digital

software

RTS/8
User's Manual

Order No. DEC-08-ORTMA-C-D

RTS/8
User's Manual

Order No. DEC-08-ORTMA-C-D

Version 2B

digital equipment corporation · maynard, massachusetts

First Printing, June 1974
Revised: September 1975
February 1977

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

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

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

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

The following are trademarks of Digital Equipment Corporation:

DIGITAL	DECsystem-10	MASSBUS
DEC	DECtape	OMNIBUS
PDP	DIBOL	OS/8
DECUS	EDUSYSTEM	PHA
UNIBUS	FLIP CHIP	RSTS
COMPUTER LABS	FOCAL	RSX
COMTEX	INDAC	TYPESET-8
DDT	LAB-8	TYPESET-10
DECCOMM	DECsystem-20	TYPESET-11
RTS/8		

CONTENTS

CONTENTS (Cont.)

CONTENTS (Cont.)

FIGURES

TABLES

PREFACE

This manual describes the PDP-8 Real-Time Operating System (RTS/8). Knowledge of PDP-8 assembly language programming (PAL8) is essential for a complete understanding of this manual. In addition, the user should be familiar with real-time systems in general and with the operation and use of the development system for the PDP-8, OS/8. The information in Chapter 9, "Advanced RTS/8 Programming Techniques" is for the experienced RTS/8 user. It should be read after the user has gained familiarity with RTS/8.

This version of the manual has been enlarged and expanded to incorporate several new RTS/8 features. The Major features include KL8-A Support, PDP-8/A Null Task, the EXIT Task, and two new Executive Requests. Other features are the nonresident implementation of the MCR, UDC/ICS support, an OS8COM facility that allows the OS/8 system to talk to an RTS/8 task, and control files that allow the user to efficiently make multiple task copies. RTS/8 flowcharts have been added to show system operation.

The following PDP-8 handbooks will be helpful for review and reference:

INTRODUCTION TO PROGRAMMING (DEC-08-XINPA-A-D)
SMALL COMPUTER HANDBOOK (90P45)
OS/8 HANDBOOK (DEC-S8-OSHBA-A-D)
UDC8 UNIVERSAL DIGITAL CONTROL SUBSYSTEM MAINTENANCE MANUAL (46H745)
PDP-8A MINICOMPUTER HANDBOOK (EB0621976)
ICS8 INDUSTRIAL CONTROL SUBSYSTEM MAINTENANCE MANUAL (EKOICS8MM)

CHAPTER 1

INTRODUCTION

1.1 RTS/8 DESCRIPTION

RTS/8 is a compact real-time system designed for the PDP-8 family of processors (except the PDP-8/S). This system allows up to 63 tasks to run concurrently and compete for resources on a fixed priority basis. It can be used for a wide range of applications in which a number of processes must be monitored and controlled. As with other real-time systems, RTS/8 responds to physical or conceptual events to permit the timely execution and scheduling of tasks.

The RTS/8 Executive controls execution and interaction among all tasks. The Executive decides which tasks should run (based on the priorities of the runnable tasks), and services the tasks by means of Executive Requests (ERs).

A task is the basic program unit within RTS/8. RTS/8 system tasks (DEC-supplied) and their file names are listed in Table 1-1. The system supports both resident and nonresident tasks. A resident task resides permanently in memory; a nonresident task is one in which the major portion of the task resides on a mass storage device and is loaded into memory only when that task becomes executable. Using nonresident tasks permits portions of several tasks to share the same areas of memory, providing economical use of memory.

RTS/8 includes system tasks that control most standard DIGITAL I/O devices. A full complement of peripherals is supported, including RK8 and RK8-E moving-head disks, DF32 and RF08 fixed-head disks, TC08 DECtape, RX8 floppy disk, LINCtape, DEC cassette, and LE8 and LS8E line printers (RTS/8 does not support TD8E DECtape). The Monitor Console Routine (MCR) task provides an interface between the user at the console terminal and the system. The MCR provides the user with a series of commands to control, inspect, and, to some extent, debug the system. The MCR commands are straightforward and easy to use. They allow the user to schedule and execute tasks at specified intervals, suspend task execution, and print system status reports.

A system task also is provided that allows a single copy of the OS/8 operating system to run in the background, creating a real-time foreground-OS/8 background system. With OS/8 in the background, the user has the facilities for program assembly, debugging, and editing. The minimum RTS/8 hardware configuration required for a foreground only system is a PDP-8 family processor, 4K words of memory, and a console terminal capable of papertape input. A system capable of running a real-time foreground and OS/8 in the background requires a PDP-8 family processor with KM8-A, KM8-E or TSS-8 Time Sharing options, a mass storage device (such as an RK8E cartridge disk or RX8 floppy disk), and two terminals (one must be dedicated to OS/8 system execution).

RTS/8 tasks are created by editing the RTS/8 master parameter file to produce a parameter file that describes the user's particular system. Task source files are then assembled with the edited parameter file using the PAL8 assembler. The assembler can run either under OS/8, or the OS/8 support system under RTS/8. Then using ABSLDR under OS/8, all task binaries are joined into a complete RTS/8 system.

Table 1-1
RTS/8 System Tasks

Task Name	File Name	Task Function
-	PARAM.PA	System parameter file with equates blank. Appropriate values should be inserted to create specific parameter files.
	RTS8.PA	RTS/8 Executive
MCR	MCR.PA	Monitor Console Routine
null task	MCR.PA	Null task
OS8	OS8SUP.PA	OS/8 Support Task
OS8F	OS8SUP.PA	OS/8 File Support Task
PWRF	PWRF.PA	Power Fail Task
CLOCK	CLOCK.PA	Clock Handler Task
TTY	TTY.PA	Terminal Driver Task
LPT	LPT.PA	Line Printer Driver Task
DTA	DTA.PA	TC08 DECtape Driver Task
RK8	RK8.PA	RK8 Disk Driver Task
RK8	RK8E.PA	RK8E Disk Driver Task
RF08/DF32	RF08.PA	RF08/DF32 Fixed-Head Disk Driver Task
CSA	CSA.PA	Cassette Driver Task
CSAF	CSAF.PA	Cassette File Support Task
UDC/ICS	UDCICS.PA	Universal Digital Controller/Industrial Controller Subsystem Handler Task
RX8A	RX01RT.PA	Floppy Disk Handler (1st controller)
RX8B	RX01RT.PA	Floppy Disk Handler (2nd controller)
RX8C	RX01RT.PA	Floppy Disk Handler (3rd controller)
RX8D	RX01RT.PA	Floppy Disk Handler (4th controller)
LTA	LTA.PA	LINCtape Driver Task
SWAPPER	SWAP.PA	Nonresident Task swapper
NULL8A	NULL8A.PA	Null Task for PDP-8A
EXIT	EXIT.PA	Exit Task

1.2 REAL-TIME SYSTEM OPERATION

A multiprogramming system is a software framework that allows available resources (such as memory, CPU time, and peripheral devices) to be shared by several tasks. Basically, a task is a portion of machine code that performs a specific function; a task is defined by convention since it overlaps with the definitions of "program" and "subroutine."

Multiprogramming allows many tasks to be in some state of execution simultaneously. If a task cannot use available central processor time because it is waiting for completion of an I/O operation (or is blocked by some other condition), the central processor can be switched to another task to use the available time, thus increasing system efficiency.

Most real-time systems are required to serve a group of tasks that run at varying times or frequencies and alternate between being compute bound or I/O bound. If machine resources are to be efficiently used, these tasks cannot be run in series since the central processor will be poorly utilized during the periods the tasks are I/O bound. In addition, most real-time tasks are essentially time-critical and should not wait for a slow, less important I/O or compute bound task to finish before being executed. Thus, multiprogramming and a priority scheme for scheduling the central processor provide maximum resource utilization.

Thus, a real-time system is a multiprogramming system that must, in addition to the multiprogramming features, respond quickly to critical internal or external events. The real-time system is required to suspend the operation of a less important task and start the task that deals with the critical event.

A priority scheme establishes the relative importance of various tasks in the system. This allows a less important task to be interrupted to permit execution of a critical real-time task. For RTS/8, a fixed priority scheme was chosen because such a scheme is simple and reliable, and requires low scheduling overhead.

CHAPTER 2

USING TASKS UNDER RTS/8

2.1 THE STRUCTURE OF RTS/8 TASKS

The RTS/8 Executive is the controlling program in an RTS/8 System. The Executive decides which task should run based on the priorities of the runnable tasks (those tasks not waiting for completion of any I/O or other events). It also provides services to the tasks by means of Executive Requests. Executive Requests are discussed in Chapter 3.

The user assigns unique task numbers to each task in an RTS/8 System. He can assign up to 63 (77 octal) task numbers, and must account for system tasks within the total number of tasks. A task number, once assigned, cannot change during the execution of the program, since RTS/8 uses a fixed priority system. Task numbers serve the following purposes:

1. The task number is used by the RTS/8 Executive as an index to various system tables which contain information about each task.

2. The task number is used by other tasks in the system to reference a particular task when performing certain Executive Requests (such as sending a message).

3. The task number determines the task's priority - the lower the task number, the higher the priority of the task.

The Executive uses five internal tables to maintain information about the tasks in the system. A brief description of these tables is given in Section 2.3.

2.2 CONCEPTS OF TASK COMMUNICATION

RTS/8 is event driven, i.e., the highest priority runnable task executes continuously until it is completed or some event or condition in the system causes it to be suspended. Another change or condition can reactivate the task. Tasks can be self-starting if assembled to run at system startup, started by another task, or started by the user at the terminal console using the MCR task. RTS/8 performs two main types of task communication, as follows:

1. Task synchronization through the use of Event Flags

2. Intertask messages

2.2.1 Task Synchronization Through the Use of Event Flags

Whenever two processes occur independently, one process may need to wait until the execution of the other is finished. This can be illustrated by using the PDP-8 terminal interface as an example. The PDP-8, when ready to generate the first character, alerts the terminal by issuing a Load Teleprinter Sequence (TLS) instruction. The PDP-8 must now wait in a TSF; JMP.-1 loop if it wishes to do further I/O immediately. It cannot proceed with the next character until the terminal raises its ready flag to signal that it is finished printing the first character. When the flag is raised, the PDP-8 then exits from the wait loop and proceeds to load the next character.

Similarly, RTS/8 provides Event Flags as a signalling mechanism to synchronize tasks with each other. An Event Flag is a user-chosen location that contains the status of an event. The events are either 1) physical processes such as a clock ticking or a valve closing, or 2) conceptual occurrences such as a certain string of characters typed by the operator or scheduling a task for execution. Like device flags, an Event Flag can signify either a busy or a completed state, defined as PENDING (>0) and FINISHED (=0), respectively. Thus, a task can direct another task via a message to perform a specified action, at which time it sets a mutually-agreed upon Event Flag to the PENDING value. When the second task has completed the specified action it sets the Event Flag to the FINISHED value; this is known as Posting the Event Flag. As a simple example, if the first task has been waiting for the Event Flag with the instructions:

```
TAD Event Flag  /LOAD EVENT FLAG IN AC
SZA CLA         /IF EVENT FINISHED, SKIP
JMP .-2         /KEEP TRYING
```

then the Posting of the Event Flag will cause the first task to exit from its loop, continuing on with the knowledge that the second task has completed its processing.

Since this loop ties up the PDP-8 processor, Event Flags under RTS/8 have an additional state, WAITING (<0). Using the example just cited, the addition of the WAITING state now allows the first task to tell the RTS/8 Executive that it wants to WAIT until the Event Flag signifying the status of the second task is FINISHED. The monitor saves the contents of the waiting task's PC and sets the Event Flag Wait bit in its Task Flags Table Word. The Event Flag is set to the WAITING state. The WAITING state for an Event Flag is octal 4000 plus the waiting task's Task Number. When the Event Flag is POSTed via an RTS/8 Executive Request call, the task WAITING for it is automatically taken out of Event Flag Wait by RTS/8. (If no other blocking bits are set, the waiting task is again runnable, and will resume execution when higher priority tasks are blocked.) WAITE is the mnemonic for the RTS/8 Executive Request that Waits for an Event Flag.

This code would look as follows:

```
CAL
WAITE
event flag
```

All Executive Requests are described fully in Chapter 3. A summary of Event Flag states is shown in Table 2-1.

Table 2-1
Summary of Event Flag States

Event Flag State	Value
FINISHED (posted)	0
PENDING	>0
WAITING	<0 (4000 + Task No.)

2.2.2 Intertask Messages

Just as Event Flags under RTS/8 are analogous to hardware device flags, messages are analogous to data-sending hardware I/O instructions (for example, TLS). That is, messages start a task, providing the task is WAITING for a message, and at the same time, they pass information to the task. Messages are transmitted by Executive Requests.

An RTS/8 message consists of a three-word Message Header followed by any number of contiguous words of information to be exchanged between two tasks. The Message Header is used exclusively by RTS/8. The first word of the Message Header is the Event Flag for the message. When the message is sent, the RTS/8 Executive sets the Event Flag to the PENDING state. This signifies that the message has been sent but not yet completed. No action occurs to the Event Flag upon receipt of the message by the receiving task; however, RTS/8 requires that the receiver POSTs the Event Flag when it has performed the action specified (or implied) by the message. This posting serves two purposes:

1. It informs the task which sent the message (the "sender") that the requested action has been completed, and

2. It allows the message to be sent again (see Item 2 below).

If multiple messages are waiting to be received by a task, RTS/8 uses the second and third words of the Message Header to link these messages together (see Figure 2-1). The second word is a CDF (Change Data Field) instruction to the field of the next message. The third word is the address of the next message. A second word of 0 signifies that this is the last message. If the receiving task is not actively waiting for a message, the message is placed on the receiving task's Input Message Queue. Messages are then queued in order of decreasing priority of the sender (increasing task number). Messages sent from the same task are queued in the order in which they were issued.

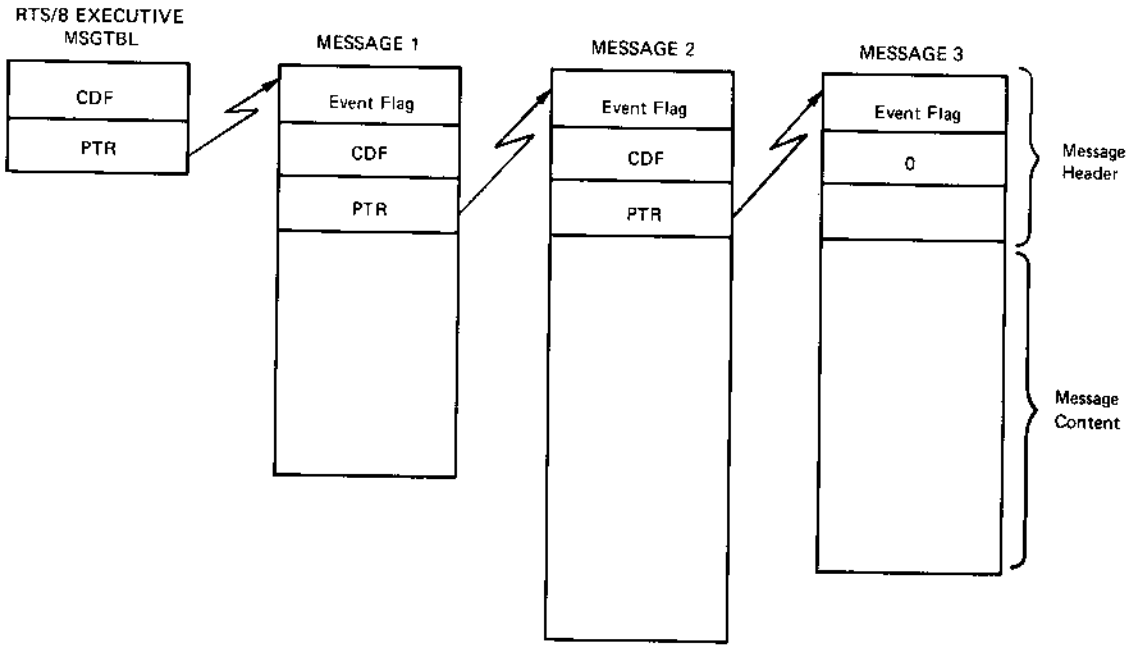


Figure 2-1 Message Format and Linking of Messages

The rest of the message can contain any desired information. Sending a message does not physically move the message information to the receiving task, but provides the receiver with the field and address of the first data word of the message.

It should be noted that the information in a message is not copied into the receiver's area. This has the following implications:

1. Data in a message should not be modified by the sender while the Event Flag for the message is PENDING.

2. The same message cannot be sent a second time before its Event Flag is FINISHED the first time. RTS/8 enforces this by checking the message Event Flag on a SEND operation and putting the sender into Event Flag Wait if the message is still PENDING.

3. It is legal for the receiving task to store information in the body of the message. In this way, an "answer" to the message can be returned without the complications of sending a return message back to the sender. For example, when a task sends a message to the disk driver task requesting I/O, the driver places the error status of the completed operation in a specific word in the message to indicate whether an error occurred.

2.3 EXECUTIVE INTERNAL TASK TABLES

The Executive uses five internal tables to maintain information about the tasks in the system. A brief description of these tables is as follows:

Table	Description
Task State Table (TSTABL)	Contains information such as the contents of the task's PC, Link and AC at the time the task stopped running.
Task Flags Table (TFTABL)	Contains information about why the task is not running, i.e., it indicates for what conditions (blocking or wait bits) the task is waiting.
Task Input Message Queue Header Table (MSGTBL)	Contains messages that have been sent to this task. This table is referred to simply as the Message Table.
Residency Table (RESTBL)	Used for nonresident tasks, this table specifies where the task is to reside in memory, and where it resides on the swap device.
Partition Table (PARTBL)	Used for nonresident tasks, this table contains information about each memory partition, such as length and location of the partition. Memory partitions are shared by nonresident tasks.

The user does not need to know the format of these tables to use RTS/8. However, a detailed explanation of these tables is given in Appendix E.

CHAPTER 3

RTS/8 EXECUTIVE REQUESTS

3.1 COMMUNICATION WITH THE RTS/8 EXECUTIVE

RTS/8 tasks communicate with the RTS/8 Executive via Executive Requests (ERs). RTS/8 uses locations 20-27 in every field as a communication region for ERs to facilitate Executive Requests across field boundaries. The Executive can be called in any field via a JMS 20 instruction (designated symbolically as CAL). The Data Field (DF) does not have to be any specific value when the CAL is given, since the code in location 20 sets the DF to the Instruction Field (IF), sets the IF to 0 and jumps to the RTS/8 Executive.

A summary of Executive Requests is given in Table 3-1. Most of the Executive Requests are explained in detail in this chapter. The RESCHD, WAITX and DERAIL Executive Requests are described in Chapter 9.

The RTS/8 Executive will not honor any request to switch tasks arising from an interrupt if the interrupted task's Program Counter (PC) was less than 100(octal). This protects the RTS/8 Executive's entry point (location 20 in each field) from being destroyed. User tasks must be written so that instructions are never executed below 100 in any field.

All ER's except DERAIL and SKPINS can relinquish processor control to higher-priority tasks as a result of their action.

3.2 ERs USED TO COMMUNICATE BETWEEN TASKS

The five ERs associated with the Intertask Messages and the Event Flags are SEND, WAITE, SENDW, RECEIVE and POST. An example of their use is shown in Section 3.2.6. In addition, a sixth ER called WAITX is described in Chapter 9, Advanced Programming Techniques (Section 9.2.2).

Table 3-1
Summary of Executive Requests

Code	Symbolic Name	Description	Section Reference
0	SEND	Send a message to a task	3.2.1
1	RECEIVE	Look for and/or receive a message from a task	3.2.4
2	WAITE	Wait for an Event Flag to be posted	3.2.2
3	RUN	Run a task	3.3.4
4	SUSPND	Suspend execution of a task	3.3.3
5	POST	Post an Event Flag	3.2.5
6	SKPINS	Insert code into interrupt skip chain	3.4
7	DERAIL	Derail or force a task's execution to a new address	9.2.3
10	BLKARG	Block a task from running for a specific reason	3.3.1
11	SENDW	Send a message and wait for it to be received	3.2.3
12	UNBARG	Remove a reason that a task is blocked from running	3.3.2
13	RESCHD	Force the RTS/8 Scheduler to run	9.1.2
14	WAITX	Wait for a particular Event Flag to be posted	9.2.2

3.2.1 SEND - Send Message

Format:
```
CAL
SEND
TSKNUM
MESSAG
```

The SEND ER sends the message located at MESSAG in the field of the CAL instruction to the task whose number is TSKNUM. If the receiving task has a higher priority than the sender and is waiting for a message, the sender is temporarily suspended and the receiver runs. In this case, the sender is not put into any WAIT state once the message is sent. However, if the Event Flag in location MESSAG is PENDING (nonzero), meaning the message is still busy from a previous SEND, the sender will be put into Event Flag Wait on location MESSAG, and only when the Event Flag becomes FINISHED (zero) will this SEND be performed. Care should be taken that a message is sent from only one task as only the last request to send a busy message is remembered; the first task can go to sleep in Event Wait permanently.

3.2.2 WAITE - Wait on Event Flag

```
Format:  CAL
         WAITE
         EFLG
```

The WAITE ER checks the status of location EFLG and if it is FINISHED, returns control to the caller. If EFLG is PENDING, its state is changed to WAITING and the calling task is put into Event Flag Wait. When location EFLG is POSTed by another task or interrupt routine, the calling task becomes runnable again. The Event Flag must be initialized (set to 1) before use in most cases, particularly when a task is initiating an event to be completed by another task. The waiting task must reset the Event Flag before using it again in that the Event Flag does not reset itself.

NOTE

In advanced applications, the user may be waiting for multiple Event Flags (see Section 9.2.1 for description of WAITM). In this case the task will run whenever any one of the Event Flags is posted, and not necessarily the one specified in the WAITE. To insure that a particular Event Flag is posted, use the WAITX ER described in Section 9.2.2.

3.2.3 SENDW - Send and Wait

```
Format:  CAL
         SENDW
         TSKNUM
         MESSAG
```

The SENDW ER is exactly equivalent to the sequence:

```
         CAL
         SEND      /SEND THE MESSAGE
         TSKNUM
         MESSAG
         CAL
         WAITE     /WAIT FOR RECEIVER TO ACKNOWLEDGE
         MESSAG
```

3.2.4 RECEIVE - Receive Message

```
Format:             TAD TSKNUM          /ONLY TO RESTRICT TO ONE
                    CAL                 /SENDING TASK
                    RECEIVE
           MADDR,   0                   /MESSAGE ADDRESS STORED
                                        /HERE;  CDF TO MESSAGE
                                        /FIELD IN AC ON RETURN
```

If the AC is zero when the RECEIVE ER is issued, the calling task's Input Message Queue is examined. If there are messages in the calling task's Input Message Queue, the first (i.e., highest-priority) message is dequeued and the address of its first data word is placed in MADDR. A CDF to the field of the message is stored in the AC.

If there are no messages, the task is placed in Message Wait until such time as a message is sent to this task. However, a task may first examine its Input Message Queue Header in field 0 to determine the state of the Input Message Queue.

If the AC is nonzero when the RECEIVE ER is issued, the calling task's Input Message Queue is searched for a message whose sender's Task Number matches the contents of bits 1-11 of the AC. If such a message is found, it is removed from the queue as specified above; if a message is not found, the issuing task is placed in Message Wait. This allows a message from only one given task to be received.

NOTE

The following information is useful to the advanced user. When a task is in MSGWT, after just having done a RECEIVE, its PC as stored in the TSTABL points back to the location containing the CAL. Thus, when a message comes in, the task re-executes the RECEIVE ER and accepts the message. This mechanism is normally transparent to the user. One implication is that no harm is caused by taking a task out of MSGWT because once the task starts up again, it will re-execute the RECEIVE ER, and go back into MSGWT.

Normally, if there are no messages in the Input Message Queue when a task performs a RECEIVE, the task is put into Message Wait. However, a 1 in bit 0 of the AC (i.e., the AC is negative) when the RECEIVE is issued indicates that the task is not willing to wait. Thus, with no messages in the Input Message Queue (or none sent by the task specified in bits 1-11 of the AC), the task will then continue to run (at CAL +3) with the AC equal to zero. The zero AC provides the means for the RTS/8 Executive to inform the task that there were no messages (of the desired type) pending.

3.2.5 POST - Post Event Flag

Format:

```
TAD EFPTR          /POINTER TO EVENT FLAG
CAL
POST
CDF EFFLD          /FIELD OF EVENT FLAG
```

The Event Flag pointed to by the AC, in the field specified by the CDF, is set to the FINISHED (zero) state. If its previous state was WAITING, the task that was waiting for it is cleared of its Event Flag Wait. This ER never sets the calling task in a WAIT state. If the task waiting for the Event Flag is of a higher priority than the calling task, the calling task is temporarily suspended while the other is run.

3.2.6 Example of ERs for Message and Event Flags

The following example illustrates the RTS/8 ERs dealing with Messages and Event Flags. Since I/O and interrupts under RTS/8 have not been discussed yet, this example is elementary. There is no advantage to keeping the functions of the two tasks separate, and the entire send/receive structure is being used here as an elaborate subroutine call. A description of the execution sequence follows the example.

Task A

```
A1          ALOOP,      CAL
                        SEND            /SEND TASK B MESSAGE 1
                        B
                        MES1
A2                      CAL
                        SEND            /SEND TASK B MESSAGE 2
                        B
                        MES2
A3                      CAL
                        WAITE           /WAIT FOR MESSAGE 1
                        MES1
A4                      JMP ALOOP       /LOOP
            MES1,       ZBLOCK 3        /MESSAGE 1
                        15              /RANDOM NUMBERS
                        37
                        23

            MES2,       ZBLOCK 3        /MESSAGE 2
                        -1              /RANDOM NUMBERS
                        4
```

Task B

```
B1          BLOOP,      CAL
                        RECEIVE             /GET A MESSAGE
            MADDR,      0
B2                      DCA EFCDF           /SAVE MESSAGE CDF FOR POST
B3                      TAD EFCDF
B4                      DCA .+1             /PUT CDF INLINE
B5                      HLT                 /CDF TO MESSAGE FIELD
B6                      TAD I MADDR         /GET 1ST DATA WORD OF
                                            /MESSAGE (DO NOTHING WITH IT)
B7                      CLA
B8                      STA CLL RTL         /-3 IN AC
B9                      TAD MADDR           /AC POINTS TO MESSAGE
                                            /EVENT FLAG
B10                     CAL
                        POST                /DECLARE MESSAGE RECEIVED
            EFCDF,      HLT                 /CDF TO MESSAGE FIELD HERE
B11                     JMP BLOOP           /LOOP
```

The flow of execution in this example depends on which of the two tasks has higher priority. Assuming that at some time both A and B become runnable and task A has higher priority, the sequence of execution is as follows:

Sequence	Reason For Execution
A1	Task A has higher priority than task B.
A2	Task A has higher priority than task B.
A3	Task A has higher priority than task B.
B1	Task A is now in Event Flag Wait since MES1 was PENDING; MES1 is now in WAITING state.

Sequence	Reason For Execution
B2-B10	Task A is still waiting; the RECEIVE at B1 received MES1
A4	The POST at B10 posted MES1 and "woke up" A, which has higher priority than B.
A1	A continues executing.
A2	A tries to send MES2 again; B has not yet processed it; MES2 is PENDING.
B11	Therefore, A is put into Event Flag Wait and B is resumed; MES2 is now WAITING.
B1-B10	B now RECEIVes and POSTs MES2.
A2	This brings A out of Event Flag Wait; the RTS/8 Executive has modified task A's program counter so that it will re-execute the offending SEND.
A3	A3 now waits for MES1 to be POSTed.

If task B has higher priority, the sequence of execution is:

Sequence	Reason For Execution
B1	Task B has higher priority than task A.
A1	Task B is placed in Message Wait since there are no messages in its input queue. Task A then sends MES1 to Task B.
B2-B10	Task A's message brings task B out of Message Wait; since B has higher priority, A is stopped and B runs.
B11	The POST at B10 sets MES1 to FINISHED but has no other effect.
B1	Now task B tries to get another message.
A2	There are no other messages, so task B is put in Message Wait and A is run.
B2-B11	Task A sends MES2 which "wakes up" B; B processes MES2 and
B1	returns for more,
A3	and is put in Message Wait. Since MES1 is FINISHED
A4	the WAITE at A3 has no effect and task A
A1	loops back to A1 and sends MES1 again.

3.3 ERs USED TO SET AND CLEAR TASK FLAGS

Several ERs allow a task to explicitly set and clear flags in the Task Flags Table entry of another task, and to set flags in its own table entry. These ERs are BLKARG, UNBARG, SUSPND and RUN.

3.3.1 BLKARG - Block Task for Specified Reason

```
Format:   TAD TASKNUM     /OR 0 IF SELF
          CAL
          BLKARG
          WAITBITS
```

TASKNUM contains the number of the task to be blocked (that is, not allowed to run). WAITBITS specifies one or more bits to be set in that task's Task Flags word. Assuming WAITBITS is nonzero, this will cause the specified task to become non-runnable. If TASKNUM contains zero, the issuing task will be blocked on the specified wait bits.

The TASKNUM=0 form of this ER is the only legal way to specify the issuing task as the task to be blocked; if TASKNUM is equal to the issuing task number, the action of this ER is undefined.

Example:

Task 14 is placed into User Wait by executing the following code.

```
TAD     (14
CAL
BLKARG
USERWT
```

Symbolic names for specifying the condition for blocking or unblocking a task in the WAITBITS word is given in Table 3-2.

Table 3-2
Symbolic Names for Specifying WAITBITS

Symbolic Name	Value	Meaning
NONRWT	4000	Nonresident Wait - This task cannot run because it is not in memory.
EFWT	2000	Event Flag Wait - This task is waiting for an Event Flag (which contains a WAITING value corresponding to this task) to be POSTed.
RUNWT	1000	Run Wait - This task is waiting for a RUN ER to be executed with its number in the AC, or for the operator to type "REQUEST task" to the Monitor Console Routine (see Chapter 5).
SWPWT	0400	Swap Wait - This task cannot run because it is in the process of being brought into memory.
EORMWT	0200	Event or Message Wait - This task is waiting for an Event Flag to be posted or a message to arrive, whichever happens first.
USERWT	0100	User Wait - This bit is reserved for use by user-written tasks. RTS/8 does not use this bit.
ENABWT	0040	Enable Wait - This task is waiting to be Enabled. Use of this bit is restricted to the Monitor Console Routine for the "ENABLE task" and "DISABLE task" commands (see Chapter 5).
MSGWT	0020	Message Wait - This task is waiting to be sent a message.
DNEWT	0001	Task does not exist. This bit should never be set or cleared by a user task.

3.3.2 UNBARG - Unblock Task for Specified Reason

```
Format:   TAD TASKNUM
          CAL
          UNBARG
          WAITBITS
```

TASKNUM contains the number of the task to unblock, and WAITBITS specifies one or more bits to be cleared in that task's Task Flags word. If the Task Flags word becomes zero as a result of this operation, the specified task becomes runnable; if the specified task has higher priority than the issuing task and becomes runnable, the issuing task is temporarily suspended while the higher-priority task runs.

This ER is a no-op (no operation) if issued with TASKNUM equal to the issuing task's number.

Example:

Task 14 is taken out of User Wait by executing the following code.

```
TAD     (14
CAL
UNBARG
USERWT
```

3.3.3 SUSPND - Suspend a Task's Execution

```
Format:   TAD TASKNUM /0 IF SELF
          CAL
          SUSPND
```

This SUSPND ER is identical in action to the following instructions:

```
TAD TASKNUM
CAL
BLKARG
RUNWT
```

3.3.4 RUN - Run a Task

```
Format:   TAD TASKNUM
          CAL
          RUN
```

This RUN ER is identical in action to the following instructions:

```
TAD TASKNUM
CAL
UNBARG
RUNWT
```

The SUSPND and RUN ERs exist because their function is performed often enough to warrant a shorthand version. An example that shows how they can be used in a task follows.

A data collection task is to print a report every 1000 data points without interrupting the data collection/reduction process. When executed, the Report Generation Task comes up running, so that the

first report occurs on the first data. In this simplified example, the data operated on by the report program may have been already updated for the next cycle before being reported. A full example would require a scheme such as double buffering to protect the data.

Data Control Task

```
DLOOP,    TAD (-1750      /1000 DECIMAL
          DCA COUNT
DATALP,   CAL
          WAITE
          DATAEF          /WAIT FOR DATA READY
          .               /CODE TO STORE DATA
          .               /POINT IN BUFFER
          .
          .               /GET A DATA POINT
          ISZ COUNT       /AND PROCESS IT
          JMP DATALP      /COUNT OFF 1000 POINTS
          TAD (REPORT
          CAL             /RUN REPORT TASK
          RUN
          JMP DLOOP       /KEEP COLLECTING
COUNT,    0
```

Report Generation Task

```
RLOOP,    CAL                   /AC=0, SUSPEND
          SUSPND                /UNTIL NEEDED
          JMS TITLE             /HAS BEEN RUN
          .
          .                     /PRINT REPORT
          .                     /WITH TITLE
          .
          .
          JMP RLOOP             /REPORT OVER-GO
                                /BACK AND WAIT
```

To eliminate interference with the data collection, REPORT should have a lower priority than DATA.

3.4 USING INTERRUPTS IN RTS/8

The RTS/8 Executive contains code to receive and dismiss hardware interrupts and to perform interrupt-initiated task switching, but it does not provide room for an interrupt skip chain. Instead, the skip chain is literally a chain and is built up dynamically at system startup time via the SKPINS ER. A description of the SKPINS ER is as follows.

```
Format:   CAL
          SKPINS
          MODULE
```

MODULE is the address (in the current field) of an interrupt processing module.

An interrupt processing module has the following format:

```
MODULE, 0                   /THIS WORD GETS A POINTER
                            /TO THE NEXT MODULE
        0                   /MODULE ENTERED HERE - CONTAINS
                            /CDF CIF TO NEXT MODULE FIELD
        SKDR                /SKIP ON DEVICE READY
        (SKP)               /(ONLY IF SKDR REALLY MEANS SKIP
                            /ON DEVICE NOT READY)
        JMP I MODULE        /NOT READY - GO TO NEXT MODULE IN
                            /CHAIN
        CDF CIF CUR         /THIS ONE IS MINE - SET DF AND IF
                            /CORRECTLY
        .                   /INTERRUPT PROCESSING
        .
        .
        CIF 0               /DISMISS THE INTERRUPT, MAYBE POST
        POSTDS              /AN EVENT FLAG DEPENDING UPON
                            /CONTENTS OF AC
```

See item 7 below for the definition of the POSTDS instruction.

Whenever a task executes a SKPINS ER, the interrupt chain is broken at the very end and the user's interrupt module is inserted. This is usually done by tasks at system start-up time only. The last interrupt module points to the interrupt dismiss routine as its "next module". In this way, RTS/8 tries to avoid superfluous interrupts. SKPINS always inserts at the end of the skip chain. This implies that the skips in the skip chain are ordered roughly by priority of the task which inserted them, since any SKPINS ERs in a task are usually executed as once-only code at system start-up time.

Once an interrupt module receives control (i.e., its I/O skip succeeds), there are several restrictions on its execution:

1. The interrupt module must clear the interrupt request.

2. The Data Field and Instruction Field are those of the next interrupt module; the user must correct this as described above before any indirect addressing or jumps are performed.

3. An interrupt module may not issue any RTS/8 ERs.

4. An interrupt module should not compute excessively when interrupts are off. Typical execution time should be under 75us. If considerably more computing than this is needed, a task should be scheduled to perform it by POSTing an Event Flag. A POSTDS instruction is used to wake up the task from Event Wait.

5. Interrupt modules must not turn interrupts on because the state of the interrupted task will be destroyed by a second interrupt.

6. On entry to the interrupt module, the contents of the AC, Link, and Data Field have already been saved, but not the contents of the Multiplier Quotient (MQ). Therefore, interrupt modules requiring the use of the MQ should save it, and then restore it before dismissing the interrupt.

7. Interrupt modules must dismiss the interrupt by setting the Instruction Field to 0 and issuing a POSTDS instruction. POSTDS is defined as a JMP I 24 instruction. An Event Flag may be POSTed when the interrupt is dismissed by setting the

Data Field to the field of the Event Flag and placing the location of the Event Flag in the AC prior to issuing the POSTDS. For example:

```
CDF CUR          /DF = THIS FIELD
TAD (EVFLG       /EVFLG MAY NOT BE AT LOCATION 0
CIF 0
POSTDS           /DISMISS INTERRUPT AND POST EVFLG
```

If an Event Flag is not going to be posted by the interrupt routine, the AC must be cleared prior to issuing the POSTDS instruction.

For example, an RTS/8 Paper Tape Punch handler task might contain the following sections of code:

In the initialization code (contained in a task that is runnable at system start-up time):

```
START,  CAL              /LINK THE PUNCH SKIP
        SKPINS           /INTO THE SKIP CHAIN
        PTPINT
GETREQ, CAL
        RECEIVE          /WAIT FOR MESSAGE
        .
        .
        .
```

As a character punch subroutine used by the main body of the task:

```
PUNCH,  0                /ENTER WITH CHAR IN AC
        DCA TEMP         /SAVE CHAR
        CAL              /WAIT UNTIL PUNCH READY
        WAITE
        PTPEF
                         /SET PUNCH EVENT FLAG
        ISZ PTPEF        /TO THE PENDING STATE
        TAD TEMP
        PLS              /PUNCH CHAR
        CLA
        JMP I PUNCH      /RETURN
        .
        .
        .
```

Interrupt skip chain code:

```
PTPINT, ZBLOCK 2         /USED TO CHAIN SKIPS
        PSF              /CHECK PUNCH FLAG
        JMP I PTPINT     /NOT READY
        CDF CIF CUR      /SET CORRECT DF, IF
        PCF              /CLEAR PUNCH FLAG
        TAD (PTPEF
        POSTDS           /DISMISS INTERRUPT,
                         /POSTING PTPEF
        .
        .
        .
PTPEF,  0                /PUNCH INITIALLY READY
TEMP,   0
```

RTS/8 does not provide a mechanism for removal of entries from the interrupt skip chain.

3.5 EXECUTIVE REQUEST WAIT STATES

A summary of wait states generated by Executive Requests is shown in Table 3-3.

Table 3-3
Summary of Wait States Incurred by Executive Requests

ER	Wait State	Condition	PC Suspended At
SEND	none	EFWT for SEND if message busy at 'CAL'	-
RECEIVE (No wait if AC=4000)	MSGWT	If no messages in Input Queue and AC positive	'CAL'
WAITE (No wait if EF 'done')	EFWT	If Event Flag not FINISHED	'CAL'+3
RUN	none	-	-
SUSPND	RUNWT	If task = self	'CAL'+2
POST	none	-	-
SKPINS	none	-	-
DERAIL	none	-	-
BLKARG	any (given by argument)	If task = self	'CAL'+3
SENDW	EFWT	If message free but Event Flag not FINISHED	'CAL'+3
	EFWT	If message busy	'CAL'
UNBARG	none	-	-
RESCHD	none	-	-
WAITX	EORMWT	If specified Event Flag not FINISHED	'CAL'
WAITM	any (given by argument)	-	'CAL'+3

NOTE: (a) 'CAL' denotes the address of the CAL instruction in the Executive Request.

(b) A message is said to be busy if its Event Flag has not yet been POSTED by its previous user.

CHAPTER 4

RTS/8 SYSTEM TASKS

The RTS/8 system includes system tasks that control most of the standard Digital PDP-8 I/O devices. Also included is one task that provides interactive system control from the console terminal and allows a single copy of the OS/8 monitor system to run in the background. Foreground tasks are protected from background tasks; however, the reverse is not true. The complete list of system tasks available in the RTS/8 system is as follows:

- Clock Handler - accepts requests in the form of RTS/8 messages to perform actions after a specified time has elapsed.

- Console and Non-console Terminal Handlers - handle a single terminal in either line or character mode.

- Line Printer Handler - supports an LS8, LS8E, LP8 or LV8 line printer.

- Mass Storage Handlers - Control the passing of information from these devices to and from memory for the RK08 and RK8-E moving-head disks, DF32 and RF08 fixed-head disks, and TC08 DECtape unit. Data is read and written in the standard RTS/8 block format (400 octal contiguous words).

- Floppy Disk Handler - provides support for the use of the RX8 floppy disk.

- LINCtape Handler - supports both OS/8 and DIAL-format LINCtapes.

- OS/8 Files Support Task - allows the user to look up, create, and delete files in OS/8 directories from a foreground task. This task, when used with the mass storage handlers, provides the capability to read or write OS/8 files on mass storage devices.

- OS/8 Support Task - supports the execution of an OS/8 operating system in the background.

- UDC/ICS Handler - enables the user to control the various types of UDC/ICS modules.

- Cassette Handler - allows the user to read or write data on a tape cassette.

