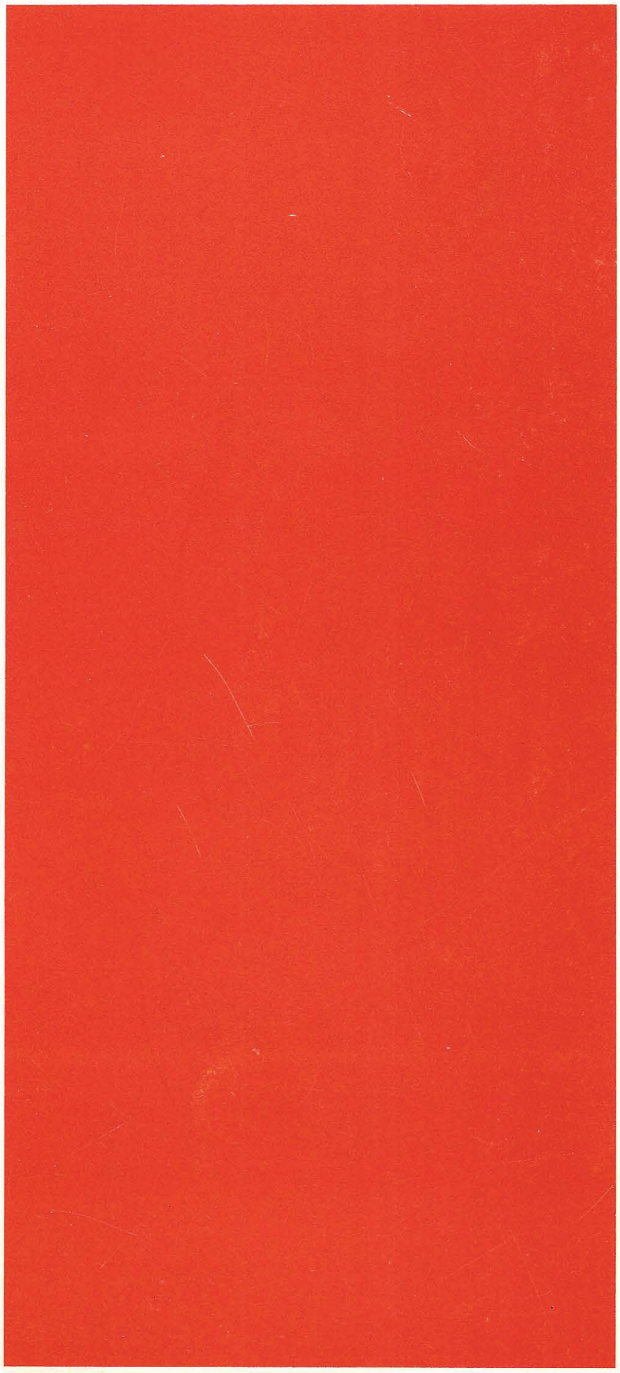
Honeywell

OP-16
USERS GUIDE

SERIES 16

SOFTWARE



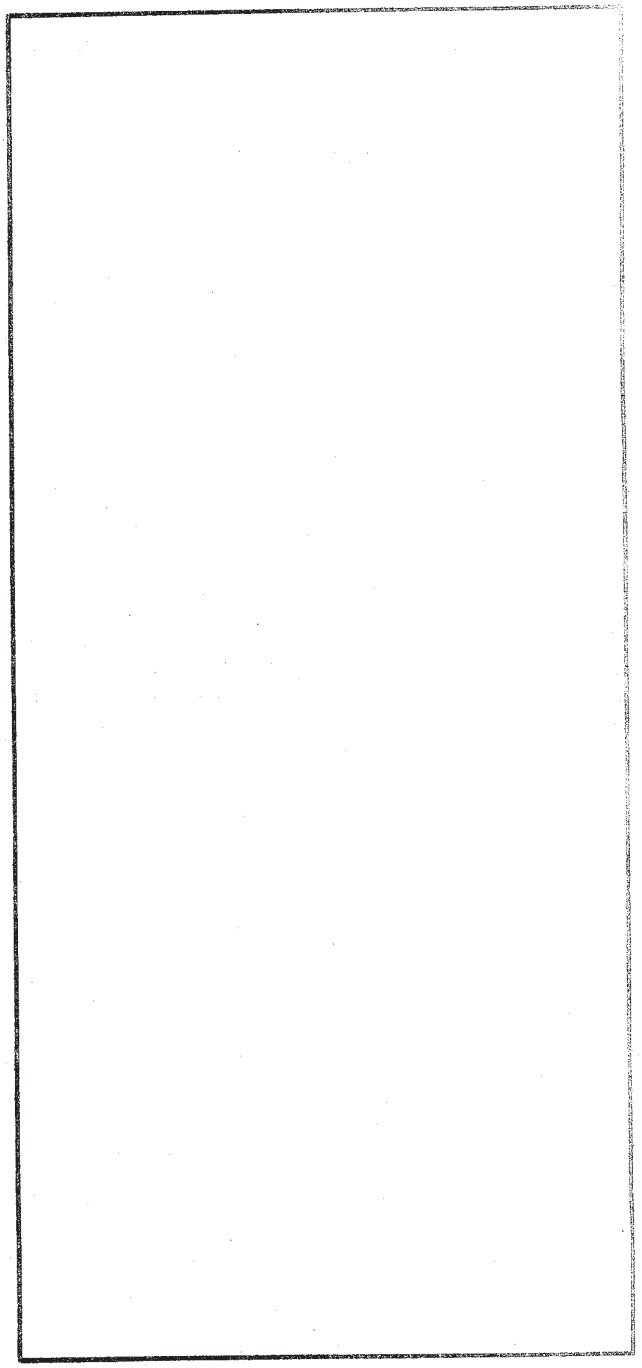
Doc. No. 41286103020C

Honeywell

OP-16
USERS GUIDE

SERIES 16

SOFTWARE



Doc. No. 41286103020C

Honeywell reserves the right, at any time and without notice, to change any information contained herein. Unless agreed in writing by Honeywell, this publication will not form part of a contract.

Originated by HCO Technical Publications Department

For further information write to:

HONEYWELL INFORMATION SYSTEMS LIMITED,
Hemel Computer Operations,
Maxted Road,
HEMEL HEMPSTEAD,
Herts.

Tel: Hemel Hempstead 2291
Telex: 82413
Cable: HONEYWELL

May 1973

ERRATUM

Page 8-18, Para 5

Delete this paragraph and insert the following:

5. After loading the Fortran programs, load the Fortran System Library tapes with the OP-16 Fortran Library tapes in the following order.

	Title	Document No	Condition
1.	CLIB	41286969-521	-
2.	DLIB H-W	41286971-521	If hardware arithmetic option is present.
	or		
	DLIB S-W	41286970-521	If hardware arithmetic option is not present.
3.	OPFRT1	70182903000	
4.	OPFRT2H	70182904000	Force load the first routine (SYSCAL) and standard load the remaining routines if re-entrant Math Subroutines are used.
5.	IRLIB H-W	41286973-521	If hardware arithmetic option is present.
	or		
	IRLIB S-W	41286972-521	If hardware arithmetic option is not present.
6.	OPFRT2S	70182905000	If re-entrant Math Subroutines are not used.
7.	OPFRT3	70182906000	-
8.	ULIB	41286974521	-
9.	ACI	70180717321	-

REVISION HISTORY

UK Doc. No.	Rev.	Date	Derived From US Doc. No.	Rev.
41286103020	C	July 1972	70130072404	E

PREFACE

This manual describes the Op-16 Operating System and the RTX-16 Executive for the Models 316 and 516. Separate manuals describe Op-16 options.

Section I provides a brief description of Op-16 features, components, and configuration requirements, and RTX-16 functions and construction. The remainder of this manual is organized as follows:

Section II: describes the RTX-16 Executive, its modules, and its functions.

Section III: describes the System Function calls and how to use them from a user program.

Section IV: describes the components of the Configuration Module and gives rules for its assembly.

Section V: describes the utility programs and device drivers.

Section VI: gives rules for writing Op-16 user programs.

Section VII: describes the procedure used to generate an Op-16 system.

Section VIII: describes additional options available to the Op-16 user.

Appendices: list the memory segments, and Executive interrupt reference numbers and special parameters.

The following conventions are followed in this manual.

The word "sector" refers to a 512-word block; the Model 316 and 516 memories are organized into sectors. The word "segment" refers to a 128-word block; the storage of secondary storage devices operating under RTX-16 is divided into segments.

Each octal number is preceded by an apostrophe or followed by the word "octal", except where there can be no ambiguity. For example, '74 = 74 (octal) = 74_8 = 60_{10}.

Diamond brackets [< >] are used throughout this manual to enclose items of a variable nature. The prefix I is used to indicate that decimal numbers are required; 'I to indicate that octal numbers are required; and A to indicate that alphabetic or ASCII information is required. Thus, <'16> means six octal digits, <A4> means four ASCII characters, and <pointer to program name in ASCII> means that the symbolic name of a location containing the program in ASCII is to be inserted. Two examples of the use of diamond brackets follow.

1. General case: DAC <A4>
   Specific example: DAC LKL3

2. General case: LDA <pointer to buffer>
   Specific example: LDA LKBP

.
.
.

LKBP DAC BUFF

Round brackets [()] are used to mean "the contents of the indicated register". For example, (A) = '333 means that the A register contains octal 333.

When a user response on the ASR is to be followed by a carriage return, the symbol c/r is used.

The reader is assumed to have a familiarity with programming in Series 16 assembly language and to have read the 316/516 Programmers' Reference Manual, Doc. No. 42400343401.

For systems using the executive module EXEC-A (Doc. No. 70181463000) Rev. H onwards, the following significant changes, that have been made to OP-16, apply.

Clock Frequency

The executive module is designed to operate with a 60Hz mains supply which gives a real time clock basic hardware interval of 16.7 ms. When using the system with a 50Hz real time clock, the following changes must be made to the executive. These changes can be made before the system is started using the computers control panel, or the 'replace core' function of the off-line utility program, or the changes can be made when the system is running using the 'replace core' function of the on-line utility program.

Change locations:

CLK2 to DEC 10 ('12)

CLK3 to DEC -5 ('177773)

XMIL to DEC 9 ('11)

These changes will give a system time interval of 100 ms.

NOTE: When changing the system time interval, XMIL must always be set to one less than the number of system units in one second (i.e. CLK2-1).

CONTENTS

		Page
Section I	Introduction	1-1
	OP-16 Operating System	1-1
	Features	1-1
	Components	1-1
	Requirements	1-2
	Hardware	1-2
	Software	1-2
	RTX-16 Executive	1-3
	Functions	1-3
	Construction	1-3
Section II	RTX-16 Executive	2-1
	Basic Executive	2-1
	Configuration Module Executive Tables	2-1
	Interrupt Handler	2-2
	Execution Priority	2-3
	Interrupt Connection	2-3
	Interrupt Response Code	2-4
	Program Scheduler	2-5
	Program Status	2-5
	Scheduled	2-5
	Running	2-5
	Waiting	2-6
	Inactive	2-6
	Program Requests	2-7
	Labels	2-7
	Real-Time Clock	2-8
	Options Available to OP-16 Programs	2-8
	Coordination Option	2-9
	Communication Option	2-10
	FIFO Routine	2-11
	Program Residency Option	2-11
	Error Print Program	2-12
	Error Message Format	2-12
	Calling Error Print Program	2-13
	System Loader (SYSLOAD)	3-1
Section III	System Function Calls	3-1
	Function 1 - Request Program	3-1
	Error Return	3-1
	Examples	3-2
	Function 2 - Schedule Label	3-3
	Function 3 - Connect Clock	3-4
	Error Return	3-4
	Examples	3-4
	Function 4 - Disconnect Clock	3-6
	Function 5 - Connect Interrupt	3-6
	Error Return	3-6
	Example	3-6
	Function 6 - Disconnect Interrupt	3-7
	Function 7 - Terminate	

		Page
Section III (cont)	Function 8 - Wait	3-7
	Compound Functions	3-7
	Examples	3-7
	Writing New System Functions	3-8
	Examples of System Functions	3-8
Section IV	Configuration Module	4-1
	XCOM Header	4-1
	XPLT - Executive Program List Table	4-2
	XPET - Executive Program Entry Table	4-5
	XIDT - Executive Interrupt Definition Table	4-5
	XIDT	4-6
	XID1	4-8
	XID2	4-8
	XPCT - Executive Program Communication Table	4-9
	XCUT - Executive Clock User's Table	4-10
	XIVT - Executive Interrupted Variables Table	4-10
	XLPT - Executive Label Parameter Table	4-11
	XFET - Executive Function Entry Table	4-11
	XINT - Executive Initialization Location	4-12
	XDCT - Executive Device Configuration Table	4-12
	XSPT - Executive Special Parameters Table	4-13
	XCOM Size Estimation	4-13
	Configuring Sample System	4-14
	Core Map	4-14
	Mass-Store Layout	4-14
	Configuration Module	4-14
Section V	RTX-16 Utility Programs	5-1
	Utility Programs	5-1
	Overview	5-1
	Preconfigured Special-Purpose Utility Programs	5-2
	Functions	5-2
	Core Requirements	5-3
	Device Drivers	5-3
Section VI	Writing a Program	6-1
	Program Header	6-1
	Program Name	6-1
	Interrupt Response Code	6-1
	Writing Program with Interrupt Response Code	6-3
	Passing Instructions to Drivers	6-3
	Bookkeeping in Drivers	6-3
	Use of Error Print Program	6-3
	Sample Program without Interrupt Response Code	6-3
	Sample Program with Interrupt Response Code	6-7
Section VII	System Building	7-1
	Layout	7-1
	Building Core-Only System	7-3
	Building RTX-16 Executive	7-3
	Building Programs for RTX-16	7-3
	Building Programs in Sectors 4 through 7	7-3
	Building Programs in Sectors 10 and Above	7-7
	Building Core Mass-Storage System	7-7

		Page
Section VII (cont)	Building RTX-16 Executive	7-9
	Building Programs for RTX-16	7-9
	Building Programs in Sectors 6 through 12	7-9
	Building Programs in Sectors 13 and Above	7-10
	System Initialization	7-10
Section VIII	Special Capabilities of RTX-16	8-1
	Relocated Base Sector	8-1
	Programs Using Sector Zero	8-1
	Programs Using Relocated Base Sector	8-1
	Writing Special Queueing Subroutine	8-2
	Writing New System Functions	8-2
	Control Interfaces	8-2
	Entry from User Program to Function Handler	8-2
	Entry to System Function from Function Handler	8-3
	Return from System Function to Function Handler	8-3
	Return to User Program from Function Handler	8-3
	Return to Scheduler from Function Handler	8-3
	Data Interfaces	8-3
	Entry from User Program to Function Handler	8-3
	Entry to Function from Function Handler	8-4
	Exit from Function to Function Handler	8-4
	Exit Back to User Program from Function Handler	8-4
	Exit Back to Scheduler from Function Handler	8-4
	General Rules	8-4
	User Initialization Routines	8-5
	Description	8-5
	Programming Notes	8-6
	Configuration Module	8-6
	Initialization Control Subroutine	8-6
	Example 1	8-6
	Example 2	8-8
	Adjusting Clock Resolution	8-8
	16.7-ms Clock Resolution	8-8
	40-ms Clock Resolution (Model 316 Only)	8-8
	Fortran Capability	8-8
	Compiler Configuration	8-8
	Programming Rules	8-9
	OP-16 Statements	8-9
	Header	8-9
	Request	8-10
	Schedule	8-10
	Connect Clock	8-11
	Disconnect Clock	8-12
	Connect Interrupt	8-13
	Disconnect Interrupt	8-13
	Terminate	8-13
	Wait	8-14
	Print Error	8-14
	Interrupt Block	8-14
	Interrupt Return	8-15
	In-Line Assembly Code	8-15
	Octal Constants	8-15
	Data Statement Enhancements	8-15
	OP-16 Fortran Package	8-16

		Page
Section VIII (cont)	Overview	8-16
	OP-16 Fortran Read/Write Statement Processor (RWSP)	8-16
	Re-entrant Math Subroutines (RMS)	8-17
	OP-16 Fortran Library Extensions (FLE)	8-17
	OP-16 Fortran Package Loading Procedures	8-18
	OP-16 I/O Editor Error Messages	8-19
	Compiler Library Generation Facility	8-19
	Register Load	8-19
	Register Store	8-20
	Register Test	8-20
	Fortran Library Additions	8-20
	Function LOC (arg)	8-20
	Function IFETCH (arg)	8-20
	Subroutine ISTORE (arg1, arg2)	8-20
	New Error Diagnostics	8-21
	Extended Memory	8-21
	Elimination of Three System Routines	8-21
	Error Print Program	8-21
	OPTRAC	8-21
	Keyboard Program	8-22
Appendix A	Segment Reference Table	A-1
Appendix B	Interrupt Reference Numbers Assigned in RTX-16	B-1

ILLUSTRATIONS

Figure 1-1.	Components of OP-16	1-2
Figure 2-1.	OP-16 System and its Parts	2-2
Figure 2-2.	RTX-16 Executive	2-3
Figure 2-3.	Sequence of Events in Response to an Interrupt	2-6
Figure 2-4.	Coordination Example	2-9
Figure 2-5.	Communication Example	2-10
Figure 2-6.	Sample Call to Error Print Program	2-12
Figure 3-1.	Examples of Request Program Executive Function	3-2
Figure 3-2.	Examples of Call to Connect Clock System Function	3-5
Figure 3-3.	Examples of Call to Connect Interrupt Executive Function	3-6
Figure 3-4.	Examples of Compound Functions	3-7
Figure 4-1.	Bit Assignment of Status Word	4-3
Figure 4-2.	Bit Assignment of Option Word	4-3
Figure 4-3.	Bit Assignments of Communication Word	4-3
Figure 4-4.	Core Map for Sample System	4-15
Figure 6-1.	Structure of Program Header	6-2
Figure 6-2.	Queueing Labels in Program Header	6-2
Figure 6-3.	Simplified Flow Diagram for Driver Programs	6-4
Figure 6-4.	Example of User Program	6-5
Figure 6-5.	Example of Driver Program	6-8
Figure 7-1.	8K Core-Only System	7-2
Figure 7-2.	12K Core/Mass-Store System	7-2
Figure 7-3.	Building Core-Only OP-16 System	7-4
Figure 7-4.	Memory Map of RTX-16 without Fortran Capability	7-5
Figure 7-5.	Memory Map of RTX-16 with Fortran Capability	7-6
Figure 7-6.	Building Core Mass-Store System	7-8

TABLES

		Page
Table 1-1.	Minimum RTX-16 Hardware	1-2
Table 2-1.	Basic Executive Tables	2-4
Table 2-2.	Real-Time Clock Resolution	2-7
Table 2-3.	Executive Error Messages	2-11
Table 3-1.	Base Frequencies for Clock Calls	3-4
Table 4-1.	Mainframe Interrupt Bits (SMK '0020)	4-9
Table 4-2.	Mass-Store Layout for Sample System	4-16
Table 4-3.	Sample Configuration Module	4-17
Table 5-1.	Utility Program Core Requirements	5-3

SECTION I
INTRODUCTION

OP-16 OPERATING SYSTEM

Op-16 is a small multiprogramming operating system complete with I/O drivers, utility and support programs, debugging aids, and on-line peripheral device test programs. It is capable of operating in a core-only or core/secondary storage environment. Use of the extended addressing mode (up to 32K) is optional (refer to Section VIII).

The Op-16 system answers the needs of users who require a small, efficient programming system to implement real-time data acquisition and control. An Op-16 system may be built around either a Model 516 (Model 1605 Computer System) or a Model 316 (Model 1603 Computer System).

Features

The Op-16 Operating System offers the following features.

- Priority scheduling
- Multiprogramming
- Centralized control of peripheral input/output operations and associated interrupt processing
- Coordination of core areas, input/output devices, and common subroutines
- Communication between user programs
- Modular organization allowing easy configuring to suit unique application requirements
- A complete set of utility, support, debugging, and peripheral device test programs
- Fortran compatible

Components

Any Op-16 system is composed of the following components (see Figure 1-1).

- RTX-16 Real-Time Executive
- Utility routines (optional)
- Debugging aids
- Real-time peripheral device drivers and test programs
- Fortran package
- Honeywell Series 16 standard support software

Requirements

HARDWARE

The minimum hardware requirements are listed in Table 1-1. Although it is feasible to operate properly-designed application programs under RTX-16 in 4K of core, the system must be built on a larger machine and transferred via a self-loading paper tape.

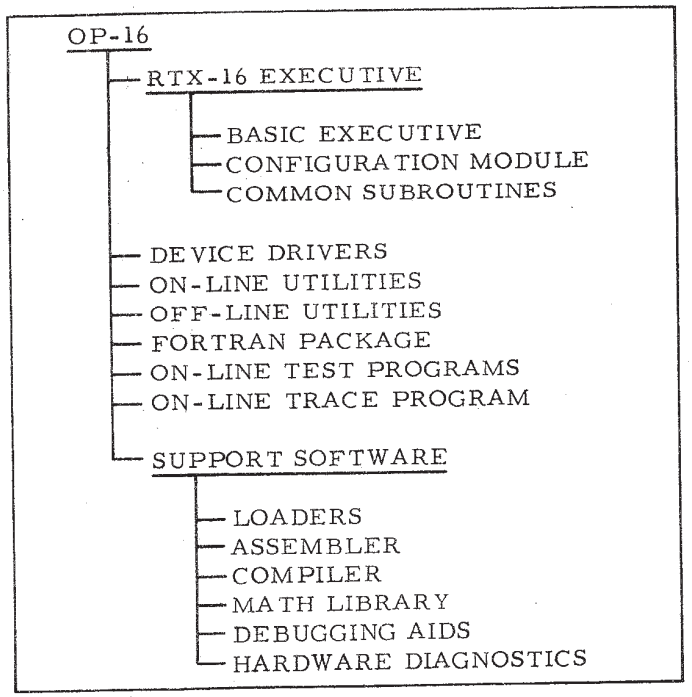


Figure 1-1. Components of Op-16

Table 1-1. Minimum RTX-16 Hardware

Component	316 Model No.	516 Model No.
Computer with 4K memory	316-01	516-01
Real-Time Clock Option	316-12	516-12
ASR-33/ASR-35 Teletypewriter	316-53/316-55	516-53/516-55

SOFTWARE

The minimum software configuration for an RTX-16 Executive system is as follows.

- Basic Executive
- Configuration Module (user-supplied)
- FIFO Communication Queueing Subroutine
- Error Print Program

In addition, RTX-16 supports an extensive set of device drivers, a p werful set of configurable on- and off-line utility programs, on-line I/O device test programs, and preconfigured system builder programs. For an up-to-date list of drivers and utilities, refer to Doc. 41286638311, Binder Table of Contents for OP-16 (BTC1OP16).

RTX-16 EXECUTIVE

Functions

The RTX-16 Executive is a collection of routines which perform the following functions.

- Execute programs according to their priority
- Keep track of the coordination requirements of programs, devices, and core storage
- Handle the interrupts which communicate external conditions to the Executive and its programs
- Keep track of the time of day in order to execute programs at certain times or after a certain delay
- Handle communications between programs and the Executive
- Handle communications between the operator and the Executive
- Detect errors in the system or individual programs
- Perform the necessary bookkeeping for a multiprogramming, multilevel system

Construction

RTX-16 is constructed of compact modules. Only the modules needed for a particular application are used. The user writes a configuration module containing all the variable information for a system.

All programs, whether normally resident in core or normally resident on secondary storage, are treated identically.

SECTION II
RTX-16 EXECUTIVE

The RTX-16 Executive is the fundamental component of every Op-16 system. It is composed of four parts: the Basic Executive, the Configuration Module, the FIFO Routine, and the Error Print Program. See Figure 2-1 for a general core layout showing the relationship of the modules.

BASIC EXECUTIVE

The Basic Executive occupies sectors 1, 2, and part of 3 in core. It contains its own constants and cross-sector links, allowing all of sector 0 (except the hardware dedicated locations) to be used by user programs. System variables needed by the Basic Executive are defined in the RTX-16 Configuration Module.

The Basic Executive consists of three major parts: the Program Scheduler, the System Function Handlers, and the Interrupt Handler (see Figure 2-2). The Program Scheduler is the most important from the point of view of the individual programs, since it is only through its use that any of them may be started up. The System Function Handlers give programs access to the Basic Executive for requesting service from it and keeping it informed. The Interrupt Handler notifies the Executive of conditions outside the computer which require its attention.

There are several other important parts of the Basic Executive. One of these is the Real-Time Clock Program, which is connected via the Interrupt Handler. Another, also connected to the Basic Executive via the Interrupt Handler, is a small routine which calls in the Executive Keyboard Program (KB) whenever a dollar sign ($) is typed on the ASR. The Error Print Program (EP) signals the operator of trouble in the hardware or the software of the system. There is also a routine in the Basic Executive (not shown in Figure 2-2) which services the Relocatable Base Sector option if the computer is so equipped.

The next several subsections describe the tables of the Configuration Module and the operation of the various portions of the Basic Executive.

CONFIGURATION MODULE EXECUTIVE TABLES

Proper understanding of the Executive Tables is necessary for an understanding of Executive operation. These tables are located in the Configuration Module. Each has a

four-word name and a four-letter mnemonic beginning with X and ending with T. Since they contain information generated by the user, a detailed description is deferred until later (Section IV). A brief description of each table in the module is given in Table 2-1.

INTERRUPT HANDLER

Whenever an interrupt occurs, the Interrupt Handler determines the source of the interrupt and jumps to the user's interrupt response code for that interrupt. This is accomplished through the Executive Interrupt Definition Table. The user's interrupt response code must be very brief. Control then returns to the Interrupt Handler, which optionally schedules a label in the user's program and returns control to the Scheduler. Further details follow.

