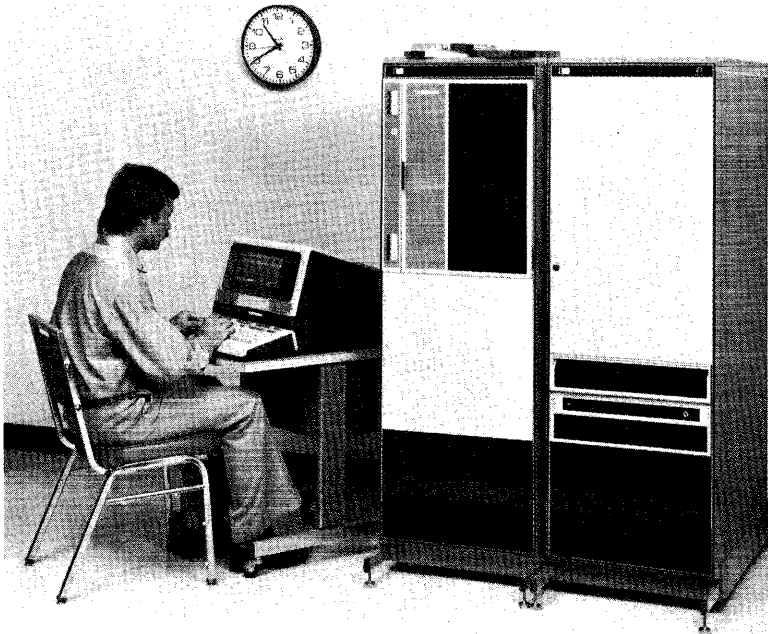


# RTE-IV Programmer's Reference Manual



# **RTE-IV Programmer's Reference Manual**

(This manual reflects information that is compatible with  
software revision code 1826.)



---

HEWLETT-PACKARD COMPANY  
11000 WOLFE ROAD, CUPERTINO, CALIFORNIA, 95014

<p><b>Library Index Number</b> <b>2RTE.320.92067-90001</b></p>
--

# PUBLICATION NOTICE

Information in this manual describes the RTE-IV operating system software. Changes in text to document software updates subsequent to the initial release are supplied in manual update notices and/or complete revisions to the manual. The history of any changes to this edition of the manual is given below under "Publication History." The last change itemized reflects the software currently documented in the manual.

Any changed pages supplied in an update package are identified by a change number adjacent to the page number. Changed information is specifically identified by a vertical line (revision bar) on the outer margin of the page.

## PUBLICATION HISTORY

First Edition ..... June 78 (Software Rev. Code 1826)

### NOTICE

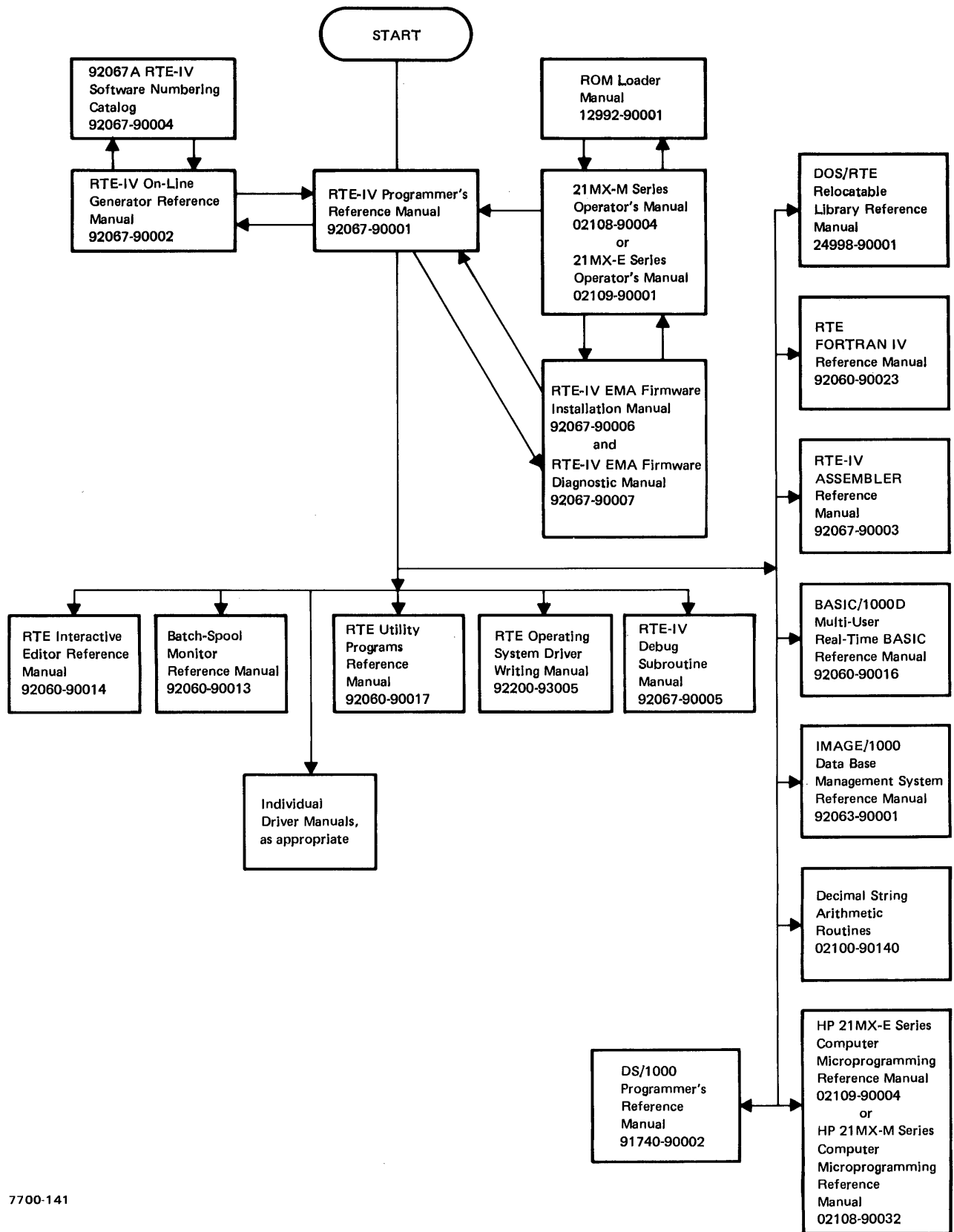
The information contained in this document is subject to change without notice.

HEWLETT-PACKARD MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Hewlett-Packard shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance or use of this material.

Hewlett-Packard assumes no responsibility for the use or reliability of its software on equipment that is not furnished by Hewlett-Packard.

This document contains proprietary information which is protected by copyright. All rights are reserved. No part of this document may be photocopied, reproduced or translated to another program language without the prior written consent of Hewlett-Packard Company.

# DOCUMENTATION MAP





This manual describes the features and functions of the RTE-IV operating system for HP 1000 Computers and Computer Systems. The manual is the primary reference source for those who will be writing or maintaining computer software, or who are otherwise involved in the design and operation of an RTE-IV operating system.

The purpose of the manual is describe the functions and requirements for utilizing all available system services for developing and executing programs in a real-time environment; i.e., use of operator commands, I/O procedures, memory allocation, hardware and software interface, system procedures, program scheduling, on-line I/O and memory reconfiguration, and various utility programs and relocatable library subroutines.

It is assumed that readers of this manual already have a working knowledge of one of the three programming languages available to RTE-IV users (RTE FORTRAN-IV, RTE-IV Assembly Language or BASIC/1000D) and that they have a general understanding of the appropriate Operators Manual for their HP computer system.

Some available manuals offering other levels of information that may be directly relevant to system operation and user applications are briefly summarized below:

\* Batch Spool Monitor Reference Manual

Describes the uses and requirements of the Batch Spool Monitor subsystem for those who wish to use the batch spooling feature. Of particular relevance for all applications users is this manual's descriptions and formats of the File Manager (FMGR) commands.

\* RTE Interactive Editor Reference Manual

Describes the format and function of the RTE Editor (EDITR) program commands and procedures for on-line editing services. The manual is of particular relevance for those involved in editing source programs or updating existing programs.

\* RTE Utility Reference Manual

Describes the functions and requirements for using a collection of utility programs, including: WHZAT (current system status information), Disc Back-Up (copy from disc-to-disc or disc to other device), KEYS and KYDUMP (creates softkey command string sets), LGTAT (logs and displays system and auxiliary disc status), etc.

\* DOS/RTE Relocatable Library Reference Manual

Describes a set of utility subroutines that are primarily used by FORTRAN-IV and Assembly Language programs.

\* RTE-IV Debug Subroutine Manual

Describes the use and formats of the interactive Debug (DBUGR) subroutine commands that can be used in checking for logical errors in a program. A subset of the most frequently-used DBUGR commands are also described in a separate section of the RTE-IV Programmer's Reference Manual.

\* RTE-IV On-Line Generator Reference Manual

Describes detailed, "cookbook" procedures for generating a new RTE-IV operating system without shutting down the existing RTE-IV system. Complete examples of each phase and heavily annotated worksheets are provided. Manual is primarily intended for system managers and system programmers who are involved in the design and maintenance of total system configurations.

\* RTE-IV Software Numbering Catalog

A cross-referenced directory of all RTE-IV software files located on supplied distribution media. Describes for users who have a file number or part number for an RTE-IV software part how to find the part number on the medium (or media) on which the part is distributed. Manual is primarily intended for system managers and others involved in generating new systems or reconfiguring existing systems. Optional software supplied with other products is listed in separate Software Numbering Catalogs distributed with those products.

\* Appropriate Driver Manuals

Individual manuals that aid users in determining the particular drivers required for site-specific combinations of devices. Manuals describe the criteria for configuring the various drivers into an operating system. The manuals are primarily intended for those in configuration design and generation.

\* RTE-IV Operating System Driver Writing Manual

Describes the system considerations, requirements and constraints in creating user-written drivers for specialized applications not covered by supplied standard device drivers.

Other manuals that may be of particular interest, including language manuals, will be found listed in the Documentation Map in this manual.

TABLE OF CONTENTS
-------------------

Foreword	Page
Glossary	

Section I

GENERAL DESCRIPTION

Real-Time Executive .....	1-1
System Hardware .....	1-2
System Software.....	1-3
Memory Management.....	1-3
Multiprogramming.....	1-3
Input/Output Processing .....	1-4
Resource Management.....	1-4
Executive Communication.....	1-5
Operator Commands.....	1-6
System Configuration .....	1-6
Multi-Terminal Operations.....	1-7
System Utility Programs.....	1-7
Relocating Loader (LOADR).....	1-8
File Management Package (FMP).....	1-8
Interactive Editor (EDITR).....	1-8
Batch Spool Monitor (BSM).....	1-8
WHZAT.....	1-8
DBUGR.....	1-9
On-Line Generator ((RT4GEN).....	1-9
SWTCH.....	1-9
Disc Backup.....	1-9
Disc Update.....	1-9
KEYS and KDUMP.....	1-10
LGTAT.....	1-10
Programming Languages.....	1-10
RTE FORTRAN IV.....	1-10
RTE Assembler.....	1-10
RTE Micro-Assembler.....	1-10
Real-Time BASIC/1000D.....	1-11
QUERY.....	1-11
RTE-IV System Summary.....	1-11

Section II

STANDARD BOOT-UP PROCEDURES

Boot Loaders and Boot Extension.....	2-1
Disc Loader ROM.....	2-1
Bootstrap Loader.....	2-3
Boot Extension Execution.....	2-3

## TABLE OF CONTENTS

### Section III

#### OPERATOR COMMANDS

Introduction.....	3-1
Command Structure.....	3-1
Command Conventions.....	3-1
RTE-IV Operator Commands.....	3-5
AB (abort).....	3-5
AS (assign partition).....	3-6
BL (buffer limits).....	3-7
BR (break).....	3-8
DN (down).....	3-9
EQ (status).....	3-10
EQ (buffering).....	3-11
FL (flush).....	3-12
GO (reschedule).....	3-13
IT (Interval Timer).....	3-15
LG (LG tracks).....	3-16
LS (source file).....	3-17
LU (assignment).....	3-18
LU (reassignment).....	3-19
OF (terminate).....	3-20
ON (schedule).....	3-21
PR (priority).....	3-23
RT (release tracks).....	3-24
RU (run).....	3-25
SS (operator suspend).....	3-27
ST (status).....	3-28
SZ (assignment).....	3-30
SZ (reassignment).....	3-31
TI (time).....	3-33
TM (set clock).....	3-34
TO (time-out).....	3-35
UP (make available).....	3-36
UR (release reserved partition).....	3-37
Operator Command Error Messages.....	3-38

### Section IV

#### EXEC CALLS

Introduction.....	4-1
Assembly Language Format.....	4-1
FORTRAN IV Format.....	4-2
EXEC Call Error Returns.....	4-4
EXEC Call Summary.....	4-6
Standard Function Calls.....	4-9
READ/WRITE .....	4-9
I/O CONTROL.....	4-12
I/O STATUS.....	4-16
DISC TRACK ALLOCATION .....	4-19
PROGRAM DISC TRACKS RELEASE .....	4-21
GLOBAL DISC TRACKS RELEASE.....	4-22
PROGRAM COMPLETION.....	4-24
PROGRAM SUSPEND.....	4-27
PROGRAM SEGMENT LOAD.....	4-28

## TABLE OF CONTENTS

PROGRAM SCHEDULE.....	4-29
TIME REQUEST.....	4-33
STRING PASSAGE.....	4-34
TIMED EXECUTION (Initial Offset).....	4-36
TIMED EXECUTION (Absolute Start Time).....	4-38
PROGRAM SWAPPING CONTROL.....	4-41
PARTITION STATUS.....	4-42
MEMORY SIZE.....	4-44
Class I/O EXEC Calls.....	4-46
Class I/O READ/WRITE.....	4-50
Class I/O GET.....	4-52
Class I/O CONTROL.....	4-55
Class I/O Applications Examples.....	4-56
Resource Numbers and Logical Unit Locks.....	4-63
Executive Error Messages.....	4-71
Memory Protect Violations.....	4-72
Dynamic Mapping Violations.....	4-72
Dispatching Errors.....	4-72
EX Errors.....	4-73
Unexpected DM and MP Errors.....	4-73
TI, RE and RQ Errors.....	4-74
Parity Errors.....	4-74
Other EXEC Errors.....	4-75
Disc Allocation Error Messages.....	4-76
Schedule Call Error Codes.....	4-76
I/O Call Error Codes.....	4-76
Program Management Error Codes.....	4-77
Logical Unit Lock Error Codes.....	4-77
Executive Halt Errors.....	4-77
Section V	
I N P U T / O U T P U T	
Software I/O Structure.....	5-1
Equipment Table.....	5-2
Device Reference Table.....	5-7
Logical Unit Numbers.....	5-9
Interrupt Table.....	5-10
System Base Page Interrupt Locations.....	5-11
Driver Mapping Table.....	5-12
I/O Processor General Operation.....	5-14
Standard I/O Calls.....	5-14
Power Fail.....	5-15
I/O Controller Time-Out.....	5-16
Privileged Interrupt Processing.....	5-17
Section VI	
M E M O R Y   M A N A G E M E N T	
Addressing.....	6-1
Memory Maps.....	6-2
Physical Memory.....	6-3
Logical Memory.....	6-7
Base Page.....	6-9

## TABLE OF CONTENTS

COMMON Areas.....	6-10
Memory Protection.....	6-10
Partitions.....	6-12
Partition Lists.....	6-12
Partition Assignment and Reservation.....	6-12
Mother Partitions.....	6-13
Subpartitions.....	6-13
Extended Memory Area.....	6-14
Memory Management Subroutines.....	6-19
.EMAP.....	6-19
.EMIO.....	6-22
MMAP.....	6-23
EMAST.....	6-24

### Section VII

#### R E L O C A T I N G   L O A D E R

RTE Relocating Loader.....	7-1
RU,LOADR Command Options.....	7-2
Program Relocation.....	7-2
On-Line Modification.....	7-3
Segmented Programs.....	7-4
Adding New Programs.....	7-4
Program Replacement.....	7-5
Addition or Replacement Limitations.....	7-5
Program Deletion.....	7-6
COMMON Allocations.....	7-6
Program Types.....	7-7
Loader Operation.....	7-8
Additional Opcode Parameters.....	7-12
Loading the Binary Code.....	7-13
Loader Command File.....	7-14
SEARCH.....	7-14
SEARCH <namr>.....	7-14
RELOCATE <namr>.....	7-14
FORCE.....	7-14
DISPLAY.....	7-14
ECHO.....	7-14
END.....	7-14
/A.....	7-14
*.....	7-15
AS,xx.....	7-15
SZ,<yy>.....	7-15
LL,<namr>.....	7-15
OP,<opcode>.....	7-15
FM,<format>.....	7-15
Loading From a Logical Unit.....	7-16
Loading Segmented Programs.....	7-16
Reducing Segmented Program Load Time.....	7-19
DBUGR Library Subroutine.....	7-20
LOADR Error Reporting.....	7-20
LOADR Error Codes.....	7-21

TABLE OF CONTENTS

Section VIII

SEGMENTED PROGRAMS

RTE FORTRAN-IV Segmentation..... 8-2  
 RTE Assembler Segmentation..... 8-3

Section IX

MULTI-TERMINAL MONITOR

System Configuration..... 9-1  
 Multipoint Initialization..... 9-1  
 Logical Unit Number Assignment..... 9-3  
 Operation..... 9-3  
 Available MTM Services..... 9-4  
 Automatic Scheduling of FMGxx..... 9-4  
 FMGxx Execution..... 9-4  
 BREAK and ABORT Command Variations..... 9-5  
 Automatic Program Renaming..... 9-7  
 Creating Program Copies..... 9-8  
 Program Swapping..... 9-9

Section X

RTE-IV SYSTEM LIBRARY

Introduction..... 10-1  
 Calling System Library Subroutines..... 10-1  
 Reentrant Subroutine Structure..... 10-2  
 Reentrant Subroutine Format..... 10-3  
 Privileged Subroutine Structure..... 10-3  
 Privileged Subroutine Format..... 10-4  
 Memory Resident Library..... 10-4  
 Utility Subroutine Structure..... 10-5  
 System Library Subroutines..... 10-5  
 REIO - Reentrant I/O Subroutine..... 10-6  
 BINRY - Disc Read/write Subroutine..... 10-6  
 RNRQ - Resource Management Subroutine..... 10-7  
 LURQ - Logical Unit Lock..... 10-11  
 \$PARS - Parse Subroutine..... 10-13  
 INPRS - Buffer Conversion Subroutine..... 10-14  
 \$CVT3, \$CVT1, CNUMD, CNUM0, KCVT - Binary to ASCII  
 Subroutines..... 10-14  
 MESSS - Message Processor Interface Subroutine..... 10-15  
 EQLU - Interrupting LU Query..... 10-16  
 PTRN, PTRM - Parameter Return Subroutines..... 10-16  
 .DRCT - Indirect Address Subroutine..... 10-18  
 IFBRK - Breakflag Test Subroutine..... 10-18  
 COR.A, COR.B - First Word Available Memory Subroutine..... 10-19  
 IDGET - Retrieve Program's ID Segment Address..... 10-20  
 TMVAL - Current Time Subroutine..... 10-21  
 GETST - Recover Parameter String..... 10-21  
 IFTTY - Query Whether Logical Unit is Interactive or Not..... 10-22  
 LOGLU - Returns LU of Terminal That Scheduled Program..... 10-22  
 .EMAP, .EMIO, MMAP, EMASST Subroutines..... 10-23

TABLE OF CONTENTS

Section XI  
 D E B U G R I N T E R A C T I V E D E B U G G I N G

Calling DBUGR.....	11-1
Entering DBUGR.....	11-2
DBUGR Commands.....	11-3
DBUGR Modes.....	11-3
Expressions and Terms.....	11-4
Examine Memory.....	11-5
Modify Memory.....	11-6
Examine Registers.....	11-6
Setting a Label.....	11-7
Execute Program.....	11-7
Breakpoints.....	11-8
Tracing.....	11-11
DBUGR Error Messages.....	11-12
DBUGR Example.....	11-13

Section XII  
 M E M O R Y A N D I/O R E C O N F I G U R A T I O N

Scheduling the Configurator From Disc Loader ROM.....	12-1
Scheduling the Configurator From Bootstrap Loader.....	12-3
Configurator Program.....	12-3
Configurator Halts and Error Messages.....	12-4
Reconfiguration Procedures.....	12-4
I/O Reconfiguration Steps.....	12-5
Memory Reconfiguration Procedures.....	12-8
Excluding Bad Pages.....	12-9
SAM Extension Reconfiguration.....	12-9
Changing Partition Definitions.....	12-10
Changing Program Partition Assignments.....	12-14
Program Partition Assignments.....	12-15
Reconfiguration Example.....	12-16
Boot-Up and Reconfiguration Halts.....	12-19
Configurator Error Messages.....	12-21

Appendix A  
 H P C H A R A C T E R S E T..... A-1

Appendix B  
 S Y S T E M C O M M U N I C A T I O N A R E A A N D S Y S T E M T A B L E S

System Communication Area.....	B-1
Program ID Segment.....	B-4
ID Segment Extensions.....	B-8
Short ID Segments.....	B-9
Memory Allocation Table Entry.....	B-9
RTE-IV System Disc Layout.....	B-11

Appendix C  
 R E C O R D F O R M A T S

Source Record Formats.....	C-1
NAM Record.....	C-3



## TABLE OF CONTENTS

ENT Record.....	C-4
EXT Record.....	C-5
DBL Record.....	C-6
EMA Record.....	C-7
END Record.....	C-7
Absolute Tape Format.....	C-8

### Appendix D

RTE-IV VERSUS RTE-III.....	D-1
Logical User Map.....	D-1
Driver Partitions.....	D-1
Type 2, 3 and 4 Programs.....	D-1
Extended Memory Areas.....	D-2
Memory Resident Library.....	D-2
File Input/Output.....	D-2
Parity Error.....	D-2
Memory and I/O Reconfiguration.....	D-2

### Appendix E

TABLE AREA I AND II ENTRY POINTS.....	E-1
---------------------------------------	-----

### Appendix F

RTE-IV PROGRAM TYPES.....	F-1
---------------------------	-----

### Appendix E

ERROR MESSAGES INDEX.....	E-1
---------------------------	-----

## I L L U S T R A T I O N S

Read/Write (conwd) Format.....	4-10
I/O Control (conwd) Format.....	4-13
Partition Status Parameter Return.....	4-44
Partition Current Status Example.....	4-46
Class Number (ICLAS) Format.....	4-51
Class I/O Multiple Terminal Input Example.....	4-58
Dispatching Input to Subtasks for Processing.....	4-62
"Deadly Embrace" Examples.....	4-68
Equipment Table Entry Format.....	5-3
CONWD Word (EQT Entry Word 6) Expanded.....	5-6
Device Reference Table Entry Format.....	5-7
Device Reference Table.....	5-8
Driver Mapping Table.....	5-13
RTE-IV Address Scheme.....	6-2
Physical Memory Allocations.....	6-5
RTE-IV 32K Word Logical Memory Configurations.....	6-8
Base Page Structure.....	6-9
Memory Protect Fence Locations for Programs Using COMMON.....	6-11
EMA and MSEG Structure.....	6-16
Multiple Data Arrays Organization.....	6-18

## TABLE OF CONTENTS

Segmented Program Example.....	7-16
Segmented Programs.....	8-2
Main Calling Segment.....	8-3
Segment Calling Segment.....	8-4
Segment to Main Jumps.....	8-4
RNRQ Control Word Format.....	10-9
Reconfiguration Example.....	12-16
ID Segment Format.....	B-5
ID Segment Extension.....	B-8
Memory Allocation Table Entry Format.....	B-9
RTE-IV System Disc Layout.....	B-11
Source Record Formats .....	C-1

### T A B L E S

Operator Command Summary.....	3-2
Operator Command Syntax Conventions.....	3-4
Operator Command Error Messages.....	3-38
RTE EXEC Calls.....	4-7
I/O Status Word (ISTAL/ISTA2) Format.....	4-18
EQT Word 5 Status Table.....	4-19
Class Input/Output Terms.....	4-48
EXEC Call Error Summary.....	4-79
Loader Error Codes.....	7-22
DBUGR Error Messages.....	11-12
System Boot-Up and Reconfiguration Halts.....	12-20
I/O and Memory Reconfiguration Error Codes.....	12-22
System Communication Area Locations.....	B-2
RTE-IV Program Types.....	F-1

GLOSSARY OF TERMS
-------------------

**ABSOLUTE PROGRAM** - A program that has been relocated and is capable of being loaded into main memory for subsequent execution. An "absolute program" is synonymous with "relocated program."

**ABSOLUTE SYSTEM** - The binary memory image of an RTE system (stored on Logical Unit 2).

**ADDRESS SPACE** - see LOGICAL MEMORY or PHYSICAL MEMORY.

**ASYNCHRONOUS DEVICE** - A device that can perform I/O operations that are independent of time considerations but operates simultaneously with program execution. Interaction with the computer is through request/response circuitry.

**AUXILIARY DISC SUBCHANNEL** - An optional subchannel that is treated as a logical extension of the system disc subchannel, Logical Unit 2. If used, it is assigned to Logical Unit 3. The binary memory image of RTE-IV may not reside on the auxiliary subchannel.

**BACKGROUND (BG)** - An arbitrary name for one of two types of partitions in RTE; generally used for lower priority programs whose responses to interrupts are not time-critical.

**BASE PAGE** - A 1024-word area of memory corresponding to logical page 0. It contains the system's communication area, driver links, trap cells for interrupt processing, and system and/or user program links.

**BASE PAGE FENCE** - A hardware register that divides a logical base page into a portion containing the user's base page and a portion of the system's base page.

**BG** - See BACKGROUND.

**BLOCK** - Two logical disc sectors of 128 bytes each, totaling a 256 bytes.

**BOOT EXTENSION** - An absolute program that resides on the first two sectors of logical track 0 of the system subchannel. The Boot Extension itself is first loaded into memory by the Bootstrap Loader or ROM Loader.

**BOOT FILE** - An optional file to which the Bootstrap Loader produced by the On-Line Generator is stored. This may be a disc file or a logical unit (e.g., a mini-cartridge).

## GLOSSARY OF TERMS

**BOOTSTRAP LOADER** - A loader produced by the Generator and stored in the boot file. The Bootstrap Loader loads the Boot Extension into memory and then transfers control to the Boot Extension.

**BOOT-UP** - The process of bringing the Bootstrap Loader or ROM Loader contents into memory. Control is then transferred to the Boot Extension to begin the initialization process.

**BUFFER** - An area of memory (main-memory, mass memory or local peripheral memory) used to temporarily store data.

**CLASS I/O** - A means of buffering data between devices and user programs, and between programs themselves, that permits a user program to continue execution concurrently with its own I/O. The term "I/O without program wait" is a more commonly used term.

**CLOSE FILE** - A method of terminating a program's access to a file so that no further read/write operations may be performed on the file.

**COMMON** - An area of memory that can be accessed by a program and its subprograms. Usually used to pass data from a program to a subprogram. In RTE, system COMMON may be used to pass data from one program to another.

**CONFIGURATOR** - A two-part program that allows reconfiguration of an RTE system's I/O and physical memory structures without going through a new system generation. The configurator is initiated as an option during the startup process.

**CURRENT PAGE** - The memory page in which the executing instruction is located. Some 21MX memory reference instructions can only directly reference locations in two pages: current page and base page.

**DATA CONTROL BLOCK (DCB)** - A table within an executable program that contains information used by the File Management Package (FMP) in performing disc accesses. (See the RTE Batch Spool Monitor Reference Manual.)

**DCPC** - see Dual Channel Port Controller

**DEVICE DOWN** - Relates to the state of a peripheral device or I/O controller. When the device is down, it is no longer available for use by the system. The term also refers to the DN operator command.

**DEVICE INDEPENDENCE** - Refers to the ability of a program to perform I/O without knowing which physical device is being accessed (see also Logical Unit Number).

DEVICE REFERENCE TABLE (DRT) - A table created during system generation corresponding to Logical Units 1 through 63. The contents of the Device Reference Table include a pointer to the associated EQT entry, subchannel number of the device, and information as to whether or not the device is locked. The table may be modified by the user through an LU command.

DEVICE TIMEOUT - A time interval associated with a specific I/O device. If the system expects a response from such a device and this response does not occur within the timeout period, the device is assumed to be inoperative by the system. This feature is necessary to prevent a program from getting "hung up" because it is waiting for a response from a non-functioning peripheral device.

DIRECT MEMORY ACCESS - See Dual Channel Port Controller.

DIRECTORY - A list of programs and files currently stored on a disc subchannel that can be displayed by the user.

DISC - Strictly speaking, the term means the platter(s) with the storage medium only; however the term is also loosely used to mean the entire peripheral including the drive.

DISC-BASED - Refers to an operating system using a disc storage device as an integral part of the operating system.

DISC FORMATTING - The process by which physical track and sector addresses are written in the preamble of each disc track sector. Disc formatting may be performed by the appropriate disc diagnostic. After formatting is completed, the SWTCH program and Disc Backup utility may perform subchannel initialization.

DISC-RESIDENT - A term applied to programs in executable form (absolute) that are stored on disc and brought into main memory for execution by the system in response to a program or operator request, time-of-day schedule or an I/O interrupt.

DISC ROM BOOT - A loader residing in Read-Only Memory that loads (off-line) the Boot Extension from disc storage and transfers control to the Boot Extension. (See also BOOT EXTENSION and STARTUP.)

DISPATCHER - An RTE system module that selects, from the scheduled list, the highest priority program to be executed next. The dispatcher module loads the program into memory from disc (if the program is not already in memory) and transfers control to the program.

DMA - See Dual Channel Port Controller

DMS - See Dynamic Mapping System

DORMANT PROGRAM - A dormant program is one that is "sleeping" or inactive. More specifically, in RTE it is a program that is neither executing, suspended nor scheduled.

## GLOSSARY OF TERMS

DOWN - Status of a device controller EQT that is not available for use.

DRIVER - A software module that interfaces a device and its controller to an operating system. Drivers specified by EQT definitions will go into either a driver partition or into the System Driver area of memory.

DRIVER PARTITION - A block of memory that contains one or more drivers. In RTE-IV, all drivers are in physical memory; however, only the driver partition containing the driver currently being used is included (mapped) in the logical address space.

DRT - See DEVICE REFERENCE TABLE

DUAL CHANNEL PORT CONTROLLER (DCPC) - A hardware accessory that permits an I/O process to transfer data to or from memory directly, or access memory, thus providing a much faster transfer of data. The operating system controls access to the DCPC channels.

DYNAMIC BUFFER SPACE - Additional buffer space allocated to a program after the program code itself. The additional size is determined by the user. Typically used only by assembly language program.

DYNAMIC MAPPING SYSTEM - A hardware accessory allowing partitioned memory systems to address memory configurations larger than 32K words of physical memory.

EMA - See Extended Memory Area

EQT - See Equipment Table

EQT EXTENSION - A method for increasing the size of an Equipment Table entry's buffer space, during system generation, that gives the specified I/O driver more words of storage space than are available in the EQT temporary storage area.

EQUIPMENT TABLE (EQT) - A table in memory associating each physical I/O device controller with a particular software processing routine (driver). For a given device, the EQT provides status information, temporary storage and parameter passing services (see also Device Reference Table and Interrupt Table).

EXEC - One of the RTE system modules that interfaces user programs to the operating system.

EXTENDABLE FILE - An FMP file that is automatically extended in response to a write request to points beyond the range of the currently defined file. An extent is created with the same name and size as the main, and the access is continued.

EXTENDED MEMORY AREA (EMA) - An area of physical memory that may extend beyond the user's logical address space and is used for large data arrays. Its size is limited only by the amount of physical memory available. An entire array is resident in physical memory although the entire array is not currently in the logical address space.

EXTERNAL REFERENCE - A reference to a declared symbolic name not defined in the software module in which the reference occurs. An external reference is satisfied by another module that defines the reference name by an entry point definition.

FILE - A defined section of memory on a storage device used to store data or programs.

FILE EXTENTS - See EXTENDABLE FILE

FILE MANAGEMENT - The operating system functions associated with maintaining disc files (translating file names to physical disc memory areas; maintaining a directory; checking for security codes; etc.).

FILE MANAGEMENT PACKAGE (FMP) - A collection of subprograms used to access, control and maintain files.

FILE MANAGER (FMGR) - A program that provides FMP file creation, access and manipulation services through FMGR commands entered by the user.

FMGR - See File Manager

FMP - See File Management Package

FOREGROUND - A purely arbitrary name for one of the two types of partitions in RTE; generally used for higher-priority programs. The "foreground" area is synonymous with the real-time area.

GLOBAL TRACKS - Global tracks are a subset of system tracks and are accounted for in the track assignment table. Any program can read/write or release a global track (i.e., programs can share global tracks).

HP-IB - The Hewlett-Packard version of the IEEE standard 488-1975 Digital Interface for Programmable Instrumentation. The HP-IB provides two-way communication between instruments and/or between computers, instruments, or peripherals.

ID SEGMENT - A block of words, associated with each resident program, that is used by the system to keep track of the program's name, software priority field, current scheduling status and other characteristics. Every program must have its own ID segment.

## GLOSSARY OF TERMS

**ID SEGMENT EXTENSION** - A method for increasing the size of an ID segment to save additional information about its associated program. The extensions are used only for EMA programs (see EMA). ID segment extensions are automatically allocated by the generator or loader, but only if sufficient ID segment extensions were specified during system generation.

**INTERRUPT** - The process (usually initiated by an I/O device controller) that causes the computer to signal an executing program, in an orderly fashion, for the purpose of transferring information between a device and the computer.

**INTERRUPT LOCATION** - A single memory location whose contents (always an instruction) are executed upon interrupt by an I/O device controller (same as trap cell).

**INTERRUPT TABLE (INT)** - A table that associates interrupt links with the octal select codes of peripheral devices to specific EQT entries or programs.

**I/O** - A general term referring to any activity between a computer and its peripheral devices.

**I/O CONTROLLER** - A combination of interface card(s), cable, and (for some devices) controller box used to control one or more I/O devices.

**I/O DEVICE** - A physical unit defined by an EQT entry (I/O controller) and subchannel.

**I/O WITHOUT WAIT** - See Class I/O.

**KEYWORD TABLE** - A table of ID segment addresses

**LG AREA** - A group of tracks used to temporarily store relocatable code that can be accessed by the File Manager.

**LIBRARY** - A collection of relocatable subroutines that perform commonly-used (e.g., mathematical) functions. Subroutines are appended to referencing programs or are placed in the memory resident library for access by memory resident programs.

**LOADER** - A program that converts the relative addresses of relocatable programs to absolute addresses compatible with the memory layout of a particular system.

**LOCAL COMMON** - An area of COMMON appended to the beginning of a program and accessible only by that program, its subroutines or segments. This type of COMMON can be specified only during on-line relocation by the loader (LOADR).

**LOCKED DEVICE** - See Logical Unit Lock.

**LOCKED FILE** - A file opened exclusively to one program and therefore not currently accessible to any other program.



LOGICAL MEMORY - Logical memory is the 32K-word (maximum) address space described by the currently enabled memory map. If the System Map is enabled, it describes those areas of physical memory necessary for the operation of RTE-IV. When the User Map is enabled, it describes those areas needed by the currently executing program. DCPC maps describe the address space to/from which the transfer is taking place.

LOGICAL UNIT LOCK - A mechanism for temporarily acquiring exclusive use of an I/O device or devices by a program, to ensure its I/O completion before being preempted by another program.

LOGICAL UNIT NUMBER (LU) - A number used by a program to refer to an I/O device. Programs do not refer directly to the physical I/O device select code number, but rather through the LU number that has a cross-reference to the device.

LU - See LOGICAL UNIT NUMBER

MAILBOX I/O - A Class I/O term applied to a protected buffer that keeps track of the "sender" and "receiver" program for each block of data in the buffer used in program-to-program communication.

MAIN PROGRAM - The main body of a user program (as opposed to the whole program, which may include subroutines or segments).

MAP - Applied to 21MX or XE machines, the term applies to a set of 32 registers that point to 32 pages of physical memory defining a 32K-word logical address space.

MAPPING SEGMENT (MSEG) - The area of an EMA that is currently accessible within the user program's logical address space.

MEMORY PROTECT - A hardware accessory that allows an address (memory protection fence) to be set so that when in protected mode, the locations below that address cannot be accessed by writes or JSB/JMP instructions.

MEMORY-RESIDENT LIBRARY - A collection of reentrant or privileged library routines available only to memory resident programs (in RTE-IV). These routines are included in the disc-resident relocatable library for appending to disc-resident programs.

MEMORY-RESIDENT PROGRAM - A program that executes from a designated area in physical memory and remains in memory, as opposed to a disc-resident program that may be swapped out to the disc or loaded from the disc to another area in memory. Memory resident programs are loaded during system generation (only), and usually are high priority programs with short execution times.

## GLOSSARY OF TERMS

**MOTHER PARTITION** - A partition that may be larger than the maximum logical address space and which may consist of a group of subpartitions. The subpartitions allow many smaller programs to use the memory when the mother partition is not active.

**MSEG** - See Mapping Segment

**MULTIPROGRAMMING** - A technique whereby two or more routines or programs may be executed concurrently by an interleaving process, using the same computer. Multiprogramming is an attempt to improve equipment efficiency by building a queue of demands for resources, achieved by having available in main memory more than one task waiting for resource usage. The concurrent tasks are then multiplexed among each other's wait time intervals.

**MULTI-TERMINAL MONITOR** - A system software module that provides for interactive program development and editing in a multi-terminal environment controlled by a single computer.

**OFF-LINE** - Refers to use of the computer and/or I/O devices by resources other than the RTE operating system or subsystems.

**ON-LINE** - Refers to software or I/O devices recognized and controlled by the main operating system at the time they are being used.

**ON-LINE GENERATOR** - A program that permits use of an existing RTE operating system's services to generate a new system from relocatable software modules found in the File Manager Area. System control can then be transferred to the new operating system through use of a program called SWTCH. (See RTE-IV On-Line Generator Reference Manual.)

**ON-LINE LOADING** - The relocation of programs through use of the Relocating Loader (see RELOCATION).

**OPEN FILE** - A method of gaining access to a specific file to perform a read/write instruction.

**OPERATOR'S CONSOLE** - see SYSTEM CONSOLE

**OPERATING SYSTEM** - An organized collection of programs designed to optimize the usage of a computer system. It provides the means by which user programs interact with hardware and other software. (See also REAL-TIME EXECUTIVE.)

**OVERLAYS** - Also called segments, these are routines that share the same portion of main memory and are called into memory by the program itself (see SEGMENTED PROGRAMS).

**PAGE** - The largest block of memory (1024 words) that can be directly addressed by the address field of a one-word memory reference instruction.

**PARTITION** - A predefined block of memory with a fixed number of pages (redefinable at system boot-up) located in the disc resident program area of memory. The user may divide the disc resident program area into as many as 64 partitions that can be classified as a mixture of real-time and background, all real-time, or all background. Disc-resident programs run in partitions and at least one partition of sufficient size must be defined during system generation to run disc resident programs.

**PERIPHERAL DISC SUBCHANNEL** - A disc subchannel available to the user for read/write operations but for which RTE-IV does not manage nor maintain a track assignment table. It is the user's responsibility to manage these tracks; however, the File Manager may be used to manage peripheral subchannel tracks. A peripheral subchannel must have a logical unit number assignment greater than 6.

**PHYSICAL MEMORY** - Physical memory is the total amount of memory defined at generation or reconfiguration time. It refers to the actual memory in the computer; e.g., page 67 of physical memory is associated with a certain block of actual hardware, whereas the same page might be referred to as "page 5" in a particular block of logical memory.

**POWER FAIL/AUTO-RESTART** - The ability for a computer to save the current state of the system in permanent memory when power is lost, and to restore the system to defined conditions when power returns.

**PRIORITY** - A regulation of events allowing certain actions to take precedence over others in case of timing conflicts.

**PRIVILEGED DRIVERS** - I/O drivers whose interrupts are not processed by the RTE operating system. Such drivers offer improved response time but must perform their own internal housekeeping; i.e., saving status upon interrupt.

**PRIVILEGED INTERRUPTS** - Interrupts that by-pass normal interrupt processing to achieve optimum response time for interrupts having the greatest urgency. Privileged interrupts are handled by privileged I/O drivers.

**PRIVILEGED SUBROUTINE** - A privileged subroutine executes with the interrupt system off (and therefore by-passes the operating system). It allows high-speed processing at the cost of losing use of operating system housekeeping services and real-time response.

**PROGRAM STATE** - Refers to the status of an executable program at any given time. A user program is always in one of four possible states: executing, scheduled, suspended or dormant.

## GLOSSARY OF TERMS

PROGRAM SWAPPING - see Swapping

PURGE - Refers to the act of instructing an operating system to delete a file or program from its directory. Usually used with reference to disc files.

REAL-TIME (RT) - An arbitrary name for one of the two types of partitions in RTE; generally used for higher-priority programs. The real-time area is synonymous with the "foreground" area.

REAL-TIME EXECUTIVE - A collection of software modules comprising the total operating system; e.g., EXEC, SCHED, RTIOC, I/O drivers and various tables. For all practical purposes, Real-Time Executive, operating system and RTE are synonymous terms.

RECORD - A logical subdivision of a file that contains zero or more words, and is terminated by an end-of-record mark.

REENTRANT - Refers to a routine that can be shared by a number of programs simultaneously; i.e., one program can be interrupted in its usage of the routine to permit a higher-priority program to utilize the routine. The first program can then reenter the routine at the point where it was interrupted.

RELOCATABLE LIBRARIES - A collection of commonly-used subroutines in relocatable format. For example:

System Library - subroutines that are appended to each user program and that are unique to the operating system. This allows a user to write programs using operating system routines but which are independent of the operating system for subroutine execution.

DOS/RTE Relocatable Library - a collection of utility subroutines that are primarily accessed by FORTRAN and Assembly Language programs.

FORTRAN Formatters - format subroutines for FORTRAN I/O operations and other programming languages.

RELOCATING LOADER (LOADR) - A HP-supplied program that sets up communications links and forms an absolute load module from a relocatable program. LOADR creates the relocated program in conformance with current system constraints and loader commands entered by the user.

RESOURCE MANAGEMENT - A feature that allows the user to manage a specific resource shared by a particular set of cooperating programs.

RESPONSE TIME - The total amount of time required to bring a real-time program or routine into execution in response to an interrupt, interval timer, call from another program or operator call. Response time is usually measured in microseconds to milliseconds.

ROM BOOT - A loader residing in Read-Only Memory that on-line loads the Boot Extension from disc storage and transfers control to the Boot Extension. The Boot Extension must reside on the disc physical unit 0, track 0, sector 0. (See also Boot Extension and Startup definitions.)

RTE - See REAL-TIME EXECUTIVE

SAM - See System Available Memory

SCHEDULING - Entering a program in the schedule list for execution, either at the next entry into the dispatcher, or at the appropriate time when the program's priority is high enough.

SEGMENTED PROGRAM - A technique for accommodating programs larger than the available logical memory. "Segment" refers to those slices of the program that are brought into main memory as required, and overlay the previous segment.

SELECT CODE - An octal number (10 through 77) that specifies the address of an I/O device interface card.

SIMULTANEOUS PERIPHERAL OPERATIONS ON-LINE (SPOOL) - An RTE feature generally associated with batch operations. There is both in-spooling and out-spooling. In-spooling consists of a program and data being first read in from some peripheral device and placed on the disc. Program reads are translated to disc reads instead of reads from the peripheral device. Program writes are also translated to disc writes instead of peripheral device writes, so that program output is on disc. Out-spooling is the process of taking the program's output from disc to the appropriate peripheral device.

STARTUP - The startup process is initiated by the Boot Extension. During the startup process, the tables, registers and pointers required by the system are established. Control is then transferred to the Configurator.

SUBCHANNEL - One of a group of I/O devices connected to a single I/O controller. For example, RTE driver DVR23 can operate more than one magnetic tape drive through subchannel assignments. In the case of moving head discs, contiguous groups of tracks are treated as separate subchannels. For example, a 7905 disc platter may be divided into four subchannels. Each subchannel is referenced by an LU number.

SUBCHANNEL INITIALIZATION - The process of preparing a disc subchannel for use by the RTE operating system.

## GLOSSARY OF TERMS

**SUBCHANNEL NUMBERS** - Decimal numbers (0-31) associated with the LU numbers of devices with multiple functions on the same device. Each subchannel number is associated with a specific subchannel; e.g., a 2645A terminal could have four subchannels: one for the keyboard, one each for the right and left tape channels, and one for an optional line printer.

**SUBPARTITIONS** - Partitions that are optional subdivisions of a mother partition. Subpartitions have the same type (RT or BG) as the mother partition. Subpartitions are treated like other partitions except that they cannot be used while the mother partition contains an executing program.

**SUBSYSTEM GLOBAL AREA (SSGA)** - An area of memory that consists of all Type 30 modules loaded at generation time. The area is included in the system address space and in the address spaces of programs that access it (Types 17-20, and Types 25-28). The area may be used for data (i.e., COMMON).

**SWAPPING** - A technique whereby an executing program is suspended and transferred to mass storage (because another program needing the same portion of memory has been scheduled). When the interrupting program has terminated, becomes suspended, or becomes eligible to be swapped out, the previously swapped program may be reloaded into memory and resumes execution at the point where it was suspended.

**SWTCH PROGRAM** - A system utility program that transfers an RTE-IV operating system to a specific disc area from which it can be booted up.

**SYNCHRONOUS DEVICE** - Devices that perform I/O operations in a fixed timing sequence, regardless of the readiness of the computer.

**SYSTEM AVAILABLE MEMORY (SAM)** - A temporary storage area used by the system for class I/O, reentrant I/O, automatic buffering and parameter string passing.

**SYSTEM COMMON** - An area of memory that is sharable by programs.

**SYSTEM CONSOLE** - The interactive console or terminal (LU1) that controls system operation and from which all system and utility error messages are issued. In a multi-terminal environment, a system console is distinguished from "user consoles" from which users develop programs.

**SYSTEM DISC SUBCHANNEL** - The disc subchannel assigned to Logical Unit 2 that contains the memory image of the RTE-IV system.

SYSTEM DRIVER AREA - An area for privileged drivers, very large drivers, drivers that do their own mapping or drivers not included in driver partitions. It is included in the system's address space, in the address space of RT and Type 3 BG programs, and optionally in the address space of memory resident programs.

SYSTEM MAP - The 32K-word address space used by the operating system during its own execution.

SYSTEM TRACKS - All subchannel tracks assigned to RTE-IV for which a contiguous track assignment table is maintained. These tracks are located on Logical Unit 2 (system), and 3 (auxiliary).

TABLE AREA I - An area of memory that is included in all address spaces and which includes the EQTs, Device Reference Table, Interrupt Table, Track Map Table, all Type 15 modules, and some system entry points.

TABLE AREA II - An area of memory that contains the system tables, ID segments, all Type 13 modules, and some system table and entry points. Table Area II is included in the address space of the system, real-time programs, Type 3 background programs, and (optionally) memory resident programs.

TIME BASE GENERATOR (TBG) - A hardware module (real-time clock) that generates an interrupt in 10 millisecond intervals. It is used to trigger execution of time-scheduled user programs at pre-determined intervals and for device time-outs.

TIME-OUT - Relating to the state of a peripheral device. When the device has timed-out, it is no longer available for system use (down). Also (noun) the parameter itself; the amount of time RTE will wait for the device to respond to an I/O transfer command before making the device unavailable.

TIME SCHEDULING - The process of automatically scheduling a program for execution at pre-determined time intervals. Program scheduling is established through use of the IT command, and requires that the Time Base Generator be installed in the CPU.

UP - See Device Up

USER MAP - The 32K-word address space used by a user program during its execution.

## 1-1. REAL-TIME EXECUTIVE

The Real-Time Executive is the major control element and communications link within the RTE-IV operating system. It supervises and coordinates all program calls or operator requests for system services. In a typical real-time environment, the Executive handles all decision making and scheduling unless overridden by operator intervention.

A disc-based system, RTE-IV provides for real-time program execution concurrent with full program development services. RTE-IV features multiprogramming, dynamic memory mapping, access to more than one million words of main memory, and an Extended Memory Area (EMA) scheme that offers access to data arrays that are larger than a program's logical address space.

The memory management and mapping provisions allow the central processor unit (CPU) to access from 48K to 1024K words of "physical memory." Physical memory refers to all of memory actually available to the user through the memory management and mapping scheme. "Logical memory" refers to the actual 32K-word address space imposed by the 15-bit address length used in HP 21MX-series computers that is addressable by user programs. RTE-IV automatically handles all addressing and mapping of memory for the user.

Most programs previously written to execute under RTE-M, RTE-II or RTE-III systems are upward compatible with and will successfully operate under RTE-IV. Differences in features between operating systems are itemized in Appendix E.

Significant new features built into RTE-IV include the following:

- \* Improved user interface - reduced user interaction for scheduling system processes (i.e., Relocating Loader, FORTRAN IV, Assembler, etc.).
- \* Program preparation using files.
- \* Assignment of programs to partitions via operator command.
- \* Interactive Relocating Loader.
- \* Greater reliability - hardware parity error recovery, additional checks on operator scheduling command input, improved error messages, and on-line removal of defective pages. Defective pages are those in which parity errors have been detected.



## GENERAL DESCRIPTION

- \* Reconfiguration of I/O and/or main memory during system boot-up without the necessity of regenerating the entire system. Defective pages of memory can be by-passed during the memory reconfiguration process. (Defective pages are those in which parity errors have been detected.)
- \* Increased user code area of up to 27K words.
- \* A memory management scheme that accomodates unusually large data arrays. Implementation is through an easy-to-use Extended Memory Area (EMA). Using EMA, data arrays as large as physical memory may be mapped into the user's logical address space, as required. Typical applications where EMA arrays are particularly useful are as follows:
  - a. Systems with large amounts of data storage, acquisition and processing. Data access within EMA arrays is rapid, requiring no disc accesses as in virtual memory schemes.
  - b. Data acquisition and storage from fast devices at real-time rates.
  - c. Processes involving data access from random locations (e.g., sorting).
  - d. Scientific applications involving large matrices (e.g., inverting a matrix).
  - e. Applications requiring extremely large buffer areas.

## 1-2. SYSTEM HARDWARE

The RTE-IV system operates with the following minimum hardware configuration:

- \* HP 21MX Series Computer with a minimum 48K words of memory (64K is highly recommended for improved memory utilization).
- \* Time Base Generator
- \* Dual Channel Port Controller (DCPC)\*
- \* Dynamic Mapping System
- \* Memory Protect
- \* System Console Device
- \* High Speed Disc Storage
- \* Firmware Accessory Board (FAB) (21MX-E series only)

\* Either an HP Mini-Cartridge Subsystem or High Speed Paper Tape Reader.

### 1-3. SYSTEM SOFTWARE

The complete set of currently available RTE-IV operating system modules and standard subsystems is listed in the RTE-IV Software Numbering Catalog. Optional subsystem modules can be found in the various subsystem Software Numbering Catalogs.

### 1-4. MEMORY MANAGEMENT

The Dynamic Mapping System (DMS) provides the capability of addressing memory configurations larger than 32K words. Up to 1024K words of physical memory can be addressed by the user. The following brief explanation of the mapping and addressing process provides a general overview of system operation. For a more detailed description, refer to the 21MX Series Computer Reference Manual and information given in the "Memory Organization and Management" section of this text.

Addressing more than 32K words is accomplished by translating memory addresses through one of four "memory maps". A memory map is defined as a set of 32 hardware registers that provide the interface between the 32K logical and physical memory. All memory map addressing is performed internally by the system and is transparent to the user. The four memory maps managed by the system consist of a system map that defines the system's logical address space, a user map that defines the user's logical address space, and two Port maps that define a caller's I/O buffer in a DCPC transfer.

### 1-5. MULTIPROGRAMMING

RTE-IV is a multiprogramming system that allows several programs to be active concurrently. Each program executes during the unused central processor time of the others. Scheduling/dispatching modules decide when to execute programs that are competing for system resources. These modules swap disc-resident programs in and out of partitioned memory in accordance with availability of system resources, program priority and time scheduling criteria. The programs may be scheduled by pre-determined time intervals, an external event, operator command or by another program. A scheduled list maintained by the system is automatically scanned every 10 milliseconds or whenever a change is made to the list by a new entry.

Up to 254 programs may be defined by ID segments at one time (an ID segment is a table that describes the program; refer to Appendix A for more information). Additional programs may be relocated and then saved as files by using the File Manager. Thus, the number of readily accessible programs can be increased to the limits of available disc storage.

## GENERAL DESCRIPTION

### 1-6. INPUT/OUTPUT PROCESSING

All I/O and interrupt processing is controlled by the system with the single exception of privileged interrupts (privileged interrupts circumvent the system for faster response time). Input/output operations are performed concurrently with program execution; some programs execute while others are receiving I/O services.

Requests for I/O services are made by EXEC function calls coded into the calling program. The EXEC calls specify the type of transfer (Read, Write, Control) and the desired device. I/O requests for a particular program are queued to the controller I/O list according to the calling program's priority. Automatic buffering for write operations is provided if specified.

In addition to the standard I/O scheduling processes described above, there are a number of other I/O functions available that can improve system performance in a multiprogramming environment:

- \* Device Time-Out -- sets a time-out value for a device to prevent indefinite program suspension because of a malfunctioning device.
- \* I/O Buffering -- automatic buffering on slower devices allows a calling program to initiate an output operation (only) without waiting for completion before resuming execution. A read without wait operation is a function of Class I/O (see below).
- \* Reentrant I/O -- allows a disc resident program to be swapped out from a memory partition and into disc storage when it is suspended for I/O. This, in turn, permits any program to use the partition. The previous status of the swapped program is maintained so that, when it once again achieves highest priority on the scheduled list, it can resume execution and I/O processing at the point of interruption.
- \* Logical Unit Lock -- assigns a logical unit exclusively to a specific program, thus preventing any other program from accessing it until it is unlocked.
- \* Class I/O -- a special set of I/O calls that provide a method for buffering data between devices and user programs and also between programs (mailbox I/O). Class I/O permits a user program to continue execution concurrently with its own I/O (I/O without wait).

### 1-7. RESOURCE MANAGEMENT

Resource management is a user-determined method for cooperating programs to share a common resource in an orderly manner. A "resource" may be anything so defined by the user programs accessing it; an I/O device, a file, subroutine, or a memory location containing volatile data are typical examples.

This sublevel of resource sharing is initially implemented during system generation by defining the number of concurrent resources to be shared. A table of these numbers is set up and maintained by the system. An example of resource sharing would be the updating of commonly-shared data by one program. It would lock the associated resource number to prevent premature access by other programs until the data was updated. See Section IV for a complete description.

#### 1-8. EXECUTIVE COMMUNICATION

EXEC calls are the line of communication between an executing program and system services. The required calls are coded into a program during its development phase. The calls have a structured format plus a number of parameter options that further define the specific operation to be performed.

When an executing program makes a call to EXEC, it attempts to execute a jump subroutine (JSB) to that portion of the system located in the protected area of memory. This causes a memory protect violation interrupt that is then processed by the system. If the call is legal, the system processes the request.

The following is a partial list of system services available to an executing program via EXEC calls:

- \* Perform input and output operations
- \* Allocate and release disc space
- \* Terminate or suspend itself
- \* Load its segment
- \* Schedule other programs
- \* Recover scheduling strings
- \* Obtain the time of day
- \* Time-schedule program execution
- \* Obtain status information on partitions

See Section IV of this manual for complete descriptions and format considerations of EXEC calls.

## GENERAL DESCRIPTION

### 1-9. OPERATOR COMMANDS

The operator maintains final control of RTE-IV system operation through commands entered via the system console. These commands and their parameter options enable the operator to monitor current system status and/or modify system operation. The following is a partial list of operator control functions:

- \* Turn programs on and off
- \* Suspend and restart programs
- \* Examine the status of any partition, program, I/O device or controller
- \* Schedule programs to execute at specified times
- \* Change the priority of programs
- \* Declare I/O controllers or devices up or down
- \* Dynamically alter the logical I/O structure and buffering designations
- \* Delete temporarily-loaded disc resident programs from memory
- \* Examine and dynamically alter an I/O device's time-out parameter
- \* Release tracks assigned to dormant programs
- \* Initialize the real-time clock and display the time
- \* Change program size (dynamic buffer area)
- \* Assign programs to partitions
- \* Remove reserved status of partition

See Section III of this manual for descriptions and parameter options of all operator commands.

### 1-10. SYSTEM CONFIGURATION

Memory resident and disc resident user programs, system modules, library routines, device drivers and Real-Time Executive modules are incorporated into a configured RTE System. The RTE software is modular and flexible enough to permit user programs and I/O device drivers to be configured into a real-time system that is tailored to an installation's exact requirements.

Using the Real-Time On-Line Generator (RT4GN) and SWTCH, the relocatable software modules and user programs are converted into a configured real-time system in memory-image binary format. The configured system is then loaded (bootstrapped) into the computer from the system area of the disc. Any remaining disc storage is dynamically allocated by the configured system to user programs or is utilized by the scheduler for swapping operations.

#### 1-11. MULTI-TERMINAL OPERATIONS

The Multi-Terminal Monitor (MTM) provides concurrent management of multiple user consoles. Each user is provided with his own File Manager for command input. Individual copies of user programs are created whenever they are initiated at MTM consoles thus allowing concurrent execution of Assemblers, Editors, Generators, etc. See Section IX of this manual for a detailed discussion of MTM operation.

#### 1-12. SYSTEM UTILITY PROGRAMS

Standard system utilities are on-line programs that run under the RTE operating system and are called by the user to perform various program preparation, system status and housekeeping processes. The presence of any utility program in the system is optional, depending upon site-specific requirements. The programs available are:

- \* Relocating Loader (LOADR)
- \* File Management Package (FMP)
- \* Interactive Editor (EDITR)
- \* Batch Spool Monitor (BSM)
- \* On-Line Generator (RT4GN)
- \* Disc Backup
- \* Disc Update
- \* System Status Program (WHZAT)
- \* KEYS and KYDMP Programs
- \* Track Assignment Table Log Program (LGTAT)
- \* Debug Subroutine (DBUGR)

## GENERAL DESCRIPTION

### 1-13. RELOCATING LOADER

The Relocating Loader program accepts user-written relocatable programs and outputs absolute load modules in conformance with loader control command parameter options specified by the user. Other command parameters cause the loader to list system status information; i.e., currently available programs; or purge unwanted, permanently loaded programs from the system. See Section VII of this manual for a detailed discussion of LOADR operation.

### 1-14. FILE MANAGEMENT PACKAGE (FMP)

The File Management Package is a set of programs (FMGR and D.RTR) and subroutines that provide disc file housekeeping services. Service may be acquired either programatically or through interactive user commands. Files may be created, renamed, copied, purged, listed, concatenated or otherwise manipulated on disc tracks under control of the File Management Package. See the Batch Spool Monitor Reference Manual, Sections II and III for complete information regarding use of FMP.

### 1-15. INTERACTIVE EDITOR

The Editor (EDITR) program is used to create and/or edit (modify) lines of text in a source file under development or in a data file in ASCII format. See the RTE Interactive Editor Reference Manual for further information.

### 1-16. BATCH SPOOL MONITOR

The Batch Spool Monitor is a set of programs and subroutines that are used to perform disc-based job processing. That is, jobs or data can be input from a disc file and data can be output on a disc, with all the necessary I/O being performed independently of batch processing. BSM also provides a means for input and output spooling of data. See the Batch Spool Monitor Reference Manual, Section IV through VII, for more information.

### 1-17. WHZAT

The WHZAT program provides status information regarding the current system environment. Two different types of information can be displayed: a list of all active program and their current status, or a list of all partitions with their sizes and current status (occupied or non-occupied). See the RTE Utility Programs Reference Manual for more information.

## 1-18. DBUGR

The DBUGR subroutine can be appended to a user program through use of the Relocating Loader. It can then aid the user in checking for logical errors in a program through interactive control commands. Debugging is performed at the Assembly Language level. See the subset of DBUGR control commands described in the DBUGR--Interactive Debugging section of this manual or the DBUGR Reference Manual for a complete description of all DBUGR functions.

## 1-19. ON-LINE GENERATOR

The On-Line Generator permits use of an existing RTE-IV system to configure a new RTE-IV system according to user specifications. Generation can be directed from an answer file, logical input unit or operator console. See the RTE-IV On-Line Generator Reference Manual for more information.

## 1-20. SWTCH

The SWTCH program permits a user to transfer an RTE-IV operating system file created by the On-Line Generator to a specific area of a disc from which it can be booted up. See Section V of the RTE-IV On-Line Generator Manual for more information.

## 1-21. DISC BACKUP

The Disc Backup programs can be used either on-line or off-line to transfer data from disc to magnetic tape or vice versa, copy data from disc to disc, verify successful transfers or copy operation, and to initialize a disc cartridge. See the RTE Utility Programs Reference Manual for more information.

## 1-22. DISC UPDATE

The Disc Update process can be used to replace disc cartridge files with files stored on an HP mini-cartridge tape. The primary purpose is to update master software discs with either HP software distributed on mini-cartridges or user-written program modifications. See the RTE Utility Programs Reference Manual for more information.



## GENERAL DESCRIPTION

### 1-23. KEYS AND KYDMP

The KEYS and KYDMP programs are used to create user-defined command sets for programming the soft keys on the HP 2645A Display Station. Softkeys provide the capability to enter entire sequences of commands with a single keystroke. The advantages are speed of entry and a significant reduction in operator errors during terminal entry sessions. See the RTE Utility Programs Reference Manual for more information.

### 1-24. LGTAT

The LGTAT program logs and displays the status of the system and auxiliary (only) disc tracks. See the RTE Utility Programs Reference Manual for more information.

### 1-25. PROGRAMMING LANGUAGES

The language translators available for user program development under the RTE system are RTE FORTRAN IV, RTE Assembler, HP Micro Assembler and BASIC 1000/D.

#### RTE FORTRAN-IV

RTE FORTRAN IV is a problem oriented programming language that is translated by a compiler. The FORTRAN IV compiler executes in RTE and accepts source programs from either an input device or FMGR file. The resultant relocatable object programs and listed output files are stored in FMGR files or output to specified devices. For further information, see the RTE FORTRAN-IV Programmer's Reference Manual.

