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

digital equipment corporation · maynard. massachusetts

First Printing, November 1973
Revised: June 1975
January 1976

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.

CONTENTS

FIGURES

TABLES

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:

TERM	MEANING
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>.

TERM	MEANING
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.

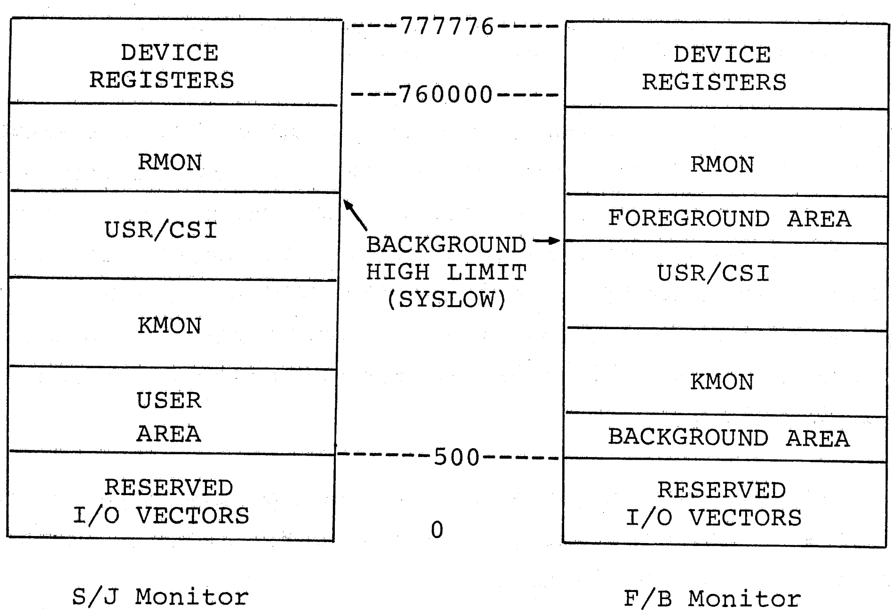


Figure 2-1
Monitor Memory Layout

The memory area diagrammed is arranged as follows:

Memory Area	Use
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
> ```

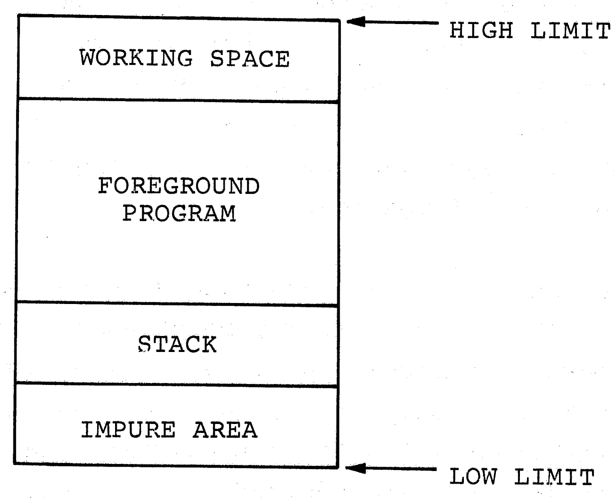


Figure 2-2
Foreground Job Area Layout

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.

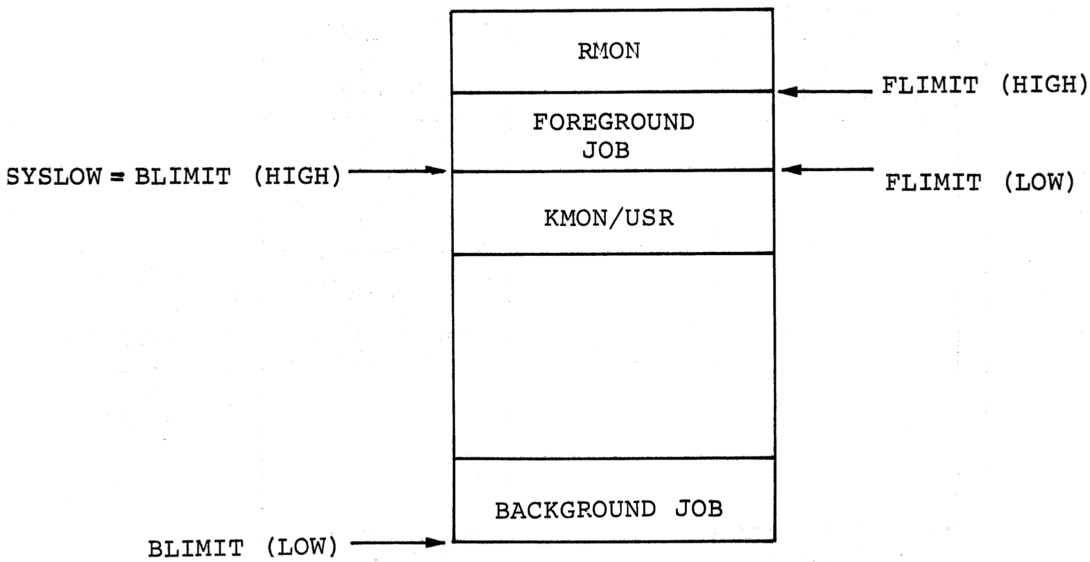


Figure 2-3
Job Limits

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.

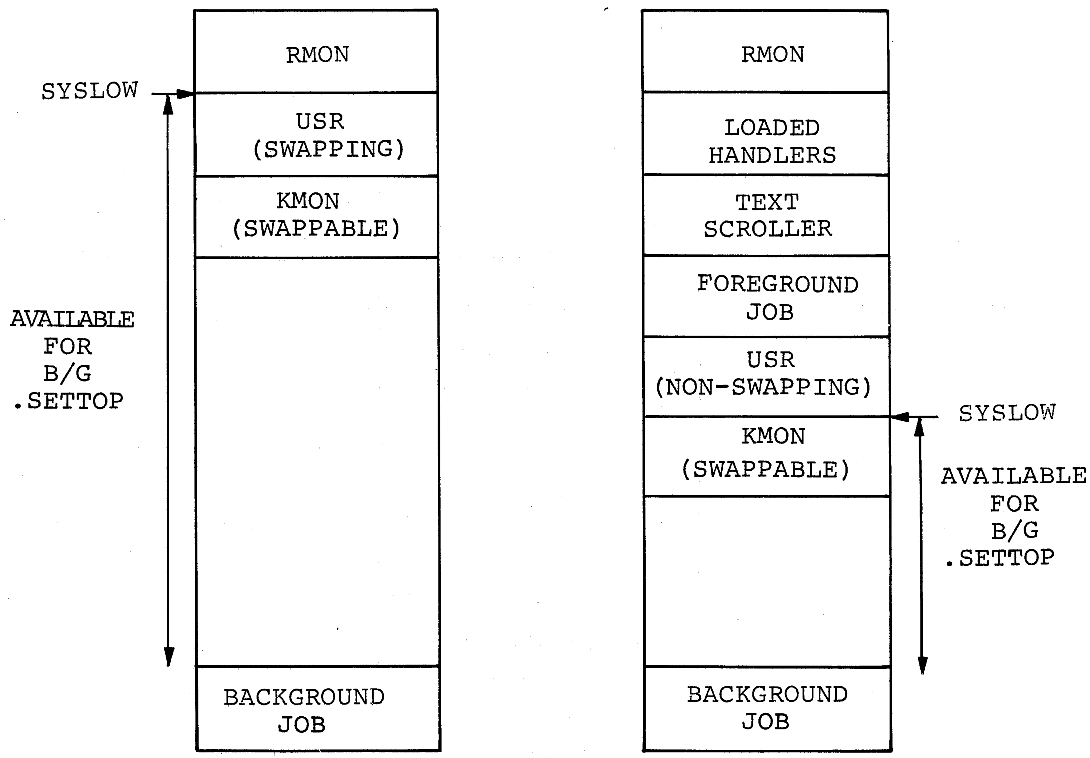


Figure 2-4
Background SYSLOW Examples

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

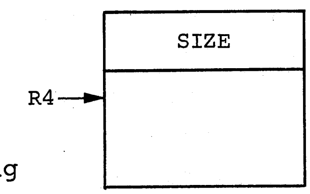
RT-11 uses a dynamic memory allocation scheme to provide memory space for LOADed handlers, foreground jobs (F/B Monitor only) and the display text scroller. Memory is allocated in the region above the KMON/USR and below RMON. If there is insufficient memory in this region (initially, after the system 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

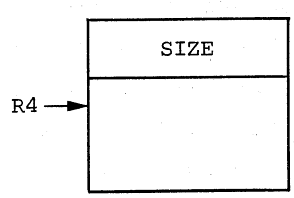
Call sequence:
RØ = SIZE
JSR PC,GETBLK



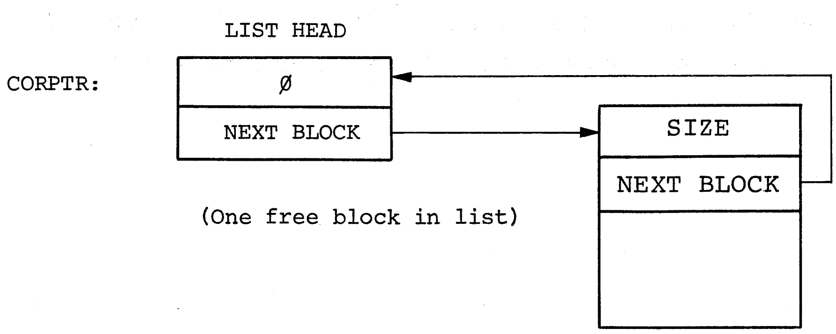
(returns with R4 pointing
to the allocated block)

b) Releasing a memory block

Call sequence:
R4 → BLOCK
JSR PC,PUTBLK



c) Free memory list



(One free block in 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: **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) Any bits not currently assigned are reserved by DIGITAL for future use and should not be used arbitrarily by user programs.
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 } (F/B)	2	Pointer to the impure area for the current executing job.
322	210	JOBNUM	2	Executing job's number (0 = B/G, 2 = F/G).
320 322	208 210	$TIME } (S/J)	4	Two words of time of day in the S/J Monitor.
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 14_{10} 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 256_{10}-word blocks contained on the device. The current devices and their entries are:

Device	Number of 256-Word Blocks	Device	Number of 256-Word Blocks
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 2000_8 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.IPUT	2	Input "PUT" pointer for interrupts.
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 20_{10} 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,RØ
BISB #↑B1111ØØØØ,342(RØ)
```

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/NOCRLF	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
*BASE;ØR<CR>                            relocation registers; open
*Ø,2141Ø\        11Ø        2Ø4<CR>     with backslash]
*E
.
```

In this example, the width is changed from 72_{10} to 132_{10} (204_8).

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 <u>RT-11 System Release Notes</u> 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;
*BASE;ØR<CR>                                set relocation register;
*Ø,TTCNFG/      212      5<CR>              change TTCNFG;
*Ø,LISTFB/      3316     1Ø3316<CR>         set bit 15 in LISTFB.]
*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:

5 header words
file entries . . .

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 $1<=N<=31_{10}$.
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:

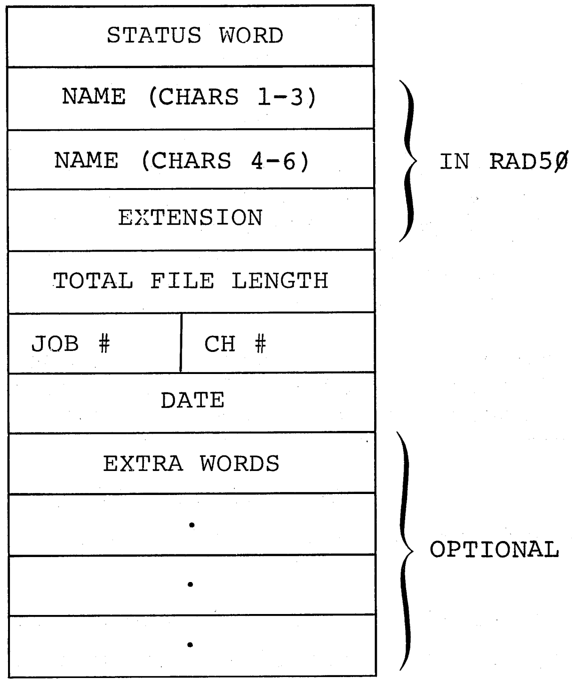


Figure 3-1
Directory Entry Format

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
UNUSED	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:

	Word	Meaning
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 AN EMPTY ENTRY
	0	
	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:

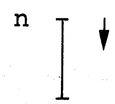
$$(M+1)\left[\frac{507}{2(7+N)}\right]$$

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

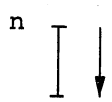
The theoretical total can be realized by compressing the device (using the PIP /S operation) when the directory fills up. PIP packs the directory segments as well as the physical device.

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:

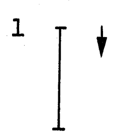


This represents a directory segment with some number of directory entries. n is the segment number.

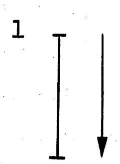


This represents a segment which is full, i.e., no more entries will fit in the segment.

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,

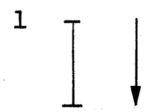
3. the directory segment links are set, and

4. the file is entered in the newly created segment.

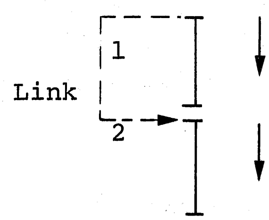
NOTE

> 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 OVFLO?

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

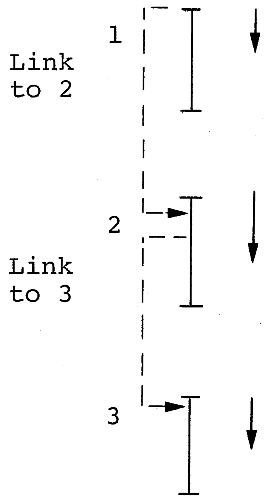


Before extension

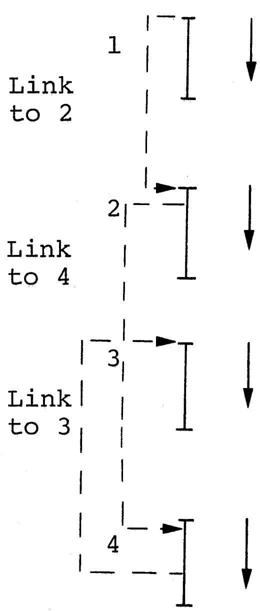


After extension; half the entries are in the new segment, half in the old; segment 1 is linked to segment 2.

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:



In this case, segment 2 overflows into segment 4 and the links are used to link logical pieces rather than physical pieces.

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 *EOF*, not *HDR*, 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 HDR1*---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 HDR1 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.

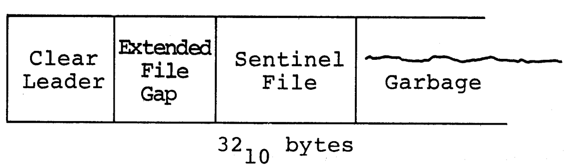
3.3.1.3 Recovering From Bad Tape Errors - When a bad tape error occurs on magtape, the magtape handler will retry the desired function, and, if the error persists, will attempt to save the tape's file structure. It does this on writes, for example, by retrying the write 10 times, using the write with 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.

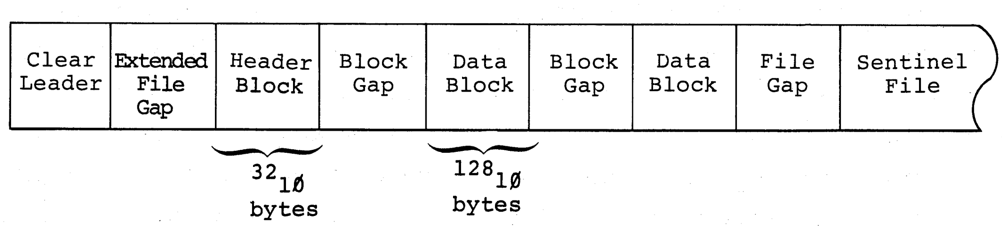
If a bad tape error occurs while writing the end of file label (EOF1) during CLOSE, the handler writes a triple tape mark to signify end of file and logical end of tape. Additional files can be added to the tape only if the last complete file can be extended (or overwritten by a file of different length) 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:

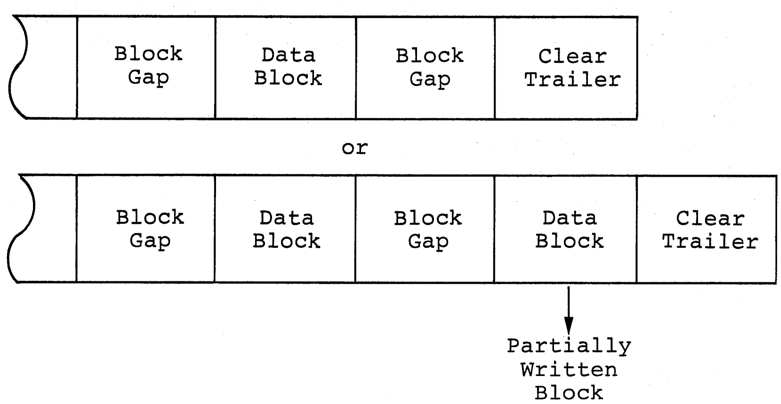


while a cassette with a file on it appears as:



Files normally have data written in 128_{10}-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 32_{10}-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.

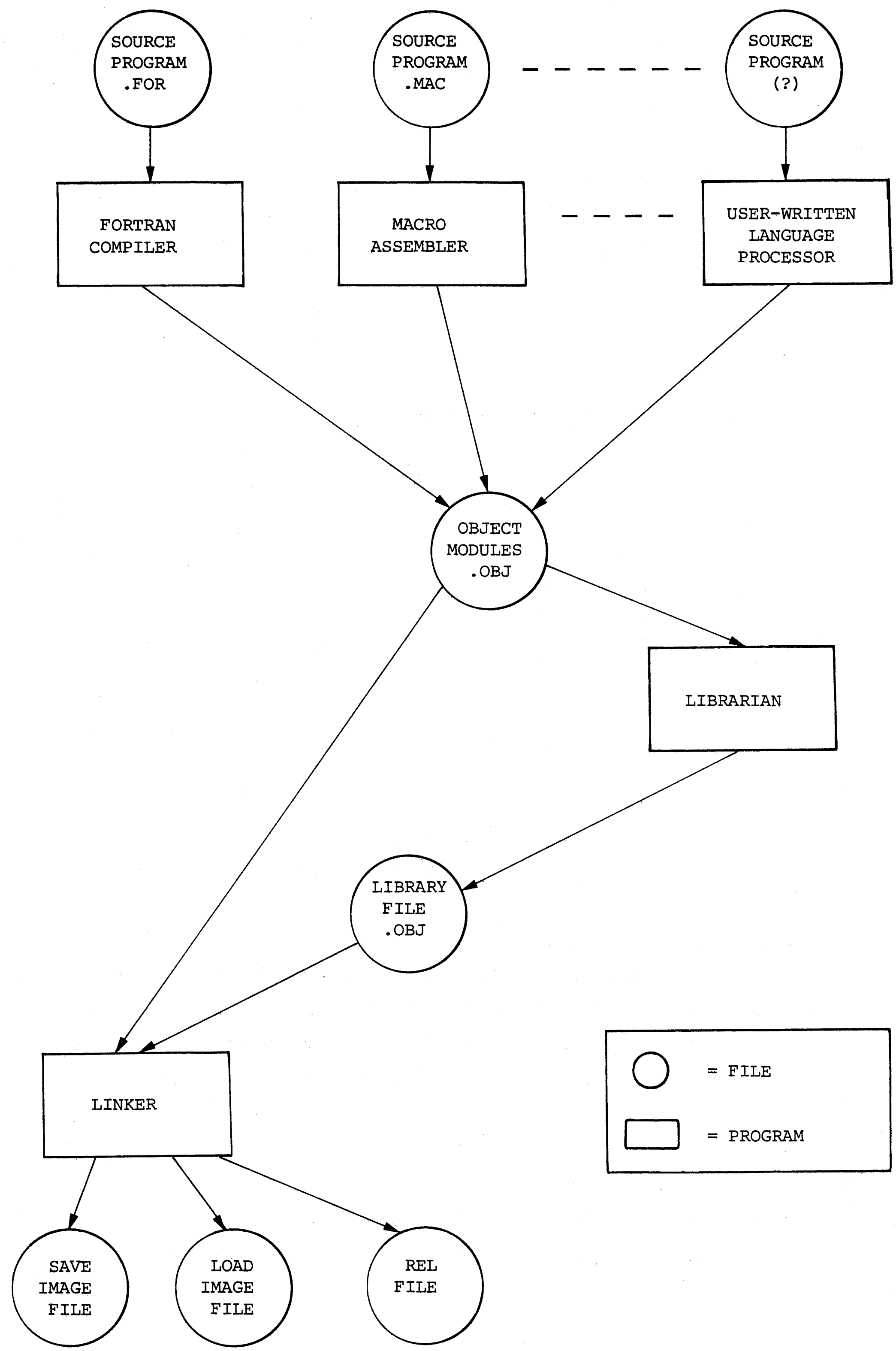


Figure 3-3
Object Module Processing

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.

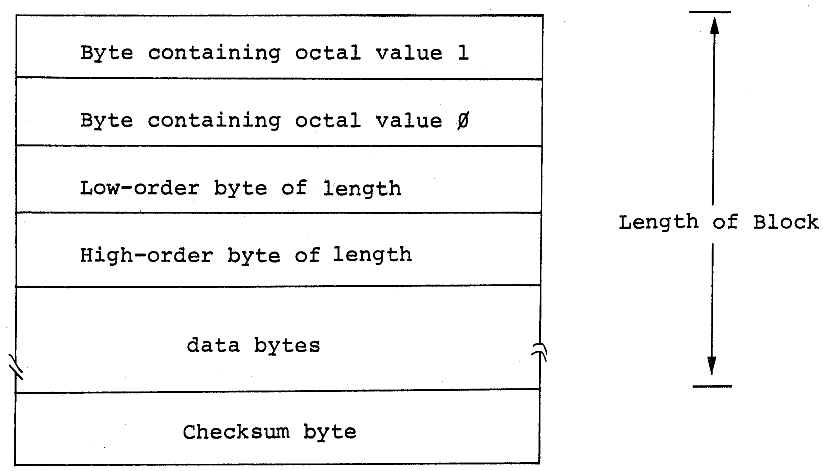


Figure 3-4
Formatted Binary Block

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:

Identification Code	Type of Block	Function	
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
1Ø	Librarian End Block	signals the end of the library file	Library File Only

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:

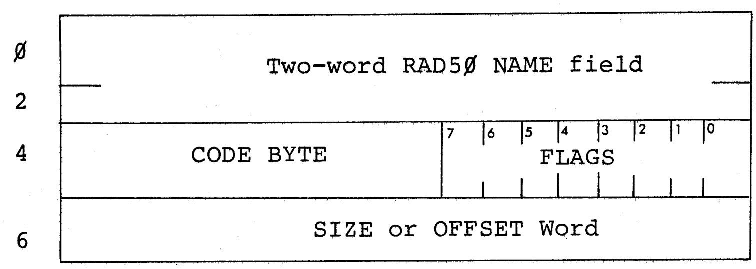


Figure 3-5
GSD Structure

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␣␣␣␣␣␣.

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:

Ø	Identification Word (TXT=3)
2	Load Address of following data
4	Absolute Load Data (up to 38 bytes)

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):

15 — 8	7	6 — Ø
RELATIVE REFERENCE	B	COMMAND
Two-word RAD5Ø NAME Field		
CONSTANT		

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	Ø/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:

a)	ABS=17755Ø TST ABS	both addresses cause internal displaced relocation to occur
b)	CLR 17755Ø	

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 10_8) - 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.

	1Ø
Constant	

Examples:

a) .=100 ;IF WE ARE IN THE ASECT

b) .=.-2Ø ;IF WE ARE IN A RELOCATABLE SECTION

9. Set Program Limits (code 11_8) - 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 14_8, 15_8, and 16_8 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.

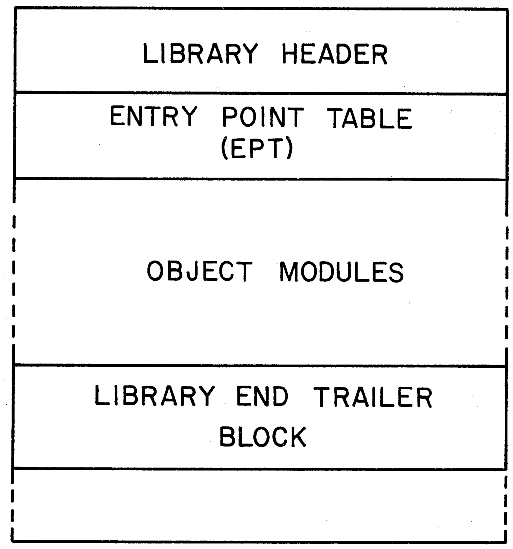


Figure 3-8
Library File Format

Diagrams of each component in the library file structure are included here, but Chapter 7 of the <u>RT-11 System Reference Manual</u> 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.

Word	Description
1	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.

Offset	Field		Notes
0	SYMBOL CHARS 1-3 (RAD5Ø)		
2	SYMBOL CHARS 4-6 (RAD5Ø)		
4	(bit 15)	ADDRESS OF BLOCK	BIT 15=1-MODULE NAME Ø-CSECT OR ENTRY POINT NAME
6	# OF CSECTS IN OBJECT MODULE	RELATIVE BYTE IN BLOCK	RELATIVE BYTE MAXIMUM=777_8 CSECTS MAXIMUM =177_8

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.

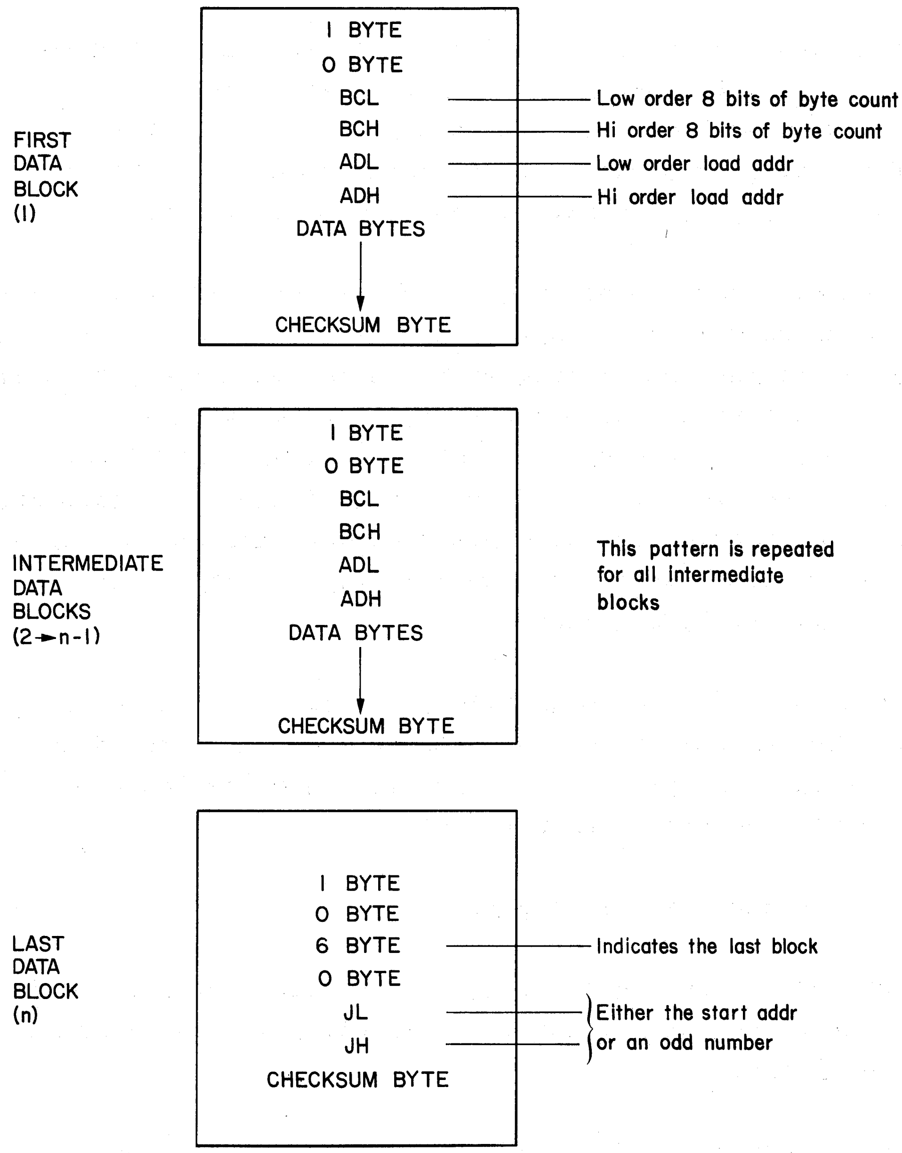


Figure 3-12
Formatted Binary Format

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 background. 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) (absolute) - always positive relocation

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

3. Internal Displaced Relocation (MOV 54,R) - always negative relocation

4. Global Displaced Relocation (MOV G,R) - negative relocation only where the global is defined as absolute elsewhere

5. Global Additive Relocation (.WORD G + C) - same as 2 above

6. Global Additive Displaced (MOV G + C,R) - same as 4 above

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:

Block Ø

	Program Text	Relocation Information

Figure 3-13
REL File Without Overlays

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

Offset from Beginning of Block 0	Contents
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'
6Ø	Relative block number of relocation information

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

Offset from Beginning of Block 0	Contents
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:

15	14 Ø
	RELATIVE WORD OFFSET
	RELATIVE WORD OFFSET
	RELATIVE WORD OFFSET
	RELATIVE WORD OFFSET
	RELATIVE WORD OFFSET
	.
	.
	.
-2	

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  FTEST
.MCALL  ..V2..,.REGDEF,.LOOKUP,.READW,.QSET,.PRINT,.EXIT
..V2..
.REGDEF
ST:     .QSET   #QLIST,#7
        .LOOKUP #AREA,#0,#PTR
        BCC     1$
        .PRINT  #LKFAIL
        .EXIT
1$:     .READW  #AREA,#0,#BUFF,#256.,#0
        BCC     2$
        .PRINT  #RDFAIL
        .EXIT
2$:     .PRINT  #OK
        .EXIT

QLIST:  .BLKW   7*7
AREA:   .BLKW   20.
PTR:    .RAD50  /PR FILE12/
BUFF:   .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                                                       .TITLE  FTEST
  3                                             .MCALL  ..V2..,.REGDEF,.LOOKUP,.READW,.QSET,.PRINT,.EXIT
  4 000000                                      ..V2..
  5 000000                                      .REGDEF
  6 000000                                      ST:     .QSET   #QLIST,#7
    000000  112700  000007                      .ITF NB <#7>,   MOVB    #7,%0
    000004  012746  000144'                                             MOV     #QLIST,-(6.)
    000010  104353                                                      EMT     ^O353
  7 000012                                              .LOOKUP #AREA,#0,#PTR
    000012  012700  000306'                                             MOV     #AREA,%0
    000016  112760  000001  000001                                      MOVB    #1,1(0)
    000024  105010                                                      CLRB    (0)
    000026  012760  000356' 000002  .ITF NB <#PTR>,                     MOV     #PTR,2.(0)
    000034  005060  000004                                              CLR     4.(0)
    000040  104375                                                      EMT     ^O375
  8 000042  103004                                      BCC     1$
  9 000044                                              .PRINT  #LKFAIL
    000044  012700  001364'         .ITF NB <#LKFAIL>,                          MOV     #LKFAIL,%0
    000050  104351                                                      EMT     ^O351
 10 000052                                              .EXIT
    000052  104350                                                      EMT     ^O350
 11 000054                          1$:                 .READW  #AREA,#0,#BUFF,#256.,#0
    000054  012700  000306'                                             MOV     #AREA,%0
    000060  112760  000010  000001                                      MOVB    #8.,1(0)
    000066  105010                                                      CLRB    (0)
    000070  012760  000000  000002  .ITF NB <#0>,                       MOV     #0,2.(0)
    000076  012760  000364' 000004  .ITF NB <#BUFF>,                            MOV     #BUFF,4.(0)
    000104  012760  000400  000006  .ITF NB <#256.>,                            MOV     #256.,6.(0)
    000112  012760  000000  000010  .ITF NB <#0>,                       MOV     #0,8.(0)
    000120  104375                  .ITF NB <X>,                        EMT     ^O375
 12 000122  103004                                      BCC     2$
 13 000124                                              .PRINT  #RDFAIL
    000124  012700  001402'         .ITF NB <#RDFAIL>,                          MOV     #RDFAIL,%0
    000130  104351                                                      EMT     ^O351
 14 000132                                              .EXIT
    000132  104350                                                      EMT     ^O350
 15 000134                          2$:                 .PRINT  #OK
    000134  012700  001417'         .ITF NB <#OK>,                      MOV     #OK,%0
    000140  104351                                                      EMT     ^O351
```

```
16 000142                                       .EXIT
   000142  104350                                               EMT     ^O350
17
18 000144                              QLTST:   .BLKW   7*7
19 000306                              AREA:    .BLKW   20.
20 000356  063320   023364   022070   PTR:     .RAD50  /PR FILE12/
26 001364     114      117      117   LKFATL:  .ASCIZ  /LOOKUP FAILED/
   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         BUFF    000364R         LKFATL  001364R         OK       001417R        PC      =%000007
PTR     000356R         QLTST   000144R         RDFATL  001402R         R0      =%000000        R1      =%000001
R2     =%000002         R3     =%000003         R4     =%000004         R5      =%000005        SP      =%000006
ST      000000R         ...V2 = 000001
. ABS.  000000     000
        001776     001
ERRORS DETECTED: 0
FREE CORE: 15895. WORDS

,LP:/N:TTM/L:MEB=FTEST
```

```
BLOCK NUMBER 0000
000/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
020/ 000000 000000 000000 000000 000000 000000 000000 000000 *................*
040/ 000000 000000 000000 000000 001776 001776 000000 070524 *........=.=...TQ*
060/ 000003 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/ 000300 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 *................*
```

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 112760 *@...F.D.K.@.F.P.*
020/ 000001 000001 105010 012760 000356 000002 005060 000004 *......P.N...0...*
040/ 104375 103004 012700 001364 104351 104350 012700 000306 *]...@.T.I.H.@.F.*
060/ 112760 000010 000001 105010 012760 000000 000002 012760 *P.......P.....P.*
100/ 000364 000004 012760 000400 000006 012760 000000 000010 *T...P.....P.....*
120/ 104375 103004 012700 001402 104351 104350 012700 001417 *]...@...I.H.@...*
140/ 104351 104350 000000 000000 000000 000000 000000 000000 *I.H.............*
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 063320 *..............PF*
360/ 023364 022070 000000 000000 000000 000000 000000 000000 *T&8$............*
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 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 FAILE*
400/ 000104 042522 042101 020127 040506 046111 042105 051000 *D.READW FAILED.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 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 1000_8 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:

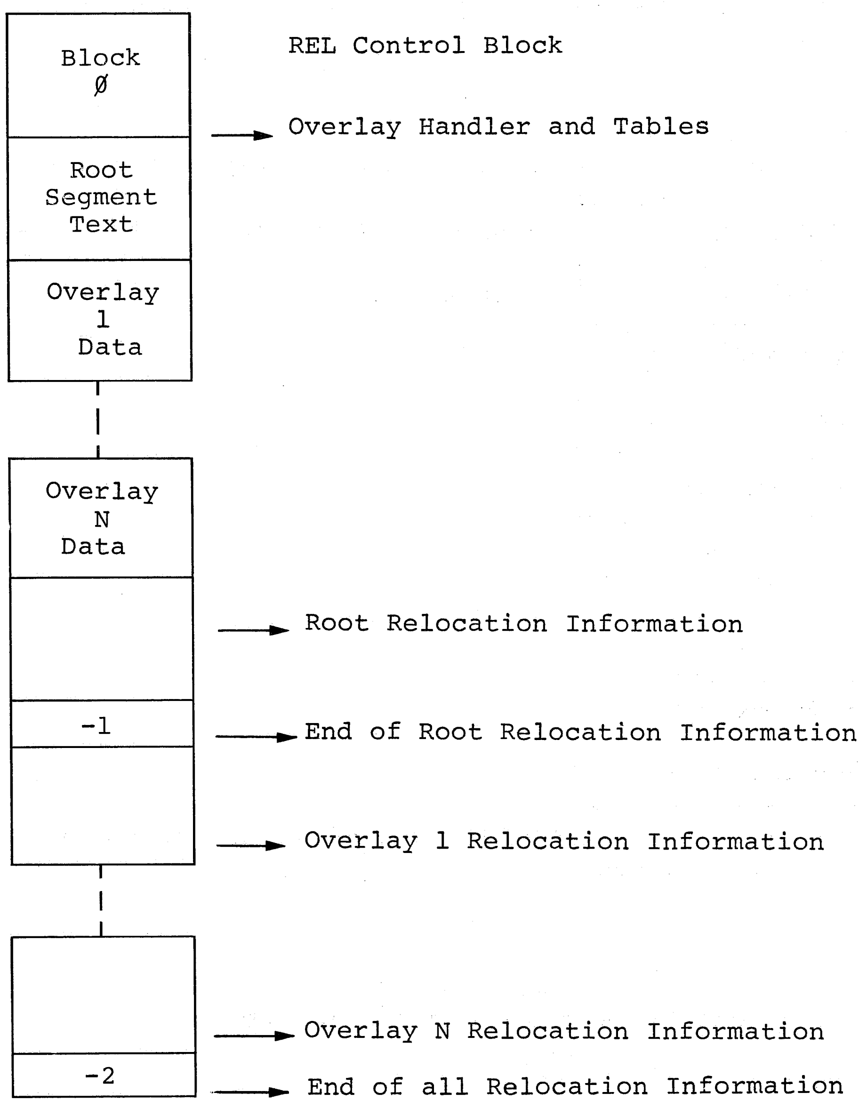


Figure 3-14
REL File With Overlays

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:

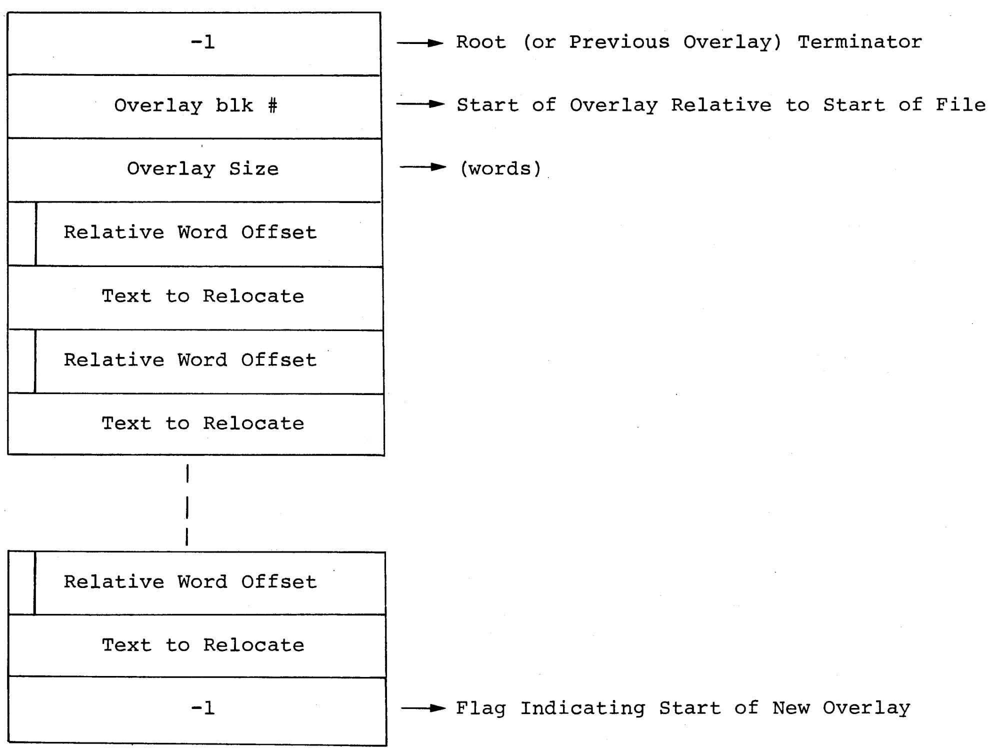


Figure 3-15
Overlay Segment Relocation Block

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:

Block #	Contents
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:

File	Contains
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

File	Contains
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.

F/B Monitor	Block # (decimal)	Contents
	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

S/J Monitor	Block # (decimal)	Contents
	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:

Overlay #	S/J	F/B
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.

Action	Explanation
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.

Action	Explanation
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:

```
.R PATCH<CR>                    [Run RT-11 PATCH program.]

PATCH Version number

FILE NAME--
*MONITR.SYS/M<CR>
*BHALT/    4Ø7     Ø<CR>
*E
.R PIP
*A=MONITR.SYS/U<CR>
*SY:/O
```

[Specifying MONITR.SYS/M indicates it is a monitor file. Change location "BHALT" from a 407 to a 0 (HALT). The correct address of BHALT can be found in Table 2 of <u>RT-11 System Release Notes</u> (V02C). E causes an exit to the monitor. 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:

Switch Register	Memory Size
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/    4Ø7      24Ø<LF>             [NOP the branch at BHALT
*BHALT+2/ 137Ø2     127Ø2<LF>           Change MOV @#SR,R2 to
*BHALT+4/ 17757Ø    xxxxxx<CR>          MOV #VAL,R2.  Address of
*E                                      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:

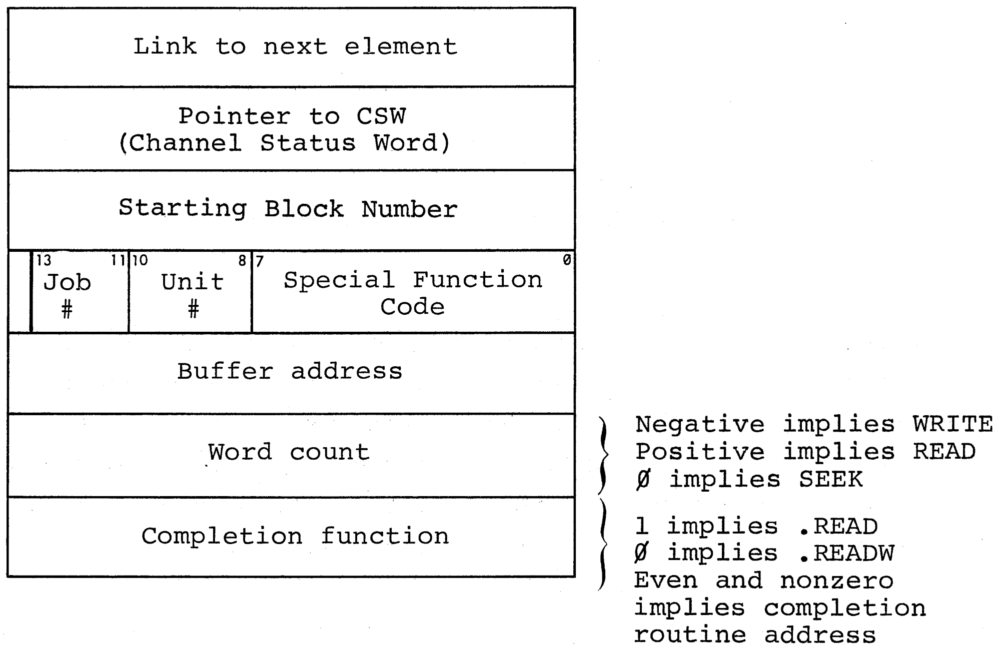
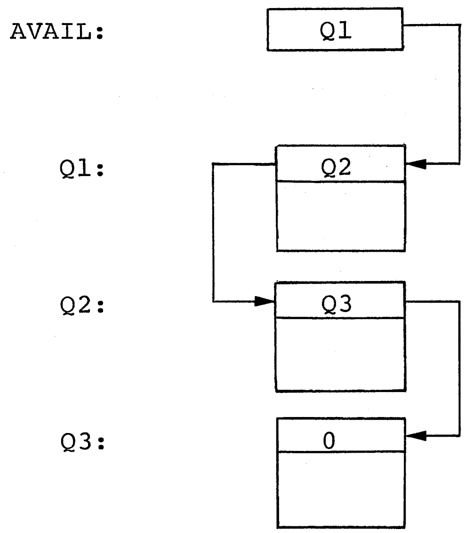


Figure 5-1
I/O Queue Element

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.

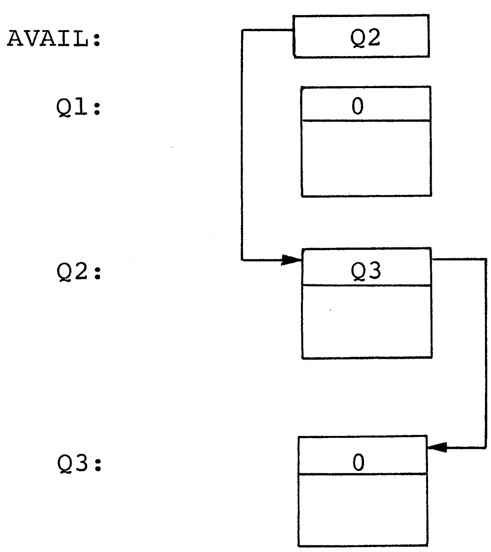
If an I/O operation is requested and a queue element is not available, RT-11 must wait until an element is free to queue the request. This obviously slows up program execution. If asynchronous I/O is desired, extra queue elements should be allocated. It is always sufficient to allocate N new queue elements, 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.

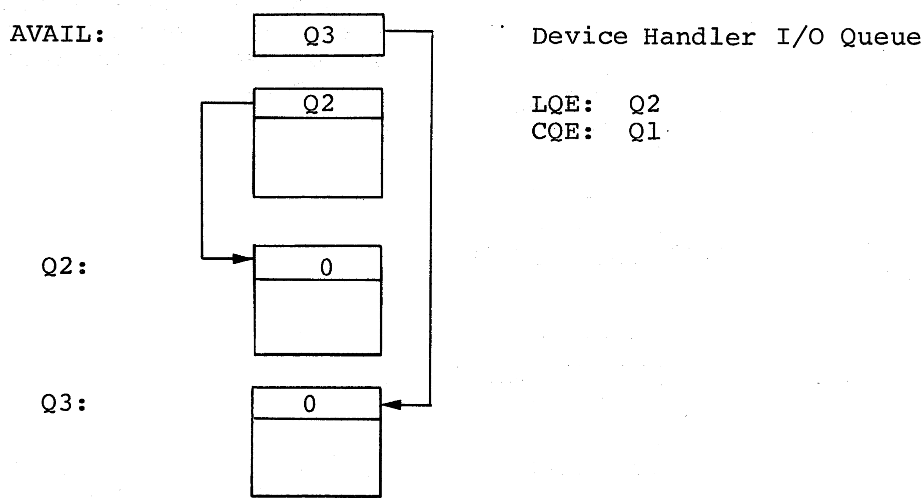
When an I/O request is initiated, an element is allocated (removed from the list of available elements) and is linked into the appropriate device handler's I/O queue. The handler's queue header consists of two pointers: the current queue element (CQE) pointer, pointing to the element at the top of the 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.



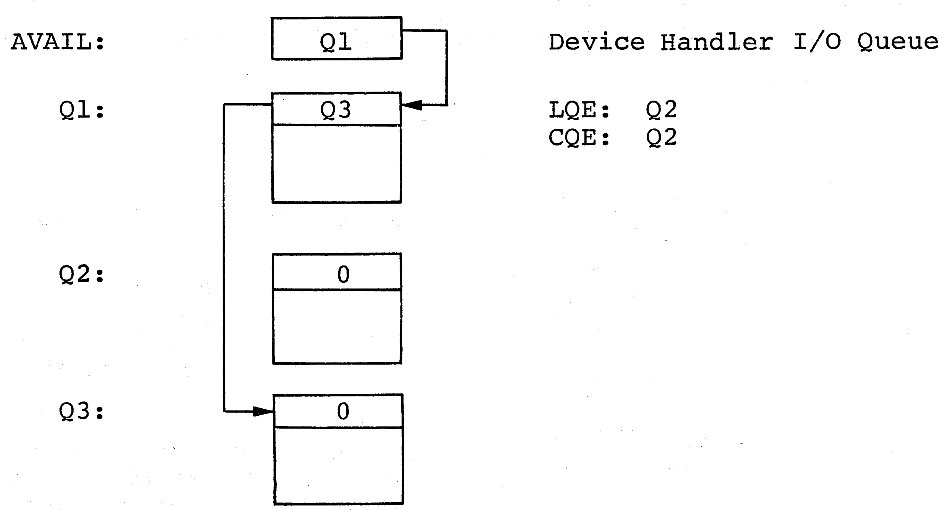
Device Handler I/O Queue

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

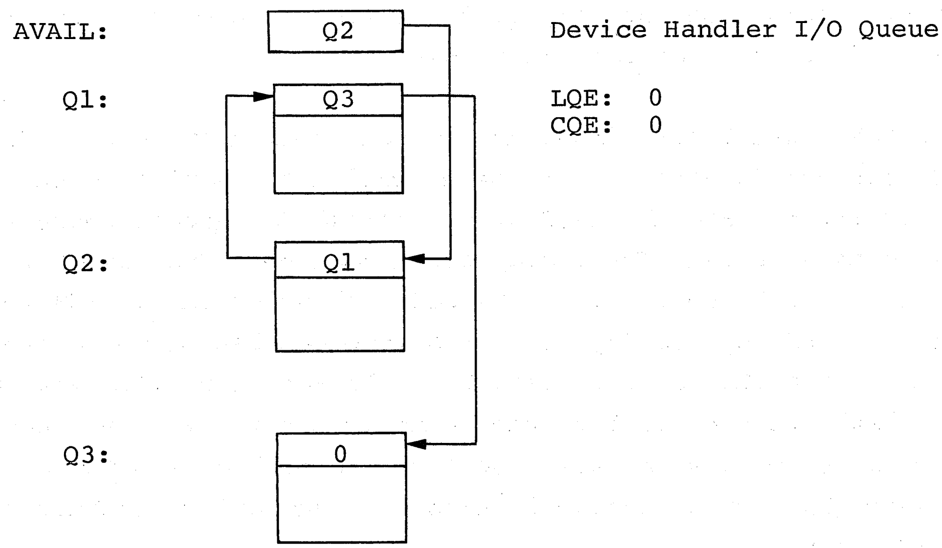
In this case, the device is associated with element Q1. If another request comes in for that same device before the first completes, a waiting queue is built up for that device.



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.

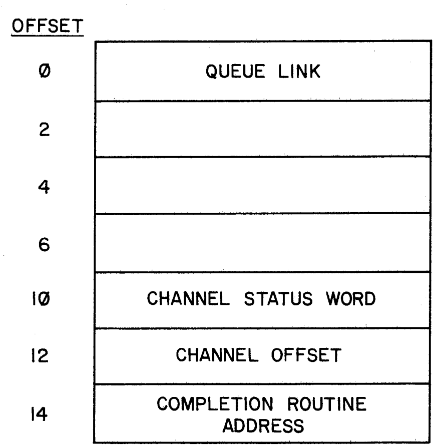


Figure 5-2
Completion Queue Element

OFFSET	
Ø	QUEUE LINK
2	JOB NUMBER
4	
6	
1Ø	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		
Ø	HIGH-ORDER TIME	C.HOT
2	LOW-ORDER TIME	C.LOT
4	LINK TO NEXT ELEMENT	C.LINK
6	JOB # OF OWNER	C.JNUM
1Ø	OWNER'S SEQUENCE #	C.SEQ
12	Ø	
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:

Word #	Contents
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 <u>RT-11 System Reference Manual</u>. 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.

Table to be Changed	Contents
$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 O in the example listing.
$DVSIZ:	2000	The disk has 1024 (decimal) 256-word blocks for storage.
$PNAME:	.RAD50 /RC/ or 7Ø37Ø	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 UNDEF GLBLS	where RC.OBJ is the handler object 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:

Table	S/J Address	F/B Address
$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>                             [/M is necessary;
*BASE;ØR<CR>                                  Monitor base;
*Ø,13652/_______4ØØØ_____316<CR>              $HSIZE table;
*Ø,137Ø6/_______Ø________2ØØØ<CR>             $DVSIZ table;
*Ø,16516/_______625Ø_____7037Ø<CR>            $PNAME table;
*Ø,16552/_______4________1ØØØ23<CR>           $STAT table;
*Ø,16336\_______77_______<CR>                 Check that vectors in
*E                                            permanent map are protected;
.                                             Exit to monitor]
```

F/B Monitor

```
.R PATCH

PATCH Version number

FILE NAME--
*MONITR.SYS/M<CR>                               [/M is necessary;
*BASE;ØR<CR>                                    Monitor base;
*Ø,14556/      4ØØØ     316<CR>                 $HSIZE table;
*Ø,14612/      Ø        2ØØØ<CR>                $DVSIZ table;
*Ø,17630/      625Ø     7Ø37Ø<CR>               $PNAME table;
*Ø,17664/      4        1ØØØ23<CR>              $STAT table;
*Ø,17336       77       <CR>                    Check that vectors in
*E                                              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 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:

Table	S/J Address	F/B Address
$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 = 2ØØØ
$PNAME = 7Ø37Ø
$STAT  = 1ØØØ23
$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;ØR <CR>                            Monitor base;
*Ø,13674/      4ØØØ     316<CR>          $HSIZE table;
*Ø,1373Ø/      Ø        2ØØØ<CR>         $DVSIZ table;
*Ø,16516/      625Ø     7Ø37Ø<CR>        $PNAME table;
*Ø,16552/      4        1ØØØ23<CR>       $STAT table;
*Ø,16612/      Ø        45Ø56<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;ØR<CR>
*Ø,146Ø2/      4ØØØ     316<CR>         [$HSIZE table;
*Ø,14636/      Ø        2ØØØ<CR>         $DVSIZ table;
*Ø,1764Ø/      625Ø     7Ø37Ø<CR>        $PNAME table;
*Ø,17674/      4        1ØØØ23<CR>       $STAT table;
*Ø,17564/      Ø        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:

Code	Function
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 <u>RT-11 System Reference Manual</u>. 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=8Ø
```

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
.RAD5Ø   /option/              [2 words of RAD50]
.BYTE    <routine-4ØØ>/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 <u>RT-11 System Reference Manual</u>. 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 (core memory)	11/20	11/40	11/45
	.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:

Tag	Bit in I.JSTA Word	Condition
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:

EMT	BATCH Handler Routine
.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:

TAG	BIT	DESCRIPTION
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 .SAVESTATUS 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,Ø
```

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
BATICT	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 Ø of .DSTATUS)
ARGARG	4732	EMT argument list for BA handler initialization

(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  BEX
MSG:    .ASCIZ  /THIS MESSAGE COMES FROM THE BATCH STREAM/
        .EVEN
        .LIST   BEX
        .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.*
$EOJ
```

EDITIT.BAT

```
$JOB/RT11
$! JOB TO EDIT FILE.MAC
        %S                  !INCREMENT S TO PREVENT RECURSION
        LET A=33            !A IS ALT MODE
.R EDIT
*EBFILE.MAC'A'R'A''A'
*GMSG:'A'KI     .ASCIZ  /MODIFIED BY EDITOR RUN BY BATCH/
*'A'EX'A''A'
$EOJ
```

EXAMPL.CTL

```
\C
$JOB

$MESSAGE EXAMPLE BATCH STREAM
\E\@ EXAMPLE BATCH STREAM
\C
$CREATE FILE.MAC
\ER BATCH
\DFILE.MAC/M=
        .MCALL  .REGDEF,.PRINT,.EXIT
        .REGDEF
START:  .PRINT  #MSG
        .EXIT
        .NLIST  BEX
MSG:    .ASCIZ  /THIS MESSAGE COMES FROM THE BATCH STREAM/
        .EVEN
        .LIST   BEX
        .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
\F\D\ER LINK
\D000000=FILE
\ER \D000000
\ER PIP
\D000000.SAV/D
\C
$PRINT FILE.LST
\ER PIP
\DLST:*.*/X=FILE.LST
\F\D\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
\DEDITIT/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$$$$$$\F\ER BATCH
\D/R
\E\F
```

EDITIT.CTL

```
\C
$JOB/RT11

$! JOB TO EDIT FILE.MAC
        %S                   !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/
\D\KA2EX\KA2\KA2
\C
$EOJ
\L$$$$$$\F\ER BATCH
\D/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

*
```

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

  000000                        .MCALL  .REGDEF,.PRINT,.EXIT
  000000                        .REGDEF
  000000              START:    .PRINT  #MSG
  000006                        .EXIT
                                .NLIST  BEX
  000010      124 MSG:          .ASCIZ  /THIS MESSAGE COMES FROM THE BATCH STREAM/
                                .EVEN
                                .LIST   BEX
          000000'               .END    START
```

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

MSG      000010R          PC      =%000007          R0      =%000000
R1      =%000001          R2      =%000002          R3      =%000003
R4      =%000004          R5      =%000005          SP      =%000006
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


$PRINT FILE.LST


$DELETE FILE.LST


$RT11
        IF(S=0) ,,L2

$CALL EDITIT    !CALL EDITIT TO EDIT FILE.MAC



$JOB/RT11

$! JOB TO EDIT FILE.MAC
        %S                  !INCREMENT S TO PREVENT RECURSION
        LET A=33            !A IS ALT MODE

*EBFILE.MAC$R$$
*
GMSG:$KI         .ASCIZ  /MODIFIED BY EDITOR RUN BY BATCH/
$EX$$


$EOJ
$$$$$$


$RT11
        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

  000000                     .MCALL  .REGDEF,.PRINT,.EXIT
  000000                     .REGDEF
  000000           START:    .PRINT  #MSG
  000006                     .EXIT
                             .NLIST  BEX
  000010     115 MSG:        .ASCIZ  /MODIFIED BY EDITOR RUN BY BATCH/
                             .EVEN
                             .LIST   BEX
          000000'            .END    START
```

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

MSG      000010R          PC     =%000007          R0     =%000000
R1     =%000001           R2     =%000002          R3     =%000003
R4     =%000004           R5     =%000005          SP     =%000006
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) ,,L2
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 $3-10_8$)
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

A.1 RC11/RS64 DEVICE HANDLER


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

;DEC-11-XXXXX-A

;JFG

;OCTOBER 1974

;COPYRIGHT (C) 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 ANY ERRORS THAT MAY APPEAR
;IN THIS DOCUMENT.

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

```
                                        .SBTTL HANDLER DEFINITIONS
                                        .MCALL  .REGDEF, ..V2..

  000000                                ..V2..

                                ;REGISTER DEFINITION
                                        .LIST ME
  000000                                .REGDEF
         000000                 R0=%0
         000001                 R1=%1
         000002                 R2=%2
         000003                 R3=%3
         000004                 R4=%4
         000005                 R5=%5
         000006                 SP=%6
         000007                 PC=%7
                                        .NLIST ME

                                ;RT-11  MONITOR DEFINED CONSTANTS
         000054                         MONLOW= 54      ;MONITOR BASE POINTER
         000270                         OFFSET= 270     ;POINTER TO Q MANAGER
                                                        ;COMPLETION ENTRY
         000001                         HDERR=  1       ;HARD ERROR BIT
         000010                         RCTRY=  8.      ;RETRY FOR ERRORS

                                ;PRIORITY CONSTANTS
         000340                         PR7 =   340     ;HANDLER ENTERED AT PR7
         000240                         PR5 =   240     ;HANDLER RUNS AT PR5

                                ; RC-11 COMMUNICATION CONSTANTS
         000103                         WR=     103     ;SET INTERRUPT ENABLE,WRITE & INITIATE FUNC.
         000105                         RD=     105     ;SET INTERRUPT ENABLE,READ & INITIATE FUNC.
         001000                         INHCA=  1000    ;INHIB. INCRE. CURRENT ADR REG (RCCA)
         000400                         ABORT=  400     ;ABORT OPERATION IN PROGRESS (RCCS)
         060000                         RTRYER= 060000  ;RETRY AFTER ERROR MASK FOR RCCS
                                                        ;BIT 14 = 1 => DATA ERROR
                                                        ;BIT 13 = 1 => ADDRESS ERROR

                                ;RC-11  CONTROL REGISTERS
         177440                         RCLA =  177440  ;LOOK AHEAD REGISTER
         177442                         RCDA =  177442  ;DISK ADDRESS REGISTER
         177444                         RCER =  177444  ;DISK ERROR STATUS REGISTER
         177446                         RCCS =  177446  ;DISK CONTROL AND STATUS
                                                        ;REGISTER
         177450                         RCWC =  177450  ;WORD COUNT REGISTER
         177452                         RCCA =  177452  ;CURRENT ADDRESS REGISTER
         177454                         RCMN =  177454  ;MAINTENANCE REGISTER
         177456                         RCDB =  177456  ;DATA BUFFER REGISTER
```

```
       000210                     RCVEC = 210     ;INTERRUPT VECTOR ADDRESS

                              ;RC SYSTEM DEFINITIONS
                   (A)        .GLOBL  RCSYS, RKSYS, RFSYS, DPSYS, DSSYS, DTSYS, DXSYS
                              .GLOBL  $INPTR, $INTEN, RCINT
                              .GLOBL  RCSIZE                ;SIZE IS REQUIRED BY BSTRAP

       000000      (B)        RKSYS = 0
```

RC11 V01-01 (FIXED HEAD DISK) RT-11 MACRO VM02-09 8-APR-75 12:04:26 PAGE 2+
HANDLER DEFINITIONS

```
       000000                 RFSYS = 0
       000000                 DPSYS = 0
       000000      (B)        DSSYS = 0
       000000                 DTSYS = 0
       000000                 DXSYS = 0
```

RC11 V01-01 (FIXED HEAD DISK) RT-11 MACRO VM02-09 8-APR-75 12:04:26 PAGE 3
RC11 DEVICE HANDLER

```
                                      .SBTTL RC11 DEVICE HANDLER
                              ;       (MAXIMUM SUPPORT 1 CONTROLLER AND 4 RS64 DISKS)
                              ;       (1024 BLOCKS OF 256 WORDS)
       000000'                        .CSECT  SYSHND

000000                        RCSTRT:
                              ;LOAD POINT

000000 000210                         .WORD   RCVEC         ;ADDRESS OF INTERRUPT VECTOR
000002 000060      (C)                .WORD   RCINT-.       ;OFFSET TO INTERRUPT ROUTINE
000004 000340                         .WORD   PR7           ;PRIORITY 7

000006             (D)        RCSYS:
000006 000000                 RCLQE:  .WORD   0             ;POINTER TO LAST Q ENTRY
000010 000000                 RCCQE:  .WORD   0             ;POINTER TO CURRENT Q ENTRY

                              ;ENTRY POINT
000012 012727 000010   (E) -->        MOV     #RETRY,(PC)+  ;SPECIFY THE RETRY COUNT
000016 000000                 RETRY:  .WORD   0             ;RETRY COUNTER
000020 004067 000226          RCRETR: JSR     R0,RCCOM1     ;SET UP THROUGH COMMON ROUTINE
000024 000000                 ZERO:   .WORD   0             ;NO ZERO FILL INITIATION
000026 062704 000010                  ADD     #10,R4        ;POINT TO RCCA
000032 012514                         MOV     (R5)+,@R4     ;SET BUFFER ADR (RCCA)
000034 012544                         MOV     (R5)+,-(R4)   ;SET WORD COUNT ADR (RCWC)
000036 012705 000103                  MOV     #WR,R5        ;ASSUME WRITE FUNCTION
```

```
000042  005714                      TST     @R4                 ;CHECK WORD COUNT
000044  001427                      BEQ     RCHOME              ;NO I/O
000046  100402                      BMI     1$                  ;WRITE SPECIFIED
000050  122525                      CMPB    (R5)+,(R5)+         ;SET READ FUNCTION CODE
000052  005414                      NEG     @R4                 ;MAKE WORD COUNT (-) (RCWC)
000054  010544              1$:     MOV     R5,-(R4)            ;SET PROPER FUNCTION IN RCCS
000056  000207          (F)         RTS     PC                  ;GO-AWAY FOR I/O
```

RC11 V01-01 (FIXED HEAD DISK)   RT-11 MACRO VM02-09   8-APR-75 12:04:26 PAGE 4
RC-11 INTERRUPT ROUTINE

```
                                    .SBTTL  RC-11 INTERRUPT ROUTINE


000060  000430          (G)         BR      RCABRT              ;ABORT ROUTINE FOR F/B
                                                                ;MONITOR

000062  004577  000226  (H) RCINT:  JSR     R5,@$INPTR          ;NOTIFY MONITOR & SET
                                                                ;PRIORITY TO LEVEL 5
000066  000100          (I)         .WORD   ^C<PR5>&PR7         ;RC RUNS AT PRIORITY 5
                            ;CHECK FOR ERROR ON RC & RETRY IF APPLICABLE

000070  016705  177714              MOV     RCCQE,R5            ;GET CURRENT QUEUE ADR
000074  012704  177446              MOV     #RCCS,R4            ;POINT TO DSK CNTRL & STATUS REG
000100  005714                      TST     @R4                 ;DATA ADDRESS OR WRITE
                                                                ;CHECK ERROR OR WRITE-LOCK
                                                                ;NON-EXISTENT DISK?
000102  100023                      BPL     RCFILL              ;NO.
000104  032714  060000              BIT     #RTRYER,@R4         ;ONLY DATA & ADR ERRORS MERIT A RETRY
000110  001403                      BEQ     1$                  ;NOPE - GO BACK WITH ERROR
000112  005367  177700              DEC     RETRY               ;RETRY FATAL ERROR 8 TIMES
000116  003340                      BGT     RCRETR              ;OK DO IT AGAIN.
000120  052755  000001      1$:     BIS     #HDERR,@-(R5)       ;CAN'T GET BY HARD ERROR
                                                                ;SET ERROR FLAG +
                                                                ;GO BACK TO MONITOR
000124  010704              RCHOME: MOV     PC,R4
000126  062704  177662  (J)         ADD     #RCCQE-.,R4         ;PIC ADDR OF CURRENT QUEUE ENTRY
000132  013705  000054              MOV     @#MONLOW,R5
000136  000175  000270              JMP     @OFFSET(R5)         ;EXIT TO MONITOR QUEUE COMPLETION

000142  012737  000400  177446  RCABRT: MOV  #ABORT,@#RCCS      ;ABORT CURRENT OPERATION
000150  000765          (K)         BR      RCHOME              ;EXIT TO MONITOR

                            ;FILL REMAINDER OF DISK BLOCK WRITTEN TOO WITH ZEROES.
000152  004067  000100      RCFILL: JSR     R0,RCCOMN           ;CALL COMMON RTN
000156  001000                      .WORD   INHCA               ;SET TOO INHIBIT INC OF CURRENT ADR (RCCA)
000160  016505  000002              MOV     2(R5),R5            ;GET WORD COUNT FROM
```

```
                                                                ;CURRENT QUE ELEMENT (RCCQE)
000164  100357                          BPL     RCHOME          ;NOFILL FOR READS
000166  105705                          TSTB    R5              ;EVEN # BLOCKS WRITTEN?
000170  001755                          BEQ     RCHOME          ;YES - FILL NOT NECESSARY.
000172  005405                          NEG     R5              ;WRITE WORD COUNTS ARE NEGATIVE IN
000174  010546                          MOV     R5,-(SP)        ;THE QUE.
                                ; CALCULATE THE # OF SECTORS IN THE CURRENT OPERATION
                                ; (32 WORDS = ONE SECTOR)

000176  012746  000005                  MOV     #5,-(SP)        ;PUSH REPEAT COUNT ONTO STACK
000202  006205                  1$:     ASR     R5              ;DIVIDE THE # WORDS
000204  106066  000001                  RORB    1(SP)           ;CHECK FOR SECTOR OVERFLOW
000210  105316                          DECB    @SP             ;DECREMENT REPEAT COUNT
000212  001373                          BNE     1$
000214  005726                          TST     (SP)+           ;SECTOR OVERFLOW ?
000216  001401                          BEQ     2$              ;NO
000220  005205                          INC     R5              ;INCLUDE NEXT SECTOR
000222                          2$:
                                ; END OF SECTOR CALCULATION
000222  060524                          ADD     R5,(R4)+        ;CALCULATE CURRENT DISK ADR. (RCDA)
000224  012605                          MOV     (SP)+,R5
```

RC11 V01-01 (FIXED HEAD DISK)   RT-11 MACRO VM02-09   8-APR-75 12:04:26 PAGE 4+
RC-11 INTERRUPT ROUTINE

```
000226  052705  177400                  BIS     #177400,R5      ;WRITE MUST BE LESS THAN
                                                                ;A BLOCK (RCWC TAKES
                                                                ;2'S COMPLEMENT NEG. VALUE.)
000232  005724                          TST     (R4)+           ;POINT TO RCCS
000234  012724  001103                  MOV     #WR+INHCA,(R4)+ ;SET WRITE FUNCTION
000240  010524                          MOV     R5,(R4)+        ;SET WORD COUNT (RCWC)
000242  010714                          MOV     PC,@R4          ;POINT MEMORY ADDRESS TO A ZERO.
000244  062714  177560                  ADD     #ZERO-.,@R4     ;PTC (INTO RCCA)
000250  000207                          RTS     PC              ;EXIT
```

(L) → RTS PC

```
                                     .SBTTL COMMON SUBROUTINE
                              ;RCCOMN
                              ;COMMON SUBROUTINE USED BY INTERRUPT
                              ;AND ENTRY ROUTINES

000252  012704  177446        RCCOM1: MOV   #RCCS,R4         ;PT TO DSK CNTRL & STATUS REG
000256  031014                RCCOMN: BIT   @R0,@R4          ;FILL IN PROGRESS
000260  001402                        BEQ   1$               ;NO
000262  012600                        MOV   (SP)+,R0         ;POP R0
000264  000717                        BR    RCHOME           ;FINIS FILL OF BLK WITH 0'S
000266  016705  177516        1$:     MOV   RCCQE,R5         ;PTR TO CURRENT QUEUE ENTRY
000272  012546                        MOV   (R5)+, -(SP)     ;GET BLOCK NUMBER
000274  006316                        ASL   @SP              ;CALCULATE DISK ADDRESS FOR RCDA
000276  006316                        ASL   @SP              ;UNIT, TRACK# + SECTOR ADDRESS)
000300  006316                        ASL   @SP              ;[32*8=256]
000302  052014                        BIS   (R0)+,@R4        ;INHIB CURR, ADR INC (IF NEEDED)
000304  024444                        CMP   -(R4),-(R4)      ;POINT TO RCDA
000306  012614                        MOV   (SP)+,@R4        ;SET DISK ADR FOR TRANSFER
000310  005725                        TST   (R5)+            ;IGNORE UNIT #
000312  000200                        RTS   R0               ;RETURN TO CALLER



000314  000000G         (N)   $INPTR: .WORD  $INTEN  (M)   ;MONITOR ENTRY ADDR.

        000316                RCSIZE= .-RCSTRT  (O)          ;SIZE OF HANDLER

        000001'                       .END
```

```
RC11 V01-01 (FIXED HEAD DISK)   RT-11 MACRO VM02-09   8-APR-75 12:04:26 PAGE 5+
SYMBOL TABLE

ABORT = 000400          DPSYS = 000000 G        DSSYS = 000000 G        DTSYS = 000000 G        DXSYS = 000000 G
HDERR = 000001          INHCA = 001000          MONLOW= 000054          OFFSET= 000270          PC    =%000007
PR5   = 000240          PR7   = 000340          RCABRT  000142R    002  RCCA  = 177452          RCCOMN  000256R    002
RCCOM1  000252R    002  RCCQE   000010R    002  RCCS  = 177446          RCDA  = 177442          RCDB  = 177456
RCER  = 177444          RCFILL  000152R    002  RCHOME  000124R    002  RCINT   000062RG   002  RCLA  = 177440
RCLQE   000006R    002  RCMN  = 177454          RCRETR  000020R    002  RCSIZE= 000316 G        RCSTRT  000000R    002
RCSYS   000006RG   002  RCTRY = 000010          RCVEC = 000210          RCWC  = 177450          RD    = 000105
RETRY   000016R    002  RFSYS = 000000 G        RKSYS = 000000 G        RTRYER= 060000          R0    =%000000
R1    =%000001          R2    =%000002          R3    =%000003          R4    =%000004          R5    =%000005
SP    =%000006          WR    = 000103          ZERO    000024R    002  SINPTR  000314RG   002  SINTEN= ****** G
...V2 = 000001
. ABS.  000000     000
        000000     001
SYSHND  000316     002
ERRORS DETECTED: 0
FREE CORE: 15627. WORDS

RC,LP:/N:TTM/C=RC
```

A.2 RC11/RS64 BOOTSTRAP

```
BOOT V02B-01     RT-11 BOOTSTRAP RT-11 MACRO VM02-09    8-APR-75 11:49:04
TABLE OF CONTENTS

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


BOOT V02B-01     RT-11 BOOTSTRAP RT-11 MACRO VM02-09    8-APR-75 11:49:04 PAGE 1

          000001                $RCSYS=1
                                .TITLE  BOOT V02B-01     RT-11 BOOTSTRAP
                                ; RT-11 BOOTSTRAP
                                ;
                                ; DEC-11-ORBTA-D
                                ;
                                ; COPYRIGHT (C) 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.
```

```
                                      .SBTTL  MACROS, GLOBALS

                                      .MCALL  ..V1..
  000000                              ..V1..
                                      .MCALL  .EXIT,  .LOOKUP,.PRINT, .SAVESTATUS

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

                                      ; GLOBAL REFERENCES TO MONITOR:
                                      .GLOBL  $OVREC, $ENTRY, $INPTR, $KMLOC, $MONBL, $PNAME, $SLOT
                                      .GLOBL  $SWPBL, $USRLC, $PNAMO
                                      .GLOBL  BSTRNG, CORPTR, DKASSG, FILLER, HWFPUS, HWDSPS, KMLOC
                                      .GLOBL  KMON,   KMONSZ, KW11LS, MAPOFF, QCOMP,  RT11SZ
                                      .GLOBL  RTLEN,  RTSIZE, SWAPSZ, SYENTO, SYINDO, SYNCH,  SYASSG
                                      .GLOBL  SYSLOW, TTIBUF, TTOBUF, USRLOC, USRSZ, MAXSYH

                                      .GLOBL  RELLST

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

                                      .IF NE BF
                                      .GLOBL  BCNTXT, BKGND1, BKGND2, BKGND3, CNTXT,  FUDGE1, FUDGE2
                                      .GLOBL  MSGENT, RMONSP, SWIPTR, SWOPTR, TTIUSR, TTOUSR, .SCRTN
                                      .IFF
                                      .GLOBL  AVAIL,  I.CSW,  FPPADD, FPPIGN, MONLOC, TRAPLC, TRAPER
                                      .ENDC

           002000                     PERM    = 2000                  ;STATUS WORD FOR PERMANENT FILE
           004000                     ENDBLK  = 4000                  ;STATUS OF END OF SEGMENT MARK
           000044                     JSW     = 44                    ;ADDRESS OF JOB STATUS
           177570                     SR      = 177570                ;CONSOLE SWITCH REGISTER

                                      ; REGISTER DEFINITIONS:

           000000                     R0=%0
           000001                     R1=%1
           000002                     R2=%2
           000003                     R3=%3
           000004                     R4=%4
           000005                     R5=%5
           000006                     SP=%6
           000007                     PC=%7

                                      ; MONITOR OFFSET CONSTANTS
```

```
        000300                      CONFIG  = 300           ;HARDWARE CONFIGURATION WORD
        000274                      SYUNIT  = 274           ;SYSTEM UNIT #

        177546                      LKCS    = 177546        ;CLOCK STATUS REGISTER
        172000                      GT40    = 172000        ;GT40 LOCATION
        177560                      TKS     = 177560        ;KEYBOARD STATUS
        177562                      TKB     = 177562        ;   "     BUFFER
        177564                      TPS     = 177564        ;PRINTER STATUS
        177566                      TPB     = 177566        ;   "     BUFFER
```



```
                                    .SBTTL  ASECT

                                    .IF NDF $RFSYS                  ;TURN ON $RKSYS IF ALL OTHERS ARE OFF
                                    .IF NDF $DTSYS
                                    .IF NDF $DPSYS
                                    .IF NDF $DSSYS
                          (AA) ---- .IF NDF $RCSYS
                                    .IF NDF $DXSYS
                                    $RKSYS= 0           ;IT MUST BE AN RK SYSTEM
                                    .ENDC
                                    .ENDC
                                    .ENDC
                                    .ENDC
                                    .ENDC
                                    .ENDC

        000000                              .ASECT
        000000                              . = 0
 000000 000240                              240                     ;BOOT VALIDATION PATTERN
 000002 000414                              BR      BOOT1           ;BRANCH TO REAL BOOT

                                            .IF NDF $DXSYS
        000034                              . = 34                  ;PUT THE JUMP BOOT IN TRAP VECTOR
 000034 000167  000460              BOOT1:  JMP     BOOT            ;START THE BOOTSTRAP

                                            .IFF
                                            CSGO=   1               ;START FUNCTION
                                            CSEBUF= 2               ;EMPTY BUFFER
                                            CSRD=   6               ;READ SECTOR
                                            CSUNIT= 20              ;UNIT 1 SELECTION
                                            CSDONE= 40              ;RX DONE
                                            CSTR=   200             ;RXDB TRANSFER READY
                                            CSERR=  100000          ;RX ERROR

                                            RXCS=   177170          ;RXCS STATUS REGISTER
```

```
                .  = 14                         ;INITIALIZE BPT AND IOT VECTORS
                .WORD   READS                   ;ON BPT INTERUPT TO READS ROUTINE
                .WORD   0                       ;PS SET TO 0
                .WORD   WAIT                    ;ON IOT INTERUPT TO WAIT ROUTINE
UNITRD: .BYTE   CSGO+CSRD               ;READ FROM UNIT 0, SETS WEIRD BUT OK PS
                .BYTE   CSGO+CSRD+CSUNIT;READ FROM UNIT 1
                .  = 34                         ;34-52 USEABLE
BOOT1:  MOVB    UNITRD(R0),RDCMD;SET READ FUNCTION FOR CORRECT UNIT
RETRY:  MOV     @PC,SP                  ;INIT SP WITH NEXT INSTRUCTION
        MOV     #200,R2                 ;AREA TO READ IN NEXT PART OF BOOT
        CLR     R0                      ;SET TRACK NUMBER
        BR      2$                      ;OUT OF ROOM HERE, GO TO CONTINUATION
                .  = 70                         ;PAPER TAPE VECTORS
2$:     MOV     SP,R1                   ;SET TO BIG WORD COUNT
        INC     R0                      ;SET TO ABSOLUTE TRACK 1
        BR      3$                      ;BRANCH TO CONTINUATION
                .  = 104                        ;PROGRAMMABLE CLOCK
3$:     MOV     @PC,R3                  ;ABSOLUTE SECTOR 3 FOR NEXT PART
        BPT                             ;CALL READS SUBROUTINE
        BR      BOOT2                   ;BRANCH TO CONTINUATION
                .  = 120                        ;LOTS OF UNUSED VECTORS, (DR-11B?)

BOOT V02B-01     RT-11 BOOTSTRAP RT-11 MACRO VM02-09    8-APR-75 11:49:04 PAGE 3+
ASECT

READS:  MOV     #RXCS,R4                ;R4 => RX STATUS REGISTER
        MOV     R4,R5                   ;R5 WILL POINT TO RX DATA BUFFER
        MOV     (PC)+,(R5)+             ;INITIATE READ FUNCTION
RDCMD:  .WORD   0                       ;GETS FILLED WITH READ COMMAND
        IOT                             ;CALL WAIT SUBROUTINE
        MOV     R3,@R5                  ;LOAD SECTOR NUMBER INTO RXDB
        IOT                             ;CALL WAIT SUBROUTINE
        MOV     R0,@R5                  ;LOAD TRACK NUMBER INTO RXDB
        IOT                             ;CALL WAIT SUBROUTINE
        MOV     #CSGO+CSEBUF,@R4;LOAD EMPTY BUFFER FUNCTION INTO RXCS
4$:     IOT                             ;CALL WAIT SUBROUTINE
        TSTB    @R4                     ;IS TRANSFER READY UP?
        BPL     RTIRET                  ;BRANCH IF NOT, SECTOR MUST BE LOADED
        MOVB    @R5,(R2)+               ;MOVE DATA BYTE TO MEMORY
        DEC     R1                      ;CHECK BYTE COUNT
        BGT     4$                      ;LOOP AS LONG AS WORD COUNT NOT UP
        CLR     R2                      ;KLUDGE TO SLUFF BUFFER IF SHORT WD CNT
        BR      4$                      ;LOOP
WAIT:   TST     @R4                     ;IS TR, ERR, DONE UP? INT ENB CAN'T BE
        BEQ     WAIT                    ;LOOP TILL SOMETHING
        BMI     RETRY                   ;START AGAIN IF ERROR
RTIRET: RTI                             ;RETURN
```

```
         . = 200                     ;SECTOR 2 OF RX BOOT
BOOT2:   CMPB    (R3)+,(R3)+         ;BUMP TO SECTOR 5
         BPT                         ;CALL READS SUBROUTINE
         CMPB    (R3)+,(R3)+         ;BUMP TO SECTOR 7
         BPT                         ;CALL READS SUBROUTINE
         BIT     #CSUNIT,RDCMD       ;CHECK UNIT ID
         BNE     1$                  ;BRANCH IF BOOTING UNIT 1, R0=1
         CLR     R0                  ;SET TO UNIT 0
1$:      MOV     R0,(PC)+            ;SAVE UNIT BOOTED FROM FOR LATER
BTUNIT:  .WORD   0                   ;SAVE THE UNIT HERE
         MOV     #TRWAIT,@#20        ;LETS HANDLE ERRORS DIFFERENTLY
         JMP     BOOT                ;NOW WE ARE READY TO DO THE REAL BOOT

         .ENDC
```



```
; FOLLOWING ARE THE BOOTSTRAP I/O DRIVERS FOR EACH VALID
; SYSTEM DEVICE.
; CALLING SEQUENCE:
;        R0 = PHYSICAL BLOCK TO READ/WRITE
;        R1 = WORD COUNT
;        R2 = BUFFER ADDRESS
;        R3,R4,R5 ARE AVAILABLE AND MAY BE DESTROYED BY THE DRIVER
; THE DRIVER MUST GO TO BIOERR IF A FATAL I/O ERROR OCCURS.
; IT MUST ALSO INVOKE THE MACRO SYSDEV
.MACRO  SYSDEV  NAME,VECTOR
.GLOBL  NAME'INT, NAME'SIZE      ;DEFINE SYSTEM DEVICE INTERRUPT & SIZE
SYNAME  = 0
.IRPC   X,<NAME>
SYNAME  = <SYNAME+''X-100>*50
.ENDR
SYVEC   = VECTOR                 ;IT VECTORS TO THIS LOCATION
. = SYVEC                        ;AT THE VECTORS
        .WORD   NAME'INT,340     ;  PUT A VECTOR TO THE SYSTEM HANDLER
. = SYSIZE
        .WORD   NAME'SIZE        ;PUT HANDLER SIZE WHERE IT CAN BE USED
. = 402                          ;AND START THE CODE AT 402
SYBITO  = VECTOR / 20            ;OFFSET INTO BIT MAP FOR PROTECTION
SYBITS  = ^B11000000             ;COMPUTE ACTUAL BITS
.REPT   <VECTOR & 17> / 4        ;VECTOR IS A MULTIPLE OF 4
SYBITS  = SYBITS / 4             ;SHIFT RIGHT 2 MORE BITS
.ENDR
.ENDM   SYSDEV
```

```
.IF DF  $DSSYS  ;RS SYSTEM
.SBTTL  BOOTSTRAP 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
        BIT     #2,16(R5)       ;WHAT IS IT?
        BNE     1$              ;IT'S AN RS04
        ASL     R4              ;IT'S AN RS03
1$:     ASL     R4              ;CONVERT TO TRACK/SECTOR
        BIC     #^C7,(SP)       ;STRIP TO UNIT BITS
        MOV     (SP)+,(R5)      ;SET UNIT
        MOV     R4,-(R5)        ;SET BLOCK
        MOV     R2,-(R5)
        MOV     R1,-(R5)
        NEG     @R5
        MOV     #71,-(R5)       ;GO, READ, NO INTERRUPT
2$:     BIT     #100200,@R5     ;WAIT FOR DONE OR ERROR
        BEQ     2$
        BMI     BIOERR          ;BOOT ERROR
        RTS     PC

.ENDC
```

```
        .IF DF  $DPSYS                          ;CONDITIONAL FOR RP11 DISK
        .SBTTL  BOOTSTRAP I/O DRIVER - RP11

; RP11 DISK DRIVER

        SYSDEV  DP,254                          ;DEVICE IS RP. IT VECTORS TO 254
RPCS=   176714                                  ;RP11 DEVICE CONTROL REG
RPDS=   176710                                  ;RP03 DEVICE STATUS REG
RPDA=   176724                                  ;RP03 DISK ADRS REGISTER
CS.GO=  000001                                  ;GO BIT IN CONTROL & STATUS
CS.RD=  000004                                  ;READ FUNCTION CODE
CS.DRV= 003400                                  ;UNIT SELECT BITS
DS.ATT= 000377                                  ;UNIT ATTN BITS

READ:   MOV     R0,R3                   ;R3 = BLOCK #
        JSR     R2,DIV                  ;GET SECTOR NUMBER
        .WORD   10.                     ;BY DIVIDING BY 10
        MOV     R4,-(SP)                ;SAVE SECTOR
        MOV     R5,R3                   ;SET NEW DIVIDEND
        JSR     R2,DIV                  ;AND COMPUTE CYL & TRACK
        .WORD   20.                     ;BY DIVIDING BY 20
        SWAB    R4                      ;POSITION TRACK IN HIGH BYTE
        BIS     (SP)+,R4                ;AND INSTALL SECTOR
        MOV     #RPDA,R3                ;R3 -> DISK ADRS REG
        MOV     R4,@R3                  ;SET TRACK & SECTOR
        MOV     R5,-(R3)                ;AND CYLINDER
        MOV     R2,-(R3)                ;AND BUS ADDRESS
        MOV     R1,-(R3)                ;AND WORD COUNT
        NEG     @R3                     ;MAKE NEGATIVE
        BIC     #^C<CS.DRV>,-(R3) ;CLEAR ALL BUT UNIT #
        BIS     #CS.RD+CS.GO,@R3 ; AND START READ
1$:     TSTB    @R3                     ;WAIT UNTIL TRANSFER COMPLETE
        BPL     1$
        TST     @R3                     ;ANY ERRORS?
        BMI     BIOERR                  ;YES
        MOVB    #DS.ATT,@#RPDS          ;CLEAR UNIT ATTN FOR BOTH
        CLRB    @#RPDS                  ;  OLD & ECO'D CONTROLLERS
        RTS     PC                      ;ELSE JUST RETURN

; DIVIDE ROUTINE FOR RP HANDLER.
; R5 = R3 / @R2, REMAINDER IN R4

DIV:    CLR     R5                      ;QUOT. = 0
        CLR     R4                      ;REM. = 0
        TST     R3                      ;IS DIVIDEND 0?
        BEQ     4$                      ;YES - JUST RETURN
        COM     R5                      ;QUOT. = -1 & SET CARRY
1$:     ROL     R3                      ;NORMALIZE
```

```
                        BCC     1$
                2$:     ROL     R4              ;SHIFT & SUBTRACT
                        CMP     R4,@R2
                        BLO     3$
                        SUB     @R2,R4
                3$:     ROL     R5
                        ASL     R3
                        BNE     2$
                        COM     R5              ;FIX QUOTIENT
```



```
                4$:     TST     (R2)+
                        RTS     R2

                .ENDC
```



```
        .IF DF  $RKSYS!$RFSYS!$RCSYS        (BB)

        .IFDF   $RFSYS                  ;CONDITIONAL FOR RF DISK
        .SBTTL  BOOTSTRAP I/O DRIVER - RF11

        ; RF11 DISK HANDLER

                SYSDEV  RF,204          ;DEVICE IS RF. IT VECTORS TO 204.
        RFCS    = 177460                ;CONTROL & STATUS REGISTER
        RFWC    = 177462                ;WORD COUNT
        RFMA    = 177464                ;MEMORY ADDRESS
        RFDA    = 177466                ;DISK ADDRESS
        RFDE    = 177470                ;DISK ADDRESS EXTENSION
        RFDB    = 177472                ;DATA BUFFER

        READ:   MOV     #RFDA,R3        ;POINT TO DISK ADDRESS
                MOV     R0,R5           ;COPY BLOCK NUMBER
                SWAB    R5              ;MULTIPLY BY 256 TO GET WORD # ON DISK
                MOV     R5,R4           ;SAVE HIGH ORDER DISK ADDRESS
                CLRB    R5              ;MAKE DA AN EVEN BLOCK NUMBER
                MOV     R5,(R3)+        ;PUT LOW ORDER ADDRESS IN CONTROLLER
                BIC     #177740,R4      ;ISOLATE HIGH ORDER ADDRESS
                MOV     R4,(R3)         ;PUT IT IN CONTROLLER
                TST     -(R3)           ;RESET POINTER
```

```
                                              .ENDC
                           (CC)               .IFDF   $RCSYS                    ;CONDITIONAL FOR RC (RS64) DISK

                                                      .SBTTL  BOOTSTRAP I/O DRIVER - RC11

                                              ; RC11 DISK HANDLER
     000040                (DD)                       SYSDEV  RC,210            ;DEVICE IS RC, IT VECTORS TO 210

                                              ; RC11 CONTROL REGISTERS

             177440                                   RCLA=   177440            ;LOOK AHEAD REGISTER
             177442                                   RCDA=   177442            ;DISK ADDRESS REGISTER
             177444                                   RCER=   177444            ;DISK ERROR STATUS REGISTER
             177446                                   RCCS=   177446            ;DISK CONTROL & STATUS REGISTER
             177450                                   RCWC=   177450            ;WORD COUNT REGISTER
             177452                                   RCCA=   177452            ;CURRENT ADDRESS REGISTER
             177454                                   RCMN=   177454            ;MAINTENANCE REGISTER
             177456                                   RCDB=   177456            ;DATA BUFFER REGISTER

     000402  012703  177442  (EE)     READ:           MOV     #RCDA,R3          ;PT TO DISK ADR REGISTER
     000406  010005                                   MOV     R0,R5             ;GET BLOCK NUMBER
     000410  006305          (FF)                     ASL     R5                ;CALCULATE DISK ADR FOR RCDA
     000412  006305                                   ASL     R5                ;(UNIT,TRACK # & SECTOR ADR)
     000414  006305                                   ASL     R5                ;[32 * 8=256]
     000416  010513                                   MOV     R5,@R3            ;IND PROPER DISK ADR
     000420  062703  000012                           ADD     #12,R3            ;PT TO CURRENT ADR REG + 12
                                              ;                                 ;(INTERFACE TO COMMON CODE)
                                              .ENDC
                                              .IFDF   $RKSYS
                                              .SBTTL  BOOTSTRAP I/O DRIVER - RK05

                                              ; RK05 DISK HANDLER
```

