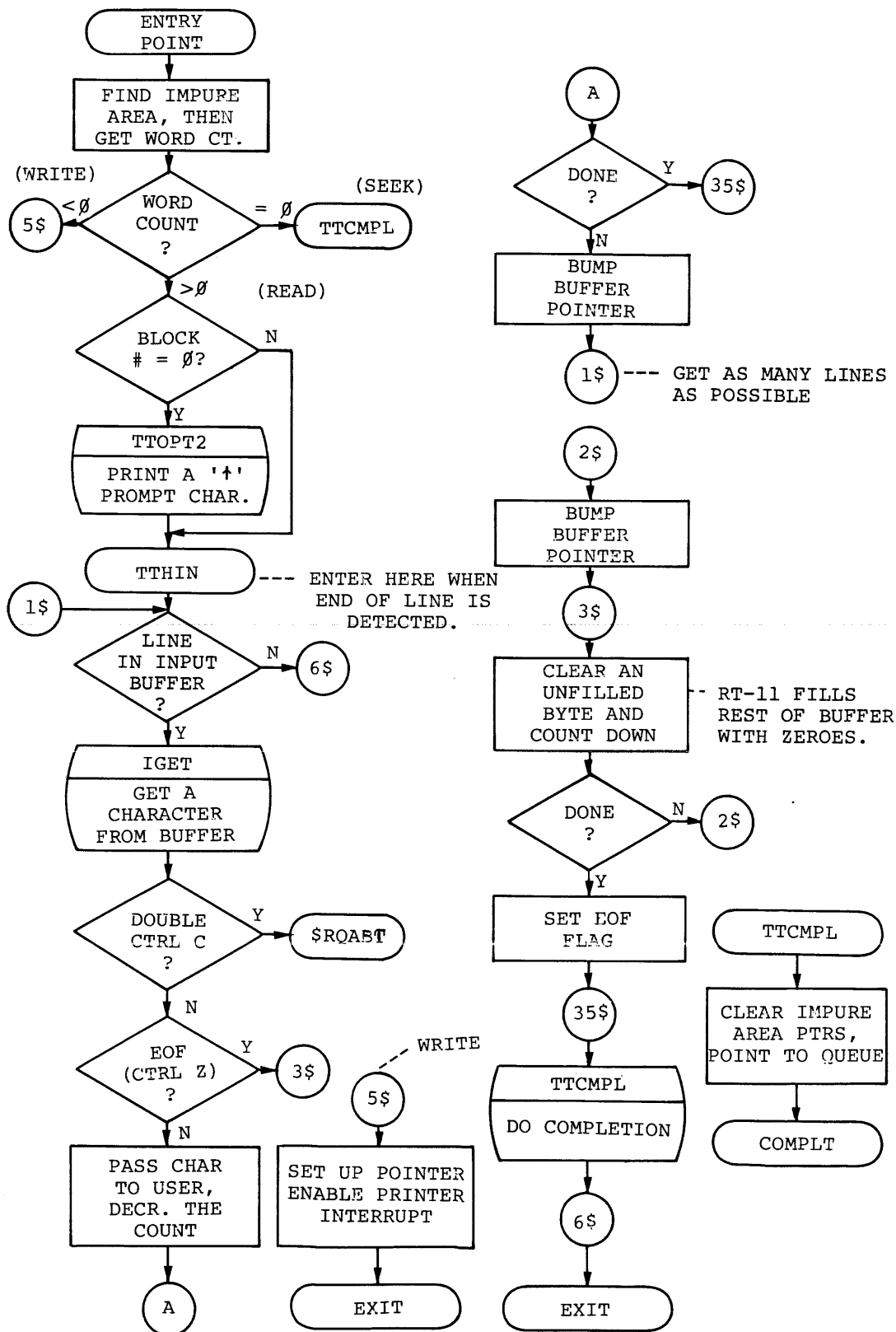


TT OUTPUT INTERRUPT ROUTINE (CONT.)