- Cassette File Support Handler - allows the user to look up, enter, and delete files from a DECcassette in CAPS-8 format.

- Power Fail Task - when used with power fail hardware, it provides for an orderly shutdown when AC power is lost. Also, it allows a programmed restart when power returns.

- Exit Task - allows the user to perform special processing before making an exit from RTS/8.

- PDP-8A Null Task - allows the user to count in decimal on the LED display of the PDP-8A.

The sources of the system tasks are supplied with the RTS/8 system. The tasks referred to as "handlers" are completely message-driven, i.e., when idle they are in the Message Wait state. Other tasks send these handlers I/O request messages. When the handler completes the I/O operation, it POSTs the Event Flag associated with the request message and issues another RECEIVE ER.

4.1 CLOCK HANDLER

The Clock Handler Task can be assembled to handle any one of four hardware clocks. The user selects the clocks by setting the symbol CLKTYP in the parameter file to 0 for KD8-EA/DK8-EC, to 1 for KW12, to 2 for PDP-8A, or to 3 for DK8-EP. The Clock Handler accepts RTS/8 messages and inserts the entries into an internal clock queue. As the entries become due, they are removed from the queue, and the request is decoded and executed. The user fixes the length of the queue at assembly time by defining the symbol CLKQLN in the parameter file to the minimum number of entry slots. The default value for CLKQLN is 20.

The format of a clock message is:

```
CLKMSG,  ZBLOCK 3              /3 WORDS RESERVED FOR RTS/8
         COMMAND+TASKNO        /TASKNO=0 MEANS TASKNO=SENDING TASK
         TIMEHI
         TIMELO
         EXTRA1
         EXTRA2
```

The words TIMEHI and TIMELO specify a time interval from the present time in terms of "system ticks". The user specifies the number of system ticks in a second in the RTS/8 parameter file by defining the parameter SHERTZ. The hardware tick rate (in ticks per second) is specified by the parameter HERTZ. CLKTYP and HERTZ are determined completely by the user's hardware configuration. SHERTZ equals the reciprocal of the software system clock resolution. HERTZ must be an exact multiple of SHERTZ. For example, the parameters for a line-frequency clock might be:

```
DECIMAL
HERTZ= 120
SHERTZ= 10
```

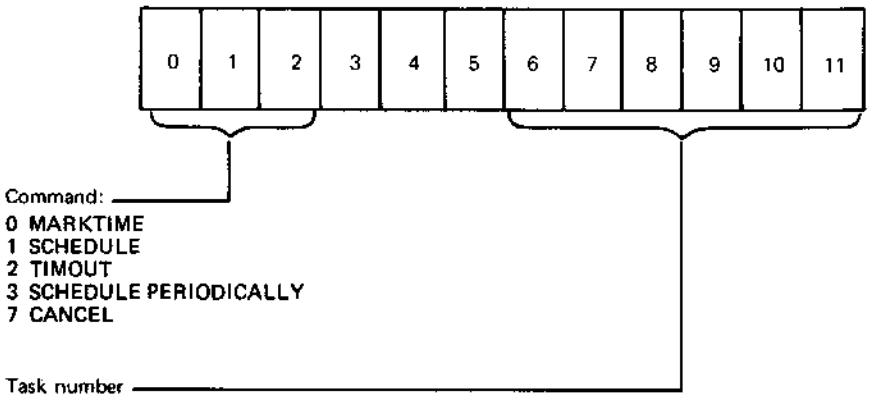
indicating that there will be 10 "system ticks" per second based on a 60-cycle clock. Such parameters might be used if only 1/10 second resolution is necessary in the Clock Handler. Note that the maximum interval that can be expressed in TIMEHI and TIMELO is (2**24)/SHERTZ seconds. This is approximately three days if SHERTZ=60.

Other RTS/8 system tasks use the symbol CLOCK when referring to the Clock Handler. The user should define this symbol in the RTS/8 parameter file to be equal to the Clock Handler's task number. It

should be undefined if a Clock Handler is not to be included in the system. (See Chapter 6 for a description of the parameter file.)

COMMAND is the type of request and has the following meanings:

Octal	Symbolic	Description
0000	MARKTIME	POST the event flag CLKMSG after the specified interval elapses. TASKNO, EXTRA1, and EXTRA2 are ignored.
1000	SCHEDULE	POST CLKMSG immediately. Execute a RUN ER on the task specified by TASKNO after the specified interval elapses. EXTRA1 and EXTRA2 are ignored.
2000	TIMOUT	POST CLKMSG immediately. DERAIL the task specified by TASKNO into a subroutine whose address is specified in EXTRA1 after the specified interval elapses. EXTRA2 is ignored.
3000	SCHEDULE PERIODICALLY	POST CLKMSG immediately. Execute a RUN ER on the task specified by TASKNO after the specified interval elapses, and re-queue this command with the parameters EXTRA1 and EXTRA2 in place of TIMEHI and TIMELO. This has the effect of running the specified task periodically with a period specified by EXTRA1 and EXTRA2.
7000	CANCEL	Cancel all the clock requests for the task specified by TASKNO. TIMEHI, TIMELO, EXTRA1, and EXTRA2 are ignored. POST CLKMSG immediately. Note that the requests are not actually deleted and that they still occupy space in the queue until they time out.



Command Word Format - Clock Handler

The Clock Handler also maintains the current time-of-day (in system ticks until midnight), in symbolic locations TODH (high-order) and TODL (low-order) in Page 0 of Field 0. When this time-of-day reaches zero (i.e., at midnight), it is reset to the quantity -(SHERTZ*86400) (24 hours until midnight) and an OS/8-format date word in symbolic location DATE in Page 0 of Field 0 is incremented by one day.

Note that in order for the quantity SHERTZ*86400 to be contained in 24 bits, SHERTZ must be less than 192. If SHERTZ is larger, an assembly error will result while assembling the Clock Handler.

4.1.1 Examples of Clock Handler Calls

```
            CAL                 /WITH A 60HZ SYSTEM TICK RATE,
            SENDW               /THIS CAUSES THE CURRENT TASK
            CLOCK               /TO "GO TO SLEEP" FOR 2 SECONDS.
            SLEEPM
            .
            .
SLEEPM,     ZBLOCK 3            /MESSAGE HEADER
            0                   /SET EVENT FLAG AFTER INTERVAL
            0;170               /INTERVAL IS 120 (DECIMAL) SYSTEM
                                /TICKS
```

If the user changes the value 170 to the assembler expression 2^SHERTZ, the preceding sequence becomes configuration-independent.

```
            CAL                 /RUN THE TASK REPORT ONCE
            SEND                /EVERY HOUR, INDEFINITELY,
            CLOCK               /ASSUMING A 60HZ SYSTEM TICK RATE
            RUNMSG
            .
            .
RUNMSG,     ZBLOCK 3            /MESSAGE HEADER
            SCHEDULE REPORT PERIODICALLY
                                /RUN REPORT AFTER SPECIFIED
                                /INTERVAL AND PERIODICALLY
                                /THEREAFTER,
            0;1                 /FIRST RUN IS ALMOST IMMEDIATELY
                                /(1/60 SECOND)
            64;5654             /PERIOD BETWEEN RUNS IS 216000
                                /(DECIMAL) SYSTEM TICKS = 3600
                                /SECONDS = 1 HOUR.
```

4.2 TERMINAL HANDLER

The RTS/8 Terminal Handler handles a single terminal in either line or character mode. Input in line mode is terminated by a carriage return or an ALTMODE character and may be edited using the RUBOUT and ^U characters. The RUBOUT character deletes the last valid character typed and prints a backslash; the ^U character deletes the entire line and returns the carriage. Character mode input is not echoed and is terminated by overflow of a specified character count.

If multiple terminals are to be handled, multiple copies of this Terminal Handler must be assembled. Assembly parameters in the body of the handler specify which device codes the handler will use to access its terminal. These parameters also specify whether the handler is to be a "console" Terminal Handler, that is, the terminal

on which the MCR program is going to be run. The console Terminal Handler invokes the MCR whenever a ^C is typed on the keyboard; nonconsole terminal handlers treat ^C as any other character. For the console handler, ^C wakes up MCR by POSTing an Event Flag.

The parameters edited into the distributed version of the Terminal Handler assemble the handler to handle the PDP-8 console terminal as a "console" device. Thus, when the MCR function is required, both the MCR task and the Terminal Handler task must be assembled and included as part of the RTS/8 system. Modification of the Terminal Handler to support a VT50 terminal and other features are described in Section 4.2.1.

The format of messages to the Terminal Handler can be either of the following:

```
ZBLOCK 3                        ZBLOCK 3
command+length                  ASSGN+tsknum
INBUF
OUTTXT

Description:                    Description:

Types text specified by         ASSGN=200
OUTTXT and command, then        Assigns Terminal Handler to task specified
reads text into INBUF.          Deassigns Terminal Handler if tsknum=0
```

Legal Commands, which can be combined, are as follows:

Octal	Symbolic	Action if specified	Action if not specified
4000	NOPACK	Output text is in unpacked ASCII, one character per word terminated by a 0000.	Output text is in packed 6-bit, two characters per word terminated by a 00.
2000	NOCRLF	Do not type a CR/LF after the message.	Type a CR/LF after typing the message.
1000	IND	OUTTXT points to the first word of the output text.	OUTTXT is the first word of the output text.
0400	NOLINE	Input is in character mode; terminated after 'length' input characters read.	Input is in line mode; terminated by a CR or ALTMODE (ESC). The length is still tested.

Length — Is a seven-bit field which specifies the maximum size of the input buffer if input is in line mode, or the number of characters to input if input is in character mode. If input is in line mode and there are LENGTH-1 characters in the input buffer, characters other than carriage return, ALTMODE, RUBOUT and ^U will not be accepted or echoed the message Event Flag is Posted.

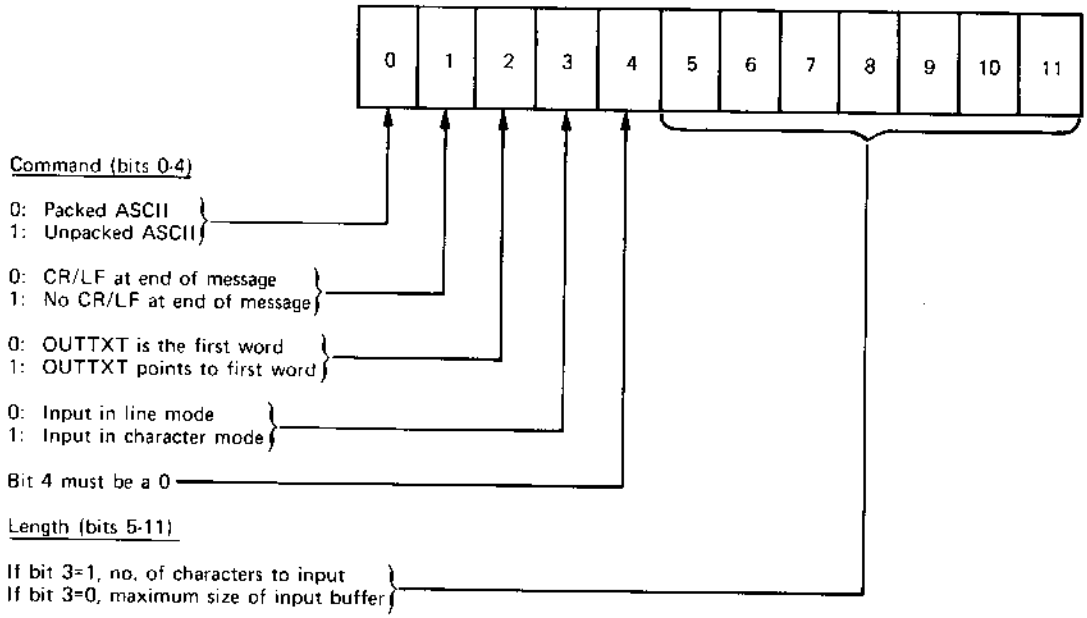
INBUF — Is a pointer to the input buffer; if it is zero, no input is taken. The input buffer is filled with input characters packed one per word with the parity bit (bit 4) forced on. If input is in line mode, the last character of the line is followed by a zero word (if a

carriage return terminated the line) or a -1(7777) word (if an ALTMODE character terminated the line).

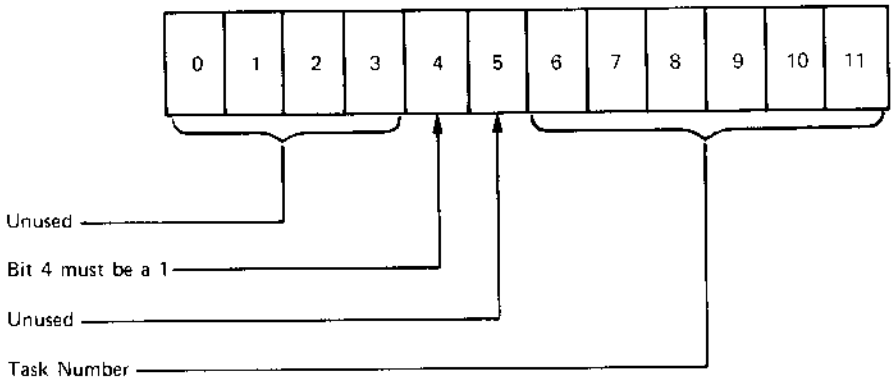
OUTTXT — Is either the first word of the output text string (if IND=0) or a pointer to the first word of the output string (if IND=1000) ir the same field as the message.

ASSGN =200 — "Assigns" the Terminal Handler to the specified task. This will cause the terminal handler to only accept messages from the specified task. If another task tries to SEND a message to the Terminal Handler while it is assigned, the message will be placed in the Terminal Handler's Message Input Queue but will not be removed for processing by the Terminal Handler until the assignment is released. The task to which the Terminal Handler is assigned can release the assignment by sending a message assigning the Terminal Handler to task number 0. No I/O operation is performed by an assignment message.

tsknum — Is a 6-bit field used with the ASSGN command to specify the task number of the task to which the terminal is to be assigned. If this field is zero, the terminal is deassigned allowing the terminal task to accept commands from any task.



Command and Length Word Format - Terminal Handler I/O Mode



Command Word Format - Terminal Handler ASSGN Mode

4.2.1 Additional Assembly Parameters Affecting Terminal Handler Properties

Several assembly parameters are available to the user as an aid in using the TTY task. This section describes these parameters. A summary of their default values is shown in Table 3-4.

VT50 =0 — Do not treat CTRL/S and CTRL/Q as special characters.

=1 — Support CTRL/S and CTRL/Q. If this feature is enabled, typing CTRL/S while data is being printed/displayed on the terminal will cause data to stop until the next CTRL/Q is typed. This can be used on fast CRT terminals to temporarily "freeze" the screen. This parameter must be set to 1 if the user's terminal is a model VT50 or VT52 since these terminals will occasionally send synchronization characters to the host computer of their own volition.

WIDTH =n — Where n is an octal number that sets the page width to n characters. TTY width is currently set to 120(octal) characters. For example, setting the parameter

```
WIDTH = 60
```

changes the TTY page width to 80(decimal) characters. After n characters are printed on the terminal, the handler will automatically type out a carriage-return line-feed. Sometimes it is desirable to suppress this CR/LF (for example, when using direct cursor addressing). In this case, WIDTH should be set equal to 0.

SCOPE — This option is used to determine treatment of the RUBOUT key as follows:

SCOPE=0 provides the normal mode of RUBOUT support (echo rubouts with a backslash).

SCOPE=1 causes RUBOUT to move the cursor left one position, physically removing the character from the screen. If the cursor is in column 1, RUBOUT still works, but has no visible effect.

TAB

This option is used to simulate tabs by the proper number of spaces. This is accomplished via the assembly parameter TAB as follows:

TAB=0 specifies that the hardware does not support tabs. The software simulates tabs with spaces.

TAB=1 specifies that the hardware does support tabs.

FILL

Fill characters are supported via the assembly parameter FILL as follows:

FILL=0 provides no fill characters.

FILL=n sends n fill characters (nulls) after a line feed. The number n must be in the range 1-5. FILL=4 is recommended for 2400 baud VT05s.

CONSOL

CONSOL = 1 means the handler is being assembled for the console terminal(default).

CONSOL = 0 means that this handler will not wake up the MCR when a ^C is typed.

OLDTTY

OLDTTY = 1 specifies the use of the old two-page handler which was supplied with RTS/8 version 1. This handler has fewer features than the new handler but it is a page shorter. The parameters VT50, WIDTH, SCOPE, TAB and FILL described herein have no effect when using this handler.

OLDTTY = 0 specifies the use of the new 3-page terminal handler.

LSBOT

LSBOT = 1 specifies the listing of both the old two-page and new three-page.

LSBOT = 0(default) causes only the handler selected by the OLDTTY parameter to be listed.

TTFLD

Specifies the field of the TTY Handler task; for example, 20 designates field 2.

TTLOC

Specifies the starting location of the TTY Handler task; for example, 3000 designates the starting location at 3000.

Table 4-1
Summary of Terminal Handler Assembly Parameter
Default Values

Parameter	Default Value	Meaning
VT50	1	Support ^S, ^Q
WIDTH	120	Page width of 80(decimal) characters
SCOPE	0	Rubouts echo as \
TAB	0	Simulate tabs
FILL	0	No fill characters
CONSOL	1	^C wakes up MCR
OLDTTY	0	Use 3-page TTY task
LSBOT	0	List only TTY task selected by OLDTTY
TTFLD	10	Not a default value, but given as an example to show that the given number assignments for TTFLD and TTLOC load the TTY Handler in field 1 starting at location 5000
TTLOC	5000	

4.2.2 Useful Equates in the Parameter File

Several useful equates (described in Section 4.2) are available which can be used when sending messages to TTY or LPT tasks. They are as follows:

NOPACK = 4000	Used if output message is not 6-bit ASCII format.
NOCRLF = 2000	Used if output message should not be followed by carriage return/line feed.
IND = 1000	Used if OUTTXT points to the first word of the output text.
NOLINE = 400	Used if input is in character mode.
ASSGN = 200	Used to assign the device handler for use only by this task.
KL8ALINE = 100	Used with KL8-A support (see Section 4.13.2).

4.2.3 Examples of Terminal Handler Messages

```
HIYA,    ZBLOCK 3            /MESSAGE HEADER
         0                   /PACKED TEXT, END WITH CR/LF,
         0                   /NO INPUT
         TEXT /HELLO/        /TEXT TO BE OUTPUT
```

Sending the above message to the Terminal Handler prints HELLO on the terminal.

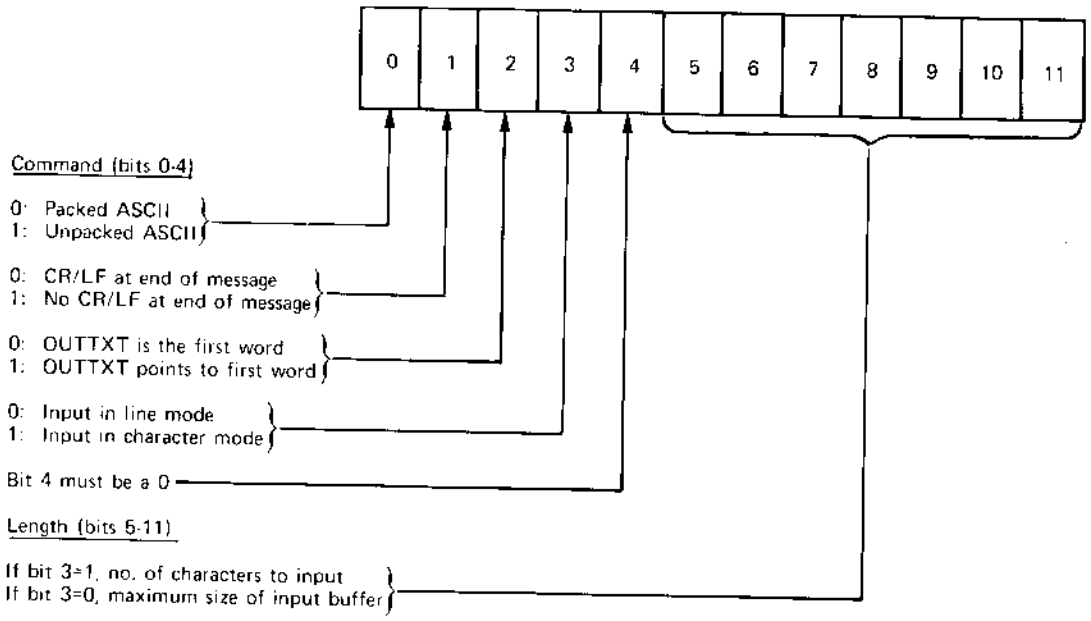
```
QUEST,  ZBLOCK 3            /MESSAGE HEADER
        NOCRLF+60           /PACKED TEXT, NO CR/LF,
                            /48-CHARACTER INPUT LIMIT
        ANSWER              /POINTER TO INPUT BUFFER
        TEXT /TYPE THE ANSWER:/
```

Sending the above message to the Terminal Handler prints TYPE THE ANSWER: on the terminal and inputs a reply without first returning the carriage. The answer obtained from the above message could be printed on the terminal by sending the following message:

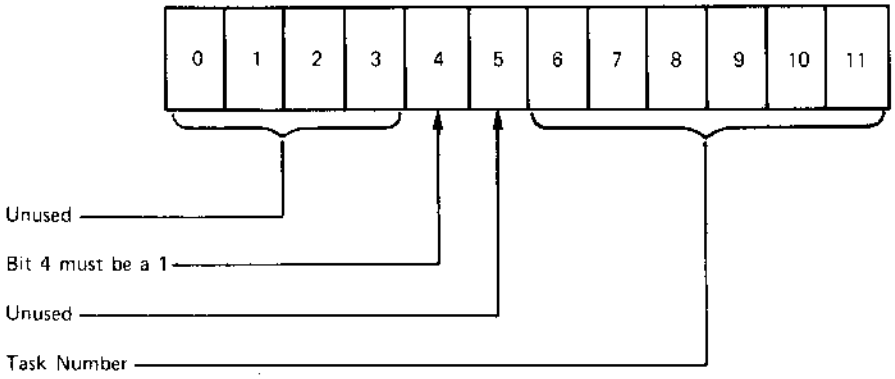
```
TYPANS, ZBLOCK 3            /MESSAGE HEADER
        NOPACK+IND          /UNPACKED TEXT, INDIRECT, WITH CR/LF
        0                   /NO INPUT
        ANSWER              /POINTER TO OUTPUT TEXT
```

4.3 LINE PRINTER HANDLER

The RTS/8 Line Printer Handler outputs to an LE8, LS8E, LP8 or LV8 line printer. The format of messages to the Line Printer Handler is identical to the format of messages to the terminal handler, but the INBUF word and the LINE bit are ignored (the INBUF word must, however, be present in the message).



Command and Length Word Format - Line Printer Handler I/O Mode



Command Word Format - Line Printer Handler ASSGN Mode

4.4 MASS STORAGE HANDLERS

Handlers are available for TC08 DECtape, DF32 and RF08 fixed-head disks, RK8 and RK8E moving-head disks, RX01 floppy disks and LINCtape. All mass storage handlers accept the same message format to read or write blocks on various mass storage devices. However, the Floppy Disk Handler and the LINCtape Handler allow the use of additional parameters other than the ones described herein. These parameters are described in Sections 4.4.1 and 4.4.2.

The format of messages to mass storage handlers is:

```
MSMESG,  ZBLOCK 3
         UNIT
         RW + PAGES + FIELD
         BUFADD
         BLOKNO
         STATUS
```

where:

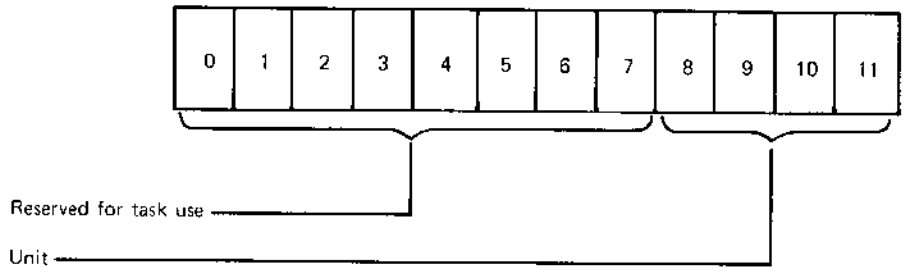
UNIT — Is the number of the logical unit on which the operation is to be performed. DF32 and RF08 disks consist of only one unit. TC08 DECtape has logical units 0-7 corresponding to its physical units 0-7. LINCtape has logical units 0-7 corresponding to its physical units 0-7. RK8 disk has logical units 0-3 corresponding to its physical units 0-3. RK8E disk has logical units 0-7. Units 0-3 correspond to the outer (lower track number) half of physical units 0-3, and units 4-7 correspond to the inner (higher track number) half of physical units 0-3, respectively. RX01 has units 0 or 1 which corresponds to the left and right drive, respectively.

RW — Is 0 for a read operation, 4000 for a write operation.

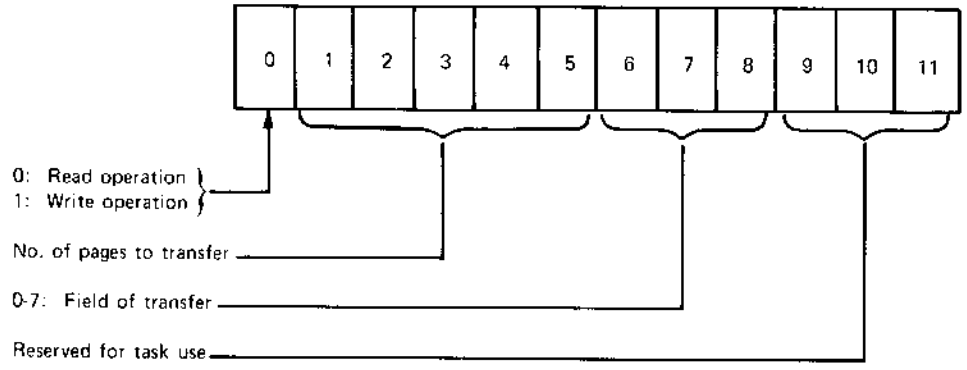
PAGES — Specifies the number of (128-word) pages to transfer (times 100 octal). For example, PAGES=2000 specifies the transfer of 20(octal) pages or 2048 words; if PAGES=0, 40(octal) pages or 4096 words are transferred.

FIELD Is the PDP-8 field in which the transfer takes place (times 10 octal). For example, if FIELD=30, the transfer takes place in field 3.

The RW+PAGES+FIELD word is sometimes called the function word of the message.



Unit Word Format - Mass Storage Handlers



Function Word Format - Mass Storage Handlers

BUFADD Is the starting address of the buffer to be transferred.

BLOKNO Is the block number on the device from which the transfer will begin. All devices are assumed to have 256-word blocks. On DECtape, the first 128 words of each of an even/odd pair of 129-word DECtape records are considered to be a block.

STATUS Is a word that the handler sets on completion of the operation. It contains a zero if the operation is successful, otherwise it will contain a nonzero quantity which is the contents of the device status register. Tasks which use the mass storage handlers should test this word after the I/O operation has been completed (that is, after the Event Flag has been POSTed) to determine if any errors occurred during the transfer. All RTS/8 mass storage handlers retry operations three times if errors are encountered before setting the STATUS word to a nonzero.

Note that the middle three words of a message to the RTS/8 mass storage handlers are identical to the arguments to an OS/8 handler when the same operation is performed.

4.4.1 Floppy Disk Handler

Each copy of the Floppy Handler can control one single or dual RX01 drive; for more than one RX01, multiple copies of the handler are required. The format of messages to the Floppy Disk Handler is:

```
ZBLOCK 3
CODE+DEL+MODE+UNIT
RW+PAGES+FIELD
BUFADD
BLOKNO
STATUS
```

where:

CODE = 0 Regular condition. BLOKNO is interpreted as an OS/8 logical record number. Also, PAGES is interpreted in the OS/8 sense to mean the number of pages of data to transfer. The DEL bit is ignored.

= 4000 Special Physical Sector Condition. PAGES is ignored. One sector is transferred. It is specified by BLOKNO which is to be interpreted as TTTTTTTSSSSS. That is, the high order 7-bits of BLOKNO represent the physical track number. This number must be in the range 0-76 decimal (0-114 octal). The low order 5 bits of BLOKNO represent the sector number on that track. This number must be in the range 1-26 decimal (1-32 octal).

DEL = 0 Deleted data marks should not be considered.

= 2000 Handle deleted data marks (if CODE=4000) as follows: If writing a sector, write deleted data indication. Do not note this fact in STATUS word. If reading a sector, set bit 5 of STATUS word to a 1 if read deleted data indication. In such a case, the STATUS word may be nonzero even though no physical error has occurred. Other STATUS bits are relevant and STATUS negative means hard error.

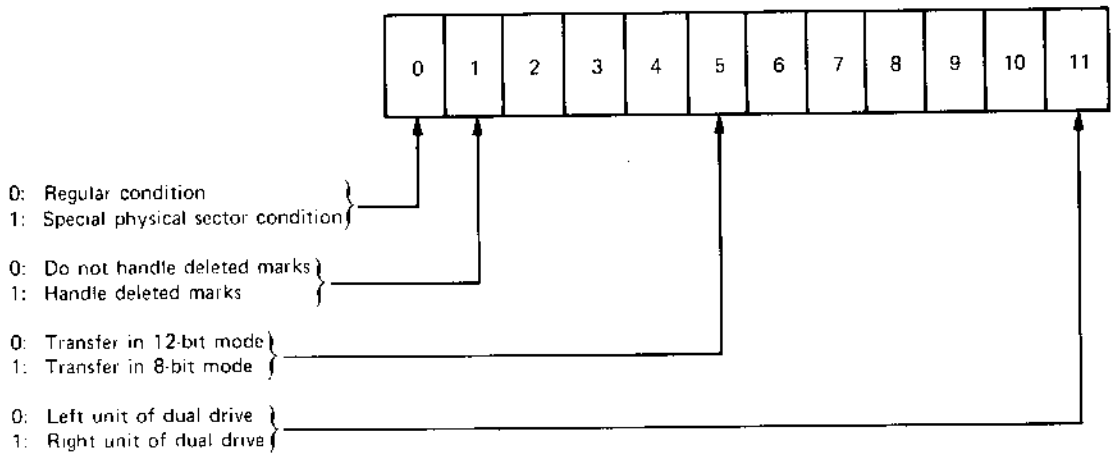
MODE = 0 Specifies transfer in 12-bit mode.

= 100 Specifies transfer in 8-bit mode.

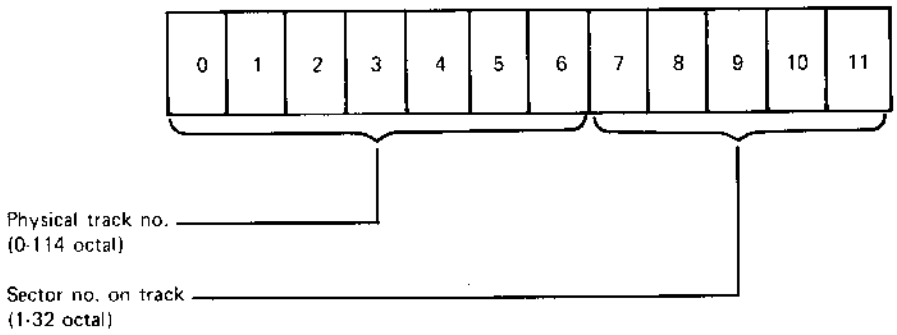
OS/8 format uses 12-bit mode. In 12-bit mode, the 64 12-bit words that comprise an OS/8 floppy sector are packed into the first 96 bytes of the sector, while the last 32 bytes contain random bit patterns. In 8-bit mode, an 8-bit byte on the floppy disk corresponds to the low order 8-bits of a 12-bit word in memory. Data in the high order 4 bits of a word in memory is not transferred to the floppy disk.

In 12-bit mode, a sector contains 64 (decimal) 12-bit words of data. In 8-bit mode, a sector contains 128 8-bit bytes of data.

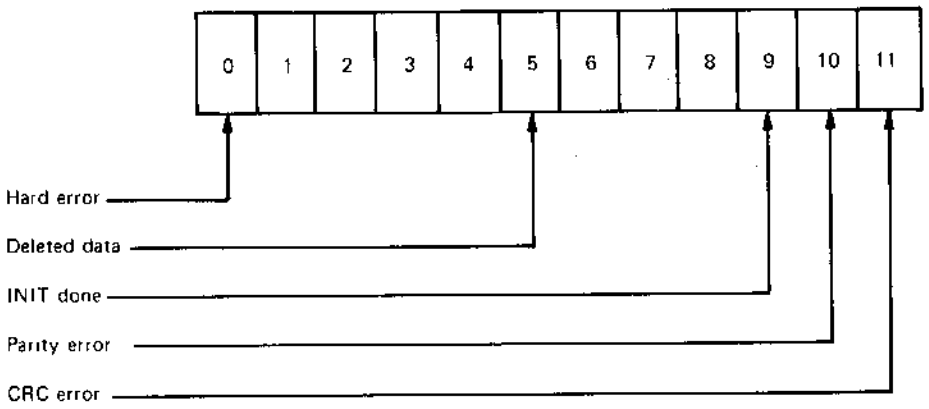
UNIT = Specifies the drive unit number. It may be 0 or 1. The number 0 refers to the unit on the left of a dual drive.



CODE = 4000 (bit 0 set to 1) transfers one sector specified by BLOCKNO as follows:



Unit Word Format - Floppy Disk Handler



Status Word Format - Floppy Disk Handler

The largest legal OS/8 block number on a floppy disk is 755 octal. If block 756 is referenced, an error is generated. Use of larger block numbers may produce unpredictable results. Specifying an illegal track or sector may produce an error with STATUS = 4000.

The standard OS/8 Interleave Scheme is as follows:

OS/8 Logical Block (octal)	Floppy Sectors (track/sector in decimal)			
0	1/1,	1/3,	1/5,	1/7
1	1/9,	1/11,	1/13,	1/15
2	1/17,	1/19,	1/21,	1/23
3	1/25,	1/2,	1/4,	1/6
4	1/8,	1/10,	1/12,	1/14
5	1/16,	1/18,	1/20,	1/22
6	1/24,	1/26,	2/1,	2/3
7	2/5,	2/7,	2/9,	2/11
10	2/13,	2/15,	2/17,	2/19
11	2/21,	2/23,	2/25,	2/2
12	2/4,	2/6,	2/8,	2/10
13	2/12,	2/14,	2/16,	2/18
14	2/20,	2/22,	2/24,	2/26
15	3/1,	3/3,	3/5,	3/7
.				
.				
.				

Track 0 is not used by OS/8, and cannot be accessed in the 12-bit mode.

4.4.2 LINCtape Handler

The LINCtape Handler supports both OS/8 and DIAL format LINCtapes. The format of messages to the LINCtape Handler is:

```
ZBLOCK 3
MODE+UNIT
RW+PAGES+FIELD
BUFADD
BLOKNO
STATUS
```

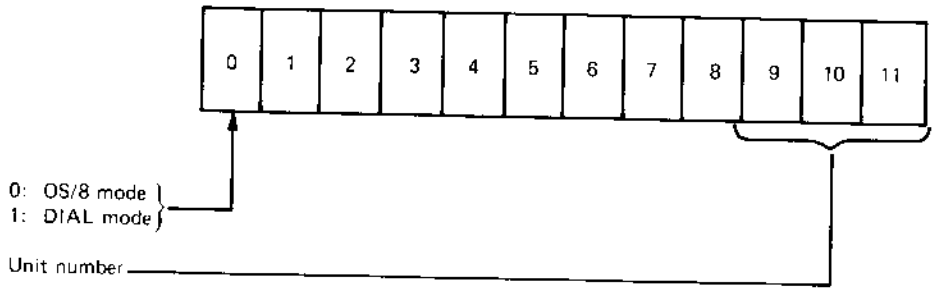
where:

UNIT=	Specifies the LINCtape unit number in range 0 to 7.
MODE=0	Specifies OS/8 Mode. A LINCtape is presumed to contain 200 or 201 (octal) words per physical block.

=4000	Specifies DIAL Mode. A LINCtape is presumed to contain 400 (octal) words per physical block.
	Note: The LINCtape used is not checked to see if it is properly formatted for the specified mode. Use of a LINCtape with improper physical format will produce unpredictable results.
RW =0	Read data from LINCtape
=4000	Write data to LINCtape
PAGES	Specifies the number of 128-word pages to transfer (times 100 octal). For example, PAGES=2000 transfers 20 octal pages or 2048 words; if pages=0, 40 octal pages or 4096 words are transferred.
FIELD	Specifies the PDP-8 field in which the transfer takes place (times 10 octal). (For example, FIELD=30, the transfer takes place in field 3).
BUFADD	Is the starting address of the buffer to be transferred.
BLOKNO	Is the block number on the device from which the transfer will begin. All devices are assumed to have 256-word blocks. On OS/8 LINCtapes, two consecutive physical blocks comprise one OS/8 logical block. Only the first 128 words in each physical block contain meaningful data.
	When running in DIAL mode, BLOKNO represents a physical LINCtape block number. In this case, PAGES must be even because an even number of pages is transferred. If PAGES is (incorrectly) odd, the last page is not transferred, except if PAGES=1 which will result in one block (2 pages) being transferred.
STATUS	is the ones complement of tape check (checksum). The value 0 means no error. STATUS is always 0 on a Write operation. Three software retries are attempted on a checksum read error. Note that the hardware performs infinite retries on most errors (write-lock-out, tape not mounted, bad spot on tape) and does not return control to RTS/8 until successful.

CAUTION

In the OS/8 mode, the word following the end of the buffer is temporarily destroyed while a LINCtape operation is in progress. The location is then restored upon completion of the operation. However, since RTS/8 is a real-time system, code may be executing while the tape operation is in progress. The user must make sure that this word is never referenced while the LINCtape is being used. Under no circumstances should the word following the end of the buffer belong to another task.



Unit Word Format - LINCtape Handler

4.4.3 Example of Mass Storage Handler Call

```
            CAL
            SENDW
            DTA                 /SEND A MESSAGE TO THE DECTAPE
                                /HANDLER
            DTAMSG              /AND WAIT FOR COMPLETION
            TAD STATUS          /CHECK THE STATUS OF THE OPERATION
            SZA CLA
            JMP ERR             /BAD - GO TO ERROR ROUTINE
                                /OK - CONTINUE PROCESSING
                .
                .
                .
DTAMSG,     ZBLOCK 3            /MESSAGE HEADER
            4                   /DECTAPE UNIT 4
            4210                /WRITE 256 WORDS FROM FIELD 1
            BUFFER              /ADDRESS OF BUFFER
            55                  /INTO BLOCK 55 (RECORDS 132 & 133)
STATUS,     0                   /STATUS OF OPERATION STORED HERE
```

4.5 POWER FAIL TASK

The Power Fail Task provides the mechanism by which the system recovers from power failure. If the power-fail/auto-restart hardware option is present and if the system parameter PWRFAL was equated to a nonzero value, the SPL (Skip on Power Low) instruction is included in the interrupt skip chain. If a power low condition occurs, the processor state is saved and the processor is halted. When power comes back, the processor state is restored and an Event Flag is POSTed which wakes up the Power Fail Task. The Power Fail Task restores the clock, console terminal, and OS/8 terminal if they are present, and also performs an action for each task in the system based on the contents of an internal table. Each task has a one-word entry in this table, which contains:

0	If nothing should be done for this task (default value)
-1	If the EFWT (Event Flag Wait) bit should be cleared in the Task Flags Table entry for this task (i.e., this task should be taken out of Event Flag Wait)

ADDR — If the task should be DERAILed to location ADDR in the field in which it is executing as well as having its EFWT bit cleared.

Each task in the system may alter its entry in the Power Fail Task's table by sending a message to the Power Fail Task. The format of the message is:

```
PWRMSG,  ZBLOCK 3
         WORD
```

where:

WORD — is the new contents of the Power Fail Task's table entry for the sending task.

4.6 OS/8 SUPPORT TASK

The OS/8 Support Task supports the execution of the OS/8 operating system as a task under RTS/8. OS/8 is run in the top two or more memory fields under control of the KM8-E memory extension and timeshare option (standard on PDP-8/E, 8/F, or 8/M with 8K or more of core memory) or TSS-8 time sharing hardware option.

NOTE

A jumper on the KM8-E module is used to select the timeshare function. The module is shipped with this jumper in place (timeshare function disabled). The PDP-8A utilizes the memory extension and timeshare option provided by the KM8-A extended option board. A switch on the KM8-A module is used to enable the timeshare function.

The OS/8 Support Task is configured at system startup time to establish a correspondence between OS/8 devices and RTS/8 handler tasks. Terminal input and output from OS/8 are ring-buffered by several characters to minimize input loss due to the usurpation of the CPU by tasks of higher priority. Because of the large number of trapped CDF instructions in OS/8 and its Commonly Used System Programs (CUSPs), response time is slower than a stand-alone OS/8 system but still quite reasonable. The background OS/8 task must have the same system device that was used by the OS/8 system to load RTS/8. The OS/8 Support Task cannot run on a stand-alone PDP-8 without OS/8.