#### RTE-IV ASSEMBLER

The RTE-IV Assembly Language is a machine-oriented programming language. Source programs written in this language are accepted by the Assembler from either input devices or disc files and translated into absolute or relocatable object programs. Absolute code is output in binary records suitable for execution on systems other than RTE-IV. For further information, see the RTE-IV Assembler Reference Manual.

#### RTE MICRO-ASSEMBLER

The Micro-Assembler is part of an optional support package for on-line users of special microprogrammed instructions. The Micro-Assembler translates source code into object microprograms. For further information, see the Micro-Assembler Reference Manual.

## REAL-TIME BASIC/1000D

Real-Time BASIC is an optional, conversational programming language that is easily learned, even by users without previous programming experience. Each statement entered by the user is immediately checked for correct syntax by the Real-Time BASIC Interpreter. No separate compilations or assembly operations are involved. A partly completed program can be run at any time to confirm that it executes as the user intended. See the Multi-User Real-Time BASIC Reference Manual.

## QUERY

QUERY is an English-like language used to access the HP data base management subsystem called IMAGE/1000. IMAGE/1000 is itself an optional subsystem that can be ordered for RTE-IV system applications involved with large data base considerations. In addition to the use of QUERY, the data base can also be accessed through RTE-IV FORTRAN, Assembler or Real-Time BASIC applications programs. See the IMAGE/1000 Reference Manual for further information.

## 1-26. RTE-IV SYSTEM SUMMARY

The HP Real-Time Executive IV software system is a multiprogramming, multi-user and multi-partitioned system that provides priority scheduling, interrupt handling and program preparation capabilities.

With multiprogramming, a number of data acquisition systems or test stands can be operated simultaneously on a 24-hour a day basis. Data reduction and report preparation functions can be scheduled to execute in the background area during times when real-time activities permit. The same computer can also be used by the programming group for ongoing development work with RTE background compilers for FORTRAN IV, and with the HP Assembler, Editor, and other auxiliary programs. Programs can be added to the system on-line. For system protection, new programs can be debugged while the memory protect fence and the Dynamic Mapping System maintains the integrity of the system area and other user programs.

Scheduling of all programs is based on priority. External events can interrupt current operations to schedule programs for execution, or a program can be scheduled by an operator request, a program request, or on a real-time clock basis. Priorities are assigned by the user during generation or on-line loading, and may be changed by an operator request.

## GENERAL DESCRIPTION

The system controls I/O processing through a central routine that directs requests and interrupts to the appropriate device driver subroutine. For efficiency, programs awaiting I/O are suspended to let other programs use the computer. Outputs to slow devices can be buffered. For processes that cannot tolerate ordinary system overhead, a privileged interrupt option lets a device contact its driver directly without going through the Executive. Program-to-program communication is provided through a mailbox (Class I/O) scheme.

The operator retains final control of system operation via commands entered through the system console. The operator can turn on programs, make status checks or perform other operations.

Configuration is efficient. System generation is performed on-line using interactive operator dialog or pre-built answer files. This results in an operating system configured for a specific hardware system.

System boot-up is the process of loading the operating system software into memory so that it is ready for execution. Boot-up begins by using either the Disc Loader ROM or Bootstrap Loader to load the Boot Extension into memory from track 0, sector 0 of the system disc subchannel. The Boot Extension, in turn, loads the operating system into memory.

At this point, the user has the option of either completing a "standard" system boot-up procedure as described in this section, or reconfiguring the current I/O and memory assignments as described in Section XII, "Memory and I/O Reconfiguration." In a standard boot-up, the operating system immediately completes the rest of the initialization process as follows:

1. Displays a SET TIME message.
2. Executes a startup program (optional).
3. Passes control to the File Manager (FMGR), which tries to execute a procedure file named WELCOM. If the WELCOM file does not exist on the system, the FMGR displays a FMGR -006 error message.

If memory and/or I/O reconfiguration are to be performed during system boot-up, completion is delayed and an interactive Configurator program is scheduled via S-register settings to make the new memory and I/O assignments. At the end of the reconfiguration process, control is returned to the system to complete the boot-up procedure as described above.

Use the procedures described below to perform a standard system boot-up. Use the procedures described in Section XII to perform a boot-up with I/O and memory reconfiguration.

#### 2-1. BOOT LOADERS AND BOOT EXTENSION

The Disc Boot Extension can be loaded into memory from the disc using either the Disc Loader ROM or Bootstrap Loader.

#### 2-2. DISC LOADER ROM

The Disc Loader ROM can be used to load the Boot Extension if the Boot Extension resides on physical track 0, sector 0 of the system disc. Refer to the HP 12992 Loader ROM's Installation Manual (12992-90001) for a description of the S-register setting to load the Boot Extension into memory.

## STANDARD BOOT-UP PROCEDURES

An example of a standard system boot-up using the 12992B RPL-compatible 7905/7906/7920 Disc Loader ROM is as follows:

1. Select the S-register for display on the computer front panel.
2. Press CLEAR DISPLAY.
3. Set the S-register bits as follows:

Bits ----	Enter: -----
0-2	Surface number of the disc where the RTE-IV system subchannel starts (surface numbers start at 0).
3-4	0 (reserved)
5	0 for standard boot-up
6-11	Octal select code of the disc.
12	1 to indicate a manual boot from the S-register.
13	0 (reserved)
14-15	Loader ROM selection (number of the ROM cell containing the Disc Boot Loader).

4. Press PRESET, IBL and PRESET (again) to load contents of Disc Loader ROM. A successful load is indicated if the OVERFLOW indicator does not light up.
5. Press RUN.

### EXAMPLE:

1. Assume a standard boot-up from ROM #2, with a 7906 in select code 21 and surface 0.
2. Set the S-register = 112100.
3. Press PRESET, IBL , PRESET (again) and RUN.

## 2-3. BOOTSTRAP LOADER

The Bootstrap Loader is used to load the Boot Extension into memory if the Boot Extension does not reside on the physical track 0, sector 0 of the system disc, or if the Disc Loader ROM is not available. The procedure is as follows:

1. Select the S-register for display on the computer front panel.
2. Press CLEAR DISPLAY.
3. Set the S-register bits as follows:

Bits: -----	Enter: -----
0-5	0
6-11	Octal select code of input device (e.g., photoreader)
12-15	0

4. Press PRESET, IBL and PRESET (again) to load the Bootstrap Loader. A successful load is indicated if the OVERFLOW indicator does not light up.
5. Press RUN.

When the HLT 77B occurs, clear the S-register, set the P-register to octal 100 and press RUN to continue.

## 2-4. BOOT EXTENSION EXECUTION

The disc Boot Extension uses the S-register to communicate with the configurator program (see Section XII). Do NOT change the S-register contents until the system boot-up procedure is completed and the SET TIME message is displayed.

### 3-1. INTRODUCTION

User control of an RTE operating system and the monitoring of system status are performed through a two-way dialog between the system and user. The system displays various status or error messages that may or may not require human intervention. The user communicates with the system through operator requests entered at the user console. Using these commands and their various parameter options, an operator may interrupt RTE at any time to determine current system status, correct error conditions or modify system performance. The operator commands and their function are summarized in Table 3-1; complete descriptions are given later in the section.

### 3-2. COMMAND STRUCTURE

The operator first gets system attention by pressing any key on the system console (LU1). On the system console, RTE responds with an asterisk (\*) prompt to indicate system attention. The user then types a command which is a two-character request word (e.g., ON, UP, etc.), followed by the appropriate parameters separated by commas.

Each command is parsed or resolved by a central routine that accepts certain conventions. Command syntax is described in Table 3-2. This syntax and the command conventions described below must be followed exactly to satisfy system requirements.

### 3-3. COMMAND CONVENTIONS

- \* When a command is entered, the items outside the brackets are required symbols. Items inside the brackets are optional.
- \* Two commas in sequence defaults a parameter to zero.
- \* Each command entered must be completed with an end-of-record terminator (RETURN key on a CRT or TTY system input device).

## OPERATOR COMMANDS

- \* An error made while entering a command parameter can be corrected by using the BACK SPACE key on a CRT system input device (the CONTROL and A keys struck simulataneously will delete the last character entered on TTY input devices). To delete an entire line, use the DEL key (RUBOUT key on TTY devices). Corrections to a command must be made before the RETURN key is pressed or the system will issue an error return. Note that line feed is supplied by the system.
- \* Whenever the operating system is rebooted, parameters changed by user command will be restored to their original values established during system generation.

Table 3-1. Operator Command Summary

Command Format	Function	See Page
AB	Aborts current batch program.	3-5
AS	Assigns program to a partition.	3-6
BL	Examines and sets buffer limits.	3-7
BR	Sets a break flag in named program's ID segment.	3-8
DN	Declares an I/O controller or device unavailable.	3-9
EQ	Examines status of any I/O device, and dynamically alters device buffering assignments.	3-10 3-11
FL	Buffer flush command used in conjunction with Multiple Terminal Monitor (MTM) only.	3-12
GO	Restarts programs in an operator suspension state (there several other suspension states).	3-13
IT	Sets time intervals for programs.	3-15
LG	Allocates LG area.	3-16
LS	Sets LS area pointer.	3-17
LU	Examines and alters device Logical Unit assignments.	3-18 3-19



Table 3-1. Operator Command Summary (cont'd)

Command Format	Function	See Page
OF	Terminates program execution.	3-20
ON	Schedules a program for execution.	3-21
PR	Changes the priority of programs.	3-23
RT	Releases program's disc tracks.	3-24
RU	Schedules a program for immediate execution.	3-25
SS	Operator suspends a program.	3-27
ST	Examines the status of a program or partition.	3-28
SZ	Examines or changes program size.	3-30
TI	Prints the current time.	3-33
TM	Sets the real-time clock.	3-34
TO	Examines and dynamically alters an I/O controller's time-out parameter.	3-35
UP	Declares an I/O controller and associated devices as available.	3-36
UR	Unreserves a previously reserved partition.	3-37

OPERATOR COMMANDS

Table 3-2. Operator Command Syntax Conventions

Item	Meaning
UPPER CASE ITALICS	These words are literals and must be specified as shown.
lower case italics	Symbolic representations indicating what type of information is to be supplied. When used in text, the italics distinguishes them from other textual words.
[,item]	Items with brackets are optional. However, if item is not supplied, its position must be accounted for with a comma; this causes item to automatically default.
<pre>--  ,item 1    ,item 2    ,item 3   --</pre>	Indicates that exactly one item may be specified.
<pre>,item 1 ,item 2 ,item 3</pre>	Indicates that there is a choice of entries for the parameter, but one parameter must be specified.
...(row of dots)	This notation means "and so on."

## 3-4. RTE-IV OPERATOR COMMANDS

All operator commands are described below in alphabetical sequence. A carriage return to terminate a command entry is not illustrated, but is assumed in every case.

## AB (abort)

Aborts the current program running under batch. Since FMGR (not a copy of FMGR) is the Batch Spool Monitor, the command applies only to "sons" of FMGR. The format is

```

      /,0 \
AB
      \,1 /

```

where:

- 0 is the default case. It terminates and removes from the time list the current batch program that is executing, scheduled, or operator suspended. It also terminates batch programs that are I/O, memory or disc suspended the next time they are scheduled. Disc tracks are not released.
- 1 immediately terminates the batch program and removes it from the time list, and releases all disc tracks. If suspended for I/O, a system generated CLEAR request is issued to the driver.

When the File Manager is waiting on a program it is running (e.g., ASMB), the AB command functions like the command

```
OF,name
```

If the File Manager is dormant or non-existent in the system, the AB command causes the error message ILLEGAL STATUS to be printed. If the File Manager is not dormant and is not running a program, AB functions like the command

```
BR,FMGR
```

Note that an AB command from an MTM terminal functions differently and has a different meaning. See the MTM section of this manual.

## OPERATOR COMMANDS

### AS (assign partition)

Assigns a program to a partition. The partition does not have to be reserved. The format is

AS,xxxxx,yy

where:

xxxxx = the program name  
yy = the partition number (1-64)

Program xxxxx will be assigned to partition yy. If yy = 0, the program will be unassigned and can be dispatched to any partition of the proper type large enough to run the program.

If the program is not dormant or is still resident in any partition (i.e., saving resources, operator suspended or serially reusable), the error 'ILLEGAL STATUS' will be returned and the input ignored. Partition yy must also be large enough to run program xxxxx. If not, the error 'ILLEGAL PART`N`' will be returned. Trying to assign a program to an undefined partition will also generate the 'ILLEGAL PART`N`' message.

If the program named xxxxx cannot be found, a "NO SUCH PROG" error message will be issued.

## BL (buffer limits)

Examines or modifies current buffer limits. The format is

BL[lower limit, upper limit]

where:

BL without parameters displays previously set upper and lower limits.

lower limit is the lower limit number.

upper limit is the upper limit number.

Setting upper and lower memory limits with this command can prevent an inoperative or slow I/O device from monopolizing System Available Memory. Each time a buffered I/O request is made (Class I/O requests are buffered), the system adds up all the words in the I/O requests queued to that entry and compares the number to the upper limit set by this command (or during generation). If the sum is less than the upper limit, the new request is added to the queue. If the sum is larger than the upper limit, the requesting program is suspended in the general wait (STATUS = 3) list.

When a buffered I/O request completes, the system adds up the remaining words in the I/O requests queued to the EQT entry and compares the number to the lower limit set by the command (or during generation). When the sum is less than the lower limit, any programs suspended for exceeding the buffer limits on this EQT are rescheduled.

Any program with a priority of 1 through 40 will not be suspended for buffer limit, so that alarm messages, etc., are not inhibited.

## OPERATOR COMMANDS

### BR (break)

Sets an attention flag in a program's ID segment. The format is

BR,name

where name is the name of the program.

The BR command allows an operator to break the execution of a program if the program requests this via the IFBRK system subroutine. When BR is executed, a break flag in the named user program's ID segment is set. The user's program can call the HP-supplied subfunction that will test the break flag and then act accordingly. The calling sequence of the subfunction is:

I=IFBRK(IDUMY)

where IDUMY is a dummy parameter to make the call appear as a function (IDUMY need not be supplied in Assembly Language). The returned value will be negative if the break flag is set, and positive if it is not. If the flag is set, it will be cleared by IFBRK. See the Multi-Terminal Monitor section for variations to the BR command for operation under MTM.

## DN (down)

Declares an I/O controller or device down (i.e., unavailable for use by the RTE system). The format is

```

      /,eqt\
DN
      \,,lu/

```

where:

eqt            is the EQT entry number of the I/O controller to be set down.

lu            is the LU entry number of the I/O device to be set down.

Setting an I/O controller (EQT entry) down effectively sets all devices connected to the I/O channel down by blocking any I/O operations on the select code. The state of the devices (LU's) associated with the select code are unchanged.

Setting the I/O device (LU) down will make only the specific device unavailable. However, all other LU's pointing to the device will also be set down. Other devices using the device's I/O select code are unaffected.

The I/O controller or I/O device remains unavailable until the I/O controller is set up by the UP command. The operator might set a device down because of equipment problems, tape change, etc.

## OPERATOR COMMANDS

### EQ (status)

Prints the description and status of an I/O controller, as recorded in the EQT entry. The format is

EQ,eqt

where:

eqt is the EQT entry number of the I/O controller.

The status information is printed as:

select code DV.nn D B Unn status

where:

select code is the I/O select code number.

DV.nn is the driver routine.

D is D if is DMA required; 0 if not.

B is B if automatic output buffering is used; 0 if not.

Unn is the last subchannel addressed.

status is the logical status:

0 = available

1 = I/O controller unavailable (down)

2 = I/O controller unavailable (busy)

3 = waiting for DMA assignment

Note that if eqt is 0, it is a bit bucket, as is any associated LU.



## EQ (buffering)

Changes the automatic output buffering designation for a particular I/O controller. The format is

```
EQ,eqt      /,UNbuffer\  
            \,BUffer  /
```

where:

eqt            is the EQT entry number of the I/O controller.

UNbuffer      turns off buffering.

BUffer        turns on buffering.

When the system is rebooted from the disc, all buffering designations are reset to the values originally specified during generation.

## OPERATOR COMMANDS

### FL (flush)

Eliminates buffered output to an I/O device. The format is

lu>FL

where:

lu is the Logical Unit Number of the interrupting user console.

The FLush command can only be used in conjunction with the Multi-Terminal Monitor (MTM), and is illegal if entered from the system console (LU1).

Other methods for clearing the buffer are using the EXEC call:

CALL EXEC (3,23B,lu)

or the File Manager command:

:ON,FMGR  
:CN,lu,23B

## GO (reschedule)

Reschedules a program previously suspended by the SS command or a Suspend EXEC Call. The format is

```

/GO \
      ,name [,p1 [...[,p5]]]]]
\GOIH/

```

where:

name is the name of a suspended program to be scheduled for execution.

p1 ... p5 is a list of parameters to be passed to name only when name has suspended itself (see Suspend EXEC Call in Section IV). The parameters are ignored if name was suspended with the SS command.

The GO command is illegal if the program has not been suspended previously by the operator or has not suspended itself.

Parameters p1 through p5 can be entered in ASCII or numeric form. Octal numbers are designated by the "B" suffix and negative numbers by a leading minus sign. For example:

```
GO,name,FI,LE,31061B
```

GO (reschedule)...cont'd

Note that only two ASCII characters per parameter will be returned by a RMPAR subroutine call: if one is given, the second character is passed as a blank (blank = 40B). If the first parameter is ASCII "NO" it must then be repeated (the system interprets it as "NOW" in the GO command). For example:

GO,name,NO,NO,FI,3,4,5

is interpreted as shown below. NO (NOW) is not used except to push out the parameters.

NO  
FI  
3  
4  
5

After a program has suspended itself and is restarted with the GO command, the address of the parameters passed by GO is in the B-register. An immediate call to the library subroutine RMPAR retrieves the parameters (see Section IV, Suspend EXEC Call). If the program has not suspended itself, the B-register is restored to its value before suspension and the parameters are ignored.

The program may also recover the ASCII command string (up to 80 characters typed after the prompt) that scheduled it by using the String Passage EXEC call (see Section IV). If the program was rescheduled with a GOIH (inhibit string passage) or if the program has not suspended itself, the command string is not passed.

## IT (Interval Timer)

Sets time values for a program so that it automatically executes at selected times when scheduled with the ON command. The format is

```
IT,name [res,mpt[,hr,min[,sec[,ms]]]]
```

where:

name is the name of the program.

res is the resolution code:

```
1 - tens of milliseconds
2 - seconds
3 - minutes
4 - hours
```

mpt is a number from 0 to 4095 and is used with res to give the actual time interval for scheduling (see below).

```
hr      hours
min     minutes      sets an initial start time.
sec     seconds
ms      tens of ms.
```

The resolution code (res) is the units in time to be multiplied by the execution interval value (mpt) to get the total time interval. Thus, if res=2 and mpt=100, name would be scheduled every 100 seconds. If hr,min,sec and ms are present, the first execution occurs at the initial start time specified by these parameters (the program must be initialized with the ON command.) If the parameters are not present (e.g., IT,name), the program's time values are set to zero and the program is removed from the time list. The program can still be called by another program or started with the ON,name,NOW or RU command.

When the system is rebooted from the disc, time values set by the IT command are lost, and the original time values set at original load time are reinstated.

The IT command is similar to the Execution Time EXEC Call (See Section IV). For example:

The commands

```
IT,WHZAT,2,5
ON,WHZAT,NO
```

will cause WHZAT to execute every five seconds, starting now.

## OPERATOR COMMANDS

### LG (LG tracks)

Allocates or releases a group of disc tracks for the LG area. LG tracks may be used as temporary storage for relocatable code in FMGR operations. The format is

LG,numb

where:

numb=0            (zero) releases the allocated LG area.

numb>0            release currently allocated LG tracks and then  
                  allocate numb contiguous tracks for an LG area.

Enough LG tracks for storing relocatable code must be allocated before storing into this area. Insufficient tracks cause the program to abort and one of the following diagnostics to be displayed on the system console:

I006 -            LG area not defined.

I009 -            LG area overflow.

An LG request should not be used while anyone is using the LG tracks. Doing so may result in the message

LGO IN USE

being displayed on the system console, and no change in the current number of LG tracks. In most cases, however, the attempt to do so results in an I006 error being issued.

## LS (source file)

Designates the disc Logical Unit number and starting track number of source code stored in the track pool prior to an EDITR operation on the code. The format is

LS,disc lu,trk numb

where:

disc lu        is the Logical Unit number of the disc containing the source file.  
              2 or 3 = system or auxiliary disc units.  
              0        = eliminate the current source file designation.

trk numb       is the starting track number (decimal) of the source code.

LS replaces any previous declarations with the current source code area. Only one area may be declared at a time.

OPERATOR COMMANDS

LU (assignment)

Prints the EQT entry number, device subchannel number, and I/O device status associated with a Logical Unit number. The format is

LU,lu

where lu is a Logical Unit number from 1 to 63.

Example:

```
          LU # 7 = E12 S 1 D
                ^   ^   ^   ^
Logical Unit number -----+
EQT number-----+
subchannel number-----+
I/O device status (down in this case)--+
```

If the Logical Unit's device is unavailable (down), a D is printed as the status; otherwise the position is left blank.



## LU (reassignment)

Changes a Logical Unit number assignment. The format is

```

LU,lu /,eqt[,subch numb]\
      \,0 /

```

where:

- lu is a Logical Unit number from 1 to 63 (decimal).
- eqt is an EQT entry number to assign lu.
- eqt if zero (0) lu becomes the bit bucket.
- subch numb is a subchannel number (0 to 31) to assign to lu.

The restrictions on changing Logical Unit assignments are:

- a. LU1 (system console) must be an interactive console device. Note that if LU1 is changed, the new console will print a double asterisk (\*\*).
- b. LU2 (system disc) and LU3 (auxiliary disc) cannot be changed to another EQT entry number.
- c. An LU cannot be changed to point at the same device as LU2 or LU3.

When an irrecoverable problem occurs on an I/O device, the operator can bypass the downed device for future requests by reassigning the Logical Unit number to an operable device on another select code.

When the system is rebooted from the disc, all LU assignments are reset to those originally established during generation.

Section V, Input/Output, explains Logical Unit numbers, equipment table entry numbers, and subchannel numbers in detail.

OF (terminate)

Terminates a program or removes a disc resident program that was loaded temporarily on-line into memory but not permanently incorporated onto the protected system disc. For options 1 and 8 below, the message "name ABORTED" will appear for programs (but NOT segments) after the command is executed. The format is

```

OF,name  /,0 \
         ,1
         \,8 /
    
```

where:

name is the name of the program.

0 terminates and removes the named program from the time list the next time it is scheduled. The program's disc tracks are not released.

1 immediately terminates the named program, removes it from the time list, and releases all disc tracks. If suspended for I/O, a system-generated request to clear the device is issued to the driver.

8 immediately terminates the named program. If the program is temporary program loaded on-line, it is removed from the system (see the Relocating Loader section of this manual).

For programs with segments, the OF, name, 8 command must be used on the segments as well as the main.

Of,name,8 will not remove permanently loaded programs, since their ID segments on the disc are not altered by this request. A permanently loaded program is defined as a program loaded during generation, or on-line with the LOADR and with a copy of its ID segment in both memory and on the disc. For temporary programs loaded on-line, the ID segment is blanked to make it available for use by another program loaded with the LOADR.

The tracks (if system tracks) containing the program are released. If the program had been stored on File Manager tracks, those tracks remain as File Manager tracks and are not returned to the system.

If the program is I/O suspended, a system generated clear request is issued to the driver. The OF,name,8 command must then be entered a second time to permanently remove the program from the system.

A permanently loaded disc resident program may only be removed permanently with the LOADR as described in Section VII.

ON (schedule)

Schedules a program for execution. Up to five parameters and the command string may be passed to the program. The format is

```

/ON \
      ,name[,NOW] [,p1[,...[,p5]]]]]
\ONIH/
    
```

where:

name is the name of a program.

NOW schedules a program immediately that is normally scheduled by the system clock (see IT).

p1 ... p5 are parameters passed to the program when it is scheduled.

Parameters p1 through p5 are the ones passed by RMPAR as described under Comments in the Program Schedule EXEC Call in Section IV. Refer also to XTEMP words 1 through 5 in the program's ID segment (see Appendix B). Note that any parameters not entered as part of the ON command will be returned as zeros by a call to RMPAR.

Parameters p1 through p5 can be entered in ASCII or numeric form. Octal numbers are designated by the "B" suffix and negative numbers by a leading minus sign. For example:

```
ON,name,FI,LE,31061B
```

Note that only two ASCII characters per parameter will be returned by a RMPAR subroutine call; if only one is given, the second character is passed as a blank. (blank = 40B). If the first parameter is ASCII "NO" then it must be repeated (the system interprets it as "NOW" in the ON command). For example:  
For example:

```
ON,name,NO,NO,FI,3,4,5
```

is interpreted as

```

NO
FI
3
4
5
    
```

The program can recover the ASCII command string (up to 80 characters typed after the prompt) by using the String Passage EXEC call (see Section IV). The ONIH command inhibits the passage of the command string.

ON (schedule) ...cont'd

String Passage Example:

ON,name,FILE1,FILE2,MISCINFOSTRING,....,3

If the resolution code in the ID segment of the program is not zero, RTE places the program in the time list for execution at specified times (unless NOW appears; in which case, the program is scheduled and put into the time list immediately). The resolution may be non-zero as a result of:

- a. Generation
  - 1. With a resolution code in the name record
  - 2. Entry of a resolution code during parameter input phase.
- b. The IT command.
- c. Scheduling the program with absolute start time or offset by some program in the system (see EXEC calls in Section IV).

Note that if there is no partition large enough to run the program, or if the program is assigned to a partition that is too small or does not exist, the error message 'SIZE ERROR' will be reported. Conditions under which the error message could be output when attempting to run are:

:SP,xxx  
reboot and reconfigure memory to remove partitions large enough for this program.  
:RP,xxx  
:RU,xxx

## PR (priority)

Changes the priority of a program. The format is

PR,name,numb

where:

name            is the name of the program.

numb            is the new priority.

One (1) is the highest priority, and 32767 is the lowest. When the system is restarted from the disc, the priority of name resets to the value set by the generator or LOADR.

RT (release tracks)

Releases all disc tracks assigned to a program. The format is

RT,name

where:

name is the program whose tracks are to be released.

The RT command is illegal if the named program is not dormant. If the program is dormant, all tracks assigned to the program are released.

Any tracks released as a result of this command cause all programs in disc track allocation suspension to be rescheduled. More information on disc tracks may be obtained from the system program LGTAT, described in the RTE Utility Programs Reference Manual.

## RU (run)

Immediately schedules a program without affecting its entry in the time list. Up to five parameters and the command string may be passed to the program. The format is

```

/ RU \
      ,name[,p1[,...[,p5]]]]]
\ RUIH/

```

where:

name            is the name of a program.

p1 ... p5        are parameters passed to the program when it is scheduled.

The RU command is used when the operator desires to run a program without affecting its entry in the time list.

Parameters p1 through p5 are passed by RMPAR as described in the the Program Schedule EXEC Call in Section IV.

Note that any parameters not entered as part of the RU command will be returned as zeros by a call to RMPAR.

Parameters p1 through p5 can be entered in ASCII or numeric form. Octal numbers are designated by the "B" suffix and negative numbers by a leading minus sign. For example:

```
RU,name,FI,LE,31061B
```

## RU (run)....cont'd

Note that only two ASCII characters per parameter will be returned by a RMPAR subroutine call; if only one is given, the second character is passed as a blank (blank = 40B). If the first parameter is ASCII "NO" then it must be repeated (the system interprets it as "NOW" in the RU command). For example:

```
RU,name,NO,NO,FI,3,4,5
```

is interpreted as shown below. NO(NOW) is not used except to push the parameters out:

```
NO
FI
3
4
5
```

The program can recover the ASCII command string (up to 80 characters typed after the prompt) by using the String Passage EXEC call (see Section IV). The RUIH command inhibits the passage of the command string. If there are no characters past name, the command string is not transmitted.

String Passage Example:

```
RU,name,STRINGWHATEVER,12345,ANOTHERONE,6789
```

Note that if there is no partition large enough to run the program, or if the program is assigned to a partition that is too small or does not exist, a 'SIZE ERROR' message will be reported. Conditions causing this error message could be as follows:

```
:SP,xxx
reboot and reconfigure memory to remove partitions large
enough for this program.
:RP,xxx
:RU,xxx
```



## SS (operator suspend)

Operator suspends a non-dormant program. The format is

SS,name

where name is the name of the program to be suspended.

The SS command places the program in the operator suspended list immediately if the program is executing or scheduled. The request is illegal if the program is dormant. If the program is suspended for I/O memory, disc or is in the time list, RTE waits until the current state is ended and then operator-suspends the program.

The SS command is similar to the Program Suspend EXEC call (see Section IV).

## ST (status)

Requests the status (priority, current list, time values) of a named program, or to determine the name and partition number of the program currently occupying memory, or print the name of the program occupying a specified partition. The formats are as follows:

## Program Status:

ST,name

## Name and Partition Number of Current Program:

ST,0

## Name of Program in Specified Partition:

ST,part numb

## where:

name is the name of the program whose status is to be printed.

0 causes the system to print the name and partition number of the program currently executing. If none, 0 is printed.

part numb is a partition number that causes the system to print the name of the program in part numb. If the partition is empty, 0 is printed. If part numb is wrong, NO SUCH PROGRAM is printed.

The status of a program is printed on one line in a fixed format:

pr s res mpt hre min sec ms T

## where:

pr is the priority (a decimal value from 1 to 32767).

s is the current state of the program:

- 0 = Dormant
- 1 = Scheduled
- 2 = I/O suspend
- 3 = General wait
- 4 = Unavailable memory suspend
- 5 = Disc allocation suspend
- 6 = Operator suspend or programmed suspend (EXEC 7 call)
- 9 = Background segment

## ST (status) ...(continued)

res, mpt, hr, min, sec and ms are all zero (0) unless the program is scheduled by the clock (see the IT command in this section for the meaning of these terms).

The letter "T" appears when the program is currently in the time list as a result of an ON command.

A program is placed in the general wait list (status = 3) whenever:

- a. It is waiting for a Resource Number (RN) to clear or become available. This includes Logical Unit (LU) locks and attempts to use a locked LU.
- b. A schedule request is made with ICODE = 23 or 24 (queue schedule), and the program being called is busy.
- c. A request is made to an I/O device that is down. This differs from a request to an I/O device that is busy.
- d. A Class I/O GET call is made and the Class Queue is empty.
- e. A program is waiting for another program to complete as a result of an EXEC 9 or 23 call.
- f. A program is waiting on a Buffer Limit (see the BL command in this section).

Programs will be removed from the general wait list when the action waited for takes place or when the program is aborted.

When the format ST,0 is used, the status is printed as:

name part numb

where:

name is the name of the program currently residing in partition number part numb.

part is the partition number.  
numb

When the format ST,part numb is used, the status is printed as:

name

## OPERATOR COMMANDS

SZ (assignment)

Causes program size information to be printed. The format is

SZ,xxxxx

where xxxxx is the program name. The output will be formatted as:

AAAAA BB CCCC DD

where:

AAAAA = the last word plus 1 of the user's program. If the program is segmented, AAAAA is the last word, plus 1 of the largest segment.

BB = minimum required size of the program. For non-EMA programs this includes the program code size plus any optional dynamic buffer space. For EMA programs, this also includes the EMA size.

NOTE: If the program is an EMA program whose EMA size was defaulted and that program had not been previously dispatched, the EMA size "CCCC" will be reported as "1".

CCCC = the program's EMA size. Printed for EMA programs only.

DD = the program's MSEG size. This will only be printed if the program is of EMA type.

## SZ (reassignment)

Allows the user to increase the page requirements of a program. Certain programs such as compilers, assemblers, loaders and generator use memory after the end of the program for symbol table or data space. The SZ command modifies the size of the additional memory used by the program. An alternate form of the command increases both program page requirements and EMA size requirements. The format is

```
SZ,name,P1          for non-EMA programs
    OR
SZ,name,P1,P2      for EMA programs
```

where:

name is the program name

P1 is the new required program size in pages for non-EMA programs; that is, the program code size plus any dynamic buffer space.

For EMA programs, P1 is the new EMA size.

P2 is the new MSEG size for the EMA program referenced.

The program must be dormant and not currently resident in a partition (i.e., it must not have terminated with save-resources or serially reusable condition), and there must be at least one partition large enough to run the program at its new size.

The following conditions will be flagged as errors and the error 'SIZE ERROR' reported:

## FOR NON-EMA PROGRAMS:

1. An attempt to make P1 larger than 32K word program address space.
2. An attempt to make P1 larger than any currently existing partitions.
3. If the program is assigned, and attempt to make P1 larger than the partition size.
4. An attempt to make P1 smaller than the actual code of the program.

SZ (reassignment) ... cont'd

FOR EMA PROGRAMS:

1. An attempt to set P1 such that the program size plus the EMA size is larger than the largest partition in the system.
2. If the program is assigned to a partition, an attempt to set P1 such that the program size plus the EMA size is larger than that partition.
3. An attempt to set P1 less than 1.
4. An attempt to set P2 such that the program size plus P2 exceeds maximum program address space.
5. An attempt to set P2 less than 1.

EMA size changes are only allowed for those programs where no EMA size was specified within the program itself; that is, the default was taken. An attempt to increase or decrease the EMA size in a program where the EMA size was specified within the program causes a 'SIZE ERROR' message to be issued. MSEG changes may be made for any EMA type program. All FTN4 programs have specified EMA sizes. Note that it is not possible to increase or decrease the dynamic buffer space for EMA programs via the SZ command. This can be done only by reloading the program.

## TI (time)

Prints the current year, day and time, as recorded in the real-time clock. The format is

TI

The computer prints out the year, day and time in the format

yyyy ddd hh mm ss

where:

yyyy is the four-digit year.

ddd is the three-digit day of the year (see Table 2-3 at the end of this section for day-of-year conversion).

hh,mm,ss is the time on a 24-hour clock in hour, minutes and seconds.

The TI command is similar to the Time Request EXEC Call (see Section IV).

## OPERATOR COMMANDS

### TM (set clock)

Sets the real-time clock. The format is

TM,yyyy,ddd[,hh,mm,ss]

where:

yyyy is a four-digit year.

ddd is a three-digit day of the year (see Table 2-3 at the end of this section).

hh,mm,ss is the current time of a 24-hour clock in hours, minutes and seconds.

The TM command is entered in response to the message

SET TIME

which is displayed when the RTE system is booted from disc.

Enter a time value ahead of real-time. When real-time equals the entered value, press RETURN key. The system is now synchronized with the time of day.

#### NOTE

The real-time clock is automatically started from 8:00 on the system release date each time the system is loaded into memory.



## TO (time out)

Prints or changes the time-out value of an I/O controller. The format is

TO,eqt[,numb]

where:

eqt is the EQT entry number of the I/O controller.

numb is the number of 10 ms intervals to be used as the time-out value (numb cannot be less than 500 (5 sec) for the system input device driven by DVR00/05).

The time-out value is calculated using numb time-base generator interrupts (the time-base generator interrupts once every 10 ms). For example, numb = 100 sets a time-out value of one second;  $100 * 10 \text{ ms} = 1 \text{ second}$ . When the system is rebooted from the disc, time-out values set by TO are reset to the values originally set during generation.

If numb is absent, the time-out value of eqt is printed in the format

TO #10 = 100

and means EQT entry number 10 has a time-out value of 100 ten-millisecond intervals or one second.

If a device has been initiated and it does not interrupt within the interval set by the time-out parameter, the following events place:

- a. The calling program is rescheduled and a zero transmission log is returned to it.
- b. The device is set to the down status and bit 11 in the fourth word of the device's EQT entry is set to 1. An error message is printed; e.g.,

I/O TO L #x E #y S #z

- c. The system issues a CLC to the device's I/O select code(s) through the EQT number located in the Interrupt Table.

See also the discussion of I/O controller time-out in the Input/Out section of this manual and "Driver Time-Out Processing" in the RTE Operating System Driver Writing Manual.

## UP (make available)

Declares an I/O controller and all associated devices as up (i.e., available for use by the RTE system). The format is

UP,eqt

where eqt is the EQT entry number of the I/O controller to be re-enabled.

When the operator has previously set an I/O controller or device down for some reason, the condition should be corrected before using the UP command to declare the item available again. If the problem is irrecoverable, the LU command can be used to switch the Logical Unit number assignment to another device for further requests (see the LU command in this section). Previous requests made to this device are switched to the new device. To prevent indefinite I/O suspension on a downed device, time-out is used. Refer to the TO command in this section and "I/O Device Time-Out" in Section V.

The UP command places all downed devices (LU's) and the I/O controller (EQT entry) in the available state. Any I/O operations associated with downed devices are queued on the EQT entry for processing. If a device's problem has not been corrected, it will be reset down and an error message will be printed:

I/O NR L #lu E #eqt S #sub

## UR (release reserved partition)

Releases a partition previously reserved during generation or slow boot.

The format is

UR,xx

where xx is the number of the partition to be released.

Once the command is entered, any program that fits into the partition may run in it. Note that although partitions may be released on-line, they may not be reserved on-line, since such action could prevent a currently swapped-out program from regaining use of its system-assigned partition when it was again scheduled.

## OPERATOR COMMANDS

### 3-5. OPERATOR COMMAND ERROR MESSAGES

When an operator command is entered incorrectly or current system conditions prevent honoring the command, RTE may reject the command and issue one of the messages listed in Table 3-3. The operator should either enter the command correctly or take appropriate action and enter the command again.

Table 3-3. Operator Command Error Messages.

Message	Meaning	Action
OP CODE ERROR	Illegal operator request word.	Enter correct opcode
NO SUCH PROG	The name entered is not a main program in the system.	Enter correct program name or load program
INPUT ERROR	A parameter is illegal.	Enter command with correct parameter
ILLEGAL STATUS	Program is already scheduled.	Check status with ST cmd. Either wait until program terminates itself or off it with OF command and reenter RU command
CMD IGNORED- NO MEM	Not enough System Available Memory exists for storing the program's command string.	Reenter the command (RU,ON,GO) or enter the inhibit form (IH) of the cmd.
ILLEGAL PART'N	Partition does not match command request.	Reenter command with correct parameter number
SIZE ERROR	Illegal program size specified or size of program specified larger than its assigned partition or any partition.	Reenter command with correct size or adjust program size with SZ cmd.

Other errors may occur when an I/O device times out because of an inoperable state. For example, assume the line printer is in the OFF-LINE condition (or the operator has failed to engage the paper tape reader clutch). In this case, the system will print one of the following error messages and suspend the program:

```
I/O NR L #lu E #eqt S #sub
```

```
I/O TO L #lu E #eqt S #sub
```

After the device problem has been corrected, simply enter the command

```
UP,eqt
```

where eqt is the downed device's Equipment Table entry number (same number given in the I/O error message). The program is automatically rescheduled and the desired I/O operation takes place.

An alternate method of handling the same problem would be to use the LU command to change the referenced device to another device that is operational.

Another example of time-out is running out of paper when a program is printing a long listing on the line printer. In this case, it is possible to switch LU's and continue the listing without interruption, as shown below:

```
I/O TO L #lu E #eqt S #sub
LU,lu,eqt
```

The error message says that the device at LU number lu, EQT number eqt, subchannel number sub has timed out and has been set down by the system. The operator switches logical units (with the LU command). The listing will continue on the new device.

#### 4-1. INTRODUCTION

An executing program may request various system services through EXEC calls coded into the program. An EXEC call is a block of words consisting of a subroutine call to EXEC with a list of parameters that define the request. Execution of the subroutine call causes a memory protect violation interrupt and transfers control into the EXEC module. EXEC then determines the type of request (from the parameter list) and initiates processing if the request was legally specified.

In RTE FORTRAN IV, EXEC calls are coded as standard CALL statements. In Assembly Language, EXEC calls are coded as JSB EXEC, followed by a series of parameter definitions. For any particular call, the object code generated for the FORTRAN CALL statement is equivalent to the corresponding Assembly Language object code.

#### 4-2. ASSEMBLY LANGUAGE FORMAT

The general format for an EXEC call in Assembly Language is as follows:

EXT	EXEC	Used to link program to RTE.
.		
.		
JSB	EXEC	Transfer control to RTE.
DEF	*+n+1	Defines a point of return from RTE (must be immediately after the last parameter), where n is the number of parameters and may not be an indirect address.
DEF	p1	Define addresses of parameters that may occur anywhere in program; may be multi-level indirect.

## EXEC CALLS

```
DEF   pn
return point  Continue execution of program.
.
.
.
.
.
pn - - -      pl = ICODE = Request Code 1<pl< 26.
              - -
              Actual parameter values
.
pn - - -
```

The example below illustrates a Read request (ICODE=1), with the read being performed on LU5:

```
JSB EXEC
DEF NEXT      Address of return point and call delimiter.
DEF D1        Address of EXEC code.
DEF LU        Address of LU number.
DEF IBUFR     Buffer address.
DEF IBUFL     Address of number of words to read.
NEXT - .
.
.
.
.
D1  DEC  1      This is ICODE; 1=read.
LU  DEC  5      LU number is 5.
IBUFL DEC 10    Buffer length to read is 10 words.
IBUFR BSS 100   This is the buffer where the data is placed.
```

The above sample request reads 10 words from LU5 and places the words into the first 10 words of the 100-word buffer called IBUFR.

### 4-3. FORTRAN IV FORMAT

In FORTRAN IV, the Executive can be called through a CALL statement or as a function. The function is used when the user wishes the A and B registers to be returned in a variable.

CALL Statement Example:

```
CALL EXEC (ICODE, p2, ...,pn)
```

where ICODE and p2 through pn are either integer values or integer variables defined elsewhere in the program.

Function Example:

```
DIMENSION IREG(2)
EQUIVALENCE (REG(1),IA,IREG),(IREG(2),IB)
      .
      .
      .
REG=EXEC (ICODE,p2...,pn)
```

The A-register is returned in IA and the B-register in IB.

As a further example of using calls in FORTRAN, the Assembly Language example given previously in paragraph 4-2 could be performed in two different ways in FORTRAN-IV:

1. As a call:

```
DIMENSION IBUFR(100)
      .
      .
      .
LU=5
IBUFL=10
CALL EXEC(1,LU,IBUFR,IBUFL)
```

2. As a function:

```
DIMENSION IBUFR(100)
      .
      .
      .
LU=5
IBUFL=10
REG=EXEC(1,LU,IBUFR,IBUFL)
```

These two FORTRAN examples and the Assembly Language call all perform the same function.



## EXEC CALLS

### 4-4. EXEC CALL ERROR RETURNS

EXEC calls that are in error will cause the offending program to be aborted if the error is severe enough. The following errors are considered to be sufficiently catastrophic to cause a program abort:

Error Code: -----	Error Type: -----
MP	Memory Protect
DM	Dynamic Mapping
RQ	Request Code
DP	Dispatching
RE	Reentrancy
PE	Parity

If an error is not severe, it will either abort the program or, at the user's option, report the error to the program itself and allow the program to continue execution. Non-severe error codes include the following:

Error Code: -----	Error Type: -----
SC	Scheduling
LU	LU Lock
IO	Input/Output Error
DR	Disc Allocation
RN	Resource Number

A detailed explanation of EXEC call error messages is given at the end of this section.

The "no-abort" option is set up by altering the return point of the EXEC call. This error return is established by setting bit 15 to "1" on the request code word (ICODE). This causes the system to execute the first line of code (it must be a one-word instruction) following the CALL EXEC if there is an error. If there is no error, the second line of code following the CALL EXEC is extended.

The special error return will also return control to the calling program on a disc parity error on the system disc or auxiliary disc. In this case, the B-register will be set to -1 instead of the transmission log, and the return will be to the normal return point. If there is an error, the A-register will be set to the ASCII error type (LU,SC,IO,DR,RN) and the B-register set to the ASCII error numbers normally displayed on the system console.

The following excerpts from a sample FORTRAN program demonstrates use of the special error return:

```

                CALL EXEC(ICODE+100000B,LU,IBUFR,IBUFL)
Error Return->  GO TO 100
No Error Return-> .
                .
                .

```

Only the GO TO statement should be entered after a no-abort EXEC call; any other FORTRAN command would cause error type information to be lost (see below). The GO TO statement also must not reference the very next statement; thus, the following sequence is illegal:

```

                CALL EXEC(ICODE+100000B,LU,IBUFR,IBUFL)
                GO TO 100
100
                .
                .
                .

```

This is illegal because FORTRAN produced code tries to optimize the two statements and will not produce a jump if the jump destination is the very next executable statement. Therefore, the GO TO would be ignored.

As mentioned previously, if an error return is made to a program, the A and B registers contain the ASCII error code. The A-register contains the error type (SC,LU,IO,DR,RN), and the B-register contains the error number (ASCII 01,02,03,etc.).

The A-register can be easily examined in Assembly Language calls. Examination is slightly more complex in FORTRAN-IV, but the A and B registers can be fetched in the following way:

```

                CALL EXEC(ICODE+100000B,...)
                GO TO 100
                .
                .
                .
100 CALL ABREG(IA,IB)

```

ABREG is an HP-supplied subroutine that returns the A-register in the first parameter (IA) and the B-register in the second parameter (IB). Since the contents of A and B are now available, the user may examine the the error and take appropriate action.

#### CAUTION

Note that the no-abort option should not be used when the EXEC call is made as a function; that is, the following should not be used:

```

                REG=EXEC(ICODE+100000B.....)
                GO TO 100

```

## EXEC CALLS

The reason is that REG forces the A and B register to be treated as a REAL subroutine instead of an integer subroutine.

### 4-5. EXEC CALL SUMMARY

Table 4-1 summarizes the available RTE EXEC calls, their function and order of appearance in this section. The error messages associated with the calls are listed at the end of this section.

Table 4-1. RTE EXEC Calls

Call	Request	Function	Page	
Read,Write	1,2	Transfers information to and from an external I/O device.	4-9	
I/O Control	3	Performs various I/O control operations.	4-12	
I/O Status	13	Requests information about a device.	4-16	
Disc Track Allocation Program	4	Assigns a specific number of disc tracks for data storage.	4-19	
Global	15			
Disc Track Release Program	5	Release assigned disc tracks.	4-21	
Global	16		4-22	
Program Completion	6	Logically terminates execution of a calling program.	4-24	
Program Suspend	7	Suspends calling program execution.	4-27	
Program Segment Load	8	Loads a program segment into background area.	4-28	
Program Schedule	9	Schedules a program for execution	4-29	
	10			Immediate with wait.
	23			Immediate without wait.
	24			Queue with wait.
	24	Queue without wait.		

## EXEC CALLS

Table 4-1. RTE EXEC Calls (cont'd)

Call	Request	Function	Page
Time Request	11	Requests current time.	4-33
String Passage	14	Retrieves program's command string or passes string to program's "Father."	4-34
Timed Execution		Schedules a program for execution	
Initial Offset	12	After an initial offset.	4-36
Absolute Start	12	At a specified time.	4-38
Program Swapping Control	22	Allows a program to lock itself into memory.	4-41
Partition Status	25	Provides information about a specified partition.	4-42
Memory Status	26	Allows a program to obtain information about its own address space.	4-44
Class I/O Read, Write	17,18,20	Starts a no-wait I/O request that results in an information transfer to and from an external I/O device or program.	4-50
Class I/O Control	19	Performs various no-wait control operations.	4-55
Class I/O Get	21	Completes the data transfer initiated by the Class I/O request (17,18,19,20).	4-52

## 4-6. STANDARD FUNCTION CALLS

## 4-7. READ/WRITE CALL

Transfers information to or from an I/O device. For a Read request or for writes to unbuffered devices, the program is placed in the I/O suspend list until the operation is complete. RTE then reschedules the program.

Assembly Language:

EXT	EXEC		
	.		
	.		
	JSB	EXEC	Transfer control to RTE
	DEF	RTN	Return address
	DEF	ICODE	Request code (1=read; 2=write)
	DEF	ICNWD	Control information
	DEF	IBUFR	Buffer location
	DEF	IBUFL	Buffer length
	DEF	IPRM1	Optional parameter (track number if disc transfer)
	DEF	IPRM2	Optional parameter (sector number if disc transfer)
RTN	return	point	Continue execution (A=status, B=transmission log. If buffered Write, A and B are meaningless.)
	.		
	.		
ICODE	DEC	1 (or 2)	1=Read;2=Write
ICNWD	OCT	conwd	conwd is described in Comments
IBUFR	BSS	n	Buffer of n words
IBUFL	DEC	n(or -2n)	Same n; words (+) or characters (-)
IPRM1	DEC	f	Optional parameter or decimal track number if disc transfer
IPRM2	DEC	q	Optional parameter or decimal sector number if disc transfer

FORTRAN

DIMENSION	IBUFR(n)	Set up buffer
IBUFL	= n	Buffer length
ICODE	= 2	Request code (1=Read; 2=Write)
ICNWD	= conwd	Set Control Word
REG=EXEC	(ICODE, ICNWD, IBUFR, IBUFL, IPRM1, IPRM2)	

EXEC CALLS

4-8. READ/WRITE COMMENTS

Parameters IPRM1 and IPRM2 are optional except in disc transfers. If the data transfer involves a disc, IPRM1 is the disc track number and IPRM2 is the disc sector number. These parameters may have further uses in calls to other I/O devices. In some cases, IPRM1 and IPRM2 may be used to pass an additional control buffer to the driver (see Z-bit below).

CONTROL WORD

Figure 4-1 shows the format of the control word (conwd) required in the Read/Write calling sequence. Function codes for DVR00/05 driven devices are given as an example. See the appropriate driver manual for other device function codes.

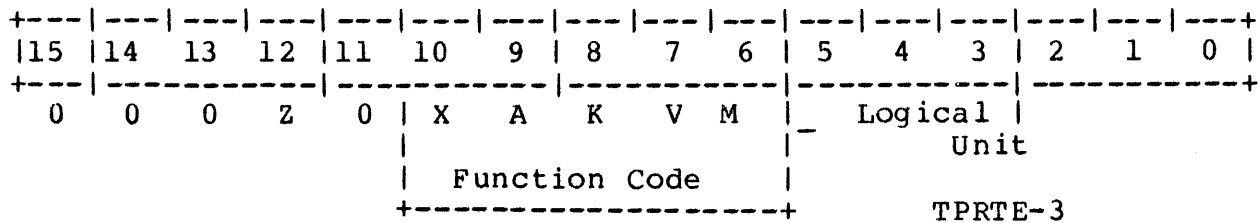


Figure 4-1. Read/Write (conwd) Format

where:

Logical Unit = the logical unit number of the devices to/from which the I/O transfer is to be sent.

Note that if the logical unit is specified as zero (the bit bucket), the call is executed but no data is transferred.

Z = When set, designates that IPRM1 is the address of a control buffer and IPRM2 is the length of that buffer in positive words or negative character (useable only when the call is to a non-disc device). The Z-bit is passed through to the driver.

Bits 11 and 13-15 are received for usage by the system and should be set to zero by the caller.

Function code bits for DVR00/05 devices are as follows:

M = 0 for ASCII.

M = 1 for binary.

V = 1, and M = 1, causes the length of punched tape input to be determined by the word count in the first non-zero character read from the tape.

V = 1 for the line printer will cause it to print column one.

V = 0, and M = 1, the length of the punched tape input is determined by the buffer length specified in the EXEC call.

K = 1 causes keyboard input to be printed as received. If K=0, input from the keyboard is not printed.

A = 1 designates punching (without printing) ASCII characters on the teleprinter (M = 0). (If A = 0, M determines mode of transfer.) This bit is effective on devices that recognize this control function.

X = When paper tape devices are used, "X" in combination with "M" and "V" will indicate an honesty mode that is defined as follows:

On input, if "X", "M", and "V" are set, absolute binary tape format is expected and handled. If "X" and "M" are set, and "V" is not, leader is not skipped and the specified number of words are read. On output the record terminator (usually four feed frames) is not punched.

On input, if "X" is set and "M" is not, ASCII tape format is expected. Leader is not skipped, bit 8 is stripped, but otherwise, all characters are passed to the user's buffer. The only exception is line-feed, which terminates the record. On output, carriage return and line-feed are suppressed; any trailing left arrow is not (i.e., left arrow is transmitted but carriage return/line feed is not).

#### A AND B REGISTER RETURNS

End-of-operation information for reads and unbuffered writes is transmitted to the program in the A- and B- registers. The A-register contains word 5 (status word) of the device EQT entry with bits 14 and 15 indicating the end-of-operation status as defined by the driver completion code. This will be either 00 (up) or 01 (down).

The B-register contains a positive number that is the number of words or characters (depending upon program specification) actually transmitted. Thus, the user can find the number of words entered on any input request by getting the contents of the B-register.



## EXEC CALLS

If the input buffer length was a negative number of characters, the contents of the B-register will be equal to the positive number of characters entered. If the requested buffer length was a positive number of words, the B-register contents will be equal to the positive number of words entered.

When a REAL array is transmitted, the buffer length must still be the total number of words required (i.e., two times REAL array length, or three times double-precision array length).

The registers are meaningless in output requests to a buffered device.

## I/O AND SWAPPING

Disc resident programs performing I/O are swappable under any one of the following conditions:

- a. The buffer is not in the partition (i.e., it is in system COMMON).
- b. The device is buffered, the request is for output, and enough SAM was allocated for buffering the record to be transferred.
- c. The input or output buffer is wholly contained in the Temporary Data Block (TDB) reentrant routine, and enough SAM was allocated to hold the TDB.

Only the first buffer of a two-buffer request (see Z-bit above) is checked to determine program swappability. It is the user's responsibility to put the second buffer in an area that implies swappability if conditions "a" or "c" are true. The system handles case "b".

## REENTRANT I/O

Use of reentrant I/O allows a program to be swapped if the read request is made via a call to the REIO subroutine. REIO is a utility library subroutine and is more fully described under Section X.

## 4-9. I/O CONTROL CALL

Carries out various I/O control operations, such as backspace, write end-of-file, rewind, etc. If the I/O device is not buffered, the program is placed in the I/O suspend list until the control operation is complete.

Assembly Language:

	EXT	EXEC	
	.		
	.		
	JSB	EXEC	Transfer control to RTE
	DEF	RTN	Return address
	DEF	ICODE	Request code (3=control)
	DEF	ICNWD	Control information
	DEF	IPRAM	Optional parameter
RTN	return	point	Continue execution (A = status, B meaningless. A and B are both meaningless if the call is buffered).
	.		
	.		
	.		
ICODE	DEC	3	Request code = 3
ICNWD	OCT	conwd	See Control Word
IPRAM	DEC	n	Required for some control functions; see Control Word

FORTRAN:

Use the FORTRAN statements or an EXEC call sequence.

```

    ICODE = 3      Request code
    ICNWD = conwd
    IPRAM = x      Optional; see Control Word
    REG = EXEC (ICODE,ICNWD,IPRAM)
  
```

CONTROL WORD

Figure 4-2 shows the format of the control word (conwd) required in the I/O control calling sequence.

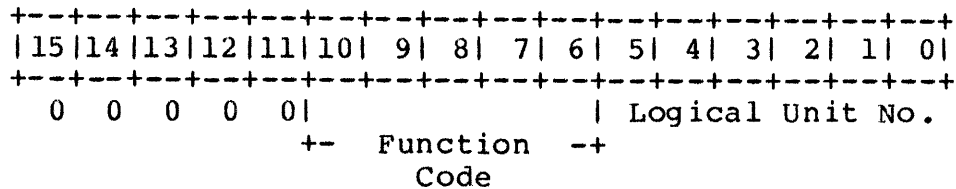


Figure 4-2. I/O Control (conwd) Format

## EXEC CALLS

The following are general function codes:

Function Code (Octal)	Action
-----	-----
00	Clear device
01	Write end-of-file (magnetic tape)
02	Backspace one record (magnetic tape)
03	Forword space one record (magnetic tape)
04	Rewind (magnetic tape)
05	Rewind standby (magnetic tape)
06*	Dynamic status (magnetic tape)
07	Set end-of-paper tape--leader skipped on next input request
10	Generate paper tape leader
11	List output line spacing
12	Write inter-record gap (magnetic tape)
13	Forward space file (magnetic tape)
14	Backward space file (magnetic tape)
15	Conditional form feed (see Line Printer Driver manual).

```
+-----+
|*FOOTNOTE: The dynamic status request (06) is unbuffered by RTIOC so|
| that the caller receives the true status of any device. This causes|
| the caller to wait for previous requests it (and lower priority |
| programs) has made to be processed. |
+-----+
```

The following functions are defined for DVR00/DVR05 (see the driver manual):

Function Code: -----	Action: -----
20	Enable terminal - allows terminal to schedule its program when any key is struck.
21	Disable terminal - inhibits scheduling of terminal's program.
22	Set timeout - the optional parameter is set as the new timeout interval.
23	Ignore all further action requests until: <ul style="list-style-type: none"> <li>a. the device queue is empty</li> <li>b. an input request is encountered in the queue</li> <li>c. a restore control request is received.</li> </ul>
24	Restore output processing (this request is usually not needed).

The following functions are defined for the 264x cartridge tape units (CTU). (Function codes 01, 02, 03, 04, 06, 13, and 14 have the same meaning for CTU as for magnetic tape.)

Function Code: -----	Action: -----
05	Rewind.
10	Write end-of-file if not just previously written or not at load point.
26	Write end-of-data.
27	Locate file number IPRAM (less than 256).

## EXEC CALLS

Function code octal 11 (list output line spacing), requires the optional parameter IPRAM which designates the number of lines to be spaced on the specified logical unit as shown below:

IPRAM -----	Teleprinter -----	Line Printer -----
+n	space n lines	space n lines
-n	space n lines	top of form
0	no line feed	no line feed

### 4-10. I/O STATUS CALL

Requests information (status condition and device type) about the device assigned to a Logical Unit number.

#### Assembly Language:

	EXT	EXEC	
	.		
	.		
	.		
	JSB	EXEC	Transfer control to RTE
	DEF	RTN	Return address
	DEF	ICODE	Request code (13=status)
	DEF	ICNWD	Control information
	DEF	ISTAL	Status word 1
	DEF	ISTA2	Status word 2 -- optional
	DEF	ISTA3	Status word 3 -- optional
RTN	return	point	Continue execution (A and B are meaningless)
	.		
	.		
	.		
ICODE	DEC	13	Request code = 13
ICNWD	DEC	n	Logical Unit number
ISTAL	NOP		Word 5 of EQT entry returned here
ISTA2	NOP		Word 4 of EQT entry returned here, optional
ISTA3	NOP		LU status returned here, optional

#### FORTTRAN:

```

ICODE = 13          Request code
ICNWD = nn         nn is the logical unit number
CALL EXEC (ICODE,ICNWD,ISTAL,ISTA2,ISTA3)

```

## 4-11. I/O STATUS COMMENTS

The calling program is not suspended when the call is made. Equipment Table entry (EQT entry) words 5 and 4 (optional) are returned in ISTA1 and ISTA2 and are defined as shown in Table 4-2. The STATUS portion of EQT entry word 5 for moving head discs is further broken down and is shown in Table 4-3. Refer to the appropriate driver manual for the format for other drivers.

The status of the specified LU is returned in ISTA3. Bit 15 indicates whether the device (LU) is up (0) or down (1). Bits 4-0 give the subchannel associated with the device.

Table 4-2 I/O Status Word (ISTA1/ISTA2) Format

WORD	CONTENTS															
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
4	D	B	P	S	T	Unit #					Channel #					
5	AV		EQUIP. TYPE CODE						STATUS (see Table 3-4)							
ISTA2	<p>D = 1 if DMA required.</p> <p>B = 1 if automatic output buffering used.</p> <p>P = 1 if driver is to process power fail.</p> <p>S = 1 if driver is to process time-out.</p> <p>T = 1 if device timed out (system sets to zero before each I/O request).</p> <p>Unit = Last sub-channel addressed.</p> <p>Channel = I/O select code for device (lower number if a multi-board interface).</p>															
ISTA1	<p>AV = I/O controller availability indicator:            0 = available for use.            1 = disabled (down); for UP/DOWN status of LU see ISTA3.            2 = busy (currently in operation).            3 = waiting for an available DMA channel.</p> <p>EQUIP. TYPE CODE = type of device. When this number is linked with "DVR." it identifies the device's software driver routine:            00 to 07<sub>8</sub> = paper tape devices (or system control devices)            00 = teleprinter (or system keyboard control device)            01 = photo-reader            02 = paper tape punch            05 subchannel 0 = interactive keyboard device (or system keyboard control devices)                  subchannel 1,2 = HP mini-cartridge device            07 subchannel 4 multi-drop driver            10 to 17 = unit record devices            10 = plotter            11 = card reader            12 = line printer            15 = mark sense card reader            20 to 37 = magnetic tape/mass storage devices            31 = 7900 moving head disc            32 = 7905 moving head disc            33 = flexible disc            40 to 77 = instruments</p> <p>STATUS = the actual physical status or simulated status at the end of each operation.            For paper tape devices, two status conditions are simulated: Bit 5 = 1 means end-of-tape on input, or tape supply low on output.</p>															

EXEC CALLS

TABLE 4-3. EQT WORD 5 STATUS TABLE

Device \ Status	7	6	5	4	3	2	1	0
7900 Moving Head Disc DVR31		NR	EOT	AE	FC	SC	DE	EE
7905/7906/7920 Moving Head Disc DVR32	PS	FS	HF	FC	SC	NR	DB	EE
(See appropriate driver manual for status bits of other devices)								
Where:								
DE = Data Error			NR = Not Ready					
DB = Device Busy			HF = Hardware Fault					
SC = Seek Check			PS = Protected Switch Set					
FC = Flagged Track (protected)			FS = Drive Format Switch is set					
AE = Address Error			EE = Error exists					
EOT = End of Tape/Track								

4-12. DISC TRACK ALLOCATION CALL

Requests that the system assign a specific number of contiguous disc tracks for data storage. The tracks are either assigned to the calling program or assigned globally.



## EXEC CALLS

### Assembly Language:

	EXT	EXEC	
	.		
	.		
	.		
	JSB	EXEC	Transfer control to RTE
	DEF	RTN	Return address
	DEF	ICODE	Request code (4=local track;15=global track)
	DEF	ITRAK	Number of contiguous tracks required
	DEF	ISTRK	Start track number
	DEF	IDISC	Disc logical unit number
	DEF	ISECT	Number of 64 word sectors/track
RTN	return	point	Continue execution (A and B are meaningless)
	.		
	.		
ICODE	DEC	4 or 15	4 = allocate track to program 15 = allocate track globally
ITRAK	DEC	n	n = number of contiguous tracks within the same disc unit requested. If bit 15 of ITRAK = 1 the program is not suspended if tracks are not available; if bit 15 = 0, the program is suspended until the tracks are available.
ISTRK	NOF		System stores starting track number here, or -1 if the tracks are not available.
IDISC	NOF		System stores Logical Unit number (2 or 3) here.
ISECT	NOF		System stores number of 64 word sectors/track here.