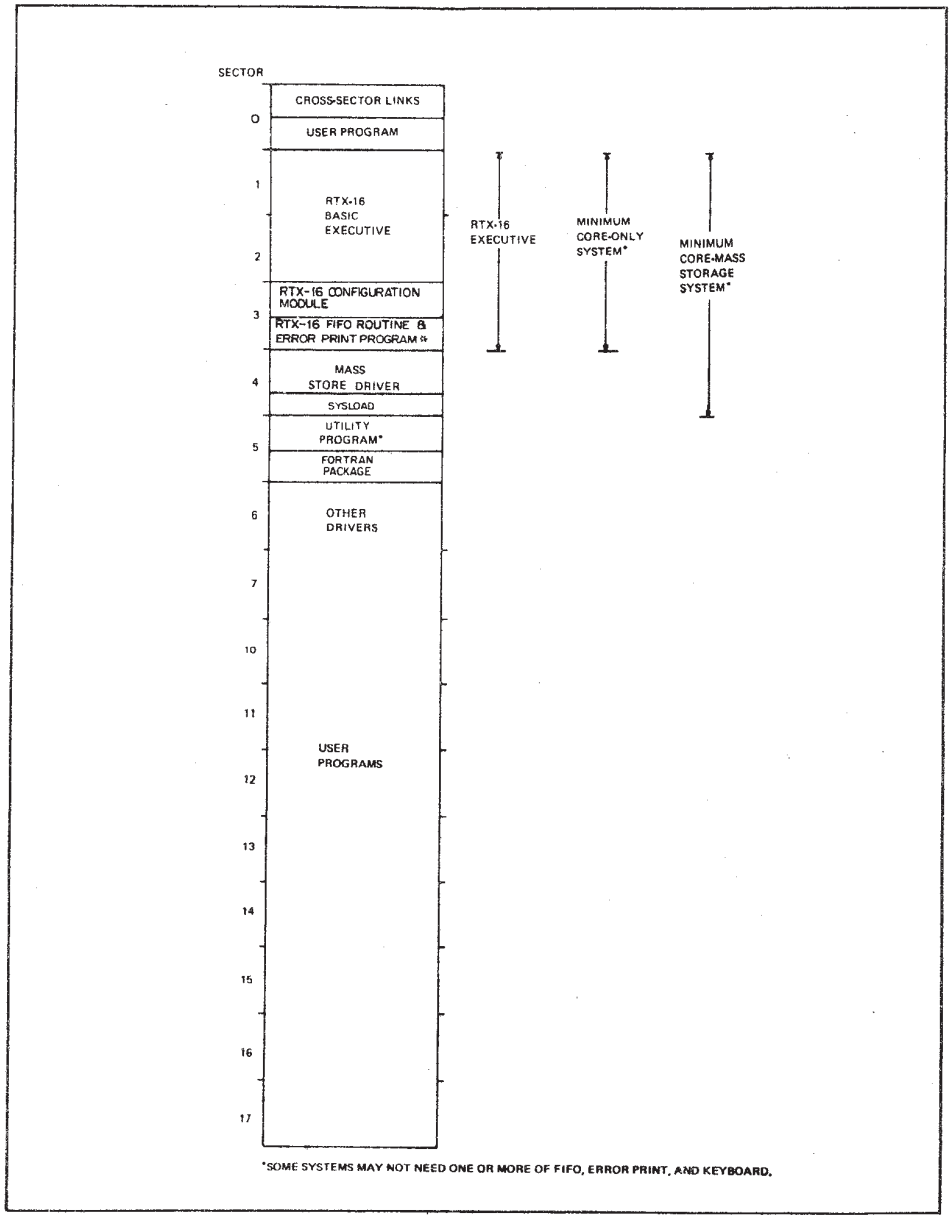


Figure 2-1. OP-16 System and its Parts

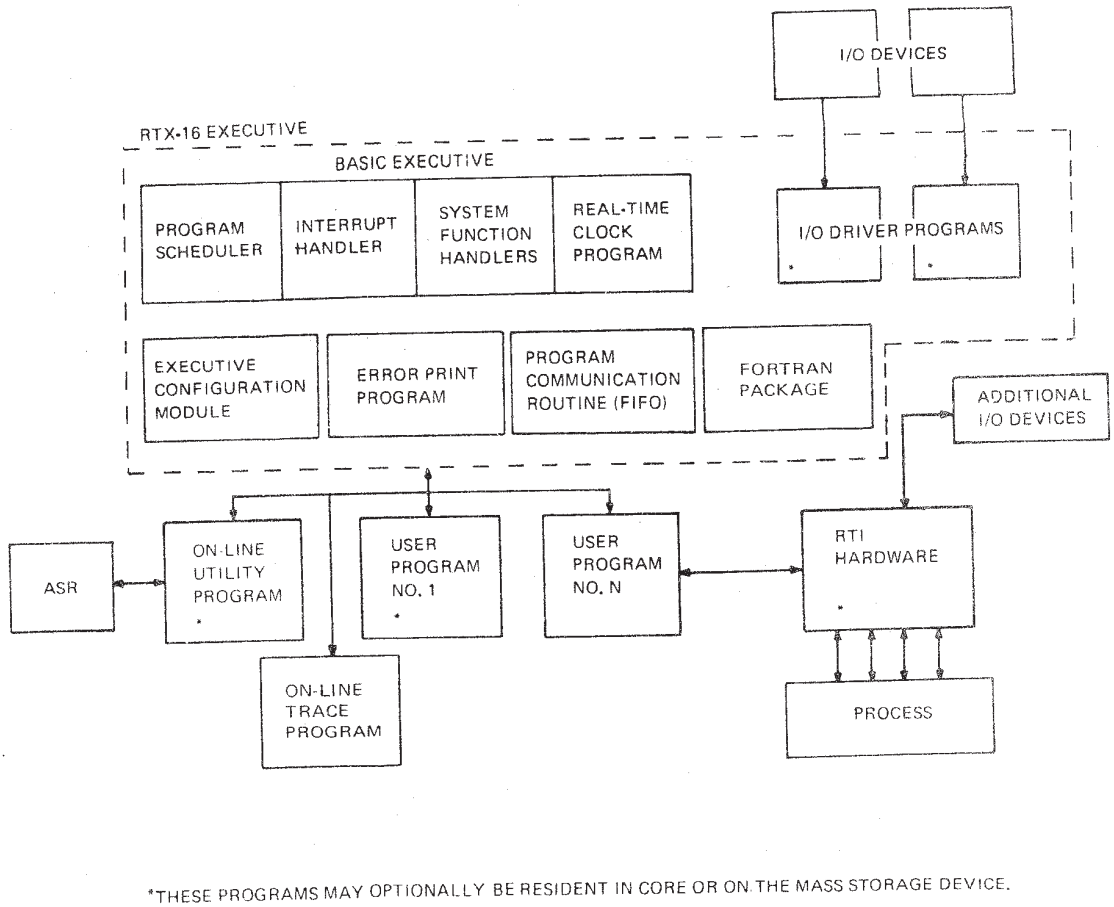


Figure 2-2. RTX-16 Executive

Execution Priority

Executable code is divided into two categories, as follows.

1. Interrupt Code – defined as the code executed as the response to an interrupt. The entry point of interrupt code is defined by the Connect Interrupt System function (see Executive System functions in Section III).
2. Noninterrupt Code – defined as all other executable code under the priority control of the Scheduler.

Interrupt code is always executed before noninterrupt code; that is, if an interrupt has occurred, noninterrupt code will be suspended until all interrupts have been serviced. If two interrupts occur simultaneously, the interrupt tested higher in XIDT is serviced first.

Table 2-1. Basic Executive Tables

Mnemonic	Name	Description
XPLT	Executive Program List Table	Contains program name, address, size, status, and option information.
XPET	Executive Program Entry Table	Defines priority of programs described in XPLT.
XIDT	Executive Interrupt Definition Table	Contains all information necessary for identifying and servicing interrupts.
XPCT	Executive Program Communication Table	Table of buffers for parameters being passed between programs.
XCUT	Executive Clock User's Table	Contains information on each program connected to clock at a given instant.
XIVT	Executive Interrupted Variables Table	Store key variables for each interrupted program.
XLPT	Executive Label Parameter Table	Temporarily stores labels scheduled by interrupt response code.
XFET	Executive Functions Entry Table	Defines number of System Functions in a system and their location.
XDCT	Executive Device Configuration Table	Contains information needed by peripheral device drivers.
XSPT	Executive Special Parameters Table	Contains additional information needed by Executive.

The priority of noninterrupt code is determined by the position of the program's pointer in XPET; that is, if two programs in XPLT may be started, the program with the higher pointer in XPET will be started first, provided that core areas and devices needed are available.

Interrupt Connection

An interrupt is connected when it has been enabled by means of its mask bit and a pointer to a routine (the interrupt response address) has been associated with it in the Executive Interrupt Definition Table. This is accomplished by the Connect Interrupt System Function. Disconnected interrupts are masked off.

Interrupts are connected and disconnected on line under program control. This allows more than one program to use the same device at different times (for example, the ASR, which is used by the Error Print Program and the ASR Driver).

Interrupt Response Code

This is the portion of code that actually handles the interrupt. It must be as short as possible since it runs with interrupts inhibited. If further work is to be done, the interrupt response code should return with the label (address) of the section that will further process

the interrupt by priority scheduling in the A register. Figure 2-3 shows the flow of control during an interrupt response. More detail on interrupt response code is given in Section VI, Writing a Program.

PROGRAM SCHEDULER

The Program Scheduler is the most basic module of the Basic Executive. Its function is to start up the highest priority program which has been requested. Whenever called, the scheduler uses the pointers in table XPET to scan the XPLT table for the first program that may be started. If no program is found, the scheduler loops through XPLT continuously until an event causes a program to be requested. The scheduler is in control of the system under the following conditions.

1. When the system is idle (that is, no user programs are active).
2. Immediately after a program has executed a Terminate Function.
3. Immediately after all interrupt code has been serviced.
4. Immediately after a program has executed a Wait Function.

The maximum time between scheduler entries is 50 ms for a 16.7-ms clock.

Program Status

Programs may be in any one of the following states.

1. Scheduled
2. Running
3. Waiting
4. Inactive

SCHEDULED

The scheduled program has been requested but not yet started. It remains scheduled until it becomes the highest priority program requiring service and all required items are available (for example, core memory, shared nonreentrant subroutines, and devices).

RUNNING

A program is running when it is in control of the CPU. It remains in this state until it does a voluntary Terminate or Wait, or is put into Wait because an interrupt has occurred.

WAITING

A waiting program is in core but has been put into the Wait state either voluntarily or because of an interrupt. If the program is waiting voluntarily, it will be restarted by a label scheduled by another program or by the occurrence of a connected interrupt. If the program has been interrupted, it will be restarted when it is again the highest priority program requesting service.

INACTIVE

An inactive program is ignored by the Executive. It may be in core or mass storage.

Program Requests

The Executive considers all requests for programs to be equal, regardless of whether the request originates from a running program or from an operator action. The program is started according to its priority. Users who wish requested programs to be executed in a fixed order should chain the requests by having the first program in the chain execute a request function for the second program in the chain, etc. Programs using the communication option will be rerun once for each request. Programs not using this option will be executed a maximum of twice for multiple requests. (See Communication Option later in this section.)

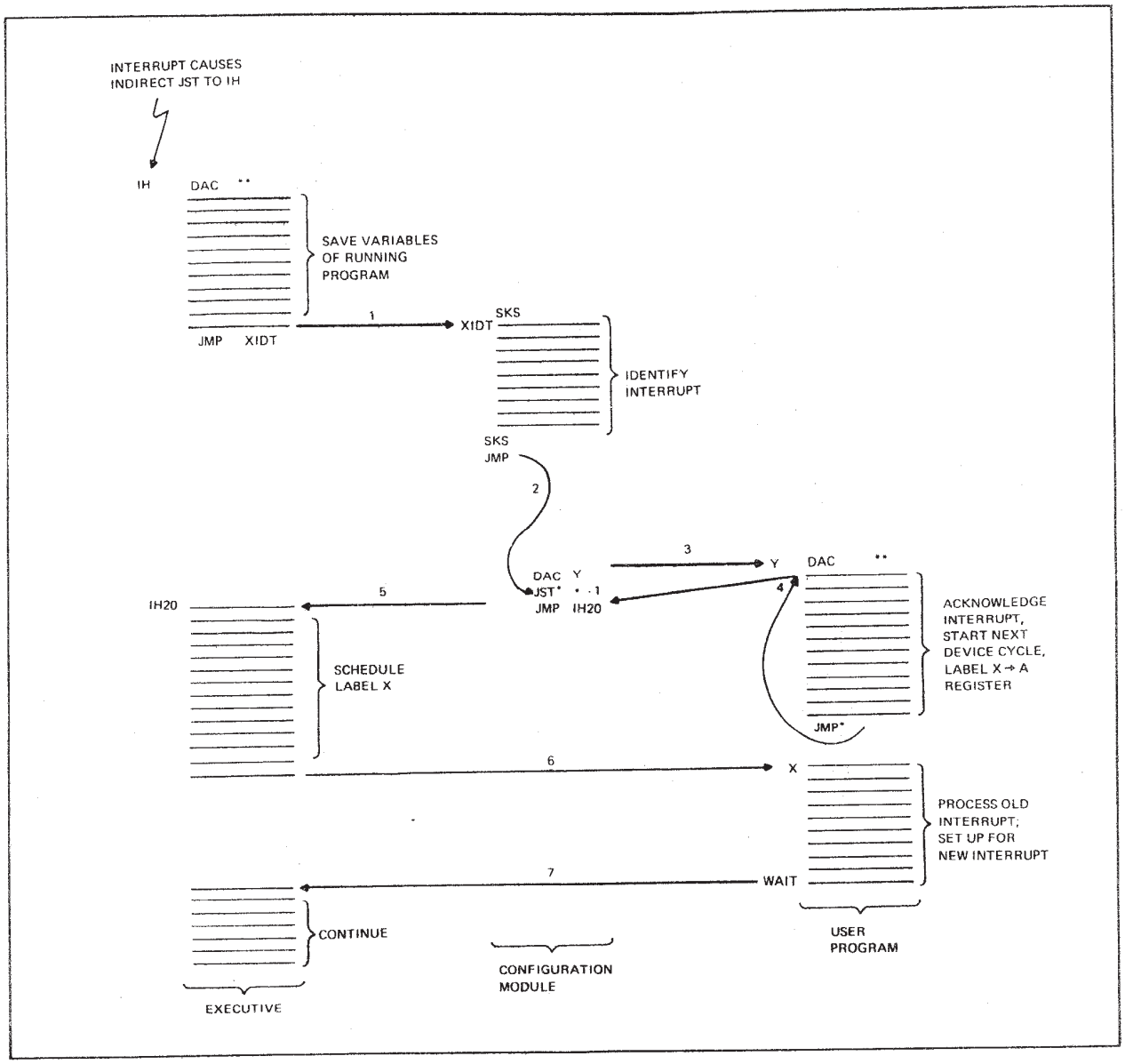


Figure 2-3. Sequence of Events in Response to an Interrupt

Labels

A label is a core address at which execution is to start. Programs use one of the system functions (described later) to instruct the Executive to schedule labels. The contents of the A register after interrupt response code has been completed are treated as a label to be scheduled unless they are zero. A program may schedule a label in any active program either running or waiting. The labels are queued in the header of the requested program; overflow of this header is an error. The label scheduled when the interrupt response code terminates must be in the same program as the interrupt response code.

If the program requested or in which a label is scheduled is of higher priority than the requesting program, the requesting program will be suspended, and the requested or scheduled program will be activated.

REAL-TIME CLOCK

The Real-Time Clock is a combination of hardware and software that allows programs to be time-related. The standard system gives a resolution of 50 ms, but this may be adjusted by the user if he desires. Table 2-2 shows how the user may vary clock resolution.

Table 2-2. Real-Time Clock Resolution

	60-Hz AC Power	50-Hz AC Power
Hardware Interval	16.7 ms[a]	20 ms[a]
Software Interval	3 hardware intervals[b]	5 hardware intervals[b]
Resolution (Model 516)		
Standard	50.0 ms	100 ms
Range	16.7 ms to 9 minutes	20 ms to 10.4 minutes
Resolution (Model 316)		
Standard	50.0 ms	100 ms
Range	5 ms to 10.4 minutes	5 ms to 10.4 minutes

[a] Fixed on the Model 516; may be varied from 5 to 20 ms on the 316.

[b] May be varied from 1 to 32,767 (see Section VIII, Special Capabilities of RTX-16).

Location '61 is dedicated to the real-time clock. At each hardware interval this location is incremented. An interrupt is generated each time the contents of location '61 become zero.

A clock interrupt calls the Real-Time Clock routine. This routine first updates the system time and then steps through the Clock User's Table, XCUT, to find which of the periodically executed programs are due to be run (see the Connect Clock system function in Section III).

The clock routine then requests execution of any program which comes due at this time. When all of XCUT has been updated, the clock interrupt is completed, and control returns to the Scheduler to start up or resume the highest priority program requiring service, possibly an interrupted program.

The system time is available for use by all programs. It may be set and displayed by the use of the Keyboard program. The locations for system time are as follows.

Location '1003	50 Millisecond units
Location '1004	1-second units (0 to 59)
Location '1005	1-minute units (0 to 59)
Location '1006	1-hour units (0 to 23)
Location '1007	1-day units

OPTIONS AVAILABLE FOR OP-16 PROGRAMS

Coordination Option

The coordination option is used for any program which can run only when certain conditions are fulfilled. In most cases the necessary condition is the availability of the space in core in which it runs.

Mass-store resident programs must use this option, and the user must make sure that the appropriate bit(s) is (are) set in the coordination word(s). It is assumed that a mass-store resident program will share an area of core with one or more other mass-store resident programs or blocks of data. Coordination is used also to ensure that peripheral devices, such as the ASR, or certain common but nonreentrant subroutines are used by only one program at a time.

The user assigns a certain bit of the master coordination word, XCCW, to each device, routine, or portion of core which must be coordinated. That bit of XCCW is set to 1 whenever a program is running that other program that has the same requirement may run. Termination of the program which had that requirement resets the bit in XCCW to 0 and allows another program requiring that item to be started. Each program using coordination has a coordination word in its entry in the Executive Program List Table, XPLT. This word has a 1 in each bit position which must be coordinated in order for the program to run.

The user must assign as many bits of XCCW as necessary when configuring the system. His assignment of a bit to a device, or portional core is completely arbitrary but must be consistent throughout the system. Figure 2-4 gives an example.

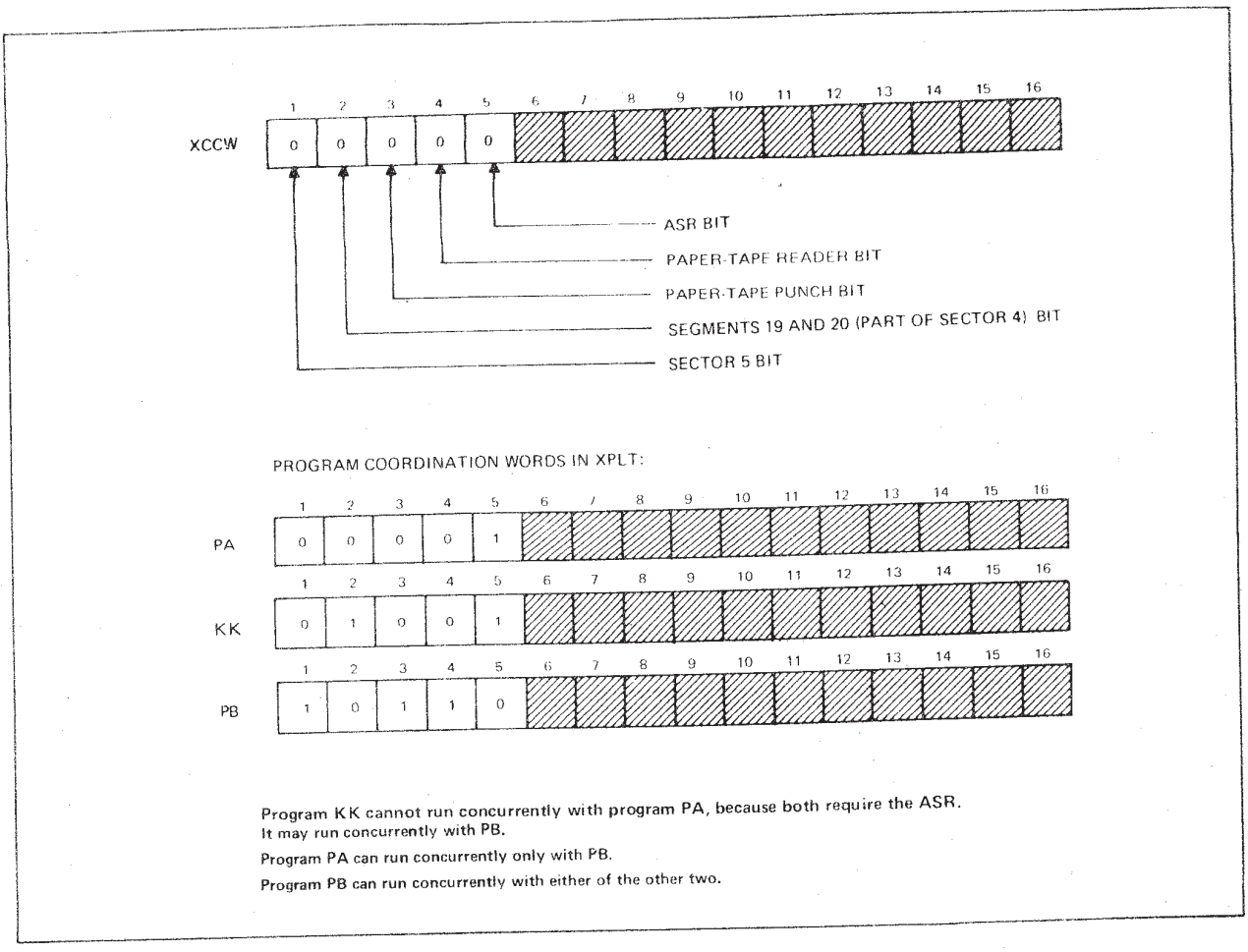


Figure 2-4. Coordination Example

Communication Option

Some programs require input parameters each time they are run. The communication option allows a program to receive any parameters it requires. If a program does not need parameters, it need not use the communication option.

When System Function 1, Request Program, is called, the Executive checks the Executive Program List Table to see whether the requested program uses communication. If it does, the requested program's communication word in XPLT tells the Executive which queueing routine and buffer to use. The Executive then passes one parameter from the calling program to the selected queueing routine. The queueing routine queues this communication parameter in the specified buffer. It then returns control to the Executive, which will return control to the calling program.

RTX-16 supplies one queueing routine, FIFO (first-in first-out). The user may write any additional queueing subroutines he desires, such as last-in first-out, or the priority of the calling program.

The next time the requested program is started up, the Executive calls the queueing routine specified in the program's communication word and asks for the latest communication parameter in its buffer. The Executive places this parameter in the program's header, and the program is started up. Figure 2-5 shows a specific example.

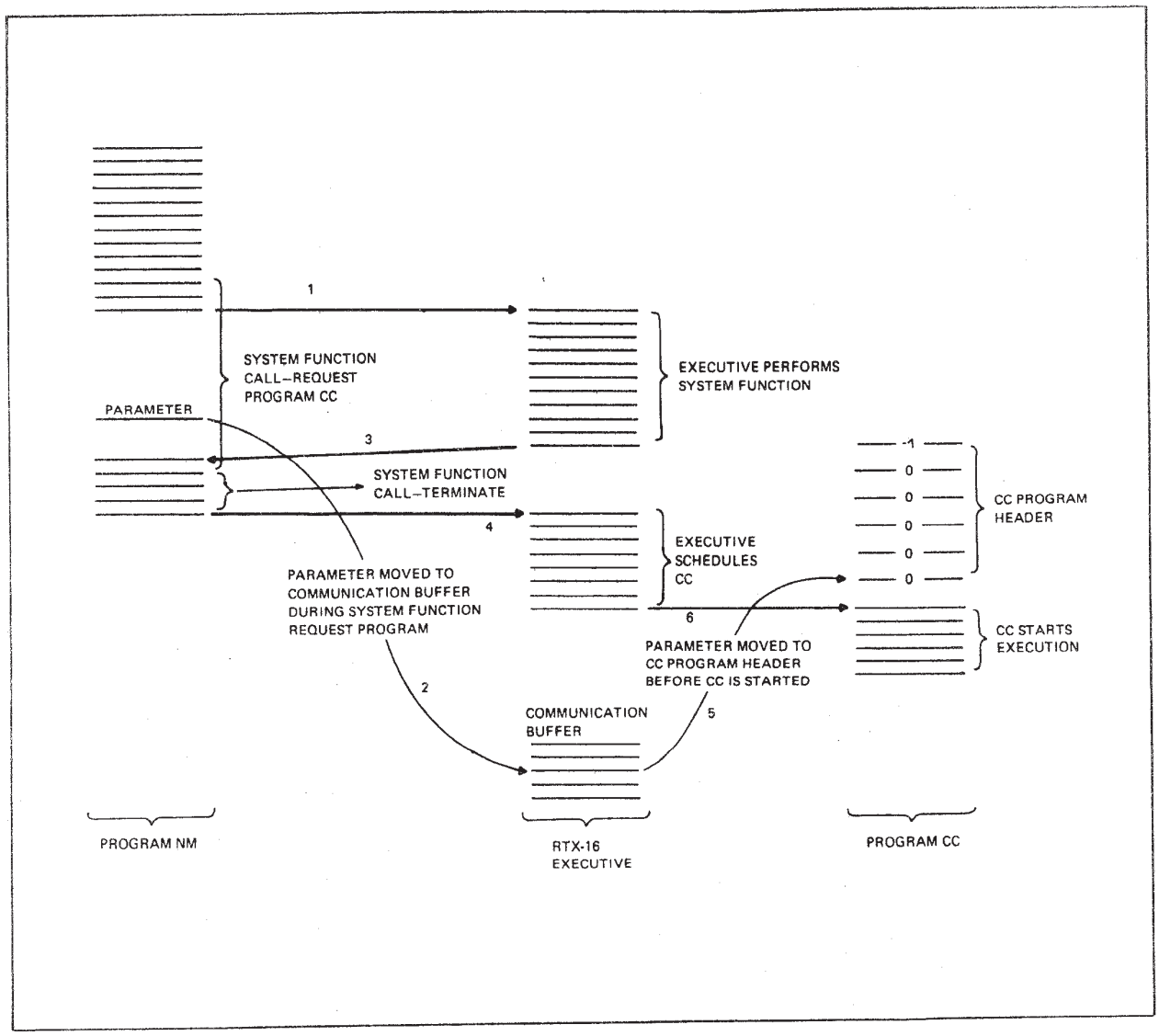


Figure 2-5. Communication Example

FIFO Routine

FIFO is the system-supplied queueing routine for the communication option. It may be omitted in systems where the communication option is never used. This routine takes care of both filling and emptying the communication buffers. Section VII includes instructions on writing special queueing routines to supplement or replace FIFO.

Program Residency Option

Programs may reside in core or on mass storage when not active. Bit 14 of the option word in the XPLT Table is 0 for core-resident programs and 1 for mass-store resident programs.

Programs that reside on the mass-store device must have the program size and starting segment indicated in the option word. See SPLT, Executive Program List Table, in Section IV, for details.

ERROR PRINT PROGRAM

RTX-16 includes an Error Print program designed to notify the computer operator by way of the ASR of any error. The errors it detects are those that may signal equipment failure or improper programming. Consequently, an error message from a properly running RTX-16 system should be of great concern to the user. Table 2-3 lists the error codes generated by the Executive. Other programs, including user programs, may use this feature. They will have their own unique error codes, which should be listed and available near the ASR.

Table 2-3. Executive Error Messages

Error Number	Program Name	Description and Result
E1	Any	A. Named program has asked for function involving another program, but requested program cannot be found in Program List Table. B. Named program has used illegal function number.
E2	Any	Program has tried to schedule label (function 2) in named program which is not active or which already has maximum number of labels scheduled.
E3	Any	Named program has tried to connect clock, but Clock User's Table (XCUT) is full.
E4	Any	Named program has tried to disconnect clock without its having been connected.
E7	Any	Named program has attempted to terminate with interrupt still connected. Program will be disabled; it can be reenabled only by operator intervention.
E11	$$	Unidentified interrupt has occurred. Interrupt is ignored.
E12	Any	Named program (interrupt-driven) has tried to schedule label from its interrupt code to its non-interrupt code, but XLPT table is full.
E13	Any	Executive has tried to schedule label (from XLPT) in named program which is not active or which already has maximum number of labels scheduled.