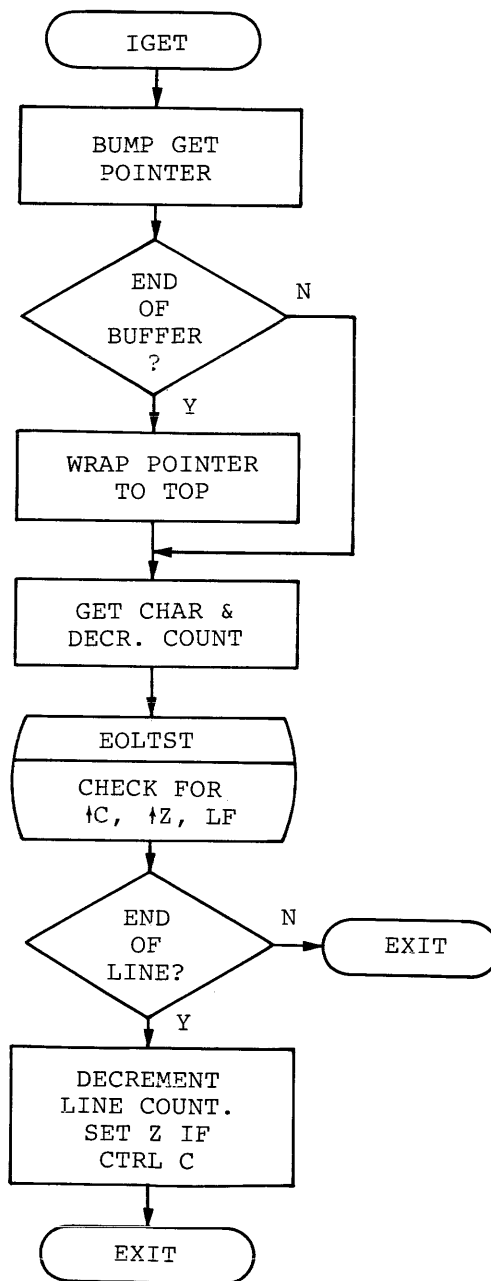


E.5.6 Resident Device Handlers (TT, Message)