Several parameters in the system parameter file control the assembly of the OS/8 Support Task. The parameters and their meanings are as follows:

OSFLDS — Defined as the number of fields to be dedicated to OS/8.
Example: OSFLDS=2 specifies two fields or 8K of memory for OS/8.

OSKBDV — Set equal to the keyboard IOT code of the OS/8 terminal.
Example: OSKBDV=03 specifies the use of the console terminal keyboard of OS/8.
Note: OS/8 requires its own dedicated terminal.

OSTTDV — Set equal to the teleprinter IOT code of the OS/8 terminal.
Example: OSTTDV=04 specifies the use of the console teleprinter for OS/8.

OSFILL — Specifies how many null characters must follow a line-feed character on the OS/8 terminal. This allows high-speed VT05 terminals to be used as OS/8 terminals. For standard Teletypes[1] and DECwriter terminals, this parameter should be set to zero.
Example: OSFILL=4 allows the use of a 2400 baud VT05.

OSSYSD — Specifies the OS/8 system device driver task.
Example: OSSYSD=DTA specifies DTA0 as the OS/8 system device.

NOTE

The user does not need to include a terminal driver for the OS/8 terminal device (it is built into OS8SUP).

The OS/8 system that runs under the OS/8 Support Task runs all OS/8 CUSPs except BUILD, BOOT, PIP10, INDUSTRIAL BASIC, and BASIC and FORTRAN LAB runtime functions. All references to the keyboard and teleprinter are diverted to the specified OS/8 keyboard and teleprinter. References in OS/8 to the LE8, LS8E, LP8 or LV8 line printers are diverted to the RTS/8 line printer handler if the system parameter LPT is defined; otherwise they are executed directly by the Support Task. References to the following OS/8 device names will be diverted to the corresponding RTS/8 handler if one is defined:

DTA0-DTA7
LTA0-LTA7
RKA0-RKA3
RKB0-RKB3
RXA0-RXA7

If one is not defined, OS/8 will perform the I/O directly using the standard OS/8 handler.

In addition, the OS/8 handlers SYS and DSK are diverted to the handler specified by the parameter OSSYSD. Other references to I/O under the supported OS/8 system may cause the OS/8 support task to hang in a loop. References to a handler called RTS8 are diverted to OS8COM (see Section 4.7).

4.6.1 Mapping of Fields with OS/8 Support Task

The parameter HGHFLD in the parameter file must specify the highest field available to the entire RTS/8-OS/8 system. This is usually the highest field available in memory (e.g., 30 for a 16K machine). The OS8SUP task maps OS/8 fields into real fields as follows. The field which OS/8 uses as field 0 is actually HGHFLD. OS/8 fields 1, 2, 3, etc. are mapped into consecutive fields beginning with field

[1]Teletype is a registered trademark of the Teletype Corporation.

HGHFLD-OSFLDS+1, proceeding upward. If an OS/8 program references a field greater than HGHFLD, unpredictable results will occur, as these fields are mapped over the lower OS/8 fields. The software core size is correctly set to OSFLDS and should be used by multi-field OS/8 programs.

4.7 OS/8 - RTS/8 COMMUNICATION (OS8COM)

The OS/8 Support Task contains a mechanism by which OS/8 can talk to an RTS/8 task. To perform this communication, the OS/8 system must be configured with a handler called RTS8. This handler can be a dummy; it need not do anything. In fact, it can be some other handler to which the name RTS8 has been assigned. The OS/8 Support Task traps all calls to this handler. The arguments that are passed to the RTS8 handler by an OS/8 program will be passed to an RTS8 task called OS8COM. The user is responsible for writing this OS8COM task.

The OS8COM task performs an RTS/8 RECEIVE ER. The task can then receive a message any time an OS/8 program reads or writes to the RTS8 handler. This message looks like any other message to a mass storage device. OS8SUP does make one change to the arguments. Bits 6 through 8 of the function word originally contain the field of the buffer. This is the field where OS/8 expects the buffer to be. When OS8COM gets control, these bits identify the actual field that contains the buffer. OS8COM can return information to OS/8 through these arguments.

4.7.1 Using the OS8COM Task

An OS/8 program that runs an RTS/8 task as specified by the OS/8 user is shown in the following example.

Example:

```
                USR=7700        /LOCATION OF OS/8 USER SERVICE ROUTINE
                JMS PRINT       /PRINT MESSAGE "WHAT TASK WOULD YOU
                                /LIKE TO RUN?" ON THE OS/8 TERMINAL
                JMS READ        /READS RESULT FROM OS/8 KEYBOARD
                                /RETURNS TASK NUMBER IN RANGE 1-77
                                / IN AC
                DCA TASKNUM     /STORE IT AWAY
                CIF 10
                JMS I (USR      /CALL USR
                1               /TO DO A FETCH
                DEVICE RTS8     /OF DEVICE 'RTS8'
ENTRY,          ADDR            /DUMMY ADDRESS (HANDLER WILL ALREADY
                                /BE RESIDENT
                HLT             /ERROR (HANDLER NOT FOUND)
/NOTE THAT THIS CODE IS NOT REUSABLE AND THAT LOCATION
/'ENTRY' IS SET TO THE ENTRY POINT FOR THIS HANDLER
                CIF 0
                JMS I ENTRY     /CALL HANDLER
                0               /DUMMY READ
TASKNUM,        0               /TASK NUMBER
                ZBLOCK 2        /DUMMY
                JMP I (7605     /RETURN TO OS/8
```

It should be noted that TASKNUM is being passed as the second argument instead of the first because OS8SUP automatically modifies bits 6-8 of the first argument, presuming that a mapped field number is located

there. OS8COM expects three arguments after the handler call plus an error return. These must be specified by the user.

Where the OS/8 portion of the program has been written, the OS8COM task that handles the RTS/8 side of the communication must be written. OS8COM is written like any other RTS/8 user task, and an example of what it might look like is as follows:

```
                TASK=OS8COM     /OS8COM IS ASSIGNED A PRIORITY IN THE
                                /PARAMETER FILE
                INIWT=0         /COMES UP RUNNING
                CUR=40          /SPECIFY FIELD HERE
                FIELD CUR%10
                *200            /STARTING ADDRESS
START,          CAL
                RECEIVE         /IMMEDIATELY GO INTO RECEIVE WAIT
MADDR,          0               /ADDRESS OF MESSAGE LEFT HERE
                DCA MSGFLD      /CDF TO MESSAGE FIELD LEFT IN AC
MSGFLD,         HLT
                ISZ MADDR       /POINT TO FUNCTION WORD
                ISZ MADDR       /POINT TO BUFFER ADDRESS
                                /(SECOND OS/8 ARGUMENT)
                TAD I MADDR     /GET TASK NUMBER
                CAL
                RUN             /RUN THIS TASK
                                /OS8COM WANTS TO BE HIGHER
                                /PRIORITY THAN TASK IT IS RUNNING
                TAD MSGFLD
                DCA EFCDF
                TAD (-5
                TAD MADDR       /GET ADDRESS OF EVENT FLAG
                                /FOR MESSAGE
                CAL
                POST            /POST MESSAGE
EFCDF,          HLT
                JMP START       /GET ANOTHER MESSAGE
```

In this example, the task number was put in the second argument of the OS/8 call. However, it became the third word of the RTS/8 message because OS8SUP always adds a word to the mass storage call argument list, namely the unit number. For a description of the OS/8 standard handler call format, see Section 4.1 of the OS/8 Software Support Manual. For a description of the standard message format for mass storage devices, see Section 4.4 of this manual.

4.7.2 Other Techniques

Other techniques which can be employed by the user are as follows:

1. If the RTS/8 handler STATUS word (word 5) of the message posted by OS8COM is nonzero, then return is taken to OS/8 at the error return of the handler call.

2. Arguments may be passed back to OS/8 through the argument list.

3. If more than three words of data need to be passed to OS8COM from OS/8, the user can pass a CDF and address of the area where the data resides. If the CDF occurs as the first argument to the handler call, it automatically will be relocated before being passed to OS8COM.

4.8 OS/8 FILE SUPPORT TASK

The OS/8 File Support Task (OS8F) allows other tasks to look up, create, and delete files in OS/8 directories. This task is included in the same source file as the OS/8 Support Task, but the user can assemble it independently of that task (depending on which tasks are defined in the system parameter file). The format of messages to OS8F is:

```
OSFMSG,  ZBLOCK 3
         DEVHND^10+UNIT+FUNCT
         FILPTR
         STATUS
         BLOKNO
         LENGTH
```

where:

DEVHND — Is the task number of the handler for the desired device.

UNIT — Is the unit number on which the operation is to be performed.

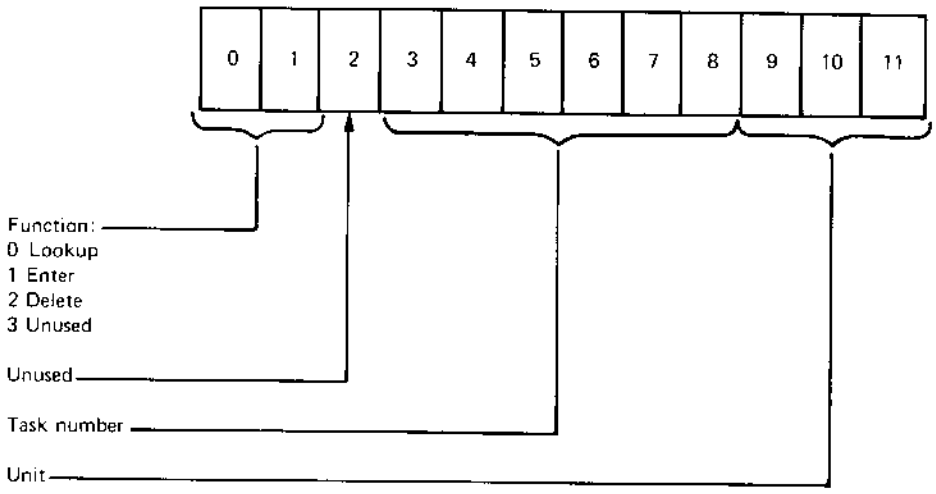
FUNCT — Represents the function to be performed. It can have the following values:

- 0 — Looks up the specified filename and returns its starting block number in BLOKNO, and its length in LENGTH (as a two's complement number).
- 2000 — Enters the specified filename into the first empty space (on the device) whose length is equal to or exceeds the value in LENGTH. Returns the starting block number of the new file in BLOKNO. If a file of the same name previously existed on the device it is deleted. The value of LENGTH is unchanged.
- 4000 — Deletes the specified filename.

FILPTR — Is a pointer to a 4-word filename in the same field as the message. The PAL8 pseudo-op FILENAME can be used to generate these filenames.

STATUS — Describes the final status of the operation as follows:

- 0 — Operation successful.
- 1 — File not found on Lookup or Delete.
- 2 — No room for file on Enter.
- >2 — I/O error occurred. The value is the hardware error status of the device.
- -1 — Invalid directory on device.



OS8F Call Function Word

If both OS8F and the OS/8 Support Task are present in a system, an interlock is set up to prevent simultaneous updating of directory blocks by both systems. Because OS/8 tends to leave directory blocks in memory for long periods of time, this interlock scheme causes lengthy delays for the OS8F task. Before a Delete or Enter operation is performed, OS8F waits until OS/8 is in a state in which:

1. There is no active temporary file on the OS/8 device corresponding to DEVHND and UNIT.

2. OS/8 has just loaded the Keyboard Monitor, Command Decoder, or USR into core.

Look up operations are not interlocked since they do not modify the directory.

4.9 UNIVERSAL DIGITAL CONTROLLER/INDUSTRIAL CONTROLLER SUBSYSTEM (UDC/ICS) HANDLER

The UDC/ICS handler gives the user the capability to control the various types of UDC/ICS functional devices. This handler performs two types of action: immediate and associated. Immediate actions include reading and sending analog and digital values to appropriate UDC/ICS functional devices. Associated actions can be linked to specified events within the UDC/ICS (counters overflowing, switches being thrown). The associated actions can do the following:

1. Run a specified task when the event occurs
2. Set the Event Flag when the event occurs
3. DERAIL a specified task when the event occurs

The number of associated requests that can be pending simultaneously is determined by the size of the buffer, which is specified by the assembly parameter RINGBUF.

The UDC/ICS handler permits the following operations:

1. Analog Output - send a 10-bit value to an analog channel
2. Analog Input - accept input from analog subchannel
3. Digital Output - send a 12-bit value to a digital channel
4. Digital Input - read a digital channel
5. Get Generic Code - determine the generic code for a specified channel
6. Enable Counter - permit interrupts from a counter channel
7. Read Counter - read current value of the counter channel
8. Disable Counter - disable interrupts from a counter channel
9. Enable Contacts - permit interrupts from a contact channel
10. Change Of State - find the current COS value for a contact channel
11. Disable Contacts - ignore interrupts from a contact channel

Each operation is discussed in detail below, including the format of the message for specifying the operation. The first three words are required for use by the Executive. Word 4 specifies one of the 11 UDC/ICS operations which are as follows: AO=0; DO=1; DI=2; GC=3; EC=4; RC=5; DC=6; ECT=7; CS=10; DCT=11; AI=12. Word 5 designates the channel being used for the indicated operation. Words 6 through 8 may be required to completely specify the operation, and the number used is dependent upon the operation. The word that follows the last word specifying the desired operation is used for the value read or the value returned. Word 10 of all UDC/ICS messages contains the error state.

The general format for a UDC/ICS message is:

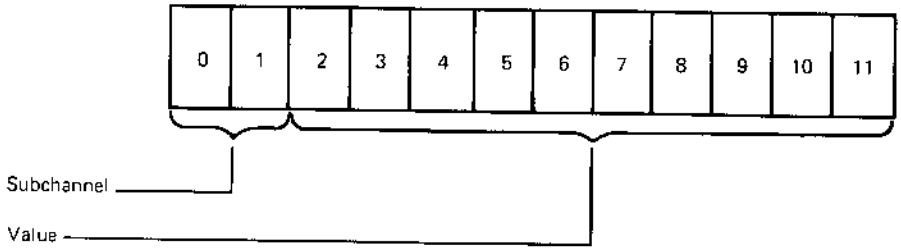
```
ZBLOCK 3
OPERATION
CHANNEL
OPWORD1
OPWORD2
OPWORD3
VALUE
STATUS
```

4.9.1 AO Analog Output

```
Format:   AO
          channel number
          subchannel & value
```

Channel number is the analog output channel. The subchannel and value word is formed by the subchannel (0-3) in bits 0 and 1 and the 10-bit value in bits 2-11. For example, a message for an analog output operation:

```
AOEX,   ZBLOCK 3
        AO                      /ANALOG OUTPUT
        23                      /CHANNEL 23
        4614                    /SUBCHANNEL 2, VALUE 614
        ZBLOCK 3
AOER,   0                       /ERROR INDICATOR
```

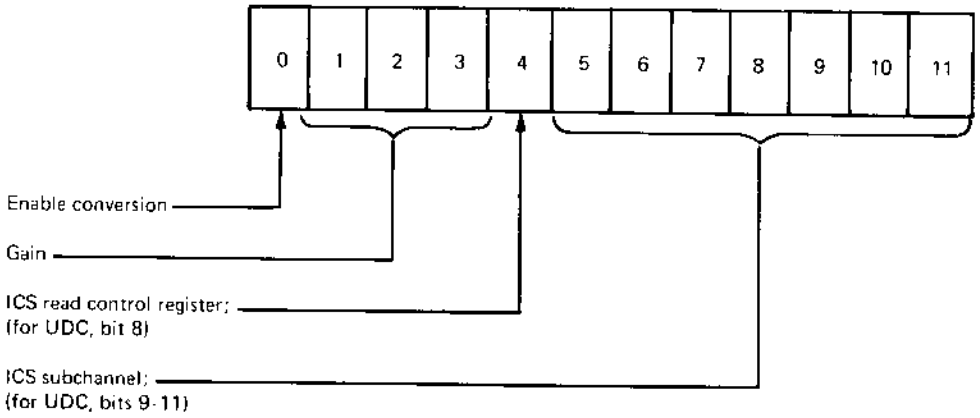


Subchannel and Value Word Format - UDC/ICS Handler

4.9.2 AI Analog Input

Format:
```
AI
channel
subchannel & gain
answer
```

where channel is the analog input channel. The subchannel and gain word need only specify the gain in bits 1-3 and the subchannel in bits 9-11 for UDC, and 5-11 for ICS. The handler automatically sets bit 0 (enable conversion) and read control register (UDC bit 8; ICS bit 4). The ICS analog converters must have addresses which are less than 20 (octal) since all converter modules must be located in the first 16 slots of the ICS unit. After conversion, the digitized value is placed in the answer word.



Subchannel and Gain Word Format - UDC/ICS Handler

An example of a message for an analog input operation is as follows:

```
AIEX,   ZBLOCK 3
        AI                      /ANALOG INPUT
        17                      /CHANNEL 17
        3                       /SUBCHANNEL 3, GAIN 1
AIANS,  0                       /RESULT HERE
        ZBLOCK 2
AIERR,  0                       /ERROR INDICATOR
```

The user should ensure that for each major channel there is sufficient time (approximately 250 microseconds for UDC; 5 milliseconds for ICS) for each subchannel conversion to be completed before another is indicated. In general, it may be helpful if all A/D conversions for a major channel are initiated from the same task.

4.9.3 DO Digital Output

```
Format:  DO
         channel
         value
```

Channel is a legal digital output channel and value is the number to be output. For example:

```
DOEX,   ZBLOCK 3
        DO                      /DIGITAL OUTPUT
        20                      /CHANNEL 20.
        7777                    /VALUE = 7777
        ZBLOCK 3
DOER,   0                       /ERROR INDICATOR
```

4.9.4 DI Digital Input

```
Format:  DI
         channel
         result
```

Channel is the appropriate digital input channel and result will contain the value of the channel when read. For example:

```
DIEX,   ZBLOCK 3
        DI                      /DIGITAL INPUT
        27                      /CHANNEL 27
DIANS,  0                       /VALUE OF CHANNEL 27 WILL BE PUT
                                /HERE
        ZBLOCK 3
DIER,   0                       /ERROR INDICATOR
```

4.9.5 GC Generic Code

```
Format:  GC
         channel
         result
```

The generic code of the specified channel is put in result. For example:

```
GCEX,   ZBLOCK 3
        GC              /DETERMINES GENERIC CODE
        27              /CHANNEL 27
GCANS,  0               /GENERIC CODE PUT HERE
        ZBLOCK 3
GCER,   0               /ERROR INDICATOR
```

Generic codes are as follows: 0 - No interrupt; 1 - Controller error; 2,3 - Contact Interrupt Modules; 4 - Counter Module; 7 - A/D converter.

4.9.6 EC Enable Counter

Format:
```
EC
channel
initial value
reload value
event action
address
```

Channel is the counter channel to be enabled, initial value is the first value to be loaded into that channel, and reload value is the value with which to reload the channel after every event. If the reload value is 0, the counter is not reloaded. The event action and address words specify what happens when the counter interrupts. There are three mutually exclusive possibilities, indicated by setting the appropriate bit in the event action word as follows:

Bit 0 = 1 - Set Event Wait Flag of this job; continue execution of this job when the event occurs. Address word not used.

Bit 1 = 1 - Run a task that sent the message; run task specified by bits 4-11 of event action word. Address word not used.

Bit 2 = 1 - DERAIL the task that sent the message; the address word is only used by the DERAIL operation and specifies the address of the DERAIL subroutine. The subroutine must be in the same field as the calling task.

Bit 3 = 1 - Do action just once. If bit 3 = 0, specified action is performed after each interrupt. Bit 3 indicates whether action is to occur once or repeatedly.

Several enable counter examples follow:

```
ECEX1,  ZBLOCK 3
        EC              /ENABLE COUNTER
        4               /CHANNEL 4
        7700            /INITIAL VALUE OF 7700
        7710            /RESET TO 7710 AFTER EACH EVENT
        4000            /POST EVENT FLAG ON EVENT EVERY TIME
                        /IT OCCURS
        0               /UNUSED
ECER1,  0               /ERROR INDICATOR
```

```
ECEX2,  ZBLOCK 3
        EC              /ENABLE COUNTER
        4               /CHANNEL 4
        1205            /INITIAL VALUE OF 1205
        0               /DON'T RESET
        2016            /RUN TASK 16 ON EVENT EVERY TIME IT
                        /OCCURS
        0               /UNUSED
ECER2,  0               /ERROR INDICATOR

ECEX3,  ZBLOCK 3
        EC              /ENABLE COUNTER
        5               /CHANNEL 5
        10              /INITIAL VALUE OF 10
        7700            /RESET TO 7700
        1015            /DERAIL TO TASK 15 EVERY TIME IT
                        /OCCURS
        5620            /AT LOCATION 5620
ECER3,  0               /ERROR INDICATOR
```

4.9.7 RC Read Counter

```
Format:  RC
         channel
         result
```

where channel is the counter channel whose current value is to be read. That value is placed in result. For example:

```
RCEX,   ZBLOCK 3
        RC              /READ COUNTER
        6               /CHANNEL 6
RCANS,  0               /VALUE OF CHANNEL 6 PUT HERE
        ZBLOCK 3
RCER,   0               /ERROR INDICATOR
```

4.9.8 DC Disable Counter

```
Format:  DC
         channel
```

where channel is the counter channel from which interrupts are to be ignored. For example:

```
DCEX,   ZBLOCK 3
        DC              /DISABLE COUNTER
        6               /CHANNEL 6
DCER,   ZBLOCK 4
        0               /ERROR INDICATOR
```

4.9.9 ECT Enab. Contacts

```
Format:  ECT
         bit & channel
         event action
         address
```

where the bit & channel word specifies the bit on the contact channel from which to enable interrupts. Channel is specified in bits 4-11 and the contact bit is packed in bits 0-3 as a value from 0-13(octal).

Event action and address are specified in the same manner as in the enable counter function. For example:

```
ECTEX1,  ZBLOCK 3
         ECT                  /ENABLE CONTACTS
         5401                 /FROM BIT 13(OCTAL) OF CHANNEL 1
         2013                 /RUN TASK 13 AFTER AN EVENT OCCURS
         ZBLOCK 3
ECTE1R,  0                    /ERROR INDICATOR

ECTEX2,  ZBLOCK 3
         ECT                  /ENABLE CONTACT
         1001                 /FROM BIT 2 OF CHANNEL 1
         4000                 /ON 1ST OCCURRENCE OF EVENT, POST
                              /EVENT FLAG
         ZBLOCK 3
ECTE2R,  0                    /ERROR INDICATOR
```

Twelve messages are required to enable the entire channel.

4.9.10 CS Change of State

```
Format:  CS
         channel
         result
```

where channel is the contact channel whose current change of state value is to be placed in result. For example:

```
COSEX,   ZBLOCK 3
         CS                   /READ COS
         1                    /CHANNEL 1
COSANS,  0                    /RESULT HERE
         ZBLOCK 3
COSER,   0                    /ERROR INDICATOR
```

4.9.11 DCT Disable Contacts

```
Format:  DCT
         bit & channel
```

where bit & channel is specified as in enable contact. That is, bits 0-3 specify the bit (0 - 13 octal) and bits 4-11 specify the channel to be disabled. For example:

```
DCTEX,   ZBLOCK 3
         DCT                  /DISABLE CONTACTS
         5401                 /FROM CHANNEL 1, BIT
                              /13(OCTAL)
         ZBLOCK 4
DCTANS,  0                    /ERROR INDICATOR
```

4.9.12 UDC/ICS Assembly Parameters

The UDC/ICS handler has several assembly parameters that the user must specify to indicate the UDC/ICS configuration. The number and address is required only for those modules that perform interrupts. They are as follows:

RINGBF	Number of interrupts that can be stored in the ring buffer.
NCNTR	Number of counter modules.
NCNTC	Number of contact modules.
NAD	Number of analog input converter modules.
FCTR	Address of the first counter module. The modules must be at contiguous module addresses.
FCT	Address of the first contact interrupt module. Interrupt modules must be at contiguous module addresses.
FAD	Address of the first A/D converter module. Analog input modules must be at contiguous module addresses.
NMPLX	Number of multiplexer modules per analog converter (ICS only).

These parameters are used mainly to specify the sizes of several tables in the UDC/ICS handler, allocated as 30(octal) words per contact module, 3(octal) words/counter module, and 16(octal) words per analog module. The UDC/ICS handler currently assumes that the handler and all its tables are entirely within the same data field (although the user could easily reprogram this).

The user must keep in mind when establishing RINGBF size that if the buffer is full, UDC/ICS interrupts are disabled until there is room in the buffer. Also, each interrupt requires two entries in the buffer; that is, the actual buffer size is 2 * RINGBF.

4.9.13 UDC/ICS Error Conditions

To indicate error conditions, the UDC/ICS handler places a value in the tenth word of the task's message. The values and meanings are:

Value	Meaning
1	Illegal generic code for specified channel and operation
2	Channel or subchannel value not valid
3	Illegal function code
5	UDC/ICS control not responding (power off or hardware error)

The user should initialize and check the error word. A no error condition puts a 0 in this location.

Only errors encountered at noninterrupt time are returned in this manner, thus they may also indicate a faulty UDC/ICS hardware functional device. Generic codes of 0 or 1 encountered at interrupt time are ignored.

4.10 CASSETTE HANDLER

The Cassette Handler (CSA) allows the user to read and write variable-length records on DEC cassettes, as well as to perform various special functions (such as rewind and write end-file). One copy of the Cassette Handler can operate eight units.

There are two general categories of cassette operation:

1. Handler functions - read and write
2. Utility functions - rewind, backspace file gap, write file gap, backspace block gap, and skip to file gap

The user should call these functions in a meaningful sequence. The first word of the message defines the cassette unit and either the handler or utility call.

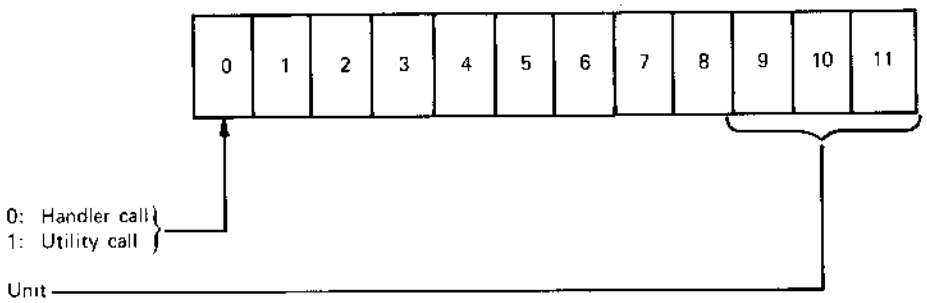
4.10.1 Handler Function

The format of a message to the Cassette Handler when using a handler call is:

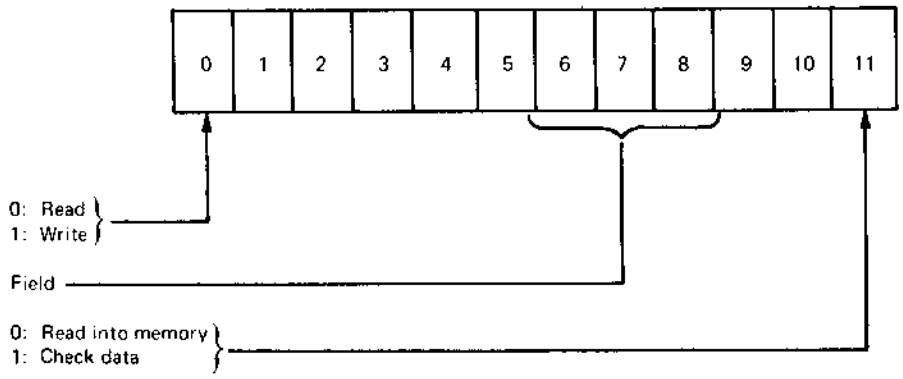
```
ZBLOCK 3
CALL + UNIT
RW + FIELD + NONSTORE
BUFADD
SIZE
STATUS
```

For a handler function, the words after the RTS/8 message header are defined as follows:

Word 1	bit	0 = 0	Utility call
	bit	1 = 0	Handler call
	bits	9-11	Cassette unit
Word 2	bit	0 = 0	Read
		0 = 1	Write
	bits	6-8	Field of buffer
	bit	11	Do not store data (applicable to read only)
Word 3			Buffer address
Word 4			Record size in bits 4-11
Word 5			Status return



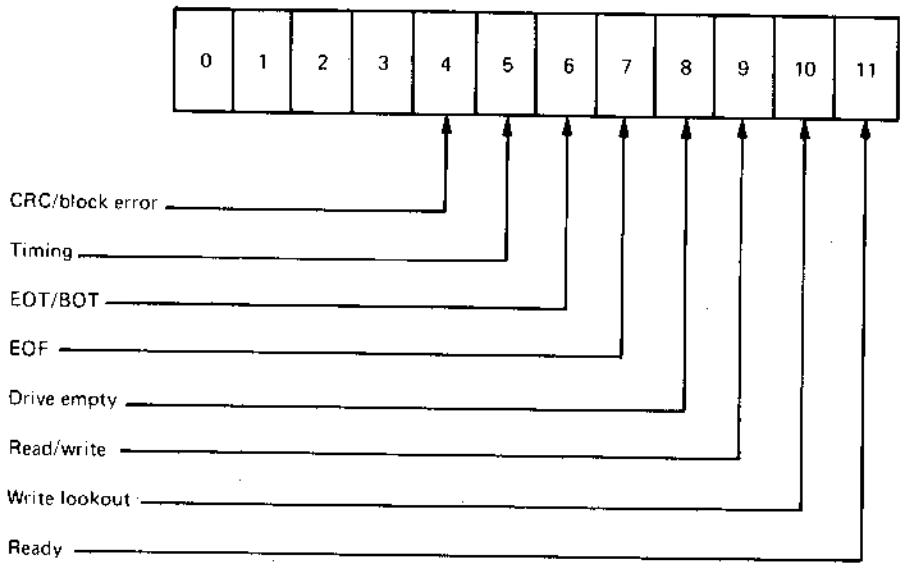
Unit Word Format - Cassette Handler



Function Word Format - Handler Call

Cassette conventions specify a record size of 200 bytes, but the user can use any size up to 377 (8 bits are transferred). The buffer specified by the message cannot cross field boundaries. For a read operation, the buffer is optional (although its word in the message must be included), according to bit 11 of word 2. The nonstore capability can be used for advancing through a long file. Word 5 contains the contents of status register B, which is defined by the bit setting as follows:

Bit	Meaning
4	= CRC /block error
5	= Timing
6	= EOT/BOT
7	= EOF
8	= Drive empty
9	= Read/write
10	= Write lockout
11	= Ready



Status Return Word Format - Cassette Handler

At the end of each cassette operation, the user should examine Word 5 to check for errors encountered.

An example of a cassette handler message to write 100 bytes from a buffer starting at 21200 to cassette unit 3 is as follows:

```
MSG1,   ZBLOCK 3
        4003            /HANDLER OPERATION ON UNIT 3
        4020            /WRITE FROM FIELD 2 THE
        1200            /BUFFER AT 1200 WHICH IS
        0100            /100 BYTES LONG
        0000            /STATUS RETURN
```

To read and not store 200 bytes from unit 2, the message is:

```
MSG2,   ZBLOCK 3
        4002            /HANDLER OPERATION ON UNIT 2
        0001            /READ AND DON'T STORE
        0000            /UNUSED
        0200            /200 BYTES
        0000            /STATUS RETURN
```

4.10.2 Utility Function

The format of a message to the Cassette Handler when using a utility call is:

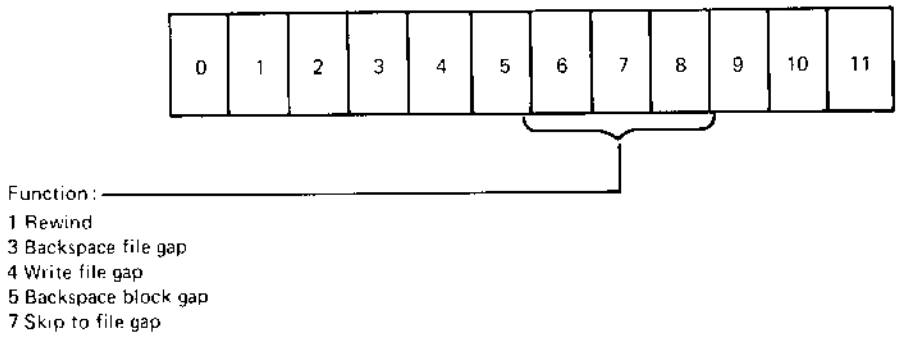
```
ZBLOCK 3
CALL+UNIT
FUNCTION
STATUS
```

For a utility function, the words after the RTS/8 message header are defined as follows:

```
Word 1    bit 0 = 0              Utility call
          bit 0 = 1              Handler call
          bits 9-11              Cassette unit

Word 2    (function in bits 6-8): 10 = Rewind
                                  30 = Backspace file gap
                                  40 = Write file gap
                                  50 = Backspace block gap
                                  70 = Skip to file gap

Word 3    Status return
```



Function Word Format - Utility Call

For example, to request a rewind on unit 1, the message is:

```
MSG3,     ZBLOCK 3
          0001                /UTILITY OPERATION ON UNIT 1
          0010                /REWIND
          0000                /STATUS RETURN
```

If an error is encountered, the operation is retried 3 times, except when a write lock out is placed on a write operation or an error occurs while reading CRC.

The CAPS-8 User's Manual (DEC-8E-OCASA-A-D) is suggested reading for users who are unsure of cassette conventions.

4.11 CASSETTE FILE SUPPORT HANDLER

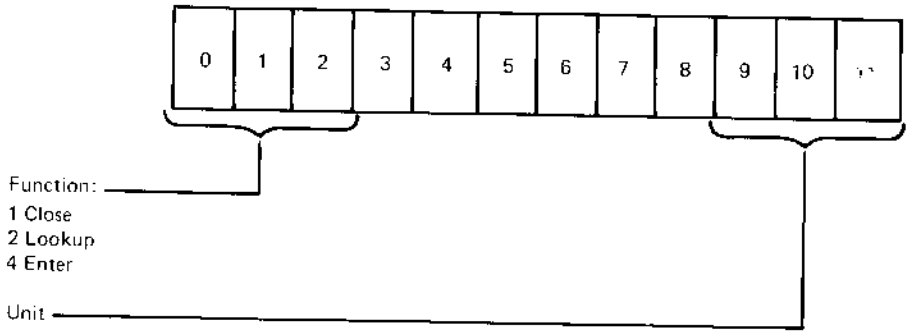
The Cassette File Support Handler (CSAF) supports the DEC standard cassette format and allows the calling task to look up and enter files on cassettes in that format. This handler requires the cassette handler (CSA) to perform the actual I/O operations involved.

The cassette operations ENTER, LOOKUP and CLOSE are performed by the Cassette File Support Handler (CSAF) which in turn calls the cassette handler (CSA). ENTER and LOOKUP require the user to put appropriate information in a record header area with which CSAF performs the file operations. The header area must be at least 40(octal) words long and cannot cross field boundaries.

Word definitions for a CSAF message are as follows:

Word 1	bit 0 = 1	- ENTER
	bit 1 = 1	- LOOKUP
	bit 2 = 1	- CLOSE
	bits 9-11	- unit
Word 2		Address of header for ENTER and LOOKUP; status return for CLOSE
Word 3		Field of header for ENTER and LOOKUP (bits 6-8)
Word 4		Status return for ENTER and LOOKUP

In all cases, the status return is the contents of Status Register B.



Unit Word Format - Cassette File Support Handler

For ENTER and LOOKUP, the format of the header area must conform with cassette standards (and therefore is compatible with CAPS-8). This format is as follows:

Byte	Use
0-5	Filename
6-10	Filename extension
11	File type 1 = ASCII 0 = undefined
12-13	File record length. Currently word 12 must be 0
14-15	Unused
16-23	Date (ASCII) specified as ddmmyy
24-35	Unused

Reference is to 8-bit bytes, one per word, right justified

For an ENTER operation, if a file with the name specified in the header area is found on the specified unit, it is deleted.

For a LOOKUP operation, the record size of the specified file is returned in location header+13 (byte 13). If the file is not found or if an error occurs, this location contains 0.

The CLOSE operation is automatically followed by a REWIND.

Examples of messages follow.

```
MSG4,   ZBLOCK 3
        4000            /ENTER ON UNIT 0
        6400            /INFORMATION IN HEADER STARTING AT
                        /6400
        0010            /OF FIELD 1
        0000            /STATUS RETURN

MSG5,   ZBLOCK 3
        1003            /CLOSE ON UNIT 3
        0000            /STATUS RETURN
```

4.12 PDP-8A NULL TASK

The PDP-8A Null Task counts from 1 to 9999 in decimal in the AC display. It also counts from 1 to 7777 in octal in the MQ display. The source which is called NULL8A, takes up a page. The user can configure the null task into an RTS/8 system by inclusion in the parameter file of its task name and the statement

```
NULL8A = NTASKS+1
```

4.13 KL8-A SUPPORT

The KL8-A is a 4-serial line asynchronous multiplexer for the PDP-8/A that has three lines with partial modem control and one line with full modem control. KL8-A support is available to the RTS/8 Executive, the TTY task, and the OS/8 Support Task. To use KL8-A support, the user should perform the procedures that are described in the following sections.

4.13.1 Executive KL8-A Support

The symbol KL8A in the parameter file is set to a value equal to the number of KL8-A units being employed by the user. If one KL8-A is being used, then KL8A=1 is specified.

If the symbol 'KL8A' is set to 0 or undefined in the parameter file, no KL8-A support will be provided by RTS/8.

KL8-A support is provided by the RTS/8 Executive. The source file KL8ASR.PA must be assembled as follows:

```
.PAL  KL8ASR<PARAM,KL8ASR
```

The parameters in the parameter file that relate to KL8-A service are as follows:

KL8A = 0 or undefined	means that no KL8-A service is desired.
= n	means support for n physical lines is desired. Each physical KL8-A provides four lines.
KL8ADV =	Device code for the first KL8-A. Default is 40. Each KL8-A uses two consecutive device codes (e.g., 40 and 41). If multiple KL8-A's are used, they should have consecutive device codes.
KL8ACT =	Specifies page for start of KL8-A connect routine. Default is 7400 (if KL8A = 1). The KL8-A connect routine must be located in field 0. It is 1-page long for one KL8-A and grows a page for every three additional KL8-A's used (or part thereof). The default value of this parameter is such that the KL8-A support routine gets jammed up against the end of field 0, ending at location 7577.

4.13.2 TTY Task KL8-A Support

KL8-A support in the TTY task is initiated by setting symbol KL8A in the parameter file to nonzero. Then the KL8-A line to be used is specified in place of the terminal IOT device code plus 100. For example, if the TTY task is to control line 3 of a KL8-A,

```
TTDEV = KL8ALINE+3
```

is specified in the parameter file. (The symbol KL8ALINE is defined to have the value 100 in the parameter file.) If more than one KL8-A interface is used, the lines are numbered consecutively beginning with 0 and continuing across interfaces. Thus, KL8-A logical line number 5 actually is physical line number 1 of the second KL8-A interface. Physical lines are numbered from 0 to 3.

KL8-A support requires additional memory in field 0 for Executive Support but does not increase the size of the TTY task. KL8-A support is included in both the old (2-page) and new (3-page) TTY task.

4.13.3 KL8-A Support for the OS/8 Support Task

KL8-A support for OS/8 is similar to that described for the TTY task. However, the following procedure is used. First, the symbol KL8A is set to nonzero in the parameter file. Then, the particular KL8-A line is specified by using a number of the form 100+line in place of the device code, where "line" is the line number of the KL8-A that is OS/8 being used. The symbol KL8ALINE is conveniently defined as being 100 in the parameter file. For OS/8 support, the parameter

```
OSTTDV = KL8ALINE+2
```

specifies that terminal output goes to line 2 of the KL8-A. When more than one KL8-A is used, the lines should be numbered successively as described for the TTY task support in Section 4.13.2.

4.13.4 KL8-A Support for a User Task

The KL8-A support in the Executive allows a user to program the KL8-A in a manner similar to the KL8-J.

First, the user task must insert the KL8-A into the interrupt skip chain and provide a keyboard and printer interrupt routine to service the line he wishes to use. This is accomplished via the following code:

```
CDF CUR
CIF 0
IOF
TAD (LINE^4
JMS I (KL8ACT
KEYBD INTERRUPT ROUTINE
PRINTER INTERRUPT ROUTINE
```

where LINE is the line number of the KL8-A desired. KL8-A line numbers are consecutive, begin at 0, and may span across KL8-As. The KL8-A line number is actually of the form 4a+b where a is the number of the KL8-A (0,1,2...) and b is the physical line number of the specific KL8-A (0-3).

