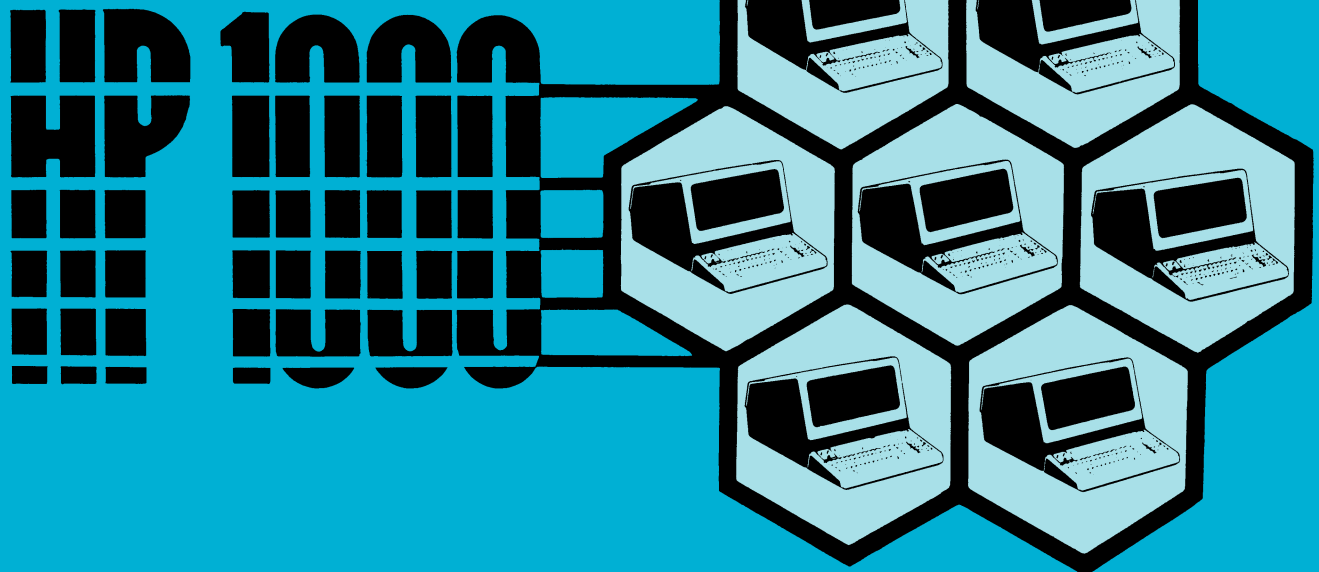# HP 92068A
RTE-IVB Programmer's
Reference Manual

RTE-IVB Programmer's

Reference Manual

Printing History

The Printing History below identifies the Edition of this Manual and any Updates that are included. Periodically, Update packages are distributed which contain replacement pages to be merged into the manual, including an updated copy of this Printing History page. Also, the update may contain write-in instructions.

Each reprinting of this manual will incorporate all past Updates, however, no new information will be added. Thus, the reprinted copy will be identical in content to prior printings of the same edition with its user-inserted update information. New editions of this manual will contain new information, as well as all Updates.

To determine what software manual edition and update is compatible with your current software revision code, refer to the appropriate Software Numbering Catalog, Software Product Catalog, or Diagnostic Configurator Manual.

Preface

This manual describes the scope, format, and use of the RTE-IVB operating system services available to user-written programs. It is intended to be the primary reference source for programmers who will be responsible for writing and maintaining software within the RTE-IVB operating environment.

This manual is divided into six sections as follows:

CHAPTER ONE gives a general description of the RTE-IVB operating system features.

CHAPTER TWO describes the use of the Executive communication module of RTE-IVB that provides user-written programs with the ability to communicate with the operating system. The Executive allows programs to perform I/O operations, manage programs, manage system resources, and obtain status information about the system.

CHAPTER THREE describes the use of files and the File Management System by user-written programs. By calling routines contained in the File Management System, user-written programs can define, access, position within, and purge disc or non-disc files.

CHAPTER FOUR defines and describes the use of the program segmentation feature of RTE-IVB.

CHAPTER FIVE defines and describes the use of the Extended Memory Area (EMA) feature of RTE-IVB.

CHAPTER SIX describes the format and use of the routines contained in the System Library. The System Library contains user-callable subroutines that provide a variety of utility and special purpose functions.

For additional information on the subsystems associated with the RTE-IVB operating system, the reader should refer to the appropriate reference manual as shown on the Documentation Map.

Two additional reference manuals that are directly related to the RTE-IVB operating system are briefly described below:

* RTE-IVB Terminal User's Reference Manual. This manual describes the features of the RTE-IVB operating system that are available to the user in an interactive mode.