Error Message Format

A sample error message is printed below. Program SD has detected an error which it identifies as 333 (octal):

E333SD

Error messages are always in this exact format, preceded by a carriage return, line feed, and bell character. Leading 0's in the error number are suppressed.

Error messages are printed as soon as the Error Print program can be started by the Scheduler. Up to 10 messages can be queued. User programs which use the ASR driver and users operating the Executive Keyboard program should not tie up the ASR interrupt too long, inhibiting error messages.

Sense switch 4 may be set if error messages are not to be printed. This should be used for debugging only.

Calling Error Print Program

A pointer to the entry point of this routine is always stored in location '1016. Before a call for an error to be printed, the program name should be in the X register and the error number in the A register. The following calling sequence should be used.

```
(L  )    INH
(L+1)    LDA     <A4>     Variable specifying binary error identification Code
(L+2)    LDX     <A4>     Variable specifying ASCII program name
(L+3)    JST*    <A4>     Variable specifying '101016
(L+4)    ENB
(L+5)                     Return point
```

Figure 2-6 is an example of a call to the Error Print program; program XJ has detected the error it calls '305.

```
         INH
         LDA     ERNM      ERROR NUMBER
         LDX     PGNM      PROGRAM NAME
         JST*    ERPE      CALL ERROR PRINT PROGRAM
         ENB
         JMP     CONT      CONTINUE AT CONT AFTER ERROR PRINTED
ERNM     OCT     305       ERROR IDENTIFICATION CODE
PGNM     BCI     1,XJ      PROGRAM NAME
ERPE     DAC*    '1016     PROGRAM ENTRY POINT

RESULTING ERROR MESSAGE:

E305XJ
```

Figure 2-6. Sample Call to Error Print Program

SYSTEM LOADER (SYSLOAD)

The System Loader is used in mass-store systems to load mass-store based programs into core for execution. It is requested by the Scheduler and runs as a program under the Executive. It makes requests on the mass-store driver, as any other program does, for data transfers. By suitable entries in the Configuration Module, the Scheduler's requests for program loading may take higher, the same, or lower priority than data transfers.

Whenever a mass-store based program is to be executed, the Scheduler first checks the program's coordination, and if it can be run, the program is temporarily disabled. The Scheduler then requests SYSLOAD using the Request program subroutine (RPRO) in the Executive, and passes the XPET entry of the program to be loaded as a parameter.

When SYSLOAD runs, it first uses the XPET entry to locate the XPLT entry for the required program. It then uses the information in XPLT to build up a transfer request for the mass-store driver. Before requesting the mass-store driver, however, SYSLOAD determines the name of the driver that is to be used for program loading. This allows the user to specify exactly the priority required for program loading. This is done by having a special entry in XPLT for the mass-store driver, in addition to the normal entry used for data transfers. This entry refers to the mass-store driver by a different name, which may be chosen by the user, and is the one used by SYSLOAD. Having found the appropriate name, SYSLOAD then requests the driver and waits for the transfer to be completed. After a successful transfer into core, the program is enabled again and ready to be started up by the Scheduler.

The following two error conditions may be reported by SYSLOAD through the Error Print program:

E50XX	Program to be loaded, XX, did not use mandatory coordination option and is left disabled.
E51XX	Transfer of program XX from mass store was not successful, so program is left disabled.

Configuration of SYSLOAD and the mass-store driver is discussed and illustrated in Section IV.

SECTION III
SYSTEM FUNCTION CALLS

Programs communicate with the Executive (and, in turn, with other programs) by means of system functions. A common entry point is used by all system functions, location '1001. This location is referenced by each program by naming an external address constant, XLNK, and giving it the value '101001 in each program. A common function handler serves as the entry and exit point for each function. If the function requires a name search of the XPLT, this is handled by the common function handler.

NOTE: Registers are not saved and restored during the execution of system functions; this responsibility lies with the programmer.

FUNCTION 1 - REQUEST PROGRAM

```
(L  )   JST*   XLNK                                  Function handler entrance
(L+1)   DEC    1                                     Function number
(L+2)   BCI    1, <A2>                               Name of requested program
(L+3)   DAC    <error return point>
(L+4)   OCT    <communication parameter>
(L+5)                                                Normal return point
```

The requested program is not started up by this function but merely requested. The Scheduler will start the program as soon as possible. The communication parameter (L+4) must be present whether or not the requested program uses communication. If it does, the queueing routine will queue the contents of L+4 for eventual transfer to the header of the requested program.

Error Return

In case of error, the Error Print program prints an error message on the ASR, and the error return is made to the program. The A register contains an indication of the error: A = 1 means no such program name; (A) = 2 means that the requested communication buffer is full of parameters and, therefore, this request cannot be processed.

Examples

Two examples of the use of the Request program function are given in Figure 3-1. In the first example, program LK is requested. Since this program does not use communication, L+4 contains zero. The execution of the program resumes at L+5 if there is no error and at

ABC if there is an error. In the second example, program SM is requested. This program uses communication. A pointer to a buffer within the calling program, therefore, is inserted in word L+4. Execution of the program resumes at L+5 if there is no error and at ABD if there is an error.

A. Request for program without communication

```
JST*    XLNK    FUNCTION ENTRANCE
DEC     1       1 = REQUEST PROGRAM
BCI     1,LK    PROGRAM NAME IS LK
DAC     ABC     ERROR RETURN ADDRESS
OCT     0       NO PARAMETER
```

B. Request for program with communication

```
JST*    XLNK    FUNCTION ENTRANCE
DEC     1       1 = REQUEST PROGRAM
BCI     1,SM    PROGRAM NAME IS SM
DAC     ABD     ERROR RETURN ADDRESS
DAC     SBUF    POINTER TO PARAMETERS
```

In example A, no parameter is passed. In example B, the parameter is a pointer to location SBUF. This is the start of a buffer containing information for program SM.

Figure 3-1. Examples of Request Program Executive Function

FUNCTION 2 - SCHEDULE LABEL

```
(L  )   JST*    XLNK                    Function handler entrance
(L+1)   DEC     2                       Function number
(L+2)   BCI     1, <A2>                 Name of program in which
                                        label is to be scheduled
(L+3)   DAC     <error return point>
(L+4)   DAC     <label to be scheduled>
(L+5)                                   Normal return point
```

As shown above, the programmer knows in advance which label is to be scheduled. It will often be the case, however, that this information will be filled in by another part of the program during execution.

The major use of the Schedule Label function allows a program which services another to call the first one back after its service is complete. Drivers use Schedule Label in this way. The calling program passes its name and label to the driver by means of the communication option. When the driver has finished its write or read, it schedules the label that was passed to it. In this case, it must fill in L+2 and L+4 each time the call is made. The program in which the label is scheduled must be in the Wait or Running state.

Labels also are scheduled by the Interrupt Handler. These are queued and automatically handled by the Executive.

Error return is handled as follows. In case of error, the Error Print program prints an error message on the ASR, and the error return is taken to the program. The A register contains an indication of the error: (A) = 1 means no such program name; (A) = 2 means that the label cannot be scheduled even though the program exists. The second case happens when the program's header is full of labels or the program is not active.

FUNCTION 3 - CONNECT CLOCK

(L)	JST*	XLNK	Function handler entrance
(L+1)	DEC	3	Function number
(L+2)	BCI	1,<A2>	Program name to be connected
(L+3)	DAC	<error return point>	
(L+4)	DEC	<time until first executions>	
(L+5)	DEC	<interval between subsequent executions>	
(L+6)	DEC	<base frequency (see Table 3-1) and schedule label flag>	
(L+7)			Normal return point

This function is used to connect a program to the clock for automatic initiation by the Clock program. The Clock program initiates a connected program by means of a Request program or a Schedule Label function. This means that a program may be entered at its start address or reentered at a user-specified label. Several options are available using variations of the basic calling sequence shown above.

A program may be requested on a periodic basis by specifying the name of the program in L+2, the time until first execution in L+4, the interval between executions in L+5, and the base frequency in L+6. A label may be scheduled in a program on a periodic basis by setting bit 1 of the base frequency and loading the label into the A register prior to executing the Connect Clock function. If the interval specified in the calling sequence is set to zero, the clock program will automatically disconnect the object program from the clock when the object program falls due. The Clock program will then either request or schedule a label in the object program. This allows a program to be initiated once only from the clock, and the initiated program need not disconnect itself to prevent periodic execution.

A program may delay itself for a specified length of time by executing a Connect Clock function with a base frequency greater than 3. The interval should be set to zero. Upon expiration of the time delay, the program is disconnected and scheduled for resumption at L+7.

Table 3-1 lists the available base frequencies.

Table 3-1. Base Frequencies for Clock Calls

Base Frequency	Meaning	
0	Time until first execution is absolute time of day in minutes. Interval between executions is in minutes thereafter.	For periodic execution
1	Time until first execution is in 50-ms units. Interval between executions is in 50-ms units.	
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.	
4	Time delay until resumption of execution is in 50-ms units.	For time delays
5	Time delay until resumption of execution is in seconds.	
6	Time delay until resumption of execution is in minutes.	

Error Return

In case of error, the Error Print program prints an error message on the ASR, and the error return is taken to the program. The A register contains an indication of the error: (A) = 1 means no such program; (A) = 2 means the maximum number of clock users in XCUT has been exceeded.

Examples

Figure 3-2 shows two examples of the Connect Clock function. In the first example, program CE is to be called in 150 ms and every 450 ms after that. In case of error, execution will resume at XYZ. Otherwise, it will resume at L+7. In case of error, execution will resume immediately at XYZ. In the third example, a label is to be scheduled in program CE every 100 ms after 1 sec.

FUNCTION 4 - DISCONNECT CLOCK

```
(L  )     JST*     XLNK                              Function handler entrance
(L+1)     DEC      4                                 Function number
(L+2)     BCI      1, <A2>                           Name of program to be
                                                       disconnected
(L+3)     DAC      <error return point>
(L+4)     DEC      <base frequency (see Table 3-1)>
(L+5)                                                Normal return point
```

This function requests the executive to stop periodic execution of a program or to cancel the automatic resumption of a program in a wait state.

A. Request for program to be periodically executed

```
JST*    XLNK    FUNCTION ENTRANCE
DEC     3       3 = CONNECT CLOCK
BCI     1,CE    PROGRAM NAME IS CE
DAC     XYZ     ERROR RETURN ADDRESS
DEC     3       OFFSET (3 X 50 MS = 150 MS)
DEC     9       INTERVAL (9 X 50 MS = 450 MS)
DEC     1       1 = 50 MS INTERVALS
```

B. Request for time delay before resumption of execution

```
JST*    XLNK    FUNCTION ENTRANCE
DEC     3       3 = CONNECT CLOCK
BCI     1,CE    PROGRAM NAME IS CE
DAC     XYZ     ERROR RETURN ADDRESS
DEC     9       OFFSET  9 X 50 MS = 45 MS)
DEC     0       NO INTERVAL
DEC     4       4 = 50 MS INTERVALS
```

C. Request for label to be periodically scheduled

```
LDA     LABL    LABEL TO BE SCHEDULED IN
                  A-REGISTER
JST*    XLNK    FUNCTION ENTRANCE
DEC     3       3 = CONNECT CLOCK
BCI     1, CE   PROGRAM NAME IS CE
DAC     XYZ     ERROR RETURN ADDRESS
DEC     20      OFFSET (20 X 50 MS = 1 SEC)
DEC     2       INTERVAL (2 X 50 MS = 100 MS)
OCT     100001  MS BASE FREQUENCY AND
                  SCHEDULE LABEL BIT
```

In example A, first execution of program CE is to occur in 150 ms. Subsequent executions are to occur every 450 ms. In example B, execution of calling program is suspended for 450 ms and then resumed. In example C, label in A register will be scheduled in program CE every 100 ms starting in 1 sec. These examples assume that standard clock period of 50 ms is in use.

Figure 3-2. Examples of Call to Connect Clock System Function

Error return is handled as follows. In case of error, the Error Print program prints an error message on the ASR, and the error return is taken to the program. The A register contains an indication of the error: (A) = 1 means no such program; (A) = 2 means that the program was not connected to the clock.

NOTE: The preceding four System Function Calls have an alternate format: A program number, representing its priority, may be used in place of the two-character program name. If a number is specified, it is used directly to index down the XPET table to get to the required program entry in the XPLT table (see Section IV). Programs are numbered from 1 upwards, program 1 being the highest priority. This feature considerably reduces system overhead, but to preserve flexibility in priority placement of programs, it should be used only in exceptional cases.

FUNCTION 5 - CONNECT INTERRUPT

```
(L  )    JST*    XLNK                                    Function handler entrance
(L+1)    DEC     5                                       Function number
(L+2)    DEC     <interrupt reference number (see
                  Appendix B)>
(L+3)    DAC     <error return point>
(L+4)    DAC     <start of interrupt response code>
(L+5)                                                    Normal return point
```

On return from a successful Connect Interrupt call (at L+5), the address of the eleventh word of the interrupts' device list in XIDT is returned to the caller in the A register. This address may be the start location of configuration data for the device (see Section IV for more information on the XIDT table).

Error Return

In case of error, the Error Print program prints an error message on the ASR, and the error return is taken to the program. The A register contains a 2, signifying that the interrupt is already connected.

Example

Figure 3-3 shows an example. Interrupt number 2 (the high-speed paper-tape reader) is to be connected. In case of error, execution is to resume at EEE. The interrupt response code starts at RRR.

```
JST*    XLNK    FUNCTION ENTRANCE
DEC     5       5 = CONNECT INTERRUPT
DEC     2       2 = PAPER TAPE READER
DAC     EEE     ERROR RETURN ADDRESS
DAC     RRR     START OF INTERRUPT CODE
```

Figure 3-3. Example of Call to Connect Interrupt Executive Function

FUNCTION 6 - DISCONNECT INTERRUPT

```
(L  )    JST*    XLNK                                    Function handler entrance
(L+1)    DEC     6                                       Function number
(L+2)    DEC     <interrupt reference number (see
                  Appendix B)>
(L+3)                                                    Return point
```

This function informs the Executive that the calling program no longer wishes to respond to the named interrupt.

FUNCTION 7 – TERMINATE

```
(L  )     JST*    XLNK        Function handler entrance
(L+1)     DEC     7           Function number
```

This function enables the program to inform the executive when it has finished execution. Control is returned to the Executive with no return to the program. If the program attempts to terminate with an interrupt still connected, an error message will be printed, and the program will be disabled and left in core. Otherwise, all parameters associated with the running of the program will be reset.

FUNCTION 8 – WAIT

```
(L  )     JST*    XLNK        Function handler entrance
(L+1)     DEC     8           Function number
```

This function informs the Executive that the program wishes to suspend execution because it will be restarted at a label or in its interrupt response code.

COMPOUND FUNCTIONS

A Wait or Terminate function may be performed immediately after the execution of another function this allows compound functions, such as Request Program and Wait, Schedule Label and Terminate, and Connect Interrupt and Wait, to be specified. The compounding is achieved by regarding the function number as two bytes, the right-hand byte giving the primary function and the left-hand byte giving the secondary function.

Examples

Figure 3-4 shows two examples of compound functions.

```
                A. Request Program and Wait:

JST*      XLNK
HEX       801           8 = Wait, 1 = Request Program
  .
  .
  .


                B. Schedule Label and Terminate:

JST*      XLNK
HEX       702           7 = Terminate, 2 - Schedule Label
  .
  .
  .
```

Figure 3-4. Examples of Compound Functions

WRITING NEW SYSTEM FUNCTIONS

New system functions maybe added to a system by the user. Refer to Section VIII, Special Capabilities of RTX-16, for details.

EXAMPLES OF SYSTEM FUNCTIONS

Two sample programs in Section VI show the use of system functions.

SECTION IV
CONFIGURATION MODULE

The Configuration Module (XCOM) consists of a series of tables created by the user. In this section, each table is described, and rules are given for generating it. A sample configuration module is presented at the end of this section (Table 4-3).

XCOM HEADER

The XCOM header consists of a list of SUBR pseudo-operations which allows the loader to link the XCOM to the Executive. There is a SUBR for each major table and others to special entry points. A REL pseudo-operation should be placed at the end of the list. The format is:

```
SUBR    XPLT
SUBR    XPET
SUBR    XIDT
SUBR    XID1
SUBR    XID2
SUBR    XPCT
SUBR    XCUT
SUBR    XIVT
SUBR    XLPT
SUBR    XFET
SUBR    XDCT
SUBR    XSPT
SUBR    XPFP
SUBR    XEXA
SUBR    XINT
SUBR    ER
SUBR    KB1
SUBR    CLK2
SUBR    CLK3
SUBR    MSD
SUBR    LO1
REL
```

Systems having no mass-store device should use the following form of the SUBR MSD and SUBR LO1:

```
        SUBR  MSD, XPLT
        SUBR  LO1
LO1     EQU   0
        REL
```

In this case, the REL must follow the LO1 EQU O.

XPLT - EXECUTIVE PROGRAM LIST TABLE

This table defines all the programs in the RTX-16 system; it is central to all operations of the Executive. There must be an entry for every program in the system, including device drivers. The format of the table is:

```
XPLT  EQU   *
P1    BCI   1, P1  1st entry
        .
        .
P2    BCI   1, P2  2nd entry
        .
        .
        .
PN    BCI   1, PN  Nth entry
        .
        .
        .
PZ    OCT   0  End-of-table entry
```

The format of each entry is:

```
<Label>  BCI   1, <program name>
         OCT   <start address>
         BSZ   1                Status
         OCT   <options>
         OCT   <coordination>
         OCT   <communication>
         OCT   <secondary storage>
```

The table must start with XPLT EQU * and end with a word containing zero. An entry may be four, five, six, or seven words long, depending on the options in use. The first word of each entry must have a unique label to be used by table XPET. The order of the entries in the XPLT table is not significant, because their priority is established by XPET. Each entry contains the following information.

Word 1: Contains the unique two-character name of the program (in ASCII).

Word 2: Contains the address of the location at which the program is to be started.

Word 3: Contains all dynamic information pertinent to execution of the program (see Figure 4-1).

Word 4: Indicates which of the coordination, communication, secondary storage, and relocated base sector options are in use. If the program is to be mass-store resident, it also contains the size in segments (128-word blocks), and the starting segment number in core (see Figure 4-2 for exact bit assignments).

Word 5: This optional word is used by the Executive to ensure that this program runs only when it conflicts with no other, and that no other program which would conflict with it can run concurrently (see Coordination Option in Section II).

Word 6: This optional word specifies the programs' base sector, and which queueing subroutine and which buffer are to be used for passing parameters to the program. Subroutine and buffer numbers are determined when configuring table XPCT (see Figure 4-3 for exact bit assignments).

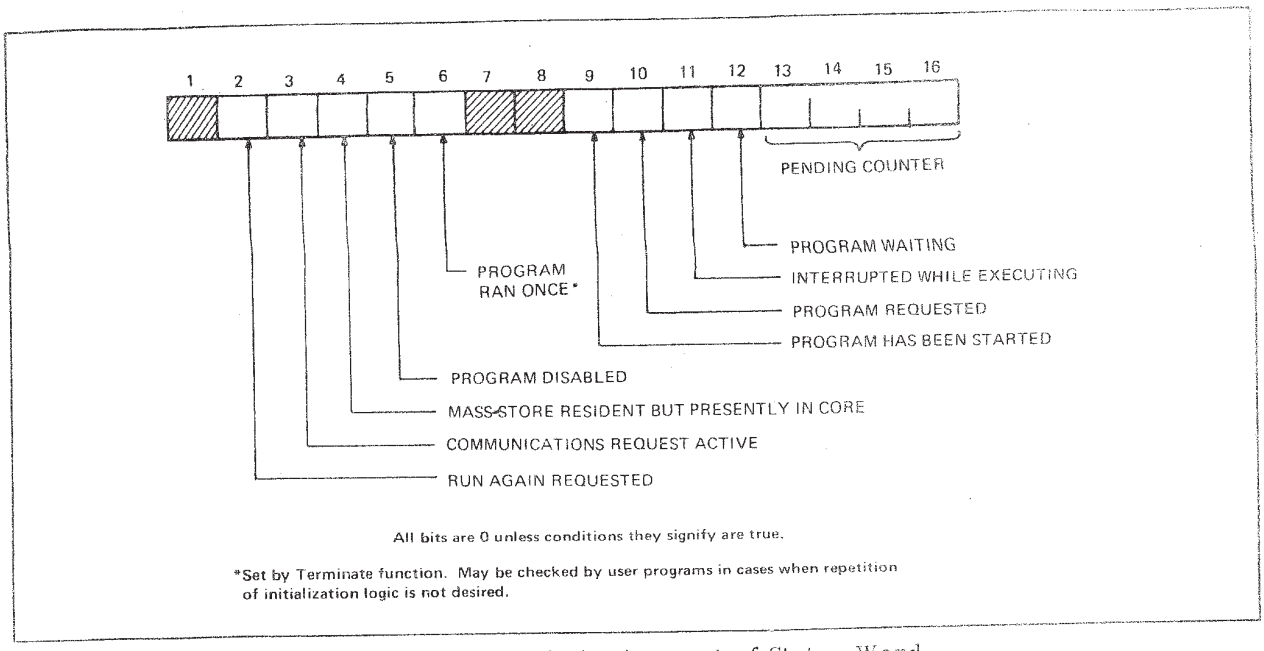


Figure 4-1. Bit Assignment of Status Word

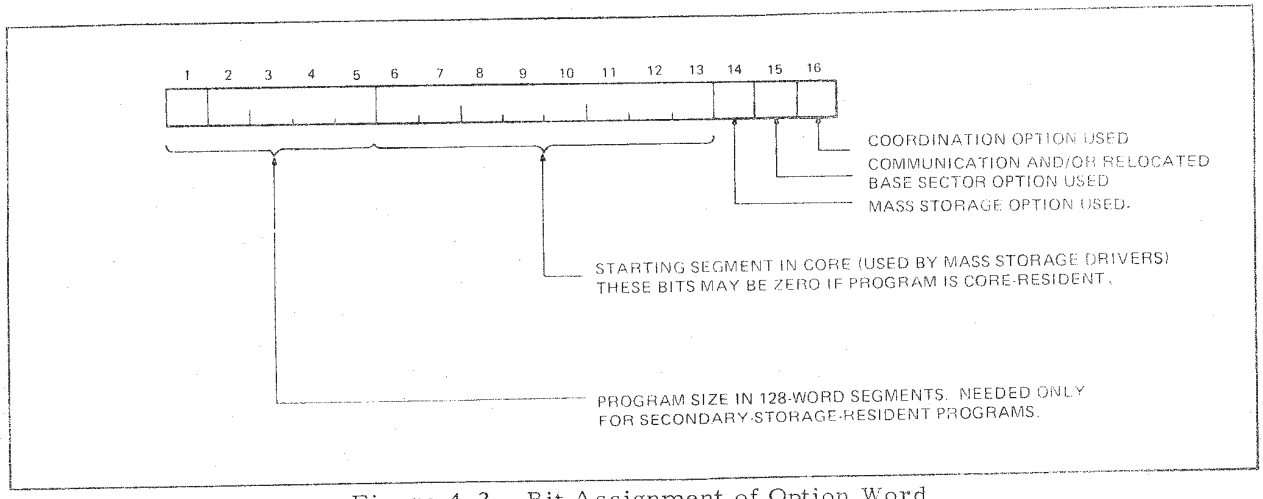


Figure 4-2. Bit Assignment of Option Word

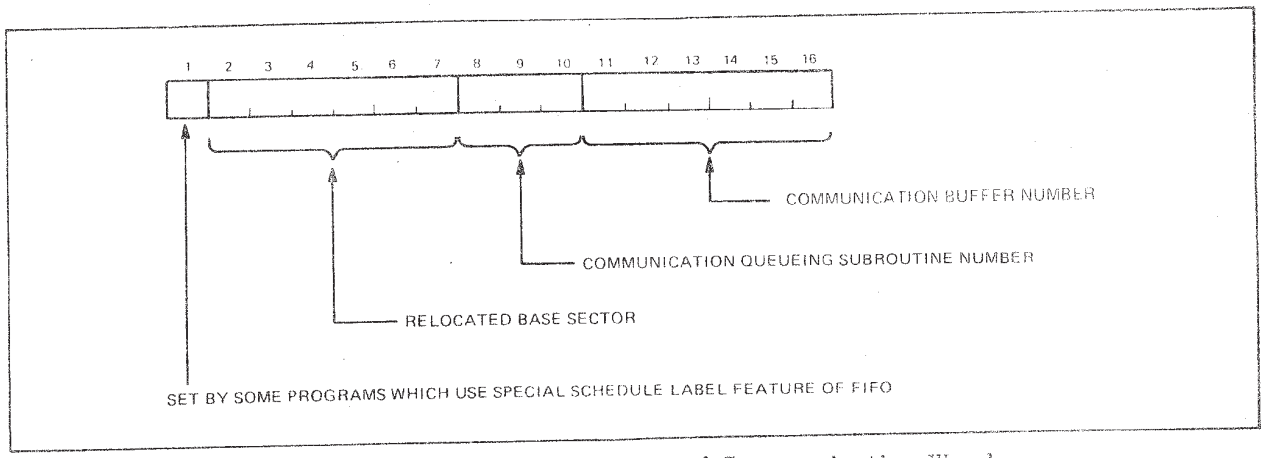


Figure 4-3. Bit Assignments of Communication Word

Word 7: This optional word defines the programs' location on the mass storage device. Mass storage is divided into segments of 128 words, numbered consecutively from 0. This word identifies the first segment in which the program resides on mass storage.

There are five special cases in the XPLT table:

Programs CL, EP, KB, LO, and SM.

Program CL is the Clock program, which is part of the Executive. Its entry must be exactly as follows:

```
CL      BCI     1,CL
        XAC     CL
        OCT     221
        OCT     0
```

Program EP is the Error Print program. Its entry should be as follows:

```
ER      BCI     1,EP
        XAC     EP
        BSZ     1
        OCT     1           Coordination used for ASR device
        OCT     <coordination bit(s)>
```

Program KB is the Keyboard utility program. A minimal entry for this program is shown below; however, the user should refer to Doc. No. 70130072519, OP-16 Utility Programs, for further information.

```
KB      BCI     1,KB
        OCT     <start address>
        BSZ     1
        OCT     1           Coordination used for ASR driver
        OCT     <coordination bit(s)>
```

Program LO is the System Loader (SYSLOAD) used by the Executive to bring mass-store resident programs into core for execution. Its entry should be as follows:

```
LO      BCI     1,LO
        XAC     LO
        BSZ     1
        OCT     2           Communication used
        OCT     <subroutine and buffer numbers>
```

Program SM is the Mass-Store driver. Since it may be used for both program loading and data transfers, it may have one or two entries in XPLT, depending on whether the user wishes to have the same or different priorities for these two operations. For the different priority case the following entries must be made.

```
*       PROGRAM LOADING ENTRY
MSD     BCI     1,ML
        OCT     <start address of mass-store driver>
```

```
        BSZ     1
        OCT     3           Communication and coordination used
        OCT     <coordination bit> same as for SM
        OCT     <subroutine and buffer> buffer must be different from SM

*       DATA TRANSFER ENTRY
SM      BCI     1,SM
        OCT     <start address of mass-store driver>
        BSZ     1
        OCT     3           Communication and coordination used
        OCT     <coordination bit> same as for ML
        OCT     <subroutine and buffer> buffer must be different from ML
```

Note that the two entries must be coordinated so that they are mutually exclusive.