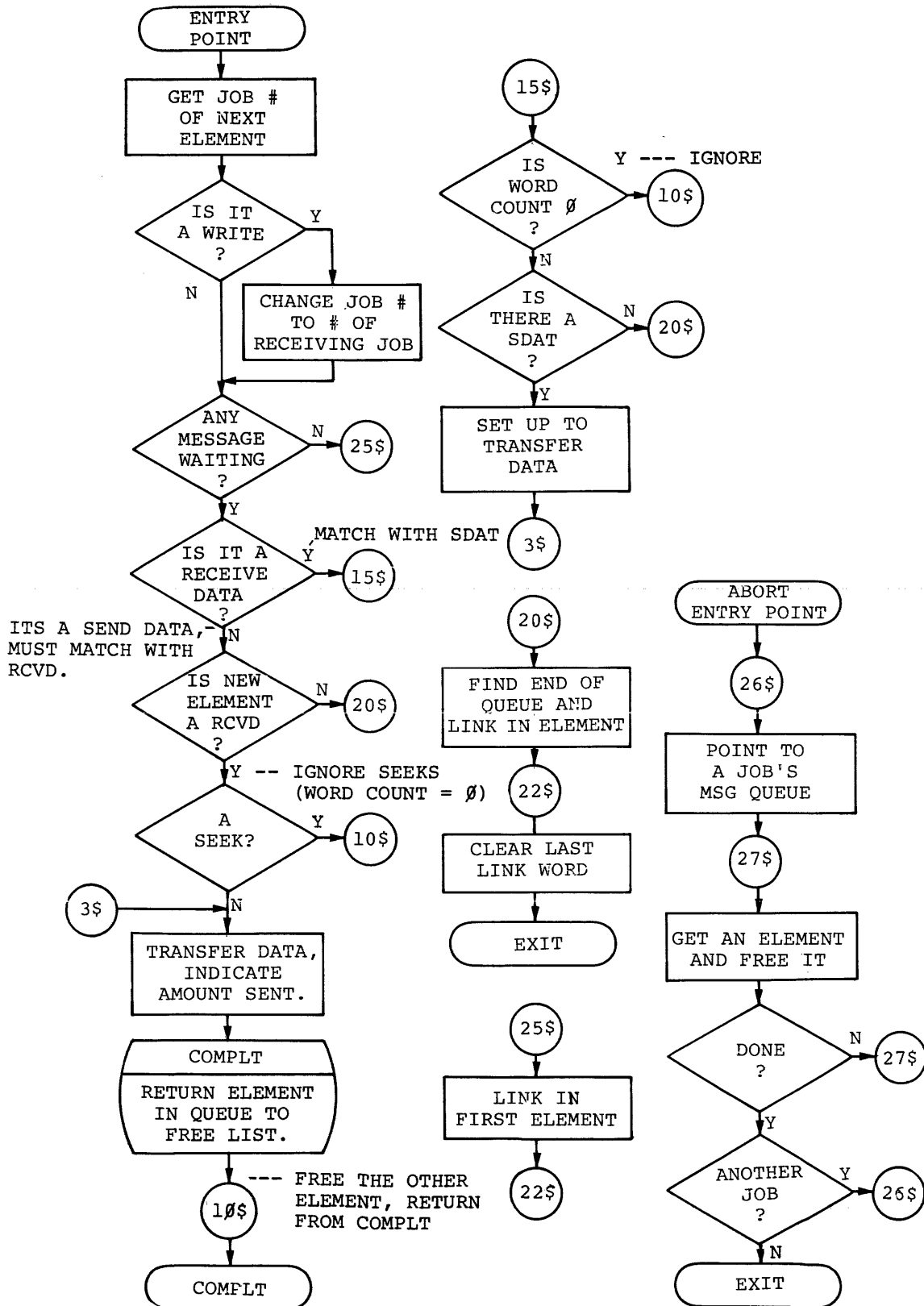
TT: RESIDENT HANDLER



TT: RESIDENT HANDLER (CONT.)



MESSAGE HANDLER



ENTRY POINT INDEX

Page numbers marked by an asterisk indicate the flowchart of the entry point.

\$CRTNE, E-135*	CTRLS, E-89*
\$SENSYS, E-122*	CTRLU, E-91*
\$INTEN, E-122*	
\$RQABT, E-141*, E-148	D\$DTAT, E-105*
\$RQSIG, E-126*, E-133	D\$LETE, E-66*, E-105*
\$RQTSW, E-126*	D\$STAT, E-66*
\$SYNCH, E-114*	D\$VICE, E-113
\$SYSWT, E-126*	D, E-4, E-6*
ABORT, E-123, E-124, E-150*	DATE, E-18*
ADTRAN, E-13*	DELETE, E-36*
ALT, E-89*, E-140*	DELOUT, E-34*, E-36, E-38, E-42,
AQLINK, E-134*	E-43
ASSIGN, E-20*	DEQUSR, E-117, E-132*
	DLEET, E-56*
B, E-4, E-5*	DLYUSR, E-126*, E-132
BADCOM, E-4*, E-7, E-9	DOSAVE, E-19
BEGIN, E-7*, E-9	
BLKCHK, E-56, E-57*	
	E\$NTER, E-66*, E-105*
C\$DFN, E-72*, E-112*	E\$XIT, E-78*, E-115
C\$DFN2, E-66*, E-72	E, E-4, E-5*
C\$HAIN, E-75*, E-115*	E376, E-64, E-81*, E-104
C\$LOS2, E-66*, E-69, E-105*, E-106	E5ER0, E-111*
C\$LOSE, E-69*, E-106*	E5ER1, E-111*
C\$MKT, E-119*	ECHO, E-143*
C\$PYCH, E-110*	ECHOOC, E-143*
C\$SIGN, E-66*, E-106*	ECHOR0, E-143*
C\$SISP, E-66*, E-106*	EMTCOM, E-104*
C\$STAT, E-111*	EMTDON, E-65*, E-68, E-69, E-70,
CALUSR, E-82*, E-120*	E-72, E-73, E-74, E-75, E-76,
CCBB0, E-15*	E-77, E-79, E-80*, E-88,
CDFN, E-30*	E-89, E-90, E-91, E-107*,
CHANER, E-64, E-65*	E-108, E-120
CHKSP, E-113*	EMTER0, E-107*, E-112, E-113,
CHXIT, E-75, E-78*	E-120
CLOCOM, E-33, E-42*	EMTOUT, E-64, E-65*, E-68, E-81
CLSQSH, E-36, E-42*	EMTPRO, E-64*, E-104*
CMARKT, E-116, E-119*	EMTRTI, E-105, E-106, E-107*,
CMLPT2, E-134*	E-110, E-111, E-112, E-113,
CNTXSW, E-127*	E-114, E-117, E-120
COMERR, E-59	EMTUSR, E-105*, E-106
COMPLT, E-100*, E-134*, E-148,	ENQUSR, E-132*
E-150	ENTER, E-37*
COMXIT, E-30, E-33, E-34, E-35,	ENTRPG, E-7, E-69, E-117
E-37, E-38, E-40, E-41,	ENTRY, E-59*
E-43, E-54, E-59*	EOLTST, E-92*
CONSOL, E-60*	ERRCOM, E-129*
CSI, E-46*	EXINT, E-122*, E-138
CTRL.B, E-141*	EXSWAP, E-122, E-123*
CTRL.C, E-141*	EXTEND, E-38, E-39*
CTRL.F, E-141*	EXUSER, E-122*, E-123, E-124,
CTRL.O, E-142*	E-138
CTRL.Q, E-142*	
CTRL.S, E-142*	F\$ETCH, E-66*, E-105*
CTRL.U, E-142*	FATAL, E-29*
CTRLC, E-90*	FILE, E-14*
CTRLO, E-89*	FOON, E-144*, E-146
CTRLQ, E-89*	FRUN, E-24*