BOOT V02B-01 RT-11 BOOTSTRAP RT-11 MACRO VM02-09 8-APR-75 11:49:04 PAGE 7+
BOOTSTRAP I/O DRIVER - RC11

```
                                                      SYSDEV  RK,220            ;DEVICE IS RK, IT VECTORS TO 220
                                              RKDA    = 177412                  ;RK DISK ADDRESS

                                              READ:   MOV     #14,R3            ;PHYSICAL BLOCK TO RK DISK ADD.
                                                      BR      2$                ;ENTER BLOCK # COMPUTATION
                                              1$:     ADD     #20,R3            ;CONVERT DISK ADDRESS
                                              2$:     SUB     #14,R0
                                                      BPL     1$
                                                      ADD     R3,R0             ;R0 HAS DISK ADDRESS
                                              5$:     MOV     #RKDA,R3          ;POINT TO HARDWARE DISK ADDR REGISTER
                                                      BIC     #17777,@R3        ;LEAVE THE UNIT NUMBER
                                                      BIS     R0,(R3)           ;PUT DISK ADDRESS INTO CONTROLLER
```

GG

```
                          .ENDC
                          ; THIS CODE IS COMMON TO RK05,RC11 AND RF11 HANDLERS
000424  010243                    MOV     R2,-(R3)          ;BUFFER ADD.
000426  010143                    MOV     R1,-(R3)          ;WORD COUNT
000430  005413                    NEG     (R3)              ;(NEGATIVE)
000432  012743  000005            MOV     #5,-(R3)          ;START DISK READ
000436  105713            3$:     TSTB    (R3)              ;WAIT UNTIL COMPLETE
000440  100376                    BPL     3$
000442  005713                    TST     (R3)              ;ANY ERRORS?
000444  100401                    BMI     BIOERR            ;HARD HALT ON ERROR
000446  000207                    RTS     PC

                          .ENDC
```

BOOT V02B-01   RT-11 BOOTSTRAP RT-11 MACRO VM02-09   8-APR-75 11:49:04 PAGE 8
BOOTSTRAP I/O DRIVER - RC11

```
        .IF DF SDTSYS

        .SBTTL  BOOTSTRAP I/O DRIVER - DECTAPE

        ; DECTAPE BOOTSRAP HANDLER
                SYSDEV  DT,214          ;DEVICE IS DT. IT VECTORS TO 214.
        TCCM    = 177342                ;COMMAND REGISTER
        TCDT    = 177350                ;DATA REGISTER
        TCST    = 177340                ;STATUS REGISTER

        READ:   MOV     #TCCM,R4        ;R4 => COMMAND REG
                MOV     #TCDT,R3        ;R3 => DATA REG
        DTSRCH: MOV     R0,R5           ;COPY BLOCK NUMBER
                SUB     #2,R5           ;SEARCH FOR 2 EARLIER
                MOV     #4003,@R4       ;REVERSE,RNUM
        2$:     BIT     #100200,@R4     ;WAIT TILL BLOCK FOUND
                BEQ     2$
                BMI     DTERR
                CMP     R5,@R3          ;IS IT THE DESIRED BLOCK
                BLT     DTSRCH          ;NO,CONTINUE SEARCHING
        DTFWRD: MOV     #3,@R4          ;SEARCH FORWARD (RNUM)
        4$:     BIT     #100200,@R4     ;WAIT
                BEQ     4$
                BMI     DTERR
                CMP     R0,@R3          ;DESIRED BLOCK
                BGT     DTFWRD          ;NO-SEARCH FORWARD
                BLT     DTSRCH          ;NO-SEARCH REVERSE
                MOV     R2,-(R3)        ;BUFFER ADDRESS
                NEG     R1
                MOV     R1,-(R3)        ;WORD COUNT
```

```
                MOV     #5,@R4          ;READ
DT4:            BIT     #100200,@R4     ;WAIT FOR COMPLETION
                BEQ     DT4
                BMI     BIOERR          ;READ ERROR
                CLR     @R4             ;STOP DT
                RTS     PC
DTERR:          TST     @#TCST          ;WHAT KIND OF ERROR ?
                BPL     BIOERR          ;NOT END ZONE
                BIT     #4000,@R4       ;REVERSE?
                BNE     DTFWRD          ;THEN GO SEARCH FORWARD
                BR      DTSRCH          ;ELSE SEARCH REVERSE

        .ENDC
```

BOOT V02B-01   RT-11 BOOTSTRAP RT-11 MACRO VM02-09   8-APR-75 11:49:04 PAGE 9
BOOTSTRAP I/O DRIVER - RC11

```
                .IF DF  SDXSYS          ;FLOPPY SYSTEM
                .SBTTL  BOOTSTRAP I/O DRIVER - FLOPPY

                SYSDEV  DX,264          ;FLOPPY VECTORS THROUGH 264

READ:           ASL     R0              ;CONVERT BLOCK TO LOGICAL SECTOR
                ASL     R0              ;LSN=BLOCK*4
                ASL     R1              ;MAKE WORD COUNT BYTE COUNT
1$:             MOV     R0,-(SP)        ;SAVE LSN FOR LATER
                MOV     R0,R3           ;WE NEED 2 COPIES OF LSN FOR MAPPER
                MOV     R0,R4
                CLR     R0              ;INIT FOR TRACK QUOTIENT
                BR      3$              ;JUMP INTO DIVIDE LOOP
2$:             SUB     #23.,R3         ;PERFORM MAGIC TRACK DISPLACEMENT
3$:             INC     R0              ;BUMP QUOTIENT, STARTS AT TRACK 1
                SUB     #26.,R4         ;TRACK=INTEGER(LSN/26)
                BPL     2$              ;LOOP - R4=REM(LSN/26)-26
                CMP     #-14.,R4        ;SET C IF SECTOR MAPS TO 1-13
                ROL     R3              ;PERFORM 2:1 INTERLEAVE
4$:             SUB     #26.,R3         ;ADJUST SECTOR INTO RANGE -1,-26
                BPL     4$              ;(DIVIDE FOR REMAINDER ONLY)
                ADD     #27.,R3         ;NOW PUT SECTOR INTO RANGE 1-26
                BPT                     ;CALL READS SUBROUTINE
                MOV     (SP)+,R0        ;GET THE LSN AGAIN
                INC     R0              ;SET UP FOR NEXT LSN
                TST     R1              ;WHATS LEFT IN THE WORD COUNT
                BGT     1$              ;BRANCH TO TRANSFER ANOTHER SECTOR
                RTS     PC              ;RETURN
TRWAIT:         TST     @R4             ;NEW WAIT SUBROUTINE, PRINTS ERRORS
                BEQ     TRWAIT
                BPL     RTIRET          ;RETURN FROM INTERUPT
```

```
                                                ;*****  THIS MUST FALL INTO BIOERR  ******

                                                .ENDC


   000450  004067  000024          BIOERR: JSR     R0,REPORT         ;SAY THAT WE GOT ERROR
   000454     015     012     077          .ASCIZ  <15><12>\?B-I/O ERROR\<12>
   000457     102     055     111
   000462     057     117     040
   000465     105     122     122
   000470     117     122     012
   000473     000
                                           .EVEN
```

```
BOOT V02B-01    RT-11 BOOTSTRAP RT-11 MACRO VM02-09    8-APR-75 11:49:04 PAGE 10
BOOTSTRAP CORE DETERMINATION

                                   .SBTTL  BOOTSTRAP CORE DETERMINATION

   000474  112037  177566  REPOR1: MOVB    (R0)+,@#TPB       ;PUT ANOTHER CHARACTER OUT
   000500  105737  177564  REPORT: TSTB    @#TPS             ;WAIT FOR TYPER READY
   000504  100375                  BPL     REPORT            ; ...
   000506  105710                  TSTB    @R0               ;ANYTHING MORE ?
   000510  001371                  BNE     REPOR1            ;YES, LOOP
   000512  000005                  RESET                     ;STOP ALL DEVICES
   000514  000000                  HALT
   000516  000776                  BR      .-2               ;KEEP HIM FROM CONTINUING

   000520  012706  010000  BOOT:   MOV     #10000,SP         ;SET STACK POINTER
   000524  012700  000002          MOV     #2,R0             ;READ IN SECOND PART OF BOOT
   000530  012701  000400          MOV     #<BOOTSZ-1>*400,R1 ;EVERY BLOCK BUT THE ONE WE ARE IN
   000534  012702  001000          MOV     #1000,R2          ;INTO LOCATION 1000
   000540  004767  177636          JSR     PC,READ
                           .IIF GT .-1000, .ERROR            ;BOOTSTRAP BLOCK 0 TOO BIG
   000544  012703  000004          MOV     #4,R3             ;POINT TO TRAP LOCATIONS
   000550  011305                  MOV     @R3,R5            ;SAVE TRAP LOC
   000552  012723  000620          MOV     #NXM,(R3)+        ;SET TRAP FOR NON EXISTENT MEMORY
   000556  005013                  CLR     @R3

                           ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                           ; THIS BOOTSTRAP CAN SIMULATE ANY SIZE PDP-11.
                           ; IF LOCATION 'FIDDLE' IS A HALT, THE CPU WILL STOP DURING THE BOOT.
                           ; ON CONTINUE, THE TOP 5 BITS OF THE SWITCH REGISTER ARE USED TO
                           ; SET THE TOP OF AVAILABLE CORE AS A MULTIPLE OF 1K.
                           ; IF THE SR IS >= 160000 OR IF FIDDLE IS A   BR 1$ ,
                           ; THE BOOTSTRAP WILL DO A NORMAL CORE DETERMINATION.
                           ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
   000560  000407          FIDDLE: BR      1$                ;CHANGE TO HALT FOR FIDDLING
```

```
   000562  013702  177570          MOV     @#SR,R2         ;GET SWITCH VALUE
   000566  042702  003777          BIC     #3777,R2        ;ISOLATE TOP 5 BITS (1K INCREMENTS)
   000572  020227  160000          CMP     R2,#160000      ;SHOULD WE DO NORMAL CHECK ?
   000576  103410                  BLO     NXM             ;NO, USE THE SR VALUE
   000600  005002          1$:     CLR     R2              ;LOOK FOR TOP OF CORE
   000602  062702  004000  2$:     ADD     #4000,R2        ;MOVE TO NEXT 1K BANK
   000606  020227  160000          CMP     R2,#160000      ;REACHED 28K YET ?
   000612  001402                  BEQ     NXM             ;YES, DO A 28K SYSTEM
   000614  005712                  TST     @R2             ;NO, SEE IF THIS LOCATION EXISTS
   000616  000771                  BR      2$              ;KEEP GOING IF WE DIDN'T TRAP
   000620  012743  001476  NXM:    MOV     #BCLR,-(R3)     ;NONMEMORY TRAPS HERE
   000624  011367  177160          MOV     @R3,10          ;BAD INSTRUCTIONS TRAP HERE
   000630  012701  001604          MOV     #TSLIST,R1      ;BITS FOR CLEARING ON ERROR TRAPS
   000634  010100                  MOV     R1,R0
   000636  005737  177546          TST     @#LKCS          ;CHECK PRESENCE OF CLOCK
   000642  052140                  BIS     (R1)+,-(R0)     ;ADVANCE LIST
   000644  005737  172000          TST     @#GT40          ;CHECK FOR DISPLAY
   000650  052110                  BIS     (R1)+,@R0
   000652  170000                  CFCC                    ;CHECK FOR FPU
   000654  052110                  BIS     (R1)+,@R0
   000656  010523                  MOV     R5,(R3)+        ;RESTORE TRAP
   000660  012723  000340          MOV     #340,(R3)+      ;TRAP TO 4 IS PR7, CARRY OFF
   000664  010523                  MOV     R5,(R3)+        ;RESTORE 10
   000666  012723  000341          MOV     #341,(R3)+      ;TRAP TO 10 IS PR7, CARRY ON
   000672  162702  000000G         SUB     #RTSIZE,R2      ;R2 NOW POINTS TO WHERE WE WANT THE KMON
   000676  062702  000000G         ADD     #FILLER,R2      ;ABUT IT AGAINST THE TOP OF CORE
```

BOOT V02B-01   RT-11 BOOTSTRAP RT-11 MACRO VM02-09   8-APR-75 11:49:04 PAGE 10+
BOOTSTRAP CORE DETERMINATION

```
   000702  162702                  SUB     (PC)+,R2        ;RECOVER UNUSED CORE FROM SY:
           000704          SYSIZE = .
           000706          . = .+2                         ;SYSTEM HANDLER SIZE PUT HERE
   000706  062702  000000G         ADD     #MAXSYH,R2      ;THIS WAY BECAUSE NO GLOBAL ARITH.
   000712  020227  010000          CMP     R2,#10000       ;IS IT JUST TOO TINY ?
   000716  103466                  BLO     TOOSML          ;YES
   000720  010246                  MOV     R2,-(SP)        ;PUT LOAD ADDRESS ON STACK
   000722  012700  000001          MOV     #1,R0           ;NOW READ FIRST DIRECTORY BLOCK
   000726  006300          DFND:   ASL     R0
   000730  062700  000004          ADD     #4,R0           ;DIRECTORY STARTS AT 6
   000734  012701  001000          MOV     #1000,R1
   000740  012702  001634          MOV     #BUFFB,R2
   000744  004767  177432          JSR     PC,READ         ;READ THE SEGMENT
   000750  012701  001644          MOV     #BUFFB+10,R1    ;POINT TO START BLOCK WORD
   000754  012100                  MOV     (R1)+,R0
   000756  010102          MONF:   MOV     R1,R2           ;SAVE ADDRESS OF STATUS WORD
```

```
000760  032721  002000                  BIT     #PERM,(R1)+     ;IS IT A PERMANENT FILE?
000764  001411                          BEQ     1S              ;NO. WE ARE TRYING TO FIND THE
000766  162721                          SUB     (PC)+,(R1)+     ;FILE  MONITR.SYS, AS THAT IS
000770  051646                           .RAD50 /MON/           ;THE CURRENT MONITOR.
000772  162721                          SUB     (PC)+,(R1)+
000774  035562                           .RAD50 /ITR/
000776  162711                          SUB     (PC)+,(R1)
001000  075273                           .RAD50 /SYS/
001002  001002                          BNE     1S              ;LAST WAS NOT .SYS EXTENSION
001004  054141                          BIS     -(R1),-(R1)     ;BOTH MUST BE 0
001006  001447                          BEQ     MONFND          ;FOUND THE MONITOR
001010  032712  004000          1S:     BIT     #ENDBLK,(R2)    ;IS THIS ALL IN SEGMENT?
001014  001010                          BNE     2S              ;YES. READ NEXT, IF ANY.
001016  066200  000010                  ADD     10(R2),R0       ;INCREASE START BLOCK
001022  062702  000016                  ADD     #16,R2          ;GET TO NEXT ENTRY
001026  066702  000610                  ADD     BUFFB+6,R2
001032  010201                          MOV     R2,R1           ;POINT R1 TO NEXT
001034  000750                          BR      MONF
001036  016700  000574          2S:     MOV     BUFFB+2,R0      ;SEE IF NEXT IS AVAILABLE
001042  001331                          BNE     DFND            ;YES. CONTINUE
001044  004067  177430                  JSR     R0,REPORT       ;HE AIN'T GOT A MONITOR
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
                                        .EVEN
001074  004067  177400          TOOSML: JSR     R0,REPORT       ;HE IS IN A TINY MACHINE
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
                                        .EVEN
```

```
                                          .SBTTL  READ MONITOR, LOOKUP HANDLERS
  001126  011602                  MONFND: MOV     @SP,R2          ;RECALL LOAD LOCATION
  001130  062700  000002                  ADD     #BOOTSZ,R0      ;BUMP R0 OVER BOOT RECORDS
  001134  010046                          MOV     R0,-(SP)        ;SAVE SWAP BLOCK POINTER
  001136  012701  000000G                 MOV     #MAXSYH,R1      ;DO GLOBAL ARITHMETIC HERE
  001142  166701  177536                  SUB     SYSIZE,R1       ;R1 = MAXSYH-SYSIZE (BYTES)
  001146  062701  000000G                 ADD     #FILLER,R1      ;ADD AMOUNT OF EXTRA STUFF
  001152  006201                          ASR     R1              ;(WORDS)
  001154  005401                          NEG     R1              ;(TO SUBTRACT)
  001156  062701  000000G                 ADD     #RTLEN,R1       ;LENGTH TO LOAD (WORDS)
  001162  062700  000000G                 ADD     #SWAPSZ,R0      ;POINT TO BLOCK WITH KMON
  001166  004767  177210                  JSR     PC,READ         ;READ THE MONITOR INTO PLACE
  001172  012700  001502                  MOV     #RELLST,R0      ;POINT TO LIST OF THINGS TO RELOCATE
  001176  012601                          MOV     (SP)+,R1        ;R1 = SWAP BLOCK NUMBER
  001200  012604                          MOV     (SP)+,R4        ;R4 => KMON IN CORE
  001202  162704  000000G                 SUB     #KMON,R4        ;SUBTRACT LOCATION KMON WAS LINKED TO
  001206  010164  000000G                 MOV     R1,$SWPBL(R4)   ;R4 = BIAS. SET UP SWAP BLOCK #
  001212  062701  000000G                 ADD     #SWAPSZ,R1
  001216  062701  000000G                 ADD     #KMONSZ,R1
  001222  010164  000000G                 MOV     R1,$MONBL(R4)   ;SET USR BLOCK #
  001226  060430                  1$:     ADD     R4,@(R0)+       ;RELOCATE A POINTER IN THE ASECT
  001230  020027  001524                  CMP     R0,#RELST2      ;DONE YET ?
  001234  103774                          BLO     1$              ;NO
  001236  012005                          MOV     (R0)+,R5        ;GET POINTER TO THING IN MONITOR
  001240  060405                  2$:     ADD     R4,R5           ;BIAS THE POINTER
  001242  060415                          ADD     R4,@R5          ;NOW RELOCATE THE WORD
  001244  012005                          MOV     (R0)+,R5        ;GET NEXT POINTER
  001246  001374                          BNE     2$
  001250  013700  000054                  MOV     @#54,R0         ;POINT TO MONITOR

                                          .IF DF  $RKSYS!$DXSYS!$DPSYS!$DSSYS ;THE RK,RX,RP,RJS03/4 CAN BOOT FROM ANY UNIT

                                          .IF DF  $RKSYS          ;CODE FOR RK
                                          MOV     @#RKDA,R1       ;GET THE RK UNIT NUMBER
                                          ROL     R1
                                          ROL     R1
                                          ROL     R1
                                          ROL     R1
                                          BIC     #^C7,R1         ;EXTRACT IT
                                          .ENDC                   ;DF $RKSYS
                                          .IF DF  $DSSYS          ;CODE FOR RJS03/4
                                          MOV     @#RSCS2,R1      ;UNIT # INTO R1
                                          BIC     #^C7,R1         ;STRIP TO 3 BITS
                                          .ENDC
                                          .IF DF  $DPSYS          ;RP11
                                          MOV     @#RPCS,R1       ;GET CONTROLLER STATUS REG INTO R1
                                          BIC     #^C<CS.DRV>,R1  ;STRIP TO UNIT NUMBER
                                          SWAB    R1              ;UNIT # INTO BITS 2-0
```

```
                                        .ENDC                   ;DF SDPSYS

                                        .IF DF  SDXSYS          ;FLOPPY
                                        MOV     BTUNIT,R1       ;GET BOOTED UNIT (STORED BY BOOT2)
                                        .ENDC                   ;DF SDXSYS

                                        ADD     R1,DKASSG(R0)   ;FIX PERMANENT PSEUDO-ASSIGNMENTS
                                        ADD     R1,SYASSG(R0)
                                        MOVB    R1,SYUNIT+1(R0) ;SET UNIT NUMBER WE BOOTED
```



```

                                        .ENDC                   ;DF SRKSYS!SDXSYS!DPSYS
                                .ENABL  LSB

   001254  056760  000322  000300       BIS     BCNFG,CONFIG(R0) ;SET HARDWARE CONFIGURATION
   001262  005003                       CLR     R3              ;COUNT DEVICE SLOTS
   001264  012701  000000G              MOV     #SENTRY,R1      ;POINT TO SENTRY TABLE IN RMON
   001270  060401                       ADD     R4,R1
   001272  005721                 4$:   TST     (R1)+           ;RESIDENT DEVICE ?
   001274  001414                       BEQ     5$              ;NO, SKIP IT
   001276  060441                       ADD     R4,-(R1)        ;YES, FIX HANDLER POINTER
   001300  026127  000000G 070370       CMP     SPNAMO(R1),#SYNAME ;IS THIS THE SYSTEM DEVICE?
   001306  001006                       BNE     45$             ;NO
   001310  010360  000000G              MOV     R3,SYINDO(R0)   ;SET SYSTEM INDEX NUMBER
   001314  060360  000000G              ADD     R3,SYINDO(R0)   ;(DOUBLED)
   001320  011160  000000G              MOV     @R1,SYENTO(R0)  ;AND SET UP SYSTEM ENTRY POINTER
   001324  005721                 45$:  TST     (R1)+
   001326  005203                 5$:   INC     R3              ;ANY MORE ?
   001330  022703  000000G              CMP     #SSLOT,R3
   001334  001356                       BNE     4$
   001336  062700  000010               ADD     #SYBITO,R0      ;ADD IN OFFSET TO SYSTEM VECTOR IN MAP
   001342  152760  000014  000000G      BISB    #SYBITS,MAPOFF(R0) ;AND PROTECT IT
   001350  012701  000000G              MOV     #SPNAME,R1      ;POINT TO PERM NAME TABLE
   001354  060401                       ADD     R4,R1
   001356  062704  000000G              ADD     #SDVREC,R4      ;POINT R4 TO SDVREC IN RMON
   001362  012703  000000G              MOV     #SSLOT,R3       ;NUMBER TO LOOK UP
   001366  012167  000222         6$:   MOV     (R1)+,FNAME     ;FILL IN NAME IN LOOKUP
   001372                               .LOOKUP 0,#BLOOK        ;LOOKUP SY:HH.SYS
   001400  103002                       BCC     7$              ;GO IF THERE
   001402  005024                       CLR     (R4)+           ;CLEAR RECORD NUMBER
   001404  000406                       BR      8$
   001406                         7$:   .SAVEST 0,#CBLOK        ;SAVE STATUS OF THING
   001414  016714  000204               MOV     CBLOK+2,@R4     ;SET STARTING RECORD
   001420  005224                       INC     (R4)+           ;FIX IT
   001422  005303                 8$:   DEC     R3
```

```
001424  001360                          BNE     6$

001426  012737  100000  000044          MOV     #100000,@#JSW   ;NOTHING TO SWAP
001434                                  .PRINT  #BSTRNG         ;PRINT BOOT HEADER
001442  005000                          CLR     R0
001444  012720                          MOV     (PC)+,(R0)+
001446  040000                           BIC    R0,R0
001450  012720                          MOV     (PC)+,(R0)+
001452                                   .EXIT
001454  032767  000000G 000120          BIT     #KW11LS,BCNFG   ;AND IF HE HAS A CLOCK
001462  001403                          BEQ     10$             ;  WE TURN IT
001464  012737  000100  177546          MOV     #100,@#LKCS     ;    ON
001472  005000                  10$:    CLR     R0
001474                                  .EXIT
                                .DSABL  LSB

001476  005011                  BCLR:   CLR     @R1             ;TRAP MEANS THIS CONFIGURATION NYET
001500  000002                          RTI                     ;UNTRAP
```

BOOT V02B-01   RT-11 BOOTSTRAP RT-11 MACRO VM02-09   8-APR-75 11:49:04 PAGE 12
RELOCATION LIST

```
                                .SBTTL  RELOCATION LIST
001502  000004                  RELLST: 4                       ;ILLEGAL MEM AND INST. TRAPS
001504  000010                          10
001506  000030                          30                      ;EMT
001510  000054                          54                      ;ADDRESS OF RMON
001512  000060                          60                      ;TTY VECTORS
001514  000064                          64
001516  000100                          100                     ;CLOCK VECTOR
001520  000210                          SYVEC                   ;SYSTEM DEVICE VECTOR
001522  000244                          244                     ;LOCATION OF FPU TRAP
001524  000000G                 RELST2: USRLOC                  ;LOCATION OF USR NOW
001526  000000G                         $USRLC                  ;ADDRESS OF 'NORMAL' USR
001530  000000G                         QCOMP                   ;QUEUE COMPLETION
001532  000000G                         $KMLOC                  ;ADDRESS OF KMON
001534  000000G                         TTIBUF                  ;TTY RING BUFFER--INPUT
001536  000002G                         TTIBUF+2
001540  000006G                         TTIBUF+6
001542  000010G                         TTIBUF+10
001544  000000G                         TTOBUF                  ;TTY RING BUFFER--OUTPUT
001546  000004G                         TTOBUF+4
001550  000006G                         TTOBUF+6
001552  000000G                         SYSLOW                  ;LOWEST USED LOCATION
001554  000002G                         CORPTR+2                ;FREE CORE LIST
001556  000000G                         $INPTR                  ;POINTER TO $INTEN IN RESIDENT HANDLER
001560  000000G                         SYNCH                   ;SYNCHRONIZATION ADDRESS
```

```
                                     .IF NE BF
                                             MSGENT                   ;RELOCATE BF STUFF HERE
                                             TTIUSR
                                             TTOUSR
                                             FUDGE1
                                             FUDGE2
                                             BKGND1
                                             BKGND2
                                             BKGND3
                                             CNTXT
                                             BCNTXT
                                             RMONSP
                                             SWIPTR
                                             SWOPTR
                                             .SCRTN

                                     .IFF
   001562  000000G                           TRAPLC
   001564  000000G                           TRAPER                   ;LOCS FOR TRAPS TO 4/10
   001566  000000G                           FPPADD
   001570  000000G                           FPPIGN                   ;FPP SERVICE FOR MONITOR
   001572  000000G                           MONLOC                   ;WHERE USR WILL SIT
   001574  000000G                           I.CSW                    ;SINGLE USER STUFF HERE
   001576  000000G                           AVAIL                    ;MONITOR FREE Q POINTER
                                     .ENDC
   001600  000000                            0                        ;END OF LIST

   001602  000000                    BCNFG:  0                        ;BOOT CONFIGURATION WORD=DO NOT MOVE
   001604  000000G 000000G 000000G   TSLIST: .WORD   KW11LS,HWDSPS,HWFPUS ;BITS IN CONFIG WORD

                                     ; BLOOK IS THE ARGUMENT AREA FOR AN RT-11 LOOKUP.
   001612  075250                    BLOOK:  .RAD50  /SY /


BOOT V02B-01     RT-11 BOOTSTRAP RT-11 MACRO VM02-09   8-APR-75 11:49:04 PAGE 12+
RELOCATION LIST

   001614  000000  000000            FNAME:  .WORD   0,0              ;FILENAME GOES HERE
   001620  075273                            .RAD50  /SYS/
   001622                            CBLOK:  .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                  BOOTSZ  = . + 777 / 1000
      3         002000                  .        = BOOTSZ * 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         BIOERR  000450
BLOOK   001612        BOOT    000520          BOOTSZ= 000002          BOOT1   000034          BSTRNG= ****** G
BUFFB = 001634        CBLOK   001622          CONFIG= 000300          CORPTR= ****** G        DFND    000726
DKASSG= ****** G      ENDBLK= 004000          FIDDLE  000560          FILLER= ****** G        FNAME   001614
FPPADD= ****** G      FPPIGN= ****** G        GT40   = 172000         HWDSPS= ****** G        HWFPUS= ****** 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     =%000007         PERM   = 002000
QCOMP = ****** G      RCCA   = 177452         RCCS   = 177446         RCDA   = 177442         RCDB   = 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     =%000000
R1     =%000001       R2     =%000002         R3     =%000003         R4     =%000004         R5     =%000005
SP     =%000006       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          TPB    = 177566         TPS    = 177564         TRAPER= ****** G
TRAPLC= ****** G      TSLIST  001604          TTIBUF= ****** G        TTOBUF= ****** G        USRLOC= ****** G
USRSZ = ****** G      $DVREC= ****** G        $ENTRY= ****** G        $INPTR= ****** G        $KMLOC= ****** G
$MONBL= ****** G      $PNAME= ****** G        $PNAMO= ****** G        $RCSYS= 000001          $SLOT = ****** G
$SWPBL= ****** G      $USRLC= ****** G        ...V1 = 000001
. ABS.  002000      000
        000000      001
ERRORS DETECTED: 0
FREE CORE: 14985. WORDS

RCBTSJ,LP:/N:TTM/C=RCSYS,BSTRAP
```

A.3 LP/LS11 DEVICE HANDLER

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

                .TITLE LP   V02-03      25-JUN-74

                ; RT-11 LINE PRINTER (LP/LS11) HANDLER
                ;
                ; DEC-11-ORTLA-D
                ;
                ; RGB/FP/ABC/EF
                ;
                ; MARCH 1973/FEBRUARY 1974
                ;
                ; 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.
                ;
```

```
000000          R0=%0
000001          R1=%1
000002          R2=%2
000003          R3=%3
000004          R4=%4
000005          R5=%5
000006          SP=%6
000007          PC=%7

                ; LINE PRINTER CONTROL REGISTERS
177514          LPS     = 177514                ;LINE PRINTER CONTROL REGISTER
177516          LPB     = 177516                ;LINE PRINTER DATA BUFFER
000200          LPVEC   = 200                   ;LINE PRINTER VECTOR ADDR

                ;CONSTANTS FOR MONITOR COMMUNICATION
000001          HDERR   = 1                     ;HARD ERROR BIT
000054          MONLOW  = 54                    ;BASE ADDR OF MONITOR
000270          OFFSET  = 270                   ;POINTER TO Q MANAGER COMP ENTRY
000340          PR7     = 340
000200          PR4     = 200

                ; ASCII CONSTANTS

000015          CR      = 15
000012          LF      = 12
000014          FF      = 14
000011          HT      = 11

000204          COLSIZ  = 132.          ;132 COLS

                .GLOBL  COLCNT
```

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

             000000              (1)       .ASECT
             000400                        .       = 400

      000400 000036              (2)         .WORD   30.      ;MINIMUM WIDTH
      000402 110454  077100                  .RAD50  /WIDTH /
      000406 040025                          .WORD   <O.WIDTH-400>/2+40000 ;NO 'NO' OPTION, NUMBER REQUIRED

      000410 000240                          NOP              ;NO CR => NOP CROPT
      000412 012620  000000                  .RAD50  /CR    /
      000414 100033                          .WORD   <O.CR-400>/2+100000 ;ALLOW 'NO'

      000420 000240                          NOP              ;NO FORM0 => NOP FFOPT
      000422 023752  052760                  .RAD50  /FORM0 /
      000426 100040                          .WORD   <O.FORM0-400>/2+100000 ;ALLOW 'NO'

      000430 100500                          BMI     LPERR-ERROPT+. ;NO HANG BY GOING TO ERROR
      000432 031066  025700                  .RAD50  /HANG  /
      000436 100045                          .WORD   <O.HANG-400>/2+100000 ;ALLOW 'NO'

      000440 000040                          .WORD   40                ;FOR NO LC, CONVERT LC TO UC
      000442 045570  000000                  .RAD50  /LC    /
      000446 100052                          .WORD   <O.LC-400>/2+100000

      000450 000000              (3)         0       ;END OF LIST

      000452 010067  000462      (4)   O.WIDTH:MOV   R0,COLCNT         ;NEW WIDTH TO 2 CONSTANTS
      000456 010067  000532                    MOV   R0,RSTC+2
      000462 020003                            CMP   R0,R3             ;ERROR IF < 30.
      000464 000207                            RTS   PC

      000466 012703                    O.CR:   MOV   (PC)+,R3          ;NO 'NO', SO SET TO DO CR
      000470 001403                             BEQ  RSTC-CROPT+.
      000472 010367  000504                    MOV   R3,CROPT          ;SET CR OPTION
      000476 000207                            RTS   PC

      000500 012703                    O.FORM0:MOV   (PC)+,R3          ;SET TO DO FORMFEEDS ON BLOCK 0
      000502 001471                             BEQ  BLK0-FFOPT+.
      000504 010367  000350                    MOV   R3,FFOPT
      000510 000207                            RTS   PC

      000512 012703                    O.HANG: MOV   (PC)+,R3          ;SET TO HANG
      000514 100441                             BMI  RET-ERROPT+.
      000516 010367  000332                    MOV   R3,ERROPT
      000522 000207                            RTS   PC

      000524 005003                    O.LC:   CLR   R3                ;FOR 'LC', LEAVE LOWER CASE STUFF ALONE
      000526 000240                            NOP
      000530 010367  000400                    MOV   R3,LCOPT
      000534 000207                            RTS   PC
```

```
           001000             (5)  .=1000
                                   ; LOAD POINT
001000     000200                  LOADPT: .WORD   LPVEC           ;ADDR OF INTERRUPT VECTOR
001002     000034                          .WORD   LPINT-.         ;OFFSET TO INTERRUPT SERVICE
001004     000200                          .WORD   PR4             ;PRIORITY 7
001006     000000                  LPLQE:  .WORD   0               ;POINTER TO LAST Q ENTRY
001010     000000                  LPCQE:  .WORD   0               ;POINTER TO CURRENT Q ENTRY

                                   ; ENTRY POINT

001012     016704  177772          LP:     MOV     LPCQE,R4        ;R4 POINTS TO CURRENT Q ENTRY
001016     006364  000006                  ASL     6(R4)           ;WORD COUNT TO BYTE COUNT
001022     103115                          BCC     LPERR           ;A READ REQUEST IS ILLEGAL
001024     052737  000100  177514          BIS     #100,@#LPS      ;CAUSE AN INTERRUPT,STARTING TRANSFER
001032     000207                          RTS     PC

                                   ; INTERRUPT SERVICE

001034     000512                          BR      LPDONE          ;ABORT ENTRY POINT

001036     004577  000242          LPINT:  JSR     R5,@INTEN       ;INTO SYSTEM STATE
001042     000140                          .WORD   ^C<PR4>&PR7
001044     016704  177740                  MOV     LPCQE,R4        ;R4 -> CURRENT QUEUE ELEMENT
001050     005737  177514                  TST     @#LPS           ;ERROR CONDITION?
001054     100441                  ERROPT: BMI     RET             ;YES-HANG TILL CORRECTED
001056     005724                          TST     (R4)+           ;IS THIS BLOCK 0?
001060     001471                  FFOPT:  BEQ     BLK0            ;YES - OUTPUT INITIAL FORM FEED
001062     005724                          TST     (R4)+           ;AND POINT TO ADRS OF NEXT CHAR
001064     105737  177514          LPNEXT: TSTB    @#LPS           ;READY FOR ANOTHER CHAR YET?
001070     100033                          BPL     RET             ;NOPE - RETURN FROM INTERRUPT
001072     106327                          ASLB    (PC)+           ;TAB IN PROGRESS?
001074     000000                  TABFLG: .WORD   0
001076     001057                          BNE     TAB             ;BRANCH IF DOING TAB
001100     113405                  IGNORE: MOVB    @(R4)+,R5       ;GET NEXT CHAR (IF ANY)
001102     042705  177600                  BIC     #177600,R5      ;7-BIT
001106     005714                          TST     (R4)            ;ANY MORE CHARS?
001110     001464                          BEQ     LPDONE          ;NO:FINISHED
001112     005214                          INC     (R4)            ;YES, DECREMENT COUNT (IT WAS NEGATIVE)
001114     005244                          INC     -(R4)           ;BUMP BUFFER POINTER
001116     120527  000040                  CMPB    R5,#40          ;PRINTING CHAR?
001122     103417                          BLO     CHRTST          ;NO-GO TEST FOR SPECIAL CHAR.
001124     122705  000140                  CMPB    #140,R5         ;LOWER CASE?
001130     103002                          BHIS    PCHAR           ;NO
001132     162705                          SUB     (PC)+,R5        ;YES, CONVERT IF DESIRED
001134     000040                  LCOPT:  40
001136     005327                  PCHAR:  DEC     (PC)+           ;ANY ROOM LEFT ON LINE?
001140     000204             (6)  COLCNT: .WORD   COLSTZ          ;# OF PRINTER COLUMNS LEFT
001142     002756                          BLT     IGNORE          ;NO MORE ROOM ON LINE,DON'T PRINT CHAR
```

```
001144  106327                         ASLB    (PC)+            ;UPDATE TAB COUNT
001146  000001                 TABCNT: .WORD   1
001150  001423                         BEQ     RSTTAB           ;RESET TAB
001152  110537  177516         PC1:    MOVB    R5,@#LPB         ;PRINT THE CHAR
001156  000742                         BR      LPNEXT           ;TRY FOR NEXT CHAR
001160  000207                 RET:    RTS     PC
001162  120527  000011         CHRTST: CMPB    R5,#HT           ;IS CHAR A TAB?
001166  001420                         BEQ     TABSET           ;YES-RESET TAB
001170  120527  000012                 CMPB    R5,#LF           ;IS IT LF?
```

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

```
001174  001406                         BEQ     RSTC             ;YES-RESTORE COLUMN COUNT
001176  120527  000015                 CMPB    R5,#CR           ;IS IT CR?
001202  000240                 CROPT:  NOP                      ;IGNORE UNLESS MODIFIED
001204  120527  000014                 CMPB    R5,#FF           ;IS IT A FF?
001210  001333                         BNE     IGNORE           ;NO-CHAR IS NON-PRINTING
001212  012767  000204  177720 RSTC:   MOV     #COLSIZ,COLCNT   ;RE-INITIALIZE COLUMN COUNTER
001220  012767  000001  177720 RSTTAB: MOV     #1,TABCNT        ;RESET TAB COUNTER
001226  000751                         BR      PC1              ;PRINT THE CHAR
001230  016767  177712  177636 TABSET: MOV     TABCNT,TABFLG    ;SET UP TAB
001236  012705  000040         TAB:    MOV     #40,R5           ;PRINT SPACES
001242  000735                         BR      PCHAR

001244  005244                 BLK0:   INC     -(R4)            ;MAKE SURE WE ONLY COME HERE ONCE
001246  022424                         CMP     (R4)+,(R4)+      ;POINT TO ADRS OF NEXT CHAR
001250  012705  000014                 MOV     #FF,R5           ;PRINT INITIAL FF
001254  000756                         BR      RSTC

001256  052754  000001         LPERR:  BIS     #HDERR,@-(R4)    ;SET HARD ERROR BIT

                               ; OPERATION COMPLETE

001262  005037  177514         LPDONE: CLR     @#LPS            ;TURN OFF INTERRUPT
001266  010704                         MOV     PC,R4
001270  062704  177520                 ADD     #LPCQE-.,R4      ;ADDR OF CQE IN R3
001274  013705  000054                 MOV     @#MONLOW,R5
001300  000175  000270                 JMP     @OFFSET(R5)      ;JUMP TO Q MANAGER

001304  000000                 INTEN:  0

        000306                 LPSIZE  = .-LOADPT

        000001'                        .END
```

(7) — RSTC: MOV #COLSIZ,COLCNT

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

BLK0      001244           CHRTST   001162          COLCNT   001140 G        COLSTZ= 000204          CR     = 000015
CROPT     001202           ERROPT   001054          FF     = 000014          FFOPT    001060         HDERR  = 000001
HT      = 000011           IGNORE   001100          INTEN    001304          LCOPT    001134         LF     = 000012
LOADPT    001000           LP       001012          LPB    = 177516          LPCQF    001010         LPDONE   001262
LPERR     001256           LPINT    001036          LPLQF    001006          LPNEXT   001064         LPS    = 177514
LPSIZE= 000306             LPVEC  = 000200          MONLOW= 000054           OFFSET= 000270          O.CR     000466
O.FORM    000500           O.HANG   000512          O.LC     000524          O.WIDT   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         TAB      001236
TABCNT    001146           TABFLG   001074          TABSET   001230
. ABS.    001306     000
          000000     001
ERRORS DETECTED: 0
FREE CORE: 18070. WORDS

,LP:/N:TTM/C=LP
```

A.4 CR11 DEVICE HANDLER

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

                                        .TITLE  CR.SYS
                                ;       RT-11 CARD READER (CR11) HANDLER
                                ;
                                ;       DEC-11-OCRHA-D
                                ;
                                ;       ECP, ABC, RRB
                                ;       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

                                        .SBTTL  MISCELLANEOUS EQUATES

             000000             R0=%0
             000001             R1=%1
             000002             R2=%2
             000003             R3=%3
             000004             R4=%4
             000005             R5=%5
             000006             SP=%6
             000007             PC=%7

                                ; CARD READER CONTROL
```

```
000230          CRVECT=230          ;INTERRUPT VECTOR
177160          CRST=177160         ;CARD READER STATUS REGISTER
177162          CRB1=177162         ;DATA BUFFER 1
177164          CRB2=177164         ;DATA BUFFER 2

                ; CONSTANTS FOR MONITOR COMMUNICATIONS
000001          HDERR=1             ;HARD ERROR
000054          MONLOW=54           ;BASE ADDRESS OF MONITOR
000270          OFFSET=270          ;OFFSET TO HANDLER RETURN
177776          PS=177776           ;PROGRAM STATUS WORD
000340          PR7=340             ;PRIORITY 7
000300          PR6=300             ;PRIORITY 6

                ; ASCII CONSTANTS
000015          CR=15               ;CARRIAGE RETURN
000012          LF=12               ;LINE FEED
000040          SPACE=40            ;SPACE
000041          EOF=41              ;END OF FILE

                ; CARD READER CONTROL AND STATUS BITS
000001          READ=1              ;READ
000002          EJECT=2             ;EJECT CARD
000100          INTER=100           ;INTERRUPT ENABLE
000200          COLD=200            ;COLUMN DONE
000400          READY=400           ;READY
001000          BUSY=1000           ;BUSY
002000          ONLIN=2000          ;ONLINE
004000          DATLAT=4000         ;DATA LATE
010000          MOTIN=10000         ;MOTION CHECK(CM11 ONLY)
020000          HOPCK=20000         ;HOPPER CHECK(CM11 ONLY)
040000          CARDN=40000         ;CARD DONE
100000          ERR=100000          ;ERROR
```

```
                                        .SBTTL  CONFIGURATION SECTION

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

         000000                         .ASECT
         000400                 .=400                           ;CONFIGURATION AREA

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

000400   000404                         BR      .+LXCRLF-XCRLF
000402   012634  022600                 .RAD50  /CRLF/
000406   100025                         .WORD   <CRLF-400>/2+100000

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

000410   000403                         BR      .+LXTRIM-XTRIM
000412   077731  050500                 .RAD50  /TRIM/
000416   100032                         .WORD   <TRIM-400>/2+100000

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

000420   001151                         BNE     .+LERROR-XHANG
000422   031066  025700                 .RAD50  /HANG/
000426   100037                         .WORD   <HANG-400>/2+100000

                                ; SET CR CODE [=] 026 [029]
                                ; SET TRANSLATION TO 026 [029] MODE

000430   000032                         .WORD   026.
000432   012434  017500                 .RAD50  /CODE/
000436   040044                         .WORD   <CODE-400>/2+40000

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

000440   000022                         .WORD   NOIMAG-IMBASE
000442   035111  026210                 .RAD50  /IMAGE/
000446   100072                         .WORD   <IMAGE-400>/2+100000

000450   000000                         .WORD   0               ;END-OF-OPTIONS FLAG
```

```
                                        .SBTTL  CONFIGURATION SUBROUTINES

000452  012703                  CRLF:   MOV     (PC)+,R3        ;NOP IF POSITIVE
000454  000240                          NOP
000456  010367  000246'                 MOV     R3,XCRLF        ;ENTRY POINT FOR NO
000462  000207                          RTS     PC

000464  012703                  TRIM:   MOV     (PC)+,R3        ;NOP IF POSITIVE
000466  000240                          NOP
000470  010367  000236'                 MOV     R3,XTRIM        ;ENTRY POINT FOR NO
000474  000207                          RTS     PC

000476  012703                  HANG:   MOV     (PC)+,R3        ;NOP IF POSITIVE
000500  000240                          NOP
000502  010367  000036'                 MOV     R3,XHANG        ;ENTRY POINT FOR NO
000506  000207                          RTS     PC

000510  010701                  CODE:   MOV     PC,R1           ;R1 -> CONVERSION TABLE FOR 026
000512  062701  000122                  ADD     #SET026-.,R1
000516  160300                          SUB     R3,R0           ;026 REQUESTED?
000520  100407                          BMI     CODEXT          ;NOPE - ERROR (NOTE THAT C IS SET!)
000522  001407                          BEQ     SETCOD          ;YES, IT'S 026 - GO DO IT
000524  062701  000050                  ADD     #SET029-SET026,R1 ;R1 -> CONVERSION TABLE FOR 029
000530  022700  000003                  CMP     #3,R0           ;WAS IT 029?
000534  001402                          BEQ     SETCOD          ;YES
000536  000261                          SEC                     ;ELSE AN ERROR - INDICATE SUCH
000540  000207                  CODEXT: RTS     PC              ;AND RETURN TO KMON
000542  010703                  SETCOD: MOV     PC,R3           ;R3 -> CHARACTER TABLE
000544  062703  177716'                 ADD     #CHRTBL-.,R3
000550  005000                  SCODE:  CLR     R0              ;PICK UP NEXT OFFSET TO BE MODIFIED
000552  152100                          BISB    (R1)+,R0        ;  FROM APPROPRIATE TABLE
000554  001771                          BEQ     CODEXT          ;ALL DONE (NOTE: C IS CLEAR)
000556  060300                          ADD     R3,R0           ;POINT TO BYTE TO MODIFY
000560  112110                          MOVB    (R1)+,@R0       ;AND PLUG IN NEW VALUE
000562  000772                          BR      SCODE           ;CONTINUE

000564  062703  000010          TMAGE:  ADD     #YAIMAG-NOTMAG,R3 ;POINT TO "YES" TABLE
000570  060703                          ADD     PC,R3           ;ENTRY FOR "NO"
000572  012367  000136'         TMBASE: MOV     (R3)+,XIM1      ;AND PATCH HANDLER
000576  012367  000154'                 MOV     (R3)+,XIM2
000602  012367  000156'                 MOV     (R3)+,XIM2+2
000606  012367  000164'                 MOV     (R3)+,XIM3+2
000612  000207                          RTS     PC

000614  001402                  NOTMAG: BEQ     .+NXTCHR-XIM1
000616  116537  000306                  MOVB    CHRTBL-XIM2(R5),@(PC)+
000622  000001                          .WORD   1
000624  000402                  YAIMAG: BR      .+NXTCHR-XIM1
000626  013737  177162                  MOV     @#CRB1,@(PC)+
000632  000002                          .WORD   2
```

```
                                   .SBTTL  026, 029 CONVERSION TABLES

                              ; 026 CONVERSION TABLE
                              ; MODIFIES CHARACTER TABLE TO ACCEPT 026 KEYPUNCH CODES

  000634    012    137        SET026: .BYTE   012,137         ;BACK ARROW             (8-2)
  000636    013    075                .BYTE   013,075         ;EQUAL                  (8-3)
  000640    015    136                .BYTE   015,136         ;UP ARROW               (8-5)
  000642    016    047                .BYTE   016,047         ;APOSTROPHE             (8-6)
  000644    017    134                .BYTE   017,134         ;BACKSLASH              (8-7)
  000646    052    073                .BYTE   052,073         ;SEMICOLON              (0-8-2)
  000650    054    050                .BYTE   054,050         ;LEFT PAREN             (0-8-4)
  000652    055    042                .BYTE   055,042         ;QUOTES                 (0-8-5)
  000654    056    043                .BYTE   056,043         ;LB. SIGN               (0-8-6)
  000656    057    045                .BYTE   057,045         ;PERCENT                (0-8-7)
  000660    112    072                .BYTE   112,072         ;COLON                  (11-8-2)
  000662    115    133                .BYTE   115,133         ;L BRACKET              (11-8-5)
  000664    116    076                .BYTE   116,076         ;GREATER THAN           (11-8-6)
  000666    117    046                .BYTE   117,046         ;AMPERSAND              (11-8-7)
  000670    200    053                .BYTE   200,053         ;PLUS                   (12)
  000672    212    077                .BYTE   212,077         ;QUESTION               (12-8-2)
  000674    214    051                .BYTE   214,051         ;RIGHT PAREN            (12-8-4)
  000676    215    135                .BYTE   215,135         ;R BRACKET              (12-8-5)
  000700    216    074                .BYTE   216,074         ;LESS THAN              (12-8-6)
  000702    000000                    .WORD   0               ;** END OF TABLE **

                              ; 029 CONVERSION TABLE
                              ; MODIFIES CHARACTER TABLE TO ACCEPT 029 KEYPUNCH CODES

  000704    012    072        SET029: .BYTE   012,072         ;COLON                  (8-2)
  000706    013    043                .BYTE   013,043         ;LB. SIGN               (8-3)
  000710    015    047                .BYTE   015,047         ;APOSTROPHE             (8-5)
  000712    016    075                .BYTE   016,075         ;EQUAL                  (8-6)
  000714    017    042                .BYTE   017,042         ;QUOTES                 (8-7)
  000716    052    134                .BYTE   052,134         ;BACKSLASH              (0-8-2)
  000720    054    045                .BYTE   054,045         ;PERCENT                (0-8-4)
  000722    055    137                .BYTE   055,137         ;BACK ARROW             (0-8-5)
  000724    056    076                .BYTE   056,076         ;GREATER THAN           (0-8-6)
  000726    057    077                .BYTE   057,077         ;QUESTION               (0-8-7)
  000730    112    135                .BYTE   112,135         ;R BRACKET              (11-8-2)
  000732    115    051                .BYTE   115,051         ;RIGHT PAREN            (11-8-5)
  000734    116    073                .BYTE   116,073         ;SEMICOLON              (11-8-6)
  000736    117    136                .BYTE   117,136         ;UP ARROW               (11-8-7)
  000740    200    046                .BYTE   200,046         ;AMPERSAND              (12)
  000742    212    133                .BYTE   212,133         ;L BRACKET              (12-8-2)
  000744    214    074                .BYTE   214,074         ;LESS THAN              (12-8-4)
  000746    215    050                .BYTE   215,050         ;LEFT PAREN             (12-8-5)
  000750    216    053                .BYTE   216,053         ;PLUS                   (12-8-6)
  000752    000000                    .WORD   0               ;** END OF TABLE **

                              .IIF GE <CHRTBL-LOADPT+216>-1000,       .ERROR  ;TABLE NOT IN BLOCK 1
```

```
                                          .SBTTL  HANDLER PROPER

          000000'                         .CSECT  CR11

                                 ; LOAD POINT
 000000   000230                 LOADPT: .WORD    CRVECT          ;INTERRUPT VECTOR
 000002   000050                         .WORD    CRINT-.         ;OFFSET TO INTERRUPT SERVICE
 000004   000300                         .WORD    300             ;PS
 000006   000000                 CRLQE:  .WORD    0               ;LAST QUEUE ENTRY
 000010   000000                 CRCQE:  .WORD    0               ;CURRENT QUEUE ENTRY

                                 ; ENTRY POINT

 000012   016705  177772         CRHAND: MOV     CRCQE,R5         ;POINT TO Q ELEMENT
 000016   016704  000136                 MOV     CHRPTR,R4        ;POINT INTO CARD IMAGE
 000022   006365  000006                 ASL     6(R5)            ;CONVERT WORD COUNT TO BYTE COUNT
 000026   101555                         BLOS    LERROR           ;NULL REQUEST OR WRITE IS LOGIC ERR
 000030   032737  001400  177160         BIT     #READY+BUSY,@#CRST ;IS READER READY?
 000036   000240                 XHANG:  NOP                      ;* PATCH HERE TO ISSUE HARD ERROR
 000040   005725                         TST     (R5)+            ;BLOCK 0 REQUEST ?
 000042   001525                         BEQ     READR            ;YES, GO INITIATE REQUEST
 000044   005725                         TST     (R5)+            ;NO, POINT TO BUFFER PTRS
 000046   000514                         BR      CONT             ;GO SEE IF ANY STUFF IS LEFT IN OLD CARD

 000050   000567                         BR      ABORT            ;ABORT

                                 ; INTERRUPT ENTRY POINT

 000052   004577  001246         CRINT:  JSR     R5,@$INPTR       ;ENTER SYSTEM STATE
 000056   000040                         .WORD   ^C<PR6>&PR7
 000060   005367  000264                 DEC     COLCNT           ;COUNT DOWN INTERRUPTS THIS CARD
 000064   013705  177160                 MOV     @#CRST,R5        ;GET STATUS
 000070   100541                         BMI     ERROR            ;WHOOPS -- ERROR
 000072   105705                         TSTB    R5               ;CHECK FOR COLUMN DONE
 000074   100041                         BPL     CARD             ;BRANCH IF NOT COLUMN DONE
 000076   013705  177164                 MOV     @#CRB2,R5        ;GET COMPRESSED CHAR
 000102   001423                         BEQ     INCOLT           ;IT'S BLANK
 000104   013746  177162                 MOV     @#CRB1,-(SP)     ;GET EXPANDED CHAR
 000110   026767  000044  000072         CMP     CHRPTR,BUFPTR    ;FIRST COLUMN?
 000116   001002                         BNE     TSTPUN           ;NOPE
 000120   011667  000100                 MOV     @SP,CHAR12       ;ELSE SAVE FOR EOF CHECK
 000124   042716  177003         TSTPUN: BIC     #177003,@SP      ;CHECK ONLY ROWS 1-7
 000130   011646                         MOV     @SP,-(SP)        ;CHECK FOR INVALID PUNCHES
 000132   005416                         NEG     @SP              ;BY CHECKING FOR 2 OR MORE PUNCHES
 000134   042626                         BIC     (SP)+,(SP)+      ;IN COLUMNS 1-7
 000136   001402                 XIM1:   BEQ     NXTCHR           ;IT'S OKAY
 000140   012705  000377                 MOV     #377,R5          ;ELSE FORCE TRANSLATION INTO 134
 000144   016727  000010         NXTCHR: MOV     CHRPTR,(PC)+     ;REMEMBER POSITION OF NON-BLANK
```

```
000150  000000                    ENDPTR: .WORD   0
000152  060705                    INCOLT: ADD     PC,R5           ;MAKE A PIC POINTER TO TRANS TABLE
000154  116537  000306            XIM2:   MOVB    CHRTBL-.(R5),@(PC)+ ;PUT TRANSLATED CHAR IN LINE
000160  000000                    CHRPTR: .WORD   0
000162  062767  000001  177770    XIM3:   ADD     #1,CHRPTR       ;PUSH BUFFER POINTER BY 1 OR 2
000170  052737  000100  177160    INTRET: BIS     #INTEB,@#CRST   ;ENABLE ONLINE INTR IF NECESSARY
000176  000207                            RTS     PC
```

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

                                  ; CARD DONE OR ERROR

000200  032705  040000            CARD:   BIT     #CARDN,R5       ;CARD DONE?
000204  001440                            BEQ     ERR1            ;NOPE -- SPURIOUS INTERRUPT - IGNORE
000206  012704                            MOV     (PC)+,R4        ;R4 -> CHRBUF
000210  000000                    BUFPTR: .WORD   0               ;POINTER TO CHRBUF
000212  016705  177572                    MOV     CRCQE,R5        ;POINT TO Q ELEMENT
000216  022525                            CMP     (R5)+,(R5)+     ;PUSH OVER BUFFER POINTER
000220  022727  007417                    CMP     #7417,(PC)+     ;END OF FILE?
000224  000000                    CHAR12: .WORD   0               ;12-BIT FOR FIRST CARD COL.
000226  001471                            BEQ     ENDFIL          ;YES
000230  010146                            MOV     R1,-(SP)        ;SAVE R1
000232  016701  177722                    MOV     CHRPTR,R1       ;POINT TO PAST END OF 80 CHARS
000236  000240                    XTRIM:  NOP                     ;* PATCH HERE TO SUPPRESS TRIMMING
000240  016701  177704                    MOV     ENDPTR,R1       ;DA, GIVE IT TO HIM
000244  005201                            INC     R1
000246                            LXTRIM:
000246  000240                    XCRLF:  NOP                     ;* PATCH HERE TO SUPPRESS CR/LF
000250  112721  000015                    MOVB    #CR,(R1)+       ;JA, GIVE IT HIM
000254  112721  000012                    MOVB    #LF,(R1)+
000260  010167  177664            LXCRLF: MOV     R1,ENDPTR       ;THIS IS NOW THE END
000264  012601                            MOV     (SP)+,R1        ;RESTORE R1
000266  000404                            BR      CONT            ;ENTER FILLING LOOP
000270  112435                    FILBUF: MOVB    (R4)+,@(R5)+    ;PUT A BYTE IN HIS BUFFER
000272  005315                            DEC     @R5             ;IS HE FULL ?
000274  001457                            BEQ     RETMON          ;YEP
000276  005245                            INC     -(R5)           ;PUSH BUFFER POINTER
000300  026704  177644            CONT:   CMP     ENDPTR,R4       ;END OF OUR CARD ?
000304  101371                            BHI     FILBUF          ;NOT YET
000306  032737  001400  177160    ERR1:   BIT     #READY+BUSY,@#CRST ;OKAY TO INIT READ?
000314  001325                            BNE     INTRET          ;NOPE - GO HANG UNTIL READY
000316  010704                    READR:  MOV     PC,R4           ;POINT TO CHRBUF
000320  062704  000542                    ADD     #CHRBUF-.,R4
```

```
000324  010467  177630                        MOV     R4,CHRPTR          ;START FILLING FROM NEW CARD
000330  010467  177614                        MOV     R4,ENDPTR          ;WHICH IS AS YET EMPTY
000334  010467  177650                        MOV     R4,BUFPTR          ;ESTABLISH BUFFER POINTER
000340  005067  177660                        CLR     CHAR12             ;CLEAR EOF FLAG
000344  012727  000120                        MOV     #80.,(PC)+         ;SET COLUMN COUNT
000350  000000                        COLCNT: .WORD   0                  ;COUNT OF COLUMNS REMAINING IN CARD
000352  012737  000101  177160                MOV     #READ+INTER,@#CRST ;START A CARD GOIN'
000360  000207                                RTS     PC                 ;BYE

                                      ; VARIOUS ERRORS

000362  016705  177422                LERROR: MOV     CRCQE,R5           ;POINT TO QUEUE ELEMENT
000366  052755  000001                        BIS     #HDERR,@-(R5)      ;YOU CAN'T WRITE ON A READER
000372  000416                                BR      ABORT

000374  032705  004000                ERROR:  BIT     #DATLAT,R5         ;DATA LATE IS ONLY NOT CURABLE
000400  001370                                BNE     LERROR             ;ISSUE HARD ERROR IF SO
000402  005767  177742                        TST     COLCNT             ;DONE WITH DATA COLUMNS?
000406  100337                                BPL     ERR1               ;NOPE -- MUST BE PICK CHECK, ETC.,
                                                                         ;  START A NEW READ TO CORRECT CONDITION.
000410  000673                                BR      CARD               ;ELSE ASSUME CARD DONE

                                      ; END OF FILE CARD FOUND
```

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

```
000412  012504                        ENDFIL: MOV     (R5)+,R4           ;POINT INTO HIS BUFFER
000414  105024                        CLRBUF: CLRB    (R4)+              ;CLEAR IT ALL
000416  005315                                DEC     @R5
000420  001375                                BNE     CLRBUF
000422  052775  020000  177770                BIS     #20000,@-10(R5)    ;SET EOF BIT IN CHANNEL
000430  005067  177514                ABORT:  CLR     ENDPTR             ;FORCE A READ NEXT TIME

                                      ; RETURN TO MONITOR (REQUEST DONE, EOF, OR ERROR)

000434  010467  177520                RETMON: MOV     R4,CHRPTR          ;SAVE POSITION IN CARD
000440  005037  177160                        CLR     @#CRST             ;NO INTERRUPTS
000444  010704                                MOV     PC,R4              ;THE USUAL MONITOR RETURN
000446  062704  177342                        ADD     #CRCQE-.,R4
000452  013705  000054                        MOV     @#MONLOW,R5
000456  000175  000270                        JMP     @OFFSET(R5)
```

```
                                        .SBTTL  CHARACTER TABLE

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

                                .MACRO  C       $LIST
                                        T=0
                                .IRP    X,<$LIST>
                                .IF NE  X'.
                                .IF LE  X'.-7
                                        T=T+X'.
                                .IFF
                                        U=10
                                .REPT   X'.-8.
                                        U=U+U
                                .ENDR
                                        T=T+U
                                .ENDC
                                .IFF
                                        T=T+40
                                .ENDC
                                .ENDR
                                        .=CHRTBL+T
                                         .BYTE  $CHAR
                                        $CHAR = $CHAR + 1
                                .ENDM   C

                                ; THE FOLLOWING TABLE TRANSLATES 029 KEYPUNCH CODES TO ASCII.

000462                          CHRTBL:
        000400                  .REPT   256.
                                        .BYTE   134             ;DEC STANDARD ERROR CHARACTER
                                .ENDR

        000462'                         .=CHRTBL
```

```
        000000                  $CHAR = 0           ;START AT OCTAL 000

000462                   C      <12,0,9,8,1>        ;NULL
000754                   C      <12,9,1>            ;CTRL-A
000704                   C      <12,9,2>            ;CTRL-B
000705                   C      <12,9,3>            ;CTRL-C
000706                   C      <9,7>               ;CTRL-D
000512                   C      <0,9,8,5>           ;CTRL-E
000560                   C      <0,9,8,6>           ;CTRL-F
000561                   C      <0,9,8,7>           ;CTRL-G
000562                   C      <11,9,6>            ;CTRL-H
000611                   C      <12,9,5>            ;CTRL-I
000710                   C      <0,9,5>             ;CTRL-J
000550                   C      <12,9,8,3>          ;CTRL-K
000716                   C      <12,9,8,4>          ;CTRL-L
000717                   C      <12,9,8,5>          ;CTRL-M
000720                   C      <12,9,8,6>          ;CTRL-N
000721                   C      <12,9,8,7>          ;CTRL-O
000722                   C      <12,11,9,8,1>       ;CTRL-P
001014                   C      <11,9,1>            ;CTRL-Q
000604                   C      <11,9,2>            ;CTRL-R
000605                   C      <11,9,3>            ;CTRL-S
```

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

```
000606                   C      <9,8,4>             ;CTRL-T
000517                   C      <9,8,5>             ;CTRL-U
000520                   C      <9,2>               ;CTRL-V
000505                   C      <0,9,6>             ;CTRL-W
000551                   C      <11,9,8>            ;CTRL-X
000613                   C      <11,9,8,1>          ;CTRL-Y
000614                   C      <9,8,7>             ;CTRL-Z
000522                   C      <0,9,7>             ;ALTMODE (ESCAPE)
000552                   C      <11,9,8,4>          ;CTRL-\
000617                   C      <11,9,8,5>          ;CTRL-]
000620                   C      <11,9,8,6>          ;CTRL-^
000621                   C      <11,9,8,7>          ;CTRL-_
000622                   C      <>                  ;SPACE
000463                   C      <12,8,7>            ;!
000702                   C      <8,7>               ;"
000502                   C      <8,3>               ;#
000476                   C      <11,8,3>            ;$
000576                   C      <0,8,4>             ;%
000537                   C      <12>                ;&
000663                   C      <8,5>               ;'
000500                   C      <12,8,5>            ;(
000700                   C      <11,8,5>            ;)
000600                   C      <11,8,4>            ;*
```

```
 81 000577                     C       <12,8,6>        ;+
 82 000701                     C       <0,8,3>         ;,
 83 000536                     C       <11>            ;-
 84 000563                     C       <12,8,3>        ;.
 85 000676                     C       <0,1>           ;/
 86 000524                     C       <0>             ;0
 87 000523                     C       <1>             ;1
 88 000464                     C       <2>             ;2
 89 000465                     C       <3>             ;3
 90 000466                     C       <4>             ;4
 91 000467                     C       <5>             ;5
 92 000470                     C       <6>             ;6
 93 000471                     C       <7>             ;7
 94 000472                     C       <8>             ;8
 95 000473                     C       <9>             ;9
 96 000503                     C       <8,2>           ;:
 97 000475                     C       <11,8,6>        ;;
 98 000601                     C       <12,8,4>        ;<
 99 000677                     C       <8,6>           ;=
100 000501                     C       <0,8,6>         ;>
101 000541                     C       <0,8,7>         ;?
102 000542                     C       <8,4>           ;@
103 000477                     C       <12,1>          ;A
104 000664                     C       <12,2>          ;B
105 000665                     C       <12,3>          ;C
106 000666                     C       <12,4>          ;D
107 000667                     C       <12,5>          ;E
108 000670                     C       <12,6>          ;F
109 000671                     C       <12,7>          ;G
110 000672                     C       <12,8>          ;H
111 000673                     C       <12,9>          ;I
112 000703                     C       <11,1>          ;J
113 000564                     C       <11,2>          ;K
114 000565                     C       <11,3>          ;L
```



```
115 000566                     C       <11,4>          ;M
116 000567                     C       <11,5>          ;N
117 000570                     C       <11,6>          ;O
118 000571                     C       <11,7>          ;P
119 000572                     C       <11,8>          ;Q
120 000573                     C       <11,9>          ;R
121 000603                     C       <0,2>           ;S
122 000525                     C       <0,3>           ;T
```

```
000526                  C       <0,4>           ;U
000527                  C       <0,5>           ;V
000530                  C       <0,6>           ;W
000531                  C       <0,7>           ;X
000532                  C       <0,8>           ;Y
000533                  C       <0,9>           ;Z
000543                  C       <12,8,2>        ;[
000675                  C       <0,8,2>         ;\
000535                  C       <11,8,2>        ;]
000575                  C       <11,8,7>        ;^
000602                  C       <0,8,5>         ;_
000540                  C       <8,1>           ;ACCENT GRAVE
000474                  C       <12,0,1>        ;LC A
000724                  C       <12,0,2>        ;LC B
000725                  C       <12,0,3>        ;LC C
000726                  C       <12,0,4>        ;LC D
000727                  C       <12,0,5>        ;LC E
000730                  C       <12,0,6>        ;LC F
000731                  C       <12,0,7>        ;LC G
000732                  C       <12,0,8>        ;LC H
000733                  C       <12,0,9>        ;LC I
000743                  C       <12,11,1>       ;LC J
000764                  C       <12,11,2>       ;LC K
000765                  C       <12,11,3>       ;LC L
000766                  C       <12,11,4>       ;LC M
000767                  C       <12,11,5>       ;LC N
000770                  C       <12,11,6>       ;LC O
000771                  C       <12,11,7>       ;LC P
000772                  C       <12,11,8>       ;LC Q
000773                  C       <12,11,9>       ;LC R
001003                  C       <11,0,2>        ;LC S
000625                  C       <11,0,3>        ;LC T
000626                  C       <11,0,4>        ;LC U
000627                  C       <11,0,5>        ;LC V
000630                  C       <11,0,6>        ;LC W
000631                  C       <11,0,7>        ;LC X
000632                  C       <11,0,8>        ;LC Y
000633                  C       <11,0,9>        ;LC Z
000643                  C       <12,0>          ;OPEN BRACE
000723                  C       <12,11>         ;VERTICAL BAR
000763                  C       <11,0>          ;CLOSE BAR
000623                  C       <11,0,1>        ;TILDE
000624                  C       <12,9,7>        ;DEL

        001062'         . = CHRTBL + 256.

001062                  CHRBUF: .BLKW   81.
001324  000000          $INPTR: .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                    CRSIZE=.-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  BUFPTR  000210R     002  BUSY  = 001000            CARD    000200R     002  CARDN = 040000
CHAR12  000224R     002  CHRBUF  001062R     002  CHRPTR  000160R     002  CHRTBL  000462R     002  CLRBUF  000414R     002
CODE    000510           CODEXT  000540           COLCNT  000350R     002  COLD  = 000200            CONT    000300R     002
CR    = 000015           CRB1  = 177162           CRB2  = 177164           CRCQF   000010R     002  CRHAND  000012R     002
CRINT   000052R     002  CRLF    000452           CRLQF   000006R     002  CRSIZE= 001326            CRST  = 177160
CRVECT= 000230           DATLAT= 004000           EJECT = 000002           ENDFIL  000412R     002  ENDPTR  000150R     002
EOF   = 000041           ERR   = 100000           ERROR   000374R     002  ERR1    000306R     002  FILBUF  000270R     002
HANG    000476           HDERR = 000001           HOPCK = 020000           IMAGE   000564            IMBASE  000572
INCOLT  000152R     002  INTER = 000100           INTRET  000170R     002  LERROR  000362R     002  LF    = 000012
LOADPT  000000R     002  LXCRLF  000260R     002  LXTRIM  000246R     002  MONLOW= 000054            MOTIN = 010000
NOTMAG  000614           NXTCHR  000144R     002  OFFSET= 000270           ONLIN = 002000            PC    =%000007
PR6   = 000300           PR7   = 000340           PS    = 177776           READ  = 000001            READR   000316R     002
READY = 000400           RETMON  000434R     002  R0    =%000000           R1    =%000001            R2    =%000002
R3    =%000003           R4    =%000004           R5    =%000005           SCODE   000550            SETCOD  000542
SET026  000634           SET029  000704           SP    =%000006           SPACE = 000040            T     = 000227
TRIM    000464           TSTPUN  000124R     002  U     = 000020           XCRLF   000246R     002  XHANG   000036R     002
XIM1    000136R     002  XIM2    000154R     002  XIM3    000162R     002  XTRIM   000236R     002  XATMAG  000624
$CHAR = 000200           $INPTR  001324R     002
. ABS.  000754      000
        000000      001
CR11    001326      002
ERRORS DETECTED: 0
FREE CORE: 17698. WORDS

.LP:/N:TTM/C=CR
```

A.5 TC11 DEVICE HANDLER


```
        .TITLE  DT V02-07 12-APR-74

        ; RT-11 DECTAPE (TC11) HANDLER
        ;
        ; DEC-11-ORTDA-D
        ;
        ; RB/EF
        ;
        ; AUGUST, 1974
        ;
        ; 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.
        ;
```

```
        000000'                 .CSECT  SYSHND

                                .ENABL  LSB

        000000                  R0=%0
        000001                  R1=%1
        000002                  R2=%2
        000003                  R3=%3
        000004                  R4=%4
        000005                  R5=%5
        000006                  SP=%6
        000007                  PC=%7
        177776                  PS=177776

                                ; DECTAPE CONTROL REGISTERS
        177350                  TCDT    = 177350                ;DATA REGISTER
        177340                  TCST    = 177340                ;CONTROL AND STATUS REGISTER
        177342                  TCCM    = 177342                ;COMMAND REGISTER
        177344                  TCWC    = 177344                ;WORD COUNT REGISTER
        177346                  TCBA    = 177346                ;BUS ADDRESS REGISTER
        000214                  TCVEC   = 214                   ;TC11 INTERRUPT VECTOR

                                ;CONSTANTS FOR MONITOR COMMUNICATION
        000001                  HDERR   = 1                     ;HARD ERROR BIT
        000054                  MONLOW  = 54                    ;MONITOR BASE POINTER
        000270                  OFFSET  = 270                   ;POINTER TO Q MANAGER COMP ENTRY

                                .GLOBL  DTSYS,  RKSYS,  RFSYS,  DPSYS,  DSSYS,  DXSYS
                                .GLOBL  $INPTR, $INTEN, DTINT

        000000                  RKSYS   = 0                     ;RK IS NOT RESIDENT
        000000                  RFSYS   = 0                     ;NEITHER IS RF
        000000                  DXSYS   = 0
        000000                  DPSYS   = 0
        000000                  DSSYS   = 0
        000340                  PR7     = 340                   ;ENTERS AT LEVEL 7
```

```
  000000                           BEG:
                                   ; LOAD POINT
  000000  000214                           .WORD  TCVEC          ;ADDRESS OF INTERRUPT VECTOR
  000002  000032                           .WORD  DTINT-.        ;OFFSET TO INTERRUPT SERVICE
  000004  000340                           .WORD  PR7            ;PRIORITY 7
  000006                           DTSYS:
  000006  000000                   DTLQE:  .WORD  0              ;POINTER TO LAST Q ENTRY
  000010  000000                   DTCQE:  .WORD  0              ;POINTER TO CURRENT Q ENTRY

                                   ; ENTRY POINT
  000012  012727  000010                   MOV #8.,(PC)+         ;INIT THE RETRY COUNT
  000016  000000                   DTTRY:  .WORD  0              ;RETRY COUNTER
  000020  013746  177776                   MOV    @#PS,-(SP)     ;FAKE AN INTERRUPT
  000024  004767  000004                   JSR    PC,DTINT
  000030  000207                           RTS    PC             ;BACK TO MONITOR

                                   ;INTERRUPT SERVICE

  000032  000510                           BR     DTSTOP         ;ABORT CALL FROM BF SYSTEM
  000034  004577  000262           DTINT:  JSR    R5,@$INPTR     ;NOW JSR TO COMMON CODE
  000040  000040                           .WORD  ^C<300>&PR7    ;RUN AT LEVEL 6
  000042  010046                           MOV    R0,-(SP)
  000044  016700  177740                   MOV    DTCQE,R0       ;R0 POINTS TO Q ELEMENT
  000050  012704  177342                   MOV    #TCCM,R4       ;R4 POINTS TO CONTROL REGISTER
  000054  012046                           MOV    (R0)+,-(SP)    ;DESIRED BLOCK # ONTO STACK
  000056  012005                           MOV    (R0)+,R5       ;UNIT # INTO R5
  000060  042705  174377                   BIC    #^C<3400>,R5   ;ISOLATE UNIT NUMBER
  000064  032714  100100                   BIT    #100100,(R4)   ;ERROR BIT ON?
  000070  100452                           BMI    DTERR          ;YES
  000072  001502                           BEQ    RETRY          ;IF INTERRUPT IS OFF,WE ARE INITIATING
                                                                 ;A REQUEST
  000074  032714  000002                   BIT    #2,(R4)        ;SEARCHING?
  000100  001463                           BEQ    DTDONE         ;NO-A READ OR WRITE JUST COMPLETED
  000102  023767  177350  000206           CMP    @#TCDT,BWANT   ;COMPARE ACTUAL BLOCK TO DESIRED BLOCK
  000110  001422                           BEQ    BLKFND         ;FOUND IT
  000112  003407                   DIRECT: BLE    FORWARD        ;SEARCH IN THE FORWARD DIRECTION
  000114  052705  004000           REVERSE:BIS    #4000,R5       ;SET REVERSE BIT
  000120  162716  000002                   SUB    #2,(SP)        ;SEARCH FOR TWO BLOCKS BEFORE ONE
                                                                 ;ACTUALLY DESIRED (TO ALLOW
                                                                 ;SPACE FOR THE TURN-AROUND
  000124  000402                           BR     FORWARD        ;DON'T SET DELAY INHIBIT
  000126  052705  010000           FORW1:  BIS    #10000,R5      ;TAPE IS ALREADY MOVING FORWARD
                                                                 ;SO INHIBIT HARDWARE DELAY
  000132  052705  000103           FORWARD:BIS    #103,R5        ;INTERRUPT ENABLE,RNUM,AND GO
  000136  012667  000154                   MOV    (SP)+,BWANT    ;REMEMBER THE BLOCK WE ARE LOOKING FOR
  000142  010514                   RETRNI: MOV    R5,(R4)        ;TELL CONTROLLER TO GO
  000144  012600                           MOV    (SP)+,R0       ;RESTORE R0
```

```
000146  000207                              RTS     PC                  ;BACK INTO MONITOR

000150  032714  004000          FNDZR:  BIT     #4000,(R4)          ;WERE WE IN REVERSE?
000154  001757                          BEQ     REVERSE             ;NO-REVERSE TAPE
000156  032714  004000          BLKFND: BIT     #4000,(R4)          ;WERE WE GOING FORWARD?
000162  001363                          BNE     FORWARD             ;NO-WE HAVE TO TURN AROUND

                                ; INITIATE READ/WRITE REQUEST


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

000164  052705  010115                  BIS     #10115,R5           ;ASSUME WRITE
000170  012037  177346                  MOV     (R0)+,@#TCBA        ;CORE ADDRESS
000174  011016                          MOV     (R0),(SP)           ;WORD COUNT (OVER BLOCK #)
000176  100404                          BMI     1$                  ;WRITE WAS A GOOD GUESS
000200  001423                          BEQ     DTDONE              ;IF ZERO,SEEK
000202  005416                          NEG     (SP)                ;READ-NEGATE WORD COUNT
000204  042705  000010                  BIC     #10,R5              ;SET READ FUNCTION
000210  012637  177344          1$:     MOV     (SP)+,@#TCWC        ;SET WORD COUNT
000214  000752                          BR      RETRNI
                                ; ERROR ROUTINE

000216  032737  104000  177340  DTERR:  BIT     #104000,@#TCST      ;ENDZ ERROR?
000224  100003                          BPL     NOTEZ               ;NOT ENDZ
000226  032714  000002                  BIT     #2,(R4)             ;WERE WE SEARCHING?
000232  001346                          BNE     FNDZR               ;YES-REVERSE TAPE
000234  005367  177556          NOTEZ:  DEC     DTTRY               ;MORE TRIES LEFT?
000240  003017                          BGT     RETRY               ;YES
000242  052770  000001  177772          BIS     #HDERR,@-6(R0)      ;NO-SET HARD ERROR BIT

                                ; OPERATION FINISHED

000250  005726                  DTDONE: TST     (SP)+               ;POP BLOCK
000252  012600                          MOV     (SP)+,R0            ;RESTORE R0
000254  112737  000011  177342  DTSTOP: MOVB    #11,@#TCCM          ;STOP SELECTED DRIVE
000262  010704                          MOV     PC,R4
000264  062704  177524                  ADD     #DTCQE-.,R4                 ;ADDR OF CQE IN R4
000270  013705  000054                  MOV     @#MONLOW,R5
000274  000175  000270                  JMP     @OFFSET(R5)

                                ; RETRY CODE

000300  105737  177340          RETRY:  TSTB    @#TCST              ;TAPE UP TO SPEED?
000304  100710                          BMI     FORW1               ;YES-AVOID STOPPING TAPE
000306  062767  000004  000002          ADD     #4,BWANT            ;NO-IT TAKES 4 BLOCKS TO START AND STOP
000314  022716                          CMP     (PC)+,(SP)          ;MAKE AN ATTEMPT TO START IN THE
```

```
     000316  000000           RWANT:  0                          ;RIGHT DIRECTION BASED ON LAST BLOCK
     000320  000674                   BR      DIRECT             ;DESIRED

     000322  000000G          $INPTR: .WORD   $INTEN

             000324           DTSIZE  = .-REG           ;SIZE OF DT HANDLER

             000001'          .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  BLKEND  000156R   002  RWANT   000316R   002  DIRECT  000112R   002  DPSYS = 000000 G
DSSYS = 000000 G          DTCQE   000010R   002  DTDONE  000250R   002  DTERR   000216R   002  DTINT   000034RG   002
DTLQE     000006R    002  DTSIZE= 000324         DTSTOP  000254R   002  DTSYS   000006RG  002  DTTRY   000016R    002
DXSYS = 000000 G          ENDZR   000150R   002  FORWAR  000132R   002  FORW1   000126R   002  HDERR = 000001
MONLOW= 000054            NOTEZ   000234R   002  OFFSET= 000270         PC    =%000007          PR7   = 000340
PS    = 177776            RETRNI  000142R   002  RETRY   000300R   002  REVERS  000114R   002  RFSYS = 000000 G
RKSYS = 000000 G          R0    =%000000         R1    =%000001         R2    =%000002          R3    =%000003
R4    =%000004            R5    =%000005         SP    =%000006         TCBA  = 177346          TCCM  = 177342
TCDT  = 177350            TCST  = 177340         TCVEC = 000214         TCWC  = 177344          $INPTR  000322RG   002
$INTEN= ****** G
. ABS.    000000     000
          000000     001
SYSHND    000324     002
ERRORS DETECTED: 0
FREE CORE: 18078. 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.

```
        .TITLE  KB.MAC  V01-01
; RT-11 V2 DEVICE INDEPENDENT TERMINAL HANDLER, KB.
;
; DEC-11-ORKBA-D
;
; COPYRIGHT (C) 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.
;
;       MARCH 1975
;       RGB
```



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

```
;       TRANSFERS ONE LINE AT A TIME,NO MATTER HOW LONG THE
;       INPUT BUFFER IS FOR THE READ REQUEST; THE UNUSED PORTION
;       OF THE BUFFER IS ZERO-FILLED. CARRIAGE RETURN ECHOES
;       CARRIAGE RETURN,/LINE-FEED, AND INSERTS CR AND LF CHARACTERS
;       IN THE BUFFER IF THERE IS ROOM,ELSE ONLY CR IS PLACED IN THE BUFFER.
;     2)FORM FEED ECHOES 7 LINE FEEDS,AND INSERTS A FF CHARACTER IN
;       BUFFER.
;     3)RUBOUT ECHOES "\" AND DELETES THE LAST CHARACTER IN THE BUFFER.
;       IF THERE ARE NO CHARACTERS IN THE BUFFER,RUBOUT DOES NOT ECHO
;       AND IS IGNORED.
;     4)TAB ECHOES ENOUGH SPACES TO POSITION THE PRINT HEAD AT THE
;       NEXT TAB STOP,AND INSERTS A TAB CHARACTER IN THE BUFFER.
;     5)CTRL U ECHOES "^U" AND ERASES THE CURRENT LINE.
;     6)CTRL Z ECHOES "^Z" AND CAUSES THE HANDLER TO REPORT END-OF-FILE.
;       THE CTRL Z CHARACTER IS NOT INSERTED IN THE BUFFER.
;     7)THE LOW-SPEED READER WILL RUN IF IT IS TURNED ON WHILE A READ
;       REQUEST IS PENDING TO THE HANDLER. IF THE TAPE BEING READ HAS
;       MANY TABS,HOWEVER,THE TIME NECESSARY TO ECHO THE TABS WILL
;       CAUSE CHARACTERS FOLLOWING THE TABS TO BE LOST. TO DISABLE THE
;       ECHOING OF TABS,THE "SET" COMMAND CAN BE USED AS FOLLOWS:
;               "SET KB LSR" WILL DISABLE TAB ECHOING,ALLOWING A TAPE
;                           TO BE READ WITHOUT CHARACTER LOSS.
;               "SET KB NOLSR" WILL ENABLE TAB ECHOING,FOR NORMAL KEYBOARD
;                             INPUT. THIS IS THE DEFAULT.
;     8)WHEN THE HANDLER RECIEVES A READ REQUEST,A ">" CHARACTER IS
;       PRINTED IN THE LEFT MARGIN OF THE TERMINAL TO SIGNIFY THAT THE
;       HANDLER IS READY FOR INPUT.THIS CHARACTER CAN BE CHANGED, OR THE
;       PROMPT FEATURE CAN BE REMOVED, BY RE-ASSIGNING THE SYMBOL
;       "PROMPT" TO THE ASCII VALUE OF THE
;       DESIRED CHARACTER. SETTING PROMPT TO "0" WILL CAUSE NO CHARACTER
;       TO BE PRINTED.
;     9)IF NO READ REQUEST IS ACTIVE,THE HANDLER WILL NOT ACCEPT INPUT,
;       AND THE KEYBOARD WILL NOT ECHO. IF IT DOES ECHO,THE HANDLER IS
;       ACCEPTING INPUT.
;
;THIS HANDLER CONTAINS CONDITIONAL CODE TO SUPPORT TERMINALS THAT
;REQUIRE FILLER CHARACTERS AFTER A PARTICULAR CHARACTER. TO ENABLE THE
;FILLER FUNCTION, DEFINE THE SYMBOL "FILCHR" EQUAL TO THE ASCII
;VALUE FOR THE CHARACTER TO BE FILLED AFTER,AND THE SYMBOL "FILCNT"
;TO BE THE OCTAL NUMBER OF NULLS TO BE ISSUED AFTER EACH OCCURANCE
;OF THE CHARACTER DEFINED BY "FILCHR". FOR EXMAPLE,TO PROVIDE
;12 FILLER CHARACTERS AFTER A CARRIAGE RETURN, SET "FILCHR=15" AND
;"FILCNT=14".
```

```
;THE HANDLER IS INSTALLED VIA THE FOLLOWING PROCEDURE:
;       1)ASSEMBLE IT AS FOLLOWS:
;         DEFINE FILLER CONDITIONALS IF NECESSARY
;         .R MACRO
;         *KB=KB
;       2)LINK IT AS FOLLOWS:
;         .R LINK
;         *KB.SYS=KB
;       3)INSTALL IT AS DEVICE "KB:",AS DESCRIBED IN SECTION XXXXXX
;         OF THE RT-11 V2 SOFTWARE SUPPORT MANUAL. REMEMBER THAT
;         THE VECTORS FOR THE TERMINAL MUST BE PROTECTED IN THE BIT MAP
;         AS DESCRIBED IN THAT SECTION.
;         THE VALUES FOR THE VARIOUS TABLE ENTRIES SHOULD BE
;                 HSIZE=VALUE OF SYMBOL "KBSIZE" ON LAST LINE OF LISTING
;                 DVSIZE=0 (NON-FILE STRUCTURED DEVICE)
;                 PNAME=42420 (RAD50 FOR "KB ")
;                 STAT= HIGH ORDER BYTE=0,LOW ORDER BYTE=ANY DEVICE NUMBER
;                         AVAILABLE. NOTE THAT IT CANNOT BE 4. A VALUE >50
;                         IS RECOMMENDED.
;       4)ONCE INSTALLED, KB: WILL BE AVAILABLE WHEN THE SYSTEM IS REBOOTED.
;
;THE HANDLER ITSELF IS ACTIVATED WITH READ AND WRITE REQUESTS,AS ARE ALL
;RT-11 DEVICE HANDLERS. WHEN USING SYSTEM PROGRAMS WHICH OPERATE ON
;LARGE BUFFERS,SEVERAL LINES MAY ACCUMULATE IN THE BUFFER BEFORE
;THEY APPEAR ON THE TERMINAL,AND THEN ALL AT ONCE. TO AVOID THIS PROBLEM,
;EACH OUTPUT BUFFER CAN BE ZERO-FILLED AND SENT TO THE TERMINAL TO PRINT
;EACH LINE-THE HANDLER WILL IGNORE NULLS ON OUTPUT.
;IN FORTRAN,EACH LINE CAN BE FORCED IN OR OUT BY USING A REWIND
;FOLLOWING EACH READ OR WRITE TO THE DEVICE. FOR EXAMPLE:
;         LOGICAL*1 INPUTL(80)
;         CALL ASSIGN (7,'KB:/C')
;         WRITE (7,1)
;         REWIND 7
;         WRITE (7,2)
;         REWIND 7
;         READ (7,3) INPUTL
;         REWIND 7
;         .
;         .
;         .
;1        FORMAT . . . .
;2        FORMAT . . .
;3        FORMAT . . .

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

```
                                        .MCALL  .REGDEF,..V2..,.INTEN
  000000                                .REGDEF
  000000                                ..V2..