### FORTTRAN:

Example (with no suspension):

```
ICODE = 4
ITRAK = 100000B + n
CALL EXEC (ICODE,ITRAK,ISTRK,IDISC,ISECT)
```

Example (with suspension until tracks available):

```
ICODE = 4
ITRAK = n
CALL EXEC (ICODE,ITRAK,ISTRK,IDISC,ISECT)
```

## 4-13. DISC TRACK ALLOCATION COMMENTS

RTE supplies only whole tracks within one disc. When writing or reading from the tracks (see Read/Write EXEC call), RTE does not provide automatic track switching; when using this call, the user program is completely responsible for track management. RTE will prevent other programs from writing on program-assigned tracks but not from reading them.

The program retains the tracks until released by itself, the operator, or if the program is aborted. Globally assigned tracks are available to any program for READ, WRITE, or release. The user is completely responsible for their management. RTE will not prevent other programs from writing on globally assigned tracks or from releasing them.

## 4-14. PROGRAM DISC TRACKS RELEASE CALL

Releases some contiguous disc tracks previously assigned to a program (see Disc Allocation EXEC call).

Assembly Language:

	EXT	EXEC	
	.		
	.		
	.		
	JSB	EXEC	Transfer control to RTE
	DEF	RTN	Return address
	DEF	ICODE	Request code (5=release local tracks)
	DEF	ITRAK	Number of contiguous tracks, or -1
	DEF	ISTRK	Starting track number
	DEF	IDISC	Disc logical unit = 2 or 3
RTN	return	point	Continue execution (A and B are meaningless)
	.		
	.		
	.		
ICODE	DEC	5	Release program's tracks
ITRAK	DEC	n	If n = -1, release all tracks assigned to program; ISTRK and IDISC are unnecessary. Otherwise, n is the number of contiguous tracks to be released starting at ISTRK.
ISTRK	DEC	m	Starting track number
IDISC	DEC	p	Disc logical unit

## EXEC CALLS

### FORTTRAN:

Release of n contiguous tracks starting at m on LU p:

```
        ICODE = 5
        ITRAK = n
        ISTRK = m
        IDISC = p
        CALL EXEC (ICODE, ITRAK, ISTRK, IDISC)
```

Release all tracks allocated to the program.

```
        ICODE = 5
        ITRAK = -1
        CALL EXEC (ICODE, ITRAK)
```

### 4-15. PROGRAM TRACKS RELEASE COMMENTS

Any suspended program waiting for tracks is rescheduled when enough tracks are released to honor the request.

### 4-16. GLOBAL DISC TRACKS RELEASE CALL

Releases a specified number of contiguous disc tracks that were previously assigned globally (see Disc Allocation EXEC call).

Assembly Language:

	EXT	EXEC	
	.		
	.		
	.		
	JSB	EXEC	Transfer control to RTE
	DEF	RTN	Return address
	DEF	ICODE	Request code (16=release global track)
	DEF	ITRAK	Number of contiguous tracks
	DEF	ISTRK	Starting track number
	DEF	IDISC	Disc logical unit
RTN	return	point	Continue execution (A = track release status, B meaningless)
	.		
	.		
	.		
ICODE	DEC	16	Release global tracks
ITRAK	DEC	n	The number of contiguous tracks to be released starting at ISTRK
ISTRK	DEC	m	Starting track number
IDISC	DEC	p	Disc logical unit

## FORTRAN:

Release of n contiguous global tracks starting at m on LU p:

```
        ICODE = 16  
        ITRAK = n  
        ISTRK = m  
        IDISC = p  
        REG = EXEC (ICODE,ITRAK,ISTRK,IDISC)
```

## 4-17. GLOBAL DISC TRACK RELEASE COMMENTS

If any one of the tracks to be released is either not assigned globally or is currently in use (i.e., some program is queued to read or write on the track at the time of the release request), none of the tracks are released.

The requesting program is rescheduled after the request with the A-Register set as follows:

A=0 The tracks have been released.

A=-1 No tracks have been released (at least one track was in use).

A=-2 No tracks have been released (one or more tracks was not assigned globally).

EXEC CALLS

4-18. PROGRAM COMPLETION CALL

Notifies RTE that the calling program wishes to terminate itself or another program.

Assembly Language:

	EXT	EXEC	
	.		
	.		
	JSB	EXEC	Transfer control to RTE
	DEF	RTN	Return address
	DEF	ICODE	Request code (6=terminate)
	DEF	INAME	Name of program to be terminated (optional)
	DEF	INUMB	Type of completion (optional)
	DEF	IPRML	
	.		
	.		Up to five optional parameters
	.		
RTN	DEF	IPRM5	
	return	point	Continue execution (A = as it was; B = as it was or parameter address)
	.		
	.		
ICODE	DEC	6	Request code = 6
	DEC	0	Terminate this program
INAME		or	
	ASC	3,name	name = Name of subordinate program to be terminated.
			name = 0 if terminating itself.
INUMB	DEC	n	n = 0, Normal completion
			n = -1, Serial Reusability Completion. When rescheduled, program is not reloaded into memory if it is still resident.
			n = 1, Save Resources Completion. Make program dormant but save current suspension point and save all resources the program has; that is, any system resource the program asked for but did not itself release is retained.

n = 2, Terminates and removes from the time list the named program. If the program is I/O suspended, the system waits until the I/O completes before setting the program dormant; however, this call does not wait. The program's disc tracks are not released. CALL EXEC (6, 0, 2 or 3) is equivalent to issuing an OF,name,0 or 1 command (respectively) and therefore is treated like an abnormal termination.

n = 3, Immediately terminates the named program, removes it from the time list, and releases all disc tracks. If suspended for I/O, a system generated clear request is issued to the driver. An abort message is printed on the system console. CALL EXEC (6, 0, 2 or 2) is equivalent to issuing an Of,name, 0 or 1 command (respectively) and therefore is treated as an abnormal condition.

IPRML

.  
.  
.  
.  
.

These parameters are saved in the terminating program's ID segment and thus may be picked up by a call to RMPAR when the program next executes. In this manner a terminating program may retain parameters for all future executions.

IPRM5

FORTTRAN: DIMENSION INAME(3) See INAME above

ICODE = 6

INUMB = 0

See INUMB above

INAME(1) = 2Hcc

First two characters

INAME(2) = 2Hcc

Second two

INAME(3) = 2Hc

Last character in upper eight bits

REG = EXEC (ICODE, INAME, INUMB)

## EXEC CALLS

### 4-19. PROGRAM COMPLETION COMMENTS

The optional parameters in this call makes it possible to selectively terminate programs that only the user has scheduled. That is, if PROG1 ("Father") schedules PROG2 ("Son") to run, and PROG2 later schedules PROG3, then PROG2 becomes the "Father" to PROG3 (a "son"). In this case, only the following calls for Program Completion are legal:

- \* PROG1 terminates itself or PROG2
- \* PROG2 terminates itself or PROG3
- \* PROG3 terminates itself only.

Option -1 (INUMB=-1) should be used only for programs that are serially reusable; that is, disc resident programs that can initialize their own buffers or storage locations. When INUMB=-1, the program is reloaded from disc only if it has been overlaid by another program. The program must be able to maintain the integrity of its data in memory.

Option 1 (INUMB=1) is nearly identical to the Program Suspend EXEC call (see below), and also functions similarly to the SS operator command. When INUMB=1, the program starts from its point of suspension with all resources saved. Unless the program terminated itself in this manner, it could only be restarted by the program that scheduled it ("Father") or through the ON or RUN operator commands. If the program terminated itself (INAME=0), it may be restarted by any normal run stimulus (i.e., schedule, ON, RUN, TIME and interrupt).

IPRM1 through IPRM5 are optional parameters that are passed back to the Program when it is next scheduled. They are passed only when INAME=0, and may be recovered by a call to RMPAR when the program next executes. This permits a program in the time list to run with the same parameters each time.

Note that the FORTRAN compiler automatically generates a Program Completion EXEC call when it compiles an END statement.

Note also that a father may either terminate a son normally or with the son saving resources.

## 4-20. PROGRAM SUSPEND CALL

Suspends execution of the calling program until it is restarted by a GO operator request.

Assembly Language:

	EXT	EXEC	
	.		
	.		
	JSB	EXEC	Transfer control to RTE
	DEF	RTN	Return address
	DEF	ICODE	Request code (7=suspend)
RTN	return	point	Continue execution (A=as it was; B=as it s or parameter address)
	.		
	.		
	.		
ICODE	DEC	7	Request code = 7

## 4-21. PROGRAM SUSPEND COMMENTS

The FORTRAN library subroutine PAUSE, which is automatically called by a PAUSE statement, generates the Program suspend EXEC call. In addition, it logs the pause and any supplied number on the system console.

It is illegal to suspend a program running under batch with the Program Suspend call. This results in a SC00 error return.

The Program Suspend call is similar in function to the SS operator command. When a program is suspended either by this call or by the SS operator command, both the A- and B-registers are saved and the program is placed in the operator suspension list. When the program is restarted via a GO command without parameters, all registers are restored to the same status they had at the point of suspension and the program resumes execution.



## EXEC CALLS

When the program is restarted via a GO command with parameters, the B-register contains the address of a five-word array set by the GO command. In a FORTRAN program, a call to the RMPAR library subroutine can load these parameters, providing the RMPAR call occurs immediately following the Program Suspend call. However, it should be noted that when RMPAR is used, parameters MUST accompany the GO command. Otherwise, RMPAR will use the restored B-register as an address to parameters that do not exist. When it is suspected that there might not be any parameters, the following example shows how to allow for it:

```

DIMENSION I(5),IREG(2)
EQUIVALENCE (IREG,REG),(IREG(2),IB)
REG=0.0
REG=EXEC (7)      Suspend
IF (IB) 20,20,10
10 CALL RMPAR (I)  Return point; get
                  parameters
20 CONTINUE      Return point; no
                  parameters

```

### 4-22. PROGRAM SEGMENT LOAD CALL

Loads a calling program's background segment from disc into the background segment area and transfers control to the segment's entry point. (See "Segmented Programs" in the Program Preparation section of this manual for information on segmented programs.)

#### Assembly Language:

	EXT	EXEC	
	.		
	.		
	JSB	EXEC	Transfer control to RTE
	DEF	RTN	Return address
	DEF	ICODE	Request code
	DEF	INAME	Segment name
	DEF	IPRML	} Up to five optional parameters
	.		
	.		
RTN	DEF	IPRMS	} Control is transferred to the segment. (A = segment ID seg. address; B = as it was or parameter address.)
	return	point	
	.		
	.		
ICODE	DEC	8	Request code = 8
INAME	ASC	3,name	name is the segment name

## FORTRAN:

```
DIMENSION NAME (3)
ICODE=8
INAME (1)=2Hcc      First two characters
INAME (2)=2Hcc      Second two characters
INAME (3)=2Hc       Last character in bits 8-15
CALL EXEC (ICODE,INAME,IPRM1...IPRM5)
```

## 4-23. PROGRAM SEGMENT LOAD COMMENTS

On segment entry the registers are set as follows:

A = Segment ID segment address.

B = As it is unless parameters are passed, in which case it is the address of parameter list address (see RMPAR).

If the segment loaded does not exist, an SC05 error results.

## 4-24. PROGRAM SCHEDULE CALL

Schedules a program for execution and passes up to five parameters and a buffer to the program.

# EXEC CALLS

## Assembly Language:

	EXT	EXEC	
	.		
	.		
	JSB	EXEC	Transfers control to RTE
	DEF	RTN	Return address
	DEF	ICODE	Request code
	DEF	INAME	Name of program to schedule
	DEF	IPRML	} Up to five optional parameters
	.		
	.		
	DEF	IPRM5	
	DEF	IBUFR	
	DEF	IBUFL	Optional buffer length
RTN	return	point	Continue execution (A=program status; B=as it was or parameter address)
	.		
	.		
ICODE	DEC	numb	9=immediate schedule with wait 10=immediate schedule with no wait 23=queue schedule with wait 24=queue schedule with no wait
INAME	ASC	3,name	name is the named program to schedule
IPRML			
	.		} Up to five optional parameters
	.		
	.		
IPRM5			
IBUFR	BSS n		Optional buffer of n words
n (or-2n)		Same n;	words (+) or characters (-)
			IBUFL DEC

## FORTTRAN:

```

DIMENSION      INAME(3),IBUFR(n)      IBUFL      =      n
Set buffer length ICODE = numb      See
ICODE above INAME(1) = 2Hcc      First two
characters INAME(2) = 2Hcc      Second two
characters INAME(3) = 2Hc      Last character
CALL = EXEC(ICODE,INAME,IPRML,...IPRM5,IBUFR,IBUFL)

```

## 4-25. PROGRAM SCHEDULE COMMENTS

The ICODE parameter determines whether or not the calling program will wait, and whether the calling program's schedule request will be queued until the currently scheduled program becomes dormant.

When a program is scheduled, a pointer is placed in its ID segment that will:

- a. Point back to the program that scheduled it.
- b. Be set to 0 if the program was scheduled by the operator, from an interrupt or from the time list.

The pointer is cleared when the program terminates or is aborted. Note that the pointer established the program performing the scheduling as the "Father" and the program being scheduled as the "Son".

When a program that had been scheduled with wait completes, the Father may recover the system's copy of optional parameter 1 to determine whether or not the Son terminated normally.

Abnormal termination of the Son is caused by any of the following conditions:

- a. System abort of program.
- b. An OF operator command.
- c. Self-termination via CALL EXEC (6,0,2) or CALL EXEC (6,0,3).

Abnormal termination causes the system's copy of optional parameter 1 to be set to 100000B. This occurs even if the Son attempted or planned to pass back parameters via PRTN. The Father can recover the system's copy of optional parameter 1 by calling RMPAR.

If the Son terminated normally and no parameters were passed back via PRTN, the value of optional parameter 1 returned by RMPAR will then be equal to its original value. Alternately, it will be the value set up in the Son's PRTN call. The PRTN subroutine allows Sons to pass parameters back to Fathers.

ICODE = 9 OR 10

If a program to be scheduled is dormant, it is scheduled and a zero is returned to the calling program in the A-register. If the program to be scheduled is not dormant, it is not scheduled by this call and its status (some non-zero value) is returned to the calling program in the A-register. If the program to be scheduled is a Son that was suspended with the EXEC 6 call, some high bits may be set in the A-register. Only the least four-bits should be checked for zero in this case.

## EXEC CALLS

A schedule with wait (ICODE=9) call causes RTE to put the "Father" in a wait status by setting the wait bit in the status word of the Father's ID segment. If required, the Father may be swapped by the system to make way for a program that needs to run. The "Son" runs at its own priority, which may be greater than, less than or equal to that of the calling program. Only when the Son terminates does RTE resume execution of the Father at the point immediately following the Program Schedule call.

A disc resident program may schedule another disc resident program with wait, since disc resident programs are swapped according to their priority when they are in conflict over use of their memory area.

A Program Schedule call without wait (ICODE=10) causes the specified program to be scheduled for execution according to its priority. The Father program continues at its own priority without wait. Again note that ICODES of 9 and 10 will not schedule the program if the program to be scheduled is busy (i.e., not dormant).

ICODE = 23 or 24

These requests are identical to 9 and 10 except that the system places the "Father" in a queue if the "Son" is not dormant. The Father's request will then be honored when the Son becomes available. Note that status will not be available in the A-register and the Father will be impeded until the request is honored. The queue means that if the Son is not dormant, the potential Father is suspended until the Son may be scheduled by this Father. When the potential Son can be scheduled, the request is reissued and execution proceeds as EXEC 9 and 10 described above.

### OPTIONAL PARAMETERS

When the Son begins executing, the B-register contains the address of a five-word parameter list from the Father (parameters = 0 as the default). A call to the RMPAR library subroutine, as the first executable statement of a called program, transfers these parameters to a specified five-word array within the called program. For example:

```
PROGRAM XQF
DIMENSION IPRAM (5)
CALL RMPAR (IPRAM)
```

Note that IPRAM must be a minimum dimension of five words.

If the optional buffer is included in the Father's scheduling call, the buffer is moved to System Available Memory and assigned to the Son. The Son can recover the string by using the GETST library routine or the String Passage call. The Father is memory suspended if there is not enough System Available Memory to currently hold the buffer but there will be in the future. The Father is aborted and an SC10 error is returned if there never be enough System Available Memory for the buffer. The Father will not abort if the no-abort bit (bit 15 in ICODE) is set. The length of the string is limited only by the amount of usable System Available Memory.

For schedule with wait requests (ICODE = 9 or 23), the Son may pass back five words to the Father by calling the PRTN library routine; for example:

```
PROGRAM SCHED
DIMENSION IBACK (5)
CALL PRTN (IBACK)
CALL EXEC (6)
```

The EXEC (6) call (termination call) must immediately follow the PRTN call. The Father may recover these parameters by calling RMPAR immediately after the Son call. The Son may pass back a buffer to the Father (see the String Passage call).

The Program Schedule call is similar in function to the RUN operator command.

#### 4-26. TIME REQUEST CALL

Requests the current time as recorded in the real-time clock.

Assembly Language:

	EXT	EXEC	
	.		
	.		
	JSB	EXEC	Transfer control to RTE
	DEF	RTN	Return address
	DEF	ICODE	Request code (11=time request)
	DEF	ITIME	Time value array
	DEF	IYEAR	Optional year parameter
RTN	return	point	Continue execution (A=meaningless; B=as it was)
	.		
	.		
	.		
ICODE	DEC	11	Request code = 11
ITIME	BSS	5	Time value array
IYEAR	BSS	1	Year (optional)

## EXEC CALLS

### FORTTRAN:

```
DIMENSION ITIME(5),IYEAR(1)
ICODE=11
CALL EXEC (ICODE,ITIME,IYEAR)
```

### 4-27. TIME REQUEST COMMENTS

The time value array contains the time on a 24-hour clock, with the year in an optional parameter, when RTE returns. The year is a full four digits (e.g., 1978).

Assembler		FORTTRAN
-----		-----
ITIME	or	ITIME(1) = Tens of milliseconds
ITIME+1	or	ITIME(2) = Seconds
ITIME+2	or	ITIME(3) = Minutes
ITIME+3	or	ITIME(4) = Hours
ITIME+4	or	ITIME(5) = Day of the year

Another method of obtaining the current time is through a double-word load from the \$TIME Table Area II entry point. \$TIME contains the double-word integer of the current time of day. If this double-word is passed to the TMVAL library subroutine, then TMVAL returns milliseconds, seconds, minutes and hours. Refer to the Library Subroutine section of this manual for more information.

The Time Request call is similar in function to the TI operator command.

### 4-28. STRING PASSAGE CALL

Retrieves the command string that scheduled the program or passes a buffer back to the "Father" program.

## Assembly Language:

	EXT	EXEC	
	.		
	.		
	JSB	EXEC	Transfer control to RTE
	DEF	RTN	Return address
	DEF	ICODE	Request code (14=string passage)
	DEF	IRCOD	Retrieve/write code
	DEF	IBUFR	Buffer location (string location)
	DEF	IBUFL	Buffer length (string length)
RTN	return	point	Continue execution (A = status; B = positive number of words/characters)
	.		
	.		
ICODE	DEC	14	Request code
IRCOD	DEC	1 or 2	1 = retrieve parameter string 2 = write buffer to "Father"
IBUFR	BSS	n	Buffer of n words
IBUFL	DEC	n(or -2n)	Same n; words (+) or characters (-)

## FORTRAN:

```

DIMENSION IBUFR(n)
IBUFL = n
ICODE = 14
IRCOD = 1
REG = EXEC(ICODE,IRCOD,IBUFR,IBUFL)

```

## 4-29. STRING PASSAGE COMMENTS

The command string retrieved is exactly like the string used in scheduling the program via RU, ON, GO commands, or EXEC 9, 10, 23, or 24. The block of System Available Memory used to store the command string (buffer) is released by this call or when the calling program goes dormant. Any parsing of the returned string is left to the calling program. The RTE system library routine GETST can be used to recover the parameter string portion of the command string.

Upon return from a retrieve operation, the A-Register contains status information: 0 if the operation was successful or 1 if no string was found. The B-Register is a positive number giving the number of words (or characters) transmitted. If the string is longer than IBUFL, only IBUFL words are transmitted. If an odd number of characters are requested in a retrieve operation, the right half of the last word is undefined.

If the write parameter string option is used, the call returns any block of system available memory associated with the "Father" and allocates a new block for the "Father" into which the string will be stored.



## EXEC CALLS

If no memory is currently available, the calling program is memory suspended.

If there will never be enough memory and bit 15 of ICODE is not set, the calling program is aborted with an SC10 error.

If there is no "Father," execution continues at the return point with the A-register equal to 1. If the write parameter operation was successful, the A-register is set to 0.

Example:

```
RU,PROGX,ABCDSTRING
```

Where RU,PROGX,ABCDSTRING is returned by EXEC (14,...) and ABCDSTRING is returned by GETST.

### NOTE

Be careful when writing a buffer to a "Father" when the Father scheduled the "Son" without wait (EXEC 10 or 24). It is the user's responsibility to ensure synchronization of the Son's write and the Father's read.

### 4-30. TIMED EXECUTION CALL (Initial Offset)

Schedules a program for execution at specified time intervals, starting after an initial offset time. RTE places the specified program in the time list and returns to the calling program.

## Assembly Language:

	EXT	EXEC	
	.		
	.		
	JSB	EXEC	Transfer control to RTE
	DEF	RTN	Return address
	DEF	ICODE	Request code (12=initial offset schedule
	DEF	I PROG	Program to put in time list
	DEF	IRESL	Resolution code
	DEF	MTPLE	Execution multiple (0-4095)
	DEF	IOFST	Initial time offset
RTN	return	point	Continue execution (A=meaningless; B as it was)
	.		
	.		
ICODE	DEC	12	Request code = 12
	DEC	0	Put calling program in time list
I PROG		or	
	ASC	3,name	name is the program to put in the time list
IRESL	DEC	x	Resolution code (1=10's/ms;2=ses;3=mins;4=hrs)
MTPLE	DEC	y	Execution multiple
IOFST	DEC	-z	z (units set by x) gives the initial offset

## FORTRAN:

DIMENSION I PROG(3)	See I PROG above
I PROG(1) = 2Hcc	First two characters
I PROG(2) = 2Hcc	Second two
I PROG(3) = 2Hc	Last character in upper 8 bits
ICODE = 12	
IRESL = x	(1=10's/ms;2=secs;3=mins;4=hrs)
MRPLE = y	(0-4095)
IOFST = -z	z (units set by x) gives the initial offset
CALL EXEC (ICODE,I PROG,IRESL,MTPLE,-IOFST)	

## EXEC CALLS

### 4-31. INITIAL OFFSET COMMENTS

The Execution Time EXEC call is similar to the IT Operator request (see Section II). However, the EXEC call places the program in the time list whereas IT does not. This call can schedule a program to execute in one of three ways as described in the following paragraphs:

#### 1. RUN ONCE

After a time offset and the program to be scheduled is dormant, the program will execute once and then be made dormant. This is accomplished as shown in the following example:

IRESL = 3 (specifies minutes)

MTPLE = 0 (specifies run once)

IOFST = -45 (specifies run after 45 minutes have elapsed from current time)

#### 2. RUN REPEATEDLY

After a time offset and the program to be scheduled is dormant, the program will execute, go dormant, and then re-execute at specified intervals. This is accomplished as shown in the following example.

IRESL = 3 (specifies minutes)

MTPLE = 60 (specifies run every 60 minutes)

IOFST = -30 (specifies run after 30 minutes have elapsed from current time)

#### 3. GO DORMANT; THEN RUN

If IPROG=0, the current/calling program is made dormant, but the point of suspension is retained. The program is then placed in the time list for rescheduling from the point of suspension after a delay. When the program is rescheduled, it can be either to run once or repeatedly.

### 4-32. TIMED EXECUTION CALL (Absolute Start Time)

Schedules a program for execution at specified time intervals, starting at a particular absolute time. RTE places the specified program in the time list and returns to the calling program.

## Assembly Language:

	EXT		EXEC	
	.		.	
	.		.	
	JSB		EXEC	Transfer control to RTE
	DEF		RTN	Return address
	DEF		ICODE	Request code (12=absolute start time sched.)
	DEF		IPROG	Program to put in time list
	DEF		IRESL	Resolution code
	DEF		MTPLE	Execution multiple
	DEF		IHRS	Hours
	DEF		MINS	Minutes
	DEF		ISECS	Seconds
	DEF		MSECS	Tens of milliseconds
RTN	return		point	Continue execution (A = meaningless, B as it was)
	.		.	
	.		.	
ICODE	DEC		12	Request code = 12
	DEC		0	Putting calling program in time list
IPROG		or		
	ASC		3,name	name is the program to put in the time list
IRESL	DEC		x	Resolution code (1=10's/ms;2=secs;3=mins;4=hrs)
MTPLE	DEC		y	Execution multiple
IHRS	DEC		a	Absolute starting time
MINS	DEC		b	In hours, minutes, seconds
ISECS	DEC		c	and tens of milliseconds
MSEC	DEC		d	on a 24-hour clock

## FORTRAN:

```

IPROG=0 or DIMENSION IPROG(3)
IPROG(1) = 2Hcc    First two characters
IPROG(2) = 2Hcc    Second two
IPROG(3) = 2Hc     Last character in upper 8 bits
ICODE = 12
IRESL = x          (1=10's/ms;2=secs;3=mins;4=hrs)
MTPLE = y          (0-4095)
IHRS = h
MINS = m
ISECS = s
MSECS = ms
CALL EXEC (ICODE,IPROG,IRESL,MTPLE,IHRS,MINS,ISECS,MSECS)

```

## EXEC CALLS

### 4-33. ABSOLUTE START TIME COMMENTS

The Execution Time EXEC call is similar to the IT operator request (see Section II). However, the EXEC call places the program in the time list whereas IT does not. This call differs from the Initial Offset version in that a future starting time is specified instead of an offset. For example, if the current time is 1400 hours and you wish the program to run at 1545 hours the parameters would be as follows:

```
IHRS = 15
MINS = 45
ISECS = 0
MSECS = 0
```

This call can schedule a program to execute in one of two ways as described in the following paragraphs:

#### 1. RUN ONCE

After a time offset and the program to be scheduled is dormant, the program will execute once and then be made dormant. This is accomplished as shown in the following example.

```
IRESL = 3      (specifies minutes)
MTPLE = 0      (specifies run once)
IHRS   = h
MINS   = m      (specifies absolute start-time)
ISECS  = s
MSECS  = ms
```

#### 2. RUN REPEATEDLY

After a time offset and the program to be scheduled is dormant, the program will execute, go dormant, and then re-execute at specified intervals. This is accomplished as shown in the following example:

```
IRESL = 3      (specifies minutes)
MTPLE = 60     (specifies run every 60 minutes)
IHRS   = h
MINS   = m      (specifies absolute start-time)
ISECS  = s
MSECS  = ms
```

## 4-34. PROGRAM SWAPPING CONTROL CALL

Allows a program to lock itself into memory (real-time or background) if the ability to perform a memory lock was specified during generation.

Assembly Language:

	EXT	EXEC	
	.		
	.		
	JSB	EXEC	Transfer control to RTE
	DEF	RTN	Return address
	DEF	ICODE	Request code
	DEF	IOPTN	Control information
RTN	return	point	Continue execution (A=meaningless; B=as it was)
	.		
	.		
	.		
ICODE	DEC	22	Request code = 22
IOPTN	DEC	numb	0 = program may be swapped 1 = program may not be swapped

FORTTRAN:

```

      ICODE = 22
      IOPTN = numb
      CALL EXEC (ICODE,IOPTN)

```

## 4-35. PROGRAM SWAPPING CONTROL COMMENTS

This call allows a programmer to lock a program into memory so it cannot be swapped out for a program of higher priority.

NOTE
The program cannot be locked into memory if the memory lock bits (base page word 1736B, bits 2 and 3) are not set (SC07 error results). The bits are set during generation.

## EXEC CALLS

The program's memory lock bit (IOPTN = 0 or 1) is set or cleared by the request (refer to ID segment word 15, bit 6 in Table A-1). This bit is also cleared (making the program swappable) if the program aborts or terminates except on the Save Resources Program Completion EXEC call.

### 4-36. PARTITION STATUS CALL

Returns status information about any specified partition.

Assembly Language:

```
        EXT  EXEC
        .
        .
        JSB  EXEC      Transfer control to RTE
        DEF  RTN      Return address
        DEF  ICODE     Request code (25=partition status)
        DEF  IPART     Partition no. that information is desired
                   about
        DEF  IPAGE     Returned no. of starting page for partition
        DEF  NPGS     Returned no. of pages in partition (includes
                   Base Page)
        DEF  IPST     Partition status word (defined below)
        .
        .
        ICODE DEC 25
        IPAGE NOP
        NPGS  NOP
        IPST  NOP
```

FORTTRAN:

```
CALL EXEC(25,IPART,IPAGE,INPGS,IPST)
```

## 4-37. PARTITION STATUS COMMENTS

The format of PSTAT is as follows:

15	14	13	12	11		8	7		0
---	---	---	---	---	-----	-----	-----	-----	-----
RS	RT	M	S	C	-----0-----				ID Seg. no.
---	---	---	---	---	-----				-----

where

- RS = 1 if partition reserved.
- RT = 1 if partition is real time.
- M = 1 if partition is a mother partition.
- S = 1 if partition is subpartition of a mother partition.
- C = 1 if chain is in effect; that is, if subpartition is locked because Mother partition is active.

ID Seg. no. is the ordinal number (i.e., counting from 1) of the ID segment for the program that occupies the partition. If ID Seg. no. = 0, the partition is unoccupied.

The values returned for number of pages and starting page number will be identical to those displayed by the WHZAT system program.

If the partition number is illegal (i.e., undefined or illegal), a -1 will be returned in the number of pages word and a 0 returned to the page number word.

The interaction between physical memory and logical memory for the partition status is illustrated in Figure 4-3. Note that the Table Area in the illustrated User Map is the system-supplied space that contains the necessary software to enable the user to communicate with the system.



EXEC CALLS

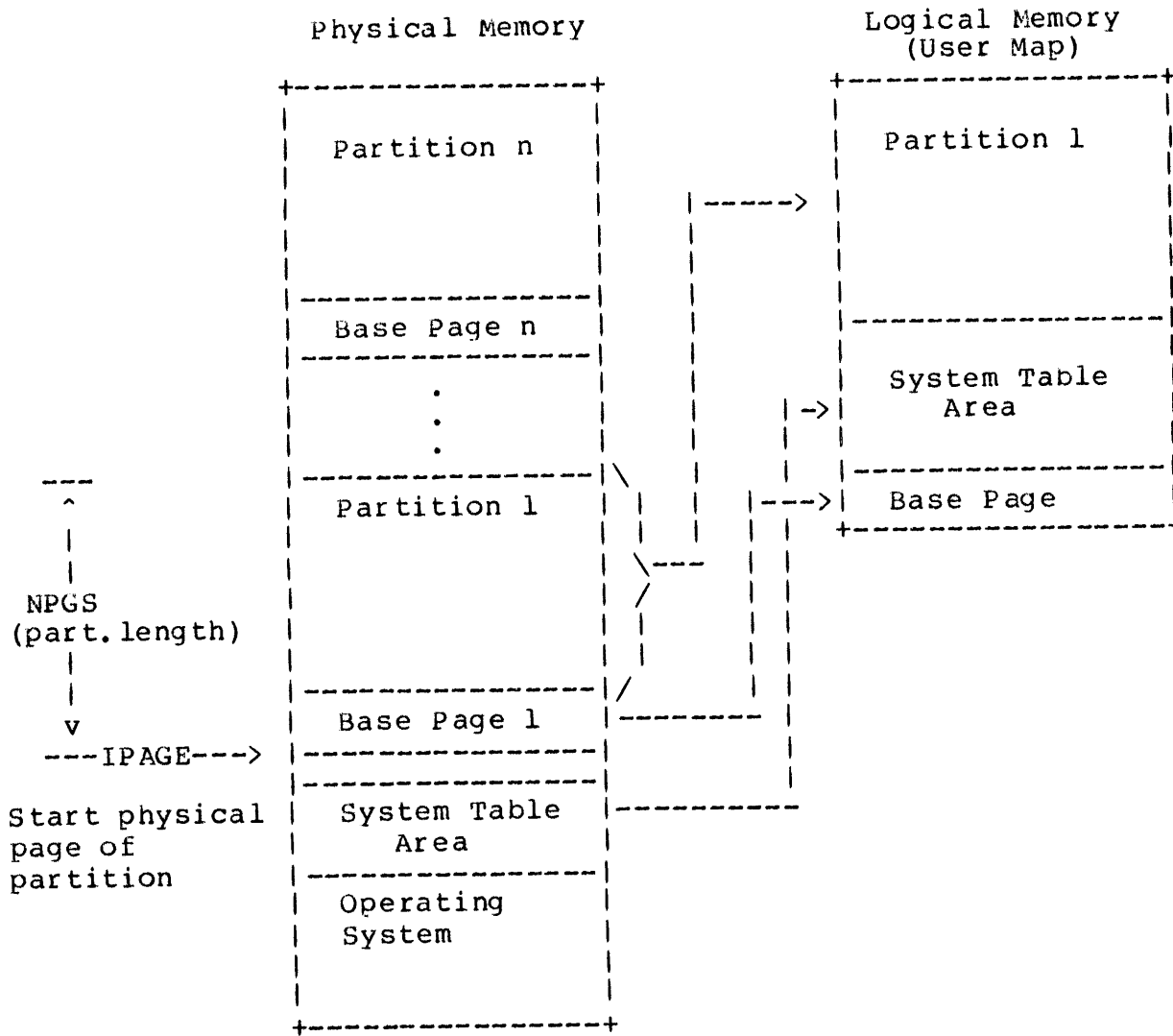


Figure 4-3. Partition Status Parameter Return

4-38. MEMORY SIZE CALL

Returns current memory limits of the partition in which the calling program is executing.

## Assembly Language:

```

      EXT EXEC
      .
      .
      JSB EXEC      Transfer control to RTE
      DEF RTN       Return Address
      DEF ICODE     Request Code (26=memory size)
      DEF IFPG      First available word address behind the
                   program (i.e., last word of program + largest
                   segment + 1)
      DEF ILMEM     Number of words available between end of
                   program and end of program's address space.
      DEF NPGS      Length of current partition in pages (includes
                   base page)
      DEF IMAP      Return copy of current user map (optional).
                   IMAP must be a 32-word buffer address.
      .
      .
      ICODE DEC 26
      IFPG  NOP
      ILMEM NOP
      NPGS  NOP
      IMAP  BSS 32

```

## FORTRAN:

```
CALL EXEC (26,IFPG,ILMEM,NPGS,IMAP)
```

## 4-39. MEMORY SIZE COMMENTS

The number of words of logical memory (ILMEM) is calculated by subtracting IFPG, the program's high main plus one (including its largest segment), from the last word of the program's logical address space. The logical address space, which may be smaller than the partition, is determined at load time and may be greater than (if size override option taken) or equal to the program size.

For EMA program, ILMEM is the number of words between the end of the program and the start of MSEG. This includes any dynamic buffer area requested at load time.

## EXEC CALLS

The manner in which the current status of the partition is calculated is illustrated in Figure 4-4. Sample data is provided.

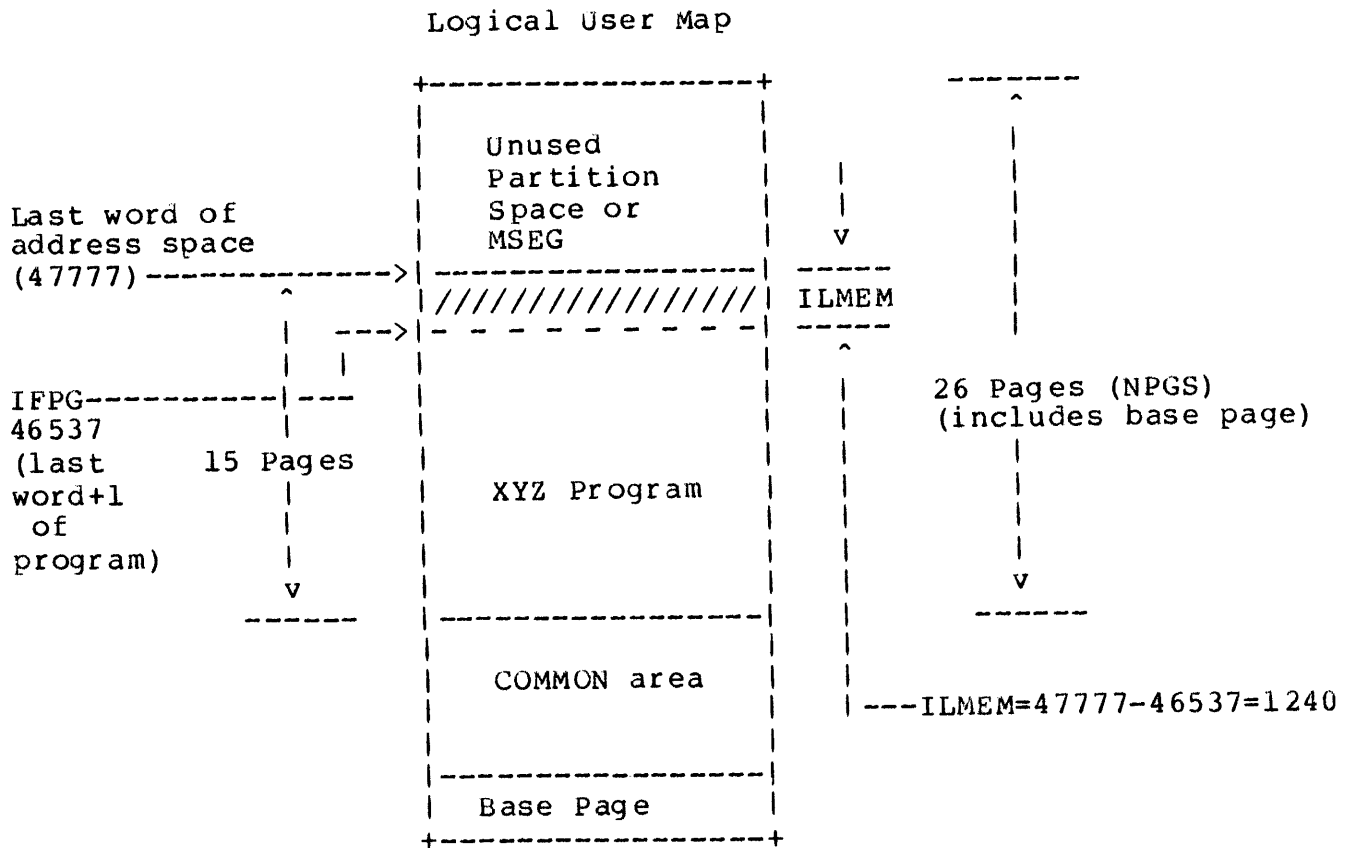


Figure 4-4. Partition Current Status Example

### 4-40. CLASS I/O EXEC CALLS

The Class I/O feature consists of a special set of I/O EXEC calls that give user programs a level of I/O independence beyond that provided by standard I/O. Use of the Class I/O scheme can provide the following benefits:

- a. A program doing an input operation can proceed with execution even though the data is not yet ready (I/O without wait).
- b. Program-to-program communication with controlled access via a mailbox scheme.
- c. Synchronized program-to-program data passing that avoids processing of incomplete or non-updated data. A calling program can put itself to sleep until it receives a signal that updated data processed by another program is available for further processing.

Implementation of Class I/O is based on use of a buffer with an exclusive access key, thus avoiding the possibility of unplanned alteration of existing data or access to incomplete data. Use of such keyed buffers or "classes" is exclusive of system or local COMMON resources utilized in standard program-to-program data passing.

A definition of the term "class" and other terms unique to Class I/O considerations is given in Table 4-4.

The maximum number of classes is established during system generation as a value between 1 and 255. Once the numbers are established, the system keeps track of them and assigns them (if available) to the calling program when a Class I/O call is made and the Class Number parameter is set to zero. Once the number has been allocated, the user can keep it as long as desired and use it to make multiple Class I/O Calls. When the user is finished with the number, it must be returned to the system for use by some other class user.

The system allocates a buffer from System Available Memory (SAM) when a user program issues a Class I/O call. The "key" is also issued to the calling program in the form of a Class Number, which is the only mechanism by which a calling program may thereafter access the buffer. Note that there may be more than one buffer associated with a single Class Number (key) and that a user program may have more than one Class Number allocated to itself.

For "I/O without wait" operations, data can be read from or written to an I/O device by first transferring the data to the buffer. The user program can thus either continue execution of other tasks without waiting for the I/O transfer to complete, or can suspend or terminate itself (releasing system services to other waiting programs) until the data transfer is completed.

The user program recovers the results of its Class I/O call by later issuing a Class I/O Get call. If the results are not present, the caller either can wait or return to execute more code before re-issuing the Class Get call.

A simple example of I/O without wait would be a program that issues a Class I/O READ call in its code, followed by a series of other coded operations. While these following operations were being executed, the system simultaneously would be reading the data into the allocated keyed buffer. The calling program would issue a Class I/O GET call to fetch the data from the buffer. A more detailed example of I/O without wait is given later in this section.

Table 4-4. Class Input/Output Terms

Term	Description
Class	An account owned by one program that may be used by a group of programs.
Class Number	The account number referred to above.
Class Users	Programs that use the Class Number.
Class Request	An access to a Logical Unit number with a class number.
Class Members	Logical Unit numbers that are currently being accessed in behalf of a class. Completion of access removes the association between class number and Logical Unit number (completion of access is defined as when the driver completes the request).
Class Queue (pending)	The set of uncompleted class requests.
Class Queue (completed)	The set of all completed class requests. The structure is first-in- first-out.

The system handles a Class I/O call in the following manner:

- a. When the class user issues a Class I/O call (and the call is received), the system allocates a buffer from System Available Memory and puts the call in the header (first eight words) of the buffer. The call is placed in the pending Class Queue and the system returns control to the class user.
- b. If this is the only call pending on the EQT, the driver is called immediately; otherwise, the system returns control to the class user and queues the request according to program priority.
- c. If buffer space is not available, the class user is memory suspended unless bit 15 ("no wait") is set. If the "no wait" bit is set, control is returned to the class user with the A-register containing a -2, indicating no memory available. If the program is suspended, no memory will be granted to lower priority programs until this program's Class I/O request is satisfied.
- d. If too much memory was asked for (more than all of System Available Memory) the program is aborted with an I004 error return.

- e. If the Class Number is not available or the I/O device is down, the Class User is placed in the general wait list (status = 3) until the condition changes.
- f. If the call is successful, the A-register will contain zero on return to the program.

The buffer area furnished by the system is filled with the caller's data if the request is either a WRITE, or a WRITE/READ call. The buffer is then queued (pending) on the EQT entry specified by the Logical Unit Number.

After the driver receives the Class I/O call (in the form of a standard I/O call) and completes, the system will:

- a. Release the buffer portion of the request if a WRITE. The header is retained for the GET call.
- b. Queue the header portion of the buffer in the Completed Class Queue.
- c. If a GET call is pending on the Class Number, reschedule the calling program. (This means that if the user issues a Class GET call or examines the completed Class Queue before the driver completes, the user has effectively beat the system to the completed Class Queue.) Note that the program that issued the Class I/O call and the program that issued the Class GET call do not have to be the same program.
- d. If there is no GET call outstanding, the system continues and the driver is free for other calls.

When the user issues the GET call, the completed Class Queue is checked and only one of the following paths is taken:

- a. If the driver has completed, the header of the buffer is returned (plus the data). The user (calling program) has the option of leaving the I/O request in the completed Class Queue so as not to lose the data. In this case a subsequent GET call will obtain the same data. Or, the user can dequeue the request and release the header and buffer, and can also release the Class Number back to the system.

## EXEC CALLS

- b. If the driver has not yet completed (GET is issued before the Class I/O operation is completed), the calling program is suspended in the general wait list (status = 3) and a marker so stating is entered in the completed Class Queue header. If desired, the program can set the "no wait" bit to avoid suspension. In any case, when the driver completes, any program waiting in the general wait list for this class is automatically rescheduled. Note that only one program can be waiting for any given class at any instant. If a second program attempts a GET call on the same Class Number before the first one has been satisfied, it will be aborted (I/O error IO10).

### 4-41. CLASS I/O - WRITE/READ CALL

Transfers information to or from an external (non-disc) I/O device or another program. Depending upon parameter specifications, the calling program will not be suspended while the call completes.

Assembly Language:

	EXT	EXEC	
	.		
	.		
	.		
	JSB	EXEC	Transfer control to RTE
	DEF	RTN	Return address
	DEF	ICODE	Request code (17=Read; 18=Write; 20=Write/Read)
	DEF	ICNWD	Control information
	DEF	IBUFR	Buffer location
	DEF	IBUFL	Buffer length
	DEF	IPRM1	Optional parameter
	DEF	IPRM2	Optional parameter
	DEF	ICLAS	Class word
RTN	return	point	Continue execution (A=zero or status; B meaningless)
	.		
	.		
ICODE	DEC	numb	17=Read; 18=Write; 20=Write/Read
ICNWD	OCT	conwd	conwd is described in Figure 4-1
IBUFR	BSS	n	Buffer of n words
IBUFL	DEC	n(or-2n)	Same n; words (+) or characters (-)
IPRM1	DEC	f	Optional parameter
IPRM2	DEC	g	Optional parameter
ICLAS	OCT	class	Class is described in comments

FORTTRAN:

```

DIMENSION IBUF
IBUFL = n
ICODE = numb
ICNWD = conwd
ICLAS = class
CALL=EXEC (ICODE,ICNWD,IBUFR,IBUFL,IPRML,IPRM2,ICLAS)
  
```

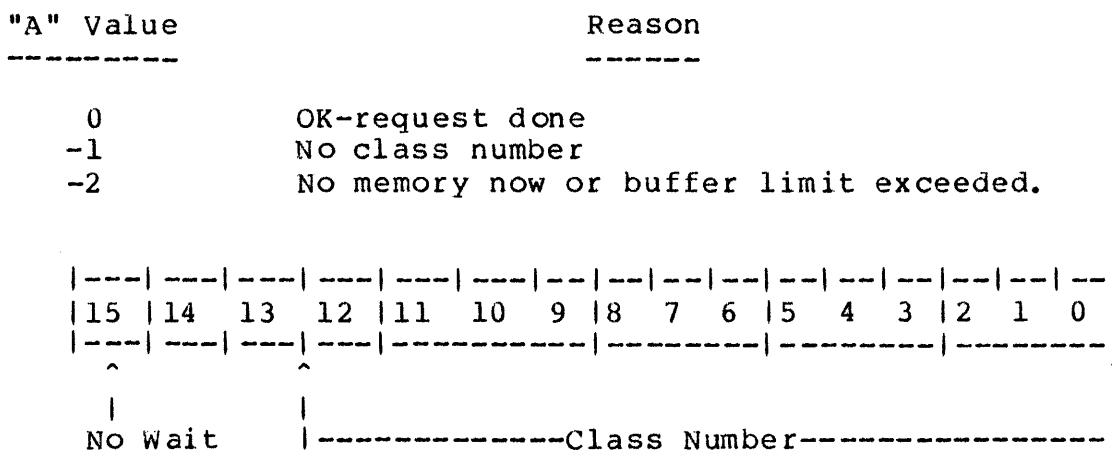
4-42. CLASS I/O WRITE/READ COMMENTS

For a combination Class Write/Read call, the driver should expect control data in the buffer IBUF. The system will treat the request as a Class Write in that the buffer must be moved prior to the driver call, and as a Class Read in that the buffer must be saved after driver completion. Note that the driver will receive a standard Read request (ICODE = 1) on this request.

Refer back to Figure 4-2 for the format of the control word (conwd) required in the Class I/O Write/Read calling sequence.

IPRML and IPRM2 are required as place holders in this request. They may also be used to pass information through to the Class GET call to aid in processing the request.

Figure 4-5 shows the format of the class word (ICLAS) required in the calling sequence. To obtain a Class number from the system, the class portion (bits 12-0) of the word is set to zero. This causes the system to allocate a Class Number (if one is available) to the calling program. The number is returned in the ICLAS parameter when the call completes and the user must specify this parameter (unaltered) when using it for later calls. Bit 15 is the "no-wait" bit. When set the calling program does not memory suspend if memory (or a class number) is not available. The A-register value when the program returns is as follows:





EXEC CALLS

When the user's program issues a Class I/O call the system allocates a buffer from System Available Memory and puts the call in this buffer. The call is queued and the system returns control to the user's program. If memory is not available, three possible conditions exist:

1. The program is requesting more memory space than will ever be available. In this case, the program is aborted with a IO04 error.
2. The program is requesting a reasonable amount of memory but the system must wait until memory is returned before it can satisfy the calling program. The program is suspended unless the "no wait" bit is set, in which case a return is made with the A-register set to -2.
3. If the buffer limit is exceeded, the program will be suspended until this condition clears. If the "no wait" bit is set, the program is not suspended and the A-register is set to -2.

4-43. CLASS I/O - GET CALL

Completes the data transfer between the system and user program that was previously initiated by a class request.

Assembly Language:

	EXT	EXEC	
	.		
	.		
	.		
	JSB	EXEC	Transfer control to RTE
	DEF	RTN	Return address
	DEF	ICODE	Request code
	DEF	ICLAS	Class word
	DEF	IBUFR	Buffer location
	DEF	IBUFL	Buffer length
	DEF	IRTN1	Optional parameter status word
	DEF	IRTN2	Optional parameter status word
	DEF	IRTN3	Optional parameter class word
RTN	return	address	Continue execution (A=status; B=Transmission Log)
	.		
	.		
ICODE	DEC	21	21 = class GET call
ICLAS	NOP		class is described below
IBUFR	BSS	n	Buffer of n words
IBUFL	DEC	n (or -2n)	Same n; words (+) or characters (-)
IRTN1	NOP		Location for IPRM1 from Write/Read call
IRTN2	NOP		Location for IPRM2 from Write/Read call
IRTN3	NOP		Request code passed to driver or initial Read or Write call

FORTRAN:

```

DIMENSION IBUFR (n)
ICODE = 21
IBUFL = n
ICLAS = x0
REG = EXEC(ICODE,ICLAS,IBUFR,IBUFL,IRTN1,IRTN2,IRTN3)

```

## 4-44. CLASS I/O - GET CALL COMMENTS

One of the features of the GET call is that one or more user programs waiting for system resources can suspend themselves without CPU overhead or program overhead such as polling. A program can perform a deliberate GET on a Class Number associated with a device or another program and put itself to sleep. The program will only be awakened when there is something to process. The desired data will be resident in the program's buffer. After the data is processed, the program can put itself to sleep again with another GET.

When the calling program issues a Class GET call, the program is telling the system that it is ready to accept returned data from a Class READ call or remove a completed Class WRITE or CONTROL call from the completed class list. If the driver has not yet completed (GET call got to the completed class before the system), the calling program is suspended in the general wait list (status = 3) and a marker so stating is entered in the Class Queue header. When the driver completes, the program is automatically rescheduled. If desired, the program can set the "no wait" bit to avoid suspension.

Figure 4-5 shows the format of the class word (ICLAS) required in a class GET call. Bits 12-0 represent the Class Number and security code that the GET call is seeking. This Class Number is obtained (in unaltered form) from the original Class I/O READ, WRITE, CONTROL or WRITE/READ call.

Bit 15 is the "no wait" bit. When set, the calling program does not suspend if the class request has not yet completed.

Bit 14 is the "save" bit. When set, the buffer is not released; therefore, a subsequent GET call will return the same the same data.

Bit 13 is the "de-allocate" bit. When set, the Class Number is not returned to the system. If bit 13 is zero and no requests are left in the Pending Class Queue, and no class requests for this class are waiting for driver processing, the class is returned to the system.

It is possible for the call to return the Class Number and data, or no data, depending on whether or not there is one class call left.

Bits 14 and 13 work in conjunction with each other. If bit 14 is set, then the buffer will not be released. Therefore you cannot deallocate the Class Number. That is, the Class Number cannot be released because there is still an outstanding request against it.

## EXEC CALLS

Only when the GET call gets the last class request on a class, or on an empty class queue (completed and pending) can the user release the Class Number by clearing bit 13 in the ICLAS word.

Three parameters in the call are return locations: that is, values from the system are returned to the calling program in these locations. Optional parameters IPRM1 and IPRM2 from the Class I/O WRITE/READ or CONTROL calls are returned in IRTN1 and IRTN2. These words are protected from modification by the driver. The original request code received by the driver is returned in IRTN3, as follows:

Original Request Code -----	Value Returned in IRTN3 -----
17/20 (READ, WRITE/READ)	1
18 (WRITE)	2
19 (CONTROL)	3

IRTN3 is typically used by a program that performs Class I/O with one Class Number to devices (such as slow consoles) to ensure that input and output are buffered and so that input (IRTN3=1) and output (IRTN3=2) are processed along separate paths.

## BUFFER CONSIDERATIONS

There are several buffer considerations in using the Class I/O GET call:

- a. The number of words returned to the user's buffer is the minimum of the requested number and the number in the Completed Class queue element being returned (that was specified in the initial Read/Write in the READ/WRITE call).
- b. If the original request was made with the "Z" bit set in the 1 control word, then IPRM1 returned by this call will be meaningless.
- c. The "Z" buffer will be returned if there is room for it (see "a" above) only if the original request was a READ or WRITE/READ (i.e., for WRITE requests no data is returned in the buffer area).
- d. The remaining words in the user buffer (of any) past the number of words indicated by the transmission log count (in the B-register) are undefined. If a "Z" buffer is used, the words remaining in the buffer past the end of the Z buffer are undefined. Users should not initialize a buffer before doing a GET call, but should clear out the unused words according to the count returned by the transmission log or other parameters returned by the driver.

## A AND B REGISTER RETURNS

The A and B registers are set as follows after a Class I/O GET call:

A-Register -----	B-Register -----
A15 = 0 then A = status	B = transmission log (positive words or characters depending on original request)
A15 = 1 then A = -(numb+1)	B = meaningless

On return with data, bit 15 is set to zero and the rest of the A-register contains the status word (EQT5). If a return is made without data (the "no wait bit" was set in the class word) then bit 15 is set to one and the A-register contains the number of requests numb made to the class bit not yet serviced by the driver (i.e., pending class requests).

## 4-45. CLASS I/O - CONTROL CALL

Carries out various I/O control operations such as backspace, write end-of-file, rewind, etc. The calling program does not wait for the function to be completed.

Assembly Language:

	EXT	EXEC	
	.		
	.		
	JSB	EXEC	Transfer control to RTE
	DEF	RTN	Return address
	DEF	ICODE	Request code
	DEF	ICNWD	Control information
	DEF	IPRAM	Optional parameter
	DEF	ICLAS	Class word
	DEF	IPRM1	Optional parameter
	DEF	IPRM2	Optional parameter
RTN	return	point	Continue execution (A=Class number; B meaningless)
	.		
	.		
ICODE	DEC	19	Request code = 19
ICNWD	OCT	conwd	See Control Word
IPRAM	DEC	n	Required for some control functions; see Control Word
ICLAS	OCT	class	class is described in Comments
IPRM1	DEC	f	Optional parameter passed to GET call
IPRM2	DEC	g	Optional parameter passed to GET call

## EXEC CALLS

### FORTRAN:

Use the FORTRAN I/O statements or an EXEC call sequence.

ICODE = 19	Request code
ICNWD = conwd	See Control Word format in Figure 4-2
IPRAM = x	See Control Word format in Figure 4-2
ICLAS = y	Class Word
REG = EXEC(ICODE, ICNWD, IPRAM, ICLAS)	

### 4-46. CLASS I/O CONTROL COMMENTS

Refer to Figure 4-2 for the format of the control word (conwd) required in the Class I/O Control calling sequence.

Note that this call, with the exception of the ICLAS, IPRM1 and IPRM2 parameters, is the same as the standard I/O Control call. Also refer to the Class I/O GET call for additional information.

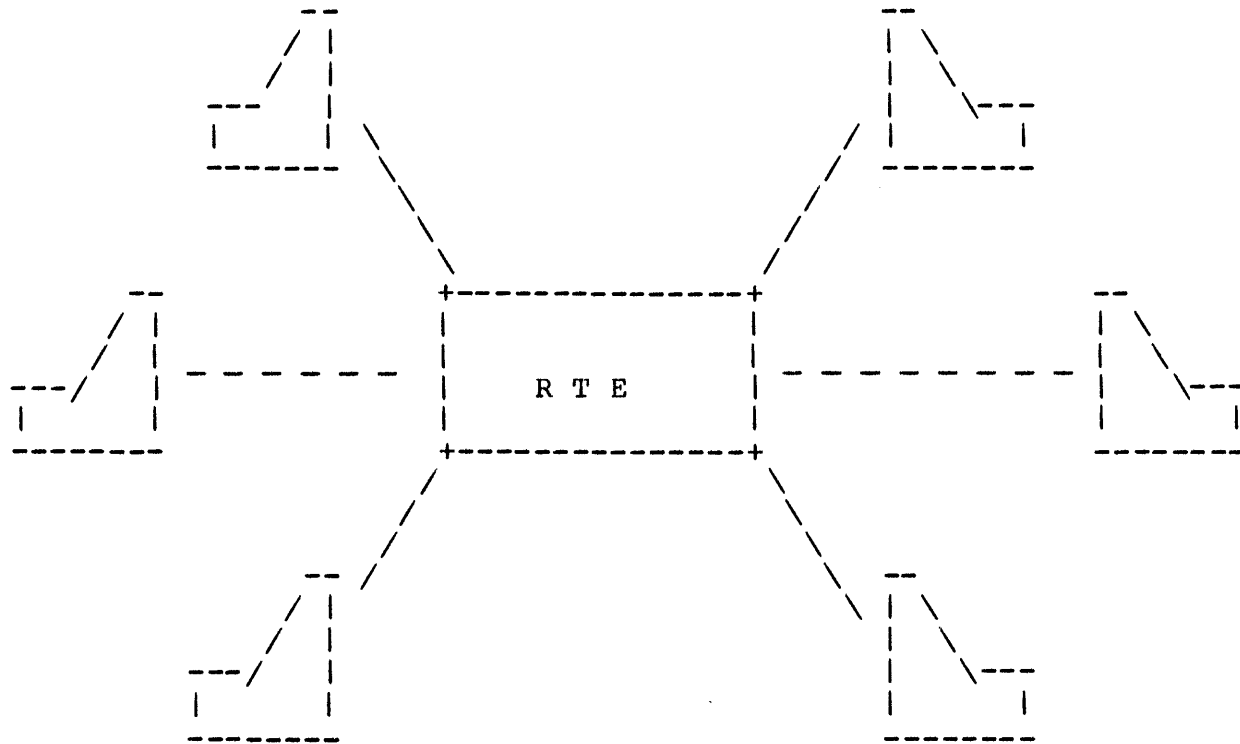
### 4-47. CLASS I/O APPLICATIONS EXAMPLES

One example of using Class I/O is program-to-program (mailbox) communication. The sequence of events that occur are described below, and the calling sequence is illustrated in Figure 4-6.

The range of possible areas where Class I/O could be used to improve applications program performance is too wide and varied to show "typical" examples. The two examples given below are intended only to demonstrate some of the considerations and procedures used in designing specific applications.

## EXAMPLE 1. MULTIPLE TERMINALS WITH A SINGLE APPLICATIONS PROGRAM

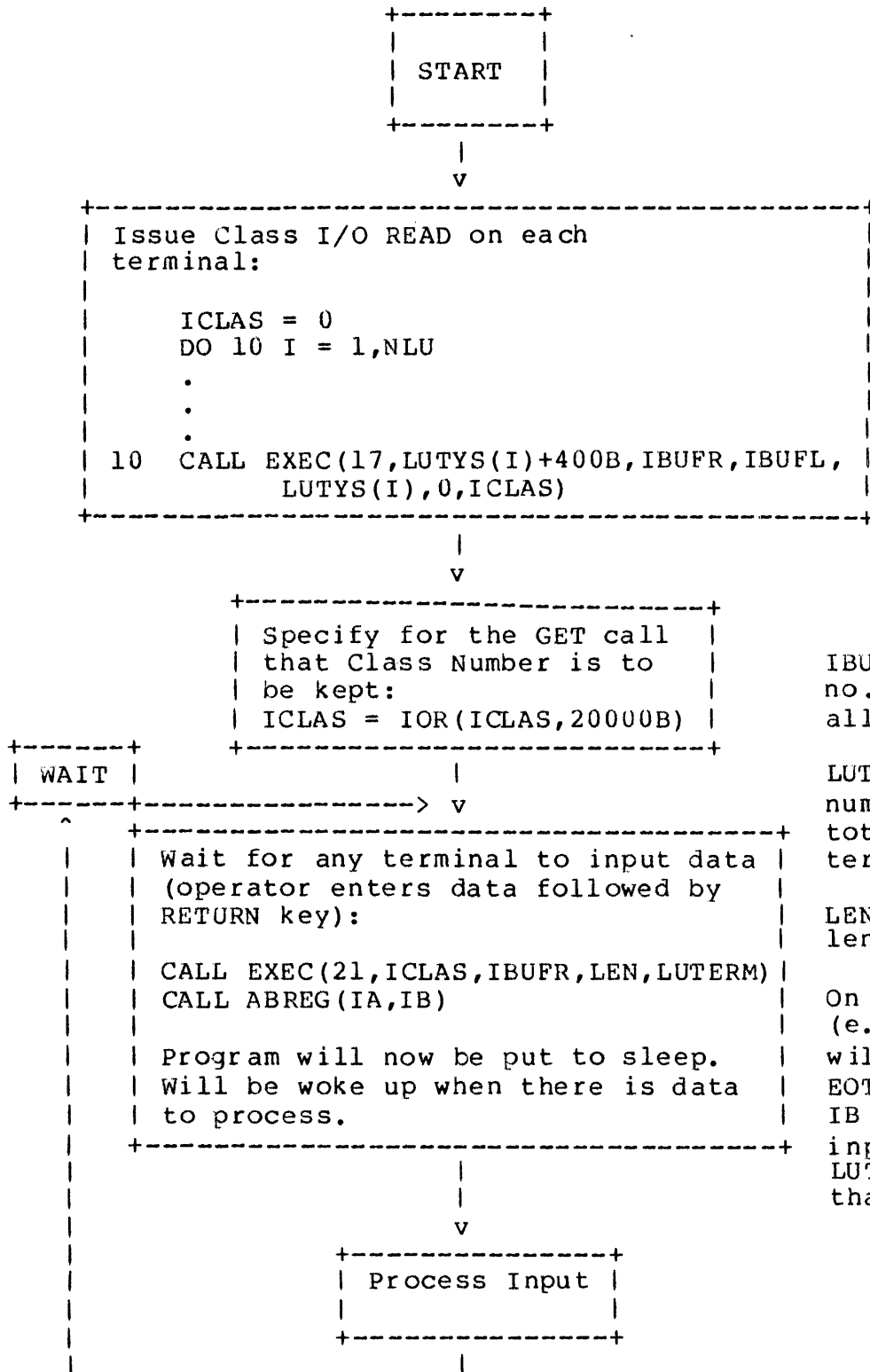
In the following example, any one of many users could be providing input to the program:



Assume an order-entry situation in which there are several operators but only one program. If standard I/O was used, it would be possible to read from only one terminal at a time. However, by using Class I/O, the program permits all operators to enter data seemingly at once. RTE handles all queueing so that the program operates on a single transaction at a time, thus simplifying the programming while giving the appearance of simultaneous processing on all transactions.