G\$TIM, E-73*, E-114*
G\$TJB, E-73*, E-111*
GESTAT, E-40*
GET, E-4, E-8*
GETBLK, E-23*
GETFD, E-53*
GETNAM, E-53*
GT, E-25*

H\$ERR, E-74*
H\$RSET, E-66*, E-116*
HDRSET, E-35*

IGET, E-149*
INCR1, E-59
INIT, E-4*
IORSET, E-116*

KLOSE, E-42*

L\$OCK, E-75*
L\$OOK, E-66*, E-105*
LK4DEV, E-61*
LKER1, E-34*, E-36
LKINT, E-138*
LNFILE, E-33, E-34*
LOAD, E-21*
LOOKUP, E-33*

M\$RKT, E-112*
M\$WAIT, E-111*
MARKTM, E-113*
MEXIT, E-4*, E-79
MEXIT2, E-4*, E-78
MONOUT, E-49, E-75*, E-120*

NFREAD, E-70*, E-71
NFWRITE, E-71*, E-109
NOSWIT, E-48*
NXBLK, E-56*
NXTFIL, E-47*, E-48

OBUMP, E-145*, E-146
OCKCR, E-145*, E-146
OFILL, E-144*
OPRINT, E-5*, E-24
OPUT, E-96*
OUSTUF, E-52*
OVLINK, E-12*
OVREAD, E-12*

P\$RINT, E-76*, E-105*
P\$ROTE, E-110*

P\$SURGE, E-74*, E-106
PHETCH, E-40*
PUTBLK, E-23*

Q\$SET, E-66*, E-105*
QFULL, E-133*
QMANGR, E-98*, E-133*
QRESET, E-117*
QSET, E-43*
QUIESCE, E-117*

R\$CTLO, E-105*
R\$CVD, E-108*
R\$EAD, E-70*, E-74, E-108
R\$NAME, E-66*, E-105*
R\$OPEN, E-69*, E-111*
R\$SUME, E-112*
R, E-4, E-7*
RDOVLY, E-7, E-69*, E-117*
REENTR, E-4, E-9*
RENAME, E-33*
RENT, E-37*, E-39
RESUM, E-112*
RESUME, E-22*
REVERT, E-117*
RIDUSR, E-132*
RSTSR, E-35*
RTORG, E-29*
RUB, E-91*, E-142
RUBCM2, E-92*
RUBCOM, E-92*
RUN, E-4, E-9*
RWXT, E-108*, E-109
RWXTE0, E-108*

S\$AVST, E-68*, E-111*
S\$DAT, E-109*
S\$ERR, E-74*
S\$ETOP, E-77*, E-118*
S\$FPA, E-73*
S\$FPP, E-112*
S\$PFUN, E-74*, E-108*
S\$RSET, E-66*
S\$SPND, E-112*
S\$SRET, E-117*
S\$SWAP, E-112*
S\$TRAP, E-113*
SAVE, E-19*
SAVEVC, E-6, E-14*
SEGRW, E-58*
SEGRW1, E-58*
SEGRW2, E-58*
SOFIRST, E-35*
SPDEL, E-34*, E-36
SPECIAL, E-47, E-48*
SPENTR, E-37, E-38*
SPESHL, E-59
SPLOOK, E-33, E-34*

