Prime Computer User Guide

For
Real Time Operating System

PRIME COMPUTER, INC., 145 PENNSYLVANIA AVENUE, FRAMINGHAM, MA. 01701

RTOS USER GUIDE

ORIGINAL EDITION OF MARCH 1974

Reprinted February 1975

INSTRUCTIONS

This is a partial reprint of the original document. To form a complete RTOS manual, Addendum A (supplied separately) must be combined with this document in the following sequence:

Original (This Document)	Addendum A	Contents
	X	Title Page, Contents and Foreword
	X	Section 1
X		Section 2
	X	Section 3
X		Section 4
	X	Section 5
	X	Section 6
X		Section 7
X		Section 8
	X	Section 9
X		Appendix A
	X	Other Appendices

SECTION 2

ACTION OF EXECUTIVE

This section describes the standard RTOS Executive, with emphasis on the functions of the Scheduler, Function Handler, and Interrupt Handler. The Communication, Coordination and Mass Storage options, the Error Print routine, the Clock Program, and The System Loader are also discussed. Detailed flow charts are included at the end of the section.

SUMMARY OF EXECUTIVE CONTROL FLOW

Figure 2-1 provides an overview of program control through the main components of the Executive. Main entry and exit points are identified by the same labels (INP1, SC, FE, etc.) that appear on the detailed flow charts and the Executive assembly language listing.

After a manual start, the executive initialization routine (INP1) initializes the Real time Clock and ASR interrupts, sets the addressing mode, etc., and jumps to the user's initialization routines, if present. Control then passes to the Scheduler.

The scheduler determines whether there is a program ready for execution by examining entries in the SPLT table of SIM. If several programs are ready, they are started in the order of priority determined by the SPLT table of SIM. A program's readiness for execution is determined by bit settings in the status word of the program's SPLT table entry. The main categories are:

Requested	Ready for execution beginning at the program's starting location. Requested status is set by a Request Program or Connect Clock executive function.
Scheduled	Ready to resume execution at a label (usually other than the starting location). A scheduled program must be in memory and waiting. Scheduled status is set by a Schedule Label or Connect Clock executive function, or if there is a label present in the A register during the return from interrupt response code.
Running	In memory and executing.

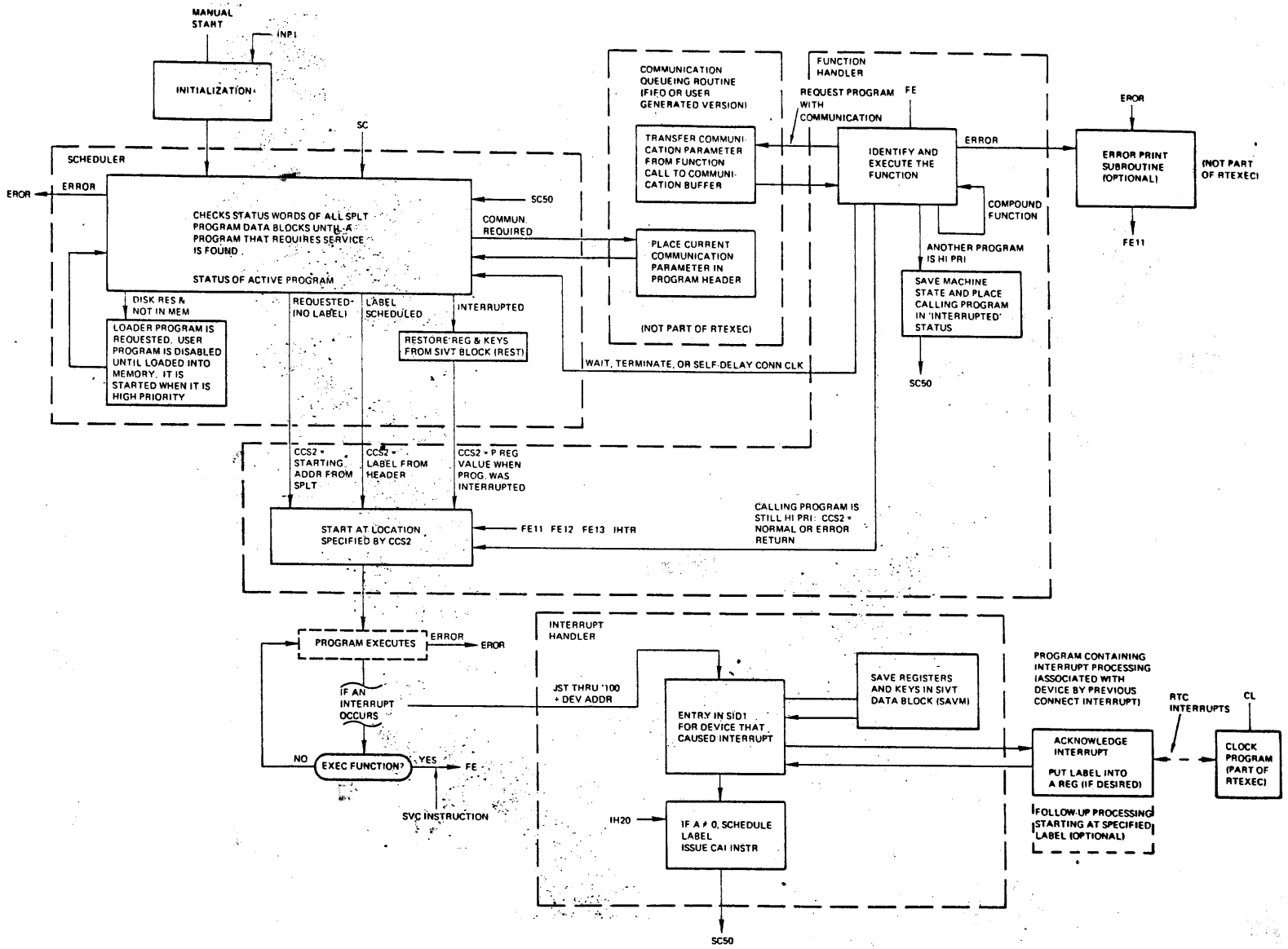


Figure 2-1. Control Flow Through Executive Software

Interrupted	In memory but not executing because of a hardware interrupt or scheduling of a higher priority program during an executive function. The CPU registers and keys that were in effect at the time of the interrupt are saved in a data block of the SIVT table of SIM.
Active	Any of the above.
Inactive	None of the above.
Waiting	In memory but not executing as a result of a Wait executive function.
Terminated	Having run to completion at least once and ended with a Terminate executive function or Connect Clock (self-delay).
Disabled	Incapable of being made active (a mass store resident program that is requested is disabled until the system loader transfers it to memory).

Program status is discussed in more detail in the description of the Scheduler to follow.

If the current high-priority program is disk resident and not in memory, the scheduler requests the system loader program. When the loader has high priority, it executes and brings the active program into memory. When the active program itself is again high priority, it is started.

The current active program may be started at the starting location (if requested), at a label (if scheduled) or at the location where it was interrupted. Once a user program is executing, it is in full control of of the CPU until a hardware interrupt occurs, an executive function is encountered, or the RTOS error print routine is called. This applies to RTOS device drivers, the clock program, and all utility programs as well as user-generated application programs.

Vectored interrupt mode is used for all RTOS device drivers. Interrupts can occur only when a device has been associated with interrupt response code by a Connect Interrupt executive function. The Connect Interrupt function inserts a pointer to the interrupt response code in the interrupt data block for the interrupting device in the SID1 table of SIM. When the interrupt occurs, control passes immediately to the device's SID1 table entry, which controls the events immediately following the interrupt. The Connect Interrupt function inserts a pointer to the interrupt response code in the interrupt data block for the interrupting device in the SID1 table of SIM. When the interrupt occurs, control passes immediately to the device's SID1 table entry, which controls the events immediately following the interrupt. The first event is a JST to SAVM, an executive subroutine which saves the registers and keys of the interrupted program and then enables higher-priority interrupts. Next is a jump to the response code in the program which connected the interrupt. The response code should be limited to

essential operations such as reading the current data word or enabling the next device cycle, to make interrupt response time as brief as possible. The response code may also place a label (an entry point within the same program) in the A register before returning to the executive. The label can specify a portion of the program that does follow-up processing of the data when time permits. Return is then made to the interrupt handler, which does the equivalent of a Schedule Label executive function (if a label was present), clears the active interrupt, and returns to the scheduler. The scheduler starts processing at the label when the interrupt processing program has high priority.

Immediately after the system is initialized, the Real Time Clock begins to issue interrupts at a rate configured into the CLK3 entry of SIM, (typically every 50 ms). Each RTC interrupt causes execution of the clock program, an independent interrupt-driven program supplied as a part of the Executive, starting at entry point CL. The clock program maintains a time of day count and checks for programs scheduled for execution by Connect Clock executive functions. Such programs are set to Requested status and become candidates for startup by the scheduler according to their priority.

Executive functions start with a SVC instruction (an interrupt through location '65, which points to function handler entry point FE). The function handler identifies the function and performs the appropriate actions. All executive functions cause one or more changes in program status (to be described in more detail elsewhere) to the calling program or another program specified in the function call. Certain functions return to the scheduler since they do not require a return to the calling program. The scheduler then starts whatever program is high in priority at the time. Most functions can return directly to the calling program, either to the normal return or an error return point. If a program that is higher in priority than the calling program is requested, the function handler places the calling program in 'interrupted' status, saves the P register as if the calling program itself had been interrupted, and returns to the Scheduler.

When communication is required by a program that is being made active by executive function, the function handler calls a part of the system's communication queueing subroutine. (FIFO is the standard version supplied with RTOS.) The queueing routine transfers a communication parameter from the function calling sequence to a communication buffer area defined by the SPCT table of SIM. Later, when the scheduler is ready to start the program requiring communication, the parameter is moved from the buffer area to the header of the program.

The scheduler and function handler make use of the standard error print program to print messages to the operator when certain types of errors occur. User programs may make use of the error printout as well.

STARTUP AND INITIALIZATION

See Section 9

SCHEDULER

The scheduler is the main control element of the Executive. Its purpose is to identify and start execution of the highest priority program that has been requested or otherwise placed in an active status.

The scheduler is executing and in control of the CPU under the following condition:

a. No user programs are requested or active.

b. Immediately after a program has executed a Terminate function.

c. Immediately after all interrupt code has been executed (for example, after the Real-Time Clock routine has done all processing following an RTC interrupt).

d. Immediately after a program has executed a wait function.

Because of item c., the scheduler is entered at least once for every real-time clock interrupt period (50 ms for a 60 Hz clock and a CLK3 entry of -3; 40 ms for a 50 Hz clock and a CLK3 entry of -2).

During execution, the scheduler uses the pointers from the SPET table of SIM (Section 3) to access program data blocks from the associated SPLT table. By examining the program's status word in the SPLT table data block, the scheduler can tell whether the program is to be started. If not, the next SPLT entry is examined with priority being established by the order of pointers in the SPET table. If no active or requested program is found, the scheduler loops through SPLT continuously until an interrupt, a keyboard function, or a clock-controlled event causes a program to be requested or active. The scheduler starts such a program at its starting location or at a label (if scheduled).

Program Status

Figure 2-2 shows the relationship between SPET and SPLT and identifies the entries in a typical program data block.

Each executable user program and device driver in the system is assigned a status, determined by bit settings in the status word, part of the program's entry in the SPLT table. Bit assignments are shown in Figure 2-3. Initially, the status of all programs is predetermined by the status words configured in SIM by the user. Thereafter, the executive alters program status in response to executive functions and utility keyboard functions. Primary program status is determined by bit 9 (Started), bit 10 (Requested), bit 11 (Interrupted) and bit 12 (Waiting).

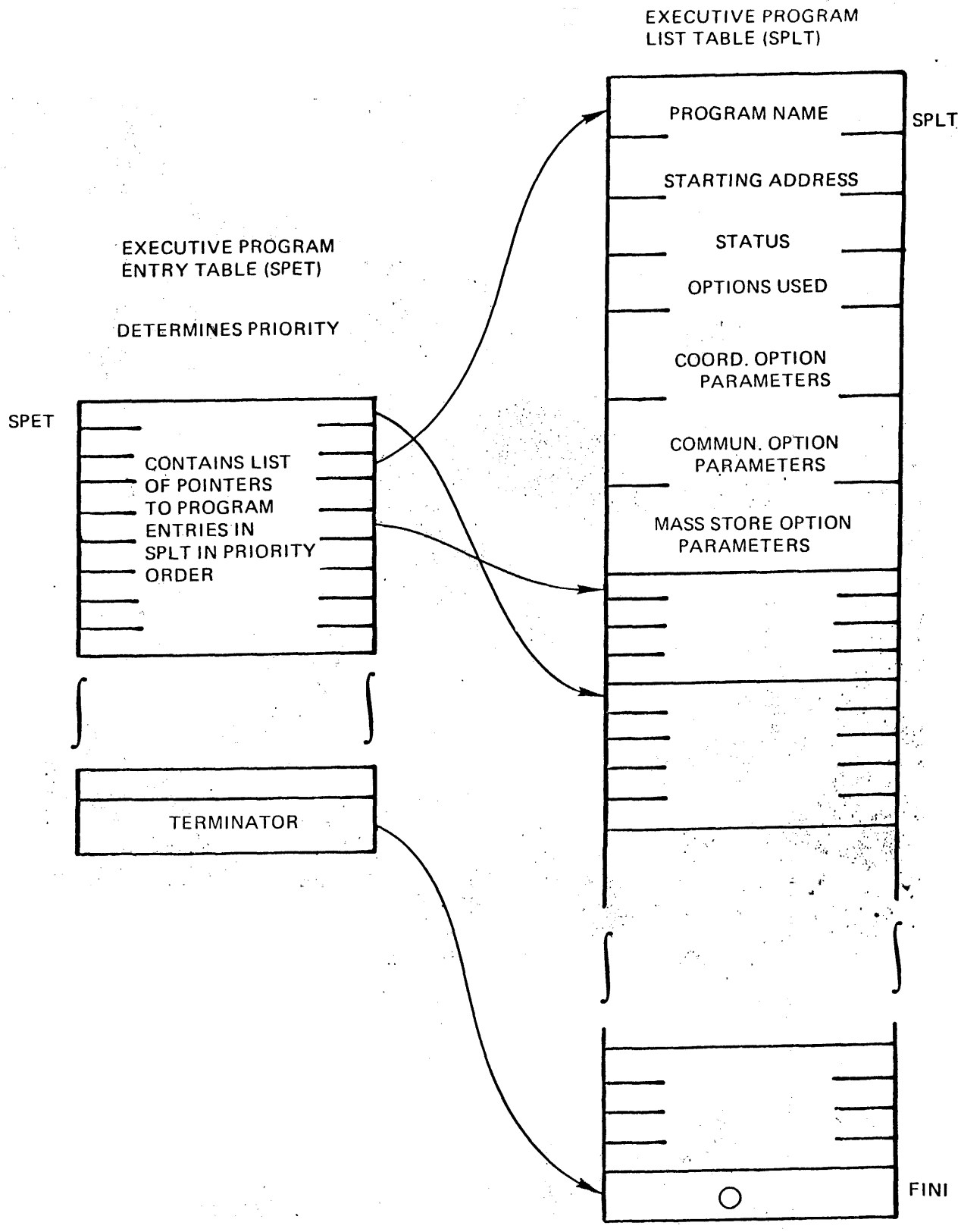


Figure 2-2. SIM Tables Used by Scheduler

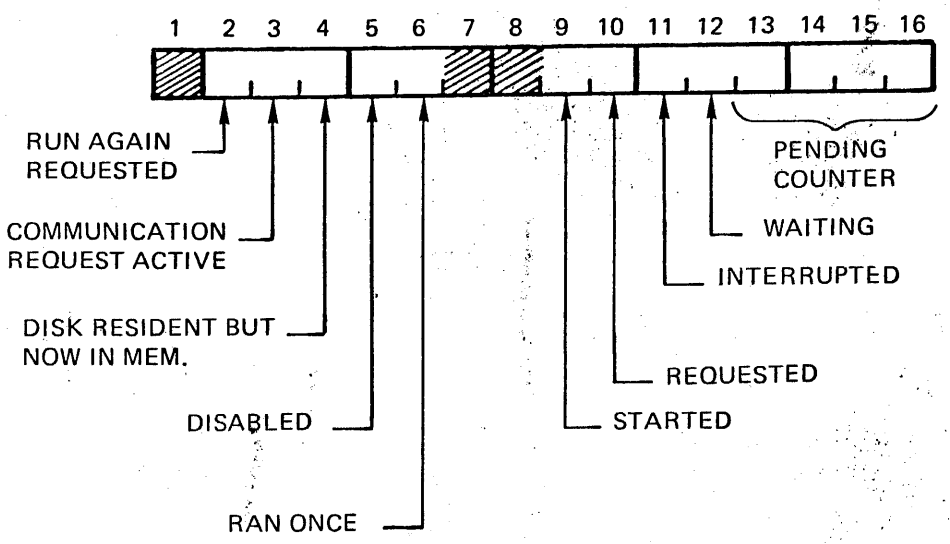


Figure 2-3. Status Word Bit Assignments

Requested: A program is 'Requested' if bit 10 of its status word is set. It remains requested until the scheduler identifies it as the highest priority program requesting service. The program is then started (placed in 'Running' status) if there are no coordination conflicts. A program may be placed in requested status by one of the following:

a. A Request Program or Schedule Label executive function (Section 4) or an RP utility keyboard function (Section 5).

b. A Connect Clock executive function or CC utility keyboard function.

c. Setting the 'Requested' bit and a pending count of 1 in the status word during configuration of SIM.

Running: A program is 'Running' when it is in memory and executing. Bit 9 of the status word for a running program is set, and bit 10 (Requested) is reset. The scheduler places a program in running status when it is the highest priority active program, assuming there are no coordination conflicts or other problems that will prevent execution.

The location at which the scheduler starts a running program depends on its previous status:

Previous Program Status	Starting Location
Requested	Starting address specified in program's data block in SPLT table of SIM.
Interrupted	P register value saved, when interrupt occurred, in an interrupt data block in the SIVT table of SIM.
Waiting	At a label specified by a Schedule Label, Connect Clock, or Connect Interrupt executive function, or at a label placed in the A Register during interrupt response code.
Running and returning from an Executive Function	Executive Function's normal or error return.

Waiting: A 'Waiting' program is in memory but not executing. Bit 12 of its status word is set. A program can place itself in the waiting state by a Wait executive function or when it delays itself by a Connect Clock executive function. A program with a label scheduled for periodic execution by a Connect Clock function is in the waiting state between periods of active execution.

A waiting program can be restarted (placed in 'Running' status) by a Schedule Label executive function in another running program, a connected interrupt, or when a connected clock period comes due. A program that has initiated a self-delay by a Connect Clock function is restarted when the delay has elapsed. The 'Wait' bit is cleared when the program is started at a scheduled label.

Interrupted: A program that was running when an interrupt occurred is set to 'Interrupted' status; bit 11 of its status word is set. The interrupt handler saves the registers and keys (including the P register value at the time of the interrupt) in an interrupt data block in the SIVT table of SIM.

When the scheduler finds that an interrupted program is the highest priority active program and there are no coordination conflicts, it restores the machine state from the proper SIVT data block and starts execution at the saved P register value. The interrupted program resumes where it left off and is in 'Running' status (the 'Wait' bit is cleared and the 'Started' bit is set).

Active: A program is considered active when it is in memory and in interrupted, running, or waiting status.

Inactive: A program is inactive when it is not interrupted, running, or waiting.

Terminated: A terminated program is one that has run at least once and has concluded itself by a Terminate executive function. Bit 6 of the program's status word (the 'Ran once' bit) is set and all other primary status bits are cleared. A terminated program can be returned to active status by a Request Program or Connect Clock function.

Other Status Word Functions

In addition to maintaining primary program status, the status word contains several other optional features that can influence scheduler action.

Run Again Bit: Bit 2 of the status word is set whenever a program is requested while it is in any way active. If this bit is set during a 'Terminate' system function the program is placed in 'Requested' status so that it can run at least once more. (Bit 2 is cleared.)

Communication Request Active Bit: Bit 3 of the status word is set during any Request Program executive function that uses the communication option (described later). Such a function uses part of the system communications routine to queue a communication parameter in a buffer. Before the program is started by the scheduler, another part of the communications routine is used to transfer the parameter from the buffer to the program header, as shown in Figure 2-1. If this bit is set during a 'Terminate' executive function, the program is placed in 'Requested' status to satisfy outstanding program requests that use communication. When all queued requests have been processed, the bit is cleared and the program is terminated.

Memory Resident Bit: Bit 4 of the status word is set when an RTOS-B program that is normally disk-resident is present in memory. If the bit is zero, the scheduler knows that the program must be brought in from the disk before it can be made active. The disk transfer is controlled by the System Loader program which is requested by the scheduler.

Disabled Bit: When status bit 5 is set, a program is not allowed to start even if it is the high priority active program. For example, when there is is a request for a disk resident program that is not in memory (status bit 4 is zero), the scheduler disables the requested program until the system loader program brings it in from disk. The loader clears the disabled bit when loading is complete, thus enabling the program to execute when it has high priority.

Ran Once Bit: Bit 6 is set when the program has successfully run through one complete execution and has ended with a Terminate function. User programs can check this bit before subsequent requests in order to avoid repetition of initializing routines, for example.

Started Bit: Bit 9 is set by the scheduler every time a requested program is started. The bit is tested by several system functions; it means that the program is currently running, waiting, or interrupted. The bit is reset by a Terminate function.

Pending Counter: The pending counter (status bits 13-16) records the number of outstanding program execution passes (scheduled labels) and the potential for further requests.

The counter is incremented every time the program is requested, a label is scheduled, or an interrupt is connected. The counter is decremented each time a program is started from the scheduler (either from a start or a scheduled label) or an interrupt is disconnected.

The counter is tested by the 'Terminate' system function in order to terminate a program correctly.

Option Word

Word 'Label + 3' of a program's SPLT entry is the Option Word, which determines the number of words to follow. For bit assignments, see Figure 2-4.

If bits 14, 15 and 16 are all zero, no options are used and the option word itself is the last word in the program's SPLT entry. An additional word is present in the entry for each option that is used.

If bit 14 is set, the rest of the option word is used to define the size and starting segment for a disk-resident program (RTOS-B).

See Figure 8-5

Figure 2-4. Option Word Bit Assignments

Coordination Option

If the program is to use the coordination option, the coordination word follows the option word in the SPLT block. Bits of the coordination word may be assigned meanings arbitrarily by the user. While the program is running, its coordination word is loaded into the Executive's master coordination word, XCCW. Other programs that use one or more of the same coordination bits are not allowed to run until the first program is complete.

In general, the coordination word is used by the Executive to prevent concurrently running programs from having conflicts in memory occupancy, use of peripherals, or use of non-reentrant subroutines. Typical bit assigments are shown in Figure 2-5. In this example, programs AS, EP, and user program BB all use the ASR, so are mutually exclusive. User programs AA, BB, and CC are shown as disk resident programs that use adjacent memory areas. Only programs AA and CC can be memory resident at the same time. (See Section 6 for more information on this type of coordination.)

Communication Option

The optional communication word is present if bit 15 of the option is set. It provides a place to select a queueing routine and identify a buffer to be used in passing communication parameters. Numbers for subroutines and buffers are determined when the SPCT table of SIM is configured. (See Section 3.) Bit assignments are shown in detail in Figure 2-6. The Relocated Base Sector function is discussed in Chapter 8. The interaction between the Scheduler and Function Handler during communication is discussed later in this section.

Mass Store Option

When bit 14 of the Option Word is set, the last word in the SPLT table is interpreted as the mass-store option word. It defines the starting 128-word segment on the mass storage device where the program will be resident while it is not in main memory. The contents of the word are treated as an unsigned binary integer. (See Figure 2-7.) Bits 1 through 13 of the Option Word (Figure 2-4) define other characteristics of mass-store resident programs.