                                ;VECTOR AND DEVICE REGISTER ADDRESSES-EDIT THESE TWO TO RECONFIGURE
         000300                         KBVEC=300               ;KEYBOARD VECTOR
         176500                         KBCSR=176500            ;KEYBOARD CONTROL REGISTER

                                ;OTHER DEVICE ADDRESSES
         000304                         TPVEC=KBVEC+4           ;PRINTER VECTOR
         176502                         KBBUF=KBCSR+2           ;KEYBOARD BUFFER REGISTER
         176504                         TPCSR=KBCSR+4           ;PRINTER CONTROL REGISTER
         176506                         TPBUF=KBCSR+6           ;PRINTER BUFFER

                                ;CONSTANTS
         000270                         OFFSET=270              ;OFFSET TO MONITOR COMPLETION ENTRY

         020000                         EOF=20000               ;EOF BIT IN CSW
         000340                         PR7=340                 ;PSW VALUE FOR PRIORITY 7
         000200                         PR4=200                 ;PSW VALUE FOR PRIORITY 4

         000011                         HT=11                   ;TAB
         000012                         LF=12                   ;LINE FEED
         000014                         FF=14                   ;FORM FEED
         000015                         CR=15                   ;CARRIAGE RETURN=15
         000025                         CTRLU=25                ;CTRL/U
         000032                         CTRLZ=32                ;CTRL/Z
         000040                         SPACE=40                ;SPACE
         000177                         DELET=177               ;RUBOUT

         000024                         EBLENGTH=20.            ;LENGTH OF ECHO BUFFER

         000076                         PROMPT='>               ;PROMPT CHARACTER

                                ;SET LSR CODE
                                ;THE FOLLOWING IS THE HANDLER INTERFACE TO THE MONITOR SET COMMAND.
                                ;FOR DETAILS OF INTERFACING TO THE SET COMMAND,SEE THE RT-11 V2 SOFTWARE
                                ;SUPPORT MANUAL
         000000                         .ASECT
         000400                         .=400
  000400 000011                         HT                      ;FOR NOLSR,SET LSROPT TO "HT"
  000402 047012  000000                 .RAD50  /LSR   /
  000406 100005                         .WORD   <OPLSR-400>/2+100000
  000410 000000                         0
  000412 012703                 OPLSR:  MOV     (PC)+,R3        ;FOR LSR,SET LSROPT TO 377
  000414 000377                         377
  000416 010367  000640'                MOV     R3,LSROPT       ;MODIFY OPTION VARIABLE IN HANDLER
  000422 000207                         RTS     PC              ;RETURN TO SET PROCESSOR

         000000'                        .CSECT
```

```
                                        ;THIS IS THE HANDLER HEADER AREA,USED BY FETCH AND THE
                                        ;QUEUE MANAGER TO STORE VARIABLES CRITICAL TO HANDLER OPERATION.
000000  000304                          KBSTRT: .WORD   TPVEC           ;PRINTER VECTOR ADDRESS
000002  000116                                  .WORD   TPINT-.         ;OFFSET TO PRINTER INTERRUPT SERVICE
000004  000340                                  .WORD   PR7
000006  000000                          KBLQE:  .WORD   0               ;LAST QUEUE ENTRY
000010  000000                          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 (FETCH 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.
000012  010700                                  MOV     PC,R0
000014  062700  000246                          ADD     #KBINT-.,R0     ;CALCULATE ABSOLUTE ADDRESS OF KEYBOARD INTERRUPT SERVICE
000020  010037  000300                          MOV     R0,@#KBVEC      ;SET UP KEYBOARD VECTOR
000024  012737  000340  000302                  MOV     #PR7,@#KBVEC+2
000032  005067  000702                  RETRY:  CLR     READFL          ;INIT READ FLAG AND TAB COUNT
000036  016705  177746                          MOV     KBCQE,R5        ;POINT TO CURRENT Q ELEMENT
000042  022525                                  CMP     (R5)+,(R5)+     ;ADD 4 TO R5
000044  011567  000672                          MOV     (R5),UBPTR      ;SET UP POINTER TO USER BUFFER
000050  012567  000670                          MOV     (R5)+,UBPTR1    ;AND SAVE ORIGINAL POINTER FOR LATER
000054  006315                                  ASL     (R5)            ;MAKE WORD COUNT INTO BYTE COUNT
000056  011567  000654                          MOV     (R5),BYTCNT     ;AND SAVE IT
000062  001466                                  BEQ     DONE            ;WORD COUNT OF 0 IS SEEK,WHICH IS NOP IN THIS HANDLER
000064  103420                                  BCS     TPOUT2          ;IF NEGATIVE,WRITE TO PRINTER
000066  011504                                  MOV     (R5),R4         ;BYTE COUNT TO R4
000070  014505                                  MOV     -(R5),R5        ;USER BUFFER POINTER IN R5
000072  105025                          3$:     CLRB    (R5)+           ;ZERO USER BUFFER BEFORE STARTING TRANSFER
000074  005304                                  DEC     R4
000076  001375                                  BNE     3$              ;BRANCH IF NOT DONE
000100  105267  000634                          INCB    READFL          ;SET "READ IN PROGRESS" FLAG
000104  004167  000456                          JSR     R1,ECHO         ;PROMPT INPUT WITH ">"
000110     076     377                          .BYTE   PROMPT,377
000112  000167  000440                          JMP     KBIN            ;ENABLE KEYBOARD INTERRUPT AND RETURN
```

```
                                         ;THIS IS THE ABORT ENTRY POINT-THE HANDLER IS ENTERED AT THIS ADDRESS
                                         ;IF THE MONITOR RECIEVES A REQUEST TO ABORT ANY IO TRANSFER IN PROGRESS
000116  000446                                   BR      ABORT

                                         ;THIS IS THE TERMINAL OUTPUT INTERRUPT SERVICE. AFTER ENTERING SYSTEM STATE,
                                         ;IT DETERMINES IF THERE ARE ANY CHARACTERS IN THE ECHO BUFFER TO BE
                                         ;PRINTED. IF NOT,IT THEN DETERMINES WHETHER A WRITE REQUEST IS IN PROGRESS
                                         ;OR NOT. IF SO,THE NEXT CHARACTER IN THE USER BUFFER IS PRINTED.
                                         ;IF NOT,THE INTERRUPT IS DISMISSED.
                                         ;IF THERE ARE CHARACTERS IN THE ECHO BUFFER,THE FIRST CHARCTER IN THE
                                         ;LIST IS FETCHED INTO R4,THE LIST IN THE ECHO BUFFER IS "SLID UP"
                                         ;BY ONE CHARACTER,AND THE CHARACTER IN R4 IS THEN PRINTED.
                                         ;IF THE FILLER CONDITIONAL CODE IS INCLUDED AT ASSEMBLY TIME,
                                         ;THE CHARACTER IN R4 IS COMPARED AGAINST THE CHARACTER TO BE FILLED AFTER.
                                         ;IF THE SAME,A COUNT OF NECESSARY FILLS IS STUFFED IN "FILCN1" AND THE
                                         ;CHARACTER IS PRINTED. THE INTERRUPT SERVICE THEN CHECKS THE NUMBER
                                         ;OF FILLS NEEDED AS THE FIRST ITEM,AND PRINTS NULLS IF ANY ARE LEFT
000120  004577  000622                   TPINT:  JSR     R5,@$INPTR      ;ENTER SYSTEM STATE
000124  000140                                   .WORD   ^C<PR4>&PR7
000126  032737  000200  176504           TPOUT2: BIT     #200,@#TPCSR    ;IS THE PRINTER READY
000134  001436                                   BEQ     RTSPC           ;YES-THEN WAIT FOR INTERRUPT TO PRINT ANYTHING

                                                 .IFDF   FILCHR          ;CONDITIONAL CODE FOR FILLER
                                                 TSTB    FILCN1          ;ANY FILLS NEED TO BE OUTPUT?
                                                 BLE     3$              ;BRANCH IF NOT
                                                 DECB    FILCN1          ;YES-DECREASE NUMBER BY ONE
                                                 CLR     R4              ;NULL IS FILLER
                                                 BR      TPOUT3          ;GO PRINT IT
                                                 .ENDC
000136  010705                           3$:     MOV     PC,R5           ;CALC ABSOLUTE ADDRESS
000140  062705  000550                           ADD     #RBSTRT-.,R5    ;OF ECHO BUFFER
000144  111504                                   MOVB    (R5),R4         ;GET CHAR TO ECHO FROM ECHO BUFFER
000146  001410                                   BEQ     1$              ;BRANCH IF BUFFER EMPTY
000150  012746  000024                           MOV     #EBLENGTH,-(SP) ;NUMBER OF CHARS IN ECHO BUFFER ON STACK
000154  116525  000001                   2$:     MOVB    1(R5),(R5)+     ;SLIDE ECHO LIST UP
000160  005316                                   DEC     (SP)            ;DECREASE COUNT OF CHARS TO SLIDE
000162  003374                                   BGT     2$              ;BRANCH IF NOT FINISHED
000164  005726                                   TST     (SP)+           ;DONE-CLEAN UP STACK
000166  000412                                   BR      TPOUT1          ;AND PRINT CHAR

000170  105767  000544                   1$:     TSTB    READFL          ;ARE WE READING OR WRITING?
000174  001016                                   BNE     RTSPC           ;BRANCH IF READING
000176  117704  000540                   TPOUT:  MOVB    @UBPTR,R4       ;GET CHAR FROM USER BUFFER INTO R4
000202  005267  000534                           INC     UBPTR           ;BUMP BUFFER POINTER
000206  005267  000524                           INC     BYTCNT          ;AND DECREASE TRANSFER COUNT
000212  003012                                   BGT     DONE            ;BRANCH IF TRANSFER COMPLETE
000214  105704                           TPOUT1: TSTB    R4              ;DON'T PRINT NULLS
```

```
000216  001743                          BEQ     TPOUT2          ;BRANCH IF NULL
                                        .IFDF   FILCHR          ;CONDITIONAL CODE FOR FILLER
                                        CMPB    R4,FILCR:       ;DOES THIS CHAR NEED TO BE FILLED AFTER?
                                        BNE     TPOUT3          ;BRANCH IF NOT
                                        MOVB    #FILCNT,FILCN:  ;YES-SET UP COUNT OF FILLS NEEDED
                                        .ENDC
000220  012737  000100  176504  TPOUT3: MOV     #100,@#TPCSR    ;ENABLE PRINTER INTERRUPT
000226  110437  176506                  MOVB    R4,@#TPBUF      ;PRINT CHARACTER
000232  000207                  RTSPC:  RTS     PC              ;RETURN TO MONITOR
```



```
                                ;REQUEST TERMINATION AND ABORT CODE
                                ;THIS ROUTINE IS ENTERED WHEN THE I/O TRANSFER IS
                                ;COMPLETED OR ABORTED. THE DEVICE INTERRUPTS ARE DISABLED,AND
                                ;STANDARD MONITOR COMPLETION ENTRY CODE IS EXECUTED.
000234  005037  176504          ABORT:  CLR     @#TPCSR         ;DISABLE OUTPUT INTERRUPTS
000240  005037  176500          DONE:   CLR     @#KBCSR         ;DISABLE INPUT INTERRUPTS
000244  010704                          MOV     PC,R4           ;STANDARD MONITOR
000246  062704  177542                  ADD     #KBCQE-.,R4     ;COMPLETION ENTRY
000252  013705  000054                  MOV     @#54,R5         ;CODE
000256  000175  000270                  JMP     @OFFSET(R5)
```



```
                                ;KEYBOARD INTERRUPT SERVICE
                                ;THIS IS THE KEYBOARD INTERRUPT SERVICE ROUTINE. AFTER ENTERING
                                ;SYSTEM STATE,IT GETS THE TYPED CHARACTER INTO R4,THEN
                                ;PROCEEDS DOWN A CHAIN OF CHECKS FOR THE SPECIAL CASE CHARACTERS
                                ;(RUBOUT,CTRL U,CTRL Z,CR,FF). IF IT IS ONE OF THE SPECIAL
                                ;CHARACTERS, THE ROUTINE "ECHO" IS CALLED TO ECHO APPROPRIATE
                                ;CHARACTERS ON THE TERMINAL,THEN APPROPRITE ACTION FOR THE SPECIAL CASE
                                ;IS TAKEN. IF A NORMAL CHARACTER IS TYPED,IT IS ECHOED AND PLACED
                                ;IN THE USER BUFFER BEFORE THE INTERRUPT IS DISMISSED.
000262  004577  000460          KBINT:  JSR     R5,@SINPTR      ;ENTER SYSTEM STATE
000266  000140                          .WORD   "C<PR4>&PR7
000270  113704  176502                  MOVB    @#KBBUF,R4      ;GET CHAR
000274  042704  177600                  BIC     #177600,R4      ;STRIP TO SEVEN BITS
000300  120427  000177                  CMPB    R4,#DELET       ;IS THIS CHARACTER A RUBOUT?
000304  001020                          BNE     11$             ;BRANCH IF NOT
000306  026767  000430  000430          CMP     UBPTR,UBPTR1    ;ANY CHARS LEFT TO RUB OUT?
```

```
000314  001520                          BEQ     KBIN            ;NO-IGNORE RUBOUT
000316  005367  000420                  DEC     UBPTR           ;BACK UP POINTER INTO USER BUFFER
000322  004167  000240                  JSR     R1,ECHO
000326     134     377                  .BYTE   '\,377
000330  105267  000405                  INCB    TABCNT          ;BUMP TAB COUNTER FOR "\"
000334  105077  000402                  CLRB    @UBPTR          ;ZERO RUBBED OUT CHAR
000340  005267  000372                  INC     BYTCNT          ;AND INCREASE TRANSFER COUNT TO REFLECT LOST CHAR
000344  000504                          BR      KBIN            ;RE-ENABLE INTERRUPTS AND EXIT
000346  120427  000014          11$:    CMPB    R4,#FF          ;IS THIS CHAR A FORM FEED?
000352  001006                          BNE     6$              ;BRANCH IF NOT
000354  004167  000206                  JSR     R1,ECHO         ;YES-ECHO 7 LINE FEEDS
000360     012     012     012          .BYTE   LF,LF,LF,LF,LF,LF,LF,377
000363     012     012     012
000366     012     377
000370  120427  000015          6$:     CMPB    R4,#CR          ;IS THIS CHAR A CR?
000374  001017                          BNE     7$              ;BRANCH IF NOT
000376  004167  000164                  JSR     R1,ECHO         ;YES-ECHO CR,LF
000402     015     012     000          .BYTE   CR,LF,0,377
000405     377
000406  110477  000330                  MOVB    R4,@UBPTR       ;PUT CR IN USER BUFFER
000412  005267  000324                  INC     UBPTR           ;BUMP USER BUFFER POINTER
000416  005367  000314                  DEC     BYTCNT          ;ROOM IN BUFFER FOR LF TOO?
000422  001706                          BEQ     DONE            ;DON'T INSERT IT IF NOT
000424  112777  000012  000310          MOVB    #LF,@UBPTR      ;ELSE ADD LF TO BUFFER
000432  000702                          BR      DONE
```

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

```
000434  120427  000025          7$:     CMPB    R4,#CTRLU       ;IS CHAR CTRL U?
000440  001007                          BNE     8$              ;BRANCH IF NOT
000442  004167  000120                  JSR     R1,ECHO         ;ECHO "^U"
000446     136     125     015          .BYTE   '^,'U,CR,LF,0,377
000451     012     000     377
000454  000167  177352                  JMP     RETRY           ;AND RESTART READ

000460  120427  000032          8$:     CMPB    R4,#CTRLZ       ;IS CHAR CTRL Z?
000464  001013                          BNE     9$              ;BRANCH IF NOT
000466  004167  000074                  JSR     R1,ECHO         ;ECHO "^Z"
000472     136     132     015          .BYTE   '^,'Z,CR,LF,0,377
000475     012     000     377
000500  016705  177304                  MOV     KBCQE,R5        ;POINT R5 TO Q ELEMENT
000504  052775  020000  177776          BIS     #EOF,@-2(R5)    ;AND SET EOF FLAG IN CSW
000512  000652                          BR      DONE            ;STOP TRANSFER

000514  120427  000040          9$:     CMPB    R4,#40          ;IS THIS A PRINTING CHAR?
000520  002402                          BLT     21$             ;BRANCH IF NOT
000522  105267  000213                  INCB    TABCNT          ;YES-INCREASE TAB POSITION
```

```
000526  110467  000004          21$:    MOVB    R4,20$          ;SET UP TO ECHO CHAR
000532  004167  000030                  JSR     R1,ECHO
000536     000     377          20$:    .BYTE   0,377
000540  110477  000176                  MOVB    R4,@UBPTR       ;PUT CHAR IN USER BUFFER
000544  005267  000172                  INC     UBPTR           ;BUMP BUFFER POINTER
000550  005367  000162                  DEC     BYTCNT          ;ANY MORE TO TRANSFER
000554  001631                          BEQ     DONE            ;BRANCH IF NOT
000556  012737  000101  176500  KBIN:   MOV     #101,@#KBCSR    ;ENABLE KEYBOARD INTERRUPT
000564  000207                          RTS     PC              ;RETURN TO MONITOR
```



```
                                ;SUBROUTINE ECHO
                                ;THIS SUBROUTINE SERVES TO PLACE THE SPECIFIED CHARACTERS IN THE
                                ;ECHO BUFFER,AND START THE PRINTER IN CASE IT IS IDLE.
                                ;THE CALLING SEQUENCE IS
                                ;       JSR     R1,ECHO
                                ;       .BYTE   CHAR1,CHAR2,CHAR3,...,CHARN,377
                                ;ON ENTRY,R4 CONTAINS THE CHAR TYPED AT THE KEYBOARD.
                                ;NOTE THAT THERE MUST BE AN EVEN NUMBER OF BYTES IN THE ARGUMENT LIST
                                ;AND THEREFORE THE NUMBER OF CHARACTERS EXCLUDING THE 377
                                ;MUST BE ODD.
                                ;WHEN ENTERED,ECHO SCANS THE ECHO BUFFER TO FIND THE END OF THE
                                ;ECHO LIST,WHICH IS MARKED BY A NULL BYTE. WHEN THE END OF THE LIST
                                ;IS FOUND,IT IS DETERMINED IF THERE ARE AT LEAST 8 FREE SLOTS IN THE LIST
                                ;TO ACCOMODATE A POSSBLE LINE FEED OR FORM FEED. IF NOT,THE
                                ;CHARACTER JUST TYPED IS IGNORED. IF SO,THE CHARACTERS FROM THE
                                ;ARGUMENT LIST FOLLOWING THE CALL ARE INSERTED IN THE BUFFER,
                                ;THE PRINTER IS STARTED IF IT IS IDLE,AND THE ROUTINE RETURNS.
                                ;NOTE THAT TAB IS A SPECIAL CASE; IF R4 CONTAINS A TAB CHARACTER
                                ;WHEN THIS ROUTINE IS ENTERED,THE ARGUMENT LIST IS NOT USED. RATHER,
                                ;AN APPROPRIATE NUMBER OF SPACES TO MOVE THE PRINT HEAD TO THE
                                ;NEXT TAB STOP ARE PLACED IN THE ECHO BUFFER,AND THE ROUTINE RETURNS
                                        .ENABL  LSB
000566  010705                  ECHO:   MOV     PC,R5           ;CALC ABSOLUTE ADDRESS
000570  062705  000120                  ADD     #EBSTRT-.,R5    ;OF ECHO BUFFER
000574  010567  000134                  MOV     R5,TEMP         ;SAVE ADDRESS OF ECHO BUFFER
000600  062767  000023  000126          ADD     #EBLENGTH-1,TEMP;TEMP POINTS TO END OF ECHO BUFFER
000606  105725                  4$:     TSTB    (R5)+           ;IS THIS END OF ECHO LIST?
000610  001376                          BNE     4$              ;BRANCH IF NOT
000612  005305                          DEC     R5              ;YES-R5 POINTS TO FIRST FREE SLOT IN ECHO LIST
000614  160567  000114                  SUB     R5,TEMP         ;FIND NUMBER OF FREE SLOTS IN ECHO LIST
000620  026727  000110  000010          CMP     TEMP,#8.        ;IS THERE ENOUGH ROOM TO ECHO TAB OR FF?
000626  003002                          BGT     3$              ;BRANCH IF YES
000630  012601                          MOV     (SP)+,R1        ;NO-IGNORE THIS CHAR THEN
000632  000751                          BR      KBIN            ;DISMISS INTERRUPT
```

```
000634  010446                  3$:     MOV     R4,-(SP)        ;SAVE CHAR
000636  120427                          CMPB    R4,(PC)+        ;IS THIS CHAR A TAB?
000640  000011                  LSROPT: HT                      ;THIS COMPARE OPERAND CAN BE CHANGED BY SET LSR
000642  001013                          BNE     1$              ;BRANCH IF NOT
000644  112725  000040          5$:     MOVB    #SPACE,(R5)+    ;ECHO A SPACE
000650  105267  000065                  INCB    TABCNT          ;BUMP POSITION COUNTER
000654  132767  000007  000057          BITB    #7,TABCNT       ;AT TAB STOP YET?
000662  001370                          BNE     5$              ;BRANCH IF NOT
000664  005721                          TST     (R1)+           ;YES-ARTIFICIALLY BUMP RETURN
000666  105015                          CLRB    (R5)            ;END ECHO LIST
000670  000403                          BR      6$              ;AND START ECHO
000672  112125                  1$:     MOVB    (R1)+,(R5)+     ;MOVE CHAR INTO ECHO LIST
000674  100376                          BPL     1$              ;BRANCH IF END-OF-LIST NOT SEEN
000676  105045                          CLRB    -(R5)           ;ELSE USE 0 TO MARK END OF ECHO LIST
000700  004767  177222          6$:     JSR     PC,TPOUT2       ;PRINT A CHAR TO START PRINTER
000704  012604                          MOV     (SP)+,R4        ;RESTORE CHAR
000706  000201                          RTS     R1              ;RETURN
                                        .DSABL  LSB

```

KB.MAC V01-01 RT-11 MACRO VM02-09   8-APR-75 12:33:51 PAGE 10

```
                                ;DATA AREA

                                ;ECHO RING BUFFER-EBLENGTH CHARACTERS LONG
000710     000                  RBSTRT: .BYTE   0
                                        .BLKB   EBLENGTH-1

                                ;VARIABLE AREA

                                        .IFDF   FILCHR          ;FILLER CONDITIONAL
                                FILCR1: .BYTE   FILCHR          ;CHARACTER TO BE FILLED AFTER
                                FILCN1: .BYTE   0               ;NUMBER OF FILLS REMAINING
                                        .ENDC
000734  000000                  TEMP:   .WORD   0               ;TEMPORARY
000736  000000                  BYTCNT: .WORD   0               ;USER TRANSFER COUNT
000740     000                  READFL: .BYTE   0               ;FLAG FOR "READ IN PROGRESS"
000741     000                  TABCNT: .BYTE   0               ;TAB POSITION COUNTER
000742  000000                  UBPTR:  .WORD   0               ;POINTER INTO USER BUFFER
000744  000000                  UBPTR1: .WORD   0               ;POINTER TO START OF USER BUFFER

                                ;MONITOR SYSTEM STATE ENTRY LINK
000746  000000                  SINPTR: .WORD   0

        000750                  KBSIZE=.-KBSTRT
        000001'                         .END
```

```
KB.MAC  V01-01  RT-11 MACRO VM02-09   8-APR-75 12:33:51 PAGE 10+
SYMBOL TABLE

ABORT   000234R       BYTCNT  000736R       CR     = 000015       CTRLU = 000025       CTRLZ = 000032
DELET = 000177        DONE    000240R       EBLENG= 000024        ECHO    000566R      EOF   = 020000
FF    = 000014        HT    = 000011        KBBUF = 176502        KBCQE   000010R      KBCSR = 176500
KBIN    000556R       KBINT   000262R       KBLQE   000006R       KBSIZE= 000750       KBSTRT  000000R
KBVEC = 000300        LF    = 000012        LSROPT  000640R       OFFSET= 000270       OPLSR   000412
PC    =%000007        PROMPT= 000076        PR4   = 000200        PR7   = 000340       RBSTRT  000710R
READFL  000740R       RETRY   000032R       RTSPC   000232R       R0    =%000000       R1    =%000001
R2    =%000002        R3    =%000003        R4    =%000004        R5    =%000005       SP    =%000006
SPACE = 000040        TABCNT  000741R       TEMP    000734R       TPBUF = 176506       TPCSR = 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,LP:/N:TTM/C=KB
```

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	.CLOSE
.CSISPC	.DELETE
.DATE	.ENTER
.DSTAT	.LOOKUP
.EXIT	.READ
.FETCH	.READC

(continued on next page)

Table C-1 (Cont.)
V1 Programmed Requests

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

The formats of V1-specific requests (those listed in the right column) follow. Definitions of arguments used in these macro calls are:

.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).

.crtn — 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 transferred to or from the buffer during an I/O operation.

```
.CLOSE .chan

.DELETE .chan,.dblk

.ENTER .chan,.dblk,.length

.LOOKUP .chan,.dblk

.READ  )
.READC }    .chan,.buff,.wcnt,.crtn,.blk   [ .crtn is required ]
.READW )                                   [ only for .READC   ]

.RENAME .chan,.dblk

.REOPEN .chan,.cblk

.SAVESTATUS .chan,.cblk
```

```
.WAIT .chan

.WRITE )
.WRITC }    .chan,.buff,.wcnt,.crtn,.blk     [ .crtn 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 LSPOOL.MAC.

2. Next assemble it under MACRO and then link it to create the REL format output file:

```
.R MACRO
*LSPOOL=LSPOOL

.R LINK
*LSPOOL=LSPOOL/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 LSPOOL<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.

```
                                        .TITLE  LSPOOL - LINE PRINTER SPOOLER
                                        .SBTTL  A USEFUL FOREGROUND PROGRAM

                                ; THIS PROGRAM FOR THE FOREGROUND IS A LINE PRINTER SPOOLER.
                                ; IT SEARCHES A SPECIFIED DEVICE FOR FILES WITH A PARTICULAR
                                ; EXTENSION (THE DEFAULT IS LPT) AND PRINTS THEM, DELETING
                                ; AFTER PRINTING. IF NONE ARE FOUND, IT WILL GO TO SLEEP FOR
                                ; HALF A MINUTE, PERMITTING THE BACKGROUND TO RUN.
                                ;
                                ; TO RUN LSPOOL, FIRST LOAD LP HANDLER AND INPUT DEVICE HANDLER
                                ; IF IT IS NOT THE SYSTEM DEVICE TYPE.
                                ;
                                ; E.G.,
                                ;
                                ; .LOA LP,RF
                                ; .FRU LSPOOL
                                ;
                                ; LSPOOL WILL TYPE: "DEVICE TO SPOOL?"
                                ; TYPE INPUT DEVICE AND FILE DESCRIPTION, E.G.:
                                ;
                                ; RF:*.LST


                                        .MCALL  ..V2..,.REGDEF
                                        .MCALL  .READW,.WRITW,.LOOKUP,.DELETE,.CSISPC,.TTYIN
                                        .MCALL  .PRINT,.TTYOUT,.SRESET,.RCTRLO,.CLOSE,.EXIT
                                        .MCALL  .DSTATUS,.TWAIT

 000000                                 ..V2..
 000000                                 .REGDEF

         000046                 USRSWP  =       46              ;USR SWAP LOCATION POINTER
         000052                 ERRBYT  =       52              ;ERROR CODE
         000015                 CR      =       15              ;CARRIAGE RETURN
         000012                 LF      =       12              ;LINE FEED

 000000  012737  001260' 000046 START:  MOV     #BUFF,@#USRSWP  ;MAKE USR SWAP OVER BUFF
 000006  010627                         MOV     SP,(PC)+        ;SAVE STACK POINTER FOR RESET
 000010  000000                 STKSAV: .WORD   0
 000012                                 .DSTATUS #IOB,#LP       ;MUST BE IN MEMORY.
 000024  103403                         BCS     1$              ;ILLEGAL DEVICE
 000026  005767  000750                 TST     IOB+4           ;TEST ENTRY POINT
 000032  001011                         BNE     BEGIN           ;BR TO BEGIN IF LOADED
 000034                         1$:     .PRINT  #MSG0           ;LP NOT IN MEMORY!
```

```
000042                          .EXIT                       ;BACK TO USER FOR A LOAD LP

                        ; COME HERE ON BAD COMMAND STRING
000044                  BADCOM: .PRINT  #MSG2               ;PRINT ERROR MESSAGE
000052  016706  177732          MOV     STKSAV,SP           ;RESET STACK, FALL THRU TO BEGIN

000056                  BEGIN:  .CLOSE  #0                  ;WE WILL USE CH 0, SO CLEAN IT UP
000070                          .RCTRLO                     ;RESET CTRL/O FLAG SO
000072                          .PRINT  #MSG1               ;PROMPTING MSG WILL PRINT.
000100  012702  001142'         MOV     #CSIBLK,R2          ;POINT TO COMMAND STRING BUFFER
000104  010201                  MOV     R2,R1               ;COPY THE POINTER AND INPUT COMMAND
000106                  1$:     .TTYIN                      ;A CHARACTER AT A TIME.
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

000116  001773                  BEQ     1$                  ;YES, IGNORE IT.
000120  110022                  MOVB    R0,(R2)+            ;MOVE IT INTO BUFFER AND
000122  122700  000012          CMPB    #LF,R0              ;TEST FOR END OF LINE.
000126  001367                  BNE     1$                  ;NO, GET ANOTHER CHARACTER.
000130  105042                  CLRB    -(R2)               ;YES, CLEAR OUT THE LINE FEED
000132                          .CSISPC R1,#DEFEXT,R1       ;PROCESS THE COMMAND
000144  012600                  MOV     (SP)+,R0            ;TEST # OF SWITCHES;C BIT UNCHANGED.
000146  001336                  BNE     BADCOM              ;NO SWITCHES ALLOWED
000150  103735                  BCS     BADCOM              ;SYNTAX ERROR
000152                          .LOOKUP #IOB,#1,#LP
000204  016700  000776          MOV     CSIBLK+36.,R0       ;GET FILE EXTENSION TO PRINT.
000210  001002                  BNE     3$                  ;BRANCH IF USER SPECIFIED,
000212  012700                  MOV     (PC)+,R0            ;ELSE USE LPT EXTENSION
000214  046624                  .RAD50  /LPT/
000216  010067  000244  3$:     MOV     R0,LPT...           ;SAVE THE EXTENSION FOR LATER.
000222  016700  000752          MOV     CSIBLK+30.,R0       ;GET THE INPUT DEVICE NAME
000226  001002                  BNE     4$                  ;BRANCH IF USER SPECIFIED,
000230  012700                  MOV     (PC)+,R0            ;ELSE USE THE
000232  015326                  .RAD50  /DK0/               ;DEFAULT DEVICE.
000234  010011          4$:     MOV     R0,@R1              ;SET DEVICE NAME IN FILE
000236  005061  000002          CLR     2(R1)               ;DESCRIPTOR BLOCK, CLEARING
                                                            ;OUT ANY FILE NAME.
000242                          .DSTATUS #IOB,R1            ;INPUT DEVICE HANDLER MUST BE RESIDENT
000252  103403                  BCS     5$                  ;ILLEGAL DEVICE
000254  005767  000522          TST     IOB+4               ;TEST ENTRY POINT
000260  001004                  BNE     6$                  ;BRANCH IF O.K.,
000262                  5$:     .PRINT  #MSG3               ;ELSE PRINT MESSAGE
000270  000672                  BR      BEGIN
```

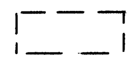
```
000272                          6$:     .LOOKUP #IOB,#0,R1          ;OPEN CHANNEL TO READ DIRECTORY
000320  103467                          BCS     BADERR
                                .ENABL  LSB
000322  012702  001260'         FINDLP: MOV     #BUFF,R2            ;R2 => BUFFER
000326  012703  000001                  MOV     #1,R3               ;INIT R3
000332  006303                  1$:     ASL     R3                  ;MULTIPLY BY 2
000334  062703  000004                  ADD     #4,R3               ;AND ADD 4 TO GET BLOCK NUMBER
                                                                    ;OF DIRECTORY SEGMENT (STARTING
                                                                    ;AT BLOCK 6 OF DEVICE).
000340                                  .READW  #IOB,#0,R2,#1000,R3 ;READ A DIRECTORY SEGMENT
000402  103436                          BCS     BADERR              ;ERROR READING DIRECTORY!!
000404  010205                          MOV     R2,R5               ;COPY POINTER
000406  062705  000012                  ADD     #12,R5              ;MOVE PAST DIRECTORY HEADER
000412  032715  004000          2$:     BIT     #4000,@R5           ;TEST FOR END OF SEGMENT
000416  001416                          BEQ     3$                  ;BRANCH IF MORE TO GO
000420  016203  000002                  MOV     2(R2),R3            ;GET LINK TO NEXT SEGMENT
000424  001342                          BNE     1$                  ;BRANCH IF ANOTHER SEGMENT EXISTS,
000426                                  .TWAIT  #IOB,#TIMBLK        ;ELSE WAIT A WHILE
000452  000723                          BR      FINDLP              ;WE'RE AWAKE, LOOK FOR A FILE
000454  032725  002000          3$:     BIT     #2000,(R5)+         ;TEMPORARY FILE?
000460  001404                          BEQ     4$                  ;YES, SKIP IT
000462  026527  000004                  CMP     4(R5),(PC)+         ;DOES THE EXTENSION MATCH?
000466  000000                  LPT...: .WORD   0
000470  001407                          BEQ     COPIER              ;YES, GO PRINT IT.
000472  062705  000014          4$:     ADD     #14,R5              ;NO, ADVANCE TO NEXT ENTRY
000476  000745                          BR      2$                  ;AND GO LOOK AT IT.
000500                          BADERR: .PRINT  #MSG4               ;PRINT A MESSAGE
000506  000711                          BR      1$                  ;THEN TRY NEXT SEGMENT
                                .DSABL  LSB
```

LSPOOL - LINE PRINTER SPOOLER   RT-11 MACRO VM02-11   26-NOV-75 00:07:21 PAGE 1+
A USEFUL FOREGROUND PROGRAM

```
                                ; THIS ROUTINE PRINTS THE SPOOLED FILE JUST FOUND AND THEN DELETES IT.

000510  012561  000002          COPIER: MOV     (R5)+,2(R1)         ;COPY FILE NAME AND EXTENSION
000514  012561  000004                  MOV     (R5)+,4(R1)         ;INTO FILE DESCRIPTOR BLOCK
000520  012561  000006                  MOV     (R5)+,6(R1)         ;FOR A LOOKUP.
000524                                  .LOOKUP #IOB,#2,R1          ;LOOKUP THE FILE ON CHANNEL 2.
000554  103662                          BCS     FINDLP              ;SOMETHING FUNNY HAPPENED
000556  005005                          CLR     R5                  ;O.K., COPY IT, R5 IS BLOCK #
000560                          1$:     .READW  #IOB,#2,R2,#1000,R5 ;READ 1000 WORDS
000624  103424                          BCS     2$                  ;ERROR ON READ
000626  010004                          MOV     R0,R4   ;COPY ACTUAL WORD COUNT TRANSFERRED
```

```
000630                                   .WRITW  #IOB,#1,R2,R4,R5 ;AND WRITE IT TO CHANNEL 1
000672  005725                           TST     (R5)+            ;BUMP BLOCK # BY 2
000674  000731                           BR      1$               ;CONTINUE UNTIL EOF.
000676  105737  000052           2$:     TSTB    @#ERRBYT         ;WAS ERROR JUST AN EOF?
000702  001412                           BEQ     4$               ;YES.
000704                                   .CLOSE  #2               ;NO, CLOSE THE FILE AND
000716                                   .PRINT  #ERRIN           ;REPORT AN INPUT ERROR
000724  000167  177372           3$:     JMP     FINDLP           ;THEN FIND ANOTHER FILE.
000730                           4$:     .CLOSE  #2               ;ON EOF, CLOSE THE FILE
000742                                   .DELETE #IOB,#3,R1       ;MUST DELETE USING AN INACTIVE CHANNEL
000772  000754                           BR      3$               ;THEN CONTINUE

000774  046636                   LP:     .RAD50  /LP0/            ;SPOOLER OUTPUT DEVICE
000776                           IOB:    .BLKW   5                ;EMT ARGUMENT BLOCK
        .NLIST  BIN
001010  MSG0:   .ASCIZ  /NO LP/
001016  MSG1:   .ASCIZ  /DEVICE TO SPOOL?/
001037  MSG2:   .ASCIZ  /TRY AGAIN/
001051  MSG3:   .ASCIZ  /DEVICE?/
001061  MSG4:   .ASCIZ  /ERROR READING DIRECTORY/
001111  ERRIN:  .ASCIZ  /INPUT ERROR/
        .EVEN
                                         .LIST   BIN
001126  000000  001604           TIMBLK: .WORD   0,60.*15.
001132  046624                   DEFEXT: .RAD50  /LPT/            ;CSI DEFAULT EXTENSIONS
001134  000000  000000  000000           .WORD   0,0,0
001142                           CSTBLK: .BLKW   39.              ;CSI WORK AREA
001260                           BUFF:   .BLKB   10000            ;BUFFER AREA
        000000'                          .END    START

LSPOOL - LINE PRINTER SPOOLER   RT-11 MACRO VM02-11   26-NOV-75 00:07:21 PAGE 1+
SYMBOL TABLE

BADCOM   000044R     BADERR   000500R     BEGIN    000056R     BUFF     001260R     COPIER   000510R
CR     = 000015      CSTBLK   001142R     DEFEXT   001132R     ERRBYT=  000052      ERRIN    001111R
FINDLP   000322R     IOB      000776R     LF     = 000012      LP       000774R     LPT...   000466R
MSG0     001010R     MSG1     001016R     MSG2     001037R     MSG3     001051R     MSG4     001061R
PC     =%000007      R0     =%000000      R1     =%000001      R2     =%000002      R3     =%000003
R4     =%000004      R5     =%000005      SP     =%000006      START    000000R     STKSAV   000010R
TIMBLK   001126R     USRSWP=  000046      ...V2  = 000001
. ABS.   000000      000
         011260      001
ERRORS DETECTED: 0
FREE CORE: 15035. WORDS