* RTE-IVB System Manager's Manual. This manual contains the information necessary to plan, generate, and maintain the RTE-IVB operating system.
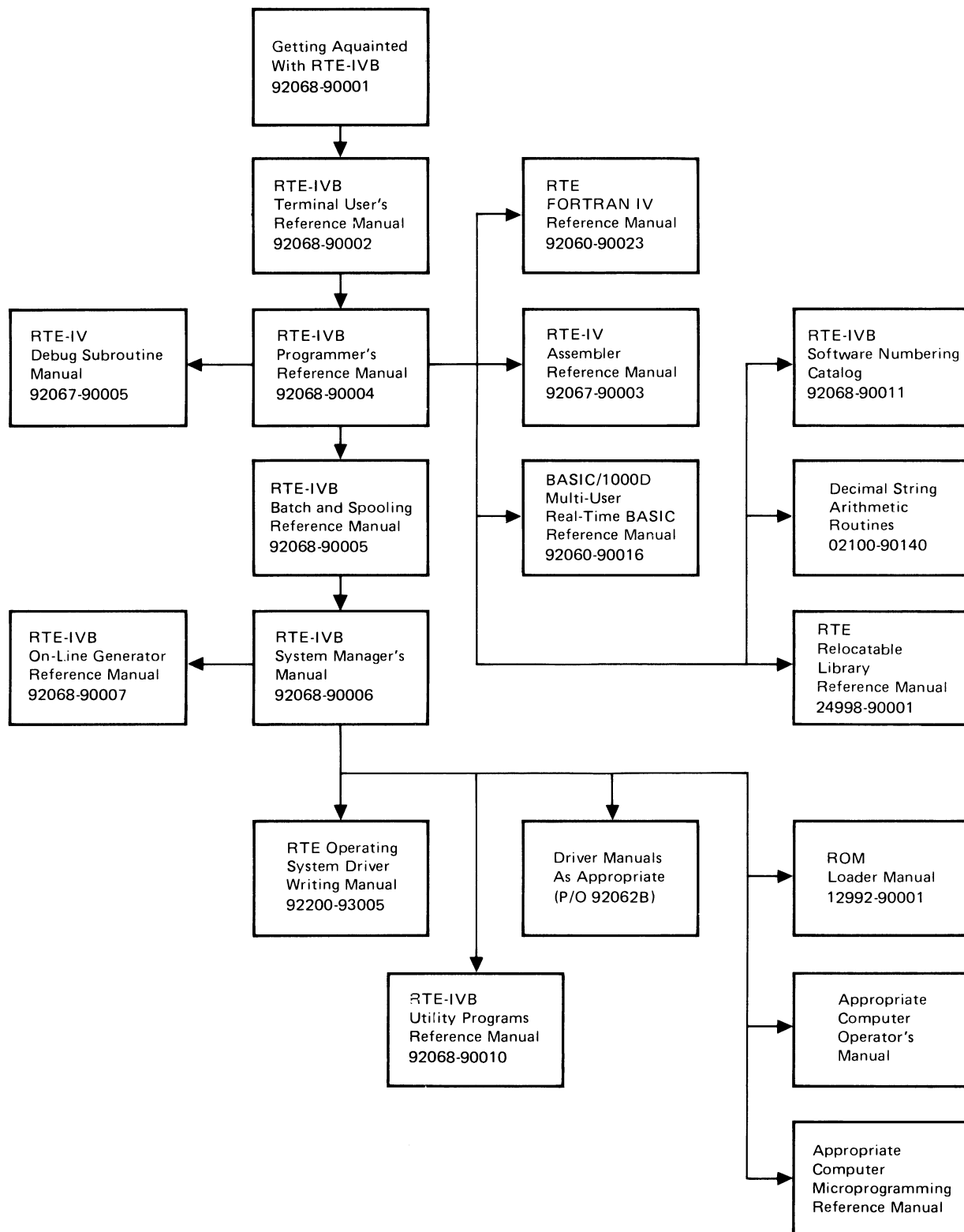
RTE-IVB Documentation Map

```
                    ┌─────────────────┐
                    │ Getting Aquainted│
                    │ With RTE-IVB     │
                    │ 92068-90001      │
                    └────────┬────────┘
                             │
                             ▼
                    ┌─────────────────┐      ┌─────────────────┐
                    │ RTE-IVB         │      │ RTE             │
                    │ Terminal User's │─────►│ FORTRAN IV      │
                    │ Reference Manual│      │ Reference Manual│
                    │ 92068-90002     │      │ 92060-90023     │
                    └────────┬────────┘      └─────────────────┘
                             │
┌─────────────────┐ ┌───────▼─────────┐      ┌─────────────────┐      ┌─────────────────┐
│ RTE-IV          │ │ RTE-IVB         │      │ RTE-IV          │      │ RTE-IVB         │
│ Debug Subroutine│◄│ Programmer's    │─────►│ Assembler       │─────►│ Software        │
│ Manual          │ │ Reference Manual│      │ Reference Manual│      │ Numbering       │
│ 92067-90005     │ │ 92068-90004     │      │ 92067-90003     │      │ Catalog         │
└─────────────────┘ └───────┬─────────┘      └─────────────────┘      │ 92068-90011     │
                             │                                        └─────────────────┘
                             ▼
                    ┌─────────────────┐      ┌─────────────────┐      ┌─────────────────┐
                    │ RTE-IVB         │      │ BASIC/1000D     │      │ Decimal String  │
                    │ Batch and       │      │ Multi-User      │─────►│ Arithmetic      │
                    │ Spooling        │      │ Real-Time BASIC │      │ Routines        │
                    │ Reference Manual│      │ Reference Manual│      │ 02100-90140     │
                    │ 92068-90005     │      │ 92060-90016     │      └─────────────────┘
                    └───────┬─────────┘      └─────────────────┘
                             │
┌─────────────────┐ ┌───────▼─────────┐                              ┌─────────────────┐
│ RTE-IVB         │ │ RTE-IVB         │                              │ RTE             │
│ On-Line Generator│◄│ System Manager's│                             │ Relocatable     │
│ Reference Manual│ │ Manual          │────────────────────────────►│ Library         │
│ 92068-90007     │ │ 92068-90006     │                              │ Reference Manual│
└─────────────────┘ └───────┬─────────┘                              │ 24998-90001     │
                             │                                        └─────────────────┘
```

Table Of Contents

Chapter 3
File Management Via FMP

Chapter 4

Program Segmentation

Chapter 5

EMA Programming

Chapter 6
RTE-IVB Library Subroutines

Appendix A
HP Character Set

Appendix B
System Communication Area and System Tables

Appendix C
I/O Tables and Processing

Appendix D
Record Formats

Appendix E
Differences In RTE Operating Systems

Appendix F
Program Types

Appendix G
Program States

Appendix H
DCB and Directory Formats

Appendix I
Memory Management and Related Tables

Appendix J
Session Monitor Tables

Appendix K
Class I/O Application Examples

Appendix L
Resource Numbers and Logical Unit Lock Examples

Appendix M
Schedule FMGR Programmatically

List Of Illustrations

List Of Tables

Chapter 1
General Description

Introduction

RTE-IVB is a disc based operating system that provides the supervisory functions necessary to coordinate requests for, and allocation of, system services and resources. Being a real-time system, RTE-IVB processes all decision and scheduling tasks internally unless overridden by user intervention. User requests for system action can be made by a "call" from within a program or interactively via an operator command.

As the major control element within the operating environment, RTE-IVB provides the user with various services and automatically handles the machine-related functions associated with each service. The major services provided by` RTE-IVB are briefly summarized below:

*   Executive Communication scheme that provides a communication link between user-written programs and system services.

*   Segmentation technique that allows a large program to be separated into a main program and related segments, thereby allowing it to execute in a memory partition smaller than its total size.

*   Resource Management capabilities that allow cooperating user-written programs to share system resources (files, I/O devices, etc.).

*   I/O scheme which allows a program to continue executing while its own I/O requests are being processed.

*   Program execution control that features multiprogramming (allows several programs to be active concurrently) and time-slicing (prevents compute intensive programs from dominating the CPU).

*   Partitioned memory technique that takes advantage of the hardware Dynamic Mapping System (DMS) to provide access to 2048k bytes of physical memory.

*   Extended Memory Area (EMA) that allows user-written programs to access large data arrays; the size of the arrays being limited only by the size of physical memory.

*   Operator interface that provides the user with the ability to control system action via operator commands.

In addition to the features listed above that are inherent to RTE-IVB, software modules are included with the operating system that provides the user with additional capabilities. These features are as follows:

* FORTRAN-IV Compiler
* RTE-IV Assembler
* Interactive Relocating Loader
* Interactive Editor
* Debug Utility
* Compile Utility
* Compile and Load Utility
* On-Line Generator
* Disc Save/Restore Utility
* Disc Backup and Update Utilities
* System Status Utilities
* File Management System
* Spooling System
* Session Monitor
* File Merge Utility
* Terminal Soft Keys Utility

The features described above that relate to the programmatic control of system action are described in later sections of this manual along with background information on the RTE-IVB operating system. The features of RTE-IVB that relate to the interactive control of system action are described in the RTE-IVB Terminal User's Reference Manual and the appropriate subsystem manuals (see documentation map). For information concerning the generation and configuring of the RTE-IVB operating system, refer to the RTE-IVB On-Line Generator Manual or obtain access to the RTE-IVB System Manager's Manual.

Multiprogramming and Timeslicing

RTE-IVB 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 in RTE-IVB decide when to execute programs that are simultaneously requesting system services and/or resources. The scheduling module places programs into a scheduled list in order of their priority (the highest priority program at the head of the list) and the dispatching module initiates the execution of the highest priority program. Programs with the same priority are scheduled on a first-come-first-serve basis. When the executing program completes, is terminated, or is suspended, it is removed from the scheduled list and the dispatching module transfers control to the next program with the highest priority. Note that the next program to be executed could have the same priority as the program that was just removed from the list.

The scheduled list can be logically divided into two areas by placing a time-slicing boundary at a priority level. Programs with priorities that place them above the boundary (higher priority, lower numerically) are executed in the linear fashion described above.

Programs with priorities that place them below the boundary (lower priority, higher numerically) are executed in a similiar fashion with one exception; programs are assigned an execution interval when they are scheduled. When a program exceeds its interval, it is moved within its priority level in the scheduled list.