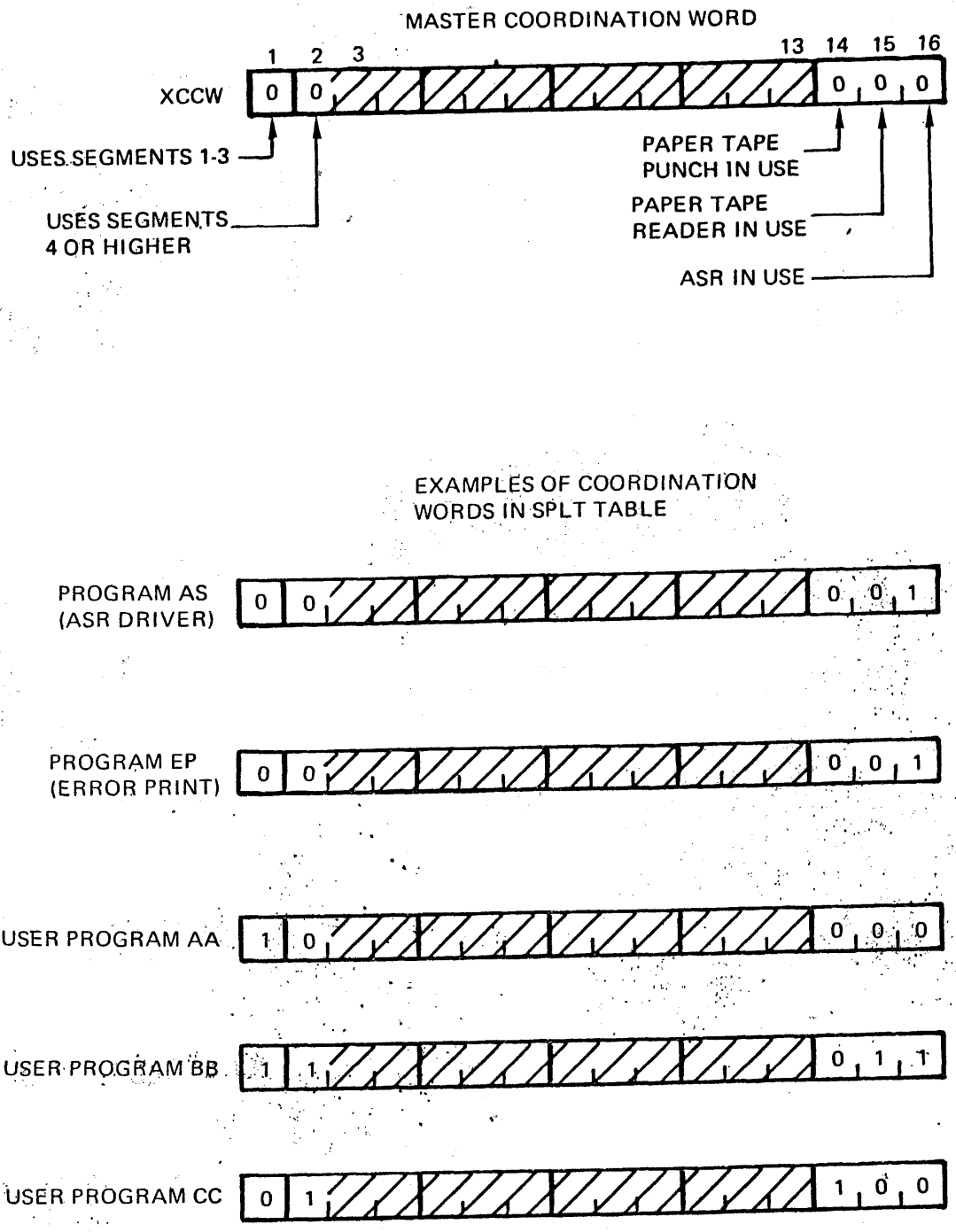


Figure 2-5. Typical Coordination Word Bit Assignments

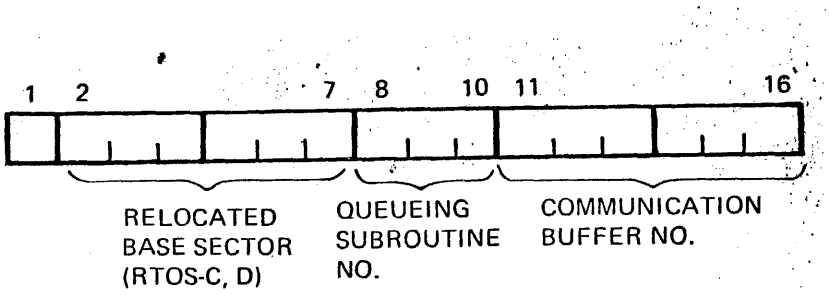
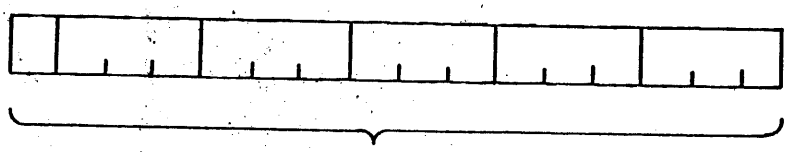


Figure 2-6. Communication Word Bit Assignments



Number of first 128-word segment on mass-storage device that is occupied by program

Figure 2-7. Mass Storage Option Word

INTERRUPT HANDLER

A typical series of events during processing of an interrupt is shown in Figure 2-8. In order for an interrupt to occur, a program must use a Connect Interrupt executive function to associate a label in itself with the interrupt for a particular device. The Connect Interrupt function loads the response code label and the SPET entry of the connecting program in the device's SID1 entry in SIM. The connecting program may then return to the Waiting state, or be interrupted by a program of higher priority. Assume that program BB in Figure 2-8 has already connected itself to the device and entered a wait, and that another program, AA, is running when the interrupt occurs.

The code in the device's SID1 table entry determines the events that follow the interrupt immediately. The first action is taken by the SAVM. (Save Machine State) component of the Executive, which saves the machine state (registers X, A, B, S, P, 4, 5, 6 and keys) in a vacant interrupt data block in SIVT. (IHP1 is the pointer to the appropriate block.) SAVM also forces single precision arithmetic mode, enables higher priority interrupts (issues an ENB instruction), and increments ICNT (the number of interrupted interrupts location). SAVM returns to SID1 and an indirect JST through the pointer 'Intresp', loaded previously by the Connect Interrupt function. This starts the user's interrupt response code, which must at least acknowledge the interrupt, and may also start a new device cycle or other control function. If a label is to be scheduled for further non-interrupt processing of the data associated with the interrupt, the label must be present in the A register before the interrupt response code returns to SID1. (Otherwise A=0.) The IH20 subroutine schedules the label (if present) and clears the active interrupt. Control then passes to the scheduler, which resumes its function of determining the highest priority active program and starting it. A label scheduled by the interrupt response code may be started at this time.

When program AA finally has the highest priority, it is restarted by the scheduler. The REST (Restore Machine State) subroutine returns the registers and keys to the state that was in effect when the interrupt occurred. The saved value of the P register is placed in Executive location XPSP. An indirect jump through that location then resumes program AA at the point where it was interrupted.

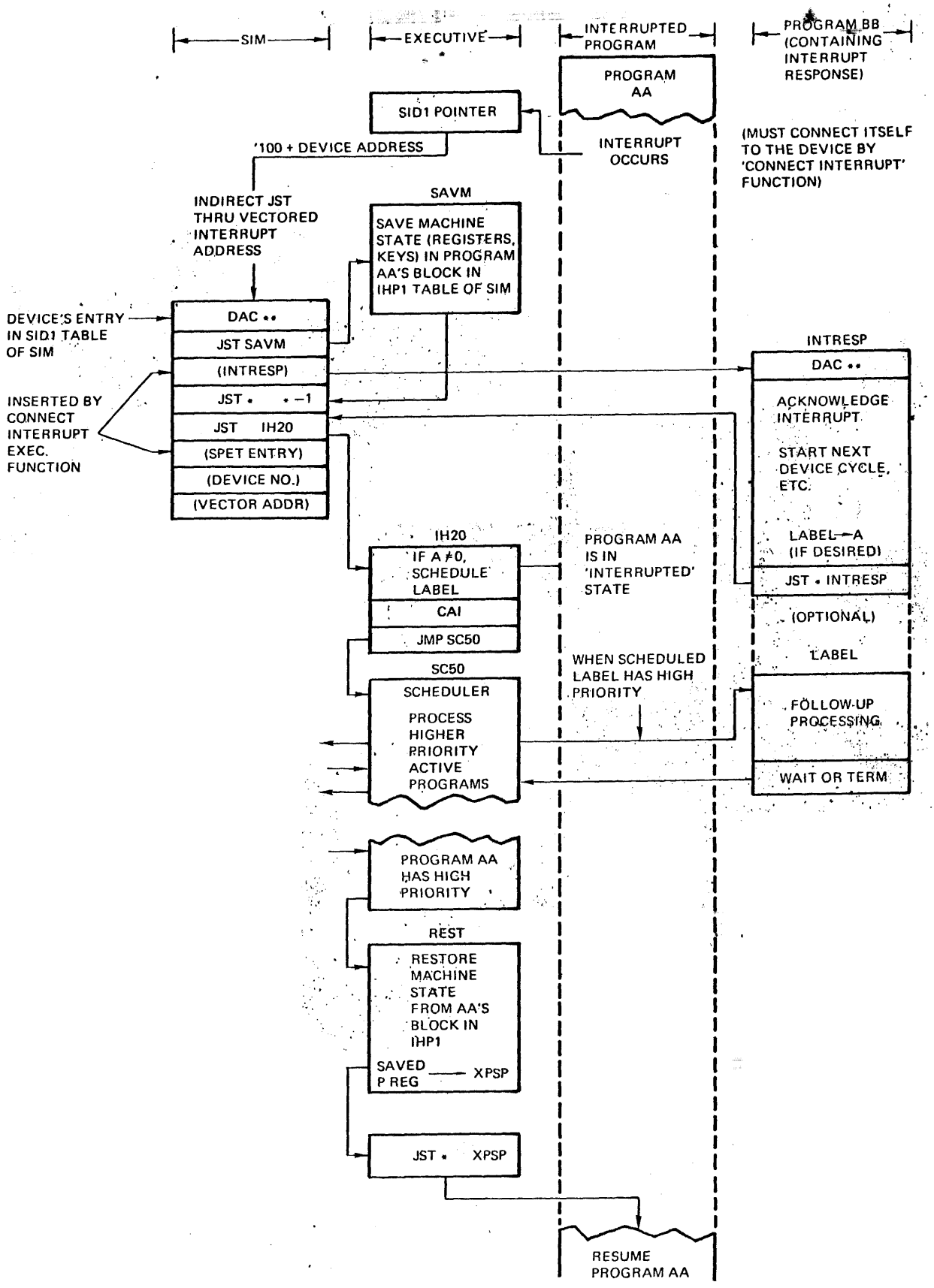


Figure 2-8. Interrupt Response Sequence

FUNCTION HANDLER

The function handler is the portion of the executive that does preliminary processing of all executive function calls, routes control to the proper function subroutine, and arranges return conditions. The function handler begins at location FE.

Figure 2-9 summarizes the function handler entry process. Every assembly-language function call begins with a SVC instruction. (See Section 4.) The SVC forces an interrupt (an indirect JST) through location '65, which contains a pointer to the location preceding FE. The pointer is loaded into location '65 during executive initialization. Once control has been passed to the function handler, the word preceding FE points to the second word of the active function call. This address base is used to obtain the function number, program name, and other parameters of the function call.

Following execution of the required system function, the Function Handler checks flag FE50. This is set by the Request Program and Schedule Label Executive functions when a program of higher priority than the one currently running is requested or has a label scheduled. FE50 is set also by the Interrupt Handler when it schedules a label in a program of higher priority than the one interrupted. If the flag is set during execution of a function, the Function Handler simulates an interrupt, saving the necessary registers, and exits to the Scheduler so that the newly requested high priority program may be started.

The effect of each executive function is defined in Section 4. Custom executive functions can be added by the user; for instructions, refer to Section 7.

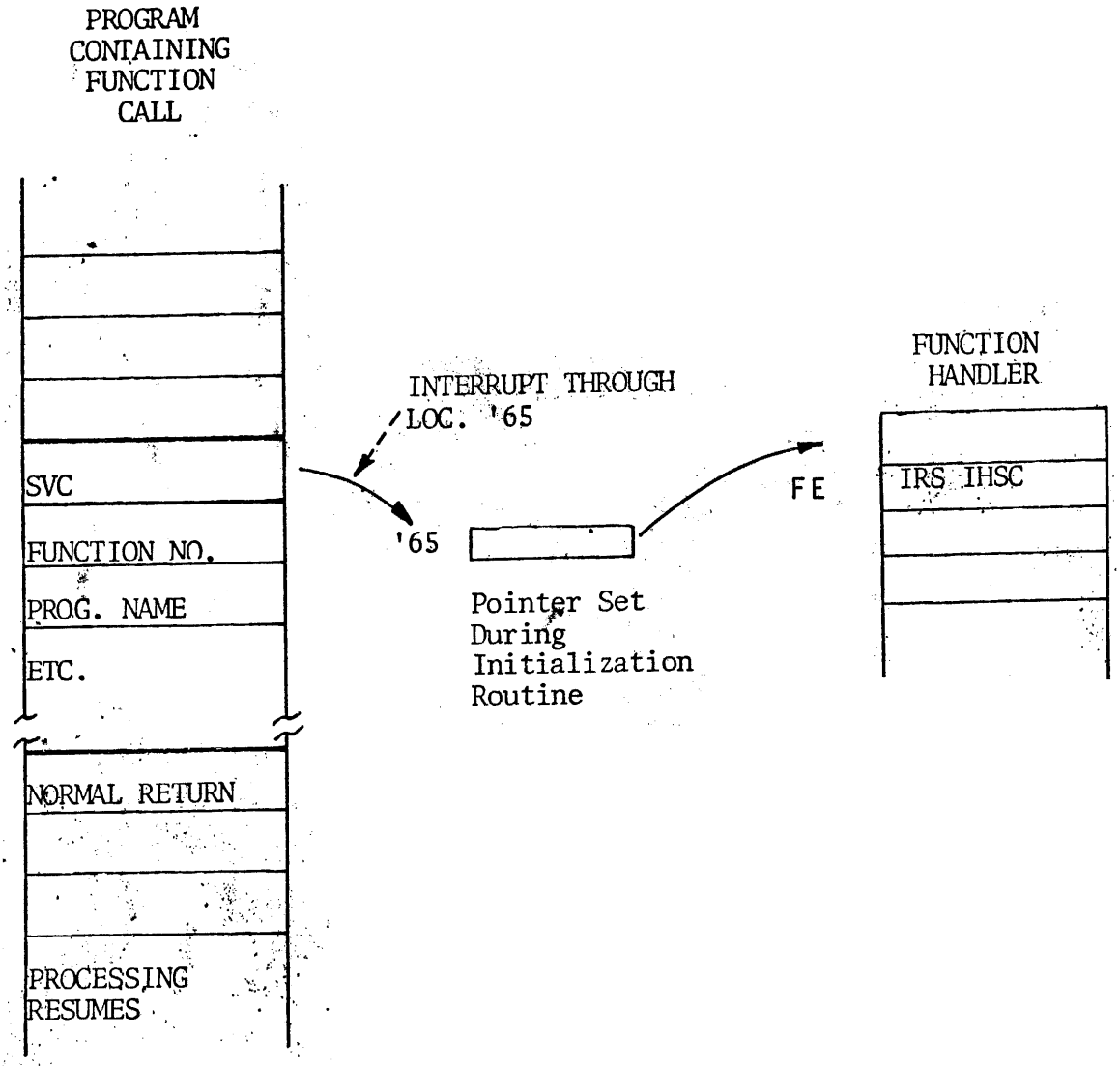


Figure 2-9. Entering Function Handler

COMMUNICATION OPTION

Included in the Request Program executive function is the optional ability to transfer a communication parameter from a requesting program to the requested program. The parameter may be a single data word, a pointer to a data buffer, or a control word to influence the program's processing functions. This feature is put into effect when the communication parameter in word L+4 of a Request Program calling sequence is non-zero. The Request Program function in that case includes a call to one of the system's queueing routines (FIFO, for example) as shown in Figure 2-10. Part of the queueing routine transfers the communication parameter from the calling sequence to a communication buffer area configured into the SPCT table of SIM. The communication word in the requested program's SPLT table entry specifies the queueing subroutine and current buffer number. (See Figure 2-6.) The queueing routine automatically keeps track of, and updates, pointers to the available communication buffer area, so that several programs can request a given program and specify different communication parameters. The parameters are "queued" in the order in which they are received.

If the communication parameter in a Request Program function is zero, the communication-related functions are not used.

When the program that requires communication is finally the highest priority program, the scheduler transfers control to another part of the queueing routine, which obtains the current parameter from the buffer area and places it in the 'communication word' location in the requested program's header. (The word preceding the starting address is reserved for a communication parameter.) The FIFO queueing routine provided as the standard RTOS communication method passes communication parameters on a first-in, first-out basis. The user may write queueing routines that employ other strategies such as last-in, first-out or one based on the priority of the calling programs.

By the time the scheduler is ready to start execution of the requested program, the communication parameter is present in its header location and available for use during execution of the program.

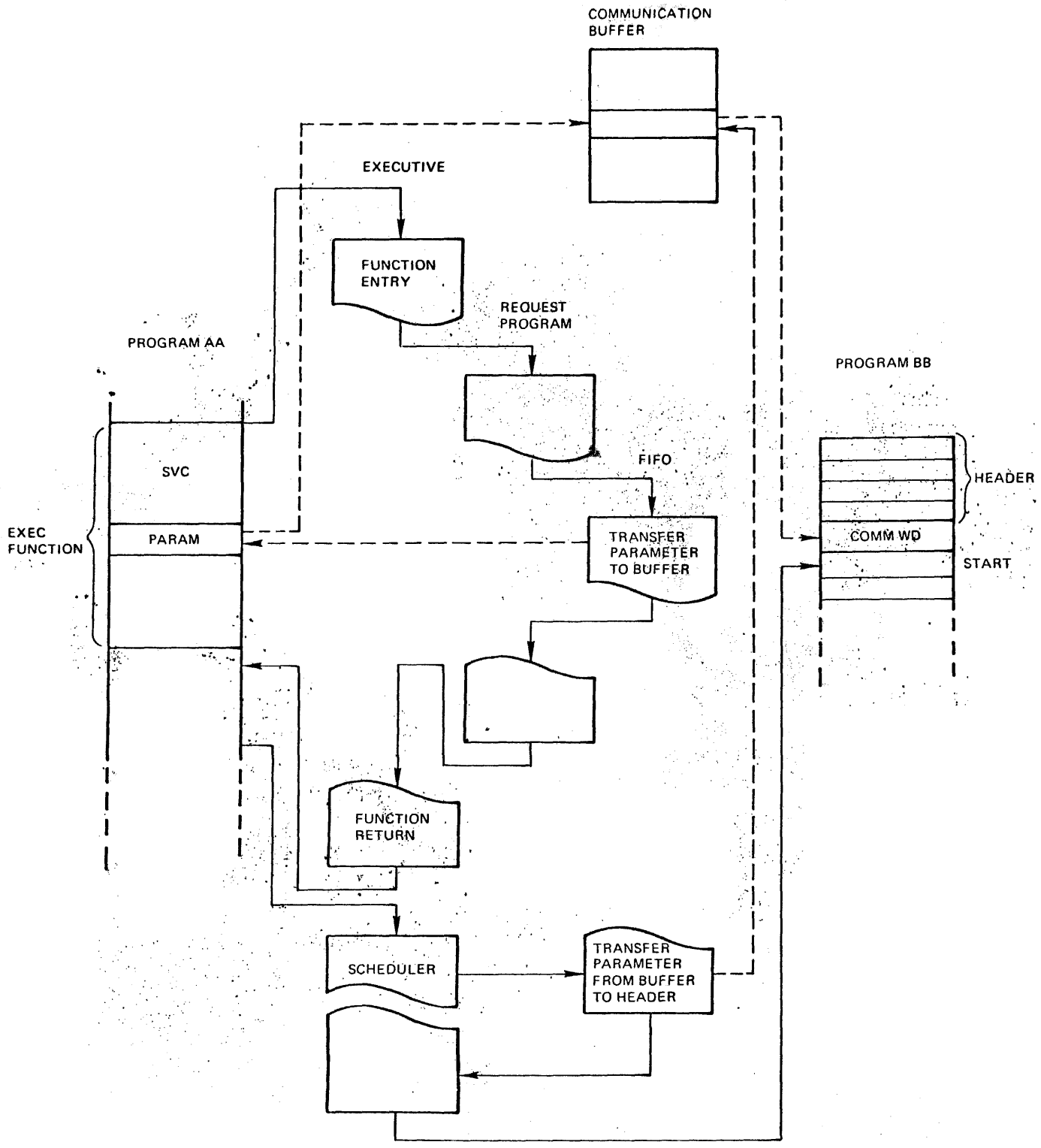


Figure 2-10. Processing an Optional Communication Request

CLOCK PROGRAM

The Clock Program is the section of the Executive beginning at entry point CL. In response to hardware interrupts from the PRIME Real Time Clock, it maintains time and date counts and handles startup of programs controlled by Connect Clock and Disconnect Clock executive functions.

Real Time Clock

The PRIME real time clock (RTC) operates by automatically incrementing memory location '61 at the power line frequency (60 or 50 Hz). When location '61 overflows to zero, the RTC issues a vectored interrupt through location '63, which contains a pointer to program CL.

The actual RTC interrupt rate is determined both by the line frequency and the count present in location '61. Initially, and after each RTC interrupt, location '61 is set to the value specified by the user in location CLK3 of SIM. The value in CLK3 is the 2's complement of the desired number of clock pulses between RTC interrupts. For example, the user might enter the value -3 in CLK3 when the system is configured. When the system is started, the RTC overflows to 0 and issues an interrupt after the third line-frequency clock pulse (50 ms for a 60 Hz frequency). The interrupt response code in program CL resets location '61 to the CLK3 value after each RTC interrupt.

Time and Date

The interrupt response code in program CL maintains time-of day and date counts in the following memory locations:

Location	Contents	Starting Value
XMIL	Millisecond counter	(CLK2)-1
XSEC	Time of Day - seconds (0 to 59)	59
XMIN	Time of Day - minutes (0 to 59)	59
XHR	Time of Day - hours (0 to 23)	23
XDAY	Elapsed days counter	0

During every RTC interrupt, the response code in program CL increments location XMIL and compares it to location CLK2 of SIM, which is preset by the user to show the number of interrupts per second. For example, when the interrupt period is 50 ms, location CLK2 is set to 20 interrupts per second. When XMIL equals CLK2, one second has elapsed, so XMIL is reset to 0 and the seconds counter (XSEC) is incremented.

When XSEC overflows from 59 to 60, it is returned to 0 and the minutes counter is incremented. Similarly, when the minutes count overflows from 59 to 0, the hours count is incremented, and when hours overflow from 23 to 0, the days count is incremented.

Scheduling Programs Due

When the time and date words are updated, the interrupt response code also schedules a label (entry point CL14 in the CL program) as the non-interrupt code for follow-up processing. When this label has high priority, it is started by the scheduler to update the entries in the SCUT table of SIM. This table consists of a four-word header followed by a series of 5-word blocks reserved for data associated with Connect Clock executive functions.

Initially, the first word of the header and each data block are strung together by pointers, as shown in Figure 2-11A. After a period of operation, Connect Clock executive functions modify the table to a condition such as that shown in Figure 2-11B. Separate pointer threads are maintained for programs that become due at intervals of minutes, seconds, or milliseconds. Three reserved words in the header point to the first block of each thread. Within a given thread, each block contains a pointer to the next block (the pointer is 0 if it is the last in the thread).

Each SCUT table entry is created by a Connect Clock function in the following format:

Word	Definition
1	Pointer to next block in same interval-unit thread (0 if last block in thread)
2	SPET pointer for program connected to clock
3	2's complement of time to first execution
4	2's complement of interval, if repetitive execution is specified (0 for one-time execution after a delay)
5	Label to be scheduled (0 if no label)

Words 3 and 4 are in units that correspond to the timebase thread for the entry (minutes, seconds, or RTC interrupts).

During the non-interrupt code that follows every RTC interrupt, each entry of SCUT is processed in the following manner. (In general, all entries in the RTC interrupt thread are checked first. The seconds and minutes entries are processed only when their interval is updated.

1. The offset entry (time to first execution) is incremented. If it overflows to 0, it is replaced by the interval value and the program proceeds to request the program or schedule a label, as required. The calling sequence is built up using the program name and label from the program's SCUT entry.

2. If the entry was for a time delay (execution once after a delay), the SCUT entry is deleted and thread pointers altered accordingly.

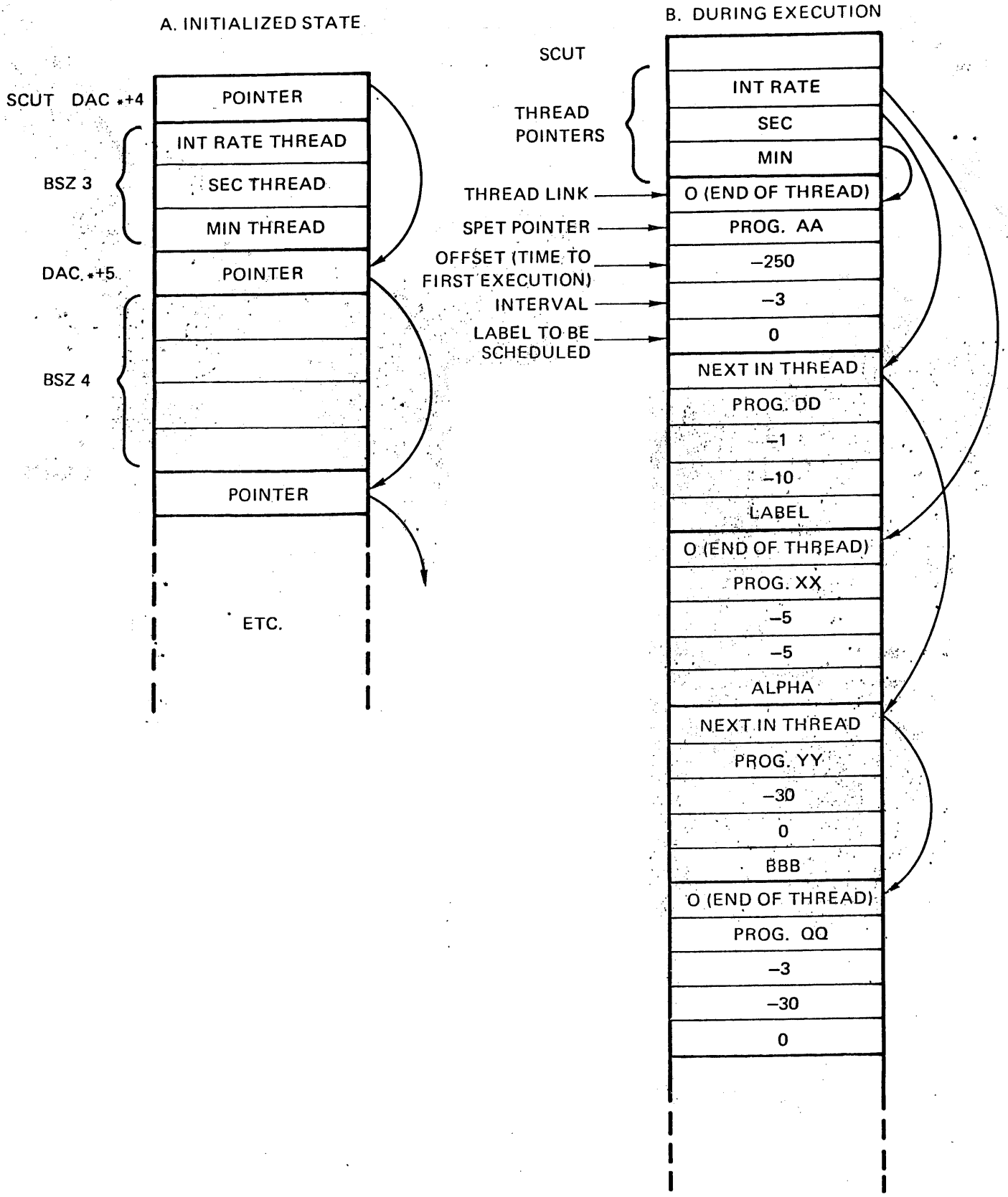


Figure 2-11. Use of SCUT Table by Clock Program

3. The current program's thread pointer is used to locate the next program requiring service. It is also processed as in steps 1 and 2.