For the same priority case, only the *DATA TRANSFER ENTRY is required, in which case the first word should be MSD BCI 1,SM. Coordination is not necessary.

XPET - EXECUTIVE PROGRAM ENTRY TABLE

This table consists of a list of DAC pointers to program entries in XPLT (the pointers are to the first word of each entry). The order in which the pointers are arranged determines the priority of the programs to which they relate. The highest priority program has its XPLT entry pointer at the top of the list. The last entry must be a DAC to the location containing OCT 0 at the end of XPLT. The format of the tables is as follows.

```
XPET    EQU     *
        DAC     <start address of highest priority programs XPLT entry>
        DAC     <start address of second highest priority programs XPLT entry>
          .
          .
          .
          .
        DAC     <start address of lowest priority programs XPLT entry>
        DAC     <start address of dummy XPLT entry>
```

There are three special cases in this table for the Clock program (CL), the Keyboard utility program (KB), and the System Loader (LO). The pointer to program CL, CL1 DAC CL, is always present and must be at the top of the list. The pointer to program KB, if present, should be KB1 DAC KB. The pointer to program LO, if present, should be LO1 DAC LO.

XIDT - EXECUTIVE INTERRUPT DEFINITION TABLE

This table contains the information necessary to identify an interrupt when it arrives and to cause control to be transferred to the user's interrupt response code.

It consists of three parts. The first (labelled XIDT) contains all the data specific to an interrupt. The second (labelled XID1) is used by the Executive to keep track of the interrupt status of the system. The third (labelled XID2) is used by the Executive to locate the interrupt data in XIDT.

XIDT

This table consists of a number of entries, called device lists, one per interrupt. The device lists must be ordered in the priority with which the user wishes them to be handled, with the highest priority interrupt first. The format of the table is as follows.

```
XIDT    EQU     *
        Device list of highest priority interrupt
          .
          .
          .
          .
          .
          .
        Device list of lowest priority interrupt
IH20    XAC     IH20
IH40    XAC     IH40
```

Each device list must contain the following information.

```
<label>  SKS     <interrupt identification code>
         NOP
         JMP     <address of first word of next device list>
         XAC     IH40
<label>  JST*    *-1
         JST*    IH20
         DAC     **
         DEC     <interrupt reference number>
         OCT     <index to SMK instruction for interrupt>
         OCT     <mask bit for interrupt>
```

Word 1: Is an SKS <'I4>, skip if not interrupting, instruction. The octal number is the code which identifies a certain interrupt. After the last entry in the table, the following instruction must be inserted.

JST* IH40

This jump causes an "interrupt not identified" error message (E11$$). The first SKS must be preceded by XIDT EQU * or labelled XIDT.

Word 2: Should be a NOP instruction, except in the special cases of the real-time clock and ASR. When an interrupt is connected, the NOP is replaced with a JMP *+3 that eventually sends control to the user's interrupt response code. The JMP *+3 is restored to a NOP when the interrupt is disconnected. Two interrupts, the real-time clock and ASR, are connected from the start. The user fills the JMP's in directly instead of NOP's. (see below).

Word 3: A JMP instruction to the first word (SKS instruction) of the next device list. For the last device list, this may be the JST* IH40 referred to under Word 1.

Word 4: Should contain an XAC pointer to IH40, except in the special cases of the real-time clock and ASR (see below). The interrupt response address is placed in this word when the interrupt is connected.

Word 5: Always contains a JST* *-1 for jumping to the user's interrupt response code. It should have a unique label for use by the XID2 table.

Word 6: Always contains a JST* IH20 for returning to the Interrupt Handler after the user's response code is finished.

Word 7: Should be left blank (OCT 0 or DAC **), except in the special case of the real-time clock (see below). It is used by the Executive to store the address of user's XPET entry.

Word 8: Contains the interrupt reference number (see Appendix B) for the particular device associated with this entry.

Word 9: Identifies which SMK instruction is used to enable the device to interrupt (see Word 1 of XID1). Its value will be 1, 2, 3, etc., depending on the rank of the SMK in XID1.

Word 10: Contains the mask bit for the interrupt (see Table 4-1 for mainframe mask bits).

This minimum device list may be followed by any number of device configuration parameters, as required. The address of the first of such parameters is returned to a user in the A register immediately after an interrupt is connected.

As already mentioned above, one exception to the format described is the real-time clock device list, which may be abbreviated as follows.

```
XS01    SKS     '20
        JMP     *+3     Interrupt always connected
        JMP     XS02    Jump to next SKS
        XAC     CL      Address of interrupt response code for clock

XL01    JST*    *-1
        JST*    IH20
        DAC     CL1     Address of XPET entry for clock
```

Note that words 8, 9, and 10 of the device list are not required, because the clock is always connected. However, the clock program does require two configuration parameters which should follow the list above:

```
CLK2 DEC  <number of clock interrupts per second>
CLK3 DEC- <number of hardware intervals between interrupts>
```

Another exception is the ASR device list as follows:

```
XS03    SKS     '404
        JMP     *+3     Interrupt always connected
        JMP     XS04    Jump to next SKS
        XAC     AI      Address of ASR interrupt monitor routine in Executive

XL03    JST*    *-1
        JST*    IH20
        DAC     **
        DEC     4
        OCT     1
        OCT     40
```

At the very end of the XIDT table there must be two XAC's to appropriate portions of the Interrupt Handler:

```
IH20    XAC     IH20
IH40    XAC     IH40
```

XID1

This table has the following format.

```
XID1    DEC     <number of SMK instructions in system>
        OCT     <1st SMK mask>          }
          .                             }
          .                             }  Set 1
          .                             }
        OCT     <Nth SMK mask>          }
        SMK     <1st SMK instruction>   }
          .                             }
          .                             }  Set 2
          .                             }
          .                             }
        SMK     <Nth SMK instruction>   }
```

Word 1: Contains the number of different interrupt mask SMK instructions in the system. It must be labelled XID1. If only a main frame is present, it should contain a 1. If both a main frame and an RTI are present, it should contain a 2, etc. Each set of the words, below, contains as many words as there are different SMK instructions.

Set 1: These words have 1's in each of the bits which correspond to interrupts that are presently connected. The first word (main-frame mask) should be initialized to OCT 41, meaning that the real-time clock and ASR are connected immediately.

Set 2: The first word should be an SMK '20, the main-frame mask-setting instruction. Each following word is the SMK instruction for each of the other interrupt mask registers (RTI, etc.)

XID2

This table has the following format.

```
XID2    DEC  -<number of device lists in XIDT (N)>
        DAC   <address of 5th word of 1st device list>
          .
          .
          .
          .
        DAC   <address of 5th word of Nth device list>
```

The table consists of a negative count of the number of device lists (this count must be labelled XID2), followed by a DAC pointer to the fifth word (the JST* *-1) of each device list. There is no significance in the relative position of the pointers, except that those interrupts most frequently connected and disconnected should be at the top end.

Table 4-1. Main-frame Interrupt Bits (SMK '0020)

Bit	Octal Value	Device
1	100000	Magnetic Tape Control Unit 1
2	40000	Magnetic Tape Control Unit 2
3	20000	
4	10000	Moving-Head Disk
5	4000	I/O Channel 1
6	2000	I/O Channel 2
7	1000	I/O Channel 3
8	400	Small Mass Store
9	200	Paper-Tape Reader
10	100	Paper-Tape Punch
11	40	ASR
12	20	Card Reader
13	10	Card Reader/Punch
14	4	Line Printer
15	2	
16	1	Real-Time Clock

XPCT - EXECUTIVE PROGRAM COMMUNICATION TABLE

This table establishes communication buffers and identifies the parameter passing subroutines required for program communication. Its format is as follows.

```
XPCT      DEC     <number of queueing subroutines (N)>
          DEC     <number of communication buffers (M)>
          XAC     <address of queueing subroutine 1>      \
            .                                              |
            .                                              |  Set 1
            .                                              |
          XAC     <address of queueing subroutine N>      /
          DAC     <address of buffer 1 (label 1)>         \
            .                                              |
            .                                              |  Set 2
            .                                              |
            .                                              |
            .                                              |
          DAC     <address of buffer M (label M)>         /
          DAC     XPCE pointer to end of buffers          \
* BUFFER 1                                                 |
<label 1> DAC     *+2  In pointer                         |
          DAC     *+1  Out pointer                        |
          BSZ     <size of buffer>                        |
            .                                             |
            .                                             |  Set 3
            .                                             |
            .                                             |
* BUFFER M                                                |
<label M> DAC     *+2  In pointer                         |
          DAC     *+1  Out pointer                        |
          BSZ     <size of buffer>                        |
* END OF BUFFERS                                          |
XPCE      OCT     0                                       /
```

Word 1: Specifies the number of queueing subroutines in the system (a maximum of seven is allowed). It must be labelled XPCT.

Word 2: Specifies the number of communication buffers in the system.

Set 1: These words are pointers to the queueing subroutines which are usually external to XCOM. There must be as many pointers as specified by Word 1. An entry for the standard queueing subroutine FIFO would be XAC FIFO.

Set 2: These words are pointers to the communication buffers. There must be as many pointers as specified by Word 2. After the last pointer to a buffer is a pointer (DAC XPCE) to a zero word that follows the last buffer (XPCE OCT 0).

Set 3: The buffers themselves are defined here. Buffers for the standard queueing routine FIFO must contain two words more than the user needs for bookkeeping purposes (In pointer and Out pointer). The first word of each buffer should have a unique label. Buffers to be used by other queueing subroutines may have other requirements.

XCUT - EXECUTIVE CLOCK USER'S TABLE

This table is serviced by the Executive and the Clock program and has the following format.

```
XCUT    DAC     *+4     Header
        BSZ     3
*
        DAC     *+5     1st Entry
        BSZ     4
          .
          .
          .
          .
*         .
          .
        DAC     *+5     (N-1)th entry
        BSZ     4
*
        BSZ     5       Nth and last entry
```

Where N is the maximum number of simultaneous clock users.

XIVT - EXECUTIVE INTERRUPTED VARIABLES TABLE

This table is where the Executive stores the registers and status of interrupted programs. The user should decide for himself how many programs he is going to allow to be in an interrupted state at any one time. This limit governs the size of the table, and when the table is full, no more programs are interrupted. The Executive concentrates on completing interrupted programs before starting up any new programs.

Five words are required for each interrupted program, thus the amount of variables storage space Z can be calculated as:

$$Z = 5 (1 = y)$$

where Y is the maximum number of programs, or program priority groups, the user will allow to be in an interrupted state at the same time. (A 'priority group' is a group of programs that can never run together because they share common coordination bits.)

The format of XIVT is as follows.

```
XIVT    DEC     Y
        BSZ     Z
```

where Y and Z are as defined above.

XLPT - EXECUTIVE LABEL PARAMETER TABLE

This is a table in which the Executive temporarily stores labels that have been scheduled by the interrupt response code. Its format is as follows.

```
XLPT    BSZ     <2 times the number of interrupts>
        DEC     -1 End of table
```

The table consists of two words for each interrupt in the system and an end-of-table marker. Should the table become full, the error message E12XX is printed, where XX is the name of the program in which the label should have been scheduled.

XFET - EXECUTIVE FUNCTION ENTRY TABLE

This table defines the system functions in a system and has the following format.

```
XFET    DEC     <largest function number in system (N)>
        XAC *   RP    Address of function 1   }
        XAC *   SL    Address of function 2   }
        XAC *   CC    Address of function 3   }
        XAC *   DC    Address of function 4   }   Set 1
        XAC     CI    Address of function 5   }
        XAC     DI    Address of function 6   }
        XAC     TE    Address of function 7   }
        XAC     WA    Address of function 8   }
*
        XAC     <address of function 9>       }
        DAC     ** Dummy for function 10      }
          .                                   }   Set 2
          .                                   }
          .                                   }
        XAC     <address of function N>       }
```

Word 1: Contains the largest function number in the system; must be labelled XFET. For systems without user-written functions, this word should be XFET DEC 8.

Set 1: A mandatory list of pointers to the start locations of the eight standard functions in the Executive. The position in the list corresponds to the function number; that is, the first entry is function number 1, the second entry function 2, etc. The indirect flag on the first four XAC's indicates that a name search is required by the function.

Set 2: An optional list of pointers to additional user-written functions. For every function of number less than that of the largest function number in the system that is not incorporated, dummy items (DAC **) must be used. (See Section VIII for information on how to write new system functions.)

XINT - EXECUTIVE INITIALIZATION LOCATION

This is a location which contains a pointer to the user's initialization control subroutine. The format of this pointer is:

```
XINT XAC INIT
```

if the subroutine (INIT) is external to XCOM, or:

```
XINT DAC INIT
```

if the subroutine is internal to XCOM.

If no user initialization is required the format is:

```
XINT XAC SC
```

to link directly to the Scheduler. (See Section VIII for details on writing the "Initialization Control Subroutine.")

XDCT - EXECUTIVE DEVICE CONFIGURATION TABLE

This table contains configurable information required by certain device drivers. Its format is as follows.

```
XDCT    DEC     <Mass-store data>
        DEC     <ASR data>
        DAC     <Pointer to RO-35 Alarm Typewriter data>
        DAC     <Pointer to Model B Logging Typewriter data>   } Set 1
        DEC     <Line Printer data>
        DEC     <Card Reader or Card Reader/Punch data>
        DEC     <Magnetic Tape Unit data>
          .
          .
          .
          .
          .                                                    } Set 2
        RO-35   Alarm Typewriter data
          .
          .
          .
        Model B Logging Typewriter data
```

Set 1: A variable-length, fixed sequence of single-word entries. Each word relates to a specific device. This word may contain data or may be a pointer to two or more data parameters. The sequence of entries may be expanded or contracted but never changed.

Set 2: All multiparameter data groups are to be placed here after the last fixed-sequence entry. Data group sequence is not significant, because access is made via the pointers in Set 1.

Refer to the specific device-driver manuals for further information on the data values. User-written device drivers may use the device lists in the XIDT table for placement of configurable information (see XIDT table description in this section).

XSPT - EXECUTIVE SPECIAL PARAMETERS TABLE

This table contains those special parameters that do not fit properly in any of the other tables. It also serves as a predefined expansion area for user or system needs. Its format is as follows:

```
XSPT    OCT    <Relocated Base Sector option>
XPFP    OCT    <power failure interrupt response address>
XEXA    OCT    <Extended Addressing option>
        OCT    <low-core protection limit>
        OCT    <high-core protection limit>
        OCT    <base segment number for overlays>
```

Word 1: Should be nonzero when the Relocatable Base Sector option is present in the user's hardware configuration or zero if it is not present. It must be labelled XSPT.

Word 2: Contains the address of the power failure interrupt response code. This address will be inserted in location '60 by the Executive at system startup. If a halt is sufficient, the user may use '1023, which will cause the system to halt at location '1024. It must be labelled XPFP.

Word 3: Should be nonzero when the Extended Addressing option is present in the user's hardware configuration or zero if it is not present. It must be labelled XEXA.

Words 4, 5 and 6: Contain information required by the keyboard utility programs. (See Doc. No. 70130072519, OP-16 Utility Programs, for details.)

XCOM SIZE ESTIMATION

The approximate formula for estimating the size(s) of the RTX-16 Configuration Module (in words) is as follows.

$$S=5A+B+C+D+13E+2F+G+3H+I+5J+5K+L+42$$

where:

A = Number of programs in the system, including RTX-16 Clock program, RTX-16 Error Print program, RTX-16 Keyboard program, device drivers and user programs.

B = Number of programs using the coordination option of the Executive. (All mass-store resident programs must use this option; also the Keyboard, ASR Driver, and Error Print programs if any two or all three are present; and any user programs requiring the option.

C = Number of programs using the communication or relocated base sector option of the Executive.

D = Number of programs residing on the mass-store device.

E = Number of hardware devices which generate interrupts. (Examples are real-time clock, I/O devices, etc.)

F = Number of different SMK instructions for interrupt masking. (Examples are SMK '20 for main frame, SMK '620 for RTI, etc.)

G = Number of parameter-passing subroutines. (One is standard and is supplied with the system.)

H = Number of buffers for saving parameters. (Generally, each program using the communication option will need its own buffer.)

I = Total number of buffer locations for saving parameters. (Example: five programs use the communication option, and a maximum of four requests will be saved for each program: I=(4) (5) = 20.)

J = Maximum number of programs to be connected to the clock for periodic execution. (Programs are connected by using System Function 3.)

K = Maximum number of programs which may be in an interrupted state at any given time. (If the maximum is exceeded, no error occurs, except that the priority scheme for determining the next program to be executed is not used.)

L = The number of user-written system functions.

CONFIGURING SAMPLE SYSTEM

This subsection describes a sample system and shows its Configuration Module. All the programs used as examples throughout this manual are included in this sample system. The hardware supported is either a Model 316 or 516 main frame with 16K of memory, a Real-Time Clock, a High-Speed Paper-Tape Punch, a High-Speed Paper-Tape Reader, a Mass Store, an ASR-33, a Line Printer, a Card Reader/Punch, a Magnetic Tape Unit, and an RO-35 Typewriter connected via a Real-Time Interface.

Core Map

Figure 4-4 shows the core map. The core-resident part of the system is in sectors 1 through 11 octal. All programs above sector 11 are mass-store resident.

Mass-Store Layout

Table 4-2 shows the storage allocation on the Mass Store.

Configuration Module

Table 4-3 is an assembly listing of the Configuration Module for this system. Comments have been included where appropriate for clarification.

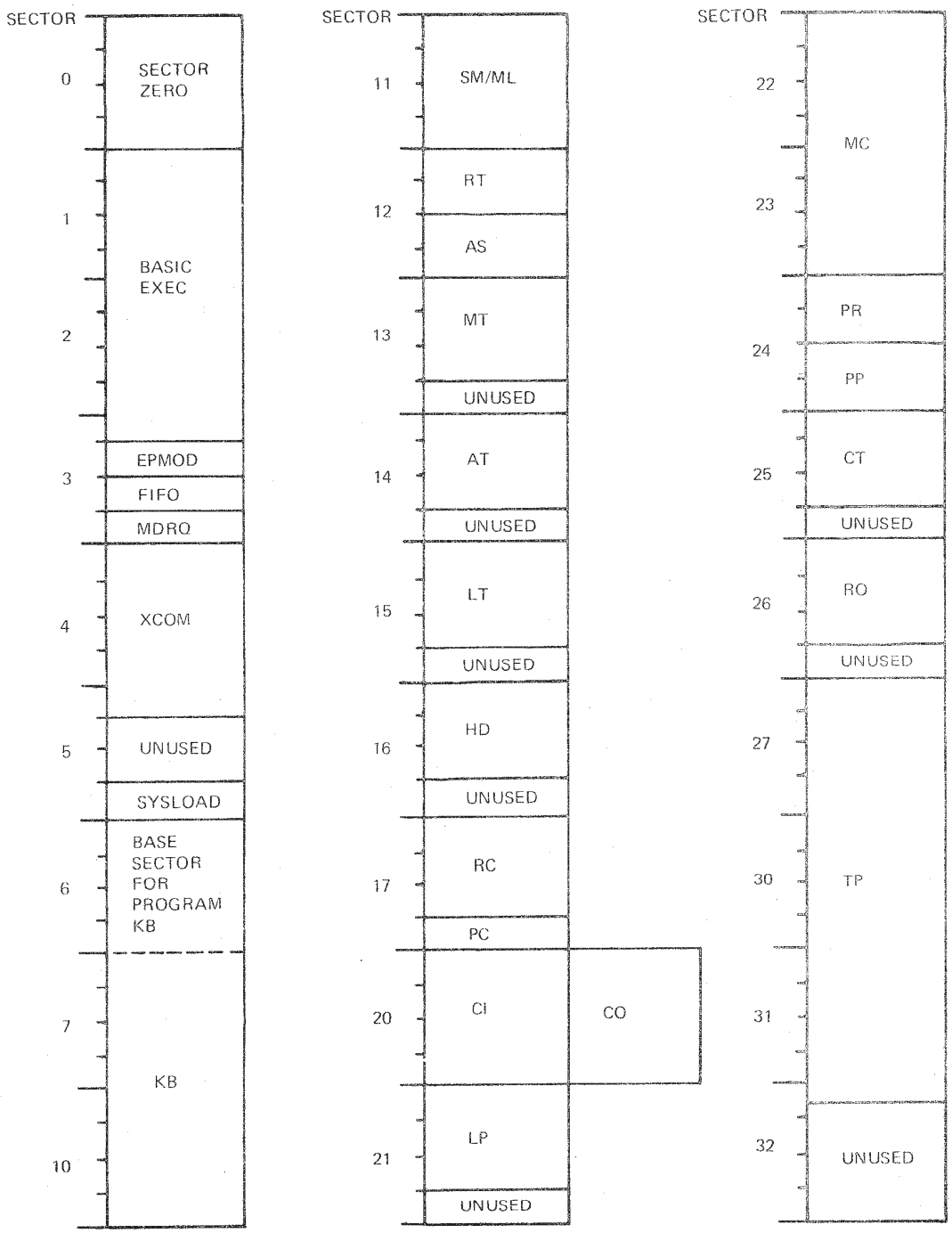
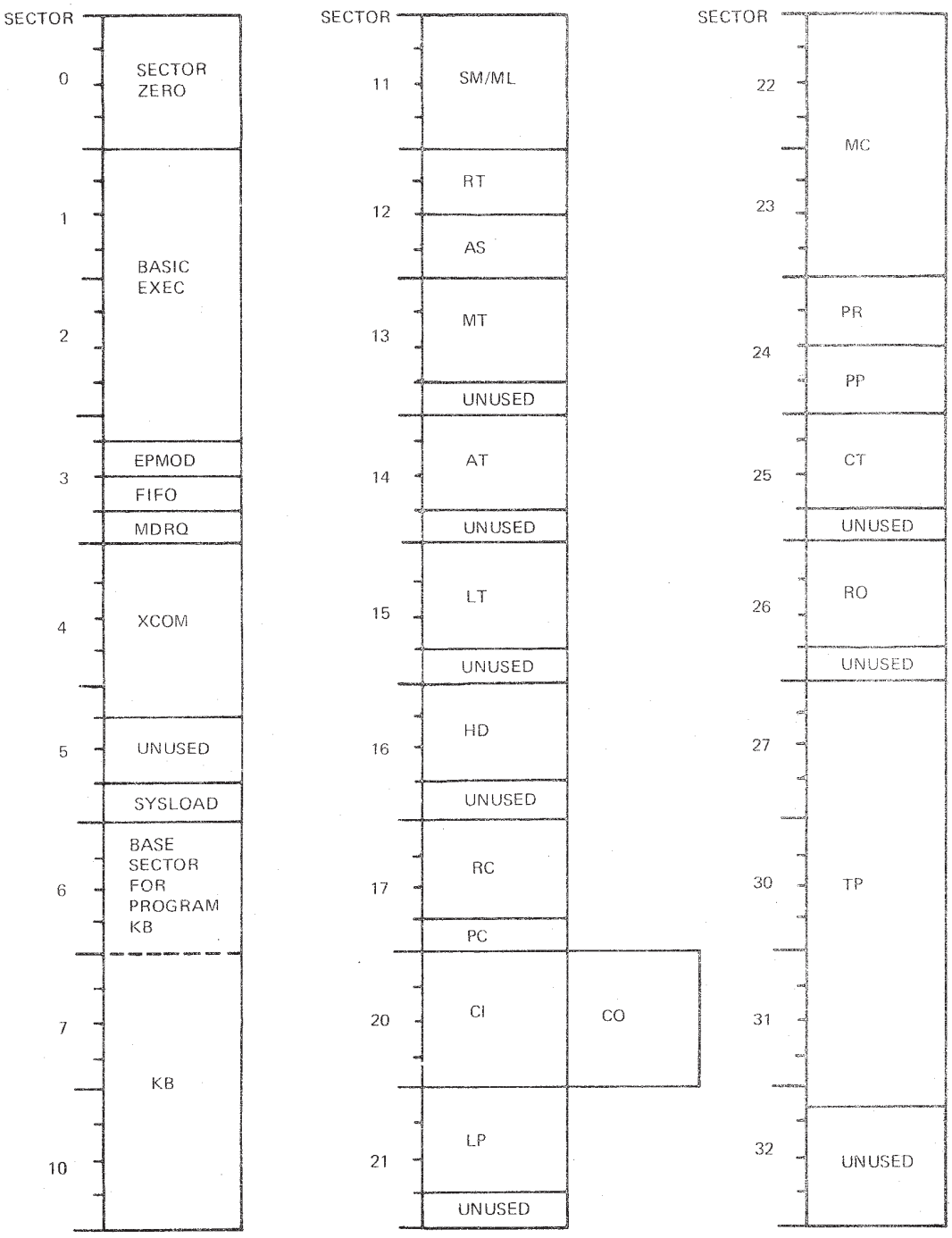


Figure 4-4. Core Map for Sample System



Long ticks separate sectors; short ticks separate segments.
All programs above Sector 11 are mass-store resident.