SPREAD, E-108*
STARTK, E-4, E-9*
STRE, E-9*
STR TIN, E-46*, E-48
SUSPEND, E-22*
SWAPME, E-126*
SWITCH, E-47, E-48*
SYNERR, E-47
SYSK, E-16*

T\$LOCK, E-120*
T\$RPST, E-68*
T\$TIN, E-79*, E-107*
T\$TOUT, E-80*, E-107*
T\$WAIT, E-112*
TCHKSP, E-93, E-94*
TIME, E-18*
TIMER, E-122, E-138*
TOOBIG, E-64, E-65*
TPRNT7, E-93*, E-94
TRAP10, E-129*
TRAP4, E-129*
TSWCNT, E-109*
TTCMPL, E-148*
TTHIN, E-148*
TTHOCM, E-146*
TTHOUT, E-145, E-146*
TTIBUM, E-91*

TTIDSP, E-140*, E-142
TTIEXZ, E-88*, E-91
TTIINT, E-86*, E-140*
TTINC3, E-87*, E-88, E-89, E-90,
E-140*, E-141
TTODON, E-93, E-94*
TTOENB, E-141*, E-143
TTOINT, E-93*, E-144*
TTOPT2, E-143*
TTOPT3, E-143*
TTOPT4, E-141, E-143*
TTOPUT, E-95*
TTORUB, E-95*
TTPOXT, E-93, E-94*

U\$NLOK, E-75*, E-120*
UABORT, E-115, E-124*, E-129,
E-132
UNBLOK, E-126*
UNLOAD, E-22*
USR, E-32*
USRBUF, E-29*
USRCOM, E-54*
USRNF, E-55*
USWAPO, E-115, E-133*

W\$AIT, E-72*, E-111
W\$RITE, E-71*, E-109

INDEX

- Abbreviations, 1-2
- Abort,
 - code, 5-8
 - entry point, 6-4
- Absolute section, 3-21
- Adding a handler to system, 5-11
- Allocating space for F/G, 2-4
- ANSI MT labels under RT-11, 3-12
- ASSIGN command, 2-15

- Backslash character (\) (BATCH), 7-1
- Bad tape errors, 3-13
- BATCH, 7-1
 - compiler, 7-4, 7-6
 - CTL format, 7-1
 - CTT temporary files, 7-22
 - directives, 7-1, 7-2
 - example, 7-11
 - job termination, 7-6
 - language, 7-1
 - run-time handler, 7-2
- BATSW1 switches, 7-3
- B/G (definition), 1-3
- Bitmap byte table, 2-21
- BLIMIT table, 2-4
- Block,
 - ENDGSD, 3-19
 - GSD, 3-19
- Blocking a job, 6-3
- Blocks, data, 3-19
- Bootable magtape, 3-12.1
- Bootstrap, 2-24
 - operation, 4-3
 - system, 5-15
- Building a new system, 5-16

- \$CALL command (BATCH), 7-12
- Carriage width, 2-26
- Cassette,
 - file header, 3-15
 - file structure, 3-14
- Channel number, 3-4
- Channel status word, 1-2
- Checksum byte, 3-18
- Clock interrupt service,
 - flowcharts, E-83, E-137
- Command field, 3-24
- Command string interpreter (CSI),
 - see CSI
- Compatibility between RT-11
 - versions, C-1
- Compiler (BATCH), 7-4
 - construction, 7-6
 - temporary files, 7-22
- Completion queue elements, 5-5, 5-6

- Console terminal,
 - interrupt service (flowcharts), E-85, E-139
 - substitution, 2-23
- Constant field, 3-24
- Context switch, 6-2, 6-3
- Converting user-written handlers,
 - requirements, 5-23
- \$COPY command (BATCH), 7-11
- <CR> (definition), 1-3
- \$CREATE command (BATCH), 7-11
- CR11 device handler, A-35
- CSECTs, 3-21
- CSI (Command String Interpreter),
 - definition, 1-2
 - flowcharts, E-45
 - requests, 6-5
 - subroutines (flowcharts), E-51
- \$CSW (definition), 1-2
- CT file header format, 3-16
- CTL file (BATCH), 7-1, 7-22
- CTT file (BATCH), 7-22
- Current queue element, 5-3