4. When a program's entry contains a pointer of 0, the end of the current thread has been reached. If any other intervals have been updated (seconds or minutes) during the interrupt, the first entry in the next thread is accessed and steps 1 and 2 repeat.

5. When all threads have been processed for a given interrupt interval, the program executes a WAIT function and awaits the next RTC interrupt.

6. During the return from the Request Program or Schedule Label executive function, an interrupting program may execute a connect clock function that causes a different thread to require service. When control returns to the clock program, the original thread is identified and processing resumes where it was interrupted.

In Figure 2-11B, program AA is the only entry in the minutes thread. It is set up for a first execution time in 250 minutes, and thereafter in 3 minute intervals. There is no label to be scheduled, so when this program comes due, a Request Program function will be scheduled.

When a Connect Clock function specifies a first execution time as a time of day, the difference between that value and the current time of day is converted into a value in minutes, and entered as the offset value. For such an entry, the CL program increments the offset location every time the minutes thread is processed. When the offset overflows to 0, the program is requested and the offset is replaced by the interval value.

Program YY, in the seconds thread, is set up to schedule the label BB in 30 seconds. Since the interval is 0, it will not be repeated. This is characteristic of a program that has connected itself for one execution after a time delay.

ERROR PRINT PROGRAM

See ERROR PRINT FUNCTION, Section 9

Table 2-1. Executive Error Messages

Error Message ('PN'=Program Name)	Meaning
E1PN	Named program has requested an executive function that is not present in the system, or has used an incorrect function number.
E2PN	Named program has tried to schedule a label in an inactive program or one with the maximum number of labels already scheduled.
E3PN	Named program has tried to connect the clock when clock user's table (SCUT) is full.
E4PN	Named program has tried to terminate with an interrupt connected. The program is disabled.
E11$$	Unidentified interrupt has occurred. (Interrupt is ignored.)
E12PN	Named program has attempted to schedule a label from interrupt to non-interrupt code but header is already full of labels.
E13PN	Executive has attempted to schedule a label in named program which is inactive or which already has maximum number of labels scheduled.

Message Format

Each error message is preceded by a carriage return, line feed and bell character and is printed in the following format:

Edddnn

where each d is an octal digit and nn is the name of the program that initiated the message. Leading 0's in the error code are suppressed.

If console sense switch 4 is set, error message printout is suppressed. (This feature should be used during debugging only.)

In the following example, a program named AA calls EPMOD to print a message numbered '123:

```
            LDA     ERCOD       ERROR CODE
            LDX     PNAME       PROGRAM NAME
            INH
            JST*    EROR        ENTRY TO ERROR PRINT PROGRAM
            ENB
            JMP     RETN        CONTINUE AT RETN AFTER ERROR PRINTED
ERCOD       OCT     123
PNAME       DATA    C'AA'
EROR        XAC     EROR
```

Resulting error printout:

E123AA

SYSTEM LOADER PROGRAM (SYSLDR)

The system loader is used in RTOS-B systems to load disk-resident programs into main memory for execution. SYSLDR runs as a program under the executive. Its main function is to request the mass-store driver to read the program into memory from disk as if it were a block of data.

When a disk-resident program that is not currently in memory is requested, and is capable of running (from the standpoint of coordination, etc.), the scheduler places it in disabled status and requests the loader program. When SYSLDR is the high-priority program, it is started; it in turn requests the disk driver, supplying the disk-resident program's location on the disk and size as parameters. SYSLDR obtains the parameters from the requested programs SPLT table entries. Also supplied as a parameter is label OK within the SYSLDR program. When the disk driver completes a successful transfer, it schedules label OK in SYSLDR and terminates. When SYSLDR is restarted by the scheduler at label OK, the status word of the original requested program is updated to show that it is now memory resident. The 'Disabled' bit is also cleared, so that the requested program can become active when it has high priority.

In the SPLT table, two different entries at different priority levels may be used for the disk driver. For example, SYSLDR may refer to the driver as program ML, while user programs may request the disk driver as program SM. This enables program loading to take place at a different priority than data transfers initiated by an application program, even though the same driver is called. (For details, see Section 3.)

FLOWCHARTS

The actions of the scheduler, function handler, and other major components of the Executive are summarized in a series of simplified flowcharts. These charts are intended as roadmaps through the multitude of conditional paths that can be taken during RTOS system execution. They summarize the actions taken by the executive to determine program priority, update program status, and start execution of the current program. Actual entry points are identified so that the user can refer easily to the sections of the Executive listing for complete detail.

In order to use these flowcharts effectively, the user should be thoroughly familiar with each table and entry of the System Information Module (SIM) described in Section 3.

Figure 2-12 defines the symbols and graphic conventions used in these charts.

NOTE

The flowcharts were current for Rev. 3 but do not include changes through Rev. 6. For more current flowcharts, see the following:

Figure 9-1 RTOSVM Starting & Initialization

9-2 Scheduler Dispatch Table Operation

9-3 Requesting Disk Resident Programs

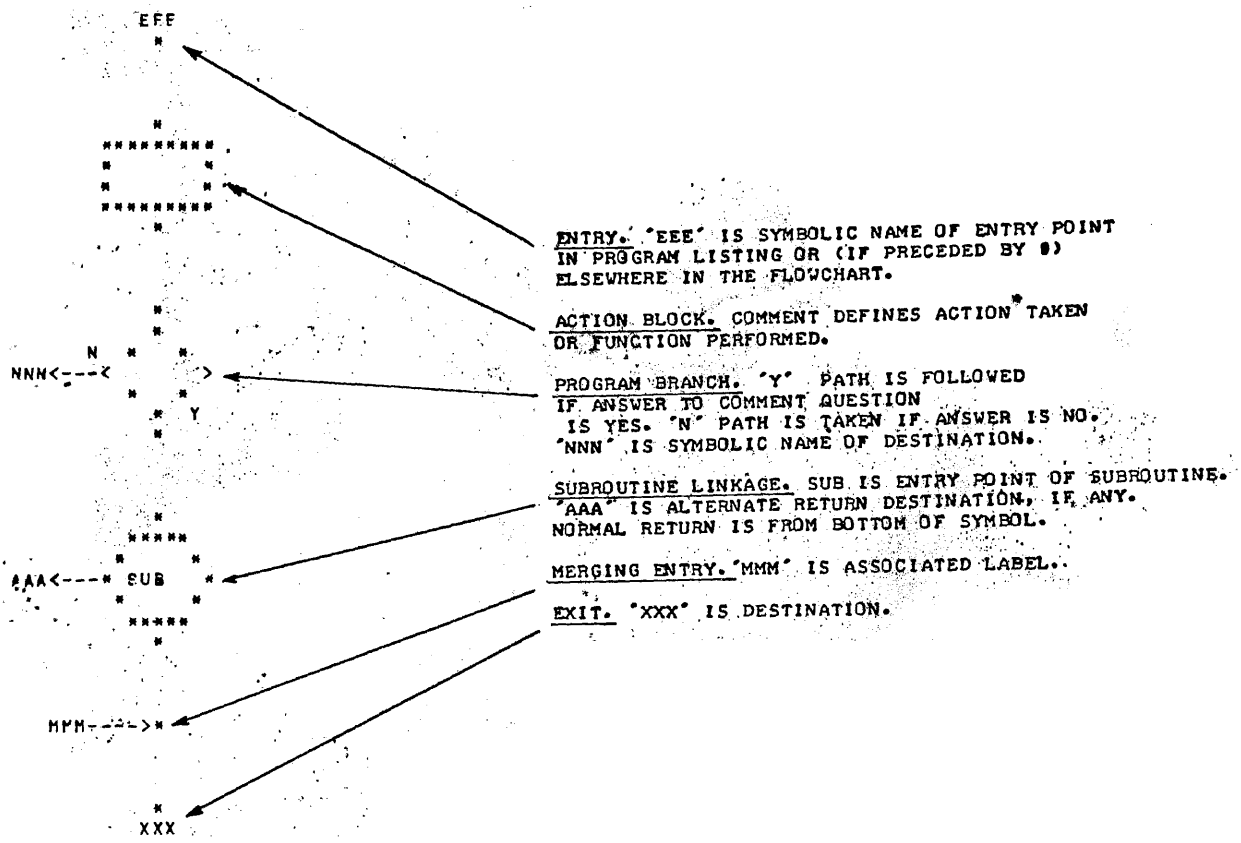


Figure 2-12. Flowchart Symbols

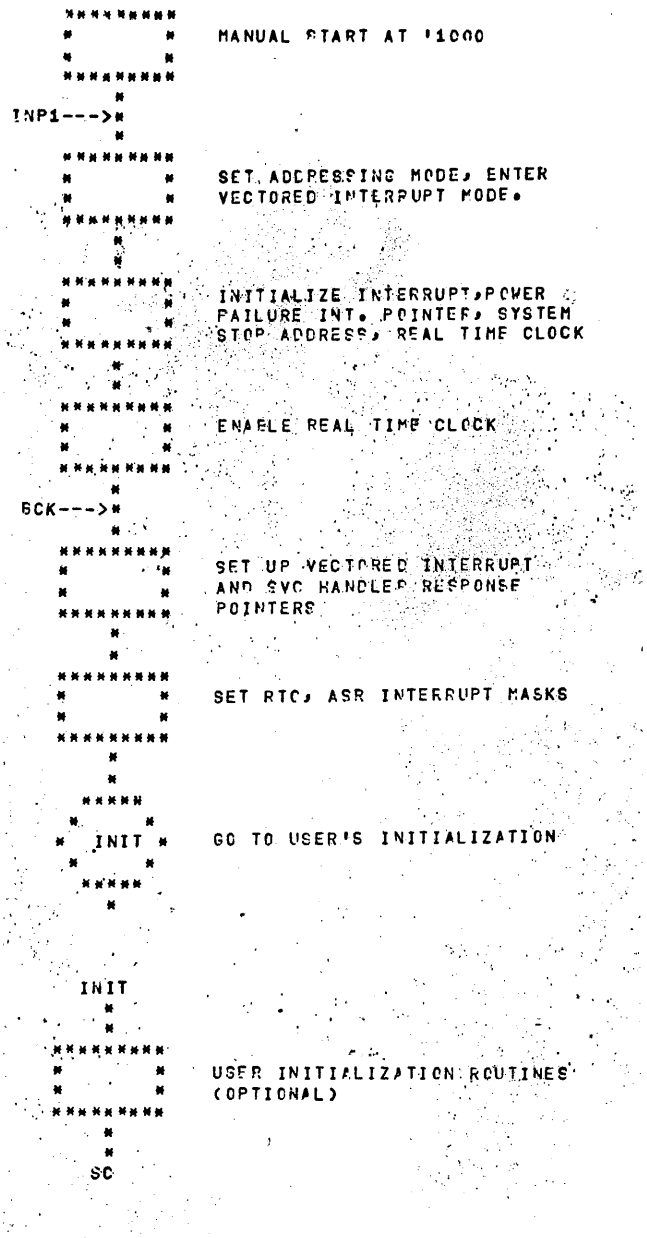


Figure 2-13 Startup and Inititalization Flowchart

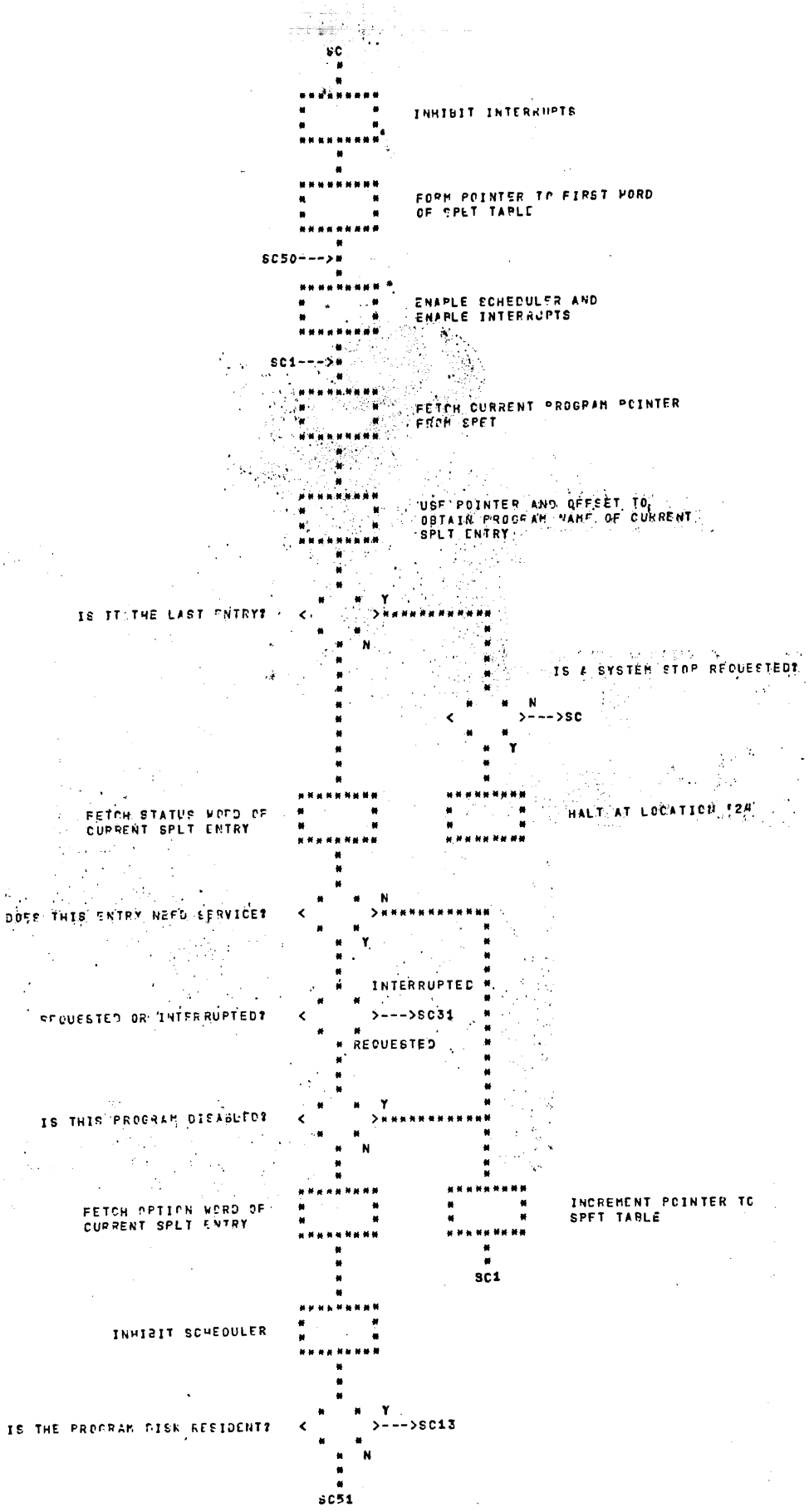


Figure 2-14. Scheduler Flowchart (Sheet 1 of 5)

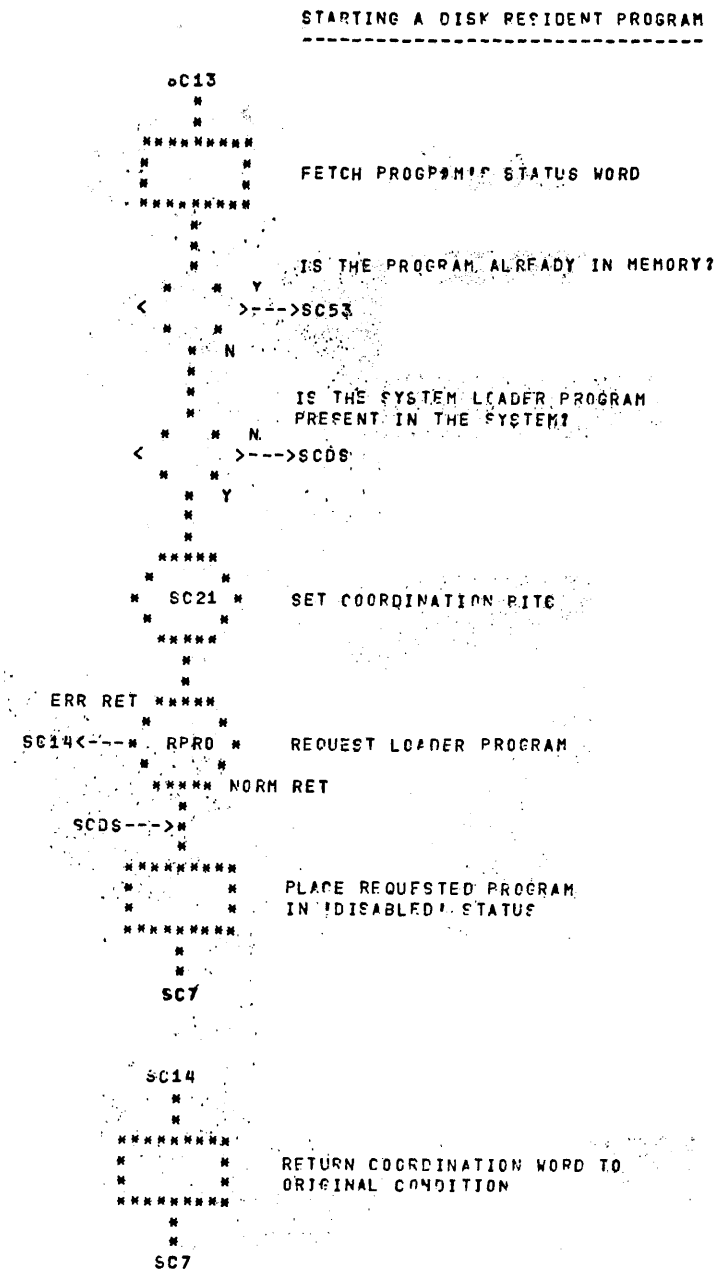


Figure 2-14. Scheduler Flowchart (Sheet 2 of 5)

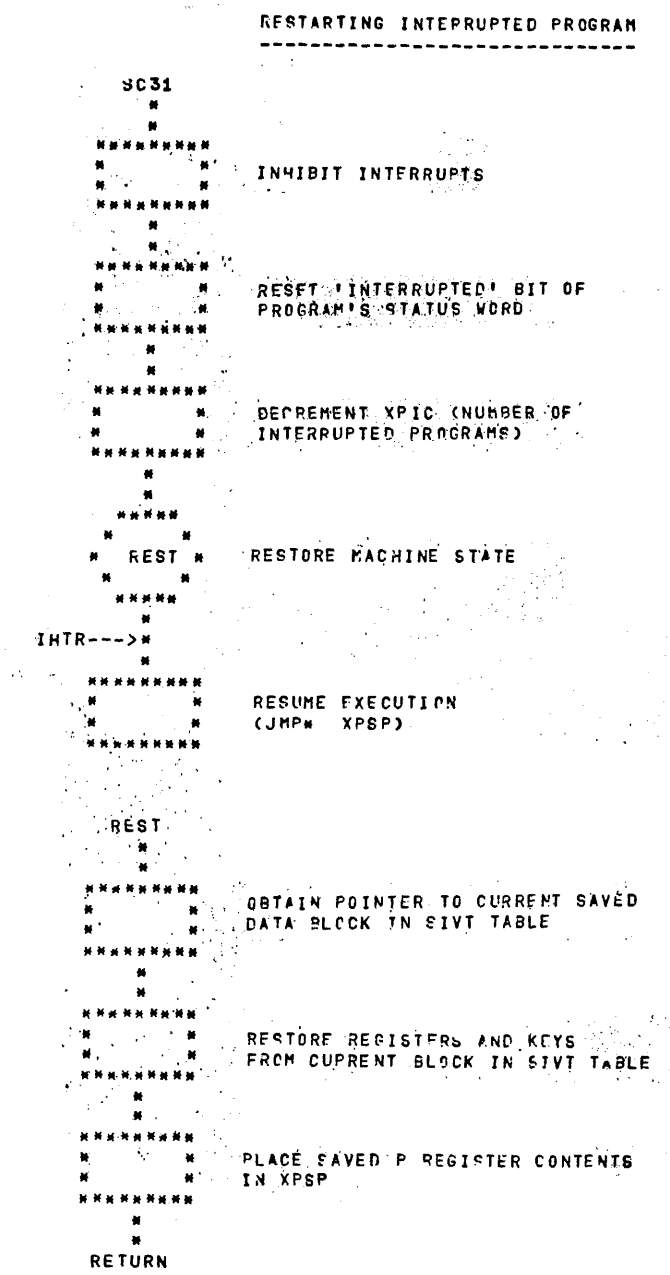


Figure 2-14. Scheduler Flowchart (Sheet 3 of 5)

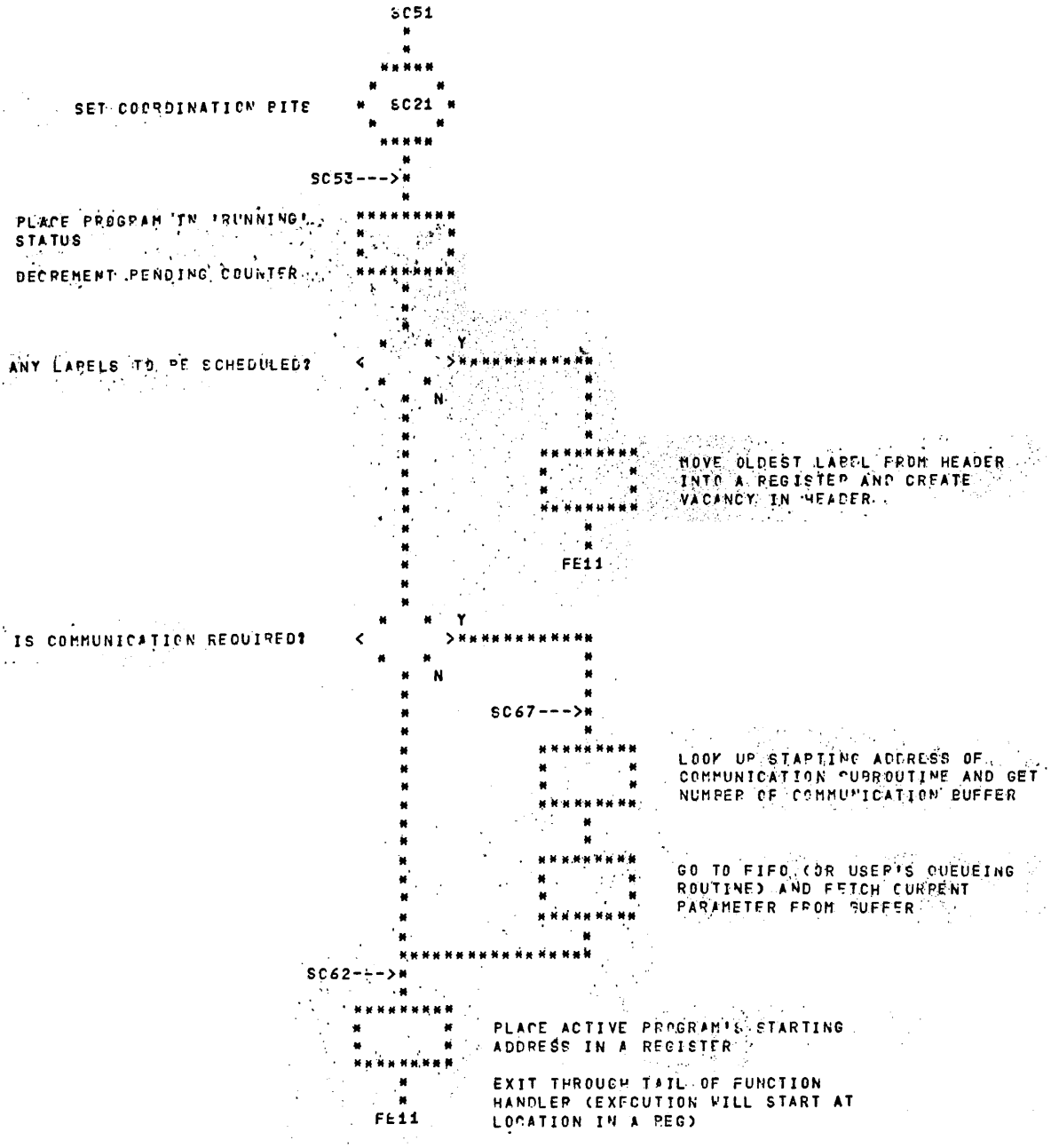


Figure 2-14. Scheduler Flowchart (Sheet 4 of 5)