Each priority level below the time-slicing boundary can be considered a queue. The program at the head of each priority queue represents the next program of that priority to be executed. When the execution of the program at the head of the queue is initiated, a maximum time interval for execution (time quantum) is calculated by the operating system. The program is allowed to execute until one of the following occurs:

1. The program leaves the scheduled list (I/O suspended, memory suspended, etc.)

2. A higher priority program is ready to execute.

3. The program exceeds its time quantum.

If a program leaves the scheduled list, its time quantum is assumed exhausted. When the program is again ready to execute, it is placed at the end of the queue within its priority in the scheduled list and a new time quantum is established.

If a higher priority program causes the suspension of a time-slicing program, the remaining portion of the suspended program's time quantum is saved in its ID segment. When the suspended program is scheduled to continue executing, the saved quantum value is restored.

When a time-slicing program exceeds its time quantum, it is placed at the end of the queue within its priority in the scheduled list and control is transferred to the new head of the queue.

The time value used to calculate the quantum and the time-slicing boundary are manipulated by the QU command described in the RTE-IVB Terminal User's Reference Manual; the considerations for manipulating them are discussed in the RTE-IVB System Manager's Manual. Figure 1-1 shows a diagram of scheduling with time-slicing.

Program Types

Programs within the RTE-IVB operating system are categorized according to where they reside (memory- or disc-resident), what type of memory partition they execute in, by the COMMON areas they have access to, and whether or not they may be duplicated. A program's type can also indicate whether it is a main program, program segment, or a subroutine (see Appendix F, "Program Types").

SCHEDULED LIST

```
                   ┌─────────────┐
                   │  PRIORITY   │
                   │     10      │
                   └─────────────┘
                          │
                          ▼
                   ┌─────────────┐
                   │  PRIORITY   │◀──────┐
                   │     20      │        \
                   └─────────────┘         \     PROGRAMS WITH SAME
                          │                  \    PRIORITY ARE SCHEDULED
                          ▼                   /   "FIRST-COME-FIRST-SERVE"
                   ┌─────────────┐          /
                   │  PRIORITY   │◀────────
                   │     20      │
                   └─────────────┘
                                        BOUNDARY SET AT LEVEL 40
          ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─
                          │
                          ▼
      ┌─────────────┐  ┌─────────────┐  ┌─────────────┐
      │  PRIORITY   │◀─│  PRIORITY   │◀─│  PRIORITY   │
      │     50      │  │     50      │  │     60      │
      └─────────────┘  └─────────────┘  └─────────────┘
          │    ▲                              ▲
          ▼    └──────────────────────────────┘
      ┌─────────────┐       ╲
      │  PRIORITY   │        ╲   PROGRAMS WITH SAME
      │     50      │         ╲  PRIORITY ARE SCHEDULED
      └─────────────┘            WITH TIMESLICING
```
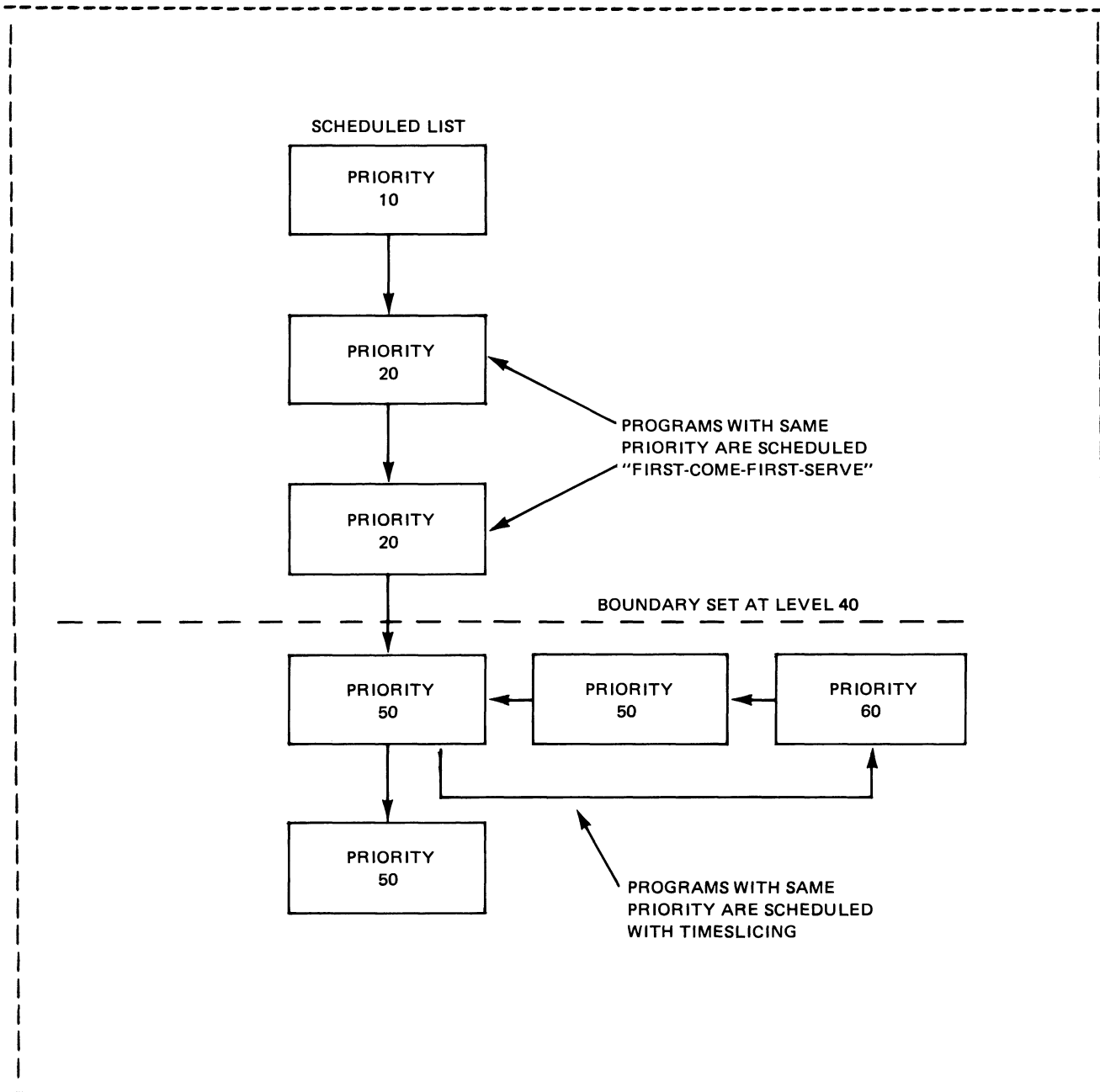
Figure 1-1.   Scheduling with Time-Slicing

A program's type is user-defined in the program definition statement (PROGRAM statement in FORTRAN, NAM statement in Assembly Language) or is assigned by the user when the program is loaded. If no type parameter is specified, a default value is determined and assigned by the loader or generator.

A   program's type, along with other necessary information, is maintained by the system in the program's ID segment (see Appendix B, "ID Segment Format").

Memory Management

The RTE-IVB operating system is written to take advantage of the hardware Dynamic Mapping System (DMS) available with the HP/1000 computers. The cooperation between the software operating system and the hardware mapping system allows access to 1024K words of physical memory.

The basic addressing space of the HP/1000 computer is 32,768 words (32K) as defined by the 15-bit address length used by the CPU. This is referred to as logical memory. The amount of memory actually installed in the computer system is referred to as physical memory. The DMS maps the 32K words of physical memory into logical memory by translating the 15-bit address through "memory maps".

Memory Maps

The page pointers contained in the map registers that comprise the memory maps are loaded by software modules within the operating system. Each map is configured to represent the 32 pages of physical memory (not necessarily contiguous) that contain the tables, buffers, data, program code, etc., necessary to perform specific processing tasks. Since there are four maps, four different 32 page sections of physical memory can be described simultaneously, with only one map being enabled at a time to indicate the section of memory currently in use. The maps are altered by the operating system to reflect dynamic changes in the operating environment, i.e., when a program is scheduled to execute, a map has to be configured to describe the physical memory pages it requires (known as the program's logical address space).