,LP:/N:TTM/C=LSPOOL.MAC
```

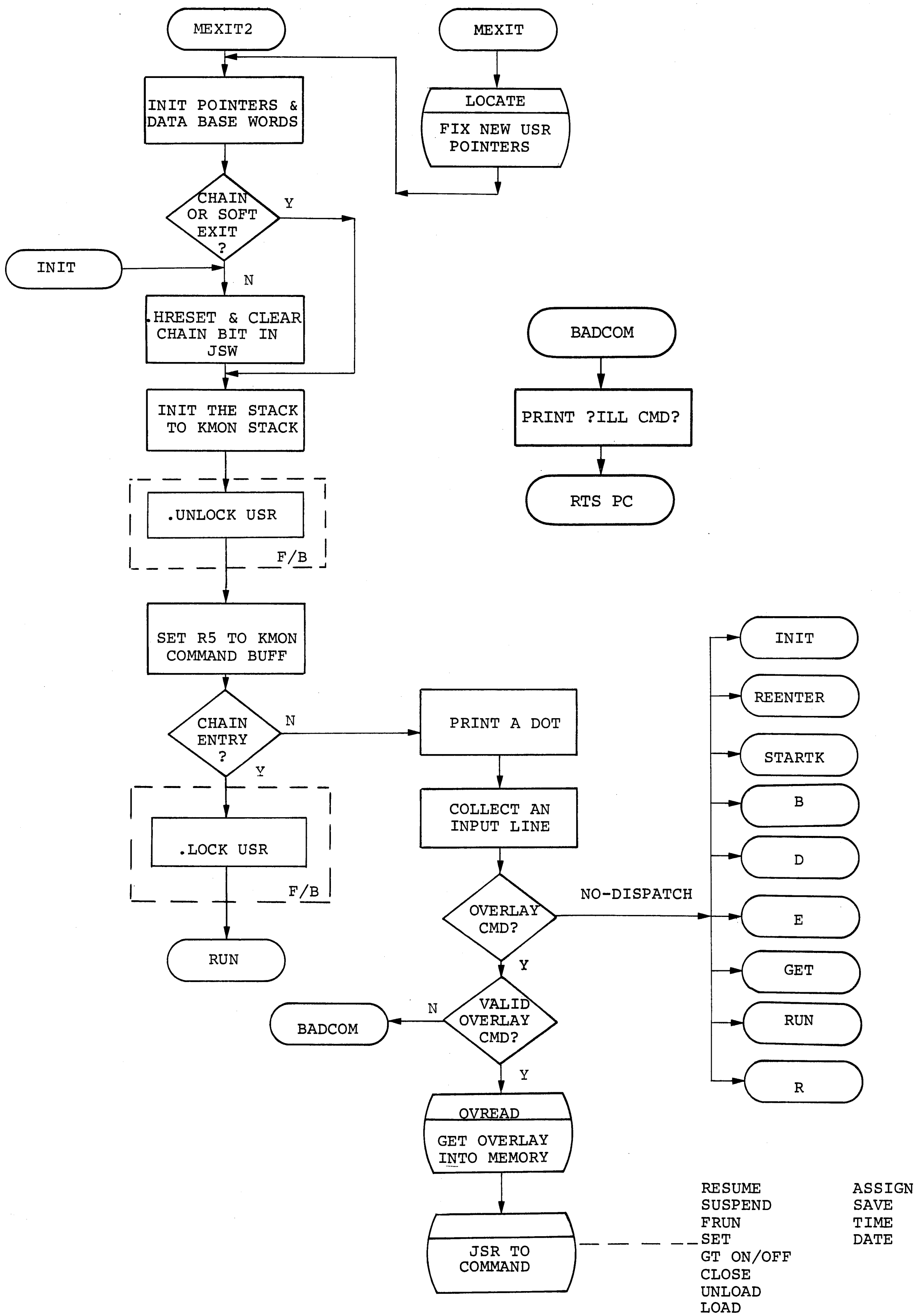
APPENDIX E
S/J AND F/B MONITOR FLOWCHARTS

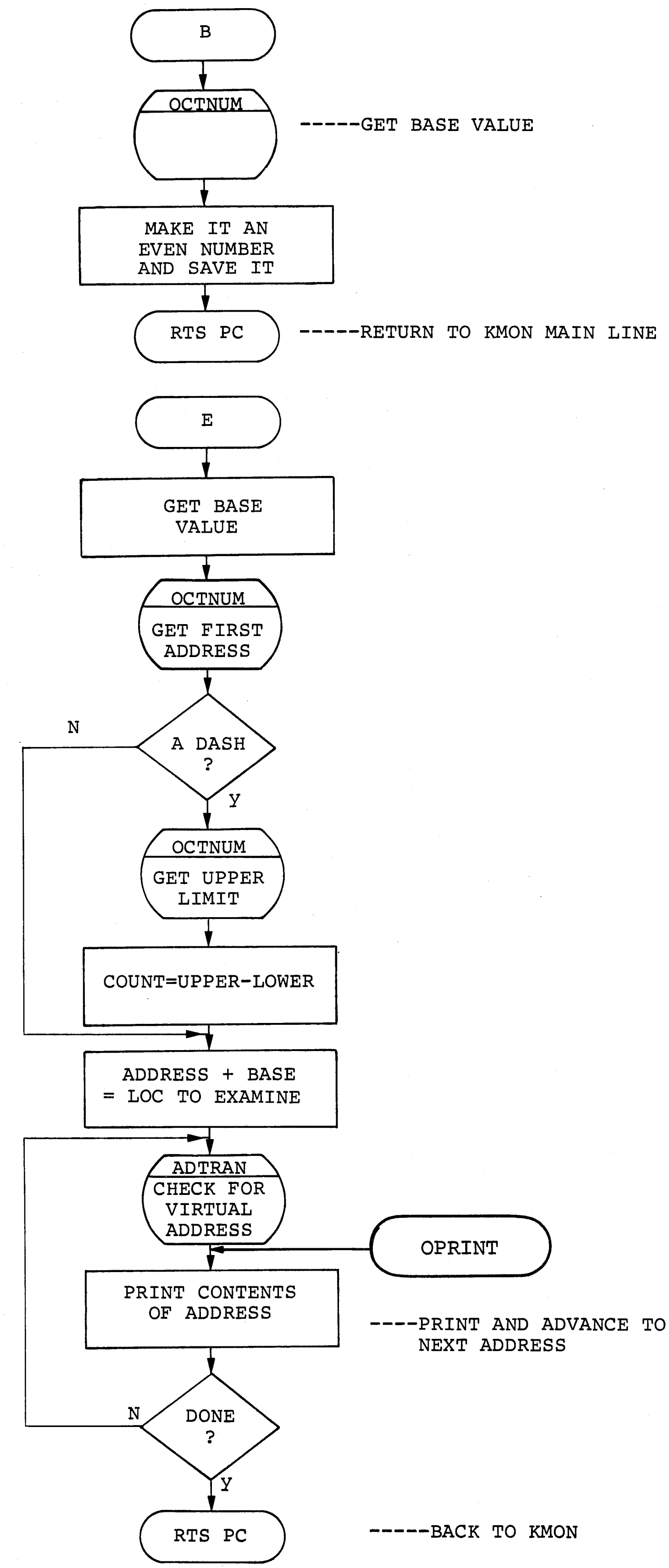
The following flowcharts are of the Single-Job and Foreground/ Background Monitors. It is recommended that the reader have source listings available for reference. Steps inside  are performed only in the F/B or S/J Monitor, as noted.

An index of all entry points appears at the end of the appendix.

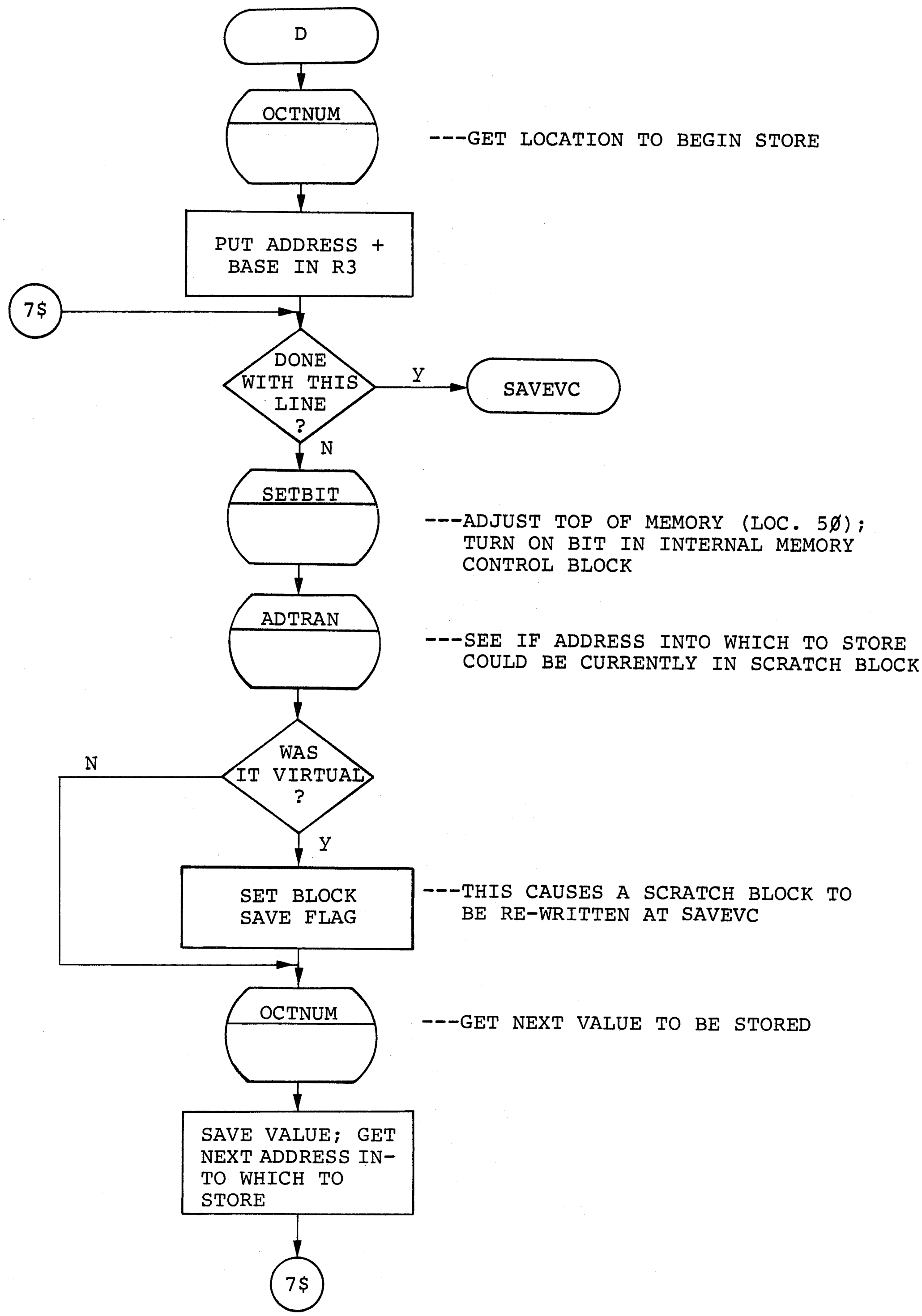
E.1 KMON (KEYBOARD MONITOR) FLOWCHARTS

KMON

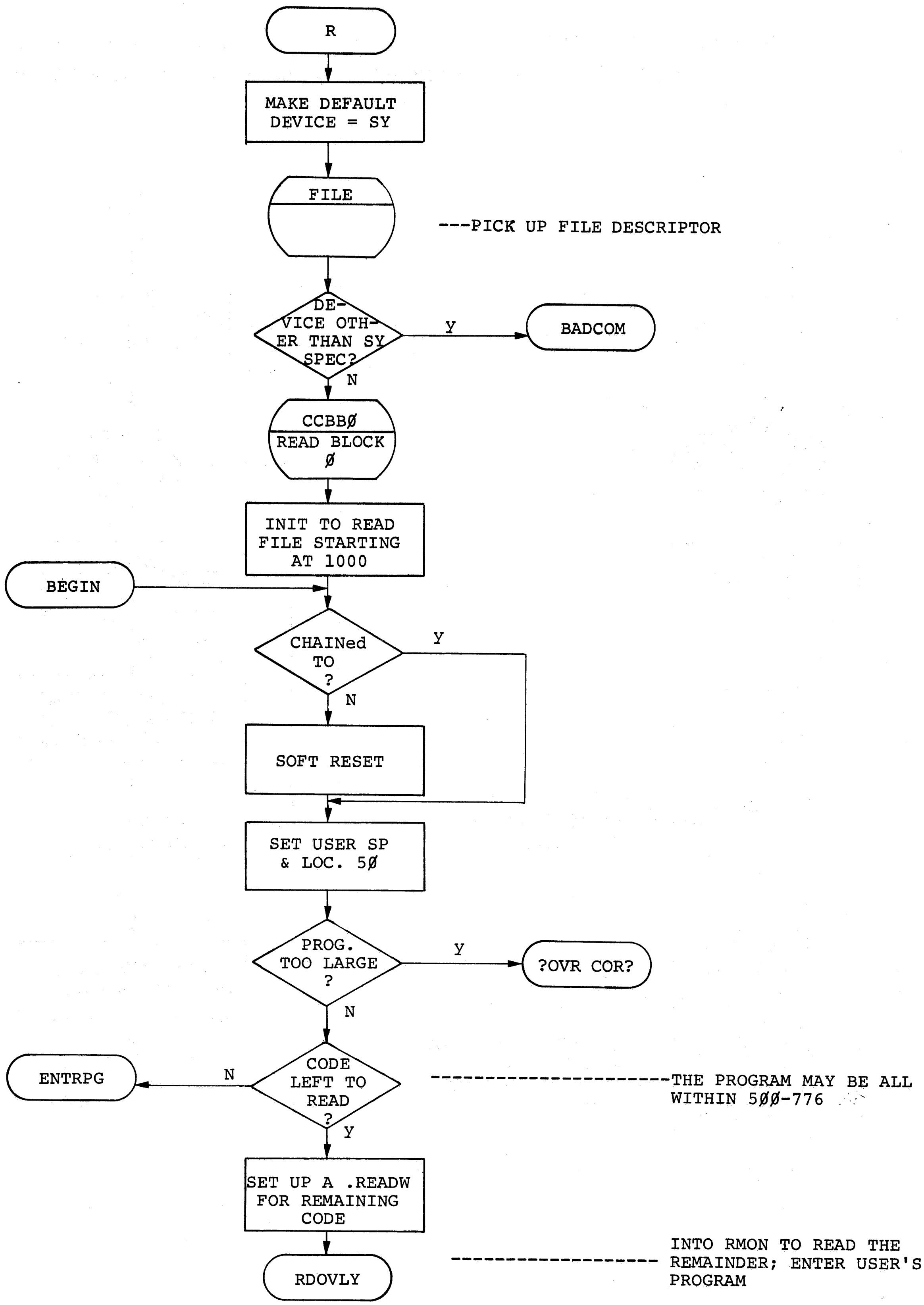

MEXIT2
MEXIT
INIT POINTERS & DATA BASE WORDS
LOCATE
FIX NEW USR POINTERS
CHAIN OR SOFT EXIT ?
Y
N
INIT
HRESET & CLEAR CHAIN BIT IN JSW
BADCOM
PRINT ?ILL CMD?
RTS PC
INIT THE STACK TO KMON STACK
.UNLOCK USR
F/B
SET R5 TO KMON COMMAND BUFF
CHAIN ENTRY ?
N
Y
PRINT A DOT
COLLECT AN INPUT LINE
.LOCK USR
F/B
OVERLAY CMD?
NO-DISPATCH
INIT
REENTER
STARTK
B
D
E
GET
RUN
R
RUN
Y
VALID OVERLAY CMD?
N
BADCOM
Y
OVREAD
GET OVERLAY INTO MEMORY
JSR TO COMMAND
RESUME
SUSPEND
FRUN
SET
GT ON/OFF
CLOSE
UNLOAD
LOAD
ASSIGN
SAVE
TIME
DATE


B
OCTNUM
-----GET BASE VALUE
MAKE IT AN EVEN NUMBER AND SAVE IT
RTS PC
-----RETURN TO KMON MAIN LINE
E
GET BASE VALUE
OCTNUM
GET FIRST ADDRESS
A DASH ?
N
Y
OCTNUM
GET UPPER LIMIT
COUNT=UPPER-LOWER
ADDRESS + BASE = LOC TO EXAMINE
ADTRAN
CHECK FOR VIRTUAL ADDRESS
OPRINT
PRINT CONTENTS OF ADDRESS
----PRINT AND ADVANCE TO NEXT ADDRESS
DONE ?
N
Y
RTS PC
-----BACK TO KMON

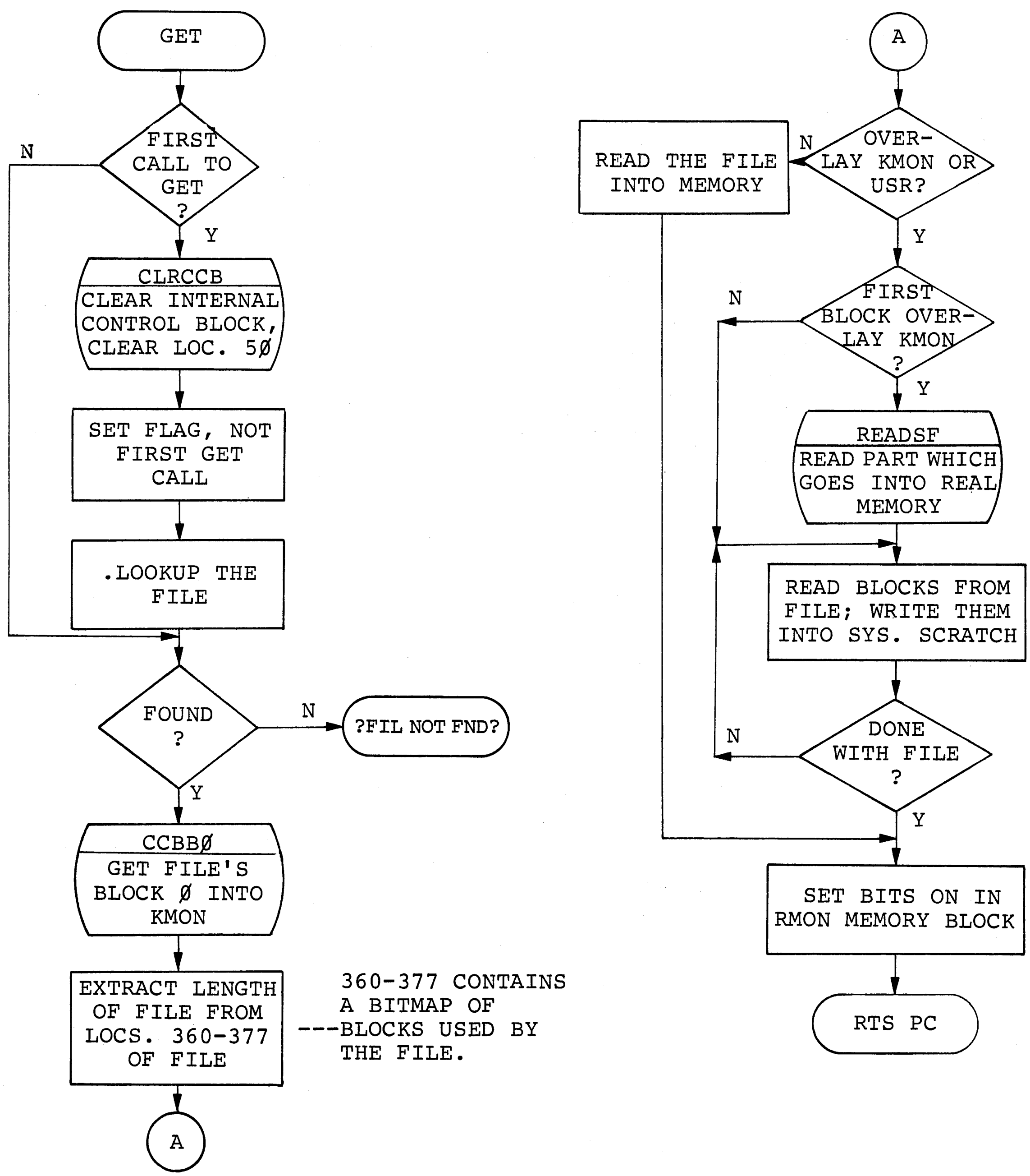
DEPOSIT



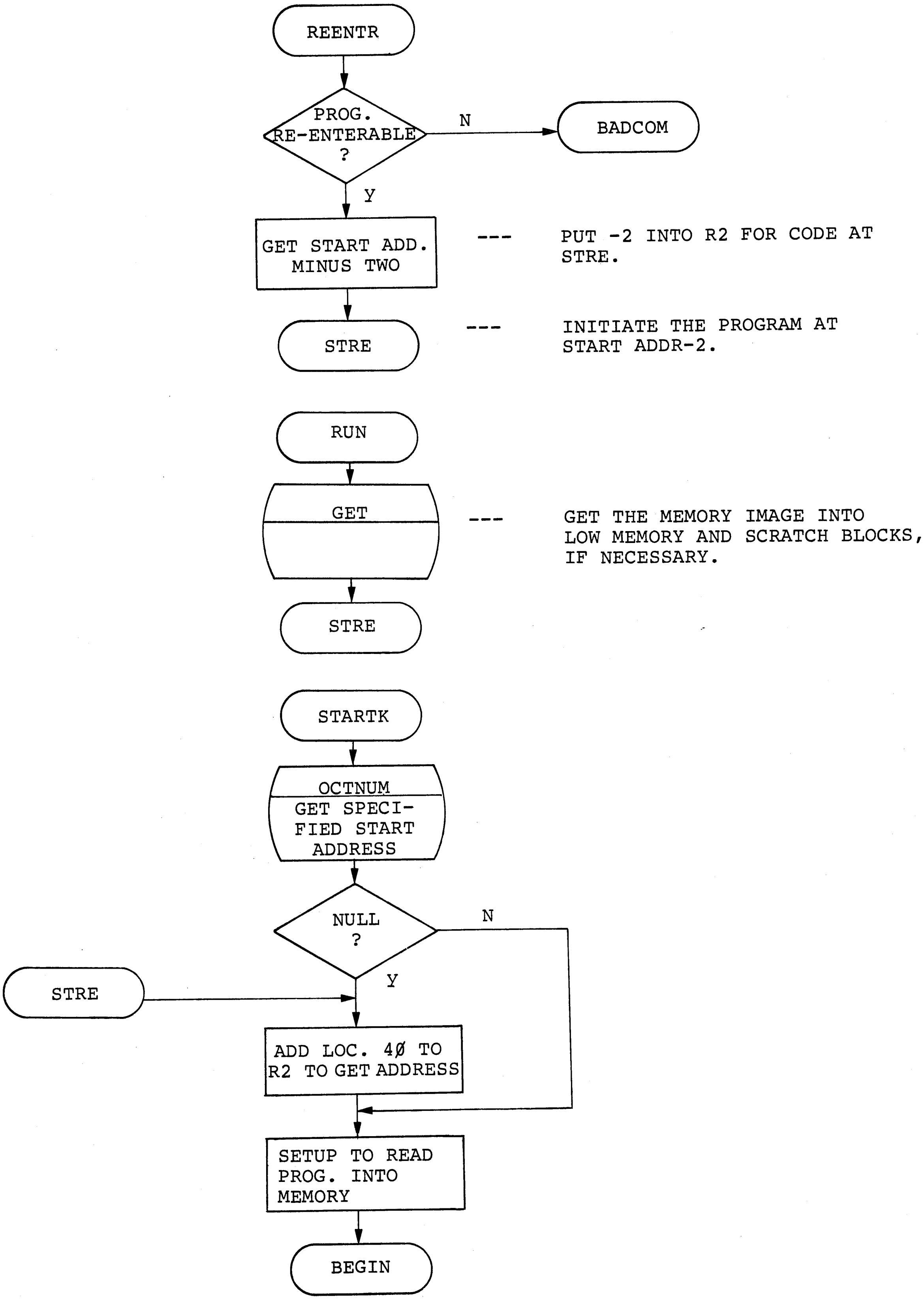
SAVEVC - Entered to rewrite the current virtual block back into the system scratch area. It also acts as the exit point for Deposit; The RTS PC will return control to KMON.


R
MAKE DEFAULT DEVICE = SY
FILE
---PICK UP FILE DESCRIPTOR
DE-VICE OTH-ER THAN SY SPEC?
Y
BADCOM
N
CCBBØ
READ BLOCK Ø
INIT TO READ FILE STARTING AT 1000
BEGIN
CHAINed TO ?
Y
N
SOFT RESET
SET USER SP & LOC. 5Ø
PROG. TOO LARGE ?
Y
?OVR COR?
N
ENTRPG
N
CODE LEFT TO READ ?
---------------------THE PROGRAM MAY BE ALL WITHIN 5ØØ-776
Y
SET UP A .READW FOR REMAINING CODE
RDOVLY
------------------- INTO RMON TO READ THE REMAINDER; ENTER USER'S PROGRAM

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.



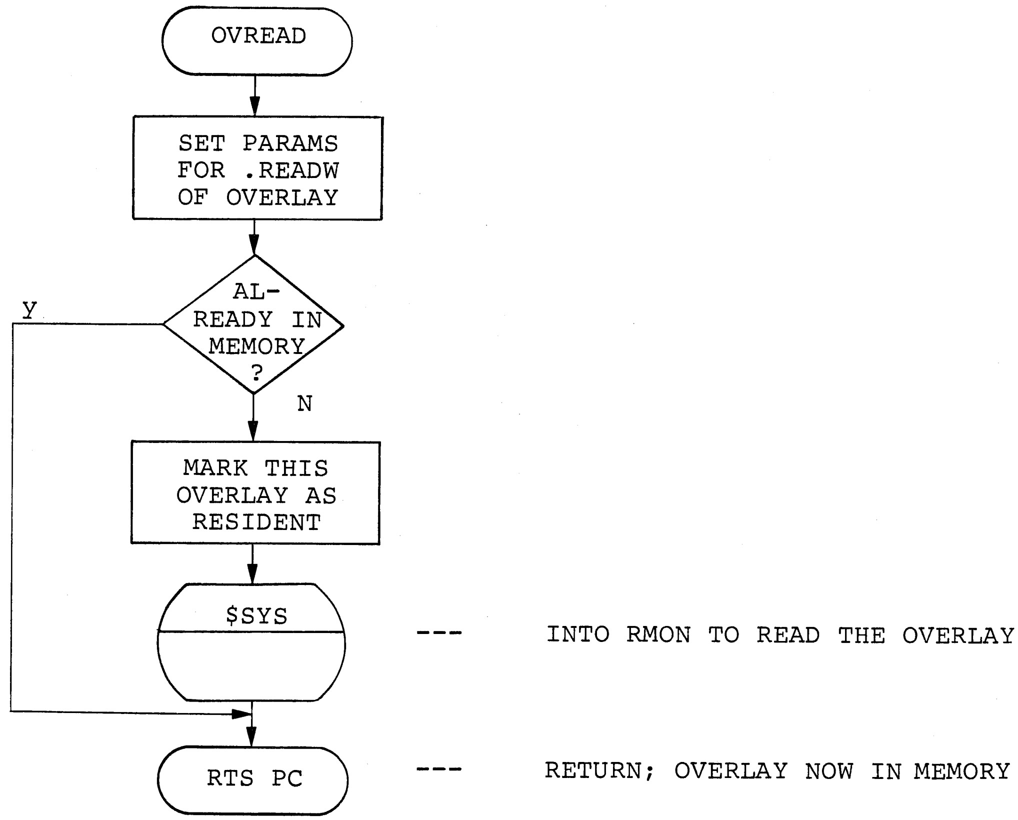
REENTER/RUN/START



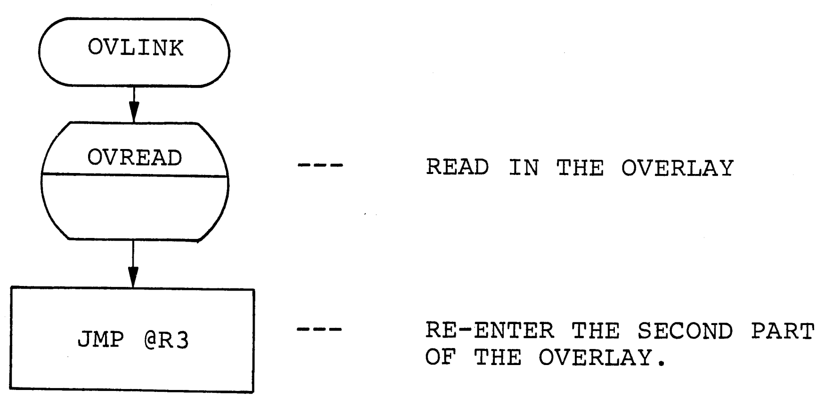
E.1.1 KMON Subroutines

OVREAD/OVLINK

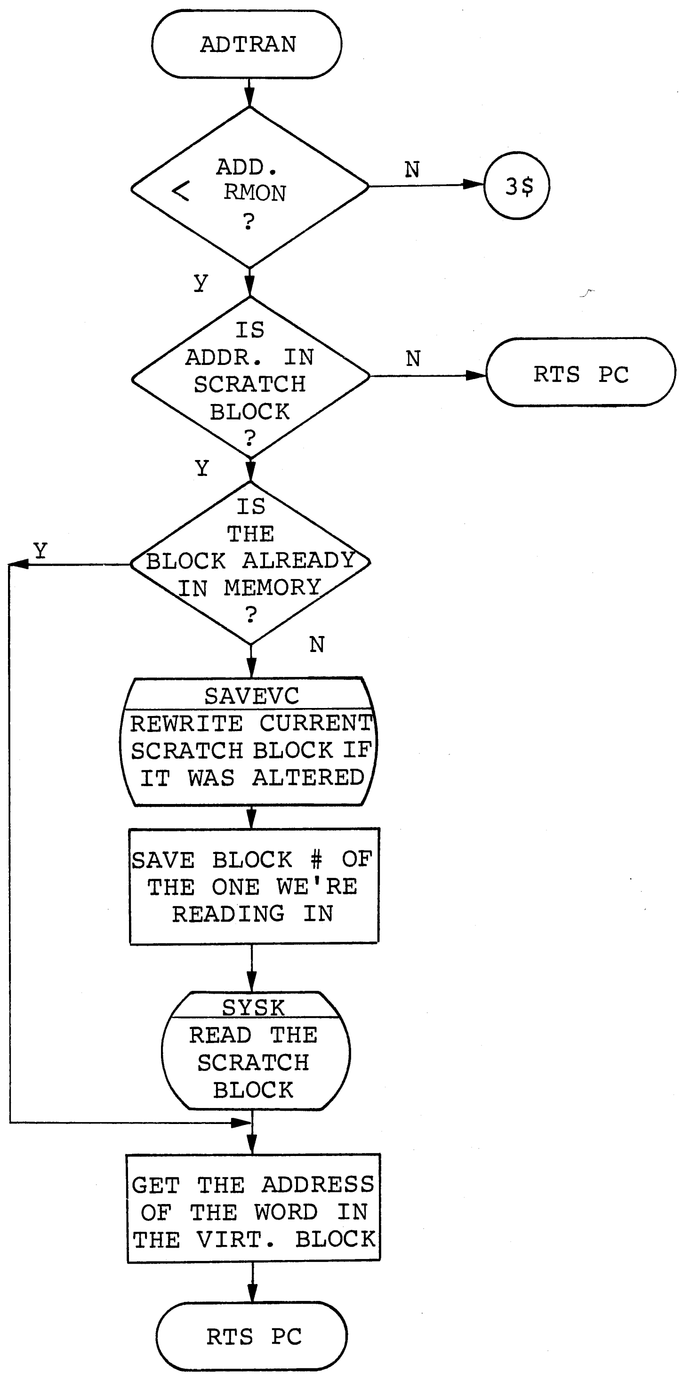
OVREAD - Used to read overlay command processors into memory.



OVLINK - Called from overlay processors to allow linking from one overlay to the other.



ADTRAN - Used to determine if a user-typed address is a) legal (i.e., address of RMON), b) in scratch blocks on system device.



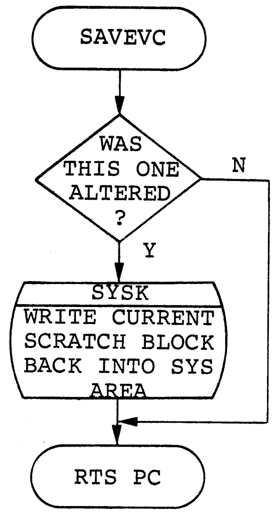
---GENERATE ?ADDR?

---BACK TO KMON.

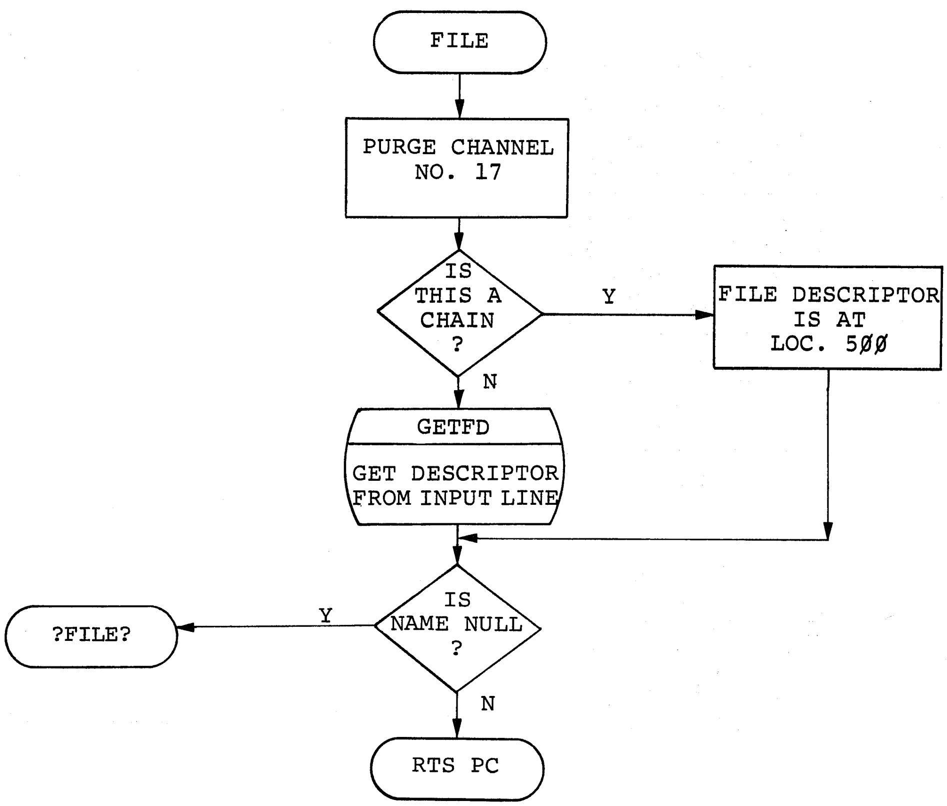
---BACK TO KMON.

SAVEVC/FILE

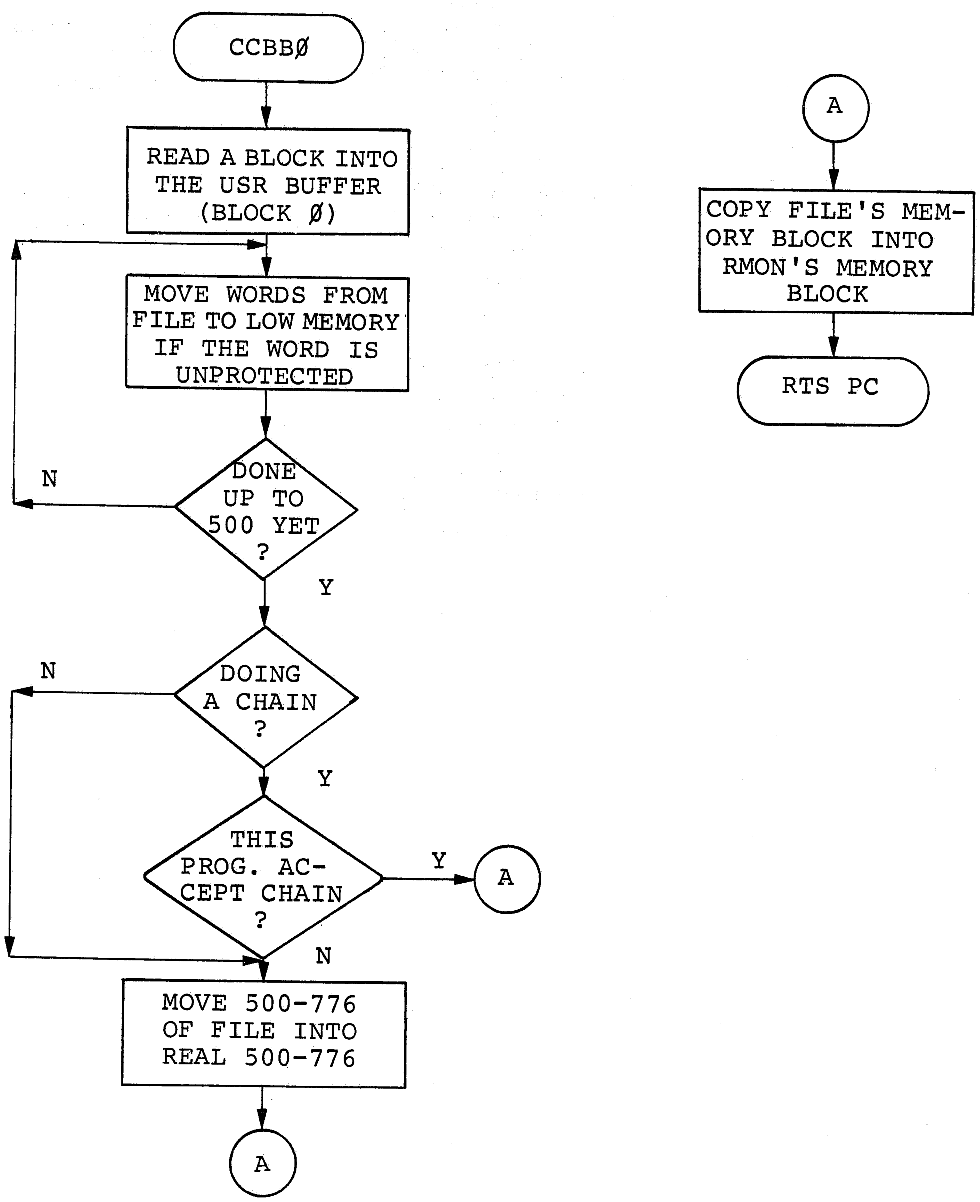
SAVEVC - Rewrites a block of memory back to the system scratch area if the block's contents were altered with a Deposit.



FILE - Called to pick up the .RAD5Ø representation of DEV:FILE.EXT. It will assume a default extension of .SAV.

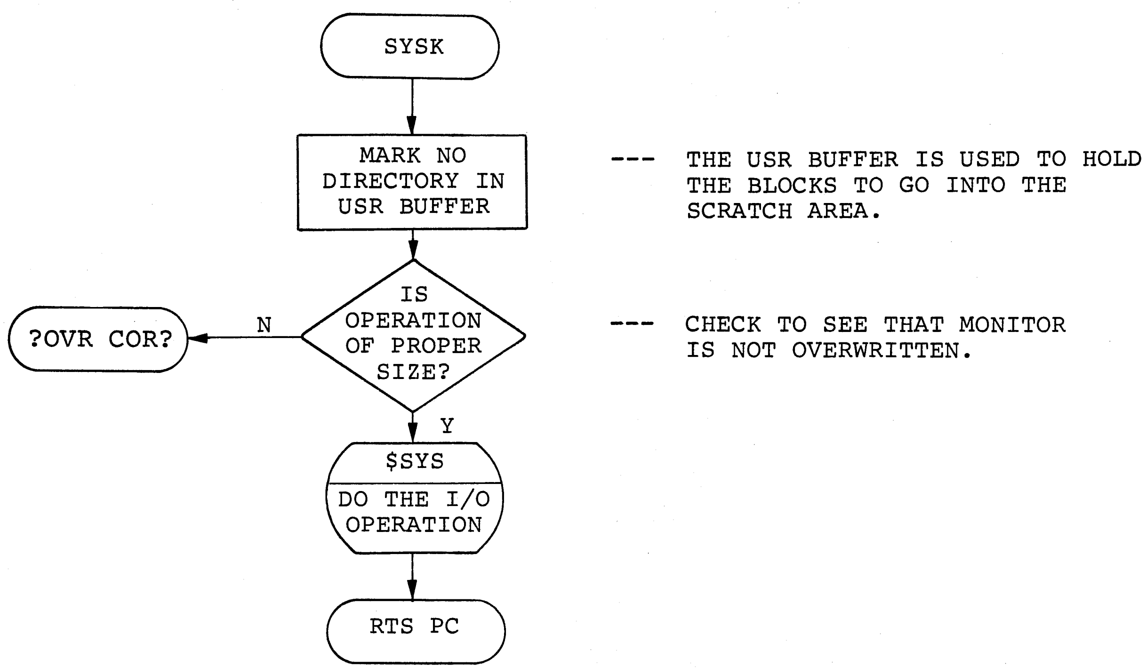


CCBBØ - The CCBBØ routine reads the first block of a .SAV file into the USR buffer, then moves selected locations from that block into the corresponding physical memory locations. The words moved are those marked with Ø's in the RMON bitmap. This procedure protects the system from having its vectors overlaid. If a chain is being done to a program which does not accept a CHAIN, 5ØØ-776 will be loaded with the contents of the file.



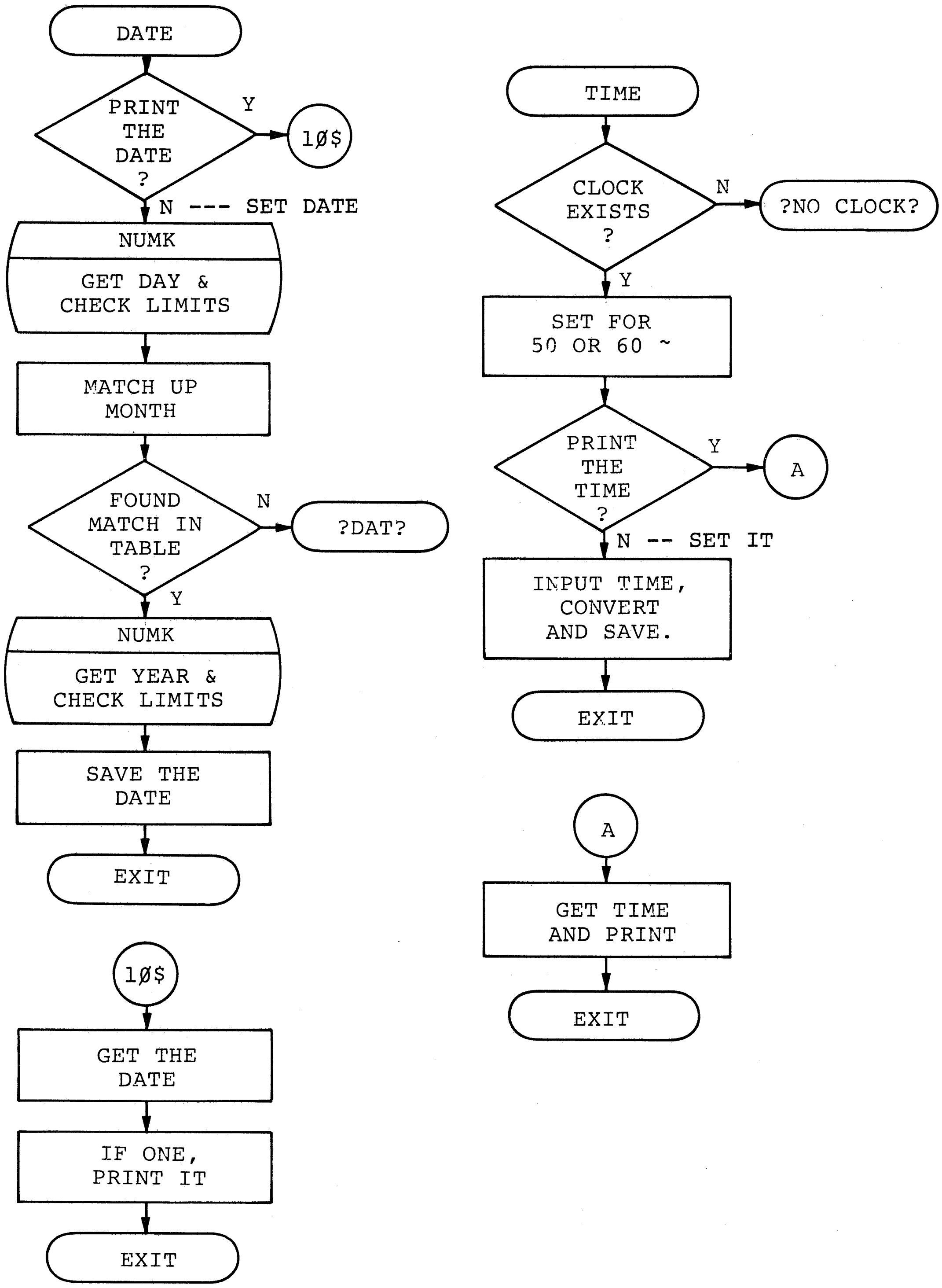
SYSK

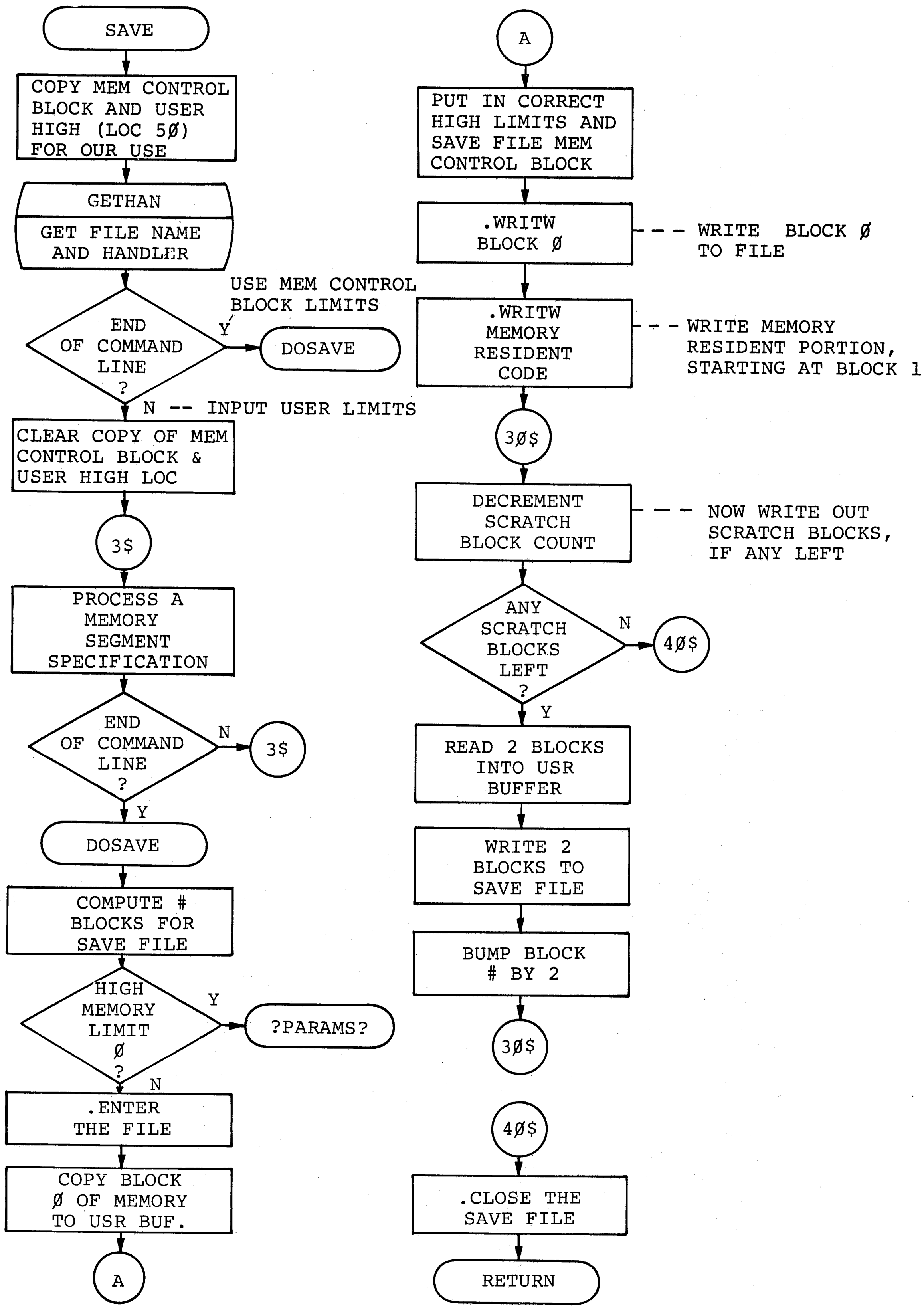
SYSK - Used to read/write blocks into and out of the system scratch area.



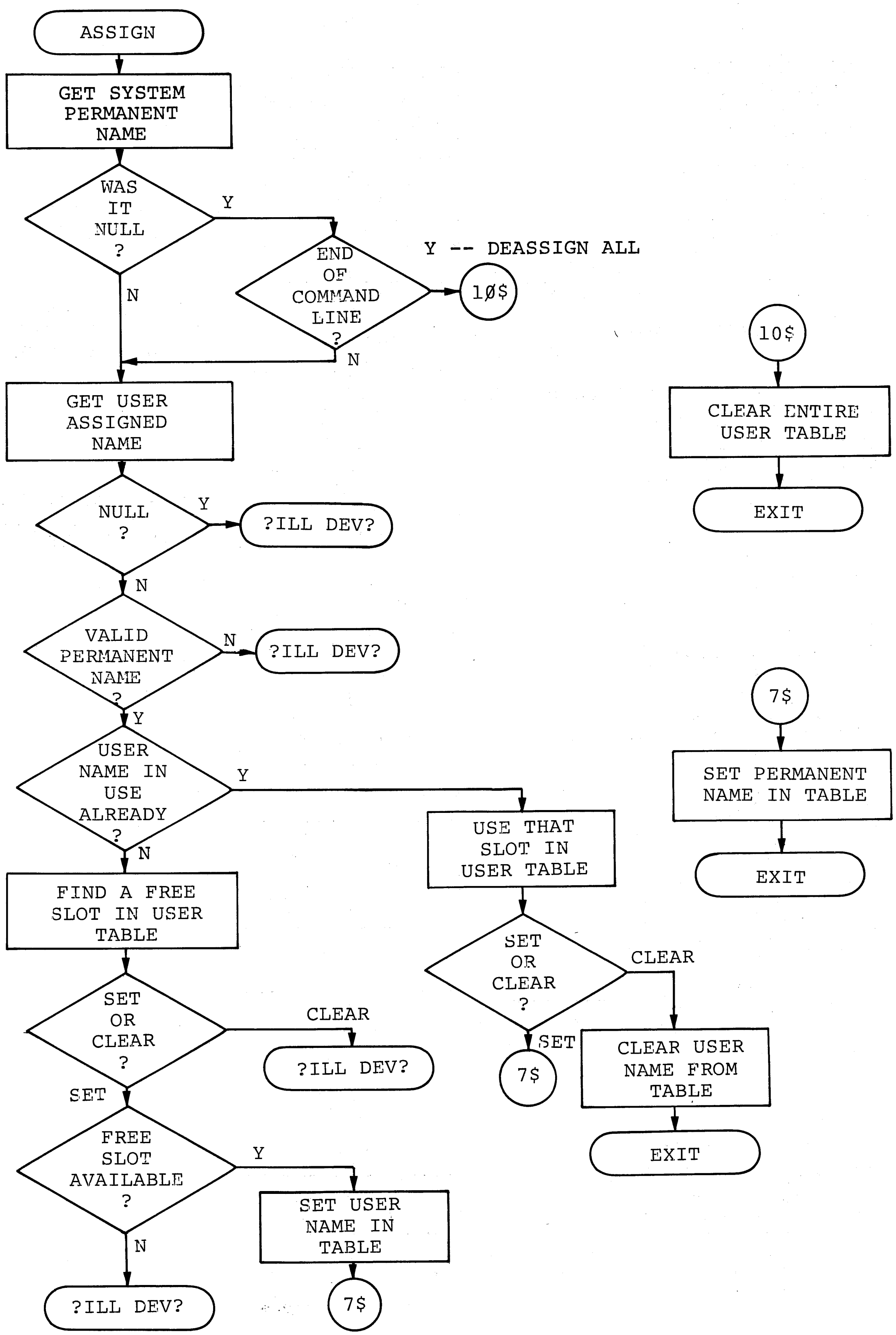
E.1.2 KMON Overlays

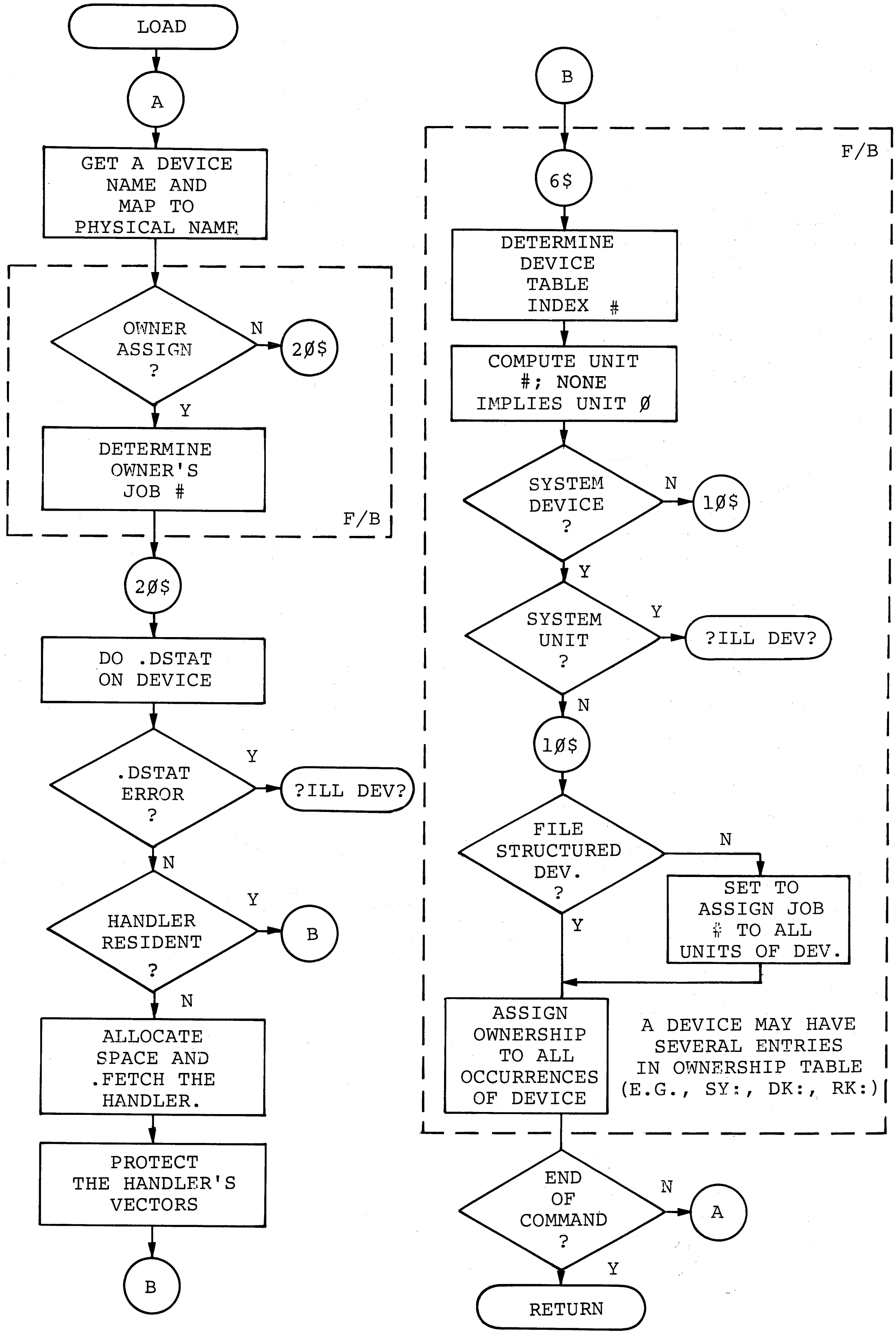
DATE/TIME


DATE
PRINT THE DATE ?
Y
1∅$
N --- SET DATE
NUMK
GET DAY & CHECK LIMITS
MATCH UP MONTH
FOUND MATCH IN TABLE ?
N
?DAT?
Y
NUMK
GET YEAR & CHECK LIMITS
SAVE THE DATE
EXIT
1∅$
GET THE DATE
IF ONE, PRINT IT
EXIT
TIME
CLOCK EXISTS ?
N
?NO CLOCK?
Y
SET FOR 50 OR 60 ~
PRINT THE TIME ?
Y
A
N -- SET IT
INPUT TIME, CONVERT AND SAVE.
EXIT
A
GET TIME AND PRINT
EXIT

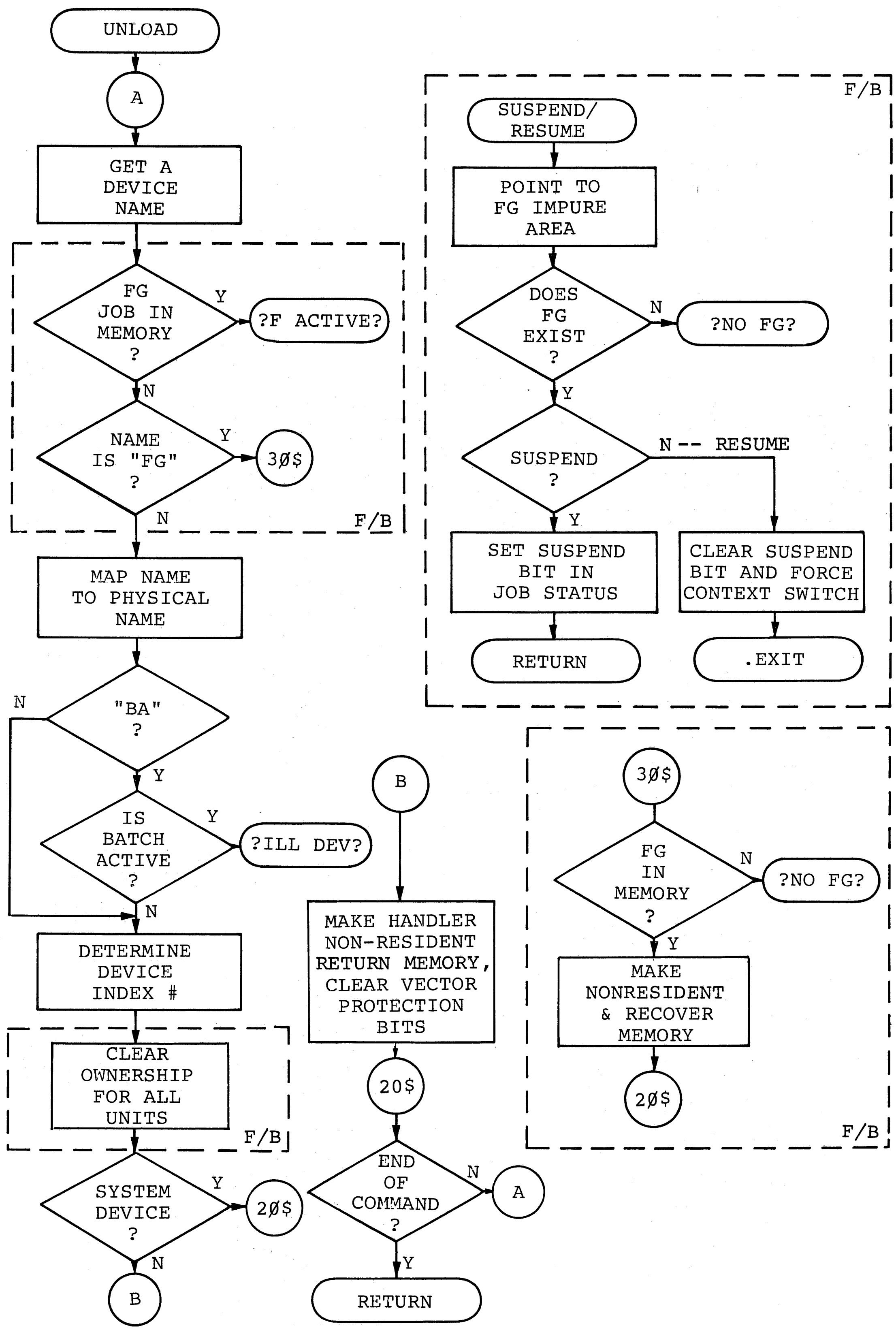

SAVE
COPY MEM CONTROL BLOCK AND USER HIGH (LOC 5Ø) FOR OUR USE
GETHAN
GET FILE NAME AND HANDLER
USE MEM CONTROL BLOCK LIMITS
END OF COMMAND LINE ?
Y
DOSAVE
N -- INPUT USER LIMITS
CLEAR COPY OF MEM CONTROL BLOCK & USER HIGH LOC
3$
PROCESS A MEMORY SEGMENT SPECIFICATION
END OF COMMAND LINE ?
N
3$
Y
DOSAVE
COMPUTE # BLOCKS FOR SAVE FILE
HIGH MEMORY LIMIT Ø ?
Y
?PARAMS?
N
.ENTER THE FILE
COPY BLOCK Ø OF MEMORY TO USR BUF.
A
A
PUT IN CORRECT HIGH LIMITS AND SAVE FILE MEM CONTROL BLOCK
.WRITW BLOCK Ø
WRITE BLOCK Ø TO FILE
.WRITW MEMORY RESIDENT CODE
WRITE MEMORY RESIDENT PORTION, STARTING AT BLOCK 1
3Ø$
DECREMENT SCRATCH BLOCK COUNT
NOW WRITE OUT SCRATCH BLOCKS, IF ANY LEFT
ANY SCRATCH BLOCKS LEFT ?
N
4Ø$
Y
READ 2 BLOCKS INTO USR BUFFER
WRITE 2 BLOCKS TO SAVE FILE
BUMP BLOCK # BY 2
3Ø$
4Ø$
.CLOSE THE SAVE FILE
RETURN

ASSIGN

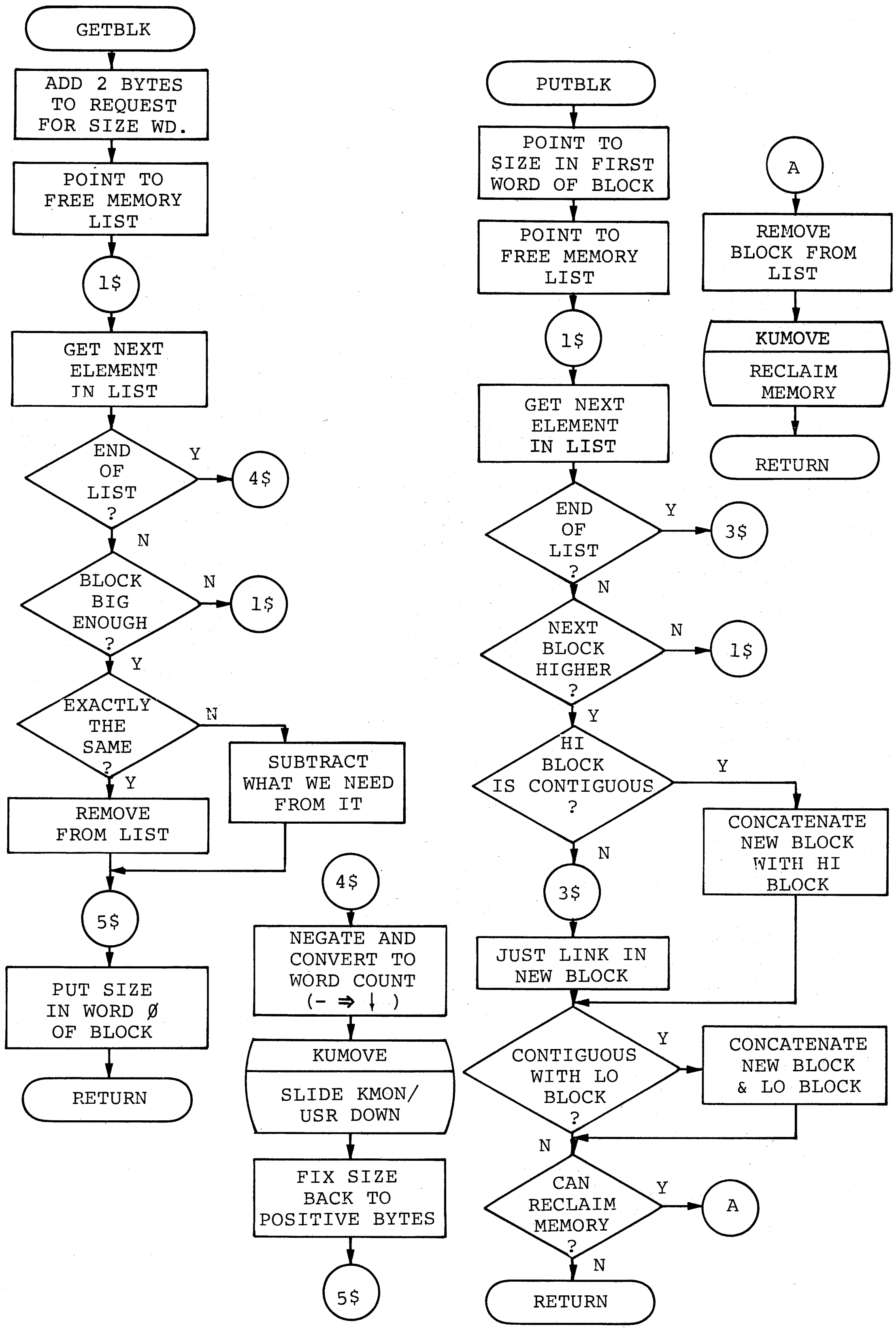

ASSIGN
GET SYSTEM PERMANENT NAME
WAS IT NULL ?
Y
N
END OF COMMAND LINE ?
Y -- DEASSIGN ALL
10$
N
GET USER ASSIGNED NAME
NULL ?
Y
?ILL DEV?
N
VALID PERMANENT NAME ?
N
?ILL DEV?
Y
USER NAME IN USE ALREADY ?
Y
N
USE THAT SLOT IN USER TABLE
FIND A FREE SLOT IN USER TABLE
SET OR CLEAR ?
CLEAR
?ILL DEV?
SET
FREE SLOT AVAILABLE ?
Y
SET USER NAME IN TABLE
7$
N
?ILL DEV?
SET OR CLEAR ?
CLEAR
SET
7$
CLEAR USER NAME FROM TABLE
EXIT
10$
CLEAR ENTIRE USER TABLE
EXIT
7$
SET PERMANENT NAME IN TABLE
EXIT


LOAD
A
GET A DEVICE NAME AND MAP TO PHYSICAL NAME
OWNER ASSIGN ?
N
20$
Y
DETERMINE OWNER'S JOB #
F/B
20$
DO .DSTAT ON DEVICE
.DSTAT ERROR ?
Y
?ILL DEV?
N
HANDLER RESIDENT ?
Y
B
N
ALLOCATE SPACE AND .FETCH THE HANDLER.
PROTECT THE HANDLER'S VECTORS
B
B
F/B
6$
DETERMINE DEVICE TABLE INDEX #
COMPUTE UNIT #; NONE IMPLIES UNIT 0
SYSTEM DEVICE ?
N
10$
Y
SYSTEM UNIT ?
Y
?ILL DEV?
N
10$
FILE STRUCTURED DEV. ?
N
SET TO ASSIGN JOB # TO ALL UNITS OF DEV.
Y
ASSIGN OWNERSHIP TO ALL OCCURRENCES OF DEVICE
A DEVICE MAY HAVE SEVERAL ENTRIES IN OWNERSHIP TABLE (E.G., SY:, DK:, RK:)
END OF COMMAND ?
N
A
Y
RETURN

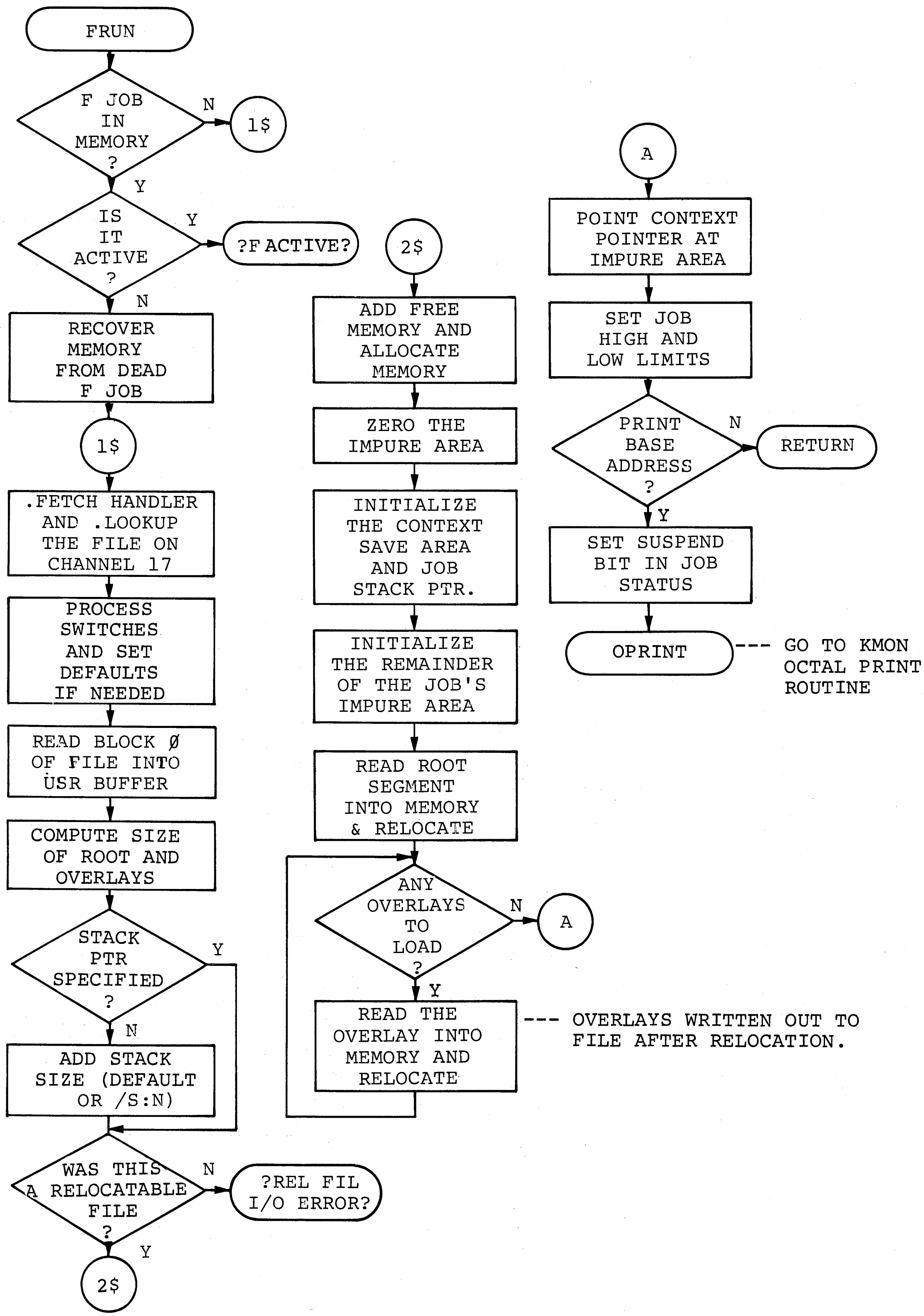
UNLOAD/SUSPEND/RESUME



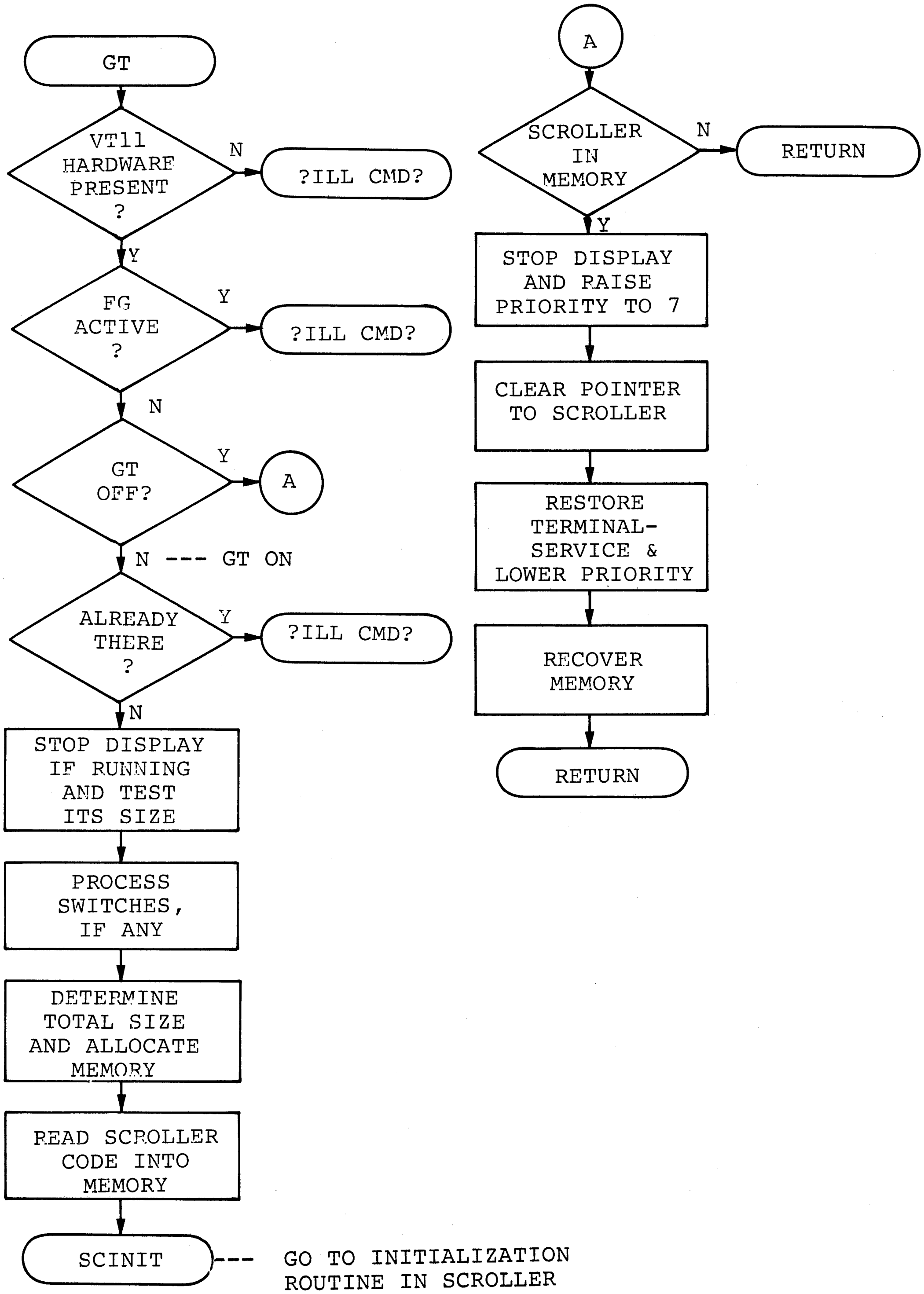
GET/PUT A BLOCK OF MEMORY



FRUN



FRUN
F JOB IN MEMORY ?
N
1$
Y
IS IT ACTIVE ?
Y
?F ACTIVE?
N
RECOVER MEMORY FROM DEAD F JOB
1$
.FETCH HANDLER AND .LOOKUP THE FILE ON CHANNEL 17
PROCESS SWITCHES AND SET DEFAULTS IF NEEDED
READ BLOCK Ø OF FILE INTO USR BUFFER
COMPUTE SIZE OF ROOT AND OVERLAYS
STACK PTR SPECIFIED ?
Y
N
ADD STACK SIZE (DEFAULT OR /S:N)
WAS THIS A RELOCATABLE FILE ?
N
?REL FIL I/O ERROR?
Y
2$
2$
ADD FREE MEMORY AND ALLOCATE MEMORY
ZERO THE IMPURE AREA
INITIALIZE THE CONTEXT SAVE AREA AND JOB STACK PTR.
INITIALIZE THE REMAINDER OF THE JOB'S IMPURE AREA
READ ROOT SEGMENT INTO MEMORY & RELOCATE
ANY OVERLAYS TO LOAD ?
N
A
Y
READ THE OVERLAY INTO MEMORY AND RELOCATE
--- OVERLAYS WRITTEN OUT TO FILE AFTER RELOCATION.
A
POINT CONTEXT POINTER AT IMPURE AREA
SET JOB HIGH AND LOW LIMITS
PRINT BASE ADDRESS ?
N
RETURN
Y
SET SUSPEND BIT IN JOB STATUS
OPRINT
--- GO TO KMON OCTAL PRINT ROUTINE


GT
VT11 HARDWARE PRESENT ?
N
?ILL CMD?
Y
FG ACTIVE ?
Y
?ILL CMD?
N
GT OFF?
Y
A
N --- GT ON
ALREADY THERE ?
Y
?ILL CMD?
N
STOP DISPLAY IF RUNNING AND TEST ITS SIZE
PROCESS SWITCHES, IF ANY
DETERMINE TOTAL SIZE AND ALLOCATE MEMORY
READ SCROLLER CODE INTO MEMORY
SCINIT
GO TO INITIALIZATION ROUTINE IN SCROLLER
A
SCROLLER IN MEMORY
N
RETURN
Y
STOP DISPLAY AND RAISE PRIORITY TO 7
CLEAR POINTER TO SCROLLER
RESTORE TERMINAL-SERVICE & LOWER PRIORITY
RECOVER MEMORY
RETURN

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)

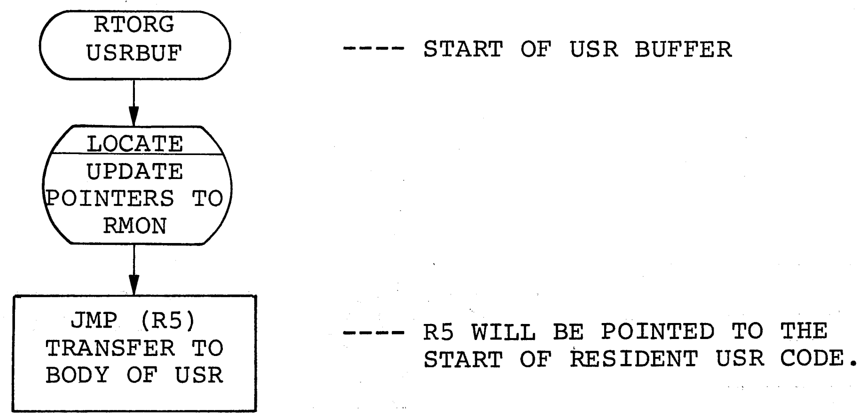
   The CDFN EMT call forces a new copy of the USR into memory to guarantee the presence of the EMT processor.

The flows for these functions follow.

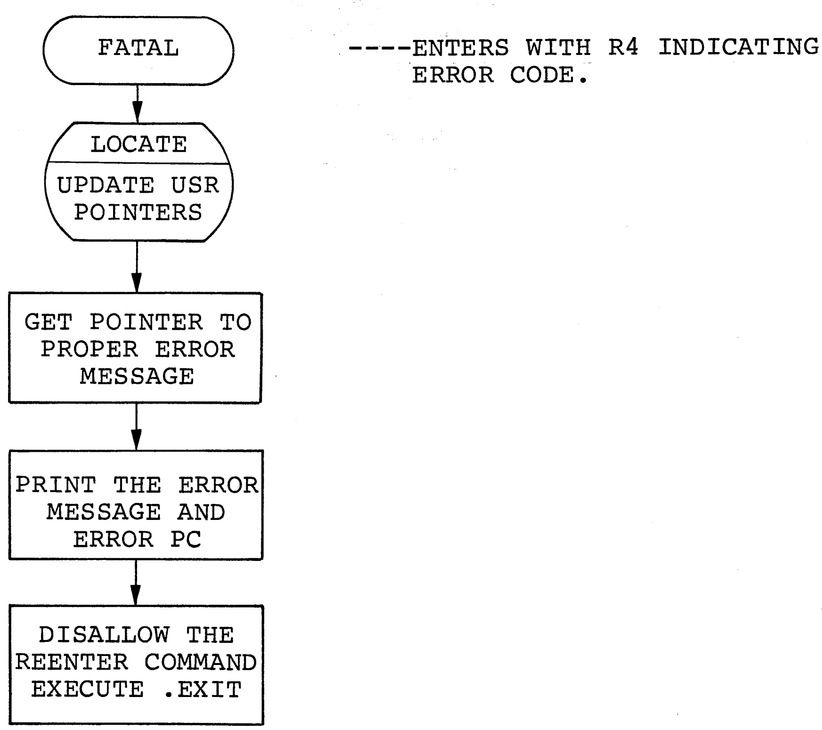
NOTE

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 is the initial entry point for USR calls when the USR has just been read into memory. LOCATE sets up pointers into RMON.

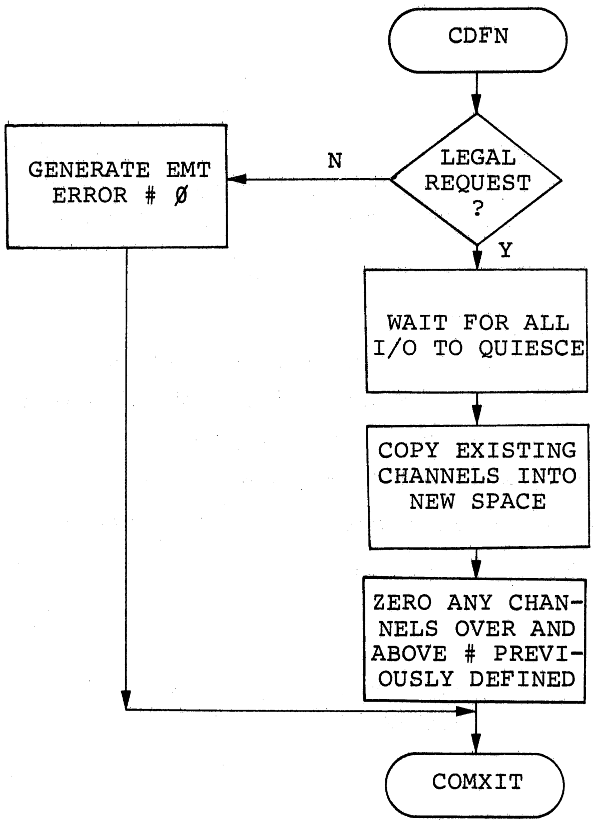


The LOCATE routine is called to update the list of pointers at RELIST. The list is initially a list of address differences (i.e., VALUE-$RMON where VALUE is the desired location and $RMON is the address of the start of RMON). LOCATE then makes all the differences into absolute addresses. Any errors which 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.

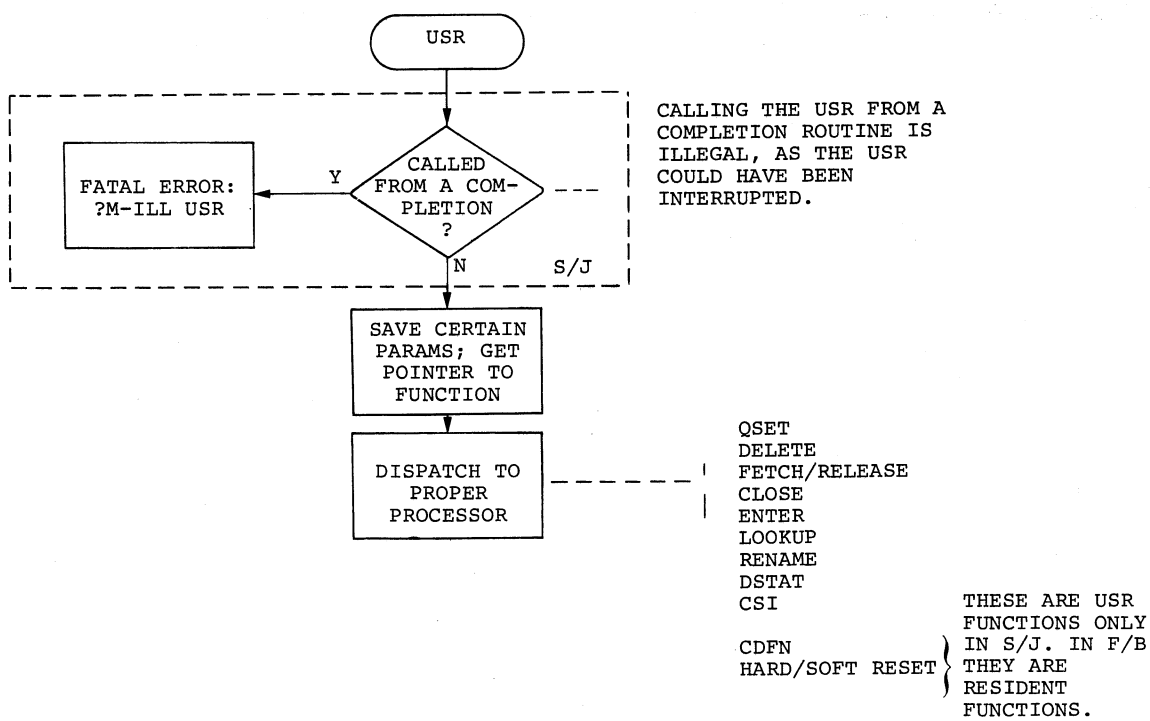


----IF REQUEST IS FOR FEWER THAN THAN ALREADY EXIST, IT IS AN ERROR.

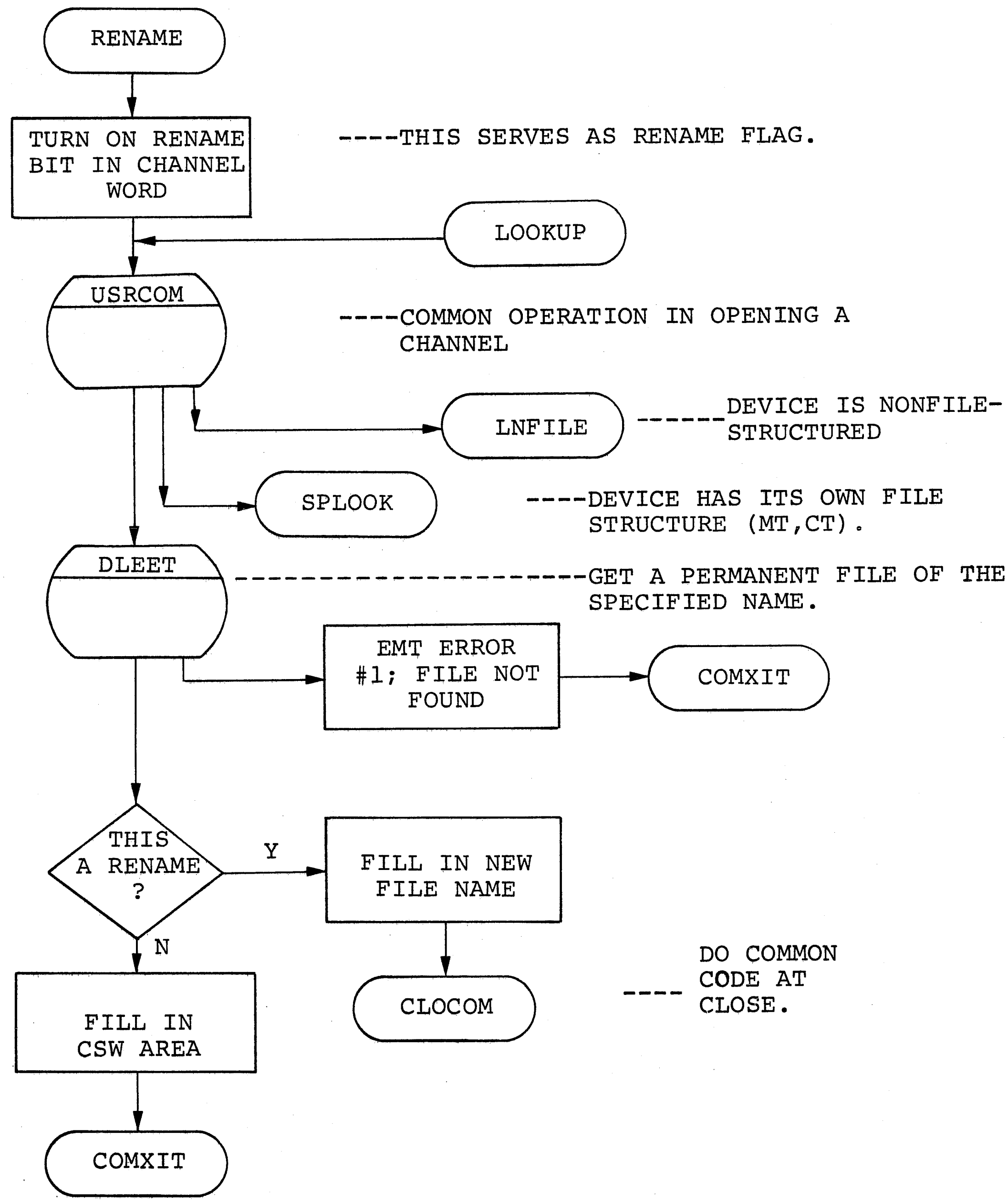
----TAKE COMMON USR EXIT.

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.

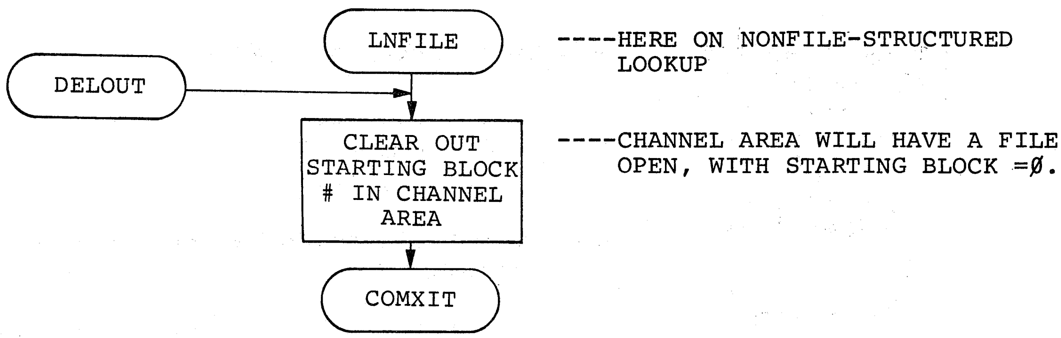
USR CODE

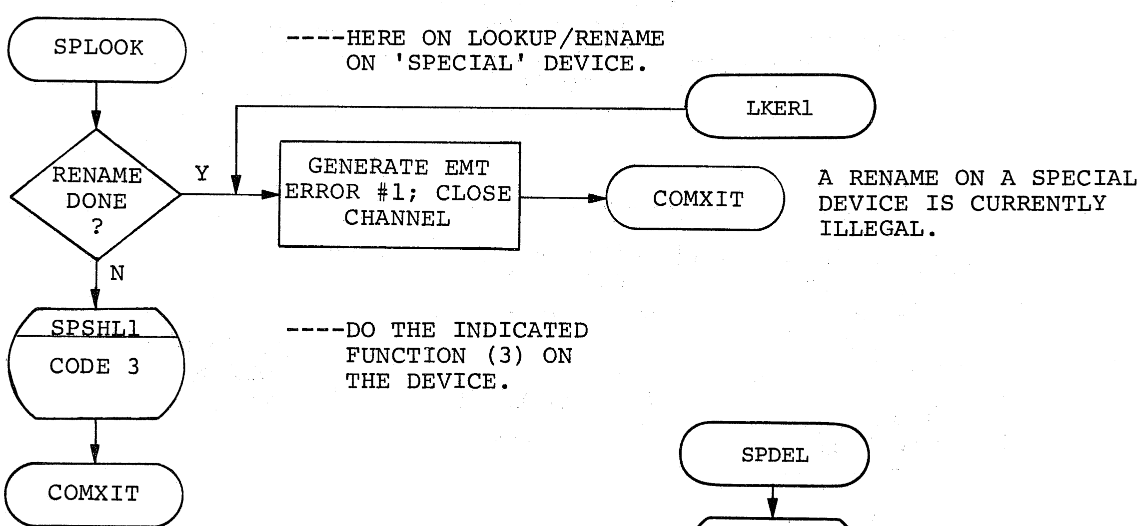


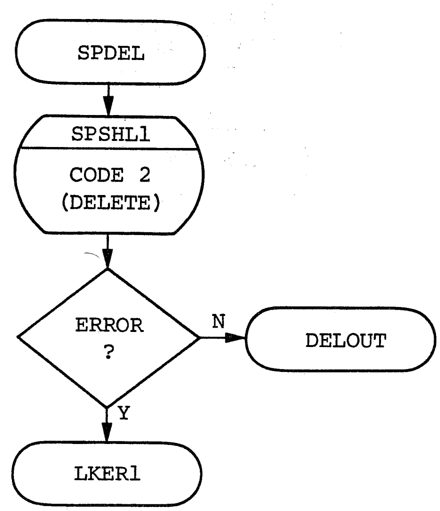
USR
CALLED FROM A COMPLETION ?
Y
FATAL ERROR: ?M-ILL USR
N
S/J
CALLING THE USR FROM A COMPLETION ROUTINE IS ILLEGAL, AS THE USR COULD HAVE BEEN INTERRUPTED.
SAVE CERTAIN PARAMS; GET POINTER TO FUNCTION
DISPATCH TO PROPER PROCESSOR
QSET
DELETE
FETCH/RELEASE
CLOSE
ENTER
LOOKUP
RENAME
DSTAT
CSI
CDFN
HARD/SOFT RESET
THESE ARE USR FUNCTIONS ONLY IN S/J. IN F/B THEY ARE RESIDENT FUNCTIONS.


RENAME
TURN ON RENAME BIT IN CHANNEL WORD
----THIS SERVES AS RENAME FLAG.
LOOKUP
USRCOM
----COMMON OPERATION IN OPENING A CHANNEL
LNFILE
------DEVICE IS NONFILE-STRUCTURED
SPLOOK
----DEVICE HAS ITS OWN FILE STRUCTURE (MT,CT).
DLEET
--------------------GET A PERMANENT FILE OF THE SPECIFIED NAME.
EMT ERROR #1; FILE NOT FOUND
COMXIT
THIS A RENAME ?
Y
FILL IN NEW FILE NAME
N
CLOCOM
---- DO COMMON CODE AT CLOSE.
FILL IN CSW AREA
COMXIT

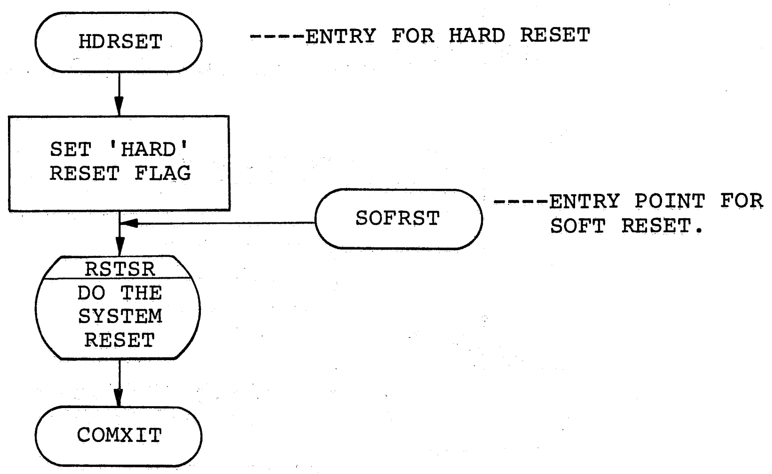
LOOKUP/RENAME (CONT.)

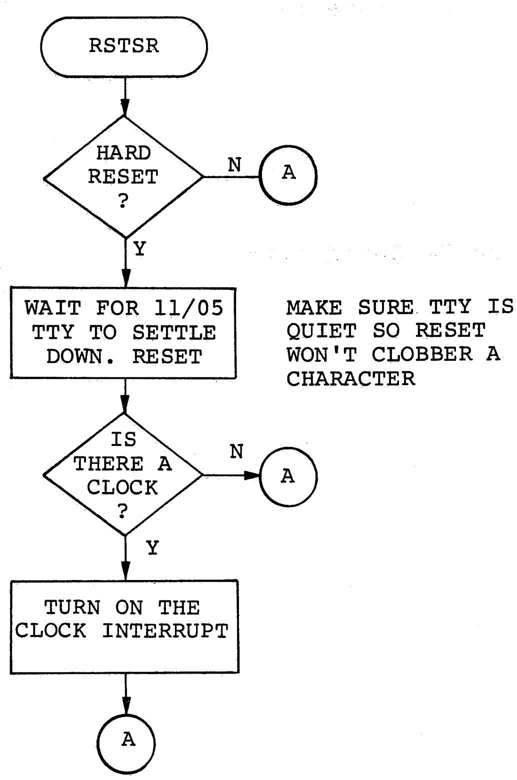


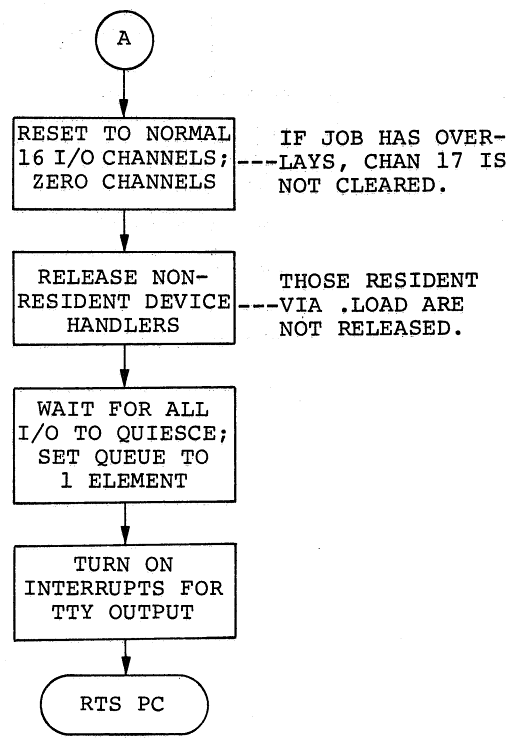




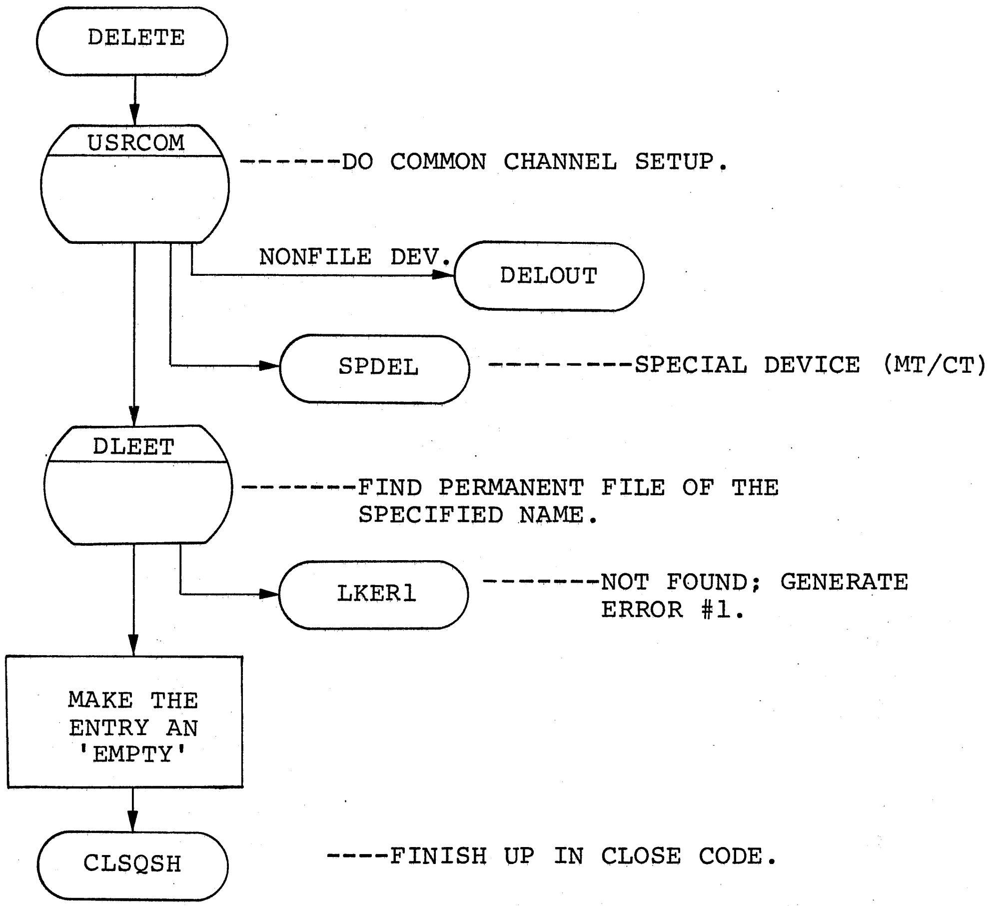
These are resident functions in F/B; USR functions in S/J.

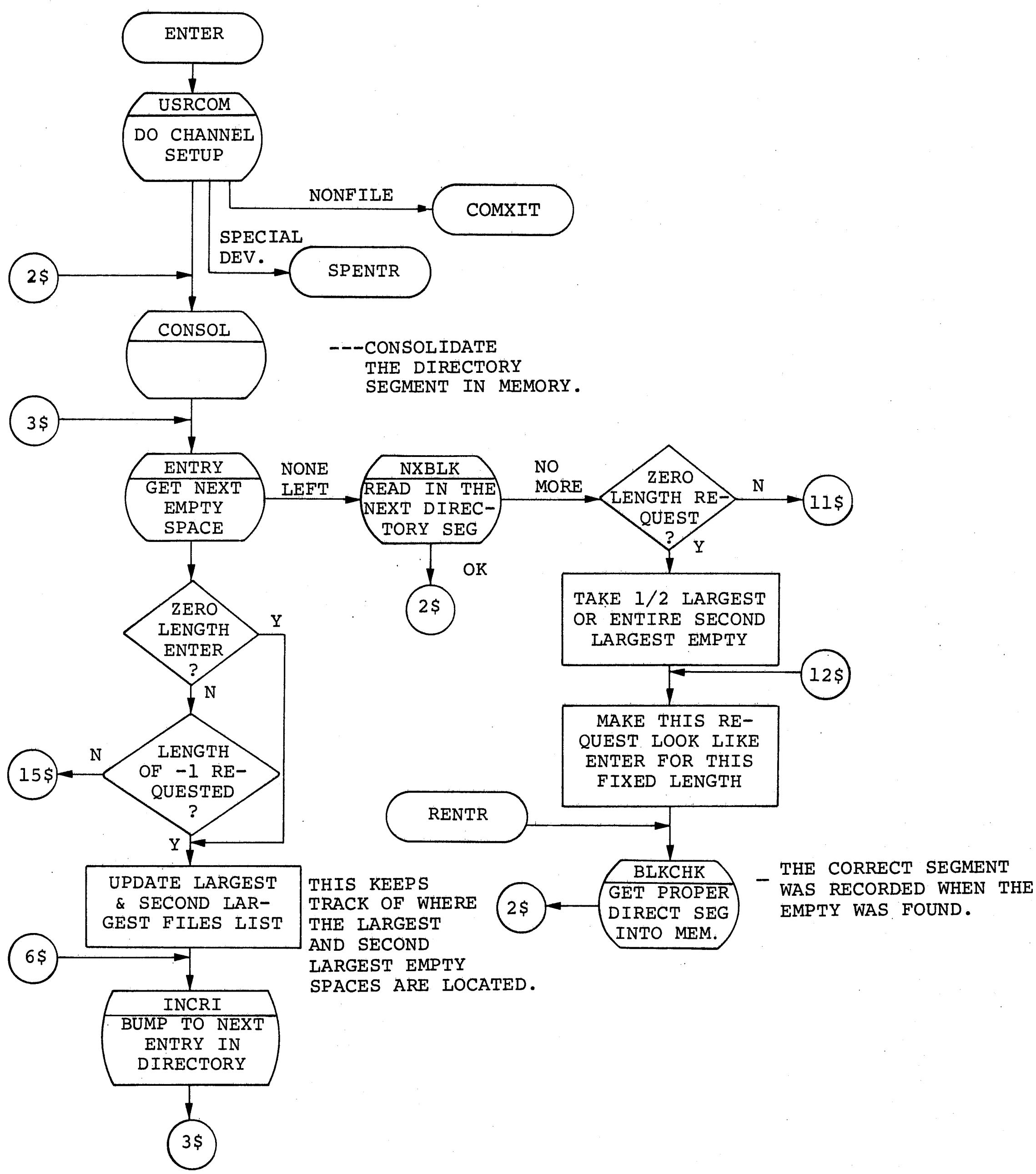






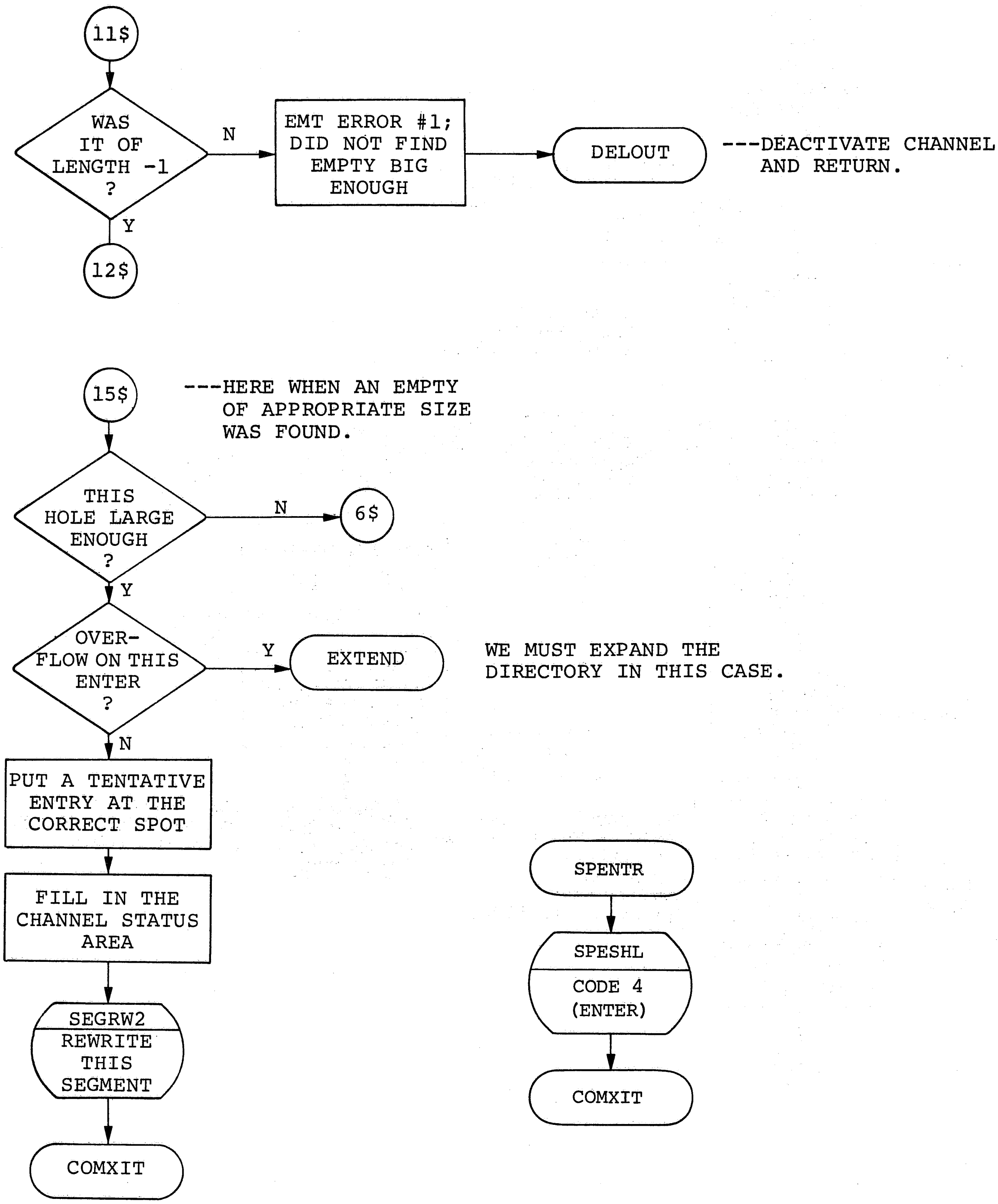
DELETE


DELETE
USRCOM
------DO COMMON CHANNEL SETUP.
NONFILE DEV.
DELOUT
SPDEL
---------SPECIAL DEVICE (MT/CT)
DLEET
-------FIND PERMANENT FILE OF THE
SPECIFIED NAME.
LKER1
-------NOT FOUND; GENERATE
ERROR #1.
MAKE THE
ENTRY AN
'EMPTY'
CLSQSH
----FINISH UP IN CLOSE CODE.

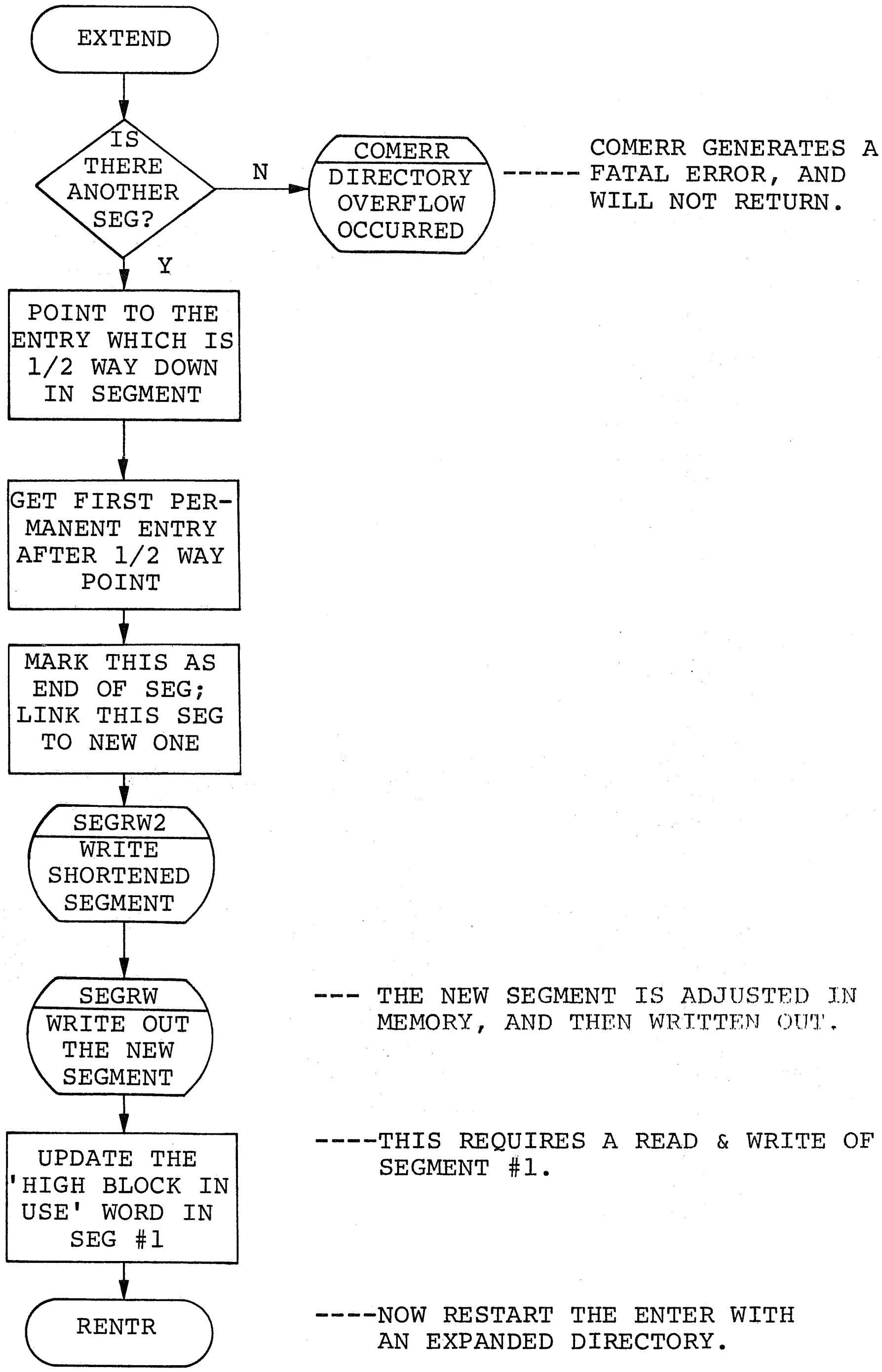


ENTER
USRCOM