Figure 4-4. Core Map for Sample System

Table 4-3. Sample Configuration Module

```
* XCOM HEADER
*
      SUBR  XPLT
      SUBR  XIDT
      SUBR  XID1
      SUBR  XID2
      SUBR  XPCT
      SUBR  XCUT
      SUBR  XIVT
      SUBR  XPET
      SUBR  XLPT
      SUBR  XFET
      SUBR  XINT
      SUBR  XDCT
      SUBR  XSPT
      SUBR  XPFP
      SUBR  XEXA
      SUBR  KB1
      SUBR  ER
      SUBR  CLK2
      SUBR  CLK3
      SUBR  MSD
      SUBR  LOI
      REL
*
*     NOTE:  IF NO PROGRAMS ARE MASS-STORE RESIDENT THE LAST
*            ENTRIES IN THE ABOVE HEADER MUST BE:
*            SUBR  MSD,XPLT
*            SUBR  LOI
*      LOI   EQU   0
*            REL
*
*
*
* XPLT - EXECUTIVE PROGRAM LIST TABLE
XPLT EQU   *
*
*     CLOCK PROGRAM
CL    BCI   1,CL                  PROGRAM NAME
      XAC   CL                    START ADDRESS
      OCT   221                   STATUS (STARTED/WAITING/INT. CONNECTED)
      OCT   0                     NO OPTIONS
*
*     PAPER TAPE READER DRIVER
PR    BCI   1,PR
      OCT   24006
      BSZ   1
      OCT   011207
      OCT   0                     NO CONFLICT
      OCT   00104
      OCT   320

*     ASR DRIVER
AS    BCI   1,AS
      OCT   12406
      BSZ   1
      OCT   010527
      OCT   2000                  COORDINATION FOR ASR DEVICE
      OCT   00102
      OCT   314
*
*     ERROR PRINT PROGRAM
ER    BCI   1,EP
      XAC   EP
      BSZ   1
```

Table 4-3 (cont). Sample Configuration Module

```
        OCT   1
        OCT   2000              COORDINATION FOR ASR DEVICE
*
*       SYSTEM LOADER
LO      BCI   1,LO
        XAC   LO
        BSZ   1
        OCT   2
        OCT   00105
*
*       MASS STORE DRIVER ENTRY FOR PROGRAM LOADING
MSD     BCI   1,ML
        OCT   11006
        BSZ   1
        OCT   3
        OCT   2                 COORDINATION WITH SM
        OCT   00101
*
*       MASS STORE DRIVER ENTRY FOR DATA TRANSFERS
SM      BCI   1,SM
        OCT   11006
        BSZ   1
        OCT   3
        OCT   2                 COORDINATION WITH ML
        OCT   00121
*
*       MAG TAPE DRIVER
MT      BCI   1,MT
        OCT   13006
        BSZ   1
        OCT   014547
        OCT   0                 NO CONFLICT
        OCT   00112
        OCT   330
*
*       LINE PRINTER DRIVER
LP      BCI   1,LP
        OCT   21007
        BSZ   1
        OCT   015047
        OCT   0                 NO CONFLICT
        OCT   00114
        OCT   301
*
*       CARD READER DRIVER
CI      BCI   1,CI
        OCT   20010
        BSZ   1
        OCT   021007
        OCT   10000             COORDINATION FOR SECTOR 20
        OCT   00116
        OCT   324
*
*       CARD PUNCH DRIVER
CO      BCI   1,CO
        OCT   20010
        BSZ   1
        OCT   021007
        OCT   10000             COORDINATION FOR SECTOR 20
        OCT   00120
        OCT   351
*
*       PAPER TAPE PUNCH DRIVER
PP      BCI   1,PP
        OCT   24406
```

Table 4-3 (cont). Sample Configuration Module

```
        BSZ   1
        OCT   011227
        OCT   0                    NO CONFLICT
        OCT   00103
        OCT   316
*
*       RO-35 TYPEWRITER DRIVER
RO      BCI   1,RO
        OCT   26006
        BSZ   1
        OCT   015307
        OCT   0                    NO CONFLICT
        OCT   00200
        OCT   360
*
*       MASS STORE TEST
ID      BCI   1,HD
        OCT   16006
        BSZ   1
        OCT   014707
        OCT   0                    NO CONFLICT
        OCT   00111
        OCT   333
*
*       MAG TAPE TEST
MC      BCI   1,MC
        OCT   22006
        BSZ   1
        OCT   041107
        OCT   0                    NO CONFLICT
        OCT   00113
        OCT   304
*
*       LINE PRINTER TEST
LT      BCI   1,LT
        OCT   15006
        BSZ   1
        OCT   014647
        OCT   0                    NO CONFLICT
        OCT   00115
        OCT   336
*
*       PAPER TAPE READER TEST
RC      BCI   1,RC
        OCT   17006
        BSZ   1
        OCT   014747
        OCT   0                    NO CONFLICT
        OCT   00107
        OCT   341
*
*       ASR TEST
AT      BCI   1,AT
        OCT   14006
        BSZ   1
        OCT   014607
        OCT   1                    COORDINATION FOR ASR DRIVER
        OCT   00106
        OCT   346
*
*       PAPER TAPE PUNCH TEST
PC      BCI   1,PC
        OCT   17606
        BSZ   1
        OCT   004777
```

Table 4-3 (cont). Sample Configuration Module

```
        OCT   0                    NO CONFLICT
        OCT   00110
        OCT   344
*
*       CARD READER/PUNCH TEST
CT      BCI   1,CT
        OCT   25006
        BSZ   1
        OCT   015247
        OCT   0                    NO CONFLICT
        OCT   00117
        OCT   355
*
*       RO-35 TEST
RT      BCI   1,RT
        OCT   12006
        BSZ   1
        OCT   010507
        OCT   0                    NO CONFLICT
        OCT   00122
        OCT   363
*
*       ON-LINE TRACE PROGRAM
TP      BCI   1,TP
        OCT   27006
        BSZ   1
        OCT   2
        OCT   00123
*
*       KEYBOARD UTILITY PROGRAM
KB      BCI   1,KB
        OCT   7003
        BSZ   1
        OCT   3
        OCT   1                    COORDINATION FOR ASR DRIVER
        OCT   06000                USES SECTOR 6 FOR BASE SECTOR
*
*       END OF PROGRAM ENTRIES
FF      OCT   0                    END OF XPLT TABLE
*
*
* XPET - EXECUTIVE PROGRAM ENTRY TABLE
XPET EQU   *
*
CL1     DAC   CL                   HIGHEST PRIORITY PROGRAM
        DAC   PR
        DAC   AS
        DAC   ER
LO1     DAC   LO
        DAC   MSD
        DAC   SM
        DAC   MT
        DAC   LP
        DAC   CI
        DAC   CO
        DAC   PP
        DAC   RO
        DAC   HD
        DAC   MC
        DAC   LT
        DAC   RC
        DAC   AT
        DAC   PC
        DAC   CT
        DAC   RT
```

Table 4-3 (cont). Sample Configuration Module

```
      DAC   TP
KB1   DAC   KB                    LOWEST PRIORITY PROGRAM
      DAC   FF
*
*
* EXECUTIVE INTERRUPT DEFINITION TABLES
IDT EQU   *
*
*     REAL TIME CLOCK (SPECIAL CASE)
XS01 SKS    '0020
      JMP   *+3                   INTERRUPT ALWAYS CONNECTED
      JMP   XS02
      XAC   CL
XL01 JST*   *-1
      JST*  IH20
      DAC   CL1
CLK2 DEC    20                    NUMBER OF CLOCK INTERRUPTS PER SECOND
CLK3 DEC    -3                    NUMBER OF INTERVALS BETWEEN INTERRUPTS
*
*     CARD READER/PUNCH
XS02 SKS    '0106                 SKIP IF NOT INTERRUPTING INSTRUCTION
      NOP                         INTERRUPT TO BE CONNECTED
      JMP   XS03                  JUMP TO NEXT ENTRY
      XAC   IH40                  INTERRUPT RESPONSE ADDRESS
XL02 JST*   *-1                   JUMP TO INTERRUPT RESPONSE CODE
      JST*  IH20                  RETURN TO INTERRUPT HANDLER
      DAC   **                    XPET POINTER STORAGE
      DEC   8                     INTERRUPT REFERENCE NUMBER
      OCT   1                     INDEX TO SMK INSTRUCTION
      OCT   10                    MASK BIT
*
*     PAPER TAPE READER
XS03 SKS    '0401
      NOP
      JMP   XS04
      XAC   IH40
XL03 JST*   *-1
      JST*  IH20
      DAC   **
      DEC   2
      OCT   1
      OCT   000200
*
*     ASR TELETYPE (SPECIAL CASE)
XS04 SKS    '0404
      JMP   *+3                   INTERRUPT ALWAYS CONNECTED
      JMP   XS05
      XAC   AI
XL04 JST*   *-1
      JST*  IH20
      DAC   **
      DEC   4
      OCT   1
      OCT   000040
*
*     MOVING HEAD DISC
XS05 SKS    '0125
      NOP
      JMP   XS06
      XAC   IH40
XL05 JST*   *-1
      JST*  IH20
      DAC   **
      DEC   1
```

Table 4-3 (cont). Sample Configuration Module

```
        OCT   1
        OCT   10000
*
*       PAPER TAPE PUNCH
XS06    SKS   '0402
        NOP
        JMP   XS07
        XAC   IH40
XL06    JST*  *-1
        JST*  IH20
        DAC   **
        DEC   3
        OCT   1
        OCT   000100
*
*       LINE PRINTER
XS07    SKS   '0103
        NOP
        JMP   XS08
        XAC   IH40
XL07    JST*  *-1
        JST*  IH20
        DAC   **
        DEC   9
        OCT   1
        OCT   000004
*
*       MAG TAPE UNIT
XS08    SKS   '0410
        NOP
        JMP   XS09
        XAC   IH40
XL08    JST*  *-1
        JST*  IH20
        DAC   **
        DEC   7
        OCT   1
        OCT   100000
*
*       RO-35 TYPEWRITER
XS09    SKS   '0327
        NOP
        JST*  IH40                  INTERRUPT NOT IDENTIFIED
        XAC   IH40
XL09    JST*  *-1
        JST*  IH20
        DAC   **
        DEC   5
        OCT   2
        OCT   10000
*
IH40    XAC   IH40                  UNIDENTIFIED INTERRUPT RETURN ADDRESS
IH20    XAC   IH20                  NORMAL INTERRUPT RETURN ADDRESS
*
*
XID2    DEC   -9                    NINE DEVICE LISTS
        DAC   XL01                  DEVICE LIST POINTERS
        DAC   XL02
        DAC   XL03
        DAC   XL04
        DAC   XL05
        DAC   XL06
        DAC   XL07
        DAC   XL08
        DAC   XL09
```

Table 4-3 (cont). Sample Configuration Module

```
*
*
XIDI  OCT   2                     TWO SMK INSTRUCTIONS
      OCT   41                    MAINFRAME SMK MASK (ASR AND RTC)
      OCT   0                     RTI SMK MASK
      SMK   °20                   MAINFRAME SMK INSTRUCTION
      SMK   °620                  RTI SMK INSTRUCTION
*
* XPCT - EXECUTIVE PROGRAM COMMUNICATION TABLE
* HEADER
XPCT  DEC   2                     TWO QUEUEING SUBROUTINES
      DEC   21                    TWENTY-ONE BUFFERS
      XAC   FIFO                  STANDARD QUEUEING ROUTINE
      XAC   MDRQ                  SPECIAL QUEUEING ROUTINE
      DAC   BF1                   BUFFER POINTERS
      DAC   BF2
      DAC   BF3
      DAC   BF4
      DAC   BF5
      DAC   BF6
      DAC   BF7
      DAC   BF8
      DAC   BF9
      DAC   BF10
      DAC   BF11
      DAC   BF12
      DAC   BF13
      DAC   BF14
      DAC   BF15
      DAC   BF16
      DAC   BF17
      DAC   BF18
      DAC   BF19
      DAC   BF20
      DAC   BF21
      DAC   XPCE
*
* BUFFERS
*
*     USED BY MASS STORE DRIVER (PROGRAM LOADING)
BF1   DAC   *+2                   IN POINTER
      DAC   *+1                   OUT POINTER
      BSZ   1                     PARAMETER STORAGE
*
*     USED BY ASR DRIVER
BF2   DAC   *+2
      DAC   *+1
      BSZ   5
*
*     USED BY PAPER TAPE PUNCH DRIVER
BF3   DAC   *+2
      DAC   *+1
      BSZ   5
*
*     USED BY PAPER TAPE READER DRIVER
BF4   DAC   *+2
      DAC   *+1
      BSZ   5
*
*     USED BY SYSTEM LOADER
BF5   DAC   *+2
      DAC   *+1
      BSZ   2
```

Table 4-3 (cont). Sample Configuration Module

```
*
*     USED BY ASR TEST
BF6   DAC   *+2
      DAC   *+1
      BSZ   5
*
*     USED BY PAPER TAPE READER TEST
BF7   DAC   *+2
      DAC   *+1
      BSZ   5
*
*     USED BY PAPER TAPE PUNCH TEST
BF8   DAC   *+2
      DAC   *+1
      BSZ   5
*
*     USED BY MASS STORE TEST
BF9   DAC   *+2
      DAC   *+1
      BSZ   5
*
*     USED BY MAG TAPE DRIVER
BF10  DAC   *+2
      DAC   *+1
      BSZ   5
*
*     USED BY MAG TAPE TEST
BF11  DAC   *+2
      DAC   *+1
      BSZ   5
*
*     USED BY LINE PRINTER DRIVER
BF12  DAC   *+2
      DAC   *+1
      BSZ   5
*
*     USED BY LINE PRINTER TEST
BF13  DAC   *+2
      DAC   *+1
      BSZ   5
*
*     USED BY CARD READER DRIVER
BF14  DAC   *+2
      DAC   *+1
      BSZ   5
*
*     USED BY CARD READER/PUNCH TEST
BF15  DAC   *+2
      DAC   *+1
      BSZ   5
*
*     USED BY CARD PUNCH DRIVER
BF16  DAC   *+2
      DAC   *+1
      BSZ   5
*
*     USED BY MASS STORE DRIVER (DATA TRANSFERS)
BF17  DAC   *+2
      DAC   *+1
      BSZ   5
*
*     USED BY RO-35 TEST
BF18  DAC   *+2
      DAC   *+1
      BSZ   5
```

Table 4-3 (cont). Sample Configuration Module

```
*
*     USED BY TRACE PROGRAM
BF19  DAC   *+2
      DAC   *+1
      BSZ   2
*
*     USED BY RO-35 TYPER 1
BF20  DAC   *+1                 IN POINTER
      BSZ   5                   PARAMETER STORAGE
      DEC   -1                  END OF BUFFER MARKER
*
*     USED BY RO-35 TYPER 2
BF21  DAC   *+1
      BSZ   5
      DEC   -1
*
XPCE  BSZ   1                   END OF BUFFERS
* XCUT - EXECUTIVE CLOCK USER'S TABLE
*
XCUT  DAC   *+4                 EMPTY THREAD START
      BSZ   1                   MILLISECOND UNITS THREAD START
      BSZ   1                   SECONDS THREAD START
      BSZ   1                   MINUTES THREAD START
*
      DAC   *+5                 ENTRY  1
      BSZ   4
*
      DAC   *+5                 ENTRY  2
      BSZ   4
*
      DAC   *+5                 ENTRY  3
      BSZ   4
*
      DAC   *+5                 ENTRY  4
      BSZ   4
*
      BSZ   5                   ENTRY  5
* XIVT - EXECUTIVE INTERRUPTED VARIABLES TABLE
*
XIVT  DEC   5                   5  PROGRAMS MAY BE INTERRUPTED
      BSZ   30
*
* XLPT - EXECUTIVE LABEL PARAMETER TABLE
XLPT  BSZ   18                  NINE INTERRUPTS IN SYSTEM
      DEC   -1                  END OF TABLE MARKER
*
* XFET - EXECUTIVE FUNCTIONS ENTRY TABLE
XFET  DEC   8                   HIGHEST FUNCTION NUMBER IN SYSTEM
*
      XAC*  RP                  REQUEST PROGRAM
      XAC*  SL                  SCHEDULE LABEL
      XAC*  CC                  CONNECT CLOCK
      XAC*  DC                  DISCONNECT CLOCK
      XAC   CI                  CONNECT INTERRUPT
      XAC   DI                  DISCONNECT INTERRUPT
      XAC   TE                  TERMINATE
      XAC   WA                  WAIT
*
* XDCT - EXECUTIVE DEVICE CONFIGURATION TABLE
XDCT  DEC   20                  20-SURFACE MOVING HEAD DISC
      DEC   1                   ASR-33
      DAC   ROD                 POINTER TO RO-35 DATA
      BSZ   1                   NOT USED
      DEC   3                   LINE PRINTER ON CHANNEL 3
```

Table 4-3 (cont). Sample Configuration Module

```
      OCT   0                    CARD READER/PUNCH ON I/O BUS
      DEC   2                    MAG TAPE UNIT ON CHANNEL 2
*
*     RO-35 DATA
ROD   DEC   2                    NUMBER OF RO-35 TYPERS
      DAC   ROD1                 POINTER TO DATA FOR FIRST TYPER
      DAC   ROD2                 POINTER TO DATA FOR SECOND TYPER
      OCP   '327                 ACKNOWLEDGE INTERRUPT INSTRUCTION
ROD1  DAC   BF20                 COMMUNICATION BUFFER 20
      OCT   13                   PAGE 0, PAC 13
      BSZ   6                    DEVICE ID STORAGE
ROD2  DAC   BF21                 COMMUNICATION BUFFER 21
      OCT   14                   PAGE 0, PAC 14
      BSZ   6                    DEVICE ID STORAGE
*
* XSPT - EXECUTIVE SPECIAL PARAMETER TABLE
XSPT  OCT   1                    RELOCATED BASE SECTOR OPTION USED
XPFP  XAC   XSSA                 POWER FAILURE INT. RESPONSE ADDRESS
XEXA  OCT   0                    EXTENDED ADDRESSING OPTION NOT USED
      OCT   7777                 LOW CORE PROTECTION LIMIT
      OCT   17777                HIGH CORE PROTECTION LIMIT
*
* XINT - INITIALISATION ROUTINE POINTER
XINT  XAC   SC                   NO USER INITIALIZATION
      END
```

SECTION V
RTX-16 UTILITY PROGRAMS

UTILITY PROGRAMS

Overview

Four families of utility programs are included as part of OP-16:

1. Off-line core only (OFLCUP)
2. Off-line core mass store (OFLMUP)
3. On-line core only (ONLCUP)
4. On-line core mass store (ONLMUP)

OFLCUP provides for off-line debugging of core-resident programs. OFLMUP is designed to load programs, data, and a copy of the Executive (if desired) onto the system mass store at the time when a system is being built and to provide for off-line debugging. On-line core mass-store utilities are designed to provide for programmer-computer communication, on-line transfer of information from sending devices to receiving devices (for example, core, mass store, ASR, paper tape, magnetic tape, etc.), and on-line debugging.

All utility programs are configurable for a variety of I/O devices as well as for a variety of functions. In general, the user is expected to configure the utility programs to suit a particular hardware environment and to include only a desired set of functions. In designing the utilities, configurability was achieved by emphasizing modular construction at the cost of a slight increase in core usage.

The structure and components of off- and on-line utilities of equivalent scope are identical. The essential difference is that on-line utilities run under the control of the RTX-16 Executive, while off-line utilities run under the control of a small Executive simulator. In both cases the configured and loaded utility programs are treated by the Executives as any other programs. Components comprising the the utility programs are included in the OP-16 Utility Program Library and are listed in Doc. No. 70181898-311, Binder Table of Contents for OP-16 (BTC1OP16).

A detailed discussion of utility program operation, features, and building procedures is included in Doc. No. 70130072519, OP-16 Utility Programs. The following paragraphs provide only a brief summary of utility program features for the reader's convenience.

Preconfigured Special-Purpose Utility Programs

A set of preconfigured utility programs is provided for the user's convenience:

1. One set of off-line core-only utilities (OFLUT-1) for an ASR and high-speed paper-tape reader/punch environment with debugging features. This version provides the user with an immediate initial debugging tool.

2. Two sets of basic off-line core mass-store utilities (OFLUT-2 and -3), each supporting a different mass-store device. These basic core mass-store utilities are built specifically to assist the user in creating other core mass-store utilities with expanded capabilities as desired for the application.

Functions

By selecting the appropriate components from the Utility Program Library, the following features may be incorporated by the user into the custom-tailored run-time utility package in accordance with the building procedures described in Doc. No. 70130072519, OP-16 Utility Programs.

1. Media transfer and verify features (used in system building, loading, and dumping):

   TR COSM - Transfers specified number of segments from core to mass store.

   TR SMCO - Transfers specified number of segments from mass store to core.

   VE SMCO - Verifies specified number of segments on mass store against core and prints differences.

   TR COPP - Transfers specified area of core to paper tape.

   TR PRCO - Transfers information from paper tape to core.

   VE PRCO - Verifies information from paper tape against core and prints differences.

   TR SMPP - Transfers specified number of segments from mass store to paper tape.

   TR PRSM - Transfers information from paper tape to mass store.

   VE PRSM - Verifies information from paper tape against mass store and prints differences.

   TR COMT - Transfers specified area of core to magnetic tape.

   TR MTCO - Transfers information from magnetic tape to core.

   VE MTCO - Verifies information from magnetic tape against core and prints differences.

   TR SMMT - Transfers specified segments from mass store to magnetic tape.

   TR MTSM - Transfers information from magnetic tape to mass store.

   VE MTSM - Verifies information from magnetic tape against mass store and prints differences.

2. Debugging features:

   RC - Replace core; enables contents of specific core locations to be printed and optionally replaced with octal data.

   PC - Print core; enables contents of specific core locations to be printed in octal.

   FC - Fill core; enables specific block of core to be filled with given octal value.

SC - Search core; enables specific block of core to be searched for given value under a mask, enables matching conditions to be printed out.

3. Real-time commands (on-line utilities only):

RP - Requests execution of program and optionally passes parameter to it.

CC - Connects program to system clock for automatic periodic execution.

DC - Disconnects program from system clock.

PT - Prints system time in hours and minutes.

RT - Replaces system time.

4. On-line core protection features:

PL - Prints core protection limits; utility functions are prevented from modifying core outside these limits.

RL - Replaces core protection limits.

Core Requirements

Approximate core requirements are shown in Table 5-1.

Table 5-1. Utility Program Core Requirements

Program	Core required (words decimal)
On-line core mass store[a,b]	512
On-line core only[a]	512-1152
Off-line core mass store	2304
Off-line core only	2560

[a] Utilize on-line drivers. Driver program core requirements are not included.

[b] Core required by on-line core mass-store utilities is independent of number of functions and is always 512 words.

DEVICE DRIVERS

RTX-16 supports an expandable library of real-time driver programs. These drivers are described in detail in separate manuals. For a list of drivers available to date, refer to Doc. No. 70181898-311, Binder Table of Contents for OP-16 (BTC1OP16).

All drivers are interrupt driven, allowing simultaneous operation of drivers and user programs.

The RTX-16 Executive makes no distinction between user programs and device drivers. Each is a program The user is wise, however, to give device drivers high priority in the Executive Program List Table (XPLT) so that I/O operations may begin quickly after requests. Because drivers are programs, they need not reside in core but may optionally reside in secondary storage when they are not being used. The secondary storage driver, of course, must always remain in core.

Any special configuration information required by a driver is included as part of the Executive Device Configuration Table (XDCT). Most drivers must also include or be linked to a communication buffer.

Because a driver is a program, it is called by the Request Program function. Drivers invariably use the Communication option in order to receive a pointer to the data buffer in the calling program. Each program using this option must specify a queueing buffer and define the length of this buffer in the Executive Configuration Module. In other words, requests for drivers may be queued to whatever extent the user desires. Likewise, each program using Communication must specify a queueing subroutine. Although the standard FIFO (first-in first-out) subroutine is usually the most convenient, any other scheme may be used if the user writes the subroutine and links it into the system.

SECTION VI
WRITING A PROGRAM

Programs to be run under RTX-16 should be written in DAP-16 or Fortran. The following discussion is applicable to both types of program. However, Fortran programs must conform to the restrictions outlined in the Section VIII.

PROGRAM HEADER

Each program run under the RTX-16 Executive must have a header with the structure illustrated in Figure 6-1. The first word (-1) indicates the top of the header. The last word before the start location is used to store the communication parameter for programs using communication. It is unused (but must be present) for programs not using communication. The intervening one to ten words are used to queue labels to be scheduled. Figure 6-2 shows the queueing of labels.

Most programs should leave room for four labels. If the user knows that four labels are too many or too few he may change the size of the block of zeros. However, the minimum header must include the -1 and two blank locations immediately before the start address.

PROGRAM NAME

Each program is identified by a unique two-character name. This name (for example, SM for the System Mass-Store Driver) is used for reference by the Executive and is defined by the user in the Configuration Module.

INTERRUPT RESPONSE CODE

Interrupt response code has been defined earlier in Section II. Any program which must respond to an interrupt (such as a device driver) must have a section of interrupt response code. The starting address of this section is transmitted to the Executive in the Connect Interrupt call.

The following restrictions apply to interrupt response code:

1. Must not execute for longer than 50 cycles
2. Cannot make any Executive calls
3. Must not contain any ENB instructions
4. Must acknowledge the interrupt with an appropriate INA, OTA, or OCP instruction.

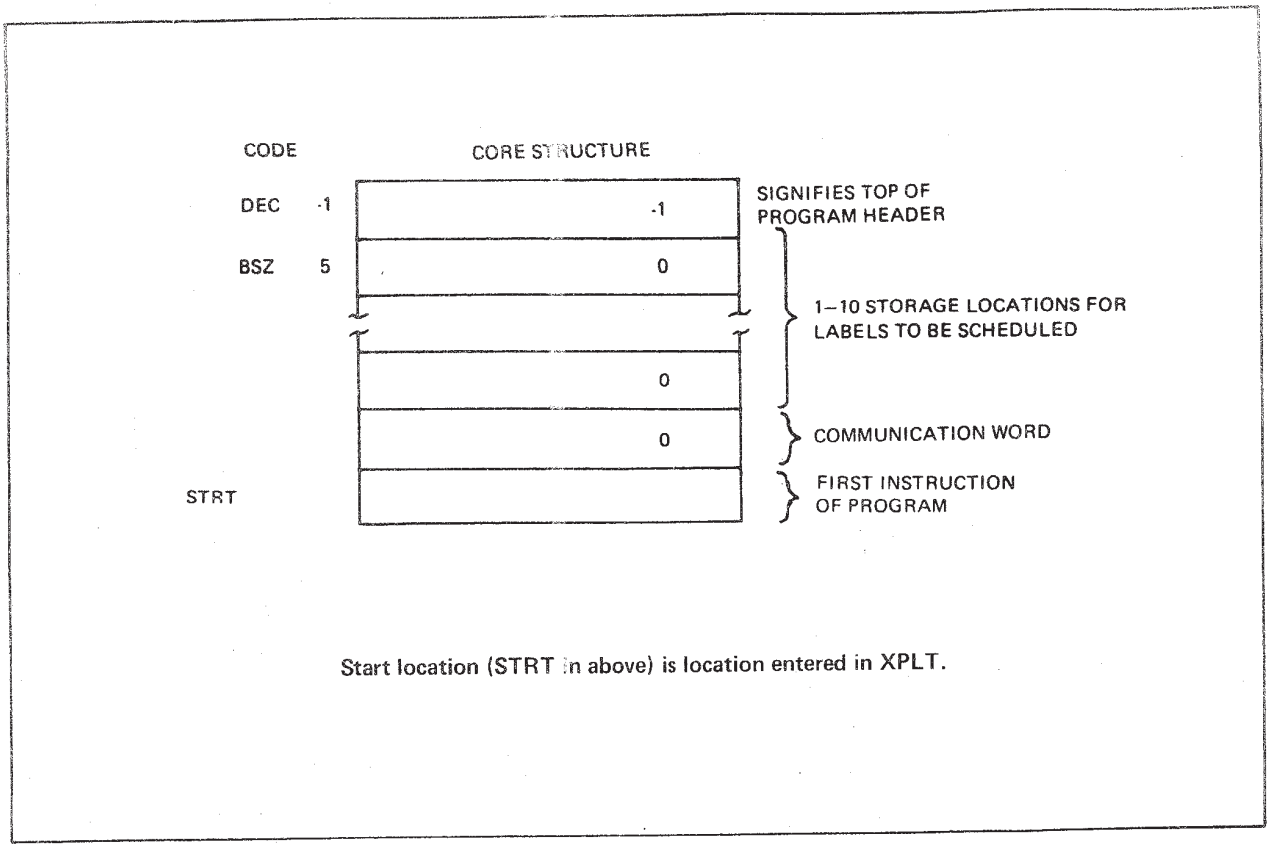


Figure 6-1. Structure of Program Header

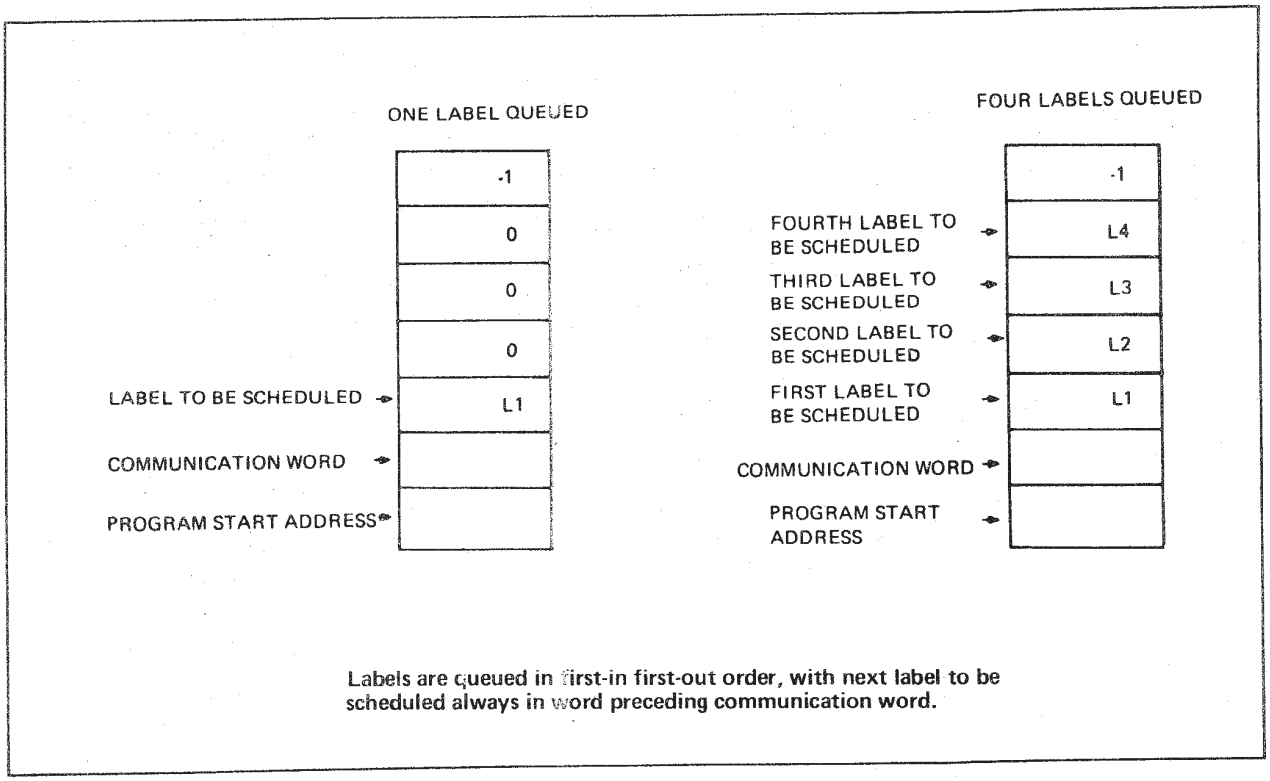


Figure 6-2. Queueing Labels in Program Header

Interrupt response code is in the subroutine format; that is, its first word is used for storing a return pointer (DAC **), and it returns by jumping through this pointer.