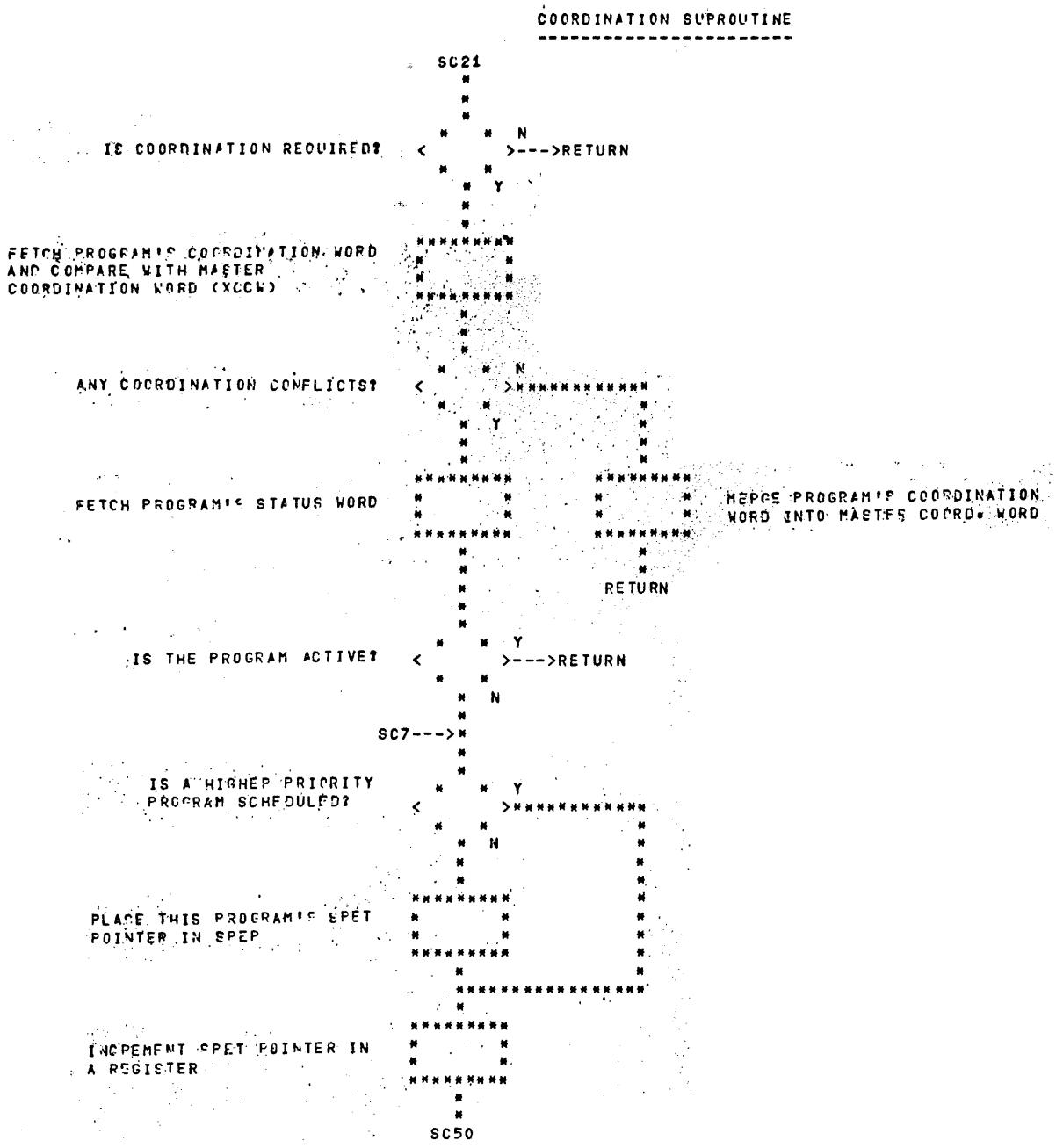


Figure 2-14. Scheduler Flowchart (Sheet 5 of 5)

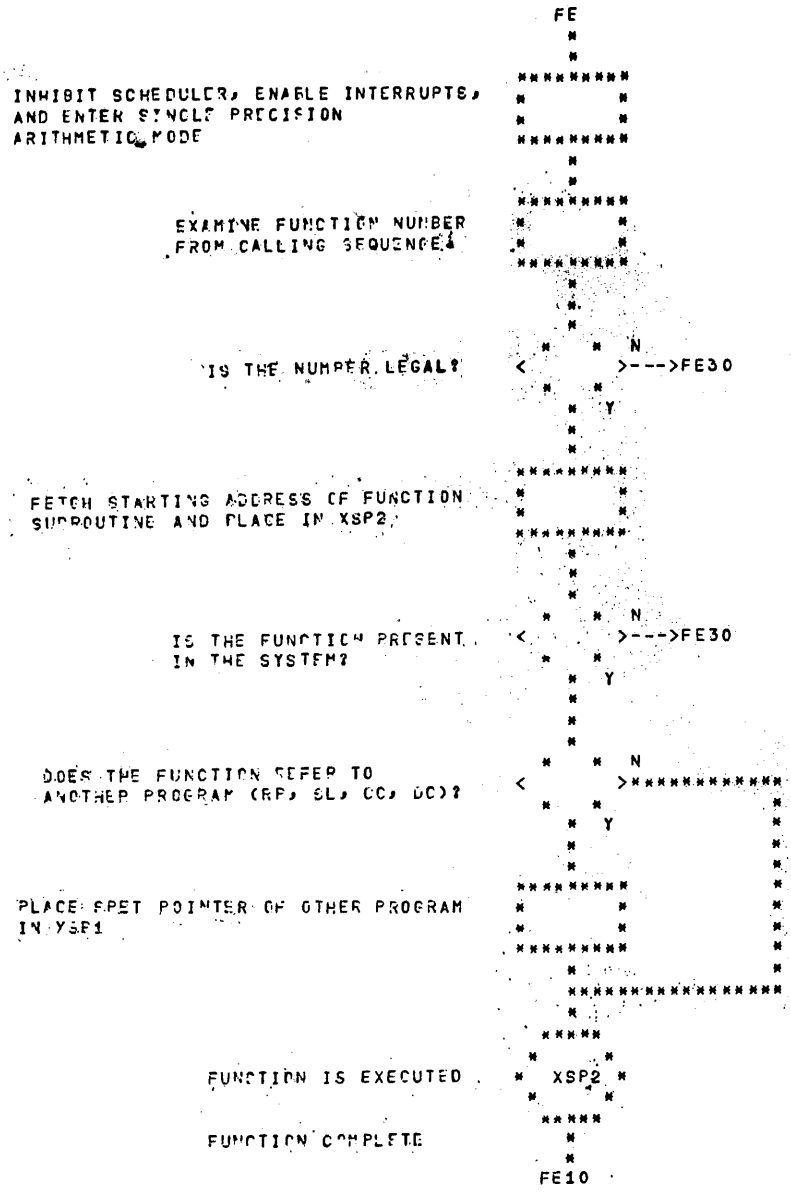


Figure 2-15. Function Handler Flowchart (Sheet 1 of 3)

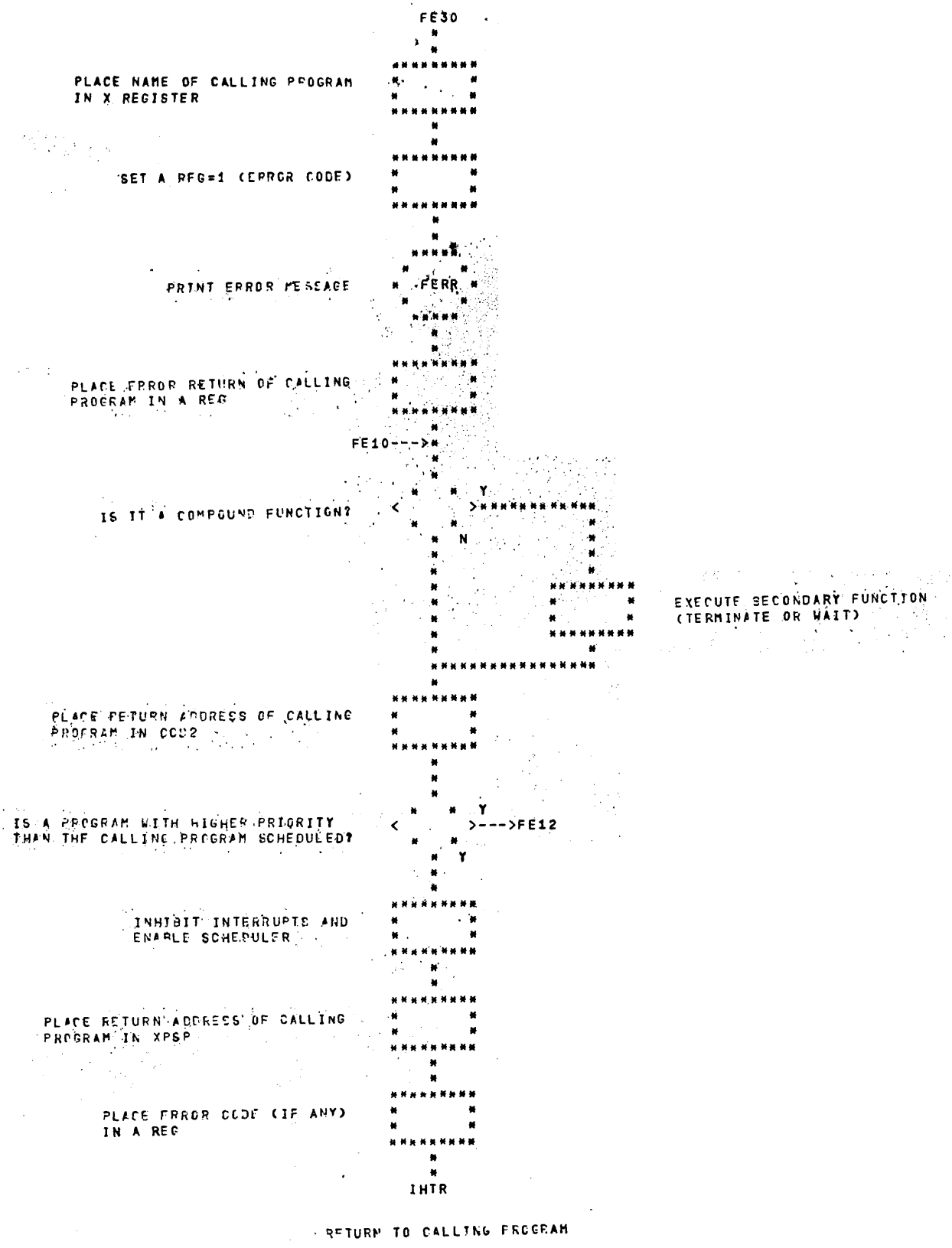


Figure 2-15. Function Handler Flowchart (Sheet 2 of 3)

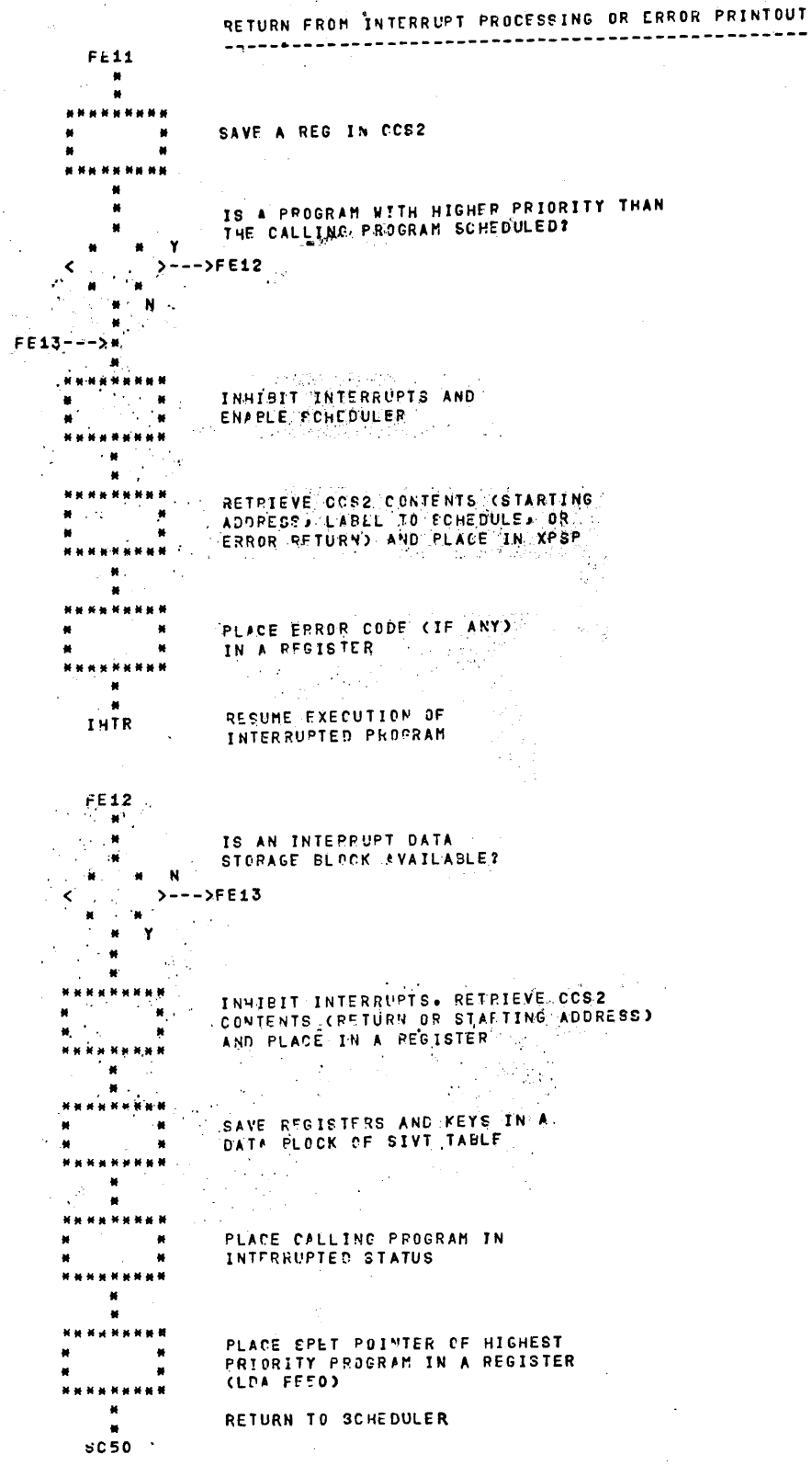


Figure 2-15. Function Handler Flowchart (Sheet 3 of 3)

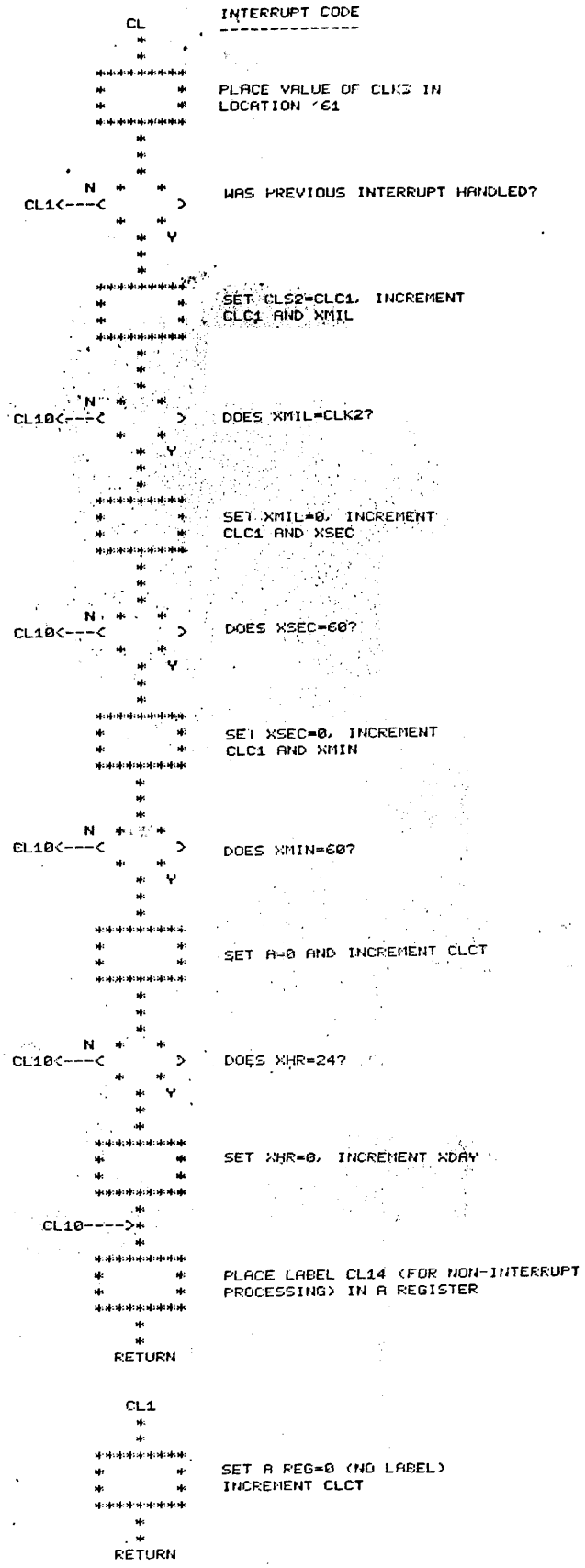


Figure 2-16. Clock Handler Flowchart (Sheet 1 of 3)

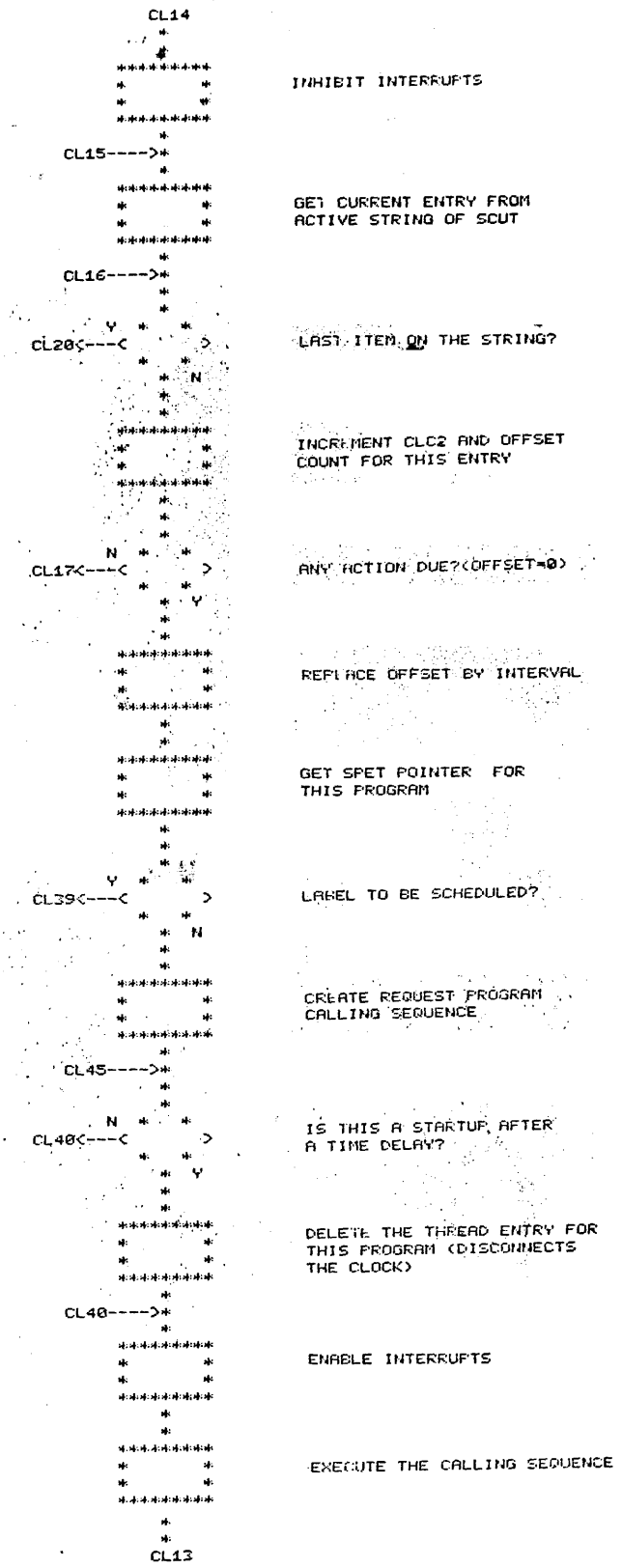


Figure 2-16. Clock Handler Flowchart (Sheet 2 of 3)

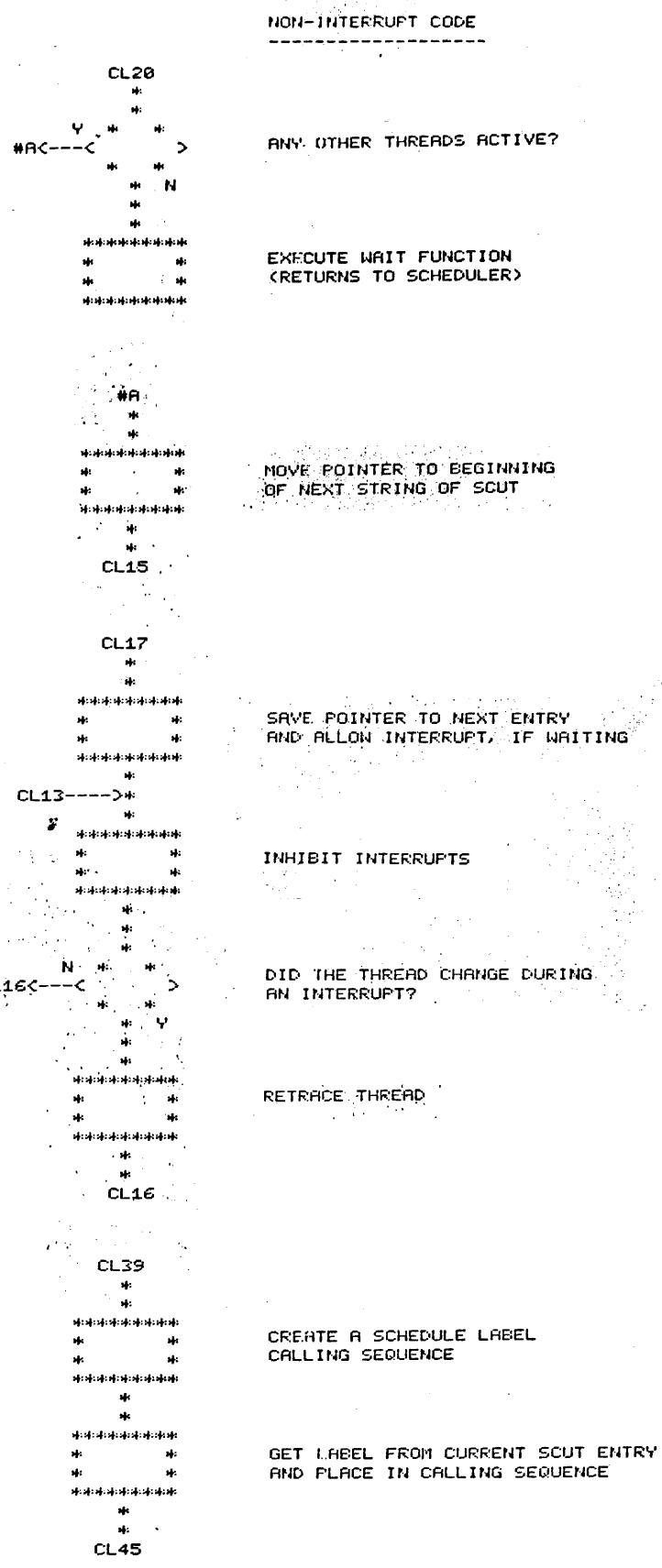


Figure 2-16. Clock Handler Flowchart (Sheet 3 of 3)

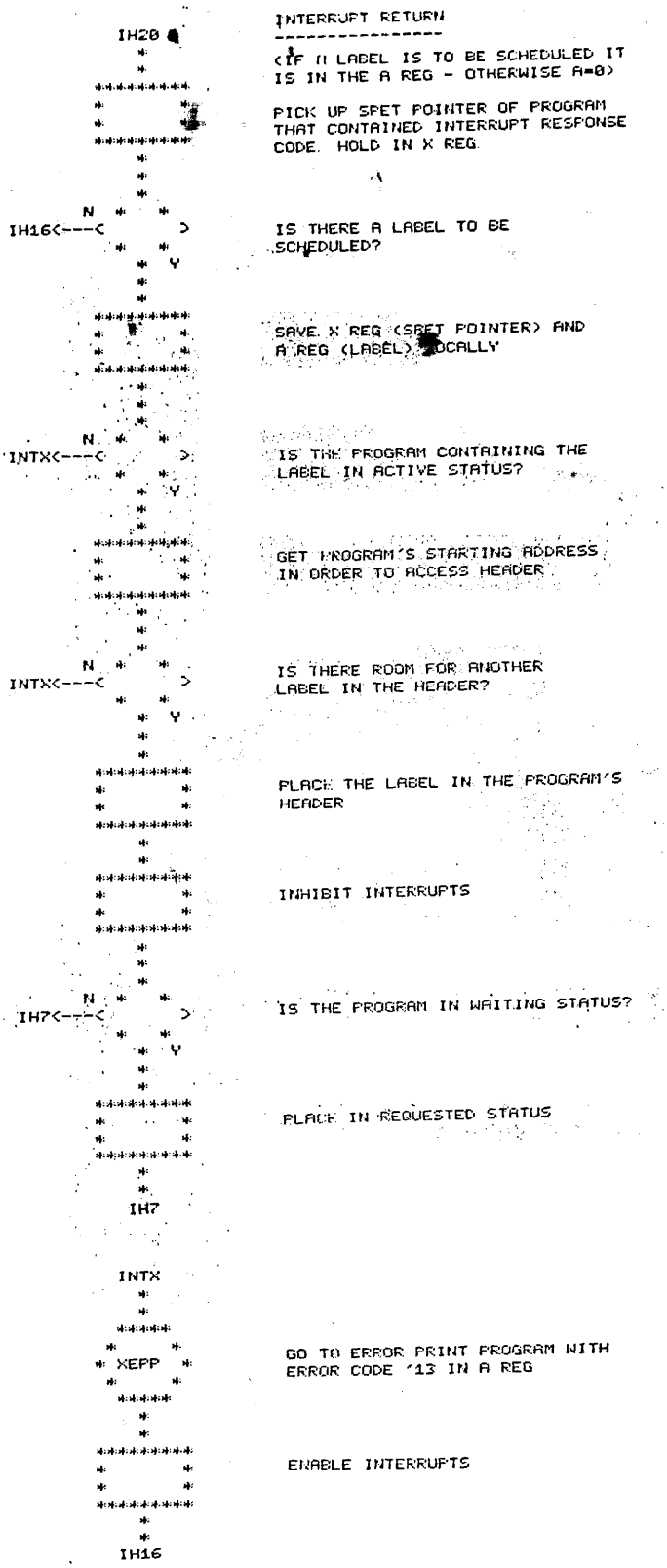


Figure 2-17. Interrupt Handler Flowchart (Sheet 1 of 3)