EXEC CALLS

The flowchart for such an application is illustrated in Figure 4-6. Note that although operators and terminal devices are shown, the input could be received from any one of a series of indential devices.



Notes:

IBUFL contains negative no. of characters allowed for input.

LUTYS is an array of LU numbers. NLU is the total number of terminals.

LEN contains maximum length of IBUFR.

On return, IA = status (e.g., bit 7 or bit 5 will be set for EOF or EOT respectively). IB = no. of characters input (will be positive). LUTERM will be the LU that responded.

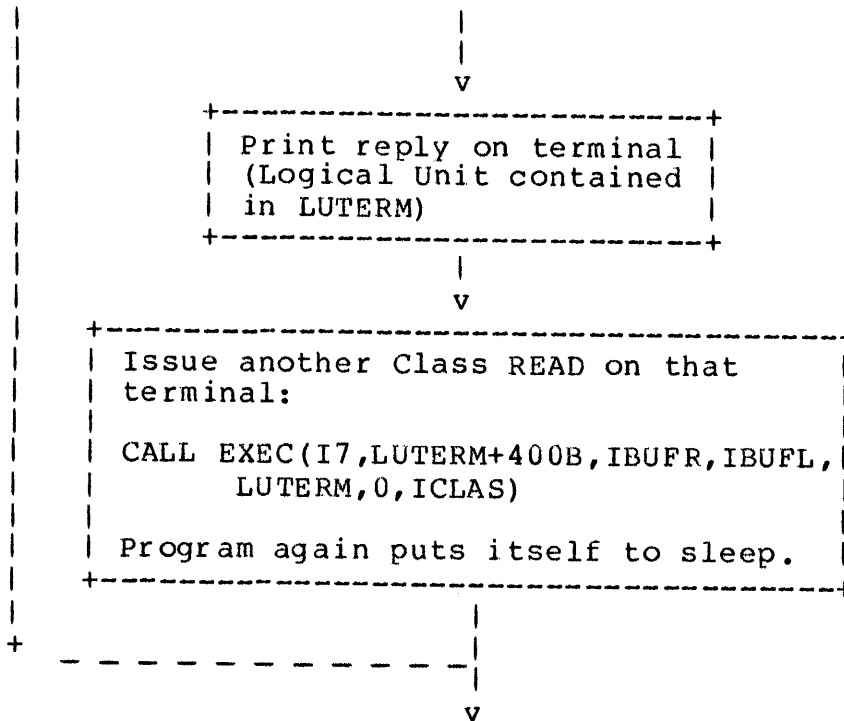


Figure 4-6. Class I/O Multiple Terminal Input Example

In some applications, it may be necessary to maintain contextual information for each operator; for example, a code indicating the type of input expected next, or the operator's name to be used in friendly dialog, etc. This information can be kept in a two-dimensional array that is indexed by the terminal LU number.

For simplicity's sake, let's assume that all terminals have consecutive Logical Unit numbers, starting from 15. The index of the array can then be calculated by subtracting 14 from the LU.



## EXEC CALLS

### EXAMPLE 2. MAILBOX COMMUNICATION BETWEEN PROGRAMS

Program-to-program communication involves a "mailbox" scheme to pass data buffers back and forth in the most expeditious manner. Instead of implementing one large program to process all user inputs, it is often more efficient to separate these into subtasks that are processed by separate programs. In the example below, the program given in the previous example is still used as the "main control," but it now sends user inputs to the appropriate processor by using mailbox I/O. This separation allows the various processors to be given different priorities, with the highest priority being assigned to those items that are most urgently needed. An added benefit is that the separation reduces the partition size requirements.

Assume that the box labeled "Process Input" in Figure 4-6 actually involved several programs, one each for a number of general categories:

- a. Order entry
- b. Inventory quantity look-up
- c. Report generation
- d. Display of status or recent history of several critical real-time activities.

The program illustrated in Figure 4-6 might then serve only as a keyboard entry controller that checks input for legality and calls on other programs to process operator commands. Many operators could now enter commands, with the applications software relying on RTE to queue the commands according to the priority of the category.

The real-time display program might have the highest priority, perhaps followed by order entry, inventory quantity look-up, and report generation last.

Other orderings are possible, depending upon the application. Some management summary reports might be considered most important, or categories may be ordered so that those involving the least processing may have the highest priority to minimize waiting time for users with "short jobs."

The significant point to note is that RTE's priority-driven scheduling functions can be used to process commands according to priority. This is done through the simple means of separating the processes of those commands into separate programs that run at different priority levels, and coordinating the processing via Class I/O.

Figure 4-7 below provides a revised version of the sample program given in Example 1 (Figure 4-6). In this new version, Class Numbers must be allocated for each of the process subprograms and these subprograms must be scheduled. This is performed in the initialization section of the original program as follows:

```

DO 20 I=1,NSUBP
JCLAS=0
CALL EXEC(18,0,IBUFR,0,0,0,JCLAS)
JCLAS=IOR (JCLAS,20000B)
CALL EXEC(21,JCLAS,IBUFR,0)
CALL EXEC(10,<processing program name>,JCLAS)
20 ISUBCL(I)=JCLAS

```

NOTES:

Every Class I/O WRITE, READ, WRITE/READ and CONTROL call issued must ALWAYS be matched with a corresponding GET call issued at some point in the calling sequence. The time sequence is not important (GET's can be issued before Class calls) but there must be a GET for every Class call. Failure to do so will tie up system resources (the Class Number and the system buffer memory) that other programs may need.

When a program is finished with a Class Number, it should explicitly release it with a GET call in which Class Number bits 13 and 14 are cleared and bit 15 is set. Repeat until all buffers are released. The system does NOT release a Class Number when the allocating program terminates.

Programs that issue Class I/O calls may be thought of as "manufacturers," with programs that issue GET calls being thought of as "consumers." It should be clear from the analogy why Class I/O and GET calls must be issued in equal numbers.

## EXEC CALLS

The "Process Input" box previously illustrated Figure 4-6 can then be expanded as illustrated in Figure 4-7 below.

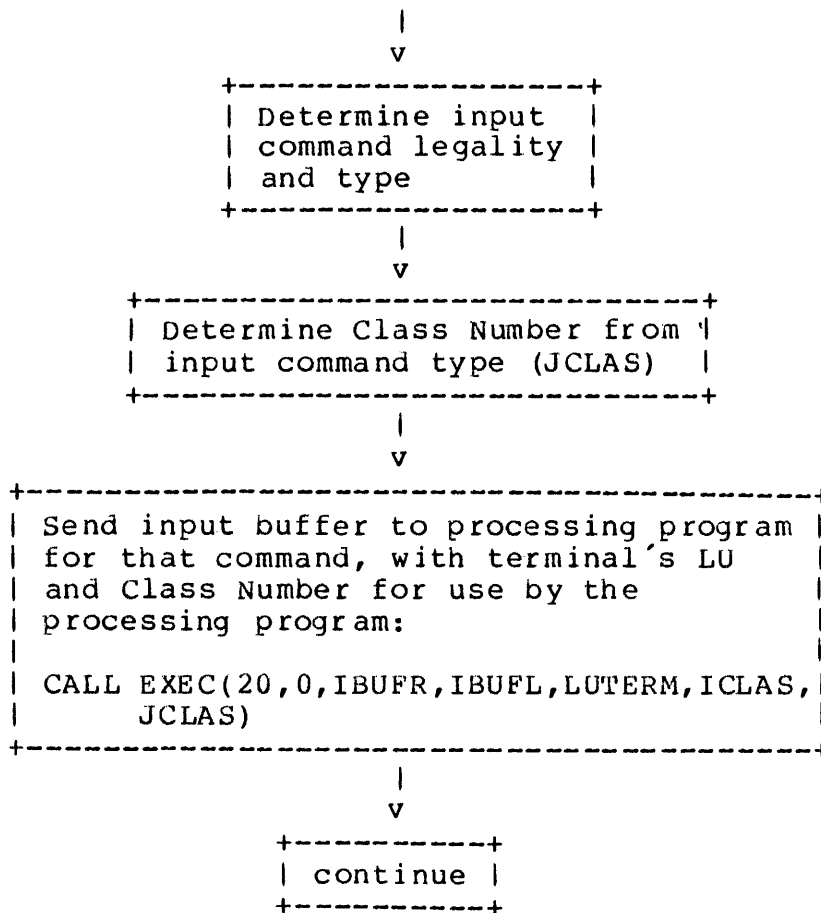


Figure 4-7. Dispatching Input to Subtasks for Processing

Since no devices are involved in mailbox I/O, the CNTWD (second parameter) of the request is zero. For this case, it is usually desirable to let the processing program print an acknowledgement or error return and then issue another Class READ on the terminal. The Class Number to use for this purpose is placed in the second optional parameter. For this reason, in the original example (Figure 4-6), the last two boxes for "printing a reply" and "issuing another Class READ" are deleted and included in the processing programs.

The processing programs obtain the Class Number to use for the above procedure by calling the RMPAR subroutine, as follows:

```

      CALL RMPAR(IPRAM(1)
      MYCLAS = IPRAM(1)
      .
      .
      .
(Initialization code may go here)
      Waits for processing input
      .
      .
      .
100  CALL EXEC(21,MYCLAS,IBUFR,MAXLEN,LUTERM,ITRMCL)
      .
      .
      .
      Process input
      .
      .
      .
      WRITE(LUTERM,1100)
1100  FORMAT (<acknowledgement or error message>)
      .
      .
      .
200  CALL EXEC(17,LUTERM+400B,IBUF,IBUFL,0,ITRMCL)
      GO TO 100

```

The processing programs now issue another Class READ to ITRMCL and return to line 100 to reissue the Class GET on MYCLAS, putting themselves to sleep until other transactions are available for the programs to process.

#### 4-48. RESOURCE NUMBERS AND LOGICAL UNIT LOCKS

Although Resource Numbering and Logical Unit locking services are not implemented through EXEC calls, their discussion logically fits in this section because their ability to synchronize use of system services between cooperating programs is closely associated with Class I/O capabilities. (See the RNRQ subroutine call in the Relocatable Libraries section of this manual.)

Like Class Numbers, the number of Resource Numbers available to the on-site RTE system is determined during system generation. Resource Numbers provide the capability of synchronizing programs that access the same resource. The resource might be a device (locking a Logical Unit requires a Resource Number), a table in memory, a file or even another program or subroutine.

## EXEC CALLS

The use of Resource Numbers is only required when:

- a. TWO or more programs use the same device, or CHANGE the contents of a memory location or disc file.
- b. ONE or more programs make decisions based upon the contents of a data item that can be modified by at least one other program.

To relate the Resource Number mechanism to applications considerations, assume the following "problem" conditions:

PROGRAM A	PROGRAM B
COMMON J	COMMON J
IF(J.EQ.2) J=J+1	IF(J.EQ.2) J=J+3
.	.
.	.
.	.

Assume Programs A and B are both scheduled memory-resident programs and that J, which they share through System COMMON, is initially 2. Further assume Program A executes the IF statement but before it can execute J=J+1, Program B gets scheduled (with B having the highest priority).

Program B sets J to J+3 (making it 5), perhaps performing other tasks, and then terminates.

Program A then increments J, making it 6. Notice that Program A running alone would leave J=3. Program B running alone would leave J=5. Programs A and B running together might leave J=3, 5 or 6.

Now assume that J is a table of tasks to be executed and that there are several programs scanning the table. Also assume the tasks are sufficiently I/O bound that the applications software has several identical programs, each of which may select any task. Without synchronization via Resource Numbers, two or more of these programs might select the same task to work on.

Such "race conditions" can be defined as any code that will execute unexpectedly, depending upon when other programs execute relative to the code. These conditions are an elusive form of software bug, causing unusual errors that can seldom be successfully repeated. Consequently, these errors are much harder to locate and identify.

Standard program priority cannot be relied upon to solve the described problems. Under the dynamics of real-time applications, there are too many other conditions under which a lower priority program occasionally may run when a higher priority program is scheduled.

A high priority program may have to be swapped because a still higher priority disc resident program has been scheduled, and it either has been assigned to the same partition, or the partition is the smallest that the highest priority program will fit into. Meanwhile, the lower priority program may be running in another partition while the other programs are being swapped.

The proper way to avoid race conditions is to assign a Resource Number to all data accesses that are updated by more than one program, or updated by one program and read by others. However, it is extremely important to note and remember three items:

1. The association between a Resource Number (RN) and a shared data area is created through the user's software design. RTE's only role is to make RN's available for allocation, locking, clearing and releasing, and the system will suspend any program that attempts to lock an RN that is already locked. RTE will reschedule the program only when the RN is cleared.
2. All programs that access the same resource MUST cooperate with each other in controlling "simultaneous" access; that is, an RN must be allocated for each resource when RTE is booted up. An RN may be saved anywhere that the cooperating programs can find it. SSGA and COMMON are typical. Programs must lock the RN locally before accessing the associated data base and clear the RN when finished with it.
3. RTE automatically clears all RN's locked locally whenever the locking program is aborted or terminates (unless it terminated saving resources).

#### EXAMPLE 1. TWO PROGRAMS UPDATING A DISC FILE

In this example the file may be either an FMP file or an area in the system track pool on LU 2 or 3. In the first case, the file must be opened non-exclusively (shared). Note that FMP files are normally opened for exclusive use and therefore are NOT sharable. No RN's are necessary to control exclusively opened files. In the second case, the disc tracks must be allocated globally. In either case, the RN must be kept in some area common to all programs (COMMON, SSGA or in the file itself).

## EXEC CALLS

It is poor practice to assume the RN's will always be allocated in the same order; changes in initialization sequences or different RTE generations may change the RN's allocated. When RTE is booted up, an initialization program should be run automatically that will allocate all required RN's and store them where required.

You might possibly choose to use one RN to control access to all data bases. Although this practice consumes the least number of RN's, it is inefficient when several programs need to update different files.

Increasing the number of RN's so that each controls a smaller number of files or area of memory increases the probability that the RN will be clear when the associated resource is required. More RN's therefore reduces the probability of incurring a delay. The number of RN's allowed is limited to 255.

The application itself may limit the minimum area of control, depending upon the circumstances. Typically, one RN per file is the limit. However, one RN should control the set if several files are updated together.

### EXAMPLE 2. DEVICE CONTROL

Programs using a device that many other programs also use (e.g., line printer) should usually lock it first. The Batch/Spool Monitor system provides users with this exclusive control and therefore LU locking is not required. Whenever any other program attempts to access the LU, the calling program will be suspended until the locking program unlocks the LU, terminates or aborts. Note that in this case, cooperation among programs is not required because RTE performs the LU/RN association.

However, when two or more programs employ LU locking, a condition known as "the Deadly Embrace" can sometimes occur.

#### "THE DEADLY EMBRACE"

This potential lock-up condition can occur when programs attempt to lock more than one resource in separate calls. For example, assume the following situation:

- a. Programs PROGA and PROGB are running. PROGA locks the line printer and then begins to output to it, causing PROGA to be suspended.

- b. PROGB runs, locks the magnetic tape unit and outputs to it, causing PROGB to be suspended.
- c. Now assume that PROGA is rescheduled and attempts to use or lock the magnetic tape unit. Since it is already locked by PROGB, PROGA gets suspended.
- d. If PROGB attempts to use or lock the line printer, then it also will be suspended.
- e. PROGA and PROGB each now requires a resource the other currently "owns," and so neither can proceed and will stay "locked-up" together forever unless an operator intervenes.

Figures 4-8A through 4-8C illustrate a typical "deadly embrace" condition. Programs LOCKA and LOCKB share the same COMMON. Program LOCKA allocates and locks LOCK1, and then waits one minute while the operator schedules the LOCKB program. Program LOCKB allocates and locks LOCK2 and then waits one minute.

When program LOCKA runs again, it attempts to lock LOCK2 and is suspended. Program LOCKB attempts to lock LOCK1 and is also suspended.

Figure 4-8C shows a printout by the WHZAT program of this lock-up condition.



PAGE 0001 FTN. 8:07 AM THU., 5 JAN., 1978

```
0001 FTN4,L
0002 PROGRAM LOCKA(3,90)
0003 COMMON LOCK1,LOCK2
0004 ICODE=11B
0005 CALL RNRQ(ICODE,LOCK1,ISTAT)
0006 WRITE(7,1) ISTAT
0007 1 FORMAT("LOCKA:STATUS LOCK # 1="I5)
0008 CALL EXEC(12,0,3,0,-1)
0009 ICODE2 = 1
0010 CALL RNRQ(ICODE2,LOCK2,ISTAT)
0011 WRITE(7,2) ISTAT
0012 2 FORMAT("LOCKA:STATUS LOCK#2="I5)
0013 END
```

FTN4 COMPILER: HP92060-16092 REV. 1805 (780113)

\*\* NO WARNINGS \*\* NO ERRORS \*\* PROGRAM = 00081 COMMON = 00002

Figure 4-8A. "Deadly Embrace" Example

PAGE 0001 FTN. 8:07 AM THU., 5 JAN., 1978

```
0001 FTN4,L
0002 PROGRAM LOCKB(3,90)
0003 COMMON LOCK1,LOCK2
0004 ICODE=118
0005 CALL RNRQ(ICODE,LOCK2,ISTAT)
0006 WRITE(7,1) ISTAT
0007 1 FORMAT("LOCKB:STATUS LOCK # 1="I5)
0008 CALL EXEC(12,0,3,0,-1)
0009 ICODE2 = 1
0010 CALL RNRQ(ICODE2,LOCK1,ISTAT)
0011 WRITE(7,2) ISTAT
0012 2 FORMAT("LOCKB:STATUS LOCK#2="I5)
0013 END
```

FTN4 COMPILER: HP92060-16092 REV. 1805 (780113)

\*\* NO WARNINGS \*\* NO ERRORS \*\* PROGRAM = 00081 COMMON = 00002

Figure 4-8B. "Deadly Embrace" Example

```

***
*** 8: 8: 2:480
*****
**PT SZ PRGRM,T ,PRIOR*DRMT*SCHD*I/O *WAIT*MEMY*DISC*OPER * NEXT TIME *
*****
*** 0 ** R$PNS*1 *00010 ***** 3,CL 032
*** 0 ** WHZAT*1 *00001 ***** 1
*** 1 7 FMG07*3 *00090 ***** 3,WHZAT
*** 2 4 LOCKA*3 *00090 ***** 3,RN 028,LKPRG=LOCKB
*** 3 4 LOCKB*3 *00090 ***** 3,RN 029,LKPRG=LOCKA
*****
**DOWN LU'S
*****
**DOWN EQT'S
*****
*** 8: 8: 2:540
***

```

Figure 4-8C. "Deadly Embrace" WHZAT Example

In the case of devices, this condition can be avoided by locking all devices that may be required at once in the same call. The program will be suspended until all devices are available.

In the case of Resource Numbers, the condition can generally be avoided by increasing the "area of control" of the Resource Numbers so that a program requiring simultaneous and exclusive access to two files (for instance) merely locks one RN, rather than one for each file. If an applications problem does not allow this solution, then the user should attempt to lock all RN's required without suspension (bit 15 of ICODE is set).

If a lock cannot be granted, attempt the following steps:

1. DO NOT update any of the related files; post whatever has already been processed (ONLY for those files to which exclusive access has already been obtained).
2. Release all RN's that are locked and re-attempt to lock the last RN, this time with suspension.
3. When the lock is granted, re-lock all the previous RN's and continue. Note that RTE will allow a program to locally lock an RN that it has already locked locally.

In summary, if a program MUST lock more than one resource and finds one or more of these resources already in use, the program should "back off," release all RN's it has already locked (if any), wait for the resource it wanted to become available, and then re-attempt to lock all RN's it needs. The program must NOT fully or partially update any files, unless it has all the RN's locked that control access to the file and any related files that must be updated simultaneously.

#### 4-49. EXECUTIVE ERROR MESSAGES

When RTE-IV discovers an Executive error, it normally terminates the program, releases any disc tracks assigned to the program, issues an error message to the system console and proceeds to execute the next program in the scheduled list.

The user may specify the non-abortion of a program for some Executive error conditions. See Section 4-4 for a detailed discussion of this option.

The error messages described below are those that may occur while accessing the Executive. They are grouped according to type. Table 4-2 contains a summary of all possible errors associated with EXEC calls.

## EXEC CALLS

### 4-50. MEMORY PROTECT VIOLATIONS

The RTE-IV operating system is protected by a hardware memory protect. Consequently, any user program that illegally tries to modify or jump to the operating system will cause a memory protect interrupt. The operating system intercepts the interrupt and determines its legality. If the memory protect is illegal, the program is aborted and the following message is displayed on the system console:

```
MP INST = xxxxxx      (offending octal instruction code)
ABE pppppp qqqqqq r  (contents of A, B and E registers at abort)
XYO pppppp qqqqqq r  (contents of X, Y and O registers at abort)
MP yyyyyy zzzzz      (yyyyyy=program name; zzzzz=violation address)
yyyyy ABORTED
```

### 4-51. DYNAMIC MAPPING VIOLATIONS

A dynamic mapping violation occurs when an illegal read or write occurs to a protected page of memory. This may happen when a user program tries to write beyond its own address space to non-existent memory or to some other program's memory. In this case, the program is aborted and the following message is issued:

```
DM VIOL = wwww      (contents of DMS violation register)
DM INST = xxxxxx      (offending octal instruction code)
ABE pppppp qqqqqq r  (contents of A, B and E registers at abort)
XYO pppppp qqqqqq r  (contents of X, Y and O registers at abort)
DM yyyyyy zzzzz      (yyyyyy=program name; zzzzz=violation address)
yyyyy ABORTED
```

### 4-52. DISPATCHING ERRORS

It is possible for programs to be scheduled and discover at a later time that there is no partition large enough to dispatch the program. This could occur if a parity error downed a partition and that partition was the largest of its type (i.e., BG, RT, or EMA). If this occurs, the program will be aborted with a DP error. The format of the error message is:

```
ABE pppppp qqqqqq r  (contents of A,B, and E registers at abort)
XYO pppppp qqqqqq r  (contents of X,Y, and O registers at abort)
DP yyyyyy zzzzz      (yyyyyy = program name; zzzzz = violation address)
yyyyy aborted
```

## 4-53. EX ERRORS

It is possible to execute in the privileged mode; that is, with the interrupt system off. Therefore, the user may not make EXEC calls in this mode because the memory protect, which is the access vehicle to EXEC, is off. An attempt to make an EXEC call with the interrupt system off causes the calling program to be aborted and the following message issued:

```

ABE pppppp qqqqqq r      (contents of A,B and E registers at abort)
XYO pppppp qqqqqq r      (contents of X,Y and O registers at abort)
EX   YYYYYY zzzzz      (YYYYYY=program name; zzzzz=violation address)
YYYYY ABORTED

```

## 4-54. UNEXPECTED DM AND MP ERRORS

The operating system handles all DM and MP violations. Some of these violations are legal; others are not. In any case, the operating system associates these violations with program activity. A DM or MP violation occurring when no program is active is an unexpected violation. Since no program is present there is no program to abort. In such a case, one of the following messages will be issued:

```

DM VIOL = wwwwww      (contents of DMS violation register)
DM INST = xxxxxx      (offending octal instruction code)
ABE pppppp qqqqqq r  (contents of A, B and E registers at abort)
XYO PPPPPP qqqqqq r  (contents of X, Y and O registers at abort)
DM <INT>      0

```

or

```

MP INST = xxxxxx      (offending octal instruction code)
ABE pppppp qqqqqq r  (contents of A, B and E registers at abort)
XYO pppppp qqqqqq r  (contents of X, Y and O registers at abort)
MP <INT>      0

```

Both of the above messages specify <INT> as the program name to signal the user that an unexpected memory protect or dynamic mapping violation error has occurred. Either is a serious violation of the operating system integrity. Usually, it indicates that user-written software (driver, privileged subroutine, etc.) has damaged the operating system integrity or has inadequately performed required (driver) system housekeeping. However, it could also mean that the CPU has failed and that the operating system detected the failure in time to prevent a system crash.

If this error occurs, it is recommended that all users on the system save whatever they were doing (i.e., finish up editing, etc.) and reboot the system. If only HP modules are present in the operating system, CPU failure is a highly likely cause of the error and CPU diagnostics should be run prior to rebooting.

## EXEC CALLS

### 4-55. TI, RE AND RQ ERRORS

The following errors have the same format as the MP and DM error returns except that the register contents are not reported:

Error -----	Meaning -----
TI	Batch program exceeds allowed time.
RE	Reentrant subroutine attempted recursion.
RQ	Illegal request code is not between 1 and 26, or (in text) an RQ00 means that the address of a returned parameter is below the memory protect fence.  An RQ00 error means that the address of a returned parameter is below the memory protect fence.

### 4-56. PARITY ERRORS

Upon detecting a "hard" parity error (i.e., one that is reproducible). RTE will abort the program that encountered the parity error and the following message will be issued:

```
PE PG#   nnnnn BAD
ABE aaaaa bbbbb e
XYO xxxxx yyyyy o
PE ppppp mmmmm
ppppp ABORTED
```

where:

nnnnn = physical page number where the parity error was detected (page number counting starts at 0).

ABE = contents of the A, B, and E-registers respectively when the parity error was detected.

XYO = contents of the X, Y, and O-registers respectively when the parity error was detected.

ppppp = program name.

mmmmm = logical memory address of parity error.

If the program was disc resident, the following message will be issued:

```
PART'N xx DOWN
PART'N yy DOWN
```

where:

xx = the partition the program was running in.

yy = the mother partition program if any are affected

Alternately, if xx is a mother partition, then yy is a subpartition that contained the parity error. In either case, partition xx and yy will no longer be available for running user programs until the system is next booted up.

Upon detecting a "soft" parity error (i.e., one that is not reproducible), RTE is not able to locate the physical page number of the parity error. The following message is then issued:

```
PE @ mmmmm
DMS STAT = zzzzz
```

where:

mmmmm = logical address of parity error.

zzzzz = DMS status register.

A parity error occurring within the operating system itself, a driver or system table area causes the system to execute a HLT 102005,

where:

A-register = physical page number where the parity error was detected (page number counting starts at 0).

B-register = logical memory address of the parity error.

A parity error occurring in a DCPC transfer when the operating system is executing in the System Map causes the system to execute a HLT 103005, where the A and B-registers are as above.

#### 4-57. OTHER EXEC ERRORS

The general format for the following errors is

```
type name address
```

where:

type = a four-character error code (DR, SC, IO, RN, LU)

name = the program that made the call.

address = the location of the call (equal to the exit point if the error is detected after the program suspends).



## EXEC CALLS

### 4-58. DISC ALLOCATION ERROR MESSAGES

- DR01 = Not enough parameters
- DR02 = Number of tracks zero, illegal logical unit, or number of tracks to release is zero or negative.
- DR03 = Attempt to release track assigned to another program.

### 4-59. SCHEDULE CALL ERROR CODES

- SC00 = Batch program attempted to suspend (EXEC (7)).
- SC01 = Missing parameter.
- SC02 = Illegal parameter.
- SC03 = Program cannot be scheduled.
- SC03 INT = Occurs when an external interrupt attempts to schedule a program that is already scheduled. RTE-III ignores the interrupt and returns to the point of interruption.
- SC04 = name is not a subordinate (or "son") of the program issuing the completion call.
- SC05 = Program given is not defined.
- SC06 = No resolution code in Execution Time EXEC Call (not 1, 2, 3, or 4).
- SC07 = Prohibited memory lock attempted.
- SC08 = The program just scheduled is assigned to partition smaller than the program itself or to an undefined partition. Unassign the program or reassign the program to a partition that is as large or larger than the program.
- SC09 = The program just scheduled is too large for any partition of the same type. For example, trying to schedule a 23K background program when the largest background partition is only 21K.
- SC10 = Not enough system available memory for string passage.

### 4-60. I/O CALL ERROR CODES

- IO00 = Illegal call number. Outside table, not allocated, or bad security code.

- IO01 = Not enough parameters.  
X bit set.
- IO03 = Illegal EQT referenced by LU in I/O call (Select code=0).
- IO04 = Illegal user buffer. Extends beyond RT/BG area or not enough system available memory to buffer the request.
- IO05 = Illegal disc track or sector.
- IO06 = Reference to a protected track; or using LG tracks before assigning them (see LG, Section III).
- IO07 = Driver has rejected call.
- IO08 = Disc transfer longer than track boundary.
- IO09 = Overflow of LG area.
- IO10 = Class GET call issued while one call already outstanding.
- IO11 = Type 4 program made an unbuffered I/O request to a driver that did not do its own mapping.

#### 4-61. PROGRAM MANAGEMENT ERROR CODES

- RN00 = No option bits set in call.
- RN01 = Not used.
- RN02 = Resource Number not defined.
- RN03 = Unauthorized attempt to clear a LOCAL Resource Number.

#### 4-62. LOGICAL UNIT LOCK ERROR CODES

- LU01 = Program has one more logical units locked and is trying to LOCK another with wait.
- LU02 = Illegal logical unit reference (greater than maximum number).
- LU03 = Not enough parameters furnished in the call. Logical unit reference less than one. Logical unit not locked to caller.

#### 4-63. EXECUTIVE HALT ERRORS

## EXEC CALLS

There are several HLT instructions included in the RTE operating system that indicate a serious violation of the integrity of the operating system. Usually, these errors indicate that the CPU or one of its subsystems (DCPC, Memory Protect, etc.) has failed. However, they could indicate that user-written software (driver, priveleged subroutine, etc.) has damaged the operating system integrity or has inadequately performed required (driver) system housekeeping. If these HLT's occur, it is recommended that the user check out his hardware with the appropriate diagnostics.

HLT 0 Located in Table Area I

HLT 2 Located in location 2 of the system map

HLT 3 Located in location 3 of the system map

HLT 6 System tried to remove a partition from a list and the partition was not found there.

Other system HLT's exist for which there is some corrective action:

HLT 4 Powerfail occured and the powerfail/autorestart subsystem was not installed.

HLT 5 Parity error in system map. See Parity Error discussion in this section.

HLT 5,C Parity error in a DCPC transfer when operating system was executing in the system map. See Parity Error discussion in this section.

HLT 10 At startup, the system discovered that there was no partition large enough to execute FMGR or D.RTR.

A summary of EXEC call error messages is provided in Table 4-5.



Table 4-5 EXEC Call Error Summary

ERROR	MEANING	READ	WRITE	CONTROL	PROGRAM TRACK ALLOCATE	PROGRAM TRACK RELEASE	PROGRAM COMPLETION	PROGRAM SUSPEND	PROGRAM SEGMENT LOAD	PROGRAM SCHEDULE W/WAIT	PROGRAM SCHEDULE WO/WAIT	TIME REQUEST
		1	2	3	4	5	6	7	8	9	10	11
DR01	Not Enough Parameters 1. Less than 4 parameters. 2. Less than 1 parameter. 3. Number = -1. 4. Less than 3 (not -1).				1	2 4						
DR02	Illegal Track Number or Logical Unit Number. 1. Track number = 0. 2. Logical Unit not 2 or 3. 3. Deallocate 0 or less Tracks.				1	2 3						
DR03	Attempt to release Track assigned to another program.					X						
IO00	Illegal Class Number 1. Outside Table. 2. Not allocated. 3. Bad Security Code.											
IO01	Not Enough Parameters. 1. Zero parameters. 2. Less than 3 parameters. 3. Less than 5/disc. 4. Less than 2 parameters. 5. Class word missing.	1 2 3	1 2 3	1								
IO02	Illegal Logical Unit 1. 0 or maximum. 2. Class request on disc LU. 3. Less than 5 parameters and X-bit set.	1 3	1 3	1								
IO03	Illegal EQT command by LU in I/O call; delete code = 0	X	X	X								
IO04	Illegal User Buffer. 1. Extends beyond RT/BG area. 2. Not enough system memory to buffer the request.	1										
IO05	Illegal Disc Track or Sector 1. Track number maximum. 2. Sector number 0 or maximum	1 2	1 2									
IO06	Attempted to WRITE to LU2/3 and track not assigned to user or globally, or not to next load-and-go sector. Illegal WRITE to a FMP track. Attempted to use copy of loader to make permanent load or delete		X									
IO07	Driver has rejected request and request is not buffered.	X	X	X								
IO08	Disc transfer implies track switch (LU2/3)	X	X									
IO09	Overflow of LG area		X									
IO10	Class GET and one call already outstanding											
IO11	Illegal User Map request for System Driver area	X	X	X								

Table 4-5 EXEC Call Error Summary (Cont.)

PROGRAM SCHEDULE TIME 12	I/O STATUS 13	STRING PASSAGE 14	GLOBAL TRACK ALLOCATE 15	GLOBAL TRACK RELEASE 16	CLASS I/O READ 17	CLASS I/O WRITE 18	CLASS I/O CONTROL 19	CLASS I/O WRITE/READ 20	CLASS I/O GET 21	PROGRAM SWAPPING CONTROL 22	PROGRAM SCHED QUEUE W/WAIT 23	PROGRAM SCHED QUEUE WO/WAIT 24	RNRQ	LURQ
			1	3 4										
			1	2 3										
					1 2 3	1 2 3	1 2 3	1 2 3	1 2 3					
	1  4				1 2  5	1 2  5	1  5	1 2  5						
	1				1 2 3	1 2 3	1 2 3	1 2 3						
	X				X	X	X	X	X					
					2	2	2	2	1					
									X					

Table 4-5 EXEC Call Error Summary (Cont.)

ERROR	MEANING	READ 1	WRITE 2	CONTROL 3	PROGRAM TRACK ALLOCATE 4	PROGRAM TRACK RELEASE 5	PROGRAM COMPLETION 6	PROGRAM SUSPEND 7	PROGRAM SEGMENT LOAD 8	PROGRAM SCHEDULE W-WAIT 9	PROGRAM SCHEDULE WO-WAIT 10	TIME REQUEST 11
LU01	Program has one or more logical units locked and is trying to LOCK another with WAIT.											
LU02	Illegal logical unit reference (greater than maximum number).											
LU03	Not enough parameters furnished in the call. Illegal logical unit reference (less than one). Logical unit not locked to caller.											
RQ00	Return buffer below memory protection fence.	X			X							X
RQ	EXEC call contains an illegal request code. 1. Return address indicates less than one or more than seven parameters. 2. Parameter address indirect through A- or B-Register. 3. Request code not defined or not loaded.	X	X	X	X	X	X	X	X	X	X	X
RN00	No option bits set.											
RN01	Not used.											
RN02	Resource number not in Table (undefined).											
RN03	Unauthorized attempt to clear a LOCAL Resource Number.											
SC00	Batch program cannot suspend.							X				
SC01	Missing Parameter. 1. Segment name missing. 2. Not 4 or 7 parameters in Time Call. 3. Not 4 parameters in String Passage Call or partition status call.								1			
SC02	Illegal Parameter 1. Option word is missing or not 0, 1, 2, or 3. 2. Read/write word in String Passage Call is not 1 or 2.						1					
SC03	Program Cannot Be Scheduled. 1. Not a segment. 2. Is a segment.								1	2	2	
SC04	Attempted to control a program that is not a "Son."						X					
SC05	Program Given is Not Defined. 1. No segment. 2. No program. 3. "Son" not found.								1	2	2	
SC06	Resolution not 1, 2, 3, or 4.						3					
SC07	Prohibited core memory lock attempted.											
SC08	Assigned partition is too small for program									X	X	
SC09	Program too large for any partition of same type									X	X	
SC10	Not enough system available memory for string passage.									X	X	

Table 4-5 EXEC Call Error Summary (Cont.)

PROGRAM SCHEDULE TIME 12	I/O STATUS 13	STRING PASSAGE 14	GLOBAL TRACK ALLOCATE 15	GLOBAL TRACK RELEASE 16	CLASS I/O READ 17	CLASS I/O WRITE 18	CLASS I/O CONTROL 19	CLASS I/O WRITE/READ 20	CLASS I/O GET 21	PROGRAM SWAPPING CONTROL 22	PROGRAM SCHEDULE UI W-WAIT 23	PROGRAM SCHEDULE UI WO-WAIT 24	RNRO	IURO
														x
														x
														x
	x		x			x	x	x	x					
x	x	x	x	x	x	x	x	x	x	x	x	x		
														x
														x
														x
2		3												
		2								1				
											2	2		
2											2	2		
										x				
														x
														x
		x									x	x		



INPUT/OUTPUT	SECTION V
--------------	-----------

In the Real-Time Executive System, centralized control and logical referencing of I/O operations effect simple, device-independent programming. Each I/O device is interfaced to the computer through an I/O controller associated with one or more I/O select codes that are hardware-linked to corresponding memory locations for interrupt processing. By means of several user-defined I/O tables, self-contained multi-device drivers and program EXEC calls, RTE relieves the programmer of many I/O processing details.

For details on the hardware input/output organization, consult the appropriate computer manuals (refer to the documentation map at the beginning of this manual). For details on writing drivers, see the RTE Driver Writing Reference Manual.

For a full understanding of the software I/O characteristics of RTE as described in this manual section, the user should be familiar with two hardware-related terms:

1. I/O Controller - a combination of I/O card, cable and, for some devices, a controller box used to control one or more I/O devices on a computer I/O select code.
2. I/O Device - a physical unit (or portion of a unit) identified in the operating system by means of an Equipment Table (EQT) entry and a subchannel assignment.

Each I/O device is interfaced to the computer through an I/O controller that is associated with one or more of the computer I/O select codes. Interrupts from controllers on specific select codes are directed to specific computer memory locations for system processing.

#### 5-1. SOFTWARE I/O STRUCTURE

This description of the I/O software is primarily intended for those who will be using I/O EXEC calls for standard programming applications. Users who will be writing their own drivers or who may otherwise require a more detailed knowledge of the I/O internal structure should consult the RTE Driver Writing Reference Manual.

The I/O structure is made up of two general types of software:

## INPUT/OUTPUT

1. The system I/O processor (RTIOC) and various device drivers.
2. A number of I/O tables, including: Equipment Table, Device Reference Table, Interrupt Table, Driver Mapping Table, plus a Base Page Communications area.

These tables and areas are used for communication between the system and the drivers, and for control of the many I/O operations that can be in progress simultaneously.

An Equipment Table entry records each controller's I/O select code, driver, DCPC, buffering and time-out specifications. A Device Reference Table assigns one or more Logical Unit numbers to each device and points each device to the appropriate Equipment Table entry. This allows the programmer to reference changeable logical units instead of fixed physical units.

An Interrupt Table directs the system's action when an interrupt occurs on any select code. RTE can call a driver that is responsible for initiating and continuing operations on all devices' controllers of an equivalent type, schedule a specified program, or handle the interrupt itself.

The programmer requests I/O by means of an EXEC call that specifies the logical unit, control information, buffer location, buffer length, and type of operation. Some subsystems may require additional parameters.

### 5-2. EQUIPMENT TABLE

The Equipment Table (EQT) is used to maintain a list of all the I/O equipment in the system. The table consists of a number of EQT entries, with one EQT entry for each I/O controller defined in the system at generation time. Each EQT entry contains all of the information required by the system and associated driver to operate the device, including:

- \* I/O select code in which the controller is interfaced with the computer.
- \* Driver type.
- \* Various driver or controller requirements and specifications, such as DCPC, buffering, time-out, power fail, etc.

These 15-word EQT entries reside in the system and have the format illustrated in Figures 5-1 and 5-2. Note that some information in an EQT entry is static; other parts are dynamic. Information marked <A> is fixed at generation time or during I/O reconfiguration at boot-up time and never changes during on-line system operation. Words marked <B> are also fixed during generation or I/O reconfiguration but can be changed on-line through operator commands. Information marked <C>, <D> and <E> are driver considerations. <F> is maintained by the system.

Word	Contents																
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
1	R	I/O Request List Pointer <C>															
2	R	Driver Initiation Section Address <A >															
3	R	Driver Continuation/Completion Section Address <A>															
4	D	B	P	S	T	Subchannel					I/O Select Code #						
	<A>	<B>	<E>	<E>	<C>	<C>					<A>						
5	AV		EQUIPMENT TYPE CODE							STATUS							
	<F>		<A>							<E>							
6	CONWD (Current I/O Request Word) <C>																
7	Request Buffer Address <C>																
8	Request Buffer Length <C>																
9	Temporary Storage <D> or Optional Parameter <C>																
10	Temporary Storage <D> or Optional Parameter <C>																
11	Temporary Storage for Driver <D>																
12	Temporary Storage for Driver <D>							or	EQT Extension Size, any <A>								
13	Temporary Storage for Driver <D>							or	EQT Extension Starting Address, if any <A>								
14	Device Time-Out Reset Value <B>																
15	Device Time-Out Clock <C>																

Figure 5-1. Equipment Table Entry Format

where:

R = reserved for system use.

## INPUT/OUTPUT

### I/O Request

List Pointer = points to list of requests queued up on this  
EQT entry. First entry in list is current  
request in progress (zero if no request).

D = 1 if DCPC required.

B = 1 if automatic output buffering used.

P = 1 if driver is to process power fail.

S = 1 if driver is to process time-out.

T = 1 if device timed out (system sets to zero before  
each I/O request).

Subchannel# = last subchannel addressed.

I/O Select = I/O select code for the I/O controller  
Code# (lower number if a multi-board interface).

AV = I/O controller availability indicator:

0 = available for use.

1 = disabled (down).

2 = busy (currently in operation).

3 = waiting for an available DCPC channel.

EQUIPMENT = type of device on this controller. When this octal  
TYPE CODE number is linked with "DVy," it identifies the  
device's software driver routine. Some standard driver  
numbers are:

00 to 07 = paper tape devices or consoles

00 = teleprinter or keyboard control device

01 = photoreader

02 = paper tape punch

05 = 264x-series terminals

07 = multi-point devices

10 to 17 = unit record devices

10 = plotter

11 = card reader

12 = line printer

15 = mark sense card reader

20 to 37 = magnetic tape/mass storage devices

23 = 9-track magnetic tape

31 = 7900 moving head disc

32 = 7905/06/20 moving head disc

33 = flexible disc drives

36 = writable control store

37 = HPIB

40 to 77 = instruments

STATUS = actual physical status or simulated status at the end of each operation.

CONWD = combination of user control word and user request code word in the I/O EXEC call (see Section IV; see also Figure 5-2 below).

and where the letters in brackets (<>) indicate the nature of each data item as follows:

<A> = fixed at generation or reconfiguration time; never changes

<B> = fixed at generation or reconfiguration time; can be changed on-line

<C> = set up or modified at each I/O initialization

<D> = available as temporary storage by driver

<E> = can be set driver

<F> = maintained by system

INPUT/OUTPUT

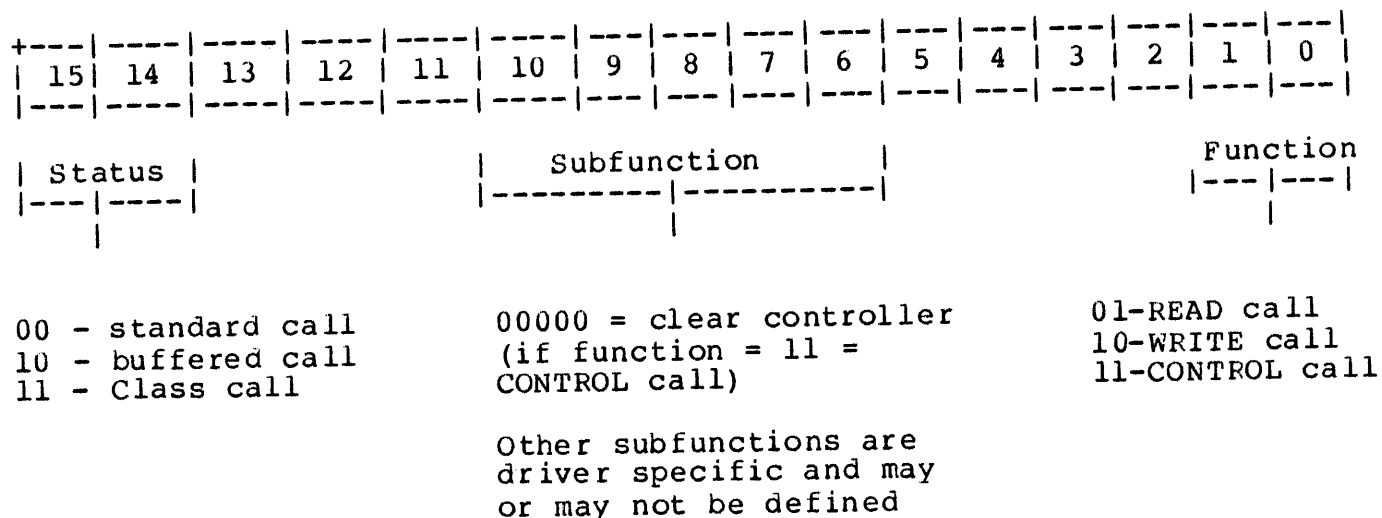


Figure 5-2. CONWD Word (EQT Entry Word 6) Expanded

When RTE initiates or continues an I/O operation (except for privileged driver constructions), it places the address of the EQT entry for the device's controller into the Base Page Communications area before calling the driver routine.

All Equipment Table entries are located sequentially in memory, beginning with EQT entry number 1. The address of the first entry and the total number of entries in the table can be found in the Base Page Communications area.

## 5-3. DEVICE REFERENCE TABLE

The Device Reference Table (DRT) is part of the mechanism by which Logical Unit numbers for I/O are implemented (see Logical Unit Numbers below). Users request I/O by specifying a Logical Unit (LU) number. The DRT translates this Logical Unit number into a physical device as specified by an EQT entry number and subchannel. The DRT is also used to queue requests for I/O on an unavailable (down) device. The request list for available (up) devices originates from word 1 of the EQT entry, as illustrated in Figure 5-1.

Each DRT entry is two words long (see Figure 5-3). There is one entry for each Logical Unit number defined at generation time, beginning with Logical Unit 1.

The first word of each entry includes the EQT entry number of the controller assigned to the logical unit and the subchannel number of the specific device on that controller to be referenced.

The second word of each DRT entry contains the current status of the logical unit; up (available) or down (unavailable). If the device is down, word 2 also contains a pointer to the list of requests waiting to access the LU. Figure 5-3 illustrates the format of a Device Reference Table entry, and Figure 5-4 illustrates the Device Reference Table.

Subchannel No.					(Reserved)					EQT Entry Number						
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	word 1
F	Downed I/O Request List Pointer														word 2	

where:

F (up/down flag) = 0 if device is up  
 = 1 if device is down

Figure 5-3. Device Reference Table Entry Format

INPUT/OUTPUT

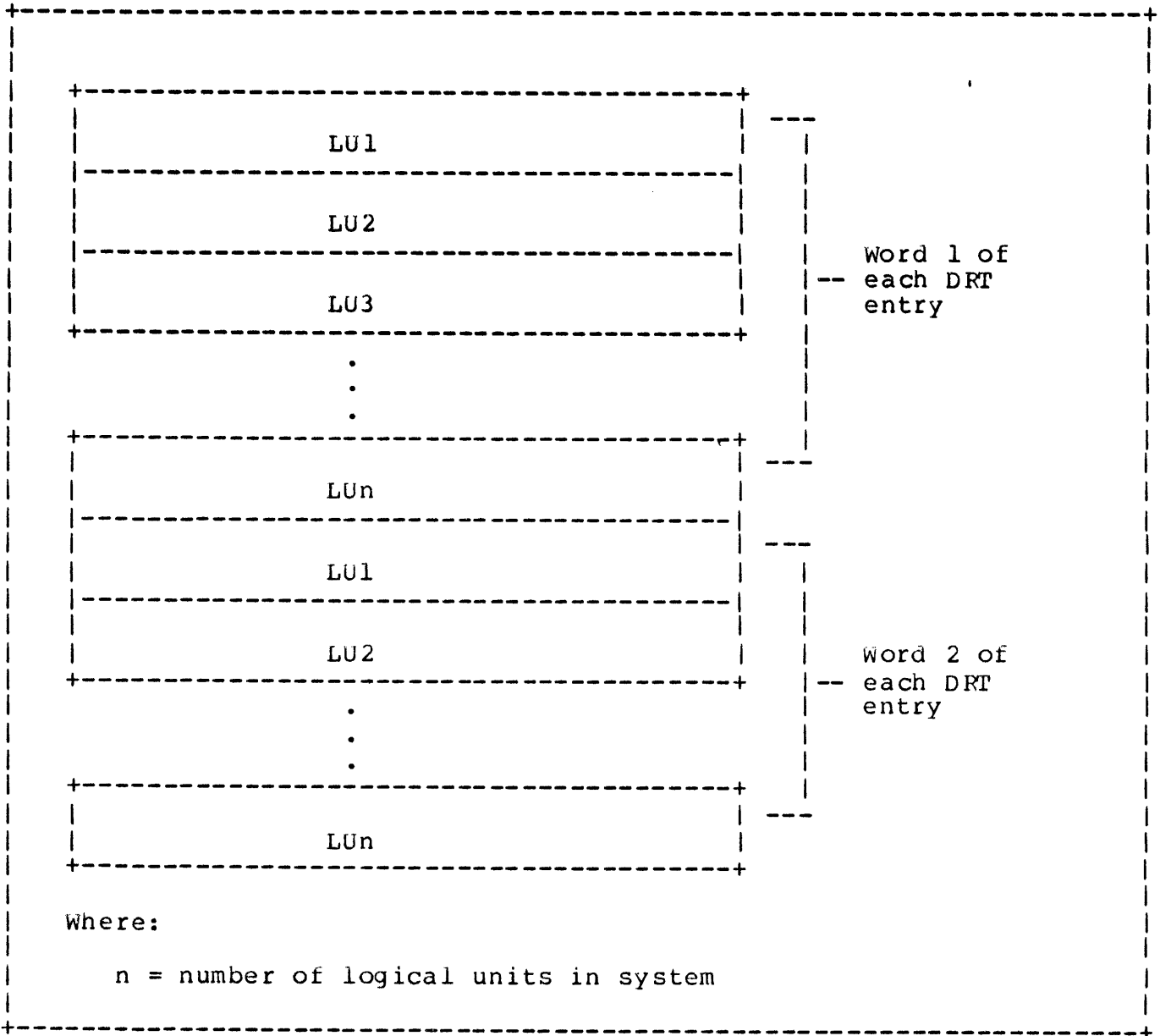


Figure 5-4. Device Reference Table

Note that there are separate tables for words 1 and 2, with the word 2 table being located in memory immediately following the word 1 table. The starting address and length of the word 1 table are recorded in the base page.



## 5-4. LOGICAL UNIT NUMBERS

Logical Unit numbers provide RTE users with the capability of logically addressing the physical devices defined by the Equipment Table. Logical Unit numbers are used by executing programs to specify on which device I/O transfers are to be carried out. In an I/O EXEC call, the program simply specifies an LU number and does not need to know which physical device or which I/O controller handles the transfer.

Although many devices such as line printers are addressed by a single LU number, others such as disc drives have subchannels, with each subchannel addressed by a different LU number.

If on-line changes to existing LU assignments become necessary or desirable, this can be achieved through use of the LU operator command. LU numbers are maintained by the Device Reference Table (see above).

Logical Unit numbers are decimal integers. The functions of Logical Units 0 through 6 are predefined in the RTE-IV system as follows:

- 0 -- bit bucket (null device; no entry in Device Reference Table)
- 1 -- system console
- 2 -- reserved for system (system disc subchannel)
- 3 -- reserved for system (auxiliary disc subchannel)
- 4 -- standard output device
- 5 -- standard input device
- 6 -- standard list device

Logical Unit 8 is recommended to be the magnetic tape device, if one is present on the system. Peripheral discs must be assigned logical units greater than 6. Additional logical units may be assigned for any function desired.

## INPUT/OUTPUT

### 5-5. INTERRUPT TABLE

The Interrupt Table contains an entry, established at system generation time, for each I/O select code in the computer. If the entry is equal to 0, the select code is undefined in the system. If an interrupt occurs on one of these select codes and is processed by the Central Interrupt Controller (CIC), RTE outputs the message

ILL INT xx

where xx is the octal I/O select code number. RTE-IV then clears the interrupt flag on the channel and returns to the point of interruption.

The ILL INT message is also issued if the driver completes and the system cannot find the processed I/O request queued to the EQT entry.

If the content of the entry is positive, the entry contains the address of the EQT entry for the I/O controller on the channel (refer to the EQT option for the Interrupt Table entry during system generation).

If the content of the entry is negative, the entry contains the negated ID segment address of a program to be scheduled. If the program is not dormant when an interrupt occurs on that I/O select code, the following message is output on the system console:

SC03 INT xxxxx

where xxxxx is the program name. The interrupt flag is then cleared for that channel and control is returned to the point of interruption. (Refer to the PRG option for the Interrupt Table entry in the RTE-IV On-Line Generator Reference Manual.)

## 5-6. SYSTEM BASE PAGE INTERRUPT LOCATIONS

When an interrupt is received, the computer transfers control to one of a group of memory locations, known as trap cells, in the system base page. The I/O select code of the interrupting controller determines the location of the transfer. For instance, interrupts from select code 12 cause a transfer to memory location 12; interrupts from select code 13 cause a transfer to location 13, et cetera. Memory locations from octal 4-77 comprise the entire set of interrupt trap cells, where

4 = powerfail

5 = memory protect/DMS/parity error

6 = DCPC Port 1

7 = DCPC Port 2

10-77 = I/O slots

Transferring control to an interrupt trap cell causes the instruction located there to be executed. For all devices operating under control of the Central Interrupt Controller (CIC), this instruction is a JSB LINK,I, where LINK contains the address of the entry point to CIC. This instruction is initially set up at generation time and is reset each time the system is rebooted. There are three different ways that interrupts are serviced, according to the contents of the trap cell and the Interrupt Table:

Generation Entry (examples) -----	Interrupt Table Contents -----	Trap Cell Contents -----
12,EQT,1	EQT entry address	JSB LINK,I
12,PRG,name	Negative ID segment address	JSB LINK,I
12,ENT,entry	0	JSB entry,I

JSB LINK,I trap cells are processed by CIC. JSB entry,I trap cells by-pass the Interrupt Table and CIC for time-critical events such as Power Fail and privileged interrupts.

## INPUT/OUTPUT

### 5-7. DRIVER MAPPING TABLE

Each EQT entry has an associated two-word Driver Mapping table entry that indicates whether the driver for that EQT entry is in the System Driver Area (SDA) or a driver partition; and whether or not the driver (if it is in SDA) performs its own mapping. If the driver is in a partition, the entry contains the physical starting page number of the partition. This page number is used to map the driver into the appropriate System Map or User Map.

The second word of each entry is set up when I/O is started on the corresponding driver. The sign bit of the second word indicates whether or not I/O is being performed for a memory resident program. The word is 0 for system I/O. The low 10 bits contain the page number of the user's physical base page if it is a partition resident program. This word is used to save time on setting up the proper map on processing interrupts.

The format of the Driver Map Table is illustrated in Figure 5-5.

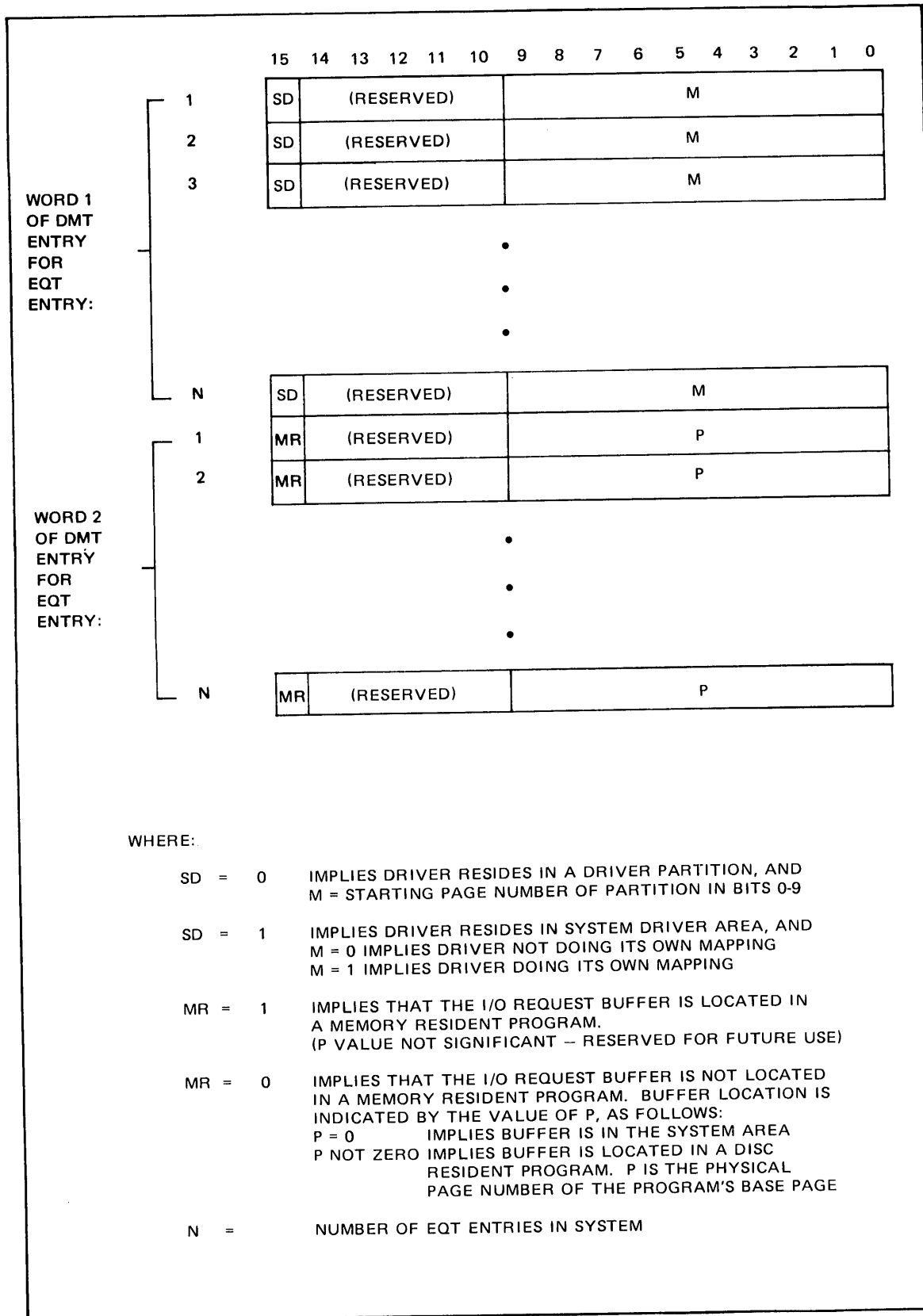


Figure 5-5. Driver Mapping Table

## INPUT/OUTPUT

### 5-8. I/O PROCESSOR GENERAL OPERATION

### 5-9. STANDARD I/O CALLS

A user program makes an EXEC call to initiate I/O transfers. If the device's controller is not buffered or the I/O transfer is for input, the calling user program is suspended until the transmission is completed (see Class I/O, Section IV for exceptions). The next lower priority program is allocated execution time during the suspension of a higher priority program.

An I/O request (i.e., READ, WRITE, CONTROL) is channeled to RTIOC by the executive request processor. After the necessary legality checks are made, the request is linked into the request list corresponding to the referenced I/O controller.

If the device's controller is available (i.e., no prior requests were pending), preparation is made to enter the driver's initiation section. The parameters from the request are set in the temporary storage area of the EQT entry.

The proper mapping registers are set up if the Driver Mapping Table indicates they are needed. The decision to choose the User Map or the System Map is decided by the type of I/O request. All system I/O, class I/O, and buffered user I/O requests require the use of the System Map.

Unbuffered user requests require the User Map. Note that in the case of a driver located in the System Driver Area making unbuffered requests, the program must be Type 2 or 3.

If the disc resident program's User Map needs to be modified to map in a partition resident driver, the User Map is saved in the program's physical base page. The second word of the driver's mapping table entry is modified to record the type of map needed and if it is a disc resident program's map the physical base page number is also kept. This second word is used to save time on setting up the map registers for a subsequent continuation interrupt. The initiation section initializes the device's controller and starts the data transfer or control function.

If the device's controller is busy upon return from the initiation section or else a required DCPC channel is not available, RTIOC returns to the scheduling module to execute the next lower-priority program.

If the device's controller (EQT entry) or the device (LU) is down, the calling program is automatically suspended in the general wait list (status=3). While in this list, the program is swappable. If any LU or EQT entry is set UP, the program is automatically rescheduled. Refer to the ST command in Section III for more information on the general wait list.

Interrupts from the device's controller cause the Central Interrupt Control (CIC) module of RTIOC to call the continuation/completion section of the driver. RTIOC sets up the correct map before entering the driver. This is done by checking the Driver Mapping Table entry associated with the EQT entry. If a User Map is being reset, its contents are obtained from the program's physical base page. At the end of the operation, the driver returns to CIC and consequently to RTIOC.

RTIOC causes the requesting program to be placed back into the schedule list and checks for an I/O stacked request. If there are no stacked requests, RTIOC exits to the dispatching module; otherwise, the initiation section is called to begin the next operation before returning.