Interrupt response code must exit with a label in the A register if a section of non-interrupt code is to be executed in response to the interrupt. If the A register contains zero, no label will be scheduled.

A typical section of interrupt response code is shown later (Figure 6-5), starting at SDIN.

WRITING PROGRAM WITH INTERRUPT RESPONSE CODE

Most programs that respond to interrupts are driver programs. They are handled by RTX-16 exactly the same as other user programs are. Figure 6-3 shows a skeleton flow chart for any program responding to interrupts. The hexagonal boxes enclose Executive function calls.

Passing Instructions to Drivers

The communication parameter is usually a pointer to a buffer. The first locations of this buffer contain the name of the calling program, normal and error return pointers, the desired function (for multipurpose drivers), and other pertinent information. Each calling program must provide exactly the information that the driver requires. The data buffer to be used between the calling program and the driver usually follows these information words.

Bookkeeping in Drivers

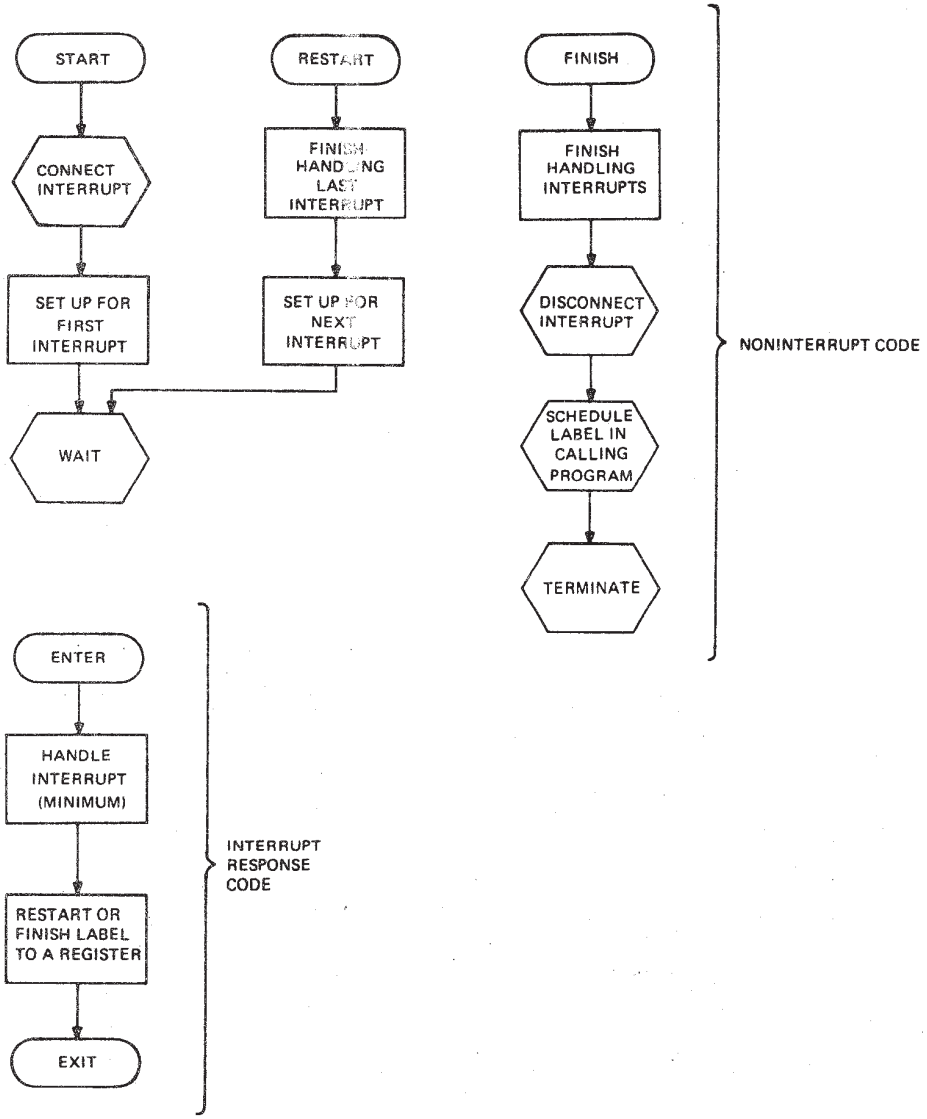
Most driver programs should provide a flag between the interrupt and noninterrupt code. This flag is set by the interrupt code and reset by the noninterrupt code. An interrupt response that finds the flag still set would know that noninterrupt code had not kept up with the interrupts. A special return could then be scheduled to signal the condition. Another flag or counter should be used to indicate when the job is done. This flag would cause control to transfer to the Finish entry in Figure 6-3.

Use of Error Print Program

Drivers should use the Error Print program to notify the operator of any errors discovered. Usually these will be device failures or overloads. Each program using the Error Print program should establish unique error codes.

SAMPLE PROGRAM WITHOUT INTERRUPT RESPONSE CODE

Figure 6-4 shows a sample user program. This program, NM, calls another, SD, to receive six ASCII characters (packed one per word). SD is described below. NM then takes these six characters and converts them to a decimal number. Blanks are ignored. The resultant number is then passed as a parameter to CC. Two negative numbers are used as error indicators to CC. Minus 1 means that SD gave NM a character other than blank or 0 through 9. Minus 2 means that SD gave an error return to NM or could not be executed.



Some indicators must be kept within program to determine when Finish entry must be scheduled rather than Restart entry.

Figure 6-3. Simplified Flow Diagram for Driver Programs

This program (NM) calls program SD for input, converts decimal number inASCII to binary number, and passes that number to CC. Provisions for errors are included.

```
*     SAMPLE RTX PROGRAM.  SEE TEXT.
*     CALLS SD TO GET ASR INPUT.  CONVERTS TO BINARY NUMBER.
*     PASSES THE BINARY NUMBER TO CC AND TERMINATES.
*
      DEC   -1                  TOP OF HEADER
      BSZ   5                   4 LABELS AND 1 UNUSED COMMUNICATION WORD
NM    LDA   =-6                 BLANK THE SIX BUFFER WORDS FIRST
      STA   TEMP                STORE COUNTER IN TEMP
      LDA   FRST                POINTER TO TOP OF BUFFER
      STA   ADRS
      LDA   ='240               BLANK
BLNK  STA*  ADRS                STORE IN BUFFER
      IRS   ADRS                UPDATE POINTER
      IRS   TEMP                UPDATE INDEX
      JMP   BLNK                LOOP
*
*     REQUEST PROGRAM SD TO GET NUMBER
      JST*  XLNK
      DEC   1                   FUNCTION 1, REQUEST PROGRAM
      BCI   1,SD                PROGRAM SD REQUESTED
      DAC   ERR                 RETURN ADDRESS IF EXEC CAN'T EXECUTE SD
      DAC   BUF                 PARAMETER TO BE COMMUNICATED TO SD
*
*     EXECUTE WAIT FUNCTION
      JST*  XLNK
      DEC   8                   FUNCTION 8, WAIT
*
*     PROGRAM SD SCHEDULES NM TO START HERE IF NO ERROR
NORM  LDX   =-6                 COUNTER OF WORDS PROCESSED
      LDA   LAST                POINTER TO END OF BUFFER
      STA   ADRS                STORE IN TEMPORARY LOCATION
      CRA
      STA   VALU                INITIALIZE VALU TO ZERO
      STA   POWR                INDICATES TEN TO THE ZERO POWER
LOOP  LDA*  ADRS                PICK UP WORD FROM BUFFER
      SUB   ='260               SUBTRACT ASCII ZERO
      SPL                       TEST SIGN OF RESULT
      JMP   NEG                 -NEGATIVE MAY MEAN AN ERROR
      CAS   =9                  -POSITIVE, CHECK FOR 0-9
      JMP   VM1                 >9, JUMP TO HANDLE
      NOP                       =9, OKAY
      STA   TEMP                <9, SAVE VALUE FOR MULTIPLICATION
      LDA   POWR                WHAT POWER OF TEN IS TO BE USED?
MLP   STA   PWR1                STORE POWER OF TEN YET TO BE USED
      SNZ                       DONE YET?
      JMP   ADD                 -YES, ADD IT TO VALU
      LDA   TEMP                VALUE TO BE MULTIPLIED
      LGL   2                   MULTIPLY BY FOUR
      ADD   TEMP                (4 X TEMP) + TEMP = 5 X TEMP
      LGL   1                   10 X TEMP
      STA   TEMP                NEW VALUE
```

Figure 6-4. Example of User Program (Part 1 of 2)

```
      LDA   PWR1                OLD POWER OF TEN
      SUB   =1                  DECREMENT TO NEW POWER OF TEN
      JMP   MLP                 MULTIPLY AGAIN OR EXIT
ADD   LDA   TEMP                GET VALUE
      ADD   VALU                ADD TO VALU
      STA   VALU                SAVE NEW VALUE OF VALU
      IRS   POWR                POWER OF TEN
NDLP  LDA   ADRS                BUFFER POINTER
      SUB   =-1                 POINT TO PREVIOUS WORD
      STA   ADRS                STORE NEW VALUE OF POINTER
      IRS   0                   UPDATE COUNTER
      JMP   LOOP                WORK ON NEXT WORD
*
*     NOW REQUEST PROGRAM CC USING VALU AS PARAMETER
RPCC  JST*  XLNK
      DEC   1                   FUNCTION 1, REQUEST PROGRAM
      BCI   1,CC                REQUESTED PROGRAM IS CC
      DAC   TERM                NORMAL RETURN EVEN IF AN ERROR
VALU  BSZ   1                   VALU IS THE PARAMETER PASSED TO CC
*
*     NOW TERMINATE
TERM  JST*  XLNK
      DEC   7                   FUNCTION 7, TERMINATE
*
*     HERE TO HANDLE CHARACTERS < ASCII ZERO
NEG   ERA   ='-20               CHECK FOR BLANK
      SNZ
      JMP   NDLP                IGNORE BLANKS
VM1   LDA   =-1                 AN ERROR IF NOT BLANK OR 0-9
      STA   VALU                STORE IN VALU
      JMP   RPCC                REQUEST CC WITH VALU = -1
*
*     IF SD DETECTS AN ERROR IT SCHEDULES NM TO START HERE
ERR   LDA   =-2                 SIGNAL TO CC THAT SD MADE AN ERROR
      STA   VALU                STORE IN VALU
      JMP   RPCC                REQUEST CC WITH VALU = -2
*
*     HERE IS THE BUFFER THAT SD OPERATES UPON
BUF   BCI   1,NM                FIRST WORD IS PROGRAM NAME
      DAC   NORM                SECOND WORD IS NORMAL RETURN LABEL
      DAC   ERR                 THIRD WORD IS ERROR RETURN LABEL
      BSS   6                   NEXT SIX WORDS ARE FOR DATA
*
LAST  DAC   *-1                 POINTER TO LAST WORD IN BUFFER
FRST  DAC   BUF+3               POINTER TO FIRST WORD IN BUFFER
ADRS  DAC   **                  STORAGE FOR BUFFER POINTER
TEMP  BSZ   1                   STORAGE FOR VALUE DURING MULTIPLICATION
POWR  BSZ   1                   STORAGE FOR POWER OF TEN
PWR1  BSZ   1                   TEMPORARY STORAGE FOR POWER OF TEN
XLNK  OCT   101001              INDIRECT POINTER TO FUNCTION ENTRANCE
      END
```

Figure 6-4. Example of User Program (Part 2 of 2)

An actual program to run under RTX-16 would probably include more error checking and more toleration of different inputs by SD. This example shows proper use of the Executive functions.

SAMPLE PROGRAM WITH INTERRUPT RESPONSE CODE

Figure 6-5 shows program SD, the driver program used by the previous example. This program connects the ASR and waits for input. It terminates after either the input of six characters or the input of a carriage return. The characters are passed to the calling program (for example, NM) without being packed.

Two kinds of errors are detected by SD. Both cause a return to the calling program's error label. The more serious error is caused by a busy ASR after the interrupt has been connected. The Error Print routine is called in this case and given the error number '333. The less serious error is caused by inability to connect the interrupt. The Error Print routine is called by the Executive in this case.

This example illustrates the proper use of function calls, calls to the Error Print program, and interrupt and noninterrupt code. All the necessary functions of a driver are performed.

This is program SD, called by NM. Up to six characters are read from ASR before program returns.

```
*     SAMPLE RTX DRIVER.  SEE TEXT.
*     USER BUFFER:  FIRST WORD  - NAME OF CALLING PROGRAM
*                   SECOND WORD - NORMAL RETURN LABEL
*                   THIRD WORD  - ERROR RETURN LABEL
*          FOURTH - NINTH WORDS - DATA BUFFER
*
      DEC   -1                  TOP OF HEADER
      BSZ   5                   4 LABELS AND 1 COMMUNICATION WORD
SD    LDA*  SD-1                PICK UP FIRST WORD IN BUFFER (PROGRAM NAME)

      STA   NAME                STORE IN SCHEDULE LABEL CALL
      IRS   SD-1                INCREMENT BUFFER POINTER
      LDA*  SD-1                PICK UP SECOND WORD IN BUFFER (NORMAL
*                                 RETURN ADDRESS TO CALLING PROGRAM)
      STA   ADDR                STORE IN SCHEDULE LABEL CALL
      IRS   SD-1                INCREMENT BUFFER POINTER
      LDA*  SD-1                PICK UP THIRD WORD IN BUFFER (ERROR
*                                 RETURN ADDRESS TO CALLING PROGRAM)
      STA   ERRL                STORE IN TEMPORARY STORAGE
      IRS   SD-1                INCREMENT BUFFER POINTER TO POINT
*                                 TO FIRST DATA LOCATION
      LDA   =-6                 MAXIMUM NUMBER OF CHARACTERS ACCEPTED
      STA   CCNT                STORE IN CHARACTER COUNTER
*
*     CONNECT INTERRUPT CALL
      JST*  XLNK
      DEC   5                   FUNCTION 5, CONNECT INTERRUPT
      DEC   4                   INTERRUPT REFERENCE NUMBER FOR ASR
      DAC   ERRA                POINTER TO ERROR RETURN FOR THIS CALL
      DAC   SDIN                POINTER TO START OF INTERRUPT
*                                 RESPONSE CODE
      SKS   '0104               IS ASR BUSY?
      JMP   ERR3                AN ERROR IF IT IS
      OCP   '0004               ENABLE ASR FOR INPUT
*
*     WAIT FUNCTION CALL
WAIT  JST*  XLNK
      DEC   8
*
*     INTERRUPT RESPONSE CODE STARTS HERE
SDIN  DAC   **                  EXECUTIVE STARTS INTERRUPT RESPONSE CODE
*                                 WITH A JST.  SAVE RETURN ADDRESS HERE.
      INA   '1004               CLEAR A REGISTER AND INPUT CHARACTER
      NOP                       ASR WILL BE READY SO INA WILL SKIP
      STA   TEMP                SAVE CHARACTER FOR NON-INTERRUPT CODE
      LDA   HDAC                POINTER TO NON-INTERRUPT PART OF PROGRAM
      JMP*  SDIN                RETURN TO EXECUTIVE.  TOTAL TIME 7 CYCLES.
*
*     HERE THE NON-INTERRUPT PORTION STARTS
HNDL  LDA   TEMP                GET CHARACTER
      ERA   CR                  IS IT A CARRIAGE RETURN?
      SNZ                       ZERO MEANS CARRIAGE RETURN
```

Figure 6-5. Example of Driver Program (Part 1 of 2)

```
      JMP   TERM                JUMP TO TERMINATING ROUTINE
*
      LDA   TEMP                NOT A CARRIAGE RETURN; GET CHARACTER AGAIN
      STA*  SD-1                PUT IN CALLING PROGRAM BUFFER
      IRS   SD-1                INCREMENT BUFFER POINTER
      IRS   CCNT                INCREMENT CHARACTER COUNT
      JMP   WAIT                NOT DONE, EXECUTE WAIT FUNCTION
      JMP   TERM                DONE, JUMP TO TERMINATING ROUTINE
*
ERRB  LDA   ='333               PROGRAM'S UNIQUE ERROR CODE
      LDX   NAME                NAME OF CALLING PROGRAM
      INH                       MUST INHIBIT INTERRUPTS FOR ERROR PRINT
      JST*  XEPR                CALL ERROR PRINT ROUTINE
      ENB                       REENABLE INTERRUPTS
*     FALL THROUGH TO ERROR RETURN TO CALLING PROGRAM
*
ERRA  LDA   ERRL                USER'S ERROR RETURN POINT
      STA   ADDR                STORE IN SCHEDULE LABEL CALL
*     FALL THROUGH TO TERMINATING ROUTINE
*
*     TERMINATING ROUTINE; FIRST DISCONNECT INTERRUPT
TERM  JST*  XLNK
      DEC   6                   FUNCTION 6, DISCONNECT INTERRUPT
      DEC   4                   INTERRUPT REFERENCE NUMBER FOR ASR
*
      LDA   ADDR                USERS'S RETURN ADDRESS
      SNZ                       ZERO MEANS DO NOT SCHEDULE LABEL
      JMP   LAST                IF ZERO GO DIRECTLY TO TERMINATE CALL
*
*     SCHEDULE LABEL CALL FOR RETURN TO USER
      JST*  XLNK
      DEC   2                   FUNCTION 2, SCHEDULE LABEL
NAME  ***   **                  PROGRAM NAME GOES HERE
      DAC   LAST                TAKE NORMAL RETURN EVEN IF ERROR
ADDR  ***   **                  ADDRESS TO BE SCHEDULED GOES HERE
*                                 (NORMAL RETURN UNLESS AN ERROR ABOVE)
*
*     TERMINATE FUNCTION CALL
LAST  JST*  XLNK
      DEC   7                   FUNCTION 7, TERMINATE
*
ERRL  BSZ   1                   STORAGE FOR ERROR RETURN ADDRESS
CCNT  BSZ   1                   CHARACTER COUNTER
TEMP  BSZ   1                   STORAGE FOR CHARACTER BETWEEN
*                                 INTERRUPT AND NON-INTERRUPT CODE
CR    OCT   215                 CARRIAGE RETURN CHARACTER
HDAC  DAC   HNDL                POINTER TO BEGINNING OF INTERRUPT CODE
XLNK  OCT   101001              INDIRECT POINTER TO FUNCTION ENTRANCE
XEPR  OCT   101016              INDIRECT POINTER TO ERROR PRINT ENTRANCE
      END
```

Figure 6-5. Example of Driver Program (Part 2 of 2)

SECTION VII
SYSTEM BUILDING

LAYOUT

Before an Op-16 system is built, its layout must be established. As mentioned earlier, the Basic Executive and its support modules have a fixed location. However, the position of the end of this section depends on the length of the Configuration Module. Use the size estimation formula discussed in Section IV, Configuration Module, in order to know where to end this module.

Except for the dedicated locations below '64, sector zero is entirely free for cross-sector links or data tables. Systems with the Relocatable Base Sector option may use another sector for the cross-sector links of programs high in core. (See Special Capabilities of RTX-16, Section VIII.)

Each program must be assigned a fixed location in core. For systems with mass storage, extreme care must be taken to see that programs which occupy the same core area will never need to be in core at the same time. Because the mass storage drivers operate upon 128-word segments, programs that reside on mass storage must begin on segment boundaries (using ORG).

Figure 7-1 shows the core-storage layout of a small Op-16 system with 8K of memory and no secondary storage. Figure 7-2 shows a larger system with 12K of memory and a mass storage device.

The user must also layout the segments on the mass storage. There are no special requirements. He should reserve an area capable of holding the entire system for easy initialization.

Once the core- and mass-storage layout is completed, the Configuration Module must be assembled. The rules given in Section IV and the Sample Configuration Module (Table 4-3) will help with this. When a proper module that assembles with no errors is created, the system may be built.

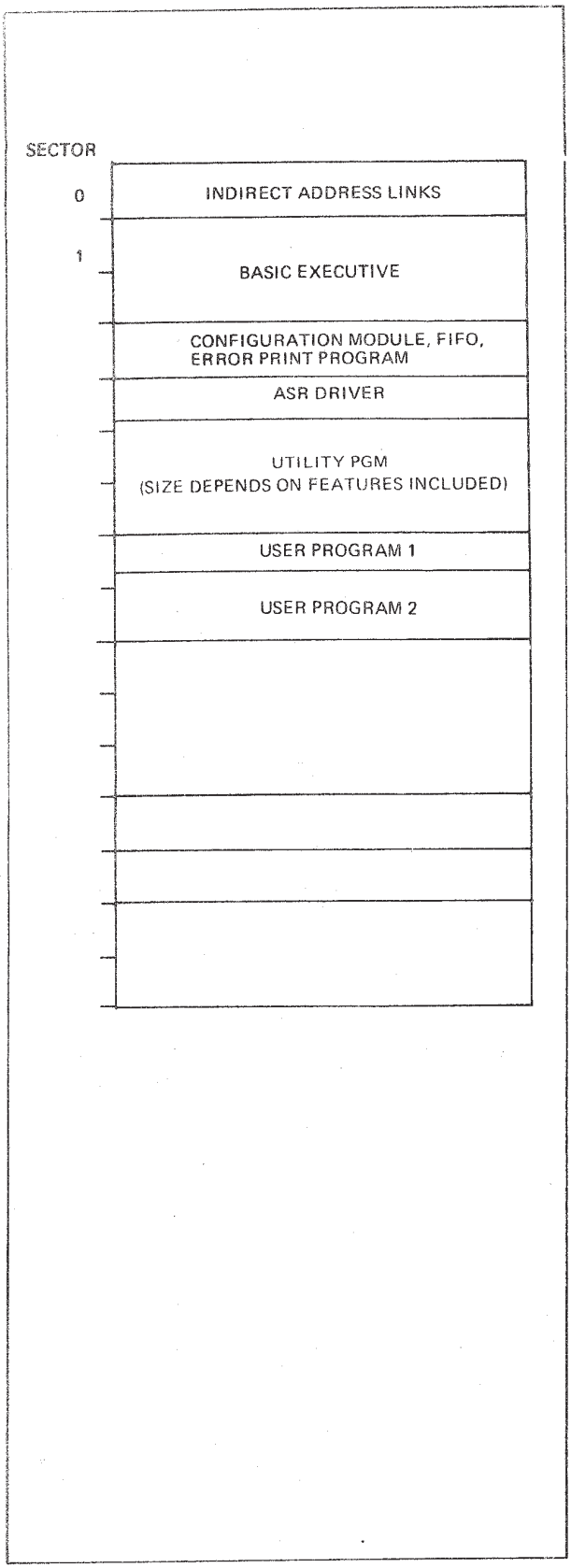


Figure 7-1. 8K Core-Only System

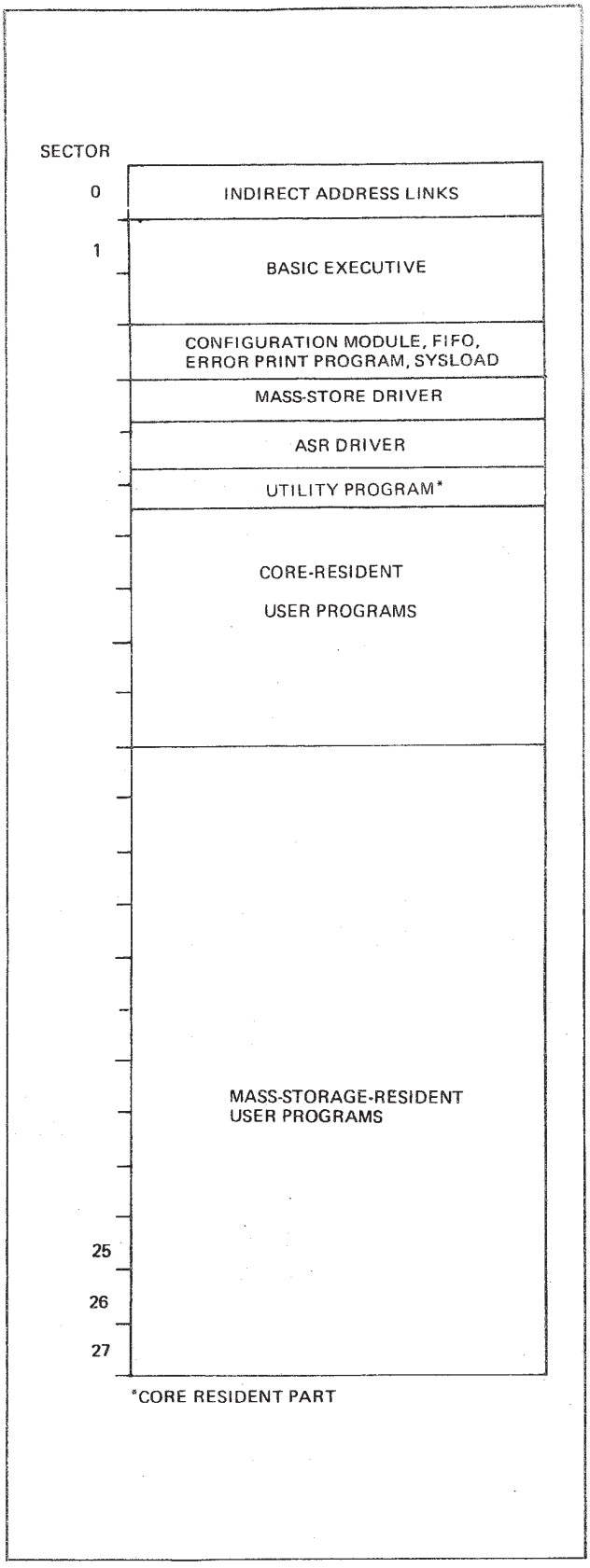


Figure 7-2. 12K Core/Mass-Store System

The user must be familiar with two or three off-line support programs. PAL-AP and LDR=APM are described in the 316/516 Operators' Guide, Doc. No. 70130072165. The OP-16 utility program is discussed in Section V of this manual.