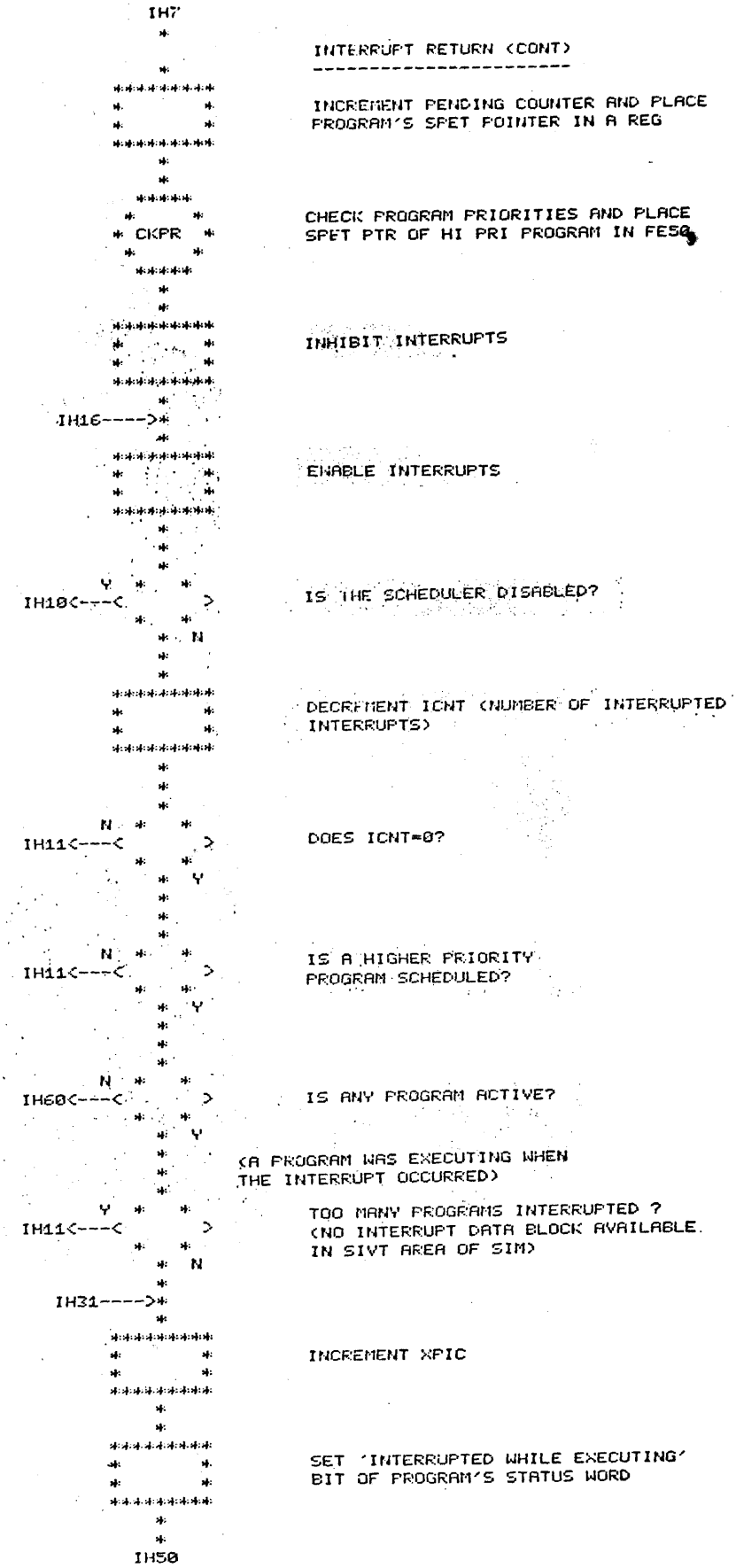


Figure 2-17. Interrupt Handler Flowchart (Sheet 2 of 3)

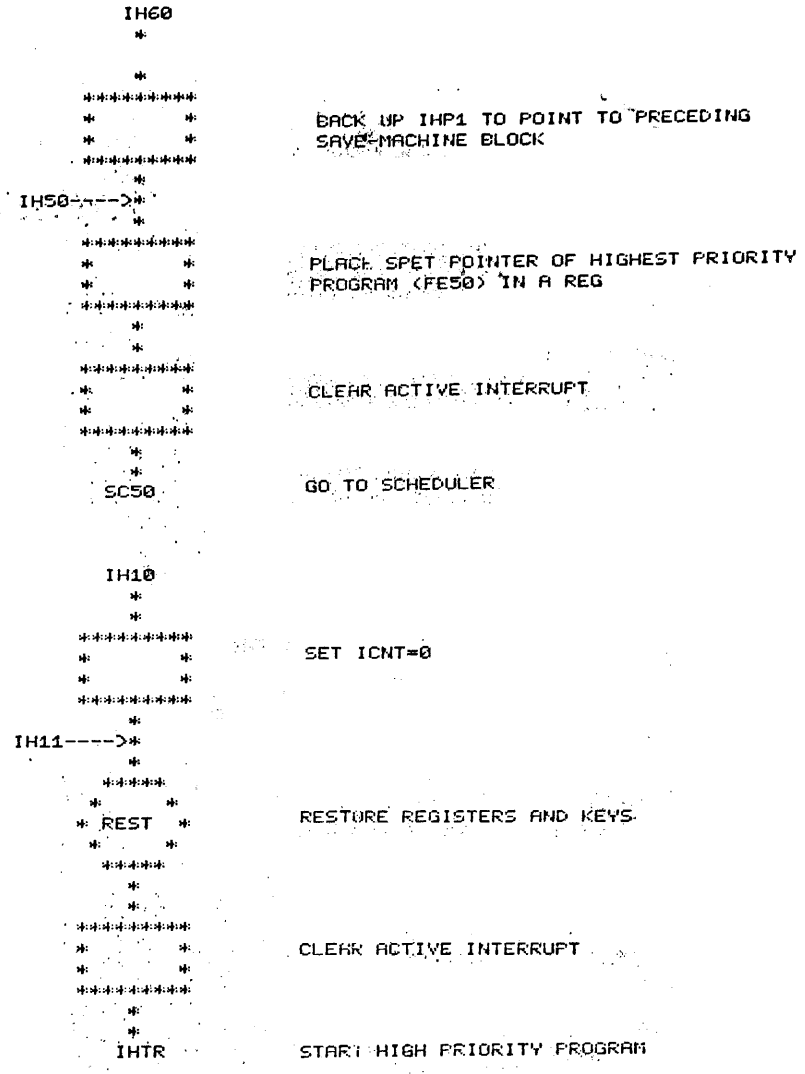


Figure 2-17. Interrupt Handler Flowchart (Sheet 3 of 3)

SECTION 4

EXECUTIVE FUNCTION CALLS

GENERAL INFORMATION

Programs communicate with the Executive and each other by means of executive function calls. Each function call is a brief assembly-language or FORTRAN calling sequence that supplies the Executive with a function code and other parameters.

Assembly Language Calling Sequences

All executive function calls begin with a SVC instruction that forces an interrupt to location '65. The Executive initialization routine stores a pointer to the Executive's function handler entry point in that location. The second word specifies the function number. The number and meaning of other words varies from function to function.

Registers are not saved and restored during execution of these functions. This is the user's responsibility.

Following is an example of a line from a PMA calling sequence:

Label	Operation	Variable	Comments
(L+2)	DEC	Pnumber	(Offset in SPET)

Line numbers (L+2, etc.) in the label field are for reference only. The user can leave the label field blank or assign labels as required. Items in capitals (DEC, etc.) are to be copied verbatim. Initial-cap items (Pnumber, etc.) are variables to be assigned symbolic names or values by the user. Comments in the examples are for purposes of explanation.

FORTRAN Calling Format

FORTRAN calls are in the form of a Subroutine Subprogram reference, as in:

```
                    |Pname   |
CALL REQST  ( arg1  |Pnumber | ,Status  [,Commun] )
```

Items between vertical bars (Pname, Pnumber) are alternatives, of which one is to be chosen. Items enclosed in brackets (i.e., [,Commun]) are optional and can be omitted. All parameters are integer constants, variables or array names, as required.

Using Program Numbers

Word 3 of function calls 1 through 4 may contain either a two letter ASCII program name or a program number. A symbolic name is convenient during system building and checkout, but for real-time execution a number saves time by eliminating a name search. The numbers are determined by the position of the program in the SPET table of SIM; the highest-priority program is program number 1, and so one. When the Executive finds a number instead of a name, it uses the number directly to index down to the correct position in SPET.

The user may want to enter ASCII program names during system configuration for ease of making changes; then, after the RTOS package is firm, program numbers can be substituted for the names to speed execution.

In FORTRAN, the program can also be identified by an ASCII name or SPET offset number:

```
INAM='AA'      (Refers to program AA)

INAM=3         (if program AA is 3rd item in SPET)
   .
   .
   .
   .

CALL REQST     (INAM, - - - - - )
```

Return Conditions

After executing a function, the function handler in the executive usually returns to the program that issued the function call. (Terminate and Wait functions do not return, and a program that delays itself with a Connect Clock function enters the waiting state and does not return.)

Most functions provide an error return location that is taken if the function cannot be performed (for example, if a nonexistent program is requested, a communication buffer is full, etc.). Normally the Executive prints an error message on the ASR and returns to the error return location of the originating program with an error code in the A register. The Disconnect Interrupt function does not provide an error return (return is always to the normal return location).

During execution of the function, if a program request or an interrupt causes another program to be higher in priority than the calling program, the latter is placed in 'interrupted' status. When it again has highest priority, the calling program resumes at its normal or error return location.

FORTRAN versions of these functions do not provide an error return as such. Return is always made to the first executable statement following the function call. Most FORTRAN calls include a 'Status' variable that is set by the Executive to indicate an error condition, if any. The user can check this variable after each function call to detect an error condition.

After a normal or error return, the processor is in single-precision arithmetic mode and the same addressing mode as when the function was called.

Address References

In PMA, address references such as 'Errtn' and 'Label' must be valid address expression in which all symbols are previously defined. In FORTRAN, an address reference can be made by a variable that is ASSIGNed a statement number value, or by an address constant in the form $n, where n is a statement number.

REQUEST PROGRAM (FUNCTION 1)

This function call places a specified program in the 'requested' state, and passes an optional communication parameter.

The PMA calling sequence is:

```
(L  )    SVC                (Function handler entry)
(L+1)    DEC    1           (Function number)
(L+2)    DATA   'Pname'     (Program to be requested)
                or
         DEC    Pnumber
(L+3)    DAC    Errtn
(L+4)    OCT    Param       (OCT 0 if commun. not required)
          or
         DAC
(L+5)    . . . . . .        (Normal return)
```

and the FORTRAN format is:

```
                 |Pname   |
CALL REQST   ( |Pnumber|, Status [, Param ])
```

where:

Pname	is a 2-letter ASCII program name
Pnumber	is an integer value that specifies the position of the program's entry in the SPET table of SIM (for faster execution than search for 'Pname')
Errtn	is an error return address (PMA only)
Status	is set to indicate the status of the request (FORTRAN only)
Param	is a single communication parameter or a pointer to a communication buffer

The communication parameter or pointer, if present, is handled by the queuing routine for eventual transfer to the header of the requested program.

In PMA, if the error return is taken, an error code is present in the A register. In FORTRAN, the usual return is taken but a status code is present in the 'Status' variable. The codes are:

Code	Condition
0	Function accepted
1	Function rejected - no such program name or number
2	Function rejected - communication buffer full of parameters.

The effect of a successful Request Program function depends on the relative priority of the requesting and requested programs. Assume for example that program AA is requesting program BB. Program BB is always set to 'requested' status. If AA has higher priority, processing resumes at the 'normal return' location of program AA. Program BB is started at the starting location specified in its SPLT entry when it has high priority.

If program BB is higher in priority than AA, the Executive attempts to start program BB and place program AA in 'interrupted' status. If either of these operations is not possible, program AA resumes at the normal return location. (A program cannot be placed in 'interrupted' status if the maximum number of interrupting devices (SID1 in SIM) is exceeded. A program cannot be started if there is a coordination conflict, etc.)

If the requested program is disk resident and not in memory, the Executive requests the SYSLDR routine in order to read the program from disk. The program is then started, if possible.

The pending counter of the requested program is incremented the first time the program is requested and once more on the first request after the "ran once" flag is set. Thereafter, it is incremented only during requests that use the communication option. The counter is decremented when the program is started.

The following PMA example requests program QR, specifies error return XYZ and does not use communication:

```
SVC
DEC     1       (request program)
DATA    'QR'    (program name is QR)
DAC     XYZ     (error return)
OCT     0       (communication not used)
```

In FORTRAN, the same function is:

```
INAM= 'QR'
  .
  .
  .
  .
CALL REQST(INAM,ISTAT)
```

SCHEDULE LABEL (FUNCTION 2)

This function allows one program to restart another waiting program (or itself) at a specified location (label). The program in which the label is scheduled must be memory-resident and active (i.e., either 'waiting' or 'interrupted'). It is started at the specified label when it is the highest priority active program. After the function es executed, the normal (or error) return is taken. The pending counter is incremented.

The PMA calling sequence is:

```
(L  )   SVC                 (Function handler entry)
(L+1)   DEC   2             (Function number)
(L+2)   DATA  'Pname'       (Program in which label
            or               is to be scheduled)
        DEC   Pnumber
(L+3)   DAC   Errtn
(L+4)   DAC   Label
(L+5)   . . . . . .         (Normal return)
```

and the FORTRAN format is:

```
                    | Pname   |
CALL SCHED  (       | Pnumber | , Status,  Label  )
```

where:

Pname	is a 2-letter ASCII program name
Pnumber	is an integer value that specifies the position of the program's entry in the SPET table of SIM (for faster execution than search for "Pname')
Errtn	is an error return address (PMA only)
Status	is set to indicate the status of the request (FORTRAN only)
Label	is the label to be scheduled

In PMA, if the error return is taken, an error code is present in the A register. In FORTRAN, the usual return is taken but a status code is present in the 'Status' variable. The codes are:

Code	Condition
0	Function accepted
1	Function rejected - no such program name or number
2	Function rejected - program header is full of labels to be scheduled

A primary use of SCHEDULE LABEL is to allow a program which services another to call the first one back after service is complete. For example, a program that requests a driver can pass its own program name and a return label to the driver by means of the communication option. When the driver has finished an operation, it returns to the original program by scheduling the label passed to it.

CONNECT CLOCK (FUNCTION 3)

This function sets up a program for automatic initiation by the clock program at recurrent intervals or once after a time delay. A flag bit determines whether the clock program will use a REQUEST PROGRAM or SCHEDULE LABEL function to initiate a program; thus the program can be entered either at the starting address (in the program's entry in the SPLT table of SIM) or at a label provided in the A Register.

The PMA calling sequence is:

```
(L  )   SVC     3           (Function handler entry)
(L+1)   DEC     3           (Function number)
(L+2)   DATA    'Pname'
          or
        DEC     Pnumber
(L+3)   DAC     Errtn
(L+4)   DEC     Delay
(L+5)   DEC     Interval    (OCT 0 for execution once after
                             a time delay
(L+6)   OCT     Flag/Base
                Freq.
(L+7)   . . . . . .         (Normal return)
```

and the FORTRAN format is:

```
                | Pname    |
CALL CONCLK  ( | Pnumber  | , Status, Delay, Interval, Base Freq.[,Label])
```

where:

Pname	is a 2-letter ASCII program name
Pnumber	is an integer value that specifies the position of the program's entry in the SPET table of SIM (for faster execution than search for 'Pname')
Errtn	is an error return address (PMA only)
Status	is set to indicate the status of the request (FORTRAN only)
Delay	is the number of base frequency units to elapse before first execution
Interval	is the number of base frequency units between periodic executions

Flag/Base Freq.	PMA: contains the schedule label flag (bit 01) and a base frequency code (bits 14-16)
Base Freq.	is the base frequency code (FORTRAN)
Label	is an optional label for periodic or delayed execution (FORTRAN). (Should be 0 for REQUEST PROGRAM function.)

In PMA, the label to be scheduled must be in the A register before this function is entered.

Error Codes

In PMA, if the error return is taken, an error code is present in the A register. In FORTRAN, the usual return is taken but status code is present in the 'Status' variable. The codes are:

Code	Condition
0	Function accepted
1	Function rejected - no such program name or number
2	Function rejected - maximum number of clock users in SCUT table of SIM is exceeded

Periodic Execution

When the specified 'Base Freq' is 0 to 3, this function requests or schedules a label in the specified program when it comes due. A program can connect itself or another program in this manner.

The 'Delay' parameter specifies the elapsed time before the program is first executed, and 'Interval' specifies the interval between successive executions. (During these intervals, the program is terminated.) See Table 4-1 for time units. Once a program has been set up for this mode of operation, it continues execution at intervals until reset by a DISCONNECT CLOCK function.

Table 4-1. Base Frequencies for Clock Calls

Base Frequency	Definition
For periodic execution	(Program requests itself or another program)
0	Time until first execution is absolute time of day in minutes. Interval between executions is in minutes thereafter.
1	Time until first execution and interval between executions is at RTC interrupt rate.*
2	Time until first execution is in seconds. Interval between executions is in seconds.
3	Time until first execution is in minutes. Interval between executions is in minutes.
For time delays	(Program delays itself)
4	Time delay until resumption of execution is at RTC interrupt rate.*
5	Time delay until resumption of execution is in seconds.
6	Time delay until resumption of execution is in minutes.

*RTC interrupt rate is determined by the CLK2 and CLK3 entries in SIM. See Clock Program description in Section 2.

In the following example, program 'KL' is set up to be requested after an initial delay of 300 ms and thereafter at 1-second intervals (assuming a 50 ms RTC interrupt period):

PMA:

```
        SVC
        DEC     3           (connect clock)
        DATA    'KL'        (program name)
        DAC     PQR         (error return)
        DEC     6           (6 x 50 = 300 ms delay)
        DEC     20          (20 x 50 = 1000 ms intervals)
        OCT     1           (RTC int. rate)
```

To schedule a label instead of requesting the program, word L+6 could be OCT 100001.

FORTRAN:

```
      DATA    INAM,IDEL,INT,IBASE/'KL',6,20,1
      .
      .
      .
      CALL CONCLK (INAM,ISTAT,IDEL,INT,IBASE)
```

One-Time Execution

If 'Interval' is zero and the base frequency code is 1 to 3, the clock program requests or schedules a label in the specified program, and disconnects the program from the clock the first time it falls due. A program initiated in this way executes once only, and need not disconnect itself to prevent periodic execution.

Delayed Execution

If the base frequency code is 4 to 6, a program can delay itself for the duration of the 'delay' value. See Table 4-1 for delay intervals. ('Interval' should be set to 0.) The program is placed in a waiting state during the delay. Upon completion of the time delay, the program is disconnected from the clock and scheduled to resume at the normal return. In this example, program 'KL' delays itself for 2 minutes:

PMA:

```
        SVC
        DEC     3
        DATA    'KL'
        DAC     PQR
        DEC     2               (2 x 1 min. delay)
        DEC     0               (no interval)
        OCT     6               (1 min. delay units)
        . . . . . .             (Return location)
```

FORTRAN:

```
        DATA    INAM,IDEL,INT,IBASE/'KL',2,0,6
        .
        .
        .
        CALL CONCLK (INAM,ISTAT,IDEL,INT,IBASE)
```

Schedule Label vs. Request Program

In PMA, if bit 01 of word L+6 is a 1, the Request Clock function schedules a label in the specified program. The label must be in the A register before the function is entered. If bit 01 of L+6 is 0, the effect is the same as a Request Program function.

In FORTRAN, if the 'Label' parameter is present and non-zero, the Executive schedules that label. If 'Label' is 0 or omitted, the specified program is requested.

DISCONNECT CLOCK (FUNCTION 4)

This function is used to remove a specified program from the periodic execution mode (as set up by CONNECT CLOCK), or to cancel the automatic resumption of a self-delayed program in the waiting state.

The PMA calling sequence is:

```
(L  )    SVC                (Function handler entry)
(L+1)    DEC  4             (Function number)
(L+2)    DATA 'Pname'       (Program to be disconnected)
           or
         DEC  Pnumber
(L+3)    DAC  Errtn
(L+4)    OCT  Base Freq.    (Table 4-1)
(L+5)    . . . . . .        (Normal return)
```

and the FORTRAN format is:

```
                  |Pname   |
CALL DISCLK  ( |Pnumber| , Status, Base Freq.)
```

where:

Pname	is a 2-letter ASCII program name
Pnumber	is an integer value that specifies the position of the program's entry in the SPET table of SIM (for faster execution than search for 'Pname')
Errtn	is an error return address (PMA only)
Status	is set to indicate the status of the request (FORTRAN only)
Base Freq.	is the same base frequency code specified in the original connect clock function

In PMA, if the error return is taken, an error code is present in the A register. In FORTRAN, the usual return is taken but a status code is present in the 'Status' variable. The codes are:

Code	Condition
0	Function accepted
1	Function rejected - no such program name or number
2	Function rejected - specified program is not connected to the clock

CONNECT INTERRUPT (FUNCTION 5)

This function defines the starting address of interrupt response code for a particular device hardware interrupt and increments the pending counter of the calling program. The address ('Intresp', below) is entered as the third word of the device's entry in the SID1 table of SIM. Thereafter, when the interrupt occurs, the Executive transfers control to the specified entry point and decrements the counter. The response code must be in the calling program itself.

The PMA calling sequence is:

```
(L  )    SVC                (Function handler entry)
(L+1)    DEC  5             (Function number)
(L+2)    OCT  Devadd
(L+3)    DAC  Errtn
(L+4)    DAC  Intresp
(L+5)    . . . . . .        (Normal return)
```

and the FORTRAN format is:

```
CALL CONINT (Devadd, Status,
```

where:

Devadd	is the device address (Appendix A)
Errtn	is an error return address (PMA only)
Status	is set to indicate the status of the request (FORTRAN only)
Intresp	is the starting address of the interrupt response code.

In FORTRAN, the label identified by 'Intresp' should be a CALL INTSET statement, and the interrupt reponse code should terminate with a CALL ISKED statement (both described later in this section).

In PMA, if the error return is taken, an error code is present in the A register. In FORTRAN, the usual return is taken but a status code is present in the 'Status' variable. The codes are:

Code	Condition
0	Function accepted
1	Function rejected - interrupt already connected or device does not exist

The following example connects device no. 3 (line printer) to interrupt response code starting at symbolic location LPINT, with an error return location LPER specified:

PMA:

```
        SVC
        DEC     5               (connect interrupt)
        DEC     3               (device no. 3)
        DAC     LPER            (error return)
        DAC     LPINT           (int. response code)
```

FORTRAN:

```
        IDEV = 3
         .
         .
        CALL CONINT (IDEV,ISTAT,$100)
         .
         .
         .
    100 CALL INTSET
         .
         .
        etc.
```

DISCONNECT INTERRUPT (FUNCTION 6)

This function dissociates a block of interrupt response code from a device by turning off the mask bit, resetting the SID1 entry to return to the Executive, and decrementing the pending counter.

The PMA calling sequence is:

```
(L  )   SVC                 (Function handler entry)
(L+1)   DEC     6           (Function number)
(L+2)   OCT     Devadd
(L+5)   . . . . . .         (Normal return)
```

and the FORTRAN format is:

```
CALL DISINT (Devadd)
```

where:

Devadd is the device address (Appendix A).

This function always takes the normal return. These are no error codes.

TERMINATE (FUNCTION 7)

A program can use this function to inform the Executive that it has finished execution. If the pending counter is 0, the program is terminated. (The status word is cleared except for the 'Ran Once' bit, which is set, and the coordination bits used by the program are cleared from the Master coordination word.)

If the pending counter is non-zero, the executive determines whether the program has been requested or if there are any labels to be scheduled. If so, the executive does what is required. If not, the program has apparently tried to terminate with an interrupt connected. In that case, the Executive sets the 'Disabled' bit in the program's status word, so that the program cannot run again. (Unrecoverable error.)

The PMA calling sequence is:

```
(L  )   SVC   0          (Function hander entry)
(L+1)   DEC   7          (Function number)
```

and the FORTRAN format is:

```
CALL TERM
```

WAIT (FUNCTION 8)

A program uses this function to voluntarily suspend execution when the program is to be restarted at label by another program or by its own interrupt response code. This function sets bit 12 (Waiting) of the calling program's status word and returns to the scheduler. To restart a program after a Wait, a Schedule Label function must be used.

The PMA calling sequence is:

```
(L  )   SVC     (Function handler entry)
(L+1)   DEC  8  (Function number)
```

and the FORTRAN format is:

```
CALL WAIT
```

(Note: In Revision 3 of RTOS, this function name is changed to XWAIT to distinguish it from the ISA function WAIT.)

FORTRAN INTERRUPT EXTENSIONS

Three special functions are included to aid the user in setting up interrupt response code in a FORTRAN program. They are:

```
CALL INTSET
```

which must be the first instruction in user interrupt response code:

```
CALL INTACK (Iack)
```

where Iack is executed as an instruction such as an OCP interrupt acknowledge; and

```
CALL ISKED (Label, Entry)
```

where 'Label' is the label to be scheduled on exit from the interrupt response code and 'Entry' is the label associated with the CALL INTSET statement. If no label is to be scheduled, 'Label' must be 0. The following example shows how these functions can be used:

```
C       SETUP OPERATIONS:
C       OCP '0160
           IACK = '30160
C       CONNECT INTERRUPT TO DEVICE 'IDEV'
           CALL CONINT (IDEV, ISTAT, $100)
C       INTERRUPT RESPONSE CODE
100        CALL INTSET
           CALL INTACK (IACK)
           .   (interrupt response processing -
           .   minimum execution time)
C       SCHEDULE OPTIONAL LABEL IN NON-INTERRUPT CODE
           CALL ISKED ($200, $100)
           .   (control returns to scheduler)
           .
C       NON-INTERRUPT FOLLOW-UP CODE
200        (Follow-up processing)
           .
           .
           CALL DISINT (IDEV)
           CALL TERM
```

EXECUTIVE AND USER EXTENSIONS

The following functions are provided as a convenience to the user:

CALL FETPAR (Ibuff, Iwords)

This function enables a program to access parameters passed to its header from another program by the Executive's queueing routine. 'Iwords' is the number of words transferred. If 'Iwords' is greater than 1, 'Ibuff' is the name of an integer array that will receive the parameters. If 'Iwords' is 1, 'Ibuff' is the parameter itself.

CALL ERPRNT(Icode, Iname)

This function enables a FORTRAN program to call the executive's error print routine. 'Icode' is an error code of up to 3 octal digits, and 'Iname' is the two-letter ASCII name of the calling program. The ASR error printout format is described in Section 2.