Second, the user must define the instruction corresponding to the TLS instruction that will be used when outputting to the KL8-A line.

For example, if the device code for the KL8-A is 40, then the user will probably want an instruction such as

```
TLSX=6404
```

in his task.

Normally, a program would contain the following code to output a character:

```
TAD char
TLS
```

When using a KL8-A, the task would first connect up the KL8-A support by calling KL8ACT. Then, to output a character, the following code would be used:

```
TAD line ^400
TAD char
TLSX
```

The AC is not cleared by the TLSX.

4.14 EXIT TASK

The EXIT Task is not required for RTS/8 operation. If this task is included in a system, it is run by the MCR EXIT command. The EXIT task performs the same functions as those performed by the MCR EXIT command, that is, it waits for any pending operations to be completed,

then turns off interrupts and returns to the OS/8 operating system. In addition, the EXIT Task allows a user task to request additional special exit processing just prior to shutting down RTS/8. This is done by having the user task send a message to the EXIT Task. This message contains a single word. This word is the address of a routine (in the same field as the message) that will be called (via a JMS) at the time of the exit. When the MCR EXIT command is typed, these routines will be called and executed in the order that they were sent to the EXIT Task.

NOTE

Any message sent to the EXIT Task will not get posted. Also, do not use the MCR REquest command to run the EXIT Task.

CHAPTER 5

MONITOR CONSOLE ROUTINE

The Monitor Console Routine (MCR) provides functions that the user can request from the console terminal to control, inspect, and debug (to some extent) his system.

The MCR indicates that it is active and ready to accept commands by printing the prompting character > on the system console terminal. An MCR command consists of a command word followed by arguments and terminated by either a carriage return or an ALTMODE. Only the first two characters of the command are significant except for the EXIT command. Commands can be a maximum of 40 characters long. If a carriage return terminates the command line, the MCR returns to the terminal for another command when it finishes processing the current command. If an ALTMODE terminates the command line, the MCR puts itself in a wait state when it finishes processing the command. The MCR is brought out of this wait state by typing ^C (CTRL C) on the console terminal.

When the MCR prompts with its > and is waiting for input, no other RTS/8 task can use the terminal. Therefore, if the terminal is used for something other than an exclusive MCR terminal (for instance, error logging), type ^C, type the MCR command and terminate it with an ALTMODE character. This procedure prevents the MCR from tying up the terminal.

5.1 MCR COMMAND ARGUMENTS

Certain syntactic constructions are used as arguments to several MCR commands. The definitions of these arguments follow.

	A single comma or a single space may be used interchangeably to separate arguments to MCR commands.
Task-ID	A Task-ID is either an octal number or a name. If it is a number, it represents the internal RTS/8 Task Number. This number also designates the priority of a task. If it is a name, the first 4 characters of the name are looked up in the MCR's Task Name table to produce a Task Number.
Time-of-day	A time-of-day is of the form hh:mm, where hh represents hours past midnight and mm represents minutes past hh:00.
Address	An Address is an octal number from 1 to 5 digits that represents a PDP-8 memory

address. If the address is less than five digits long it is assumed the high order digits are 0.

Word — A Word is an octal number from 1 to 4 digits long.

5.2 MCR COMMANDS

In the MCR command descriptions that follow, the significant portion of the command word is capitalized. Optional arguments are enclosed in square brackets ([]) and choices are embedded in parentheses, separated by exclamation points (!). Commands preceded by asterisks (*) are not present if the user did not define the symbol CLOCK in the RTS/8 parameter file (indicating that a clock is not in the system), or if the symbol MCRCLK is set to 0 (in order to shorten the MCR code length).

5.2.1 * DAte [mm/dd/yyyy [,Time-of-day]]

The date mm/dd/yyyy, if specified, becomes the system date. For the year portion of the date, only the last digit is significant; the others are ignored since 197 assumed. The RTS/8 system date is automatically incremented at midnight, but all months are treated as being 31 days in length. The second argument, if specified, is set equal to the system time-of-day. If no arguments are specified, the current system date is printed on the console terminal in the form mm/dd/7y.

```
>DATE 07/31/76
>DATE
07/31/76
```

5.2.2 * TIme [Time-of-day]

If a Time-of-day command is specified, it becomes the system time-of-day. If no argument is specified the current system time-of-day is printed out on the console terminal in the form hh:mm.

```
>TIME 14:00
>TIME
14:00
```

5.2.3 NAme Task-ID,Newname

The character string Newname becomes the new name of the task if specified by this command. The old name of that task (if any) is lost. Newname can be any length, but only the first 4 characters are stored. Newname should not be the name of any other task or an error message results.

Examples:

```
>NAME 7 REPORT
```

Task number 7 is given the name REPO.

```
>NAME REPORT,FOO
```

Task number 7, which is known as REPO, is now known as FOO.

NOTE

The system initializes the MCR name table at assembly time to contain the names of any DEC-supplied tasks that are listed in the parameter file (e.g., if the symbol CLOCK is defined in the parameter file as CLOCK=2, task number 2 gets the name CLCK). By editing the file MCR.PA after the label NMTBL, user task names can be permanently included by modifying the MCR name table.

5.2.4 REquest Task-ID [,(@Time-of-day ! Interval)[,Interval]]

The REquest Task-ID command requests a task to run immediately (if only Task-ID is specified), at a given time-of-day, after a given interval, or at a given interval.

Interval is of the form:

nH	n hours
nM	n minutes
nS	n seconds
nT	n system ticks

Requesting a task clears the RUNWT bit in the Task Flags Table entry for that task. The interval, given in the third argument, specifies the period at which the task is rerun. If the parameter CLOCK in the RTS/8 parameter file is not defined, the second and third arguments of this command are ignored and the given task runs immediately. In the examples given below, three different formats are used for the REquest command, but only the first two characters are significant except when using the EXIT command.

Examples:

```
>REQUEST X
```

runs task X immediately.

```
>RE FOO,@2:00
```

runs task FOO at 2:00 am (if it is after 2 am, FOO will run tomorrow at 2 am).

```
>RE 5,10M,5M
```

runs task number 5 ten minutes from now and every five minutes thereafter.

```
>REQ HIPR,1T,6T
```

On a machine with a 60 Hz clock, this command runs the task HIPR immediately (that is, .016 seconds from now, and 10 times per second thereafter).

NOTE

If, at the time the REquest command is executed, (which may be several hours after it is typed in) the task specified by Task-ID does not have the RUNWT bit set in its Task Flags Table entry, then the REquest command is a no-op (no operation), that is, the command has no effect. Similarly, the task will not run upon execution of the REquest command if it had other bits set beside RUNWT; the task will run only when the other blocking bits are cleared.

5.2.5 STop Task-ID

The STop Task-ID command suspends execution of the task specified by Task-ID by turning the RUNWT bit on in the Task Flags Table entry for that task. A task that has been stopped can be restarted by using the REquest MCR command (in this instance it is easier to think of it as the REsume MCR command).

5.2.6 DIsable Task-ID

The DIsable Task-ID command disables future execution of the specified task by setting the ENABWT bit on in the Task Flags Table entry for that task.

5.2.7 ENable Task-ID

The ENable Task-ID command clears the ENABWT bit in the Task Flags Table entry for the specified task, thus enabling it to run. If the ENABWT bit was not set, the command is a no-op.

5.2.8 * CAncel Task-ID

The CAncel Task-ID command cancels any clock queue entries involving the task specified by Task-ID. This includes 1)any entries made by the MCR (from previous timed Request commands), 2)entries involving the specified task made by other tasks (e.g., a timed DERAIL) and 3)entries made by the specified task involving itself (e.g., a timed POST). In the case of the timed POST, the event flag is not POSTed and the task may hang up forever waiting for it.

5.2.9 SYstat [Task-ID]

The SYstat command, depending on whether an argument is specified, prints either a general system status report or a status report in greater detail on a single task. If no argument is specified, the SYstat command prints a system status report. Each line of the report describes an existent task in the system. For each task the report prints the task number/priority, task name (if it has one), and what blocking bits are on in its Task Flags Table entry. Each blocking bit

is printed as a one-letter code, preceded by a space. The one letter codes and their meanings are:

```
E Waiting for event flag
M Waiting for a message
O Waiting for an event flag or a message
R Waiting to be REquested or RUN
S Waiting to be swapped in
D Disabled
U USERWT bit set
N Nonresident wait
```

In addition, an asterisk printed at the end of the line means the task has messages waiting in its input queue.

A more detailed status report on a single task is obtained by specifying the Task-ID of that task as an argument to the SYstat command. The detailed report contains all the information in the general status report, followed by five octal words:

```
WORD 1  The location of the Task State Table
        entry containing words 2-5; this
        word is followed by a colon
WORD 2  Task Link in sign bit, IF in bits 6-8,
        DF in bits 9-11 (PDP-8/E and 8/A Flags
        Register)
WORD 3  Task PC
WORD 4  Task AC
WORD 5  Task MQ
```

Examples:

A general SYstat command might produce the following sample output line:

```
13 CARD E *
```

This line means task number 13, named CARD, is in Event Flag Wait and has input messages pending. The command:

```
>SYSTAT CARD
```

might produce the single line:

```
13 CARD E * 1320:  0022 1741 0000 2525
```

This line indicates that CARD is stopped at location 21741 with its AC and Link zero and 2525 in its MQ.

The user can leave the SYstat facility out of the MCR assembly by setting the system parameter MCRSYS to 0 in the RTS/8 parameter file. Leaving it out saves one page of code.

5.2.10 OPen Address [,Count]

The OPen Address command displays the Count locations in octal starting at Address on the console terminal in the form:

```
11111/ cccc
```

The range of locations displayed may cross a field boundary. If Count is not specified, it is assumed to be 1.

CHAPTER 6

ASSEMBLING AND LOADING TASKS FOR RTS/8

The user assembles RTS/8 tasks with parameter files, using the OS/8 PAL8 assembler. RTS/8 parameter files are all edited versions of a master parameter file (PARAM.PA) that is included in the distributed sources. Appendix A lists the RTS/8 source files. All definitions in the master file which are to be supplied by the user are left blank in the file. For example, a sample line in the file is:

```
PDP8E=  /1 IF PDP 8/E OR PDP 8/A, ELSE 0
```

If this parameter is set to 1, the specified machine is a PDP-8/E or a PDP-8/A. If either machine is not used, this parameter is set to 0.

Thus, a unique parameter file is created for the particular RTS/8 environment, where environment is a combination of the available hardware and the set of tasks being run.

The structure of the parameter file is discussed in the next section. Other sections in this chapter describe 1) the OS/8 BITMAP program which allows the user to construct a map showing the memory locations used by given binary files, 2) a sample RTS/8 program 3) a general procedure for creating an RTS/8 system, and 4) a listing of parameters and their functions that affect the individual RTS/8 system tasks.

6.1 PARAMETER FILE STRUCTURE

The parameter file contains the parameters that the user must define to specify a particular RTS/8 system configuration. A parameter file that has been modified for the demonstration program is shown in Section 8.1. This file also contains user-defined symbols for DECNET/8. For further information on DECNET/8, see RTS/8 DECNET/8 Programmer's Guide and Reference Manual (DEC-08-LDPRA-A-D).

The parameter file is divided into the following five sections. These sections are labeled as follows:

1. Executive Specifications
2. Task Definitions
3. System Task Specifications
4. System Wide Definitions
5. Task Setup

6.1.1 Executive Specifications

The parameters in the Executive Specification section control the assembly of the Executive, and therefore are essential to the RTS/8 system. The parameters in this section and their meaning are as follows:

Symbol	Meaning
PDP8E	Set to 1 if PDP-8/E, PDP-8F, PDP-8M or PDP-8/A is the machine being used; if not, this symbol must be set to 0.
PDP12	Set to 1 if PDP12 is the machine being used; if 0 or undefined, the PDP-12 is not being used.
EAE	Set to 1 if the system should save contents of the MQ during an interrupt or task switching.
PWRFAL	Set to 1 if power fail/restart is enabled in the hardware.
KL8A	Set to a nonzero if KL8-A support routines should be loaded into system.
HGHFLD	Set to a value designating the highest field used; for example, HGHFLD = 30 specifies field 3 when using a machine with 16K core memory.
NTASKS	Set to an octal value that specifies the total possible number of tasks in the system. It also represents the highest number that can be assigned to any task in the system. Not all possible task numbers need be assigned to actual tasks; this symbol merely sets the length of system tables.
CHECKP	Set to 1 if any nonresident task is checkpointable.
PARTNS	Set to the number of memory partitions allocated in the system. Set to zero if there are no memory partitions defined in the system. For example, PARTNS = 2 indicates that there are two memory partitions defined, that is, partition number 0 and partition number 1.

6.1.2 Task Definitions

The Task Definitions section defines symbolic names for the various system tasks. The names of all system tasks which are to be included in the system are defined here. Any system task not included should have the line which defines it deleted from this section. Perform this deletion by inserting a slash (/) character at the beginning of the line, which makes the entire line a comment. Symbolic definitions of the user's own tasks can be added to this section. The user is reminded that the assignment of task numbers in octal indicates task priority, that is, the lower the number, the higher the priority of the task.

The Task Definitions section, as it initially appears to the user, is shown below.

```
/COMMON TASK NUMBERS - EDITED BY USER
/IT IS ADVISABLE TO DEFINE ALL TASKS HERE.  NAMES GIVEN BELOW
/ARE USED BY SOME SYSTEM TASKS AND SHOULD REMAIN COMMENTED OUT
/IF THE CORRESPONDING TASK IS NOT INCLUDED IN THE SYSTEM

/PWRF=            /POWER FAIL HANDLING TASK
/CLOCK=           /CLOCK HANDLER - SHOULD BE HIGH PRIORITY
/SWAPPER=         /NONRESIDENT TASK SWAPPER TASK
/TTY=             /TELETYPE DRIVER TASK
/LPT=             /LINE PRINTER DRIVER TASK
/MCR=             /MONITOR CONSOLE ROUTINE
/DTA=             /DECTAPE DRIVER TASK
/LTA=             /LINCTAPE DRIVER TASK
/RK8=             /RK8 OR RK8E DISK DRIVER TASK
/RF08=            /RF08 DISK DRIVER TASK
/DF32=            /DF32 DISK DRIVER TASK
/CSA=             /CASSETTE DRIVER TASK
/CSAF=            /CASSETTE FILE SUPPORT TASK
/UDC=             /UNIVERSAL DIGITAL CONTROLLER TASK
/RX8A=            /FIRST FLOPPY CONTROLLER
/RX8B=            /SECOND FLOPPY CONTROLLER
/RX8C=            /THIRD FLOPPY CONTROLLER
/RX8D=            /FOURTH FLOPPY CONTROLLER
/OS8=   NTASKS    /OS/8 SUPPORT - NORMALLY LOWEST PRIORITY
/OS8F=            /OS/8 FILE SUPPORT
/DDCMP=           /DDCMP TASK FOR DECNET
/NSP=             /NETWORK SERVICES PROTOCOL TASK
/NIP=             /NETWORK INFORMATION PROGRAM
/TLK=             /NETWORK TERMINAL COMMUNICATIONS TASK TRANSMITTER
/LSN=             /NETWORK TERMINAL COMMUNICATIONS TASK RECEIVER
/NULL8A=          /NULL JOB FOR PDP-8/A
/EXIT=            /EXIT TASK
/DKC8A=           /AUXILIARY DKC8A HANDLER
```

This section of the parameter file is shown in Section 8.1 after it has been modified for the demonstration program. It also shows the addition of the two nonresident tasks used in the demonstration program.

6.1.3 System Task Specifications

The parameters in the System Task Specifications section control the assemblies of the various RTS/8 system tasks. The set of parameters controlling a specific task are all grouped together and assembled conditionally only if that task name is defined in the Task Definitions section of the parameter file. The user edits the parameters in this section. The parameters and their meanings are listed in Section 6.4.

6.1.4 System Wide Definitions

The System Wide Definitions section includes the definitions of the symbols that RTS/8 uses to describe Executive Requests and Task Status Flag bits. It also contains useful definitions such as instruction equivalences, monitor call values, UDC/ICS functional values and system locations. The user should not alter this section.

6.1.5 Task Setup

The Task Setup section uses five symbols that the user defines in the body of this task to initialize the RTS/8 table entries needed to put that task in the system. These five symbols and their definitions are:

TASK — Defines the task number of the task by a statement of the form:

```
TASK=symbol
```

where "symbol" is the symbolic name for the task that the user has defined in the RTS/8 parameter file.

CUR — Defines the field of the task's starting address in bits 6-8 (e.g., CUR=10).

NOTE

The user must place the task's starting code in the field specified by CUR. This is done by using the PAL8 assembler pseudo-op FIELD.

For example, FIELD CUR%10 places the task's starting code in field 1.

START — Defines the task's starting address (not necessarily the lowest address in the task)

INIWT — Defines the initial wait bits in the Task Flags Table entry for this task. For example, INIWT = 0 means the task is runnable when the system starts up; INIWT = RUNWT (1000 octal) specifies that this task is not runnable initially and is in a Run Wait condition. This task becomes runnable when another task issues a RUN ER or when the operator types a Request command to the MCR. If INIWT is undefined, the task starts up being runnable.

VERS — Defines the task's version number, this is an optional parameter. By convention, the task's version number becomes the task's initial MQ value.

The user can define up to three tasks in one assembly. The corresponding symbols for the other tasks are TASK2 and TASK3, CUR2 and CUR3, etc. The task setup section places its data into the RTS/8 tables by origining into them; no executable code is generated. If desired, more than three tasks can be created in one assembly by adding the code for any additional tasks at the end of the PARAM.PA file. It should be noted that only one task is defined in the demonstration program in Chapter 8.

6.2 CREATING AN RTS/8 SYSTEM

An RTS/8 System can be created by using the general procedure that is described in this section. It is assumed that the user has physically mounted a copy of the distribution medium, and has bootstrapped the development system. Although nonresident tasks are treated in this procedure, greater detail on employing nonresident tasks is given in Chapter 7.

The general procedure for creating an RTS/8 System is as follows:

1. Layout on paper the system and user tasks required for the particular RTS/8 configuration to be employed. Utilize the tables and memory map given in Appendix B that show the RTS/8 components, their sizes, and their default origins to determine where the tasks are to be loaded into memory.

2. Assign task names and task priorities. If nonresident tasks are used, assign the Swapper Task a higher priority than any of the nonresident tasks. Remember that the lower the value assigned to a task, the higher its priority.

3. When large programs are involved, a documentation file should be created as a user convenience to maintain a directory of the system configuration. This file can contain information such as the tasks employed in the system, task names, task priorities, and control files.

4. Obtain a listing of the master parameter file (PARAM.PA). Use the OS/8 command

   ```
   .LIST PARAM.PA
   ```

   to get a listing from a line printer, or

   ```
   .TYPE PARAM.PA
   ```

   to get a listing from a terminal.

5. Use an editor (EDIT or TECO under OS/8) to establish the values of the parameters in the parameter file (PARAM.PA). The structure of the parameter file is described in Section 6.1, and the parameters affecting the individual RTS/8 system tasks are described in Section 6.6.

   PARAM.PA should be read in as an input file, edited, and then renamed as an output file. This procedure maintains the integrity of DIGITAL-supplied sources.

6. Create and edit any control files that are used (See Section 6.5).

7. Assemble the tasks with the parameter file after all the required parameters are defined. This can be accomplished by individually assembling each task with the parameter file as follows:

   ```
   .PAL RTS8<PARAM,RTS8
   ```

   or

   ```
   .PAL PARAM-NB, RTS8
   ```

The CCL option -NB indicates that a binary file should not be created. Shown is the assembly of the RTS/8 Executive. This has to be done for every task that is included in the system. Each control file used must also be assembled. Assemble the control file with the parameter file, placing the control file between the parameter file and the required module as follows:

```
.PAL TTY1<PARAM,TTYCF1,TTY
```

An alternate and more efficient method that can be employed is to use a Batch stream. The assembly, load and save commands for the system are generated as a Batch job. The OS/8 SUBMIT command is then used to run BATCH which will use the Batch job commands as inputs and execute them. This method can also be used for installing nonresident tasks into the system.

The assembly of each task, preceded by the parameter file, is required because there is no linking loader function with RTS/8. The assembly process creates binary files from the sources that are ready to be loaded and run. Each RTS/8 system task contains assembler code that assembles it for loading into a specific area of core memory. The user can assemble tasks to load into areas of memory not used by system tasks, or edit the system tasks for loading into specific areas. Page 0 locations and autoindex registers used by system tasks can also be redefined by editing the affected tasks.

8. Obtain a bitmap of RTS/8 tasks to determine if two or more tasks are erroneously loaded into the same memory area or use the same page 0 locations (See section 6.3).

9. Load the system after all the required tasks are assembled. All required system tasks, user tasks, and control files can be loaded at one time as follows:

```
.LOAD RTS8,CLOCK,TTY,MCR,UT1,UT2
```

The RTS/8 Executive Task always must be loaded first. Also, nonresident tasks are included in this step in order to load their resident portions and executive table entries.

10. Save the system after it is loaded by using the following command:

```
.SAVE SYS filename
```

If the system is saved, the user does not have to rebuild it each time it is needed.

11. When using nonresident tasks, create a SAVE image file (nonresident disk image) for each nonresident task from its binary file as follows:

```
.LOAD TASKX
```

Then save the nonresident portion of the task on the swap device:

```
.SAVE DSK TASKX N1-N2
```

where DSK is the swap device, N1 is the lowest address in the partition, and N2 is the highest address in the partition.

12. Start the system by using the following command:

```
.R filename
```

The following is applicable when using nonresident tasks. When the system starts, it calls the OS/8 command decoder, which types an asterisk on the console terminal. At this time, initialize the block address of each nonresident task core image as follows:

```
*DSK:TASKX=N
```

In this command line, DSK is the swap device and TASKX is the core image file containing the nonresident portion of task N. Repeat this procedure for each nonresident task, one task per line, and terminate the last line with an ALTMODE. This procedure automatically initializes and starts the real time system.

An RTS/8 system created with the procedure just described has a starting address of 00200. If an RTS/8 system was not specifically configured for a PDP-8/E or PDP-12, it halts initially to allow the operator to clear any stray device flags by operating nonstandard hardware switches or by pressing START. Press START to resume operation on a PDP-8, 8/I or 8/L.

6.3 USING THE OS/8 BITMAP PROGRAM

The OS/8 BITMAP program can be extremely useful in determining that no two RTS/8 tasks are loading into the same area or using the same Page 0 locations. OS/8 BITMAP accepts a list of binary files as input and produces as its output a map of core memory. Each location of core memory is represented in this map by a single digit which has the following meaning:

0	Nothing has been loaded into this location
1	Information has been loaded into this location
2	Information has been loaded into this location twice
3	Information has been loaded into this location three or more times

There are certain places in core memory where 2's are allowed to appear in the bit map. These areas are the RTS/8 Executive Tables (starting at location 01200), the MCR name table in the MCR, the power fail action table, and nonresident task partitions (which may contain 2's and 3's). Appearance of a 3 or a 2 in an area other than these three areas just mentioned in the bit map indicates that two or more tasks are being loading into the same location.

6.4 SAMPLE RTS/8 TASK PROGRAM

The task that is used as an example in this section was selected for its simplicity, and to show the basic concepts of RTS/8 operation. The purpose of the task is simply to print "HELLO". The user requires an RTS/8 system configuration that includes a console Terminal Handler (TTY) and the Monitor Console Routine (MCR). It is assumed that the

task will be running on a PDP-8 E/F/M/A with 8K of core memory and a standard console terminal.

The program (SAMPLE.PA) listed below is complete, and when assembled with the parameter file, will run as task 5 in a properly-configured system. The task is initially in the Run Wait state. When requested by the user through the Monitor Console Routine, the task prints "HELLO" on the console terminal, and then suspends itself. If again requested by the user, the same sequence will occur.

```
/SAMPLE RTS/8 PROGRAM
        TASK=5          /GIVEN A PRIORITY OF 5
        CUR=0           /IN FIELD 0
        INIWT=RUNWT     /SET TO RUN WAIT STATE
        FIELD 0         /PLACED IN FIELD 0
        *3000           /ANY AVAILABLE PAGE
START,  CAL
        SENDW           /MUST BE DEFINED AS "START"
        TTY             /SEND AND WAIT
        MSG1            /THE TERMINAL
        CAL             /THE MESSAGE BLOCK
        SUSPND          /SUSPEND THE TASK
        JMP START       /RESUMES HERE IF REQUESTED
MSG1,   ZBLOCK 3        /RTS/8 LINKAGE
        0;0             /TERMINAL OPTIONS
        TEXT /HELLO/    /TYPE HELLO
        $
```

An edited version of PARAM.PA now is required for this configuration, and it is named PARAMS.PA. To produce PARAMS.PA from PARAM.PA, the following definitions are edited into the parameter file:

```
PDP8E=1
HGHFLD=10
NTASKS=10
TTY=4
MCR=3
```

NTASKS is the largest task number used with this system. It is assigned any number greater than the values for MCR, TTY and the sample task, and which is smaller than 100 octal. The task values for SAMPLE, MCR and TTY were arbitrarily chosen in the range greater than zero and less than NTASKS.

The program is then assembled as follows:

```
.R PAL8

*SAMPLE.BN<PARAMS.PA,SAMPLE.PA
```

The system for the sample task requires the Executive Task RTS8 plus the TTY and MCR Task to run. This can be accomplished by using the following Batch stream:

```
$JOB
.PAL PARAMS-NB,RTS8
.PAL PARAMS-NB,MCR
.PAL PARAMS-NB,TTY
.PAL PARAMS-NB,SAMPLE
.LOAD RTS8,MCR,TTY,SAMPLE
.SAVE SYS:SAMP
$END
```

Assembly and loading of the tasks, including saving the program, is now complete. The CCL option -NB indicates that PARAM is not to be used as the name of the binary file being created.

The system now can be run as follows (user input is underlined):

```
.R SAMP                 /OS/8 RUN COMMAND
>RE 5($)                /REQUEST SAMPLE FROM MCR
HELLO                   /TASK EXECUTES AND SAYS 'HELLO'
^C                      /RETURNS CONTROL TO MCR
>SYSTAT                 /SYSTEM STATUS COMMAND
03    MCR               /MCR TASK
04    TTY               /TERMINAL TASK
05          R           /TASK 5 WAITING TO BE RUN
>RE 5($)                /RUN TASK AGAIN
HELLO                   /TASK SAYS 'HELLO' AGAIN
^C                      /RETURN CONTROL TO MCR
>EXIT                   /RETURN TO OS/8
.                       /OS/8 MONITOR
```

NOTE

($) is the ALTMODE character; all other input lines are terminated by a Carriage Return.

6.5 USE OF CONTROL FILES UNDER RTS/8

There are times when a user may want to assemble a given source module in more than one way and use the results under RTS/8. For example, suppose there are three terminals that a user wants to service under RTS/8. Each terminal has its own characteristics, and each copy of the TTY task needs to have its own set of parameters. The user must load three copies of the TTY task into memory at different locations, and possibly in different fields. This cannot be accomplished efficiently when using a single parameter file. With a single parameter file, three copies of the file TTY.PA must be made and each one edited to produce three individual tailored copies of the TTY task. This procedure is not convenient or modular.

A better way to do this is to use a control file that contains all the equates necessary to define the parameters needed by a particular TTY task. The control file then is assembled together with and placed between the parameter file and the TTY module. For example,

```
.PAL TTY1 <PARAM,TTYCF1,TTY
```

creates a binary file called TTY1 from TTY.PA using a control file called TTYCF1.

To facilitate using this procedure, a skeleton TTY control file is supplied with the RTS/8 task sources. It contains all the parameters that the user normally defines in the parameter file. Thus, a user who wants to use multiple terminals, instead of editing the parameters in the parameter file, can create a control file for each terminal that is used, and then edit the control file to make multiple copies as necessary.

An example of a TTY Control file after it has been edited by a user is shown below.

```
        TASK=TTY1
                        /DEFAULT IS 'TTY'
/       TTDEV=          /PRINTER DEVICE CODE - DEFAULT IS 04
/       KBDEV=          /KEYBOARD DEVICE CODE - DEFAULT IS TTDEV-1
        CONSOL=1
/       VT50=           /1 ENABLES CTRL/S AND CTRL/Q
        SCOPE=1
/       FILL=           /NUMBER OF FILL CHARACTERS, I.E. 4
/       WIDTH=          /TTY LINE WIDTH (0 MEANS INFINITE), DEFAULT
                        /120
/       TAB=            /1 IF TTY HAS HARDWARE TABS
        OLDTTY=0
/       LSBOT=          /1 LISTS BOTH HANDLERS (DEFAULT 0)
        TTFLD=20
        TTLOC=3000
```

This example shows a TTY control file that has been edited for task TTY1. This terminal handler task will be assembled for the console terminal. Setting SCOPE=1 causes RUBOUT to move the cursor left one position, physically removing the character from the screen. The new three-page handler is specified. It is placed in field 2, starting at location 3000.

A skeleton control file is also supplied for the RX01 floppy. Users can easily generate and use control files for other purposes.

6.6 RTS/8 SYSTEM TASK PARAMETERS

This section provides a convenient grouping of those parameters which affect the individual RTS/8 system tasks. Given for each system task is the parameter, its function and where applicable, an example. The section or chapter where a detailed description of the task appears in the text of this manual is noted after the task subhead.

6.6.1 Clock Handler Parameters (Section 4.1)

PARAMETER	MEANING
CLKTYP	Specifies selection of hardware clocks as follows: 0 = DK8-EA/DK8-EC 1 = KW12 2 = PDP-8/A 3 = DK8-EP
CLKQLN	Specifies minimum number of entry slots in the clock queue (default is 20).
HERTZ	Specifies number of hardware ticks per second. HERTZ and SHERTZ are decimal values in that they are preceded in the parameter file by the pseudo-operator DECIMAL.
SHERTZ	Specifies the number of system ticks per second. This parameter is followed by the pseudo-operator OCTAL, which resets the radix to its original octal base.

6.6.2 Swapper Parameters (Section 7.4)

SYS	Specifies the swap driver task; for example, SYS = RK8 specifies the RK8 driver task.
SUNIT	Specifies the swap device physical drive unit; SUNIT = 0 selects RKA0.

6.6.3 Terminal Handler Parameters (Section 4.2)

PARAMETER	MEANING
TTDEV	Is set to the proper printer device code; default value is 4.
KBDEV	Is set to the proper keyboard device code; default value is TTDEV-1.

The following parameters are available to the user to facilitate the use of the TTY task.

VT50	Is set to 1 (default) to enable CTRL/S and CTRL/Q functions. When set to 0, CTRL/S and CTRL/Q are not treated as special characters. Typing CTRL/S while data is being printed/displayed on the screen stops the data presentation until the next CTRL/Q is typed. This parameter must be set to 1 if the user's terminal is a VT50 or VT52. Both CTRL/S and CTRL/Q turn off the echo flag.
CONSOL	Is set to 1 to specify that the handler is being assembled for the console TTY (default). Set to 0 to specify that this handler should not wake up the MCR when ^C is typed.
WIDTH	Is set to an octal number that specifies the TTY page width. TTY width is currently set to 120 (octal), that is, a page width of 80 decimal characters. WIDTH = 60 sets the TTY page width to 48 decimal characters.
SCOPE	This option is used to determine the treatment of the RUBOUT key as follows: SCOPE=0 (default) provides the normal mode of RUBOUT support. SCOPE=1 causes RUBOUT to move the cursor left one position, physically removing the character from the screen. If the cursor is in column 1, RUBOUT still works, but has no visible effect.
TAB	This option simulates tabs by the proper number of spaces. This is accomplished via the assembly parameter TAB as follows: TAB=0 (default) specifies that the hardware does not support tabs. The software simulates tabs by spaces. TAB=1 specifies that the hardware does support tabs.

FILL	Fill characters are supported via the assembly parameter FILL as follows: FILL=0 (default) does not provide any fill characters. FILL=n sends n fill characters (nulls) after a line feed; n must be in the range 1-5. FILL=4 is recommended for 2400 baud VT5's.
OLDTTY	Is set to 1 to specify the use of the old 2-page TTY handler. Set to 0 (default) to use the standard 3-page handler. The old handler has fewer features, but it is a page shorter. The parameters VT50, WIDTH, SCOPE, TAB and FILL have no effect when using the old handler.
LSBOT	Is set to 1 to list both the old 2-page and new 3-page handler. Set to 0(default) when only the handler selected by OLDTTY is to be listed.
TTFLD	Is set to specify the field of TTY Task; for example, 20 specifies field 2.
TTLOC	Is set to specify the location of TTY Task; for example, 3000 specifies a starting location of 3000.

Several equates are listed in the parameter file. They are useful for sending messages to TTY or LPT Tasks. These equates follow:

NOPACK=4000	Used if the output message is not 6-bit ASCII.
NOCRLF=2000	Used if the output message should not be followed by carriage return/line feed.
IND=1000	Used if OUTTXT points to the first word of the output text.
NOLINE=400	Used if the output is in character mode.
ASSGN=200	Used to assign the device handler for use by only this task.
KL8ALINE=100	Used with KL8-A support (see Section 4.13.2).

6.6.4 Monitor Console Routine Parameters (Chapter 5)

PARAMETER	MEANING
MCRSYS	Is set to 1 if the SYSTAT facility is desired for printing system status reports of existent system tasks.
MCRCLK	Is set to 0 if the clock functions are not wanted by the user.
MCRFLD	Is set to the field in which it is desired to locate MCR. MCRFLD = 30 places the MCR in field 3.

MCRPRT	Is set to the number of the partition into which the nonresident portion of the MCR will be swapped. This parameter makes the MCR nonresident; however, the first page of the MCR is always resident.
MCRORG	Is set to specify the starting location of the MCR. Default causes the MCR to load against the end of the field.
MCRCDV	Set to task name of task which is to be the MCR console device. Default is TTY.

6.6.5 OS/8 Support Task Parameters (Section 4.6)

PARAMETER	MEANING
OSFLDS	Is set to the number of fields allocated for OS/8. OSFLDS = 2 specifies two fields or 8K of memory for OS/8.
OSKBDV	Is set to the device code that selects desired OS/8 keyboard terminal. OSKBDV = 03 specifies the use of the console terminal keyboard for OS/8. Note: OS/8 requires its own dedicated terminal.
OSTTDV	Is set to the device code which selects desired OS/8 teleprinter. OSTTDV = 04 specifies the use of the console teleprinter for OS/8.
OSSYSD	Is set to select the OS/8 system device driver task. OSSYSD = DTA specifies DTA0 as the OS/8 system device.
OSFILL	Is set to the number of null characters that must follow a line feed character on the OS/8 terminal. OSFILL = 4 is specified when using a 2400-baud VT05 terminal. Set to 0 when using standard hard-copy terminals. Note: A terminal device handler does not have to be included for the OS/8 terminal device.

6.6.6 KL8-A Support Parameters (Section 4.13)

PARAMETER	MEANING
KL8ADV	Specifies the device code for the first KL8-A. Default is 40. If multiple KL8-A's are used, they should have consecutive device codes.
KL8ACT	Specifies the page for the KL8-A connect routine. Default is 7400 (if KL8A=1). The KL8-A connect routine must be located in field 0.

6.6.7 Line Printer Handler Parameters (Section 4.3)

PARAMETER	MEANING
LPTLOC	Specifies the starting location of the Line Printer Handler Task.
LPTFLD	Specifies the field of the Line Printer Task.

6.6.8 DECtape Handler Parameters (Section 4.4)

PARAMETER	MEANING
DTALOC	Specifies the starting location of the DECtape Handler Task.
DTAFLD	Specifies the field of the DECtape Handler Task.

6.6.9 EXIT Task (Section 4.14)

PARAMETER	MEANING
EXITFLD	Specifies the field of the EXIT Task.
EXITLOC	Specifies the starting location of the EXIT Task.

CHAPTER 7

NONRESIDENT TASKS

7.1 OVERVIEW

A nonresident task is a task or a portion of a task that resides on a mass storage device during the time the task is not runnable. The mass storage device is called the swap device. It can be any mass storage medium (e.g., an RK8 cartridge disk or an RX8 floppy disk). When a nonresident task becomes executable, the Executive posts a residency request. The Executive then runs a special task called the Swapper to load the nonresident or portion of the nonresident task into memory. The loading process is called a swap. The memory area into which or out of which the nonresident task is swapped is called a partition. A partition is a contiguous block of memory that is used for task execution; it is readable in a single mass storage call. The use of nonresident tasks permits several tasks to share the same areas of memory, optimizing the use of available memory.

Tasks can be either totally or partially nonresident. Very few tasks are totally nonresident; in most applications, the nonresident portion includes all, or nearly all, of the active locations. Active locations are those that contain executable instructions or data that are never accessed by other tasks. The resident portion of a task includes messages, event flags, buffers and similar passive registers that may be needed by other tasks.

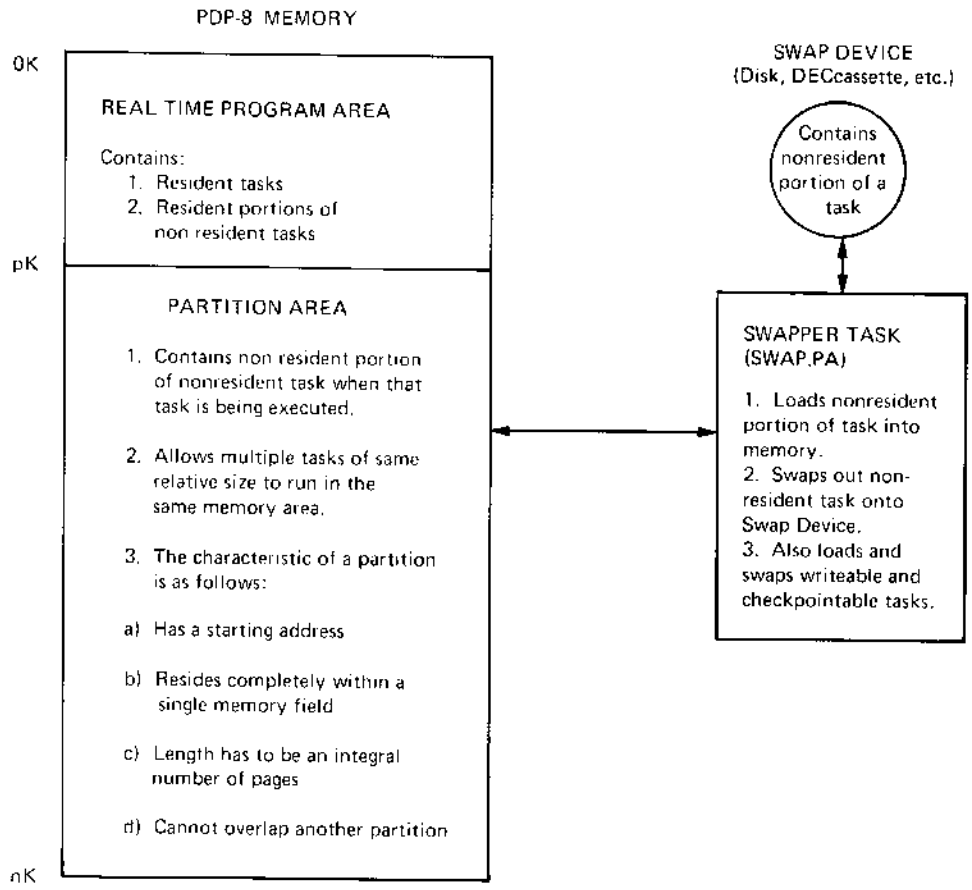


Figure 7-1 Nonresident Task Implementation

The process that swaps the nonresident portion of a task into memory is similar to the overlay capability found in OS/8 FORTRAN IV and related programs. However, swapping is much more powerful than overlaying. Every nonresident task has two properties that establish when and how it is swapped into or out of memory. These two properties are "writeability" and "checkpointability".

A task is made "writeable" if its nonresident portion must be written onto the swap device whenever it is swapped out to make room for another task. A writeable task is any task that is self-modifying - i.e., the task's code is changeable during task execution, and any task that must initialize before it can start. The writeability feature guarantees that the nonresident portion of a task is always up to date by refreshing the swap device image of the nonresident portion whenever the task is swapped out of memory.

A task is made "checkpointable" if it may be swapped out of memory automatically, without its consent, to make room for a higher priority task. A task that was checkpointed and swapped out of memory is swapped back automatically as soon as all higher priority tasks have relinquished the necessary memory space. Execution then continues at the point where it was interrupted, and the task is not aware that it was interrupted.

Some nonresident tasks have both writeable and checkpointable characteristics. Writeable tasks are utilized when a task must modify itself (for example, it includes JMS instructions or temporary locations). Tasks suitable for checkpointing are those that are 1) fairly long-running, and 2) are required only occasionally.