The four memory maps are classified by the type of processing tasks they are associated with. The maps are comprised of the following:

*   **System Map** ---The System Map describes the logical address space associated with the RTE-IVB operating system and its base page, COMMON, Subsystem Global Area, Table Area I and II, driver partition, System Driver Area (SDA), and System Available Memory (SAM). The system map is loaded during system initialization and is changed only to map in different driver partitions. Since the RTE-IVB operating system handles all interrupt processing, the system map is automatically enabled whenever an interrupt occurs.

*   **User Map** ---The User Map is associated with each disc resident program. It is a unique set of pages that describe the logical address space containing the program's code, the program's base page, Table Area I, driver partition, and optionally, Table Area II, System Driver Area, and COMMON.

All memory resident programs use a common set of pages that define the memory occupied by the memory resident program and its base page, the Memory Resident Library, Table Area I, driver partition, COMMON, and optionally, Table Area II and System Driver Area.

Each time a new memory or disc resident program is dispatched, the system reloads the User Map with the appropriate set of pages.

* **Port Map A and Port Map B**   The Port maps are associated with DCPC transfers. DCPC transfers are software assignable direct data paths between memory and a high speed peripheral device.

This function is provided by the Dual Channel Port Controller (DCPC). There are two DCPC channels, each of which may be assigned to operate with an I/O device. Port A Map is automatically enabled when a transfer occurs on DCPC channel 1, and Port B Map is enabled when a DCPC channel 2 transfer occurs.

DCPC transfers are accomplished by "stealing" CPU cycles instead of interrupting the CPU and transfering to an I/O service routine. Having separate maps associated with DCPC transfers, and having the transfer implemented by "cycle-stealing", facilitates multiprogramming since one program can be executing via the User Map while a DCPC transfer is in progress on another programs data buffer.

The Port Maps are reloaded by the system each time a DCPC channel is assigned for an I/O request. The Port Maps will be the same as the System Map or the User Map associated with the program being serviced, depending on the type of request. Once initiated, the DCPC transfer is transparent to the user since the currently enabled map (System or User) shares the CPU with the Port Maps, i.e., during a given instruction cycle (comprised of several CPU cycles) the System or User Map is enabled alternately with the Port Map.

Physical Memory

At generation time, the user plans physical memory allocations and 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. See the RTE-IVB On-Line Generator Manual and the RTE-IVB System Manager's Manual for a description of the procedures used to configure physical memory.

USER PARTITION M(1<M<64)
• • •
USER PARTITION 1(+ USER BP)
SAM EXTENSION
MEMORY RESIDENT PROGRAMS
RESIDENT LIBRARY
MEMORY RESIDENT BASE PAGE
DRIVER PARTITION N
• • •
DRIVER PARTITION 2
SYSTEM AVAILABLE MEMORY
OPERATING SYSTEM
TABLE AREA II
SYSTEM DRIVER AREA
BACKGROUND
REAL-TIME
SUBSYSTEM GLOBAL AREA
DRIVER PARTITION 1
SYSTEM AVAILABLE MEMORY
TABLE AREA I
SYSTEM BASE PAGE

COMMON

Figure 1-2. Physical Memory Allocations

The following is a brief description of the Physical Memory Configuration shown in Figure 1-2:

*   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.

    Base page links for the 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 Areas, SSGA, 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.

*   Table Area I - Contains the Equipment Table entries, Driver Mapping Table, Device Reference Table, Interrupt Table, the Disc Track Map Table, some system and HP subsystem entry points, and all Type 15 modules.

*   Driver Partition - An area, established at generation time containing one or more drivers. All driver partitions are the same length, and only one driver partition 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 when the system is generated.

*   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 (i.e., RTIOC, SCHED, EXEC, etc.).

*   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 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 Keyword Table, ID segments, ID
  Segment Extensions, Class Table, RN Table, Batch 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 others that are defined by Type 13 modules.

* System Available Memory - This is a temporary storage area used by
  the system for buffered, Class I/O, reentrant I/O, and parameter
  string passing.

* Partition - This is an area established by the user for a disc
  resident program to execute in. 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.

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 1-2). 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 a large background program does not use COMMON, it will
not be included in the User Map, providing the user with a larger
program area in the 32K-words 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 F.

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 enabled 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.

Memory Protection

Memory protection is provided by a combination of the Dynamic Mapping System and the Memory Protect Fence. DMS provides protection between program partitions by not allowing a program to access memory locations that are not defined by its memory map. The Memory Protect Fence prevents a program from addressing memory locations below a given address within its memory map.

A combination of DMS and the Memory Protect Fence, provides protection for the driver partition, Table Area I, and (optionally) System Driver Area, Table Area II, and COMMON by preventing stores and jumps to locations below a specified address.

The Memory Protect Fence indicates 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 memory resident area.

For programs using COMMON, the memory mapped includes all COMMON areas and the Memory Protect Fence is set at one of three possible locations, depending on the portion of COMMON being used. A hierarchy of protection is 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 1-3 expands the COMMON area and shows these three fence settings as A, B, and C respectively.

Partitions

Partitions are blocks of physical memory reserved for disc resident programs. Program partitions are defined during system generation and may be redefined during the reconfiguration process at system boot-up (see RTE-IVB System Manager's Manuel).
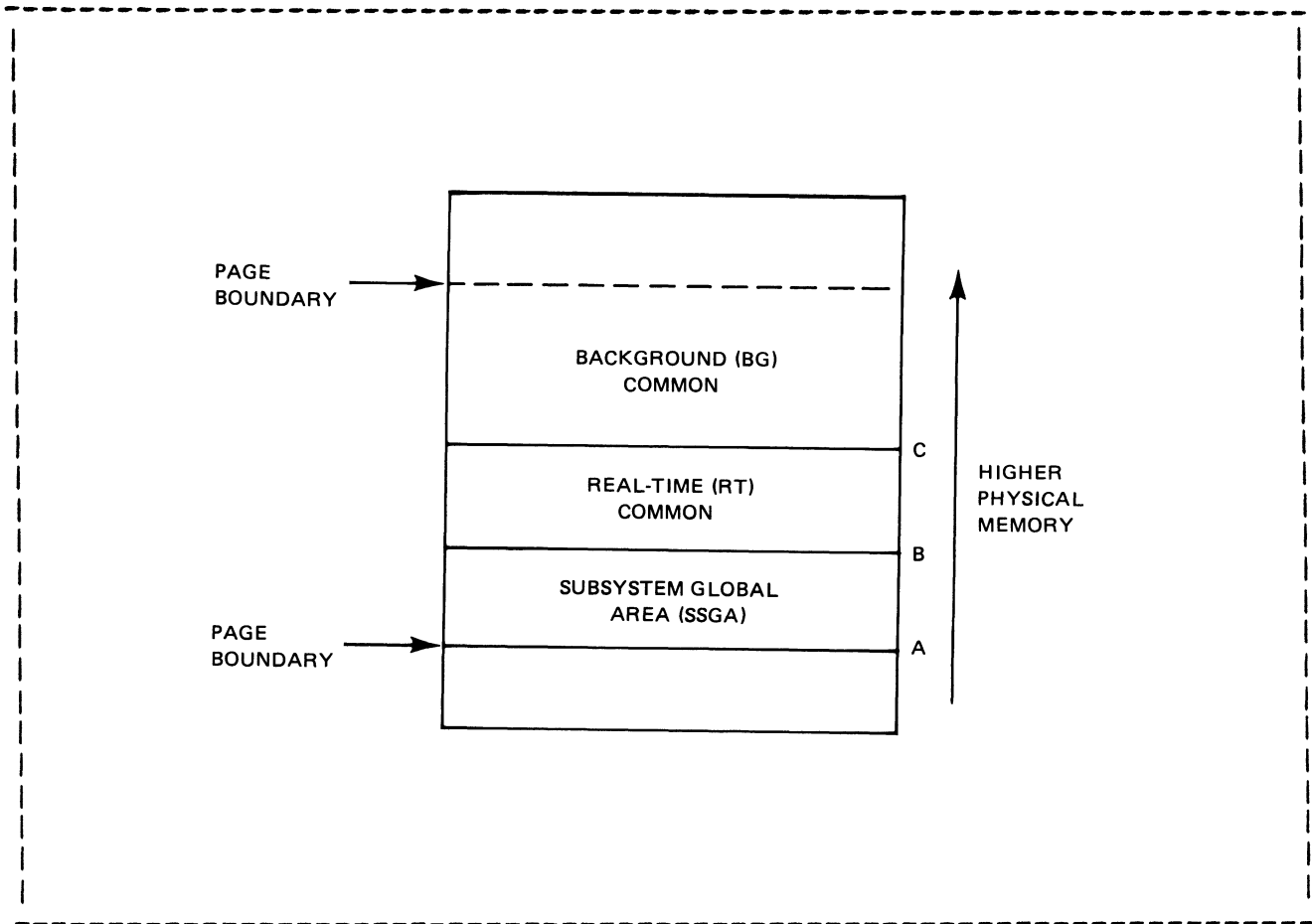
Figure 1-3. Memory Protect Fence Locations