ISA EXTENSIONS

The following calls are alternate versions of the Connect Clock executive function. These forms are supplied for compatibility with ISA Standard S61.1, 1972.

CALL START (K, J, K, M)

This function has the same effect of a Connect Clock function with a Base Freq. value of 0-3 and an interval of 0. When the specified delay has elapsed, the program is placed in the 'Requested' state and disconnected from the clock. The parameters are:

I — Two-letter ASCII program name (or number)

J — Number of time units to delay before starting

K — Specifies the time units:

- 0 — Real time clock interrupt rate (depends on CLK2 and CLK3 entries in SIM)
- 2 — Seconds
- 3 — Minutes

M — Returns the status of the request to the calling program:

0 or less	Undefined
1	Request Accepted
2 or greater	Request Rejected

CALL TRNON (I, J, K)

This function has the effect of a Connect Clock function with an interval of 0 and a starting time expressed as a time of day. When the specified time matches the Executive's time-of-day count, the program is placed in the 'Requested' state and disconnected from the clock. The parameters are:

I — Two-letter ASCII program name (or number)

J — Array name. First three words of array must contain hours, minutes, and seconds value, in that order, for time of day.

K Return the status of the request to the calling program:

0 or less	Undefined
1	Request Accepted
2 or more	Request Rejected

CALL DELAY (I, J, K)

This function enables a program to delay itself for a specified period. The program is placed in the 'Waiting' state and connected to the clock. When the delay has elapsed, the program is scheduled to resume at the next executable statement and disconnected.

NOTE:

In Revision 3 of RTOS, this function is CALL WAIT and the normal WAIT function is CALL XWAIT.

The parameters are:

I Number of delay units to elapse

J Time units

0	RTC clock rate*
2	Seconds
3	Minutes

*Determined by CLK2 and CLK3 entries in SIM

K Returns the status of the request to the calling program:

0 or less	Undefined
1	Request Accepted
2 or more	Request Rejected

SECTION 7

USING RTOS

This section describes user program requirements, startup and operating procedures, and various options available to the experienced RTOS user.

USER PROGRAM REQUIREMENTS

Programs to run under RTOS must have the header described below and conform to all other requirements of RTOS. The general format is shown in Figure 7-1.

Header

Every RTOS program must begin with the following header:

```
        SUBR    Name1, Name2 . . .      (Program identification
                                         for program linkage)

        REL

        DEC     -1                      (Upper terminator of
                                         header)

        BSZ     n                       (Reserves n words for
                                         scheduled labels;
                                         n=1 to 10)

        BSZ     1                       (Reserved for a communication
                                         parameter)

Start   ---     ---                     (Starting address specified
                                         by program's entry in
                                         SPLT table of SIM)
```

This arrangement permits the executive to access parameters by a negative offset from the program's starting address, available in SPLT. Any number of words can be reserved for labels. A typical value is 4. A communication word must be reserved even if the communication option is not used. Figure 7-2 shows the condition of a typical header during program execution.

For programs that use the communication option, the system's queueing routine (FIFO, etc.) places a communication parameter in the word preceding the starting address before the program is started. Communication is discussed in Section 2.

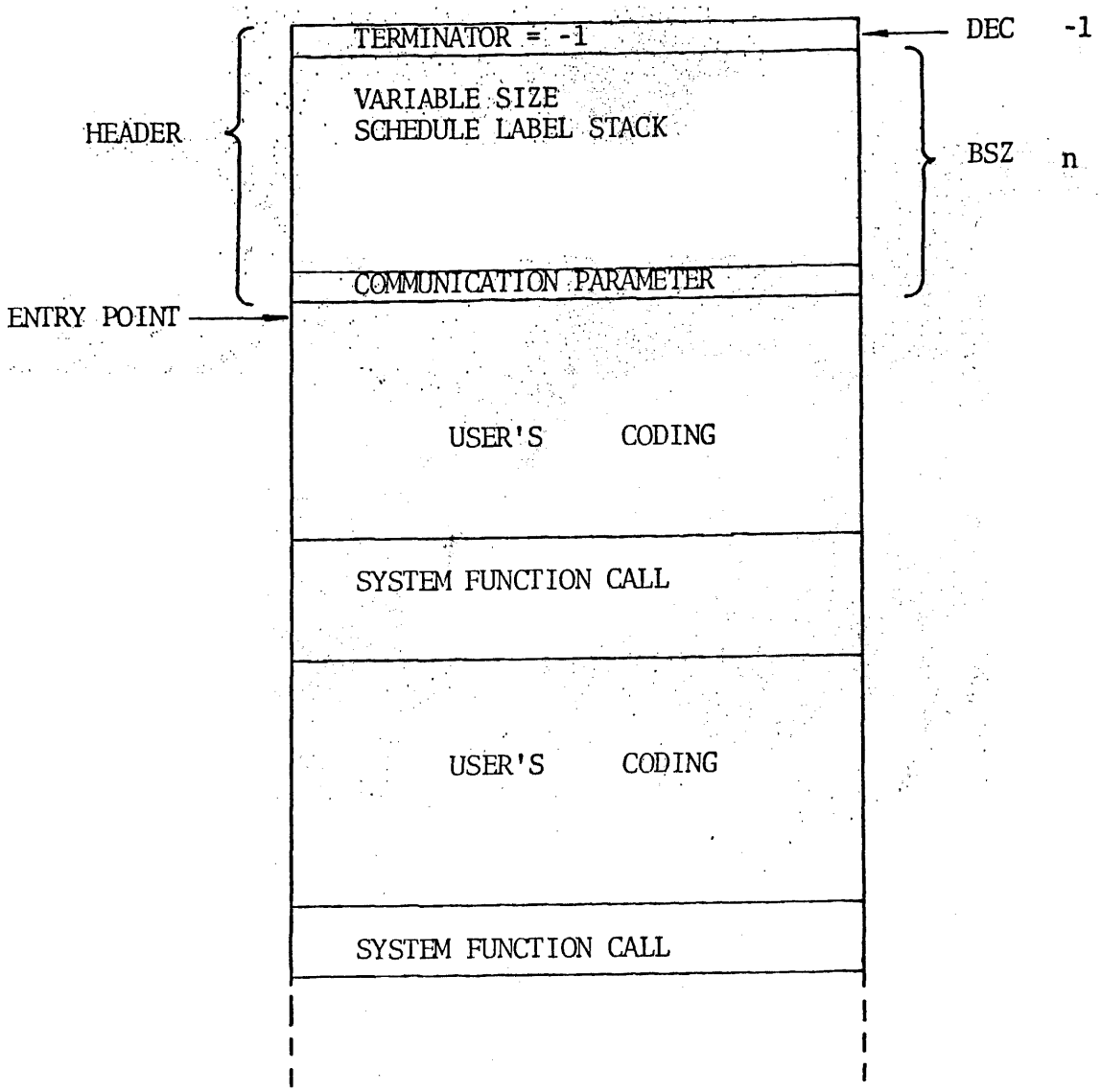
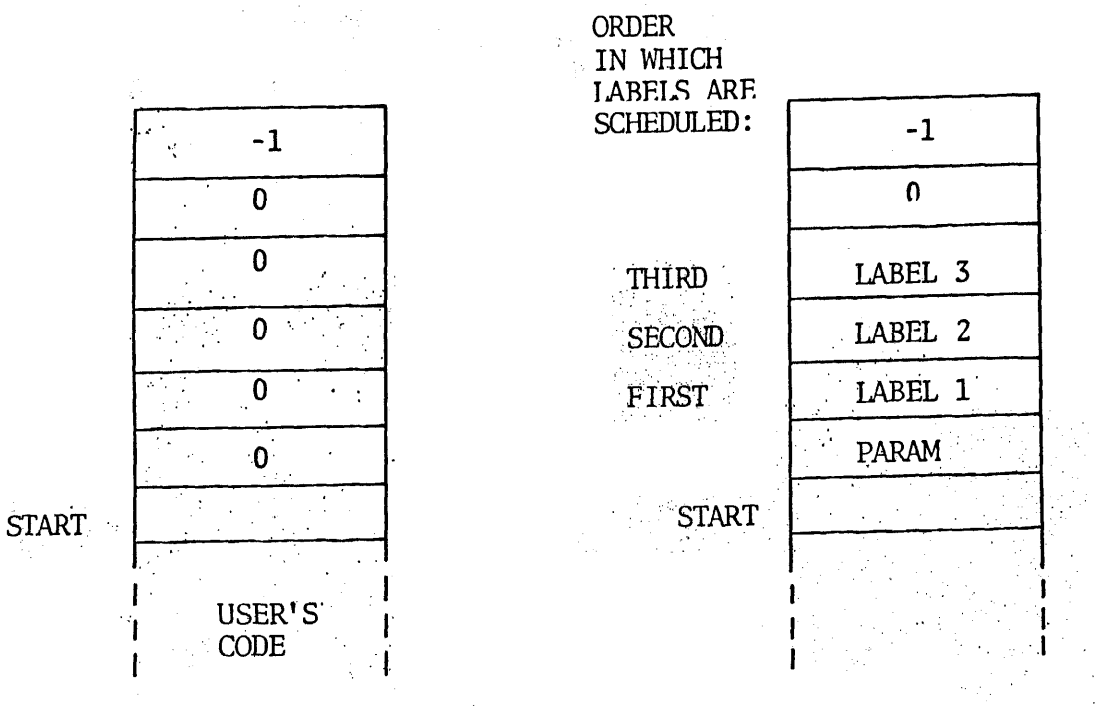


Figure 7-1. User Program Format



A. Starting Condition

B. Three Labels queued and a communication parameter present.

Figure 7-2. Use of Program Header During System Execution

When labels are scheduled, they are placed on the stack starting with the word preceding the communication word. If the system attempts to enter labels when all reserved words are used, the executive prints an error message. Labels are started in the order in which they were scheduled. As each label is started, the Executive pushes the stack toward the communication word and leaves a zero word as space for another label.

Program Name

User programs that are written to be link-loaded as programs external to the executive should begin with a SUBR statement that defines the names to be cited in the executive and other programs.

In executive functions, programs are identified by two-letter ASCII program names or an octal number equal to the program's offset position in the SPET table of SIM. These references are related to the program's actual starting location by the starting address location of the program's SPLT entry. For example, assume the user has prepared a general-purpose signal conditioning routine name GS, with a starting location identified by the label SIGCON. If the program begins with a statement such as:

```
SUBR    GS, SIGCON
```

the program's SPLT entry should begin with:

```
GSP     DATA    'GS'
        XAC     GS      (Starting address)
        .
        .
```

If the program is disk-resident or loaded separately, the actual starting address can be entered, as in:

```
GSP     DATA    'GS'
        OCT     '10400
```

Interrupt Response Code

In order to respond to device hardware interrupts, RTOS programs must issue a Connect Interrupt executive function that associates the device interrupt with an interrupt response subroutine entry point elsewhere in the program. When the interrupt occurs, the immediate actions are controlled by the device's entry in the SID1 table of SIM. After saving the machine state, the SID1 entry calls the user's interrupt response code as a subroutine. The process is described in detail in Section 2.

The user's interrupt response subroutine must be in the following format:

```
Intresp     DAC     **
              .
              .     (user's response code)
              .
            LDA     Label   (Optional)
            JST*    Intresp
```

The following requirements apply to the response code:

1. For good system response, the code should not require more than 50 microseconds of execution time.
2. Executive function calls are not permitted.
3. The interrupt must be acknowledged by the INA, OTA, or OCP instructions required to clear the interrupt request at the device interface.
4. It should run in SGL arithmetic mode and use true Sector 0.

Figure 7-3 shows the organization of all interrupt processing code for a typical driver program. When the program is first requested, it connects itself to the appropriate device interrupt, does any associated setup operations, and goes into a wait. The interrupt response code provides the immediate response to each interrupt and optionally schedules a label such as RESTART for subsequent processing. It may be necessary for the response code to set a flag (or increment a counter) that is reset (decremented) by the noninterrupt code. If system priorities cause the noninterrupt code to fall behind the interrupts, the interrupt code will find the flag to be set (count non-zero) and can schedule a special return to identify this condition. A flag can also be set when the last operation is complete so that the FINISH label can be scheduled.

The calling program normally passes instructions to a driver through the communication word, which is passed to the driver program's header by the system queueing routine. The word is usually a pointer to a buffer that contains the name of the calling program, normal and error return pointers, the function or mode of operation (for multipurpose drivers) and other necessary information. If a data buffer is used, it normally follows the control words.

When the block transfer or other driver function is complete, the FINISH label is scheduled. This performs actions appropriate to the last interrupt period, disconnects the device interrupt from the driver, and may schedule a label in the calling program. (The label would be one of the items in the communication buffer.) The driver then terminates.

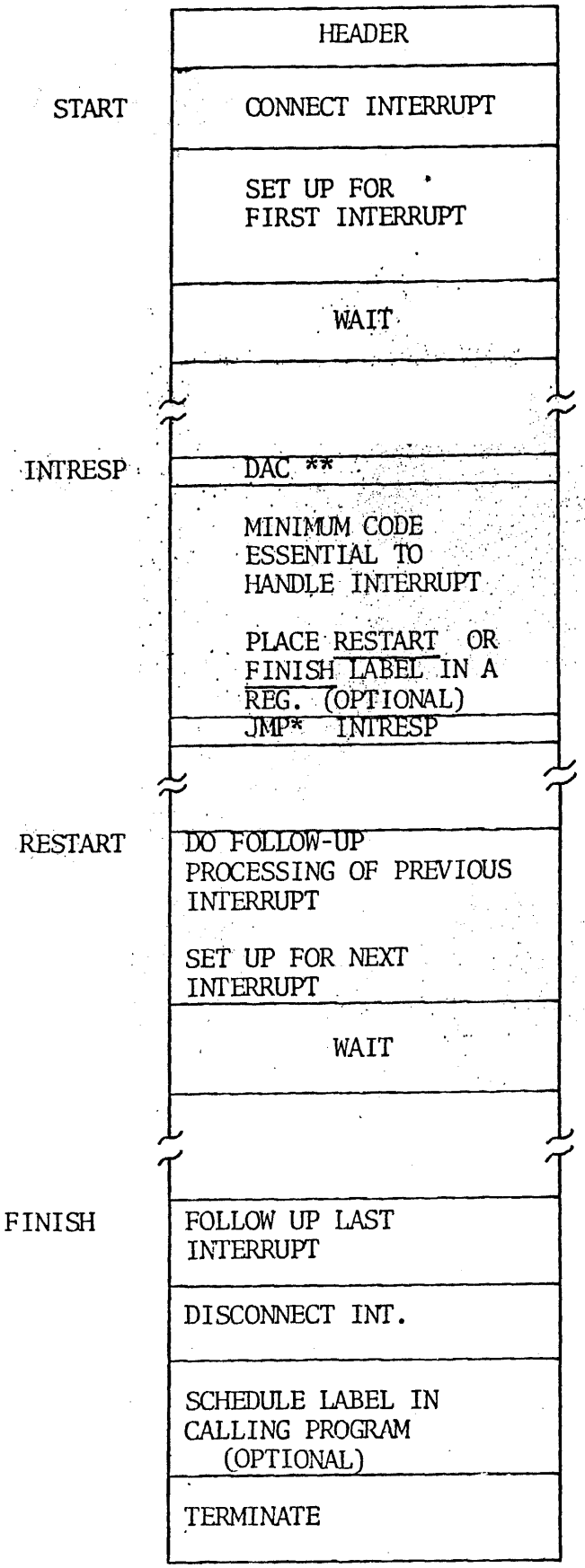


Figure 7-3. Typical Interrupt Response Code in a Driver Program

User-designed drivers should use the standard error print program (ERPRNT) to notify the operator if there are device failures or overloads. The calling sequence and code formats for ERPRNT are described in Section 2.

WRITING SPECIAL QUEUEING SUBROUTINE

Before writing a custom queueing subroutine, the user should thoroughly study the listing of FIFO, the standard queueing subroutine provided with RTOS. When the user's subroutine is called, the following information will be available to it:

1. A pointer to the function call and a flag indicating whether this is to be a fetch or store operation is available indirectly through XLNK. The flag is bit 1; a 1 means a store operation is required, and a 0 means a fetch operation is required.

2. XPCP is the pointer to the beginning of the Executive Program Communication Table.

3. XPLP is the pointer to the requesting program's entry in the Executive Program List Table.

4. The A register contains an index to the proper communication buffer in XPCT.

5. The X register contains a pointer to the first word of the program header of the program being called.

Besides fetching and storing communication parameters, any queueing subroutine must be able to make normal and error returns and to reset the 'communications active' bit in the program's SPLT entry when the buffer is emptied. Much of this may be copied directly from FIFO.

WRITING NEW SYSTEM FUNCTIONS

New system functions may be added to the system by the user. The standard calling sequence is as follows:

(L)	SVC		Function Handler entry
(L+1)	DEC	Function No.	
(L+2)	Parameter		(Optional)
(L+3)	DAC	Error Return	(Optional)

If the name or number of another program is required, it must be specified in location (L+2) of the calling sequence, using a DATA 'Program Name' or a DEC 'Program Number' pseudo-operation. The address of the user's error code, if any, must be specified in location (L+3) of the calling sequence. Other parameters may be specified in locations (L+4), (L+5), etc. of the calling sequence, and one additional parameter may be passed in the A register.

Interaction of Function, Function Handler, and Scheduler

The Function Handler enters the required function at its starting location via an indirect JMP instruction through an SFET pointer. The Scheduler is disabled and the A register is restored to the value it contained on entry to the Function Handler from the user program. If the required function requires a program name search, the Function Handler locates the required program and sets system variable XSP1 to the address of the first word of the associated SPLT entry. Location YLNK contains a pointer to the second word of the function call (the function number).

Following execution, the new system function must return to the Function Handler at label FE10 via a JMP FE10 instruction (if no error occurred) and to label FE20 (if an error occurred). If the return is to label FE10, the B register may contain a parameter to be passed to the user. (This parameter is returned to the user in the A register by the Function Handler.) The A register must contain the return address offset. If the error return to label FE20 is taken, the A register must contain the appropriate error code.

Following execution of the system function, the Function Handler enables the Scheduler and returns control to the appropriate location in the user's calling sequence via an indirect JMP instruction. The A register may contain a parameter. The contents of the other registers are unspecified, and the calling sequence locations are unaltered. If the Function Handler returns to the user's error code, the A register should contain the relevant error code.

When the Function Handler returns to the Scheduler, the saved return address, the keys, and the contents of the A register are available to the Scheduler in the SIVT table.

General Rules

The following general rules should be observed when writing system functions:

1. If the function inhibits interrupts, it should not do so for more than 50 microseconds, and it must enable interrupts before returning to the Function Handler.

2. If a function is to schedule a label, it may use a subroutine within the Executive called SLBL. The calling sequence is:

LDX	Pointer to XPET entry of program in which label is to be scheduled
LDA	Label to be scheduled
JST	SLBL
Normal return	(interrupts inhibited)
Error return	(interrupts enabled)

If the error return is made, the A register will be set as follows:

A = 0	Program not active
A = 1	No room in header for label

The pointer to the relevant SPET entry will be set in XSP1 by the Function Handler, if bit 1 of the relevant pointer in SFET is set.

3. Functions should not normally call other system functions. Exceptions may be made in the case of Wait and Terminate under certain circumstances.

4. User-coded functions should be loaded with the Executive and System Information Module to ensure that all the necessary address links are resolved.

5. If a function is to request a program, use may be made of the Executive subroutine RPRO. The calling sequence is:

```
LDA  (Address of commun, param.)-3

LDB  Pointer to program's SPET entry

JST  RPRO

Error return

Normal return
```

USER INITIALIZATION ROUTINES

The RTOS Initialization routine, INP1 in the Executive, permits the running of user initialization routines after preliminary system initialization has been performed. This enables systems running under RTOS to perform once-only system initialization without using an 'Initialization Program' which would run only once.

A user initialization routine is written as a normal closed subroutine located anywhere in memory. It could be located in a buffer area that is overwritten after the routine has executed once. There may be any number of these closed subroutines or none.

After performing its normal functions and before entering the Scheduler, INP1 does an indirect subroutine jump to a location SINT in the Configuration Module. SINT is a pointer to the initialization control subroutine:

```
*
*   EXECUTIVE INITIALIZATION TABLE
SINT    XAC     INIT        (Pointer to initialization control
                            subroutine
```

If no user initialization is required, SINT should point to entry SC of the Scheduler:

```
SINT    XAC     SC
```

The user's initialization control subroutine consists of a list of JST's to the individual initialization subroutines. It has the form:

```
*
*   INITIALIZATION CONTROL SUBROUTINE
INIT    DAC     **      (Link)
        JST     INT1    (Call initialization routine 1)
        JST     INT2    (Call initialization routine 2)
        JST     INT3    (Call initialization routine 3)
         .
         .
         .
        JST     INTN    (Call initizliation routine N)
        JMP*    INIT
```

ADJUSTING CLOCK RESOLUTION

Two parameters required by the clock program have been made configurable to allow users to adjust the clock resolution. These parameters, labelled CLK2 and CLK3, are located in SIM. CLK2 is the number of clock interrupts per second and CLK 3 is the 2's complement of the number of hardware intervals between interrupts. For a standard system with a 60 Hz line grequency, CLK2 is set to 20 and CLK3 is set to -3, for a clock resolution of 3 x 16.7 ms = 50 ms (20 interrupts per second). In a 50 Hz system, if CLK3 is -2 for 40 ms resolution, CLK2 should be 25.

ELIMINATION OF SYSTEM ROUTINES

RTOS systems may run without the Error Print Program, and the Keyboard Program.

Error Print Program

The following subroutine must be linked into the system in place of the Error Print Program:

```
        SUBR    ED
                REL
ED      DAC     **
        JMP*    ED
        END
```

In SIM, the entries for program EP in SPLT and SPET, as well as the SUBR ER, EPP statement in the header, must be omitted.

Keyboard Program

In SIM, the entries for program KB should be omitted from SPLT and SPET and in the executive, location (AI+5) should be changed to a NOP.

USING RTOS DRIVERS

The standard RTOS drivers (for ASR, paper tape unit, and mass storage devices) are accessible to the user as well as the Executive. All RTOS drivers are entered by a standard Request Program calling sequence. For program names and the format of entries in communication buffer and error printouts, refer to the driver listings.

DEBUGGING USER PROGRAMS

Programs to run under RTOS should first be written as a stand-alone program and debugged as much as possible. Then the RTOS header and address links can be added, the necessary entries can be made in SIM, and the program can be loaded to run under RTOS for final debugging.

USING EXECUTIVE POINTERS AND SIM DATA

The pointers in the Executive and data tables in SIM are in memory at all times, so the user can take advantage of them:

Exec. Location	Points to
XPET	First word of SPET table in SIM
XPLP	First word of current program's data block in the SPLT table of SIM
FE50	SPET table entry of highest-priority program
IHP1	Next available interrupt data block in SIVT table of SIM

For example, the ran-once bit in a program's status word can be checked as follows, to determine whether the program needs to be initialized.

```
LDX*    XPLP        (Pointer to SPLT entry)
LDA     2,1         (Pick up status word)
SAS     6
JMP     Init        (Needs initializing)
...                 (Ran once-continue)
```

To resolve address links in a program that is loaded separately from RTEXEC, load RTOS component XLNKS.

STARTUP AND OPERATION

The result of system building for RTOS A and B is usually a self-loading paper tape of the memory-resident portion of the system. The tape is loaded according to the procedures in the PRIME CPU Operator's Guide. The system is started manually from the control panel at location EXEC (the first location in the executive). Subsequent events depend on the program status configured into SIM. If no programs are requested, the scheduler loops while waiting for interrupts. If the system has on-line utilities, the user can enter a $ character at the ASR keyboard to place the utility keyboard handler in control. It types "SF=" and awaits one of the commands described in Section 5.

ON-LINE PERIPHERAL TEST PROGRAMS

If peripheral device test programs are included in the system, they can be started by RP utility keyboard commands.

ASR Test: SF=RP AT

The user types up to 40 characters at the keyboard and enters a carriage return. The test program types out the same characters and awaits a new input line. This cycle repeats until the user types END.

High Speed Punch Test: SF=RP PC, 1000ddd

Punches 10 records of 'ddd characters each. (If no parameter is given, each record contains '120 characters.) The resulting tape can be used in the reader test.

High Speed Reader Test: SF=RP RT, 100dddd

The punch test tape should be placed in the reader before this function is entered. This function reads one record of 'dddd characters each. (If no parameter is given, the default value is '1000 characters.) If sense switch 2 is set (up), the record is duplicated on the punch.

Disk Test: SF=RP HP Seg

Writes test data to disk segment 'Seg' (0 to 'ddddd'). If sense switch 5 is set, the next segment is selected. If switch 6 is set, the segment is read and compared to the original data.

SECTION 8

RTOS-C

INTRODUCTION

RTOS-C, supplied on revision C of the PRIME master software disk, provides the following extensions to the capabilities of versions A and B:

- DOS disk access and file handling (sequential and mapped random access)
- Dynamic run-time allocation of memory blocks
- General-purpose task and data queueing structures
- Dedicated, coordinated sector 0 areas for disk-resident programs
- Relocatable base sector (virtual-memory systems only)

To implement these features, the following basic changes are made in the RTOS software:

- An abbreviated version of PRIME DOS is provided to be installed as a memory-resident program that runs under RTOS.
- The Executive includes new executive functions for run-time control of memory allocation and queueing by user programs.
- SIM requires a SPLT table entry for resident DOS. The SPLT entry for each user program has format changes to describe characteristics of disk resident programs, and the option word contains fields for dedicated base sector coordination. Memory allocation and queueing features, if present, are configured into SIM.
- Off-line utilities are not required to build disk-resident systems, since on-line DOS may be used to load and save RTOS programs or data as named DOS files.
- A new system loader program, DOSLDR, is provided. It communicates with the Executive to handle DOS-format disk-resident programs with optional dedicated sector 0 or relocatable base sector.

- The RTOS-C software is fully compatible with previous revisions. Memory-only systems and disk-resident systems that use the RTOS-B disk format may be generated by proper configuration of SIM. (When the RTOS-B disk format is used, SYSLDR rather than DOSLDR must be present in the system.)