7.1.1 Writeable Tasks

A writeable task is one that includes code that is self-modifying, or code that must be initialized before execution. Before execution begins, all nonresident tasks must reside as core image files on the swap device. The core images are created by loading each nonresident task separately, and executing an OS/8 monitor SAVE command; this can be done under BATCH. When a task is executed, its nonresident portion is read from this core image file by the swapper. If a task is writeable, its nonresident portion will be written into the same file when it is swapped from memory. It is likely that a task, writeable at execution start-up, would flag itself not writeable at some later time when all initialization is completed. The nonresident portion of a task is writeable if it must be saved on a mass storage device (swap device) before overwriting it in memory with another task.

7.1.2 Checkpointable Tasks

Checkpointing is ideally suited for long-running tasks. The system can run short tasks in the same area, swap them out of memory, and swap the long running task back again.

7.1.3 Interaction Between Tasks

Resident tasks can interact with the resident portion of any nonresident task. Two nonresident tasks that occupy overlapping memory regions can interact with each other through their resident partitions. For example, if a nonresident task is executing, it sends a message and is then checkpointed. The message recipient can acknowledge the sender's message even though the sender is not totally resident. However, the message sent must be resident.

7.2 MEMORY PARTITIONS

The swappable portion of a nonresident task resides in a memory partition. This partition is simply a contiguous block of memory locations that is readable in one mass storage call. Every partition has the following characteristics:

- It is wholly contained in one memory field.
- It has a starting address.
- It has a length (size) that must divide evenly by 200 (octal) since OS/8 file structured devices read and write in one-half block (one page) increments.
- It normally begins at an address which is a multiple of 400.

The user can set the parameters for establishing the partition either in the parameter file or the source of the Swapper Task. This procedure is described in Section 7.4.

Partitions are mutually distinct, that is, one partition cannot overlap another. Any number of partitions can be defined. The n partitions are numbered in any order, from 0 to n-1, with partition number 0 being the first partition. They need not be adjacent.

Only one task can occupy a partition at any given time. The occupying task owns the partition until that task executes a "free" command (described in the next section), or (if the task is checkpointable) as long as higher priority tasks that share the partition remain nonexecutable.

It is most convenient if every partition begins on the first location of even-numbered pages, that is, the starting address is a multiple of 400 octal.

7.2.1 FREE Command

A free request can be appended to the function argument of some ER's (for example, SEND, RECEIVE, etc.) as follows:

```
CAL             /A CALL TO THE EXECUTIVE
COMMAND+FREE    /"COMMAND" IS ANY RTS/8 EXECUTIVE
                /REQUEST.  THE
                /ISSUING TASK IS TO BE SWAPPED OUT OF
                /MEMORY IF SOME OTHER TASK
                /BECOMES EXECUTABLE AND REQUIRES THE
                /PARTITION BEFORE THE FREEING TASK
                /CANCELS THE FREE REQUEST.
```

A free request must be combined with some other executive request. If the sequence:

```
CAL
FREE
```

is issued, it will be interpreted as:

```
CAL
SEND+FREE
```

Only nonresident tasks may issue free requests; other free requests are ignored. A task may free its own partition, but never the partition of another task.

A task normally frees its partition whenever it must wait for an event. If any other task has a pending request for the partition, it is swapped into the free partition immediately. If there is no pending request for the partition, the freeing task continues to wait in the free partition until some other occupant requests residency. In either case, once the freeing task becomes executable again, it must compete for the partition along with any other executable occupants. The freeing task may become executable before any other task requests residency in the partition. In this case, the free command is cancelled, and the freeing task retains possession of the partition. No read or write operation is necessary to effect this swap in this case. By freeing the partition whenever the occupying task (which may or may not be writeable) must wait for an event, the programmer is assured that the partition contains a running (i.e.,

nonwaiting) task whenever possible. If there are no writeable tasks in the partition, no swap device I/O is involved in freeing the partition.

7.3 NONRESIDENT TASK INITIALIZATION

The following procedure is recommended to implement any RTS/8 system containing nonresident tasks.

1. Code and debug each nonresident task as a resident task. During the debugging, load the task being debugged and only the tasks required for execution of the task to be debugged. Once the task executes correctly while resident, make it nonresident. Making a task nonresident is described in the sections that follow.

2. On a listing, mark the nonresident portion of each nonresident task. Determine the size of each task's nonresident portion. Then design the partitioning scheme and allocate nonresident tasks to memory partitions by modifying the parameter file as described in Section 7.4.

3. Re-origin each nonresident task so that its nonresident portion lies within its partition. If necessary, ensure that resident portions of nonresident tasks do not overlap into another partition. BITMAP, an OS/8 utility program, is useful for this since it allows the user to determine if two or more tasks are erroneously being loaded into the same memory section. The use of BITMAP is described in Section 6.3. A detailed description of BITMAP is given in Chapter 2 of the OS/8 Handbook. Execute the task as a resident task once more to make sure it does not contain location dependent code. Also, if any memory partition begins on an odd numbered page, temporarily relocate each task that resides in that partition. An example of starting a nonresident task at an arbitrary boundary is given in Section 9.3.

7.3.1 Parameters for Nonresident Tasks

Several assembly parameters must be initialized when employing nonresident tasks. Five of the parameters are located in the parameter file (PARAM.PA) and three must be included in the nonresident task itself.

The parameter file (PARAM.PA) contains five parameters (PARTNS, CHECKP, SWAPPER, SYS and SUNIT) which must be initialized when employing nonresident tasking. These parameters are defined as follows:

PARTNS is set to the number of memory partitions defined. PARTNS is set to zero in the parameter file to indicate that no memory partitions are defined in the system.

CHECKP is set to 1 if any nonresident task is checkpointable.

SWAPPER is the nonresident task swapper task; it must be assigned a task number that is of higher priority than the task it swaps, that is, a number lower in value than that of any nonresident task in the system.

SYS is set to designate the swap device driver task; for example, SYS = RK8 specifies the disk driver task.

SUNIT is set to specify the swap device physical drive unit; for example, SUNIT = 0 specifies the disk cartridge drive.

Every nonresident task source must include the following three parameters:

```
PARTNO= n, where n is the task's partition number (starting at 0)
CPABLE= 0, if the task is not checkpointable
        1, if the task is checkpointable
WRITE=  0, if the task is not writeable
        1, if the task is writeable
```

Failure to initialize these parameters correctly causes the program to execute unpredictably. The presence of parameter PARTNO identifies a task as a nonresident task. Hence, the variable name PARTNO should not be used except in nonresident tasks.

A nonresident task is not appreciably different from a resident task. However, buffers should not be in nonresident portions of a task since the buffers will be out of memory when that task is not being executed. There are no special coding restrictions. The nonresident portion is always present in memory while a nonresident task is executing. It is generally safe to assume that the nonresident portion is never present when the task is not executing. There is a slight structural difference between resident and nonresident tasks; nonresident tasks have clearly defined resident and nonresident portions that cannot be intermixed.

7.3.2 Assembling Nonresident Tasks

Each nonresident task is assembled separately. The nonresident task must include the parameter TASK, but never the parameters TASK2 and TASK3 (see Section 6.1.5, Task Setup).

The swap device file that contains the nonresident portion of the nonresident task requires special treatment. This file must contain the first word of the nonresident portion in the first location of relative block 1, the second word of the nonresident portion in the second location, and so on. This file must be a core image; however, the OS/8 monitor requires that the first location of any core image section load into a memory address that divides evenly by 400(octal). If the lowest address in the partition also divides evenly by 400(octal), this condition is met. There is no problem because every task can be assembled in the partition directly.

Modify the parameter file (PARAM.PA) to establish the desired parameters. Then assemble the nonresident task and the parameter file together.

7.3.3 Creating the SAVE Image File

Create a SAVE image file (nonresident disk image) from the binary image file as follows:

```
.R ABSLDR
*TASKX.BN$
.SAVE DEV TASKX.SV N1-N2
```

For each task, where DEV is the swap device, N1 is the lowest address in the partition, and N2 is the highest address in the partition. The resulting core images contain only the nonresident portions of each task and meet all the requirements previously outlined. However, the first block of the core image (relative block 0) is the core control block, which is not used by the swapper. As stated previously, it is strongly recommended that the partitions begin at a location that is a multiple of 400(octal) since the OS/8 SAVE command only saves areas starting at 400(octal) boundaries.

NOTE

SAVE image files can be constructed under OS/8 BATCH. Also, the OS/8 CCL command LOAD may be used to create the SAVE image.

7.4 PARAMETER INITIALIZATION FOR PARTITIONS

The user must initialize certain parameters to define the partitioning scheme. They can be set in either PARAM.PA or SWAP.PA. The following three variables are required for each of the partitions:

```
MFLDnn=  MEMORY FIELD OF PARTITION N
ADDRnn=  MEMORY ADDRESS OF PARTITION N
SIZEnn=  SIZE OF PARTITION N, SPECIFIED IN PAGES
```

The user can set up the parameter file to accept eight sets of partition parameters. When more than eight partitions are defined, the parameter table must be extended according to instructions contained in source. Since adding extra partitions does not increase system overhead, it is best to have as many partitions as possible. This minimizes the number of tasks that must share a partition. Each partition should have at least two occupants; otherwise, there is no reason for making the task nonresident. The partitioning scheme can also be defined by initializing the required parameters within the swapper source.

7.4.1 General Information

The entire partition table appears on the swapper PAL or CREF output listing. The user should check it carefully to ensure that all partition parameters (MFLDnn, ADDRnn and SIZEnn for each partition) were defined correctly. The parameter file generates the residency table entry for each nonresident task and it appears on the PAL or CREF output listing for that task. The user should examine word 1 of this entry to verify that the nonresident task parameters PARTNO, CPABLE, and WRITE were initialized correctly. Word 2 of the residency table entry will be zero because the task's block address on the swap device is unknown at assembly time. This location is initialized by the OS/8 command decoder shortly after program startup, and it can be examined anytime thereafter. In the usual case, where the file is a core image, this location should contain M+1, where M is the block address returned by the following command:

```
.DIR DEV:TASKX.SV/B
```

TASKX.SV is the core image on the swap device containing the nonresident portion of the task.

7.5 NONRESIDENT TASK IMPLEMENTATION

Perform the following procedure to implement nonresident tasks.

1. Assemble every task that will be included in the program as described in Section 7.3.2. Obtain PAL or CREF listings and bitmaps. Finally, obtain a bitmap of the entire system and verify that memory is allocated correctly. On the bitmap, 3s are legal only within partitions.

2. Create the SAVE image file as described in Section 7.3.3.

3. Load the binaries of the RTS/8 executive and each task, including nonresident tasks. Always load the executive first. Nonresident tasks are included in this operation in order to load their resident portions and Executive table entries. Either save the loaded program as a core image file or start it from the keyboard.

4. Start the real-time program using the monitor R command. This calls the OS/8 command decoder, which types an asterisk on the console terminal. At this time, initialize the block address of each nonresident task core image as follows:

```
*DEV:TASKX.SV=N
```

   In this command line, DEV is the swap device and TASKX.SV is the core image file containing the nonresident portion of task N. Repeat this procedure for each nonresident task, one task per line, and terminate the last line with an ALTMODE. This procedure automatically initializes and starts the real-time program.

   This initialization procedure may be executed automatically under OS/8 BATCH control.

   Example:

```
$JOB SYSTEM
/RUN "SYSTEM" AND INSTALL
/NONRESIDENT TASKS.
.RUN SYS SYSTEM
*TASK1=35  /TASK 35 IS "TASK1" SAVE IMAGE
*TASK2=36$ /TASK 36 IS "TASK2" SAVE IMAGE
$END
```

   Submitting SYSTEM.BI runs RTS/8 user system SYSTEM.SV, installing its two nonresident tasks.

5. Debug the entire program. Accomplish this by selectively placing HLT instructions (preceded by IOF, NOP, if appropriate) and examining the memory once the HLT has been executed. Use the MCR and/or the console switch register to place and remove HLT instructions and to modify all permanently resident areas. When using MCR, remember that MCR output represents a snapshot of memory at some undetermined point in time that is long past by the time the MCR has output to the terminal.

CHAPTER 8

DEMONSTRATION PROGRAM

This chapter contains a demonstration of RTS/8 with nonresident tasks executing in the foreground. Included is a listing of the modified parameter file (PARAM.PA). In addition, there is an example of nonresident tasks (Tasks NR20 and NR22) and the assembly and load instructions required for implementing the demonstration program.

8.1 MODIFIED PARAMETER FILE (PARAM.PA)

```
/2.8 PARAMETERS FOR RTS/8 TASKS          V2B+DECNET
        LSTFLG=0        /CHANGE 0 TO 1 TO PREVENT LISTING PARAM
        XLIST LSTFLG
/
/
/
/
/
/
/
/
/
/COPYRIGHT (C) 1974,1975,1976 BY DIGITAL EQUIPMENT CORPORATION
/
/
/
/
/
/
/
/
/
/
/THE INFORMATION IN THIS DOCUMENT IS SUBJECT TO CHANGE WITHOUT NOTICE
/AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT
/CORPORATION. DIGITAL EQUIPMENT CORPORATION ASSUMES NO RESPONSIBILITY
/FOR ANY ERRORS THAT MAY APPEAR IN THIS DOCUMENT.
/
/THE SOFTWARE DESCRIBED IN THIS DOCUMENT IS FURNISHED TO THE PURCHASER
/UNDER A LICENSE FOR USE ON A SINGLE COMPUTER SYSTEM AND CAN BE COPIED
/(WITH INCLUSION OF DIGITAL'S COPYRIGHT NOTICE) ONLY FOR USE IN SUCH
/SYSTEM, EXCEPT AS MAY OTHERWISE BE PROVIDED IN WRITING BY DIGITAL.
/
/DIGITAL EQUIPMENT CORPORATION ASSUMES NO RESPONSIBILITY FOR THE USE
/OR RELIABILITY OF ITS SOFTWARE ON EQUIPMENT THAT IS NOT SUPPLIED BY
/DIGITAL.
/
/
/
/
/
/
/
/
/
/
/
```

```
/RTS8 V2 EXEC PARAMETERS - EDITED BY USER
PDP8E=1
PDP12=0
EAE=0
PWRFAL=0
KL8A=0              /NUMBER OF KL8A'S IN USE
                    /I.E. 1 IF  ONE KL8A (UP TO 4 LINES)
HGHFLD=30
        IFDEF   CUR     <
        IFNZRO  HGHFLD-CUR&4000 <CURBIG,,ERROR,,>       /FLAG WARNING IF UNDEFINED FIELD SEEN
NTASKS=23
CHECKP=1
PARTNS=1
                    /(THE N PARTITIONS ARE NUMBERED FROM 0 TO N-1)

/COMMON TASK NUMBERS - EDITED BY USER
/IT IS ADVISABLE TO DEFINE ALL TASKS HERE,  NAMES GIVEN BELOW
/ARE USED BY SOME SYSTEM TASKS AND SHOULD REMAIN COMMENTED OUT
/IF THE CORRESPONDING TASK IS NOT INCLUDED IN THE SYSTEM

/PWRF=              /POWER FAIL HANDLING TASK
CLOCK=1
SWAPPER=16
TTY=2
LPT=3
MCR=4
/DTA=               /DECTAPE DRIVER TASK
/LTA=               /LINCTAPE DRIVER TASK
RK8=7
/RF08=              /RF08 DISK DRIVER TASK
/DF32=              /DF32 DISK DRIVER TASK
/CSA=               /CASSETTE DRIVER TASK
/CSAF=              /CASSETTE FILE SUPPORT TASK
/UDC=               /UNIVERSAL DIGITAL CONTROLLER TASK
/RX8A=              /FIRST FLOPPY CONTROLLER
/RX8B=              /SECOND FLOPPY CONTROLLER
/RX8C=              /THIRD FLOPPY CONTROLLER
/RX8D=              /FOURTH FLOPPY CONTROLLER
OS8=NTASKS
OS8F=15
NR20=20
NR22=22
/DDCMP=             /DDCMP TASK FOR DECNET
/NSP=               /NETWORK SERVICES PROTOCOL TASK
/NIP=               /NETWORK INFORMATION PROGRAM
/TLK=               /NETWORK TERMINAL COMMUNICATIONS TASK TRANSMITTER
/LSN=               /NETWORK TERMINAL COMMUNICATIONS TASK RECEIVER
/NULL8A=            /NULL JOB FOR PDP-8/A
/EXIT=              /EXIT TASK
/DKC8A=             /AUXILIARY DKC8A HANDLER

/SOFTWARE PARAMETERS - EDITED BY USER

        XLIST   1
        IFDEF   OS8     <
        XLIST   LSTFLG
OSFLDS=2
OSTTDV=31
OSKBDV=30
                    /DEFAULT IS OSTTDV=1
OSSYSD=RK8
OSFILL=0
                    /(EG 4 FOR 2400 BAUD VT05)
/OS8ORG=            /ORIGIN (IN FIELD 0)
        XLIST   1>
        IFDEF   MCR     <
        XLIST   LSTFLG
/MCRCLK=                    /0 IF MCR TOO FACILITIES TO BE OMITTED (DEFAULT = 1)
MCRSYS=1
                            /1 IF DESIRED (DEFAULT)
/MCRFLD=                    /FIELD OF MCR
/MCRORG=                    /LOCATION OF MCR (DEFAULT IS END OF FIELD)
/MCRPRT=                    /PARTITION NUMBER OF MCR (IF NON-RESIDENT)
/MCRCDV=                    /CONSOLE TO BE USED BY MCR, E.G, TTY
                            /DEFAULT IS TTY
        XLIST   1>
        IFDEF   CLOCK   <
        XLIST   LSTFLG
CLKTYP=0
CLKQLN=20
                    /MAY BE CHANGED BY USER
        DECIMAL
HERTZ=1000
```

```
SHERTZ=1
        IFNZRO  CLKTYP&1 <HERTZ=1750>   /FORCE DKBEP,KW12 TO 1 KHZ
        OCTAL
        XLIST   1>

        IFDEF   LPT     <
/LPTLOC=
/LPTFLD=
        >
        IFDEF   DTA     <
/DTALOC=
/DTAFLD=
        >
        TBLLST= 0       /SET TO 'LSTFLG' IF YOU DON'T DESIRE
                        /TO SEE TABLES WHEN PARAMETER FILE IS
                        /NOT LISTED.

/SYSTEM LOCATIONS:

MSGTBL= 1200-2                  /TASK MESSAGE TABLE
TSTABL= NTASKS+2^2+MSGTBL-4     /TASK STATE TABLE = HOLDS
                                /TASK LINK,UM,OF,IF,PC,AC,MQ
TFTABL= NTASKS+2^4+TSTABL-1     /TASK FLAGS TABLE = HOLDS
                                /TASK STATUS FLAGS
        XLIST   1
        IFDEF   SWAPPER <
        XLIST   LSTFLG
SYS=RK8
SUNIT=0
        IFNDEF  SUNIT   <SUNIT= 0>      /DEFAULT SWAP UNIT IS 0

        FIELD 0
/
/PARTITION TABLE (PARTBL) ENTRIES:
/MUST BE INITIALIZED BY USER AS EXPLAINED IN THE COMMENTS
/DON'T FORGET TO REMOVE LEADING "/" FROM LINES USED
/
RESTBL= TFTABL+NTASKS+2                 /RESIDENCY TABLE
PARTBL= NTASKS-SWAPPER^2+RESTBL+3&7774  /PARTITION TABLE

        *PARTBL
        XLIST   1
        IFNZRO  PARTNS          <
        XLIST   TBLLST
MFLD00=1
ADDR00=400
SIZE00=1
        SIZE00^10+MFLD00^10+4000
        ADDR00
        ZBLOCK  2
        XLIST
        IFNZRO  PARTNS-1        <
        XLIST   TBLLST
MFLD01=                 /MEMORY FIELD OF PARTITION #1
ADDR01=                 /LOWEST ADDRESS IN PARTITION #1
SIZE01=                 /SIZE OF PARTITION #1 (CORE PAGES)
        SIZE01^10+MFLD01^10+4000
        ADDR01
        ZBLOCK  2
        XLIST
        IFNZRO  PARTNS-2        <
        XLIST   TBLLST
MFLD02=                 /MEMORY FIELD OF PARTITION #2
ADDR02=                 /LOWEST ADDRESS IN PARTITION #2
SIZE02=                 /SIZE OF PARTITION #2
        SIZE02^10+MFLD02^10+4000
        ADDR02
        ZBLOCK  2
        XLIST
        IFNZRO  PARTNS-3        <
        XLIST   TBLLST
MFLD03=         /PARTITION #3
ADDR03=
SIZE03=
        SIZE03^10+MFLD03^10+4000
        ADDR03
        ZBLOCK  2
        XLIST
        IFNZRO  PARTNS-4        <
        XLIST   TBLLST
MFLD04=         /PARTITION #4
ADDR04=
SIZE04=
        SIZE04^10+MFLD04^10+4000
        ADDR04
```

```
        ZBLOCK  2
        XLIST
        IFNZRO  PARTNS-5        <
        XLIST   TBLLST
MFLD05= 
        /PARTITION #5
ADDR05=
SIZE05=

        SIZE05^10+MFLD05^10+4000
        ADDR05
        ZBLOCK  2
        XLIST
        IFNZRO  PARTNS-6        <
        XLIST   TBLLST
MFLD06= 
        /PARTITION #6
ADDR06=
SIZE06=

        SIZE06^10+MFLD06^10+4000
        ADDR06
        ZBLOCK  2
        XLIST
        IFNZRO  PARTNS-7        <
        XLIST   TBLLST
MFLD07= 
        /PARTITION #7
ADDR07=
SIZE07=

        SIZE07^10+MFLD07^10+4000
        ADDR07
        ZBLOCK  2
        XLIST 1>>>>>>>>
/
/ADDITIONAL PARTITIONS MAY BE DEFINED BY THE USER AS SHOWN ABOVE
/FURTHERMORE, THE PARTITION TABLE MAY RESIDE ANYWHERE IN FIELD ZERO
/
        PRTEND=.        /NOTE END OF PARTITION TABLE

        XLIST   1>
        IFDEF   TTY     <
        XLIST   LSTFLG
/       TTDEV=          /PRINTER DEVICE CODE - DEFAULT IS 4
/       KBDEV=          /KEYBOARD DEVICE CODE - DEFAULT IS TTDEV+1
/       CONSOL=         /1 MEANS CONSOLE TTY (DEFAULT)
/       VT50=           /1 ENABLES CTRL/S AND CTRL/Q
/       SCOPE=          /1 MEANS TTY CAN DO A BACKSPACE
/       FILL=           /NUMBER OF FILL CHARACTERS, I.E. 4
/       WIDTH=          /TTY LINE WIDTH (0 MEANS INFINITE), DEFAULT IS 120
/       TAB=            /1 IF TTY HAS HARDWARE TABS
/       OLDTTY=         /1 TO USE OLD 2-PAGE TTY HANDLER
/       LSBOT=          /1 LISTS BOTH HANDLERS (DEFAULT 0)
/       TTFLD=          /FIELD OF TTY TASK (TIMES 10)
/       TTLOC=          /LOCATION OF TTY TASK

        XLIST   1>
        IFNZRO  KL8A    <
        XLIST   LSTFLG
/KL8ADV=                /KL8A DEVICE CODE - DEFAULT IS 40
/KL8ACT=                /KL8A CONNECT ROUTINE PAGE - DEFAULT IS 7400
        XLIST   1>
        IFDEF   EXIT    <
        XLIST   LSTFLG
/EXITFLD=       /FIELD OF EXIT TASK (TIMES 10)
/EXITLOC=       /LOCATION OF EXIT TASK
        XLIST   1>
        XLIST   LSTFLG

        IFNDEF  PDP8E   <PDP8E=1>
        IFNDEF  PDP12   <PDP12=0>
        IFNDEF  EAE     <EAE=0>
        IFNDEF  PWRFAL  <PWRFAL=0>
        IFNDEF  KL8A    <KL8A=0>

        XLIST   1
        IFDEF   NSP <
        XLIST   LSTFLG
MAXCCB=         /NUMBER OF LOGICAL CHANNELS (CCB'S) BEING USED
                /E.G. 3 FOR 3 CHANNELS
                /THESE ARE NUMBERED 1,2,3
MAXNOD=         /NUMBER OF NODE NAMES IN NODE TABLE
NSPFLD=         /FIELD OF NSP TASK AND MOST NETWORK TABLES (E.G. 30)
                /(TABLES INCLUDE CCBTAB, LNKTAB, NODTAB AND NETTAB)
NSPLOC= 2600    /ORIGIN OF NSP TASK.  MUST BE .LE. 3200
                /THE DEFAULT IS CURRENTLY 3200
NODNUM=         /NODE NUMBER OF THIS NODE
```

```
        /IMPORTANT RELATIVE ORIGINS WITHIN NETWORKS TASKS

        DRLXIT= NSPLOC+4400     /ADDRESS OF AST DE-QUEUER
        CCBTAB= DRLXIT+200      /ADDRESS OF CCB TABLE
        NODTAB= CCBTAB+100      /ADDRESS OF NODE TABLE
        NETTAB= NODTAB+60       /ADDRESS OF NETWORK 'INFORMATION' TABLE

/THE DEFAULT NETWORKS TASKS USE CORE AS FOLLOWS:

/DDCMP: PAGE 0, 0200-3577       (1 LINE, 2 PAGE NODE POOL)
/NSP:   PAGE 0, 3400-7577


/NETWORK TASKS USE PAGE 0 AS FOLLOWS:

/DDCFLD:        10-12, 30-77
/NSPFLD:        13-17, 77-177

/NODE TABLE ENTRIES

/EACH ENTRY HAS THE FORM
/WORDS 1-3      NODE NAME (6-BIT, 0-PADDED)
/WORD 4         LINE NUMBER
/WORD 5         BIT 0=1 IF ADJACENT NODE
/               BITS 4-11 CONTAIN NODE NUMBER

        IFDEF TASK <    IFZERO TASK-NSP <
        FIELD NSPFLD*10
        *NODTAB

NODTAB, TEXT    /NAME/
        0               /LINE NUMBER
        0               /NODE NUMBER
        *NETTAB+4
        NODNUM          /OUR NODE NUMBER
        TEXT    /NAME/  /OUR NODE NAME
        FIELD 0
        >>
        >

        XLIST   1
        IFDEF   DDCMP   <
        XLIST   LSTFLG
MAXLIN= 3               /NUMBER OF PHYSICAL LINES BEING USED
                        /E.G. 3 FOR 3 LINES
                        /THESE ARE NUMBERED 0,1,2
MAXPKT= 24              /SET TO NUMBER OF NODE POOL PACKETS TO ALLOW
                        /THE NODE POOL EXISTS AT THE END OF DDCMP
                        /JUST BEFORE THE LCB TABLE (SIMILAR TO THE CLOCK QUEUE)
                        /EACH PACKET REQUIRES 14 WORDS OCTAL. (ABOUT 10. PER PAGE)
                        /(THE DEFAULT REQUIRES 2 PAGES CORE)
/KGBE=                  /SET TO IOT SKELETON IF KGBE IS PRESENT (E.G. 6110)
DDCFLD=         10      /FIELD OF DDCMP TASK,LCBTAB AND 'NODE POOL' (E.G. 20)
                        /THIS FIELD MUST BE DIFFERENT FROM NSPFLD
DDCLOC= 0200            /ORIGIN OF DDCMP TASK
                        /THE ABOVE MUST BE BELOW 5000-SIZE OF NODE POOL AND LCBTAB
                        /THE DEFAULT IS CURRENTLY 200

LCBSIZ= 32              /GLOBAL DEFINITION OF LCB SIZE (DO NOT ALTER)
PKSIZE= 14              /GLOBAL DEFINITION OF PACKET SIZE (DO NOT ALTER)

        DDCFNC= DDCLOC          /ADDRESS OF DDCMP 'FUNCTION CALL' ROUTINE
        HEADPK= DDCLOC+3020     /ADDRESS OF START OF PACKET FREELIST
        LCBTAB= MAXPKT*PKSIZE+HEADPK    /ADDRESS OF LINE CONTROL BLOCK TABLE

        /IMPORTANT NETWORKS PAGE 0 GLOBALS

        DDCEF=  46      /DDCMP I/O EVENT FLAG
        FREHD=  47      /LOCATION OF I/O PACKET FREELIST HEAD
        DDCTL=  50      /POINTS TO TAIL OF DDCMP INPUT QUEUE
        DDCHD=  51      /POINTS TO HEAD OF DDCMP INPUT QUEUE
        ATNINP= 52      /POINTS TO TRANSMIT COMPLETE RING BUFFER
        OHDR=   53      /LOCATION OF HEADER BUFFER FOR TRANSMITS
        OCRCL=  63      /HEADER CRC FOR TRANSMITS
        OOCRCL= 65      /DATA CRC FOR TRANSMITS
        OUTCDF= 67      /DATA DESCRIPTOR FOR TRANSMITS
        DDCUSR= NSP     /DEFAULT USER OF DDCMP TASK
        XLIST   1>

        XLIST   1
        IFDEF   NIP     <
        XLIST   LSTFLG
/NIPFLD=                /FIELD OF NIP (TIMES 10)
/NIPLOC=                /LOCATION OF NIP
/NIPART=                /PARTITION FOR NIP
/SKIMP=                 /SET TO 1 TO GET SHORT NIP
```

```
/NIPLOG=                /DEVICE NIP OUTPUTS TO
                        /DEFAULT IS LPT IF IT EXISTS (OTHERWISE TTY)
/NIPRES=                /LOCATION FOR RESIDENT PORTION OF NIP
                        /REQUIRED ONLY IF NIPART DEFINED
                        /DEFAULT IS NIPLOC=200
        XLIST   1       >

        IFDEF   TLK     <
        XLIST   LSTFLG
/TLKFLD=                /FIELD OF TLK TASK
/TLKLOC=                /START OF TLK TASK
TLKCHN=         /CCB CHANNL TO ASSIGN TO TLK TASK
        XLIST   1       >
        IFDEF   LSN     <
        XLIST   LSTFLG
/LSNFLD=                /FIELD OF LSN TASK (TIMES 10)
/LSNLOC=                /START OF TLK TASK
LSNCHN=         /CCB CHANNL TO ASSIGN TO LSN TASK
        XLIST   1       >
        XLIST   LSTFLG

/EQUIVALENCES:

AC7776= CLL STA RAL
AC7775= CLL STA RTL
AC4000= CLA STL RAR
AC3777= CLL STA RAR
AC2000= CLA STL RTR
AC0002= CLA STL RTL

/MONITOR CALL VALUES:

CAL=    JMS     20      /CALL THE EXECUTIVE
POSTDS= JMP I   24      /DISMISS AN INTERRUPT
WAITM=  JMS I   25      /WAIT FOR MULTIPLE EVENTS

                /NOTE:  "**" MEANS CRITICAL VALUE MAY NOT
                /BE CHANGED WITHOUT MODIFYING SYSTEM CODE!!
SEND=   0       /SEND MESSAGE
RECEIV= 1       /RECEIVE MESSAGE
WAITE=  2       /WAIT FOR EVENT FLAG
RUN=    3       /CONTINUE TASK EXECUTION
SUSPND= 4       /SUSPEND TASK EXECUTION
POST=   5       /POST AN EVENT FLAG
SKPINS= 6       /INSERT CODE INTO INTERRUPT SKIP CHAIN
DERAIL= 7       /INITIATE END-ACTION
BLKARG= 10      /BLOCK TASK FOR REASON SPECIFIED IN ARG
SENDW=  11      /SEND MESSAGE AND WAIT
UNBARG= 12      /UNBLOCK TASK FOR REASON SPECIFIED IN ARG
RESCHD= 13      /FORCE A RESCHEDULE
WAITX=  14      /WAIT FOR EXACTLY THIS EVENT FLAG
FREE=   4000    /**FREE PARTITION

        XLIST   1
        IFDEF DOC       <
        XLIST   LSTFLG
        AD=0;DD=1;DI=2;GC=3;EC=4;RC=5
        DC=6;ECT=7;CS=10;DCT=11;AI=12
        XLIST   1>
        XLIST   LSTFLG

/TASK STATUS FLAGS:

NONRWT= 4000    /**NONRESIDENT TASK WAIT
EFWT=   2000    /EVENT FLAG WAIT
RUNWT=  1000    /SCHEDULE WAIT
SWPWT=  0400    /**SWAPPER WAIT
EORMWT= 0200    /EVENT FLAG OR MESSAGE WAIT
USERWT= 0100    /USER SPECIFIED WAIT
ENABWT= 0040    /ENABLE WAIT
MSGWT=  0020    /MESSAGE WAIT
NETWT=  0010    /NETWORK WAIT  (RESERVED FOR POSSIBLE FUTURE USE)
DNEWT=  0001    /**DOES NOT EXIST WAIT

        IFNZRO  KL8A    <IFNDEF KL8ACT  <
        KLUD=   KL8A-1/3^200
        KL8ACT= 7400+KLUD>>

TSWFLG= 35      /TASK SW INHIBIT FLAG   IN FIELD 0
TODL=   36      /LOW ORDER TIME OF DAY  IN FIELD 0
TODH=   37      /HIGH ORDER TIME OF DAY IN FIELD 0
DATE=   40      /DATE IN DSS FORMAT     IN FIELD 0
MCREF=  41      /MCR START EVENT FLAG   IN FIELD 0
```

```
/SOME USEFUL EQUATES FOR TTY AND LPT MESSAGES:

        NOPACK=4000     /TEXT IS NOT PACKED IN 6-BIT
        NOCRLF=2000     /OUTPUT SHOULD NOT BE FOLLOWED BY CR/LF
        IND=1000        /OUTTXT PTS TO FIRST WORD OF TEXT
        NOLINE=400      /INPUT IS IN CHARACTER MODE
        ASSGN=200       /ASSIGNS DEVICE
        KL8ALINE=100    /USED TO SPECIFY A LINE OF A KL8A

        XLIST   1
        IFDEF   CLOCK   <
        XLIST   LSTFLG
        SOME USEFUL EQUATES FOR STANDARD CLOCK MESSAGES:

        MARKTIME=       0       /POST EVENT FLAG AFTER SPECIFIED INTERVAL
        SCHEDULE=       1000    /RUN TASK AFTER SPECIFIED INTERVAL
        TIMOUT=         2000    /DERAIL TASK AFTER SPECIFIED INTERVAL
        PERIODICALLY=   2000    /USED AS MODIFIER TO "SCHEDULE"
                                /RE-QUEUES RUN REQUEST AFTER SPECIFIED INTERVAL
                                /E.G.     "SCHEDULE FOO PERIODICALLY"
        CANCEL=         7000    /DELETE ALL REQUESTS FROM SPECIFIED TASK FROM QUEUE
        XLIST   1>

        XLIST   1       /FORCE LISTING OFF
        IFDEF   TASK    <
        XLIST

/TASK TABLE SETUP - "TASK", "CUR","INIWT", AND "START"
/MUST BE DEFINED BY TASK:

        IFNDEF  INIWT   <INIWT=0>
        IFNDEF  INIWT2  <INIWT2=0>
        IFNDEF  INIWT3  <INIWT3=0>

        *TASK^2+MSGTBL
        ZBLOCK  2       /MESSAGE BUFFER INITIALLY CLEAR
        *TASK^4+TSTABL
        CUR%10+CUR      /INITIAL FLAGS
        START
        0               /INITIAL AC 0
        XLIST;  IFDEF   VERS    <
        XLIST
        VERS            /INITIAL MQ
        XLIST   >
        XLIST
        *TASK+TFTABL
        INIWT
        XLIST
        >

        IFDEF   TASK2   <
        XLIST
        *TASK2^2+MSGTBL
        ZBLOCK  2       /MESSAGE BUFFER INITIALLY CLEAR
        *TASK2^4+TSTABL
        CUR2%10+CUR2    /INITIAL FLAGS2
        START2
        0               /INITIAL AC 0
        XLIST;  IFDEF   VERS2   <
        XLIST
        VERS2           /INITIAL MQ
        XLIST   >
        XLIST
        *TASK2+TFTABL
        INIWT2
        XLIST
        >

        IFDEF   TASK3   <
        XLIST
        *TASK3^2+MSGTBL
        ZBLOCK  2       /MESSAGE BUFFER INITIALLY CLEAR
        *TASK3^4+TSTABL
        CUR3%10+CUR3    /INITIAL FLAGS3
        START3
        0               /INITIAL AC 0
        XLIST;  IFDEF   VERS3   <
        XLIST
        VERS3           /INITIAL MQ
        XLIST   >
        XLIST
        *TASK3+TFTABL
        INIWT3
        XLIST
        >
```

```
        IFDEF   TASK    <
        IFDEF   PARTNO  <
        XLIST

/RESIDENCY TABLE (RESTBL) ENTRY:
/INITIALIZED FOR NONRESIDENT TASKS ONLY

        *TASK=SWAPPER-1^2+RESTBL
        PARTNO^4+PARTBL+CPABLE+CPABLE+WRITE
        XLIST
        IFNDEF  SWAPPER <NOSWAP,␣ERROR␣>        /SWAPPER MISSING
        IFNZRO  TASK-SWAPPER&4000       <SWPRIO,␣ERROR␣>/NON-RESIDENT TASK
                                                /HAS PRIORITY HIGHER THAN SWAPPER
        >>

        IFDEF   PARTNO  <
        IFNDEF  TASK    <NOTASK,␣ERROR␣>        /PARTITION BUT NO TASK
        IFNDEF  SWAPPER <NOSWAP,␣ERROR␣>        /PARTITION BUT NO SWAPPER
        IFNDEF  PARTNS  <NOPART,␣ERROR␣>        /MISSING PARTITIONS
        IFZERO  PARTNO-PARTNS&4000      <PRTERR,␣ERROR␣>        /PARTNO.GE.PARTNS
        >
        XLIST   0
```

8.2 NONRESIDENT TASK LISTINGS

The following are listings of nonresident tasks (NR20 and NR22):

8.2.1 Nonresident Task NR20

```
TASK=20

        0020  TASK=20
        0020          TASK=20
        0400          START=400
        0001          WRITE=1
        0001          CPABLE=1
        0010          CUR=10
        5000          INIWT=RUNWT+NONRWT
        0000          PARTNO=0
        0001          FIELD CUR%10
        0400          *400
10400   4020          CAL
10401   4011          SENDW+FREE
10402   0001          CLOCK
10403   0600          SLPMSG
10404   4020          CAL
10405   4011          SENDW+FREE
10406   0003          LPT
10407   0606          LPTMSG
10410   5200          JMP START
        0600          *600
10600   0000  SLPMSG, ZBLOCK 3
10603   0000          0
10604   0000          0!SHERTZ
10605   0001
10606   0000  LPTMSG, ZBLOCK 3
10611   0000          0
10612   0000          0
10613   2401          TEXT /TASK 20 RUNNING/
10614   2313
10615   4062
10616   6040
10617   2225
10620   1616
10621   1116
10622   0700
```

8.2.2 Nonresident Task NR22

```
TASK=22

        0022  TASK=22
        0022          TASK=22
        0400          START=400
        0001          WRITE=1
        0001          CPABLE=1
        0010          CUR=10
        5000          INIWT=RUNWT+NONRWT
        0000          PARTNO=0
        0001          FIELD CUR%10
        0400          *400
10400   4020          CAL
10401   4011          SENDW+FREE
10402   0001          CLOCK
10403   0630          SLPMSG
10404   4020          CAL
10405   4011          SENDW+FREE
10406   0003          LPT
10407   0636          LPTMSG
10410   5200          JMP START
        0630          *630
10630   0000  SLPMSG, ZBLOCK 3
10633   0000          0
10634   0000          0!SHERTZ
10635   0001
10636   0000  LPTMSG, ZBLOCK 3
10641   0000          0
10642   0000          0
10643   2401          TEXT /TASK 22 RUNNING/
10644   2313
10645   4062
10646   6240
10647   2225
10650   1616
10651   1116
10652   0700
```

8.3 ASSEMBLY AND LOAD PROCEDURE

The assembly and load procedure for the demonstration program is as follows:

```
.PAL RTS8<PARAM,RTS8
ERRORS DETECTED:  0
LINKS GENERATED:  0

.PAL CLOCK<PARAM,CLOCK
ERRORS DETECTED:  0
LINKS GENERATED:  0

.PAL MCR<PARAM,MCR
ERRORS DETECTED:  0
LINKS GENERATED:  0

.PAL OS8SUP<PARAM,OS8SUP
ERRORS DETECTED:  0
LINKS GENERATED:  0

.PAL TTY<PARAM,TTY
ERRORS DETECTED:  0
LINKS GENERATED:  0

.PAL LPT<PARAM,LPT
ERRORS DETECTED:  0
LINKS GENERATED:  0

.PAL RK8E<PARAM,RK8E
ERRORS DETECTED:  0
LINKS GENERATED:  0

.PAL NR20<PARAM,NR20
ERRORS DETECTED:  0
LINKS GENERATED:  0

.PAL NR22<PARAM,NR22
ERRORS DETECTED:  0
LINKS GENERATED:  0

.PAL SWAP<PARAM,SWAP
ERRORS DETECTED:  0
LINKS GENERATED:  0

.R ABSLDR
*NR20$
.SA SYS NR20 10400-10577

.R ABSLDR
*NR22$
.SA SYS NR22 10400-10577

.R ABSLDR
*RTS8,MCR,CLOCK,RK8E,TTY,LPT,OS8SUP
*SWAP,NR20,NR22$
.SA SYS RTS8V2
```

This example shows the assembly of the parameter file and the source task itself. The binaries of each nonresident task are then loaded into memory, and their nonresident portions saved on the swap device. All tasks are then loaded using the ABSLDR with the RTS/8 Executive being loaded first. Finally, the core image of all the tasks in memory is saved using the OS/8 SAVE command.