#### 5-10. POWER FAIL

Power Fail is an optional hardware/software feature that saves all system status and context up to the point at which the computer signals a power failure. If generated into the system, the Power Fail routine performs the following steps:

1. When power fails, it saves all registers, stops DCPC transfers and saves maps. If not enough time was available, Power Fail issues a HLT 4.
2. When power comes on, it restarts the real-time clock, sets up a time-out entry (TO) back to its EQT entry, and then returns to the Power Fail interrupt location so that it can do more recovery type work after the power fail system and operating system are reenabled.
3. When the EQT entry times-out, the Power Fail routine checks EQT entry word 5, bits 14 and 15 of each I/O controller. The status of bits 14 and 15 will indicate whether the I/O controller is "down" or "busy." The routine also checks bit 13 of EQT entry word 4 (set by driver), which indicates if the driver is to process the power failure.
4. If the I/O controller was busy when the power failed and the power fail bit was set when power resumed, the Power Fail routine calls the driver. The proper map is set up, according to the Driver Mapping Table entry and the driver is entered at Ixnn with its EQT entry unchanged. If the power fail bit was not set, the Power Fail routine calls the IOC module to set the controller and all downed LU's "up", reinitializes the EQT entry, and enters the driver at Ixnn.

## INPUT/OUTPUT

To summarize, assuming the controller was reading or writing data when power failure occurred (and the driver is designed to handle power fail), the controller driver will perform the power fail recovery when power resumes. If the controller was busy when power failure occurred and the controller driver cannot handle power failure, the routine attempts to restart the I/O operation.

5. If the controller or device was down when the power failed and the power fail bit is set or not set, the system "ups" the controller (EQT entry) and associated LU's, resets the EQT entry and enters the driver at Ixnn when power resumes.
6. An HP-supplied program called AUTOR will be scheduled. AUTOR sends the time of power failure to all teletypes on the system (which reenables all terminals). AUTOR is written in FORTRAN, with the source program supplied to the user so that the program may easily be modified to meet on-site requirements.

### 5-11. I/O CONTROLLER TIME-OUT

Each I/O controller may have a time-out clock to prevent indefinite I/O suspension. Indefinite I/O suspension can occur when a program initiates I/O and the device's controller fails to return a flag (possible hardware malfunction or improper program encoding). Without the controller time-out, the program that made the I/O call would remain in I/O suspension indefinitely, awaiting the "operation done" indication from the device's controller.

For privileged drivers, the time-out parameter must be long enough to cover the period from I/O initiation to transfer completion.

EQT entry words 14 and 15 in the EQT entry for each I/O controller function as a controller time-out clock. EQT entry word 15 is the actual working clock. Before each I/O transfer is initiated, it is set to a value  $m$ , where  $m$  is a negative number of 10 ms time intervals stored in EQT entry word 14. If the controller does not interrupt within the required time interval, it is to be considered as having "timed out." The EQT 15 clock word for each controller can be individually set by three methods:

1. The system inserts the contents of EQT entry word 14 into EQT entry word 15 before a driver (initiation or completion section) is entered. EQT entry word 14 can be preset to  $m$  by entering (T=) at generation time.
2. By use of the TO operator command (see Section III).
3. By driver.



## 5-12. PRIVILEGED INTERRUPT PROCESSING

Privileged interrupt processing provides access to specific elements for more rapid operations than are possible in standard I/O processing. I/O transfers are performed directly rather than going through the Central Interrupt Control module and other standard system services.

Including a special I/O interface card is the means by which RTE allows a class of privileged interrupts to be processed independently of system operation. The presence and location of the special I/O card is determined at system generation time. Its actual hardware location is stored in the word DUMMY in the Base Page Communication Area (or, if the card is not preset, zero). See the RTE-IV On-Line Generator Reference Manual for the exact specification procedure.

The special I/O card physically separates the higher priority privileged interrupts from the regular system-controlled interrupts. When an interrupt occurs, the card has its flag set which enables the card to hold off non-privileged, lower priority interrupts. This means that the system does not operate with the interrupt system disabled, but in a hold-off state. Furthermore, the privileged interrupts are always enabled when RTE is running and can interrupt any process taking place. See the RTE Operating System Driver Writing Manual for further details on writing privileged drivers.

RTE uses the Dynamic Mapping System (DMS) of 21MX-series computers to address memory configurations larger than 32K words. The user can address up to 1024K words of physical memory using the DMS feature. This is accomplished by translating memory addresses through one of four "memory maps". A memory map is defined as 32 hardware registers that provide the interface between the 32K words of logical memory and physical memory. All memory map addressing is done internally by the system and is transparent to the user.

The following brief explanation of the addressing and mapping process provides a general understanding of the overall operation of the system; for a more detailed description of the Dynamic Mapping System, refer to the appropriate 21MX Series Computer Reference Manual.

#### 6-1. ADDRESSING

The basic addressing scheme of the computer uses a 15-bit number that describes a location in memory numbered 0 to 32767 (see Figure 6-1). The 32768 (32K word) locations are grouped into 32 pages, with each page containing 1024 (1K) words. DMS takes the 15-bit address and splits it into two parts. The upper five bits (bits 10-14), become the logical page number, an index pointing to one of the 32 registers within a memory map (only one of the four maps can be enabled at a time). The lower 10 bits point to a relative address (or offset) within the destination page and do not require translation.

When the address is converted, the index is used to determine which of the 32 registers of the currently enabled map has the 10-bit physical page address. This page address is then concatenated to the relative address to provide the ultimate 20-bit address in physical memory.

# MEMORY MANAGEMENT

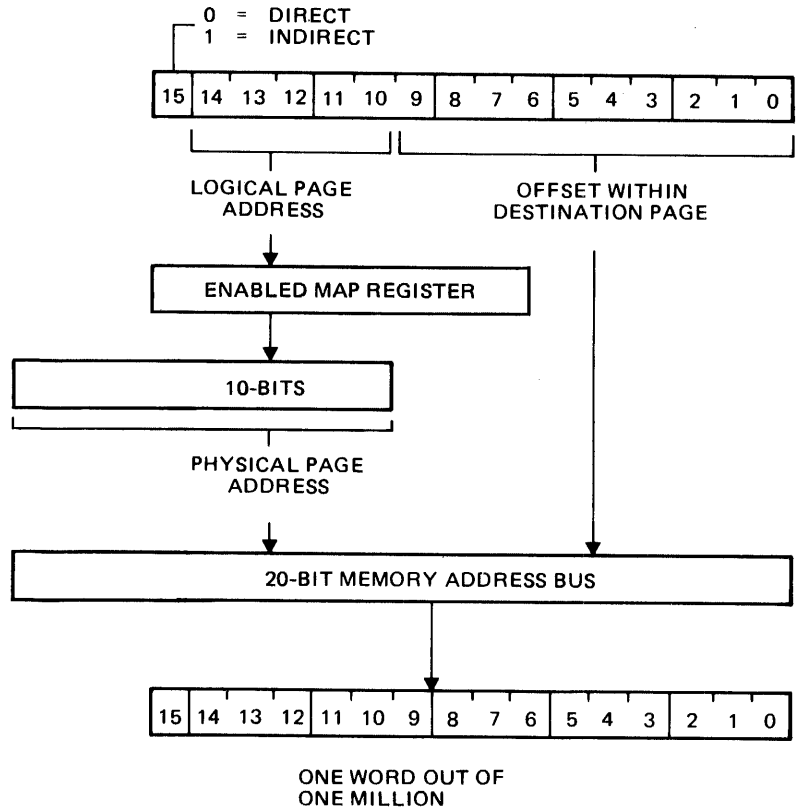


Figure 6-1. RTE-IV Address Scheme

## 6-2. MEMORY MAPS

There are four memory maps managed by the system: the User Map for describing current user programs, the System Map for describing the system and System Available Memory (SAM), and two Dual Channel Port Controller (DCPC) maps called Port A Map and Port B Map for defining the memory space of the DCPC transfer.

At any one instant, only one memory map is enabled. This map defines the 32K words of logical address space currently being used. Either the System or User Map will be enabled. A DCPC transfer is handled under the appropriate Port Map, and once initiated, is essentially transparent to the user.

**SYSTEM MAP.** This map is automatically enabled whenever an interrupt occurs and is loaded by the system during system initialization. It is changed only to map different driver partitions. It describes the logical address space used for the operating system and its base page, COMMON, Subsystem Global Area, System Driver Area, Table Areas I and II, driver partition, and System Available Memory.

**USER MAP.** Associated with each disc resident program is a unique set of pages that describe the logical address space for the program.

These pages define the memory occupied by Table Area I, driver partition, optional Table Area II and optional System Driver Area, COMMON (if the program uses it), the program's base page, and the program.

All memory resident programs use a common set of pages that define the memory ~~occupied by Table Area I, driver partition, COMMON, optional Table Area II and System Driver Area, base page, the memory resident library, and the memory resident program area.~~

Each time a new memory or disc resident program is dispatched, the system reloads the User Map with the appropriate set of pages. The User Map, therefore, provides the interface between logical memory and physical memory.

PORT A MAP. DCPC transfers are a software assignable direct data path between memory and a high speed peripheral device. This function is provided by the 21MX series Dual Channel Port Controller (DCPC). There are two DCPC channels, each of which may be assigned to operate with an I/O device. The Port A Map is automatically enabled when a transfer on DCPC channel 1 takes place.

The Port Map must be reloaded by the system each time the channel is assigned for a new I/O call so that the data buffer is mapped in. Having separate maps for DCPC facilitates multiprogramming, since DCPC may be accessing one program's buffer while another program (in a different area of physical memory) is using the CPU under the User Map (i.e., when one program is using DCPC, another program can be executing).

PORT B MAP. This map is handled in the same way the Port A Map is handled except that it applies to DCPC channel 2.

### 6-3. PHYSICAL MEMORY

At generation time, the user plans the physical memory allocations as illustrated in Figure 6-2 and then loads the system components and drivers for the most efficient configuration. The user determines the size of System Available Memory, (SAM), the number and size of each partition, the size of COMMON, and the size and composition of the resident library and memory resident program area.

## MEMORY MANAGEMENT

The areas shown in Figure 6-2 are used as follows:

- \* System Base Page - contains system communication area and is used by the system to define request parameters, I/O tables, scheduling lists, pointers, operating parameters, memory bounds, etc. System links and trap cells are also located on the system base page.

The base page links for memory resident library and memory resident programs are only in the memory resident base page and are not accessible by disc resident programs. The Table Area, SSGA and driver links, and the system communication area are accessible to all programs. Partition base pages, used for disc resident program links, are described below with partitions. For all practical purposes, the memory resident programs are in a single partition separate (protected) from all other partitions.

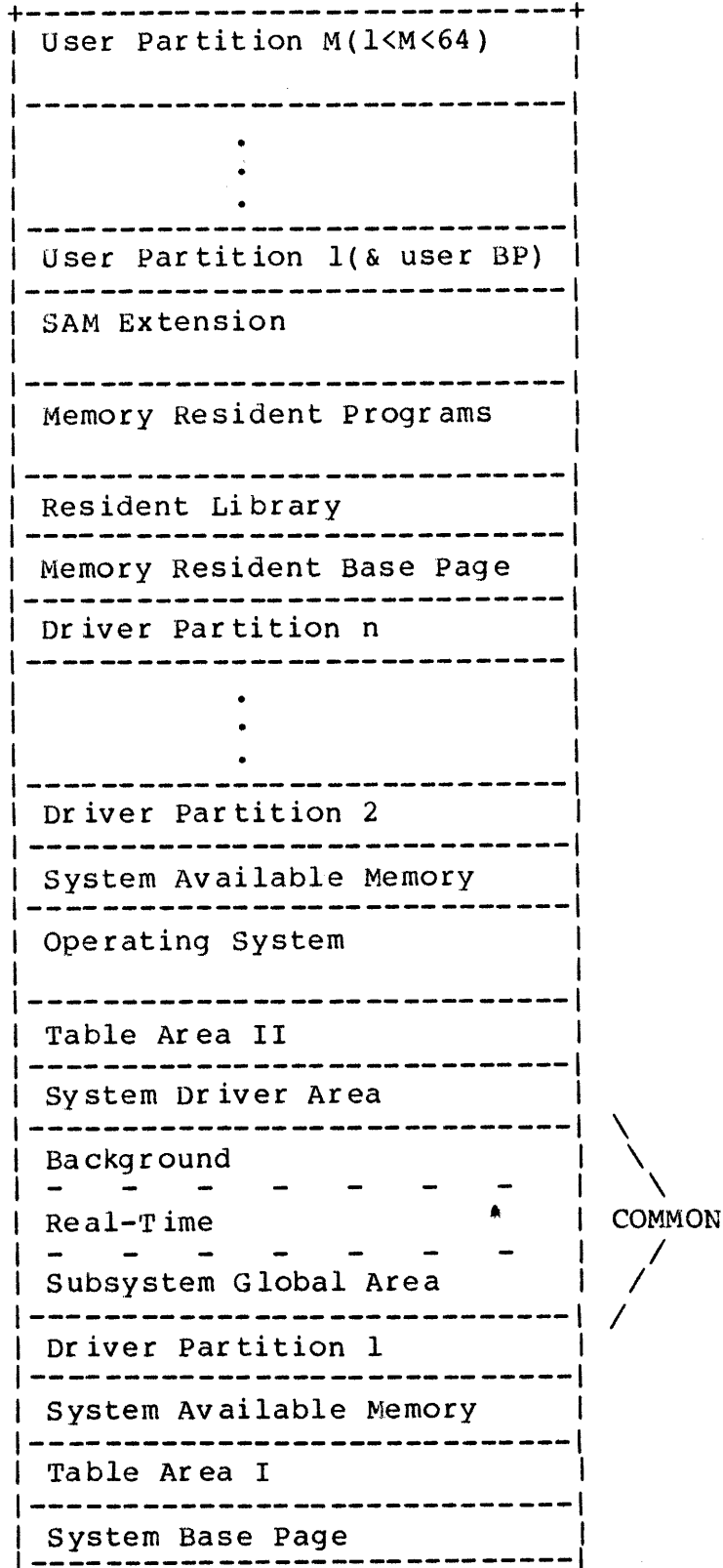


Figure 6-2. Physical Memory Allocations

## MEMORY MANAGEMENT

- \* Table Area I - Contains the Equipment Table entries, Driver Mapping Table, Device Reference Table, Interrupt Table, the Disc Track Map Table, some system entry points and all Type 15 modules.
- \* Driver Partition - An area set aside at generation time containing one or more drivers. All driver partitions are the same length, and only one is included in a 32K-word address space at any one point in time. The minimum partition size is two pages but may be increased.
- \* System Driver Area - An area for privileged drivers, large drivers, or drivers that do their own mapping. The drivers that go into this area are specified during the EQT definition phase of system generation. The System Driver Area (SDA) is included in the logical address space of both the system and Type 2 and 3 programs. It is included in the memory resident program area (if requested) at generation time.
- \* System - Contains the absolute code of the Type 0 system modules (e.g., RTIOC, SCHED, EXEC).
- \* Memory Resident Library - Contains the reentrant or privileged library routines (Type 6) that are used by the memory resident programs, or which are force loaded at generation time (Type 14). It is accessible only by memory resident programs. All routines loaded into the resident library also go into the relocatable library for appending to disc resident programs that require them.
- \* COMMON - This area is divided into three subareas: The Subsystem Global Area (SSGA), the Real-time COMMON area, and the Background COMMON area. SSGA is used by some Hewlett-Packard software subsystems for buffering and communications. The Real-time and Background sub-areas (system COMMON) are reserved for user-written programs that declare COMMON. All programs relocated during generation time that declare COMMON will reference this system COMMON. Programs relocated on-line with LOADR may choose to reference system COMMON or use local COMMON.
- \* Memory Resident Programs - This area contains all Type 1 programs that were relocated during generation.
- \* Table Area II - Contains the Memory Protect Fence Table, ID segments, Keyword Table, ID Segment Extensions, Class Table, RN Table, LU Switch Table, Memory Resident Map, and a number of entry points for system pointers. This area has entry points that are created by the generator and some that are defined by Type 13 modules.
- \* System Available Memory - This is a temporary storage area used by the system for buffered and Class I/O reentrant I/O (refer to Section IV), and parameter string passing.

- \* Partition - This is an area set aside by the user for a disc resident program to run. Each partition has its own base page that describes the linkages for the program running in the partition. Up to 64 partitions are allowed, within the constraints of available physical memory.

All of the above areas are established during system generation.

#### 6-4. LOGICAL MEMORY

Logical memory is the 32K word (maximum) address space described by the currently enabled memory map. If the System Map is enabled, logical memory includes the operating system and its base page, Table Areas I and II, System Driver Area, driver partition and System Available Memory. It also includes COMMON and Subsystem Global Area.

If the User Map is enabled for a disc resident program, logical memory includes Table Area I, a driver partition, optional Table Area II, optional System Driver Area, COMMON (if used), and the currently executing program and its base page.

The logical memory of a memory resident program includes the memory resident program area and base page, Table Area I, a driver partition, COMMON, optional Table Area II and System Driver Area.

Port Maps are used DCPC transfers and describe the logical memory containing a data buffer. A Port map will be the same as either the System Map or the map of the program being serviced, depending on type of I/O call.

Figure 6-3 shows the four configurations of the 32K word logical address space. The first configuration illustrates how this space appears under control of the System Map. Note that there is always a total of 32 pages to be divided up; however, the particular boundaries shown for the various parts are examples only, and a user's system could be larger or smaller.

The second configuration illustrates how the logical address space appears under control of the User Map when a memory resident program is executing.

The third configuration illustrates how the logical address space appears under control of the User Map when either an RT or Type 3 (BG) disc resident program is executing.

The fourth configuration illustrates how the logical address space appears under control of the User Map when a Type 4 (BG) disc resident programs is executing.

Many programs will not require a full 32K of space, and unneeded pages will be READ/WRITE protected as illustrated in the User Map given in Figure 6-3, configuration 3.



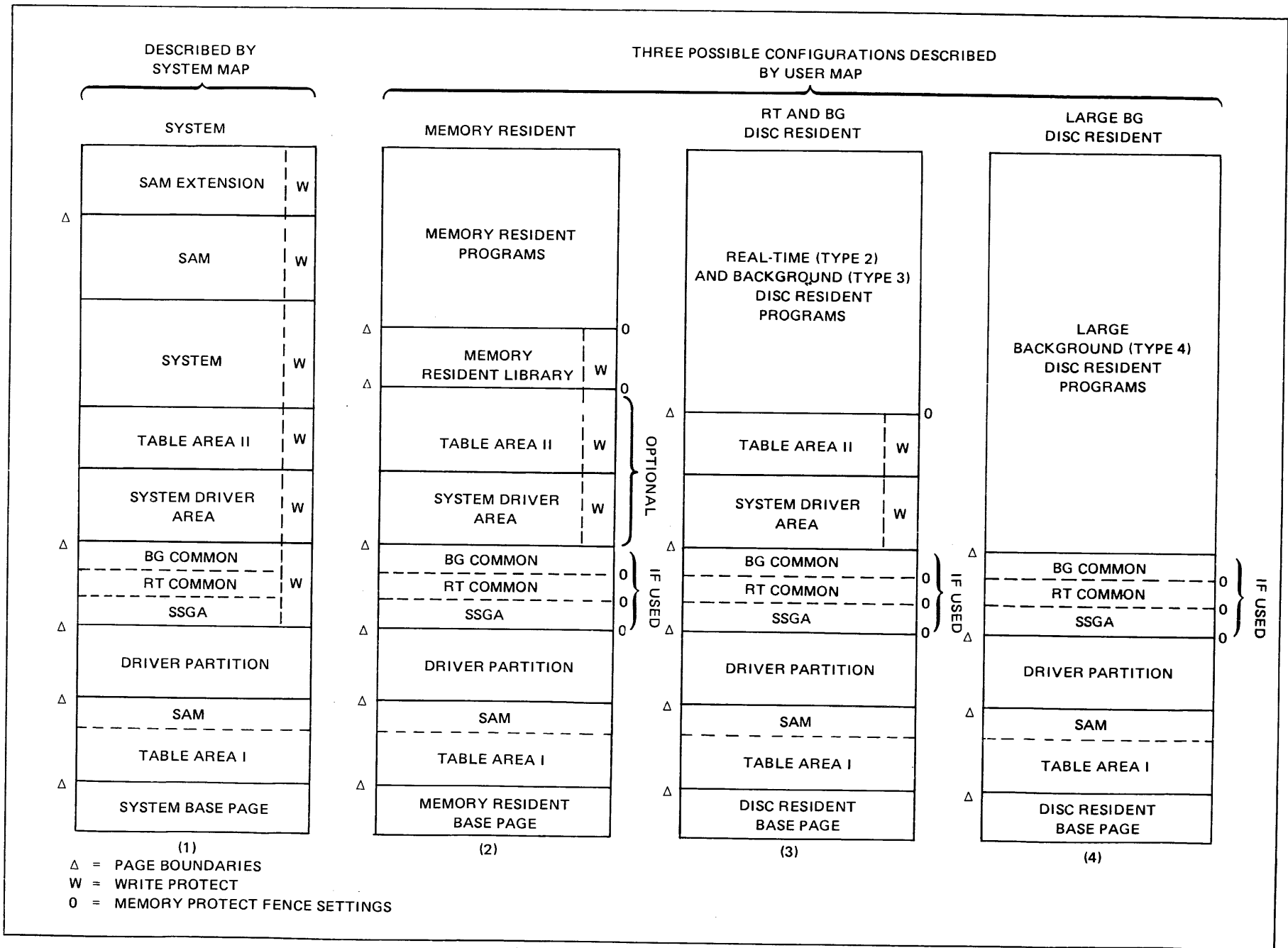


Figure 6-3. RTE-IV 32K WORD LOGICAL MEMORY CONFIGURATIONS

6-5. BASE PAGE

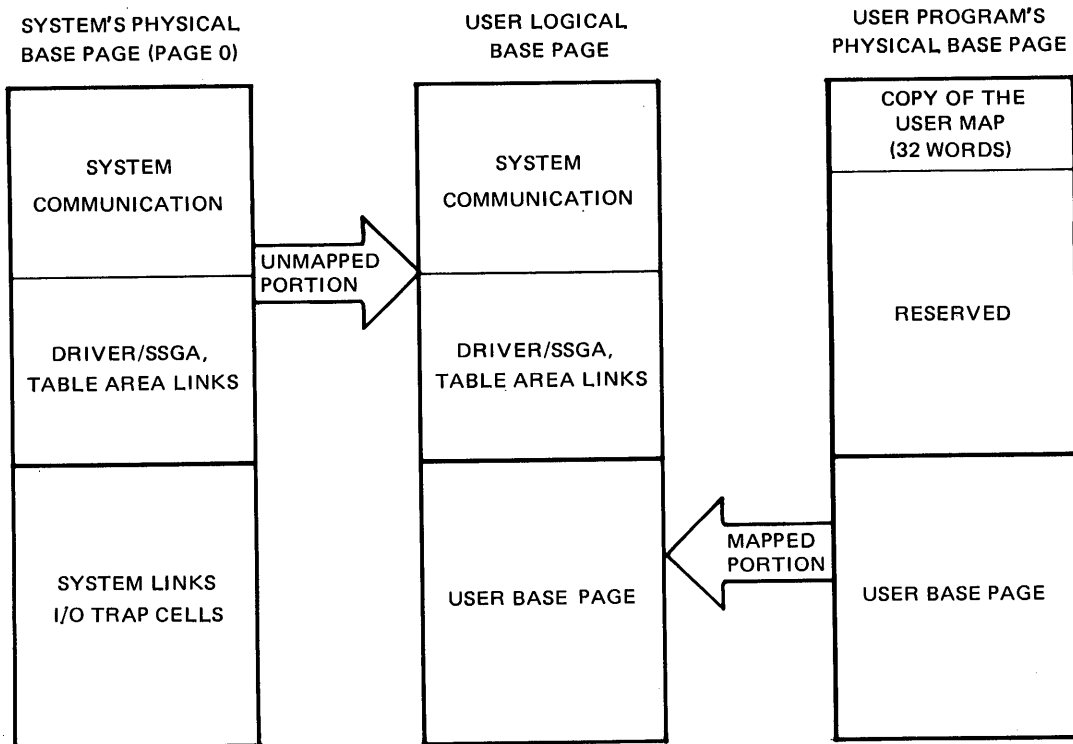
The system area, memory resident program area and each disc resident program have their own logical base pages, as follows:

- a. The system base page contains the system communication area, system links, driver links, SSGA links, table area links and trap cells for interrupt processing.
- b. The disc resident program base page contains the system communication area, driver links, SSGA links, table area links and disc resident program links.
- c. The memory resident base page has the memory resident program links, resident library links, System Communication area, table area links, SSGA links, and driver links.

The Base Page Communications area (see Appendix B), driver links, SSGA and table area links located in physical page 0 will be common to all base pages. Base page structures are illustrated in in Figure 6-4.

The Base Page Fence (refer to the 21MX and 21MX E-series Operating and Reference Manual) is automatically set by the system for all user base pages so that the bottom portion of the base page will contain the user program links.

Figure 6-4. Base Page Structure



## MEMORY MANAGEMENT

### 6-6. COMMON AREAS

The real-time and background COMMON, along with Subsystem Global Area occupy a contiguous area in memory and are treated as a single group for mapping purposes (refer to Figure 6-2). The use of COMMON is optional on a program basis; that is, any program may use real-time COMMON, background COMMON or no COMMON. If the program declares COMMON and the user chooses not to use local COMMON, both COMMON areas and the Subsystem Global Area will be included in the User Map. If the Type 4 program does not use COMMON it is not included in the User Map, thereby possibly (if SSGA, COMMON is not empty) providing the user a larger program area in the 32K of logical address space.

REAL-TIME AND BACKGROUND COMMON. If a program declares at least one word of COMMON, the use of real-time or background COMMON is selected by program type (at generation) or parameters with the on-line loader. Program types are summarized in Appendix E. Note that the memory protect fence protects areas below the selected COMMON.

These system COMMON areas are not to be confused with the local COMMON area that may be specified for programs loaded on-line. The system COMMON areas are sharable by programs operating in different partitions, whereas the local COMMON area is appended to the program (i.e., it will be in its partition) and is accessible only to that program, its subroutines and segments.

SUBSYSTEM GLOBAL AREA. The Subsystem Global Area consists of all Type 30 modules input to the generator. Accessed by entry point (using EXT statements) rather than COMMON declarations, SSGA provides multiple communication and buffer areas for Hewlett-Packard subsystems. SSGA access is authorized by program type at generation or through special parameters during on-line loading. Programs authorized for SSGA access have the COMMON area included in their maps and have the memory protect fence set below SSGA.

### 6-7. MEMORY PROTECTION

Memory protection between disc resident program partitions and between disc and memory resident programs is provided by the Dynamic Mapping System. Protection between the program and the operating system is handled by memory protect. A program cannot access a page not included in its logical memory, either directly or through a DMA transfer. Since many programs do not use all of the possible 32K word logical area, unused logical pages above the program are READ/WRITE protected; it is possible for a user to read from system logical memory via cross-map reads but the system is write protected.

A different form of protection is required for the driver partition, Table Area I, and (optionally) System Driver Area, Table Area II, and COMMON. The memory protect fence provides this protection by preventing stores and jumps to locations below a specified address. All possible fence positions are illustrated in Figure 6-3.

The memory protect fence applies to the logical address space where addresses are compared to the fence before translation. If a disc resident program does not use any of the COMMON areas, the memory protect fence is set at the bottom of the program area. Similarly, for a memory resident program not using COMMON, the memory protect fence is set at the base of the entire memory resident area.

For programs using COMMON, all of logical memory including COMMON is mapped and the fence is set at one of three possible locations, depending on the portion of COMMON being used. A hierarchy of protection is thereby established within COMMON due to their physical locations. Background COMMON is the least protected (any program using any common can modify it) and SSGA is the most protected (only programs authorized for SSGA access can modify it). Figure 6-5 expands the COMMON area and shows these three fence settings as a, b, and c respectively.

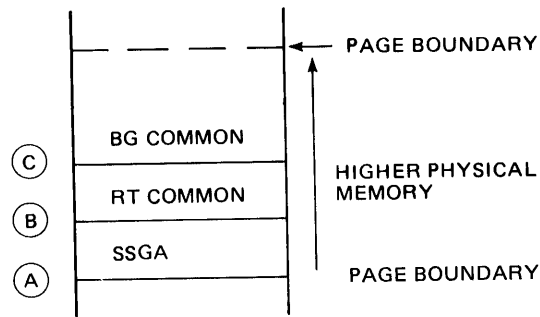


Figure 6-5. Memory Protect Fence Locations for Programs using COMMON

## MEMORY MANAGEMENT

### 6-8. PARTITIONS

Partitions are blocks of physical memory that are reserved for disc resident programs. Program partitions are defined during system generation and ordinarily are not changed. However the partitions may be redefined during the reconfiguration process at system boot-up (see Section XII).

The number of partitions depends on the amount of available physical memory. Partition types can be specified as a mixture of real-time and background, all real-time, or all background. A program can be assigned at load time to run in any partition large enough to accommodate it. Several programs can be assigned to the same partition, but only one program can run in that partition at a time. If a program is not assigned to a partition, then by default, real-time programs will run in real-time partitions, background programs in background partitions, and EMA programs will run in Mother partitions. If only one type of partition is defined, all programs will run in that type partition.

### 6-9. PARTITION LISTS

The system generator links all partitions into one of three free lists: BG, RT or mother partitions. During system initialization, if one of the free lists is empty, it is substituted by one of the other non-empty lists. For example, if no RT partitions were generated into a system, RT programs will be dispatched in BG partitions by default.

### 6-10. PARTITION ASSIGNMENT AND RESERVATION

Disc resident programs may be assigned to specific partitions during system generation, memory reconfiguration at system boot-up, or during on-line program relocation. A program may be unassigned or reassigned via an AS operator command.

A program assigned to a specific partition may only be dispatched to that partition. Program contention for a partition may be minimized by careful assignment of programs to partitions, especially if the partitions are reserved. A reserved partition may be used only to dispatch programs that are assigned to the partition. Programs not assigned to the reserved partition will not be able to use it as a default, even if no other partitions are available. A partition's reserved status may be removed by the UR operator command.

A disc resident program may be assigned to any partition large enough to accommodate it, regardless of type. For example, an RT program may be assigned to a BG partition even though both RT and BG partitions are available. Although this type of assignment is not recommended because of potential partition contention, it may be necessary when there are no partitions of sufficient size within the same partition type as the program.

## 6-11. MOTHER PARTITIONS

Mother partitions are large partitions that may be defined for executing large programs or EMA programs. When a mother partition is not in use, the memory may be used by programs executing in the subpartitions chained to the mother (see "Subpartitions" below). EMA programs that are not assigned to a partition use the largest mother partition by default.

When an EMA program needs to run in a mother partition or when an RT or BG program is assigned to a mother partition, more handling is involved than is the case with RT or BG partitions. If a mother partition is available in the free list, each subpartition is checked. If all subpartitions are either free or occupied by swappable programs, the subpartitions are marked as being used for a mother partition and all the programs in the subpartitions are swapped out. The subpartitions are then removed from all partition free lists. Note that the swapped-out programs may go back into any other partition large enough to accept them.

It is now apparent that when a mother partition is required and its subpartitions are in use, there may be a delay before the program can be dispatched in the mother partition. A subpartition cannot be made available by swapping if any one of the subpartitions has a memory-locked program, contains a program that is performing I/O in its own area, or contains a scheduled program of higher priority. There may be additional delay when the mother partition is checked (if not assigned to a specific one) or until the program in the subpartition becomes swappable.

If a mother partition is needed to dispatch a program and the partition is already allocated, the current occupant must be swapped out if the occupant's priority and status permit it. If the program to be swapped out is an EMA program, the program's code and EMA data must both be swapped. The EMA area is swapped out in large blocks equal in size to the maximum logical address space in the User Map (up to a maximum of 28K words). Each block is mapped and written to the swap tracks on the disc until all of the EMA area is swapped. Because of the many disc accesses that may be needed to swap out an EMA program, caution should be exercised when assigning ANY program to a mother partition.

## 6-12. SUBPARTITIONS

Subpartitions are not available for dispatching programs when the mother partition is in use (chain mode is in effect) by an active program. When a program in a mother partition terminates normally or is aborted, the subpartitions are released from chain mode and again become available. The mother partition occupant is swapped only under the following conditions:

## MEMORY MANAGEMENT

1. The occupant is swappable and another program needs the same mother partition.
2. The occupant is dormant (terminated with the save-resources option, operator-suspended or serially reusable), and a subpartition is needed for another program.
3. A higher-priority program is assigned to a subpartition and the mother partition occupant is in a swappable state.

when an RT or BG program is scheduled and is not assigned to a partition, a search is made for a partition of the same type that is large enough to accommodate the program. If none can be found in the free list, dormant list, nor in the allocated list (or it contains non-swappable programs), then the dormant mother partition list will be searched for one with a subpartition of the correct type and size. If a suitable subpartition can be found, the dormant program in the mother partition will be swapped out.

### 6-13. EXTENDED MEMORY AREA

The Extended Memory Area (EMA) is a large area of memory within a partition, limited only by the size of the physical memory. An EMA can extend well beyond a program's maximum logical addressable space. A section of the EMA must be included within the program's logical address space before data within that section can be addressed. Because an EMA area is in a program's partition, it is not accessible by other programs (EMA is not shared between programs). The maximum number of pages of the EMA that can be included in the logical address space is called the mapping segment (MSEG).

The philosophy behind the mapping segment function is quite similar to page faulting in a virtual memory system. If an EMA element needs to be accessed and is not within the currently mapped mapping segment, a fault occurs and the appropriate segment of the EMA containing the element is mapped into the program's logical address space. This mapping is very fast since no disc swaps are required. The entire EMA is divided into sections of the length of MSEG. They are numbered sequentially starting from 0. Mapping segments are then referred to by using these mapping segment numbers. When a program is first dispatched, none of the EMA is mapped in the user's logical address space until a call is made to .EMAP, .EMIO or MMAP.

System library routines .EMAP and .EMIO can be used to determine the location of the element within the EMA to be accessed and to map the appropriate pages.

The .EMAP routine is used to resolve the address of an element in an array. .EMAP insures that the referenced element is mapped into the current logical address space and returns its logical address.

The .EMIO routine is used to access a buffer within the EMA and also ensure that the entire buffer will be included in the logical address at one time. This buffer must be of the same length or smaller than the mapping segment size. The EMAST routine in the system library may be used to determine the standard MSEG size and EMA size for default EMA.

.EMIO checks to see if the upper and lower bounds of the buffer are completely included within a standard mapping segment. If so, .EMIO maps the appropriate MSEG into the program's logical address space. If the bounds of this buffer do not fit completely within a standard mapping segment, .EMIO will then map in the necessary pages to include the entire buffer. A flag is set to indicate that a standard mapping segment is not in the current MSEG.

The MMAP routine, with the help of EMAST, can be used independently to do MSEG mapping. This may be needed if the array handling procedure for a given application differs from the array handling tools provided by .EMAP and .EMIO. (See the .EMAP, .EMIO, MMAP and EMAST subroutine description at the end of this section for more detailed information.)

Figure 6-6 illustrates the structure of EMA's and MSEG's.



# MEMORY MANAGEMENT

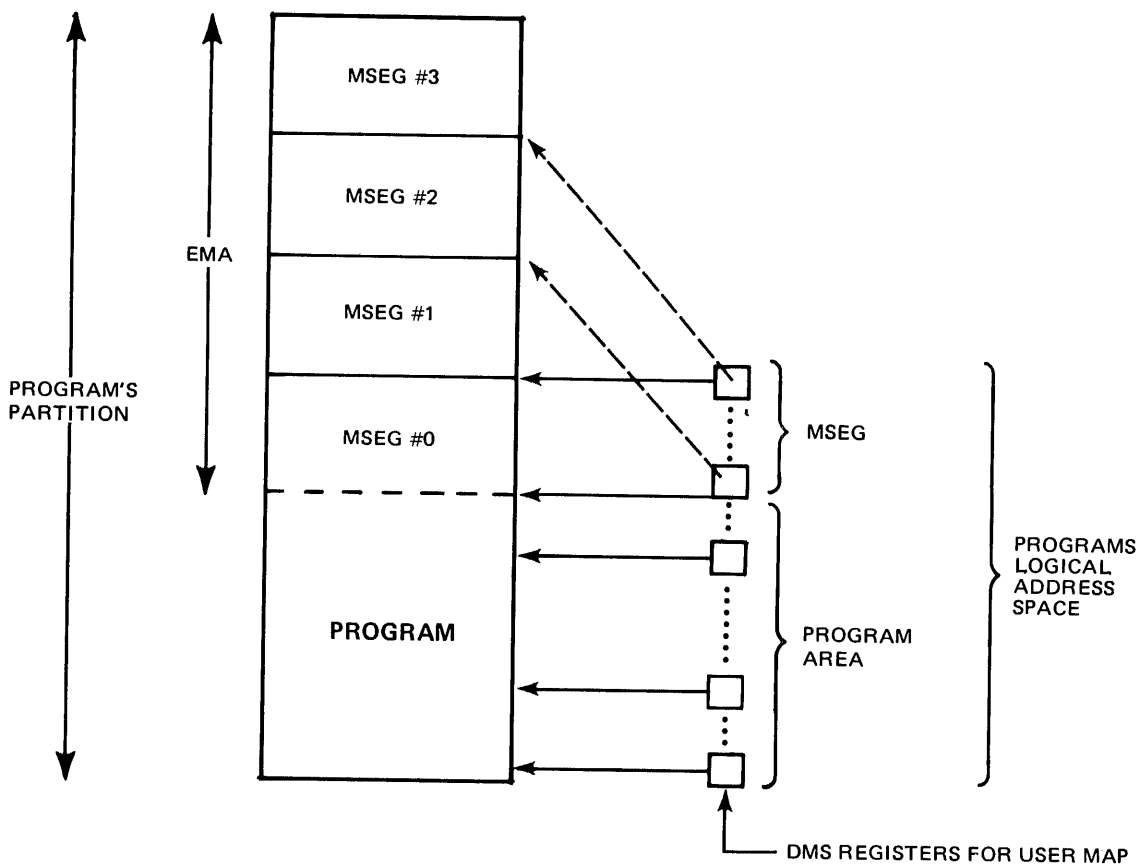


Figure 6-6. EMA and MSEG Structure

One extra page above the MSEG size is always mapped. This allows for overflow of elements containing more than one word per element, and for overflow of records for the formatter beyond the last page of the MSEG.

Only one extended memory area is allowed to be defined per program. An EMA is declared in an Assembly Language program by using the pseudo instruction:

```
label EMA m1,m2
```

where label is the EMA label and must be defined, m1 is the EMA size in pages, and m2 is the mapping segment size in pages. The EMA size can vary from 0 to 1023 pages. The MSEG size must be less than 32 pages. The default case on either EMA size, MSEG size or both, can be taken by specifying 0 as their values. If a default is taken on the MSEG size, its size is determined at load time as the program's maximum logical address space - the program size - 1. The EMA size is determined at the time of the first dispatch as the program's partitions size minus program size. EMA or MSEG size can be modified on-line only if the default was taken.

An EMA may be further subdivided into more than one data array. This is accomplished through use of optional offset parameters supplied in assembly language programs to the .EMAP and .EMIO routines. The offset is defined as the number of words from the start of the Extended Memory Area to the start of the particular array, and consists of a positive value that is 20 bits wide and is contained in two successive memory locations. The general memory structure for multiple data arrays is illustrated in Figure 6-7.

MEMORY MANAGEMENT

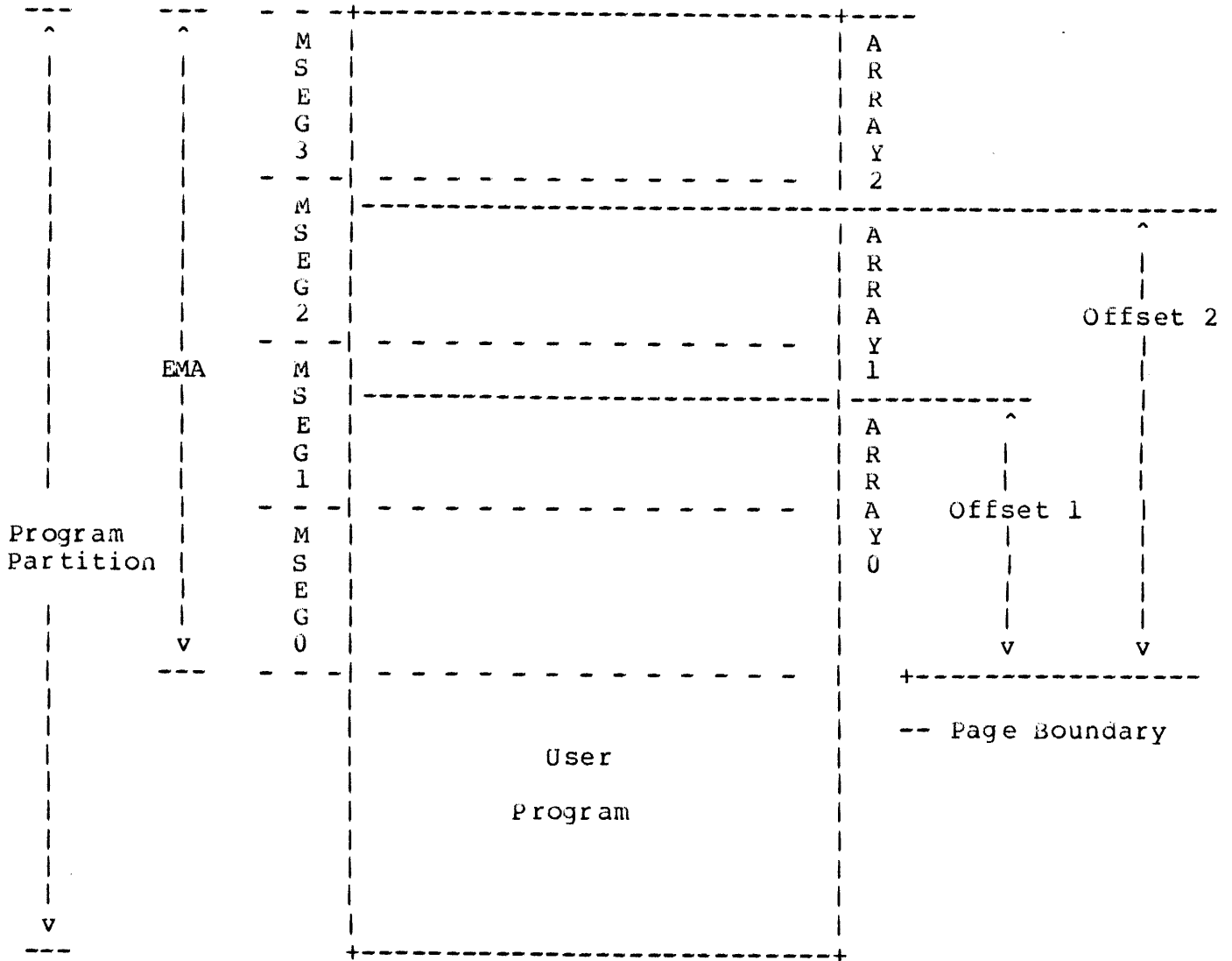


Figure 6-7. Multiple Data Arrays Organization

Locations within an EMA cannot be accessed using the EMA label with an offset, nor can EMA labels be referenced indirectly. External routines and segments can use EMA by declaring EMA as an external. For further information on using EMA as a pseudo-instruction, see the RTE IV Assembler Reference Manual.

Since EMA's can extend well beyond a program's 32K logical address space, they should be managed by defining several dimensions over them. The .EMAP or .EMIO routines can then be used to resolve the address of a specified element by using subscripts for each dimension, thus making the array addressing and mapping procedures transparent to the user.

Standard FORTRAN I/O and array accesses using subscripts are handled without any special user action. In FORTRAN, EMA's are used like any other array. Refer to the RTE FORTRAN IV Reference Manual for further information.

A segmented program may use EMA. This allows many separate operations to be performed on the same EMA; e.g., one segment reads the data, a second processes the data, and a third saves the results.

#### 6-14. MEMORY MANAGEMENT SUBROUTINES

Four subroutines implement the Extended Memory Area (EMA) capability in the RTE-IV Operating System. These are: .EMAP, .EMIO, MMAP, and EMAST. Although the software versions of these subroutines are actually part of the system library described in Section X of this manual, they are described here because they are an integral part of memory management.

Firmware versions of .EMAP, .EMIO, and MMAP exist for use on the 21MX-E series computer. The firmware version of .EMAP operates slightly differently than the software version, as described in the following discussion of .EMAP.

#### 6-15. .EMAP SUBROUTINE (Resolves Array Element Addresses)

The .EMAP subroutine resolves an address for an element in both EMA and non-EMA arrays. .EMAP returns the address of the referenced element in the current logical address space.

The software version of .EMAP calls on MMAP (if necessary) to map the appropriate mapping segment into the logical address space of the user program. The firmware version of .EMAP always maps two pages into the logical address space of the program, the first of which contains the referenced element.

-----+  
 | CAUTION |  
 |-----+  
 |

| The firmware version of .EMAP maps in the page containing the |  
 | element and the following page (if the following page is in the EMA |  
 | area). Therefore, a call to the firmware version of .EMAP will not |  
 | ensure that an entire MSEG is mapped. .EMIO can be used to ensure |  
 | this if necessary. |  
 |-----+  
 +-----

## MEMORY MANAGEMENT

The calling sequence is:

```
EXT .EMAP
JSB .EMAP
DEF RTN
DEF ARRAY          address of the start of the array
DEF TABLE        address of table containing array parameters
DEF An            address of nth subscript value
DEF An-1          address of (n-1) subscript value
.
.
.
DEF A2            address of 2nd subscript value
DEF A1            address of 1st subscript value
RTN error return
      normal return
```

**ERROR RETURN** On an error return, the A-register equals 15 (ASCII) and and the B-register equals EM (ASCII). If the relocatable library subroutine ERR0 is called to handle the error, the following message will be displayed on the console:

```
name 15-EM @ address
```

where name is the name of the program executing when the error occurred, and address is the address from which ERR0 was called.

.EMAP makes an error return under any of the following conditions:

- \* one of the subscript values is less than the lower bound of its dimension.
- \* the size of a dimension  $d(i)$  is negative.
- \* the number of words per element is specified as negative.
- \* the double precision offset is specified as negative.
- \* the number of dimensions is specified as negative.
- \* the element address for an EMA variable does not fall within the Extended Memory Area bounds.
- \* for a non-EMA array, the displacement is larger than 32767 words.

**NORMAL RETURN** On a normal return, the B-register contains the logical address of the element referenced. The A-register is meaningless.

ARRAY is the starting address of the array in which the element address is to be resolved. If EMA is declared in the calling program and the element address specified is greater than or equal to the logical start address of EMA, the array is assumed to be an EMA array. In this case, the start address actually used by .EMAP is the logical start address of EMA.

TABLE is a table of array parameters containing the number of dimensions in the array; the negative of the lower bounds for every dimension; the number of elements in every dimension (upper bound-lower bound + 1); and the number of words per element.

For EMA arrays only, a two-word offset value is required at the end of the table. The use of this offset enables several arrays to be defined in the same EMA by allowing the array origin to be higher than the logical start of the EMA. The offset is a double precision integer value with the low 16 bits (bits 15-0) in offset word 1 and the high 16 bits (bits 31-16) in word 2. This value must be positive.

The lower bound must be between -32767 and +32767.

The number of words per element must be between 1 and 1024.

The content and structure of TABLE is as follows:

```

Number of Dimensions
-L(n)
d(n-1)
-L(n-1)
d(n-2)
.
.
.
-L(2)
d(1)
-L(1)
number of words per element
offset word 1 (bits 15-0)          (used for EMA only)
offset word 2 (bits 31-16)       (used for EMA only)

```

where:

L(i) is the lower bound of the ith dimension.

d(i) is the number of elements in the ith dimension.

The .EMAP subroutine assumes the array is stored in column-major order (the left subscript varies the quickest).

## MEMORY MANAGEMENT

### 6-16. .EMIO SUBROUTINE (EMA I/O)

.EMIO is a subroutine used only in an EMA environment to ensure that a buffer to be accessed is entirely within the logical address space of the program. It will call MMAP (if appropriate) to alter the logical address space to contain the buffer, or if this is impossible it will return with an error.

.EMIO first checks whether the buffer fits in a standard mapping segment. If so, the standard mapping segment is mapped into the logical address space and .EMIO returns the logical address of the start of the buffer. If the buffer does not fall within a standard mapping segment, then .EMIO alters the mapping segment boundaries to contain the buffer.

The number of pages mapped in this special mapping segment is normally equal to the number of pages in the standard mapping segment. When this mapping segment starts within an MSEG size from the end of the EMA, all those pages up to the end of the EMA are mapped. The rest of the pages are read-write protected.

The buffer length plus the offset between the start of the buffer and its page boundary must be less than or equal to the mapping segment size. To ensure this, it is recommended that the buffer length be less than or equal to (MSEG size - 1) pages.

.EMIO maps the special mapping segment if necessary and returns with the logical address of the start of the buffer.

The calling sequence is:

```
EXT .EMIO
JSB .EMIO
DEF RTN          address for error-return
DEF BUFL        number of words in the buffer
DEF TABLE      table containing array parameters
DEF An          subscript value for nth dimension
DEF An-1        subscript value for (n-1)st dimension
.
.
.
DEF A2          subscript value for 2nd dimension
DEF A1          subscript value for 1st dimension
RTN error return
   normal return
```

where:

TABLE is as defined in .EMAP description

**ERROR RETURN** .EMIO makes an error return at location RTN with the A-register containing 16 (ASCII) and the B-register containing EM (ASCII). If the relocatable subroutine ERRO is called to handle the error, the following message is displayed on the console:

name 16-EM @ address

where name is the name of the program and address is the location from which ERRO was called.

.EMIO makes an error return under any of the following conditions:

1. One of the subscript values is less than the lower bound of its dimension.
2. The size of a dimension  $d(i)$  is negative.
3. The number of words per element is negative.
4. The double precision offset word is negative.
5. The number of dimensions is negative.
6. The buffer length is negative.
7. An EMA is not declared in the calling program.
8. The buffer length plus the page offset of the start of the buffer is greater than the mapping segment size.
9. The entire buffer does not fall within EMA bounds.

**NORMAL RETURN** When .EMIO makes a normal return, the B-register contains the logical address of the element. The contents of the A-register are meaningless.

#### 6-17. MMAP SUBROUTINE (Maps Physical Memory Into Logical Memory)

MMAP is a subroutine that maps a sequence of physical pages into the mapping segment area of the logical address space of a program. It is callable from both Assembly Language and FORTRAN programs.

The Assembly Language calling sequence is:

```

EXT MMAP
JSB MMAP
DEF RTN
DEF IPGS           Page displacement from the start of EMA to the
                   start of the segment to be mapped.

DEF NPGS           Number of pages to be mapped.
RTN return point

```



## MEMORY MANAGEMENT

The RTE FORTRAN IV calling sequence is:

```
CALL MMAP(IPGS,NPGS)
```

Upon return:

```
A-register = 0 if normal return
            = -1 if an error occurred.
```

MMAP returns an error under any of the following conditions:

1. IPGS or NPGS is negative.
2. NPGS is greater than MSEG size.
3. All NPGS to be mapped do not fall within EMA bounds.
4. EMA was not declared in the calling program.
5. IPGS is greater than or equal to EMA size.

If NPGS is less than the standard mapping segment size, the number of pages actually mapped will normally be equal to the standard mapping segment size. The number of pages mapped will be less than this if the starting page of the segment to be mapped lies within an MSEG size of the end of EMA. In this case, the number of pages mapped will include all pages from the starting page to the end of EMA.

MMAP maps one more page than the size of the mapping segment if the end of the EMA is not reached. This is done to prevent dynamic mapping system (DMS) errors in case a multiple word element or a buffer for an I/O transfer crosses the end of the last mapping segment page.

### 6-18. EMAST SUBROUTINE (Returns Information on EMA)

EMAST is a subroutine that returns information about the extended memory area (EMA) of the calling program. It is callable from Assembly Language and FORTRAN programs.

The Assembly Language calling sequence is:

```
EXT EMAST
JSB EMAST
DEF RTN
DEF NEMA (returned) Total size of EMA
DEF NMSEG (returned) Total size of mapping segment (MSEG)
DEF IMSEG (returned) Starting logical page MSEG
RTN return point
```

The RTE FORTRAN IV calling sequence is:

```
CALL EMAST(NEMA,NMSEG,IMSEG).
```

Upon return:

```
A-register = 0 if normal return  
           = -1 if error occurred
```

An error return is made if an EMA is not defined in the calling program.

### 7-1. RTE RELOCATING LOADER

The Relocating Loader (LOADR) reads relocatable code from any input device or FMP file, and produces an absolute load module that is ready for execution. The loader automatically sets up the linkage between the program and any required library files. That is, the user does not have to specify library searches during the load process. The program may be relocated as a background disc resident program, foreground disc disc resident program or optionally have a debug routine appended.

In addition to its linking functions, the LOADR's command parameter options may also be used to list program names and blank ID segments, purge permanent programs from the system and add or replace permanent programs.

The Relocating Loader has the following features:

- \* Can be operated under control of the File Manager in batch mode.
- \* Is swappable and can be operated in either real-time or background disc-resident areas.
- \* Allows programs declaring COMMON to reference either a system COMMON area (shared with other programs) or a local COMMON area (not shared with other programs).
- \* Can relocate programs from relocatable files (Type 5 files).
- \* Can scan and relocate from user library files.
- \* Allows a program to be permanently added or deleted from the system. Only the loader can be used to purge a permanent program. (The OF, name, 8 command will not remove a permanent program from the system.)
- \* Can read LOADR commands from a command file to control the load process. Allows temporary loads into either the real-time or background area for execution with an optional debug routine.
- \* Allows a program to reference absolute and code replacement type ENT macros.
- \* Uses system area disc tracks left vacant by deleted programs.

## RELOCATING LOADER

- \* Uses a short ID segment when loading a background program segment (when available; see "On-Line Modification below").

### 7-2. RU,LOADR COMMAND OPTIONS

Parameter options are available in the RU,LOADR statement that permit user specification of the following items:

1. Command file name.
2. File or the logical unit number of the input device for relocatable code.
3. File or the Logical Unit number of the list destination.
4. An operation code that allows Subsystem Global Area (SSGA) flag access together with COMMON type and program type.
5. A program format code that includes temporary loads with DBUGR features.
6. Listing characteristics.

A detailed description of the RU,LOADR statement is given under Loader operation in this section.

At load time, the user need not know the actual address of the partition in which the program will run because each partition appears to be within the first 32K words of memory. The location at which a program area appears to begin is a logical address, and the program is relocated with respect to this logical address. Logical memory address space configurations are illustrated in Section VI, Figure 6-3. It is not necessary to declare the partition number that a program will execute in, since a program will run in any partition large enough to accomodate it.

### 7-3. PROGRAM RELOCATION

During loading, programs are relocated to start at the beginning of the disc-resident program area of logical memory. If COMMON is declared, the program will be preceded by the COMMON area. The logical address of the program location always begins at a page boundary. The first two words of the program location are allocated for saving the contents of the X and Y registers whenever the program is suspended. Once relocated, the program is linked to external references such as EXEC or the Relocatable Library.

Any program segments will overlay the memory area immediately following the main program and its subroutines.

The loader stores the absolute version of the program, its subroutines and linkages on a disc track or a group of contiguous disc tracks and then assigns the disc tracks to the system.

The program, together with its subroutines and its largest segment, may be as large as the largest partition of the same type. If a program is assigned to a partition, it must not be larger than the partition or an L17 error results (see Loader Error Messages). COMMON may be allocated in one of several areas according to the needs of the programmer (see the optional parameter list for the RU,LOADR request).

#### 7-4. ON-LINE MODIFICATION

The operator can use the loader to permanently modify the set of disc resident programs previously loaded during generation. The loader adds new disc-resident real-time or background programs, and also replaces disc-resident programs with updated versions having the same name. A program to be replaced must have all the following conditions present:

- \* Must be dormant
- \* Not currently occupying a partition
- \* Not in the time list
- \* Have a zero point of suspension.

The OF,xxxxx,8 operator command deletes disc-resident programs or segments that were loaded temporarily into the system by the loader. The OF command cannot delete programs or segments that were permanently added on-line using the loader, or stored during generation using the On-Line Generator (RT4GN).

The On-Line Generator stores disc-resident programs on disc in an absolute, packed format. Each main program is identified and located by a 33-word ID segment. The ID segments are stored in the ID segment area of the system disc area and brought into main memory when the system is started up. For disc-resident programs, the program's disc location as well as its main memory and base page addresses are kept in the ID segment. When a main program and segments are loaded, the segments are identified and located by a nine-word short ID segment. When a main program declares an External Memory Area, three-word ID extension is allocated. See Appendix B for the ID segment and extension format.

RT4GN can create a number of blank 33-word and 9-word ID segments so that the loader can later add new programs and segments to the permanent system. It can also create blank ID extensions. The addition or replacement of a program involves the conversion of relocatable programs into an absolute unit, finding space on the disc to store it, and recording information in the ID segment.

## RELOCATING LOADER

The loader always attempts to use the short ID segment for identifying a program segment. However, a standard 33-word ID segment is used if short ID segment is not available.

A program declaring an EMA cannot be loaded if an ID extension does not exist for the program.

When replacing a program, the new program may overlay the old program's disc space only if the length of the new program (plus base page linkages) does not exceed the disc space formerly occupied by the previous program. A track or group of tracks is allocated for program storage when adding a program or if space requirements of a replacement program exceed those of the old. These newly allocated tracks are software-protected but not hardware-protected. Memory resident programs can neither be added nor replaced in the system.

When performing an on-line modification, the disc hardware protect must be physically disabled prior to the loading (and then enabled afterwards) unless the protection is always kept disabled. RTE provides additional software protection for any tracks containing system programs or user programs.

### 7-5. SEGMENTED PROGRAMS

Segmented modules can be added and replaced in any order provided that the main program is always entered first. Permanent replacement of a permanent program or main segment programs will not necessarily result in the main and segments being stored on contiguous tracks.

When replacing segmented program modules, the operator must either replace every segment with a new segment having the same name, or else remove the original segments permanently from the system.

Note that a main and all its segments must be relocated at the same time (see "Loading Segmented Programs" later in this section).

### 7-6. ADDING NEW PROGRAMS

A new program to be added to the system is stored on a complete disc track or several contiguous tracks. A blank ID segment is allocated to record the program's memory and disc boundaries, name, type, priority, assigned partition, and time values. The loader attempts to use available disc space in the system before allocating new full tracks. If new tracks must be allocated, they are assigned to the system and are software-protected.

## 7-7. PROGRAM REPLACEMENT

When replacing one program with another, the following sequence of events take place as appropriate to the current conditions:

1. The new program is first relocated onto scratch disc tracks.
2. The new program will use the same ID segment as the old program but will only use the same disc space if the length of the code and base page does not exceed the old program size.
3. If the new program cannot be fitted into the disc area of the replaced program, the loader then looks for another area of appropriate size if one was previously freed by the user through deleting a program incorporated during generation. In this case, the deleted program's ID segment had its name blanked but its disc space was retained. That disc space is given to the new program.
4. If neither condition exists (items 2 and 3), the scratch tracks on which the new program was generated become system protected and the old ID segment is retained.

## 7-8. ADDITION OR REPLACEMENT LIMITATIONS

Several limitations may prohibit the final addition or replacement of disc-resident programs:

1. System or reverse COMMON is requested but the program's COMMON length exceeds that of the COMMON area.
2. Local COMMON is requested and COMMON is not declared by the first relocatable module encountered by the loader, even though the module is a dummy module that contains no executable code.
3. The base page linkages exceed the corresponding linkage are for disc-resident programs established by the system during generation.
4. The length of the absolute program unit exceeds the area available.
5. Disc space is not available to store the program.
6. A blank ID segment is not available for adding a program (program previously loaded can be deleted to create a blank ID segment) or its segments.
7. An ID extension is not available for adding a program with an EMA.

## RELOCATING LOADER

### 7-9. PROGRAM DELETION

A temporary program is deleted from the system with the OF,name,8 command. A permanent program (i.e., a program loaded during generation, or on-line with the loader as a permanent addition or replacement load) is deleted with the loader. When using the loader to delete a permanent program, the opcode parameter is set to PU, which blanks the program's ID segment and makes it available for loading another program.

The tracks containing the program are released unless they are system tracks. If the program had been saved through the File Manager on FMP tracks, the tracks are not released to the system but remain as FMP tracks.

Any time a temporary or permanent program is deleted from the system, all its segments must also be deleted. This is required since segments may have occupied tracks that were released by deleting the main program.

+-----+  
|  
| NOTE  
|  
+-----+

Only the LOADR may perform permanent loads or deletes. Copies of LOADR may perform temporary loads but will be aborted with an IO06 error return if the attempt is made to perform permanent loads or purges.

### 7-10. COMMON ALLOCATIONS

Three options can be specified when allocating a COMMON area for a program:

**SYSTEM COMMON.** This implies a background program with COMMON in the background system COMMON area, or a real-time program with COMMON in the real-time COMMON area. System COMMON is established when the system is generated.

**LOCAL COMMON.** The local COMMON area for a program is established at the beginning of the background program's area. The COMMON area will be swapped together with the program. It is necessary for the first COMMON allocation to be the largest declared. RTE FORTRAN IV named COMMON is handled the same as local COMMON.

**REVERSE COMMON.** This implies a background program with its COMMON in the real-time COMMON area. Conversely, a real-time program can reference and use the background system COMMON area. Reverse COMMON is established when the system is generated.



## 7-11. PROGRAM TYPES

When a program is assembled or compiled, it may be assigned to a program type that is kept in the NAM record. The type information is used by the On-Line System Generator and, in some cases, by the Relocating Loader. (Refer to the RTE-IV On-Line Generator Reference Manual for information on program types handled by the generator.)

The Relocating Loader handles Type 6, 7, 8 and 14 modules as though they were normal subroutines (Type 7) to be appended to the program making reference to them. The loader SE command (see below) will relocate these types of modules if an entry point in a module satisfies a previous external reference.

The loader opcodes corresponding to module's NAM types are as follows:

<u>NAM Type</u>	<u>LOADR Opcode</u>
2	RT
3	BG
4	LB
0	BG (default)

where NAM Types:

- 2,3,4 and 0 are main programs (NAM Type 5 is a program segment)
- Type 2 programs are real-time programs that are relocated with access to Table Area II (see (3) of Figure 6-3, Section VI).
- Type 3 programs are background programs that are relocated with access to Table Area II (see (3) of Figure 6-3, Section VI).
- Type 4 programs are background programs that require a larger logical address space for the program. A larger address space can be acquired, since Table Area II and the System Driver area are not included in the program's address space.