BUILDING CORE-ONLY SYSTEM

When an RTX-16 system is first constructed, object tapes produced by the assembler are used, and LDR-APM is used to produce executable code in core. The PAL-AP utility program allows the user to punch the contents of core onto paper tape in a self-loading form. The punching of self-loading tapes enables the user to have a permanent copy of his system.

Figure 7-3 shows the loading process schematically. Figure 7-4 shows two core maps for an actual RTX-16 system without the Fortran capability. Figure 7-5 shows an RTX-16 system with the Fortran capability.

Building RTX-16 Executive

The RTX-16 Executive is supplied by Honeywell as three object tapes (Basic Executive, FIFO, and Error Print program). The Configuration Module must be written by the user and assembled, using the DAP-16 Assembler to provide an object tape. Utility programs are to be configured and loaded in accordance with the procedures described in Doc. No. 70130072519, OP-16 Utility Programs. The following steps are used to load and punch the Basic Executive and the Configuration Module.

1. Load the loading program LDR-APM into the four highest sectors of memory (in the example in Figure 7-1, sectors 14 through 17 octal). This program is provided as a self-loading tape.
2. Load the utility program that punches self-loading tapes (PAL-AP) into memory, starting at location 7000 octal.
3. Use LDR-APM to load the object tapes of the RTX-16 Basic Executive, Configuration Module, queueing routine FIFO, and Error Print program. Load the system-level Fortran Run-Time routines if required.
4. Use PAL-AP to punch a self-loading tape of locations 1000 through the end of the section just loaded.

Building Programs for RTX-16

Programs to be run under RTX-16 normally will be in the form of object tapes when the system is being built. In order to load all of memory with RTX-16 and its programs, the system must be built in stages which are punched out as self-loading tapes and finally all loaded into place.

BUILDING PROGRAMS IN SECTORS 4 THROUGH 7

The user may build programs in the area between the end of the Configuration Module (probably in sector 3 or 4) and location 7537. Locations 7540 through 7777 may not be used

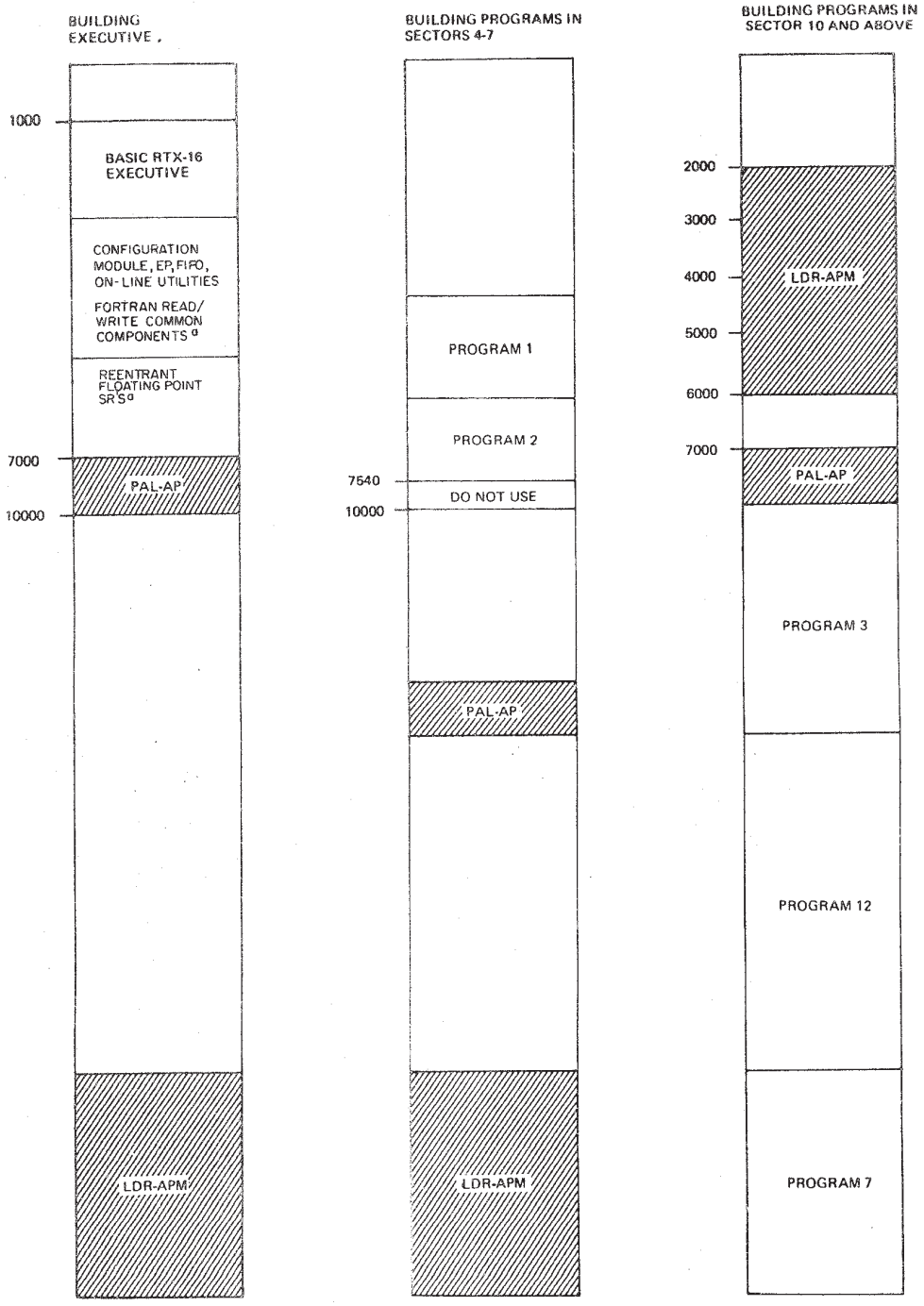


Figure 7-3. Building Core-Only Op-16 System

PART 1

```
*LOW    01000
*START  01000
*HIGH   05066
*NAMES  33256
*COMN   77777
*BASE   03134
*BASE   02776
*BASE   01772
EXEC    01000
XPIC    01011
XPEP    01020
SYSF    01021**
XSSA    01023
OPTRAC  01037**
XHPT    01041**
INPI    01044
SC      01103
IH      01343
IH20    01372
IHPI    01507
IH40    01513
IHSC    01521
CLH     01525
CL      01527
XLNK    02000
FE10    02057
FE20    02104
FERR    02123
XSPI    02131
LB      02134
RP      02167
RPRO    02174
SL      02264
SLBL    02275
CC      02342
DC      02444
CI      02516
DI      02626
TE      02641
WA      03000
AI      03063
```

PART 2

```
FIFO    03134
EP      03323
ED      03432
LO      03603
MDRQ    03714
XPLT    04000
ER      04022
MSD     04034
XPET    04226
LOI     04232
KBI     04254
XIDT    04256
CLK2    04265
CLK3    04266
XID2    04411
XIDI    04423
XPCT    04430
XCUT    04674
XIVT    04731
XLPT    04770
XFET    05013
XDCT    05024
XSPT    05057
XPFP    05060
XEXA    05061
XINT    05064

MR
```

Figure 7-4. Memory Map of RTX-16 Without Fortran Capability

PART 1

```
*LOW    01000
*START  01000
*HIGH   11706
*NAMES  33143
*COMN   77777
*BASE   03134
*BASE   02776
*BASE   06675
*BASE   07727
*BASE   10725
*BASE   11254
*BASE   01772
EXEC    01000
XPIC    01011
XPEP    01020
XSSA    01023
OPTRAC  01037**
XHPT    01041**
INPI    01044
SC      01103
IH      01343
IH20    01372
IHPI    01507
IH40    01513
IHSC    01521
CLH     01525
CL      01527
XLNK    02000
FE10    02057
FE20    02104
FERR    02123
XSPI    02131
LB      02134
RP      02167
RPRO    02174
SL      02264
SLBL    02275
CC      02342
DC      02444
CI      02516
DI      02626
TE      02641
WA      03000
AI      03063
```

PART 2

```
FIFO    03134
EP      03323
ED      03432
LO      03603
MDRQ    03714
XPLT    04000
ER      04022
MSD     04034
XPET    04226
LOI     04232
KBI     04254
XIDT    04256
CLK2    04265
CLK3    04266
XID2    04411
XID1    04423
XPCT    04430
XCUT    04674
XIVT    04731
XLPT    04770
XFET    05013
XDCT    05024
XSPT    05057
XPFP    05060
XEXA    05061
XINT    05064
SYSF    05066
CONV    05100
M$2J    05140
D$2J    05367
F$ER    05676
N$22    05716
OPED    06000
FDLS    11000
F$AT    11260
S$2J    11346
A$2J    11364
JBASE   11670
JBAS    11670

MR
```

Figure 7-5. Memory Map of RTX-16 With Fortran Capability

by programs (although they may be used later for data or as buffers), because they are used by PAL-AP. The following steps are used to load and punch programs in this area.

1. Load the loading program LDR-APM into the four highest sectors of memory (in the example in Figure 7-1, sectors 14 through 17 octal). This program is provided as a self-loading tape.
2. Load the utility program that punches self-loading tapes (PAL-AP) into memory, starting at the beginning of any sector, except 4 through 7 or the five uppermost.
3. Load all user programs and subroutines that go in this area, using LDR-APM.
4. Use PAL-AP to punch a self-loading tape of this entire portion of memory. If desired, PAL-AP can also be used to punch out individual programs.

BUILDING PROGRAMS IN SECTORS 10 AND ABOVE

To build programs in these sectors, LDR-APM and PAL-AP must be in sectors 1 through 7. After the programs have been built in the upper sectors, the self-loading tapes of the lower sectors may be loaded, and the entire system will be in core. The following steps are used to load and punch programs in this part of core.

1. Load the loading program LDR-APM into memory, starting at location 2000 octal.
2. Load PAL-AP into memory, starting at location 7000 octal.
3. Load all programs and subroutines that go in Sector 10 and above, using LDR-APM.
4. Use PAL-AP to punch a self-loading tape of this entire portion of memory. If desired, PAL-AP can also be used to punch out individual programs.

BUILDING CORE MASS-STORAGE SYSTEM

The tools necessary to build a core mass-store system are LDR-APM, PAL-AP, and a suitable mass-store utility program (MSU, Figure 7-6).

The purpose of the loader, LDR-APM, is described in Doc. No. 70130072165, 316/516 Operators' Guide. The purpose of MSU is to transfer the programs loaded into core by LDR-APM onto the mass-store device as specified by the programmer via the ASR keyboard. The preconfigured basic off-line utility programs (OFLUT-2 and -3, each supporting a different mass-store device) may readily be used for this purpose; however, these programs are core resident and occupy a relatively large amount of core.

In order to reduce core requirements and include additional features (debugging, magnetic tape support, etc.), it is recommended that the user build, using the components provided in the utility program library, his own off-line core-mass-store utilities in which several components are made mass-store resident.

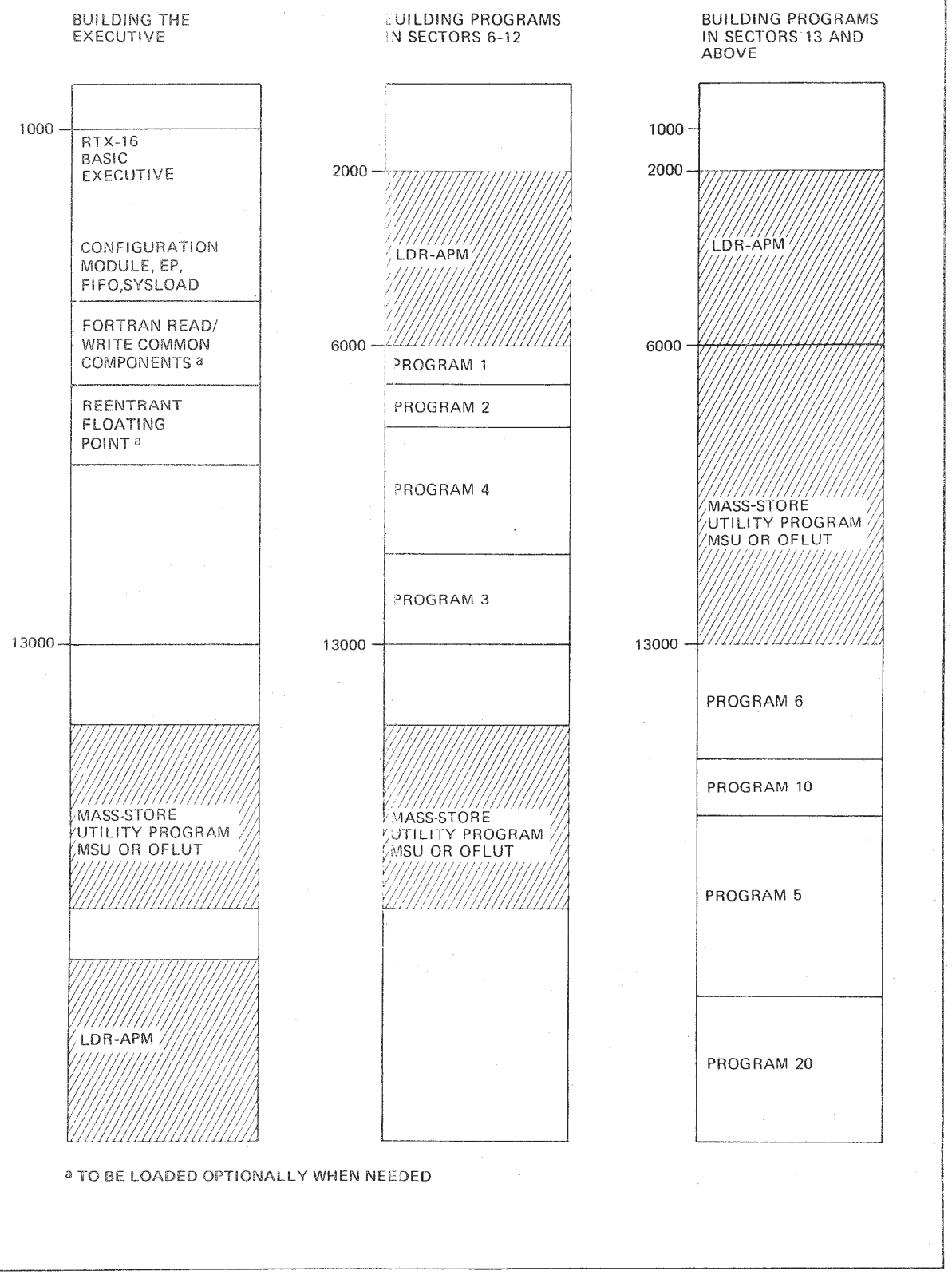


Figure 7-6. Building Core Mass-Store System

Assuming that LDR-APM and OFLUT or a user-built expanded OFLUT are available, the following building procedure is recommended.

Building RTX-16 Executive

The following steps are used to load and store the Basic Executive, Configuration module, Error Print program, FIFO, and mandatory core-resident mass-store driver.

1. Load LDR-APM into four highest core sectors (14 through 17 octal in Figure 7-1).
2. Using LDR-APM, load PAL-AP (which punches self-loading tape) at convenient location, for example, 7000 octal.
3. If basic mass-store utility OFLUT is to be used for storing programs on mass store, use LDR-APM to load OFLUT into any memory sector, except uppermost five (where LDR-APM resides) and lowermost seven (where Executive will be loaded). Use PAL-AP to punch self-loading tape of OFLUT for later convenience.

   If it is necessary to build a partially mass-store resident off-line mass-store utility program, which includes debugging features and support for additional I/O devices (MSU), use LDR-APM to load OFLUT into any memory sector, except uppermost ten. In this case, OFLUT will be used to build MSU. Use PAL-AP to punch self-loading tape of OFLUT for later convenience.
4. Build MSU, using components available from OP-16 Utility Program Library, as described in the OP-16 Utility Program Manual, Doc. No. 70130072519.
5. Use LDR-APM to load object tapes of RTX-16 Basic Executive, Configuration Module, Error Print Program, FIFO, and SYSLOAD. Load the system level Fortran run-time routines if required. Use OFLUT or MSU to store these core-resident components on mass store, if bootstrap procedures are planned for application. If appropriate features are included in MSU, it may also be used to punch paper tapes from core or from mass store which, while not self loading, may be loaded using MSU.

Building Programs for RTX-16

Programs to be run under RTX-16 normally will be in the form of object tapes when the system is being built. When using a mass-storage device as part of RTX-16, several programs will often be designed to run in the same core location. Thus, the process of loading and saving may be somewhat intricate. In order not to waste core space occupied by the Loader and Mass-Store Utility programs, the following two-stage procedure is recommended.

BUILDING PROGRAMS IN SECTORS 6 THROUGH 12

The user may build programs in the area between the end of the Mass-Store Driver (probably in sector 5) and the end of sector 12. The following steps are used to load and store programs in this area.

1. Load the loading program LDR-APM into sectors 2 through 5 of memory.
2. Load OFLUT or MSU into any memory sector, except 0 through 12.

3. Use LDR-APM to load each program that is to be drum-resident. When the program is loaded, store it on the mass store, using the utility program. Programs may also be punched on paper tape.

4. Use LDR-APM to load each program that is not to be drum-resident. Store each program on the mass store, or use OFLUT or MSU to punch a paper tape of each program. Careful layout of core and drum storage will enable the user to store large blocks in one operation.

BUILDING PROGRAMS IN SECTORS 13 AND ABOVE

To build programs in this area, LDR-APM and MSU must be in sectors 0 through 12. The following steps are used to load and punch programs in this part of core.

1. Load the loading program LDR-APM into sectors 2 through 5 of memory.

2. Build MSU into memory beginning at sector 6.

3. Use LDR-APM to load each program that is to be drum-resident. When the program is loaded, store it on the mass store, using the utility program. Programs may also be punched on paper tape.

4. Use LDR-APM to load each program that is not to be drum-resident. Store each program on the mass store, or use OFLUT or MSU to punch a paper tape of each program. Careful layout of core and drum storage will enable the user to store large blocks in one operation.

SYSTEM INITIALIZATION

The entire core-resident part of the system must be loaded. Tapes punched by PAL-AP are self-loading as long as the Key-In Loader in locations '1 through '17 has not been tampered with. Tapes produced by the Mass-Store Utility program require that it be present in core to be loaded. Then start execution of the Executive at location 1000 octal. RTX-16 will be fully operational. If the computer is stopped for any reason, the Basic Executive and Configuration Module should be reloaded before restarting at location 1000.

The RTX-16 Executive goes through the following initializing routine before starting normal operation.

1. The power failure response address from XSPT is placed in location '60 (power failure interrupt location).

2. A predefined system stop address (location '1023) is placed in location '62 in order to stop the computer in case of a memory lockout violation (not normally expected).

3. The Real-Time Clock is started. At the first clock cycle, the time will be 00:00.

4. A pointer to the Executive Interrupt Handler is placed in location '63 (standard interrupt location).

5. The interrrupt mask is established (the Real-Time Clock and ASR are the only interrupts enabled at initialization).

6. The ASR is put in the input mode.

SECTION VIII
SPECIAL CAPABILITIES OF RTX-16

RELOCATED BASE SECTOR

The Models 316 and 516 use a sectorized addressing scheme. This allows any instruction word to have direct access to any word in the same sector and any word in the base sector (sector zero). The base sector ordinarily is used by the loader to store indirect address links generated by cross-sector references.

The Model 516 Memory Lockout option or the Model 316 Relocatable Base Sector option allows any sector to be specified temporarily as the base sector. References to the base sector then specify the relocated sector. This option is used to allow different programs to store their indirect links in different sectors. The availability and use of this hardware is specified by the first word of the XSPT table in the Configuration Module; nonzero means present, zero means absent.

RTX-16 leaves all of sector zero in core permanently. This means that it may be filled with links for programs, some of which may reside on mass storage. Programs for which there is no room in sector zero must either keep all the indirect address words in their own sector (using the SETB pseudo-operation) or use the relocatable base sector option. The following subsections describe the use of the relocated base sector option under these conditions.

Programs Using Sector Zero

Any number of programs may use this sector. Locations '64 to '777 are available for indirect addresses.

All programs that respond to interrupts (including all RTX-16-supplied drivers) must use sector zero for indirect links.

Programs Using Relocated Base Sector

Programs that use the relocated base sector option must set the appropriate bits in their XPLT entries; bit 15 of the option word to specify that the communication word is present, and bits 2 through 7 of the communication word to specify the sector.

WRITING SPECIAL QUEUEING SUBROUTINE

Before the user attempts to write his own queueing subroutine, he should examine and thoroughly understand the operation of FIFO, the subroutine provided with RTX-16.

When the subroutine is called, it will have the following information available to it.

1. Indirectly through XLNK — a pointer to the function call and a flag indicating whether this is to be a fetch or store operation. This flag is bit 1 — if it is set (1), it means a store operation is required; if it is reset (0) it means a fetch operation is required.
2. XPCP — the pointer to the beginning of the Executive Program Communication Table.
3. XPLP — the pointer to the requesting program's entry in the Executive Program List Table.
4. The contents of the A register — an index to the proper communication buffer in XPCT.
5. The contents of the X register — 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 both normal and error returns and to reset the "communications active" bit in the program's entry in XPLT 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. To enable users to incorporate new functions into their systems, the following information is provided. Certain function numbers will, from time to time, be used for new standard functions.

Control Interfaces

ENTRY FROM USER PROGRAM TO FUNCTION HANDLER

The standard calling sequence is as follows.

(L)	JST*	XLNK	Function Handler entry
(L+1)	DEC	< function no.>	
(L+2)	Parameter		(Optional)
(L+3)	DAC	< address of error code>	(Optional)

If the name or number of another program is specified, this must be specified in location (L+2) of the calling sequence, using a BCI 1, < program name> or a DEC < program number> pseudo-operation respectively.

The address of the user's error code, if any, must be specified in location (L+3) of the calling sequence.

Further parameters as required by particular system functions may be specified in locations (L+4), (L+5), etc. of the calling sequence, and one further parameter may be passed in the A register.

ENTRY TO SYSTEM FUNCTION FROM FUNCTION HANDLER

The Function Handler enters the required function at its first instruction via an indirect JMP instruction through the XFET pointer. The Scheduler is disabled and the J base set to zero.

RETURN FROM SYSTEM FUNCTION TO FUNCTION HANDLER

Following execution of the required system function the function must return to the Function Handler at label FE10 via a JMP FE10 instruction if no error occurred, and to label FE20 instruction if an error occurred.

RETURN TO USER PROGRAM FROM FUNCTION HANDLER

Following execution of the system function the Function Handler enables the Scheduler, restores the J base (if in use), and returns control to the appropriate location in the user's calling sequence via an indirect JMP instruction.

RETURN TO SCHEDULER FROM FUNCTION HANDLER

The Function Handler, following execution of the required system function, checks the flag FE50. This is set by system Functions 1 and 2, Request Program and Schedule Label, when a program of higher priority than the one currently running is requested or has a label scheduled. The flag 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 by the Scheduler.

Data Interface

ENTRY FROM USER PROGRAM TO FUNCTION HANDLER

When the user program enters the Function Handler, any input data required by the Function Handler and the function is contained in the calling sequence. A parameter may also be passed in the A register.

The function number must be in location (L+1) of the calling sequence, and the error return address, if any, must be in location (L+3) of the calling sequence.

Any program specified must be specified by a BCI ← 1, < program name> or a < program or a < program number> pseudo-operation in location (L+2) of the calling sequence.

ENTRY TO FUNCTION FROM FUNCTION HANDLER

On entry to the required function, the A register is restored to the value it contained on entry to the Function Handler from the user program.

If the required function is specified as one requiring a program name search, then the Function Handler will have located the required program and set system variable XSP1 to the address of the first word of the associated XPLT entry.

Location XLNK will contain a pointer to the second word of the function call (that is, to the function number).

EXIT FROM FUNCTION TO FUNCTION HANDLER

When the required function terminates execution and returns to label FE10 of the Function Handler, the B register may contain a parameter to be passed to the user. This parameter will be 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 of the Function Handler is taken, the A register must contain the appropriate error code.

EXIT BACK TO USER PROGRAM FROM FUNCTION HANDLER

When the Function Handler returns to the user program following execution of the required system function, 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 contains the relevant error code. The contents of the other registers are unspecified, and the calling sequence locations are unaltered.