Interaction of RTOS-C with DOS

It is important to distinguish the two different versions of DOS used in connection with RTOS-C.

Off Line DOS: Regular DOS Rev. 3 is used off-line for system building, debugging, storage of disk resident components, and loading and starting of the on-line RTOS system. After disk-resident programs are loaded, they may be saved as named DOS SAVEd files under a specific UFD devoted to RTOS disk-resident programs. Two external commands have been added to generate DOS files that are to be accessed by random-access methods. RTOSRA creates and names a file of the proper length within a specified UFD and disk unit. FILBLK is used to exchange data between memory and a disk file created by RTOSRA; it is equivalent to the RTOS-B off-line utility functions TRDM and TRMD.

On-Line DOS File Manager: RTOS-C requires the on-line use of a modified version of DOS that acts primarily as a DOS file manager. This program is identified by the SDOS entry of SIM:

SDOS	DAC	DOO	DAC to SPET for DOS
	OCT	2	Logical disk device containing RTOS UFD
	DATA	C'xxxxxx'	RTOS UFD name for programs
	DEC	n	No. of random access files

The UFD 'xxxxxx' is the dedicated UFD for RTOS disk-resident programs which are filed in the DOS 'SAVE' format under 2 character RTOS names. If DOS-compatible random access files are used, the number 'n' must be specified; otherwise n=0.

The DOS file manager gives the user access to the DOS FORTRAN subroutines (CALL READ, CALL SEARCH, etc.) described in Section 11 of the DOS manual. When such a CALL is encountered, program DO is requested. It processes the call and schedules a return label in the calling program.

Using the CALL READ and CALL WRITE subroutines, the user can create and read named sequential data files from FORTRAN or assembly-language programs without having to deal directly with the disk driver. (However, the user has the option of assembly language calls to the disk driver to communicate with DOS random access files as well as RTOS-B type non-DOS compatible random access files.) All other convenience features of DOS (CALL TNOU, CALL CMREAD, etc.) are also accessible to user programs.

Disk Organization

RTOS-C disk records are maintained in the standard DOS format of chained random records scattered over the disk. These are compared with the RTOS-B structure of 128-word segments in Figure 8-1. The RTOS-C executive allows the user to access disk records by either method.

Disk Resident Programs

Disk resident programs in RTOS-B are written to disk in 128-word segments by off-line utility commands. When such a program is requested by the scheduler, the system loader program (SYSLDR) communicates with the disk driver and identifies the program by the starting segment number. In RTOS-C, disk-resident programs are stored on the DOS disk as SAVEd files identified by two-letter filenames. A different system loader (DOSLDR) identifies the program by a two-letter RTOS program name, which is the same as the filename of the program in the dedicated RTOS UFD. To support this operation, each disk-resident program's SPLT entry specifies the number of memory words occupied by the program.

Action of RTOS-C System Loader (DOSLDR)

Under RTOS-C, disk segment addresses are not required since each program is located by its two-letter RTOS filename. Therefore, the first time the system loader runs, it locates the RTOS UFD specified in the SDOS entry of SIM, and keeps it memory resident for the current and subsequent load requests. The loader searches that UFD for the file having the specified 2-letter program name and the first record is located. The disk driver is then requested to transfer the program data for this record into memory. The next record address is located and the disk driver is called again. In this manner, the file's forward pointer is followed until the entire program is loaded.

Sequential Access Data Files

Data in named, sequential DOS files may be stored or accessed by FORTRAN formatted I/O or by the DOS subroutines CALL READ and CALL WRITE (or their assembly language equivalents). (See "User Program Requirements".)

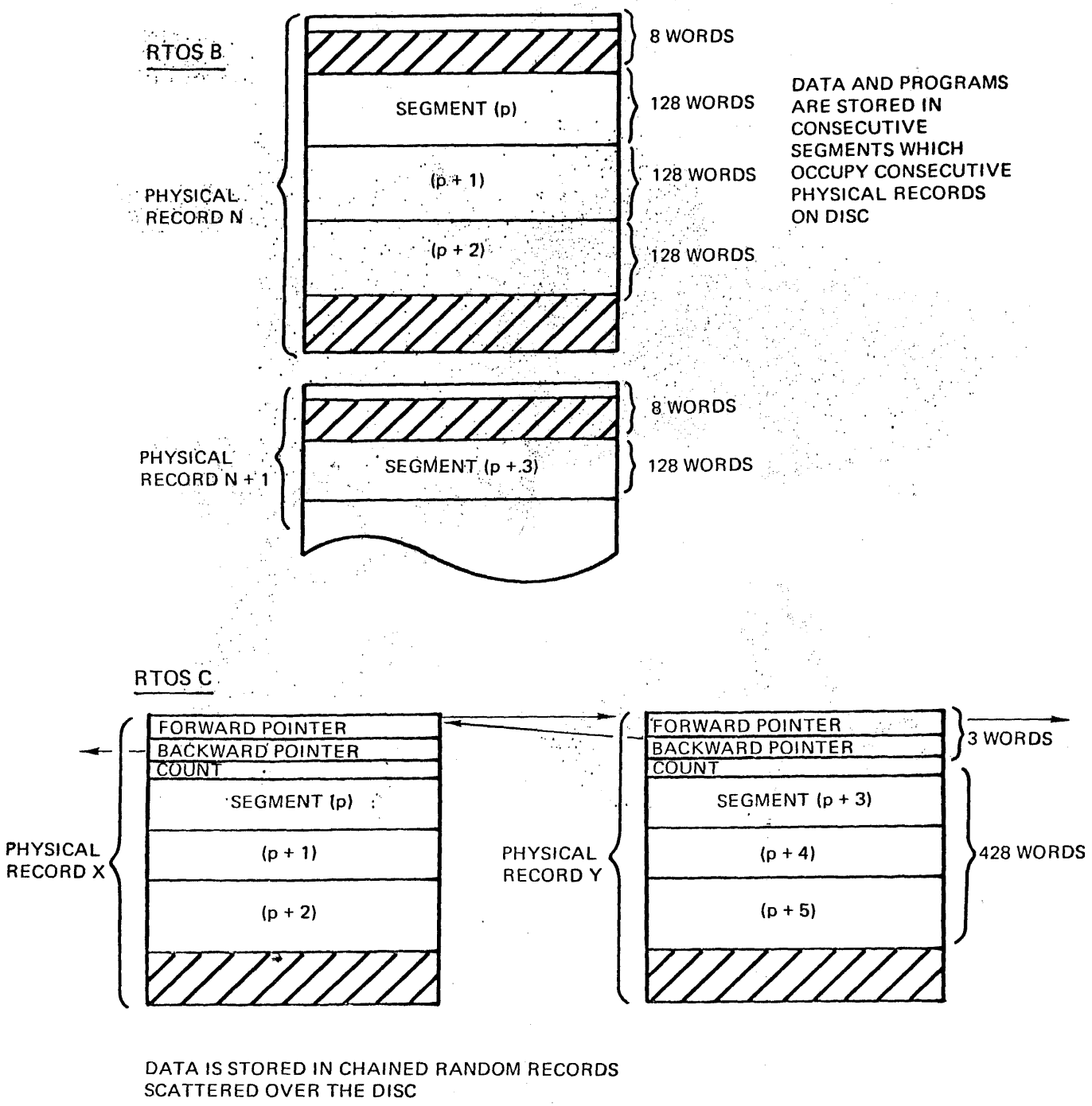


Figure 8-1. RTOS-B and RTOS-C Disk Formats Compared

Random Access Data Files

When the linked file structure of DOS is used by a real time program for data transfers, each disk record must be read in turn in order to follow the forward pointer until the desired record is reached. As an alternative, the disk driver used with RTOS-C permits a form of random access to 128-word segments in named DOS data files. To support this type of access, two external commands (described under "SYSTEM BUILDING") are added to off-line DOS. The RTOSRA command creates a random-access file of the required size in a specified UFD and logical disk unit. The FILBLK command transfers data between such files and memory; it is equivalent to the TRMD and TRDM off-line utility commands of RTOS-B. For each random-access file in the system, space for a mapping buffer must be reserved in SIM.

Such files are accessed at run-time by direct calls to the disk driver. (Calling formats and parameters are defined under "RTOS-C USER PROGRAM REQUIREMENTS".) The random access files are defined to the RTOS system by the following entries in the SSPT table:

```
        OCT                             (Logical disk device)
        DATA    C'Filnam'               (File name)
        DATA    C'Ufdnam'               (UFD name)
        DAC     Mapbuf                  (Pointer to mapping buffer)
        .
        .
        .
Mapbuf  BSZ     Size                    (Buffer Size)
```

The first time the disk driver is called, it reads every segment in the chained file to create a mapping buffer. The mapping buffer maps logical sequential record numbers against actual physical non-sequential record numbers. Thereafter, the driver reads and writes 128-word segments as in the usual RTOS-B operation.

RTOS-C EXECUTIVE EXTENSIONS

The RTOS-C executive performs all the functions described in Section 2 and implements the following new features in conjunction with the modified System Information Module (SIM) and system loader program (DOSLDR).

There are no significant changes to the interrupt handler, FIFO communication option, clock program, or error print program. The main differences appear in the initialization logic, scheduler, and function handler.

The initializing routine automatically loads the memory-image of the DOS file manager (disk-resident program D0) by requesting the DOSLDR during startup. Also, if SIM is configured for block allocation of free memory, the blocks are linked and initialized at this time.

The scheduler checks dedicated sector 0 coordination and the need for base sector relocation before starting a requested program. Physical to virtual mapping of base sectors is done, if required.

The function handler is expanded to support the new executive functions for queue handling and memory block allocation. It also checks unidentified SVC calls to see whether they are calls to DOS subroutines. If so, program D0 is requested and the calling program is placed in WAIT status.

Disk Based Programs and the Base Sector

It is convenient to allow disk resident programs that are not desectorized (i.e. that generate cross-sector references). This can be achieved (in both RTOS-B and RTOS-C) by leaving all base sector references for disk based programs in memory permanently. For systems containing a large number of disk resident programs, this is impractical since sector zero will soon become full. To solve this problem, RTOS-C provides the following enhancements:

1. Base sector coordination - when hardware paging is not available.

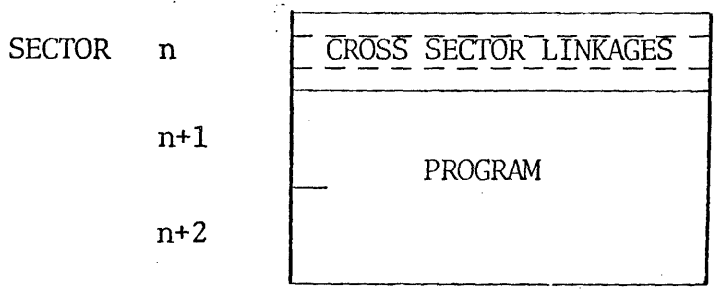
2. Base sector relocation - when hardware paging is available.

Base Sector Coordination: Disk resident RTOS programs can reserve parts of true sector 0 into which the disk-resident base sector references are loaded at program load time. The option word in each program's 'SPLT' entry specifies those parts of the base sector used by the program and coordinates their use between programs. The base sector is divided into the following parts:

```
'200 - '377    (Option word bit 2)
'400 - '577    (  "      "    "  3)
'600 - '777    (  "      "    "  4)
```

A program can coordinate one or more of these parts.

This feature is implemented when a program is link-loaded during system building. The user selects the B Register option of the linking loader to generate cross sector linkages in the specified parts of the sector immediately below the sector containing the program:

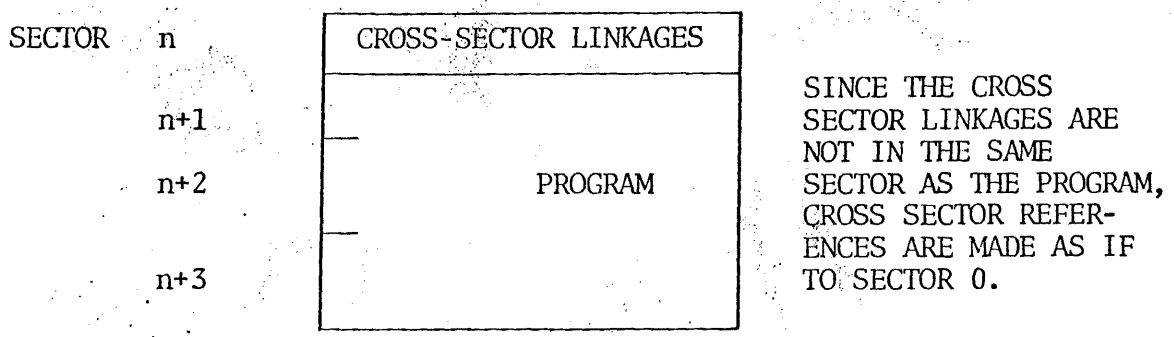


The above sectors n through n+2 are stored on disk consecutively. Those parts of the base sector containing cross-sector linkages are specified in the program's SPLT entry. For more information, see "RTOS-C SYSTEM Building".

When a disk resident program is loaded, the specified parts of sector 0 must be available (no coordination conflicts). The system loader uses the 'scatter read' mechanism of the disk to load the cross-sector linkages into the specified parts of sector 0 and the program proper into its specified area. Disk-resident programs using the same parts of sector 0 are thus mutually exclusive.

Base Sector Relocation: In Virtual memory PRIME system, both memory and disk-resident RTOS programs may individually relocate their base sector to any other sector in memory. Base sector relocation is achieved by the paging hardware option. The physical sector actually used as the base sector for the program is specified by the communication word of each program's 'SPLT' entry (Figure 2-6).

This feature is implemented when a program is link loaded during system building. The user selects the B Register option of the linking loader to generate linkages in the sector preceding the program:

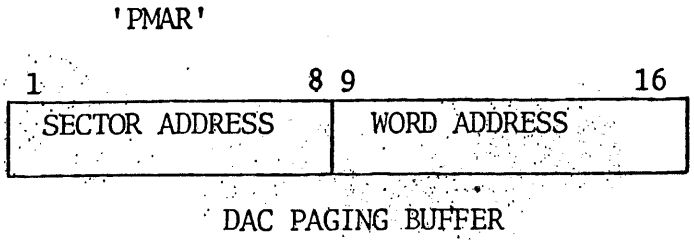


The above sectors n through n+3 are stored on disk consecutively by a DOS SAVE (RTOS-C) or the off-line utility (RTOS-B) if the program is disk-resident. The communication word of the program's 'SPLT' entry specifies the sector containing the cross-sector linkages (the relocated base sector for the program). Virtual memory is used to 'map' virtual sector 0 to the relocated sector whenever this program is running. Each memory reference instruction is elongated by 80 ns.

To support Base Sector Relocation SIM must contain the entry:

```
          SSPT        DAC          PAGINGBUFFER
          .
          .
          .
PAGINGBUFFER          BSZ          128
```

During initialization, the executive sets 'PMAR' (Page Map Address Register), location '10, as follows:



DAC PAGING BUFFER

and maps virtual memory to physical memory in the page map so that virtual sectors are equated to physical sectors. Page maps must begin on a 1/2- sector boundary.

Whenever a program is started or restarted from the scheduler or interrupt response, the relocated base sector is mapped as the physical sector, addressed as virtual sector 0, and paging mode ('EMPJ') is entered. Whenever the program returns control to the executive, normal mode is restored by the 'SVC' interrupt generated by each system function call.

Memory Block Allocation

The RTOS-C Executive includes a capability to organize memory reserves into non-contiguous blocks of various sizes and allocate them on demand from user programs.

Several additional executive functions (Fetch Block, etc.) enable user programs to fetch blocks for use and return them to the pool of reserve blocks when they are no longer required. The executive, in conjunction with the SBLK tables configured into SIM, handles the stringing and allocation of the blocks. Executive functions for block handling are:

FBLCK	Fetch Block
CFBLCK	Conditional Fetch Block
RTNBLK	Return Block
RTNSTG	Return String of Blocks
CHECKB	Check Block Count

Calling sequences and functions are defined later in this section.

Figure 8-2 shows how a group of blocks of particular size are configured and allocated. Initially, a block definition entry in the SBLK area of SIM defines a number of blocks of a particular size. (Figure 8-2A.) A pointer designates the first block of the group, and the first word of each block points to the next block. (The last block contains a pointer of 0.) This is done automatically by the executive during initialization. All block pointers are 16-bit absolute addresses. Figure 8-2B shows a situation that could occur after a period of active operation. At different times, program X has fetched two different blocks, A and C. By altering the pointers, program X has taken the option of forming them into a two-block string. Independently, program Y has fetched block E. Blocks B and D remain to be allocated by the executive.

To return a block for allocation, the user supplies the original block address. The Executive automatically updates the pointers in the block allocation table to include the returned element.

SBLK entries required to define the groups of reserved blocks are explained in the description of SIM, later in this section.

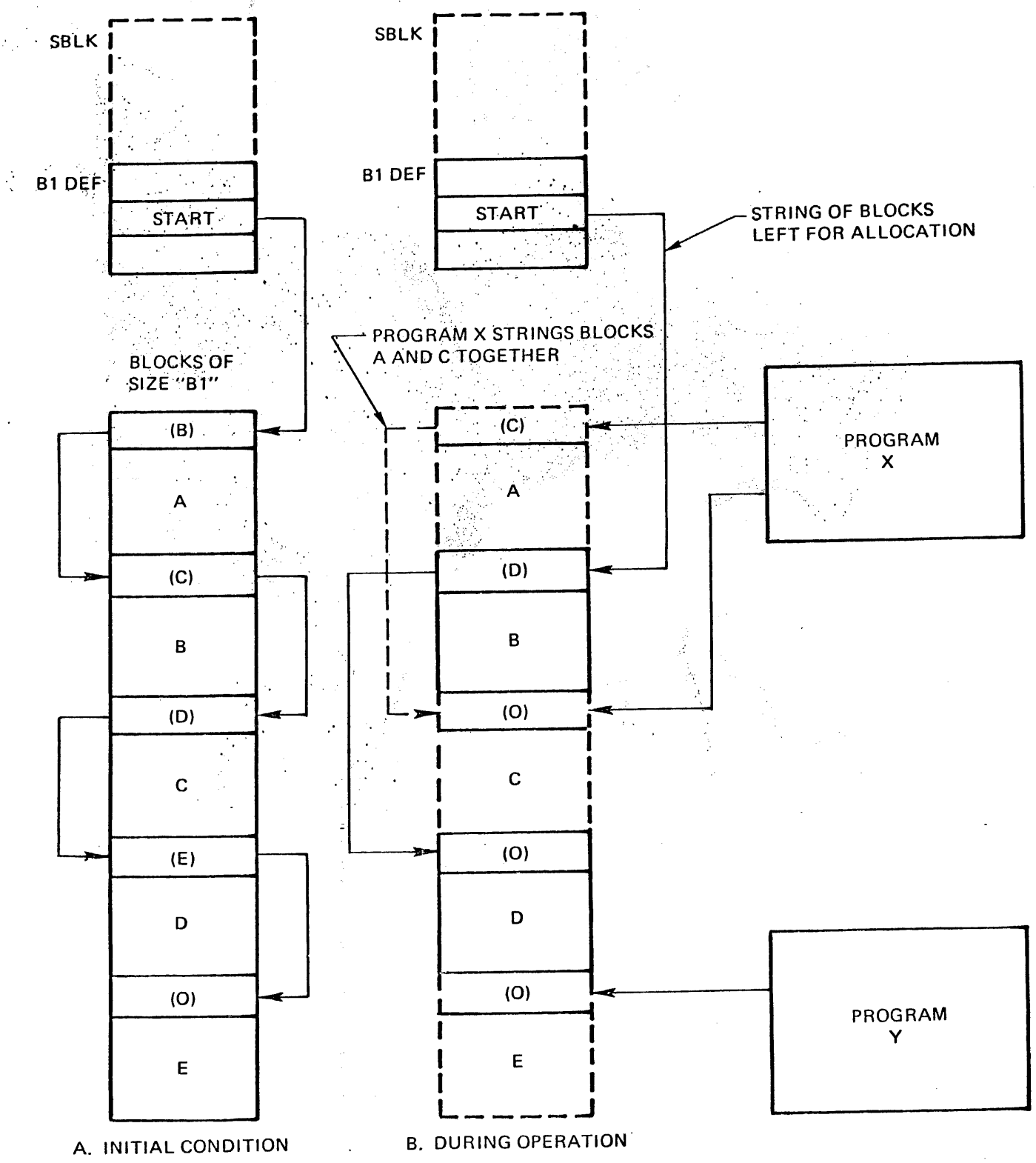


Figure 8-2. Memory Block Allocation

Task Queueing

The RTOS-C executive includes a task queueing capability which provides for automatic run-time inter-program communication and task scheduling. This capability is implemented by a new group of executive functions (Put at Top of Queue, etc.) and the SQUE tables configured into SIM.

A queue is a series of memory blocks or strings threaded together by pointers. The blocks and strings may be the same ones handled by the Memory allocation functions of the executive, or they may be generated independently by the user. They may contain data, control information, pointers, or any other information meaningful to the program which uses them.

Initially, each of the queues defined by the user in the SQUE area of SIM is empty. During execution, user programs may put blocks (or strings) at the top or the end of a given queue, and may remove blocks from the top of the queue. The executive automatically handles the threading of pointers from one block or string to the next in the queue. Executive functions for queue operations are:

PBLEND	Put Block at End of Queue
PBLTOP	Put Block at Top of Queue
PSTEND	Put String at End of Queue
PSTTOP	Put String at Top of Queue
TAKBLK	Take Block from Top of Queue

Calling sequences and functions are defined later in this section.

Figure 8-3 shows how the blocks obtained in Figure 8-2 by programs X and Y could be placed on queue Q1. Program X places the string consisting of blocks A and C on Q1 and the Executive enters a pointer to block A in the "Top of Queue" entry in the Q1 definition block. It also follows the string pointer, identifies block C as the end of the string, and points to it as the "end of queue" entry.

When program Y adds block E to the end of the queue, the executive links block C to block E and uses a block E pointer as the "end of queue" entry.

A further capability of the executive's queue handler is automatic task scheduling. When each queue is defined at configuration time, the user may specify an action to be initiated automatically when the first block is placed on an initialized queue, or a queue that has already provided one 'empty' return to a 'Take Block' function. The action is specified in the definition block for each queue in the SQUE area of SIM (described later). The choices are:

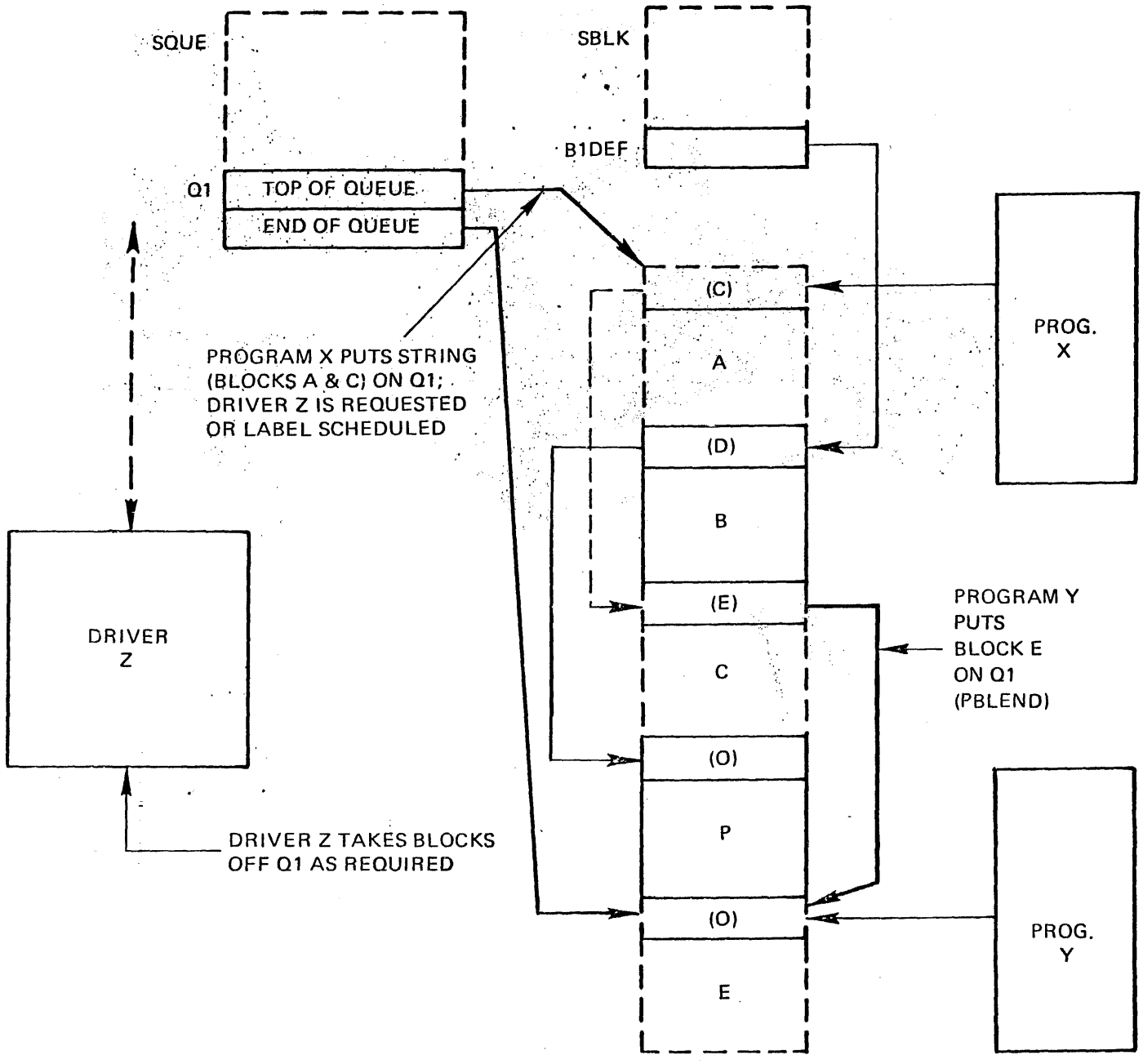


Figure 8-3. Task Queueing

Request a Program

Schedule a label in a waiting program

Execute a subroutine (usually a brief subroutine within SIM itself)

Do nothing

This automatic startup feature enables a queue to be the controlling agent for a program such as a device driver that is used in common by many other programs. Each program that wants to use the function simply places a block or string on the queue for processing. No program request is necessary. The program is requested or scheduled automatically by the function handler. Figure 8-4 shows the control flow of a typical device driver that takes advantage of the queueing facility. SIM is configured to request the driver during system initialization. Once started at label STARTUP by the scueduler, the driver connects the device to interrupt response code associated with the label INTRPT, and checks the queue to see if there are any blocks waiting to be processed. If not, the driver goes into a wait state. When the first block or string is placed on the queue, the queue handler schedules the label SCHED. When the driver is started at that location, it takes a block from the queue, starts the device action, and waits for the first interrupt. At each interrupt, the interrupt response code does any time-dependent processing, acknowledges the interrupt, and schedules the label NONINT for non-interrupt follow-up processing.