Other information regarding program types is provided in Appendixes D and F of this manual.

## RELOCATING LOADER

### 7-12. LOADER OPERATION

The loader is scheduled for execution with the RU or ON operator command in the format

```
RU,LOADR,command[,input[,list[,opcode[,format[,partition  
[,size]]]]]]]
```

where:

**command** The command file structure must be used for loads when more than one relocatable file is required. The <command> parameter specifies:

1. A command file <namr>.

2. An interactive input device from which commands may be entered. When commands are entered interactively on such device, a /LOADR: prompt is displaced when the loader is ready for a new command.

3. A non-interactive input device, such as a tape cassette, from which commands may be entered. No prompt is issued by the loader to solicit new commands.

If this and all other parameters are omitted, command entry defaults to the Logical Unit number of the user's terminal.

**input** The file name of the relocatable main program or the Logical Unit number of the relocatable input. There is no default case.

**list** List output device. The default setting is the Logical Unit number specified in the <command> parameter. If the <command> parameter is a file or is not interactive, the default is Logical Unit 6. Refer to the <opcode> parameter below for list options. The list device is locked for the duration of the load if the LU is not interactive and is not a file.

Alternately, a list file <namr> may be specified. The listing will then go to a file. The file named must not already exist. The loader must create the file. The one exception to this is if the specified file name has an apostrophe as its first character; for example:

```
`name
```

In this case, the loader will create the file if it does not exist, or simply open the file if it does exist.

opcode Mnemonic operation code. The parameter defines the program type, COMMON type, and whether or not the program requires the Subsystem Global Area (SSGA). To determine the operation code mnemonic, select one or more (or none) from each of the following columns:

Program Type	COMMON Type	Load Type
-----	-----	-----
BG	SC	PE
RT	RC	TE
LB	NC	RP
	SS	

where:

BG = Background program  
 RT = Real-Time program  
 LB = Large background program  
 SC = System COMMON  
 RC = Reverse COMMON  
 NC = No COMMON (or local COMMON)  
 SS = Use Subsystem Global (SSGA). SS may also be used with other elements in its same column.  
 PE = Permanent Program  
 TE = Temporary program  
 RP = Replace permanent program (do not also specify PE).

The default setting is BGNCTE.

The elements of the selected mnemonic code may be specified in any order with no intervening commas or blanks. For example, PEBGSS will be interpreted the same as SSBGPE, which specifies a background program using Subsystem Global to be made a permanent program. One, two or all three parameters may be specified.

## RELOCATING LOADER

**format** Mnemonic format code. This actually is an extension of the opcode that was filled. The parameter defines the format for the program load operation. To determine the format code, select one or none from each of the following columns:

DEBUG Append -----	List Options -----	File Scan -----
DB	LE NL	RS

where:

DB = append DBUGR subroutine to the program

LE = list entry points

NL = no listing desired

RS = reverse scan. RS changes the order of loading for segmented programs. The default is load segment, rescan file and load system library routines. However, when RS is specified, rescan of the file is performed only if undefined external references remain after a library search. Selection of this option can significantly speed up segment loading. See "Loading Segmented Programs" later in this section.

Do not specify RS if a system library routine is to be replaced by a user routine.

Format and opcode parameters may be intermixed and intermingled in any order. For instance, SSBGRT will be relocated as a real-time program using SSGA. Note that later specifications will override earlier specifications.

**partition** The specific partition number in which program is to be executed. If not specified, the program will execute in any available partition of sufficient size. This is the same as using the AS operator command.

**size** Allows a logical address space larger than the program size. Permits use of a dynamic buffer at the end of the program for use as a data array, symbol table space, etc., when the program requires such space. If the program is an EMA program, the EMA area immediately follows the dynamic buffer area.

The <opcode> and <format> parameter mnemonics can intermingled in any order. That is, <opcode> mnemonics can be mixed with <format> mnemonics, and vice versa. A comma must be included as a parameter position marker if:

1. The character count within the parameter exceeds six, or
2. Subsequent parameters such as <partition> are to be specified.

The following examples show typical usage of the <opcode> and <format> parameters:

```
*RU,LOADR,PROG1, , ,RTDBSS,NL
      ^   ^   ^   ^   ^
      |   |   |   |   |----- <format/opcode> parameter
      |   |   |   |   |----- <opcode/format> parameter
      |   |   |   |   |----- <list output> parameter position
      |   |   |   |   |----- <input> parameter position
      |----- <command> parameter
```

```
*RU,LOADR, , , ,RP, ,7
      ^ ^ ^ ^ ^ ^ ^
      | | | | | | |----- <partition> parameter
      | | | | | | |----- <format/opcode> parameter position
      | | | | | | |----- <opcode/format> parameter
      | | | | | | |----- <list output> parameter position
      | | | | | | |----- <input> parameter position
      |----- <command> parameter position
```

If a track allocation cannot be made for a relocation, the loader displays the message WAITING FOR DISC SPACE. The loader repeats the disc request and is suspended until space becomes available.

Following the relocation of a program that has its external references satisfied, the loader terminates with one of the following messages:

```
ww PAGES RELOCATED  xx PAGES REQ'D  NO PAGES EMA  NO PAGES MSEG
      or
ww PAGES RELOCATED  xx PAGES REQ'D  DEFAULT    zz PAGES MSEG
      or
ww PAGES RELOCATED  xx PAGES REQ'D  yy PAGES EMA  zz PAGES MSEG
/LOADR name READY

/LOADR:$END
```

where:

ww = the number of pages occupied by the relocated code  
(includes base page.)

xx = size in pages of the partition required by the program

## RELOCATING LOADER

yy = the EMA size in pages (for EMA programs only)

zz = the MSEG size in pages (for EMA programs only)

name = name of main program. The loader terminates and the program is ready to run.

If a new program is loaded bearing the same name as a main program already defined in the system, the following message is displayed:

```
DUPLICATE PROG NAME -<nnnnn>
```

where <nnnnn> is the duplicated program name. the loader automatically attempts to create a unique program name by replacing the first two characters of the new program's name with period characters (..). If successful, the loading process continues and when completed, the following messages are displayed:

```
/LOADR: <..nnn> READY  
/LOADR: $END
```

where <..nnn> is the modified program name.

If unsuccessful; that is, a program named <..nnn> already exists, the loader is aborted and the appropriate error message is displayed.

Whenever the loader completes a successful or unsuccessful load, it returns five words of information about the load to the program that scheduled it, via the PRTN system subroutine. The returned information can be accessed via RMPAR. For example, when the loader File Ma is run from the nager, FMGR picks up the information in 4P and parameters 1P, 2P, 3P, 5P (this is also the FMGR 10G). A successful load gives the following:

```
1P,2P,3P = program name
```

See the Batch-Spool Monitor Reference Manual for a description of global parameters. If an unsuccessful load occurred, the following information would be returned:

```
1P,2P,3P = 0  
4P      = L-  
5P      = loader error return
```

### 7-13. ADDITIONAL OPCODE PARAMETERS

The loader's <opcode> parameter has two other uses. Entering LI or PU causes the loader to, respectively, list all currently active programs in the system, or purge a permanent program. Opcodes LI and PU may be used in the interactive mode but may not be entered in batch mode or from a command file.

The syntax for the list option is as follows:

RU,LOADR,,,lu,LI

In this case, a list of all active programs in the system is typed to transmit the specified Logical Unit. The list will include the name, program type, priority, low and high main program addresses, low h Base and high Page addresses, and partition number if the program is tied to a assigned partition. Each blank ID segment available for use by the loader is noted by <long blank ID> or by <short blank ID> if the ID is a segment nine-word program segment ID segment.

It is printed as a table in the form:

NAME	TYPE	PRIORITY	LO	MAIN	HI	MAIN	LO	BP	HI	BP	SIZE	EMA	MSEG
		PART	N										

An alternate form of the request is:

RU,LOADR,,PROG,LU#,LI

This will list all of the above information only for the program named PROG.

If the opcode is PU, the message

/LOADR: PNAME?

is output on the assigned Logical Unit device. Entering a program name following the prompt causes the loader to permanently purge the referenced program from the system. Entering a /A will prevent any purge operation and terminate the loader.

The LI and PU opcodes may also be entered in the interactive mode but may not be entered during program relocation. The PU command may not be entered from a command file or under batch mode.

#### 7-14. LOADING THE BINARY CODE

The RTE-IV loader will accept binary relocatable code from any FMP on any file disc cartridge. The file <namr> of the main may be the RUN included in statement. If all segments and all subroutines are file <nin the input amr>, then no further information is needed. and subHowever, segments routines will frequently be in several files system, throughout the and in this case, additional commands to the The addLOADR are required. itional commands may be specified through a interaccommand file, an tive or non-interactive Logical Unit. The file specifi<namr> or LU is ed in the first loader RUN parameter.

In the interactive mode the loader prompt /LOADR: is issued:

/LOADR:

## RELOCATING LOADER

### 7-15. LOADER COMMAND FILE

The loader will load all relocatable input found in the file specified by the RUN statement. However, subroutines or segments will often be located in other files. In order to facilitate loading of a program broken up in this manner, the loader will take input from a command file. The command file syntax and meaning are described below. Note that only the first two characters of any command are required unless otherwise specified.

- SEARCH Searches the system disc library for undefined externals.
- SEARCH,<namr> Searches the file <namr> for undefined externals. Only the first two characters of this command need be specified for a single-pass search of the named file. If more than two characters are used in the command; that is, SExxxxx,NAMR instead of SE,NAMR, the file is searched multiple times to ensure that backward references are satisfied. The SE,NAMR form is faster but will not satisfy backward references.
- RELOCATE,<namr> Loads file <namr> as part of the program. The <namr> specified may be a program, subroutine or segment.
- FORCE Force loads a program and/or program segment. Undefined externals will be ignored.
- DISPLAY Causes a list of undefined exxternals to be printed on the list device, or in the interactive mode, on the interactive command device. Note that the undefined externals listed are those referenced by the module being loaded; that is, undefined externals in the main of a segmented program will not be listed if the current module being relocated is a segment.
- ECHO  
(see footnote) Causes the input commands from a file to be echoed on the list device as they are encountered. This is useful for debugging loader command files. The command is ignored if the commands are coming from an interactive device.
- END End of command input. Signals the loader to exit the command mode and finish up the load. If undefined externals exist at this time, an automatic scan of the system library is performed.
- /A Aborts the loader immediately. A clean termination of the load operation is performed.



Denotes a comment line when entered as the first character of an entry line. The loader ignores the entire line. Comments may also follow a command and be in the same entry line as the command, providing two commas appear in the line. For example:

```
SE,,SEARCH THE LIBRARY
RE,XTABS,LOAD PROGRAM NAMED XTABS
DI,,DISPLAY UNSATISFIED EXT REFS
```

AS,xx Assigns the relocated program to partition xx.  
(see footnote)

SZ, yy This command allows the user to request more memory for the program than the actual program code requires. The extra space is called dynamic buffer area. YY is the number of the pages of memory for the program and dynamic buffer area. For EMA programs, the EMA area will immediately follow the dynamic buffer area. Note that this dynamic buffer area may be changed on-line for non-EMA programs with the SZ operator command.

LL,<namr> Specifies the list Logical Unit number or file name. If the listing is to go to a file. If a file name is specified, the file must not already exist unless its name begins with an apostrophe (').  
(see footnote)

OP,<opcode> Specifies an <opcode> parameter where <opcode> is as defined previously. Note that opcodes LI or PU are illegal in a file, but are legal in the interactive mode.  
(see footnote)

FM,<format> Specifies a <format> parameter, where <format> is as defined previously.  
(see footnote)

At the end of every segment load, main load, and at the end of a command file, the system library is searched for undefined externals. If undefined externals still exist and the commands come from a file, then the undefined externals will be listed and the loader will abort.

The loader prints the message:

```
UNDEFINED EXTS
```

The external references are listed, one per line.

-----+  
| FOOTNOTE:

| Specification of these commands must precede specification of any  
| RELOCATE or SEARCH command. Otherwise, the control command will be  
| ignored if entered from an interactive device, or cause errors if  
| entered from a file. These commands may be entered either within the  
| RU command or from a command file. Note that RU command parameters  
| will be overridden by any commands subsequently entered from a  
| command file.  
|-----+

## RELOCATING LOADER

Note that during the load process, undefined externals are allowed in the main of a segmented program because they might be satisfied in a segment. When the user specified the end of the loading process, the main is then checked for undefined externals. If undefined externals exist, the following error message is issued:

```
MAINS
UNDEFINED EXT
```

and the loader will then abort unless the FORCE option is in effect.

The loader will not allow undefined externals in a segment because one segment's entry points may not satisfy another segment's externals. This is because only one segment may be in memory at a given time. The DISPLAY command will list undefined externals. Note that the list refers only to the main or current segment being loaded.

The abort may be prevented by the FORCE command. The FORCE command will force load a program and/or program segment.

### 7-16. LOADING FROM A LOGICAL UNIT

Relocatable code from a Logical Unit can be accepted by the RU,LOADR,,<lu> command or interactively with the RELOCATE,<lu> command. If more than one tape is to be mounted for the load, the interactive mode must be used and the RELOCATE,<lu> command reentered for each tape.

### 7-17. LOADING SEGMENTED PROGRAMS

The loading of segmented programs requires special loader processing. The loading speed of such programs can be increased if the load process is understood and the suggestions given below are followed. Generally, all the relocatable code will be in one file or several files scattered throughout the system.

Assume the following program:

A program has three segments and seven subroutines located in one file, as illustrated in Figure 7-1.

	S	S		S		S	S	S	S
	U	U		U		U	U	U	U
Main	B	B	SEG1	B	SEG2	SEG3	B	B	B
	1	2		3			4	5	6
							7		

Figure 7-1. Segmented Program Example

The loader would relocate this program as follows:

1. Load MAIN program.
2. Load SUB1 and then SUB2.
3. If there are undefined externals references, search entire file for subroutines required by the MAIN.
4. If any subroutines are loaded in Step 3, repeat Step 3 to satisfy backward external references (i.e., assume SUB6 is loaded and it references SUB3).
5. If there any undefined external references, search the system library and relocatable library.
6. If there are still undefined externals, continue loading (they may be satisfied by a segment).
7. Load SEG1.
8. Scan any subroutines following this segment and before next segment for undefined externals (i.e., SUB3) and load them if necessary.
9. If there are undefined externals, search the entire file for referenced subroutines.
10. If any subroutines are loaded in Step 9, repeat Step 9 to satisfy backward external references.
11. If there are undefined external references, search the system and relocatable libraries.
12. If there are still undefined externals, abort the load.
13. Continue Steps 7 through 12 for each segment.

The loading sequence described above has several implications for the user when preparing a segment load:

- a. A subroutine called by many segments need only appear once in the file.
- b. Subroutines referenced in the MAIN are loaded with the MAIN and are thus sharable by all segments. Subroutines loaded with the MAIN are not loaded with segments.
- c. Any subroutines located before the first segment are relocated with the MAIN.
- d. Any subroutines located after the first segment in a file are loaded only with those segments that reference them.

## RELOCATING LOADER

What the above basically implies is that subroutines may appear anywhere in the file, even as a library concatenated at the end of a file. This provides optimal loading in terms of program size, but does not necessarily provide optimal loading speed. To optimize loading speed, subroutines that are referenced by a segment should be located directly behind that segment.

When a relocatable program is contained in several files, a command file should be used to load the program. Typically, the MAIN program would be in one file, each segment in a separate file, and perhaps a file of subroutines that are referenced by some of the segments. The command file for loading such a segmented program might consist of the following:

File Entry -----	Command -----	Resulting Action -----
a.	RE,MAIN	Relocates program named MAIN
b.	SE,LIBRY	Searches library named LIBRY
c.	RE,SEG1	Relocates segment named SEG1
d.	SE,LIBRY	Searches library named LIBRY
e.	RE,SEG2	Relocates segment named SEG2
f.	SE,LIBRY	Searches library named LIBRY
.	.	.
.	.	.
.	.	.

When the loader encounters the command in file entry c, it recognizes the program as segmented. Before SEG1 is loaded, LOADR searches the system and relocatable libraries for undefined external references. Undefined externals are still permitted at this point, since they might be satisfied in a segment.

However, at file entry e, undefined externals remaining after the system and relocatable libraries are searched will cause LOADR execution to be aborted. This is because a segment may not satisfy an undefined external reference through another segment. (The FORCE option may be specified to force load the code and prevent an abort condition.) Upon completion of the loading process, any remaining undefined external references in the MAIN program would result in the loader being aborted and display of the following messages:

```

/LOADR: MAINS
/LOADR: UNDEFINED EXTERNALS
/LOADR: <list of MAIN program's undefined externals>
      .
      .
      .

```

#### 7-18. REDUCING SEGMENTED PROGRAM LOAD TIME

There are several ways to increase segmented program loading speed. Those described below are suggestive only, and are not to be considered as required procedures:

1. Place any referenced subroutine with the segment that calls it. This eliminates unnecessary file scans in search of a subroutine that will be relocated with a segment.
2. Place subroutines into files in the sequence in which they are called. That is, if SUB1 calls SUB2, place SUB1 in the file before SUB2, etc. For example, assume these subroutines are in a library file to be searched by the loader and that the loader is looking for SUB1. Ideally, the loader would pick up SUB1 and create SUB2 as an undefined external reference. The loader would then continue the file search; if SUB2 was then encountered, it would be picked up on the same pass. However, if SUB2 was located in front of SUB1, an additional file search would then be necessary.
3. If all the relocatable code is within the same file, place the subroutines in the sequence suggested in Item 2.
4. If several segments reference the same subroutine, place that subroutine immediately following the MAIN program. Segments may share subroutines that are loaded together with the MAIN program.

## RELOCATING LOADER

5. When all the relocatable program code is within the same file and the file has been organized as described in Item 2, use the RS operation code when the loader is scheduled. RS informs the loader that all subroutines have been sequenced as suggested above, and that the system and relocatable libraries are to be searched before a file scan. That is, in the loading steps previously described for the segmented program load example, Step 5 would be placed between Steps 2 and 3, and Step 11 would be placed between Steps 8 and 9. Another scan of the file will occur if undefined external references remain following a scan of the system and relocatable libraries.

Caution should be exercised in using the RS mnemonic, since it changes the loading sequence so that the HP relocatable library is searched before a scan of the file is made. It is therefore possible that a relocatable library subroutine might be loaded instead of a user's subroutine. However, this could only occur if the subroutine had the same entry point name as a relocatable library routine (i.e., SIN, TAN, ARCTAN, etc.) and if the user's subroutine was not included at the end of the segment or main that called it.

### 7-19. DBUGR LIBRARY SUBROUTINE

DBUGR is a utility subroutine distributed with the RTE-IV operating systems. It is appended to the end of a user's program by the loader when the opcode parameter in the RU,LOADR command is DB. DBUGR allows the user to debug a program by means of Trace, Break Point and other features. Permanent loads are not allowed with DBUGR. A summary of DBUGR commands, is given in Section XI of this manual. For a detailed description of DBUGR commands, see the RTE-IV Debug Subroutine Manual.

### 7-20. LOADER ERROR REPORTING

All loader errors are reported to the list device. The list device may be specifically declared in the ON or RU scheduling command, or defaulted. The default list device is specified under "LIST = " earlier in this section.

The error codes are displayed on the list device in the following form:

```
/LOADR:<error code>
```

For some non-recoverable error conditions, LOADR aborts execution and displays the error report as follows:

```
/LOADR:<error code>  
/LOADR:LOADR ABORTED
```

At times, the user may wish to abort a load while the load is going on. Entering a BR,LOADR command will cause the loader to abort a load and perform a clean and orderly termination. This is greatly preferable to using an OF,LOADR command during a load process, which may leave files open.

For some error codes, the name of the program module and the entry point name of the subroutine being relocated are displayed prior to the error code display line, as follows:

```
/LOADR:<module name> /LOADR:<entry point name> /LOADR:<error code>
```

#### 7-21. LOADER ERROR CODES

All loader error codes, their meaning and possible recovery action are listed in Table 7-1 below. Note that the asterisks following some diagnostics have the following meaning:

\* = module name printed BEFORE diagnostic

\*\* = entry point name printed AFTER module name

The asterisks would not actually appear in the displayed error code. All error codes are prefixed by L- characters.

Note that numbered items in the "Recovery Action" column indicate possible alternatives, as appropriate, rather than sequential steps.

Table 7-1. Loader Error Codes

Error Code	Meaning	Recovery Action
L- :		
01*	Checksum error. (Was it a relocatable file?)	Specify correct relocatable file or recompile. Give loader the correct file.
02*	Loader found an entry that was not a NAM, ENT, EXT, DBL, EMA or END record. Did the compiler emit bad records? Was it a relocatable file?	Recompile. Give loader the correct file.
03*	Program code and system tables exceeded 32K or user-specified max. size. (Program size + MSEG size is too large.)	<ol style="list-style-type: none"> <li>1. Segment program (see Sect.8).</li> <li>2. Do NOT specify a size; make it a Type 4 program if possible.</li> <li>3. Move data to EMA area if possible; otherwise, make program smaller.</li> </ol>
04*	BP linkage overflow. The program requires more BP links than system has.	<ol style="list-style-type: none"> <li>1. Rearrange subroutines.</li> <li>2. Rearrange order of loading modules.</li> <li>3. Recode to decrease number of references across page boundaries.</li> </ol>
05*	Symbol table overflow. (Loader does not have enough room to relocate.)	<ol style="list-style-type: none"> <li>1. Use SZ operator command to expand size for loader.</li> <li>2. Use SE loader command to reduce loader fix-up table size.</li> <li>3. Break up code into subrs. in separate files and use SE cmd after relocating each file.</li> </ol>
06*	COMMON block error (was first COMMON declaration the largest?).	Make largest COMMON declaration the first declaration the loader encounters.
07* **	Duplicate entry points encountered or two subroutines with the same name.	Remove one of the duplicate routines or rename one.



Table 7-1. Loader Error Codes (continued)

Error Code	Meaning	Recovery Action
L- :		
08	No transfer address (only subroutines were loaded; no main was found).	1. If program was written in Assembly Language, put a label on the END statement. The label is where the program starts. 2. If program was written in FORTRAN, relocate the module with the 'PROGRAM' statement.
09*	Record out of sequence (Probably attempted to relocate from improperly positioned tape.)	Rewind tape and start over.
10	Illegal parameter in RU statement or in statement prior to a RELOCATE statement.	Start over. Make certain the run string is proper.
11	Attempted to replace a memory resident program with a program having the same name.	Rename program with a different name, recompile and reload. It is impossible to replace a memory resident program. The loader will not even rename it.
14*	Assembler produced illegal relocatable module. A DBL record was produced that referred to an undefined external; i.e., it should have been found in the program's symbol table but was not.	Recompile and try again. This could also be an Assembler or FORTRAN compiler bug.
16	Illegal partition number or corrupt map table. Partition specified does exist or is down due to a parity error.	Either specify a different partition or no partition.
17	Number of pages required exceeds partition size.	Either specify a different partition or no partition.
18	Specified program size too large for partition. (Exceeds 32 pages.)	Either specify a smaller size or no size. See also error code 03 other recovery alternatives

RELOCATING LOADER

Table 7-1. Loader Error Codes (continued)

Error Code	Meaning	Recovery Action
L- :		
19	(1) EMA declared twice (2) EMA declared in a program segment, (3) reference to the EMA label before label was declared EMA, (4) an attempt was made to declare the same label as ENT record (i.e., duplicate ENT).	Specify the EMA in the main and load the main first. An EMA must be declared in the main and any segments or subroutines that reference that EMA must be loaded after the main.
20	No ID extensions available for the EMA program.	Either abort other EMA programs to release required ID extensions, or regenerate and specify more ID extensions.
21	Program's EMA size too large for current system partitions.	Either reconfigure system at boot-up to give more EMA space, or declare less EMA in program.
24**	Attempted to access an SSGA entry point but SSGA access was not declared at beginning of load.	Restart the load, specifying the SS mnemonic; i.e., OP,SS or FM,SS.
25	Attempted to purge a program under batch, or attempted to use LI or PU commands within a file. LI or PU may only be used interactively.	Do not put LI or PU commands in a LOADR command file.
26	Not enough long and short ID segments to finish load.	Off or purge all ID's created, free up additional ID segments, and restart load.
27	Attempted to access an EMA external with offset or indirect.	Use HP-supplied .EMAP and .EMIO subroutines to access EMA arrays

Real-time or background disc-resident programs may be structured into a main program and several segments to save memory space during program execution. A segmented program is first separated by the programmer during the coding process. Once the program is relocated, the segments are then called into memory only as they are needed for execution. The program can run in a smaller partition than its total size, since only parts of the executable code are in memory at any one time.

When the code in one of the segments is required for execution, the currently executing program uses an EXEC call to request the operating system to make a segment overlay. RTE loads the segment from the disc into a memory block following the end of the main program, overlaying whatever was previously there. Control is then passed to the entry point of the segment and execution proceeds within the segment (see Figure 8-1). Note that a segment is not allowed to overlay the main program; segments may only overlay one another.

While a segment is in memory, it can freely access subroutines and data areas in the main program, and vice-versa. The main program and its segment effectively operate as a single program. When another segment is required, either the main program or the segment can make the EXEC call to request another segment overlay. The operating system will then load the new segment into memory and pass control to it.

# SEGMENTED PROGRAMS

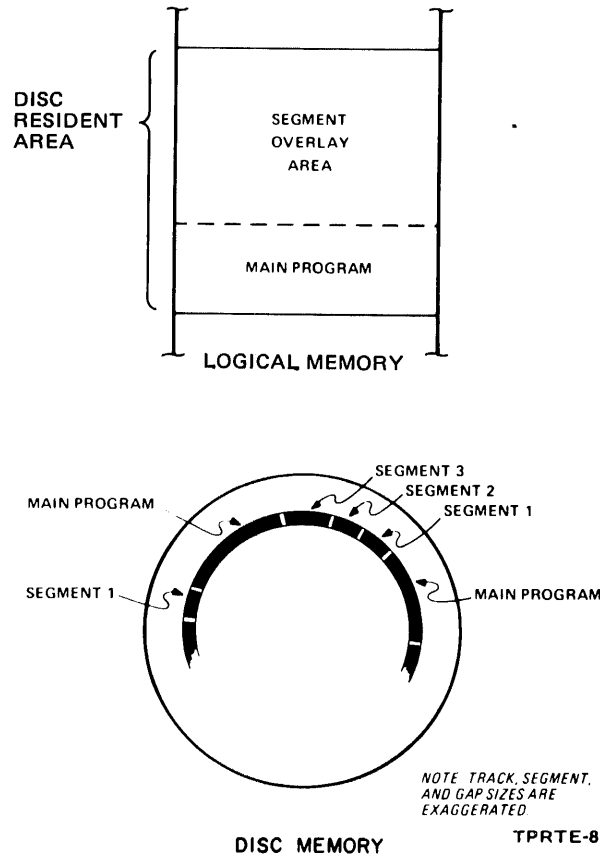


Figure 8-1. Segmented Programs

Segments may be of any size but need not necessarily be of equal length. The entire program requires a partition large enough to hold the main program plus the size of the largest segment.

## 8-1. RTE FORTRAN IV SEGMENTATION

RTE FORTRAN IV programs can be segmented if certain conventions are followed. The main program must be Type 2, 3, or 4, and the segment must be specified as Type 5 in its PROGRAM statement. The segment must be initiated using the Program Segment Load EXEC call from the main program or another segment.

If the program is to be loaded by the generator, each segment must make a non-executable dummy call to the main program. This ensures that the generator establishes the proper linkage between the main program and its segments. For example:

```

:
:
:
CALL MAIN
END
    
```

where MAIN is the name of the main program. This dummy call is not required if the program is loaded by the Relocating Loader.

Chaining of segments is uni-directional. Once a segment is loaded, execution is transferred to it. The segment, in turn, may call another segment but a segment written in FORTRAN cannot easily return to the main program. Segments can call any subroutine attached to the main program. Communication between the main program and segments may be through COMMON.

8-2. RTE ASSEMBLER SEGMENTATION

The main program must be Type 2,3, or 4 and the segments must be Type 5. One external reference from each segment to its main program is required for the generator to link the segments and main program. If the main program accesses an external symbol that will be satisfied in a segment, the symbol may appear in only one segment. Otherwise, the generator or the loader may link the segments and the main program incorrectly.

Figure 8-2 shows how an executing main program may use the JSB EXEC call to bring in any of its segments from the disc. Note that although control is passed to the transfer point of the segment, the main itself is not suspended.

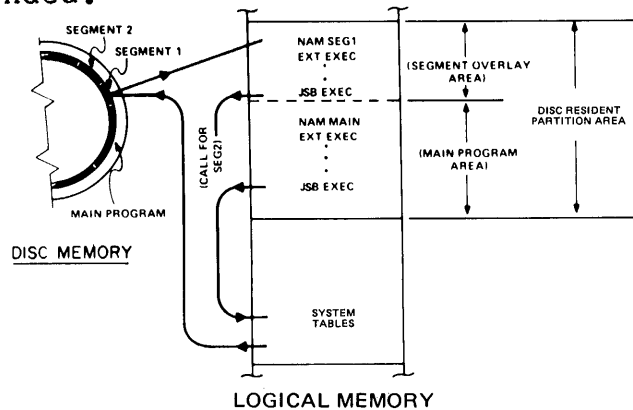


Figure 8-2. Main Calling Segment

An executing segment may itself call in another of the main program's segments by using the same "JSB EXEC" request (see Figure 8-3).

The Multi-Terminal Monitor (MTM) is a software package used to service multiple terminals in an RTE operating system. Included in the description given below are several special considerations applicable to the optional multipoint subsystem operations.

#### 9-1. SYSTEM CONFIGURATION

Multiple terminal operation requires that routines PRMPT and R\$PN\$ be configured into the operating system during generation. By default, they are memory resident and should be included in the system during the generation Program Input Phase.

Configuring a terminal for MTM servicing is performed during the Interrupt Table portion of generation. The following entry is required:

```
sc, PRG, PRMPT
```

where `sc` is the select code of the terminal being configured. This will cause interrupts to those select codes to be handled by program PRMPT.

After the RTE system is initialized and running, each terminal must be initialized with a control request either through an FMGR command:

```
:CN,lu,20B
```

or an EXEC request:

```
CALL EXEC(3,2000B+lu)
```

where `lu` is the Logical Unit number of the terminal being enabled.

#### 9-2. MULTIPOINT INITIALIZATION

Configuring a terminal for multipoint operation is performed during the Interrupt Table portion of generation. Refer to the HP 91730A Multipoint Terminal Interface User's Guide (91730-90002) for a complete description of multipoint operations. The following entry should be made for each communication line:

```
scl, PRG, PRMPT
```

where `scl` is the select code of the line being configured. This will cause interrupts to that select code to be handled by program PRMPT.

# MULTIPLE TERMINAL MONITOR

Each terminal also requires a dummy Equipment Table entry (EQT). Number 77 is a good choice. This same EQT can be used for all terminals. The following entry is then required for each terminal:

```
sct,ABS,0
```

where sct is the dummy select code that has been assigned to the terminals.

After the RTE system is initialized and running, both the communication lines and the terminals must be initialized. Each line is enabled with a control request through either an FMGR command:

```
:CN,llu,20B,100000B+ICW
```

or through an EXEC request:

```
CALL EXEC(3,2000B+llu,100000+ICW)
```

where:

llu is the Logical Unit number for the line

ICW is the control word and has the following bit configuration:

15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
1	XX														

where:

bit 15 is 1 to designate this as a line initiation

TOVAL is the timeout value in hundreds of milliseconds

LN is the logical line number

After the line has been initialized, each terminal on the line must be enabled. This is done using either of the following commands:

```
:CN,ilu,20B,ICW
```

or:

```
CALL EXEC(3,2000B+ilu,ICW)
```

where:

ilu is the Logical Unit number of the terminal.

ICW is the control word and has the following bit specification:

15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
0	LN					GID				DID					

where:

bit 15 is 0 to designate this as a terminal initialization.

LN is the Logical Line Number as specified in the Line Initiation control word.

GID is the Group Identification character as specified on the terminal's communication card.

DID is the Device Identification character as specified on the terminal's communication card.

### 9-3. LOGICAL UNIT NUMBER ASSIGNMENT

A cartridge tape with (CTU) on a 264x terminal (non-multipoint) must have a Logical Unit number either different than the Logical Unit number of the associated display (CRT). It is suggested that the CRT's and CTU's be assigned LU numbers between 09 and 63, inclusive.

### 9-4. OPERATION

MTM will perform several services for the user in conjunction with a terminal's copy of FMGR. A terminal with Logical Unit number xx has its own copy of FMGR if a program exists named FMGxx. For example, the copy of FMGR for Logical Unit 09 would be FMG09. The paragraph entitled "Creating Program Copies" (see below) explains how to make copies of a program.



## MULTIPLE TERMINAL MONITOR

If a copy of FMGR named FMGxx does not exist for a terminal, the standard MTM prompt (xx>) will be issued and the user will be conversing with the RTE operating system. The remainder of this manual section assumes that the terminal has its own copy of FMGR named FMGxx.

### 9-5. AVAILABLE MTM SERVICES

In an MTM environment, a user terminal with its own copy of FMGR has access to four services:

1. Automatic scheduling of FMGxx when the user terminal interrupts the operating system.
2. Variations of the BReak and ABort commands.
3. Automatic renaming of user programs scheduled from FMGxx.
4. Automatic execution of transfer file named HI.

### 9-6. AUTOMATIC SCHEDULING OF FMGxx

If a copy of FMGR called FMGxx exists for the terminal, striking a key on the terminal causes FMGxx to be scheduled for execution. One of two actions will then be taken, depending on whether or not FMGxx is available for execution. Normally, FMGxx is available, since it "belongs" to the terminal. If it is not available, the MTM variations of the BReak and ABort commands may be used to make it available as described below.

### 9-7. FMGxx EXECUTION

If the terminal's copy of FMGR is available for execution (not busy or suspended), three events will occur:

First, the prompt

```
xx>FMGxx
```

will be issued to the terminal.

Second, an

```
:LL,xx
```

is executed automatically (but is not displayed) on behalf of the user terminal to make its LU the list device.

Third, control is transferred to a file name HI, which must exist on LU 2, the system disc.

The HI file is a procedure file usually written by the system manager and placed on Logical Unit 2. Although the file may be empty, it must nevertheless exist or an FMGR -006 error will result. When the end of the HI file is reached, control is transferred to the interrupting terminal. The user is now conversing with the terminal's copy of FMGR.

The system manager or other user can define many useful functions to be performed in the HI file:

- \* Since the FMGxx global parameter OG always equals the turn-on LU number, the HI file can be made sensitive to the turn-on terminal.
- \* The HI file can schedule programs for execution using the RU command.
- \* Commands can also be passed directly to the operating system using the SY command.

Refer to the Batch-Spool Monitor Reference Manual for a complete description of FMGR commands.

#### 9-8. BREAK AND ABORT COMMAND VARIATIONS

Program FMGxx sometimes will be busy when the operating system attempts to schedule it to the interrupting terminal. In this case, the operating system will issue the standard MTM prompt

xx>

and the user will be conversing with the operating system. In addition to the standard Break and ABort operator commands, two variations of the commands will be accepted. These variations apply only when entered from an MTM terminal other than the system console, and only if program FMGxx exists.

Throughout the remainder of this discussion, the term "father" will be used to indicate a program that has scheduled another program and is waiting for the scheduled program to complete before resuming its own execution. The term "son" refers to the program that the father has scheduled. This form of program scheduling is commonly called "schedule with wait".

## MULTIPLE TERMINAL MONITOR

1. MTM BREAK COMMAND - The MTM command BR issued at terminal xx will set the break bit of the last son of FMGxx. The following example illustrates the interaction:

```
user hits a key  
xx>FMGxx  
.  
:  
:  
:RU,PROGA          PROGA runs, and assume  
                   PROGA schedules PROGB  
user hits a key  
xx> BR            no program name specified
```

The BR command will set the break bit in program PROGB, since it is the last son of FMGxx.

The command

```
BR,PROGX
```

will still set the break bit in PROGX and have no effect on FMGxx or any of its sons. For more information on breaking programs, refer to the IFBRK system library subroutine and the BR operator command.

If FMGxx has no sons, the break bit will be set in FMGxx itself. Whenever a FMGR program finds its break bit set, it issues the response

```
FMGR 000
```

at the turn-on terminal and prompts for the next input.

```
+-----+  
|          NOTE          |  
| The BReak command entered from the system console must |  
| still have the program name specified as the first    |  
| parameter.                                              |  
+-----+
```

2. MTM ABORT COMMAND - The MTM AB command issued at terminal xx where FMGxx exists performs the same function as the BR command except that the last son of FMGxx is aborted. Considerable care should be exercised in using this command. If FMGxx has no sons, then the break bit of FMGxx is set and the program is not aborted.

## NOTE

The batch abort command (AB) may only be entered from the system console.

## 9-9. AUTOMATIC PROGRAM RENAMING

MTM manages ID segments so that each user can have his own copy of a program. If the user wishes to run a program with FMGxx as the father (i.e. :RU,PROGX but not :SYRU,PROGX), then in certain circumstances, a copy of the program will be created belonging to the particular terminal and run for the user at the terminal.

MTM will perform this action whenever the program to be run is a son of FMGxx, and the program is a Type 6 FMGR file. A copy of the program will be created with the last two characters being xx, and be scheduled for execution to terminal xx.

For example, if the EDITR is loaded on-line as a temporary load and saved as a Type 6 file, the command:

```
:RU,EDITR
```

will create a program named EDIxx and schedule it to terminal xx. When EDIxx is finished, the ID segment will automatically be returned to the system.

The advantage of processing the ID segments in this way is that all terminals can run the same program but each user gets a personal copy of the program. Therefore, a user does not have to wait for other users to finish with a program before gaining access to it.

The above procedure will work even if the program to be run has been previously restored using the RP command. In fact, the program will be created more quickly, since there would be no disc search time before the program could be run.

If desired, the automatic renaming feature of MTM may be circumvented by using a copy of FMGR that does not "belong" to the terminal at which the user is operating. In this case, none of the features described for MTM apply, and the AB and BR commands will revert to their normal usage.

The program renaming feature of MTM may also be temporarily inhibited when running a program by using the following form of the RU command:

```
:RU,PROGX:IH
```

## MULTIPLE TERMINAL MONITOR

In this case, the actual program named PROGX will be run rather than a copy.

This capability is especially useful when loading permanent programs. The program named LOADR is the only program that can load, replace or purge programs permanently in the system. A copy of LOADR cannot perform these functions. Therefore, if the user is operating from FMGxx at terminal xx, the following command can be used to load a permanent program:

```
:RU,LOADR:IH, .....
```

All programs in Type 6 files (whether RP'ed or not) will be renamed. Permanent programs will not be renamed. A freshly-loaded temporary program will not be renamed until it has been first SP'ed, OF'ed and then at the user's discretion, is RP'ed.

The following example shows how to make a freshly-loaded program eligible for program renaming:

```
:RU,LOADR,,&ABCDE,6,TE      loads program ABCDE temporarily
:SP,ABCDE                   saves ABCDE as a Type 6 file
:OF,ABCDE,8                 gets rid of the loaded copy
:RP,ABCDE                   restores program for faster access
:RU,ABCDE                   program ABCXX
```

### 9-10. CREATING PROGRAM COPIES

The following example shows how to rename the FMGR program to give it several different names. The commands given assume that FMGR has been previously saved with the FMGR SP command:

```
:RN,FMGR,FMG01             rename the file
:RP,FMG01                  restore FMG01 from file
:RN,FMG01,FMG07            rename file again
:RP,FMG07                  restore FMG07 from file
:RN,FMG07,FMG14            rename file again
:RP,FMG14                  restore FMG14 from file
.
.
.
:RP,FMGxx                  restore program FMGxx
:RN,FMGxx,FMGR             rename file back to FMGR
                           for future use (i.e., on next entry into
                           this procedure file).
```

A similar procedure can be followed to make multiple copies of other programs.

Note that the above commands can be put in a file that will be run each time that the system is booted up. This relieves the user of the responsibility of renaming all programs for MTM use if the system went down and had to be rebooted. The last RN command restores the file's original name for future use.

It is recommended that a copy of FMGR be renamed for each terminal in the MTM environment to take advantage of the automatic scheduling capability of MTM.

For example, assume a key on the terminal with Logical Unit number 7 is struck. The terminal issues the following prompt:

```
07>FMG07
(HI file gets executed here)
.
.
.
```

The user is now conversing with FMG07 and the default list device is Logical Unit 7. The HI file has been executed and any legal FMGR command may now be issued.

#### 9-11. PROGRAM SWAPPING

In an MTM environment, there are times when a number of users could be concurrently running copies of the same program, each from their own terminal. If more programs than partitions exist, the system will try to service all users by forcing programs to share partitions; that is, it will force programs to swap. When the program being run at each terminal is interactive, it is then in the best interest of all users to code the input routines to be swappable.

All output to terminals normally is buffered, and thus a program can be swapped. However, terminal input through EXEC calls (CALL EXEC(1,LU, IBUFR,ILENGTH)) normally is NOT buffered and the program therefore is locked into its partition until the user decides on the keyboard response, types it in, and presses the RETURN key. During this time, no other program may use the partition.

If there are more interactive programs than partitions large enough to accommodate them, some users are needlessly forced to wait for others to input their data. This form of system inefficiency can be avoided by coding all terminal input to use the reentrant System Library REIO routine. The calling sequence is identical to the EXEC call for input:

```
CALL REIO(1,LU,IBUFR,ILENGTH)
```

This call allows terminal input to be buffered and ensures maximum system efficiency. See the RTE-IV Library Subroutines Section for further information regarding the use of REIO.

## MULTIPLE TERMINAL MONITOR

Note that the FORTRAN-IV Formatter uses REIO for all input READ requests to ensure that a FORTRAN-IV user's input is always buffered.

## 10-1. INTRODUCTION

RTE-IV operating systems are delivered with a collection of relocatable subroutines that comprise the system library. This group of subroutines are specific to RTE-IV operating systems and are used to interface user programs with system services.

Other collections of H-P relocatable subroutines for more general use are also available as options, and are described in the DOS/RTE Relocatable Library Reference Manual. They have been grouped into the following libraries according to function:

Library Mnemonic	Library Name
-----	-----
RLIB.N	DOS/RTE Relocatable Library
FF4.N	FORTTRAN IV Formatter

RLIB.N contains mathematical and utility subroutines such as SIN, COS, BINRY, etc. The formatter libraries contain subroutines that perform formatted data transfers, interpretation of formats, unformatted input/output of binary data, free field input, and buffer to buffer conversions. In addition, may RTE subsystems (i.e., Batch/Spool Monitor) include subroutines that may be of general use. See the appropriate subsystem manual for more information.

## 10-2. CALLING LIBRARY SUBROUTINES

Library subroutines are called by user programs and are linked to the caller either at generation or load time. These subroutines can be called either by disc-resident or memory-resident programs.

Subroutines referenced by disc-resident programs are appended to the end of the calling program and linked to it either by the loader (LOADR) or On-Line Generator.

Subroutines referenced by memory-resident programs will be placed in the memory-resident library by the generator. These subroutines must either be reentrant or privileged. Several memory-resident programs can then share one subroutine, which can save considerable space in the memory-resident area. Disc-resident programs cannot access routines in the memory-resident library; therefore, copies of these subroutines will be appended to these programs.



## RTE-IV LIBRARY SUBROUTINES

If only one memory-resident program is to access a subroutine, it is advantageous to make it a Type 7 subroutine to force it to be appended onto the calling program. A Type 7 subroutine is not placed in the memory-resident library and therefore need not be privileged or reentrant. This results in faster execution, since the subroutine will not incur the overhead associated with reentrant or privileged subroutines.

### 10-3. REENTRANT SUBROUTINE STRUCTURE

A subroutine must meet two criteria to be reentrant:

1. It must not modify any of its own instructions.
2. It must save all temporary results if it is to be called again before completing its current task.

A subroutine saves temporary results in a Temporary Data Buffer (TDB) that the operating system ensures is unique to each program. For example, assume PROGA is executing a reentrant subroutine that is interrupted by PROGB. If PROGB then begins execution of the same subroutine, the system saves PROGA's TDB until PROGA resumes execution, at which time it restores the proper TDB.

Each time a reentrant subroutine begins executing, the address and length of its temporary data block are transferred to RTE-IV through the entry point \$LIBR in order to save the data. At the end of execution, the re-entrant subroutine again calls RTE-IV through entry point \$LIBX to restore any previous temporary data.

The reentrant subroutine structure is used for subroutines with an execution time exceeding one milli-second. However, for shorter execution times, the overhead time the system uses in saving and restoring temporary data makes reentrant structure unreasonable. Faster subroutines can be structured as privileged.

-----+-----  
| NOTE |  
|-----+-----

A library (Type 6) subroutine can only call another library subroutine or Table Area I or, optionally, Table Area II entry points.

## 10-4. REENTRANT SUBROUTINE FORMAT

The format and calling sequence for reentrant subroutines is as follows:

	NAM	xxxxx,6	
	EXT	\$LIBR,\$LIBX	
ENTRY	NOP		Entry point of subroutine
	JSB	\$LIBR	Tell system to protect TDB
	DEF	TDB	Address of temporary data
	.		
	.		Subroutine instructions go here
	.		
EXIT	JSB	\$LIBX	Tell system reentrant run is finished
	DEF	TDB	Address of temporary data
	DEC	N	Return adjustment (Return point=N+ENTRY)
TDB	NOP		System-supplied link to previous TDB
	DEC	K	Total length of current TDB in words
	NOP		System-supplied return address to calling program
	BSS	K-3	
	-		
	-		
	-		Temporary data (K-3 words)
	-		

## 10-5. PRIVILEGED SUBROUTINE STRUCTURE

Privileged subroutines execute with the interrupt system turned off. This feature allows many memory resident programs to use a single privileged subroutine without incurring reentrant overhead. As a result, privileged subroutines need not save temporary data blocks but must execute very rapidly to minimize the time that the interrupt system is disabled.

Since privileged subroutines disable the interrupt system, EXEC calls are illegal within a privileged subroutine. If one is attempted, the calling program will be aborted with an EX error (See Section III).

## RTE-IV LIBRARY SUBROUTINES

### 10-6. PRIVILEGED SUBROUTINE FORMAT

The format and calling sequence for privileged subroutines is as follows:

NAM	xxxx,6	
EXT	\$LIBR,\$LIBX	
ENTRY	NOP	Entry point to the routine
	JSB \$LIBR	Call the system to disable the Interrupt system and memory protect fence
	NOP	Denotes privileged format
	.	
	.	
EXIT	JSB \$LIBX	Call the system to return to calling program, and to enable interrupts and memory protect fence
EXIT1	DEF ENTRY	Return address

It is also possible to go privileged in a block of in-line code, as follows:

-		
-		
JSB	\$LIBR	Go privileged
NOP		Denotes privileged format
-		First instruction
-		
-		
JSB	\$LIBX	Leave privileged status
DEF	*+1	Both DEF's are required
-		
-		
DEF	*+1	

### 10-6A. MEMORY RESIDENT LIBRARY

The memory resident library area in RTE-IV contains only Type 6 subroutines that are referenced by memory resident programs and Type 14 subroutines forced into the memory resident library at generation time.

Reentrant and privileged subroutines may be placed in the memory resident library during generation by either of the following methods:

1. If the routine is declared as an external (called) by a memory resident (Type 1) program, or is called by another memory resident library subroutine, the subroutine will be automatically placed in the memory resident library by the generator.
2. The routine can be changed to a Type 14 subroutine during the Parameter Input phase of generation (it also could have been assembled as a Type 14 subroutine).

NOTE

After the relocation of the resident library and all memory resident programs, all Type 6 routines are converted to Type 7 (utility) routines.

Not all subroutines referenced by memory resident programs are loaded into the memory-resident library. By declaring the subroutine to be Type 7, the user can ensure that the subroutine will be loaded with the program. Then if .ZRNT and .ZPRV are used instead of \$LIBR, the subroutine will execute faster since the system does not need to do the reentrant or privileged processing prior to executing the subroutine.

#### 10-7. UTILITY SUBROUTINE STRUCTURE

Utility subroutines are subroutines that cannot be shared by several programs because of internal design or I/O operations. Therefore, a copy of a utility subroutine is appended to every program that calls for it. The PAUSE subroutine and the library subroutines FRMTR (FF.N), and FMTIO (F4D.N) are typical examples of utility subroutines.

When the RTE system is generated, all library subroutines other than Type 8 subroutines are converted to Type 7 utility subroutines following the relocation of memory resident programs. All required utility subroutines are then relocated immediately following each user program that references them during program relocation.

#### 10-8. SYSTEM LIBRARY SUBROUTINES

All system library subroutines are described below with the exception of .EMAP, .EMIO, MMAP and EMAST. These four subroutines are the direct concern of memory management considerations and are therefore described in the Memory Management section of this manual.

## RTE-IV LIBRARY SUBROUTINES

### 10-9. REIO (Reentrant I/O)

The REIO subroutine permits user programs to perform reentrant I/O and disc resident programs to be swappable. REIO is a utility type library subroutine and has within its structure a reentrant routine that is appended to each program that calls its. The calling sequence for REIO is:

```
CALL REIO(ICODE,ICNWD,IBUFR,IBUFL)
```

where the parameters are described in the Read/Write EXEC call in Section IV of this manual. Note that REIO can only be used with Read/Write calls and that the optional parameters available in those calls are not allowed in the REIO call. REIO will always perform the requested I/O; however, it will do reentrant I/O only if the buffer is less than 130 words (to save system memory), and the buffer address is at least five words above the current fence address. If the sign bit is set on ICODE, the same error options available with the EXEC call are in effect (i.e. error return followed by normal return). REIO returns the same values in the A- and B-Registers as the standard EXEC call.

A reentrant subroutine may perform I/O using the standard EXEC requests. If the buffer is in the temporary data block (TDB) of either itself or another reentrant routine that called it, the calling program is swappable. If the buffer is in the user area, the program is not swappable (i.e., if the buffer is not in the TDB or user COMMON area, the program is not swappable).

### 10-10. BINRY (Disc Read/Write)

FORTTRAN programs can call the BINRY subroutine, to transfer information to or from the disc. The call must specify a non-EMA buffer array, the array length in words, the disc Logical Unit number, track number, sector number, and offset in words within the sector. (If the offset equals 0, the transfer begins on the sector boundary; if the offset equals n, the transfer then skips n words into the sector before starting.) BINRY has two entry points: BREAD for read operations and BWRIT for write operations.

For example:

```
CALL BWRIT (ARRAY,N,IDISC,ITRK,ISECT,IOFST)
CALL BREAD (ARRAY,N,IDISC,ITRK,ISECT,IOFST)
```

Where:

```
ARRAY = Address of the first element
N      = Number of words
IDISC  = Disc logical unit number
ITRK   = Starting track number
ISECT  = Starting sector number
IOFST  = Number of words offset within a sector
```

There are three basic ways that data can be written on the disc in relation to sector boundaries. Care must be used in planning the WRITE statement in two of the cases to avoid losing existing data:

1. Offset=n (i.e., transfer begins within a sector), and less than the sector is written, or the data transfer ends on a sector boundary. The entire first sector is initially read into an internal buffer, the data is modified according the BWRIT statement, and the entire sector is then rewritten on the disc with no data loss. No special precautions are required in this instance.
2. Offset=0 (i.e., transfer begins on a sector boundary), and less than the sector is written. The remaining data in the sector will be lost if the following precaution is not taken. The entire existing sector on the disc can first be read into a user's buffer, modified to reflect the desired changes, and then rewritten on the disc as a full sector.
3. Offset=0 or n, and a sector boundary is crossed in the data transfer. The remaining data in the final sector will be lost if the following precaution is not taken:

The entire final sector (of the data transfer) on the disc should be read into a user's buffer, modified to reflect the desired changes, and then rewritten on the disc as a full sector.

#### 10-11. RNRQ (Resource Management)

Allows cooperating programs a method of efficiently utilizing resources through a resource numbering scheme. A detailed discussion of resource management considerations is provided following the Class I/O description in the EXEC Call section of this manual.

## RTE-IV LIBRARY SUBROUTINES

The calling sequence for RNRQ is:

```
ICCODE=numb  
CALL RNRQ(ICCODE,IRN,ISTAT)
```

where:

ICCODE defines how the resource number is to be used. (See Figure 10-1.)

IRN the resource number is returned in IRN.

ISTAT status return word.

0 - normal deallocate return

1 - RN is clear (unlocked)

2 - RN is locked locally to caller

3 - RN is locked globally

4 - no RN available now

6 - RN locked locally to other program

7 - RN was locked globally when request was made

A resource number is used when one program wishes to use a resource exclusively with the cooperation of other programs in the system. This resource could be a physical device (see subroutine Logical Unit Lock) or the system itself. Using an RN prevents a low priority program from being interrupted by a higher priority program when executing.

All programs must agree that a certain RN will be used as a lock or busy indicator for a given device.

Figure 10-1 illustrates the format of the control word required in the calling sequence.

15	14	5	4	3	2	1	0
WAIT OPTION		ALLOCATE OPTION			SET DISPLAY		
NO	NO	C	G	L	C	G	L
W	A	L	L	O	L	L	O
A	B	E	O	C	E	O	C
I	O	A	B	A	A	B	A
T	R	R	A	L	R	A	L
	T		L			L	

Figure 10-1. RNRQ Control Word Format

If more than one bit is set in the control word, the following order of execution is used:

1. local allocate (skip step 2 if done)
2. global allocate
3. deallocate (exit if done)
4. local set (skip step 5 if done)
5. global set
6. clear

The system has a certain quantity of resource numbers (RNs) that are specified during generation. If a resource number is not available, the program is suspended until one is free, unless the 'no wait' bit is set. If the 'no wait' bit is set, the IRN location is set to zero. If the RN allocation is successful, the value returned in IRN is set by the system. It has no meaning to the user but must be specified (through IRN) when a lock is requested or the IRN is cleared or deallocated.



## RTE-IV LIBRARY SUBROUTINES

The no abort bit is used to alter the error return point of the call as shown in the following example:

```
CALL RNRQ(ICODE....)
GO TO error routine
normal return point
```

The above special error return is established by setting bit 14 to 1 in the request code word (ICODE). This causes the system to execute the GO TO statement following CALL RNRQ if there is an error, or skip the GO TO statement if there is no error.

### 10-12. RNRQ ALLOCATE OPTIONS

LOCAL - Allocate an RN to the calling program. The number is returned in the IRN parameter. The number is automatically released on termination of the calling program, and only the calling program can deallocate the number.

GLOBAL - Allocate an RN globally. The number is released by a request from any program.

CLEAR - Deallocate the specified number.

### 10-13. RNRQ SET OPTIONS

LOCAL - Lock the specified RN to the calling program. The RN is specified in the IRN parameter. The local lock is automatically released on termination of the calling program. Only the calling program can clear the number.

GLOBAL - Lock the specified RN globally. The RN is specified in the IRN parameter and the calling program can globally lock this number more than once. The number is released by a request from any program.

CLEAR - Release the specified number.

If the RN is already locked to someone else, the calling program is suspended (unless the no wait bit is set) until the RN is cleared. If more than one program is attempting to lock an RN, the program with the highest priority is given precedence. A single call can both lock and clear an RN.

If a program makes this call with the clear bit set, in addition to either the global or local set bits, the program will wait (in the general wait list) until the RN is cleared by another program and then continue with the RN clear.

An entry point is provided for drivers or privileged subroutines of Type 3 programs that wish to clear a global (and only global) RN:

```
LDA    RN
JSB    $CGRN
return point
```

An example on how to use ICODE follows:

Assume you wish to get an RN assigned so that any program can access it. You also want an alternate return point in case of error. Bits 4 and 14 would then be set as follows:

```
100 000 000 010 000 = 10020B
```

#### 10-14. LURQ (Logical Unit Lock)

Allows a program to exclusively dominate (lock) an input/output device. The calling sequence is:

```
DIMENSION LUARY(n)
IOPTN=numb
NOLU=aa
CALL LURQ(IOPTN,LUARY,NOLU)
```

#### Parameters

IOPTN	control parameter (an octal number) 0x0000-unlock specified LUs 1x0000-unlock all LUs the program currently has locked 0x0001-lock with wait specified LUs 1x0001-lock without wait specified LUs x (bit 14) is the no abort bit. x=4 to set, else x=0.
LUARY	an array of n LUs to be locked or unlocked.
NOLU	number of LUs to be locked or unlocked.

This request temporarily assigns a logical unit to the calling program. It prevents a higher priority program from interrupting a program's use of the device until the device is unlocked by the program that locked it.

The Logical Unit Lock request allows up to 31 programs to exclusively dominate (lock) an input/output device. Any other program attempting to use or lock a locked LU will be suspended until the original program unlocks the LU or terminates.

## RTE-IV LIBRARY SUBROUTINES

### NO ABORT BIT

The no abort bit is used to alter the error return point of this call as shown in the following example:

```
CALL LURQ(IOPTN...)  
GO TO error routine  
    normal return point
```

The above special error return is established by setting the 'x' in IOPTN to 4 which sets bit 14.

This causes the system to execute the GO TO statement following the CALL LURQ if there is an error, or to skip the GO TO statement if there is no error.

### UNLOCK

To unlock all owned LUs, the LUARY array is not used but still must be coded; the program will not abort.

Any LUS the program has locked will be unlocked when the program:

1. Performs a standard termination
2. Performs a serial reusability termination.
3. Aborts

Note that LUS will not be unlocked when the program performs a 'save resources' termination.

This subroutine calls the program management subroutine (RNRQ) for a resource number (RN) allocation; that is, the system locks an RN locally to the calling program. Therefore, before the logical unit lock subroutine can be used, a resource number must have been defined during generation. Only the first 31 RNS can be used for LU locks.

If the no-wait option is coded, the A-register will contain the following information on return:

- 0 - LU lock successful
- 1 - no RN available at this time
- 1 - one or more of the LUs is already locked.

Note that the calling program may not have LUs locked at the time of the call unless the no-wait option is used. All LUs locked by the calling program are locked to the same RN.

#### 10-15. \$PARS (Parse)

Allows a program to parse an ASCII string.

The calling sequence is:

```

LDA  IBUFA   Buffer address
LDB  ICOUN   Character count
EXT  $PARS
JSB  $PARS
DEF  IRBUF
    -return-

```

where IRBUF is 33 words long. The result of the parse of the ASCII string at IBUFA is stored in IRBUF using 4 words per parameter that are set as follows:

WORD ----	ENTRY -----	
1	FLAG WORD	0 = NULL 1 = NUMERIC 2 = ASCII
2	VALUE(1)	0 If NULL; Value if Numeric; first 2 characters if ASCII.
3	VALUE(2)	0 If NULL or numeric else the 3rd and 4th characters.
4	VALUE(3)	0 If NULL or numeric else the 5th and 6th characters.

ASCII parameters are separated from numeric parameters by examination of each character. One or more non-digit characters (except a trailing "B" or leading "-") makes a parameter ASCII. This subroutine can parse up to eight parameters.

IRBUF is initialized to 0 before parsing the string IBUFA.

The 33rd word of IRBUF will be set to the number of parameters in the string.

## RTE-IV LIBRARY SUBROUTINES

The Parse routine ignores all blanks and uses commas to delimit parameters. ASCII parameters are padded to six characters with blanks or, if more than 6 characters, the left most 6 are kept. Numbers may be negative (leading "-") and/or octal (trailing "B").

FORTTRAN interface with \$PARS is provided with the following calling sequence:

```
CALL PARSE (IBUFA,ICONN,IRBUF)
```

where the parameters are as described for the Assembly Language call above.

### 10-16. INPRS (Buffer Conversion)

This routine converts a buffer of data back into its original ASCII form. The user passes the routine a buffer (IRBUF), plus the number of parameters in the buffer, that looks like the buffer returned by the PARSE routine. INPRS then reformats the buffer into an ASCII string that is syntactically equivalent (under the rules of PARSE) to a buffer that may have been passed to PARSE to form IRBUF. The length of the ASCII string in characters will be eight times the number of parameters. The FORTRAN calling sequence is:

```
CALL INPRS(IRBUF,IRBUF(33))
```

where:

IRBUF is the buffer containing the parsed string  
IRBUF(33) is the number of parameters parsed

### 10-17. \$CVT3,\$CVT1,CNUMD,CNUM0,KCVT (Binary to ASCII Conversion Subroutines)

Converts a positive integer binary number to ASCII.

The calling sequence is:

```
LDA numb  
CLE or CCE (see text)  
EXT $CVT3  
JSB $CVT3  
-return-
```

Upon return:

E-register=1  
A-register=address of result  
B-register=value at invocation

\$CVT3 converts a positive binary number in the A-Register to ASCII, suppressing leading zeros, in either OCTAL (E=0) or decimal (E=1). On return, the A-Register contains the address of a three word array containing the resultant ASCII string.

\$CVT1 has the same calling sequence as \$CVT3 except that on return, the A-Register contains the least-two characters of the converted number. The number to be converted must be positive.

The FORTRAN interface with \$CVT3 is provided by the following calling sequence:

```

                DIMENSION IARRAY(3)
(decimal) CALL CNUMD (binary numb,IARRAY)
(octal)   CALL CNUMO (binary numb,IARRAY)

```

where binary numb is the positive binary number to be converted and IARRAY is a three word array (6 ASCII characters). Leading zeros are suppressed.

The following subroutine converts a positive number to ASCII base 10 and returns the least two digits in "I". The FORTRAN calling sequence is:

```
I=KCVT(J)
```

#### 10-18. MESSS (Message Processor Interface)

Processes all operator commands (see Section III).

The FORTRAN call to the system message processor is provided by the following calling sequence:

```
I = MESSS (IBUFA,ICOUN,LU)
                --
```

where IBUFA contains the ASCII command. ICOUN is an integer containing the character count. LU is optional.

The value on return will be zero if there is no response, or the negative of the character count if there is a message. Any message will be in IBUFA.

If the request is RU or ON (starting in first column) and the first parameter is zero or absent, then the first parameter will be replaced by LU. LU is optional. If it is not supplied, no replacement occurs.

## RTE-IV LIBRARY SUBROUTINES

### 10-19. EQLU (Interrupting LU Query)

A calling sequence is provided to find the Logical Unit number of an interrupting device from the address of word four of its equipment table entry. The address of word 4 is placed in the B-Register by the driver and used in the following sequence:

```
LDB EQT4
```

This is not necessary if the address of EQT4 has already been placed into the B-register by the driver or by another program/subroutine.