8.4 NONRESIDENT TASK ASSIGNMENT AND EXECUTION

The following is the execution of RTS/8 showing the assignment of tasks NR20 and NR22 to the system.

```
.R RTS8V2
*NR20=20
*NR22=22
*$>SY
01  CLCK 0
02  TTY  M
03  LPT  M
04  MCR
07  RK8  M
15  OS8F M
16  SWAP R
20       R N
22       R N
23 OS8   E
>RE 20
>RE 22
>EXIT
.
```

In the above example, the user terminates the installation of nonresident tasks with an ALTMODE, and returns to the MCR. A SYstat command is executed which prints a system status report (see Section 6.2.9). The REquest command is then used to run tasks NR20 and NR22. Shown below is output from tasks NR20 and NR22 on the line printer.

```
TASK 20 RUNNING
TASK 20 RUNNING
TASK 20 RUNNING
TASK 20 RUNNING
TASK 22 RUNNING
TASK 20 RUNNING
TASK 22 RUNNING
TASK 20 RUNNING
```

The EXIT command is typed to terminate RTS/8 execution and return to the OS/8 monitor.

CHAPTER 9

ADVANCED RTS/8 PROGRAMMING TECHNIQUES

9.1 PERFORMING A RESCHEDULE

9.1.1 Writing Delicate Code

Frequently, a task needs to manipulate data in another task or a common area. Since tasks are running 'simultaneously', problems will arise if two tasks want to access the same data at the same time. Consequently, delicate code wants to run with interrupts disabled while accessing data in another task.

NOTE

Interrupts may be disabled temporarily using either a IOF/ION pair or, on machines with memory extension, a CIF instruction which inhibits interrupts until the execution of the next JMP or JMS instruction.

For example, suppose Task A increments location COUNT occasionally and Task B decrements location COUNT from time to time during program execution. The code might look like the following:

```
/TASK A
LOOPA,          .
                .
                .
                .
X,              ISZ COUNT
                .
                .
                .
                .
                JMP LOOPA
COUNT,          0

/TASK B
LOOPB,          .
                .
                .
                .
                STA
Y,              TAD COUNT
Z,              DCA COUNT
                .
```

```
        .
        .
        JMP LOOPB
```

If Task A increments COUNT the same number of times that Task B decrements COUNT, it would be assumed that COUNT would be 0 at the end of the program. However, this is not necessarily so since a race condition can occur.

Suppose that Task A has a higher priority than Task B, and Task A is waiting for an event to occur with COUNT currently containing a 6. Task B is ready to decrement COUNT. However, an interrupt occurs after location Y has been executed. The AC contains a 5 and Task B is ready to store a 5 back into COUNT. The interrupt service routine, noting that the event Task A was waiting for has just occurred, now suspends Task B and resumes Task A. Task A now bumps COUNT from 6 to 7, and then goes back to sleep. Task B then resumes with the AC containing a 5 and stores a 5 into COUNT which is incorrect for proper program execution.

This situation is prevented from happening by disabling interrupts around the delicate code. Either of the following two solutions can be employed:

Solution 1

```
/TASK   B
LOOPB,  .
        .
        .
        .
        STA
        IOF
Y,      TAD COUNT
Z,      DCA COUNT
        ION
        .
        .
        .
        .
        JMP LOOPB
```

Solution 2

```
/TASK   B
LOOPB,  .
        .
        .
        .
        CIF CUR
        STA
Y,      TAD COUNT
Z,      DCA COUNT

        .
        .
        .
        .
        JMP LOOPB
```

Solution 2 (only usable on machines with memory extension) uses the CIF instruction since it temporarily inhibits interrupts until the next JMP or JMS instruction is executed.

9.1.2 Inhibiting Task Switching

Although the procedure in the previous section can be used, it at times can be very inefficient. If it is desired to perform a lot of manipulation on data which could be accessed by other tasks, it may be inappropriate to turn off interrupts. Inhibiting interrupts for long periods of time could affect other portions of the system where timed events are very important. Also, an interrupt can be lost (for example, clock interrupts) if interrupts are turned off for a significant amount of time.

For this case, another solution is possible. A task can inform the RTS/8 Executive that it wants to continue to run, and that while it is executing a certain piece of code, no other task should run even if a task of higher priority becomes runnable. This process is known as inhibiting task switching.

Task switching should be inhibited only under unusual circumstances and performed with care. While task switching is inhibited, interrupts may still occur and the interrupt service routine will get control. However, if task switching is inhibited, the interrupt service routine will always return control to the interrupted task after the interrupt has been serviced even if higher priority tasks are now runnable.

NOTE

If the user wishes to manipulate data which is accessed by an interrupt-level routine, interrupts **must** be inhibited since inhibiting task switching alone will not be sufficient in this case.

There are two methods of inhibiting task switching which are as follows:

Method 1: Task switching is automatically inhibited whenever a task's PC is less than 100. Thus, delicate code could be placed in the bottom of page 0 of any field.

Method 2: A task may inhibit task switching by zeroing location 35 in field 0. This location is symbolically referred to as TSWFLG (task switching flag) and is defined as such in the parameter file. In either case, after the task is through with its delicate code, it may not be sufficient for the task to reset TSWFLG to its original value(1). This is due to the fact that there may be some other higher-priority task that is entitled to run but did not run because task switching was inhibited. The user can find this out by interrogating location TSWFLG. If another task became runnable while task switching was inhibited, the RTS/8 executive sets the task switching flag to -1. When a task is ready to allow task switching again, it must examine this flag before resetting it to 1. If it was -1, the task returns control to the RTS/8 scheduler. This is performed by using the RESCHD ER as follows:

```
CAL
RESCHD
```

This ER causes RTS/8 to perform a reschedule that allows the runnable task of highest priority to be executed. If a user does not perform a RESCHD after re-enabling task switching, then a higher priority task which is entitled to run might not run until the next interrupt occurs. The interrupt may never occur, or if it does, it may be too late for proper program execution.

The preferred code for inhibiting task switching for Task B which was described previously is shown below:

```
/TASK   B
LOOPB,  .
        .
        .
        .
        CDF 0
        DCA I (TSWFLG   /INHIBIT TASK SWITCHING
        STA
Y,      TAD COUNT
Z,      DCA COUNT
        ISZ I (TSWFLG   /ALLOW TASK SWITCHING
```

```
JMP .+3          /SHOULD WE RESCHEDULE?
CAL              /YES
RESCHD
CDF CUR          /NO
.
.
.
.
JMP LOOPB
```

NOTE

Interrupt level routines should not look at or set the TSWFLG.

A summary of TSWFLG states is shown in Table 9-1.

Table 9-1
Summary of Task Switching Flag (TSWFLG) States

TSWFLG State	Value
Task switching allowed	1
Task switching inhibited	0
Task switching inhibited; reschedule as soon as possible	-1

9.2 EXECUTIVE REQUESTS FOR ADVANCED APPLICATIONS

9.2.1 WAITM - Waiting for Multiple Event Flags

Sometimes it is desirable to wait for a logical combination (AND or OR) of Event Flags. Waiting for the logical AND of two Event Flags is quite simple. The sequence:

```
CAL
WAITE
A                /WAIT FOR EVENT FLAG A
CAL
WAITE
B                /AND THEN WAIT FOR EVENT FLAG B
```

waits until both A and B have been POSTed.

Waiting for the logical OR of several event flags is more difficult since there is a possible race condition between the various tests and the interrupts (or task executions) which POST the Event Flags involved. The key to waiting for an OR of several Event Flags successfully is not to allow any interrupts to occur between the testing of the first Event Flag and the placing of the task in a Wait state if none of the flags were POSTed.

This is accomplished by using a special sequence of instructions and a special RTS/8 call named WAITM. WAITM is defined as JMS I 25. It must be executed with interrupts off, the Instruction Field set to 0 and the Data Field set to the current field, and it must be followed by a word containing the blocking bit(s) to be set in the Task Flags Table. The action of WAITM is equivalent to the action of the RTS/8 BLKARG ER except that a fast path through the RTS/8 Executive is taken and interrupts remain off until the blocking bits are on in the Task Flags Table.

For an example of the use of WAITM, assume that a task "TASK" wants to test two Event Flags A and B. If A is POSTed, control should go to location ADONE; if B is POSTed control should go to location BDONE. If neither is POSTed, the task must wait until one of them is POSTed. The code to perform this function is:

```
TESTAB, IOF                     /INTERRUPTS OFF - DELICATE CODE
        TAD A
        SNA CLA
        JMP ADONE               /ADONE MUST TURN INTERRUPTS ON
        TAD (4000+TASK          /SET A TO "WAITING" STATE
                                /INDICATING
        DCA A                   /THAT THIS TASK IS WAITING ON IT
        TAD B
        SNA CLA
        JMP BDONE               /BDONE MUST TURN INTERRUPTS ON
        TAD (4000+TASK          /SET B TO "WAITING" STATE
                                /INDICATING
        DCA B                   /THAT THIS TASK IS WAITING ON IT
        CIF 0
        CDF CUR
        WAITM
        EFWT                    /BLOCK TASK ON EVENT FLAG WAIT
        JMP TESTAB              /WE'RE BACK - ONE OF THE TWO
                                /EVENT FLAGS HAS BEEN POSTED.
                                /GO BACK TO FIND OUT WHICH ONE
```

9.2.2 WAITX - Wait for Exactly This Event Flag

The WAITX ER is similar to the WAITE ER. The exception is that if the Event Flag is not FINISHED, the task goes into EORMWT (instead of EFWT), and the task's PC in the TSTBL points back to the location containing the CAL of this ER. Thus, when the task resumes execution, it will re-execute the WAITX. If the EORMWT bit was cleared for some reason other than the Posting of the Event Flag in question, the task will immediately go back into EORMWT.

Consequently, control will never flow past this ER unless the Event Flag specified is actually posted (see discussion of DERAIL, Section 9.2.3). If a WAITE had been used and if the task was waiting on multiple Event Flags (which can happen using WAITM), then control conceivably could start up after the WAITE ER because some other Event Flag, and one that is no longer cared about, was posted. This situation can not occur with a WAITX.

9.2.3 DERAIL - Derail a Task's Execution

The DERAIL ER modifies the execution of a specified task and transfers control to a special subroutine of the task to process some exceptional condition. It does not cause any wait bits to get set or cleared.

Format:
```
TAD TASKNUM
CAL
DERAIL
ADDR
```

This ER simulates a "JMS ADDR" for the task whose number is contained in TASKNUM (the "derailed" task). ADDR is assumed to be in the same field in which the derailed task is executing. The derailed task's PC (from its Task State Table entry) is stored in ADDR; the PC entry in its Task State Table entry is then set to ADDR+1. Two important points concerning the operation of the DERAIL ER are as follows:

1. The derailed task's AC, Link, and Data Field settings are not saved by the DERAIL ER; therefore they must be saved and restored by the derail subroutine. In this sense, a derail subroutine is very much like an interrupt at the task level.

2. The contents of the derailed task's Task Flags word are not affected by the DERAIL ER. If the derailed task is not runnable, the derail subroutine will not be executed until the task becomes runnable.

The DERAIL ER is generally used by a high priority task to signal an emergency condition to lower priority tasks. An example would be a process-control environment where it is sometimes necessary to abort all operations if the room temperature exceeds some critical value. This can be checked by a task which measures room temperature every 10 seconds. It is inefficient and unmodular to include shutdown code in this "watchdog task" for all machinery being controlled. A better solution is to provide a location to which each equipment-controlling task can be DERAILED in order to shut down its own piece of equipment. The RTS/8 Power-Fail task uses the DERAIL ER to provide a similar facility on power-fail recovery (see Section 4.5), which can be used to reinitialize a task.

Example:

An example of a DERAIL routine is as follows:

```
DENTRY,         0                   /DERAIL ROUTINE ENTRY POINT
                DCA SAVAC           /SAVE AC
                RAR
                DCA SAVLNK          /SAVE LINK
                RDF
                TAD (CDF
                DCA DFRESET         /SAVE DATA FIELD
                CDF CUR
                .                   /HANDLE EMERGENCY CONDITION
                .
                .                   /BRANCH TO 'RESUME' IF YOU WANT
                .                   /TO RESUME WHERE YOU LEFT OFF
                .                   /BRANCH TO 'NORESUME' IF NOT
                .
RESUME,         TAD SAVLNK          /RESTORE LINK
                CLL RAL
                TAD SAVAC           /RESTORE AC
DFRESET,        HLT                 /RESTORE DF
                JMP I DENTRY        /RESUME (SAME FIELD)
NORESUME,       CLA CLL
                CDF CUR
                JMP RESTART         /RESTART TASK
SAVAC,          0
SAVLNK,         0
```

9.2.3.1 **Dangers of DERAIL -** A task can get into serious trouble if it is derailed while already in a derail routine. If this happens, the original PC, AC, link, etc., will be lost. There is no simple solution. Turning off interrupts in the derail routine may be too late to prevent this - the second derail could have already occurred before the derail routine was even entered the first time.

Consequently, a user doing a DERAIL should make sure that not more than one DERAL is done at a time. Alternatively, before doing a DERAIL, a task can check an interlock flag (which it must maintain) to see whether the target task has been derailed or not. The test and set of such a flag should be performed with interrupts inhibited.

9.2.3.2 **Restrictions Using DERAIL -** If a task is not runnable, derailing it will not make it runnable. If the task is in Event Flag Wait, it will remain in Event Flag Wait until the event occurs. When the Event Flag is POSTed, the task will wake up and begin to run in its derail routine rather than in the mainline routine. Thus, derailing a task to get it to perform some important job immediately may not always work. The task might be in one of the Wait states and may not be able to run for some time. For example, if the task were in Receive Wait at the time, the derail routine would not run until a message came in for that task.

A partial solution around this restriction is to code the task to be derailed so that it always waits on events using WAITX instead of WAITE. Then, if the user wants to derail this task, the task is first taken out of MSGWT or EORMWT and then derailed.

An example of the code for this situation is as follows:

```
TAD TASKNUM
CAL
UNBARG                  /UNBLOCK THE TASK
MSGWT!EORMWT            /FROM MESSAGE-RELATED WAITS
TAD TASKNUM
CAL
DERAIL
DENTRY
```

This will work because both the RECEIVE and the WAITX Executive Requests bump the PC back to the CAL before going into a Wait state. Thus, no harm is done if the task is taken out of that wait state for an incorrect reason. When the task resumes running at that point, it will re-execute the CAL (RECEIVE or WAITX) and go back into the Wait state as necessary. This method will not work if the task was in EFWT due to a WAITE ER because the task would resume running thinking the Event Flag had been posted when in fact it had not. A way to circumvent this (other than WAITX) is for the task to do WAITM instead of a WAITE, and poll the Event Flag upon waking up.

9.3 STARTING PARTITIONS AT AN ARBITRARY BOUNDARY

The advanced user can start a partition at an arbitrary boundary by using the following assembly and loading procedure. The example given assumes that the partition in which the user writes the nonresident portion of the task to run is three pages long (11200-11777). The PAL8 pseudo-operators FIELD and RELOC are used, and described in detail in the OS/8 handbook.

```
/TASKX
FIELD 1                 /SET FIELD
*1000                   /LOAD THIS CODE AT 11000-11577
RELOC 1200              /BUT ASSEMBLE IT TO RUN AT 11200-11777
.
.
<CODE>
.
.
PAGE
.
.
<CODE>
.
.
RELOC
```

The assembly, load and save procedures for the code are:

```
.R PAL8
*TASKX<PARAM.PA,TASKX.PA          (ASSEMBLY OF TASK)

.R ABSLDR
*TASKX$                           (LOADING OF TASK INTO 11000)

.SAVE DEV TASKX 11000-11577       (SAVING OF TASK-THREE PAGES OF
                                  CODE)
```

The swapper, upon loading this task, places it into the partition at 11200, which is where it was assembled to run.

9.4 DIRECT REFERENCES TO SYSTEM TABLES

A task may directly interrogate locations in the RTS/8 Executive tables to obtain information about itself or any other task, as long as the following two restrictions are observed:

1. Due to the interrupt-driven nature of the system, these table entries may change at any time; therefore, interrupts should be inhibited between the time these entries are tested and the time that processing which depends on the testing is completed. For example, testing the Message Queue Header for a task may show no messages, but an I/O interrupt occurring immediately after the test might allow a higher priority task to run. This task might send a message, invalidating the result of the test. To prevent this, interrupts should be turned off during and after the test.

2. System table entries may be changed only through RTS/8 Executive Requests.

Symbols have been defined in the system parameter table that permit symbolic references to be made into these tables symbolically. The symbolic expressions which yield the address of the system table entries for task N are:

N+TFTABL	Task Flags Table entry for task N; if zero, task N is runnable.
N^2+MSGTBL	Input Message Queue Header for task N; if zero, task N has no messages in its queue.

N^4+TSTABL	First word of Task State Table Entry for task N.
N^2+RESTBL	First word of Residency Table entry for nonresident task N.

For example, for a task to determine whether it had messages in its input queue without issuing a RECEIVE request, the code would be:

```
CDF 0                          /EXEC TABLES IN FIELD 0
IOF                            /TURN INTERRUPTS OFF
TAD I (TASK^2+MSGTBL
SNA CLA                        /ANY MESSAGES?
JMP NONE                       /NO
ION                            /YES
```

The code at NONE must eventually turn interrupts back on.

APPENDIX A

RTS/8 DISTRIBUTED SOURCE FILES

The RTS/8 source files included on the distributed tape are:

File Name	Task Name	Task Function
PARAM.PA		System parameter file with all equates blank. Appropriate values should be inserted to create specific parameter files.
RTS8.PA		RTS/8 Executive
MCR.PA	MCR	Monitor Console Routine
MCR.PA	null task	Null task
OS8SUP.PA	OS8	OS/8 Support Task
OS8SUP.PA	OS8F	OS/8 File Support Task
PWRF.PA	PWRF	Power Fail Task
CLOCK.PA	CLOCK	Clock Handler Task
TTY.PA	TTY	Terminal Driver Task
LPT.PA	LPT	Line Printer Driver Task
DTA.PA	DTA	TC08 DECtape Driver Task
RK8.PA	RK8	RK8 Disk Driver Task
RK8E.PA	RK8	RK8E Disk Driver Task
RF08.PA	RF08/DF32	RF08/DF32 Fixed-Head Disk Driver Task
CSA.PA	CSA	Cassette Driver Task
CSAF.PA	CSAF	Cassette File Support Task
UDCICS.PA	UDC/ICS	Universal Digital Controller/ Industrial Controller Subsystem Handler Task
RX01RT.PA	RX8A	Floppy Disk Handler (1st controller)
RX01RT.PA	RX8B	Floppy Disk Handler (2nd controller)
RX01RT.PA	RX8C	Floppy Disk Handler (3rd controller)
RX01RT.PA	RX8D	Floppy Disk Handler (4th controller)
LTA.PA	LTA	LINCtape Driver Task
SWAP.PA	SWAPPER	Nonresident Task Swapper
NULL8A.PA	NULL8A	Null Task for PDP-8A
EXIT.PA	EXIT	Exit Task

APPENDIX B

RTS/8 COMPONENT SIZES

The following table gives the approximate size and default origins of each component of the RTS/8 system. An RTS/8 memory map showing default memory allocation is shown in Figure B-1. The modules may be placed anywhere in memory at the user's discretion except for RTS8, CLOCK and SWAP. Also, certain modules must be placed in field 0 where indicated in the tables.

The following parameters, which are used in Table B-1, are defined as follows:

NTASKS = Number of tasks in system
CLKQLN = Number of entries in clock queue
MCRSYS = 1 if MCR SYSTAT function desired, else 0
MCRCLK = 1 if MCR CLOCK functions desired, else 0
KLINES = Number of physical KL8-A lines

Any fractions from divides should be dropped.

Table B-1
RTS/8 Component Sizes

Software Component	Default Origin	Number of Pages Required (1 page =128 words)	Comments
RTS/8 Executive	00200	5+NTASKS/18	Must be in locations 00200-01200. Uses page 0 locations 0-3 and 20-47 and auto-index register 17.
Clock Module	1st page after end of RTS/8 Executive (or SWAPPER)	3+CLKQLN/22	Must be in field 0. Uses auto-index register 10.
New Terminal Module, V2	03400	3	
Old Terminal Module, V1	03400	2	

(Continued on next page)

Table B-1 (Cont.)
RTS/8 Component Sizes

Software Component	Default Origin	Number of Pages Required (1 page =128 words)	Comments
Line Printer Module	14400	1	
TC08 DECtape Module	14600	2	
RK8E Disk Module	04200	2	
DF32/RF08 Module	04400	1	
LINCtape	15000	2	
RK8 Disk Module	04200	1	
Power Fail Module	10200	1+NTASKS/32	
UDC Module	10600	7-11 depending on table space desired.	Uses page 0 locations 130-144
Cassette Module	13600	3	
Cassette Label Support Module	13000	3	Requires cassette handler.
OS/8 File Support Module	04600 if OS/8 support present, otherwise 06200[1]	6	Requires a mass storage handler. Must run in field 0.
OS/8 Support Task	06200*	6	Must run in field 0 - requires 8K for OS/8 plus a mass storage handler. Uses page 0 locations 164-177.
Monitor Console Routine Task	17600 minus length (15200 with all options and no KL8A support)	5+3xMCRCLK+ MCRSYS + (NTASKS+40)/64.	Requires "console" terminal handler and page 0 locations 100-117. See Table B-2 for more details.

[1]Moves down by length of KL8A support code if KL8A support present.

(Continued on next page)

Table B-1 (Cont.)
RTS/8 Component Sizes

Software Component	Default Origin	Number of Pages Required (1 page =128 words)	Comments
RX01	13200	2	
SWAPPER	1st page after end of RTS/8 executive	2	Must be in field 0
EXIT	15000	1/2	May relocate within a page
NULL8A	13600	1	
KL8A Support	17600 minus length	1+KLINES+1 3	Must be in field 0.

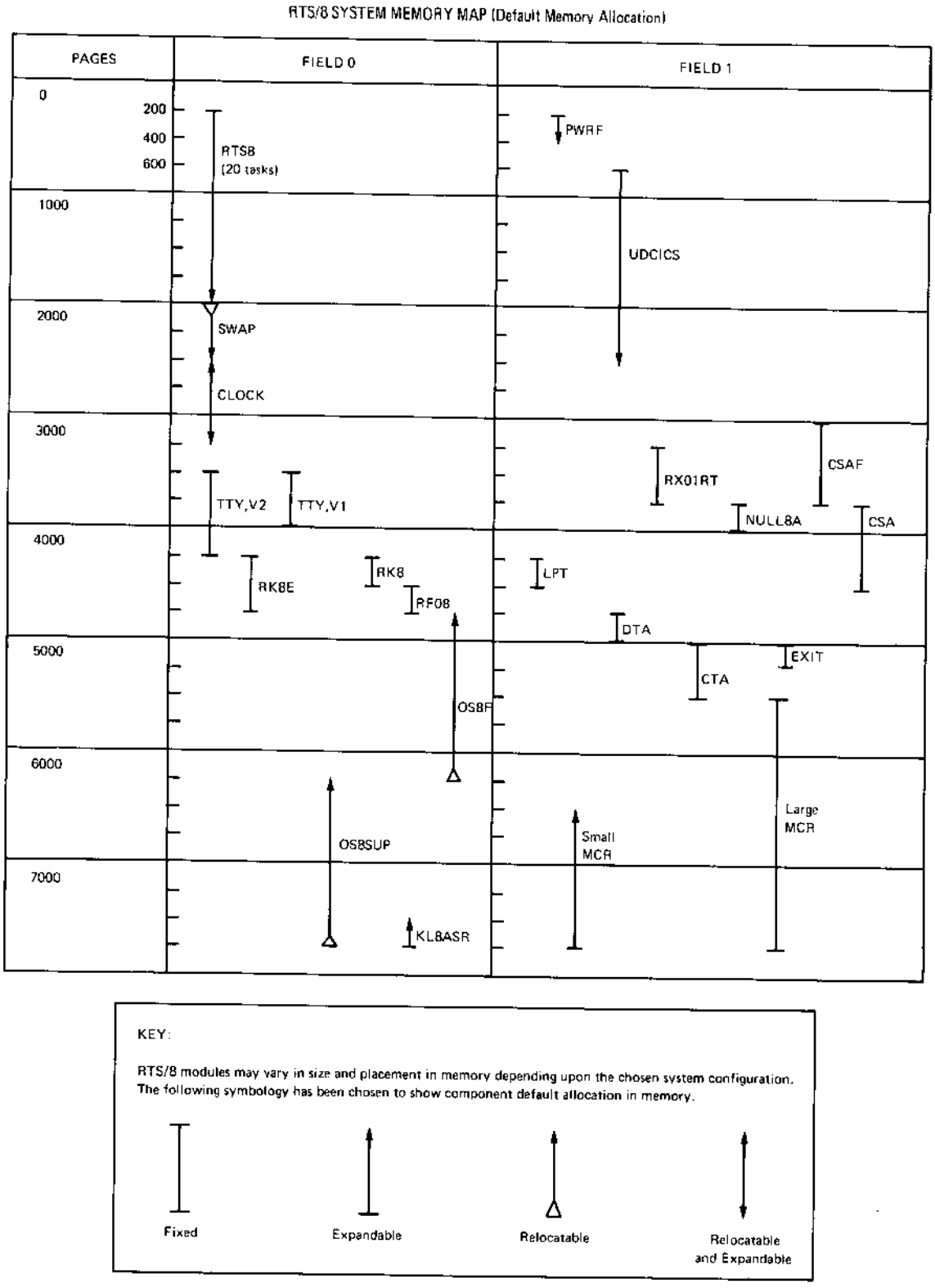


Figure B-1 RTS/8 System Memory Map (Default Memory Allocation)

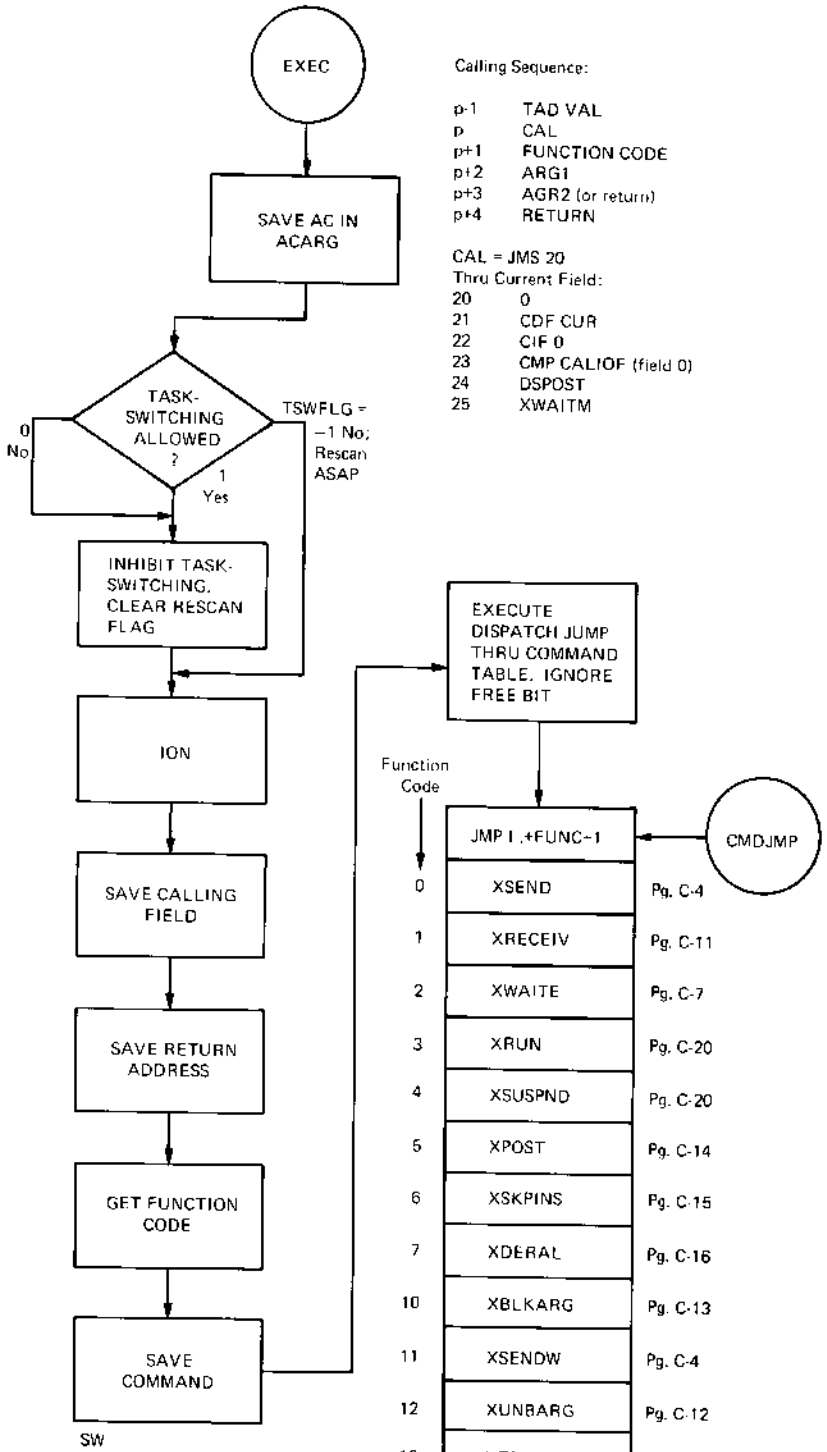
Table B-2
MCR Component Size

	If less than 34(octal) tasks	If 34(octal) or more tasks
NO SYSTAT NO CLOCK	LENGTH: 5 pages DEF ORIG: 6400 IF NONRS: 6200 PART: 6400-7377	LENGTH: 6 pages ORIG: 6200 IF NONRS: 6200 PART: 6400-7577
SYSTAT NO CLOCK	LENGTH: 6 pages ORIG: 6200 IF NONRS: 6200 PART: 6400-7577	LENGTH: 7 pages ORIG: 6000 IF NONRS: 5600 PART: 6000-7377
CLOCK NO SYSTAT	LENGTH: 10 pages ORIG: 5600 IF NONRS: 5600 PART: 6000-7577	LENGTH: 11 pages ORIG: 5400 IF NONRS: 5200 PART: 5400-7377
SYSTAT CLOCK	LENGTH: 11 pages ORIG: 5400 IF NONRS: 5200 PART: 5400-7377	LENGTH: 12 pages DEF ORIG: 5200 IF NONRS: 5200 PART: 5400-7577

APPENDIX C

RTS/8 FLOWCHARTS

This appendix contains RTS/8 flowcharts that graphically show RTS/8 system operation.



Calling Sequence:

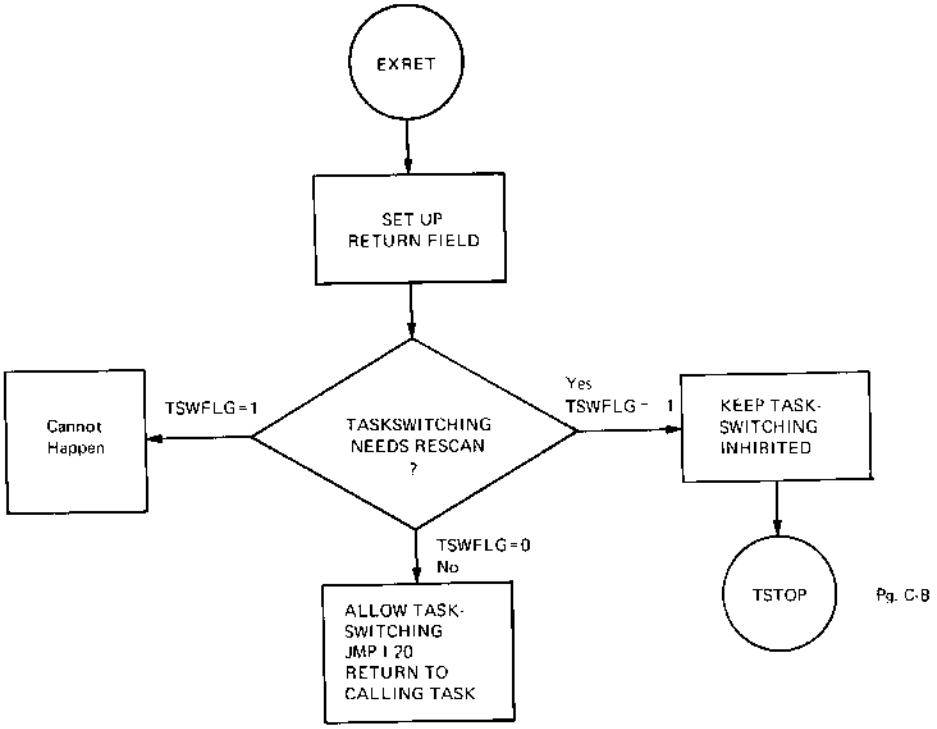
p-1	TAD VAL
p	CAL
p+1	FUNCTION CODE
p+2	ARG1
p+3	AGR2 (or return)
p+4	RETURN

CAL = JMS 20
Thru Current Field:

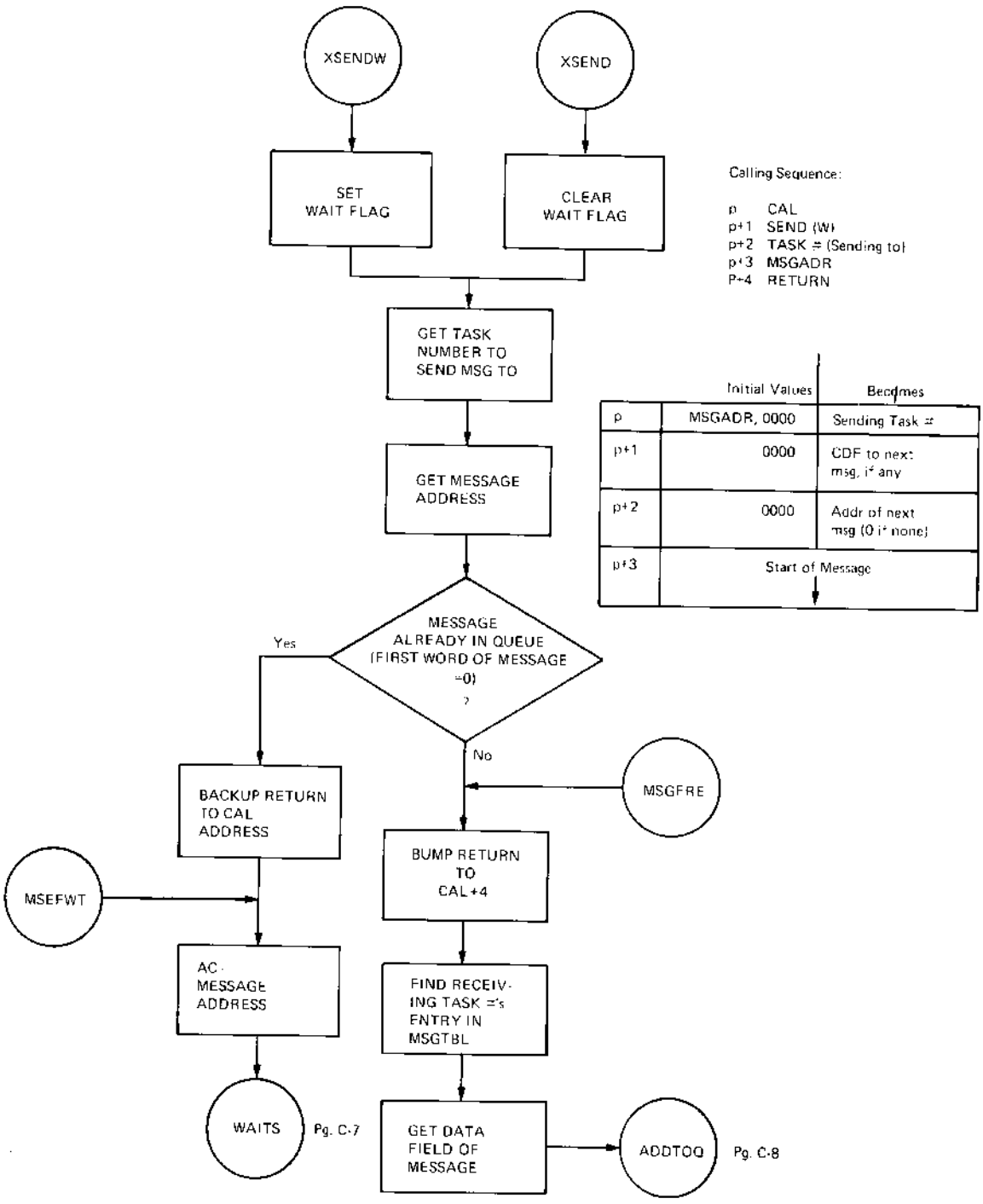
20	0
21	CDF CUR
22	CIF 0
23	CMP CAL!OF (field 0)
24	DSPOST
25	XWAITM

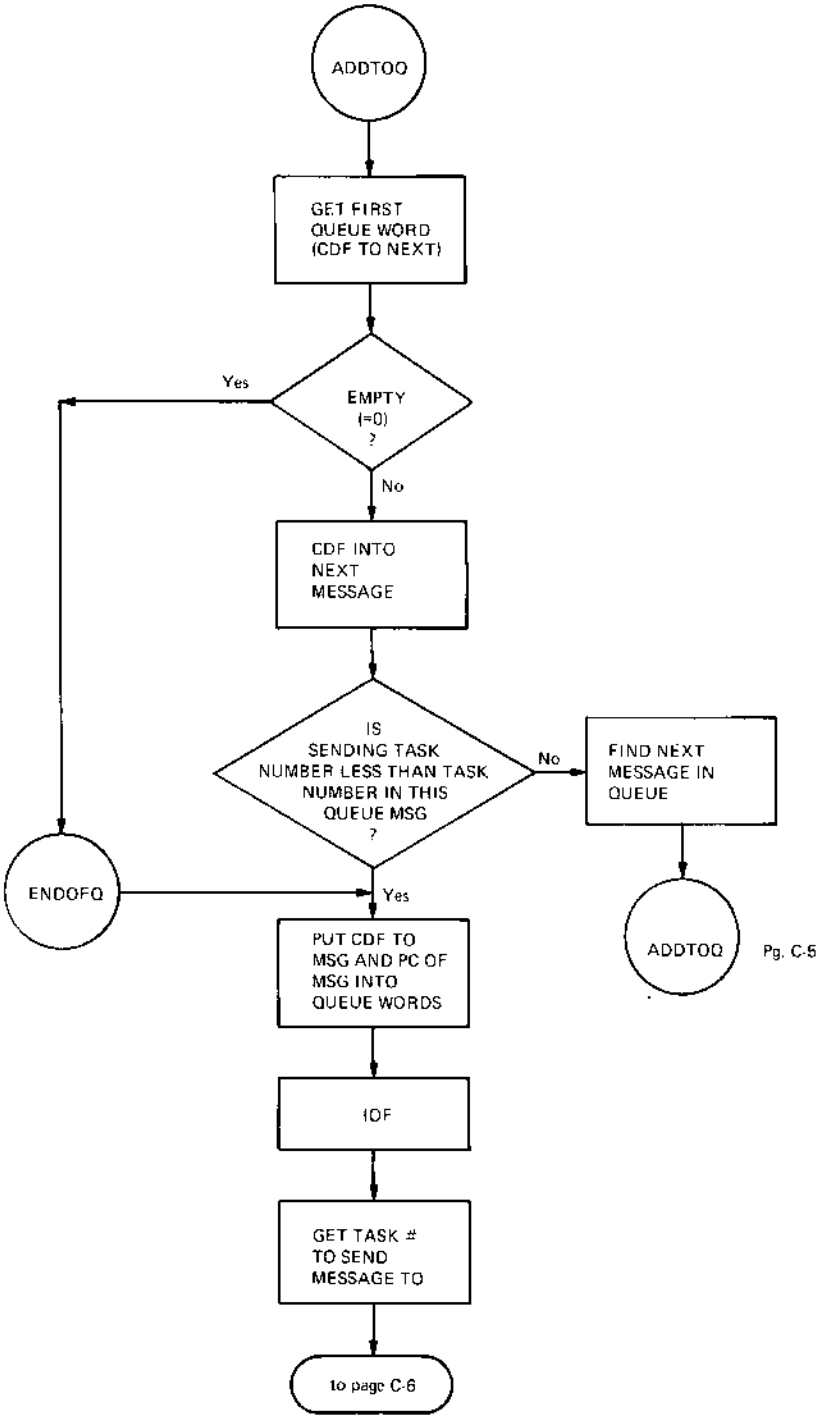
SW designates an operation that is performed only when using nonresident operations.