The number of partitions depends on the amount of available physical memory and the size of the defined partitions. 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.

Mother partitions are large partitions (defined by linking Real-Time or Background partitions) designed to execute EMA programs (see Chapter 5, "EMA Programming"). When a mother partition is not in use, its subpartitions may be used by other programs.

Input/Output Processing

In the RTE-IVB operating system, centralized control and logical referencing of I/O operations effect simple, device-independent programming. All I/O and interrupt processing is controlled by the operating system with the single exception of privileged interrupts (privileged interrupts circumvent the system for faster response time).

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

In addition to the standard EXEC 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. An input without wait operation is a function of Class I/O (see below).

* Reentrant I/O -- allows a disc resident program to be swapped out of a memory partition and into disc storage when it is suspended for I/O. This permits any program to use the partition. The previous status of the swapped program is maintained so that when the reentrant I/O request has completed, and 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 two or more programs (mailbox I/O) or between programs and I/O devices. Class I/O permits a program to continue execution concurrently with its own I/O (I/O without wait).

Hardware Considerations

For a full understanding of the software I/O characteristics of the RTE-IVB operating system described in this manual, 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 associated with a computer I/O select code. (Select code refers to a physical card slot in the backplane of the computer.)

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 of the computer I/O select codes. Controller Interrupts are directed to specific computer memory locations based on their select codes.

For details on the hardware I/O organization, consult the appropriate computer reference manual (see documentation map).

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 which I/O device requests are to be directed to. 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.

An LU is associated with an EQT entry and a subchannel. Some I/O devices have EQT entries with one subchannel designation (i.e., line printers) and are referenced by a single LU number. Other devices (disc drives and CRT terminals) have EQT entries with several subchannel designations, with an LU assignment for each subchannel. When a user makes an I/O request specifying an LU number, he can be addressing a total device (line printer) or a subsection of a device (left CTU of a terminal).

Logical Unit numbers are decimal integers that range from 1 to 254, LU numbers greater than 63 may only be accessed when operating under Session Monitor control. The functions of Logical Units 0 through 6 are predefined in the RTE-IVB system as follows (could be system or session LUs):

   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 for the magnetic tape device if one is present on the system. Peripheral disc subchannels must be assigned logical units greater than 6 and less than 64. If the Session Monitor is used, terminal LUs must be defined between 6 and 99. Additional logical units may be assigned for any function desired. On-line changes to existing LU assignments can be made by using the LU operator command described in the RTE-IVB Terminal User's Reference Manual.

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, restores registers and maps, 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 work after the power fail system and operating system are reenabled.

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, waiting for the "operation done" indication from the device's controller.

Privileged Interrupt Processing

RTE-IVB allows interrupts from specified controllers to bypass the standard system I/O processing modules and be processed instead by special routines. These I/O operations are therefore "privileged". Privileged interrupt processing is established for time-critical tasks such as power-fail processing or processing communication over a modem link.

I/O controller interrupts that are allowed to be processed as privileged are established at generation time. A special I/O card is placed in the backplane of the computer to physically separate the privileged interrupt controllers from the standard system-processed controllers. The location of the "privileged-fence" card (if present) is stored in the System Base Page. Privileged controllers are located below the fence (higher priority) and nonprivileged controllers are located above fence (lower priority).

When a privileged interrupt occurs, the privileged fence card holds off nonprivileged interrupts. The system operates in the "hold-off-interrupt" (not interrupt-disabled) state until the privileged interrupt has been processed.

The hold-off-interrupt state does not disable the interrupt system. It allows a higher priority privileged interrupt to interrupt a lower priority privileged interrupt. A nonprivileged interrupt is not allowed to interrupt a privileged interrupt. For more information on privileged driver characteristics, see the RTE-IVB Driver Writing Reference Manual.

Resource Management

The RTE-IVB operating system allows cooperating programs to manage common system resources. A resource is defined to be any element within the RTE-IVB environment that can be accessed by a user's program, e.g., an I/O device, a file, a program, or a subroutine. Cooperation between programs is established by coding them to take advantage of a utility subroutine (RNRQ) which allocates, deallocates, locks, and unlocks an arbitrary identification number known as a Resource Number (RN).

Within the cooperating programs, the RN is logically related to a particular resource by the statement stucture that comprises each program. When a program seeks exclusive access to a resource, it requests the system to lock the related RN. (The request is granted only if no other program has already locked the RN; otherwise, the program is suspended until the RN is unlocked.) When it is finished with the resource, the program requests the system to unlock the RN so that other programs can lock it.

An RN is not physically assigned to any one resource. The logical association between the RN and a resource is accomplished only by the context of the statements within the program using the RN. The RN is known to the system but the resource with which it is associated is not; therefore, all cooperating programs must agree on which RN is associated with which resource. The use of resource numbers is described in Chapter 6 of this manual.

Session Monitor

If the appropriate software modules (see the RTE-IVB System Manager's Manual) are included at generation time, the RTE-IVB operating system can be configured to provide controlled access to system services and resources by multiple users.

With the Session Monitor configured into the system, the user is required to "log on" to a station (terminal) using an account ID assigned to him by the system manager. At system initialization, the system manager sets up an account file on disc which describes the I/O devices and the command capabilities assigned to each account ID. When a user has successfully logged on, a Session Control Block (SCB) is established for his "session" using information taken from the account file. The Log-on Processor provides the session user with a copy of FMGR, and each command entered is checked to verify that the user has the capability to use the command as specified in his SCB.

The I/O devices that the session user has access to must be defined in a section of his SCB known as the Session Switch Table (SST). The SST entries are taken from the session user's specific account file entry (LUs associated with the user's ID) and from a table in the account file common to all users known as the Configuration Table (LUs associated with each session station). The function of the SST is to link the session LUs on which a user makes an I/O request, to the system LUs that the I/O request will actually be directed to. When the user makes an I/O request, his SST is searched for the LU specified in the request. If the LU is found, it is switched to the associated system LU and the request is processed. If the requested LU is not found, an error message is returned, indicating that the LU is not defined for the user's session. The SST therefore defines the system I/O devices that the session user has access to.