DO CHANNEL SETUP
NONFILE
COMXIT
SPECIAL DEV.
SPENTR
2$
CONSOL
---CONSOLIDATE THE DIRECTORY SEGMENT IN MEMORY.
3$
ENTRY
GET NEXT EMPTY SPACE
NONE LEFT
NXBLK
READ IN THE NEXT DIREC- TORY SEG
OK
2$
NO MORE
ZERO LENGTH RE- QUEST ?
N
11$
Y
TAKE 1/2 LARGEST OR ENTIRE SECOND LARGEST EMPTY
12$
MAKE THIS RE- QUEST LOOK LIKE ENTER FOR THIS FIXED LENGTH
RENTR
BLKCHK
GET PROPER DIRECT SEG INTO MEM.
2$
– THE CORRECT SEGMENT WAS RECORDED WHEN THE EMPTY WAS FOUND.
ZERO LENGTH ENTER ?
Y
N
LENGTH OF -1 RE- QUESTED ?
N
15$
Y
UPDATE LARGEST & SECOND LAR- GEST FILES LIST
THIS KEEPS TRACK OF WHERE THE LARGEST AND SECOND LARGEST EMPTY SPACES ARE LOCATED.
6$
INCRI
BUMP TO NEXT ENTRY IN DIRECTORY
3$

ENTER (CONT.)

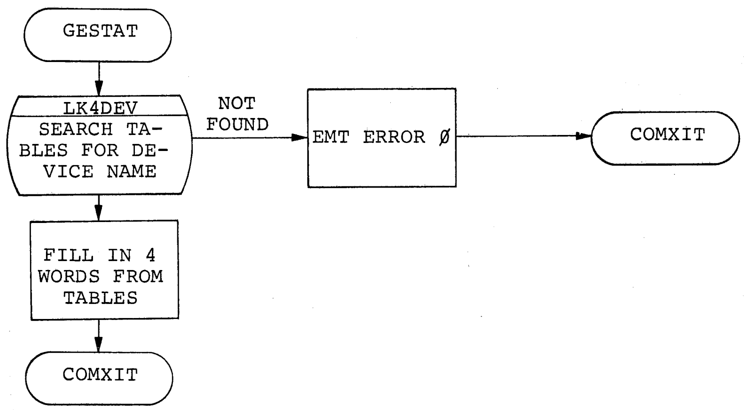


11$
WAS IT OF LENGTH -1 ?
N
EMT ERROR #1; DID NOT FIND EMPTY BIG ENOUGH
DELOUT
---DEACTIVATE CHANNEL AND RETURN.
Y
12$
15$
---HERE WHEN AN EMPTY OF APPROPRIATE SIZE WAS FOUND.
THIS HOLE LARGE ENOUGH ?
N
6$
Y
OVER-FLOW ON THIS ENTER ?
Y
EXTEND
WE MUST EXPAND THE DIRECTORY IN THIS CASE.
N
PUT A TENTATIVE ENTRY AT THE CORRECT SPOT
FILL IN THE CHANNEL STATUS AREA
SEGRW2
REWRITE THIS SEGMENT
COMXIT
SPENTR
SPESHL
CODE 4 (ENTER)
COMXIT

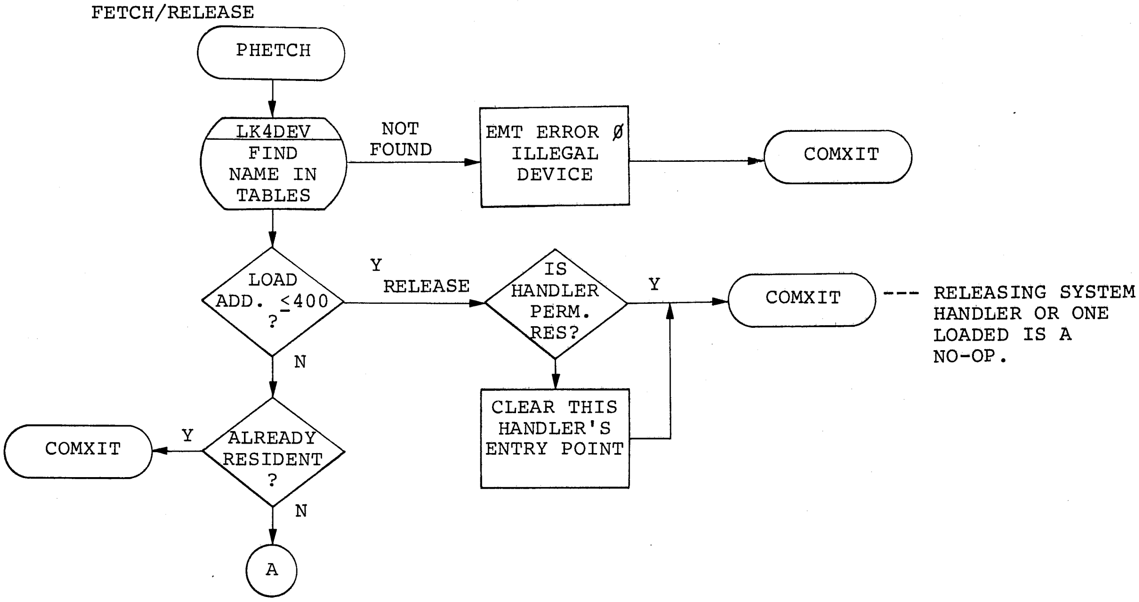

EXTEND
IS THERE ANOTHER SEG?
N
Y
COMERR
DIRECTORY OVERFLOW OCCURRED
----- COMERR GENERATES A FATAL ERROR, AND WILL NOT RETURN.
POINT TO THE ENTRY WHICH IS 1/2 WAY DOWN IN SEGMENT
GET FIRST PER-MANENT ENTRY AFTER 1/2 WAY POINT
MARK THIS AS END OF SEG; LINK THIS SEG TO NEW ONE
SEGRW2
WRITE SHORTENED SEGMENT
SEGRW
WRITE OUT THE NEW SEGMENT
--- THE NEW SEGMENT IS ADJUSTED IN MEMORY, AND THEN WRITTEN OUT.
UPDATE THE 'HIGH BLOCK IN USE' WORD IN SEG #1
----THIS REQUIRES A READ & WRITE OF SEGMENT #1.
RENTR
----NOW RESTART THE ENTER WITH AN EXPANDED DIRECTORY.

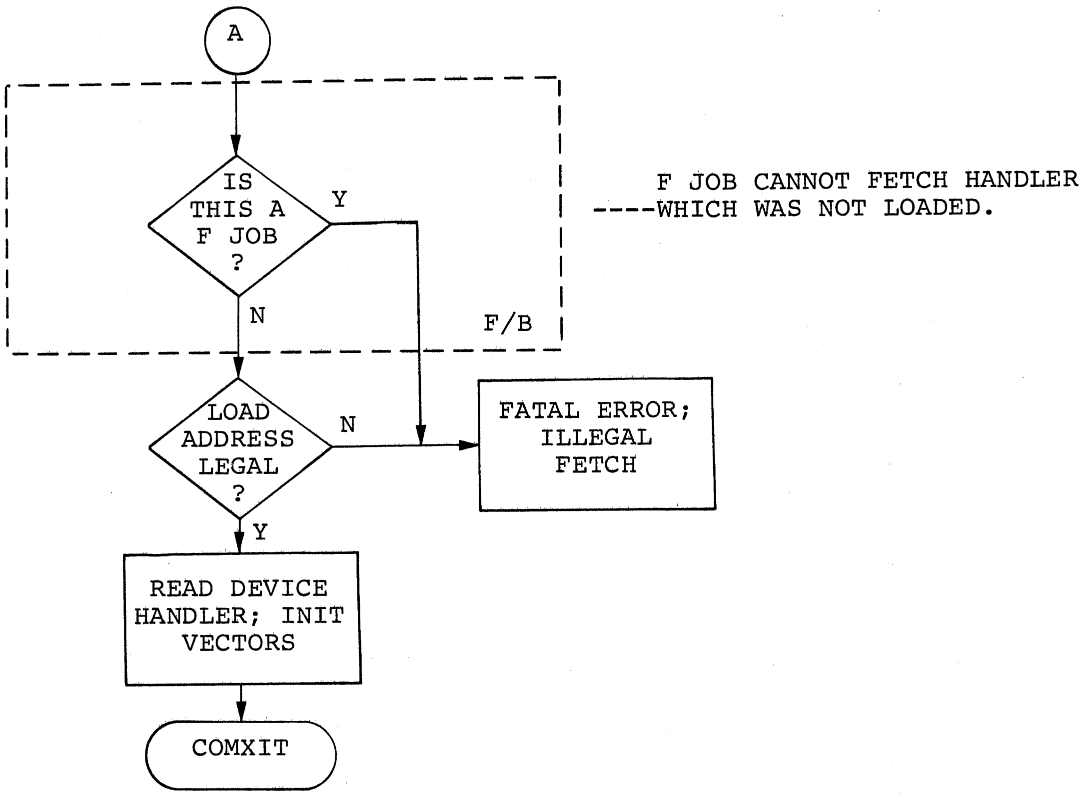
DSTAT/FETCH/RELEASE

DSTAT- GET DEVICE STATUS

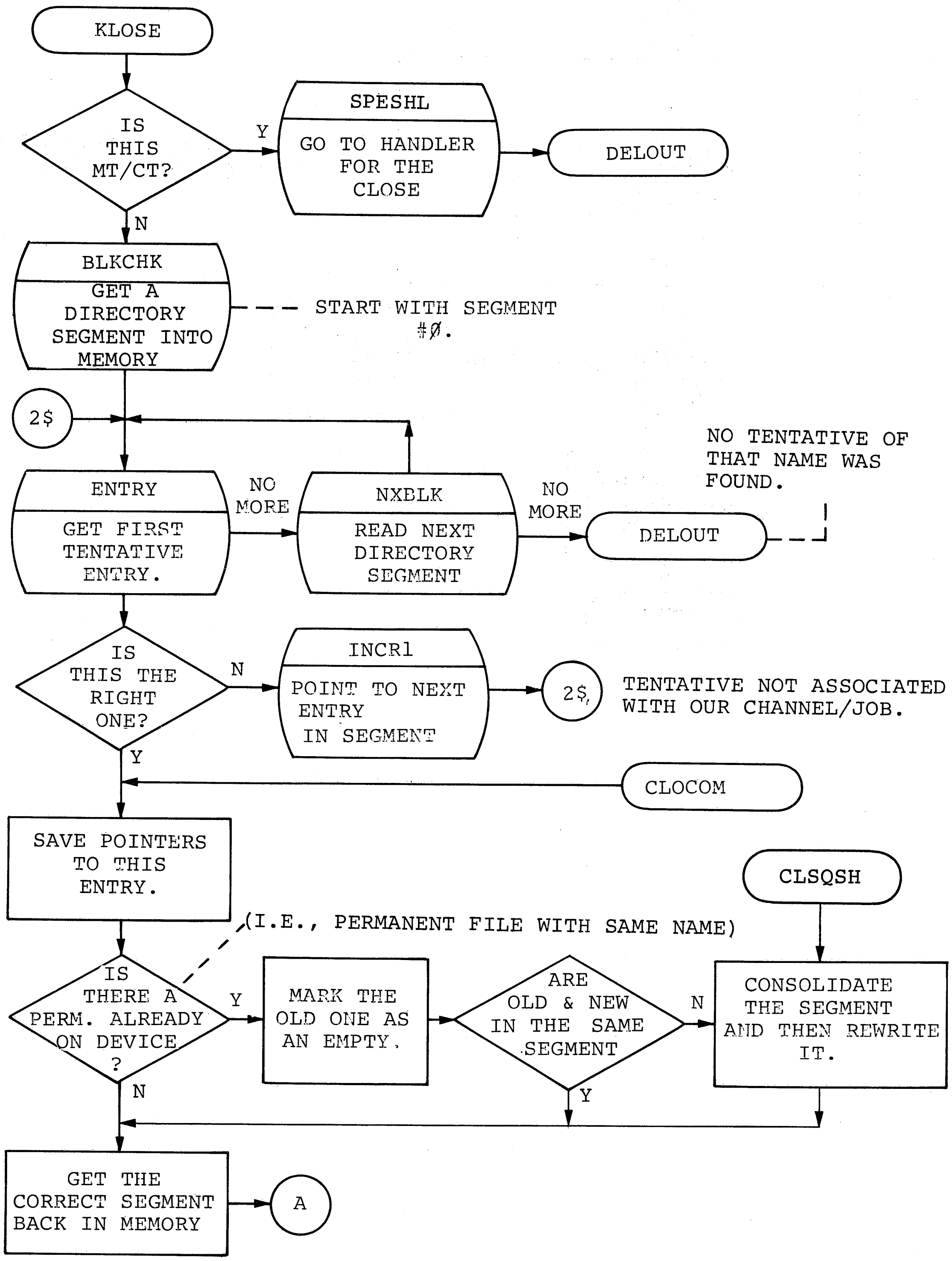


FETCH/RELEASE

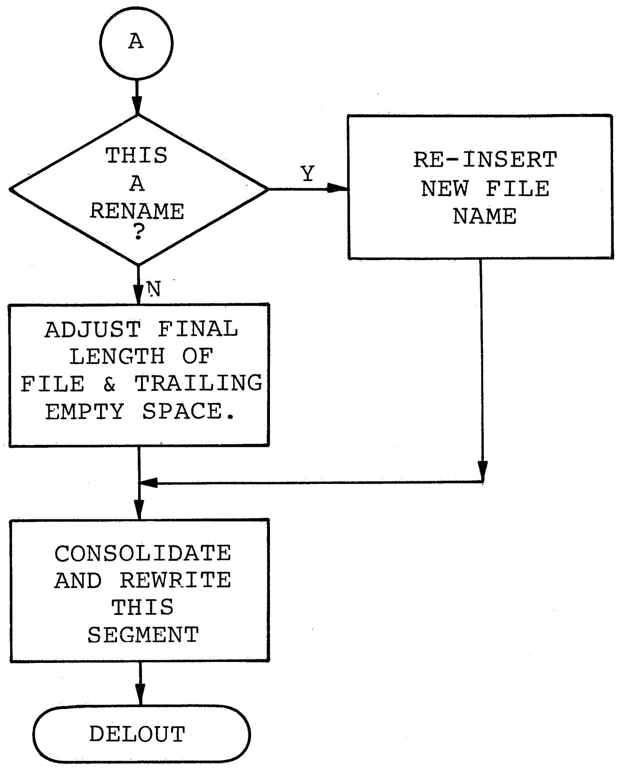



A
IS THIS A F JOB ?
Y
N
F/B
F JOB CANNOT FETCH HANDLER
----WHICH WAS NOT LOADED.
LOAD ADDRESS LEGAL ?
N
Y
FATAL ERROR; ILLEGAL FETCH
READ DEVICE HANDLER; INIT VECTORS
COMXIT

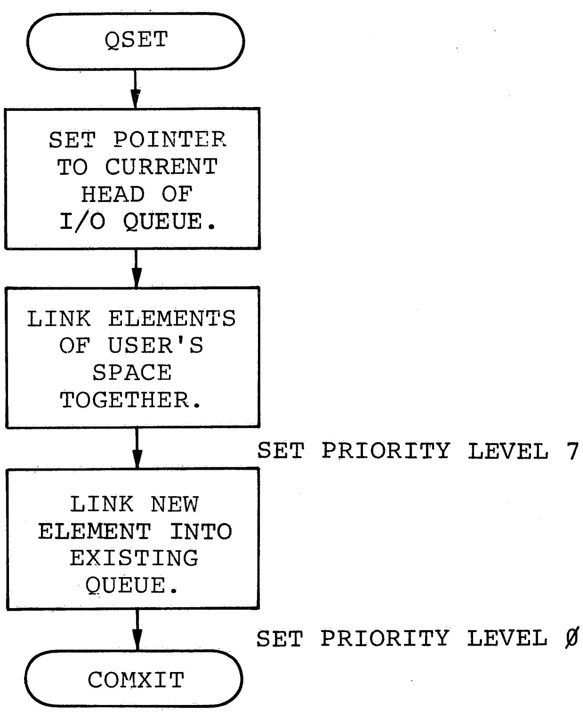
CLOSE



KLOSE
IS THIS MT/CT?
Y
SPESHL
GO TO HANDLER FOR THE CLOSE
DELOUT
N
BLKCHK
GET A DIRECTORY SEGMENT INTO MEMORY
START WITH SEGMENT #Ø.
2$
ENTRY
GET FIRST TENTATIVE ENTRY.
NO MORE
NXBLK
READ NEXT DIRECTORY SEGMENT
NO MORE
DELOUT
NO TENTATIVE OF THAT NAME WAS FOUND.
IS THIS THE RIGHT ONE?
N
INCR1
POINT TO NEXT ENTRY IN SEGMENT
2$
TENTATIVE NOT ASSOCIATED WITH OUR CHANNEL/JOB.
Y
CLOCOM
SAVE POINTERS TO THIS ENTRY.
CLSQSH
(I.E., PERMANENT FILE WITH SAME NAME)
IS THERE A PERM. ALREADY ON DEVICE ?
Y
MARK THE OLD ONE AS AN EMPTY.
ARE OLD & NEW IN THE SAME SEGMENT
N
CONSOLIDATE THE SEGMENT AND THEN REWRITE IT.
N
Y
GET THE CORRECT SEGMENT BACK IN MEMORY
A

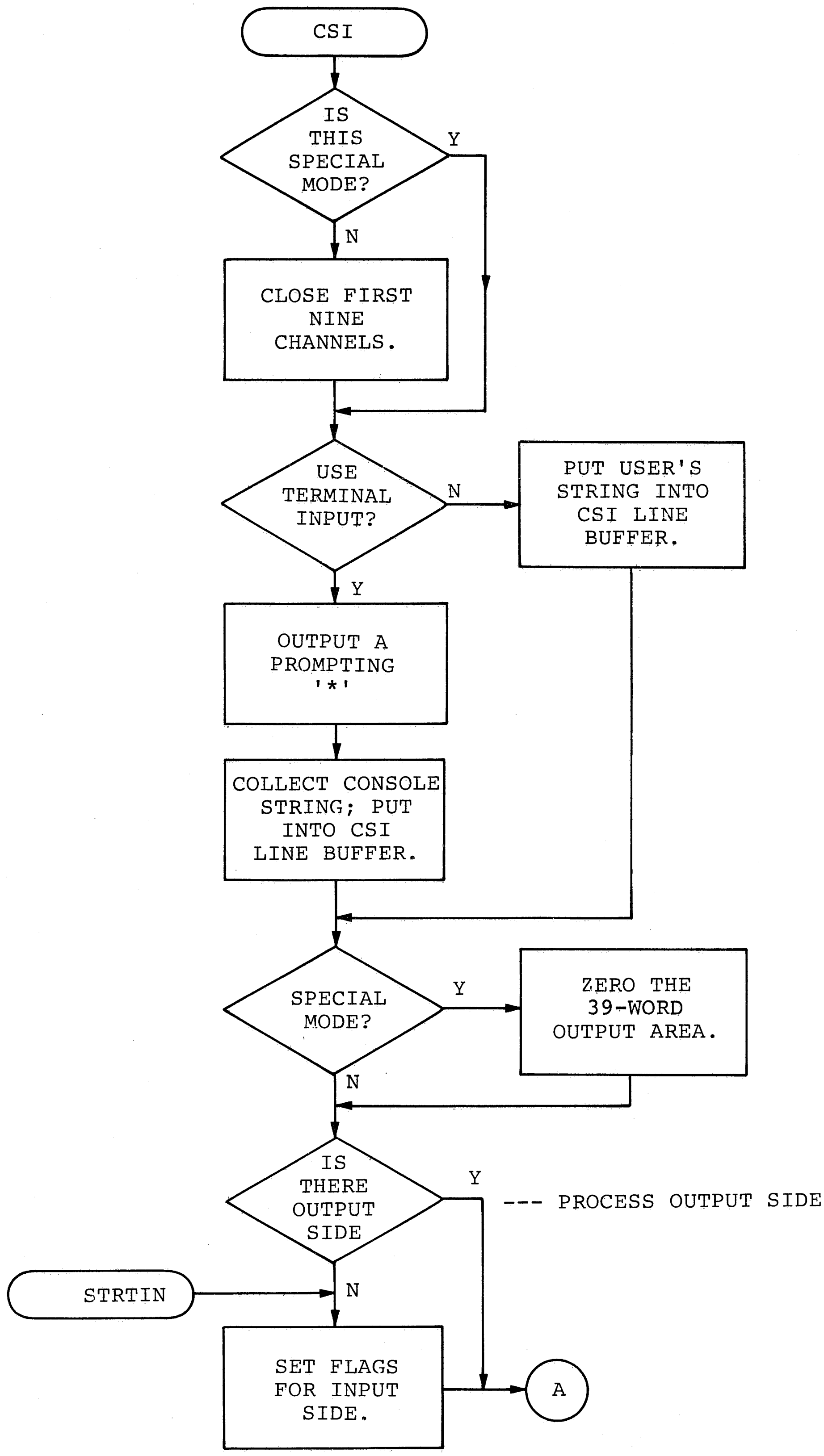


QUEUE EXTEND (QSET)

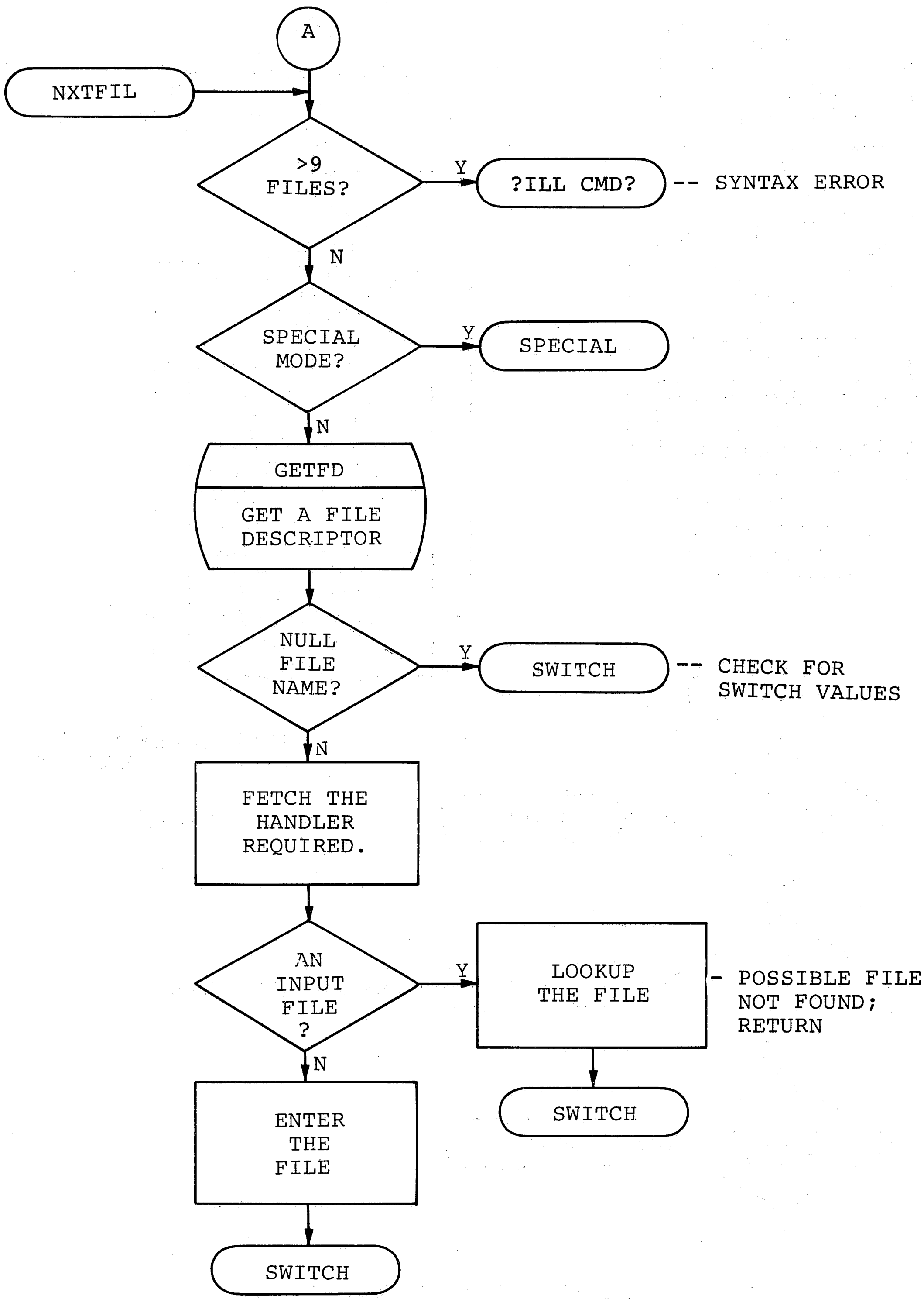


E.3 CSI (COMMAND STRING INTERPRETER) FLOWCHARTS

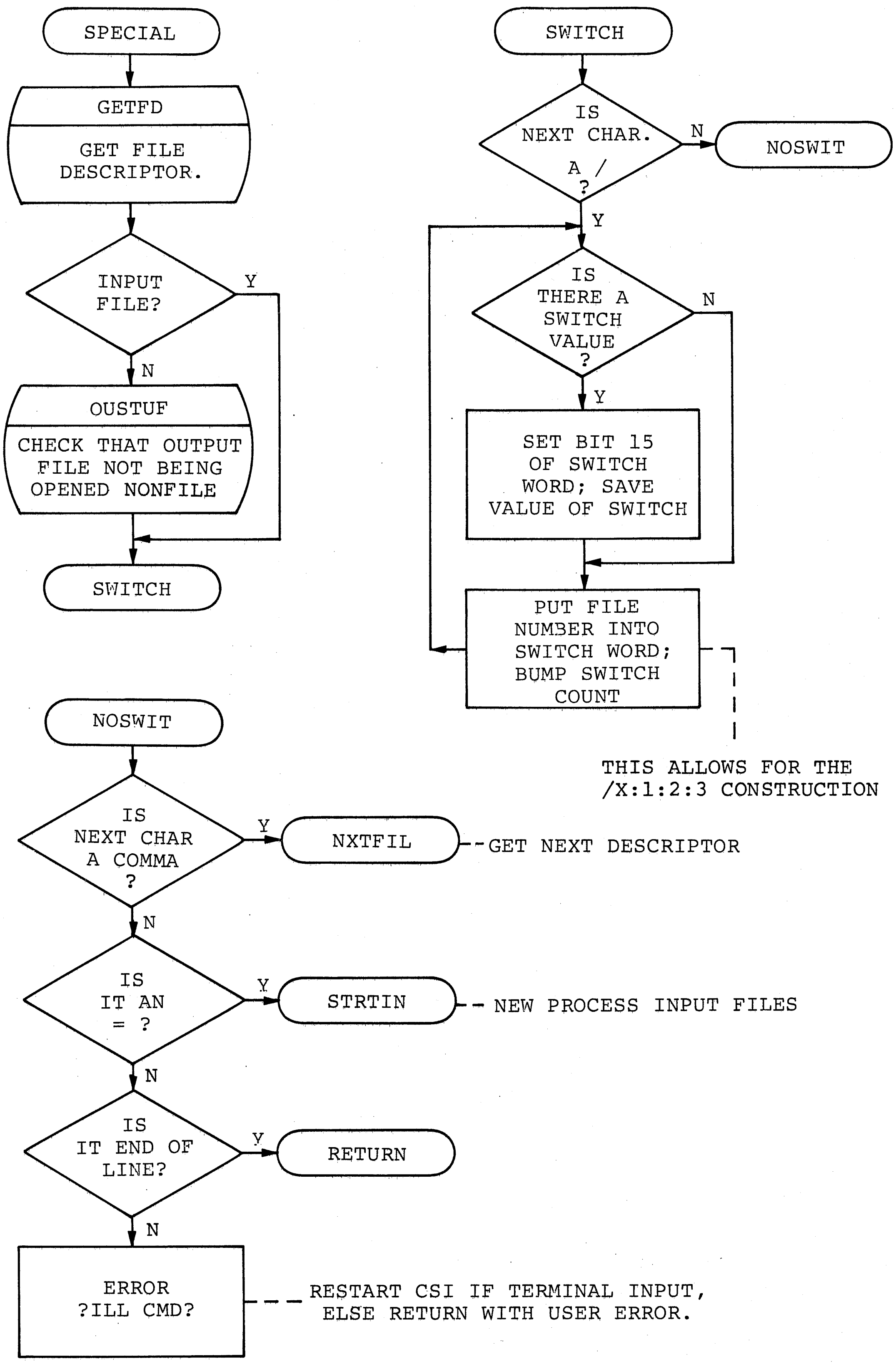
CSI CODE

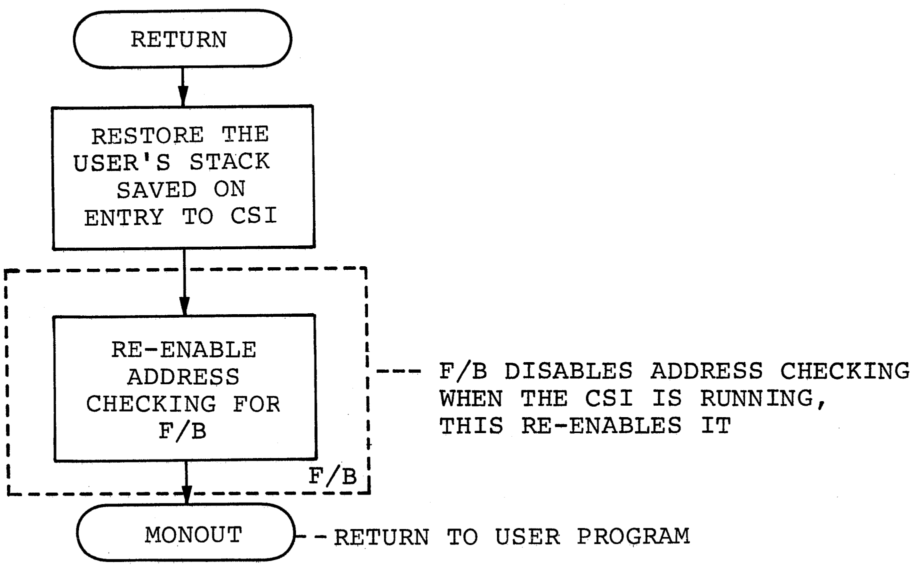


CSI
IS THIS SPECIAL MODE?
Y
N
CLOSE FIRST NINE CHANNELS.
USE TERMINAL INPUT?
N
PUT USER'S STRING INTO CSI LINE BUFFER.
Y
OUTPUT A PROMPTING '*'
COLLECT CONSOLE STRING; PUT INTO CSI LINE BUFFER.
SPECIAL MODE?
Y
ZERO THE 39-WORD OUTPUT AREA.
N
IS THERE OUTPUT SIDE
Y
--- PROCESS OUTPUT SIDE
STRTIN
N
SET FLAGS FOR INPUT SIDE.
A


A
NXTFIL
>9 FILES?
Y
?ILL CMD?
-- SYNTAX ERROR
N
SPECIAL MODE?
Y
SPECIAL
N
GETFD
GET A FILE DESCRIPTOR
NULL FILE NAME?
Y
SWITCH
-- CHECK FOR SWITCH VALUES
N
FETCH THE HANDLER REQUIRED.
AN INPUT FILE ?
Y
LOOKUP THE FILE
- POSSIBLE FILE NOT FOUND; RETURN
SWITCH
N
ENTER THE FILE
SWITCH

CSI CODE (CONT.)



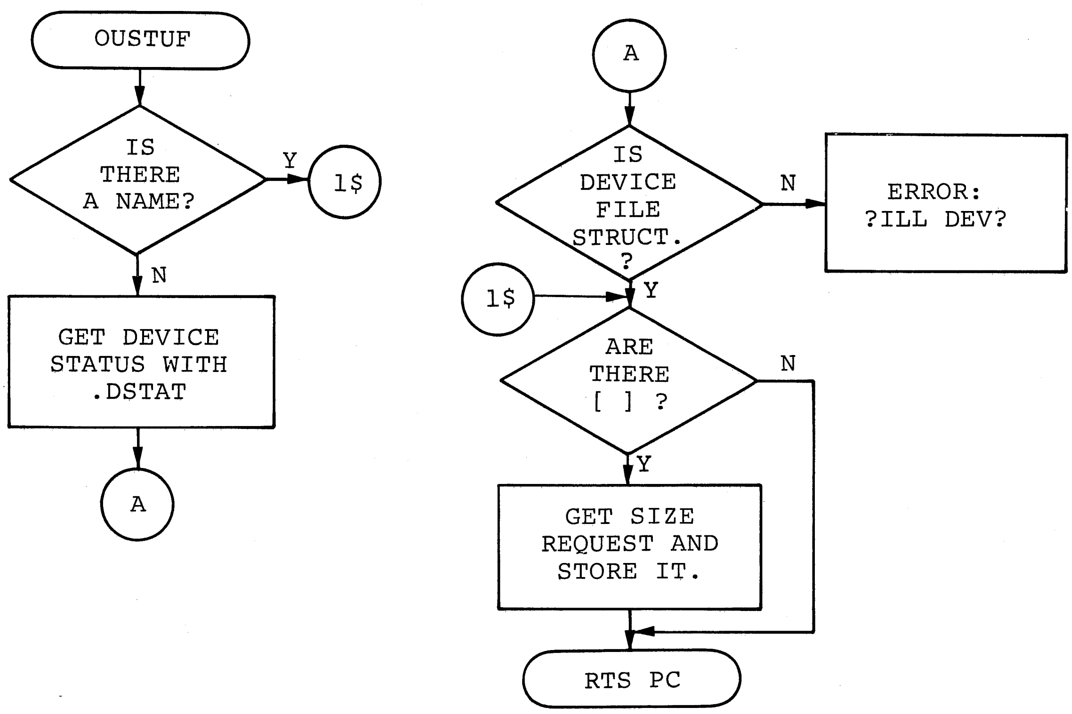

RETURN
RESTORE THE USER'S STACK SAVED ON ENTRY TO CSI
RE-ENABLE ADDRESS CHECKING FOR F/B
F/B
F/B DISABLES ADDRESS CHECKING WHEN THE CSI IS RUNNING, THIS RE-ENABLES IT
MONOUT
RETURN TO USER PROGRAM

E.3.1 CSI Subroutines

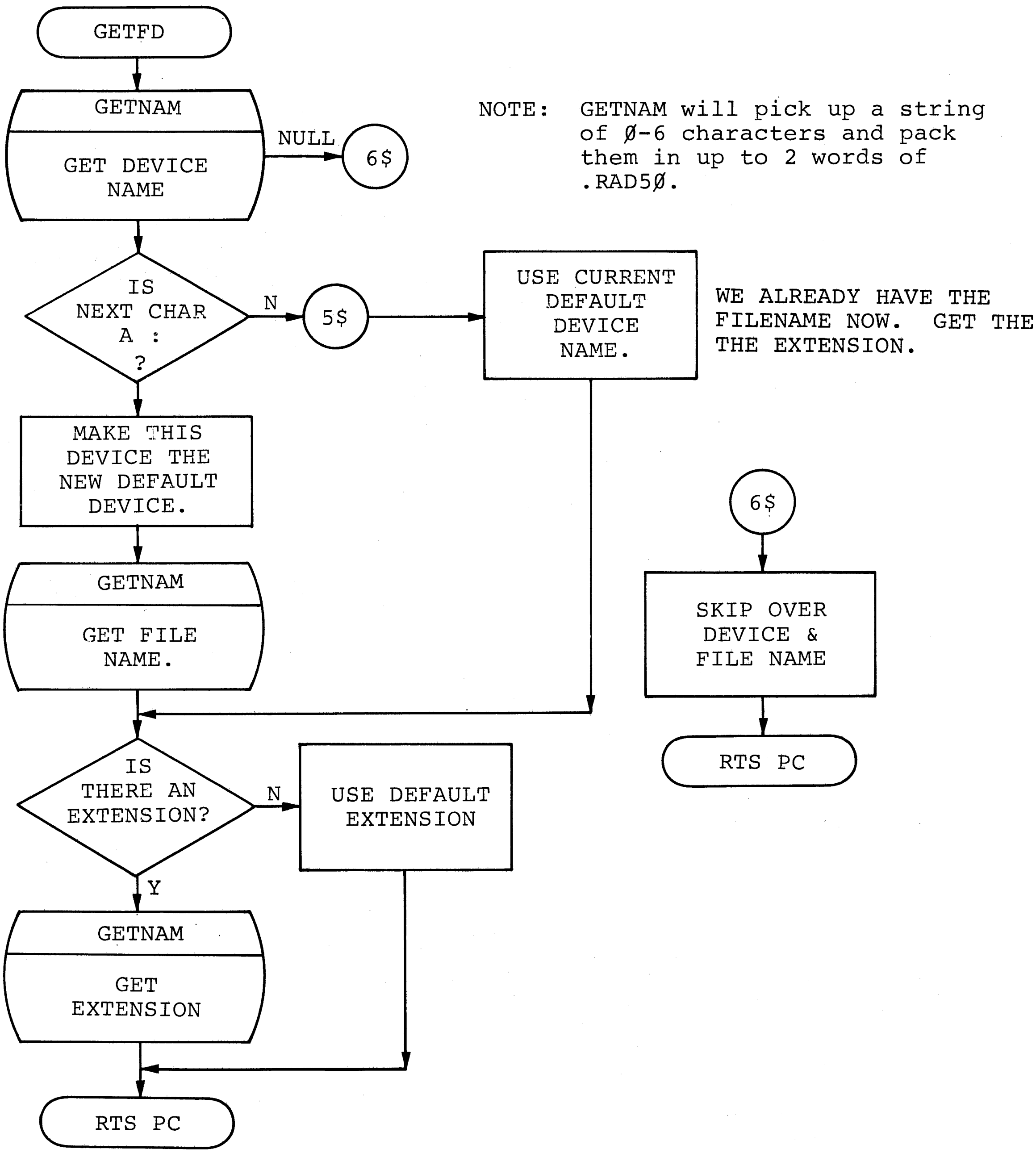
These subroutines are used by the CSI, and, in certain cases by the KMON.

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.



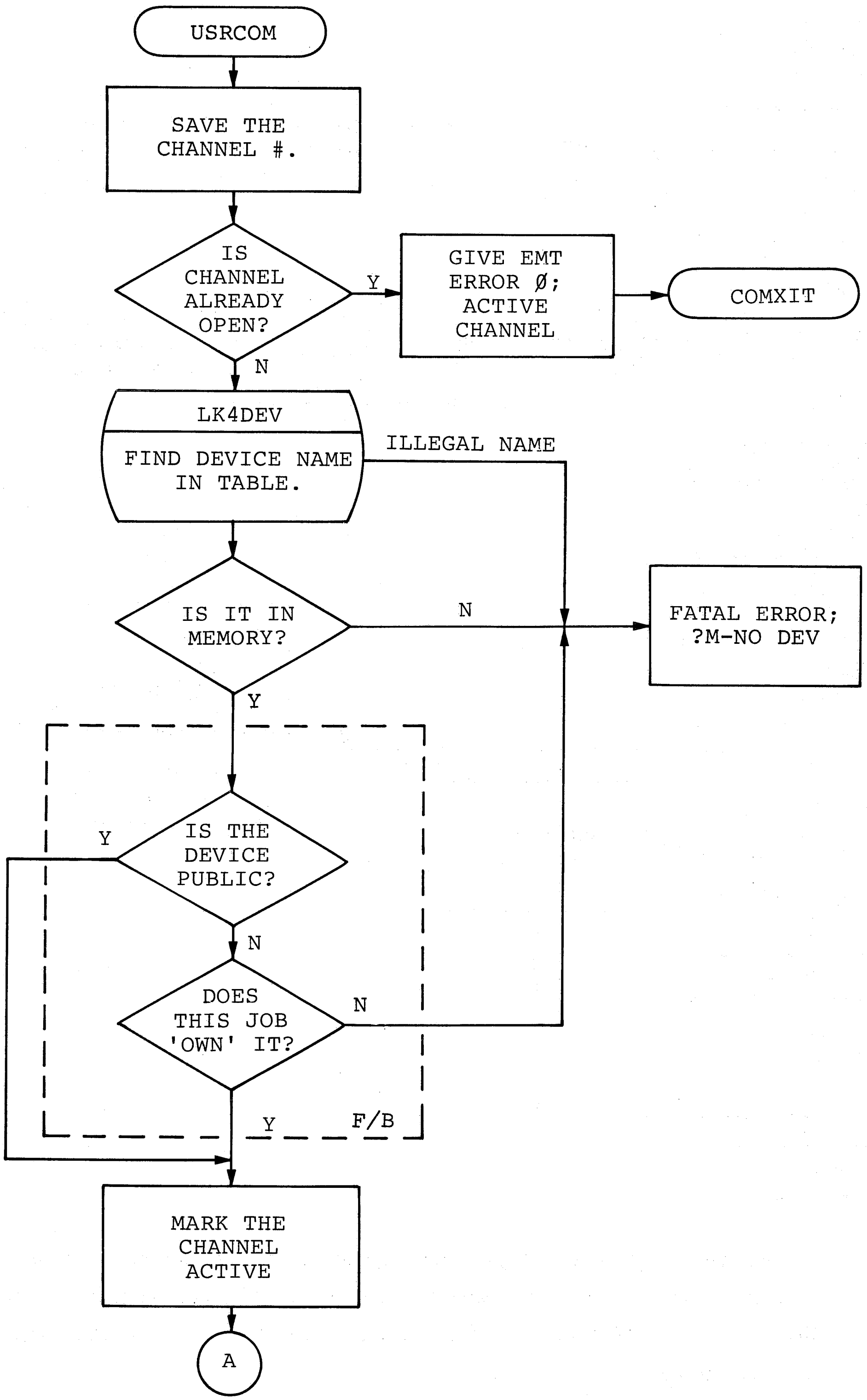
GETFD - Picks up a file descriptor (DEV:FILE.EXT) from an input string and packs it in 4 words of .RAD5Ø.

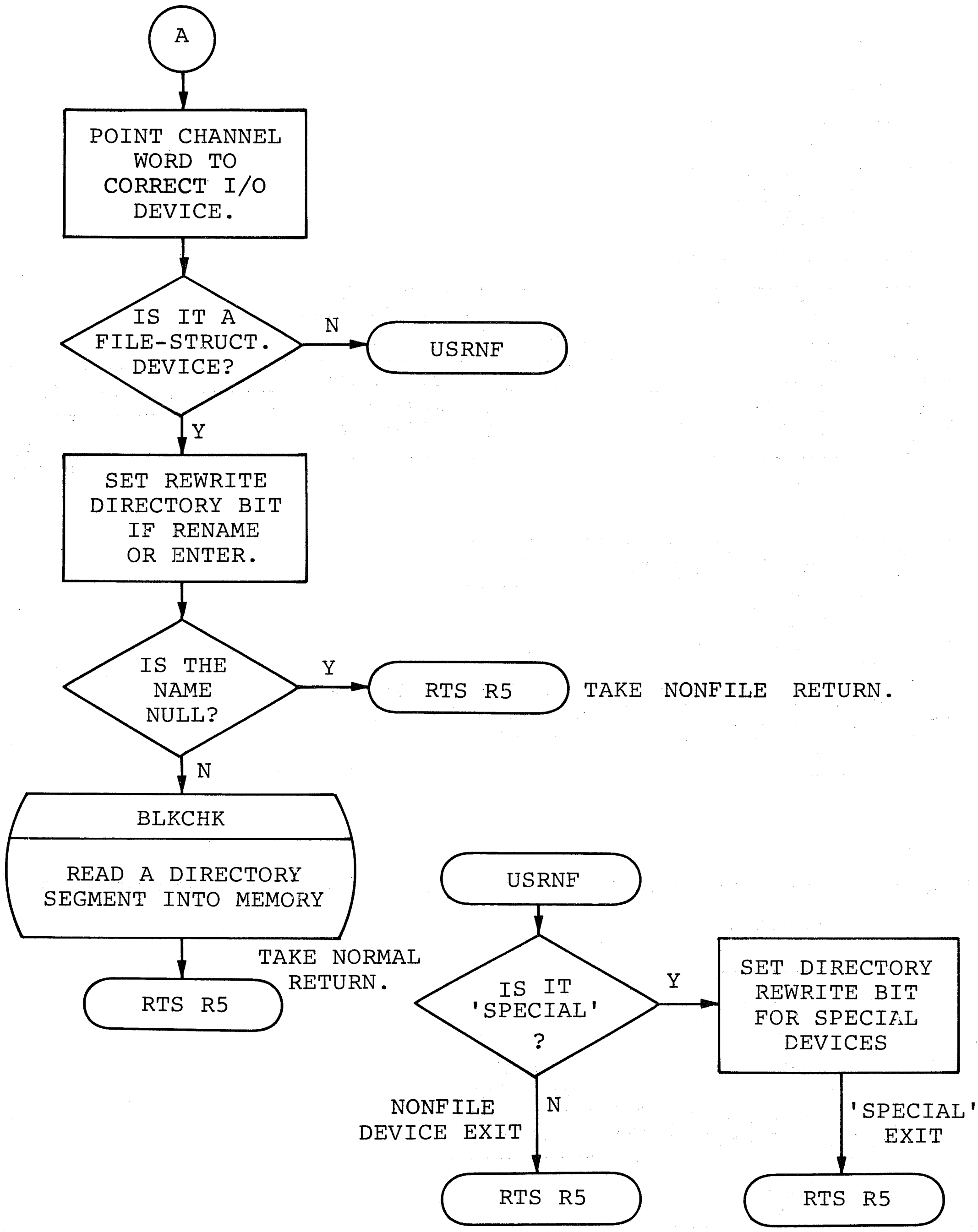


GETNAM - Converts a string of Ø-6 alphanumeric characters to a 2-word RAD5Ø group. The two words are zero filled when necessary. See code at GETNAM in the source listing if greater detail is necessary.

USRCOM

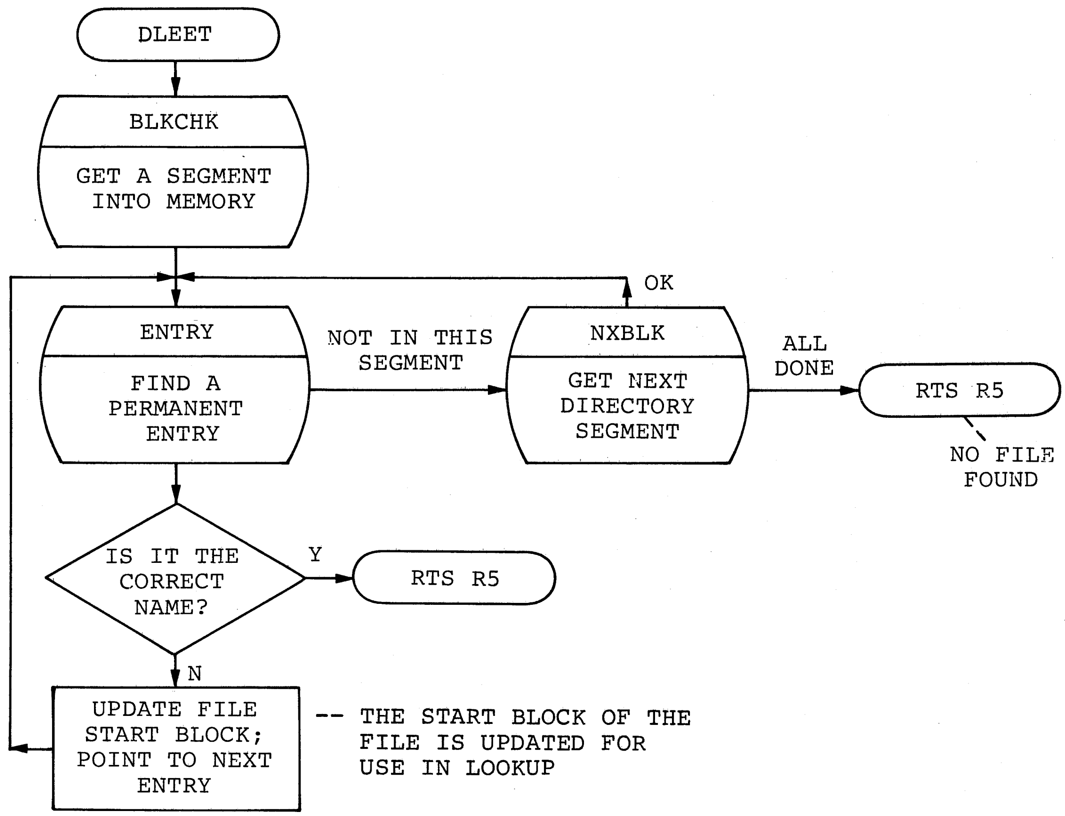
USRCOM - This routine is used to prepare a channel for I/O operations.



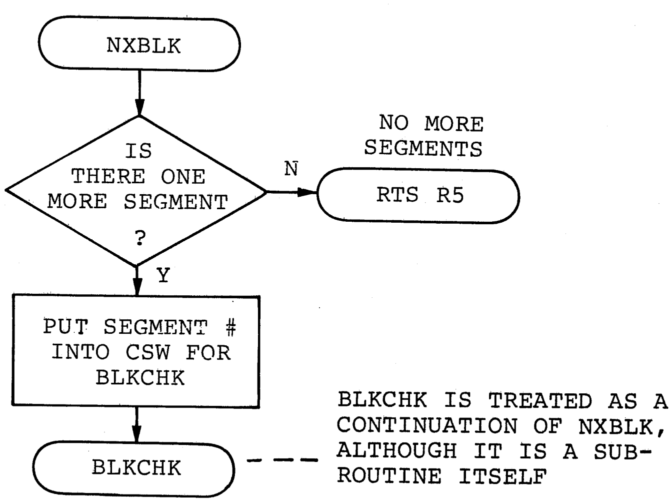

A
POINT CHANNEL WORD TO CORRECT I/O DEVICE.
IS IT A FILE-STRUCT. DEVICE?
N
USRNF
Y
SET REWRITE DIRECTORY BIT IF RENAME OR ENTER.
IS THE NAME NULL?
Y
RTS R5
TAKE NONFILE RETURN.
N
BLKCHK
READ A DIRECTORY SEGMENT INTO MEMORY
TAKE NORMAL RETURN.
RTS R5
USRNF
IS IT 'SPECIAL' ?
Y
SET DIRECTORY REWRITE BIT FOR SPECIAL DEVICES
N
NONFILE DEVICE EXIT
RTS R5
'SPECIAL' EXIT
RTS R5

DLEET/NXBLK

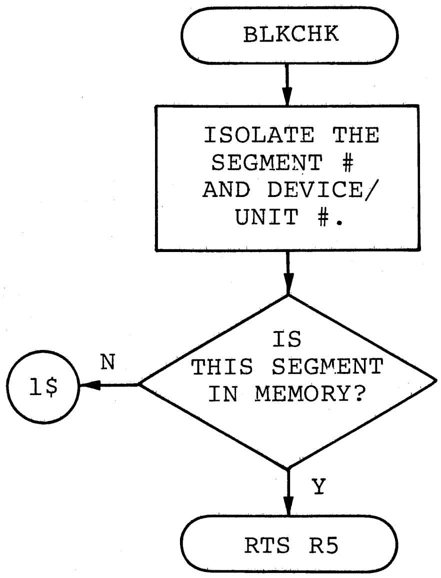
DLEET - This routine scans a device directory to find a file of a specified name.



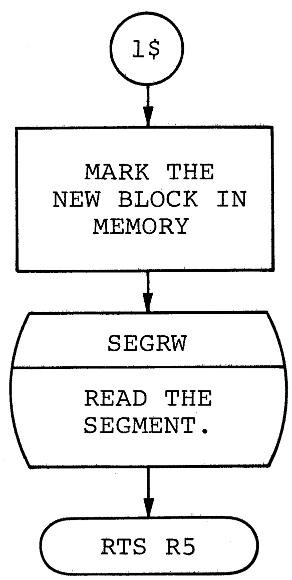
NXBLK - Gets the next in the series of directory segments, if one exists.



BLKCHK - 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.



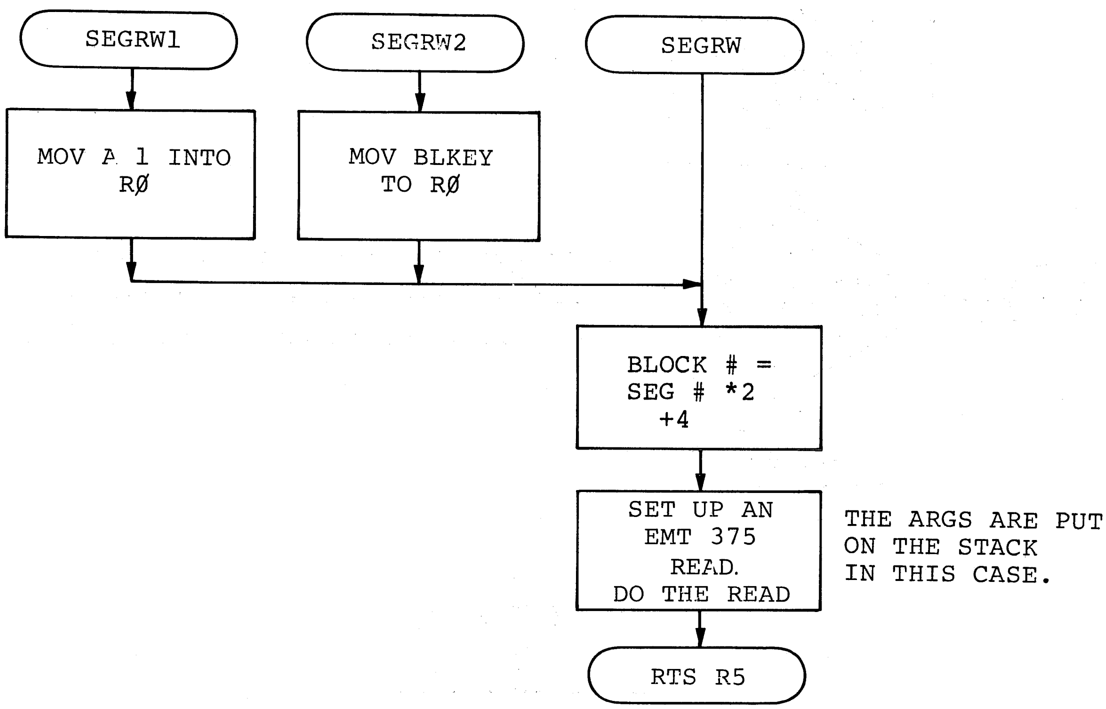
SEGRW

SEGRW - Segment Read/Write. This routine read/writes selected directory segments. There are three entry points:

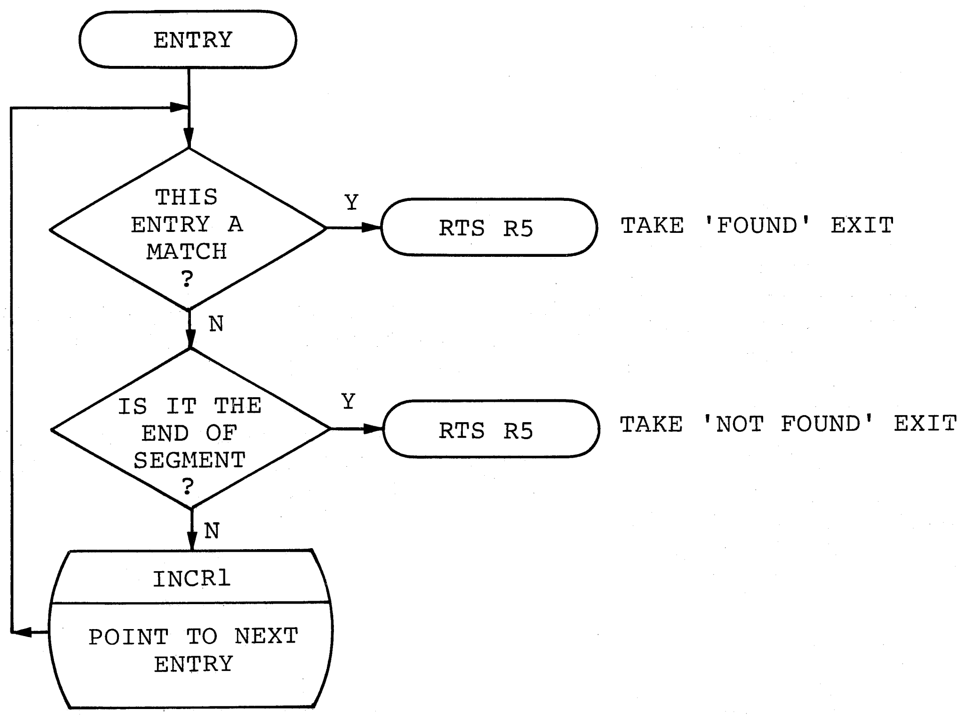
SEGRW1: Use segment #1

SEGRW2: Use the segment currently in memory (BLKEY)

SEGRW: Use the number in RØ 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:

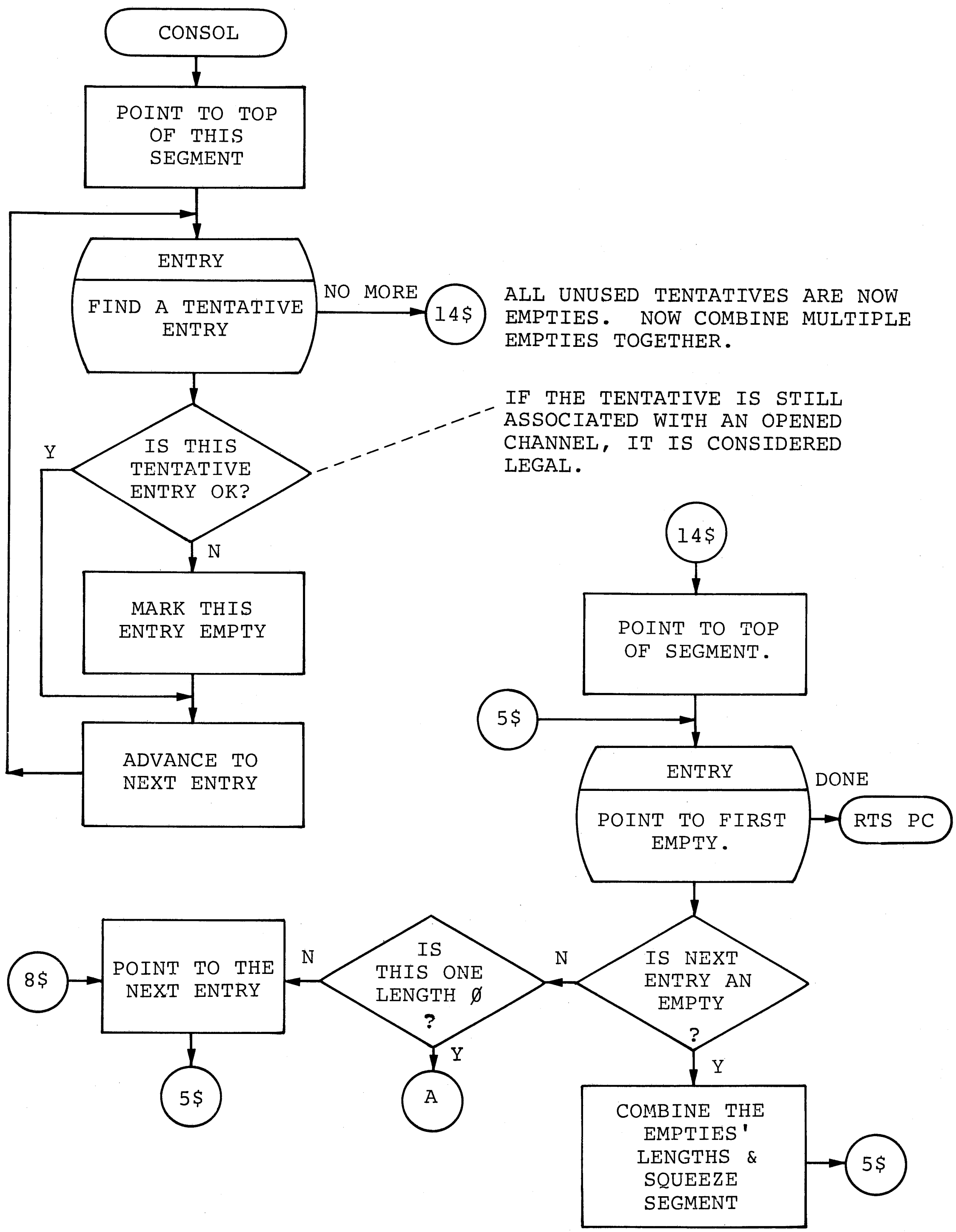
```
JSR R5,COMERR
code
```

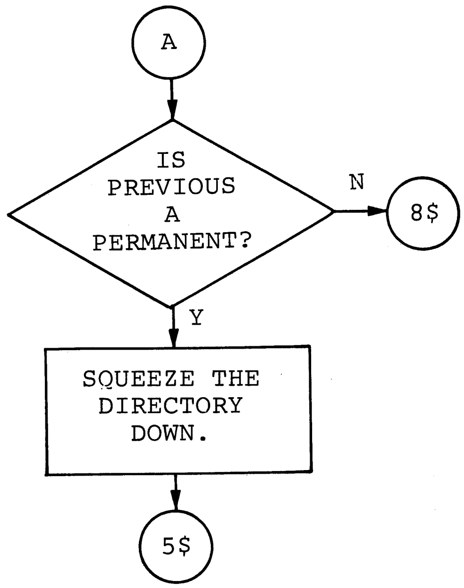
Code is used to indicate which error is to be generated. If .SERR is in effect, control passes to COMXIT, which returns to RMON.

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.

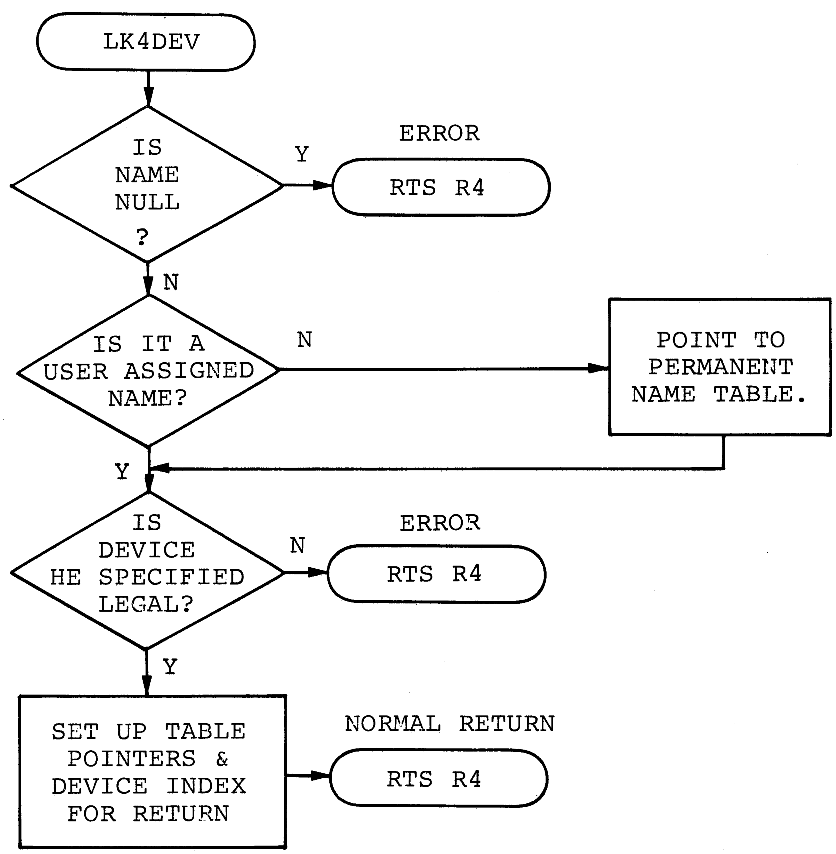
CONSOL

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.





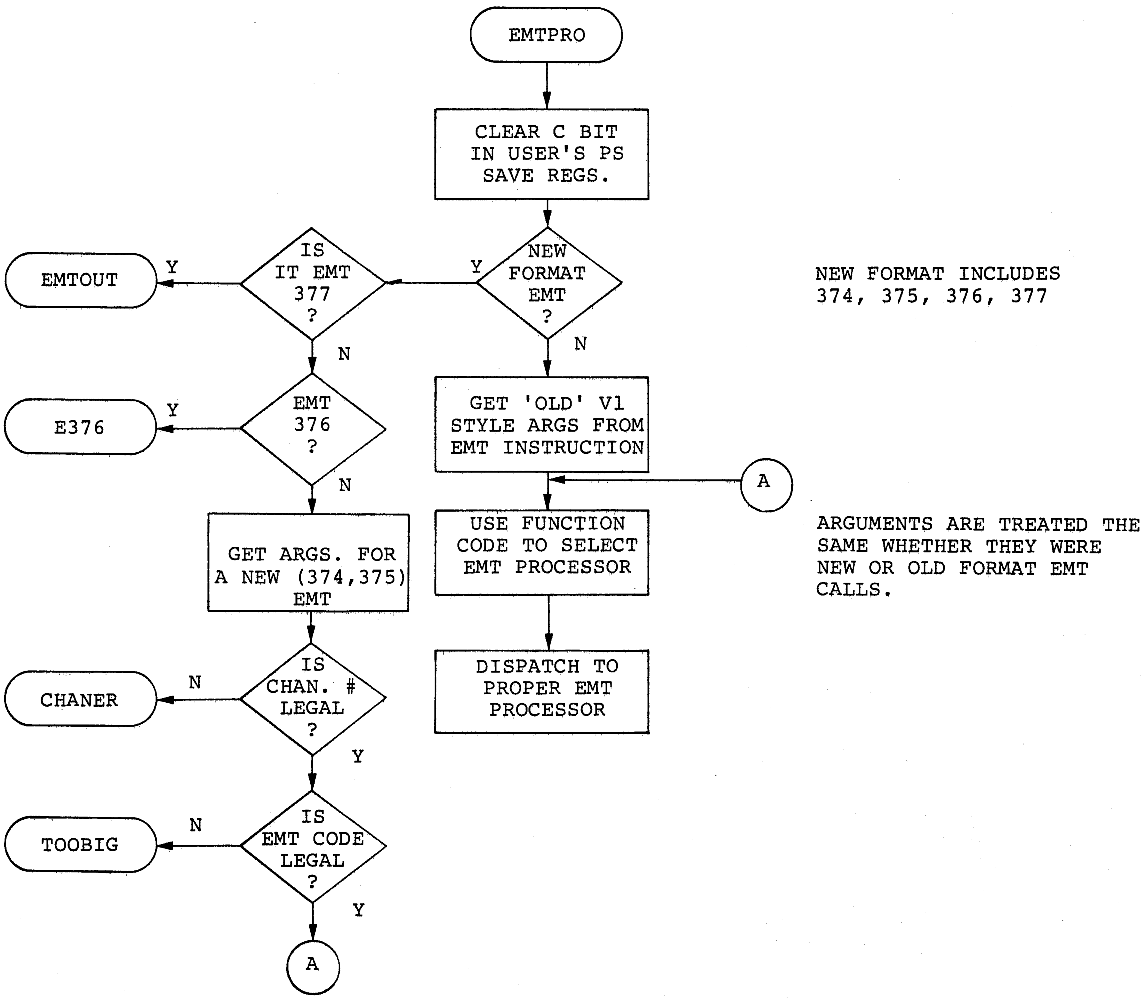
LK4DEV - This routine looks up a specified device name in the system tables. It first attempts to fine the name in the user assigned name table; failing that, the permanent name table is searched.

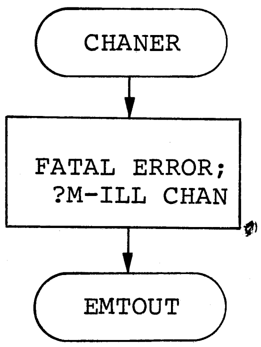


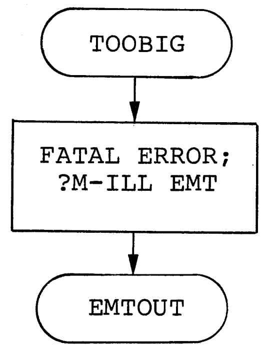
E.4 RMON (RESIDENT MONITOR) FLOWCHARTS FOR SINGLE-JOB MONITOR

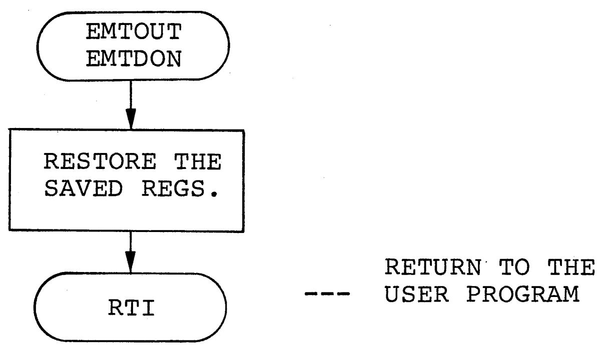
EMT DISPATCHER

The code of the EMT dispatcher is entered when an EMT instruction is executed. The EMT instruction is decoded and control passes to the approproate code for processing.









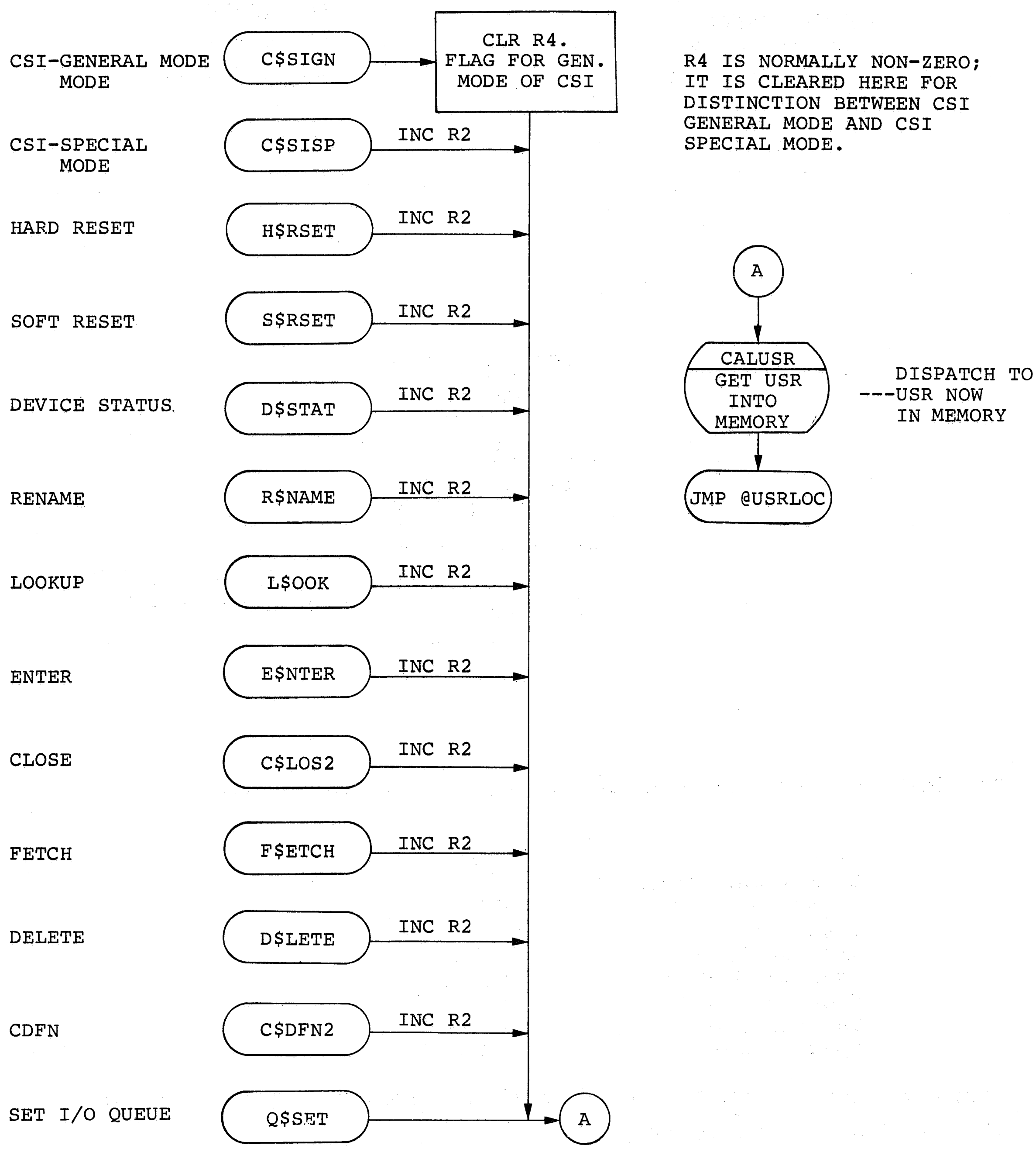
The following EMT requests are no-ops in the S/J Monitor:

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

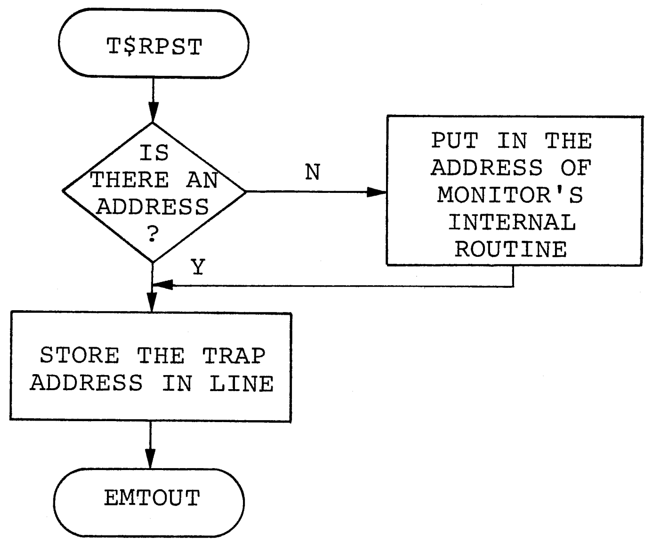
The USR Dispatch code handles dispatching those EMT's which require the USR. At each entry point, an INC R2 is performed. Thus, R2 acts as a function identifier once the USR is entered.



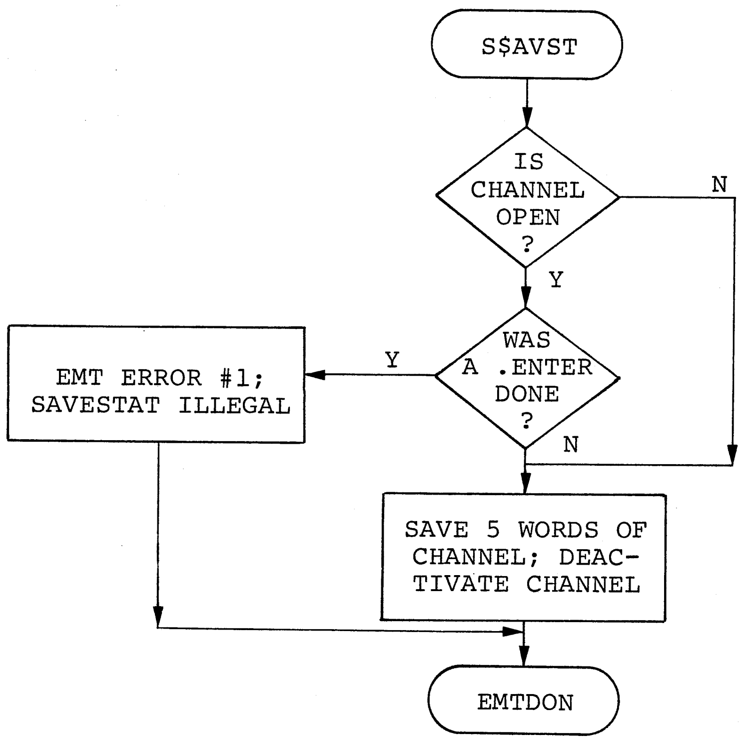
E.4.1 EMT Processors

SET TRAP/SAVE STATUS

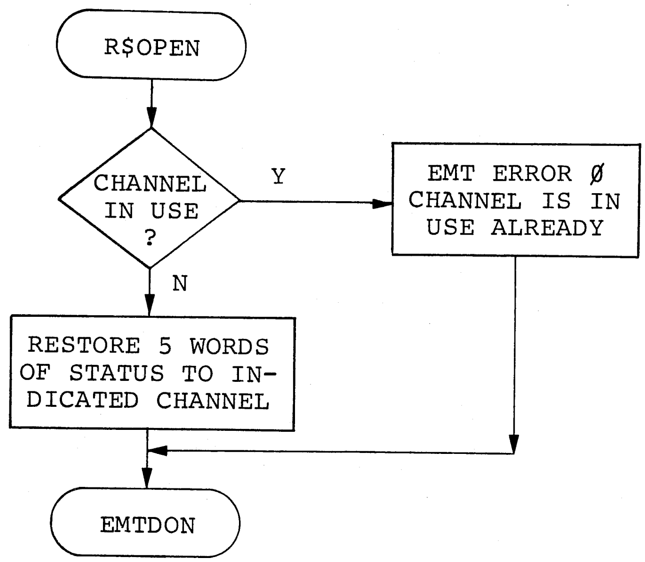
SET TRAP ADDRESS



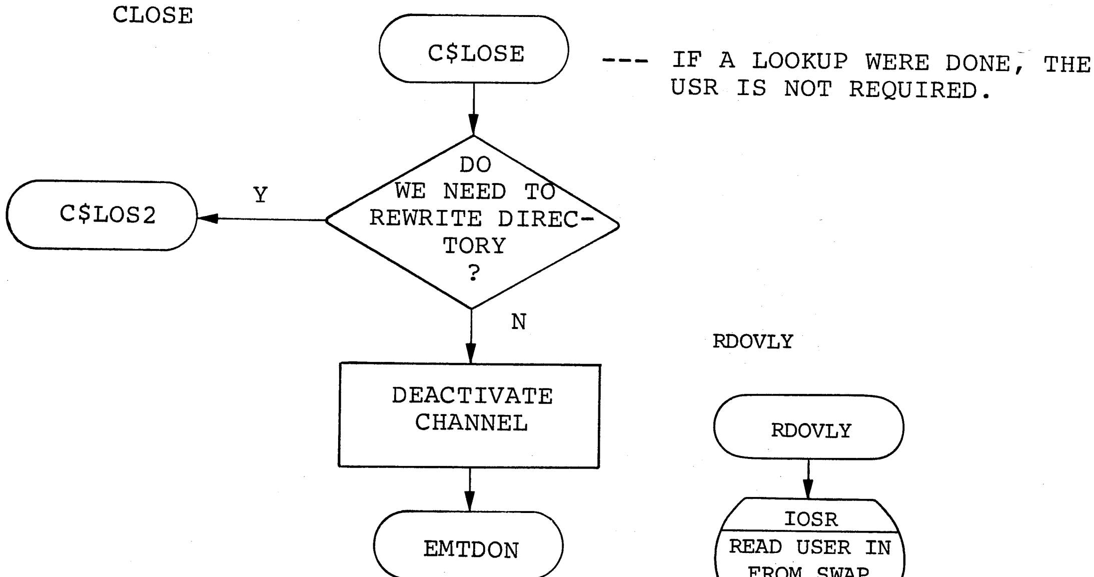
SAVESTATUS



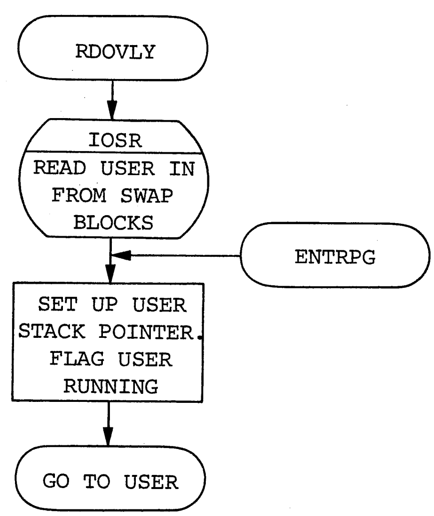
REOPEN



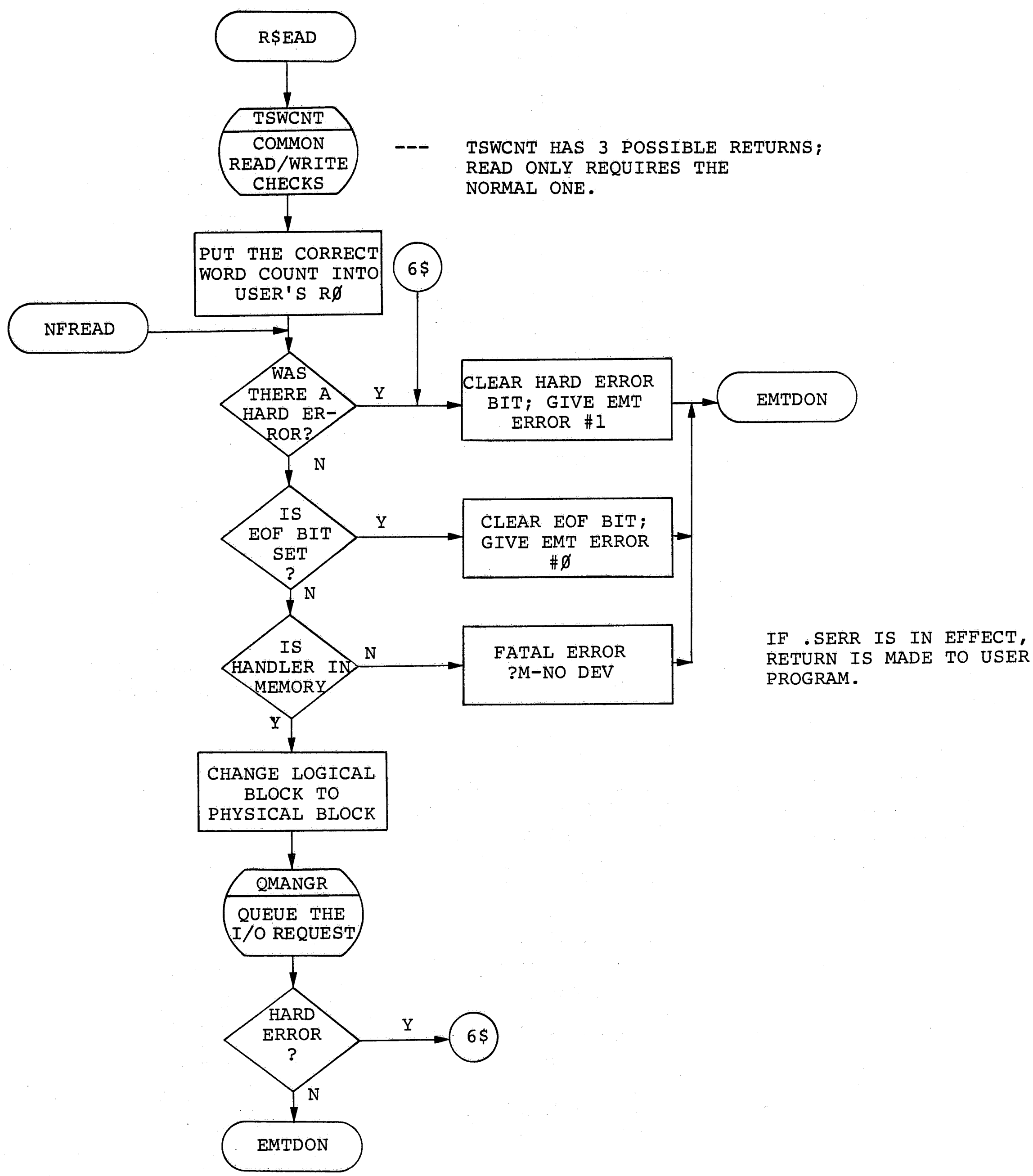
CLOSE

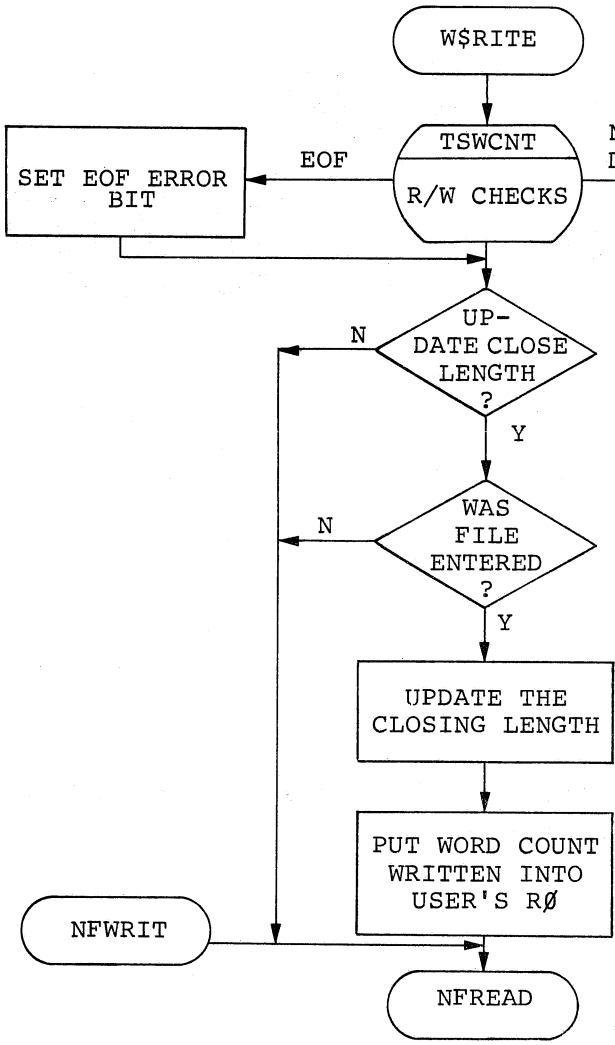


RDOVLY



READ

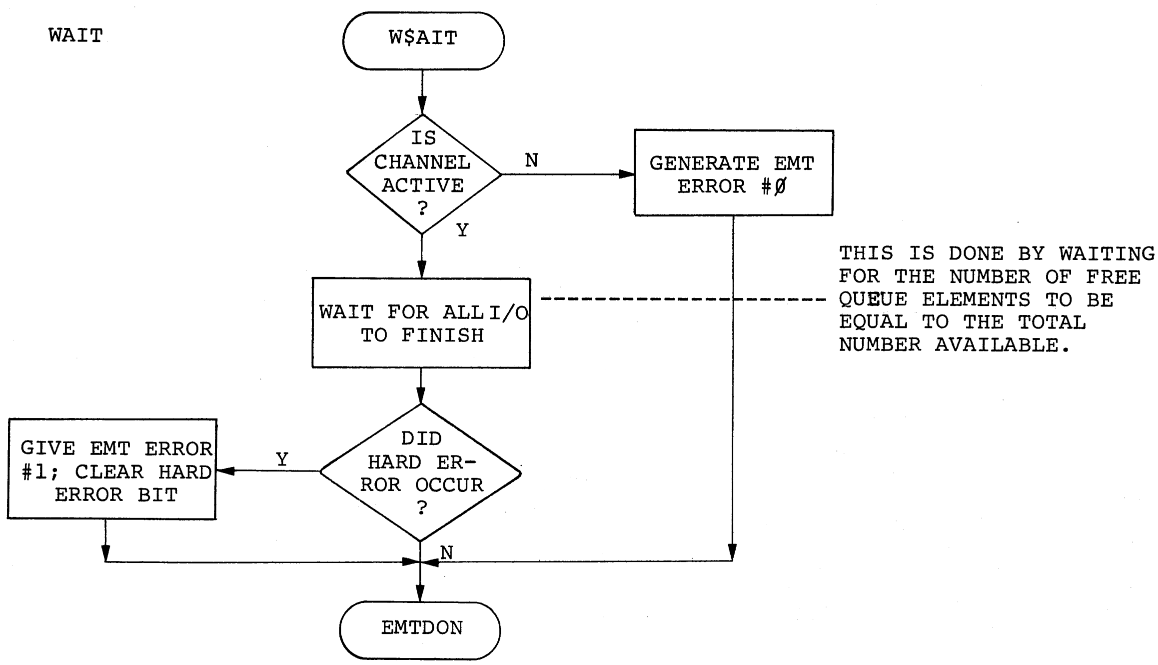

R$EAD
TSWCNT
COMMON READ/WRITE CHECKS
--- TSWCNT HAS 3 POSSIBLE RETURNS; READ ONLY REQUIRES THE NORMAL ONE.
PUT THE CORRECT WORD COUNT INTO USER'S RØ
6$
NFREAD
WAS THERE A HARD ERROR?
Y
N
CLEAR HARD ERROR BIT; GIVE EMT ERROR #1
EMTDON
IS EOF BIT SET ?
Y
N
CLEAR EOF BIT; GIVE EMT ERROR #Ø
IS HANDLER IN MEMORY
N
Y
FATAL ERROR ?M-NO DEV
IF .SERR IS IN EFFECT, RETURN IS MADE TO USER PROGRAM.
CHANGE LOGICAL BLOCK TO PHYSICAL BLOCK
QMANGR
QUEUE THE I/O REQUEST
HARD ERROR ?
Y
6$
N
EMTDON



IF THE LAST BLOCK BEING WRITTEN IS > THE CURRENT LAST BLOCK WRITTEN, CHANGE THE CLOSING FILE LENGTH.

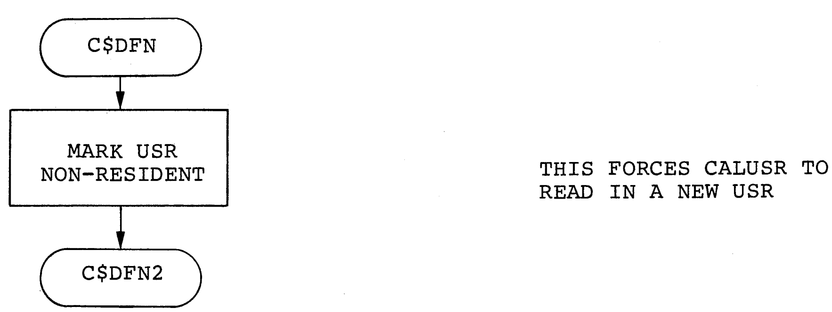
WAIT/CDFN

WAIT

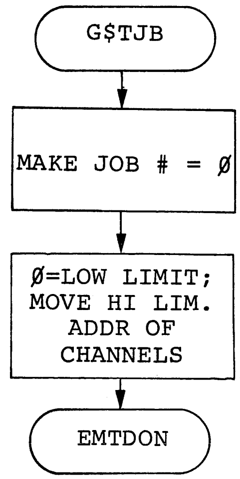


CDFN

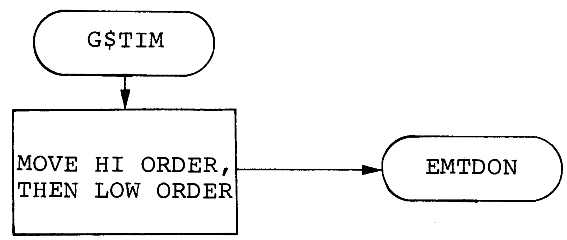
Channel Define - the resident portion of CDFN cuases a fresh copy of the USR to be read in, then enters the USR.



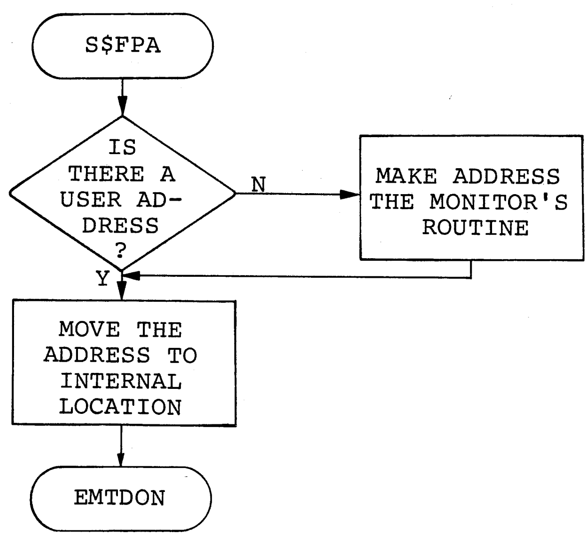
GET JOB PARAMETERS



GET TIME OF DAY



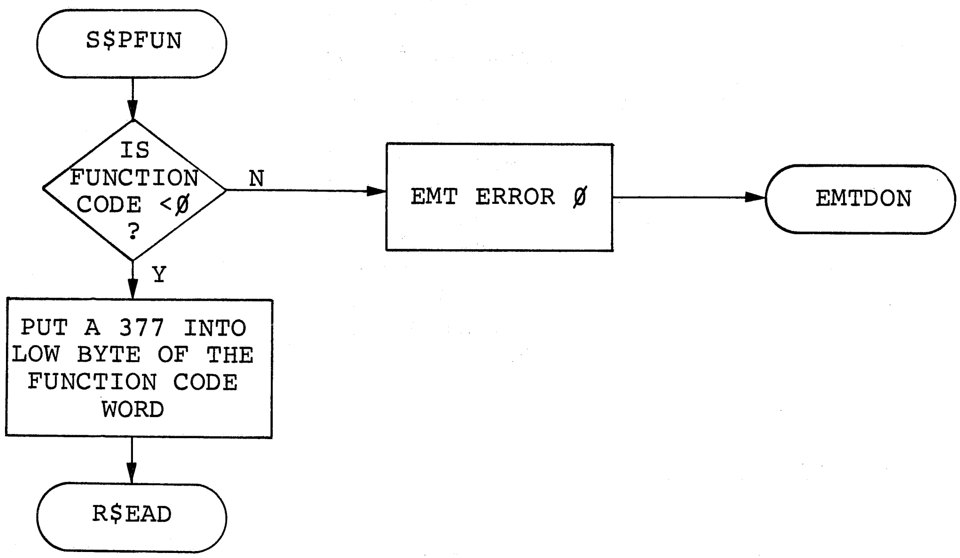
SET FPP EXCEPTION



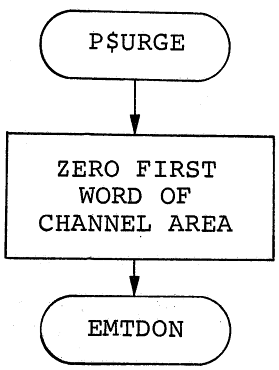
SPECIAL FUNCTIONS/PURGE
SOFT/HARD ERRORS

SPECIAL FUNCTIONS/PURGE
SOFT/HARD ERRORS

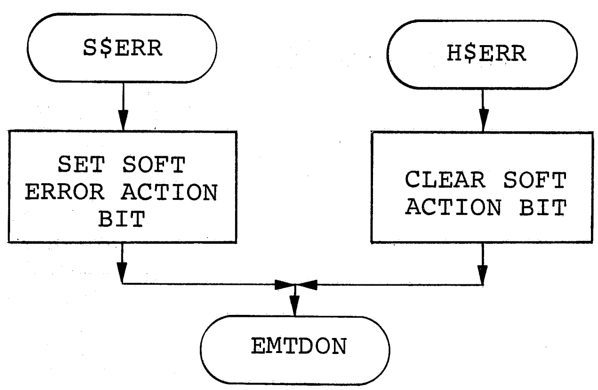
SPECIAL FUNCTIONS (MAGTAPE/CASSETTE)



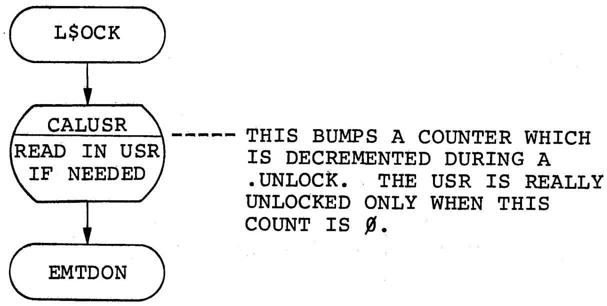
PURGE



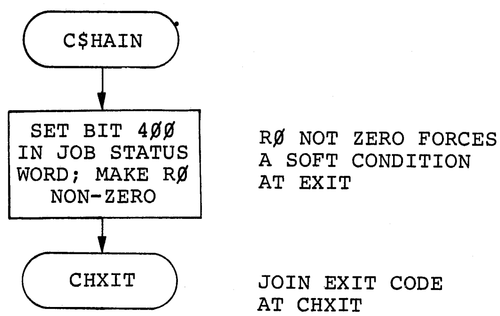
SOFT/HARD ERRORS



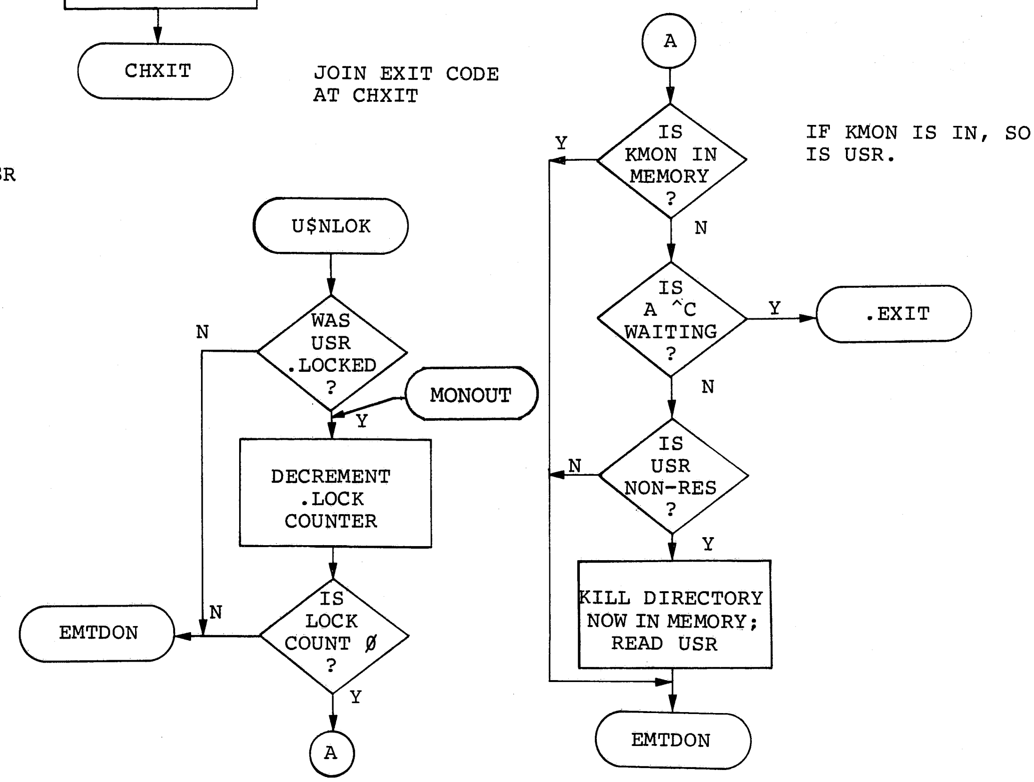
LOCK USR



CHAIN

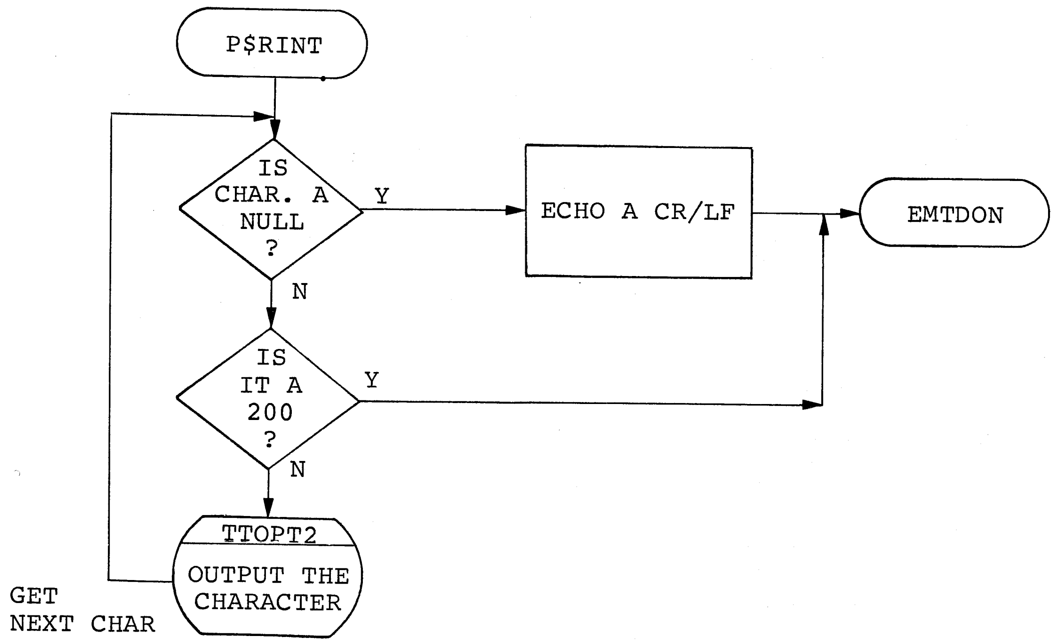


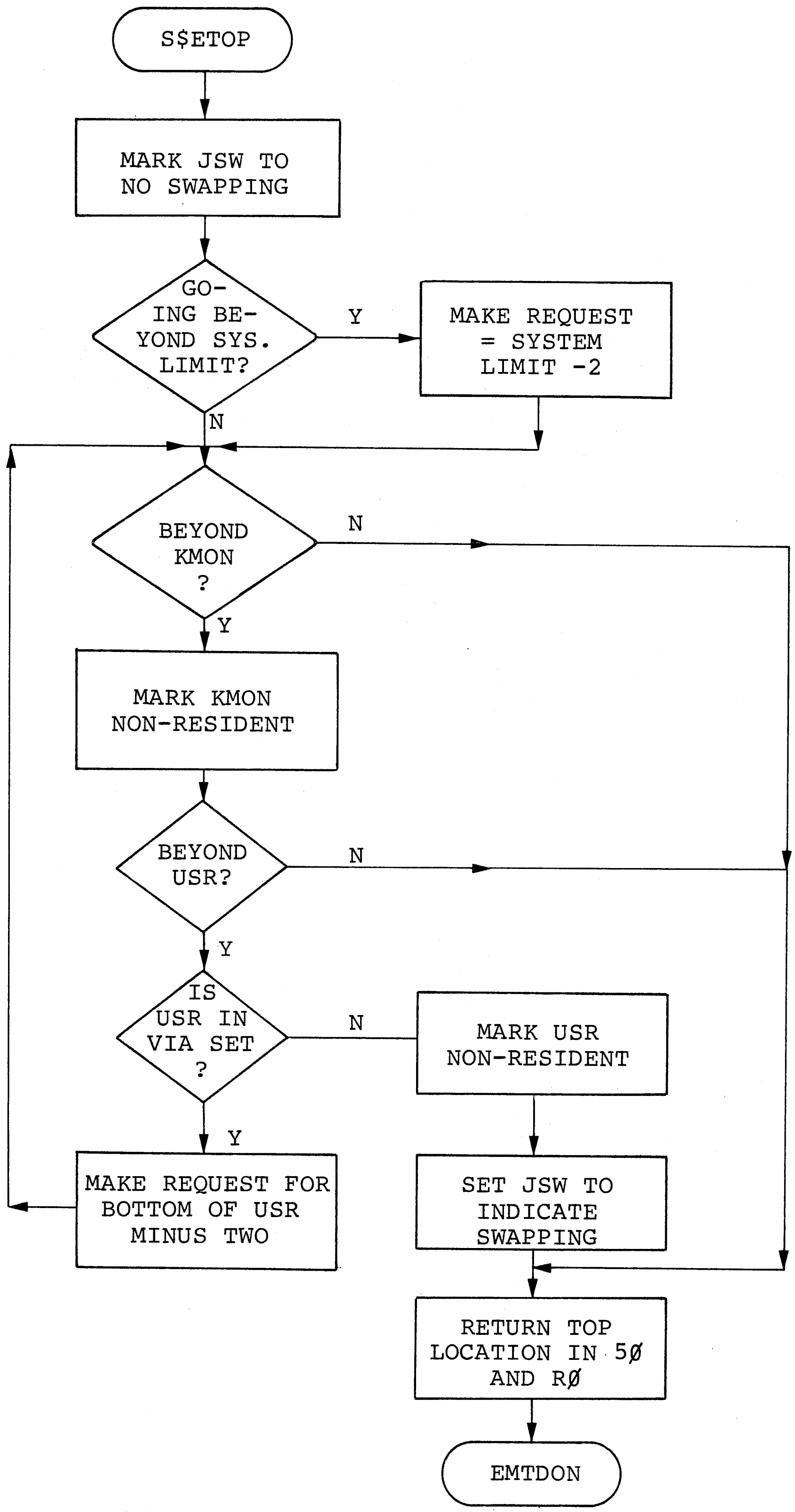
UNLOCK USR



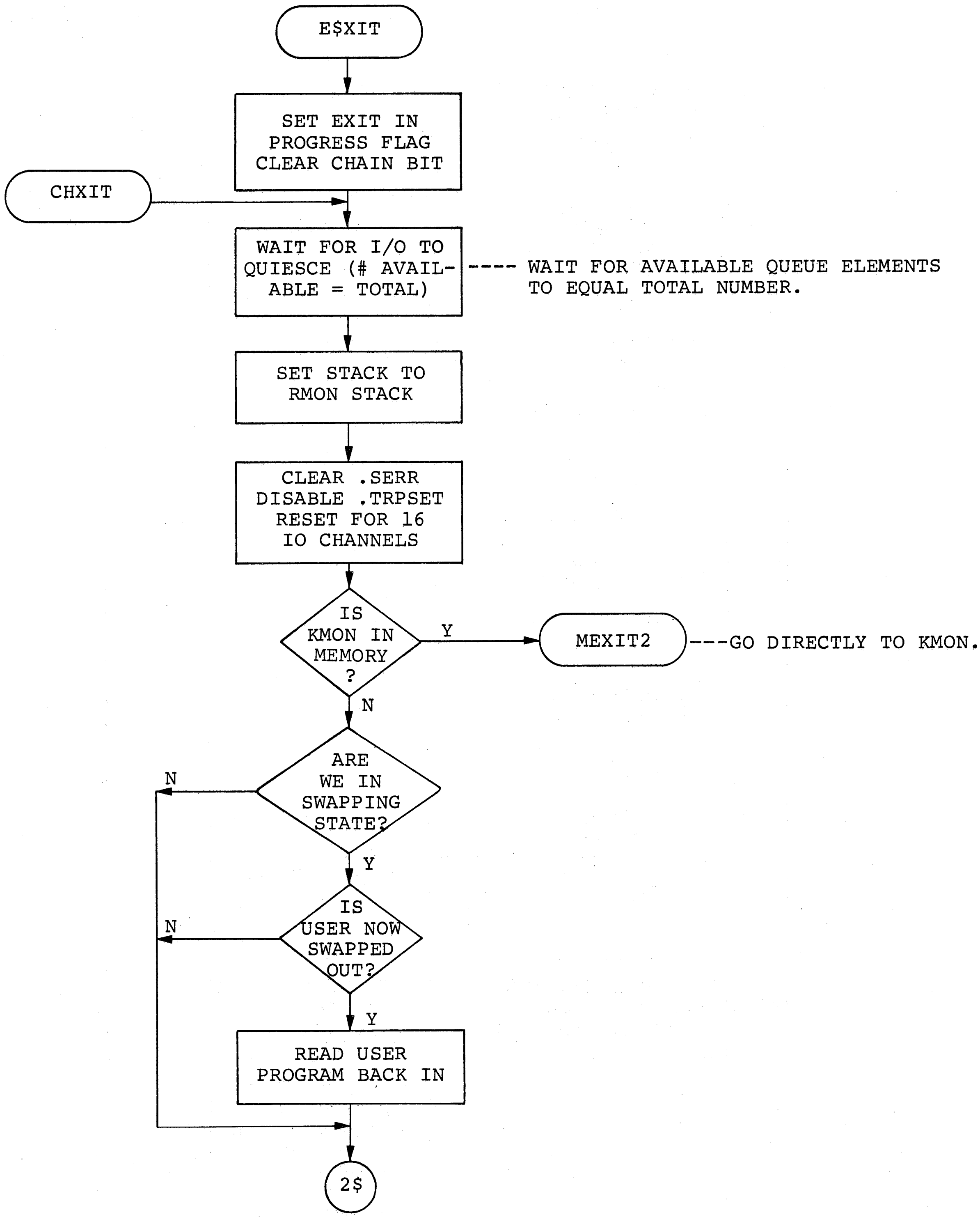
PRINT

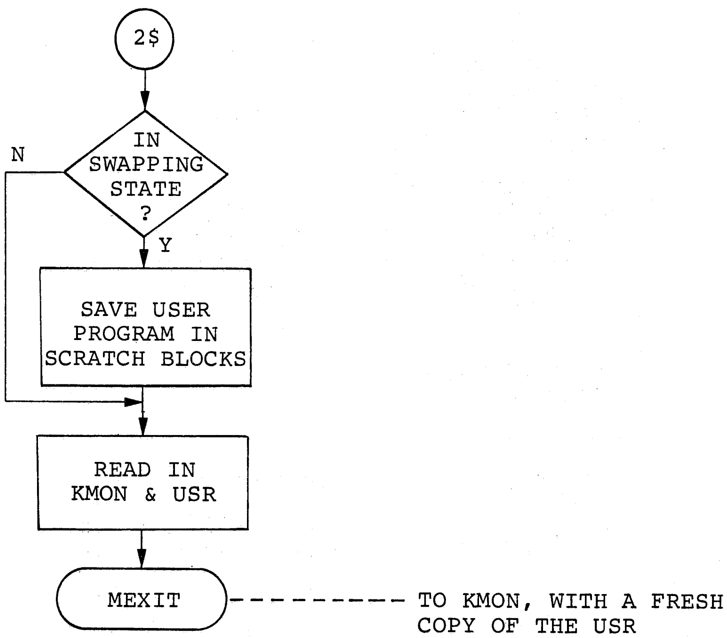
PRINT - Causes a line to be output to the console terminal.



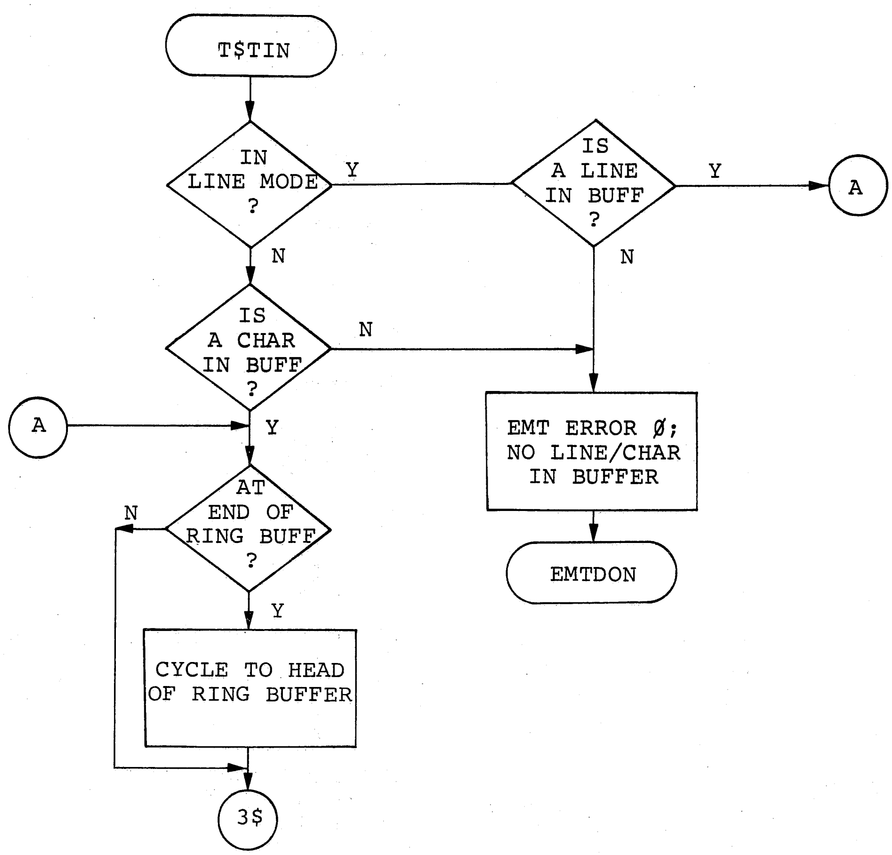

S$ETOP
MARK JSW TO NO SWAPPING
GO-ING BE-YOND SYS. LIMIT?
Y
MAKE REQUEST = SYSTEM LIMIT -2
N
BEYOND KMON ?
N
Y
MARK KMON NON-RESIDENT
BEYOND USR?
N
Y
IS USR IN VIA SET ?
N
MARK USR NON-RESIDENT
Y
MAKE REQUEST FOR BOTTOM OF USR MINUS TWO
SET JSW TO INDICATE SWAPPING
RETURN TOP LOCATION IN 5Ø AND RØ
EMTDON

EXIT

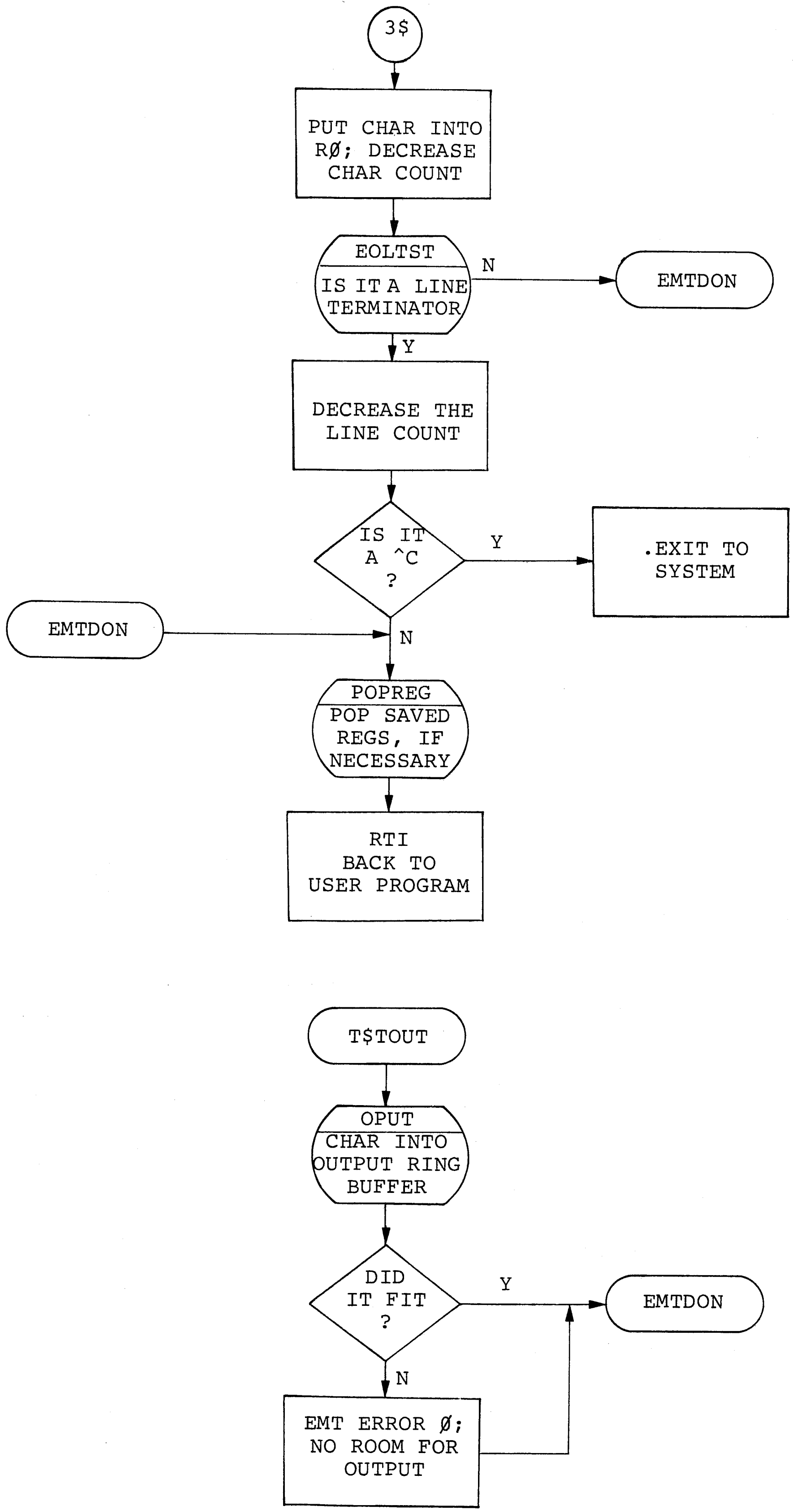

E$XIT
SET EXIT IN PROGRESS FLAG CLEAR CHAIN BIT
CHXIT
WAIT FOR I/O TO QUIESCE (# AVAILABLE = TOTAL)
---- WAIT FOR AVAILABLE QUEUE ELEMENTS TO EQUAL TOTAL NUMBER.
SET STACK TO RMON STACK
CLEAR .SERR
DISABLE .TRPSET
RESET FOR 16 IO CHANNELS
IS KMON IN MEMORY ?
Y
MEXIT2
---GO DIRECTLY TO KMON.
N
ARE WE IN SWAPPING STATE?
N
Y
IS USER NOW SWAPPED OUT?
N
Y
READ USER PROGRAM BACK IN
2$


2$
IN SWAPPING STATE ?
N
Y
SAVE USER PROGRAM IN SCRATCH BLOCKS
READ IN KMON & USR
MEXIT
TO KMON, WITH A FRESH COPY OF THE USR

TTYIN


T$TIN
IN LINE MODE ?
Y
N
IS A LINE IN BUFF ?
Y
A
N
IS A CHAR IN BUFF ?
N
A
Y
AT END OF RING BUFF ?
N
Y
CYCLE TO HEAD OF RING BUFFER
3$
EMT ERROR Ø; NO LINE/CHAR IN BUFFER
EMTDON

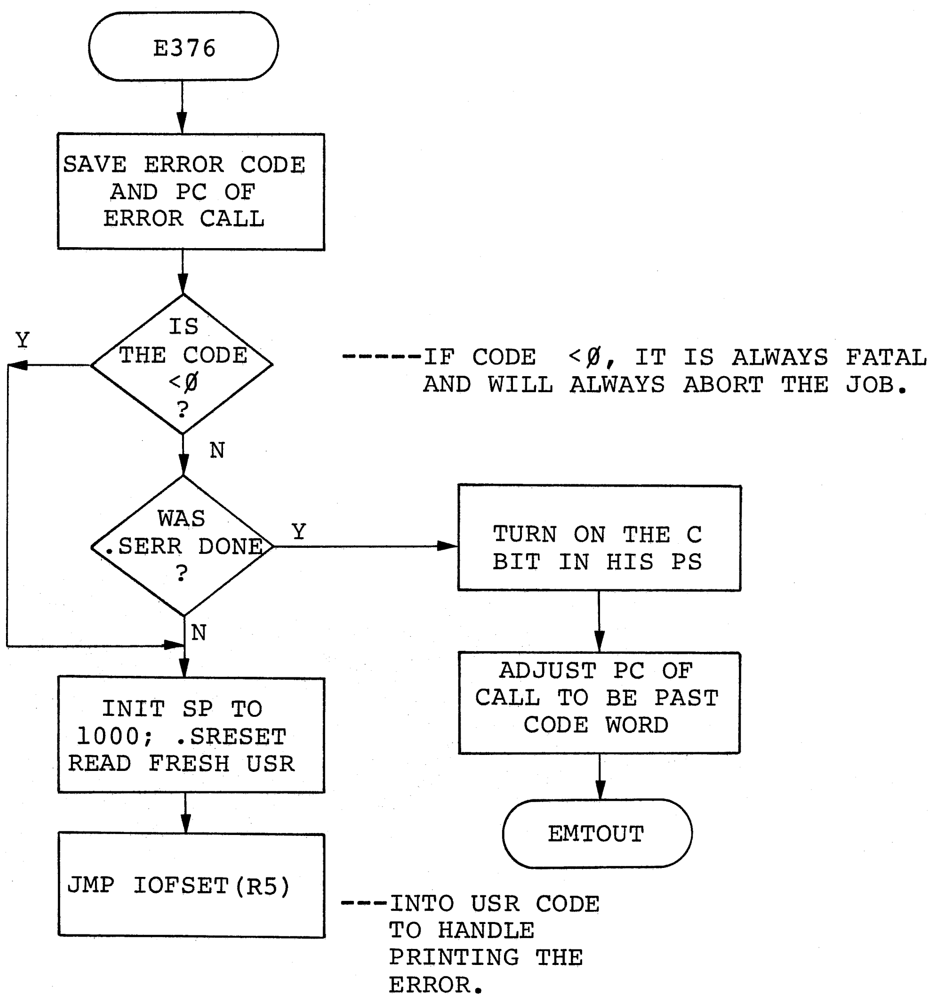
TTYIN (CONT.)/TTYOUT



EMT 376 is reserved for reporting fatal monitor errors. When a fatal error condition is encountered, a call of the form:

```
EMT 376
code
```

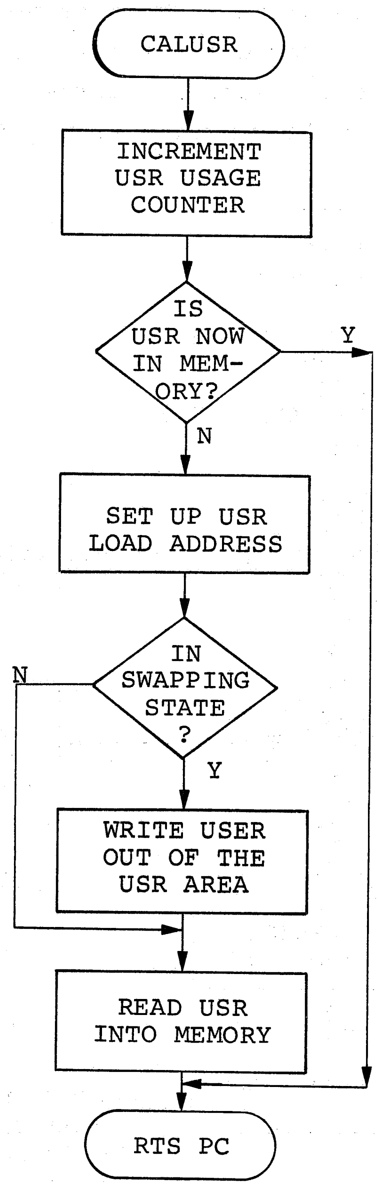
is executed. This indicates to RT-11 that a fatal error has occurred. The normal response is to print a ?M-error message and then abort the job. However, if a .SERR request has been done, no message will occur and control will pass to the user's program. The error bit (C bit) will be set and byte 52 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

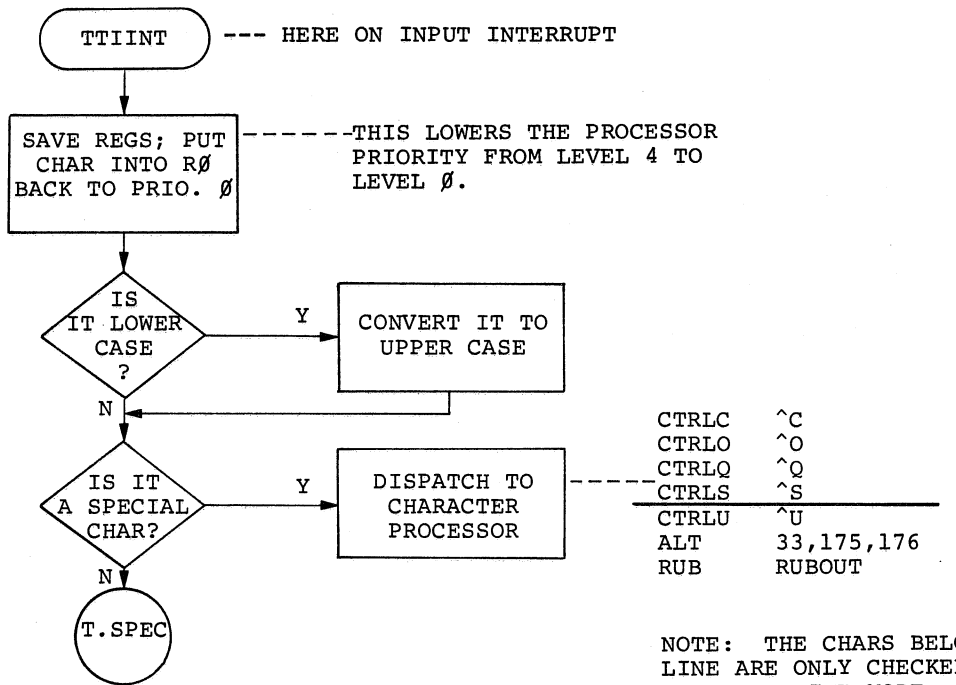
The interrupt service for the clock is primitive. The clock vector is set up such that the interrupt routine is always entered with the C bit = 1. At the interrupt routine, the code is:

```
ADC $TIME+2
ADC $TIME
RTI
```

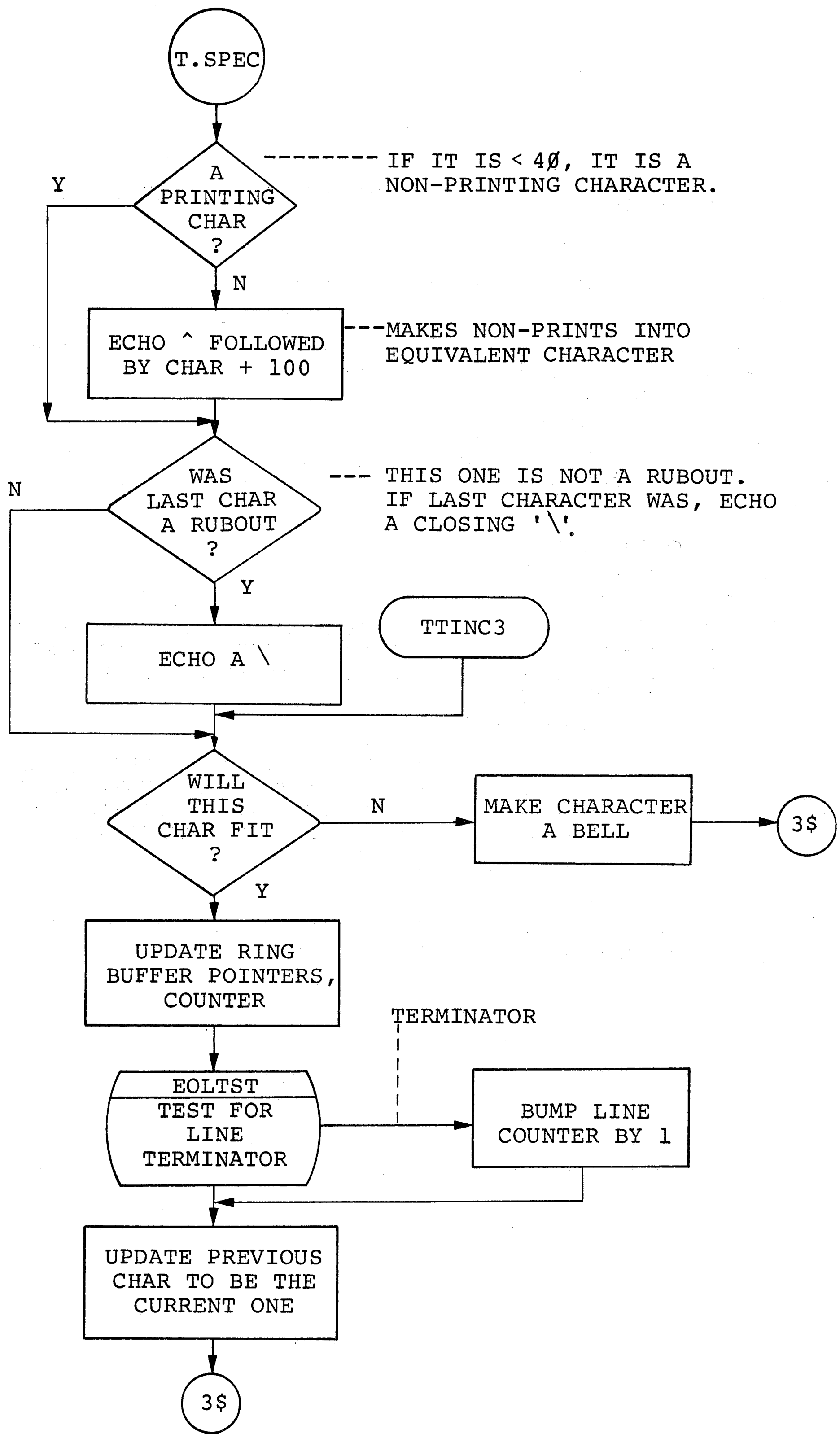
Since the C bit is 1, $TIME+2 is incremented by the ADC. When the low order word goes from 177777 to 0, the C bit remains on and $TIME is then incremented. No 24 hour wrap around is provided.

E.4.3 Console Terminal Interrupt Service

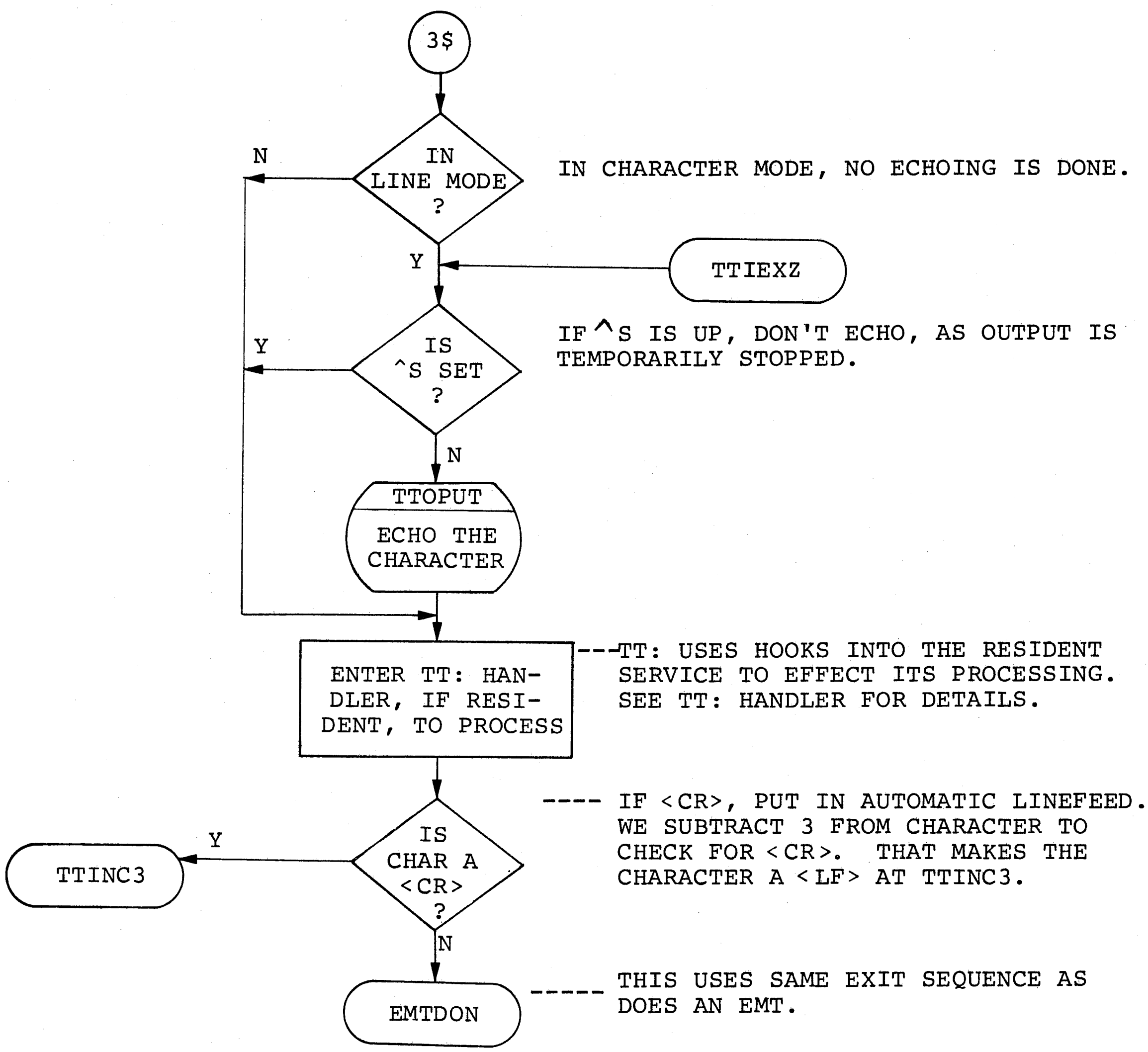
TT INPUT INTERRUPT SERVICE



NOTE: THE CHARS BELOW THE LINE ARE ONLY CHECKED WHEN TT IS IN LINE MODE. IN CHARACTER MODE THEY ARE NOT ACTED UPON.

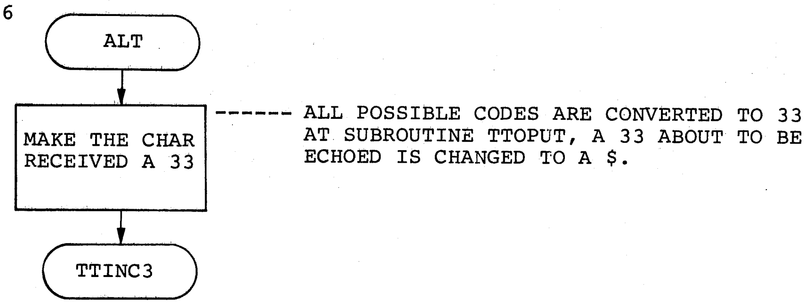

T.SPEC
A PRINTING CHAR ?
Y
N
IF IT IS < 4Ø, IT IS A NON-PRINTING CHARACTER.
ECHO ^ FOLLOWED BY CHAR + 100
MAKES NON-PRINTS INTO EQUIVALENT CHARACTER
WAS LAST CHAR A RUBOUT ?
N
Y
THIS ONE IS NOT A RUBOUT. IF LAST CHARACTER WAS, ECHO A CLOSING '\'.
ECHO A \
TTINC3
WILL THIS CHAR FIT ?
N
MAKE CHARACTER A BELL
3$
Y
UPDATE RING BUFFER POINTERS, COUNTER
TERMINATOR
EOLTST
TEST FOR LINE TERMINATOR
BUMP LINE COUNTER BY 1
UPDATE PREVIOUS CHAR TO BE THE CURRENT ONE
3$

TT INPUT INTERRUPT SERVICE (CONT.)

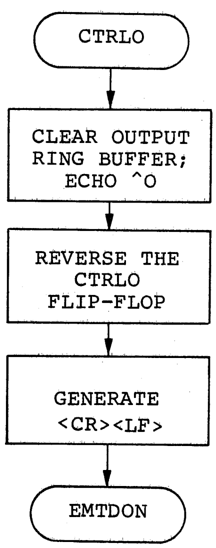


These routines are entered when any of the corresponding special characters are struck.

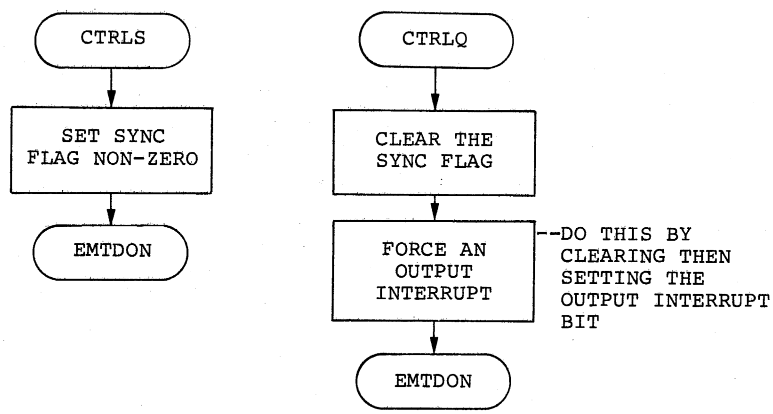
ALTMODE - 33,175,176



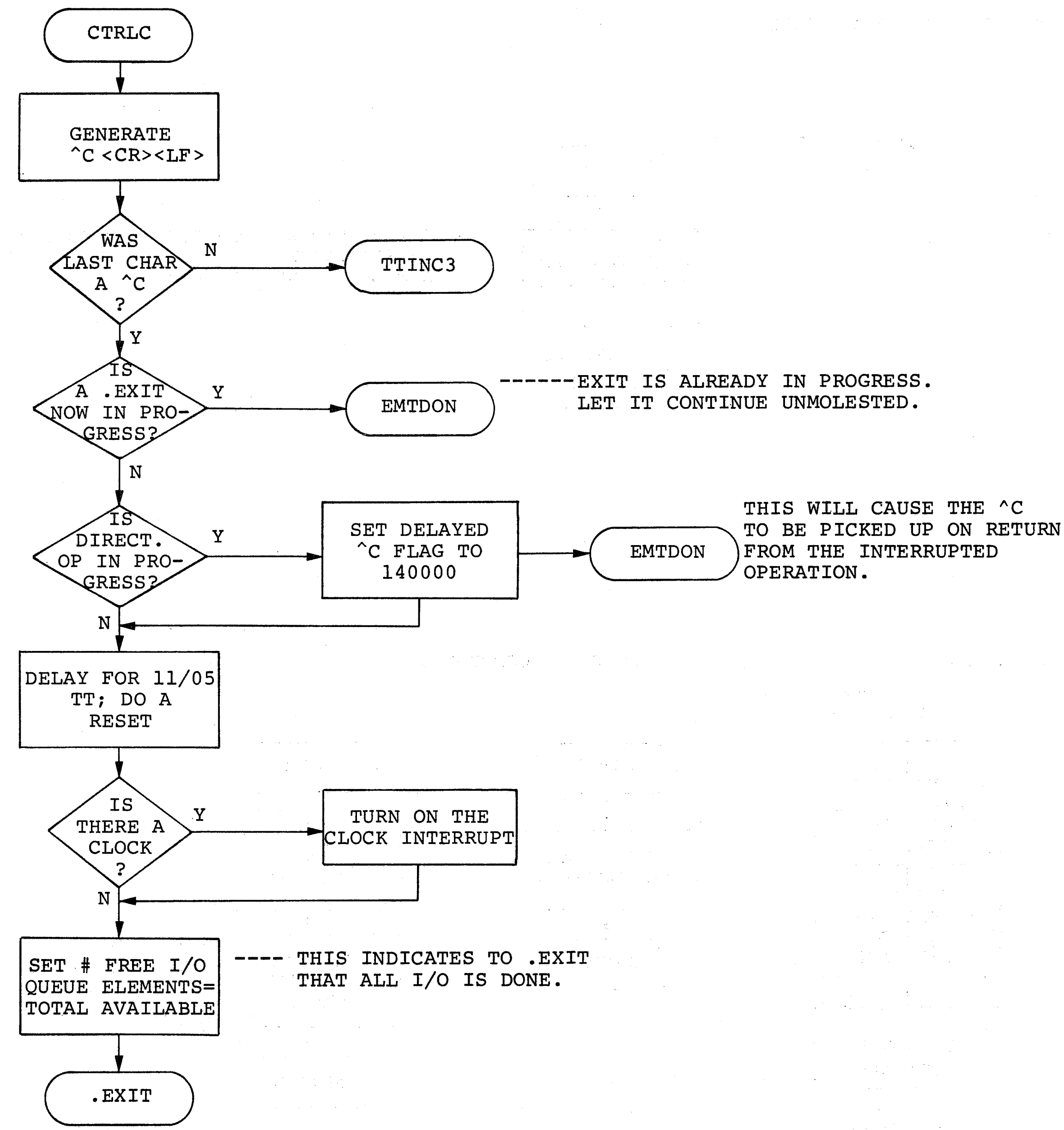
CONTROL O

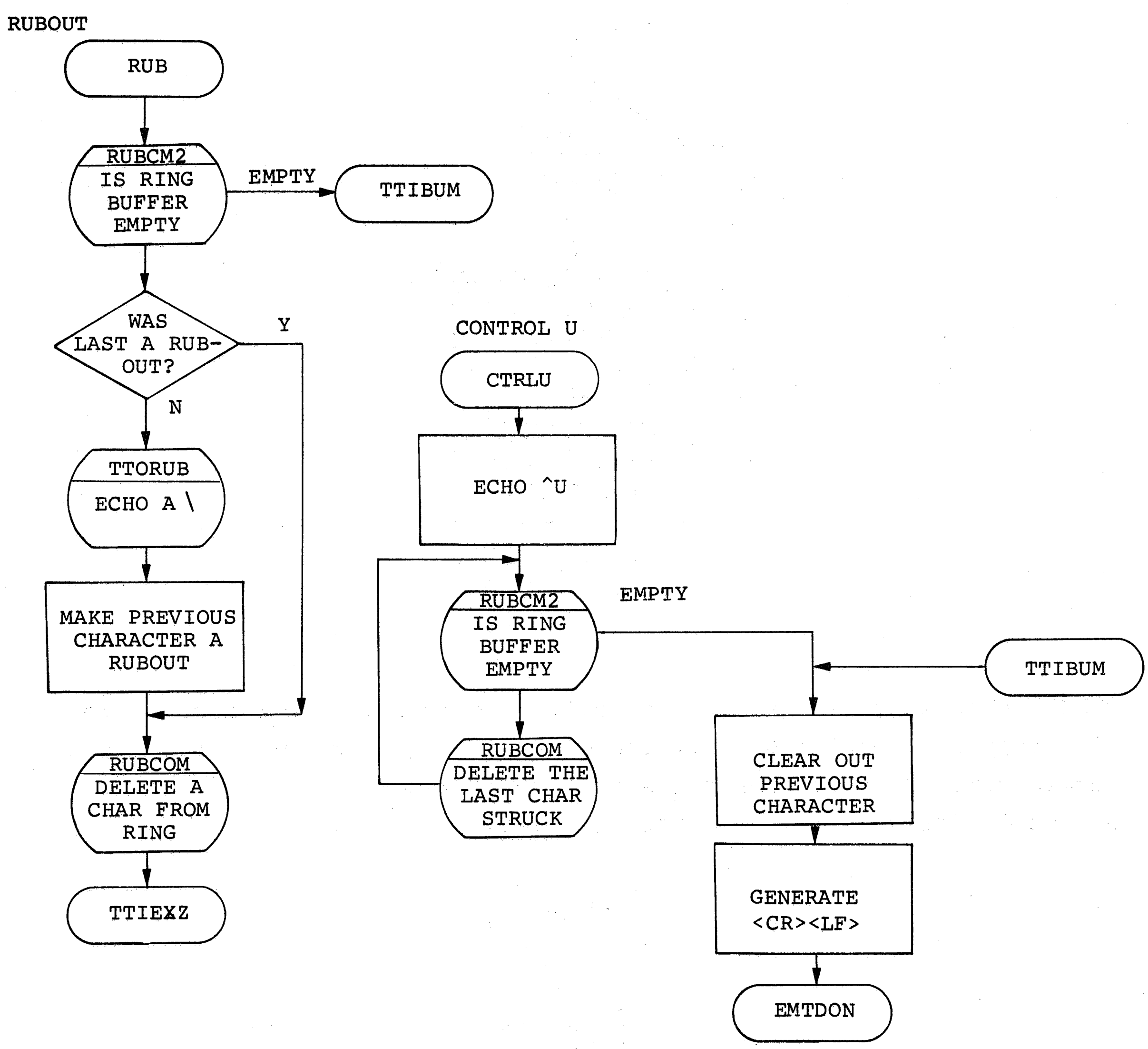


CONTROL S, CONTROL Q



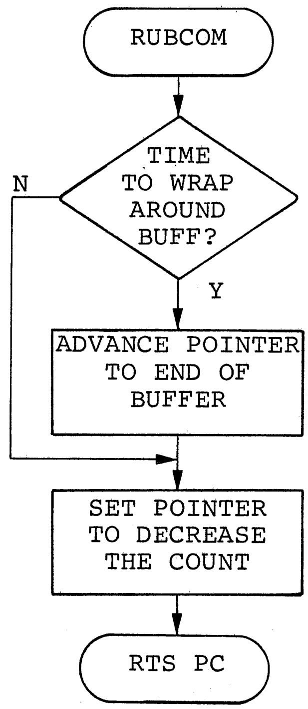
CONTROL C


CTRLC
GENERATE ^C <CR><LF>
WAS LAST CHAR A ^C ?
N
TTINC3
Y
IS A .EXIT NOW IN PRO- GRESS?
Y
EMTDON
------EXIT IS ALREADY IN PROGRESS. LET IT CONTINUE UNMOLESTED.
N
IS DIRECT. OP IN PRO- GRESS?
Y
SET DELAYED ^C FLAG TO 140000
EMTDON
THIS WILL CAUSE THE ^C TO BE PICKED UP ON RETURN FROM THE INTERRUPTED OPERATION.
N
DELAY FOR 11/05 TT; DO A RESET
IS THERE A CLOCK ?
Y
TURN ON THE CLOCK INTERRUPT
N
SET # FREE I/O QUEUE ELEMENTS= TOTAL AVAILABLE
---- THIS INDICATES TO .EXIT THAT ALL I/O IS DONE.
.EXIT

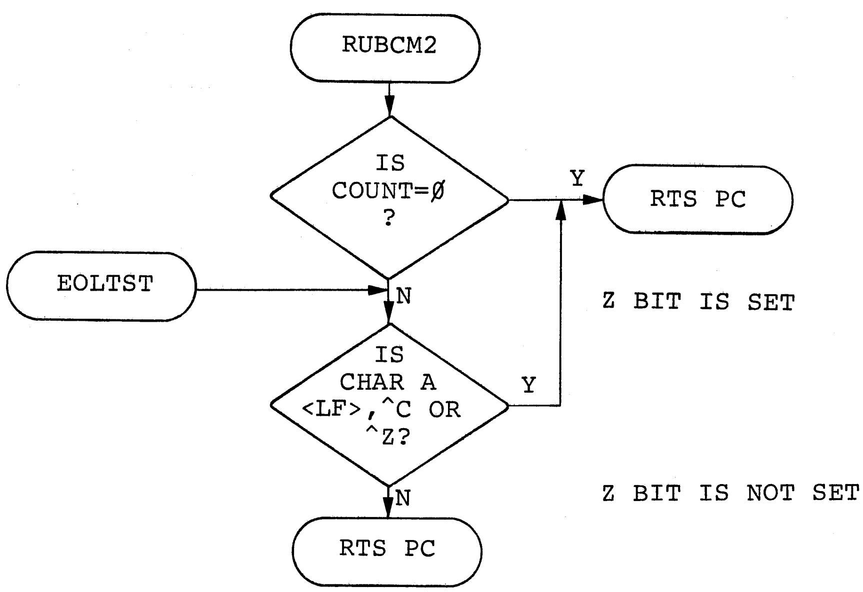

RUBOUT
RUB
RUBCM2
IS RING BUFFER EMPTY
EMPTY
TTIBUM
WAS LAST A RUB-OUT?
Y
N
TTORUB
ECHO A \
MAKE PREVIOUS CHARACTER A RUBOUT
RUBCOM
DELETE A CHAR FROM RING
TTIEXZ
CONTROL U
CTRLU
ECHO ^U
RUBCM2
IS RING BUFFER EMPTY
EMPTY
TTIBUM
RUBCOM
DELETE THE LAST CHAR STRUCK
CLEAR OUT PREVIOUS CHARACTER
GENERATE <CR><LF>
EMTDON

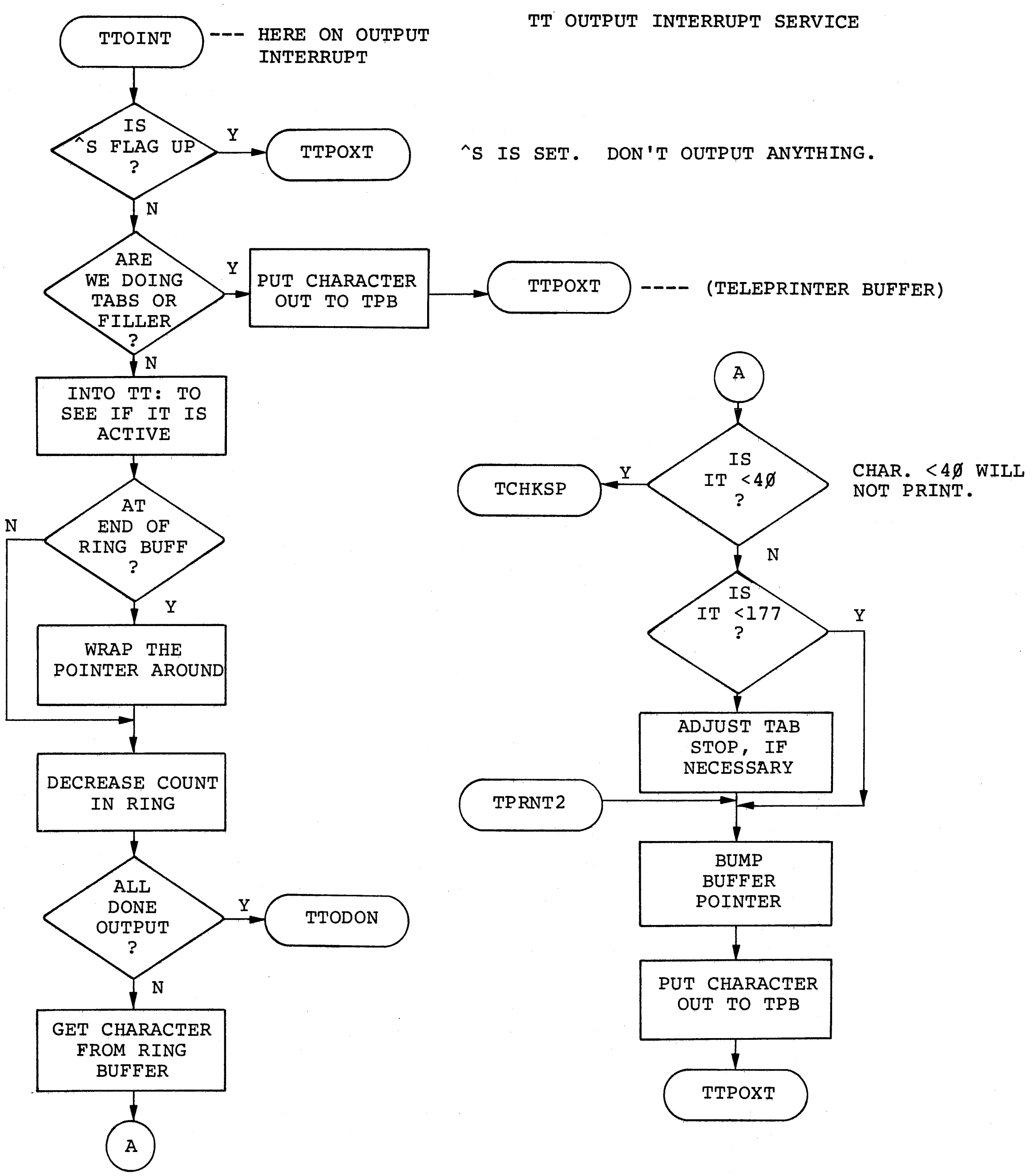
RUBCON/RUBCM2

RUBCOM will update the input ring buffer pointers when a character is to be deleted.

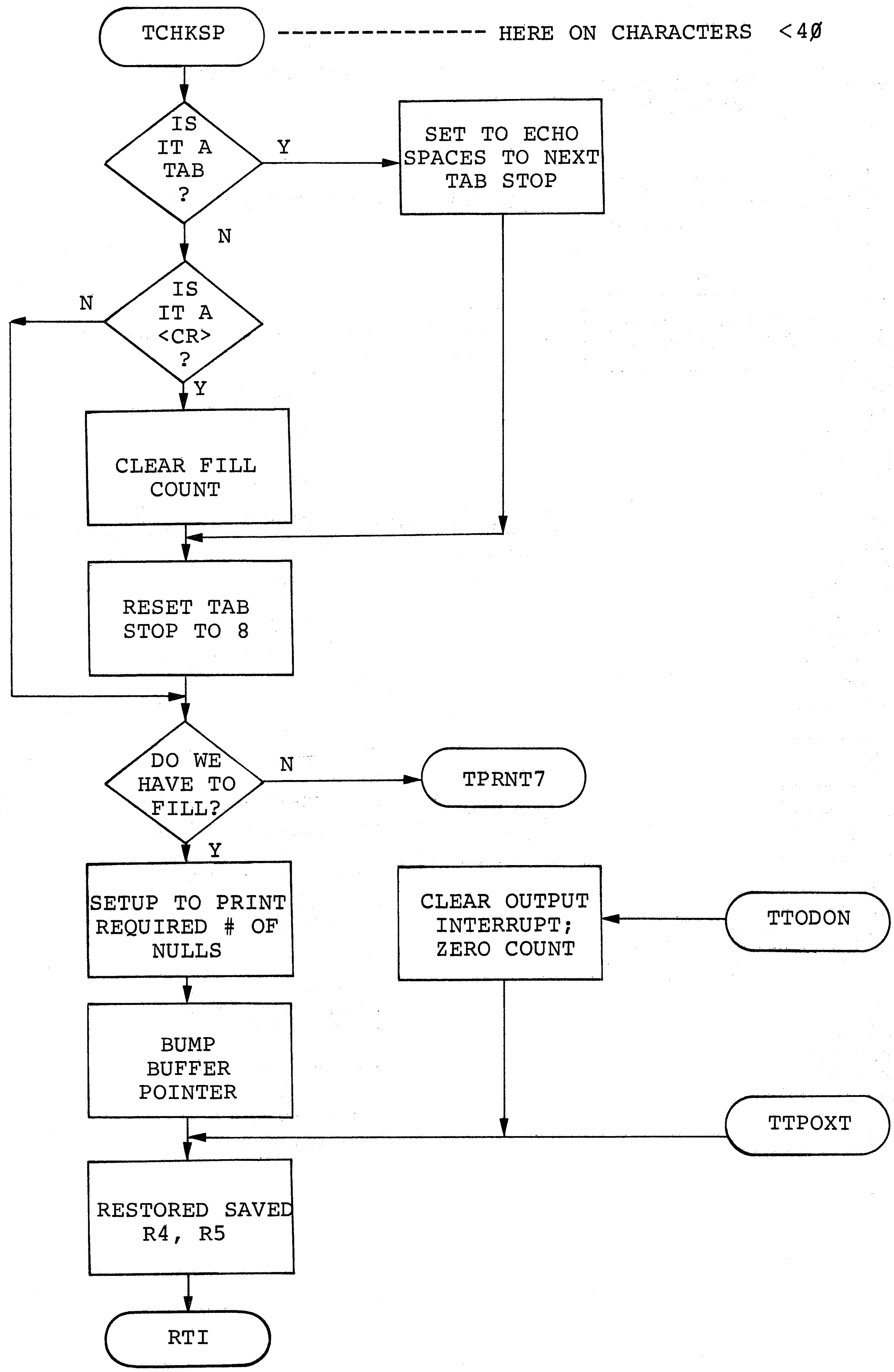


RUBCM2 checks to see if the ring buffer is empty. The buffer is empty if either the count = Ø or if the character to be deleted is a line terminator. This routine falls into routine EOLTST. The zero condition is returned if the buffer is empty.

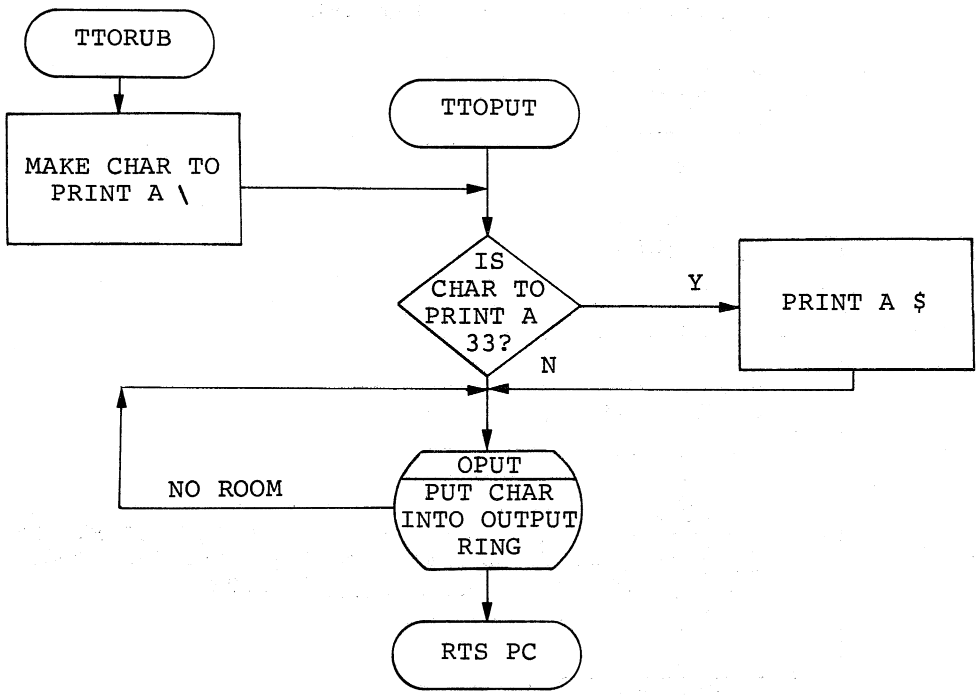



TT OUTPUT INTERRUPT SERVICE
TTOINT
--- HERE ON OUTPUT INTERRUPT
IS ^S FLAG UP ?
Y
TTPOXT
^S IS SET. DON'T OUTPUT ANYTHING.
N
ARE WE DOING TABS OR FILLER ?
Y
PUT CHARACTER OUT TO TPB
TTPOXT
---- (TELEPRINTER BUFFER)
N
INTO TT: TO SEE IF IT IS ACTIVE
AT END OF RING BUFF ?
N
Y
WRAP THE POINTER AROUND
DECREASE COUNT IN RING
ALL DONE OUTPUT ?
Y
TTODON
N
GET CHARACTER FROM RING BUFFER
A
A
IS IT <4Ø ?
Y
TCHKSP
CHAR. <4Ø WILL NOT PRINT.
N
IS IT <177 ?
Y
ADJUST TAB STOP, IF NECESSARY
TPRNT2
BUMP BUFFER POINTER
PUT CHARACTER OUT TO TPB
TTPOXT

TT OUTPUT INTERRUPT SERVICE (CONT.)

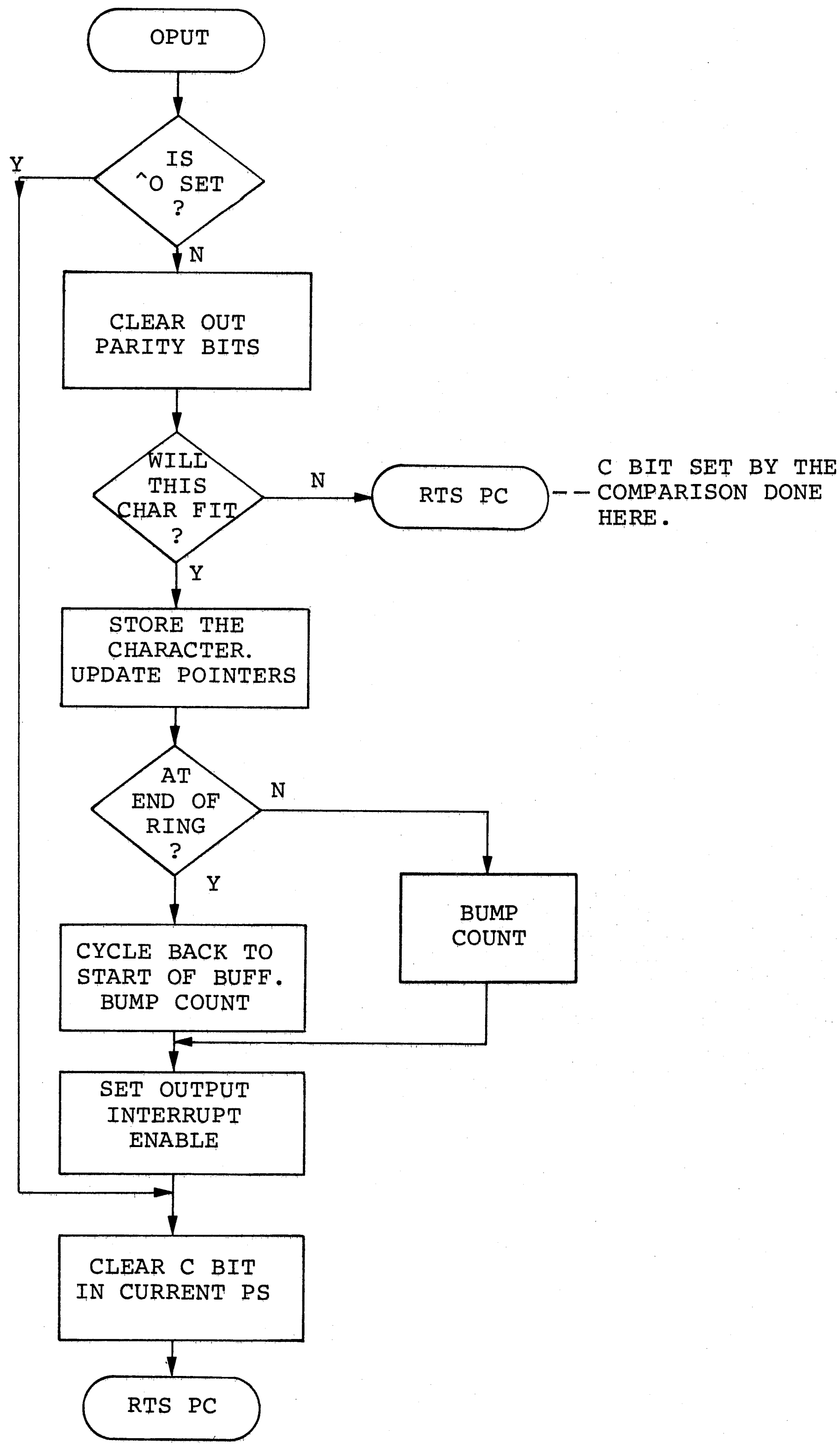


TTORUB and TTOPUT handle the printing of ALTMODE and RUBOUT. They print a $ for ALTMODE and \ for RUBOUT.



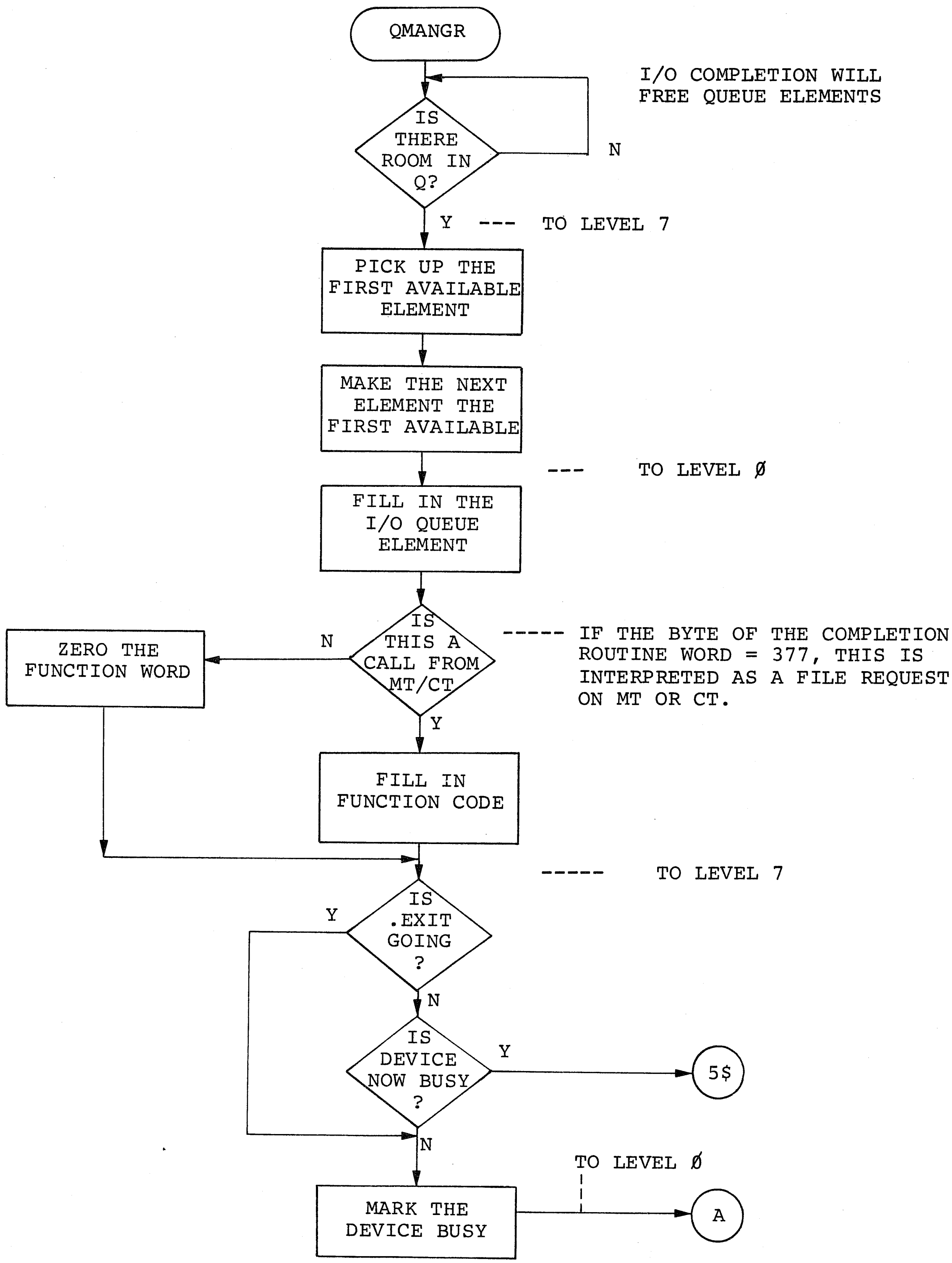
OPUT

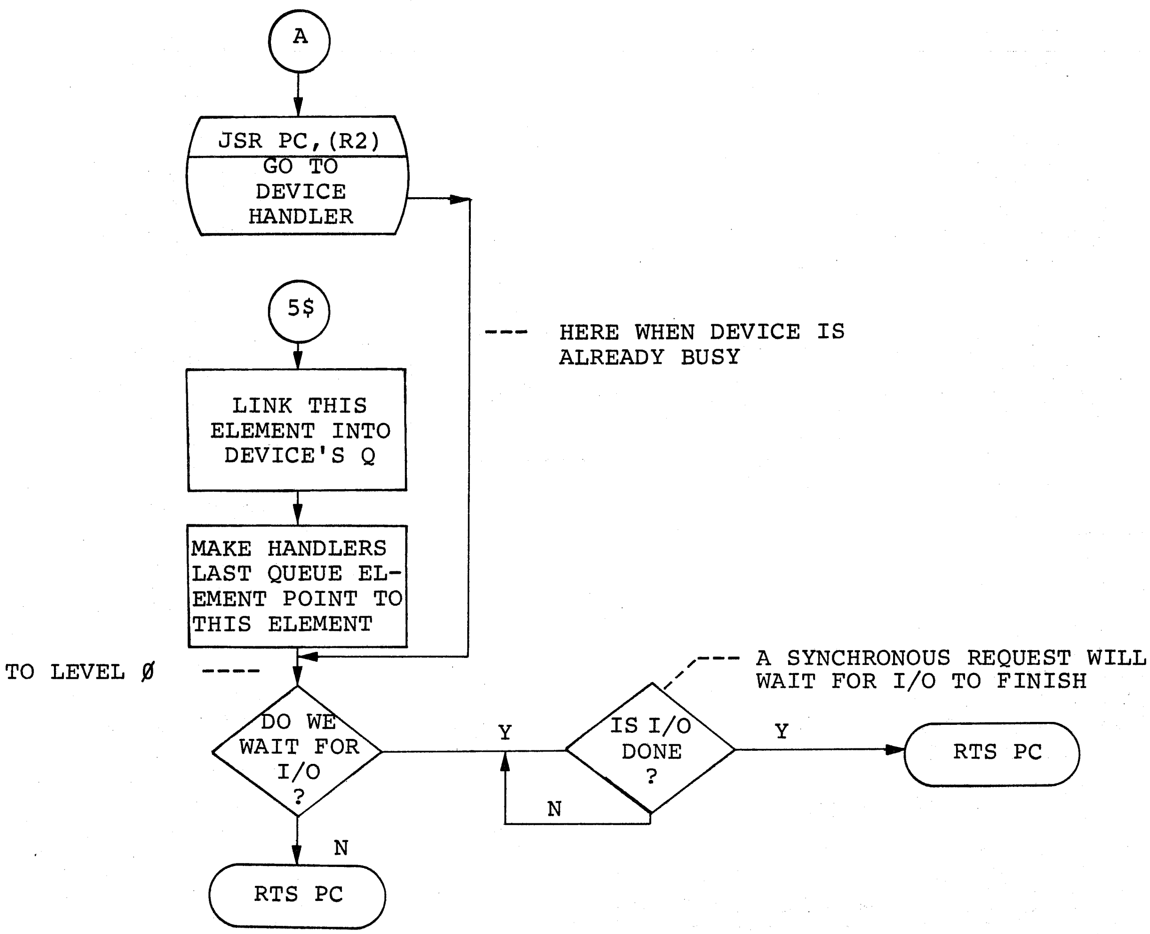
OPUT actually puts the output character into the ring buffer. It updates the ring pointers and sets the interrupt enable bit. If the buffer is full, it returns with the C bit set.



E.4.4 I/O Routines

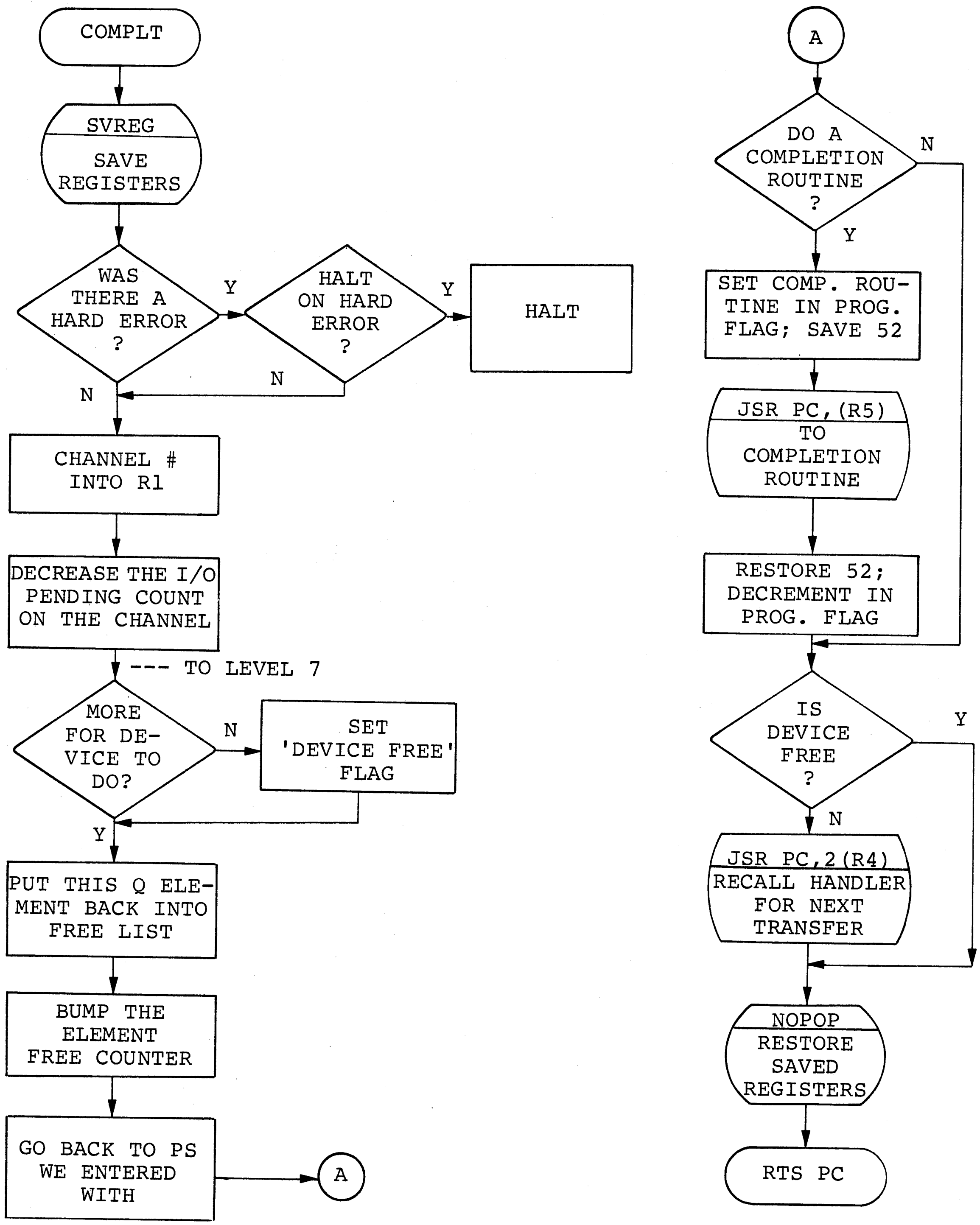
I/O QUEUE MANAGEMENT ROUTINES




A
JSR PC,(R2)
GO TO DEVICE HANDLER
5$
--- HERE WHEN DEVICE IS ALREADY BUSY
LINK THIS ELEMENT INTO DEVICE'S Q
MAKE HANDLERS LAST QUEUE EL-EMENT POINT TO THIS ELEMENT
TO LEVEL Ø ----
DO WE WAIT FOR I/O ?
Y
IS I/O DONE ?
N
--- A SYNCHRONOUS REQUEST WILL WAIT FOR I/O TO FINISH
Y
RTS PC
N
RTS PC

I/O QUEUE COMPLETION

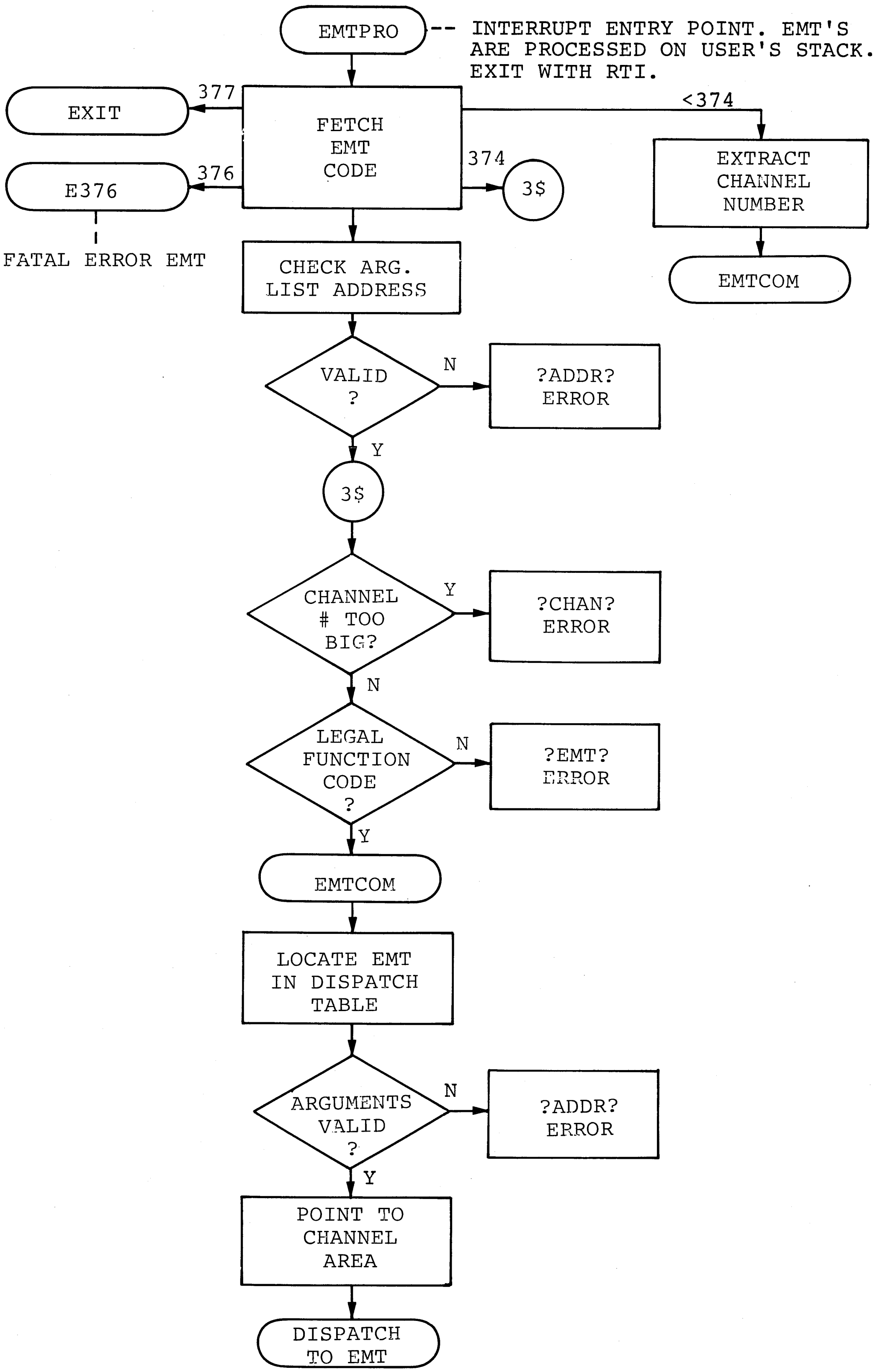
COMPLT is entered when an I/O transfer finishes.



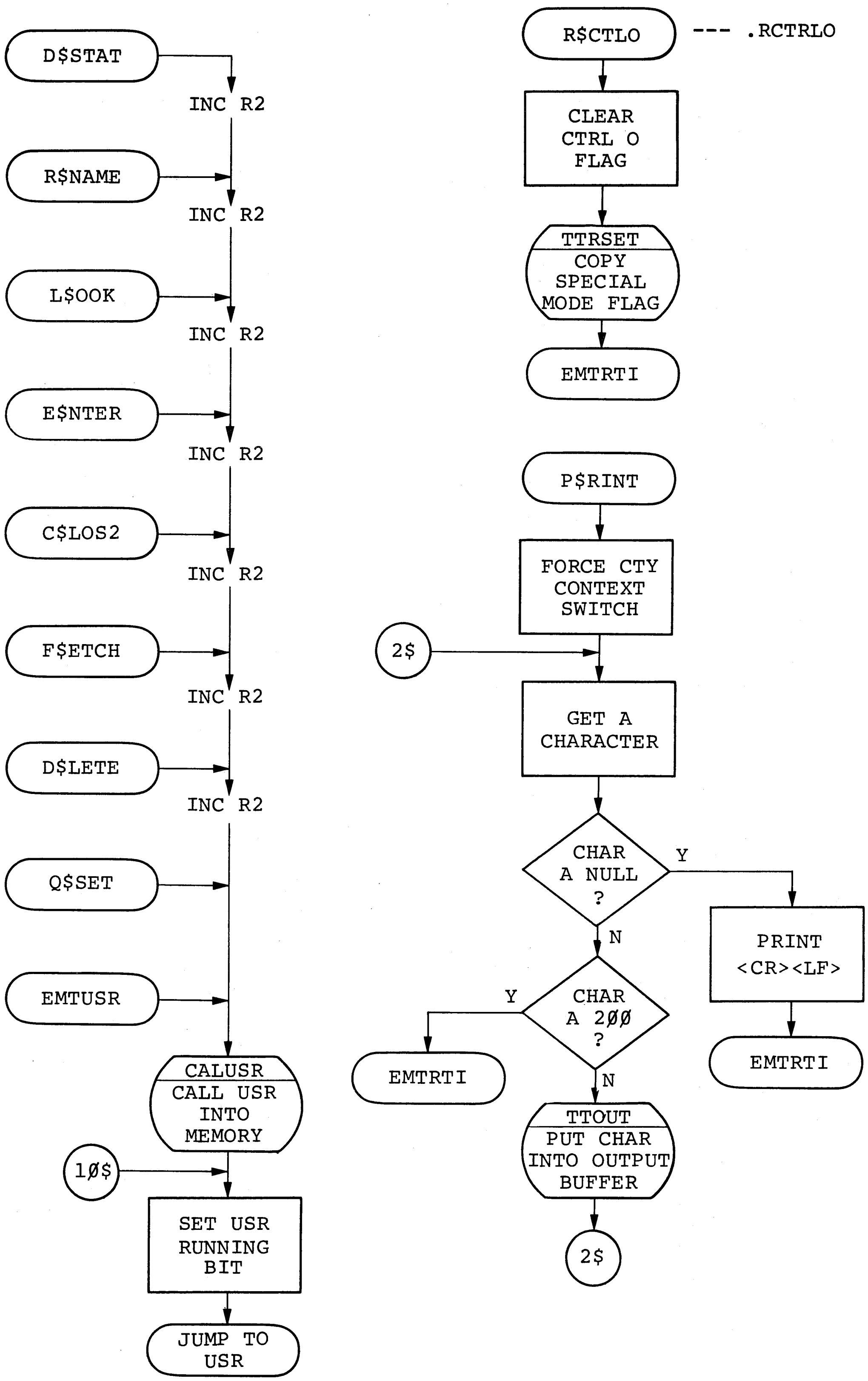
E.5 RMON (RESIDENT MONITOR) FLOWCHARTS FOR FOREGROUND/BACKGROUND MONITOR

E.5.1 EMT Processors

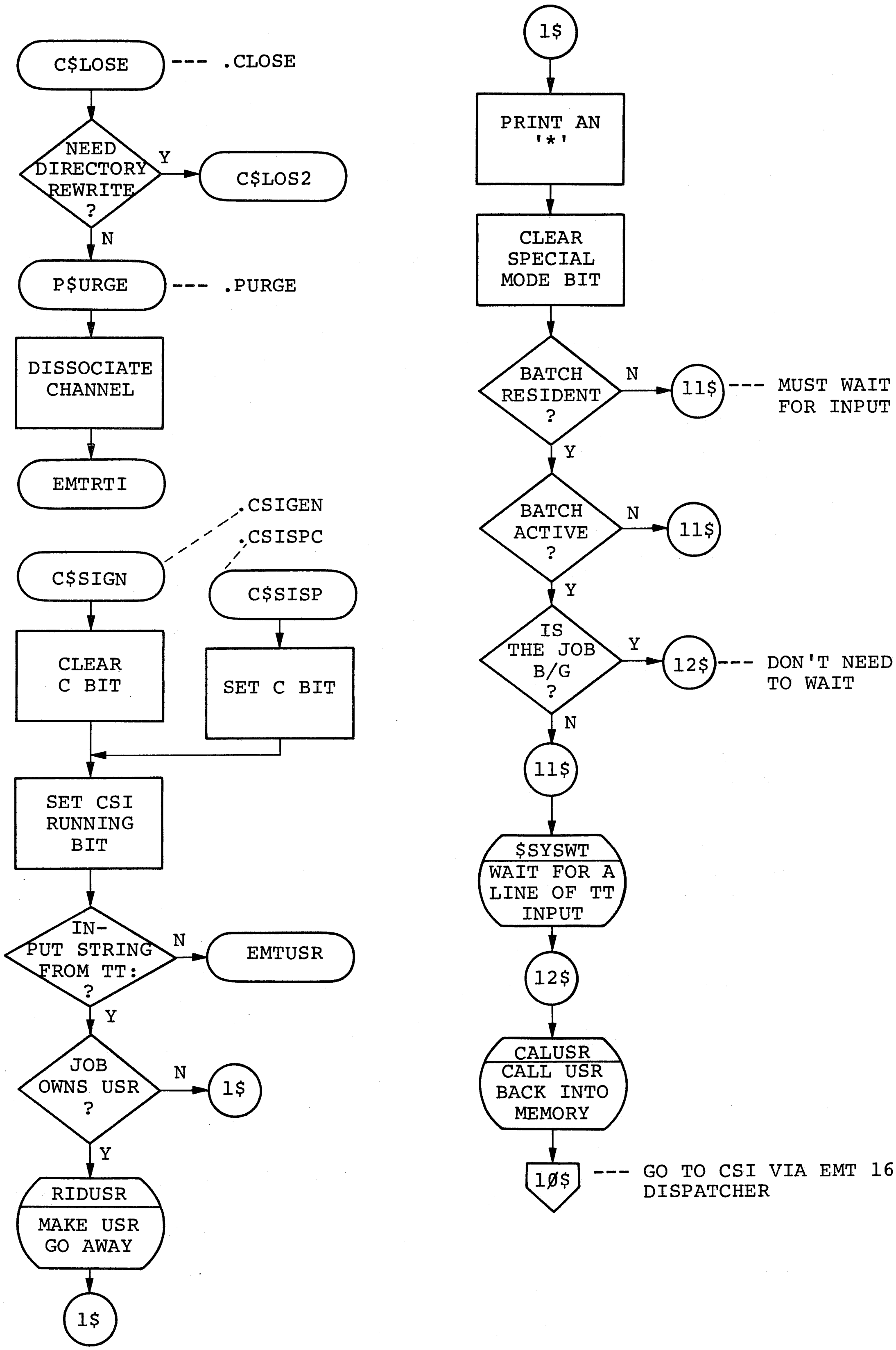
EMT DISPATCHER

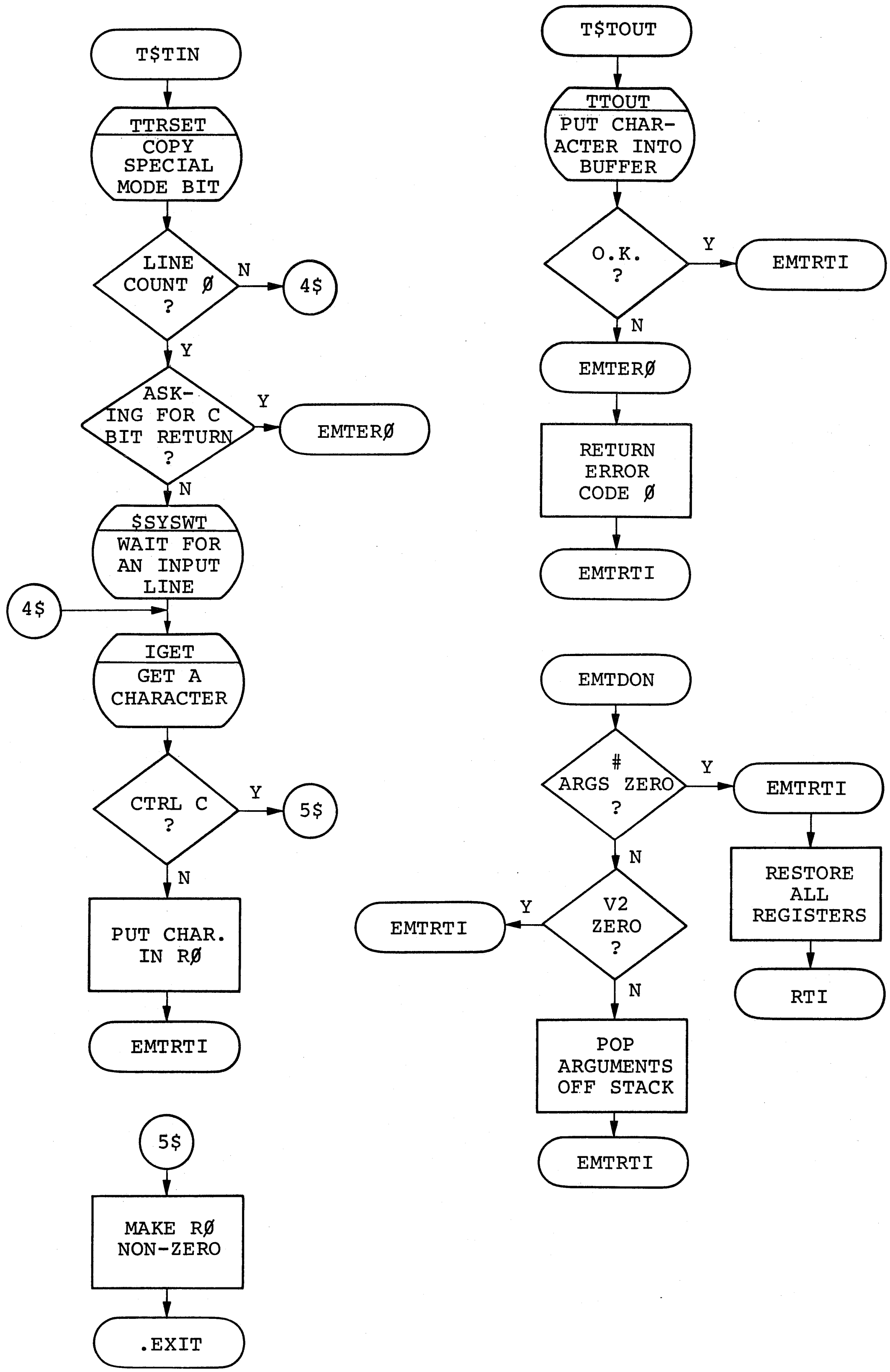


EMT 16 DISPATCH, .RCTRLO, .PRINT

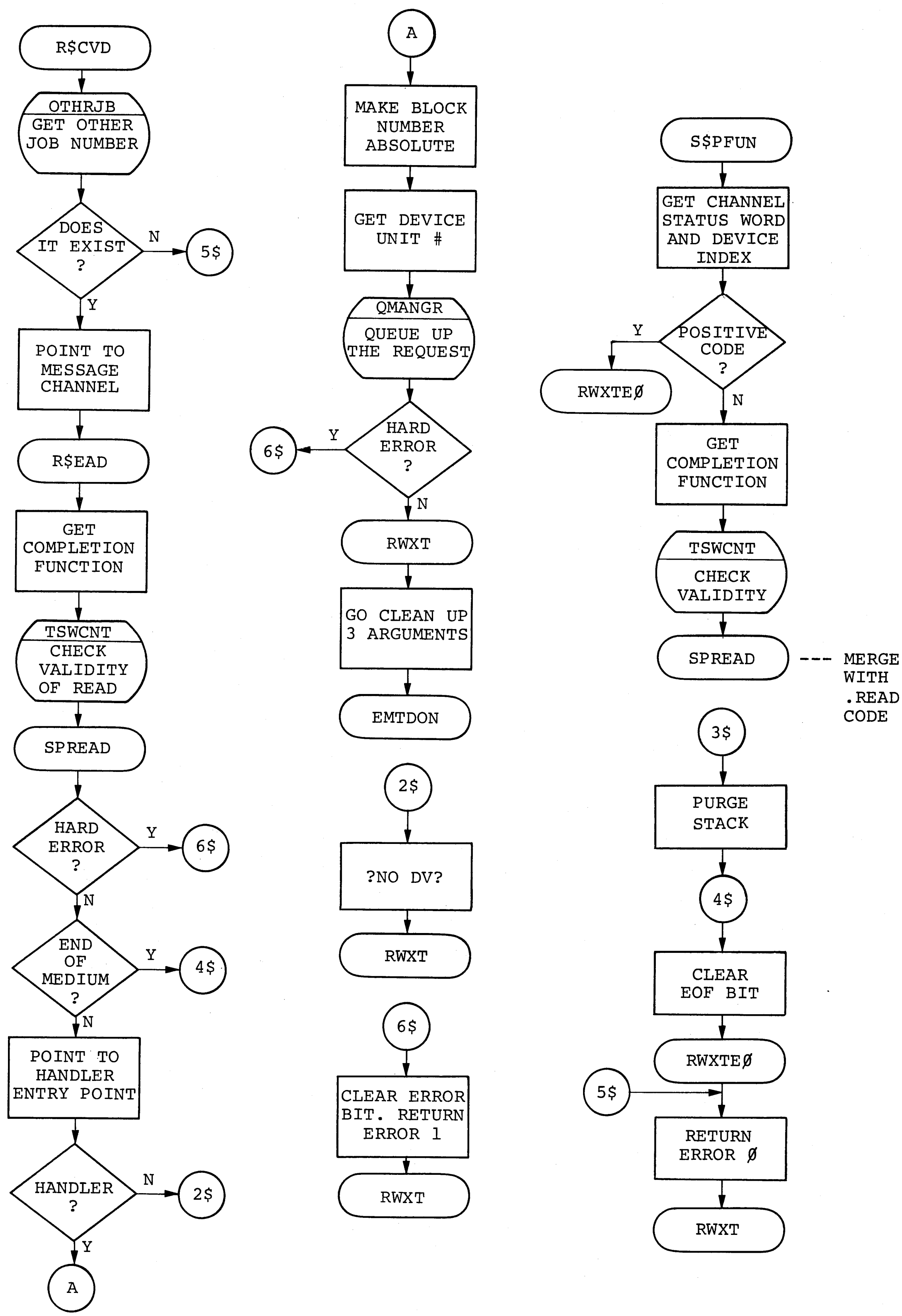


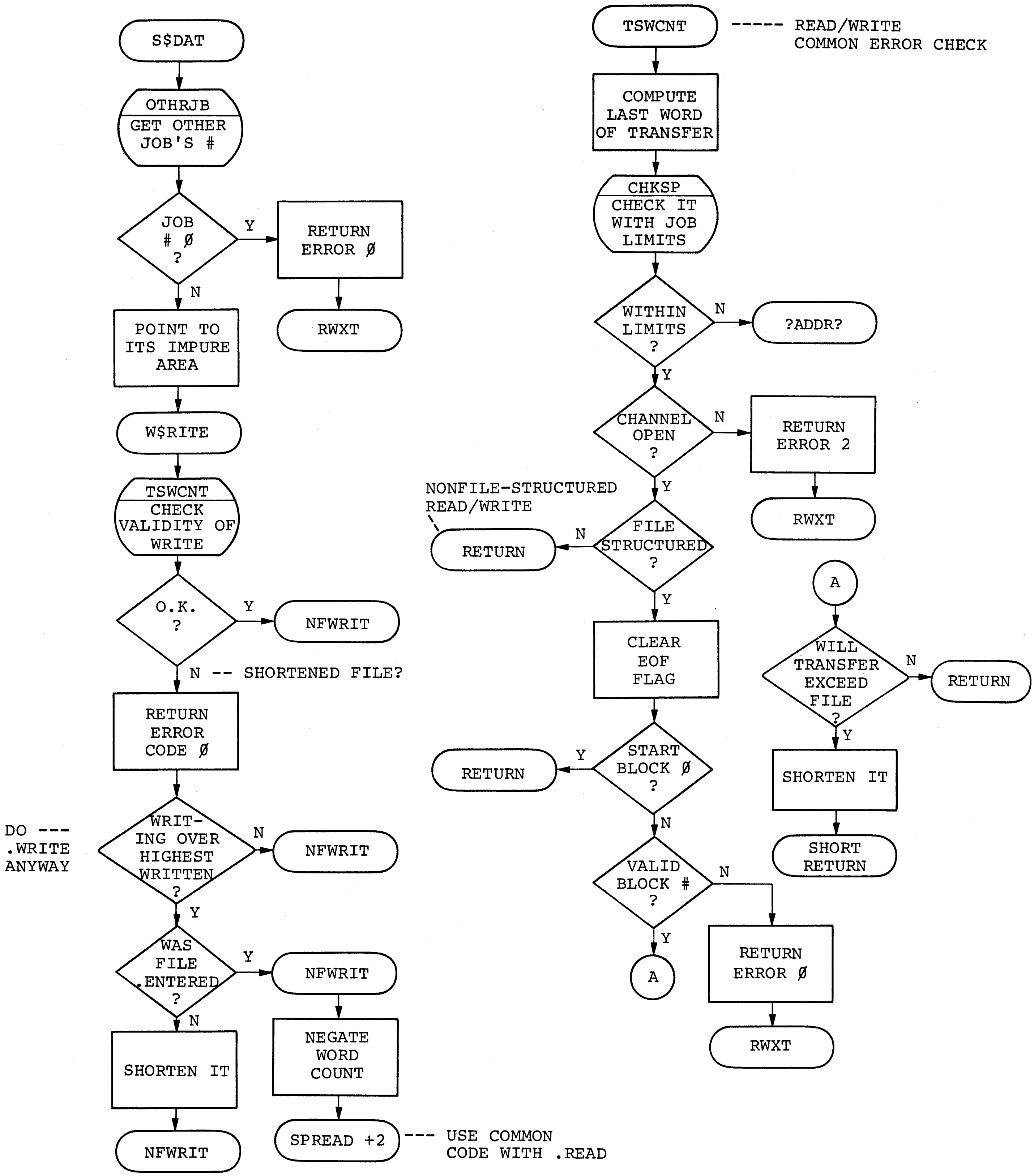
.CLOSE, .PURGE, .CSISPC, .CSIGEN



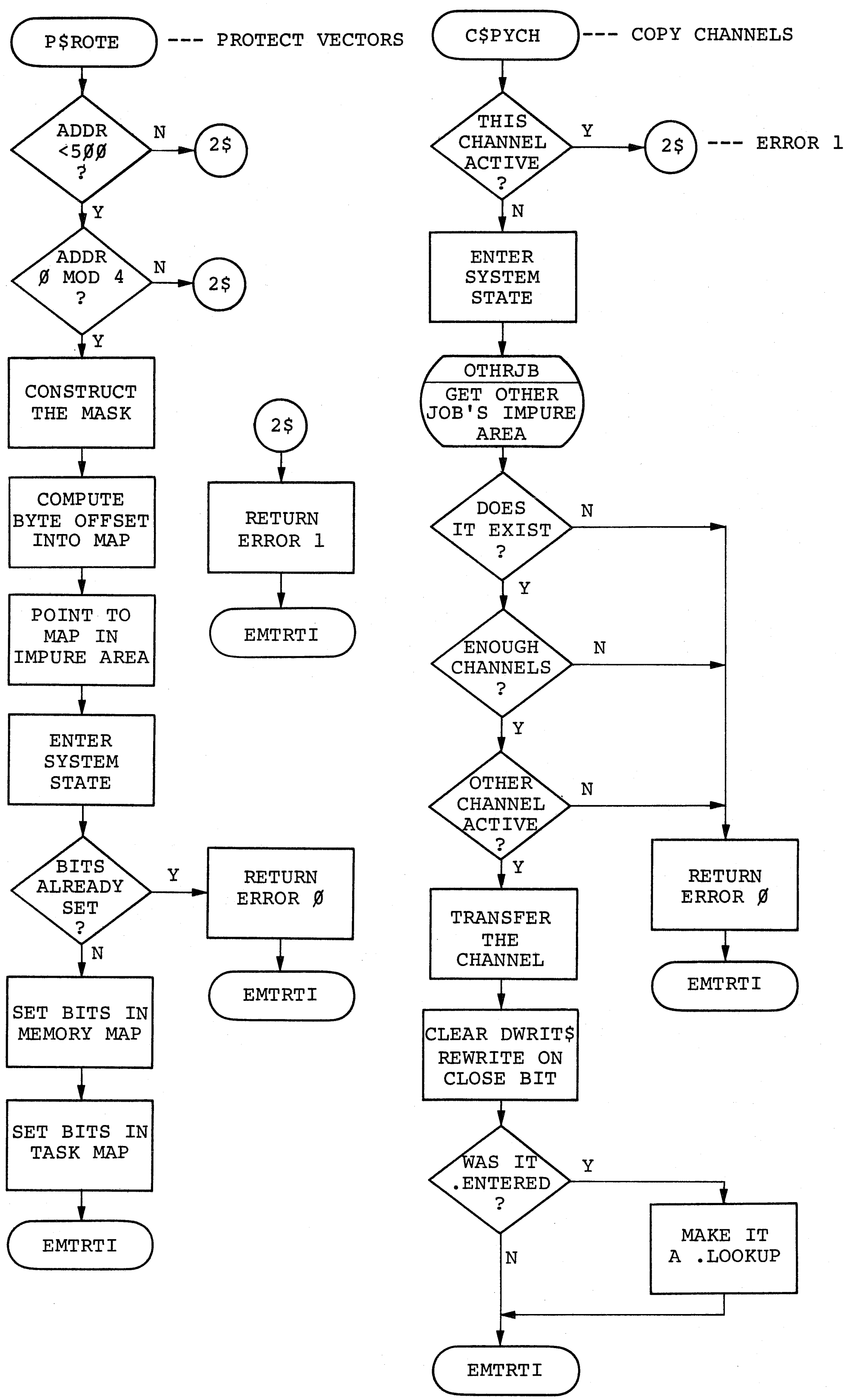

T$TIN
TTRSET
COPY SPECIAL MODE BIT
LINE COUNT Ø ?
N
4$
Y
ASK-ING FOR C BIT RETURN ?
Y
EMTERØ
N
$SYSWT
WAIT FOR AN INPUT LINE
4$
IGET
GET A CHARACTER
CTRL C ?
Y
5$
N
PUT CHAR. IN RØ
EMTRTI
5$
MAKE RØ NON-ZERO
.EXIT
T$TOUT
TTOUT
PUT CHAR-ACTER INTO BUFFER
O.K. ?
Y
EMTRTI
N
EMTERØ
RETURN ERROR CODE Ø
EMTRTI
EMTDON
ARGS ZERO ?
Y
EMTRTI
RESTORE ALL REGISTERS
RTI
N
V2 ZERO ?
Y
EMTRTI
N
POP ARGUMENTS OFF STACK
EMTRTI

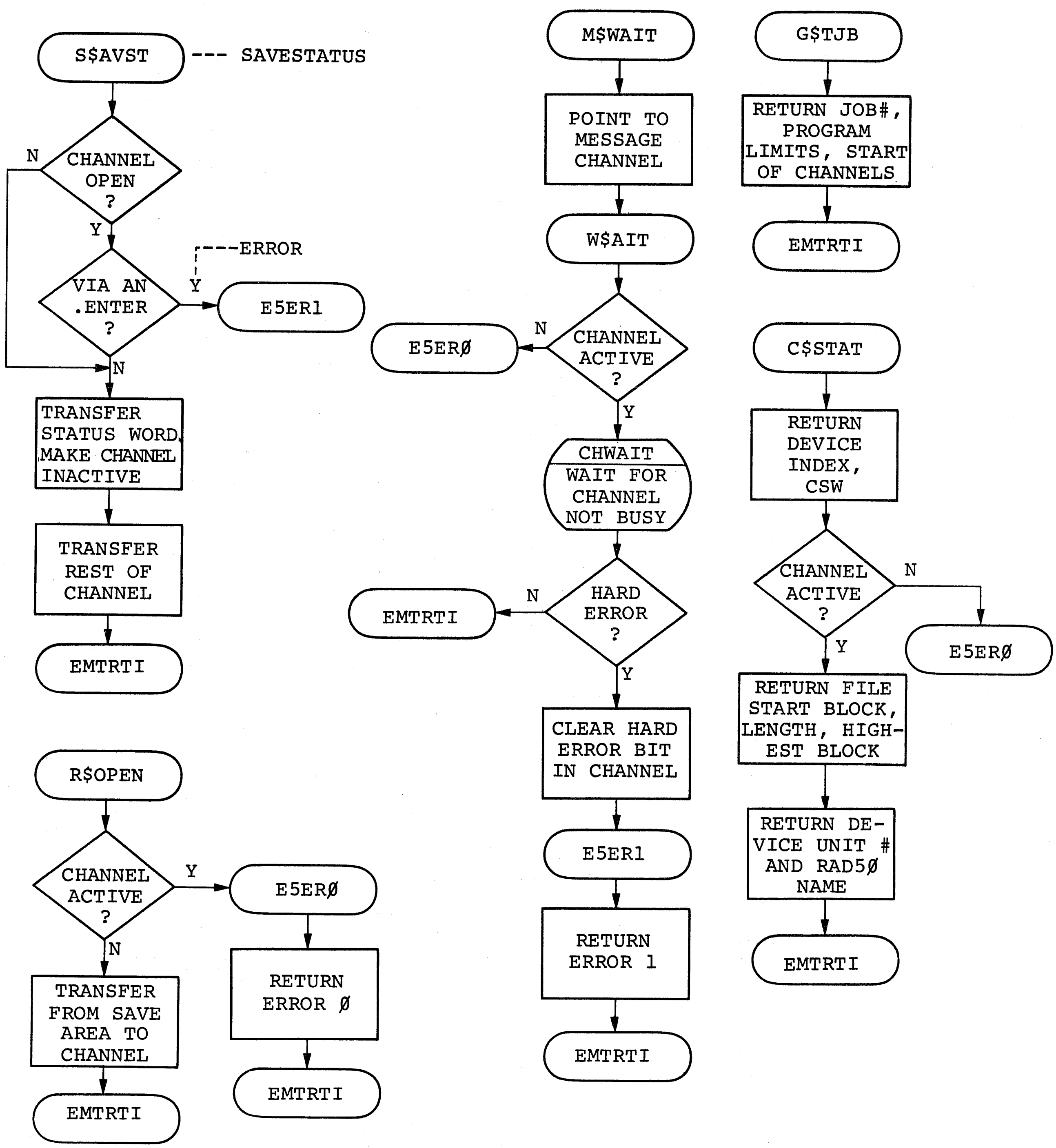
.READ, .RCVD, .SPFUN



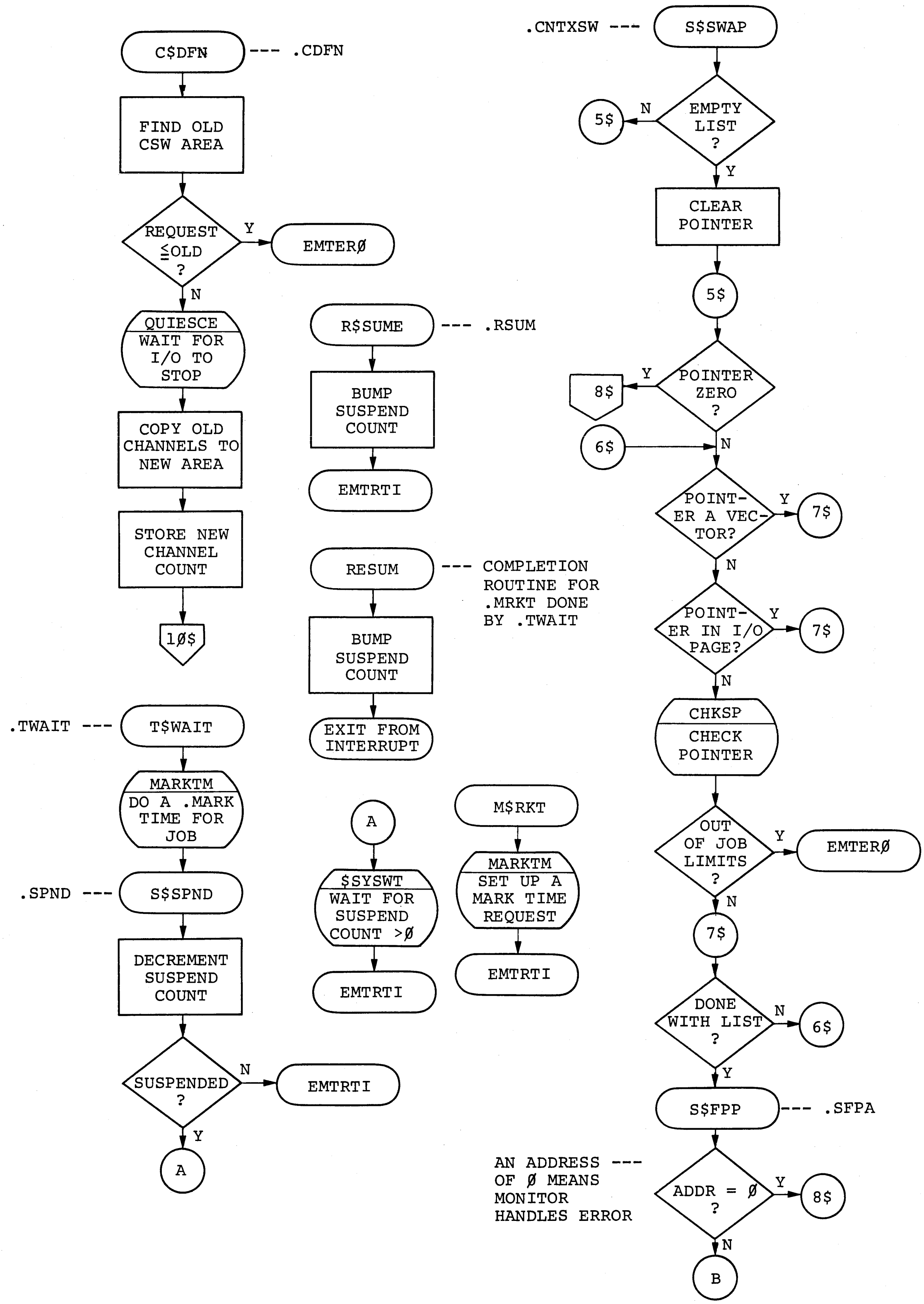

S$DAT
OTHRJB
GET OTHER JOB'S #
JOB # Ø ?
Y
RETURN ERROR Ø
RWXT
N
POINT TO ITS IMPURE AREA
W$RITE
TSWCNT
CHECK VALIDITY OF WRITE
O.K. ?
Y
NFWRIT
N -- SHORTENED FILE?
RETURN ERROR CODE Ø
DO --- .WRITE ANYWAY
WRIT-ING OVER HIGHEST WRITTEN ?
N
NFWRIT
Y
WAS FILE ENTERED ?
Y
NFWRIT
NEGATE WORD COUNT
SPREAD +2
--- USE COMMON CODE WITH .READ
N
SHORTEN IT
NFWRIT
TSWCNT
----- READ/WRITE COMMON ERROR CHECK
COMPUTE LAST WORD OF TRANSFER
CHKSP
CHECK IT WITH JOB LIMITS
WITHIN LIMITS ?
N
?ADDR?
Y
CHANNEL OPEN ?
N
RETURN ERROR 2
RWXT
Y
NONFILE-STRUCTURED READ/WRITE
RETURN
N
FILE STRUCTURED ?
Y
CLEAR EOF FLAG
START BLOCK Ø ?
Y
RETURN
N
VALID BLOCK # ?
N
RETURN ERROR Ø
RWXT
Y
A
A
WILL TRANSFER EXCEED FILE ?
N
RETURN
Y
SHORTEN IT
SHORT RETURN

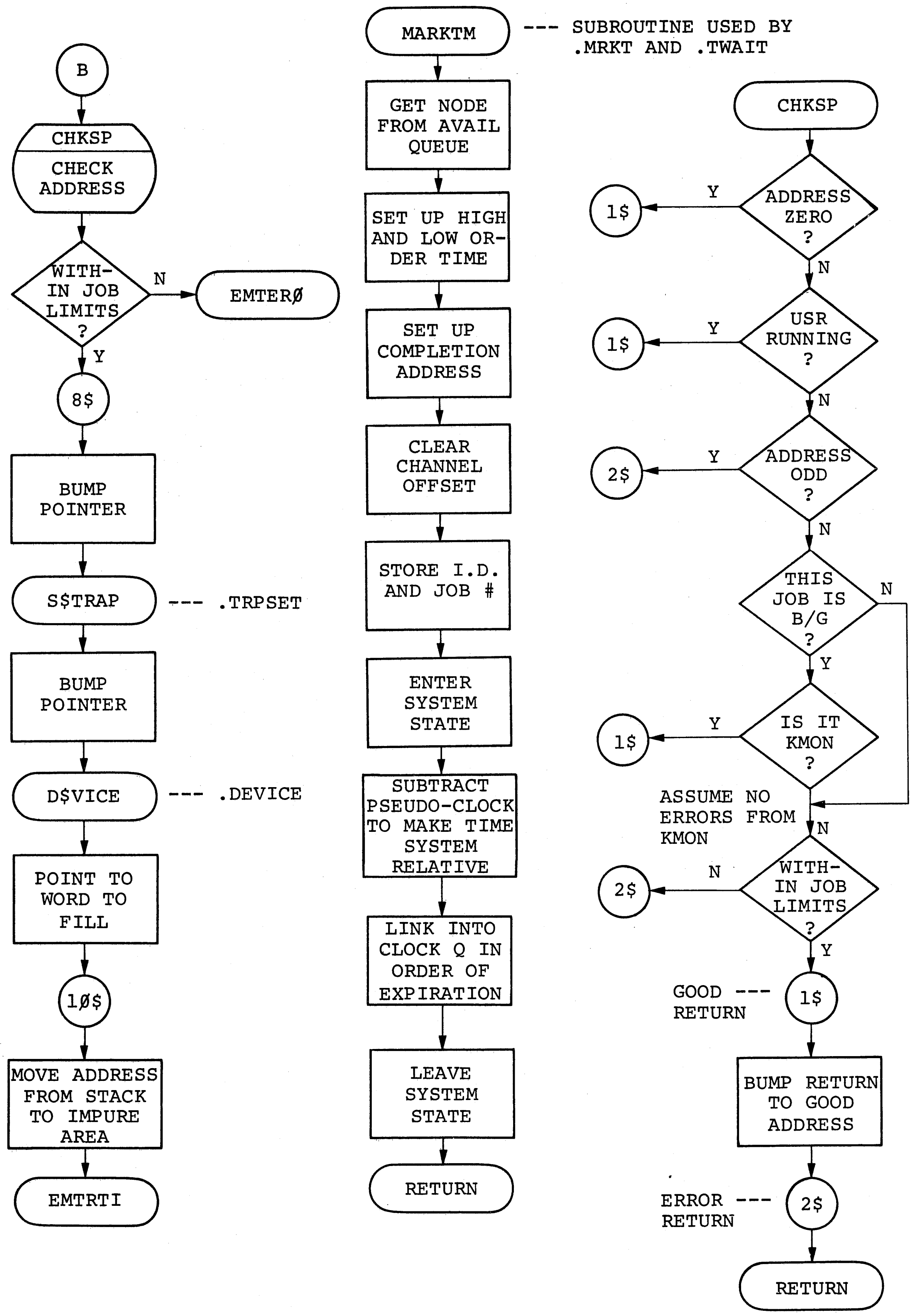
.PROTECT, .CHCOPY

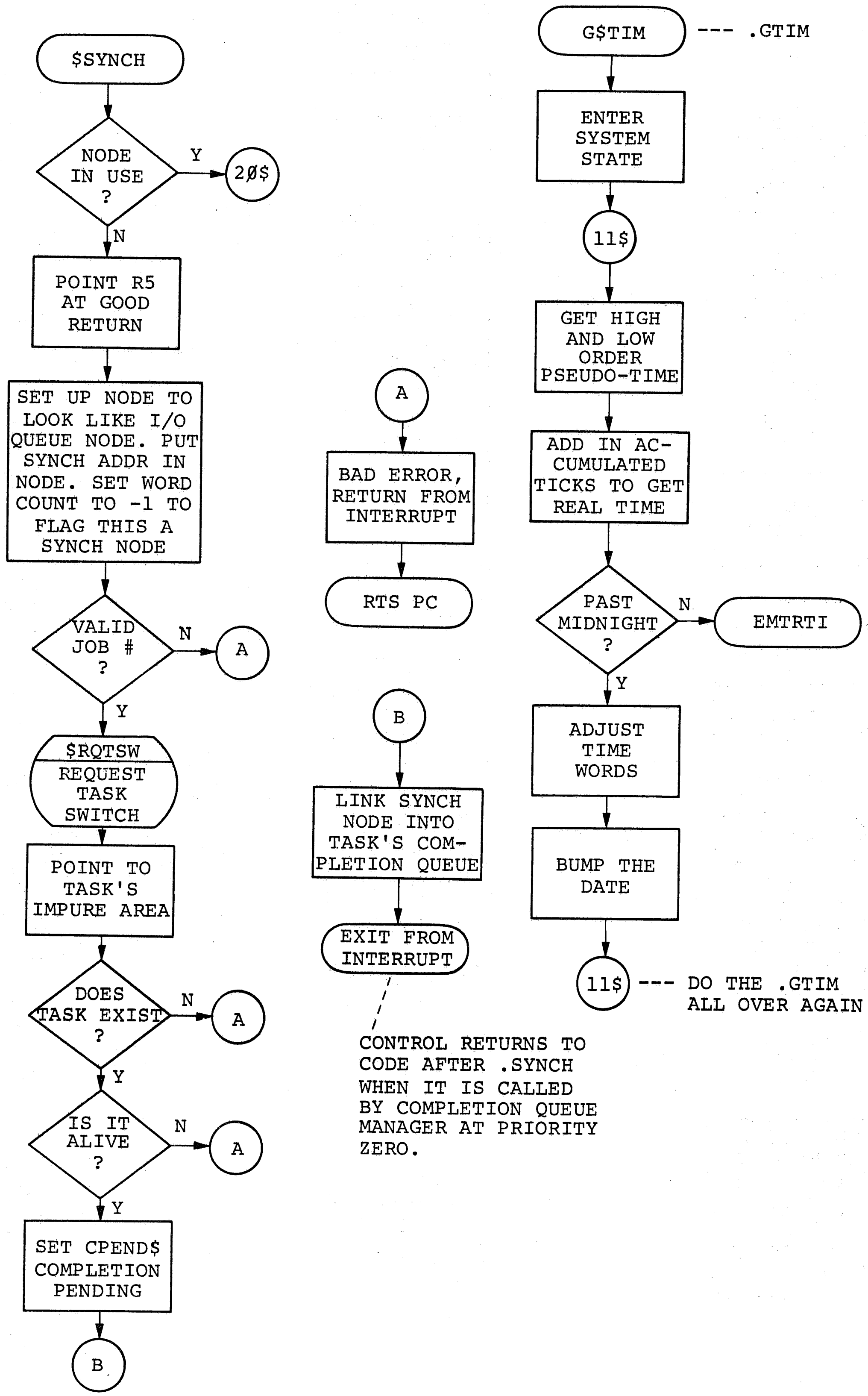


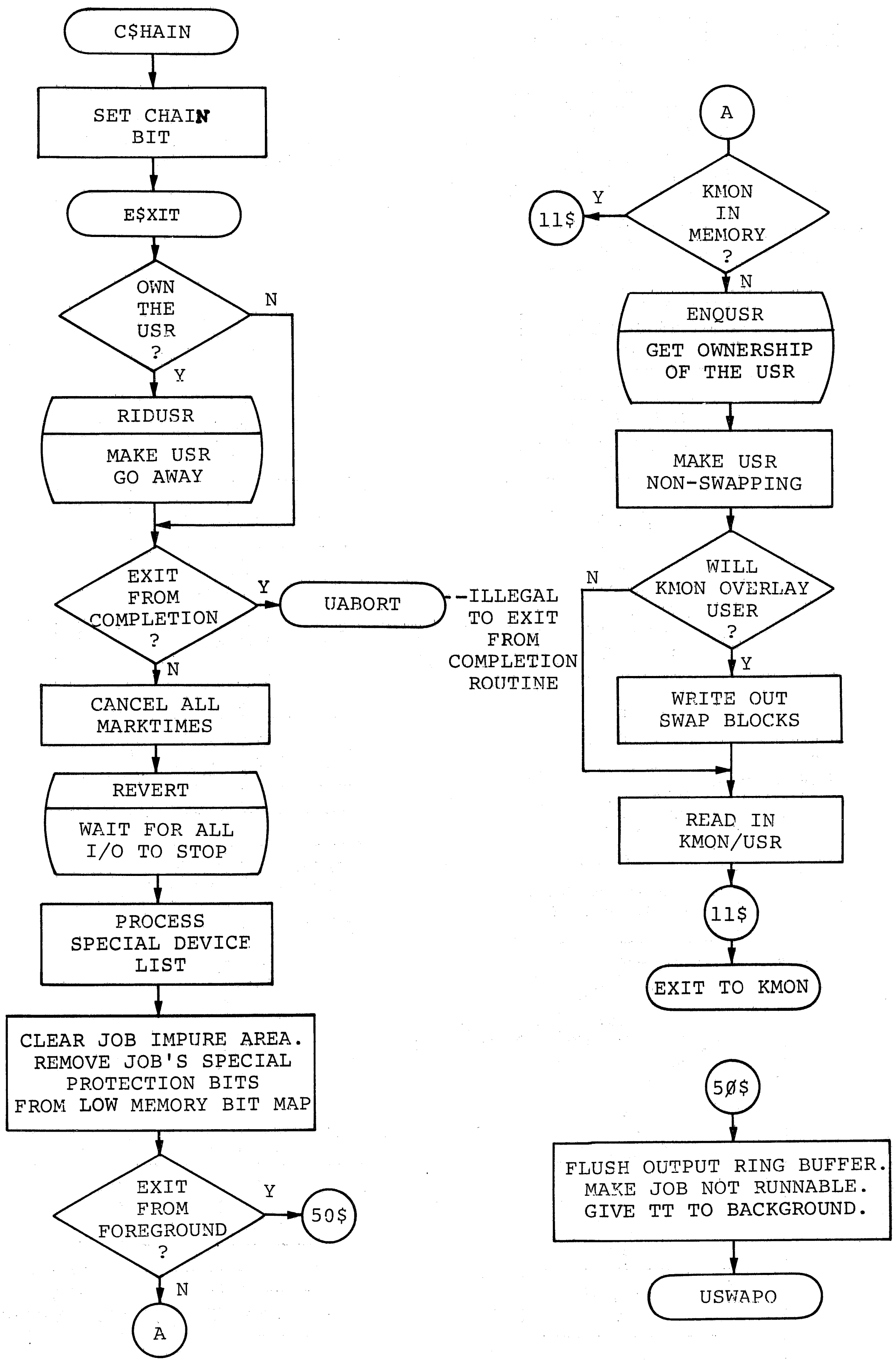

S$AVST
--- SAVESTATUS
CHANNEL OPEN ?
N
Y
VIA AN .ENTER ?
Y
N
---ERROR
E5ER1
TRANSFER STATUS WORD, MAKE CHANNEL INACTIVE
TRANSFER REST OF CHANNEL
EMTRTI
R$OPEN
CHANNEL ACTIVE ?
Y
N
E5ERØ
RETURN ERROR Ø
EMTRTI
TRANSFER FROM SAVE AREA TO CHANNEL
EMTRTI
M$WAIT
POINT TO MESSAGE CHANNEL
W$AIT
CHANNEL ACTIVE ?
N
Y
E5ERØ
CHWAIT
WAIT FOR CHANNEL NOT BUSY
HARD ERROR ?
N
Y
EMTRTI
CLEAR HARD ERROR BIT IN CHANNEL
E5ER1
RETURN ERROR 1
EMTRTI
G$TJB
RETURN JOB#, PROGRAM LIMITS, START OF CHANNELS
EMTRTI
C$STAT
RETURN DEVICE INDEX, CSW
CHANNEL ACTIVE ?
N
Y
E5ERØ
RETURN FILE START BLOCK, LENGTH, HIGH-EST BLOCK
RETURN DE-VICE UNIT # AND RAD5Ø NAME
EMTRTI

.CDFN, .TWAIT, .SPND, .RSUM,
.CNTXSW, .SFPA, .TRPSET, .DEVICE

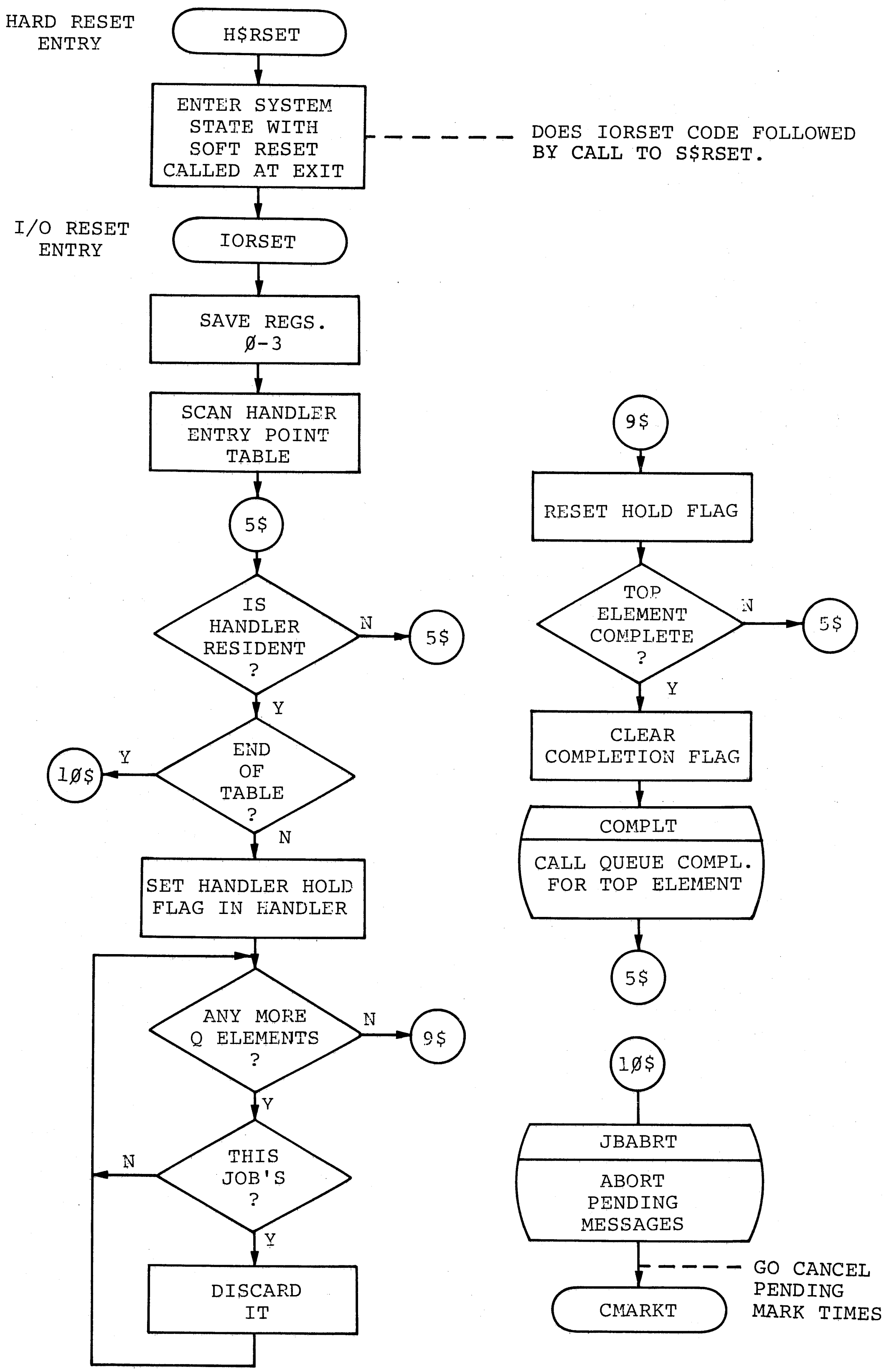


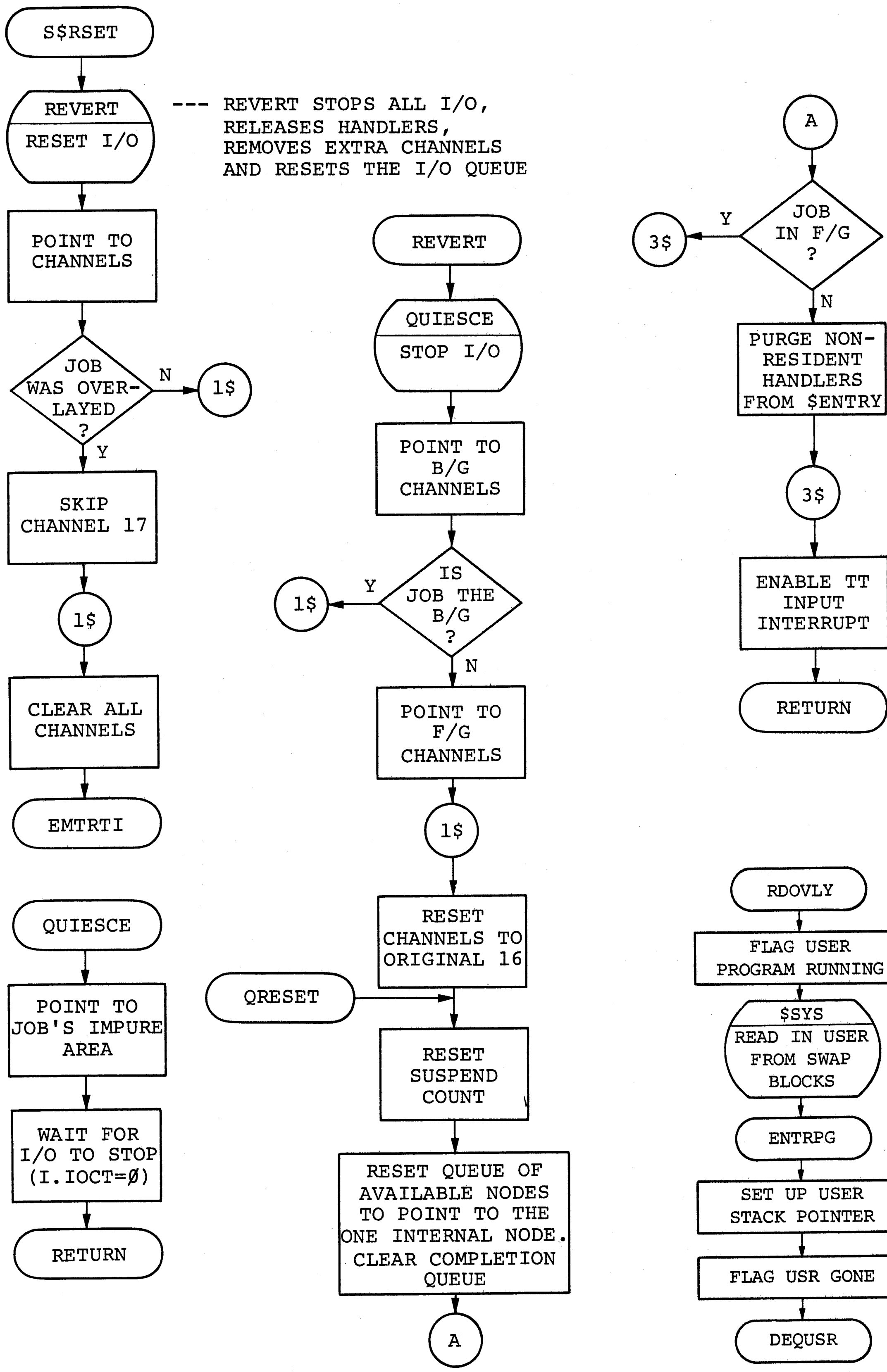

B
CHKSP
CHECK ADDRESS
WITH-IN JOB LIMITS ?
N
EMTERØ
Y
8$
BUMP POINTER
S$TRAP
--- .TRPSET
BUMP POINTER
D$VICE
--- .DEVICE
POINT TO WORD TO FILL
1Ø$
MOVE ADDRESS FROM STACK TO IMPURE AREA
EMTRTI
MARKTM
--- SUBROUTINE USED BY .MRKT AND .TWAIT
GET NODE FROM AVAIL QUEUE
SET UP HIGH AND LOW OR-DER TIME
SET UP COMPLETION ADDRESS
CLEAR CHANNEL OFFSET
STORE I.D. AND JOB #
ENTER SYSTEM STATE
SUBTRACT PSEUDO-CLOCK TO MAKE TIME SYSTEM RELATIVE
LINK INTO CLOCK Q IN ORDER OF EXPIRATION
LEAVE SYSTEM STATE
RETURN
CHKSP
ADDRESS ZERO ?
Y
1$
N
USR RUNNING ?
Y
1$
N
ADDRESS ODD ?
Y
2$
N
THIS JOB IS B/G ?
N
Y
IS IT KMON ?
Y
1$
ASSUME NO ERRORS FROM KMON
N
WITH-IN JOB LIMITS ?
N
2$
Y
GOOD RETURN ---
1$
BUMP RETURN TO GOOD ADDRESS
ERROR RETURN ---
2$
RETURN


$SYNCH
NODE IN USE ?
Y
20$
N
POINT R5 AT GOOD RETURN
SET UP NODE TO LOOK LIKE I/O QUEUE NODE. PUT SYNCH ADDR IN NODE. SET WORD COUNT TO -1 TO FLAG THIS A SYNCH NODE
VALID JOB # ?
N
A
Y
$RQTSW
REQUEST TASK SWITCH
POINT TO TASK'S IMPURE AREA
DOES TASK EXIST ?
N
A
Y
IS IT ALIVE ?
N
A
Y
SET CPEND$ COMPLETION PENDING
B
A
BAD ERROR, RETURN FROM INTERRUPT
RTS PC
B
LINK SYNCH NODE INTO TASK'S COM-PLETION QUEUE
EXIT FROM INTERRUPT
CONTROL RETURNS TO CODE AFTER .SYNCH WHEN IT IS CALLED BY COMPLETION QUEUE MANAGER AT PRIORITY ZERO.
G$TIM
--- .GTIM
ENTER SYSTEM STATE
11$
GET HIGH AND LOW ORDER PSEUDO-TIME
ADD IN AC-CUMULATED TICKS TO GET REAL TIME
PAST MIDNIGHT ?
N
EMTRTI
Y
ADJUST TIME WORDS
BUMP THE DATE
11$
--- DO THE .GTIM ALL OVER AGAIN

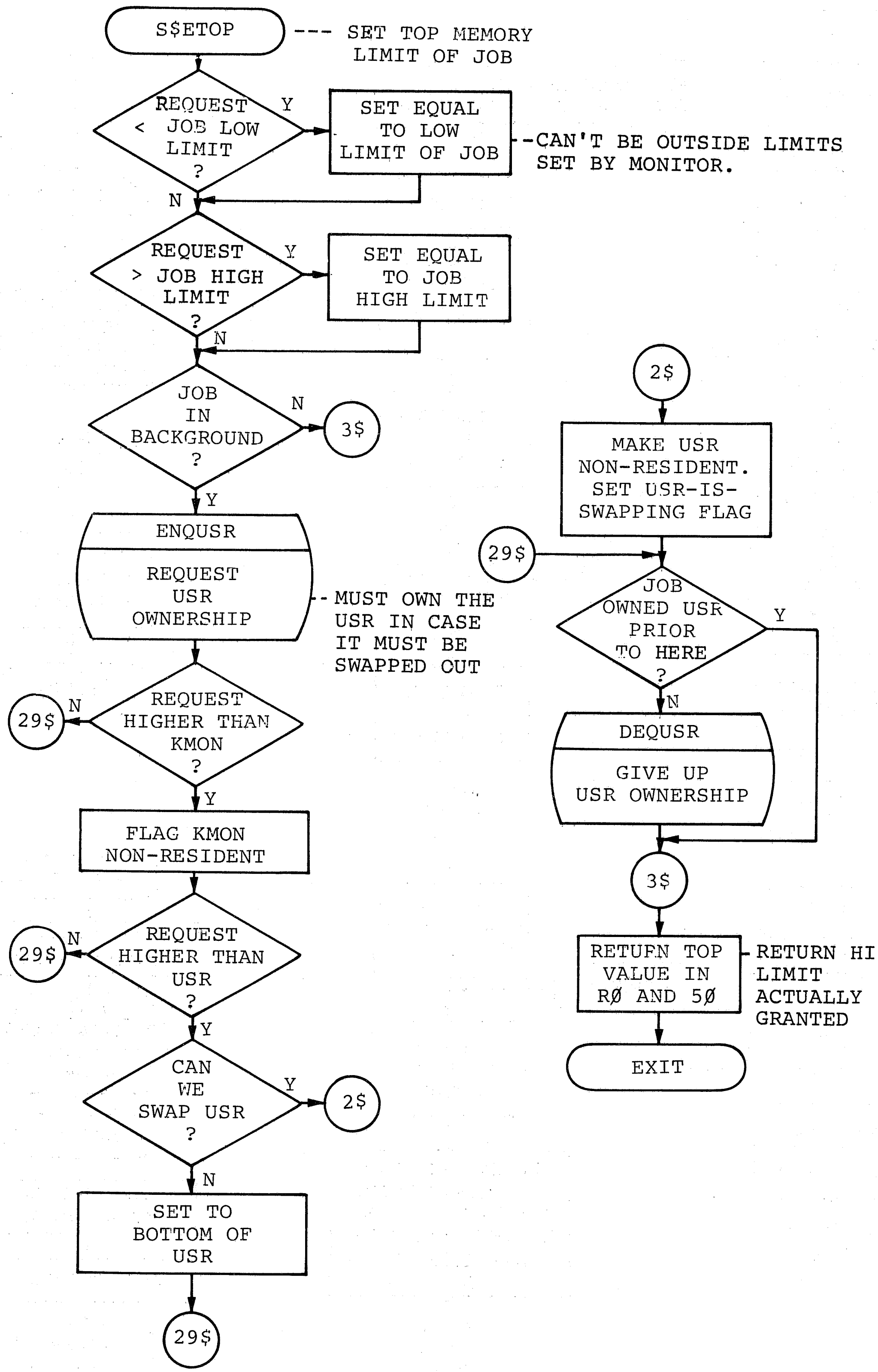

C$HAIN
SET CHAIN BIT
E$XIT
OWN THE USR ?
N
Y
RIDUSR
MAKE USR GO AWAY
EXIT FROM COMPLETION ?
Y
UABORT
--ILLEGAL TO EXIT FROM COMPLETION ROUTINE
N
CANCEL ALL MARKTIMES
REVERT
WAIT FOR ALL I/O TO STOP
PROCESS SPECIAL DEVICE LIST
CLEAR JOB IMPURE AREA. REMOVE JOB'S SPECIAL PROTECTION BITS FROM LOW MEMORY BIT MAP
EXIT FROM FOREGROUND ?
Y
50$
N
A
A
KMON IN MEMORY ?
Y
11$
N
ENQUSR
GET OWNERSHIP OF THE USR
MAKE USR NON-SWAPPING
WILL KMON OVERLAY USER ?
N
Y
WRITE OUT SWAP BLOCKS
READ IN KMON/USR
11$
EXIT TO KMON
5Ø$
FLUSH OUTPUT RING BUFFER. MAKE JOB NOT RUNNABLE. GIVE TT TO BACKGROUND.
USWAPO

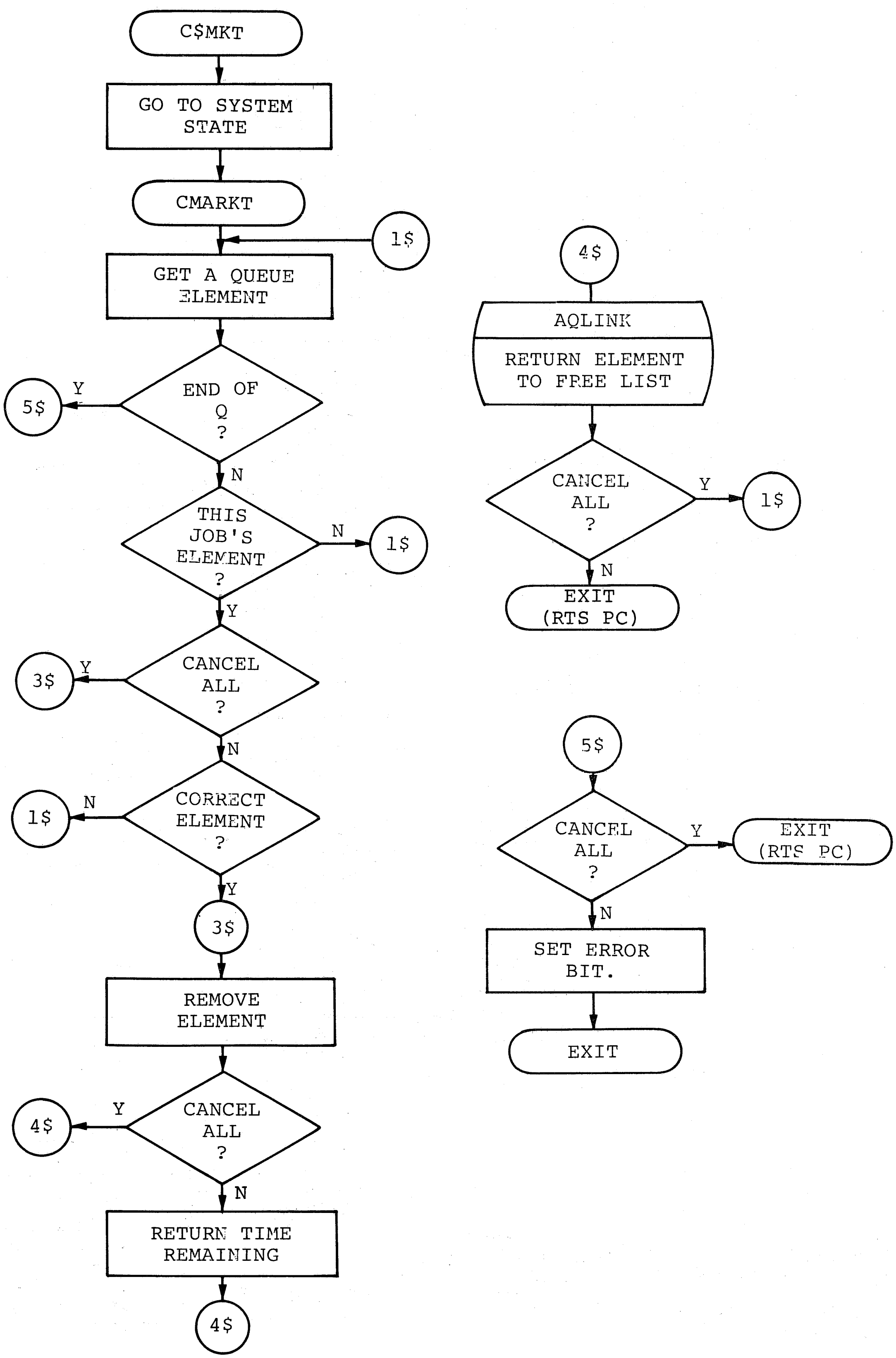
HARD AND SOFT RESET



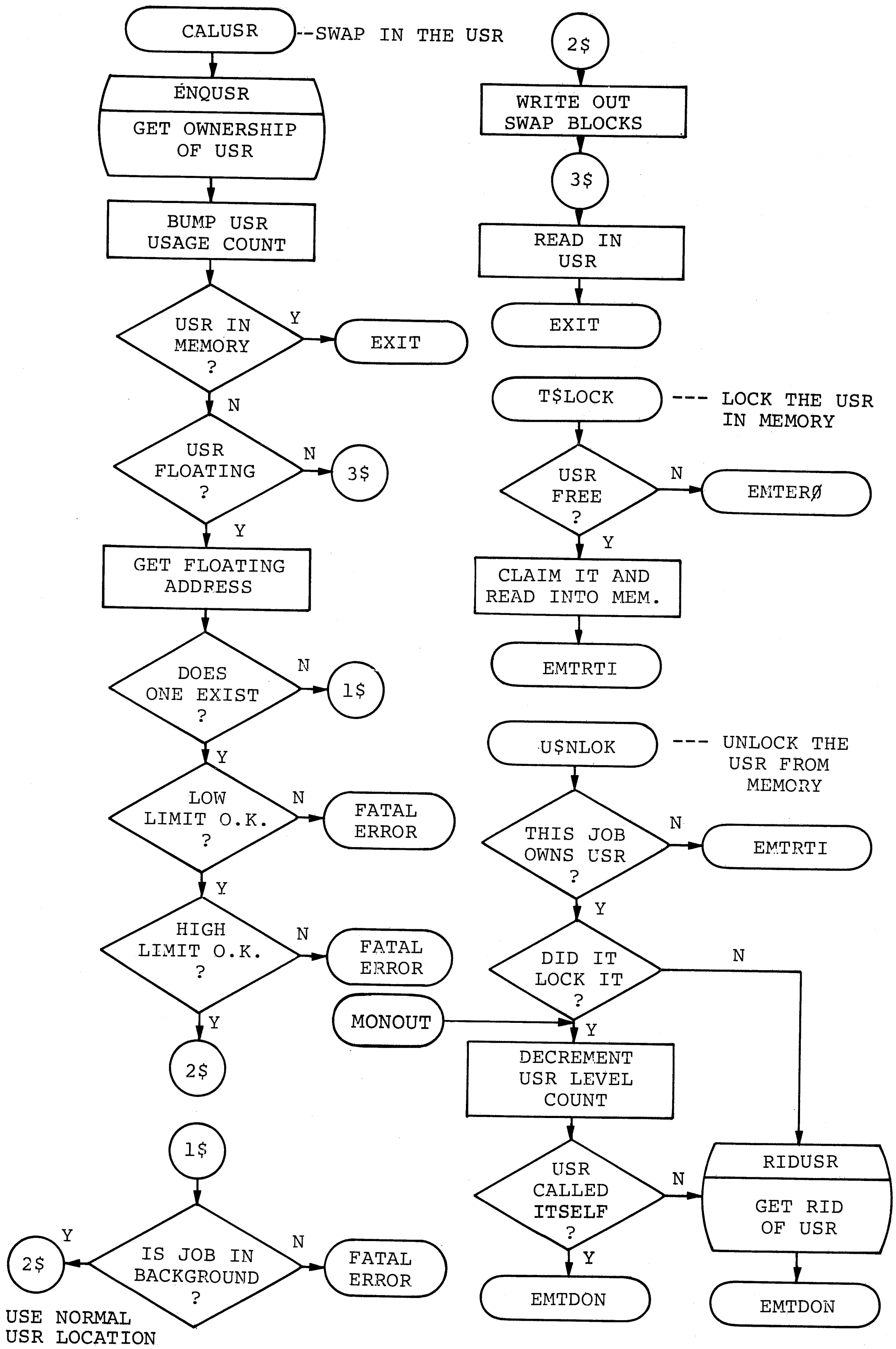

S$RSET
REVERT
RESET I/O
--- REVERT STOPS ALL I/O, RELEASES HANDLERS, REMOVES EXTRA CHANNELS AND RESETS THE I/O QUEUE
POINT TO CHANNELS
JOB WAS OVER-LAYED ?
N
1$
Y
SKIP CHANNEL 17
1$
CLEAR ALL CHANNELS
EMTRTI
QUIESCE
POINT TO JOB'S IMPURE AREA
WAIT FOR I/O TO STOP (I.IOCT=Ø)
RETURN
REVERT
QUIESCE
STOP I/O