The Assembly Language calling sequence is:

```
EXT EQLU
JSB EQLU
DEF RTN
DEF LUSDI (optional)
RTN return point
```

EQLU will return with:

A-Register = 0 if an LU referring to the EQT was not found.  
            = LU if the LU was found.

B-Register = ASCII "00" or the LU number in ASCII e.g. "16"

LUSDI = (optional parameter) value is returned to this parameter as well as in the A-Register.

This subroutine may be called from FORTRAN using the following calling sequence:

```
LU=EQLU (LUSDI)
```

### 10-20. PRTN,PRTM (Parameter Return)

These two routines are used to pass parameters to the program that scheduled the caller with wait. The scheduling program may recover these parameters with RMPAR.

The PRTN routine passes five parameters and clears the wait flag. This means that the caller should terminate immediately after the call.

The Assembly Language calling sequence is:

```

    EXT EXEC,PRTN
    JSB PRTN
    DEF *+2
    DEF IPRAM
    JSB EXEC
    DEF *+2
    DEF SIX
    .
    .
    .
    IPRAM BSS 5           Parameter buffer
    SIX   DEC 6           Program termination code

```

Note that the Program Termination EXEC call must immediately follow the PRTN call.

The FORTRAN calling sequence is:

```

    DIMENSION IPRAM(5)
    .
    .
    .
    CALL PRTN(IPRAM)
    CALL EXEC(6)

```

The PRTM routine passes four parameters and does not clear the wait flag. When the parameters are recovered with RMPAR, the first parameter is meaningless.

The Assembly Language calling sequence is:

```

    EXT PRTM
    JSB PRTM
    DEF *+2
    DEF IPRAM
    .
    .
    .
    IPRAM BSS 4

```



## RTE-IV LIBRARY SUBROUTINES

### 10-21. .DRCT (Indirect Address Subroutine)

Resolves an indirect address within the calling program's map.

The Assembly Language calling sequence is:

```
EXT .DRCT
JSB .DRCT
DEF ADDR
-return-
```

The routine returns with the A-Register set to the direct address of ADDR, the B-Register unaltered, and the E-Register lost. This routine is usually used when ADDR is external.

### 10-22. IFBRK (Breakflag Test)

This routine tests the break flag and clears it if it is set. See the BK command in Section IX.

The FORTRAN calling sequence is:

```
IF (IFBRK(IDMY)) 10,20
```

where:

10 = branch taken if the break flag is set. The flag will be cleared.

20 = branch taken if the break flag is not set.

IDMY must be used to inform the FORTRAN compiler that an external function is being called.

The Assembly Language calling sequence is:

```
JSB IFBRK
DEF *+1
-return-
```

The A-Register will = -1 if the break flag is set and =0 if not. The break flag will always be cleared if set.

## 10-23. COR.A, COR.B (First Word Available Memory)

COR.A finds the address of the first word of available memory for a given ID segment.

The Assembly Language calling sequence is:

```
EXT COR.A
LDA IDSEG
JSB COR.A
-return-
```

The ID segment address is loaded into the A-Register and the routine is called. On return the A-Register contains the first word of available memory (MEM2 from ID). Note that on entry into a segment, the A-Register contains the segment's ID segment address.

COR.B finds the high address +1 (first word of available memory) for main programs. This address is the same as that returned by COR.A for non-segmented programs. For segmented programs, this address is the high address + 1 of the largest segment. The ID segment address of only a main program must be passed to COR.B in the A-register.

The Assembly Language calling sequence is:

```
EXT COR.B
LDA IDSEG      ID segment address of a main program
JSB COR.B
-return-
```

Upon return:

A-register = 0 if normal return -1 if an error return, the B  
= register is meaningless

B-register = high address of main program (if it is not  
segmented) or the largest segment +1.

COR.B makes an error return if the ID segment address passed to it is that of a short ID segment (i.e., a segmented program).

RTE-IV LIBRARY SUBROUTINES

10-24. IDGET (Retrieve Program's ID Segment Address)

Retrieves the ID segment address of a specified program.

The FORTRAN calling sequence is:

```
IDSEG = IDGET(NAME)
```

where:

IDSEG will be set by the subroutine to the referenced program's ID segment or to 0 if the program does not exist.

NAME is a three-word (five-character) buffer with the program name in it.

The Assembly Language calling sequence is:

```
JSB IDGET
DEF *+2
DEF NAME
.
.
.
NAME ASC 3,PROG          Set aside three words of storage
                          containing ASCII equivalent of
                          PROGbb.
```

On return, the following registers are set as indicated:

A-register = ID segment address, or 0 if not found

E-register = 0 if program found, or 1 if not found

B-register = 0

## 10-25. TMVAL (Current Time)

Reformats and returns the time in milliseconds, seconds, minutes, hours, and the day.

The FORTRAN calling sequence is:

```
CALL TMVAL(ITM,ITMAR)
```

where:

ITM is the two-word negative time in tens of milliseconds. This double-word integer can be obtained from the system entry point \$TIME or the time values in the ID segment.

ITMAR is a five-word array to receive the time. The array is set up as:

```
tens of milliseconds
seconds
minutes
hours
current system day of year (not related to call values)
```

## 10-26. GETST (Recover Parameter String)

The routine GETST recovers the parameter string from a program's command string storage area. The parameter string is defined as all the characters following the second comma in the command string (third comma if the first two characters in the first parameter are NO).

The Assembly Language calling sequence is:

```
EXT GETST
.
.
.
JSB GETST      Call to subroutine
DEF RTN        Return address
DEF IBUFR      Buffer Location
DEF IBUFL      Buffer Length
DEF ILOG       Transmission Log
RTN            return point    Continue execution
.
.
.
IBUFR BSS n      Buffer of n words
IBUFL DEC n (or -2n) Same n; words (+) or characters (-)
ILOG  NOP        Error information
```

## RTE-IV LIBRARY SUBROUTINES

Upon return, ILOG contains a positive integer giving the number of words (or characters) transmitted. The A- and B-Registers may be modified by GETST. Note that if RMPAR is used, it must be called before GETST.

When an odd number of characters is specified, an extra space is transmitted in the right half of the last word.

This subroutine performs a function similar to an EXEC 14 call.

### 10-27. IFTTY (Logical Unit is or is Not Interactive)

Ascertains whether a logical unit is interactive or not.

The calling sequence in Assembly Language is:

```
EXT IFTTY
JSB IFTTY
DEF RTN
DEF LU      Logical unit being tested
RTN return point
```

The FORTRAN IV calling sequence is:

```
INT=IFTTY(LU)
```

where LU is the logical unit being tested.

Upon return:

```
INT=A-register = -1 if logical unit LU is interactive
                = 0 if logical unit LU is non-interactive
                / = upper byte is the driver type (word 5 of EQT
B-register<    table entry, bits 8-13)
                \ = lower byte is the subchannel number
```

### 10-28. LOGLU (Returns LU of Terminal that Scheduled Program)

LOGLU is a subroutine that returns the logical unit number (LU) of the terminal at which the currently executing program was scheduled.

The calling sequence in Assembly Language is:

```

EXT LOGLU
JSB LOGLU
DEF RTN
DEF IDUMY
RTN return point

```

The calling sequence in RTE Fortran IV is:

```

LU=LOGLU(ISES)

```

Upon return:

```

LU=A-register = LU number of device at which program was scheduled
B-register    = ASCII LU number
ISES          = This word will be modified by the subroutine, its
                value is reserved for future use by HP.

```

Comments:

Note that LOGLU must be called as a function. LOGLU will return the LU number of the console from which the currently executing program was scheduled. This LU number is passed down from the Father program to the Son program when one program schedules another program for execution. If the program was scheduled by interrupt or from the time list, the scheduling LU will be LU 1, the system console.

10-29. .EMAP, .EMIO, MMAP, EMAST (Extended Memory Area (EMA))s

The subroutines .EMAP, .EMIO, MMAP, and EMAST are system library subroutines that handle Extended Memory Areas. A complete description of these subroutines is provided in the Memory Management section of this manual.

DBUGR is a Hewlett-Packard utility subroutine used to interactively check programs for logical errors during execution. Using DBUGR, the user may examine and modify memory, examine and modify registers, set a breakpoint and trace instruction execution. DBUGR can only be used with consoles using drivers DVR00 and DVR05. Multipointed consoles using DVR07 will not work using DBUGR. In the following discussion, only the most frequently used DBUGR functions are described; refer to the RTE-IV DBUGR Reference Manual for the complete range of DBUGR capabilities.

#### 11-1. CALLING DBUGR

DBUGR can be automatically appended to a program at load time by calling the LOADR with the following command parameters:

```
*RU,LOADR,,filename,,DB
```

where DB instructs the LOADR to append DBUGR onto the relocatable code in file filename. Refer to the LOADR section in this manual for more information on the LOADR parameters. This command will also handle segmented programs, though there are some special procedures involving breakpoints in segmented programs. These are explained in the section on breakpoints.

When a program with appended DBUGR is subsequently run with the command:

```
*RU,program
```

DBUGR will be entered and the user will be able to give any legal DBUGR command. DBUGR calls the system subroutine LOGLU to obtain the logical unit from which the program was scheduled. It then uses this logical unit for all I/O. Refer to the Multi-Terminal Monitor section in this manual for more information.

## DBUGR INTERACTIVE DEBUGGING

DBUGR is also callable from Assembly Language and FORTRAN programs. The Assembly Language calling sequence is:

```
NAM prog
EXT DBUGR
.
.
.
JSB DBUGR          call to DBUGR
DEF RTN            address of return point
DEF LU             optional pointer to LU number

RTN -return point-
.
.
LU  BSS 1          interactive LU DBUGR will use for I/O
```

The FORTRAN calling sequence is:

```
CALL DBUGR(LU)
```

or

```
CALL DBUGR
```

according to whether the optional LU is passed in as a parameter.

In either Assembly Language or FORTRAN, if the optional LU is not passed in, DBUGR calls the system library subroutine LOGLU to determine the interactive LU to use for I/O. LOGLU returns to DBUGR the LU number of the user's interactive log device. If none exists, LU number 1 is returned specifying that the system console is to be used.

### 11-2. ENTERING DBUGR

When DBUGR is entered, it prints the following message on the appropriate LU:

```
START DBUGR
```

The user is now conversing with DBUGR and any legal command may be entered.

All DBUGR operations are conducted at the assembly language level. A load map and an Assembly language listing of the program is essential. An assembly language listing of the program is also necessary if debugging a program written in a high level language.



11-3. DBUGR COMMANDS

The following paragraphs give a concise explanation of the main features of DBUGR. Throughout these paragraphs, the conventions described in Table 11-1 apply. DBUGR supports the RUBOUT key but not the backspace key for deleting a typing mistake.

Table 11-1. DBUGR Command Conventions

SYMBOL	MEANING
\	Escape key (altmode key)
-	current position of the cursor
[CR]	carriage return
[LF]	line feed (control-J on some terminals)
<i>italics</i>	words and numbers to be supplied by the user

11-4. DBUGR MODES

DBUGR operates in one of four modes - symbolic, constant, ASCII, or address. DBUGR uses symbolic mode when it is first entered.

In symbolic mode, the contents of memory are inverse-assembled and displayed as an opcode and a memory reference (if it is a memory reference instruction). The user types "escape S" to enter symbolic mode as follows:

\S -

## DBUGR INTERACTIVE DEBUGGING

In constant mode, the contents of memory are displayed as octal constants. The user types "escape C" to enter constant mode as follows:

\C -

In ASCII mode, the contents of memory are displayed as two ASCII characters. The user types "escape H" to enter ASCII mode as follows:

\H -

In address mode, the contents of memory are displayed as an offset to a previously defined label. DBUGR will use any label that precedes the the contents by less than octal 11, or any single character label otherwise. The user types "escape A" to enter address mode as follows:

\A -

When DBUGR is in a particular mode, the mode can be temporarily switched when examining a memory location. The contents of the memory location will then be immediately displayed again in the temporary mode. With the cursor still on the displayed line of the memory location being examined, type one of the following symbols to temporarily enter the particular mode desired:

!	exclamation point - temporary symbolic mode
=	equals sign - temporary constant mode
'	single quote - temporary ASCII mode
-	underscore - temporary address mode

### 11-5. EXPRESSIONS AND TERMS

Expressions are used to specify memory locations to be examined. An expression consists of one or more terms combined with operators as in the following example:

AA+10

A term may be a previously defined symbol, a number, or certain special symbols preceded by an escape key (denoted in the text by a reverse slash (\)). The following examples are all terms:

ABC  
SYMBOL  
-32768  
1005  
\M

Legal operators are the following:

+	plus operator
blank	alternate plus operator
-	subtract operator
,	comma - inclusive or

11-6. EXAMINE MEMORY

To examine the contents of a memory location, simply type in an expression that evaluates to the memory location to be examined followed by a delimiting slash (/). For example, one way to examine memory location 50234 is:

50232+2/

DBUGR will print out on the same line the contents of the specified memory location in either octal or symbolic form. The example above might display:

50232+2/ LDA 50277 -

informing the user that location 50234 contains an LDA instruction referencing memory location 50277.

To examine the next sequential memory location, simply press the line feed (LF) key or control J. Continuing the above example, an LF is used to display the contents of memory location 50235:

50232+2/ LDA 50277 [LF] 50235/ ADA 50400 -

## DBUGR INTERACTIVE DEBUGGING

### 11-7. MODIFY MEMORY

To modify the contents of a memory location, the user must first open the memory location by examining it. After DBUGR displays the contents of the memory location, it is ready to insert new contents into the memory location examined. If an assembly language instruction is now typed in, DBUGR will assemble it and insert it into the memory location. If an octal constant is entered, DBUGR will insert it directly into the memory location. For example, to modify the contents of location 50234:

```
50234/ LDA 50277      CCA[CR]Display location 50234, change to
                        CCA instruction
50234/ CCA            [LF]  Display new contents of 50234, use
                        line feed to examine 50235
50235/ ADA 50400      100[CR]Change contents of 50235 to 100 octal
50235/ 100           _      Display new contents of location 50235
```

### 11-8. EXAMINE REGISTERS

The A and B registers are addressed as memory locations 0 and 1, respectively. The overflow register, the extend register, and the X and Y registers require special procedures for examination.

Memory location M+1 may be thought of as containing the overflow register and the extend register, each of which is one bit in length. These bits may be examined by typing "escape M+1/" as follows:

\M+1/

DBUGR will respond on the same line with an octal digit between 0 and 3 that is the status word. This octal digit may be broken down into two binary bits (EO) which are interpreted as follows:

```
E (bit 1 of \M+1) = 0 extend register is clear
                    1 extend register is set
O (bit 0 of \M+1) = 0 overflow register is clear
                    1 overflow register is set
```

The user may modify these bits immediately after examining them by typing in the new octal digit to replace the status word.

Memory locations M+3 and M+4 may be thought of as containing the X and Y registers. The X-register may be examined by typing "escape M+3/" as follows:

\M+3/

The Y-register may be examined by typing "escape M+4/" as follows:

\M+4/

DBUGR prints out the contents of the X or Y registers on the same line. They may then be modified if desired. Note that the X and Y registers are a full 16 bits wide. For example:

0/	000010	[CR]	user types 0/ to examine A-register
\M+1/	7	6[CR]	user clears the overflow register
\M+3/	677	0[CR]	examine and clear the X-register
\M+4/	50	-1[CR]	change the Y-register from octal 50 to 177777 (two's complement of -1)

#### 11-9. SETTING A LABEL

DBUGR can reference memory locations relative to a label. A label consists of one to six alphanumeric characters, the first of which must be alphabetic. To equate a label to a particular memory location, the user must first examine the memory location. After DBUGR has displayed the contents of the memory location, the label is entered followed by a colon (:). DBUGR then equates the label with the examined address. For example, the label S is equated with memory location 50234 as follows:

50234/ LDA 50277 S: [CR]

Location 50237 may now be referenced by typing:

S+3/

#### 11-10. EXECUTE PROGRAM

To proceed with execution of the user program when DBUGR has control, the user types "escape P":

\P

Upon initial entry to DBUGR, execution proceeds at the transfer address of the program. When a breakpoint is encountered, execution resumes at the instruction where the breakpoint was set.

## DBUGR INTERACTIVE DEBUGGING

When proceeding from a breakpoint, the user has the option of typing:

n\P

DBUGR will then execute the breakpoint octal n times before it will break at it.

If the proceed instruction is given and there is no breakpoint in the program, DBUGR displays the following message before control returns to the executing program:

END DBUGR

The user may instruct DBUGR where to resume execution of the program by typing the address of the instruction to be executed, followed by "escape G". For example, to resume program execution at location 50234, type:

50234\G

### 11-11. BREAKPOINTS

When an instruction with a breakpoint is encountered, control is transferred to DBUGR immediately prior to the execution of the instruction with the breakpoint. DBUGR displays information about the state of the machine, and the user may then enter any legal DBUGR command.

A breakpoint is set at an address by entering the octal address followed by "escape B". For example, to set a breakpoint at 50234, type:

50234\B

Only one breakpoint is allowed at a time.

A breakpoint that has been set is cleared either by resetting it to a new memory location, or by typing "escape B" at the beginning of a line:

\B

If the executing program reaches a breakpoint, control returns to DBUGR. DBUGR then displays the following information about the state of the machine:

ADDRESS(INSTRUCTION) A-REG B-REG X-REG Y-REG STATUS -

where:

ADDRESS is the address of the breakpoint

INSTRUCTION is the contents of the ADDRESS

A-REG,B-REG,X-REG,Y-REG are the contents of the registers

STATUS is the status of the extend and overflow bits  
as explained in the section on examining registers

For example:

```

50234\B          set breakpoint at 50234
\P             proceed with execution
50234(LDA 50277) 77 11 177776 3 3      \P
                breakpoint information displayed, user types \P to proceed
50234(LDA 50277) 77  0 177776 3 3      [CR]
                breakpoint encountered again; B-REG has changed to 0
1/           0          11[CR]         change B-REG to octal 11
\P             proceed
    
```

When a segmented program has been loaded with the command:

```
*RU,LOADR,,filename,,DB
```

use the following commands to control the setting of breakpoints within segments:

```

["A]\B        break at entry to all segments
["N]\B        break at entry to no segments
[seg]\B       break at entry to seg
    
```

## DBUGR INTERACTIVE DEBUGGING

To set a breakpoint within a segment, enter the following command:

```
addr[seg]\B
```

where:

addr is the address within the segment at which the breakpoint is set.

seg is the name of the segment in which the breakpoint is set.

The breakpoint will be set when the segment is loaded into memory. Therefore the current breakpoint will remain in effect until the segment is loaded. If seg is in memory at the time that the segment break command is entered, the current breakpoint is cleared immediately.

When a segment load clears a breakpoint, DBUGR will break at the start of the new segment and print the following message:

```
SEGMENT seg BREAK  
--BREAKPOINT INFORMATION--  
addr BREAKPOINT REMOVED
```

where:

seg is the name of the new segment

BREAKPOINT INFORMATION is the normal breakpoint information

addr is the address at which the old breakpoint was removed

DBUGR does not check the validity of the segment name. The segment name may not begin with the two characters quote A ("A) or quote N ("N). This is to avoid confusion in setting the breaks in segment entry points as explained above.

DBUGR will not allow breakpoints below the memory protect fence or outside the user's partition. An attempt to set such a breakpoint will cause a memory protect ("MP?") or a dynamic mapping ("DM?") error message to be printed.

There are certain legal instructions that DBUGR cannot execute without causing memory protect (MP) or dynamic mapping (DM) errors. The instructions "JSB EXEC" and "JSB \$LIBR" are two typical examples. When such a situation arises, DBUGR will not allow execution of the instruction, and prints out a message of "DM?" or "MP?" depending on the error that execution of the instruction would cause. To execute the instruction, simply move the breakpoint and proceed.



11-12. TRACING

When DBUGR has control, the instructions of a program can be traced (single-stepped) by typing "escape T". After each instruction is executed, the same information about the state of the machine will be displayed as after a breakpoint. For example:

```

50234\B                set a breakpoint at 50234
\P                    proceed
50234(LDA 50277)  77  11  177776  3  7      \T
                    breakpoint information displayed, start trace
50235(ADA 50101) 100  11  177776  3  7      \T
                    breakpoint information displayed, continue trace
50236(LDB 50282) 107  11  177776  3  7      -
                    .
                    .
                    .
    
```

A specified number of instructions can also be traced by specifying an octal number before the trace command. Type:

n\T

to trace octal n instructions and halt.

When DBUGR attempts to trace an instruction that will cause a memory protect or dynamic mapping violation, an "MP?" or "DM?" error will be printed. If the instruction is legal, put a breakpoint on the instruction to which control will return and then proceed.

Note: Privileged routines (see the RTE-IV Subroutines Library, Section X) cannot be traced.

DBUGR INTERACTIVE DEBUGGING

11-13. DBUGR ERROR MESSAGES

DBUGR recognizes certain errors and prints an error message. Table 11-2 lists the errors and their meanings.

Table 11-2. DBUGR Error Messages

Error	Meaning
X	The user pressed the RUBOUT key to erase a typing mistake DBUGR ignores any prior partial expression.
?	The user entered an unassigned control. Any prior expression is ignored.
U	The symbol last used is undefined, and a definition is required. The entire preceding expression is ignored.
P?	Page error. A memory reference instruction referenced an address not in the current page or the base page. The expression is ignored. DBUGR's conception of the "current page" can be changed by examining any location in the desired page.
MP?	There is a breakpoint or trace set for an instruction that if executed by DBUGR would cause a memory-protect violation to occur. Move the breakpoint and proceed.
IN?	There is a breakpoint or trace set for an instruction from which DBUGR cannot proceed. Move the breakpoint and proceed.
DM?	DBUGR is attempting to access a memory location that is not within the user's partition.
TP?	DBUGR is attempting to overload, trace, or set a breakpoint within DBUGR.

11-14. DBUGR EXAMPLE

The following example demonstrates a typical session with DBUGR.

```

*RU,PROG      (Execute program loaded with DBUGR.)

START DBUGR

16002/  CCA      M:      examine location 16002 in the main
                        program; equate 16002 to M.
23456/  NOP      S:      examine location 23456 in the
                        segment; equate 23456 to S.
S+5\B        use escape B to set a breakpoint
\P          and proceed

SEGMENT SEG1 BREAK      since a breakpoint was removed,
S (0) 17542 5608 17702 22 6 a break is executed upon entry
S+5 BREAKPOINT REMOVED to the segment

S+5[SEG2]\B      set a breakpoint within SEG2
\P          proceed

SEGMENT SEG2 BREAK      break at S+5 in SEG2.
S+5 (0) 17542 5606 45 22 4

M+50\B        set a breakpoint within the main
S+10[SEG4]\B      set a future breakpoint in SEG4
\P          proceed

M+50 (LDA M+700) 0 2234 54 72 5 break in main

M 700/  ALF,ALF = 1727 1777[LF] examine location M+700, temporary
                        octal display,change contents
                        to 1777
M+701/  0      [CR]      next location automatically displayed

M+700/  ALF,CLE,SLA,ALF [CR] re-examine location M+700

2\T          trace two instructions

M+50 (LDA M+700) 0 2234 54 72 5 breakpoint instruction is executed

M+51 (STA M+701) 1777 2234 54 72 5 \P next instruction is
                        executed; proceed with execution
    
```

DEBUGR INTERACTIVE DEBUGGING

SEGMENT SEG4 BREAK  
S (0) 17445 5562 7422 3322 5  
M+50 BREAKPOINT REMOVED

a segment breakpoint was removed, so  
break upon entry to the segment

\P

S+10 (JSB 112,I) 24 0 177777 55 6 Break at S+10 in SEG4.

["N]\B

clear segment breakpoint

\P

proceed

END DEBUGR

The ability to reconfigure the I/O and memory assignments during system boot-up without going through a complete, new system generation is a feature of the RTE-IV operating system. The reconfiguration option is exercised during system boot-up through S-register settings (described below) in order to postpone completion of the boot-up process and schedule an interactive Configurator program that performs the desired I/O and/or memory reconfiguration.

I/O reconfiguration is performed by user resassignment of I/O devices to octal select codes other than those assigned at system generation time.

Memory reconfiguration includes changing the size of the System Available Memory (SAM) extension, redefining user partitions, modifying program page requirements and assigning programs to partitions. Defective pages in memory (pages with parity errors) can be avoided by using the Configurator to redefine the SAM extension and user partitions around the defective pages.

I/O and memory reconfigurations (either or both) can be made permanent by changing the system on the disc.

#### 12-1. SCHEDULING THE CONFIGURATOR FROM DISC LOADER ROM

If a disc loader ROM is used to load the boot-extension into memory during system boot-up, the Configurator can be scheduled by setting bit 5 of the S-register, in addition to the S-register settings for the disc loader ROM. The example given below assumes the system boot-up will be performed using the 12992B RPL-compatible 7905/7906/7920 Disc Loader ROM, and that the Boot Extension resides on physical track 0, sector 0 of the system disc.

## MEMORY AND I/O RECONFIGURATION

Begin the boot-up by performing the following steps:

1. Select the S-register for display on the computer front panel.
2. Press CLEAR DISPLAY
3. Set the S-register bits for the disc loader ROM. In addition, set bit 5 of the S-register for I/O or memory reconfiguration:

Bits ----	Enter -----
0-2	Surface number of the disc where the RTE-IV system subchannel starts (surface numbers start at 0).
3-4	0 (reserved).
5	1 to specify reconfiguration is to be performed. A HLT 77B will be issued at the end of the load.
6-11	Octal select code of the disc.
12	1 to indicate a manual boot from the S-register.
13	0 (reserved).
14-15	Loader ROM selection (number of the ROM cell containing the Disc Boot Loader).

4. Press PRESET, IBL, PRESET (again) and RUN to load the contents of the Disc Loader ROM. A successful load will be indicated when the HLT 77B occurs.

5. Following the HLT 77B, set the S-register as follows:

Bits ----	Enter -----
0-5	System console octal select code if either the select code or device type is different from generation specification; otherwise, 0.
6-11	System disc octal select code if different from generation specification; otherwise, 0.
12-14	0 (reserved)
15	1 to specify reconfiguration of I/O (including disc and console, above) and/or memory assignments.

6. Press RUN to perform reconfiguration processes.

#### 12-2. SCHEDULING THE CONFIGURATOR FROM BOOTSTRAP LOADER

If the Bootstrap Loader is used to load the Boot Extension into memory, set the S-register as described above in Step 5 when the HLT 77B occurs.

Set the P-register to 100 octal and press RUN to perform reconfiguration.

#### 12-3. CONFIGURATOR PROGRAM

The Configurator works interactively with the user to make specified changes to the current I/O and memory configurations. Reconfiguration is performed in accordance with user responses to a series of Configurator prompts and queries output on the system console. When reconfiguration is completed, the Configurator queries whether it is to be made permanent. Boot-up of the RTE-IV system is then completed in accordance with the user's reply.

The Configurator is divided into two programs: \$CNFG and \$CNFX. \$CNFG is a module located at the end of the system modules. After configuration has completed, the memory area occupied by \$CNFG is allocated to SAM. \$CNFX is used to reconfigure memory and is a Type 3 disc resident program, brought into the user partition area from disc by the \$CNFG program. \$CNFG changes \$CNFX's program name to "....." and therefore \$CNFX cannot be executed on-line.

## MEMORY AND I/O RECONFIGURATION

The Configurator program first checks the contents of the S-register. If bit 15 is set, I/O and memory reconfiguration are performed. The system is reconfigured in accordance with any specified new disc and console select codes. Entering invalid disc and console select codes in the S-register will cause the system not to function properly. The Configurator then loads the driver partitions, memory resident library and memory resident programs (if they are defined for the system) into memory.

If bit 15 is not set in the S-register, control is given to the operating system.

Reconfiguration is performed interactively by using the system console and list device. Note that the standard method of getting system attention by pressing any key on the system console will not work during reconfiguration, since the system is not yet completely initialized. The bootup procedure must therefore be restarted if any equipment I/O errors occur (e.g., a device not ready or a parity error).

### 12-4. CONFIGURATOR HALTS AND ERROR MESSAGES

Various halts and Configurator error messages may occur during system boot-up or reconfiguration that require corrective action by the operator. Halts are displayed on the computer front panel. System boot-up and configuration HLTs, their meaning and required operator action are itemized in Table 12-1 at the end of this section.

Whenever the user enters an invalid response to a Configurator prompt or query, the Configurator will issue an error message in the form

CONFIG ERR xx

where xx is a Configurator error code as defined in Table 12-2 at the end of this section. Following the error message, the Configurator will usually repeat the prompt or query and the user need only enter the correct response. In the reconfiguration procedures given below, only error recovery procedures requiring further action will be described in text.

### 12-5. RECONFIGURATION PROCEDURES

The Configurator begins the reconfiguration process by first displaying the message

START RECONFIGURATION

on the system console, and followed by a set of queries to which the user enters responses on the console keyboard. The Configurator will redisplay a query if the user response is not what was expected.



The Configurator next displays the query

LIST DEVICE LU#?

Enter a Logical Unit number to which the Configurator can direct listings or press the space bar and RETURN key on the console keyboard for the default case, which is the system console. Entering a list device other than the system console causes the Configurator to display the following message:

LIST DEVICE SELECT CODE#?

Enter a list device select code or press the space bar and RETURN key for the default case, where the default is the list device select code configured into the system.

If the entered list device was not the system console, the Configurator displays the query

ECHO? (YES/NO)

Enter YES to have all output to the list device echoed on the system console.

#### 12-6. I/O RECONFIGURATION STEPS

I/O reconfiguration is performed by assigning the Interrupt Table and trap cell values for the current select code to the corresponding entries for the new select code.

The Configurator first prompts for I/O reconfiguration by displaying a list of the current I/O configuration, beginning with octal select code 10 for the operating system, in the format:

CURRENT I/O CONFIGURATION:

			+	---		---		+
						PNAME		
SELECT CODE	xx =	TBG	[,	TYPE	nn			
		PRIV	I/O			nnnnnn		
					+	---		---

## MEMORY AND I/O RECONFIGURATION

where:

xx = octal select code number ranging from 10 to 77.

EQTy = EQT entry number

TBG = Time Base Generator

PRIV I/O = privileged I/O card

TYPE nn = equipment type code

PNAME = name of program to be automatically scheduled

nnnnn = absolute instruction to be executed upon interrupt; for example, a JSB LINK,I where LINK contains the entry point address.

The CURRENT I/O CONFIGURATION data is automatically displayed to provide a basis on which to make decisions regarding reconfiguration. If the system disc, system console or the list device were assigned to a new select code, they have already been configured in memory and must NOT be reconfigured during I/O reconfiguration.

The list does not include the select codes previously configured to the system disc, system console, or list device that have been reconfigured via the SWITCH register at bootup. However, these previously-occupied select codes are still available for reassignment. Also, those devices formerly occupying the select codes now reconfigured to the system disc, console, or list device may be reassigned if referenced by their old select code.

Following display of the current configuration, the Configurator then displays the query

I/O RECONFIGURATION?(YES/NO)

Enter NO to bypass I/O reconfiguration. The Configurator will skip all further I/O reconfiguration prompts and begin prompting for memory configuration entries (see below).

Enter YES if I/O is to be reconfigured. The Configurator program will then display the message

```
CURRENT SELECT CODE#,NEW SELECT CODE#?(/E TO END)
```

-

where the hyphen (-) prompts entry of the current and new select code pairs. The current and new select codes response must be in octal and must vary between 10 and 77 octal, in the form

```
xx,yy
```

followed by a carriage return, where xx is the current select code number and yy is the new select code number. The Configurator's hyphen prompt will be repeated after each successful entry until a /E is entered to terminate the list.

A privileged I/O card's assignment can be removed by entering the current select code number of the privileged I/O card followed by zero, in the form

```
xx,0
```

where select code 0 is only used to remove the privileged I/O card's assignment. A new value of 0 will be assigned to the privileged I/O card.

```
+-----+
|                CAUTION                |
| A privileged driver will not work      |
| correctly if the privileged I/O card   |
| has been removed from the system.     |
+-----+
```

A privileged I/O card can be added to a system that does not have one by entering the specification

```
xx,PI
```

where xx is the specified select code in octal, and PI assigns the privileged I/O card to select code xx.

If a /R is entered, I/O reconfiguration is restarted with display of the CURRENT SELECT CODE#, NEW SELECT CODE#?(/E TO END) query.

If the current select code number entry is repeated in more than one response, the last entry is taken as valid and the previous entries are ignored.

## MEMORY AND I/O RECONFIGURATION

Following entry of a /E to terminate select code changes, the Configurator prints a list of the NEW I/O CONFIGURATION. The next query displayed is:

NEW I/O CONFIGURATION PERMANENT?(YES/NO)

Enter YES to modify the system on the disc to the new I/O configuration. Enter NO otherwise. If it is desirable to restart I/O reconfiguration for any reason, enter the request

/R

and I/O reconfiguration will restart by another display of the list

CURRENT I/O RECONFIGURATION:

The list will contain what the I/O configuration was changed to during the reconfiguration just completed.

### CAUTIONS:

1. It is strongly recommended that the system subchannel of the disc be backed up before making I/O reconfiguration permanent.
2. If a select code has been given a new assignment and its current I/O device has not been reassigned, the I/O device cannot be added to the system at a later date if the new I/O configuration is made permanent.
3. If a device has multiple select codes, make sure that all select codes are moved and kept in the same relative order.
4. Reassigning some devices to empty I/O slots may cause unexpected results.

## 12-7. MEMORY RECONFIGURATION PROCEDURES

After the I/O reconfiguration phase is either bypassed or terminated, the Configurator will display the following statement and query:

CURRENT PHYSICAL MEM SIZE: xxxx PAGES  
MEM RECONFIGURATION?(YES/NO)

Enter NO if memory reconfiguration is not desired. The Configurator will then transfer control to the operating system after displaying the message

RECONFIGURATION COMPLETED

Enter YES if memory is to be reconfigured. The Configurator will then display the query

PHYSICAL MEM SIZE?(#PAGES)

Enter the desired total number of memory pages, between 48 and 1024 (decimal).

## 12-8. EXCLUDING BAD PAGES

The Configurator program can be used to redefine the SAM extension and user partitions to exclude any bad pages (pages containing parity errors) within these areas. Each user partition must be a contiguous block of memory; therefore, user partitions must be defined on blocks of memory between the bad pages. Bad pages in the system area, driver partitions and the memory resident area cannot be avoided.

The Configurator displays the query

DEFINE BAD PAGES BEGINNING AT PAGE xxxx (/E TO END)

-

where the hyphen (-) prompts for the decimal number of a bad memory page. The hyphen is repeated after acceptance of each entry until a /E or 100 bad page numbers are entered, terminating the list. (The Configurator will accept up to 100 bad memory page entries.) The bad page specifications entered can range from xxxx to the maximum page number in physical memory and must be entered in an increasing order.

If /R is entered in response to the hyphen prompt, the Configurator will redisplay the query

DEFINE BAD PAGES BEGINNING AT PAGE xxxx (/E TO END)

-

and the entire list of bad pages must be re-entered.

When a /E is entered either to terminate bad page entries or bypass the entire phase, the Configurator displays the following information:

CURRENT SIZE OF SAM DEFAULT: xxxxx WORDS EXTENSION: yy PAGES  
SAM EXTENSION STARTS AT PHYSICAL PAGE xx MAX PAGES AVAIL FOR  
SAM EXTENSION: xx

The number of words displayed for default SAM are the decimal number of words assigned to the first block of SAM.

## MEMORY AND I/O RECONFIGURATION

### 12-9. SAM EXTENSION RECONFIGURATION

The Configurator next prompts for any desired change in the size of SAM extension by displaying the query

```
CHANGE SAM EXTENSION?(# PAGES/" " CR)
```

Press the space bar and RETURN key (the default case) if no change is desired.

Enter the decimal number of pages desired if the SAM extension is to be changed. The number of pages can vary from 0 (which removes SAM extension) to the maximum pages available for the SAM extension. Note that this count must not include any bad pages that fall within the SAM extension (see above).

The Configurator sets up the System Map to avoid bad pages in the SAM extension regardless of whether or not a change was requested.

If the specified SAM extension extends beyond the size of physical memory because of bad pages within this area, the Configurator displays the message

```
CONFIG ERR 12  
CHANGE SAM EXTENSION?(# PAGES/" " CR)
```

Enter a smaller number of pages for SAM extension size. The Configurator allows SAM extension to be divided up into a maximum of five blocks of memory between bad pages. If the number of pages in SAM extension requires division into more than five blocks, the Configurator displays the message

```
CONFIG ERR 22
```

and the query is redisplayed. Enter a smaller size of SAM extension.

### 12-10. CHANGING PARTITION DEFINITIONS

The Configurator next displays a list of current partition definitions is displayed in the format

CURRENT PART'N DEFINITIONS:

```
PART'N nn = pp PAGES | +--  --+ +--  --+  
                      | | ,RT | | | | |
                      | | ,BG | | | ,R | |  
                      | | ,S  | | | | |  
                      +--  --+ +--  --+
```

where

nn = the partition number

pp = is the number of pages in partition nn

RT = a real-time partition

BG = a background partition

S = a subpartition

R = a reserved partition

Following the definition list, the Configurator next displays a list of current partition requirements in the form

CURRENT PART'N REQMTS:

REALTIME

PNAME XX PAGES [E][PART'N=nn]

.

.

.

BACKGROUND

PNAME XX PAGES [\*][E][PART'N=nn]

.

.

.

where

PNAME = the real-time or background program name

E = indicates an EMA (Extended Memory Area) program

\* = indicates the background program does not include Table Area I:  
(i.e., a Type 4 program)

nn = is the number of the partition into which program PNAME is assigned.

## MEMORY AND I/O RECONFIGURATION

The Configurator then displays the following information:

```
MAX PROGRAM SIZE:
W/OUT COMMON:   xx PAGES
W/COMMON:       xx PAGES
W/TABLE II:     xx PAGES
MAX # OF PART'NS:  xx
PAGES REMAINING:  xx
DEFINE PART'NS FOR xxxx PAGES
#PAGES,RT/BG/S(,R)
PART'N x?
```

where

MAX PROGRAM SIZE = maximum logical space a program may occupy. However, the partition size may be larger than the stated maximum if the partition will be used for EMA program execution.

MAX # OF PART'NS = decimal number of partitions that can be defined in memory.

PAGES REMAINING = decimal number of pages available for defining user partitions (including bad pages that may have been listed earlier).

#PAGES,RT/BG/S(,R) = indicates the required format for user entries in response to the PART'N x? prompt described below.

PART'N x? = Configurator program prompt asking the user for the size (in pages) and format for the next partition to be defined.

If the maximum number of partitions was defined as 0 during generation time, the Configurator skips the rest of memory reconfiguration and displays the query

NEW MEMORY CONFIGURATION PERMANENT?

Since partitions must be defined contiguously, they must be within the section of memory between the bad pages. If a section of memory between bad pages has a size of one page, it is skipped by the Configurator. The Configurator will prompt for a partition definition after each accepted entry until partitions have been defined for all xxxx pages in this section of memory.



As each entry is accepted, the Configurator will reissue the prompt with a consecutively increasing partition number for the next partition. If the number of pages entered for a partition is greater than the maximum logical address space, the Configurator displays the message

SUBPARTITIONS?(YES/NO)

Enter a NO if the configurator is to ignore subpartition considerations and proceed with the normal partition definitions.

Enter a YES if subpartitions are to be defined. Subpartition definitions are specified by using the following format in response to the prompt:

#PAGES,S(,R)

where S specifies a subpartition and the optional R specifies the subpartition is to be reserved.

The memory space allocated for subpartitions is the same area occupied by the "mother" partition. Subpartition definition will end as soon as an RT or BG partition is defined, or can be terminated by entering a /E.

When an attempt is made to end the subpartition definition phase by defining an RT or BG partition and there are no more pages left in this section of memory, an ERR 13 will be displayed. In this case, either enter a /E to terminate subpartition definitions and continue partition definitions for the next block of memory, or enter /R to restart the partition definition phase.

The total number of pages defined for subpartitions must not exceed the size of the mother partition or an error code will be issued and the last subpartition must be redefined.

The Configurator analyzes each partition definition for possible errors as soon as it is entered. Any error code issued will be followed by a prompt to redefine the last partition displayed. If /R is entered instead of a partition description, the partition definition phase is restarted from the first partition definition.

## MEMORY AND I/O RECONFIGURATION

Partitions defined for each section of memory between bad pages must be defined for all pages available within the section. A running total is maintained of the number of pages currently defined within a section of good memory. The Configurator will then take one of five possible courses of action, depending upon the prevailing memory structure and size:

1. If the remaining total equals the number of pages available, the Configurator automatically requests partition definitions for the next section of good memory.
2. If the number of pages remaining to be defined is one, the Configurator increments the last defined partition by one page and then requests partition definitions for the next block of good memory.
3. If the running total exceeds the number of available pages defined within the memory block, the Configurator displays an error message and prompts for the last partition to be redefined.
4. If the number of partitions already defined is equal to the maximum number of partitions allowed and more undefined good pages remain, the Configurator displays an error message and all user partitions must be redefined. The Configurator will then prompt for new partition definitions and repeat the prompt after each accepted entry.
5. If the running total is less than the number of pages in the block of memory, definition for next partition is requested.

A list of NEW PART'N DEFINITIONS will be issued to the list device when all partitions have been defined.

### 12-11. CHANGING PROGRAM PARTITION ASSIGNMENTS

The Configurator performs a check to ensure that every program assigned to a partition fits its partition size. A program will be unassigned if the program size is larger than the partition size or if the partition: number does not exist. Following the check, the Configurator will issue a list under the heading

UNASSIGNED PROGS

·  
·  
·

followed by the query

MODIFY PROG PAGE REQMTS?(/E TO END)  
PNAME,#PAGES

-

Enter the specifications for any disc resident programs whose page requirements must be changed, using the format

program name,xx

where the number of pages entered for each program must include the base page. The number of pages must be greater than or equal to the program relocation size, and less than or equal to the maximum address space for the program. The program may only be Type 2, 3 or 4.

The hyphen prompt will be repeated after acceptance of each entry until a /E is entered to terminate the list.

Note that the page requirements for an EMA program cannot be modified.

## 12-12. PROGRAM PARTITION ASSIGNMENTS

The Configurator now asks if any programs need to be assigned to partitions by displaying the query and prompt

```
ASSIGN PROG PART'NS?(/E TO END)
PNAME, PART'N#
-
```

where the hyphen prompt will be repeated after each accepted entry until a /E is entered to terminate the list.

Enter each desired program partition assignment in the form

program name,xx

where xx is the partition number to which the program is to be assigned. If xx is 0, the program is unassigned and can be dispatched to any partition of the proper type large enough to run the program. The program must be a Type 2, 3 or 4. When a /E is entered to terminate the list, the Configurator issues the query

```
NEW MEMORY CONFIGURATION PERMANENT? (YES/NO)
```

Enter a YES to a change the appropriate tables and locations on the disc resident system. The Configurator then issues the message

```
RECONFIGURATION COMPLETED
```

and turns control over to the operating system.

If a /R is entered in response to the prompt instead of YES, memory reconfiguration is restarted from the query

```
PHYSICAL MEM SIZE?(#PAGES)
```

and the system is in the state it was changed to during the earlier reconfiguration.

## MEMORY AND I/O RECONFIGURATION

### 12-13. RECONFIGURATION EXAMPLE

The sample reconfiguration illustrated in Figure 12-1 assumes that reconfiguration was requested by setting the switch register as described at the beginning of this section of the manual. In the example, the shaded portion identifies a user response.

```
START RECONFIGURATION
LIST DEVICE LU#?
20 *SPECIFY A LIST DEVICE.
LIST DEVICE SELECT CODE#?
20 *SPECIFY LIST DEVICE'S SELECT CODE.
ECHO? (YES/NO)
YES *ECHO OUTPUT ON LIST DEVICE.
CURRENT I/O CONFIGURATION: *CURRENT I/O CONFIGURATION
SELECT CODE 10= TBG * IS DISPLAYED.
SELECT CODE 13= EQT 1,TYPE 32
SELECT CODE 14= EQT 6,TYPE 0
SELECT CODE 15= EQT 7,TYPE 1
SELECT CODE 16= EQT 3,TYPE 23
SELECT CODE 17= EQT 3,TYPE 23
SELECT CODE 20= EQT 5,TYPE 12
SELECT CODE 22= EQT 4,TYPE 2
SELECT CODE 25= EQT 2,TYPE 5
I/O RECONFIGURATION? (YES/NO)
YES *SPECIFY I/O RECONFIGURATION.
CURRENT SELECT CODE#,NEW SELECT CODE#? (/E TO END)
-
10,14 *RECONFIGURE SELECT CODES.
-
14,15
-
15,16
-
16,23
-
17,24
-
22,17
-
/E
```

Figure 12-1. Reconfiguration Example

```

NEW I/O CONFIGURATION:                                *NEW I/O CONFIGURATION
SELECT CODE 13= EQT 1,TYPE 32                        * IS DISPLAYED.
SELECT CODE 14= TBG
SELECT CODE 15= EQT 6,TYPE 0
SELECT CODE 16= EQT 7,TYPE 1
SELECT CODE 17= EQT 4,TYPE 2
SELECT CODE 20= EQT 5,TYPE 12
SELECT CODE 23= EQT 3,TYPE 23
SELECT CODE 24= EQT 3,TYPE 23
SELECT CODE 25= EQT 2,TYPE 5
NEW I/O CONFIGURATION PERMANENT?(YES/NO)             *SPECIFY NONPERMANENT.
NO                                                    *SPECIFY MEMORY RECONFIGURATION.
CURRENT PHYSICAL MEM SIZE: 48 PAGES
MEM RECONFIGURATION?(YES/NO)
YES
PHYSICAL MEM SIZE?(#PAGES)                          *SPECIFY AN INCREASE IN MEMORY SIZE.
256
DEFINE BAD PAGES BEGINNING AT PAGE 28 (/E TO END)
-
44                                                    *SPECIFY TWO BAD PAGES.
-
124
-
/E
CURRENT SIZE OF SAM:
DEFAULT: 3802 WORDS
EXTENSION: 0 PAGES
SAM EXTENSION STARTS AT PHYSICAL PAGE 28
MAX PAGES AVAIL FOR SAM EXTENSION: 12
CHANGE SAM EXTENSION?(#PAGES/" "CR)
6                                                    *INCREASE SIZE OF SAM.
CURRENT PART'N DEFINITIONS:                          *CURRENT PARTITION DEFINITIONS
PART'N 1 = 20 PAGES,BG                              * ARE DISPLAYED.
CURRENT PART'N REQMTS:                              *CURRENT PARTITION REQUIREMENTS
REALTIME                                             * FOR VARIOUS PROGRAMS ARE
BACKGROUND                                           * DISPLAYED.
$CNFX 3 PAGES
EDITR 16 PAGES
ASMB 16 PAGES
XREF 16 PAGES
LOADR 16 PAGES
WHZAT 3 PAGES
FMGR 7 PAGES
RT4GN 20 PAGES
SWTCH 11 PAGES
SAVE 16 PAGES
RSTOR 16 PAGES
COPY 16 PAGES
VERFY 16 PAGES

```

Figure 12-1. Reconfiguration Example (continued)

MEMORY AND I/O RECONFIGURATION

MAX PROGRAM SIZE:  
W/OUT COMMON: 29 PAGES  
W/ COMMON: 29 PAGES  
W/ TABLE II: 27 PAGES  
MAX # OF PART'NS: 15  
PAGES REMAINING: 222  
DEFINE PART'NS FOR 10 PAGES:  
#PAGES,RT/BG/S(,R)  
PART'N 1?  
10,RT  
DEFINE PART'NS FOR 79 PAGES:  
#PAGES,RT/BG/S(,R)  
PART'N 2?  
49,RT  
SUBPARTITIONS?(YES/NO)  
NO  
PART'N 3?  
27,RT,R  
PART'N 4?  
3,RT,R  
DEFINE PART'NS FOR 131 PAGES:  
#PAGES,RT/BG/S(,R)  
PART'N 5?  
115,BG  
SUBPARTITIONS?(YES/NO)  
YES  
PART'N 6?  
48,S  
PART'N 7?  
29,S  
PART'N 8?  
29,S  
PART'N 9?  
9,S  
PART'N 10?  
16,BG  
NEW PART'N DEFINITIONS:  
PART'N 1 = 10 PAGES,RT  
PART'N 2 = 49 PAGES,RT  
PART'N 3 = 27 PAGES,RT,R  
PART'N 4 = 3 PAGES,RT,R  
PART'N 5 = 115 PAGES,BG  
PART'N 6 = 48 PAGES,S  
PART'N 7 = 29 PAGES,S  
PART'N 8 = 29 PAGES,S  
PART'N 9 = 9 PAGES,S  
PART'N 10 = 16 PAGES,BG

\*MAXIMUM PARTITION SIZES FOR  
\* VARIOUS PROGRAM TYPES ARE  
\* DISPLAYED.  
  
\*RT PARTITION TO THE FIRST BAD PAGE.  
  
\*RT PARTITION WITH NO SUBPARTITIONS.  
  
\*RT PARTITION WHICH IS RESERVED.  
\*RT PARTITION WHICH IS RESERVED.  
  
\*BG MOTHER PARTITION BEGINS  
\* AFTER SECOND BAD PAGE.  
  
\*SUBPARTITION LARGER THAN 32K WORDS.  
\*SECOND SUBPARTITION.  
\*THIRD SUBPARTITION.  
\*FOURTH SUBPARTITION.  
\*BG PARTITION.  
\*NEW PARTITION DEFINITIONS  
\* ARE DISPLAYED.

Figure 12-1. Reconfiguration Example (continued)

## MEMORY AND I/O RECONFIGURATION

```

UNASSIGNED PROGRAMS:
MODIFY PROG PAGE REQMTS?(/E TO END)
PNAME,#PAGES          *SPECIFY NEW PROGRAM PAGE REQUIREMENTS.
-
RT4GN,27
-
ASMB,27
-
/E
ASSIGN PROG PART'NS?(/E TO END)
PNAME,PART'N#        *ASSIGN PROGRAMS TO PARTITIONS.
-
RT4GN,3
-
WHZAT,4
-
/E
NEW MEM CONFIGURATION PERMANENT?(YES/NO)
NO                    *DO NOT MAKE MEMORY CHANGES PERMANENT.

                                *END OF I/O AND MEMORY RECONFIGURATION.
                                *SYSTEM WILL NOW ATTEMPT TO BOOTUP.

SET TIME
:SV,4
TE,*****
TE,*****  92067A RTE-IV 7905 7906 7920 DISC CARTRIDGE
TE,*****           HP 92067-13101 (7905/7906)
TE,*****           HP 92067-13201 (7920)
TE,*****
:

```

Figure 12-1. Reconfiguration Example (continued)

### 12-14. BOOT-UP AND RECONFIGURATION HALTS

During either system boot-up or reconfiguration, various HLTS (of the form 1020xx) may be issued on the computer front panel. The meaning of these halts and any required operator action are given in Table 12-1.

MEMORY AND I/O RECONFIGURATION

Table 12-1. System Boot-up and Reconfiguration Halts

HLT	Meaning	User Action
4	Powerfail occurred and powerfail automatic restart is enabled.	Restart system boot-up procedure.
5	Memory protect switch was set and memory parity error occurred.	Restart system boot-up procedure.
10B	FMGR or D.RTR cannot be scheduled at startup because there is not a large enough partition (issued by the system).	Restart system boot-up and redefine memory to include a partition large enough for FMGR and D.RTR.
11B	Attempt was made to re-execute a non-RPL compatible ROM Loader Part # 12992A, or Bootstrap Loader.	Reload the ROM Loader or Bootstrap Loader before re-executing.
22B	One of the following conditions was encountered: 1. \$CNFG cannot find an ID segment for Configurator extension \$CNFX. 2. \$CNFX is not a Type 3 program. 3. A contiguous memory block of three good pages cannot be found in the user partition area.	Restart system boot-up procedure. If memory reconfiguration is desired \$CNFX must be permanently loaded as a Type 3 program and there must be at least 3 good pages of contiguous memory in the user partition area.
30B	Error was encountered in the disc I/O process by one of the RPL-compatible ROM Loaders Part #12992B & 12992F. If the disc is a 7900 the disc status is displayed in the A-register. If the disc is a 7905/20 the disc status word 1 is displayed in the B-register and disc status word 2 in the A-register.	Retry the system boot-up procedure.
31B	Error encountered in the disc I/O process by the Boot Extension. If the disc is a 7900, the disc status is displayed in the A-register. If the disc is 7905 or 7920, the disc status word 1 is displayed in the B-register and disc status word 2 is displayed in the A-register.	Retry the system boot-up procedure.
55B	An EQT with the equipment type code of console cannot be found.	Restart boot-up procedure with a console for which an EQT is generated in the system.



## 12-15. CONFIGURATOR ERROR MESSAGES

Whenever a user response to a Configurator prompt is illegal or inappropriate, the Configurator issues a CONFIG ERR message and prompts for a correct entry. All possible Configurator error codes are listed sequentially in Table 12-2. Locate the appropriate code and take the described action.

MEMORY AND I/O RECONFIGURATION

Table 12-2. I/O and Memory Reconfiguration Error Codes

CONFIG ERR	Meaning	User Action
1	Invalid LU number or a bit bucket LU.	Enter valid logical unit number.
2	Illegal select code number.	Enter valid number that must be between 10 and 77 octal.
3	New select code entered is identical to new select code assigned to disc, system console or list device, or else the current select code entered is identical to the old select code for disc, system console or list device (i.e., do not reconfigure that which was already done via the SWITCH register).	Enter different select code.
10	Specified total number of pages outside the range.	Enter valid number in the range 48-1024 for physical memory size and between 0 and maximum pages available for SAM extension.
11	Invalid bad page number.	Enter valid number greater than the previous entry and less than the physical memory size, or enter /E to terminate the list.
12	Specified SAM extension entry beyond physical memory size due to bad pages.	Enter smaller number of pages for SAM extension.
13	Current running total exceeds available pages in block of good memory or exceeds size of mother partition.	Redefine last partition or subpartition size. If there are no more pages available in the block of memory to be defined, /E or /R are the only responses accepted.

Table 12-2. I/O and Memory Reconfiguration Error Codes (cont'd)

CONFIG ERR	Meaning	User Action
14	Second parameter of partition definition entry other than RT, BG or S, or else S was entered when a subpartition definition was not expected.	Reenter definition with correct parameter.
15	Third parameter of partition definition entry other than R.	Reenter definition with R as third parameter if partition is to be reserved.
16	No such program, or the name of a segment was entered or invalid type was entered for partition assignment.	Reenter assignment with correct program name or type or /E to end this sequence.
17	Invalid partition number.	Enter valid number or /E to end this sequence.
18	Program does not fit in the assigned partition.	Assign program to larger partition if available, or continue without assigning the program.
19	Invalid number of pages was entered for program size.	Enter valid number of pages for program, between the size of the program at load time and the maximum logical address space for the program.
20	Number of defined partitions already equal to allowed maximum number and more undefined pages remain.	Redefine all partitions
21	Page requirements of an EMA program cannot be modified.	Entry is skipped.
22	Number of pages in SAM extension requires division into more than five blocks.	Enter a smaller size of SAM extension

# HP CHARACTER SET

APPENDIX

A

BITS		COLUMN	0 <sub>00</sub>	0 <sub>01</sub>	0 <sub>10</sub>	0 <sub>11</sub>	1 <sub>00</sub>	1 <sub>01</sub>	1 <sub>10</sub>	1 <sub>11</sub>
b <sub>7</sub>	b <sub>6</sub> b <sub>5</sub>	ROW ↓	0	1	2	3	4	5	6	7
0	0 0	0	NUL	DLE	SP	0	@	P	,	p
0	0 1	1	SOH	DC1	!	1	A	Q	a	q
0	1 0	2	STX	DC2	"	2	B	R	b	r
0	1 1	3	ETX	DC3	#	3	C	S	c	s
1	0 0	4	EOT	DC4	\$	4	D	T	d	t
1	0 1	5	ENQ	NAK	%	5	E	U	e	u
1	1 0	6	ACK	SYN	&	6	F	V	f	v
1	1 1	7	BEL	ETB	'	7	G	W	g	w
1	0 0	8	BS	CAN	(	8	H	X	h	x
1	0 1	9	HT	EM	)	9	I	Y	i	y
1	1 0	10	LF	SUB	*	:	J	Z	j	z
1	1 1	11	VT	ESC	+	;	K	[	k	{
1	0 0	12	FF	FS	,	<	L	\	l	;
1	0 1	13	CR	GS	-	=	M	]	m	}
1	1 0	14	SO	RS	.	>	N	^	n	~
1	1 1	15	SI	US	/	?	O	_	o	DEL

32 CONTROL CODES

64 CHARACTER SET

96 CHARACTER SET

128 CHARACTER SET

Upshifted Lower Case

EXAMPLE: The representation for the character "K" (column 4, row 11) is.

	b <sub>7</sub>	b <sub>6</sub>	b <sub>5</sub>	b <sub>4</sub>	b <sub>3</sub>	b <sub>2</sub>	b <sub>1</sub>
BINARY	1	0	0	1	0	1	1
OCTAL	1	1	3				

\* Depressing the Control key while typing an upper case letter produces the corresponding control code on most terminals. For example, Control-H is a backspace.

### HEWLETT-PACKARD CHARACTER SET FOR COMPUTER SYSTEMS

This table shows HP's implementation of ANS X3 4-1968 (USASCII) and ANS X3 32-1973. Some devices may substitute alternate characters from those shown in this chart (for example, Line Drawing Set or Scandinavian font). Consult the manual for your device.

The left and right byte columns show the octal patterns in a 16 bit word when the character occupies bits 8 to 14 (left byte) or 0 to 6 (right byte) and the rest of the bits are zero. To find the pattern of two characters in the same word, add the two values. For example, "AB" produces the octal pattern 040502. (The parity bits are zero in this chart.)

The octal values 0 through 37 and 177 are control codes. The octal values 40 through 176 are character codes.

Decimal Value	Octal Values		Mnemonic	Graphic <sup>1</sup>	Meaning
	Left Byte	Right Byte			
0	000000	000000	NUL	␣	Null
1	000400	000001	SOH	␣	Start of Heading
2	001000	000002	STX	␣	Start of Text
3	001400	000003	ETX	␣	End of Text
4	002000	000004	EOT	␣	End of Transmission
5	002400	000005	ENQ	␣	Enquiry
6	003000	000006	ACK	␣	Acknowledge
7	003400	000007	BEL	␣	Bell, Attention Signal
8	004000	000010	BS	␣	Backspace
9	004400	000011	HT	␣	Horizontal Tabulation
10	005000	000012	LF	␣	Line Feed
11	005400	000013	VT	␣	Vertical Tabulation
12	006000	000014	FF	␣	Form Feed
13	006400	000015	CR	␣	Carrriage Return
14	007000	000016	SO	␣	Shift Out
15	007400	000017	SI	␣	Shift In
16	010000	000020	DLE	␣	Data Link Escape
17	010400	000021	DC1	␣	Device Control 1 (X-ON)
18	011000	000022	DC2	␣	Device Control 2 (TAPE)
19	011400	000023	DC3	␣	Device Control 3 (X-OFF)
20	012000	000024	DC4	␣	Device Control 4 (TAPE)
21	012400	000025	NAK	␣	Negative Acknowledge
22	013000	000026	SYN	␣	Synchronous Idle
23	013400	000027	ETB	␣	End of Transmission Block
24	014000	000030	CAN	␣	Cancel
25	014400	000031	EM	␣	End of Medium
26	015000	000032	SUB	␣	Substitute
27	015400	000033	ESC	␣	Escape <sup>2</sup>
28	016000	000034	FS	␣	File Separator
29	016400	000035	GS	␣	Group Separator
30	017000	000036	RS	␣	Record Separator
31	017400	000037	US	␣	Unit Separator
127	077400	000177	DEL	␣	Delete, Rubout <sup>3</sup>

Decimal Value	Octal Values		Character	Meaning
	Left Byte	Right Byte		
32	020000	000040		Space, Blank
33	020400	000041	!	Exclamation Point
34	021000	000042	"	Quotation Mark
35	021400	000043	#	Number Sign, Pound Sign
36	022000	000044	\$	Dollar Sign
37	022400	000045	%	Percent
38	023000	000046	&	Ampersand, And Sign
39	023400	000047	'	Apostrophe, Acute Accent
40	024000	000050	(	Left (opening) Parenthesis
41	024400	000051	)	Right (closing) Parenthesis
42	025000	000052	*	Asterisk, Star
43	025400	000053	+	Plus
44	026000	000054	,	Comma, Cedilla
45	026400	000055	-	Hyphen, Minus, Dash
46	027000	000056	.	Period, Decimal Point
47	027400	000057	/	Slash, Slant
48	030000	000060	0	} Digits, Numbers
49	030400	000061	1	
50	031000	000062	2	
51	031400	000063	3	
52	032000	000064	4	
53	032400	000065	5	
54	033000	000066	6	
55	033400	000067	7	
56	034000	000070	8	} Colon
57	034400	000071	9	
58	035000	000072	:	Semicolon
59	035400	000073	;	Less Than
60	036000	000074	<	Equals
61	036400	000075	=	Greater Than
62	037000	000076	>	Question Mark
63	037400	000077	? - ?	

Decimal Value	Octal Values		Character	Meaning
	Left Byte	Right Byte		
64	040000	000100	@	Commercial At
65	040400	000101	A	Upper Case Alphabet. Capital Letters
66	041000	000102	B	
67	041400	000103	C	
68	042000	000104	D	
69	042400	000105	E	
70	043000	000106	F	
71	043400	000107	G	
72	044000	000110	H	
73	044400	000111	I	
74	045000	000112	J	
75	045400	000113	K	
76	046000	000114	L	
77	046400	000115	M	
78	047000	000116	N	
79	047400	000117	O	
80	050000	000120	P	
81	050400	000121	Q	
82	051000	000122	R	
83	051400	000123	S	
84	052000	000124	T	
85	052400	000125	U	
86	053000	000126	V	
87	053400	000127	W	
88	054000	000130	X	
89	054400	000131	Y	
90	055000	000132	Z	
91	055400	000133	[	Left (opening) Bracket
92	056000	000134	\	Backslash, Reverse Slant
93	056400	000135	]	Right (closing) Bracket
94	057000	000136	^ ↑	Caret, Circumflex; Up Arrow <sup>4</sup>
95	057400	000137	_ ←	Underline; Back Arrow <sup>4</sup>

Decimal Value	Octal Values		Character	Meaning
	Left Byte	Right Byte		
96	060000	000140	`	Grave Accent <sup>5</sup>
97	060400	000141	a	Lower Case Letters <sup>5</sup>
98	061000	000142	b	
99	061400	000143	c	
100	062000	000144	d	
101	062400	000145	e	
102	063000	000146	f	
103	063400	000147	g	
104	064000	000150	h	
105	064400	000151	i	
106	065000	000152	j	
107	065400	000153	k	
108	066000	000154	l	
109	066400	000155	m	
110	067000	000156	n	
111	067400	000157	o	
112	070000	000160	p	
113	070400	000161	q	
114	071000	000162	r	
115	071400	000163	s	
116	072000	000164	t	
117	072400	000165	u	
118	073000	000166	v	
119	073400	000167	w	
120	074000	000170	x	
121	074400	000171	y	
122	075000	000172	z	
123	075400	000173	{	Left (opening) Brace <sup>5</sup>
124	076000	000174		Vertical Line <sup>5</sup>
125	076400	000175	}	Right (closing) Brace <sup>5</sup>
126	077000	000176	~	Tilde, Overline <sup>5</sup>

9206- 1C

Notes: <sup>1</sup>This is the standard display representation. The software and hardware in your system determine if the control code is displayed, executed, or ignored. Some devices display all control codes as "|", "@", or space

<sup>2</sup>Escape is the first character of a special control sequence. For example, ESC followed by "J" clears the display on a 2640 terminal.