When operating in the session environment, access to disc cartridges is controlled by identifying them as belonging to a particular user or group of users. Disc cartridges can be mounted as:

* Private cartridges - allows Read/Write access only by the session user who mounted it.

* Group cartridges - allows Read/Write access only by members of the group that the cartridge is mounted to.

* System cartridges (LU2 and LU3) - allows Read/Write access by the system manager and non-session programs, read-only access by session users. Can only be mounted or dismounted by system manager.

* System cartridges (global) - allows Read/Write access by any system user. Can only be mounted or dismounted by system manager.

Note that the system manager has access to all cartridges.

Within the account file is a table (set up by system manager) that indicates disc LUs that are available to session users (Spare Disc Pool). If a session user wishes to mount a spare cartridge, and has the capability to do so, a disc can be allocated from the pool (a "working" copy of the Free Disc Pool maintained in memory) and an entry is made in his SST indicating that he has access to that cartridge. The entry in the disc pool is also flagged indicating the disc is allocated.

Disc cartridges are mounted and dismounted via operator commands discussed in the RTE-IVB Terminal User's Reference Manual. The formats of the session related tables are shown in Appendix J.

Language Support

The language translators available for user program development in the RTE-IVB operating environment are briefly described below. For further information on these languages, refer to the appropriate reference manual.

* RTE FORTRAN IV---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---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 HP CPUs. For further information, see the RTE-IVB Assembler Reference Manual.

* RTE MICRO-ASSEMBLER---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---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. For further information, see the Multi-User Real-Time BASIC Reference Manual.

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.

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

* 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 Chapter 2 of this manual for complete descriptions and format considerations associated with EXEC calls.

File Management System

The File Management Package (FMP) allows the user to manipulate I/O devices and files. The user interface to the FMP can be either interactive (using FMGR commands described in the RTE-IVB Terminal User's Reference Manual) or programmatic, (using FMP calls described in Chapter 3 of this manual).

The FMP library contains routines that are called from user programs and used to manipulate disc and non-disc files. Using calls to these routines, the user can create, access, purge, and obtain the status of files.

Files are classified according to the record format within the file and the type of data the system expects to find in each record. A file's type is defined when it is created and this information is placed in its file directory entry. When a file is accessed, this information is used by FMP to determine the files characteristics and initiate the appropriate action as specified by the type of file it is manipulating.

The user must also be aware of file types. Certain files are formatted to facilitate random access (fixed-length records), and others are formatted for sequential access (variable-length records). User-written programs should be coded to recognize and take advantage of a file's characteristics if efficient file manipulation is to be accomplished.

A file can contain up to (2**31)-1 records and can have a total size up to 32767 X 128 blocks (1 block = 256 bytes). For files with fixed-length records, the file size is defined at creation and cannot be expanded. For files with variable-length records the base file size is defined at creation and the file is extendable as needed.

The following is a summary list of the services available to user programs via FMP calls:

* File creation (disc file only)

* Opening files for specific modes of access.

* Read and write access to files.

* Positioning to records within a file.

* Closing files to access.

* Purging files from the system.

* Obtain position and status information on files.

* Renaming of files.

* Obtain disc cartridge list.

Refer to Chapter 3 of this manual for complete description and format considerations associated with FMP calls.

System Library

The System Library, included with the RTE-IVB Operating System, is a collection of relocatable routines that can be used to interface user programs with system services. Some of the important services available with RTE-IVB are implemented by calling these routines. For example:

* EMA programming

* Reentrant I/O processing

* Resource management

In addition, the System Library contains various general purpose routines that perform convenient services for the programmer. These services include:

* Data conversion and string manipulation

* System status query

* Session environment related services

For more details on the System Library routines, refer to Chapter 6 of this manual.

Spooling System

The Spooling System operates in conjunction with the File Management System to provide session input and output spooling, batch job processing with spooling or, through user program calls, to provide programmatic spooling without batch processing. Spooling means that jobs or data are placed on disc files for input and data is sent to disc for output. This allows input and output to be performed independently of each other and of job processing. Spooling allows programs to be processed without having to wait for completion of input from or output to slow devices. The entire spool process can proceed automatically with virtually no user intervention, or it may be directly controlled during its various phases.

Spooling can be used to increase the throughput of a job stream or program that is limited by the idle time of slow peripheral devices. It does this by allowing programs to perform I/O to disc files rather than to the slower peripheral devices. The system then manages the I/O between the disc files and the peripheral devices to assure that all I/O reaches its proper destination.

The Spooling System provides the following capabilities:

- Opens and closes the disc files known as spool files; after close, optionally writes the file contents to a user-selected non-disc device for output.

- Keeps a record of the current status of all jobs and spool files in the system.

- Translates non-disc device references in program I/O calls to references to spool files.

For details on the interactive and programmatic services provided by the Spooling System, refer to the RTE-IVB Batch and Spooling Reference Manual.

System Utility Programs

Standard system utilities are on-line programs that run under the RTE-IVB 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 following brief descriptions of the utilities available with RTE-IVB are intended to serve as an introduciton. For more details, refer to the RTE-IVB Terminal User's Reference Manual or the RTE-IVB System Manager's Manual.

* Interactive Editor (EDITR)---The Interactive Editor (EDITR) is used to create and edit ASCII files. EDITR can operate in interactive mode (accepts commands from a keyboard device) batch mode (accepts commands from a job command file), or from a user command file spooled via the SL command.

* Compile Utility (COMPL)---The compile program enables the user to invoke any of the compilers or the assembler. The utility will select the appropriate compiler or assembler by reading the control statement (first statement required in the program). Optionally, the programmer can specify a new control statement when running the compiler.

* Relocating Loader (LOADR)---The Relocating Loader program accepts relocatable programs and outputs absolute load modules in conformance with loader control 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.

* Compile and Load Utility (CLOAD)---CLOAD performs a composite function. It invokes the appropriate source translator (as COMPL does) and inputs the relocatable results to the loader (LOADR) for the creation of an executable memory-image program.

* On-Line Generator (RT4GN)---The On-Line Generator permits use of an existing RTE-IVB System to configure a new RTE-IVB system according to user specifications. Generation can be directed from an answer file, logical unit or operator console.

* 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 transfer or copy operation, and to initialize a disc cartridge.

* Disc Update---The Disc Update process can be used to replace disc cartridge files with files stored on an HP minicartridge tape. The primary purpose is to update master software discs with either HP software distributed on minicartridges or user-written program modifications.

* System Status Program (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 programs and their current status, or a list of all partitions with their sizes and current status (occupied or non-occupied).

* File Merge Utility (MERGE)---The Merge utility provides a quick and simple way for file libraries to be constructed. Taking its input from a command file or interactively from a terminal, the Merge utility concatenates files to a destination file, creating a new file if none exists.

* Save/Restore Utility (WRITT, READT)---The disc Save/Restore utility allows the user to save and restore peripheral disc cartridges through the use of magnetic tape. Saving user files or cartridges on mag tape is accomplished through the WRITT program and restoring them to the system is accomplished through the READT program.

* Interactive Debugger (DBUGR)---The DEBUGR 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. Refer to the DBUGR Reference Manual for a complete description of all DEBUGR functions.

* Soft Key Programs (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/48A Display Station. Soft Keys provide the capability to enter entire sequences of commands with a single key stroke. The advantages are speed of entry and a significant reduction in operator errors during terminal entry sessions.

* Track Assignment Table Log Program (LGTAT)---The LGTAT program logs and displays the status of the system and auxiliary (only) disc tracks.