- Data base description (BATCH), 7-7, 7-10
- Data blocks, 3-19
- Data transfer operations, 5-19
 - by device handlers, 5-9
- Date, 3-5
- Definitions, 1-2
- \$DELETE command (BATCH), 7-11
- Determining user program memory, 2-2
- Device directory, 2-15, 3-1
- Device Directory Size table,
 - \$DVSIZ, 2-15
- Device driver, 5-16
- Device Handler Block table,
 - \$DVREC, 2-14
- Device handlers, 5-1, 5-14
 - CR11, A-35
 - format, 5-8
 - LP/LS11, A-28
 - RC11/RS64, A-2
 - TC11, A-47
- Device handlers changed by SET
 - command, 5-21
- Device identifier, 2-13
- DEVICE macro, 2-16
- Device Ownership table, \$OWNER, 2-16
- Device Status table, \$STAT, 2-13
- Devices with special directories, 5-19
- Differences between V1 and V2/
 - V2B EMTs, C-1

- Directive, 7-1
- \$DIRECTORY command (BATCH), 7-11
- Directory entries, 3-2
 - format, 3-3
- Directory header,
 - format, 3-1
 - words, 3-2
- Directory of devices, 2-15
- Directory operations, 5-19
- Directory segment, 3-1, 3-6
 - extensions, 3-8
 - format, 3-1
- Double tape mark, 3-11
- \$DVREC (Device Handler Block table), 2-14
- \$DVSIZ (Device Directory Size table), 2-15
- Dynamic memory allocation, 2-7
- Empty file, 3-3
- EMT processors (flowcharts), E-67, E-103
- ENDGSD block, 3-19, 3-22
- ENDMOD block, 3-28
- Entry conditions, device handler, 5-9
- \$ENTRY (Handler Entry Point table), 2-14
- Entry point,
 - index, E-151
 - table format (library), 3-29
- \$EOJ (BATCH)
 - command, 7-13
 - statement, 7-6
- Error checking (BATCH), 7-12
- Errors, bad tape, 3-13
 - in special device handlers, 5-20
- Events, scheduler, 6-4
- Example,
 - BATCH, 7-11
 - program linked to produce REL file, 3-35
- Extra words, 3-5
- F/B (definition), 1-3
- F/B monitor,
 - description, 6-1
 - flowcharts, E-1
- F/G (definition), 1-3
- File formats, RT-11, 3-1
 - formatted binary (.LDA), 3-30
 - object (.OBJ), 3-16
 - relocatable (.REL), 3-32
 - save image (.SAV), 3-31
- File,
 - header, cassette, 3-15
 - length, 3-4
 - names and extensions, 3-4
- File structure, 3-1
 - cassette, 3-14
 - magtape, 3-11
- File types, 3-3
 - tentative, 3-3
 - empty, 3-3
 - permanent, 3-4
- Files, size and number, 3-7
- Filling directory segments, 3-7
- First end-of-file label, 3-12
- First file header label, 3-12
- Fixed offsets, 2-10-2-12
- FLIMIT table, 2-4
- Flowcharts, S/J and F/B monitor,
 - CSI (Command String Interpreter), E-45
 - KMON (Keyboard Monitor), E-3
 - RMON (Resident Monitor), F/B, E-101
 - RMON (Resident Monitor), S/J, E-63
 - USR (User Service Routines), E-27
- Foreground job area, 2-3, 2-4
- Foreground spooler example, D-1
- Foreground terminal handler, B-1
- Format, CT file header, 3-16
- Formatted binary block, 3-18
- Formatted binary format, 3-30, 3-31
- Function codes, 5-19
 - negative, 5-20
 - positive, 5-19
- Global,
 - additive displaced relocation, 3-26
 - additive relocation, 3-26
 - displaced relocation, 3-26
 - references, 3-21
 - relocation, 3-25
 - symbol directory (object module), 3-20
 - symbols, 3-21
- GOTO command (BATCH), 7-12
- GSD
 - block, 3-19
 - item, 3-20
 - structure, 3-20
- Handler Entry Point table,
 - \$ENTRY, 2-14
- Handler,
 - installation, 5-11
 - names, 5-12
 - queue header, 5-3
- Handler Size table, \$HSIZE, 2-14
- Handlers, 2-14
 - interrupt, 6-1
 - for special devices, 5-19
 - user-written, 5-23