POINT TO B/G CHANNELS
IS JOB THE B/G ?
Y
1$
N
POINT TO F/G CHANNELS
1$
RESET CHANNELS TO ORIGINAL 16
QRESET
RESET SUSPEND COUNT
RESET QUEUE OF AVAILABLE NODES TO POINT TO THE ONE INTERNAL NODE. CLEAR COMPLETION QUEUE
A
A
JOB IN F/G ?
Y
3$
N
PURGE NON-RESIDENT HANDLERS FROM $ENTRY
3$
ENABLE TT INPUT INTERRUPT
RETURN
RDOVLY
FLAG USER PROGRAM RUNNING
$SYS
READ IN USER FROM SWAP BLOCKS
ENTRPG
SET UP USER STACK POINTER
FLAG USR GONE
DEQUSR

.SETTOP


S$ETOP
SET TOP MEMORY LIMIT OF JOB
REQUEST < JOB LOW LIMIT ?
Y
N
SET EQUAL TO LOW LIMIT OF JOB
CAN'T BE OUTSIDE LIMITS SET BY MONITOR.
REQUEST > JOB HIGH LIMIT ?
SET EQUAL TO JOB HIGH LIMIT
JOB IN BACKGROUND ?
3$
ENQUSR
REQUEST USR OWNERSHIP
MUST OWN THE USR IN CASE IT MUST BE SWAPPED OUT
REQUEST HIGHER THAN KMON ?
29$
FLAG KMON NON-RESIDENT
REQUEST HIGHER THAN USR ?
CAN WE SWAP USR ?
2$
SET TO BOTTOM OF USR
MAKE USR NON-RESIDENT. SET USR-IS-SWAPPING FLAG
JOB OWNED USR PRIOR TO HERE ?
DEQUSR
GIVE UP USR OWNERSHIP
RETURN TOP VALUE IN RØ AND 5Ø
RETURN HI LIMIT ACTUALLY GRANTED
EXIT

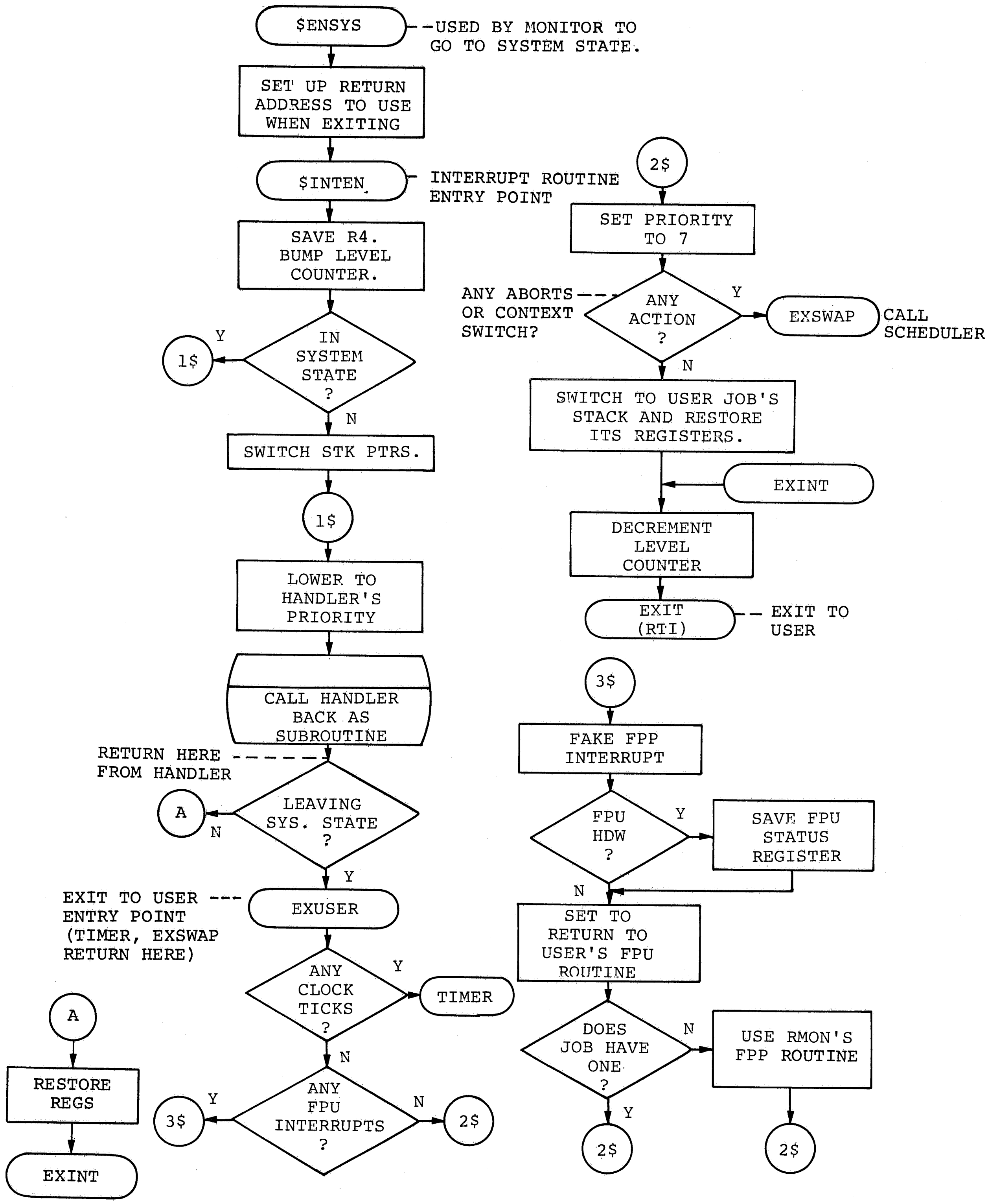

C$MKT
GO TO SYSTEM STATE
CMARKT
1$
GET A QUEUE ELEMENT
END OF Q ?
Y
5$
N
THIS JOB'S ELEMENT ?
N
1$
Y
CANCEL ALL ?
Y
3$
N
CORRECT ELEMENT ?
N
1$
Y
3$
REMOVE ELEMENT
CANCEL ALL ?
Y
4$
N
RETURN TIME REMAINING
4$
4$
AQLINK
RETURN ELEMENT TO FREE LIST
CANCEL ALL ?
Y
1$
N
EXIT (RTS PC)
5$
CANCEL ALL ?
Y
EXIT (RTS PC)
N
SET ERROR BIT.
EXIT

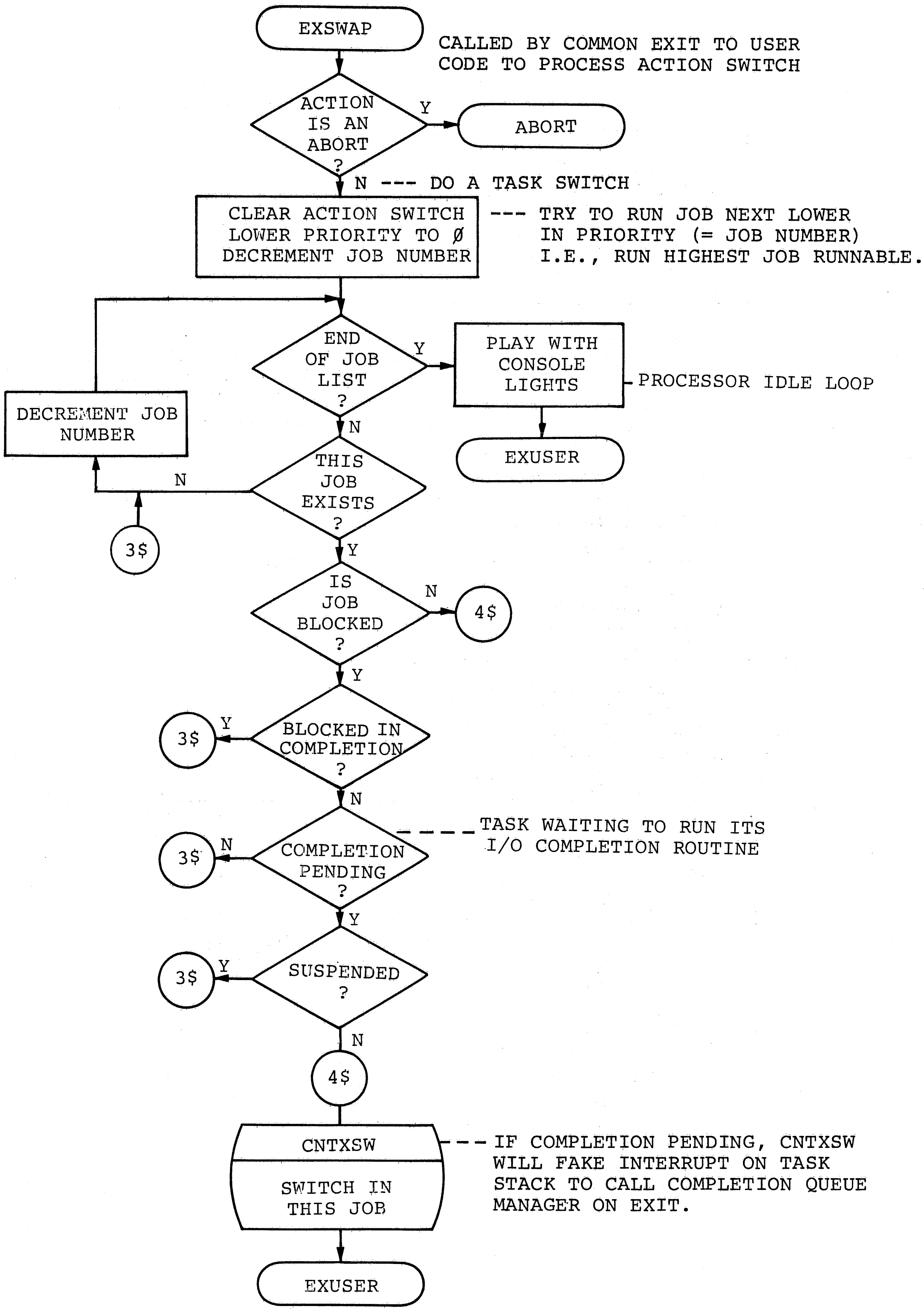
SWAP IN USR, LOCK/UNLOCK USR



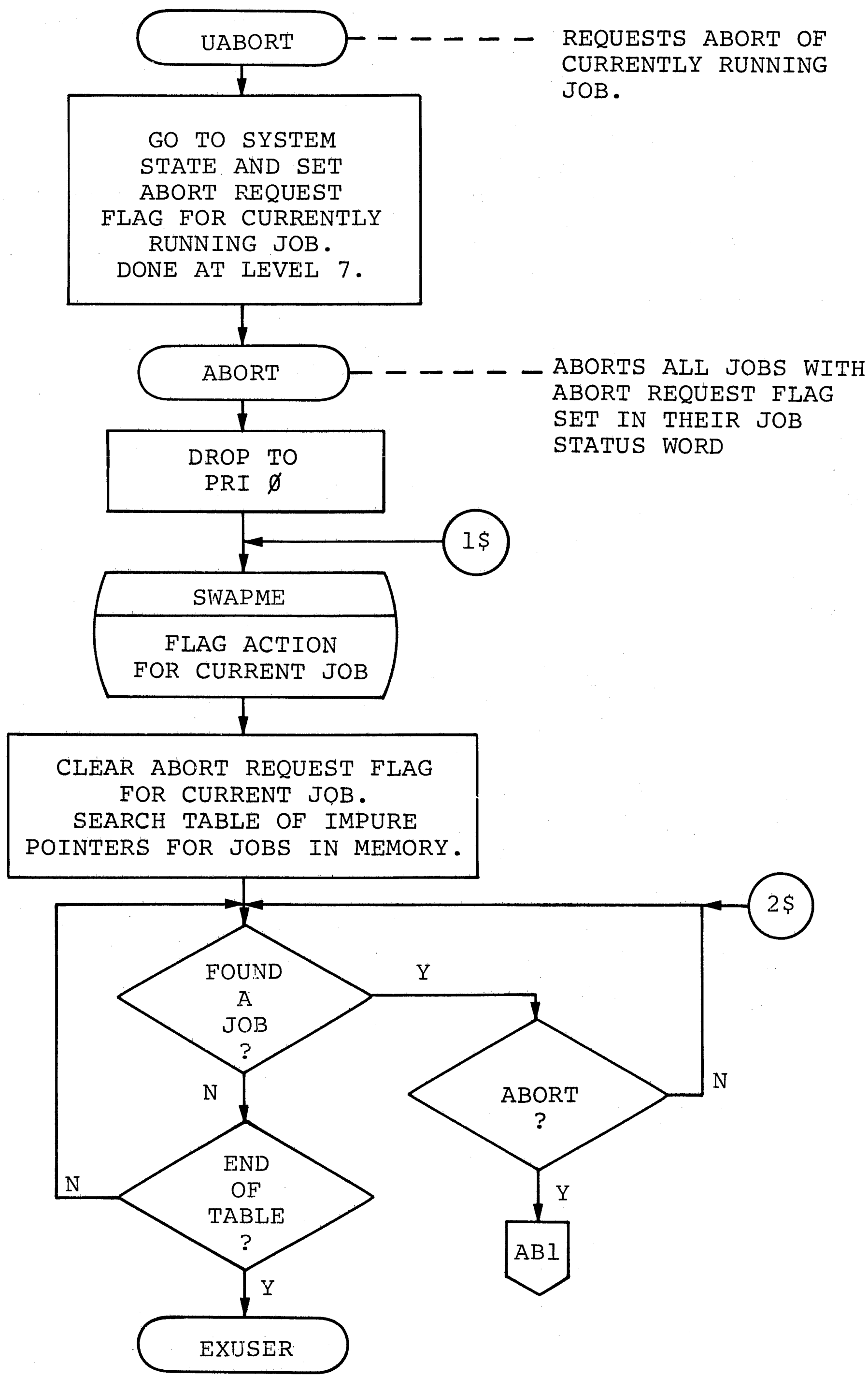
E.5.2 Job Arbitration, Error Processing

COMMON INTERRUPT
ENTRY AND EXIT

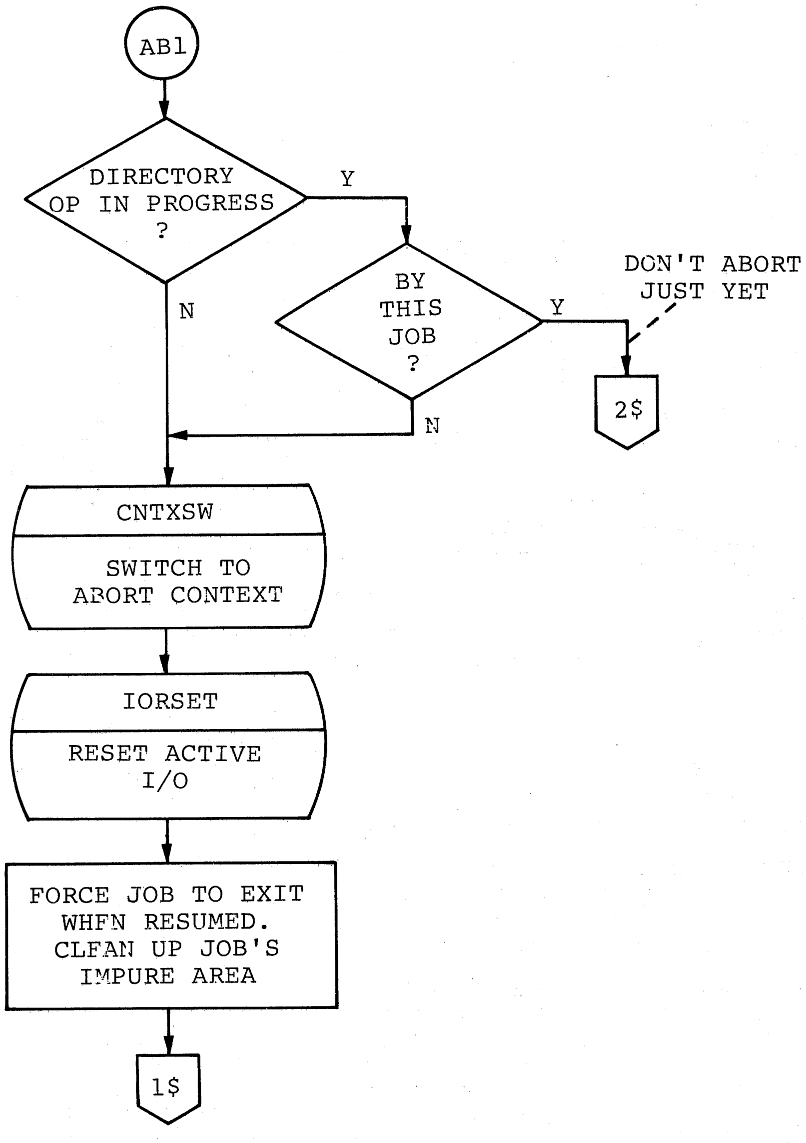



EXSWAP
CALLED BY COMMON EXIT TO USER CODE TO PROCESS ACTION SWITCH
ACTION IS AN ABORT ?
Y
ABORT
N --- DO A TASK SWITCH
CLEAR ACTION SWITCH
LOWER PRIORITY TO Ø
DECREMENT JOB NUMBER
--- TRY TO RUN JOB NEXT LOWER IN PRIORITY (= JOB NUMBER) I.E., RUN HIGHEST JOB RUNNABLE.
END OF JOB LIST ?
Y
PLAY WITH CONSOLE LIGHTS
PROCESSOR IDLE LOOP
EXUSER
N
DECREMENT JOB NUMBER
THIS JOB EXISTS ?
N
3$
Y
IS JOB BLOCKED ?
N
4$
Y
BLOCKED IN COMPLETION ?
Y
3$
N
COMPLETION PENDING ?
TASK WAITING TO RUN ITS I/O COMPLETION ROUTINE
N
3$
Y
SUSPENDED ?
Y
3$
N
4$
CNTXSW
SWITCH IN THIS JOB
IF COMPLETION PENDING, CNTXSW WILL FAKE INTERRUPT ON TASK STACK TO CALL COMPLETION QUEUE MANAGER ON EXIT.
EXUSER

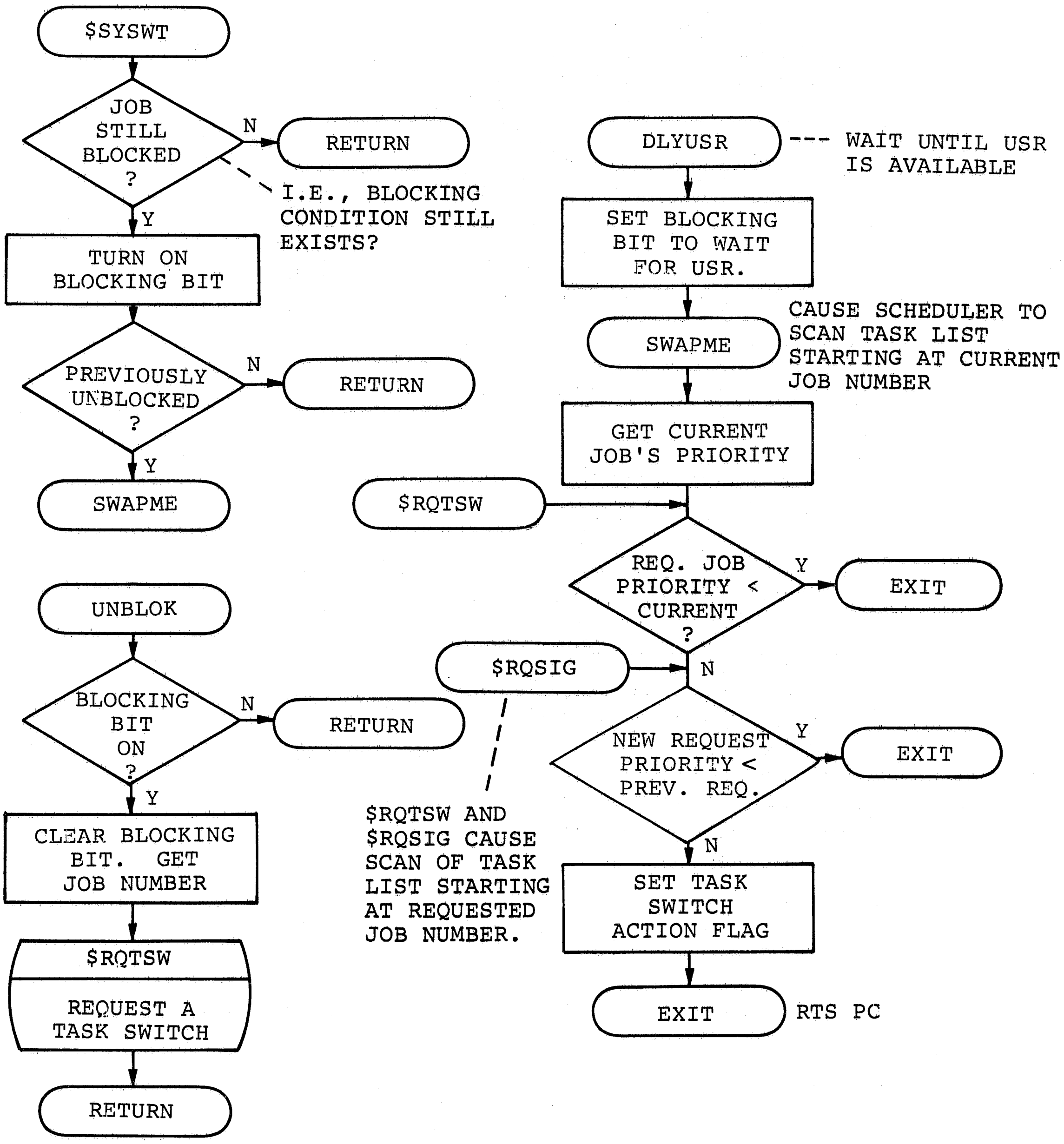
JOB ABORT

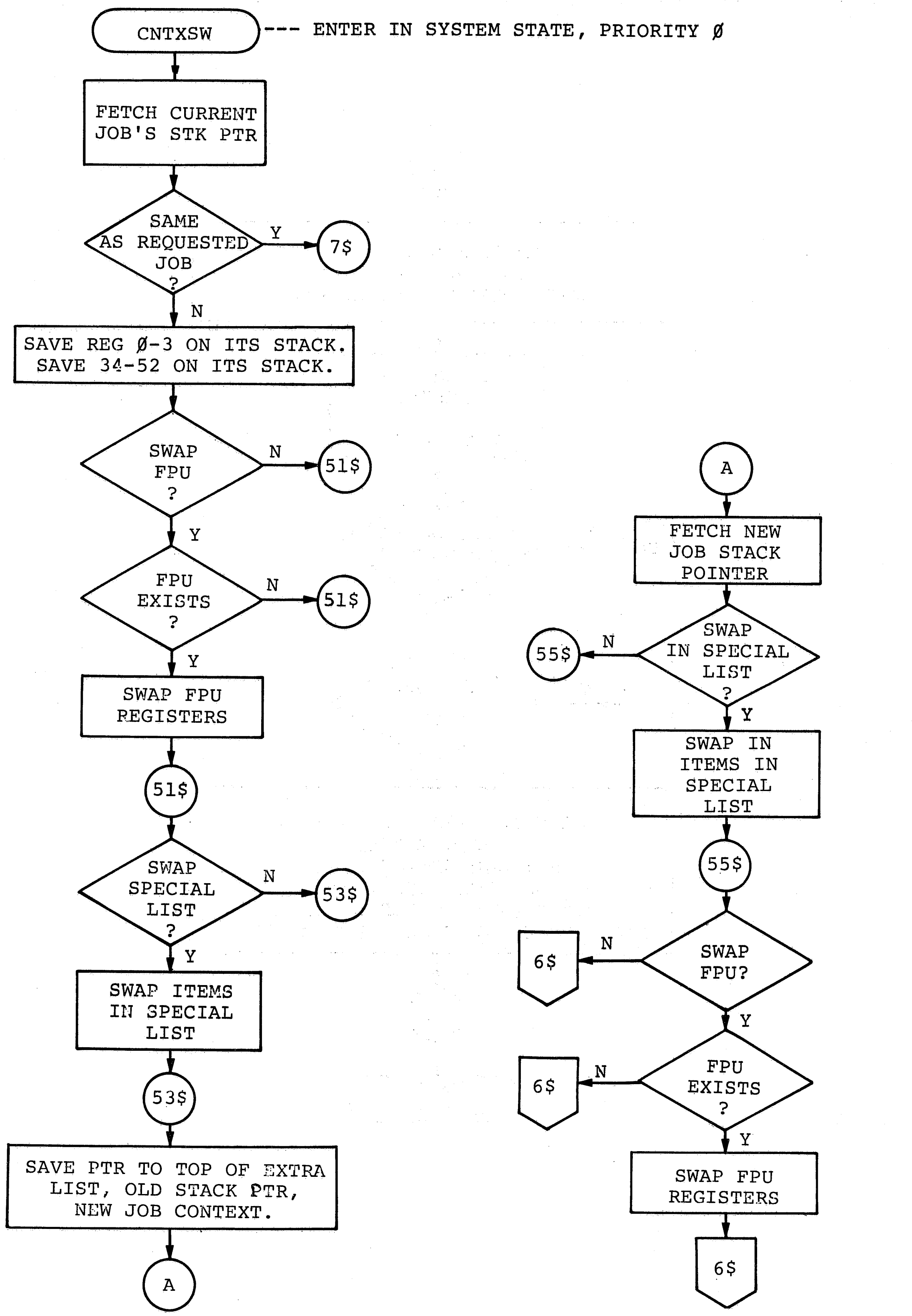


UABORT
REQUESTS ABORT OF CURRENTLY RUNNING JOB.
GO TO SYSTEM STATE AND SET ABORT REQUEST FLAG FOR CURRENTLY RUNNING JOB. DONE AT LEVEL 7.
ABORT
ABORTS ALL JOBS WITH ABORT REQUEST FLAG SET IN THEIR JOB STATUS WORD
DROP TO PRI Ø
1$
SWAPME
FLAG ACTION FOR CURRENT JOB
CLEAR ABORT REQUEST FLAG FOR CURRENT JOB. SEARCH TABLE OF IMPURE POINTERS FOR JOBS IN MEMORY.
2$
FOUND A JOB ?
Y
N
ABORT ?
N
Y
AB1
END OF TABLE ?
N
Y
EXUSER

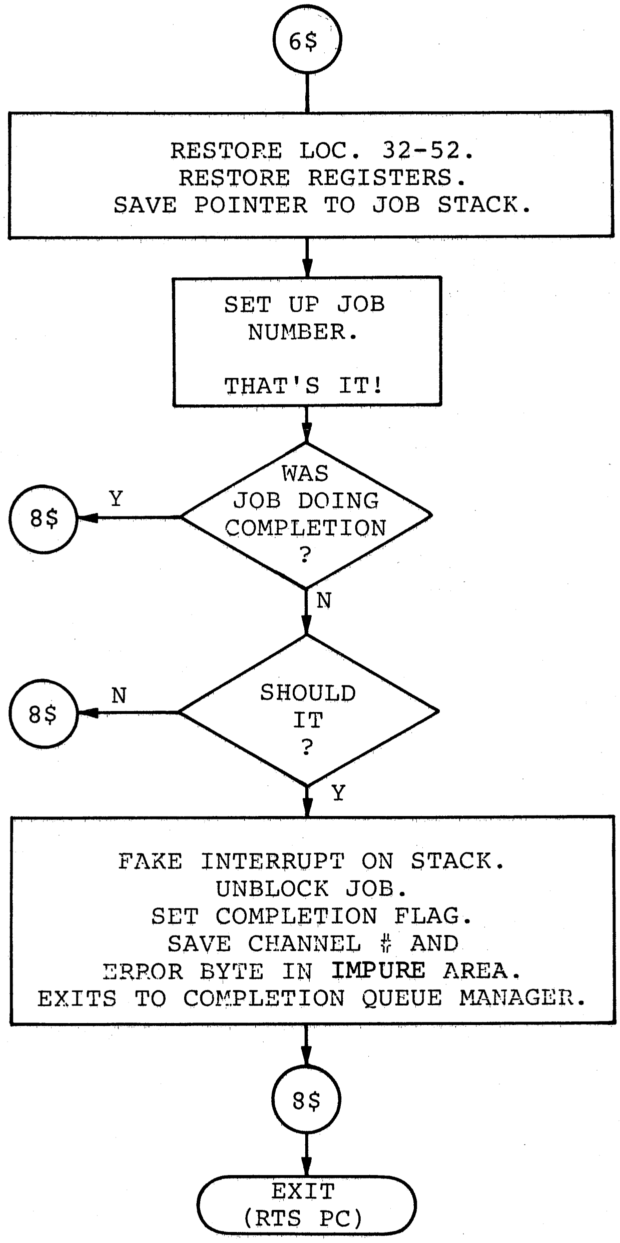

AB1
DIRECTORY OP IN PROGRESS ?
Y
N
BY THIS JOB ?
Y
N
DON'T ABORT JUST YET
2$
CNTXSW
SWITCH TO ABORT CONTEXT
IORSET
RESET ACTIVE I/O
FORCE JOB TO EXIT WHEN RESUMED. CLEAN UP JOB'S IMPURE AREA
1$

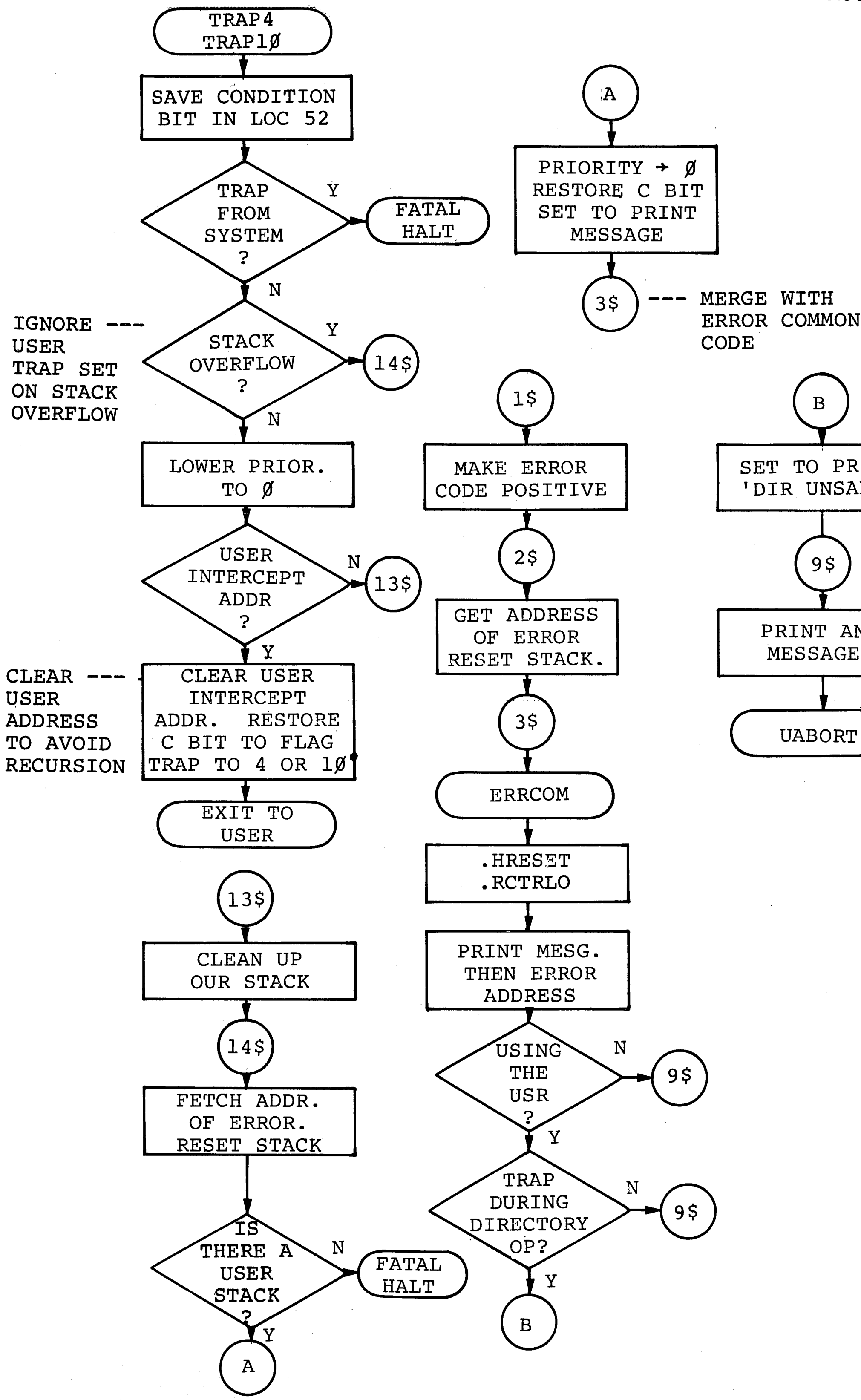
BLOCK A TASK/UNBLOCK A TASK
REQUEST TASK SWITCH




CNTXSW
--- ENTER IN SYSTEM STATE, PRIORITY Ø
FETCH CURRENT JOB'S STK PTR
SAME AS REQUESTED JOB ?
Y
7$
N
SAVE REG Ø-3 ON ITS STACK. SAVE 34-52 ON ITS STACK.
SWAP FPU ?
N
51$
Y
FPU EXISTS ?
N
51$
Y
SWAP FPU REGISTERS
51$
SWAP SPECIAL LIST ?
N
53$
Y
SWAP ITEMS IN SPECIAL LIST
53$
SAVE PTR TO TOP OF EXTRA LIST, OLD STACK PTR, NEW JOB CONTEXT.
A
A
FETCH NEW JOB STACK POINTER
SWAP IN SPECIAL LIST ?
N
55$
Y
SWAP IN ITEMS IN SPECIAL LIST
55$
SWAP FPU?
N
6$
Y
FPU EXISTS ?
N
6$
Y
SWAP FPU REGISTERS
6$

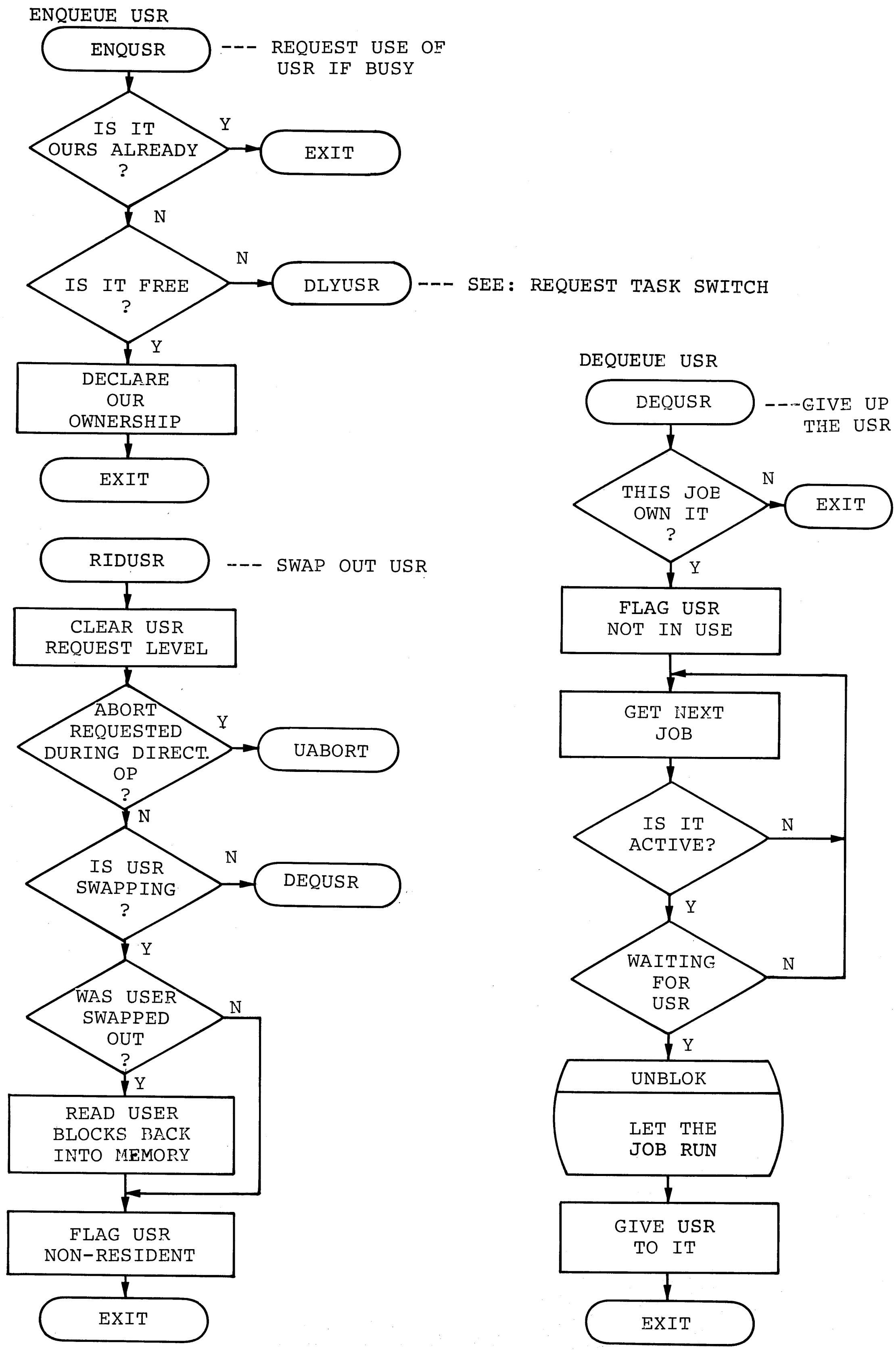
CHANGE CURRENT CONTEXT (CONT.)

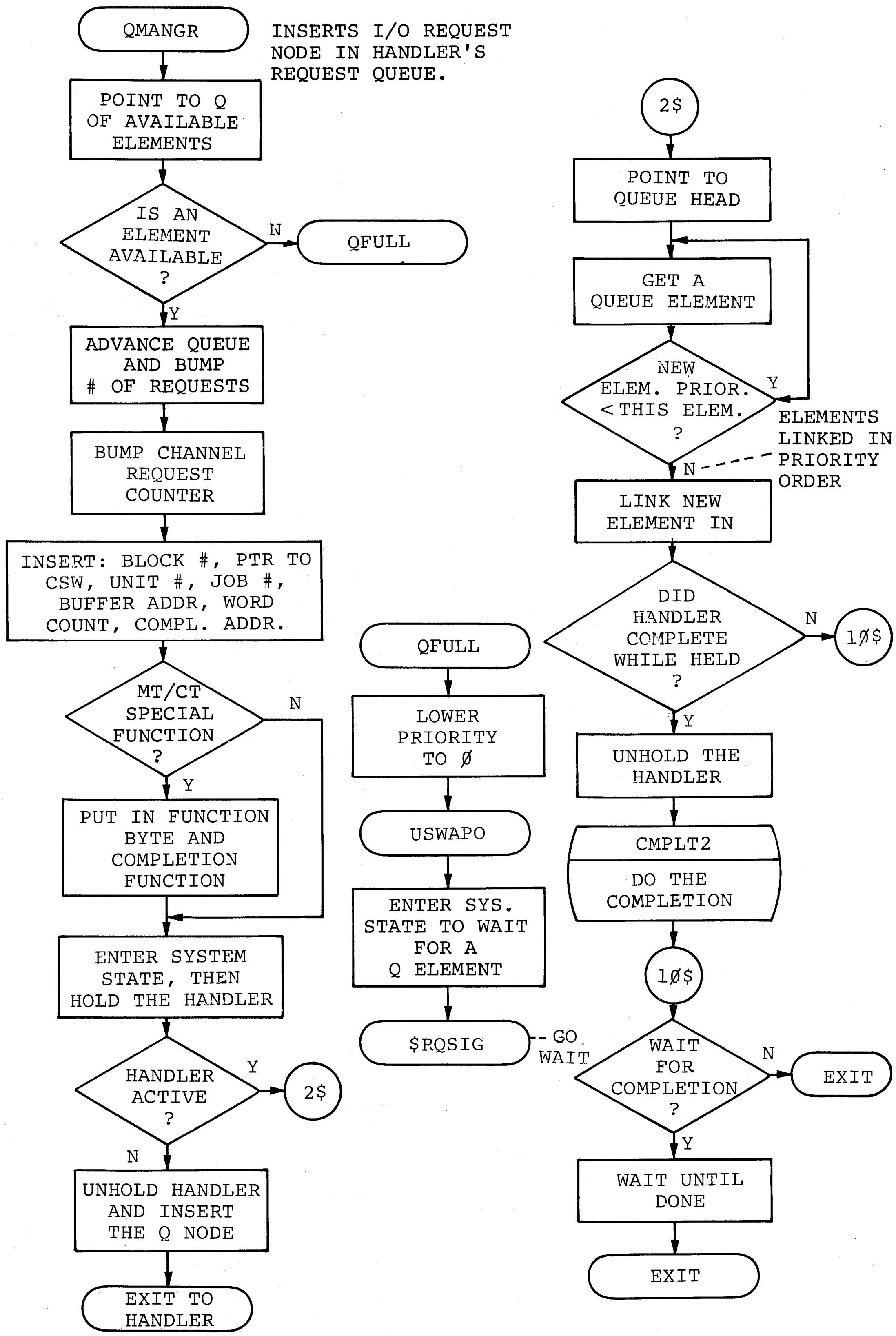



TRAP4
TRAP1Ø
SAVE CONDITION BIT IN LOC 52
TRAP FROM SYSTEM ?
Y
FATAL HALT
N
IGNORE --- USER TRAP SET ON STACK OVERFLOW
STACK OVERFLOW ?
Y
14$
N
LOWER PRIOR. TO Ø
USER INTERCEPT ADDR ?
N
13$
Y
CLEAR --- USER ADDRESS TO AVOID RECURSION
CLEAR USER INTERCEPT ADDR. RESTORE C BIT TO FLAG TRAP TO 4 OR 1Ø
EXIT TO USER
13$
CLEAN UP OUR STACK
14$
FETCH ADDR. OF ERROR. RESET STACK
IS THERE A USER STACK ?
N
FATAL HALT
Y
A
A
PRIORITY → Ø RESTORE C BIT SET TO PRINT MESSAGE
3$
--- MERGE WITH ERROR COMMON CODE
1$
MAKE ERROR CODE POSITIVE
2$
GET ADDRESS OF ERROR RESET STACK.
3$
ERRCOM
.HRESET .RCTRLO
PRINT MESG. THEN ERROR ADDRESS
USING THE USR ?
N
9$
Y
TRAP DURING DIRECTORY OP?
N
9$
Y
B
B
SET TO PRINT 'DIR UNSAFE'
9$
PRINT ANY MESSAGES
UABORT

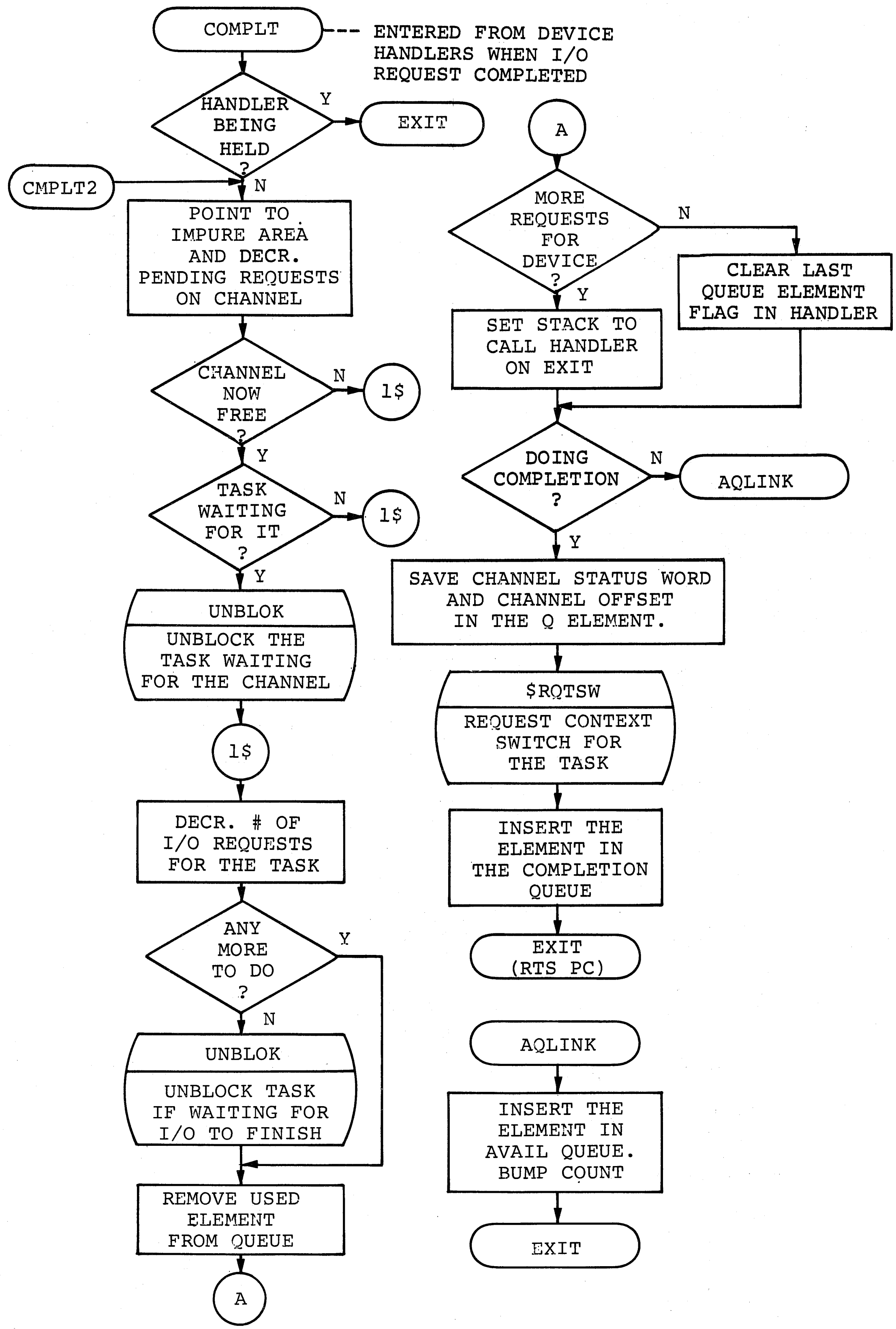
E.5.3 Queue Managers (I/O, USR, Completion)

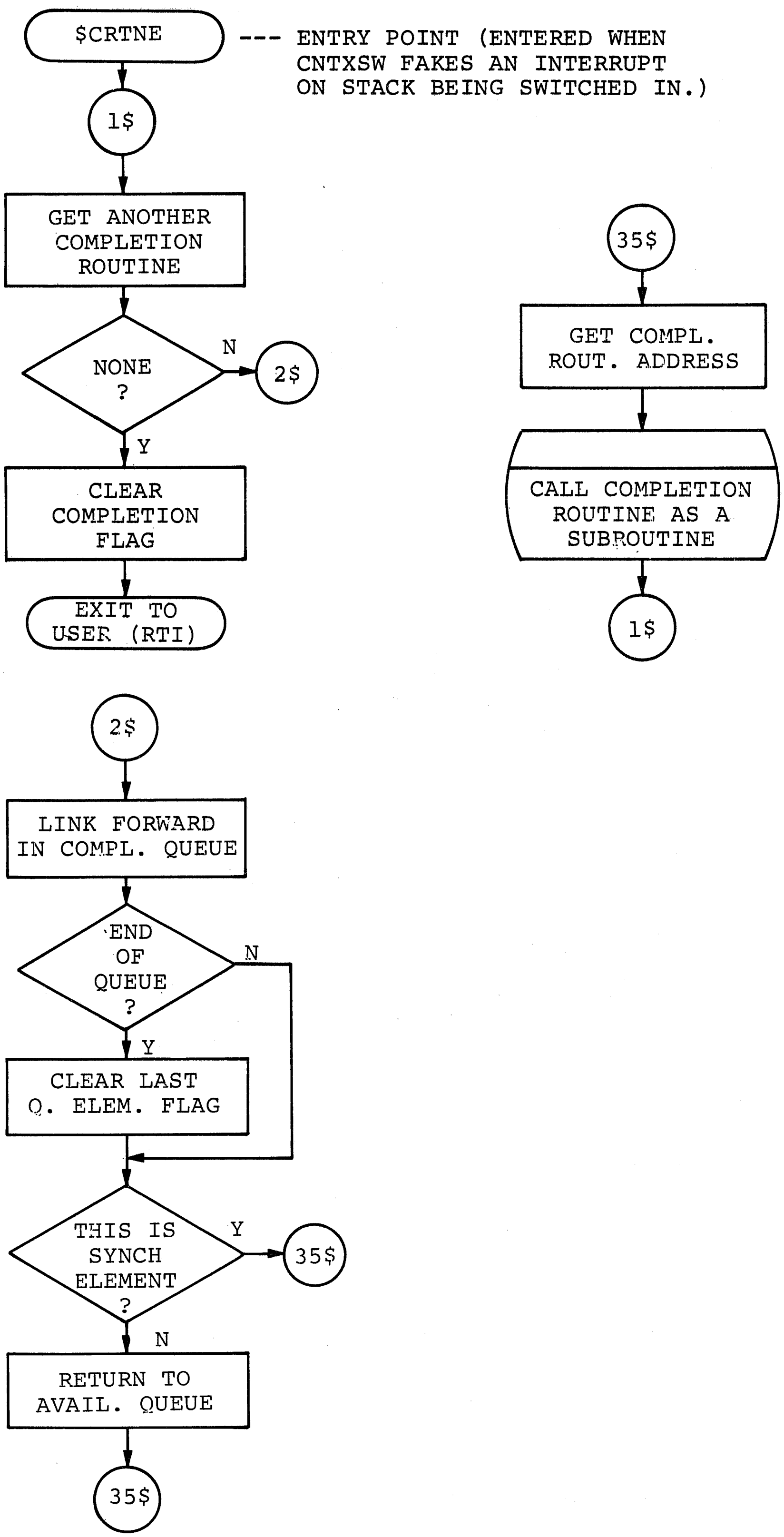
ENQUEUE/DEQUEUE USR




QMANGR
INSERTS I/O REQUEST NODE IN HANDLER'S REQUEST QUEUE.
POINT TO Q OF AVAILABLE ELEMENTS
IS AN ELEMENT AVAILABLE ?
N
QFULL
Y
ADVANCE QUEUE AND BUMP # OF REQUESTS
BUMP CHANNEL REQUEST COUNTER
INSERT: BLOCK #, PTR TO CSW, UNIT #, JOB #, BUFFER ADDR, WORD COUNT, COMPL. ADDR.
MT/CT SPECIAL FUNCTION ?
N
Y
PUT IN FUNCTION BYTE AND COMPLETION FUNCTION
ENTER SYSTEM STATE, THEN HOLD THE HANDLER
HANDLER ACTIVE ?
Y
2$
N
UNHOLD HANDLER AND INSERT THE Q NODE
EXIT TO HANDLER
QFULL
LOWER PRIORITY TO Ø
USWAPO
ENTER SYS. STATE TO WAIT FOR A Q ELEMENT
$RQSIG
GO WAIT
2$
POINT TO QUEUE HEAD
GET A QUEUE ELEMENT
NEW ELEM. PRIOR. < THIS ELEM. ?
Y
N
ELEMENTS LINKED IN PRIORITY ORDER
LINK NEW ELEMENT IN
DID HANDLER COMPLETE WHILE HELD ?
N
1Ø$
Y
UNHOLD THE HANDLER
CMPLT2
DO THE COMPLETION
1Ø$
WAIT FOR COMPLETION ?
N
EXIT
Y
WAIT UNTIL DONE
EXIT

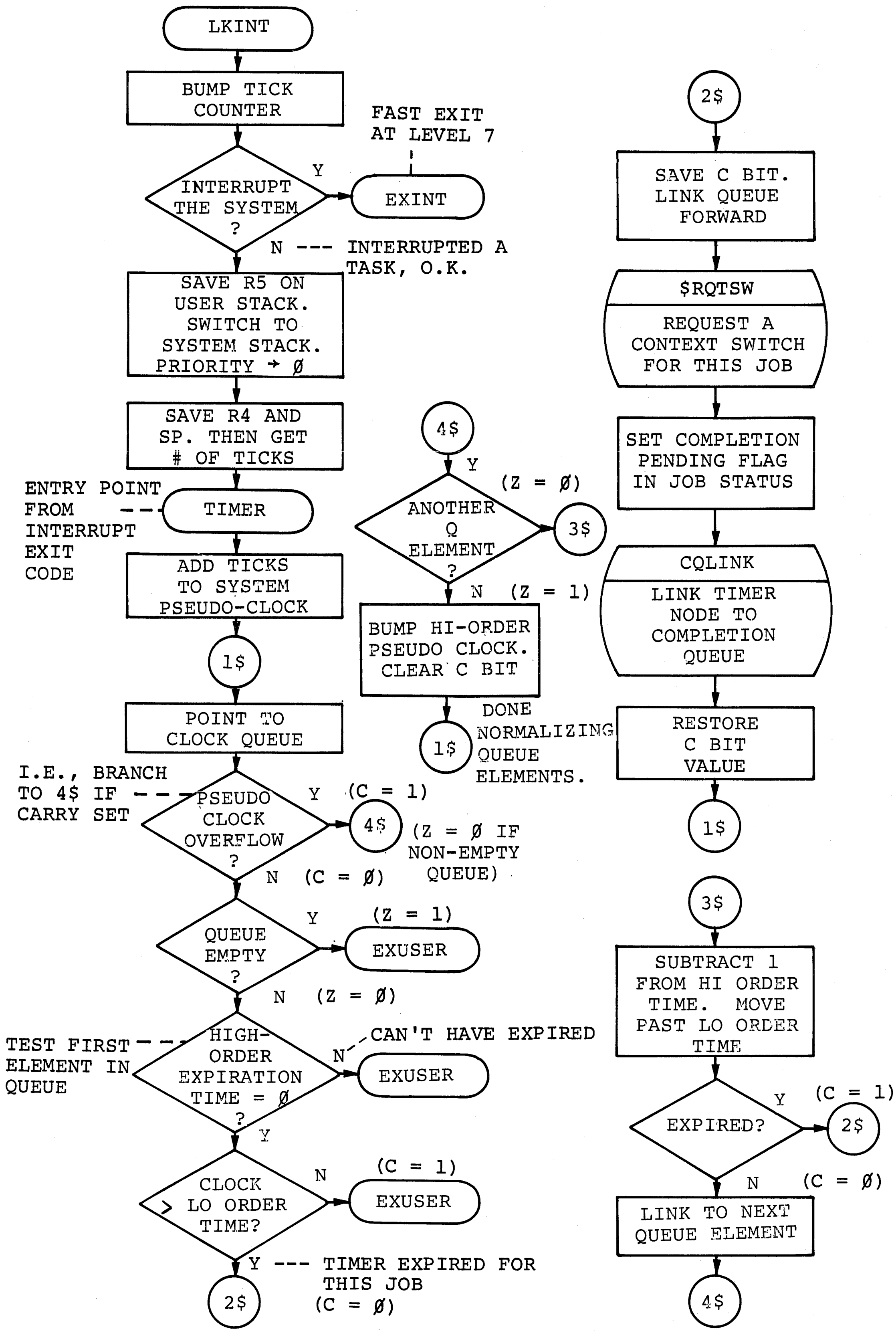
QUEUE COMPLETION




$CRTNE
--- ENTRY POINT (ENTERED WHEN CNTXSW FAKES AN INTERRUPT ON STACK BEING SWITCHED IN.)
1$
GET ANOTHER COMPLETION ROUTINE
NONE ?
N
2$
Y
CLEAR COMPLETION FLAG
EXIT TO USER (RTI)
2$
LINK FORWARD IN COMPL. QUEUE
END OF QUEUE ?
N
Y
CLEAR LAST Q. ELEM. FLAG
THIS IS SYNCH ELEMENT ?
Y
35$
N
RETURN TO AVAIL. QUEUE
35$
35$
GET COMPL. ROUT. ADDRESS
CALL COMPLETION ROUTINE AS A SUBROUTINE
1$

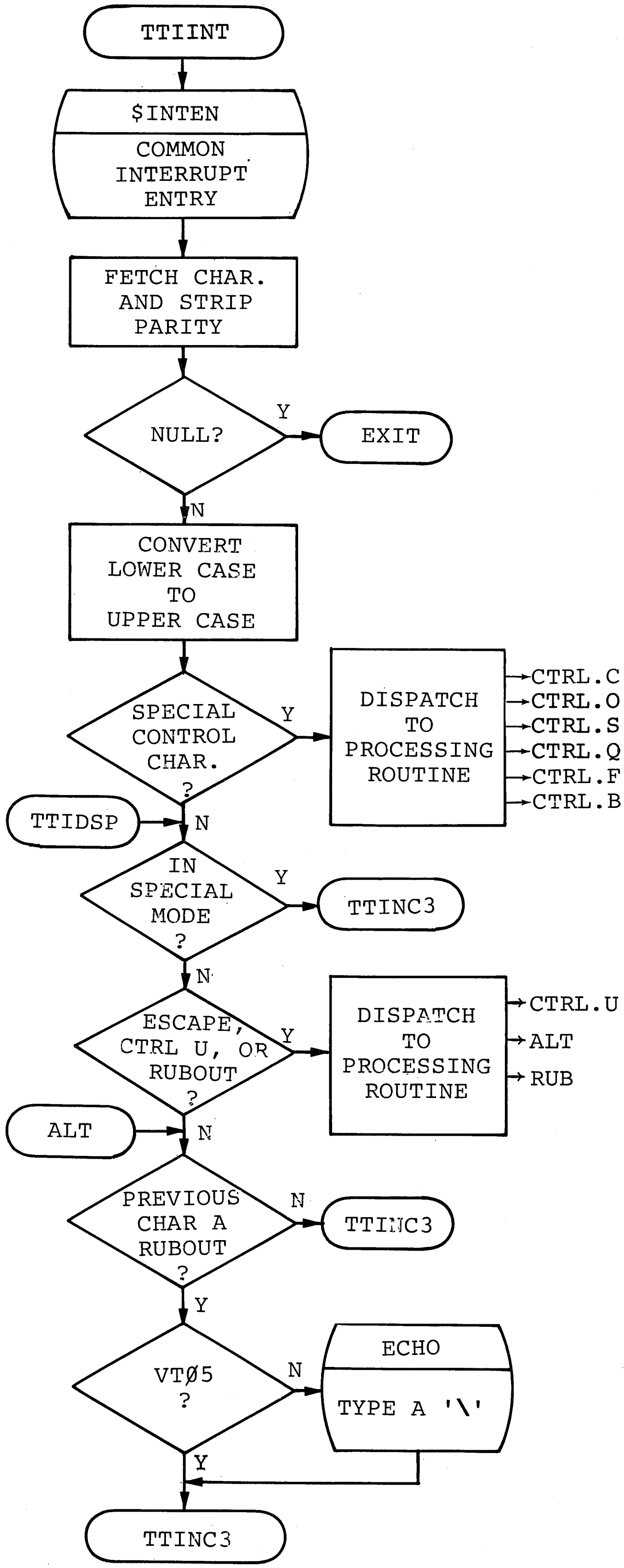
E.5.4 Clock Interrupt Service

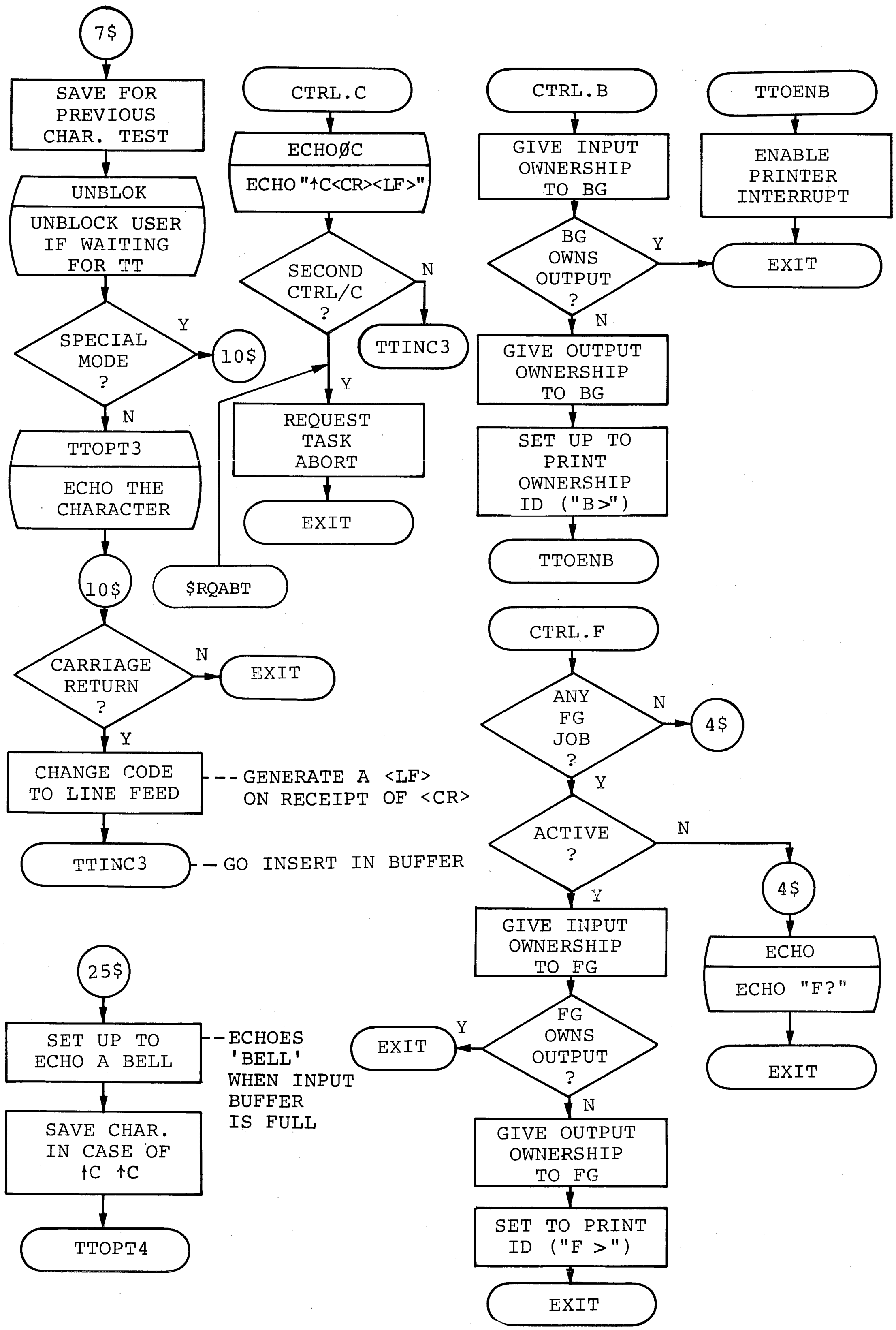
CLOCK INTERRUPT HANDLER



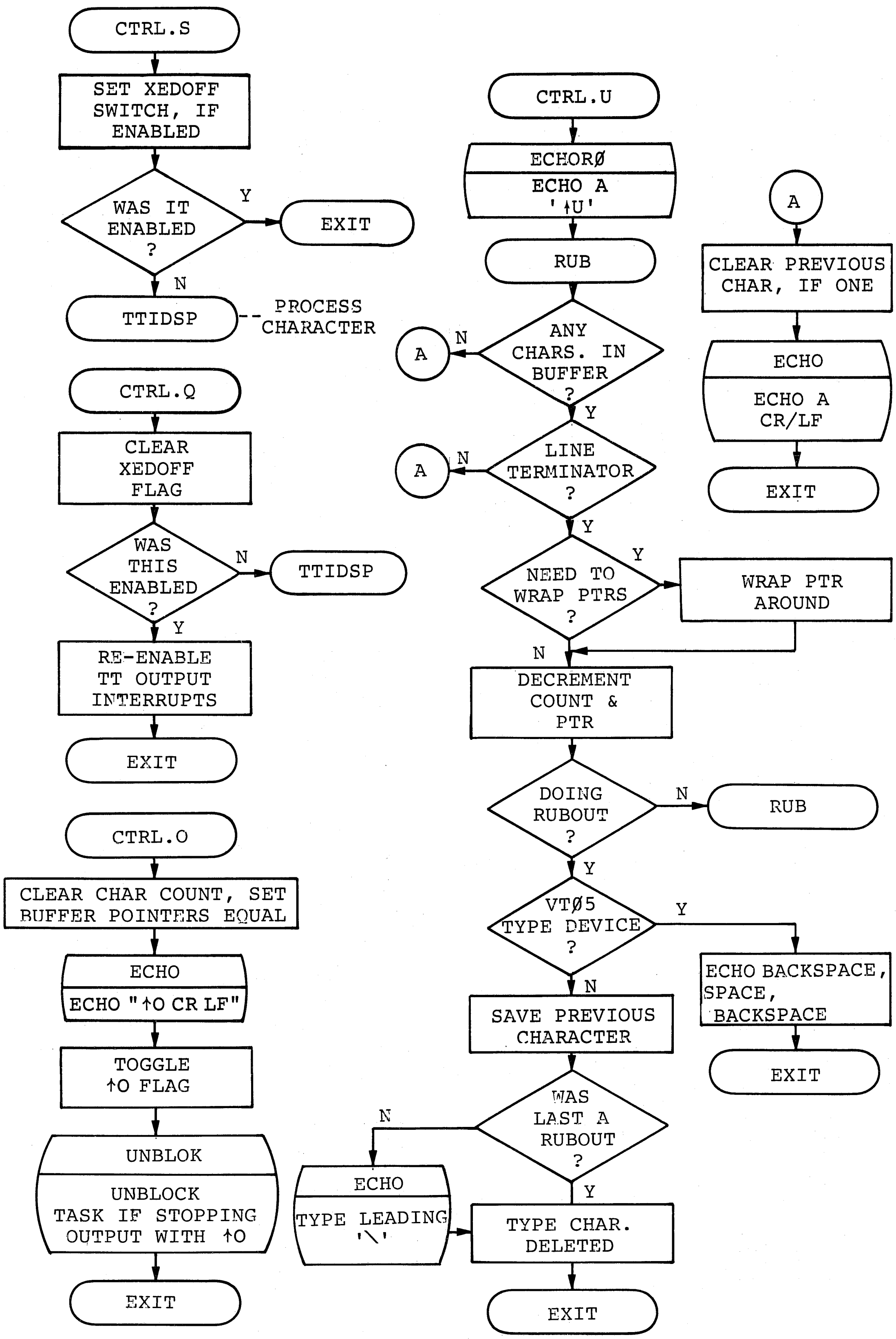
E.5.5 Console Terminal Interrupt Service

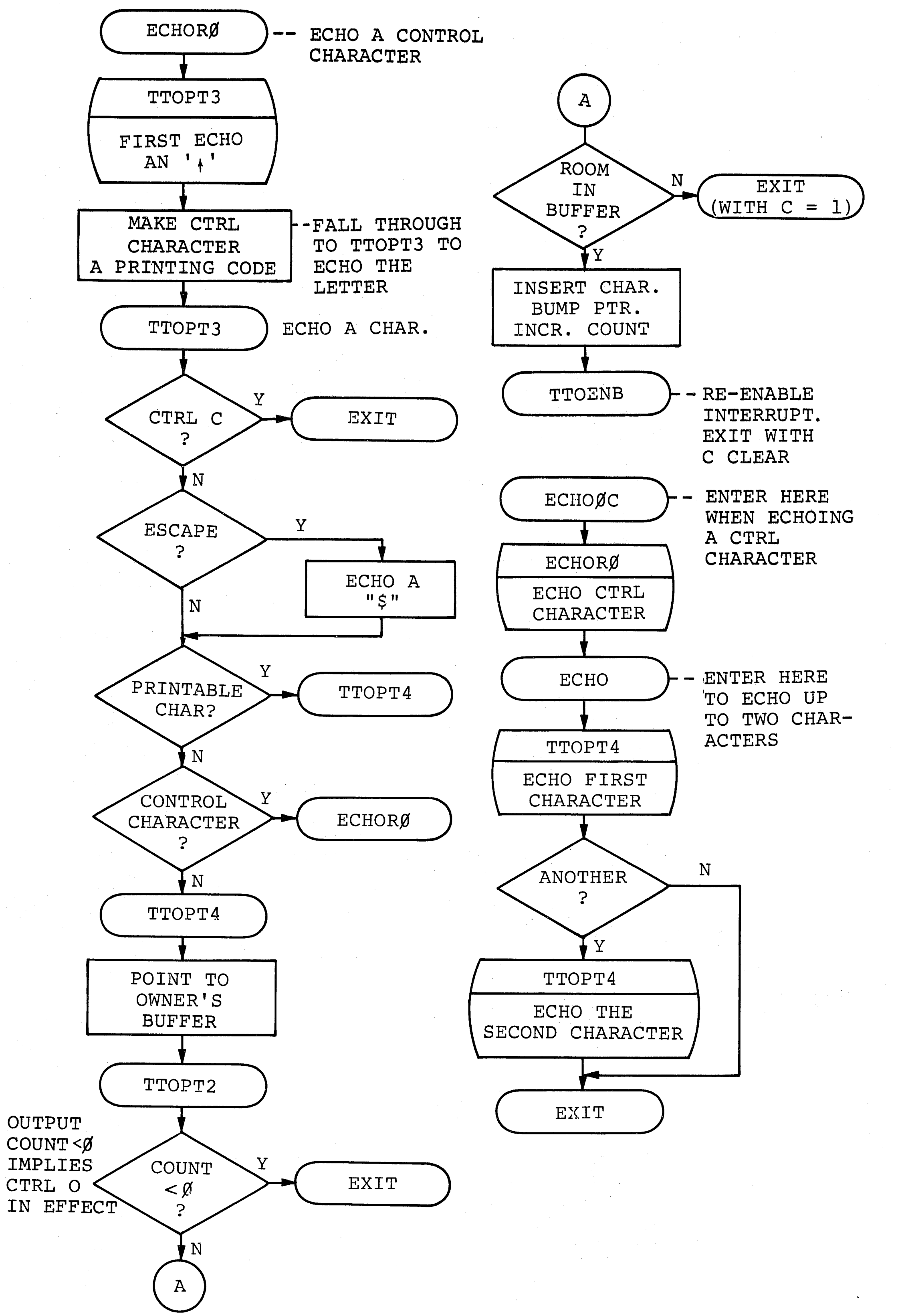
TT INPUT INTERRUPT ROUTINE



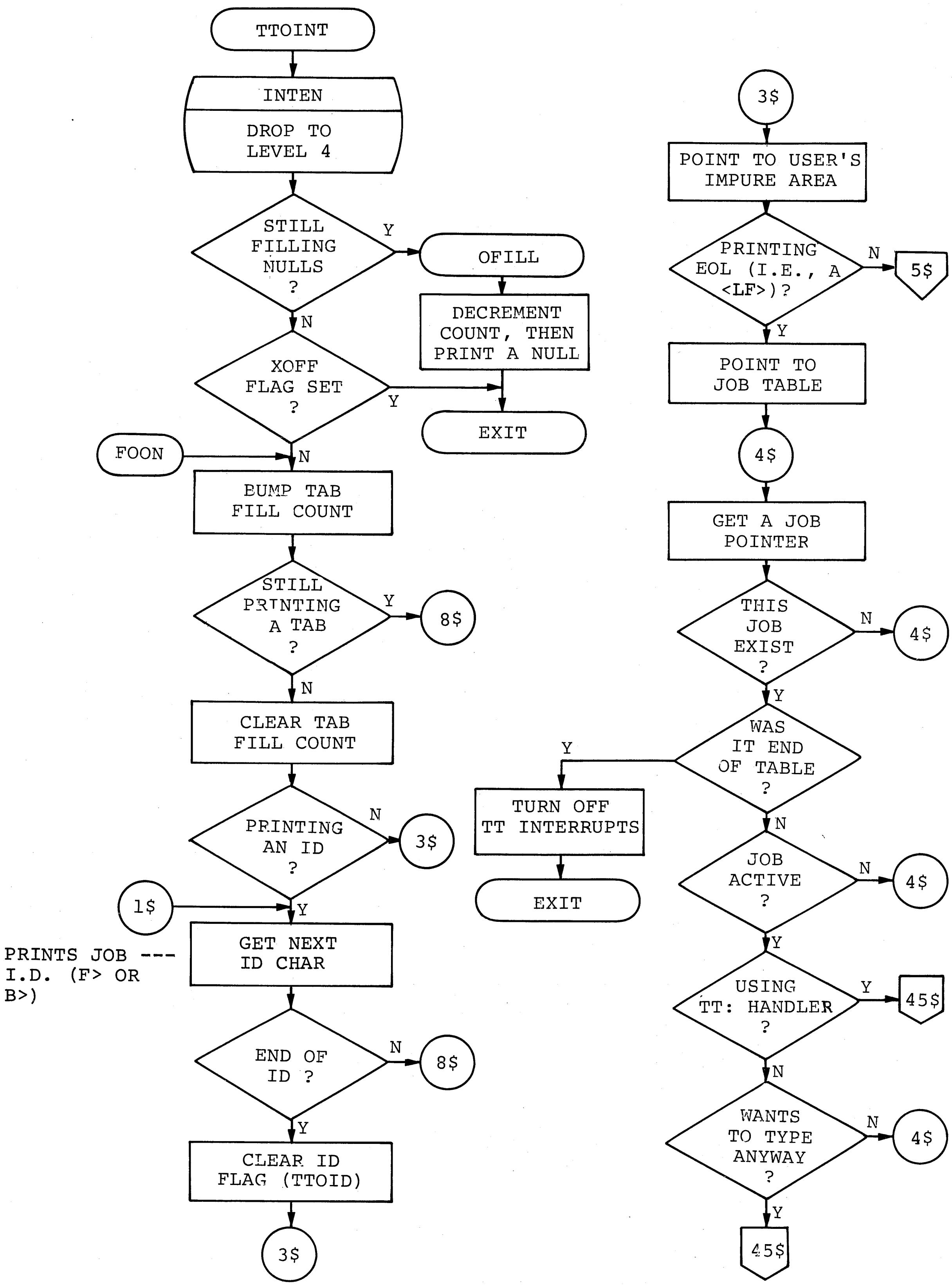

7$
SAVE FOR PREVIOUS CHAR. TEST
UNBLOK
UNBLOCK USER IF WAITING FOR TT
SPECIAL MODE ?
Y
10$
N
TTOPT3
ECHO THE CHARACTER
10$
CARRIAGE RETURN ?
N
EXIT
Y
CHANGE CODE TO LINE FEED
GENERATE A <LF> ON RECEIPT OF <CR>
TTINC3
GO INSERT IN BUFFER
25$
SET UP TO ECHO A BELL
ECHOES 'BELL' WHEN INPUT BUFFER IS FULL
SAVE CHAR. IN CASE OF ↑C ↑C
TTOPT4
CTRL.C
ECHOØC
ECHO "↑C<CR><LF>"
SECOND CTRL/C ?
N
TTINC3
Y
REQUEST TASK ABORT
EXIT
$RQABT
CTRL.B
GIVE INPUT OWNERSHIP TO BG
BG OWNS OUTPUT ?
Y
N
GIVE OUTPUT OWNERSHIP TO BG
SET UP TO PRINT OWNERSHIP ID ("B>")
TTOENB
TTOENB
ENABLE PRINTER INTERRUPT
EXIT
CTRL.F
ANY FG JOB ?
N
4$
Y
ACTIVE ?
N
4$
ECHO
ECHO "F?"
EXIT
Y
GIVE INPUT OWNERSHIP TO FG
FG OWNS OUTPUT ?
Y
EXIT
N
GIVE OUTPUT OWNERSHIP TO FG
SET TO PRINT ID ("F >")
EXIT

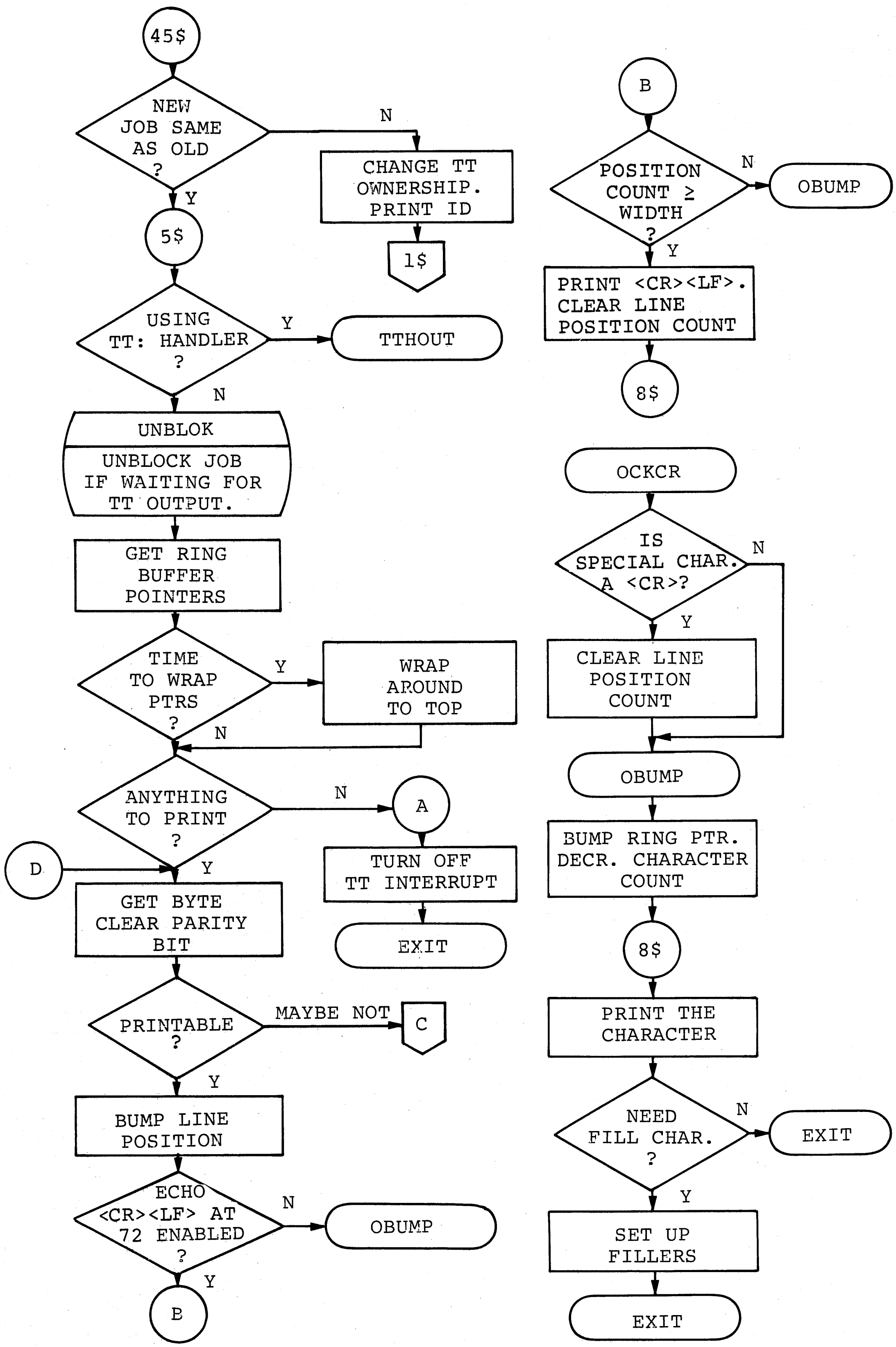
TT INPUT INTERRUPT ROUTINE (CONT.)



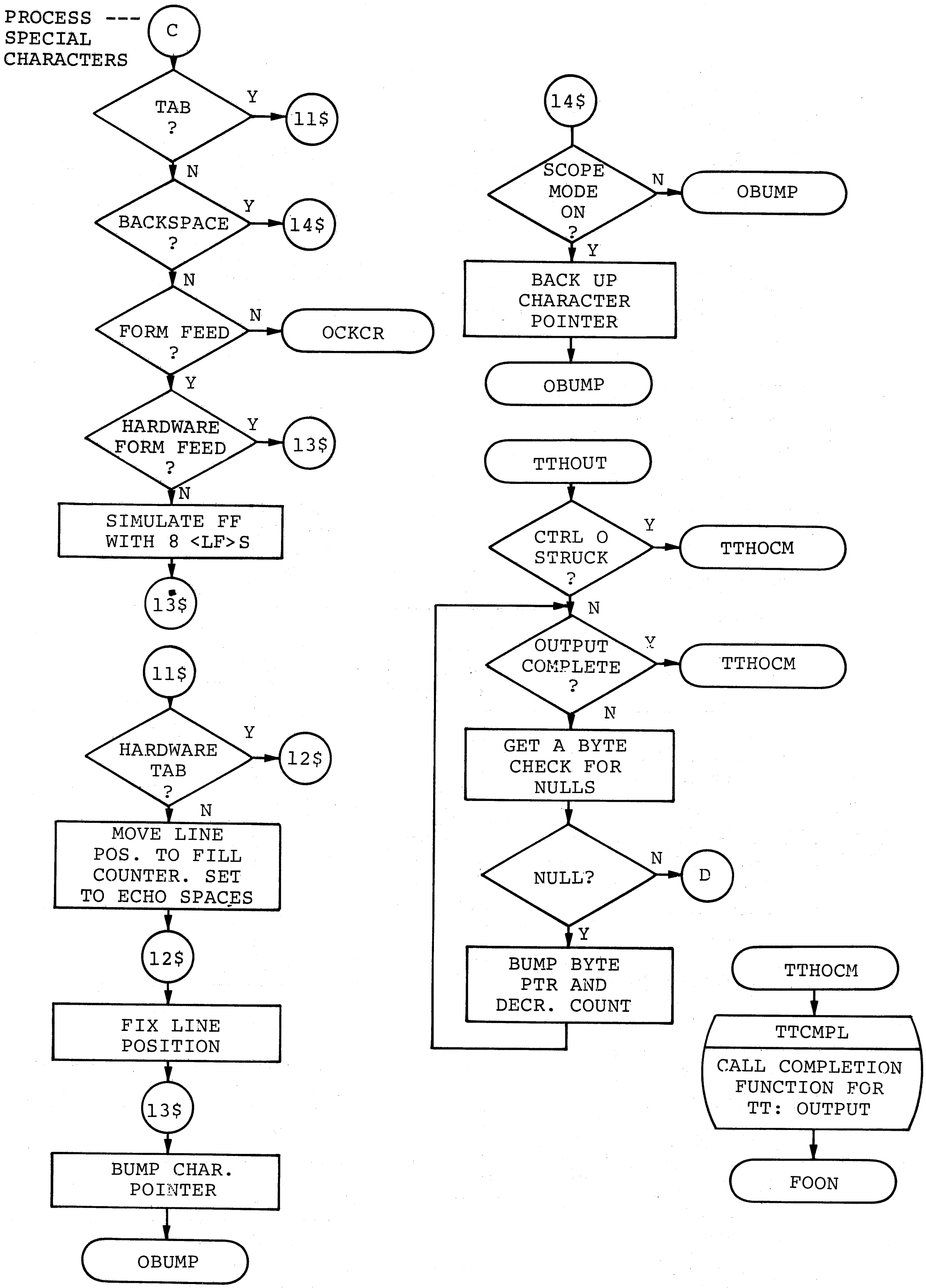

ECHORØ
-- ECHO A CONTROL CHARACTER
TTOPT3
FIRST ECHO AN '↑'
MAKE CTRL CHARACTER A PRINTING CODE
--FALL THROUGH TO TTOPT3 TO ECHO THE LETTER
TTOPT3
ECHO A CHAR.
CTRL C ?
Y
EXIT
N
ESCAPE ?
Y
ECHO A "$"
N
PRINTABLE CHAR?
Y
TTOPT4
N
CONTROL CHARACTER ?
Y
ECHORØ
N
TTOPT4
POINT TO OWNER'S BUFFER
TTOPT2
OUTPUT COUNT<Ø IMPLIES CTRL O IN EFFECT
COUNT <Ø ?
Y
EXIT
N
A
A
ROOM IN BUFFER ?
N
EXIT (WITH C = 1)
Y
INSERT CHAR. BUMP PTR. INCR. COUNT
TTOENB
-- RE-ENABLE INTERRUPT. EXIT WITH C CLEAR
ECHOØC
-- ENTER HERE WHEN ECHOING A CTRL CHARACTER
ECHORØ
ECHO CTRL CHARACTER
ECHO
-- ENTER HERE TO ECHO UP TO TWO CHAR-ACTERS
TTOPT4
ECHO FIRST CHARACTER
ANOTHER ?
N
Y
TTOPT4
ECHO THE SECOND CHARACTER
EXIT

TT OUTPUT INTERRUPT ROUTINE



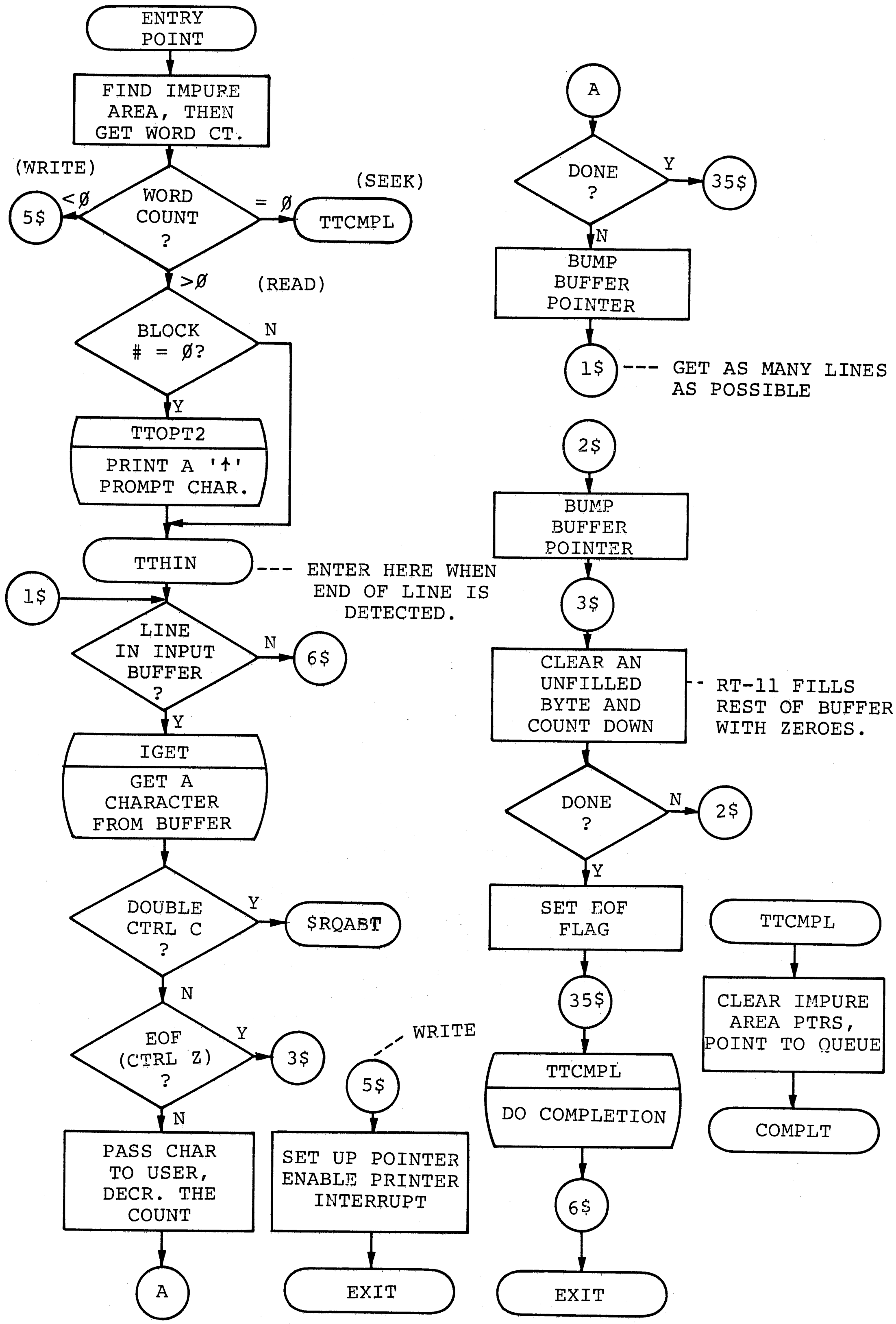

45$
NEW JOB SAME AS OLD ?
N
CHANGE TT OWNERSHIP. PRINT ID
1$
Y
5$
USING TT: HANDLER ?
Y
TTHOUT
N
UNBLOK
UNBLOCK JOB IF WAITING FOR TT OUTPUT.
GET RING BUFFER POINTERS
TIME TO WRAP PTRS ?
Y
WRAP AROUND TO TOP
N
ANYTHING TO PRINT ?
N
A
TURN OFF TT INTERRUPT
EXIT
D
Y
GET BYTE CLEAR PARITY BIT
PRINTABLE ?
MAYBE NOT
C
Y
BUMP LINE POSITION
ECHO <CR><LF> AT 72 ENABLED ?
N
OBUMP
Y
B
B
POSITION COUNT ≥ WIDTH ?
N
OBUMP
Y
PRINT <CR><LF>. CLEAR LINE POSITION COUNT
8$
OCKCR
IS SPECIAL CHAR. A <CR>?
N
Y
CLEAR LINE POSITION COUNT
OBUMP
BUMP RING PTR. DECR. CHARACTER COUNT
8$
PRINT THE CHARACTER
NEED FILL CHAR. ?
N
EXIT
Y
SET UP FILLERS
EXIT

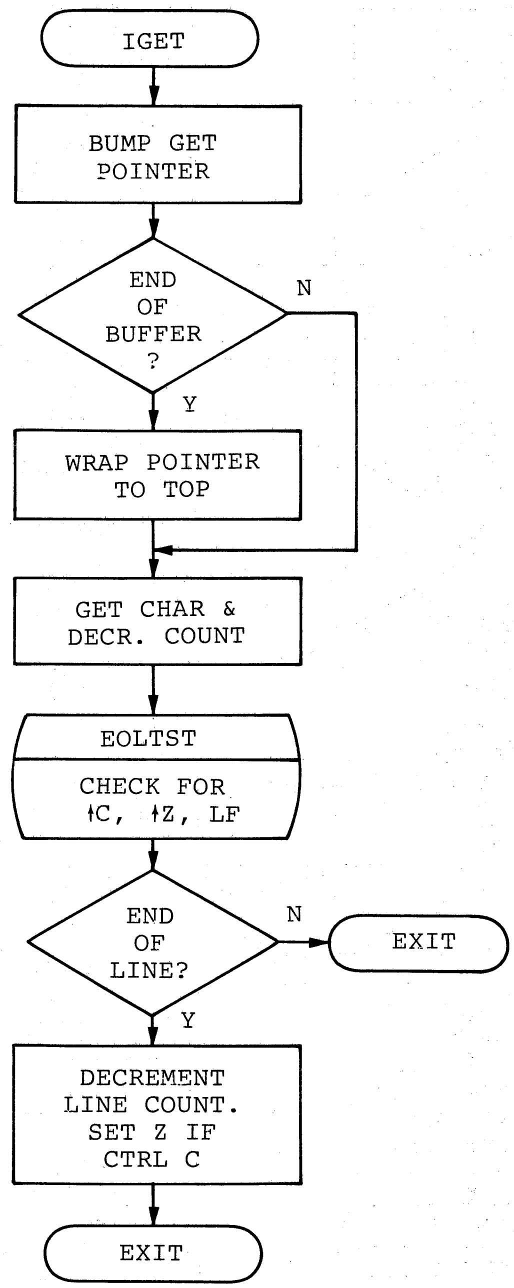
TT OUTPUT INTERRUPT ROUTINE (CONT.)



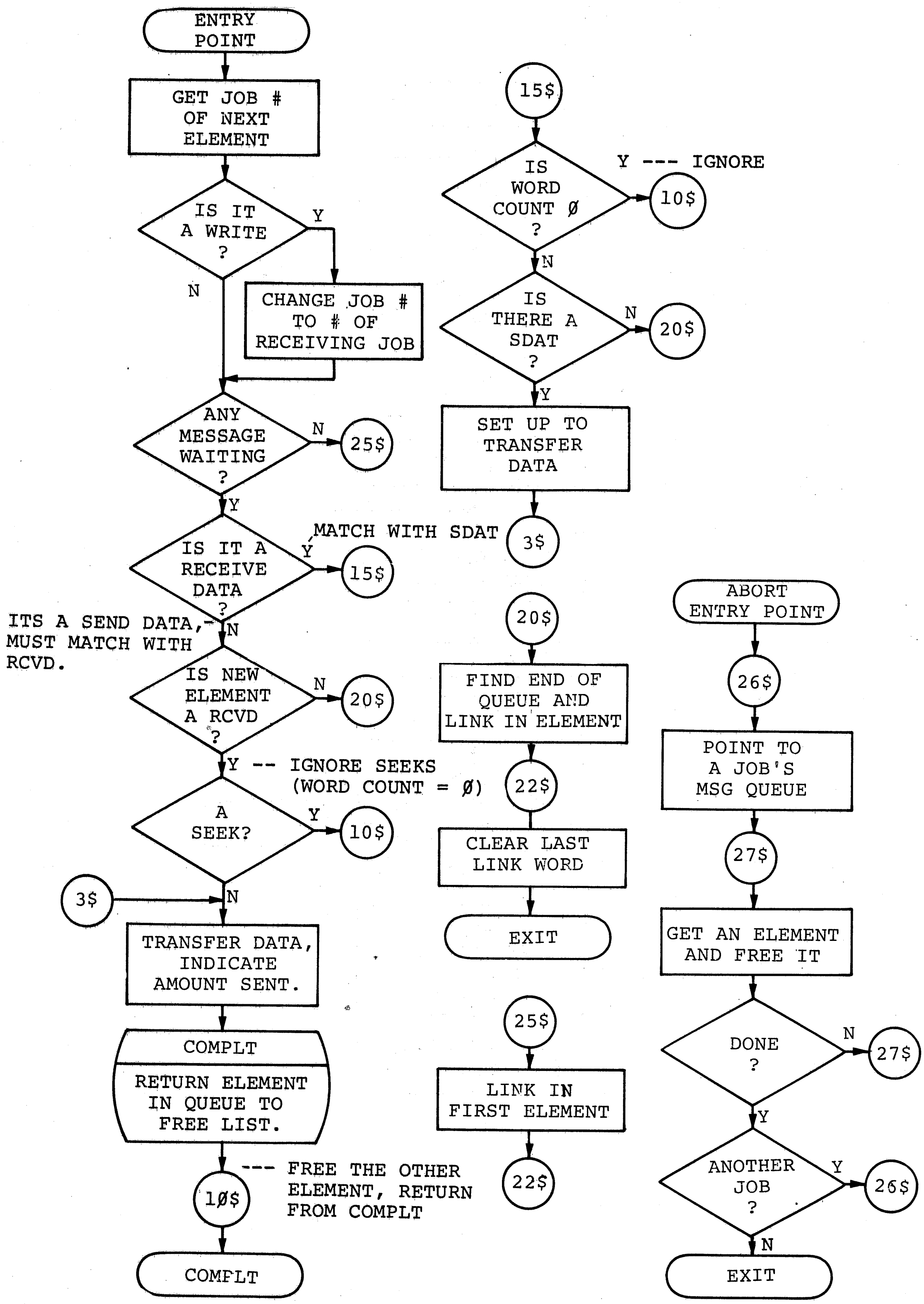
E.5.6 Resident Device Handlers (TT, Message)

TT: RESIDENT HANDLER




IGET
BUMP GET POINTER
END OF BUFFER ?
N
Y
WRAP POINTER TO TOP
GET CHAR & DECR. COUNT
EOLTST
CHECK FOR ↑C, ↑Z, LF
END OF LINE?
N
EXIT
Y
DECREMENT LINE COUNT. SET Z IF CTRL C
EXIT

MESSAGE HANDLER



ENTRY POINT INDEX

Page numbers marked by an asterisk indicate the flowchart of the entry point.

INDEX

RT-11 SOFTWARE
SUPPORT MANUAL
DEC-11-ORPGA-B-D, DN1

Please cut along this line.

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 ______________

If you require a written reply, please check here. ☐

-- 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

digital

digital equipment corporation

Printed in U.S.A.