Chapter 2
Executive Communication

Introduction

An executing program may request various system services via a call to the EXEC processor, specifying the request in a parameter string. Initiation of the call causes a memory protect violation interrupt (enables the System Map) and transfers control to the EXEC module. The EXEC module determines the type of request from the parameter string and initiates processing if the request was legally specified. A summary of the services available to the user via EXEC calls is shown below in the order of their presentation:

* Standard I/O

* Disc Track Management

* Program Management

* System Status

* Class I/O

EXEC Call Formats

In RTE FORTRAN-IV, EXEC calls are coded as either function or subroutine call statements. In RTE Assembly Language, EXEC calls are coded as JSB EXEC, followed by a series of parameter definitions. For any FORTRAN call statement, the object code generated is equivalent to the corresponding Assembly Language object code.

The discussion that follows shows the general formats used to code EXEC calls into Assembly Language and FORTRAN-IV programs. After each general format is an example of its use. Each example shows the same action being initiated, e.g., reading 10 words from LU 5 and placing them into the first 10 words of a 100 word buffer.

```
+------------------------------------------------------------+
|                                                            |
|    Assembly Language Format:                               |
|                                                            |
|             :                                              |
|             EXT EXEC      Link calling program to EXEC module. |
|             :                                              |
|             JSB EXEC      Transfer control to EXEC module. |
|             DEF RTN       Define return point from EXEC.   |
|             DEF P1        Define address of first parameter. |
|             :                                              |
|             DEF Pn        Define address of n-th parameter. |
|    RTN      :             Continue execution of program.   |
|             :                                              |
|    P1       define parameter value.                        |
|             :                                              |
|    Pn       define parameter value.                        |
|             :                                              |
|                                                            |
|                                                            |
|    Example:                                                |
|                                                            |
|             :                                              |
|             EXT EXEC                                       |
|             :                                              |
|             JSB EXEC                                       |
|             DEF NSI                                        |
|             DEF ICODE                                      |
|             DEF ICNWD                                      |
|             DEF IBUFF                                      |
|             DEF ILEN                                       |
|    NSI      :                                              |
|             :                                              |
|    ICODE    DEC 1         Request Code Word.  1 = Read.    |
|    ICNWD    DEC 5         Control Word.  LU = 5.           |
|    IBUFF    BSS 100       100 word buffer where data is placed. |
|    ILEN     DEC 10        10 words are to be read.         |
|             :                                              |
|                                                            |
|    The parameters (P1-Pn)  defined in an Assembly Language |
|    EXEC  call  are  position  dependent  and the number of |
|    parameters (n) is dependent on the function of the EXEC |
|    call.  For  example,  the  Read  example  shown  above  |
|    requires four parameter definitions  (n = 4)  and they  |
|    must be defined in the order shown.                     |
|                                                            |
+------------------------------------------------------------+
```

```
+----------------------------------------------------------------+
|                                                                |
|   FORTRAN-IV Subroutine Call Format:                           |
|                                                                |
|            :                                                   |
|           Pl          Array parameters are defined in          |
|            :          DIMENSION or COMMON statements.          |
|           Pn          One-word parameters are defined as       |
|            :          integer variables.                       |
|           CALL EXEC(Pl,...,Pn)                                 |
|            :                                                   |
|                                                                |
|                                                                |
|   Example:                                                     |
|                                                                |
|            :                                                   |
|           DIMENSION IBUFF(100)   100 word buffer.              |
|            :                                                   |
|           ICODE=1                Request Code Word.  1=Read.    |
|           ICNWD=5                Control Word.  LU=5.           |
|           ILEN=10                10 words are to be read.       |
|            :                                                   |
|           CALL EXEC(ICODE,ICNWD,IBUFF,ILEN)                    |
|                or                                              |
|           CALL EXEC(1,5,IBUFF,10)                              |
|            :                                                   |
|                                                                |
|   In  FORTRAN-IV  EXEC  calls, one-word integer parameters     |
|   can be defined as integer  variables or actual  integer      |
|   values. Array parameters must be defined in a DIMENSION      |
|   statement.   As in the Assembly Language format, param-      |
|   eters are function and position dependent.                   |
|                                                                |
+----------------------------------------------------------------+
```

```
+-----------------------------------------------------------------+
|                                                                 |
|   FORTRAN-IV Function Call Format:                              |
|                                                                 |
|          :                                                      |
|          Pl          Array parameters are defined in            |
|          :           DIMENSION or COMMON statements.            |
|          Pn          One-word parameters are defined as         |
|          :           integer variables.                         |
|          REG         Defined as real variable comprised of      |
|          :           two integer variables.                     |
|          :                                                      |
|          REG=EXEC(Pl,...,Pn)                                    |
|          :                                                      |
|                                                                 |
|                                                                 |
|   Example:                                                      |
|                                                                 |
|          :                                                      |
|          DIMENSION IBUFF(100),IAB(2)                            |
|          EQUIVALENCE (REG,IAB(1))                               |
|          :                                                      |
|          ICODE=1                                                |
|          ICNWD=5                                                |
|          ILEN=10                                                |
|          :                                                      |
|          REG=EXEC(ICODE,ICNWD,IBUFF,ILEN)                       |
|            or                                                   |
|          REG=EXEC(1,5,IBUFF,10)                                 |
|          :                                                      |
|                                                                 |
|   The purpose  for using the  function call   format is to     |
|   obtain the contents of the A- and B-registers after the      |
|   EXEC call has been executed.   The A- and B-register         |
|   contents must be returned as a real variable.  In the        |
|   example, the real variable REG has been equated to the       |
|   integer variables IAB(1) and IAB(2), making the A-           |
|   register contents available in IAB(1) and the B-register     |
|   contents available in IAB(2).  These two values can then     |
|   be examined for error indications (see EXEC CALL ERROR       |
|   RETURNS below).                                              |
|                                                                 |
+-----------------------------------------------------------------+
```

EXEC Call Error Returns

EXEC calls that are in error will cause the offending program to be aborted if the error was severe enough. If an error is not severe, it will either abort the program or, at the user's option, report the error to the program and allow it to continue executing. Shown below is a partial summary of the EXEC errors. The errors marked with an "*" are considered severe and will always cause the program to be aborted. The error code is placed in the user's Session Control Block (SCB) words 5 thru 8.

Error Code	Error Type	Error Code	Error Type
* MP	Memory Protect	SC	Scheduling
* DM	Dynamic Mapping	LU	LU Lock
* RQ	Request Code	IO	Input/Output Error
* DP	Dispatching	DR	Disc Allocation
* RE	Reentrancy	RN	Resource Number
PE	Parity		

A detailed explanation of EXEC call error messages is given at the end of this section (see EXECUTIVE ERROR MESSAGES).

No-Abort Option

If the user wishes non-severe errors reported to the calling program and the program to continue executing, the return point from the EXEC call is altered by setting the "no-abort" bit (bit 15) in the request code word (ICODE=ICODE+100000B). This causes the system to execute the first line of code (it must be a one-word instruction) following the EXEC call if an error occurs. When the "no-abort" bit is set and an error occurs, the error code is not placed in the user's SCB. If there is no error, the second line of code following the EXEC call is executed.