- Hardware mode, 3-15
- Header block, 3-1
 - format, library, 3-29
 - words, device handler, 5-8
- High limit pointer, 2-5
- \$HSIZE (Handler Size table), 2-14
- HSIZE macro, 2-17

- IF conditional branch (BATCH), 7-12
- Impure area, 2-3, 2-18, 2-19, 2-21
- Initiating a BATCH job, 7-4
- .INTEN request, 6-1
- Internal relocation, 3-24
- Internal displaced relocation, 3-25
- Interrupt,
 - code restrictions, 5-10
 - handler, 5-9, 6-1
 - vector tables, 5-13
 - vectors, 2-23
- I/O queue elements, 5-1, 5-2
- I/O queuing system, 5-1
- I/O routines, E-97
- I/O termination, 6-5
- I/O transfers, 5-1
- ISD (Internal Symbol Directory), 3-28

- Job,
 - arbitration, error processing, E-121
 - blocking, 6-3
 - boundaries (F/B), 2-4
 - ID area, 2-21
 - initiation (BATCH), 7-4
 - number, 3-4, 6-3
 - priority, 5-5, 6-3
 - scheduling, 6-3
 - status, 2-19
 - termination (BATCH), 7-6
- Job Status Word, 1-3
- \$JOB command (BATCH), 7-11
- JSW (definition), 1-3

- KMON (definition), 1-2
- KMON (Keyboard Monitor), 1-2
 - flowcharts, E-3
 - overlays, 4-3, E-17
 - subroutines, E-11

- Label,
 - first end-of-file, 3-12
 - first file header, 3-12
 - volume header, 3-12

- Labels,
 - BATCH, 7-12
 - magtape, 3-12
- Language processor, 3-16
 - object modules, 3-19
- Last queue element, 5-3
- .LDA, formatted binary format, 3-30
- <LF> (definition), 1-3
- Library,
 - end trailer, 3-30
 - file format, 3-28
 - header, 3-28
 - object format, 3-28
- Limit tables, 2-4
 - BLIMIT, 2-4
 - FLIMIT, 2-4
- Linking BATCH, 7-2
- Location counter,
 - commands, 3-23
 - definition, 3-27
 - modification, 3-27
- Low memory bitmap (LOWMAP), 2-21
- LP/LS11 device handler, A-28

- Macro,
 - DEVICE, 2-16
 - HSIZE, 2-17
- \$MACRO command (BATCH), 7-11
- Magtape
 - bootable, 3-12.1
 - compatibility, 3-13
 - file structure, 3-11
- Making SET TTY options permanent, 2-25
- Mode, hardware, 3-15
 - software, 3-15
- Memory, 1-1, 2-1, 2-2
 - allocation, 2-7, 2-8, 2-9
 - areas, 2-9
- Mnemonic names, 1-3
- Monitor,
 - description (F/B), 6-1
 - device tables, 2-13
 - fixed offsets, 2-9
 - memory allocation, 2-7
 - memory layout, 2-1
 - operation, 1-1
- MONITR.SYS contents, 4-2
- .MRKT request, 5-7

- Name field, 3-24
- Names,
 - and extensions, file, 3-4
 - handler, 5-12
 - mnemonic, 1-3
- Negative relocation, 3-32

- Object format (.OBJ), 3-16
- Object module processing, 3-17
- Offsets, fixed, 2-10-2-12
- Operations,
 - data transfer, 5-19
 - directory, 5-19
 - special, 5-19
- Output (BATCH), 7-4
- Overlay programs, 3-32
- Overlay segment relocation block, 3-44
- Overlays, 3-42
- Overview, 1-1
- \$OWNER (Device Ownership table), 2-16
- Ownership code, 2-16

- Paper tape format, 3-30
- Patch procedures, TTY options, 2-26
- Permanent file, 3-4
- Permanent name table (\$PNAME), 2-13
- \$PNAME (Permanent Name table), 2-13
- Positive relocation, 3-32
- \$PRINT command (BATCH), 7-11
- Priority level of handlers, 6-1
- Program sections, 3-21
- Programmed requests, 5-20
 - to special devices, 5-20
 - V1, C-1
- Public device, 2-16

- Queue element for a special handler, 5-20
- Queue elements,
 - completion, 5-5
 - timer, 5-7
- Queue header, handler, 5-3
- Queue manager, 5-5
 - flowcharts, E-131
- Queue structures, 5-5
- Queued I/O, 5-1