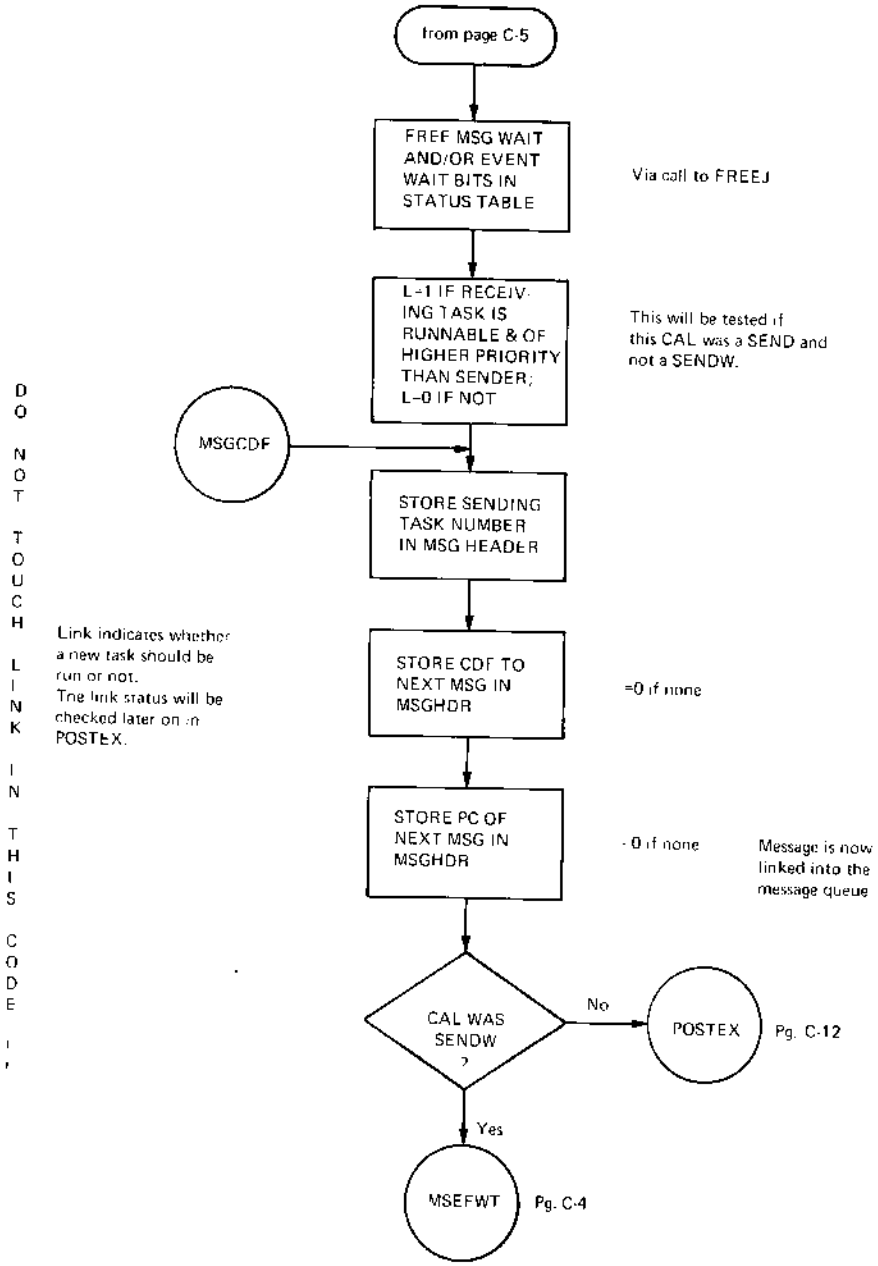
EAE designates an operation that is performed only when the Extended Arithmetic Element is used.

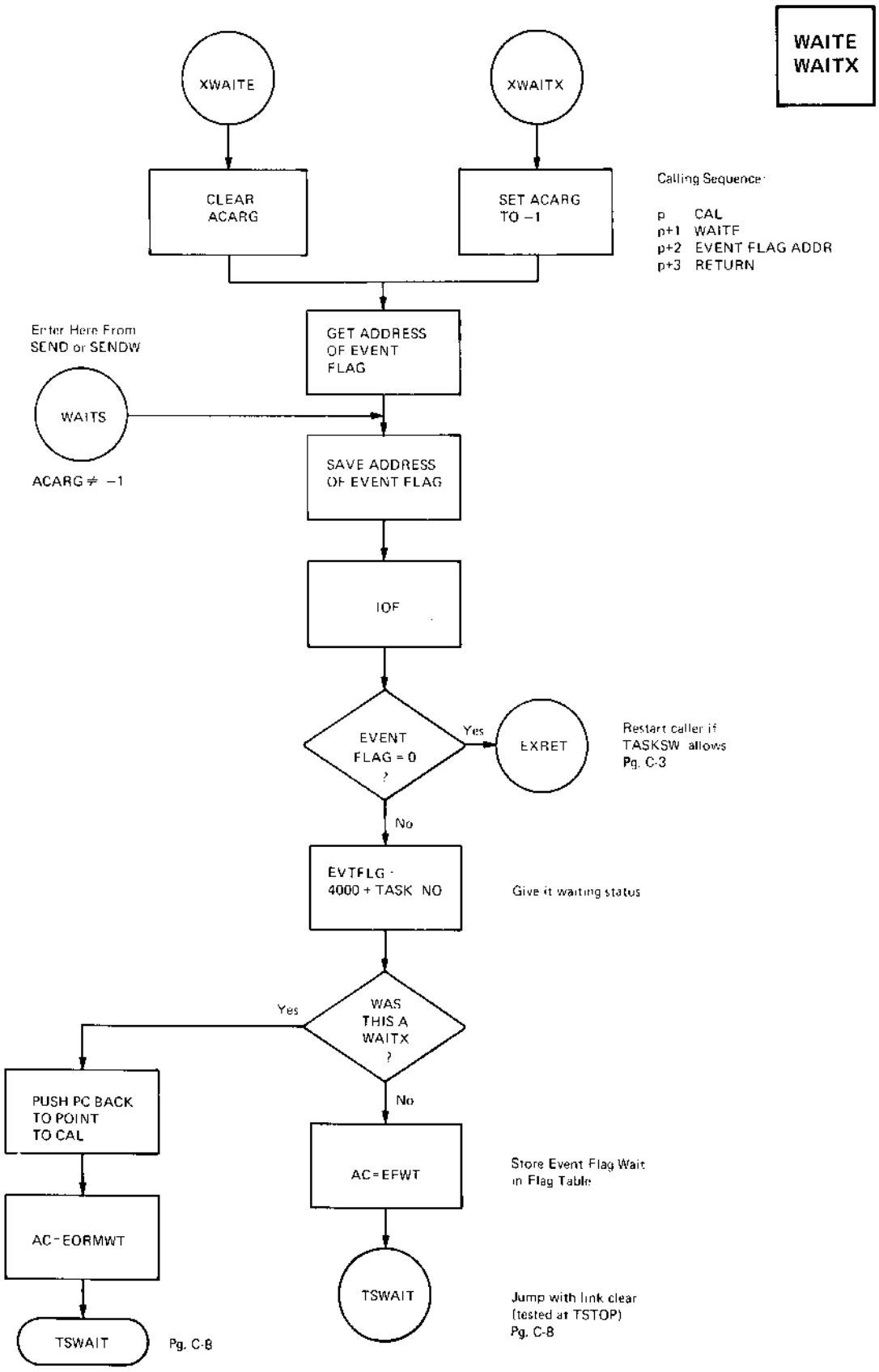

EXRET
SET UP
RETURN FIELD
TASKSWITCHING
NEEDS RESCAN
?
TSWFLG=1
Cannot
Happen
Yes
TSWFLG = 1
KEEP TASK-
SWITCHING
INHIBITED
TSTOP
Pg. C-8
TSWFLG=0
No
ALLOW TASK-
SWITCHING
JMP I 20
RETURN TO
CALLING TASK

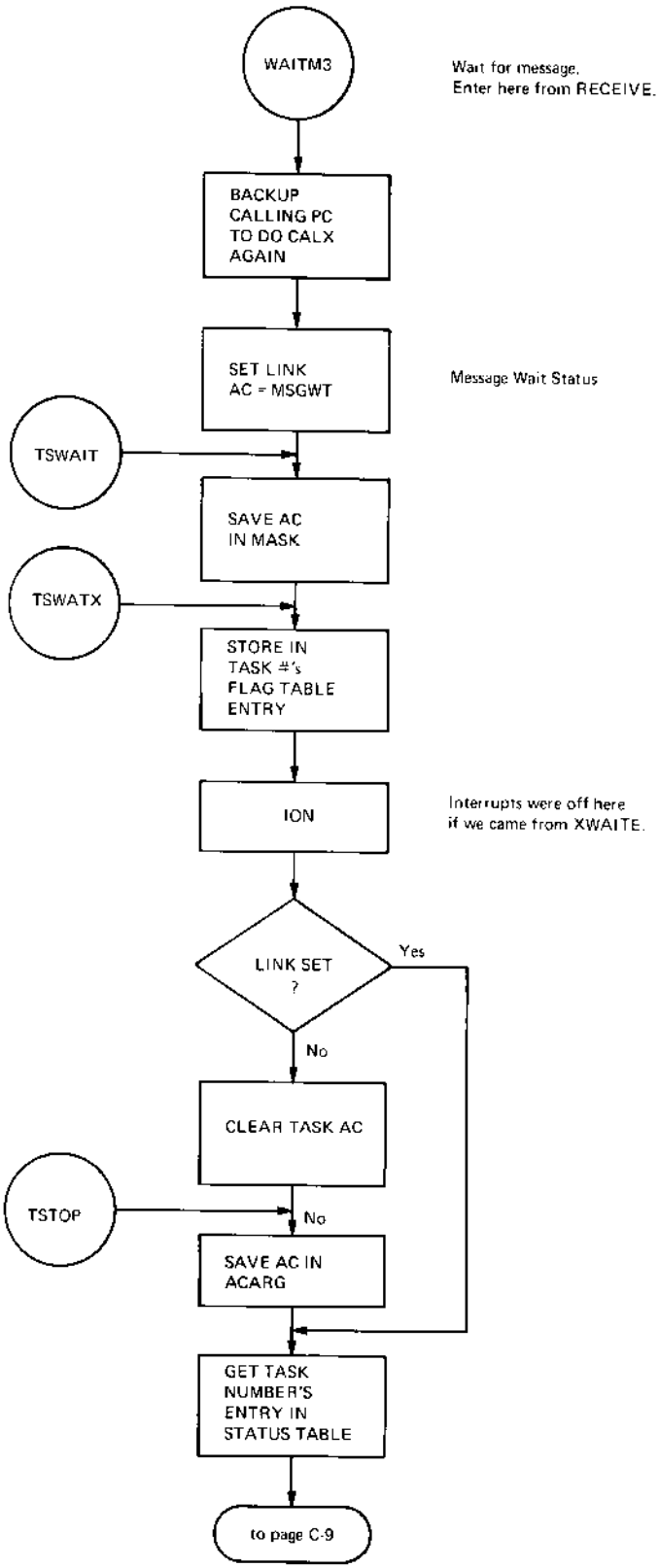
SEND
SENDW

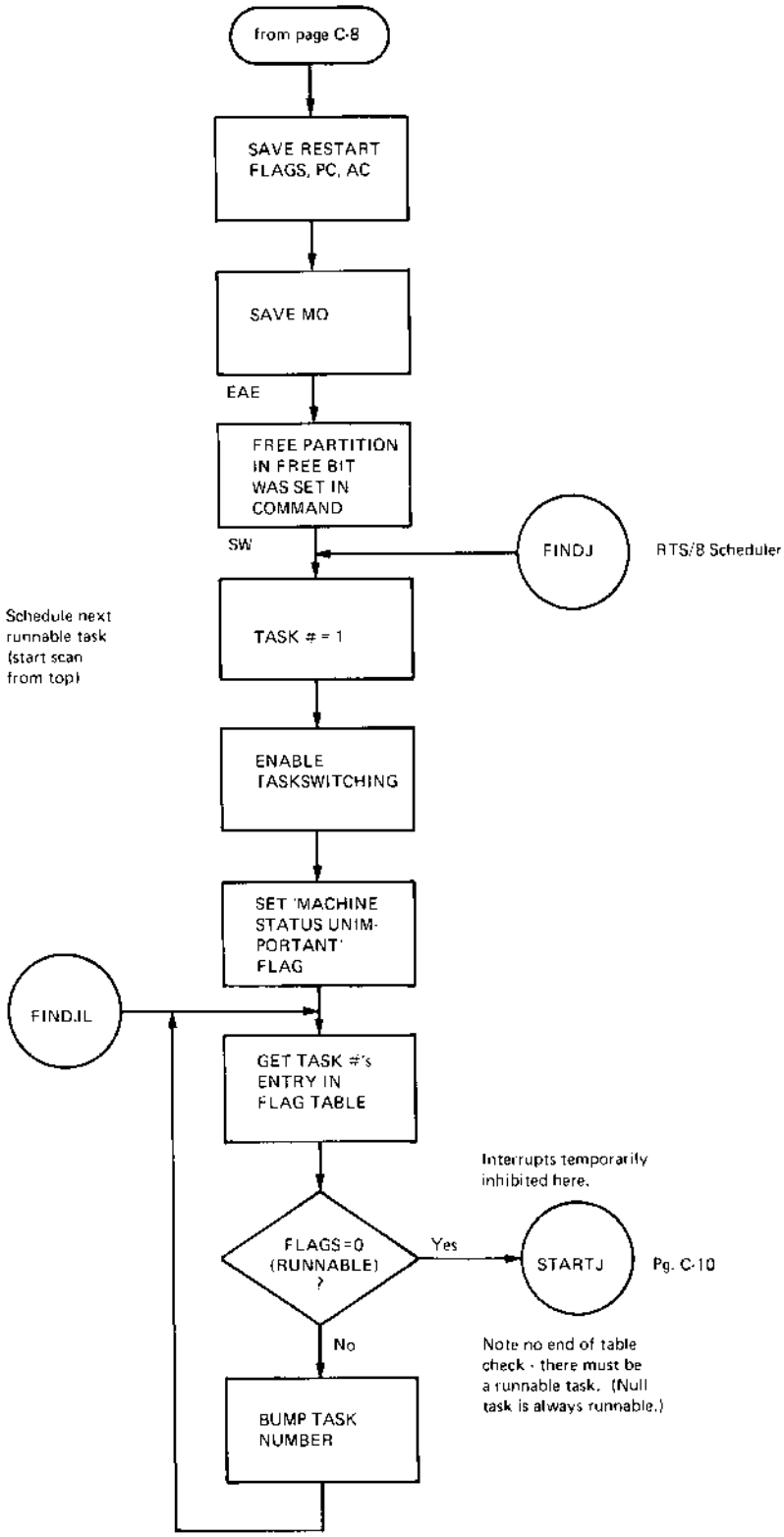

XSENDW
XSEND
SET WAIT FLAG
CLEAR WAIT FLAG
Calling Sequence:
p CAL
p+1 SEND (W)
p+2 TASK # (Sending to)
p+3 MSGADR
P+4 RETURN
GET TASK NUMBER TO SEND MSG TO
GET MESSAGE ADDRESS
Initial Values
Becomes
p
MSGADR, 0000
Sending Task #
p+1
0000
CDF to next msg, if any
p+2
0000
Addr of next msg (0 if none)
p+3
Start of Message
MESSAGE ALREADY IN QUEUE (FIRST WORD OF MESSAGE =0) ?
Yes
No
BACKUP RETURN TO CAL ADDRESS
MSGFRE
MSEFWT
BUMP RETURN TO CAL+4
AC = MESSAGE ADDRESS
FIND RECEIVING TASK ='s ENTRY IN MSGTBL
WAITS
Pg. C-7
GET DATA FIELD OF MESSAGE
ADDTOQ
Pg. C-8

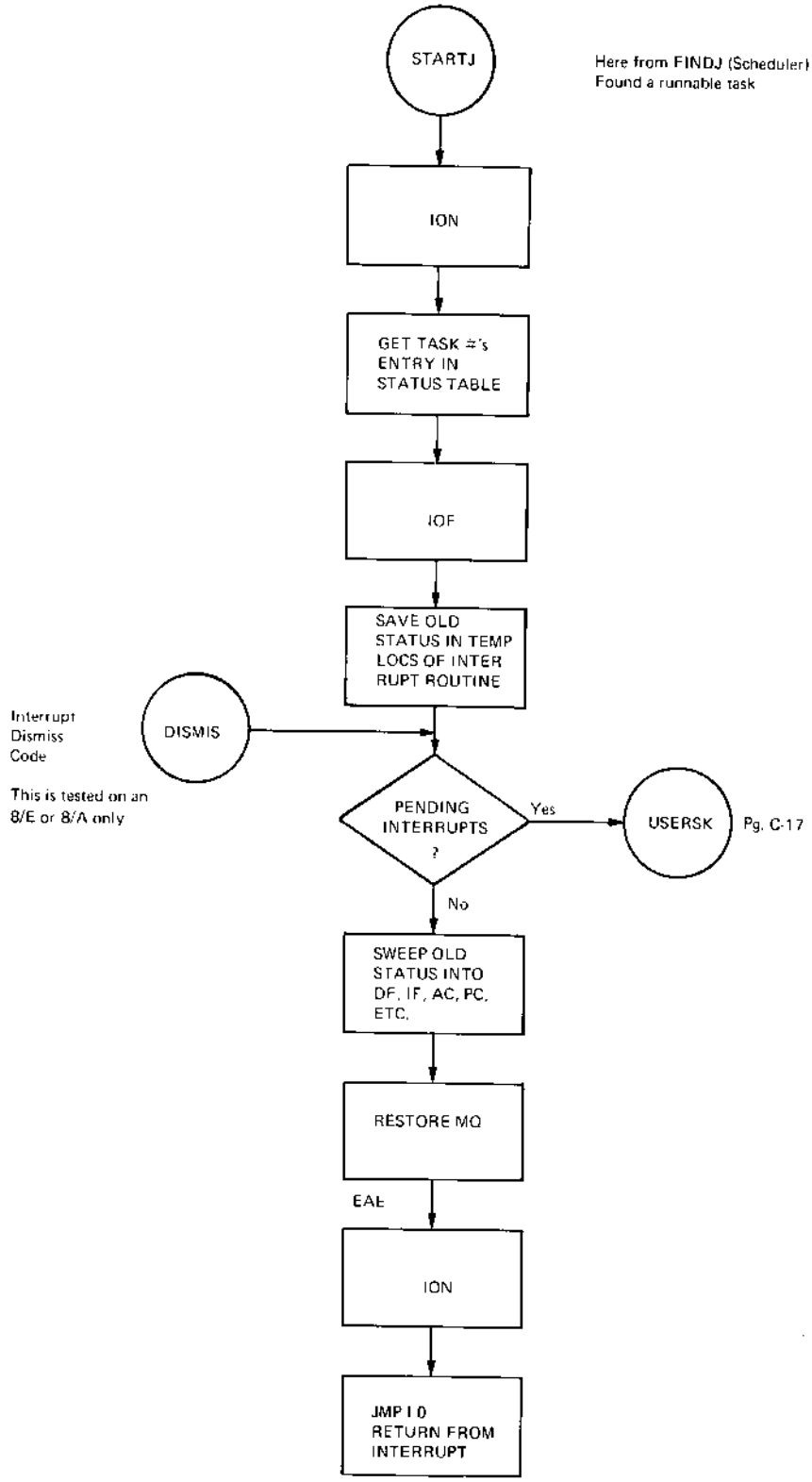

ADDTOQ
GET FIRST QUEUE WORD (CDF TO NEXT)
EMPTY (=0) ?
Yes
No
CDF INTO NEXT MESSAGE
IS SENDING TASK NUMBER LESS THAN TASK NUMBER IN THIS QUEUE MSG ?
No
FIND NEXT MESSAGE IN QUEUE
ADDTOQ
Pg. C-5
ENDOFQ
Yes
PUT CDF TO MSG AND PC OF MSG INTO QUEUE WORDS
IOF
GET TASK # TO SEND MESSAGE TO
to page C-6


from page C-5
FREE MSG WAIT AND/OR EVENT WAIT BITS IN STATUS TABLE
Via call to FREEJ
L=1 IF RECEIVING TASK IS RUNNABLE & OF HIGHER PRIORITY THAN SENDER; L=0 IF NOT
This will be tested if this CAL was a SEND and not a SENDW.
MSGCDF
STORE SENDING TASK NUMBER IN MSG HEADER
STORE CDF TO NEXT MSG IN MSGHDR
=0 if none
STORE PC OF NEXT MSG IN MSGHDR
=0 if none
Message is now linked into the message queue
CAL WAS SENDW ?
No
POSTEX
Pg. C-12
Yes
MSEFWT
Pg. C-4
Link indicates whether a new task should be run or not. The link status will be checked later on in POSTEX.
DO NOT TOUCH LINK IN THIS CODE !


WAITE
WAITX
XWAITE
XWAITX
CLEAR
ACARG
SET ACARG
TO −1
Calling Sequence:
p CAL
p+1 WAITE
p+2 EVENT FLAG ADDR
p+3 RETURN
Enter Here From
SEND or SENDW
WAITS
ACARG ≠ −1
GET ADDRESS
OF EVENT
FLAG
SAVE ADDRESS
OF EVENT FLAG
IOF
EVENT
FLAG = 0
?
Yes
EXRET
Restart caller if
TASKSW allows
Pg. C-3
No
EVTFLG +
4000 + TASK NO
Give it waiting status
WAS
THIS A
WAITX
?
Yes
No
PUSH PC BACK
TO POINT
TO CAL
AC=EFWT
Store Event Flag Wait
in Flag Table
AC=EORMWT
TSWAIT
Jump with link clear
(tested at TSTOP)
Pg. C-8
TSWAIT
Pg. C-8


WAITM3
Wait for message.
Enter here from RECEIVE.
BACKUP
CALLING PC
TO DO CALX
AGAIN
SET LINK
AC = MSGWT
Message Wait Status
TSWAIT
SAVE AC
IN MASK
TSWATX
STORE IN
TASK #'s
FLAG TABLE
ENTRY
ION
Interrupts were off here
if we came from XWAITE.
LINK SET
?
Yes
No
CLEAR TASK AC
TSTOP
No
SAVE AC IN
ACARG
GET TASK
NUMBER'S
ENTRY IN
STATUS TABLE
to page C-9

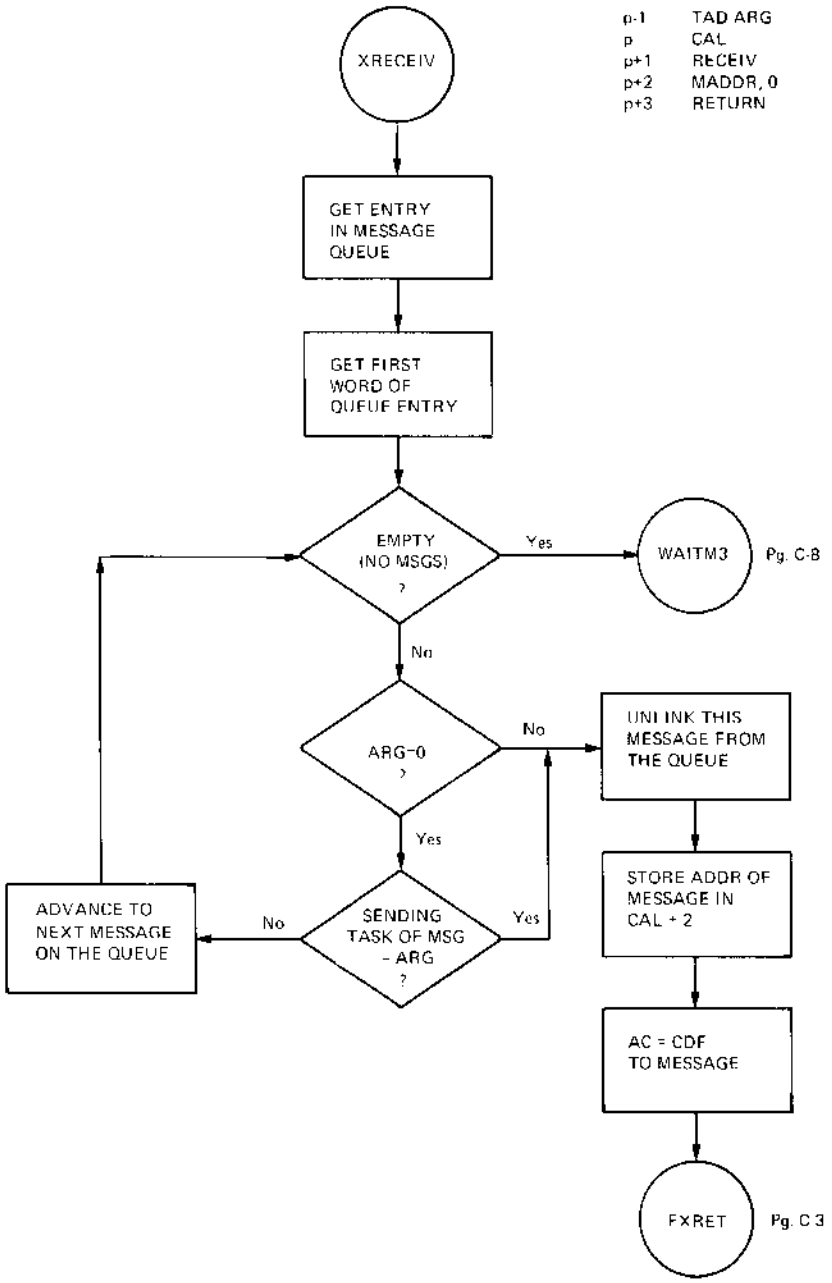

from page C-8
SAVE RESTART FLAGS, PC, AC
SAVE MQ
EAE
FREE PARTITION IN FREE BIT WAS SET IN COMMAND
SW
FINDJ
RTS/8 Scheduler
Schedule next runnable task (start scan from top)
TASK # = 1
ENABLE TASKSWITCHING
SET 'MACHINE STATUS UNIMPORTANT' FLAG
FINDJL
GET TASK #'s ENTRY IN FLAG TABLE
Interrupts temporarily inhibited here.
FLAGS=0 (RUNNABLE) ?
Yes
STARTJ
Pg. C-10
No
Note no end of table check - there must be a runnable task. (Null task is always runnable.)
BUMP TASK NUMBER


STARTJ
Here from FINDJ (Scheduler)
Found a runnable task
ION
GET TASK #'s
ENTRY IN
STATUS TABLE
IOF
SAVE OLD
STATUS IN TEMP
LOCS OF INTER
RUPT ROUTINE
Interrupt
Dismiss
Code
This is tested on an
8/E or 8/A only
DISMIS
PENDING
INTERRUPTS
?
Yes
USERSK
Pg. C-17
No
SWEEP OLD
STATUS INTO
DF, IF, AC, PC,
ETC.
RESTORE MQ
EAE
ION
JMP I 0
RETURN FROM
INTERRUPT

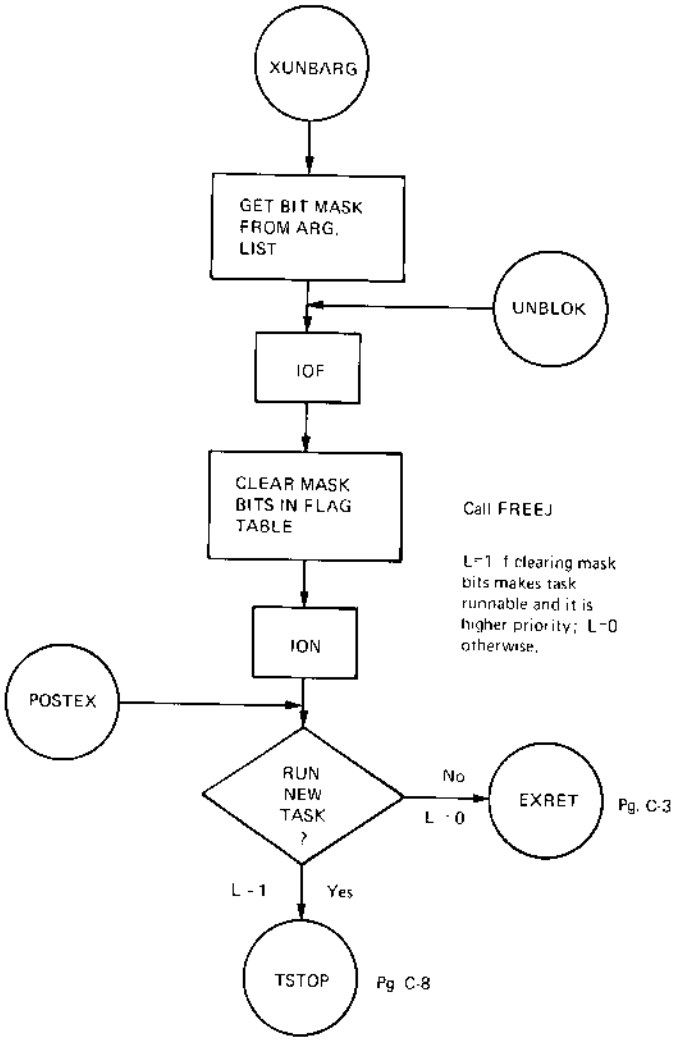
RECEIV

Calling Sequence:

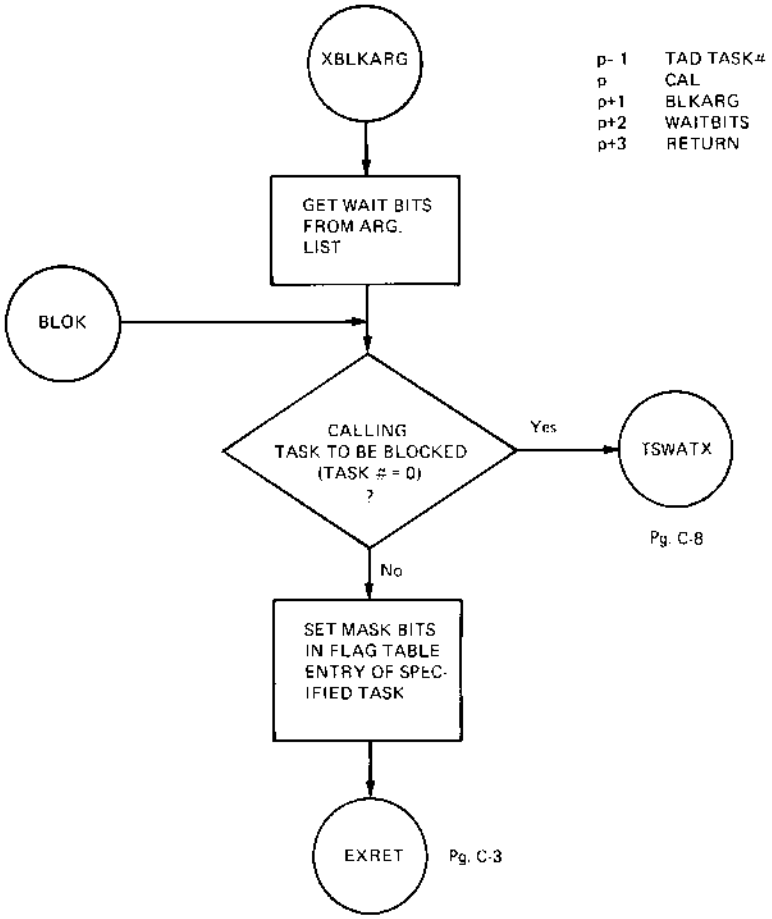
```
p-1     TAD ARG
p       CAL
p+1     RECEIV
p+2     MADDR, 0
p+3     RETURN
```



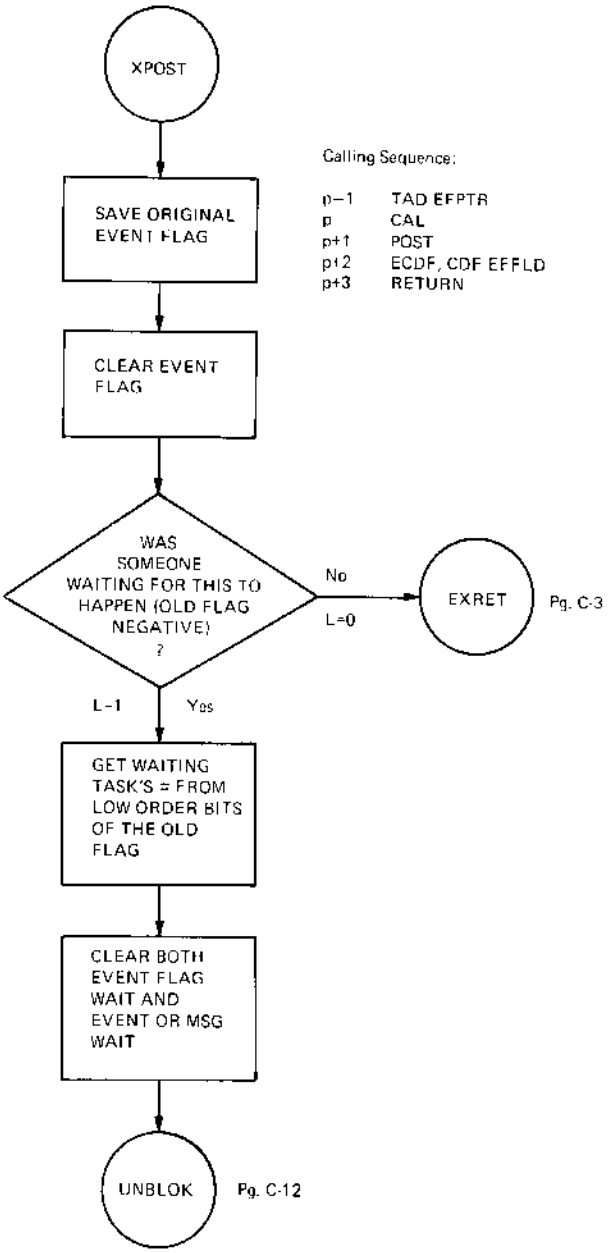
UNBARG


XUNBARG
GET BIT MASK FROM ARG. LIST
UNBLOK
IOF
CLEAR MASK BITS IN FLAG TABLE
Call FREEJ
L=1 if clearing mask bits makes task runnable and it is higher priority; L=0 otherwise.
ION
POSTEX
RUN NEW TASK ?
No
L = 0
EXRET
Pg. C-3
L = 1
Yes
TSTOP
Pg C-8

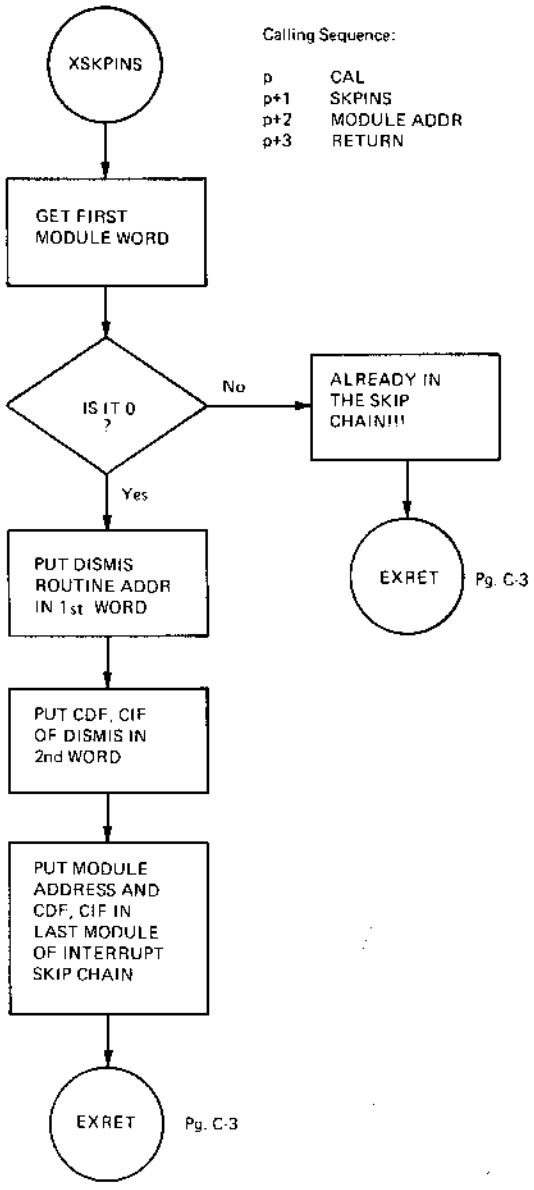
BLKARG


XBLKARG
p- 1 TAD TASK#
p CAL
p+1 BLKARG
p+2 WAITBITS
p+3 RETURN
GET WAIT BITS FROM ARG. LIST
BLOK
CALLING TASK TO BE BLOCKED (TASK # = 0) ?
Yes
TSWATX
Pg. C-8
No
SET MASK BITS IN FLAG TABLE ENTRY OF SPECIFIED TASK
EXRET
Pg. C-3

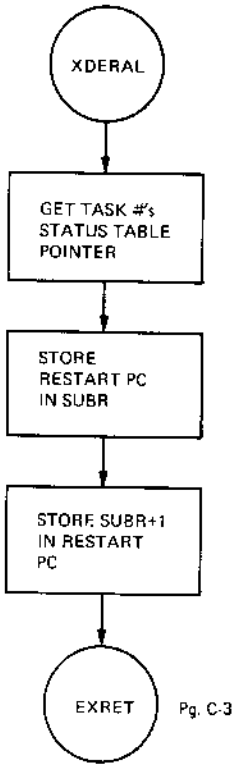
POST


XPOST
SAVE ORIGINAL EVENT FLAG
CLEAR EVENT FLAG
WAS SOMEONE WAITING FOR THIS TO HAPPEN (OLD FLAG NEGATIVE) ?
No
L=0
EXRET
Pg. C-3
L=1
Yes
GET WAITING TASK'S = FROM LOW ORDER BITS OF THE OLD FLAG
CLEAR BOTH EVENT FLAG WAIT AND EVENT OR MSG WAIT
UNBLOK
Pg. C-12
Calling Sequence:
p−1 TAD EFPTR
p CAL
p+1 POST
p+2 ECDF, CDF EFFLD
p+3 RETURN