Several different external programs may add blocks to the driver's queue for processing while the first block is being handled. When the operations appropriate to one block are complete, the non-interrupt code fetches the next block from the queue and processes it; no explicit program requests are required. When the driver has processed the last block on the queue, an attempt to take another block receives a 'queue empty' return. At this point, the driver can disconnect the interrupt and either terminate or wait with the interrupt connected.

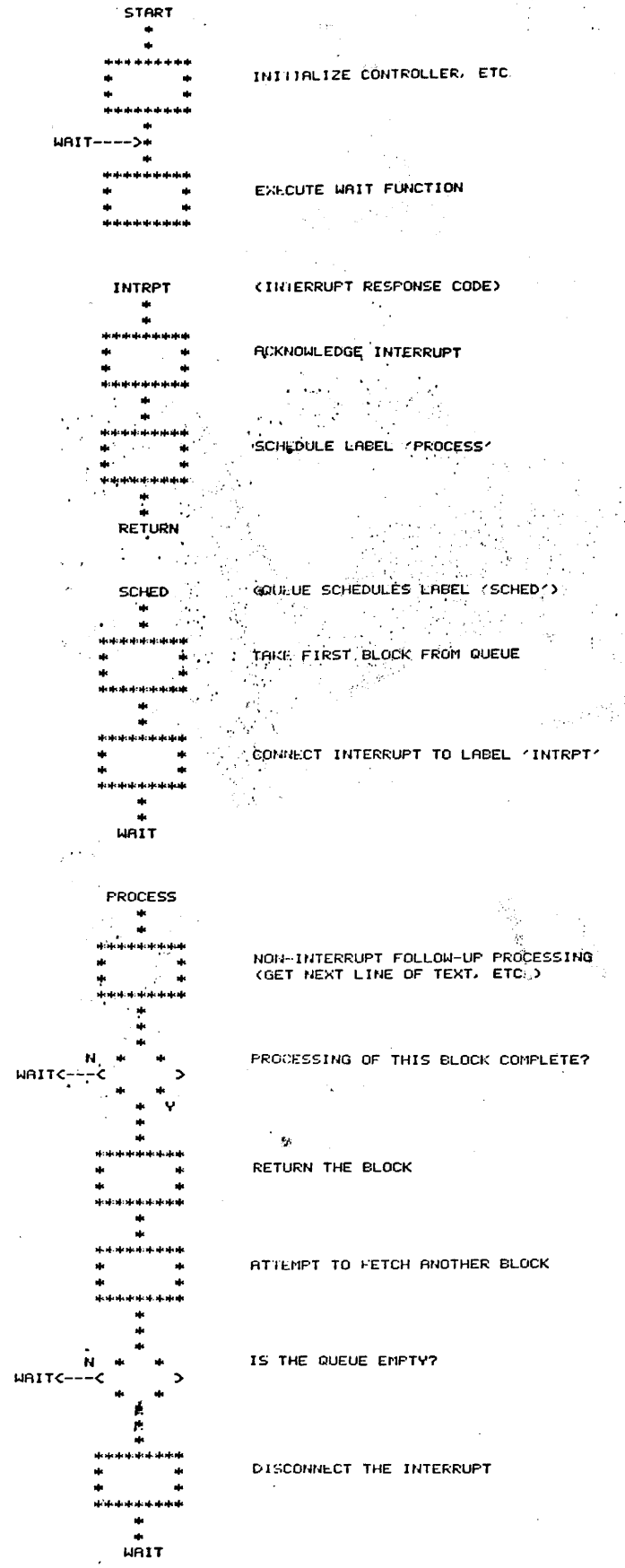


Figure 8-4. Example of Queue-driven Driver Operation

SYSTEM INFORMATION MODULE

The RTOS-C System Information Module (SIM) conforms generally to the version described in Section 3. Differences are described in following paragraphs and an example of the SIM for a disk-resident system is provided.

Header

The following SUBR statements must be added to the header:

```
        SUBR    SQUE
        SUBR    SBLK
```

SPLT Table

Each program's SPLT table entry is expanded to include a Program Size word, and the option word is modified. Each entry now has the format:

```
Label       DATA    'XY'                    (Program name)
(Label +1)  XAC     Starting Address
(      +2)  BSZ     1                       (Reserved for status word)
(      +3)  OCT     Option Word
(      +4)  OCT     Coordination Word
(      +5)  OCT     Communication Word      (Optional)
(      +6)  OCT     Program Size
(      +7)  OCT     Mass Storage Location
```

The program size word is required only for RTOS-C disk resident programs, and the mass storage location word is required only if programs are saved in the RTOS-B disk format.

Bits 2 through 4 of the option word now control coordinate three zones of dedicated sector 0, as shown in Figure 8-5. The starting segment field now refers to a 128-word segment boundary in memory. The SPLT entry for program DO must have the following form:

```
DOS     DATA    C'DO'
        OCT     0014        (XX is first sector occupied by DOS)
        BSZ     1           (Status Word)
        OCT     3403        (Coordination, Communication)
        OCT     0
        OCT     113         (FIFO)
        OCT     7777        (Program Size)
```

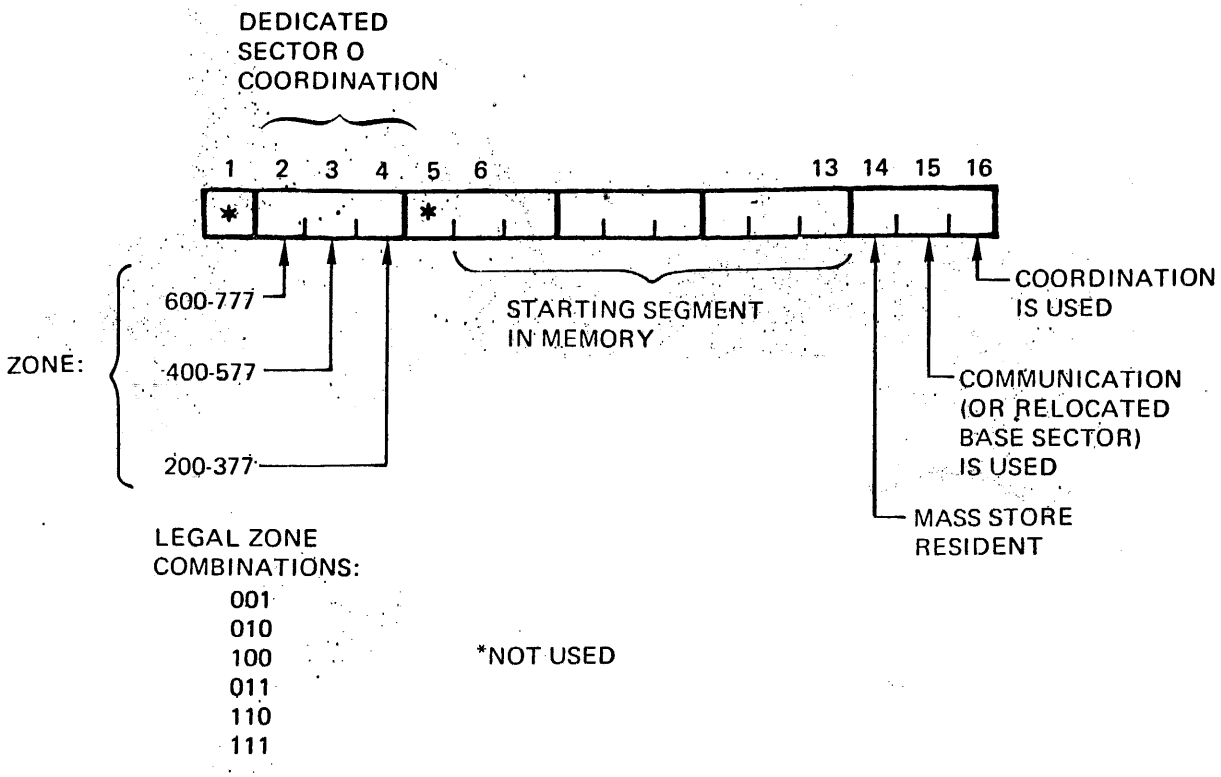


Figure 8-5. RTOS-C Option Word

SPET Table

Same as Section 3. There must be a pointer for any new entries in SPLT.

SID1 Table

Same as Section 3. Make sure there are entries for disk and magtape drivers, if used.

CLK2, CLK3, SAVM, IH20

Same as Section 3.

SPCT Table

Same as Section 3

SCUT Table

Same as Section 3.

SFET Table

If the system is to use the block and queue handling extensions to the executive functions, the following entries must appear:

```
XAC     FBL     (Fetch block)
XAC     CFBL    (Conditional fetch block)
XAC     RBL     (Return block)
XAC     RSTR    (Return string)
XAC     PBEQ    (Put block at end of queue)
XAC     PBTQ    (Put block at top of queue)
XAC     PSEQ    (Put string at end of queue)
XAC     PSTQ    (Put string at top of queue)
XAC     TKOQ    (Take block off queue)
XAC     CHCK    (Check block count)
```

SINT Entries

Same as Section 3.

SDCT Table

Same as Section 3.

SSPT, SPFP, SEXA

If base sector relocation is used, SSPT should be a DAC to the PAGINGBUFFER at the end of SIM. Otherwise, these entries are the same as Section 3.

SDOS Entries

Entries starting at the label SDOS are used when the DOS file manager program is memory-resident. If all files are to be handled by DOS sequential access, the random access entry should be DEC 0 and no random access file description blocks or mapping buffers are required.

```
SDOS    DAC     DOO             (DOS SPET pointer)
        OCT     Ldev            (Logical disk device containing
                                 RTOS UFD)
        DATA    C'Ufdnam'       (UFD for RTOS programs)
        DEC     n               (No. of random access files)

*  DEFINITION OF FIRST RANDOM ACCESS FILE
        OCT     Ldev            (Logical disk device containing
                                 first record of file)
        DATA    C'Ufdnam'       (UFD of first random access
                                 file)
        DATA    C'Filnam'       (Filename of first random
                                 access file)
        DAC     Buffer 1        (Pointer to mapping buffer for
                                 first random access file)
```

'Ldev' is a logical disk unit number defined in the SDCT area of SIM.

```
*  DEFINITION OF SECOND RANDOM ACCESS FILE
        (Same form as above)

*  DEFINITION OF Nth RANDOM ACCESS FILE
        (Same form as above)

*  MAPPING BUFFERS
Buffer 1 BSZ    Size            (Mapping buffer for random
                                 access file 1)
Buffer 2 BSZ    Size
.
.
.
Buffer n BSZ    Size            (Mapping buffer for last
                                 random access file)
```

Each 'Ufdnam' and 'Filnam' must be a 6-character ASCII DOS name (short names must be padded to 6 characters with space characters). Each 'Ldev' is a logical DOS disk device or surface. The 'Size' of each buffer is 1/3 the number of 128-word segments in the file. (Round up remainder.)

Queue Definition Tables

Entries starting at the label SQUE define the number of queue definition blocks used in the system. If queues are not used, SQUE should be a DEC 0 and none of the subsequent entries are required.

```
SQUE    DEC     n               (Number of queue definition
                                 blocks)
        DAC     Queue 1         (Pointer to first queue
                                 definition block)
        DAC     Queue 2         (Pointer to second queue
                                 definition block)
        .
        .
        DAC     Queue n         (Pointer to the n'th block)
```

```
*  FIRST QUEUE DEFINITION BLOCK

Queue 1   DEC    -1          (Reserved for start queue pointer)
          OCT    0           (Reserved for end queue pointer)
          BSZ    1           (Not used)
          OCT    Function    (See text)
          DATA   Ident
            or
          DAC

*  SECOND QUEUE DEFINITION BLOCK

Queue 2   (Same form as above)

* N'TH QUEUE DEFINITION BLOCK

Queue n   (Same form as above)
```

The 'Function' entry is a function code. Its value determines the way the 'Ident' entry is interpreted:

'Function' Code	Meaning	Use of 'Ident'
-2	Execute a subroutine	DAC pointer to subroutine entry point
-1	Request Program	DATA C Prognam'
0	No action	OCT 0 (Not used)
Other	Schedule a Label ('Function' is the label itself)	DATA C'Prognam'

'Prognam' must be a 2-character RTOS program name.

Any subroutines to be executed may be included in SIM following the queue definition blocks. An example, SUBR, is shown in the SIM listing to follows. Subroutines should be brief, must operate in INH mode, and must not use executive functions or SVC calls.

Block Definition Tables

Entries starting at the label SBLK are used to define memory blocks to be reserved for allocation and queueing. If no blocks are required, SBLK should be a DEC 0 and none of the subsequent entries are required.

```
SBLK    DEC   n          (Number of block size
                          groups to be defined)
        DAC   Block 1    (Pointer to description
                          of size 1 blocks)
        DAC   Block 2    (Pointer to description
                          of size 2 blocks)
        .
        .
        .
        DAC   Block n    (Pointer to description of
                          size n blocks)

*DESCRIPTION OF SIZE 1 BLOCKS

Block 1 OCT   Number     (Number of blocks of this
                          size to be reserved)
        DAC   Start      (Pointer to first memory
                          location used by this group
                          of blocks)
        DEC   Size       (Block size in words,
                          including linkword at
                          beginning of block)

*DESCRIPTON OF SIZE 2 BLOCKS

        (Same form as above)

*DESCRIPTION OF SIZE n BLOCKS

        (Same form as above)
```

Paging Buffer Entry

If base sector relocation is used, the SIM table must include the following entry:

```
PAGINGBUFFER    BSZ     128
```

The entry must start on a sector boundary or a sector + '400.

SIM Example

For a current SIM example, see the revised Section 6 supplied with Addendum A.

EXECUTIVE FUNCTIONS FOR BLOCK AND QUEUE HANDLING

The following functions enable the user to remove and replace blocks and queue elements defined in the SBLK and SQUE tables of SIM. Block and queue operations can be invoked in either PMA or FORTRAN.

Parameters: Symbolic names used in many of the calling sequences are:

Symbolic Name	Meaning
Blksiz	Blocksize group number
Errtn	Error return location if function cannot be executed
Baddr	Address of first location used by block or string that is fetched or returned
Qnum	Queue number

Other names are defined where they appear. Unless otherwise specified, all parameters are integer variables.

Blocks: Blocks are the reserved memory blocks defined in the SBLK area of SIM.

Strings: A string is a group of blocks of the same size that have been strung together by the user. The first word of each block must point to the first word of the next block in the string. The last block in the string must contain 0 in its first word.

Error Codes: If a function cannot be executed, the error return is taken with an error code in the A Register. In FORTRAN, the error code can be tested by the CALL GETA function.

Fetch Block (Function 9)

This function fetches the address (first location) of a memory block of a specified size.

The PMA format is:

```
SVC
DEC     9
OCT     Blksiz
DAC     Errtn
```

When the function takes the normal return, the address of the block is present in the A register.

The FORTRAN format is:

```
CALL FBLCK (Blksiz, Baddr, Errtn)
```

When the normal return is taken, the address of the block is present in 'Baddr'.

Error Code	Condition
0	Invalid blocksize code specified
-	No blocks available

Conditional Fetch Block (Function 10)

This function fetches the address of a block, provided a specified number of blocks is available.

The PMA format is:

```
LDA     Aval
SVC
DEC     10
OCT     Blksiz
DAC     Errtn
```

If there are more than 'Aval' blocks available, the normal return is taken and the address of the block is present in the A register.

The FORTRAN format is:

```
CALL CFBLCK (Blksiz, Baddr, Aval, Errtn)
```

If there are more than 'Aval' blocks available, the normal return is taken and the address of the block is present in 'Baddrs'.

Error Code	Condition
0	Invalid blocksize code specified
-	Not enough blocks remain

Return Block (Function 11)

This function returns a block to the executive for reallocation.

The PMA format is:

```
LDA     Baddr
SVC
DEC     11
OCT     Blksiz
DAC     Errtn
```

Note that the user must load the address of the block in the A register before entering the function.

The FORTRAN format is:

```
CALL RTNBLK (Blksiz, Baddr, Errtn)
```

Error Code	Condition
0	Invalid blocksize code specified

Return String of Blocks (Function 12)

This function returns a string of blocks to the executive for reallocation.

The PMA format is:

```
LDA     Baddr
SVC
DEC     12
OCT     Blksiz
DAC     Errtn
```

Note that the user must load the address of the first block of the string into the A register before the function is entered.

The FORTRAN format is:

```
CALL RTNSTG (Blksiz, Baddr,  Errtn)
```

Error Code	Condition
0	Invalid block code specified

Put Block at End of Queue (Function 13)

This function enters one block as the last element on a specific queue.

The PMA format is:

```
LDA     Baddr
SVC
DEC     13
OCT     Qnum
DAC     Errtn
```

Before entering the function, the block address must be present in the A register.

The FORTRAN format is:

```
CALL PBLEND (Qnum, Baddr, Errtn)
```

Error Code	Condition
0	Invalid queue number specified
1	Name of search error occurs if request or label startup is attempted
2	Function cannot be accomplished

Put Block at Top of Queue (Function 14)

This function enters one block as the first element on a specific queue.

The PMA format is:

```
LDA     Baddr
SVC
DEC     14
OCT     Qnum
DAC     Errtn
```

Before entering the function, the block address must be present in the A register.

The FORTRAN format is:

```
CALL PBLTOP (Qnum, Baddr, Errtn)
```

Error Code	Condition
0	Invalid queue number specified
1	Name search error occurs if request or label startup is attempted
2	Function cannot be accomplished

Put String at End of Queue (Function 15)

This function enters one string of blocks as the last element on a specific queue.

The PMA format is:

```
LDA   Baddr
SVC
DEC   15
OCT   Qnum
DAC   Errtn
```

Before entering the function, the address of the first block in the string must be present in the A register.

The FORTRAN format is:

```
CALL PSTEND (Qnum, Baddr, Errtn)
```

Error Code	Condition
0	Invalid queue number specified
1	Name search error occurs if request or label startup is attempted
2	Function cannot be accomplished

Put String at Top of Queue (Function 16)

This function enters one string of blocks as the first element on a specific queue.

The PMA format is:

```
        LDA     Baddr
        SVC
        DEC     16
        OCT     Qnum
        DAC     Errtn
```

Before entering the function, the address of the first block in the string must be present in the A register.

The FORTRAN format is:

```
CALL PSTTOP (Qnum, Baddr, Errtn)
```

Error Code	Condition
0	Invalid queue number specified
1	Name search error occurs if request or label startup is attempted
2	Function cannot be accomplished

Take Block from Top of Queue (Function 17)

This function removes the block at the top of a specific queue.

The PMA format is:

```
        SVC
        DEC     17
        OCT     Qnum
        DAC     Errtn
```

If the normal return is taken, the block address is present in the A register.

The FORTRAN format is:

```
CALL TAKBLK (Qnum, Baddr, Errtn)
```

Error Code	Condition
0	Invalid queue number specified
-	Queue empty

Check Blockcount Function 18

This function determines the number of blocks available in a particular size category.

The PMA format is:

```
SVC
DEC   18
OCT   Blksiz
DAC   Errtn
```

If the normal return is taken, the number of blocks available in the specified 'Blksiz' group is present in the A register.

The FORTRAN format is:

```
CALL CHECKB (Blksiz, Blkcnt, Errtn)
```

On a normal return, 'Blkcnt' contains the number of available blocks.

Error Code	Condition
0	Invalid block size specified

UTILITY PROGRAMS

RTOS-C uses the same complement of on-line utilities as RTOS-A and B. Off-line utilities are not required since the same functions can be performed by DOS internal or external commands (SAVE, TAP, etc.).

Two additional off-line DOS commands (RTOSRA and FILBLK) are provided for creating random-access files off-line. (See System Building.)

See the revised Section 6 supplied with Addendum A

Pages 8-44 through
8-55 deleted

RTOS-C USER PROGRAM REQUIREMENTS

User programs to run under RTOS-C should conform to the requirements of Section 7. In addition, programs that call DOS subroutines must reserve space for a secondary header following the normal RTOS header.

Using DOS Subroutines

All the facilities of DOS (and DOSVM, in a virtual-memory PRIME processor) are available under RTOS-C. These include the FORTRAN DOS functions ATTACH, READ, WRITE, SEARCH, etc., described in the DOS manual.

The real-time FORTRAN library contains an 'interlude' call for each of the FORTRAN DOS calls. Each call enters the RTOS system function logic in the executive, which is expanded to handle the extended system functions. DOS system functions cause the DOS program to be requested. (Communication is used to pass a pointer to a secondary header.) The calling program is placed in waiting status. DOS interprets and executes the request using the real time drivers and a special call on the disk driver, if required. When the function is complete, DOS schedules the return address to the calling program as a label and returns to the scheduler. If there is an error during processing of the request, the executive schedules the alternate return (or ABRTSET return) in the calling program.

The following subroutines are implemented in a different way than the standard DOS versions:

EXIT	Has same effect as a TERMINATE executive function
RESUME	Brings SAVEd program from disk to memory, retrieves starting address from header, and schedules it as a label.

Secondary Header: DOS and the executive enter and use parameters in the secondary header to control disk access, select UFD's, and control return conditions. The format is:

```
          REL
          DEC      -1       (Normal RTOS header)
          BSZ      n
          JMP      Label    (Start of secondary header)
(Hdr  )   BSZ      1        (Program name)
(Hdr+1)   BSZ      1        (SVC 1 address during exec. functions)
(  +2)    BSZ      1        (A Register after alternate return)
(  +3)    BSZ      1        (User's abort return)
(  +4     BSZ      3        (Current UFD)
(  +5)    BSZ      1        (Beginning record address)
(  +6)    BSZ      1        (Logical disk no.)
(  +7)    BSZ      3        (Home UFD)
(  +8)    BSZ      1        (Logical disk no.)
Label     ...      ...      (Start of processing)
```

All words except 'Hdr+3' are filled in by DOS or the executive during execution of DOS subroutines called by a user program. Word 'Hdr+3' may be set to an address value by the CALL ABRTST extension to FORTRAN. Word 'Hdr+2' is made accessible to user programs by the CALL RTGETA extension.

Return Conditions (ABRTST and RTGETA Functions): The user should always specify an alternate return in DOS subroutines (such as CALL ATTACH) that provide for one. Otherwise, if the function aborts, DOS is terminated and the calling program is hung in a waiting state with no return label scheduled. The same thing can happen with subroutines (CALL SAVE, etc.) that do not provide for an alternate return. Before issuing calls to these subroutines, the user should set up an abort return using the following FORTRAN extension:

```
CALL ABRTST (Return)
```

This places the value of integer variable 'Return' in the 'Abort return' location of the calling program's secondary header. 'Return' must be ASSIGNed the value of a statement number in the calling program (or be an address constant in the form $n).

Certain DOS subroutines (CALL READ, CALL WRITE) provide return information in the A register. When an abort return is taken, the A register value is placed in the secondary header of the calling program. The FORTRAN user can access the information by using the following FORTRAN extension:

```
CALL RTGETA (Areg)
```

This transfers the A register value from the secondary header to the integer variable 'Areg'.

PMA Calls to DOS Subroutines: The DOS subroutines can be called from PMA assembly language programs as well as FORTRAN. If the subroutine requires more than one parameter the calling sequence format is:

FORTRAN	PMA
CALL SUBR (A, B, C)	CALL SUBR DAC A DAC B DAC C OCT 0 Normal return

If there is only one parameter, the format is:

FORTRAN	PMA
CALL SUBR (A)	CALL SUBR DAC A Normal return

Using DOS RESUME Function: A regular RTOS program can use a call to the DOS RESUME subroutine to bring into memory files of SAVEd executable code that have no SPLT entries and are otherwise unknown to the executive. The RESUME function schedules the starting location SAVEd with the code as a label. The requirements are:

1. The RESUMEd program must be in the UFD specified in location SDOS+2 of SIM, or else the calling program must ATTACH to another UFD. Names of RESUMEd programs are not restricted to 2 letters.

2. RESUMEd files may be overlaid into any available memory locations, but the user should take care not to overlay other active programs, portions of the Executive, or SIM.

3. The header of the original program (the one with an SPLT entry) must remain intact in memory.

4. RESUMEd files may use RTOS drivers, connect interrupts, connect the clock, and do any other executive functions. However, the name of the original program (the one with the header remaining in memory) must be specified in such calls, rather than the name of the RESUMEd file. (RESUMEd files should be considered as extensions to the code of the calling program.) Labels associated with interrupts or clock function may be in the RESUMEd or the calling program itself.

Calls to Disk Driver

See Appendix H

APPENDICES

APPENDIX A

PRIME DEVICE ADDRESS ASSIGNMENTS

Number (Octal)	Device
00	Unassigned (Polling)
01	Paper Tape Reader
02	Paper Tape Punch
03	Line Printer
04	Teletype
05	Card Reader
06	Card Punch
14	Magnetic Tape
20	S-16 Set Mask Instruction, Control Panel, Real Time Clock
22	Fixed Head Disk
24	Writeable Control Store
25	Moving Head Disk
30	BPIOC 1
31	BPIOC 2
32	BPIOC 3
40	A/D Converter Type 6000
41	Digital Input Subsystem Type 6020
43	Digital Output Subsystem Type 6040
45	D/A Converter Type 6060
54	Asynch. Multiline Cont AMLC
56	Sync. Multiline Cont SMLC
60-67*	User Devices (BPI)

Note: Device Addresses not shown above are unassigned

*Device Addresses 60-67 are reserved for user assignment.