- .RAD50 conversions, 5-12
- RC11/RS64,
 - bootstrap, A-9
 - device handler, A-2
- REL file,
 - without overlays, 3-33
 - with overlays, 3-42, 3-43
- Relocatable format (.REL), 3-32
- Relocation, 3-32
 - codes, 3-24, 3-25, 3-26
 - global, 3-25, 3-32
 - global additive, 3-26, 3-33
 - global additive displaced, 3-26, 3-33
 - global displaced, 3-26, 3-33
- Relocation, (cont.)
 - information, 3-34
 - internal, 3-24, 3-32
 - internal displaced, 3-25, 3-33
- Resident device handlers (flowcharts), E-147
- Resident monitor, 1-2
- RLD,
 - block, 3-22
 - commands, 3-23
 - format, 3-24
- RMON (definition), 1-2
- RMON (Resident Monitor) flowcharts,
 - for F/B Monitor, E-101
 - for S/J Monitor, E-63
- Root relocation, 3-44
- RT-11 file formats, see file formats, RT-11

- Sample handler listings, A-1
- Save image format (.SAV), 3-31
- Scheduling, job, 6-3
- Segments of directories, 3-8
- Sentinel file, 3-15
- SET command, 5-21
- SET command options, 5-22
 - conventions for adding, 5-22
- Set program limits, 3-27
- S/J (definition), 1-3
- S/J Monitor,
 - flowcharts, E-1
 - restrictions, 2-22
- Software mode, 3-15
- Special,
 - devices, 5-19
 - operations, 5-19
- Square brackets, [and], 1-3
- Stack, 6-2
 - information, 6-2
 - location, 2-3
 - pointer, 6-2
- \$STAT (Device Status table), 2-13
- Status,
 - buffer, 2-23
 - register, 2-23
 - word, 2-13, 3-3
- Symbolic names, 1-3
- .SYNCH element, 5-7
- .SYNCH request, 5-6, 6-3, 6-4
- SYSLOW,
 - examples (background), 2-6
 - pointer, 2-4
- System,
 - bootstrap, 5-15
 - communication locations, 3-34
 - components, 4-1
 - configuration word, 2-11
 - date word, 3-5
 - size, 4-5, 4-6

System device handler,
 requirements, 5-14
 writing a, 5-14
 System device structure, 4-1

Tables, monitor device, 2-13
 TC11 device handler, A-47
 Temporary files, BATCH compiler,
 7-22
 Tentative file, 3-3
 Terminating a BATCH job, 7-6
 Terminology, 1-2
 Timer queue element, 5-7
 Trailer, library file, 3-30
 Transfer address GSD item, 3-21
 TTCNFG option bits, 2-27
 TTY options, 2-25
 .TWAIT request, 5-7
 TXT block format, 3-22

\$UNAM1, \$UNAM2 (User Name tables),
 2-15
 Underlining, 1-3
 User Name tables, \$UNAM1, \$UNAM2,
 2-15
 User service routines, 1-2
 User-written handlers, 5-23
 Using auxiliary terminals as
 console terminal, 2-23
 USR (User Service Routines),
 contention, 6-5
 definition, 1-2
 flowcharts, E-27
 ownership, 6-5
 permanently resident, 2-7
 queuing mechanism, 6-5
 swapping, 2-2, 2-6

Variables, BATCH, 7-13
 Vector protection, 2-21, 2-22
 Version 1 EMT summary, C-1
 Volume-header label, 3-12

Writing a system device handler,
 5-14

READER'S COMMENTS

NOTE: This form is for document comments only. Problems with software should be reported on a Software Problem Report (SPR) form.

Did you find errors in this manual? If so, specify by page.

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

Is there sufficient documentation on associated system programs required for use of the software described in this manual? If not, what material is missing and where should it be placed?

Please indicate the type of user/reader that you most nearly represent.

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Non-programmer interested in computer concepts and capabilities

Name _____ Date _____

Organization _____

Street _____

City _____ State _____ Zip Code _____

or
Country

Please cut along this line.

Fold Here

Do Not Tear - Fold Here and Staple

FIRST CLASS
PERMIT NO. 33
MAYNARD, MASS.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

Postage will be paid by:

digital

Software Communications
P. O. Box F
Maynard, Massachusetts 01754