EXIT BACK TO SCHEDULER FROM FUNCTION HANDLER

When the Function Handler returns to the Scheduler, no data is required by the Scheduler other than the return address, the keys, and the contents of the A register which are saved in the XIVT 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 cycles, 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:

```
EXT     SLBL
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 XPET entry will be set in XSP1 by the Function Handler, if bit 1 of the relevant pointer in XFET 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. Functions should be loaded with the Executive Module and Configuration Module to ensure that all the necessary links are completed successfully.

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

A register must contain [(address of communication parameter) -3] if the requested program uses the Communication option.

B register must contain the address of the XPET entry for the requested program:

```
EXT RPRO
JST  RPRO
Error return
Normal return
```

On return, XSP1 will contain a pointer to the XPET entry for the requested program.

The error return will be made if the requested program's communication buffer is full.

USER INITIALIZATION ROUTINES

The RTX-16 Initialization routine, INP1 in the Executive, permits the running of user initialization routines after the preliminary system initialization has been performed. This enables systems running under RTX-16 to perform once-only system initialization in a simple, straight-forward manner, without having to resort to an 'Initialization Program', which would mean an extra program in the system that would be run only once.

Description

User initialization routines are written as normal closed subroutines which may be located anywhere in core (they could be located in a buffer area so that they would be

overwritten after they have outlived their usefulness). There may be any number of these closed subroutines or none. The method employed requires one word in the Configuration Module.

After performing its normal functions and before entering the Scheduler, INP1 does an indirect subroutine jump to a location XINT in the Configuration Module. XINT (Executive Initialization Table) contains a single pointer to a series of JST's to the individual initialization subroutines, called the initialization control subroutine. This initialization control subroutine and the initialization subroutines may be located anywhere in core.

In the event that no user initialization is required, XINT is made to contain a pointer to the Scheduler.

Programming Notes

CONFIGURATION MODULE

The Configuration Module must contain the one-word item XINT, which is a pointer to the initialization control subroutine. Thus the Configuration Module must contain:

```
*
*  EXECUTIVE INITIALIZATION TABLE
        SUBR   XINT
XINT  <pointer to initialization control subroutine>
```

INITIALIZATION CONTROL SUBROUTINE

This is a closed subroutine which consists exclusively of JST's to the individual initialization subroutines. It has the form:

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

Example 1

In this example, assume it is desired to locate the initialization control subroutine and three initialization subroutines in the sector starting at location 10000_8.

The Configuration Module entry XINT is:

```
*
*  EXECUTIVE INITIALIZATION TABLE
        SUBR   XINT
XINT    OCT    10000
```

The initialization control subroutine and the initialization routines are:

```
*
*  INITIALIZATION CODE
        ORG    '10000
*
*  INITIALIZATION CONTROL
ICTL    DAC    **       LINK
        JST    INT1     CALL ROUTINE 1
        JST    INT2     CALL ROUTINE 2
        JST    INT3     CALL ROUTINE 3
        JMP*   ICTL     EXIT
*
*  INITIALIZATION SUBROUTINE 1
INT1    DAC    **       LINK
          .
          .
          .
        JMP*   INT1     EXIT
*
*  INITIALIZATION SUBROUTINE 2
INT2    DAC    **       LINK
          .
          .
          .
        JMP*   INT2     EXIT
*
*  INITIALIZATION SUBROUTINE 3
INT3    DAC    **       LINK
          .
          .
          .
        JMP*   INT3     EXIT
*
*  END OF INITIALIZATION SECTION
   FIN
```

Example 2

In this example, there is no user initialization. The Configuration Module entry XINT is:

```
*
*       EXECUTIVE INITIALIZATION TABLE
        SUBR    XINT
XINT    XAC     SC
```

This XINT contains a pointer to the scheduler.

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 the Real-Time Clock's XIDT entry in the Configuration Module.

CLK2 is the number of clock interrupts per second. This is 20 for a standard system. CLK3 is the negative of the number of hardware intevals between interrupts. This is -3 for the standard system.

The following examples show adjustments to the clock resolution.

16.7-ms Clock Resolution

CLK3 must be set to -1 to give an interrupt for every 16.7-ms clock period. CLK2 must then be set to 60, since 60 x 16.7 ms = 1 sec.

40-ms Clock Resolution (Model 316 Only)

First the Real-Time Clock period should be adjusted to 20 ms. Then, CLK3 should be adjusted to -2 (two 20-ms intervals between interrupts). CLK2 should be adjusted to 25, since 25 x 40 ms = 1 sec.

FORTRAN CAPABILITY

In Revision H of the standard Series 16 Fortran Compiler, new capabilities are added to facilitate writing Fortran programs to run under the OP-16 Operating System. The following subsections describe the configuration procedure and the programming rules for this revision of the compiler.

Compiler Configuration

The OP-16 statement processors and the on-line assembly code processor have been incorporated in a module separate from the compiler itself. If these features are desired in the Fortran system being configured, the module OPMOD, Doc. No. 70181980000, must be loaded immediately following the compiler object. If these features are not desired (that is, in an 8K system) the module OPDUM, Doc. No. 70181981000, must be loaded immediately

following the compiler object. In such a system, an ID error will be reported if any of the OP-16 statements are used on a program being compiled.

As in previous versions of the compiler, care must be taken when loading I/O driver packages to avoid cross-sector references. The area from '100 to '624 is reserved in the compiler for these references.

Programming Rules

This subsection describes the usage of the following compiler features.

1. OP-16 statements
2. In-line assembly code
3. Octal constants
4. Special data statement capabilities
5. Compiler library generation option
6. Additions to Fortran library

A Fortran system configured with OPMOD is a prerequisite for the utilization of features 1. and 2. above. The others may be used in any Fortran system.

OP-16 STATEMENTS

Header

This statement is used to generate a program header and may have the following forms.

```
HEADER    N
HEADER    N, L
HEADER    N(P)
HEADER    N(P), L
```

where

N = 1- or 2-character alphanumeric program name (used for documentation purposes only).

L = Number of words to generate for scheduled labels (if not present, 4 is assumed).

P = Communication parameter. Compiler assumes that parameter passed is an address of a parameter (or a parameter list). This name may be used in any way that a dummy variable is used.

The following coding is generated by this statement.

```
OCT     177777
OCT     0
  .
  .                    L words for scheduled labels (or 4 if L is omitted).
  .
OCT     0
OCT     0              For communication parameter.
```

```
        CALL  F$OE        If communication parameter is specified, F$OE
        DAC   P           is called to convert address so that it is identical
                          to a dummy variable.
```

If used, this statement must be the first statement in a program.

Request

This statement is used to generate an executive function call, type 1 (Request program). The forms of the statement are:

```
REQUEST   I,,E
REQUEST   I[P],,E
```

where:

I = Any integer expression which is evaluated and used as name of program being requested.

P = Communication parameter which must be either a constant or a variable name.

E = Statement number to which control is to be passed if the request fails.

The following coding is generated as a result of this statement.

```
        LDA
          .               Code to evaluate program name.
          .               If program name is a constant (i.e., 2HA  ), the
          .               constant will be generated in line.
        STA   X
        JST*  XLNK
        OCT   1
X       OCT   0           Program name.
        DAC   E
        DAC   P           Zero if parameter is not specified.
```

Note that parameter P (parameters if P is an array name) must be defined by DATA and/or DIMENSION statements. Examples for the use of the REQUEST statement, when calling I/O drivers, are included in the OP-16 driver manuals.

Schedule

This statement is used to generate a schedule label executive call and has the following form.

```
SCHEDULE          L[P],E
```

where:

L = Label to be scheduled. It may be either a statement number or an integer expression (in such a case, it is the programmer's responsibility to ensure that expression results in a valid address). In the simple case, expression would contain only an integer variable that has been assigned a statement number.

P = Program name which contains label being scheduled. May be an integer constant or an expression.

E = Statement number to which control is to be transferred if error is detected by system in the schedule request.

The following coding is generated by this statement.

```
      LDA
       .
       .                    Code to evaluate label. If label is a statement
       .                    number, a DAC to STMT number will be generated
                            in line.
      STX   X
      LDA
       .
       .                    Code to evaluate program name. If name is a
       .                    constant (i.e., 2HAB), it will be generated in line.
       .
      STA   Y
      JST*  XLNK
      OCT   2
Y     OCT                  Program name is stored or generated here.
      DAC   E              Error linkage.
X     DAC                  Label address is stored or generated here.
```

Connect Clock

This statement is used to generate a connect clock executive call and has the following form.

CONNECT CLOCK I[J, K], L, E

where:

I = Name of program to be connected, which may be an integer constant or an expressions.

J = Time of first execution, which may be a constant or an integer expression.

K = Interval, which may be an integer constant or an expression.

L = Base frequency, which may be an integer constant or an expression.

E = Statement number to which control is to be transferred if connect clock request fails.

The following coding is generated for this statement.

```
      LDA
       .                    Code to compute program name. If name is a
       .                    constant, code will be generated in line.
      STA   W
      LDA
       .                    Code to compute initial execution time.
      STA   X
```

```
        LDA
         :                      Code to compute interval.
         :
        STA   Y
        LDA
         :                      Code to compute base frequency.
         :
        STA   Z
        JST*  XLNK
        OCT   3
W       OCT                    Program name is stored or generated here.
        DAC   E
X       OCT                    Intial time is stored or generated here.
Y       OCT                    Interval is stored or generated here.
Z       OCT                    Base frequency is stored or generated here.
```

Disconnect Clock

This statement is used to generate a disconnect clock executive request. The statement has the following form.

```
DISCONNECT CLOCK     I[J], E
```

where

I = Program to be disconnected, which may be an integer constant or an expression.

J = Base frequency to which program is connected, which may be an integer constant or an expression.

E = Statement number to which control is to be transferred if request fails.

The following coding is generated by this statement.

```
        LDA
         :                      Code to evaluate program name. If name is
         :                      a constant, it will be generated in line.
         :
        STA   X
        LDA
         :                      Code to evaluate base frequency.
         :                      If frequency is an integer constant, it will be
         :                      generated in line.
         :
        STA   Y
        JST*  XLNK
        OCT   4
X       OCT                    Program name is stored or generated here.
        DAC   E                Error linkage.
Y       OCT                    Base frequency is stored or generated here.
```

Connect Interrupt

This statement is used to generate a connect interrupt executive function request and has the following form.

```
CONNECT INTERRUPT    N[R], E
```

where:

N = 1- to 6-character name of interrupt processor, which must appear as a subroutine entry point where it is defined.

R = Interrupt reference number. It may be an integer constant or an expression.

E = Statement number to which control is to be transferred if request fails.

The following coding is generated for this statement.

```
      LDA
       :                 Code to compute interrupt reference number.
       :                 If it is a constant, it will be generated in line.
      STA   X
      JST*  XLNK
      OCT   5
X     OCT               Interrupt reference number is stored or generated here.
      DAC   E           Error linkage.
      XAC   N           Address of interrupt code.
```

Disconnect Interrupt

This statement is used to generate a disconnect interrupt executive request. Its format is:

```
DISCONNECT INTERRUPT    N
```

where:

N = Reference number of interrupt to be disconnected. It may be an integer constant or an expression.

The following coding is generated for this statement.

```
      LDA
       :                 Code to evaluate interrupt reference number.
       :                 If it is a constant, it is generated in line.
      STA   X
      JST*  XLNK
      OCT   6
X     OCT               Reference number is stored or generated here.
```

Terminate

This statement is used to generate a program termination executive request. It has the following form.

```
TERMINATE
```

A path error will be reported if the next executable statement (if any) does not have a statement number.

The following coding is generated for this statement.

```
JST*  XLNK
OCT   7
```

Wait

This statement is used to generate a program wait executive request. It has the following form.

```
WAIT
```

A path error will be reported if the next executable statement does not have a line number.

Print Error

This statement is used to generate a call to the system error print routine. The statement has the following form.

```
PRINT ERROR    N[I]
```

where:

N = Program in which error occurs. It may be an integer constant (i.e., 2HXX) or a nonsubscript variable.

I = Error code. It may be an integer constant or an expression.

The following coding is generated for this statement:

```
LDA
 :                 Code to evaluate error.
 :
LDX   N           Program name.
INH
JST*  EROR
ENB
```

Interrupt Block

This statement is used to declare a section of a program that is an interrupt response code. The form for this statement is as follows.

```
INTERRUPT BLOCK    N
```

where:

N = 1- to 6-character name of interrupt block.

This interrupt block name will be declared as a subroutine entry point, and it will therefore be linked with an XAC's to the name (as in connect interrupt statements).

Either this statement may be used as the first statement in a program, in which case the entire program will be considered interrupt response code, or it may be embedded in a program, in which case the remainder of the program will be interrupt response code. The second method is advantageous, however, in that the interrupt response code has access to all the variables in the main portion of the program, thus avoiding passing parameters through common.

Interrupt Return

This statement is used to exit from interrupt response code and has the following format.

```
INTERRUPT RETURN   K
```

where:

K = Integer variable which contains address of label to be scheduled.

This statement generates the following code.

```
LDA   K          Label to A.
JMP*  ENTRY      Entry's DAC at start of interrupt code.
```

IN-LINE ASSEMBLY CODE

A small assembler incorporated into module OPMOD can process all I/O, shift, and generic instructions (except sense switch testing instructions). An assembly language line is signaled by the character A in column 1 of a source card. The op code must start in column 7 or later and must consist of a legal three-character instruction mnemonic followed by a blank.

NOTE: Embedded blanks within the op code are not allowed.

If the instruction is a shift instruction, a shift count must follow as either a decimal or an octal integer constant (octal constants are denoted by a leading colon). An I/O group instruction must be followed by an integer constant which is the instruction's function/device code. Generic instructions do not require an address, and an error will be flagged if one is present.

The error "OU" is flagged if an undefined operation is detected.

OCTAL CONSTANTS

An octal constant may now be used anywhere any integer constant is permitted. An octal constant is defined as a colon (:) followed by one to six octal digits (for example, :17710).

DATA STATEMENT ENHANCEMENTS

To facilitate compile time setup of OP-16 driver parameter lists, two extensions have been made to the data statement syntax.

First, an integer variable may be initialized to the address of a statement by placing the statement number preceded by a dollar sign ($) in the constant list position corresponding to the

variable name. For example, to initialize I to the address of statement 200, the following statement might be used.

```
DATA I/$200/
```

The second extension is used to initialize one variable to the address of another variable. When a variable appears in a constant list, the corresponding variable in the name list will be set to the address of the former variable. For example,

```
DATA N/M/
```

would be equivalent to the assembly language statement:

```
N DAC M
```

The variable whose address is being used as a constant may not have an explicit subscript. A subscripted variable name may be used, in which the address generated is the address of the first array element. If a dummy variable is used, the address generated is indirect, pointing to a word containing the actual address.

A repetition count is not allowed when using the extended features and will be set to one if used.

OP-16 FORTRAN PACKAGE

Overview

The OP-16 Fortran Package consists of the following subsystems.

OP-16 Fortran Read/Write Statement Processor (RWSP)

Re-entrant Math Subroutines (RMS)

OP-16 Fortran Library Extensions (FLE)

OP-16 Fortran Read/Write Statement Processor (RWSP)

RWSP allows the use of Read/Write statements in programs executed in the real-time multiprogramming environment of OP-16. It provides the links between the compiler-generated I/O calls and the drivers, and provides for editing of formatted calls. It is re-entrant and capable of processing formatted or nonformatted Read/Write statements to any combination of the supported devices in a simultaneous manner.

RWSP supports the following hardware.

ASR (Keyboard/Printer only)	Type 316/516 - 53/55
Line Printer	Types 5520 through 5527
Card Reader/Punch	Type 5140
7-Track Magnetic Tape (up to four tape drives)	Types 4020 and 4100
RO-35 Typewriters (output only, up to four devices)	Types 8892 through 8895

Relocated Base Sector Zero

Up to 16K of core (Compiler-Loader limitations)

RWSP consists of two groups of components: common system-level subroutines and driver Read/Write extensions.

The common system-level subroutines are:

Title	Document No.
System Level Pointer Table (SLPT)	70182762000
OP-16 I/O Editor (OPED)	70182767000
Call Converter (CONV)	70182791000

The driver Read/Write extensions are:

Title	Document No.
ASR Fortran Extension (ASF1)	70182790000
Line Printer Fortran Extension (LPI1)	70182694000
Card Reader/Punch Fortran Extension:	
Reader (CIF1)	70182697000
Punch (COF1)	70182698000
7-Track Magnetic Tape Fortran Extension (MTI1)	70182773000
RO-35 Printer Fortran Extension:	
Printer 1 (ROC1)	70182699000
Printer 2 (RO02)	70182900000
Printer 3 (RO03)	70182901000
Printer 4 (RO04)	70182902000

Loading procedures are described later in this subsection.

Re-entrant Math Subroutines (RMS)

RMS's include single-precision floating point add, subtract, multiply, and divide subroutines and their support subroutines. All subroutines included in the RMS are listed in Document No. 70182950-000 (RFMATH) (refer to information under Binder Table of Contents, Yellow Tab, in OP-16 Operating System Listing, Vol. 1).

RMS requires the High-Speed Arithmetic hardware option and the OP-16 Fortran Library component OPFRT2H.

OP-16 Fortran Library Extensions (FLE)

Fortran programs in general call upon standard subroutines. These subroutines are documented and distributed under the name of Fortran Library. The Fortran Library routines are to be link-loaded with each Fortran program.

In some cases the existing library routines had to be modified to satisfy OP-16 requirements; also, some additions were necessary. These have been incorporated in four OP-16 Fortran Library tapes:

Title	Document No.
OPFRT1	70182903000
OPFRT2H	70182904000
OPFRT2S	70182905000
OPFRT3	70182906000

OPFRT2H is required if the Re-entrant Math Subroutines are used; OPFRT2S is required otherwise.

OP-16 Fortran Package Loading Procedures

The procedures for loading the OP-16 Fortran Package are as follows.

1. If any of the Fortran programs configured in the system contain Read/Write statements, link-load the following components with the Executive.

   SLPT, System Level Pointer Table

   OPED, OP-16 Re-entrant I/O Editor (must be loaded on a sector boundary)

   CONV, Fortran I/O Call Converter.

   Note that all three components are available punched on a single paper tape, Doc. No. 70182918-000.

   If Read/Write statements are not used, omit step 1.

2. If the High-Speed Multiply/Divide hardware option is present in the system, link-load the Re-entrant Floating Point Subroutines (Doc. No. 70182950-000) with the Executive.

   Note that when the Relocated Base Sector hardware option is used, the Re-entrant Floating Point Subroutines must be loaded so as not to generate any cross-sector links.

   If this hardware option is not present, omit step 2.

3. Complete loading all the other components that need to be linked with the Executive.

4. Load the first driver. If the driver is called by Read/Write statements, link-load the appropriate driver Read/Write Extensions (and XLOCS) as described in the driver manuals. Repeat for each driver.

   If a driver is not called by Read/Write statements, omit loading the corresponding driver Read/Write Extension.

5. After loading the Fortran programs, load the Fortran Library tapes together with the OP-16 Fortran Library tapes in the following order.

	Title	Document No.	Condition
1.	LTCF1	70181876000	—
2.	LTCF2	70181877000	—
3.	LTCF3H	70181878000	If hardware Arithmetic option is present.
	or		
	LTCF3S	70181882000	If hardware Arithmetic option is not present.

	Title	Document No.	Condition
4.	OPFRT1	70182903000	—
5.	LTCF4	70181879000	—
6.	OPFRT2H	70182904000	If Re-entrant Math Subroutines are used.
	or OPFRT2S	70182905000	If Re-entrant Math Subroutines are not used.
7.	LTCF5H	70181880000	If hardware Arithmetic option is present.
	or LTCF5S	70181883000	If hardware Arithmetic option is not present.
8.	OPFRT3	70182906000	—
9.	LTCF6	70181881000	—

OP-16 I/O Editor Error Messages

The following run-time errors may be reported by the OP-16 I/O Editor on the ASR.

701XX[1]	DIGIT PRECEDES (
702XX	NO OPENING (
703XX	NO DECIMAL PT.
704XX	ILLEGAL CHARACTER
705XX	INTEGER PRECEDES -
706XX	INTEGER PRECEDES /
707XX	TOO MANY NESTED ( )
710XX	NON INTEGER
711XX	OUT OF RANGE FOR FIXED POINT
712XX	OUT OF RANGE FOR INTEGER
713XX	NOT T OR F
714XX	FIELD WIDTH EXCEEDED

COMPILER LIBRARY GENERATION FACILITY

Three special capabilities were designed into the Fortran Compiler at its inception to allow parts of the library to be written in Fortran. These features, as described below, are enabled by setting bit 1 of the A register when starting the compiler.

Register Load

A statement of the form =expr is used to evaluate the expressions and leave it in the appropriate register, depending on the mode of the expression. Examples:

= (I+10)/2 loads A register with (I + 10)/2

= 1.0 loads A and B registers with floating point value 1.0

= 1.000 loads double-precision accumulator (AC1, AC2, AC3) with double-precision value 1.0

= (1.0, 1.0) loads complex accumulator (AC1, . . . , AC4) with (1.0, 1.0)

[1] XX = Program name.

Register Store

A statement of the form var = will store the contents of the register appropriate to the variable mode in the variable. For example, I= will store the A register in I. Care must be taken if the variable is subscripted. In such a case, the operation will fail if the variable is an integer or real and if one of the following conditions is met.

1. If the variable is a dummy and the subscript is not the constant one.
2. If the subscript contains a nonconstant element; for example, I(J)= will fail.

Register Test

An arithmetic IF statement may be used to test the sign of the A register as follows.

IF( ) S1, S2, S3

where S1, S2, and S3 are statement numbers.

The register is not modified by the test.

FORTRAN LIBRARY ADDITIONS

The following three routines have been added to the standard Fortran library to facilitate writing system programs with Fortran.

Function LOC (arg)

This function returns as its value the address of its argument. For example, LOC(A(10)) will have as its value the address of the 10th element of array A.

Function IFETCH (arg)

This function returns as its value the contents of the memory location whose address is in the argument (it must be an integer). For example, IFETCH(:1000) will have the value of the contents of location '1000.

If the symbol XDCT is declared in an external statement, the following statement could be used to inspect the second word (the ASR flag) of the table XDCT in the OP-16 Configuration Module.

```
I = IFETCH(LOC(XDCT)+1)
```

Subroutine ISTORE (arg1, arg2)

This subroutine is used to set the memory location whose address is contained in the first argument to the value contained in the second argument. If used as a function, it returns the previous contents of the specified memory location. In the following example, the value -2 is stored in location '61 (Real-Time Clock).

```
CALL ISTORE(:61, -2)
```

New Error Diagnostics

'OU'	If undefined assembly op code.
'IR'	Interrupt return statement not in an interrupt block.
'IB'	If interrupt block in a subroutine or function.

EXTENDED MEMORY

The following three conditions must be met for the OP-16 Operating System to operate in Model 316 or 516 systems with extended memory (more than 16K).

1. The third word (labeled XEXA) in table XSPT (Executive Special Parameters Table) must be set to nonzero.
2. The loading option (P=XX006) of LDR-APM (loads object programs in the extended desectorizing mode) must be used when building the system.
3. Attention should be given to the programming considerations required for programs to operate in the Extend mode (see the 316/516 Programmers' Reference Manual, Doc. No. 70130072156, for discussion).

   The handling of indexed instructions is the most important difference between Normal and Extend mode. Programs written to operate in the Normal mode may not necessarily operate correctly in the Extend mode.

   Normal mode is not permitted when table XSPT indicates that the Extended Addressing option is present.

ELIMINATION OF THREE SYSTEM ROUTINES

Op-16 systems may run without any or all of the following routines: Error Print Program, OPTRAC, and Keyboard Program. The rules for eliminating these routines are given below.

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
```

The entries for program EP in XPLT and XPET, as well as the SUBR ER statement at the beginning of XCOM, must be omitted.

OPTRAC

The following subroutine must be linked into the system in place of OPTRAC.

```
        SUBR    OPTRAC,OPTR
                REL
OPTR    DAC     **
        JMP*    OPTR
        END
```

Keyboard Program

The entries for program KB should be omitted from XPLT and XPET. The following statement should be added at the end of table XSPT (Executive Special Parameters Table).

```
KBI     DAC     *
        BSZ     2
```

APPENDIX A
SEGMENT REFERENCE TABLE

The following table gives the relationship of sectors (512-word blocks defined by the hardware) and segments (128-word blocks defined by RTX-16). All numbers are in octal. Lines have been drawn at 4K intervals showing the highest sector and segment for 4K through 32K memories.

Memory Size	Sector	Segment	Starting Address	Memory Size	Sector	Segment	Starting Address
4K	0	0	0	12K	20	100	20000
		1	200			101	20200
		2	400			102	20400
		3	600			103	20600
	1	4	1000		21	104	21000
		5	1200			105	21200
		6	1400			106	21400
		7	1600			107	21600
	2	10	2000		22	110	22000
		11	2200			111	22200
		12	2400			112	22400
		13	2600			113	22600
	3	14	3000		23	114	23000
		15	3200			115	23200
		16	3400			116	23400
		17	3600			117	23600
	4	20	4000		24	120	24000
		21	4200			121	24200
		22	4400			122	24400
		23	4600			123	24600
	5	24	5000		25	124	25000
		25	5200			125	25200
		26	5400			126	25400
		27	5600			127	25600
	6	30	6000		26	130	26000
		31	6200			131	26200
		32	6400			132	26400
		33	6600			133	26600
	7	34	7000		27	134	27000
		35	7200			135	27200
		36	7400			136	27400
		37	7600			137	27600
8K	10	40	10000	16K	30	140	30000
		41	10200			141	30200
		42	10400			142	30400
		43	10600			143	30600
	11	44	11000		31	144	31000
		45	11200			145	31200
		46	11400			146	31400
		47	11600			147	31600
	12	50	12000		32	150	32000
		51	12200			151	32200
		52	12400			152	32400
		53	12600			153	32600
	13	54	13000		33	154	33000
		55	13200			155	33200
		56	13400			156	33400
		57	13600			157	33600
	14	60	14000		34	160	34000
		61	14200			161	34200
		62	14400			162	34400
		63	14600			163	34600
	15	64	15000		35	164	35000
		65	15200			165	35200
		66	15400			166	35400
		67	15600			167	35600
	16	70	16000		36	170	36000
		71	16200			171	36200
		72	16400			172	36400
		73	16600			173	36600
	17	74	17000		37	174	37000
		75	17200			175	37200
		76	17400			176	37400
		77	17600			177	37600

Memory Size	Sector	Segment	Starting Address
20K	40	200	40000
		201	40200
		202	40400
		203	40600
	41	204	41000
		205	41200
		206	41400
		207	41600
	42	210	42000
		211	42200
		212	42400
		213	42600
	43	214	43000
		215	43200
		216	43400
		217	43600
	44	220	44000
		221	44200
		222	44400
		223	44600
	45	224	45000
		225	45200
		226	45400
		227	45600
	46	230	46000
		231	46200
		232	46400
		233	46600
	47	234	47000
		235	47200
		236	47400
		237	47600
24K	50	240	50000
		241	50200
		242	50400
		243	50600
	51	244	51000
		245	51200
		246	51400
		247	51600
	52	250	52000
		251	52200
		252	52400
		253	52600
	53	254	53000
		255	53200
		256	53400
		257	53600
	54	260	54000
		261	54200
		262	54400
		263	54600
	55	264	55000
		265	55200
		266	55400
		267	55600
	56	270	56000
		271	56200
		272	56400
		273	56600
	57	274	57000
		275	57200
		276	57400
		277	57000

Memory Size	Sector	Segment	Starting Address
28K	60	300	60000
		301	60200
		302	60400
		303	60600
	61	304	61000
		305	61200
		306	61400
		307	61600
	62	310	62000
		311	62200
		312	62400
		313	62600
	63	314	63000
		315	63200
		316	63400
		317	63600
	64	320	64000
		321	64200
		322	64400
		323	64600
	65	324	65000
		325	65200
		326	65400
		327	65600
	66	330	66000
		331	66200
		332	66400
		333	66600
	67	334	67000
		335	67200
		336	67400
		337	67600
32K	70	340	70000
		341	70200
		342	70400
		343	70600
	71	344	71000
		345	71200
		346	71400
		347	71600
	72	350	72000
		351	72200
		352	72400
		353	72600
	73	354	73000
		355	73200
		356	73400
		357	73600
	74	360	74000
		361	74200
		362	74400
		363	74600
	75	364	75000
		365	75200
		366	75400
		367	75600
	76	370	76000
		371	76200
		372	76400
		373	76600
	77	374	77000
		375	77200
		376	77400
		377	77600

APPENDIX B

INTERRUPT REFERENCE NUMBERS ASSIGNED IN RTX-16

Number	Option
1	Mass-store device
2	High-Speed Paper-Tape Reader
3	High-Speed Paper-Tape Punch
4	ASR Typewriter
5	Alarm Typewriter
6	Logging Typewriter
7	Magnetic Tape TCU 1
8	CR, CRP
9	LP
10	SDLC 1
11	SDLC 2
12	SDLC 3
13	SDLC 4
14	A/D (A)
15	WD
16	TC 1
17	ASYNCH
18	COUNTER
19	A/D (B)
20	TC 2
21	TC 3