<sup>3</sup>Delete may be displayed as "|", "@", or space

<sup>4</sup>Normally, the caret and underline are displayed. Some devices substitute the up arrow and back arrow

<sup>5</sup>Some devices upshift lower case letters and symbols ( ` through ~ ) to the corresponding upper case character ( @ through ^ ). For example, the left brace would be converted to a left bracket.

## RTE SPECIAL CHARACTERS

Mnemonic	Octal Value	Use
SOH (Control A)	1	Backspace (TTY)
EM (Control Y)	31	Backspace (2600)
BS (Control H)	10	Backspace (TTY, 2615, 2640, 2644, 2645)
EOT (Control D)	4	End-of-file (TTY 2615, 2640, 2644, 2645)

9206-1D

This appendix contains information about the following topics:

- \* SYSTEM COMMUNICATIONS AREA - Base page locations of area used for system communications.
- \* PROGRAM ID SEGMENT MAP - Format of ID segments kept in system area for user programs, ID segment extension, and short ID segments.
- \* MEMORY ALLOCATION TABLE (MAT) ENTRY FORMAT
- \* DISC LAYOUT - Allocation of disc space for an RTE-IV system.

Other system tables relating to I/O considerations, such as the Equipment Table, Device Reference Table and Driver Mapping Table are described in Section V, "Input/Out."

#### B-1. SYSTEM COMMUNICATION AREA

This area is a block of storage in the system base page, starting at location 1645 octal, that is used by RTE-IV to define request parameters, I/O tables, scheduling lists, operating parameters, memory bounds, etc. The RTE-IV Assembler allows relocatable programs to reference this area by absolute addresses 1645 through 1777 octal. User programs can read information from this area but cannot alter it because of the memory protect feature.

The contents and description of each location in this area are listed in Table B-1.



SYSTEM COMMUNICATION AREA AND SYSTEM TABLES

Table B-1. System Communications Area Locations

Octal Location	Contents	Description
SYSTEM TABLE DEFINITION		
01645	XIDEX	Address of current program's ID extension
01646	XMATA	Address of current program's MAT entry
01647	XI	Address of index register save area
01650	EQTA	FWA of Equipment Table
01651	EQT#	Number of EQT entries
01652	DRT	FWA of Device Reference Table
01653	LUMAX	Number of logical units in DRT
01654	INTBA	FWA of Interrupt Table
01655	INTLG	Number of Interrupt Table Entries
01656	TAT	FWA of Track Assignment Table
01657	KEYWD	FWA of keyword block
I/O MODULE/DRIVER COMMUNICATION		
01660	EQT1 \	Addresses of first 11 words of current EQT entry (see 01771 for last four words)
01661	EQT2	
01662	EQT3	
01663	EQT4	
01664	EQT5 \	
01665	EQT6 /	
01666	EQT7	
01667	EQT8	
01670	EQT9	
01671	EQT10	
01672	EQT11 /	
01673	CHAN	Current DCPC channel number
01674	TBG	I/O address of time-base card
01675	SYSTY	EQT entry address of system console
SYSTEM REQUEST PROCESSOR/EXEC COMMUNICATION		
01676	RQCNT	Number of request parameters -1
01677	RQRTN	Return point address
01700	RQP1 \	Addresses of request parameters (set for a maximum of nine parameters)
01701	RQP2	
01702	RQP3	
01703	RQP4 \	
01704	RQP5 /	
01705	RQP6	
01706	RQP7	
01707	RQP8	
01710	RQP9 /	

SYSTEM COMMUNICATION AREA AND SYSTEM TABLES

Table B-1. System Communication Area Locations (continued)

Octal Location	Contents	Description
SYSTEM LISTS ADDRESSES		
01711	SKEDD	Schedule list header
01713	SUSP2	Wait Suspend list header
01714	SUSP3	Available Memory list header
01715	SUSP4	Disc Allocation list header
01716	SUSP5	Operator Suspend list header
PROGRAM ID SEGMENT DEFINITION		
01717	XEQT	ID segment address of current program
01720	XLINK	Linkage address
01721	XTEMP	Temporary addresses (five words)
01726	XPRIO	Priority word address
01727	XPENT	Primary entry point address
01730	XSUSP	Point of suspension address
01731	XA	A-register at suspension address
01732	XB	B-register at suspension address
01733	XEO	E and overflow register suspension address
SYSTEM MODULE COMMUNICATION FLAGS		
01734	OPATN	Operator/keyboard attention flag
01735	OPFLG	Operator communication flag
01736	SWAP	RT disc resident swapping flag
01737	DUMMY	I/O address of privileged interface card
01740	IDSDA	Disc address of first ID segment
01741	IDSDP	Position within disc sector of first ID segment
MEMORY ALLOCATION BASES DEFINITION		
01742	BPA1	FWA user base page link area
01743	BPA2	LWA user base page link area
01744	BPA3	FWA user base page link
01745	LBORG	FWA of resident library area
01746	RTORG	FWA of real-time COMMON
01747	RTCOM	Length of real-time COMMON
01750 D	RTDRA	FWA of real-time partition
01751 D	AVMEM	LWA+1 of real-time partition
01752	BGORG	FWA of background COMMON
01753	BGCOM	Length of background COMMON
01754 D	BGDRA	FWA of background partition

SYSTEM COMMUNICATION AREA AND SYSTEM TABLES

Table B-1. System Communications Area Locations (continued)

Octal Location	Contents	Description
UTILITY PARAMETERS		
01755	TATLG	Negative length of track assignment table
01756	TATSD	Number of tracks on system disc
01757	SECT2	Number of sectors/track on LU2 (system)
01760	SECT3	Number of sectors/track on LU3 (aux.)
01761	DSCLB	Disc address of user available library entry points
01762	DSCLN	Number of user available library entry points.
01763	SYSLB	Disc address of system library entry points
01764	SYSLN	Number of system library entry points
01765	LGOTK	LG Area: LU#, starting track, number of tracks (same format as ID segment word 27)
01766	LGOC	Current LG Area track/sector address (same format as ID segment word 26)
01767	SFCUN	LS: LU# and disc address (same format as ID segment word 26)
01770	MPTFL	Memory protect ON/OFF flag (0/1)
01771	EQT12 \	Address of last four words of current EQT
01772	EQT13 \	
01773	EQT14 /	
01774	EQT15 /	
01775 D	FENCE	Memory protect fence address
01777	BGLWA	LWA memory background partition

D letter indicates the contents of the location are set dynamically by the dispatcher.

B-2. PROGRAM ID SEGMENT

Each user program has a 33-word ID segment located in Table Area II that contains static and dynamic information defining the properties of the program. The static information is set during generation time or when the program is loaded on-line. The dynamic information is maintained by the operating system Executive.

SYSTEM COMMUNICATION AREA AND SYSTEM TABLES

The number of ID segments contained in a system is established during system generation, and is directly related to the number of programs that can be in main memory at any given time. If all the ID segments are in use, no more programs can be added on-line unless some other existing program is first "OFFed" (removed from the system) to recover an ID segment.

The format of the ID segment is illustrated in Figure B-1. Each ID segments's address is located in the Keyword Table (see location 01657 octal).

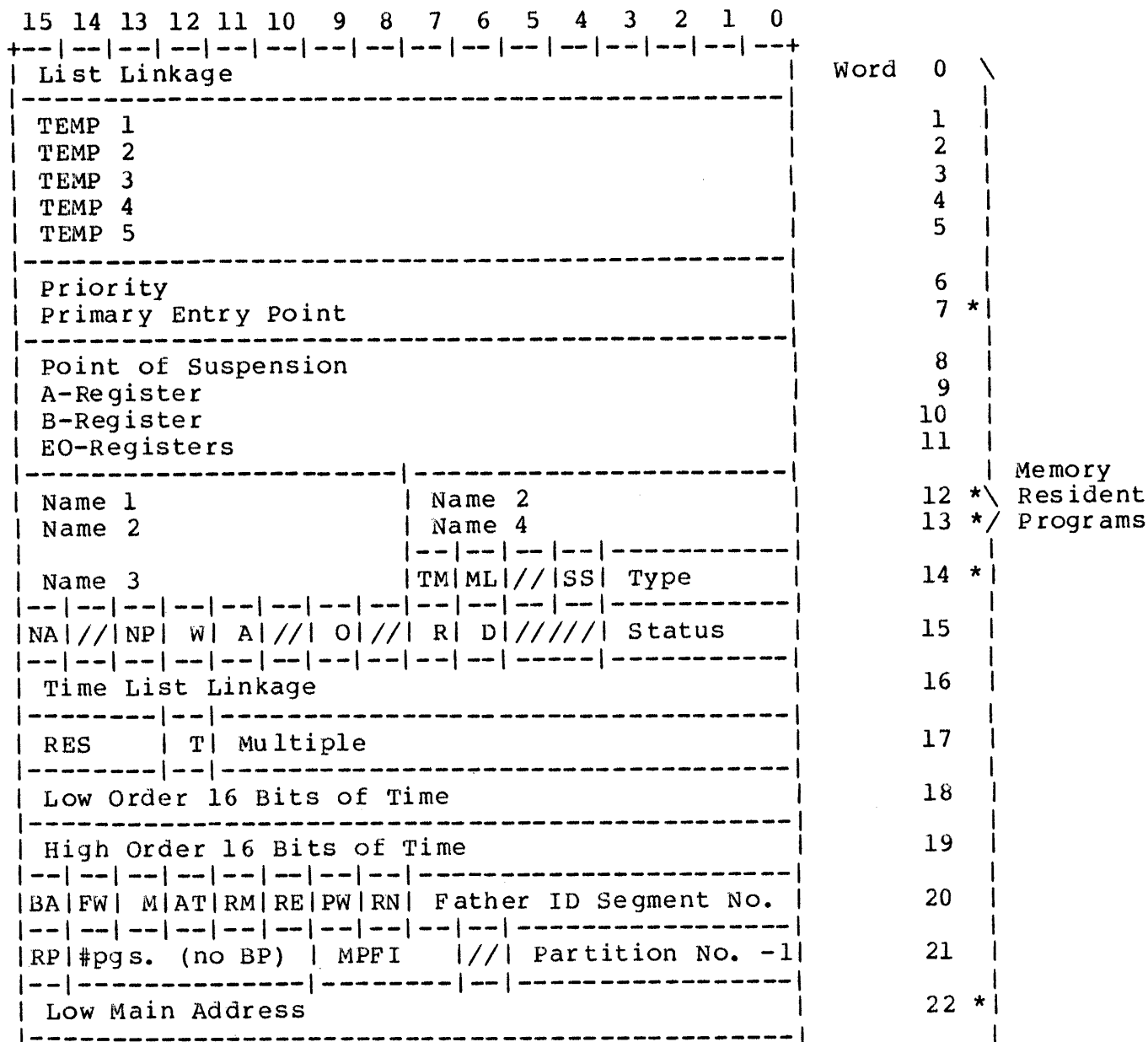


Figure B-1. ID Segment Format

SYSTEM COMMUNICATION AREA AND SYSTEM TABLES

-----			
High Main Address + 1			23 *
-----			
Low Base Page Address			24 *
-----			
High Base Page Address + 1			25 */
-----			
LU	Program: Track	Sector	26 *
-----			
LU	Swap: Track	No. Tracks	27
-----			
ID Extension No.	EMA Size		28
-----			
High Address + 1 of Largest Segment			29
-----			
Reserved			30 \
-----			
Reserved			31 \ Memory
-----			
Negative MTM LU number			32 / Residents
-----			

where:

\* = words used in short ID segments for program segments

TM = temporary load (copy of ID segment is not on the disc)

ML = memory lock (program may not be swapped)

SS = short segment (indicates a nine-word ID segment)

Type = specified program type (1-5)

NA = no abort (instead, pass abort errors to program)

NP = no parameters allowed on reschedule

W = wait bit (waiting for program whose ID segment address is in word 2)

A = abort on next list entry for this program

O = operator suspend on next schedule attempt

R = resource save (save resources when setting dormant)

D = dormant bit (set dormant on next schedule attempt)

Figure B-1. ID Segment Format (continued)

## SYSTEM COMMUNICATION AREA AND SYSTEM TABLES

Status = current program status

T = time list entry bit (program is in the time list)

BA = batch (program is running under batch)

FW = father is waiting (father scheduled with wait)

M = Multi-Terminal Monitor bit

AT = attention bit (operator has requested attention)

RM = reentrant memory must be moved before dispatching program

RE = reentrant routine now has control

PW = program wait (some other program wants to schedule this one)

RN = Resource Number either owned or locked by this program

RP = reserved partition (only for programs that request it)

MPFI = memory protect fence index

Figure B-1. ID Segment Format (continued)

# SYSTEM COMMUNICATION AREA AND SYSTEM TABLES

## B-3. ID SEGMENT EXTENSIONS

Each EMA program requires a 3-word ID segment extension in addition to its 33-word ID segment. The program's ID segment word 28 will point to its ID segment extension. The number of ID extensions contained in the system is set at generation time, and if all are in use, no more EMA programs can be added on-line. The format of the ID segment is illustrated in Figure B-2.

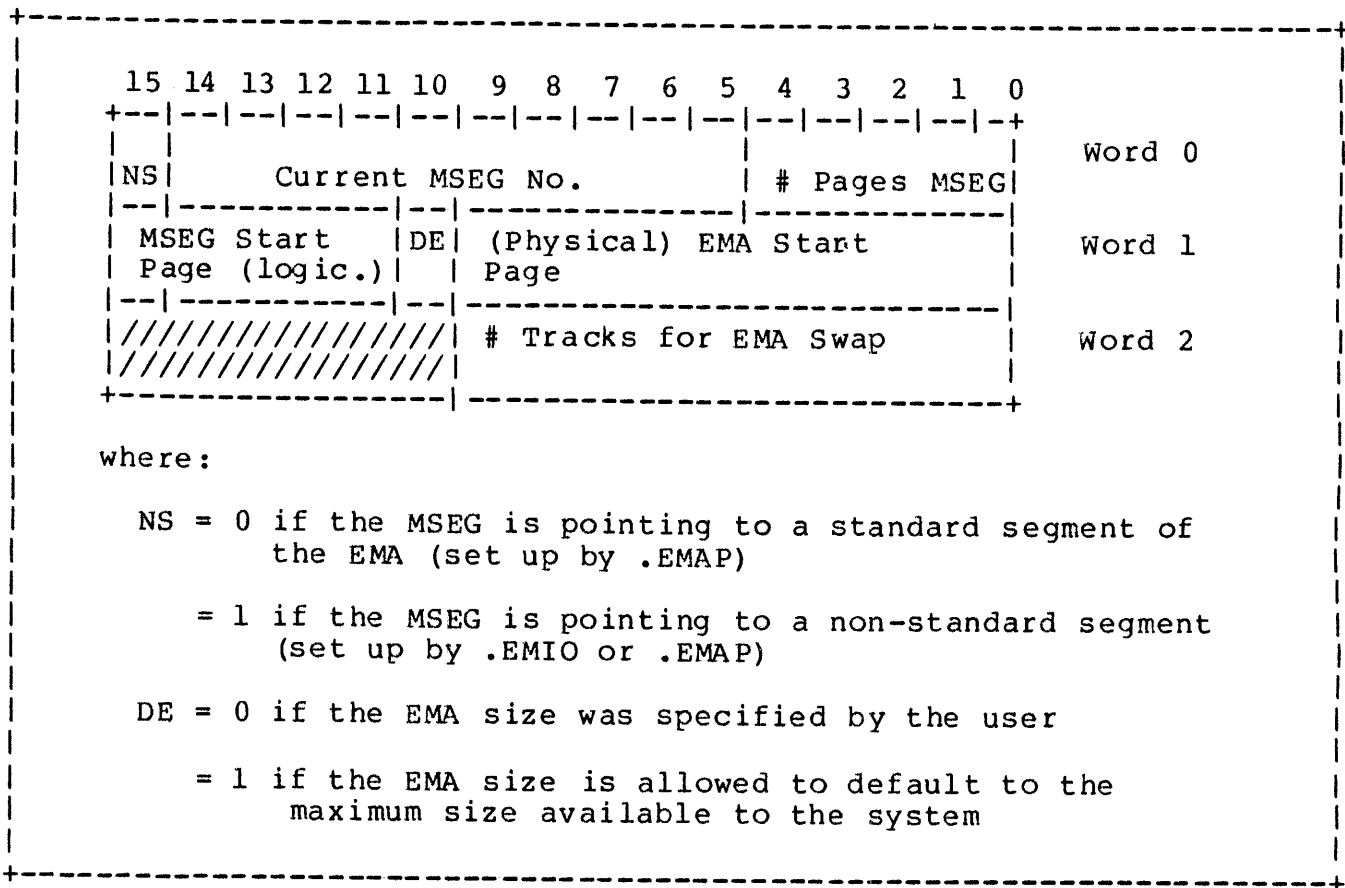


Figure B-2. ID Segment Extension

SYSTEM COMMUNICATION AREA AND SYSTEM TABLES

B-4. SHORT ID SEGMENTS

Short ID segments requiring nine words are used only for background program segments. A short ID segment is required for each segment of a segmented program. If no empty short ID segments are available during an on-line load, a standard 33-word ID segment will be used.

B-5. MEMORY ALLOCATION TABLE ENTRY

Each partition defined by the user during generation contains an entry in the Memory Allocation Table (MAT). This table starts at the system entry point \$MATA and extends upward toward high memory. Each entry is seven words long, arranged as illustrated in Figure B-5.

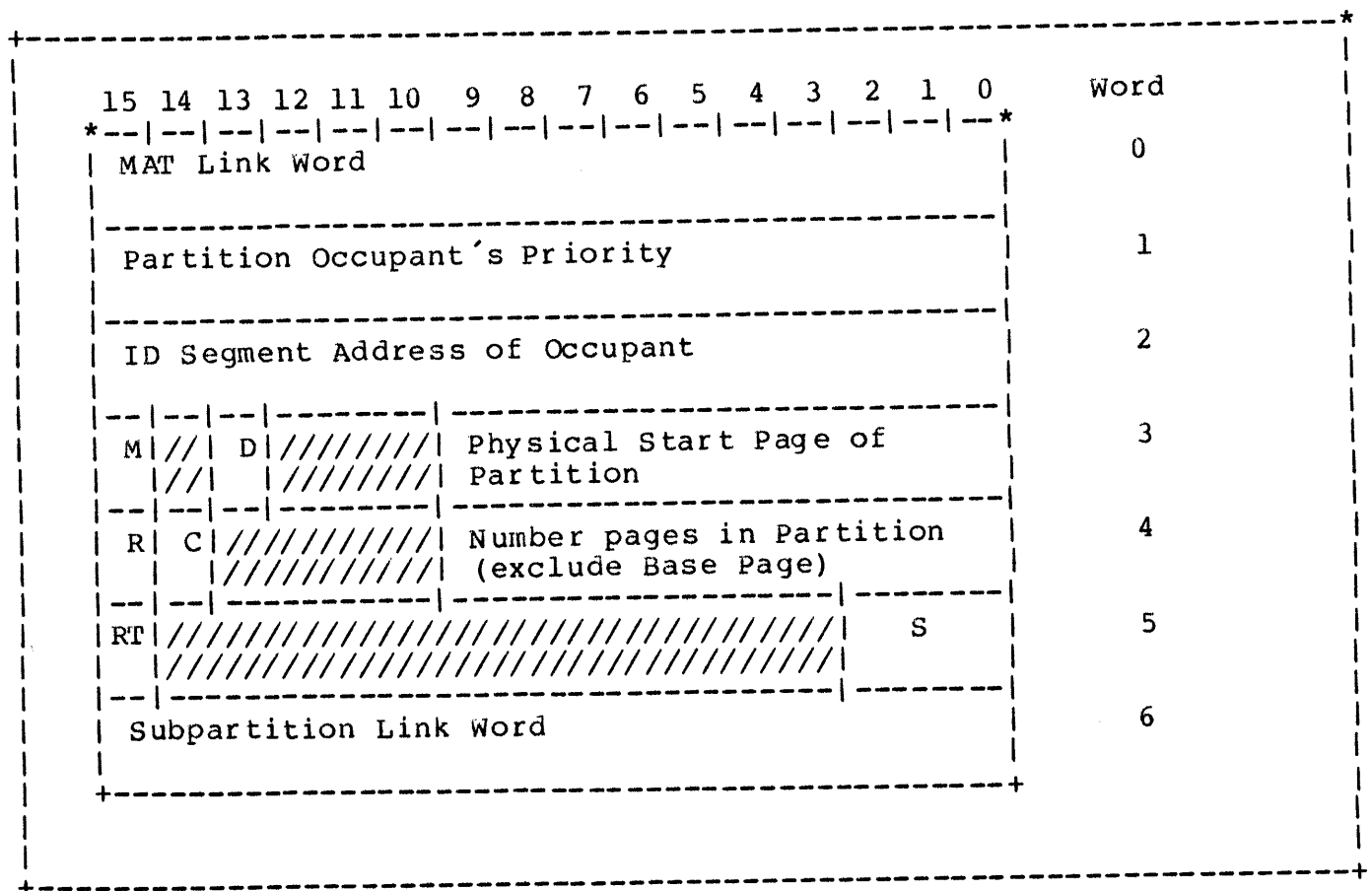


Figure B-5. Memory Allocation Table Entry Format



## SYSTEM COMMUNICATION AREA AND SYSTEM TABLES

where:

MAT Link Word = -1 if partition not defined either during  
system generation or by parity error

= 0 if end of list

M = 1 if MAT entry is for a mother partition

D = 1 if program is dormant after save-resource  
or serially reusability termination

R = 1 if partition is reserved

C = 1 if partition is in use as part of a chained  
partition

RT = 1 if MAT entry is for real-time partition

S = program's dispatching status

= 0 - program being loaded

1 - program is in memory

2 - segment is being loaded or swapped out

3 - program is swapped out

4 - subpartition swap-out started for mother  
partition

5 - subpartition completed. Mother partition  
cleared.

Subpartition Link Word

= 0 if MAT entry is not a subpartition or a mother  
partition

= next subpartition address if saubpartition

= mother partition MAT address if this entry  
is the last partition.

Figure B-3. Memory Allocation Table Entry Format

### B-6. RTE-IV SYSTEM DISC LAYOUT

Figure B-4 illustrates how disc space is allocated when an RTE-IV  
when an RTE-IV system is generated.

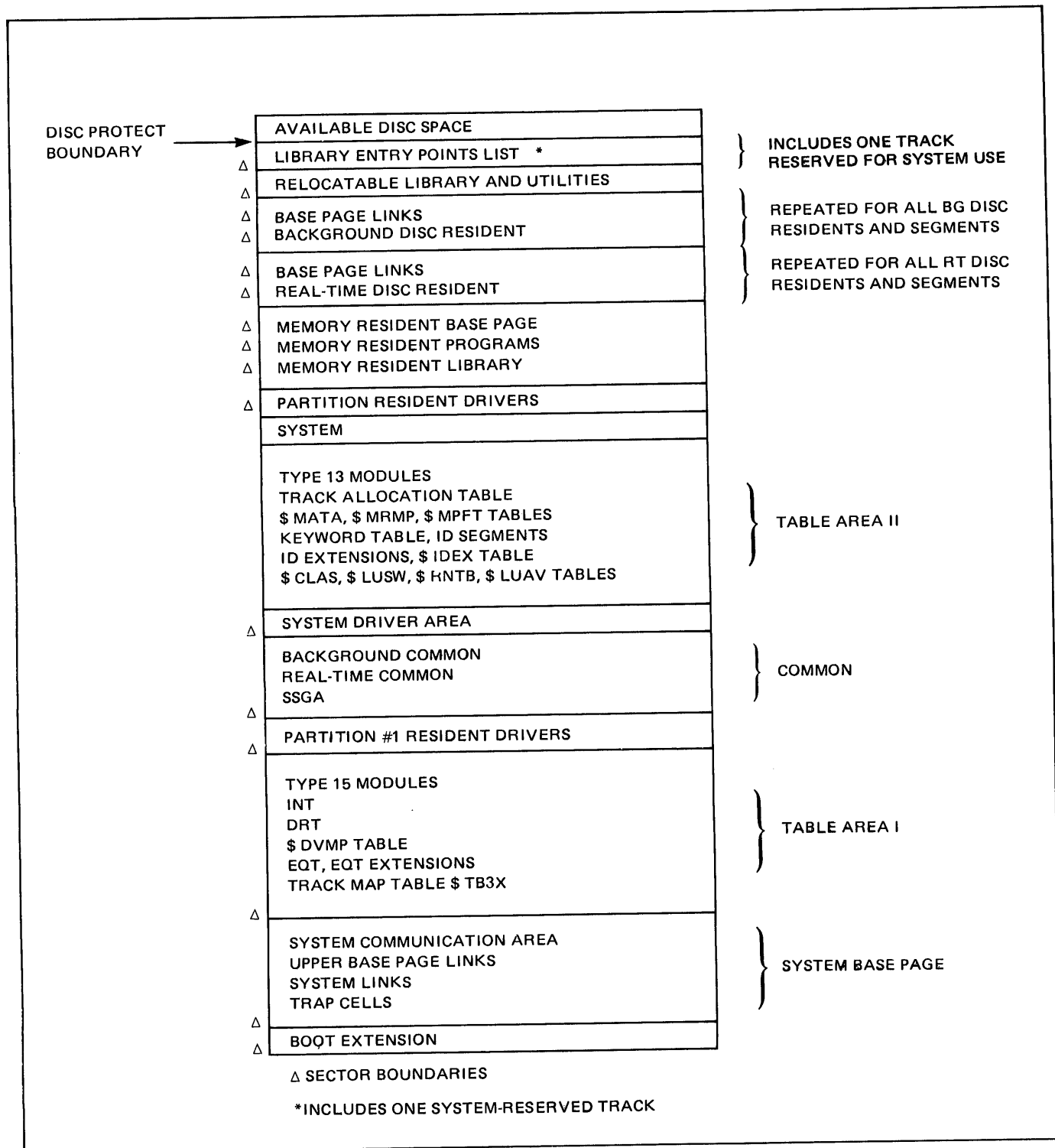
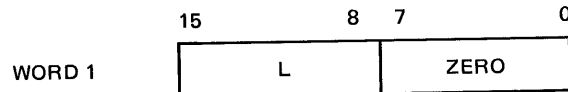


Figure B-4. RTE-IV System Disc Layout

## C-1. SOURCE RECORD FORMATS

The source format used for the disc records in the LS area by the system program EDITR and FMGR is given in Figure C-1. All records are packed ignoring sector boundaries. Binary records are packed directly onto the disc. After an END record, a zero word is written and the rest of the sector is skipped. If this zero word is the first word of the sector, it is not written. Binary files are always contiguous so a code word is not required.



WHERE L IS THE RECORD LENGTH IN WORDS EXCLUDING WORD 1

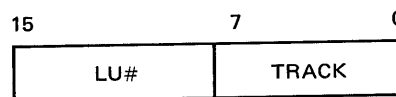


⋮

IF WORD 1 = 0 THEN END OF TAPE  
IF WORD 1 = -1 THEN END OF FILE

ODD CHARACTERS ARE PADDED WITH BLANKS TO MAKE A FULL WORD.  
THE LAST WORD ON ANY GIVEN TRACK IN A MULTI-TRACK FILE IS A  
CODE WORD THAT POINTS TO THE NEXT TRACK IN THE FILE.

## CODE WORD FORMAT



WHERE LU# IS EITHER 2 (SYSTEM) OR 3 (AUXILIARY) DEPENDING ON  
WHICH PLATTER THE TRACK IS ON.

## C-1. SOURCE RECORD FORMATS

## RECORD FORMATS

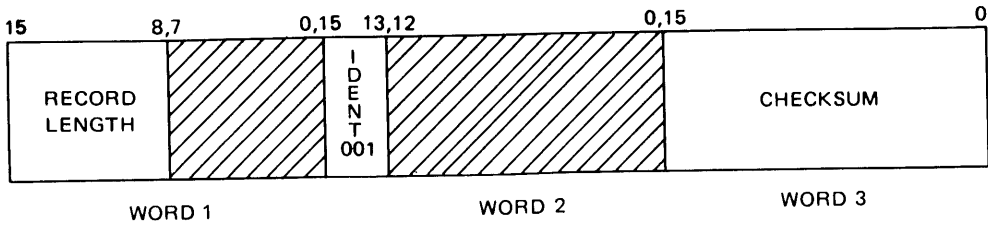
The following describes the formats of relocatable and absolute records produced as object code for a given source program. The relocatable records are generated by compilers or by the assembler for a relocatable assembly. These records are stored in a relocatable file. The generator or the loader processes these relocatable records to produce an absolute module which has all program links resolved and the program is relocated and ready to run.

The absolute records are produced by the assembler for an absolute assembly. The module of records thus produced requires no processing by the generator or loader. Absolute programs must be loaded into memory and run off-line.

# NAM RECORD

## CONTENT

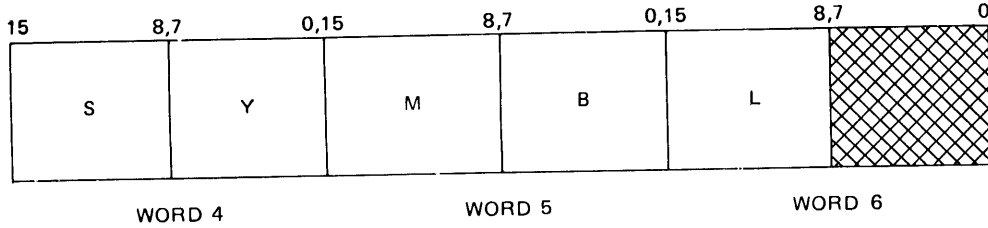
## EXPLANATION



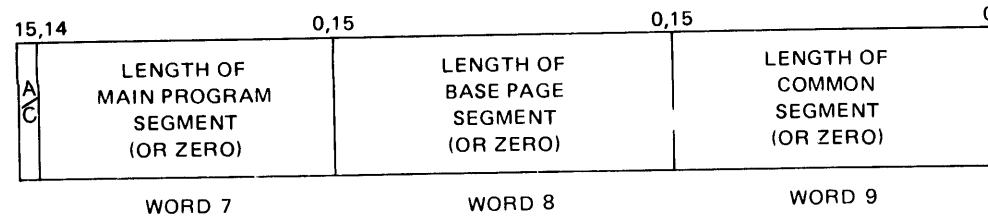
RECORD LENGTH = 9-60 WORDS

IDENT = 001

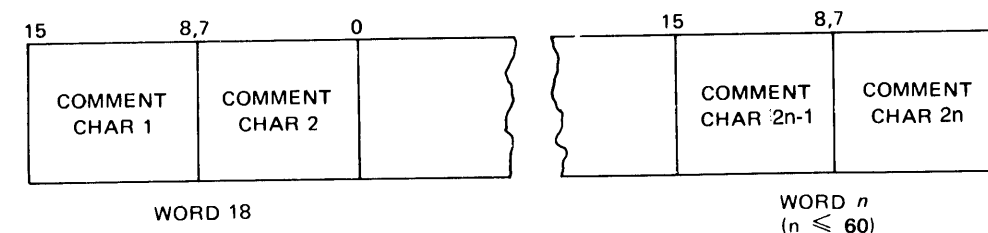
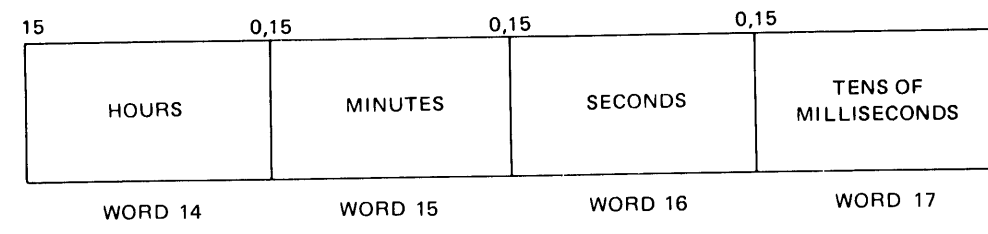
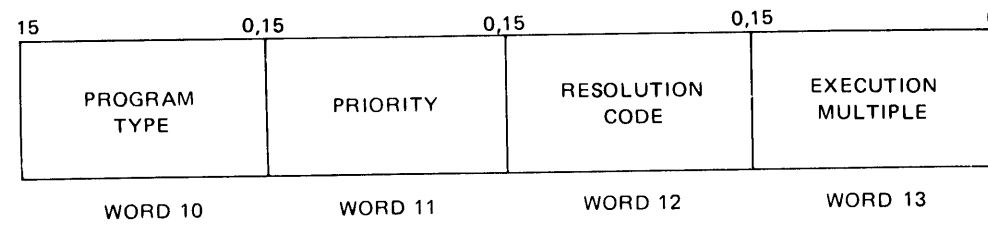
CHECKSUM: ARITHMETIC TOTAL OF ALL WORDS IN RECORD EXCLUDING WORDS 1 AND 3.




SYMBL: FIVE CHARACTER NAME OF PROGRAM



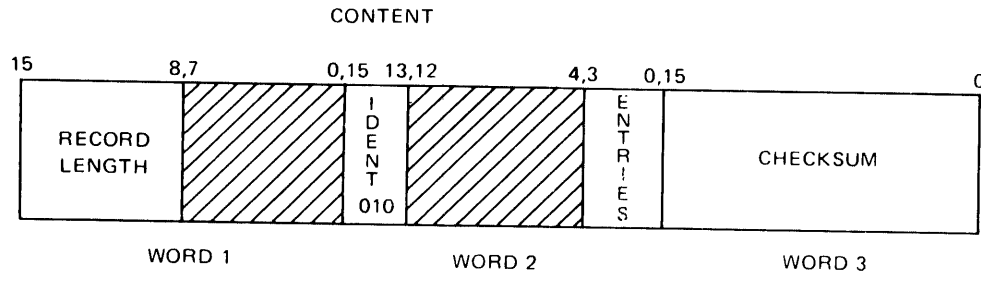
A/C: BINARY TAPE PRECESSION  
 = 0 IF ASSEMBLER PRODUCED OR LENGTH IS EXACT  
 = 1 IF COMPILER PRODUCED, AND LENGTH IS UNKNOWN



 HATCH-MARKED AREAS SHOULD BE ZERO-FILLED WHEN THE RECORDS ARE GENERATED

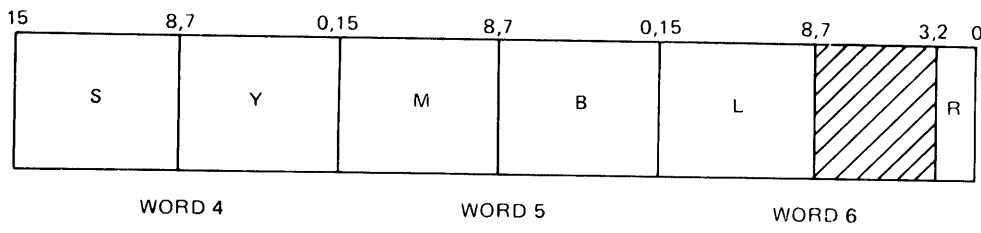
 CROSS-HATCH-MARKED AREAS SHOULD BE SPACE-FILLED WHEN THE RECORDS ARE GENERATED

# ENT RECORD

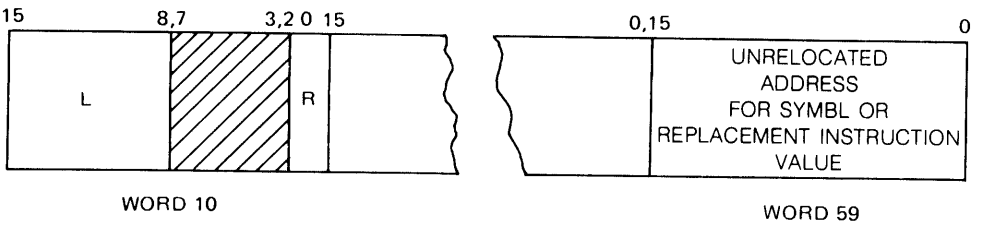
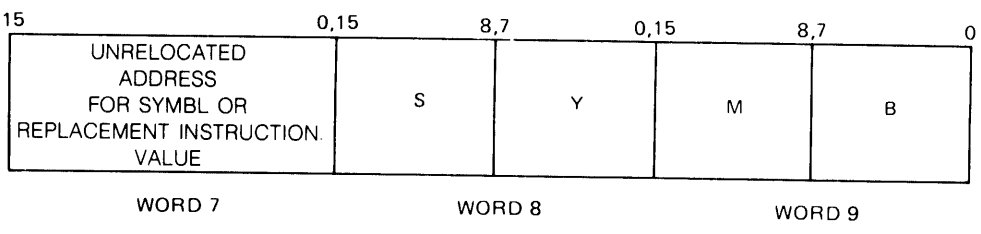


## EXPLANATION

RECORD LENGTH = 7-59 WORDS  
 IDENT = 010  
 ENTRIES: 1 TO 14 ENTRIES PER RECORD; EACH ENTRY IS FOUR WORDS LONG.



SYMBL: 5 CHARACTER ENTRY POINT SYMBOL  
 R: RELOCATION INDICATOR  
 = 0 IN PROGRAM RELOCATABLE  
 = 1 IF BASE PAGE RELOCATABLE  
 = 2 IF COMMON RELOCATABLE  
 = 3 IF ABSOLUTE  
 = 4 INSTRUCTION REPLACEMENT

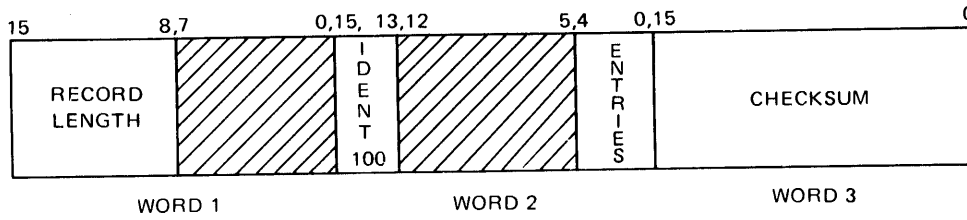


WORDS 4 THROUGH 7 ARE REPEATED FOR EACH ENTRY POINT SYMBOL.

# EXT RECORD

CONTENT

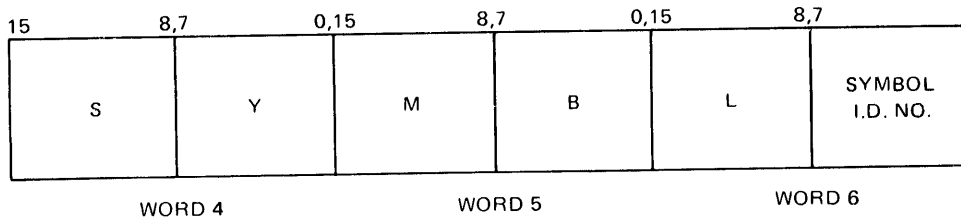
EXPLANATION



RECORD LENGTH = 6-60 WORDS

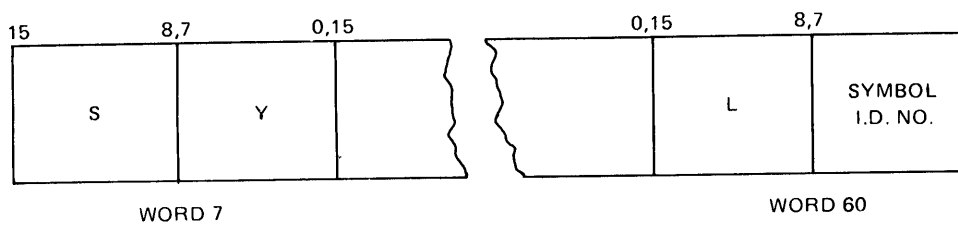
IDENT = 100

ENTRIES: 1 TO 19 PER RECORD; EACH ENTRY IS THREE WORDS LONG



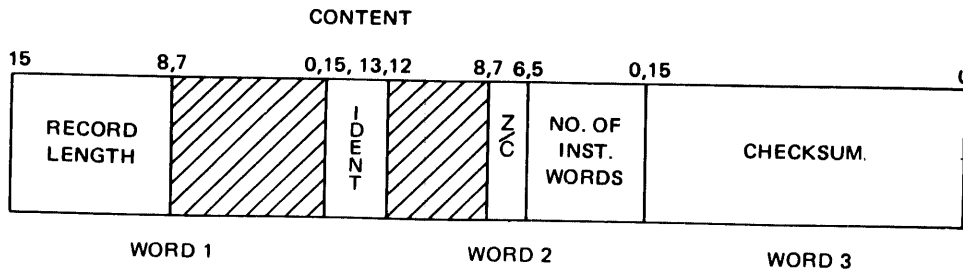
SYMBL: 5 CHARACTER EXTERNAL SYMBOL

SYMBOL ID. NO.: NUMBER ASSIGNED TO SYMBL FOR USE IN LOCATING REFERENCE IN BODY OF PROGRAM.



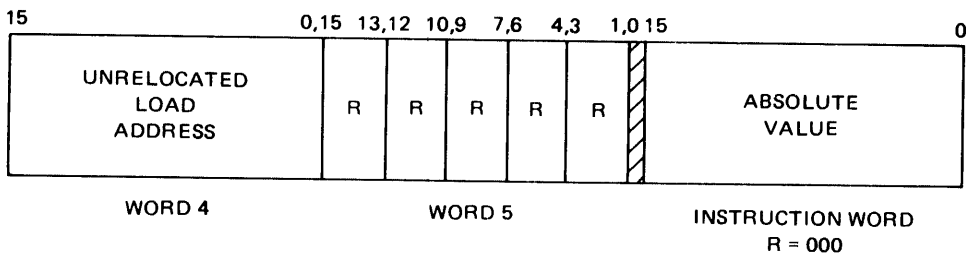
WORDS 4 THROUGH 6 REPEATED FOR EACH EXTERNAL SYMBOL (MAXIMUM OF 19 PER RECORD).

# DBL RECORD



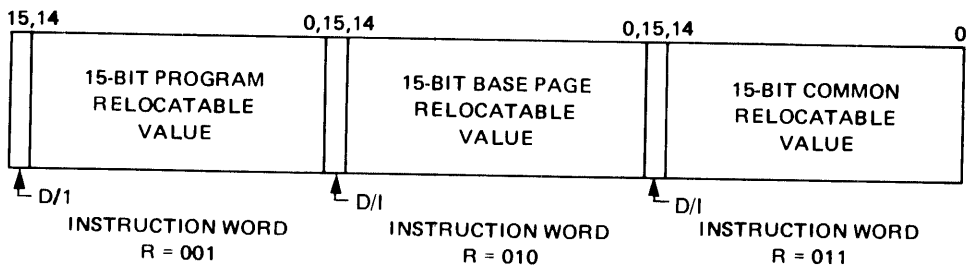
## EXPLANATION

RECORD LENGTH = 6-60 WORDS  
 IDENT = 011  
 Z/C: RELOCATION OF LOAD ADDRESS  
 = 0 FOR BASE PAGE  
 = 1 FOR PROGRAM  
 = 2 FOR ABSOLUTE  
 = 3 FOR COMMON  
 NO. OF INST. WORDS: 1 TO 45  
 LOADABLE INSTRUCTION WORDS PER RECORD

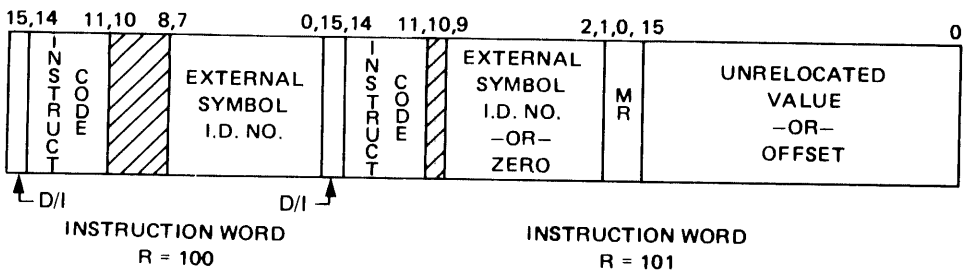


RELOCATABLE LOAD ADDRESS:  
 STARTING ADDRESS FOR LOADING THE INSTRUCTIONS WHICH FOLLOW;

R's: RELOCATION INDICATORS:  
 000 = ABSOLUTE  
 001 = 15-BIT PROGRAM RELOCATABLE  
 010 = 15-BIT BASE PAGE RELOCATABLE  
 011 = 15-BIT COMMON RELOCATABLE  
 100 = EXTERNAL REFERENCE  
 101 = MEMORY REFERENCE  
 110 = BYTE REFERENCE



R<sub>1</sub> IS RELOCATION INDICATOR FOR INSTRUCTION WORD<sub>1</sub>; R<sub>2</sub>, FOR INSTRUCTION WORD<sub>2</sub>; ETC.

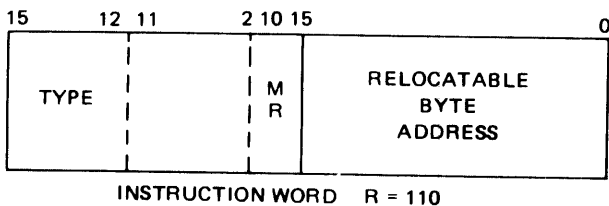


D/I: INDIRECT ADDRESSING

0 = DIRECT  
 1 = INDIRECT

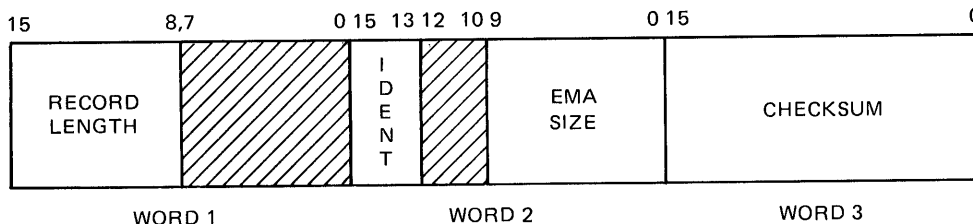
MEMORY REFERENCE INSTRUCTIONS USE TWO WORDS, WITHIN THE TWO-WORD GROUP, "MR" INDICATES RELOCATABILITY OF OPERAND SPECIFIED IN SECOND WORDS:

00 = PROGRAM RELOCATABLE  
 01 = BASE PAGE RELOCATABLE  
 10 = COMMON RELOCATABLE  
 11 = ABSOLUTE

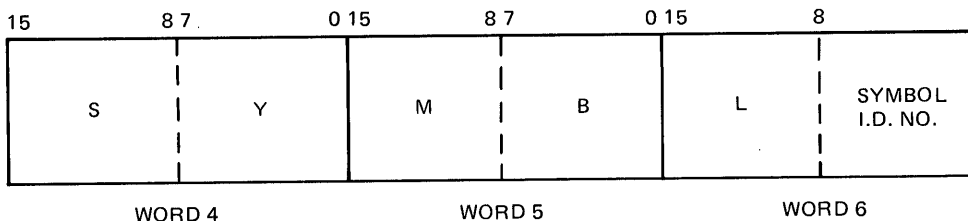




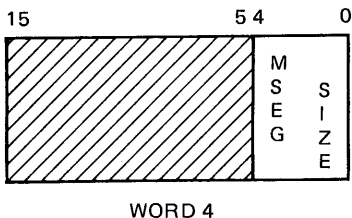
**EMA RECORD**



EXPLANATION  
 RECORD LENGTH = 7 WORDS  
 IDENT = 110



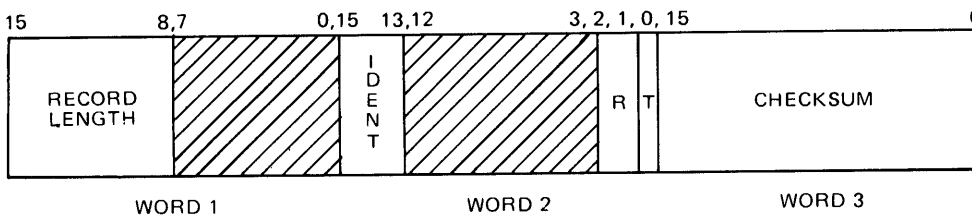
SYMBOL I.D. NO.: NUMBER  
 ASSIGNED TO SYMBOL FOR  
 USE IN LOCATING REFERENCE  
 IN COPY OF PROGRAM.



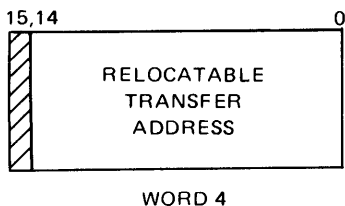
7700-189

**END RECORD**

CONTENT



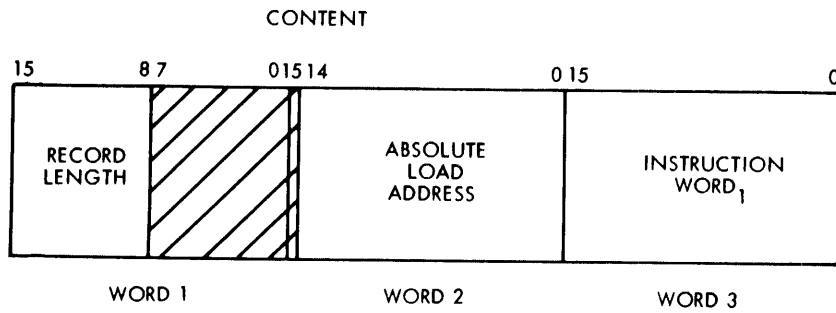
EXPLANATION  
 RECORD LENGTH = 4 WORDS  
 IDENT = 101



R: RELOCATION INDICATOR  
 FOR TRANSFER ADDRESS  
 = 0 IF PROGRAM RELOCATABLE  
 = 1 IF BASE PAGE RELOCATABLE  
 = 2 IF COMMON RELOCATABLE  
 = 3 IF ABSOLUTE  
 T: TRANSFER ADDRESS  
 INDICATOR  
 = 0 IF NO TRANSFER  
 ADDRESS IN RECORD  
 = 1 IF TRANSFER ADDRESS  
 PRESENT

TODS-18

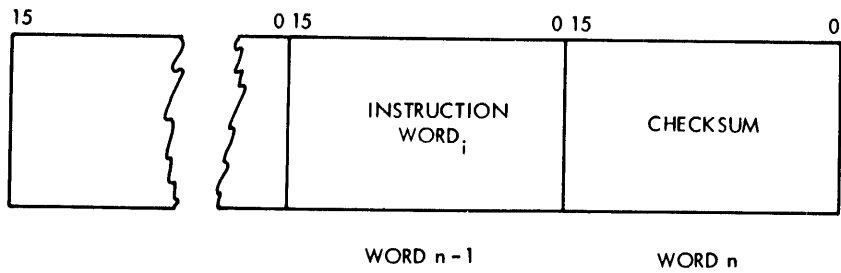
# ABSOLUTE TAPE FORMAT



## EXPLANATION

RECORD LENGTH = NUMBER OF WORDS IN RECORD EXCLUDING WORDS 1 AND 2 AND THE LAST WORD.

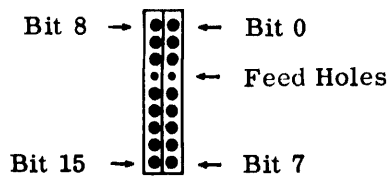
ABSOLUTE LOAD ADDRESS: STARTING ADDRESS FOR LOADING THE INSTRUCTIONS WHICH FOLLOW



INSTRUCTION WORDS: ABSOLUTE INSTRUCTIONS OR DATA

CHECKSUM: ARITHMETIC TOTAL OF ALL WORDS EXCEPT FIRST AND LAST

† On paper tape, each word represents two frames arranged as follows:



For existing installations that are upgrading from a previous RTE-III operating system to an RTE-IV software configuration, the RTE-IV features and enhancements described below are significant changes in design philosophy and the utilization of system services.

#### D-1. LOGICAL USER MAP

The operating system code and nearly all drivers are removed from the logical user map, thus allowing larger programs in RTE-IV than was possible in RTE-III. In RTE-IV, the user map is saved in the upper 32 words of the unmapped portion of the user's physical base page so that it is merely restored after being interrupted, rather than being rebuilt as in RTE-III (see Memory Management Section).

#### D-2. DRIVER PARTITIONS

A System Driver Area is available for privileged drivers, large drivers or those drivers that perform their own mapping. Driver partitions and the System Driver Area did not exist in RTE-III.

Driver partitions allow many drivers to be resident in physical memory but they share only one portion of the logical address space; that is, one driver partition is mapped only at the time it is needed. In contrast to RTE-III, RTE-IV driver partitions reduce the amount of address space used up by the driver in the user map (see Memory Management and I/O Sections).

#### D-3. TYPE 2, 3 AND 4 PROGRAMS

Type 2 (real-time) and Type 3 (background) programs have both Table Areas I and II and the System Driver area included in their address space.

Type 4 (background) programs, have only Table Area I included in their address space, although external references to Table Area II entry points will be resolved for access by cross-map instructions.

External references to system entry points (in Type 0 modules) will be resolved for Type 3 background programs only, and cross-map instructions must be used to access locations in the System Map.

## RTE-IV AND RTE-III DIFFERENCES AND COMPARISONS

Type 4 background programs did not exist in RTE-III.

Types 2, 3, and 4 programs may be segmented in RTE-IV. Only type 3 programs could be segmented in RTE-III.

### D-4. EXTENDED MEMORY AREAS

This feature allows RTE-IV programs to address memory arrays (similar to virtual memory arrays) that are beyond the standard 32K address limit. EMA arrays can be declared in both RTE-IV Assembly Language and FORTRAN-IV programs (see the Memory Management Section).

### D-5. MEMORY RESIDENT LIBRARY

Reentrant or privileged library routines located in the memory resident library can be accessed only by memory resident programs. All Type 6 and 14 library routines are treated as Type 7 utility routines when referenced outside of the memory resident area; that is, they are appended to the calling program and are placed in the disc resident relocatable library as Type 7 routines.

### D-6. FILE INPUT/OUTPUT

The loader (LOADR), FORTRAN-IV compiler and RTE-IV Assembler can perform I/O to or from FMP files in RTE-IV configurations.

### D-7. PARITY ERROR

If a parity error is detected in a user program that is running in a partition, the error will be reported and the partition removed from system use. Parity errors detected in the System Map will still halt, as in RTE-III configurations (see the EXEC Section).

### D-8. MEMORY AND I/O RECONFIGURATION

Memory and I/O specifications may be redefined during system boot-up to meet new on-site requirements, rather than going through a complete system generation. Partitions may be redefined to eliminate pages with parity errors, and devices may be reassigned to different I/O slots (see the I/O and Memory Reconfiguration Section).

## TABLE AREA I ENTRYPOINTS

EXEC  
 \$CIC  
 \$ERAB  
 \$IDLE  
 \$IDNO  
 \$LIBR  
 \$LIBX  
 \$LIST  
 \$MESS  
 \$MEU  
 \$MTM  
 \$OPSY  
 \$PVCN  
 \$SCD3  
 \$TBXX\*  
 \$UCON  
 \$UIN  
 \$ULLU  
 \$UPIO  
 \$WORK  
 \$XCIC  
 \$XDMP  
 \$XDMP  
 \$XEDQ  
 \$XEQ  
 \$YCIC

## TABLE AREA II ENTRYPOINTS

\$BATM  
 \$BGFR  
 \$CFR  
 \$CLAS\*  
 \$CMST  
 \$COML  
 \$DLP  
 \$DLTH  
 \$DVMP  
 \$DVPT  
 \$EMRP  
 \$ENDS  
 \$IDEX  
 \$LUAV\*  
 \$LUSW\*  
 \$MATA  
 \$MBGP  
 \$MCHN  
 \$MNP  
 \$MPFT  
 \$MPSA  
 \$MPSZ  
 \$MRMP  
 \$MRTP  
 \$PLP  
 \$RLB  
 \$RLN  
 \$RNTB\*  
 \$TRFR  
 \$SBTB  
 \$SDA  
 \$SDT2  
 \$TIME

\* built by the generator where 'XX' = 31 for a 7900 system  
 = 32 for a 7905/06/20 system

# RTE-IV PROGRAM TYPES

APPENDIX

F

Table F-1 provides a list of the default program types of the libraries and programs distributed with the RTE-IV operating system. The default program type is listed in the first column, and the remaining columns list the additional available program types. Each row of the table lists a program name or a library file name and indicates whether or not the corresponding program types available are allowed for that respective program or library (a "YES" meaning that the listed type is allowed, a "NO" meaning that the listed type is not allowed).

Note that several of the listed spool modules require SSGA access.

Table F-1. RTE-IV PROGRAM TYPES

PROGRAM OR LIBRARY FILE NAME	DEFAULT TYPE	TYPE 1 without TA II	TYPE 1 with TA II	TYPE 2	TYPE 3	TYPE 4	SSGA REQUIRED*
LOADR	3	NO	NO	YES	YES	NO	NO
PRMPT	1	YES	YES	YES	YES	YES	NO
RSPNS	1	YES	YES	YES	YES	YES	NO
AUTOR	2	YES	YES	YES	YES	YES	NO
\$CNFX	3	NO	NO	NO	YES	NO	NO
WHZAT	1	YES	YES	YES	YES	YES	NO
LGTAT	3	YES	YES	YES	YES	YES	NO
RT4GN	3	NO	NO	YES	YES	NO	NO
SWTCH	3	NO	NO	YES	YES	YES	NO
FMGR	3	NO	NO	YES	YES	NO	NO
D.RTR	2	YES	YES	YES	YES	YES	NO
EDITR	3	NO	NO	YES	YES	NO	NO
XREF	3	NO	NO	YES	YES	NO	NO
FTN4	3	NO	NO	YES	YES	NO	NO
ASMB	3	NO	NO	YES	YES	NO	NO
KEYS	3	YES	YES	YES	YES	YES	NO
KYDMP	3	YES	YES	YES	YES	YES	NO
#EMA	3	NO	NO	YES	YES	YES	NO
SAVE	3	NO	NO	YES	YES	NO	NO
RETOR	3	NO	NO	YES	YES	NO	NO
VERFY	3	NO	NO	YES	YES	NO	NO
COPY	3	NO	NO	YES	YES	NO	NO
MSAFD	3	NO	NO	YES	YES	NO	NO
JOB	2	NO	NO	YES	YES	NO	NO
GASP	3	NO	NO	NO	YES	NO	NO
SMP	18	NO	YES	YES	YES	NO	YES
EXTND	17	NO	YES	YES	YES	NO	YES
SPOUT	17	NO	YES	YES	YES	NO	YES
RLIB (RTE/DOS Relocatable Library)		YES	YES	YES	YES	YES	NO
BMLIB (Batch Monitor Lib.) (Spool Library)		YES	YES	YES	YES	YES	NO
CLIB (Compiler Library)		NO	YES	YES	YES	NO	NO
DECAR (Decimal String Library)		NO	NO	YES	YES	NO	NO
DECAR (Decimal String Library)		YES	YES	YES	YES	YES	NO
DBUGR (Debug Subroutine)		NO	NO	YES	YES	YES	NO
SYLIB (System Library)		YES	YES	YES	YES	YES	NO

\*Add 16 to the desired program type to obtain SSGA access.

ERROR TYPE =====	MANUAL SECTIONS =====
Operator Command Errors	3-5
Executive Error Messages	4-49
Memory Protect Violations	4-50
Dynamic Mapping Violations	4-51
Dispatching Errors	4-52
EX Errors	4-53
Unexpected DM and MP Errors	4-54
TI, RE and RQ Errors	4-55
Parity Errors	4-56
Other EXEC Errors	4-57
Disc Allocation Error Messages	4-58
Schedule Call Error Codes	4-59
I/O Call Error Codes	4-60
Program Management Error Codes	4-61
Logical Unit Lock Error Codes	4-62
Executive Halt Errors	4-63
Relocating Loader Error Codes	7-21
DBUGR Error Messages	11-13
Boot-Up and Reconfiguration Halts	12-14
Configuration Error Messages	12-15

**READER COMMENT SHEET**

**RTE-IV PROGRAMMER'S  
Reference Manual**

**92067-90001**

**June 1978**

We welcome your evaluation of this manual. Your comments and suggestions help us improve our publications. Please use additional pages if necessary.

**Is this manual technically accurate?**

**Is this manual complete?**

**Is this manual easy to read and use?**

**Other comments?**

---

**FROM:**

**Name** \_\_\_\_\_

**Company** \_\_\_\_\_

**Address** \_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_



FOLD

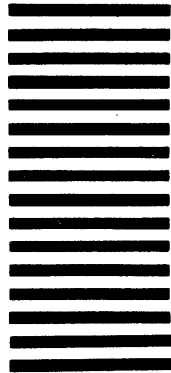
FOLD

**BUSINESS REPLY MAIL**

No Postage Necessary if Mailed in the United States Postage will be paid by

Hewlett-Packard Company  
Data Systems Division  
11000 Wolfe Road  
Cupertino, California 95014  
**ATTN: Technical Marketing Dept.**

FIRST CLASS  
PERMIT NO. 141  
CUPERTINO  
CALIFORNIA



FOLD

FOLD

PART NO. 92067-90001  
Rev. Code 1826  
Printed in U.S.A. 6/78

HEWLETT  PACKARD

Sales and service from 172 offices in 65 countries.  
11000 Wolfe Road, Cupertino, California 95014