SKPINS

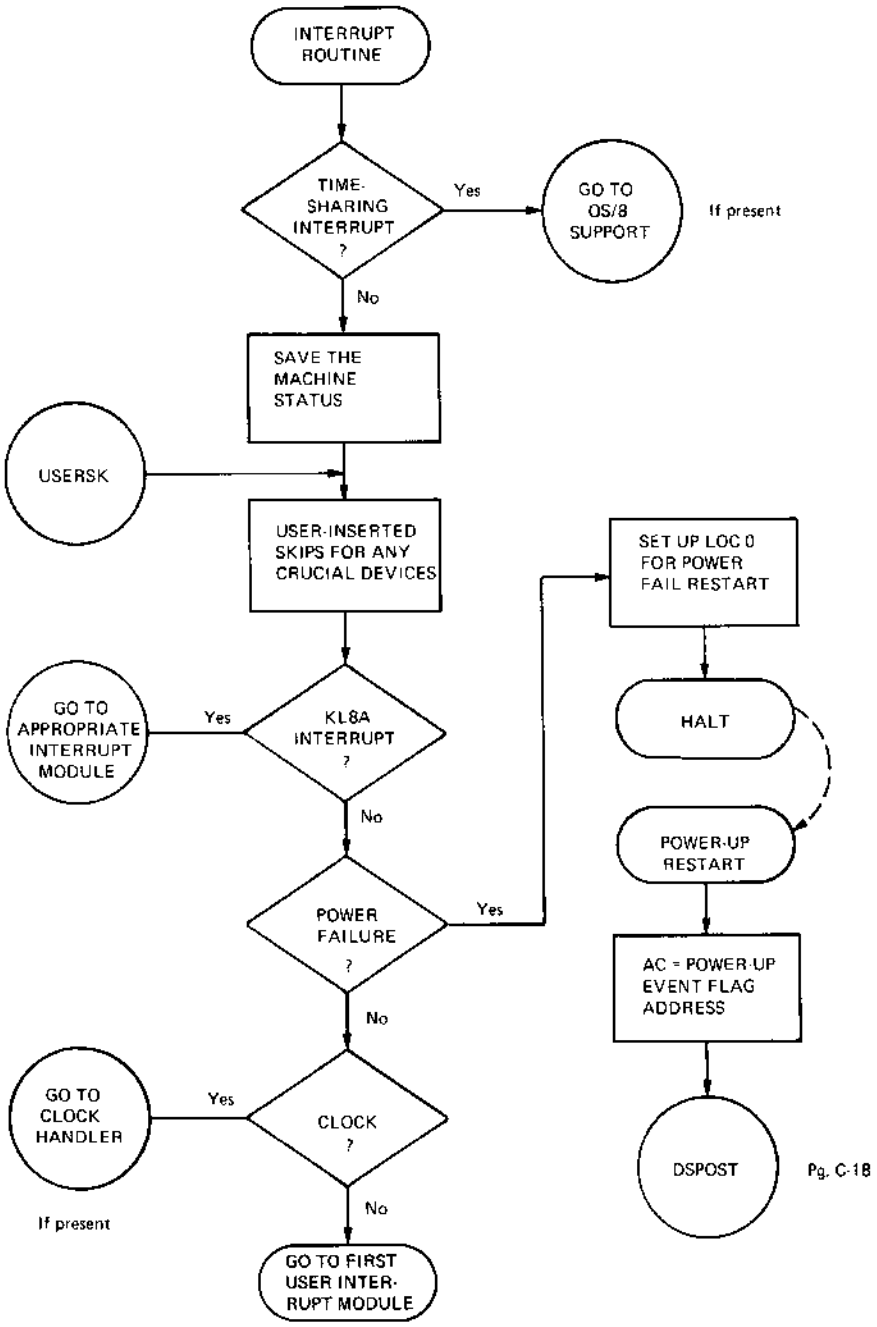

XSKPINS
Calling Sequence:
p CAL
p+1 SKPINS
p+2 MODULE ADDR
p+3 RETURN
GET FIRST MODULE WORD
IS IT 0 ?
No
ALREADY IN THE SKIP CHAIN!!!
EXRET
Pg. C-3
Yes
PUT DISMIS ROUTINE ADDR IN 1st WORD
PUT CDF, CIF OF DISMIS IN 2nd WORD
PUT MODULE ADDRESS AND CDF, CIF IN LAST MODULE OF INTERRUPT SKIP CHAIN
EXRET
Pg. C-3

DERAIL

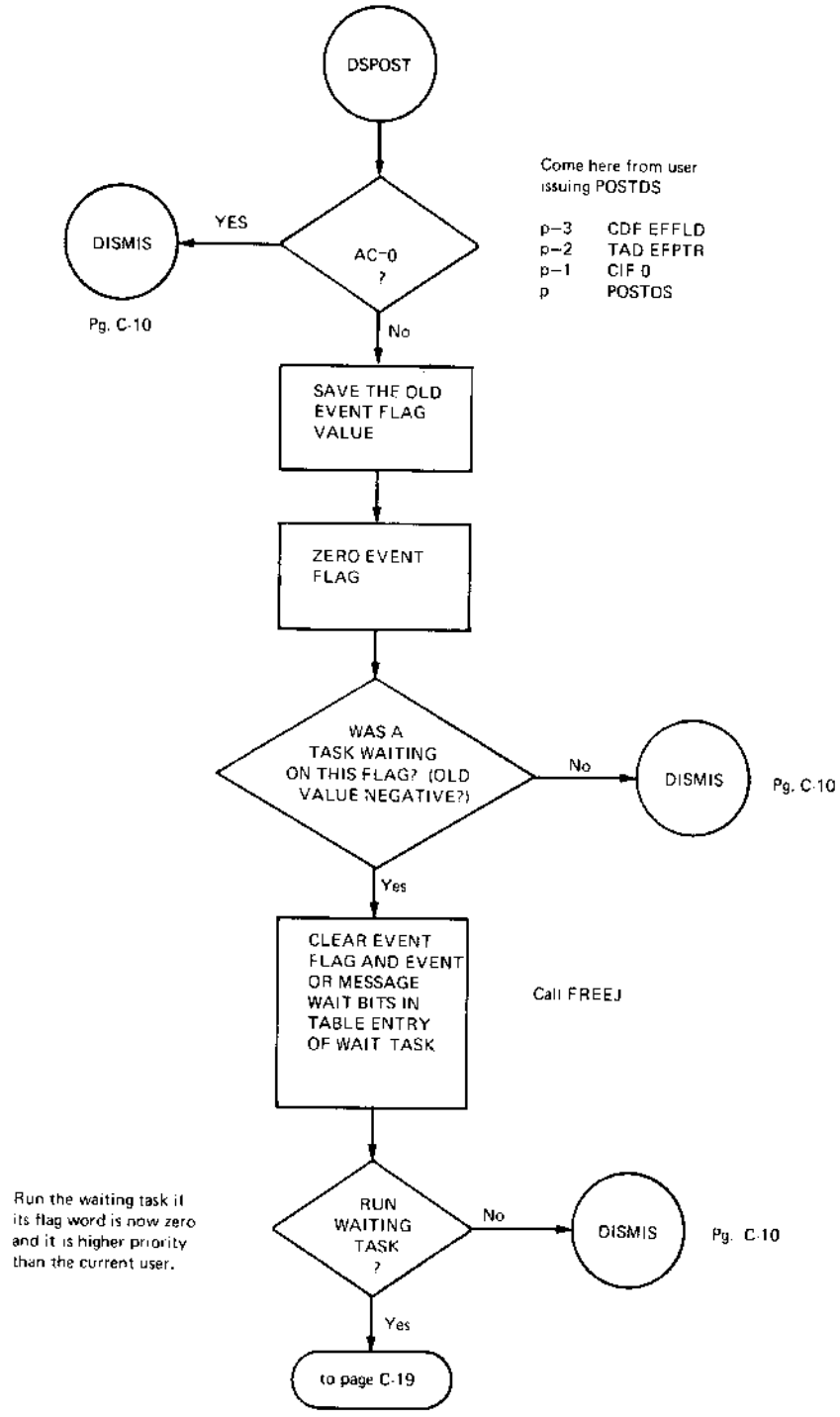


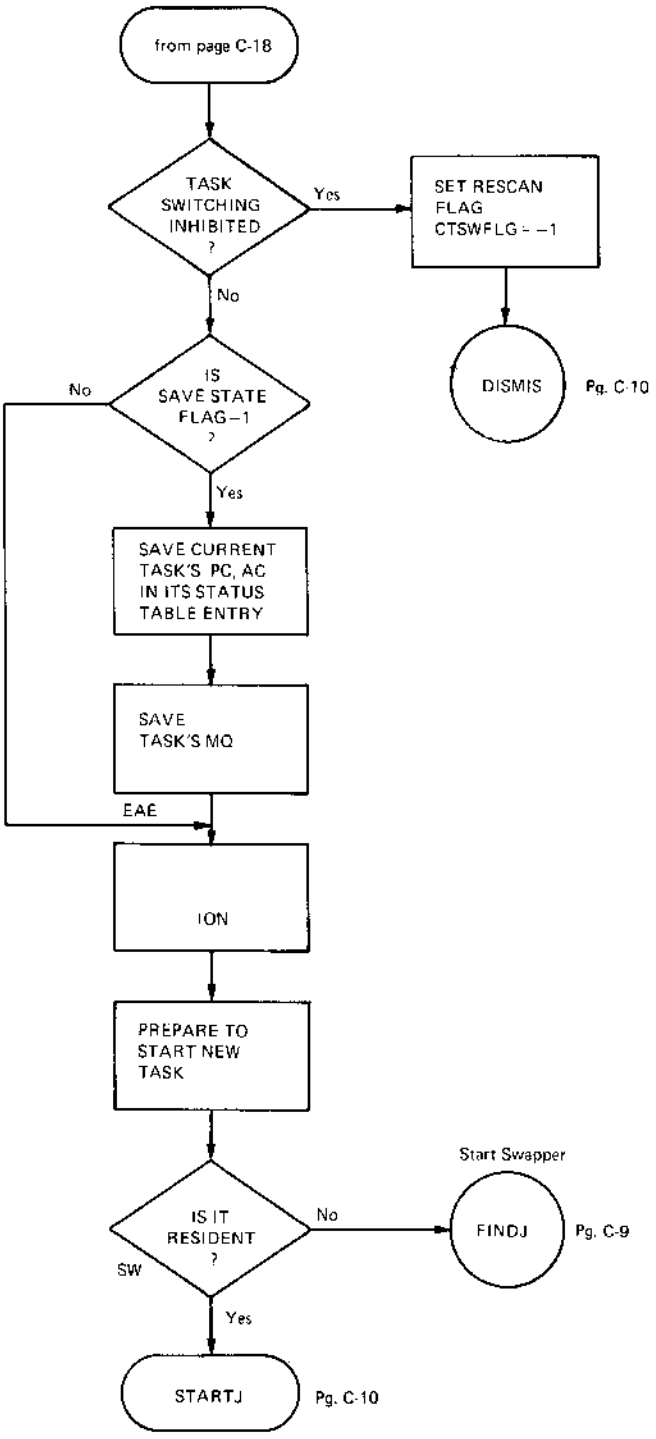
Calling Sequence:

p–1	TAD TASK#
p	CAL
p+1	DERAIL
p+2	SUBROUTINE
p+3	RETURN

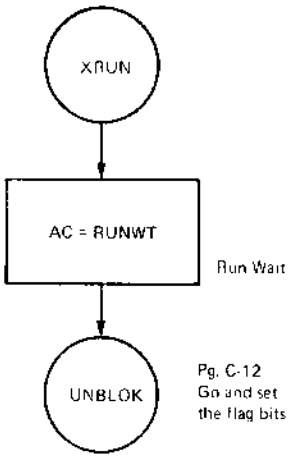

INTERRUPT ROUTINE
TIME-SHARING INTERRUPT ?
Yes
GO TO OS/8 SUPPORT
If present
No
SAVE THE MACHINE STATUS
USERSK
USER-INSERTED SKIPS FOR ANY CRUCIAL DEVICES
GO TO APPROPRIATE INTERRUPT MODULE
Yes
KL8A INTERRUPT ?
No
POWER FAILURE ?
Yes
No
GO TO CLOCK HANDLER
Yes
CLOCK ?
No
If present
GO TO FIRST USER INTER-RUPT MODULE
SET UP LOC 0 FOR POWER FAIL RESTART
HALT
POWER-UP RESTART
AC = POWER-UP EVENT FLAG ADDRESS
DSPOST
Pg. C-18

POSTDS


DSPOST
Come here from user issuing POSTDS
p−3 CDF EFFLD
p−2 TAD EFPTR
p−1 CIF 0
p POSTDS
AC=0 ?
YES
DISMIS
Pg. C-10
No
SAVE THE OLD EVENT FLAG VALUE
ZERO EVENT FLAG
WAS A TASK WAITING ON THIS FLAG? (OLD VALUE NEGATIVE?)
No
DISMIS
Pg. C-10
Yes
CLEAR EVENT FLAG AND EVENT OR MESSAGE WAIT BITS IN TABLE ENTRY OF WAIT TASK
Call FREEJ
Run the waiting task if its flag word is now zero and it is higher priority than the current user.
RUN WAITING TASK ?
No
DISMIS
Pg. C-10
Yes
to page C-19


from page C-18
TASK SWITCHING INHIBITED ?
Yes
SET RESCAN FLAG CTSWFLG = −1
DISMIS
Pg. C-10
No
IS SAVE STATE FLAG−1 ?
No
Yes
SAVE CURRENT TASK'S PC, AC IN ITS STATUS TABLE ENTRY
SAVE TASK'S MQ
EAE
ION
PREPARE TO START NEW TASK
Start Swapper
IS IT RESIDENT ?
SW
No
FINDJ
Pg. C-9
Yes
STARTJ
Pg. C-10

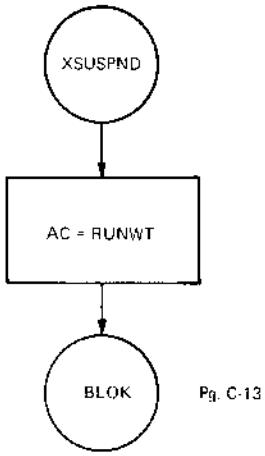
RUN



Calling Sequence:

```
p-1     TAD TASK=
p       CAL
p+1     RUN
p+2     RETURN
```

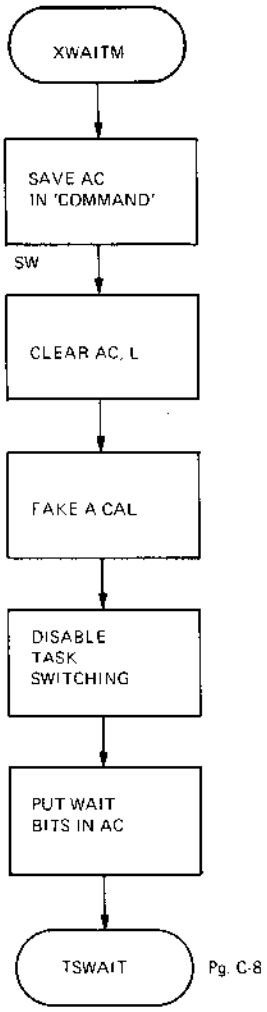
SUSPND

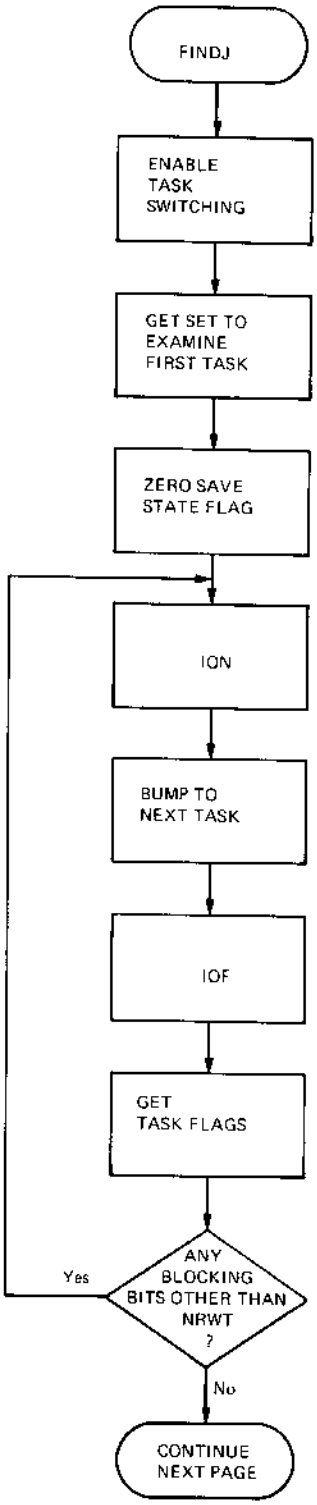


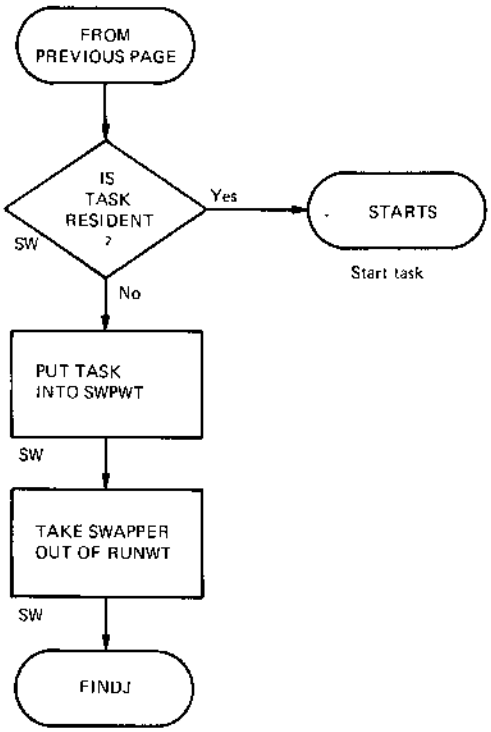
Calling Sequence:

```
p-1     TAD TASK=
p       CAL
p+1     SUSPND
p+2     RETURN
```

WAITM


XWAITM
SAVE AC
IN 'COMMAND'
SW
CLEAR AC, L
FAKE A CAL
DISABLE
TASK
SWITCHING
PUT WAIT
BITS IN AC
TSWAIT
Pg. C-8


FINDJ
RTS/8 SCHEDULER
FIND A TASK TO RUN
ENABLE
TASK
SWITCHING
GET SET TO
EXAMINE
FIRST TASK
ZERO SAVE
STATE FLAG
ION
BUMP TO
NEXT TASK
IOF
GET
TASK FLAGS
ANY
BLOCKING
BITS OTHER THAN
NRWT
?
Yes
No
CONTINUE
NEXT PAGE


FROM
PREVIOUS PAGE
IS
TASK
RESIDENT
?
SW
Yes
STARTS
Start task
No
PUT TASK
INTO SWPWT
SW
TAKE SWAPPER
OUT OF RUNWT
SW
FINDJ

APPENDIX D

RTS/8 ASSEMBLY ERROR MESSAGES

Certain user errors are caught at assembly time. They produce standard PAL8 error diagnostics on the terminal of the form

```
IC
US tag
IC
```

where the tag specified indicates the type of error as described below.

Tag	Module	Possible Error
MCRBLK	MCR.PA	MCR was declared nonresident (MCRPRT defined) and MCR origin was incorrectly a multiple of 400. Nonresident portion of MCR is second page which must start on a block boundary. Fix: Redefine MCRORG.
SYSERR	several	System error; should not occur unless user modified RTS/8 sources. Fix: See comments on source line which generated the error.
TBLERR	RTS8.PA	Internal Executive tables were generated incorrectly. See source.
CURBIG	PARAM.PA	User task specified a value for CUR which was larger than HGHFLD. Fix: Redefine CUR.
RATERR	CLOCK.PA	HERTZ is not a multiple of SHERTZ. Fix: Redefine HERTZ or SHERTZ in parameter file.
TODERR	CLOCK.PA	SHERTZ is too large. SHERTZ must be less than 192 (decimal). Fix: Redefine SHERTZ in parameter file.

Tag	Module	Possible Error
NOKL8A	KL8ASR.PA	The symbol KL8A was not defined in the parameter file. Fix: Set symbol KL8A in parameter file equal to number of physical KL8-A's present.
KL0ERR.	KL8ASR.PA	The symbol KL8A was set equal to 0 yet the file KL8ASR.PA was assembled. Fix: Do not assemble KL8ASR.PA if KL8-A support is not desired or redefine KL8A in the parameter file.
HITMON.	KL8ASR.PA	The KL8-A service overlaid location 7600 in field 0. Ignore this error if you do not want to preserve OS/8 resident code. Fix: Redefine KL8ACT in parameter file.
NOSWAP	PARAM.PA	A nonresident task was assembled in a system with no swapper. Fix: Define SWAPPER in parameter file or undefine PARTNO in user task.
SWPRIO	PARAM.PA	A nonresident task was given higher priority than the swapper. Fix: Change priority of swapper or user task.
FLDERR	PARAM.PA	Some parameter representing a field number times 10 was not in the correct form (e.g. HGHFLD, MCRFLD, etc.) Fix: Correct value of parameter in parameter file.
CURERR	PARAM.PA	One of the symbols CUR, CUR2, or CUR3 was not of the proper form. Fix: Correct value in user task to 10 times field of task.

APPENDIX E

EXECUTIVE INTERNAL TASK TABLES

The Executive uses five internal tables to maintain information about the tasks in the system. Each task's task number is used as an index into the first four tables to retrieve and update information for that task. The internal tables are as follows:

1. The Task State Table (TSTABL) - contains 4-word entries holding the most recent contents of CPU registers for each task as follows:

   Word 1 - contains the link (bit 0)
   the Greater Than Flag (bit 1) -if flag exists on machine being used
   the User Mode Flag (bit 5) -if flag exists on machine being used
   the Instruction Field (bits 6-8)
   the Data Field (bits 9-11)

   Word 2 - contains the contents of the Program Counter (PC)

   Word 3 - contains the contents of the Accumulator (AC)

   Word 4 - contains the contents of the Multiplier Quotient (MQ) register if the system has been assembled to save the MQ.

   Whenever the system executes a task, it loads the contents of the task's Task State Table entries into the corresponding CPU registers. Whenever a task stops executing, its Task State Table entries are set to the new contents of these registers. The Task State Table is located after the Message Table, that is, at location NTASKS+2^2+MSGTBL-4 in field 0.

   Example:

   Consider the following TSTABL entries for a task.

   ```
   4012
   3376
   1234
   0211
   ```

   The task is interrupted just as it is about to execute the instruction at location 3376 (entry 2) of field 1 (bits 6-8 of entry 1). At this point, the contents of the CPU registers for this task are entered in the TSTABL. The AC is 1234 (entry 3), and the MQ is 0211 (entry 4). The link is set (bit 0 of entry 1), and the data field is 2 (bits 9-11 of entry 1).

2. The Task Flags Table (TFTABL) - contains 1-word entries holding various flags (bits) for each task to determine whether the task is runnable. A task is runnable only if its Task Flags Table entry contains zero. Each flag (bit) which is set in a nonzero word indicates a reason why the task cannot run. The currently defined flags, if set, and their meanings are as follows:

Octal	Symbolic	Meaning
4000	NONRWT	Nonresident Wait - This task cannot run because it is not in memory.
2000	EFWT	Event Flag Wait - This task is waiting for an Event Flag (which contains a WAITING value corresponding to this task) to be POSTed.
1000	RUNWT	Run Wait - This task is waiting for a RUN ER to be executed with its number in the AC, or for the operator to type "REquest task" to the Monitor Console Routine (see Chapter 6).
0400	SWPWT	Swap Wait - This task cannot run because it is in the process of being brought into memory.
0200	EORMWT	Event or Message Wait - This task is waiting for an Event Flag to be set or a message to arrive, whichever happens first.
0100	USERWT	User Wait - This bit is reserved for use by user-written tasks. RTS/8 does not use this bit.
0040	ENABWT	Enable Wait - This task is waiting to be Enabled. Use of this bit is restricted to the Monitor Console Routine for the "ENable task" and "DIsable task" commands. (See Chapter 6).
0020	MSGWT	Message Wait - This task is waiting to be sent a message.
0010	NETWT	Reserved for future use.
0001	DNEWT	Does Not Exist Wait - This task cannot run because it is nonexistent.
0000	-	Task is runnable.

The Task Flags Table is located after the Task State Table, that is, at location NTASKS+2^4+TSTABL-1 in field 0.

Examples:

1. If the TFTABL entry for a task is

    1000

the task is waiting to run.

2. If the TFTABL entry for a task is

   0440

   the task was disabled from running by the operator at the MCR terminal while the nonresident portion of the task was waiting to be swapped in.

3. If the TFTABL for a task is

   0000

   the task is runnable. However, this task may not run if a task of higher priority has precedence.

3. The Task Input Message Queue Header Table (MSGTBL) - contains 2-word entries that represent the "head" (start) of the input message queue for each task:

   Word 1 - if zero, there are no messages in the queue; if non-zero, the word is a CDF to the field of the first message in the queue.

   Word 2 - if word 1 was not zero, this word is a pointer to the address of the first message in the queue.

   The Message Table is located at the end of the RTS/8 Executive in field 0.

   Example:

   Consider the following MSGTBL entries for a task.

   6211
   2044

   Since the first entry is a nonzero, there are messages in the input queue waiting for this task to receive them. The first entry is a CDF instruction to field 1. The second entry is a pointer indicating that the first message begins at location 2044 in field 1.

4. The Residency Table (RESTBL) - contains 2-word entries for each nonresident task.

   Word 1 - contains a pointer to the task's Partition Table entry in bits 0 through 9. Bit 10 is set if a task is checkpointable, and bit 11 is set if a task is writeable. Checkpointable and writeable tasks are defined in Section 5.1.

   Word 2 - contains the absolute block address (plus 1 to allow for the core control block) of the task's core image on the swap device.

   The Residency Table is located after the Task Flags Table, that is, at location TFTABL+NTASKS+2 in field 0.

Example:

Consider the following RESTBL entries for a task.

```
1611
0124
```

This task has a nonresident portion. The 4-word partition table entry used by this task begins at location 1610. The task is not checkpointable (bit 10 of entry 1 is a 0), but it is writeable (bit 11 of entry 1 is a 1).

The disk-resident portion of this task begins at block 124 on the swap device. (The save image begins at block 123.)

5. The Partition Table (PARTBL) - contains a 4-word entry for each partition. It is indexed into via a partition number.

   Word 1 - contains the length (size) in bits 1-5 and field (bits 6-8) argument of the mass storage device driver call that reads an occupant into the partition with the "WRITE" bit set. Bit 11 of this word is the partition busy flag.

   Word 2 - contains the memory address of the partition.

   Word 3 - contains a pointer to word 1 of the occupant's RESTBL entry.

   Word 4 - unused

   The Partition Table must begin at an address that is a multiple of four. It is located after the Residency Table, i.e., at location NTASKS-SWAPPER^2+RESTBL+3&7774 in field 0.

   Example:

   Consider the following PARTBL entries for a particular partition.

```
5421
1400
1553
0000
```

   This partition is currently in use (bit 11 of entry 1 is a 1) by a task whose 2-word RESTBL entry begins at location 1553 (entry 3). The partition begins at location 1400 (entry 2) of field 2 (bits 6-8 of entry 1). The partition is 14 (octal) pages long (bits 1-5 of entry 1).

Figure E-1 summarizes the internal task table structure of the Executive. The Residency Table and Partition Table are optional in that they are used only when nonresident tasks are employed. The exact location of these tables in memory depends on the number of tasks and other parameters in the parameter file. They can be found for a particular assembly under the "System Locations:" heading at the end of the parameter file assembly listing.

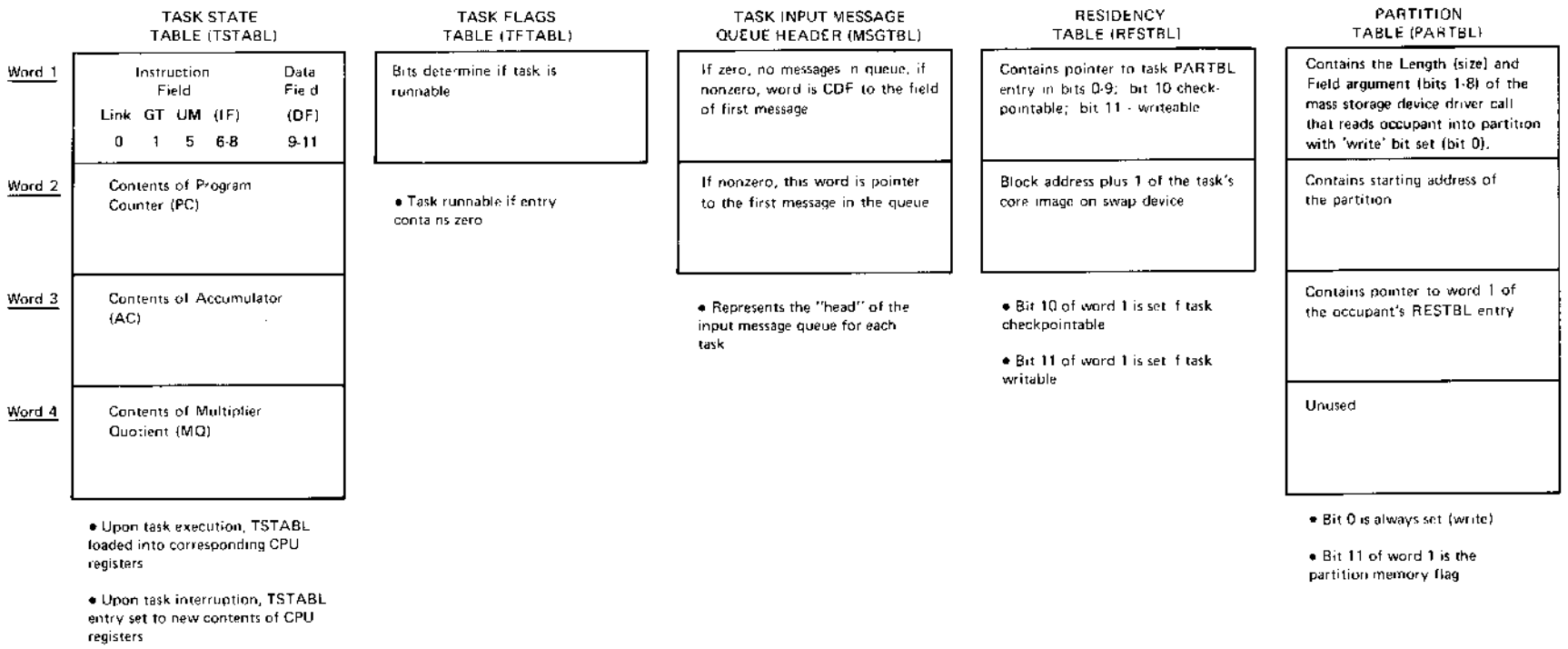


Figure E-1 Executive Internal Task Table Structure

GLOSSARY

Accumulator	The register in which the arithmetic operations are performed (abbreviated AC).
Analog	Representation of information by continuous variables.
Analog Channel	An UDC/ICS functional device.
Argument	A variable or constant which is given in the call of a subroutine as information to it; a variable upon whose value the value of a function depends; the known reference factor necessary to find an item in a table or array (i.e. the index).
Assembler	A program which translates symbolic op-codes into machine language and assigns memory locations for variables and constants.
Auto-index register	Whenever one of the absolute locations from 0010 through 0017 in any memory field is addressed indirectly, the contents of that location is incremented by one, rewritten in the same location, and used as the effective address of the current instruction.
Auto-restart	The ability to start the CPU automatically on power-up.
Baud	A unit of measure of data flow (one bit per second).
Bit	A binary digit (each PDP-8 word is composed of 12 bits).
Bit Map	A method of keeping track of used and unused entities by assigning one bit in a table to each entity.
Block	A set of consecutive machine words, characters or digits handled as a unit, particularly with reference to input and output; an OS/8 block is 400 octal contiguous words; also to inhibit a process from continuing.
Block Gap	The blank space between blocks on a recording medium.
Blocking Bits	Bits in an RTS/8 Monitor Table which specify why a given task is blocked.
Byte	A group of binary digits usually operated upon as a unit.
Cassette	A magnetic tape device used for program and data storage.

Central Processing Unit	The unit of a computing system that includes the circuits controlling the interpretation and execution of instructions (abbreviated CPU).
Checkpointable Task	A task is checkpointable if it may be swapped out of memory automatically, without its consent, to make room for higher priority tasks.
Clear	To erase the contents of a storage location by replacing the contents, normally with zeros.
Clock	A time keeping or measuring device within the computer system; provides periodic interrupts.
Communication Region	Locations 20-27 of every field, used to simplify passing executive request across field boundaries.
Compute Bound	Requiring extensive (or total) use of the CPU relative to other hardware elements (such as I/O devices).
Configuration	The number and types of hardware present on a system.
Contact Channel	A UDC/ICS functional device.
Contiguous	Code which resides in memory immediately adjacent to other sections of code.
Controller	The circuitry that controls a device.
Core Image File	A file in core image format (i.e., a 'picture' of core); also known as SAVE file.
Core Storage	The main high-speed storage of a computer in which binary data is represented by the switching polarity of magnetic cores.
Counter Channel	A UDC/ICS functional device.
CPU Registers	High-speed circuitry used to store information affecting the operation of the CPU (e.g., PC, IF, DF).
Data	A general term used to denote any or all facts, numbers, letters and symbols. It connotes basic elements of information which can be processed or produced by computer.
Data Field	A 3-bit register which determines the memory field from which operands are taken in indirectly addressed instructions (abbreviated DF).
Debug	To detect, locate and correct mistakes in a program.

Deferred Actions	Actions which are considered low-priority and are not performed until higher-priority actions are serviced.
Deferred Requests	Requests which are considered low-priority and are serviced after high-priority requests.
Derail	To transfer control or execution of a specified task to a subroutine.
Device Codes	Numbers assigned to each device in the system.
Device Status Register	A register which contains the current status of a device.
Digital	Representation of information by discrete units.
Digital Channel	A UDC/ICS functional device.
Driver	See Handler.
Dynamic	Pertaining to a quantity that is affected by some condition (such as time) and is therefore relative to the condition; also refers to features at system run-time.
Entry Point	The location in a routine to which control can be transferred and execution begun.
Event Flag	The location which contains the status of an event (event being either a result of some operation, or a physical occurrence).
Executive	The program which controls the execution of other programs or routines.
Executive Requests	Means of communication between tasks and the RTS/8 Executive.
Field	A division of memory on a PDP-8 computer referring to a 4K section of memory.
File Gap	A fixed length of blank tape separating files on a recording medium; generally several times the size of a block gap.
Functional Devices	The devices available for use under the UDC/ICS handler.
Gain	An increase in signal power.
Generic Codes	Codes used to identify which type of UDC/ICS functional device caused an interrupt and to direct program control to service routines.
Handler	A routine which is designed to control the operation of a device.
I/O Bound	A condition in which a process is performing much I/O but using very little CPU time.

Indirect Address	An address in a computer instruction which indicates a location where the address of the referenced operand is to be found.
Initialization Code	Code which sets counters, switches, and addresses to zero or other starting values at the beginning of or at prescribed points in a computer routine.
Input Buffer	A section of memory used for storage of input data.
Instruction Field	A register which holds the contents determining from which field the operand of a directly addressed instruction should be taken (abbreviated IF).
Interactive	Highly responsive to real-world inputs.
Interface	The common hardware and/or software boundary between two devices or systems.
Interlock Scheme	Arranging the control of devices so that their operation is interdependent.
Interrupt	A break in execution caused by some external event; execution is usually resumed at a later time.
Interrupt Processing Module	A routine which acts upon the external event which caused an interrupt.
Interval Queue	A list of actions to perform, each accompanied by the interval of time which is to elapse from the previous action to the current one.
Line-frequency Clock	A clock whose ticking occurs at a multiple of the power line frequency.
Link	A one-bit register in the PDP-8; an address pointer generated automatically by the PAL8 assembler to indirectly address an off-page symbol.
Logical OR	A logical function of two or more inputs which is true whenever either input is true.
Loop	A sequence of instructions that is executed repeatedly until a terminal condition prevails.
Mass Storage Device	A device such as disk or DECtape which stores large amounts of data readily accessible to the central processing unit.
Master Parameter File	A file included in the distributed sources of RTS/8 which the user can edit to indicate parameters specific to his system configuration.
Memory Address	A register which holds the address specified by a memory reference instruction.

Message	A contiguous area of memory which contains information about execution of tasks.
Message-driven	An RTS/8 task is called message-driven if it only executes in response to messages received from other tasks.
Mnemonic	Alphabetic representation of a function or octal machine instruction.
Module	A routine which handles a particular function.
Monitor Console Routine	The Monitor routine which provides the user with functions which allow him to control, inspect, and debug his system.
Multiplier Quotient	A 12-bit register used in conjunction with the accumulator to perform mathematical operations (abbreviated MQ).
Multi-programming	Two or more programs (tasks) in memory at the same time which execute alternately depending on the current state of the system.
Nonresident Task	A nonresident task is a task or a portion of a task that is swapped into memory when it becomes executable.
No-op	No operation occurs; control proceeds to the next instruction in sequence.
Null Characters	Characters with ASCII code 000.
Overflow	A condition that occurs when a mathematical operation yields a result whose magnitude is greater than the data presentation is capable of storing.
Pack	To conserve memory by combining information.
Page	A 128-word (decimal) section of PDP-8 core memory beginning at an address which is a multiple of 200(octal).
Parameter file	A file used to record arguments which may be assigned different values.
Parity Bit	A bit which indicates whether the total number of binary one digits in a word is even or odd.
Pointer	A word containing the address of another word in memory.
Post	To post an Event Flag means to set it to a FINISHED state via an RTS/8 Executive Request.
Posted	One of the states of an event flag, indicating that the event is complete; same as "FINISHED".

Power-fail	An interruption of power to the computer.
Priority Scheme	A scheme by which certain operations or the execution of a set of instructions is given preference over other operations.
Program Counter	A register which contains the address of the next instruction to be executed (abbreviated PC).
Prompting Character	A character which prints on the console terminal and cues the user to perform some action.
Queue	A waiting list (e.g., a queue of programs waiting for processing time).
Ready Flag	A bit which a device controller sets when it is ready to accept commands from the CPU.
Real-time System	A system in which computation is performed while a related physical process is occurring so that the results of the computation can be used in guiding or measuring the physical process.
Receiver	The task which has received a message from another task (sender).
Record	A collection of related items of data treated as a unit.
Record Header Area	An area of at least 40(octal) words in length which contains information necessary to perform cassette operations.
Ring-buffer	A storage area for data accessed on a first-in, first-out basis. Similar to Queue, but usually involves storage of character codes.
Scheduling	The operation of sharing resources among computing tasks.
Sender	The task which has sent a message to another task (receiver).
Sign Bit	The bit which contains the sign of a number.
Simulate	To represent the function of a device, system or program with another device, system or program.
Skip Chain	An instruction sequence which determines the source of an interrupt request on a PDP-8; contains one or more tests for each possible (hardware) interrupt condition.
State	A complete description of the condition of a piece of hardware or of a task.
Status	That portion of the state which other devices or tasks might be interested in.

Subchannel	One of the 4 channels of a UDC/ICS analog functional device.
Suspend	To temporarily halt execution of a task while another task of higher priority runs.
Symbols	Names which can be assigned values or which can be used to indicate specific locations in a program.
Synchronization	A means of coordinating tasks (through event flags) so that one task executes while others wait.
System Ticks	An RTS/8 convention designed to obviate the tasks in the system from knowing the frequency of the clock.
Task	A task is a routine which performs a specific function. A task may be "resident" or "nonresident". A resident task is permanently located in memory. A nonresident task is loaded into memory as it is needed and can be overlaid after its completion.
Task Flags Table	A table of 1-word entries in the RTS/8 Executive whose contents determine whether or not a task is runnable.
Task Input Message Queue Header Table	A table of 2-word entries which represent the head of the input message queue for each task.
Task Number	A unique number between 1 and 63 (decimal) assigned to each task in an RTS/8 system.
Task State Table	A table of 4-word entries which contains the ost recent contents of CPU registers for ach task.
Task Switching	The act of stopping execution of one task and ontinuing execution of another from the point that it was last stopped.
Terminal	A peripheral device in a system through which data can enter or leave the computer.
Utilities	Routines to perform non-Monitor related functions.
Word	In the PDP-8, a 12-bit unit of data which may be stored in one addressable location.

INDEX

READER'S COMMENTS

NOTE: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. Problems with software should be reported on a Software Performance Report (SPR) form. If you require a written reply and are eligible to receive one under SPR service, submit your comments on an SPR form.

Did you find errors in this manual? If so, specify by page.

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

Is there sufficient documentation on associated system programs required for use of the software described in this manual? If not, what material is missing and where should it be placed?

Please indicate the type of user/reader that you most nearly represent.

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Non-programmer interested in computer concepts and capabilities

Name _____ Date _____

Organization _____

Street _____

City _____ State _____ Zip Code or Country _____

Please cut along this line.

Fold Here

Do Not Tear - Fold Here and Staple

FIRST CLASS
PERMIT NO. 33
MAYNARD, MASS.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

Postage will be paid by:

digital

Software Documentation
146 Main Street ML5-5/E39
Maynard, Massachusetts 01754

RW = 0 Read data from floppy disk.

= 4000 Write data to floppy disk.

PAGES = Specifies the number of pages to transfer (times 100 octal). Pages = 0 transfers 40 pages (a full field). This value takes the range 0-37 in bits 1-5 of this word. PAGES is ignored if CODE = 4000. In that case, either 100 (octal) 12-bit words or 200 8-bit bytes (from 200 words) are transferred depending on MODE.

FIELD = Specifies the field of buffer (times 10 octal). Bits 6-8 of this word have the range 0-7.

BUFADD = Specifies the address of the first word of the buffer containing data. Field of buffer is determined by FIELD. Length of buffer depends on PAGES if CODE = 0 or on MODE if CODE = 4000.

BLOKNO = Represents first logical OS/8 block to transfer if CODE = 0. Each OS/8 block consists of 4 sectors. Track 0 is ignored and a 2-to-1 interleave scheme is employed. If CODE = 4000, this word contains physical track and sector numbers in the format TTTTTTSSSSS.

STATUS = Receives the status of the operation upon completion. If negative, a hard error has occurred. If 0, no error has occurred. This word may be positive nonzero only if DEL = 2000.

The meaning of the STATUS bits is as follows:

Bit	Meaning if 1
0	Hard error
1-3	Not used by controller
4	Not used by RTS/8
5	Deleted data indication
6-7	Not used by controller
8	Reserved for future use by controller
9	INIT done (can occur after temporary power failure to controller)
10	Parity error
11	CRC error

NOTE

On power fail restart, the INIT error might occur. When this error occurs, the calling task should send the I/O message again.