The following segments from a sample FORTRAN program demonstrate the use of the altered error return points:

```
                              :
                              parameter definitions
                              :
                              ICODE=ICODE+100000B  <--set "no-abort" bit
                              :
                              CALL EXEC(ICODE,ICNWD,IBUFF,ILEN)
     error return-->          GO TO 100
no error return-->      10    :
                              :
                      100     error processing
                              :
```

In FORTRAN, only the GO TO statement should be placed after a no-abort EXEC call; any other command would cause error information to be lost. The GO TO statement also must not reference the next statement, i.e.,

```
                    :
              CALL EXEC(ICODE+100000B,ICNWD,IBUFF,ILEN)
              GO TO 100
        100   :
              :
```

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

As discussed previously (see FORTRAN-IV Function Call Format), if a non-severe error return is made to a program, the A- and B-registers contain the ASCII error codes. 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.). Note that the no-abort error return will return control to the calling program when a parity error (PE) occurs, if the error is on the system or auxiliary disc (LU 2 or LU 3). In this case, the B-register will be set to -1. If a parity error occurs on a disc other than LU 2 or 3, the error is considered "severe" and the calling program will be aborted.

In RTE Assembly Language, the A- and B-registers can be manipulated directly to determine error information. In RTE FORTRAN-IV, an HP-supplied subroutine (ABREG) can be used to obtain the contents of the A- and B-registers. The ABREG subroutine would be used with an EXEC subroutine call; the use of the EXEC function call would make ABREG unnecessary. The use of ABREG is shown in the following example:

```
                         :
                         parameter definitions
                         :
                         CALL EXEC(ICODE+100000B,...)<--set"no-abort"bit
    error return-->      GO TO 100
no error return-->  10   :
                         :
                    100  CALL ABREG(IA,IB)
                         error processing
                         :
```

After the return from ABREG, IA contains the contents of the A-register and IB contains the contents of the B-register. The calling program can then implement error processing accordingly.

Note that if an EXEC call is successful, the A- and B-registers can contain pertinent information about the status of the call. ABREG can be used to retreive this information also. The contents of the A- and B-registers after a successful call are discussed with the descriptions of the individual EXEC calls later in this section.

ABREG is described in the DOS/RTE Relocatable Library Reference Manual.

NOTE

DO NOT set the no-abort in an EXEC function call. For example, DO NOT USE the following technique:

```
        :
        :
        REG=EXEC(ICODE+100000B,...)
        GO TO 100
        :
        :
100 error processing
        :
```

The reason for not using this method is that when the function call is compiled, a double-word store instruction is generated to save the contents of the A- and B-registers. This instruction is placed directly after the EXEC function call. If the EXEC call is successful, control is returned to the second word after the call. This would be the second half of a double word instruction and would therefore be erroneous.

No-Suspend Option

Certain programs in a real-time environment may be considered too important to be suspended due to an downed I/O device. This capability is provided by the "no-suspend" I/O option. Programs can check device status before executing I/O request. However, if the device goes down due to (or during) the I/O request, the calling program will be suspended unless the no-suspend bit (bit 14) in the I/O request code word (ICODE=ICODE+40000B) is set. The no-suspend bit can be used for the following I/O EXEC requests:

        Standard I/O EXEC calls: 1,2,3

        Class I/O EXEC calls: 17,18,19,20

When the no-suspend bit is set, the calling program resumes execution at the first line of code (it must be a one-word instruction) following the EXEC call if an error occurs. If there is no error, the second line of code following the EXEC call is executed. The use of the altered return points, and the A- and B-Registers returns are the same for the no-suspend I/O option and the no-abort option. The same restrictions on the use of the no-abort bit hold for the no-suspend bit. The alternate return label can also be uesd for the no-suspend option.

The following section from a sample FORTRAN program demonstrates the use of the no-suspend bits:

```
                        :
                parameter definitions
                        :
                ICODE =ICODE+40000B  <--set no-suspend bit.
    :
                        CALL EXEC (ICODE,ICNWD,IBUFF,ILEN)
error return----->      GO TO 100
no error return--> 10   :
                100 error processing
                        :
```

The following table outlines the various actions taken by the system depending on the condition of the system at the time of the I/O request when the no-suspend bit is set.

BUFFERED I/O REQUESTS		
System Condition at request time.	Action Taken	A- + B-Registers
Requested EQT or LU is LOCKED.	Program resumes at first line after call.	IO13
Requested EQT or LU is DOWN.	Program resumes at first line after call.	IO14
No buffer memory (Standard I/O EXEC calls 1,2,3 only).	Request continues to be processed unbuffered.	meaningless

UNBUFFERED I/O REQUESTS		
External Condition	Action Taken	A- + B-Registers
Device goes down during I/O processing.	Program resumes at first line after call.	IO14
Operator downs the device using the "DN" command.	Program resumes at first line after call.	IO14

EXEC Description Conventions

In the subsections that follow, certain conventions are used to describe EXEC calls. These conventions are summarized below.

* Parameters that are underlined, such as

      CALL EXEC(IP1,IP2,IP3)
               ---   ---

  have values returned by the system, e.g., the value is not supplied by the user.

* Parameters that are double underlined, such as

      CALL EXEC(IP1,IP2,IP3)
                ===

  have values that are returned by the system in some cases and user-supplied in other cases. The comments associated with the call description should be consulted for details concerning their use.

* Parameters enclosed in square brackets, such as

      CALL EXEC(IP1,IP2[,IP3])

  are optional.

* Parameters enclosed in angle brackets, such as

      CALL EXEC(IP1,IP2<,IP3>)

  are optional in some cases and required in others. The comments associated with the call description should be consulted for details concerning their use.

* Parameters with no qualifiers, i.e., square brackets, angle brackets, or underlines, are required and their value is supplied by the user.

* If an optional parameter is wanted, but the preceding optional parameters are not, dummy variables must be used as place holders for the missing parameters.

* All EXEC call descriptions in this section use the FORTRAN-IV subroutine call format. If desired, the description of EXEC call general formats included at the beginning of this section can be consulted to convert the calls to the FORTRAN-IV function call or Assembly Language format.

Standard I/O EXEC Calls

Standard I/O EXEC calls are used to transfer data to or from external I/O devices in addition to performing various I/O control operations. For output or control requests to unbuffered devices, the calling program is I/O suspended (state 2) until the operation completes. All input requests cause the calling program to be placed in state 2 until completion. Note that a device is specified as buffered or non-buffered at generation time (see RTE-IVB On-Line Generator Manual).

The EXEC calls included in this section are listed below in the order of their presentation:

* EXEC 1 or 2 - Read or Write

* EXEC 3 - I/O Control

Note that the I/O operations performed by EXEC 1, 2 and 3 calls can also be accomplished on a "no-wait" basis by using the class I/O calls described in Chapter 2.

Read or Write Call — EXEC 1 or 2

Transfers information to (write) or from (read) a disc or non-disc I/O device.

```
+------------------------------------------------------------------+
| CALL EXEC(ICODE,ICNWD,IBUFF,ILEN[,IOP1[],IOP2])                  |
|           =====                                                  |
+------------------------------------------------------------------+
| ICODE - Request code.  1=read, 2=write.                          |
|                                                                  |
| ICNWD - Control word. Specifies the LU of the device involved    |
|         in the data transfer, driver dependent informa-          |
|         tion, and optional parameter (IOP1 & IOP2) considera-    |
|         tions.                                                   |
|                                                                  |
| IBUFF - Data buffer. For read operations (ICODE=1), the array    |
|         into which the system returns data;  for write opera-    |
|         tions (ICODE = 2),  the array into which the program     |
|         places data to be written.                              |
|                                                                  |
| ILEN - Data length. Positive number of words or negative num-    |
|        ber of characters to be read or written.                  |
|                                                                  |
| IOP1 - Track number for disc transfers, optional parameter       |
|        for non-disc transfers.                                   |
|                                                                  |
| IOP2 - Sector number for disc transfers, optional parameter      |
|        for non-disc transfers.                                   |
+------------------------------------------------------------------+
```

COMMENTS:

Control word (ICNWD)---The format of the control word is as follows:

```
     (UB)      (Z)      (X) (A) (K) (V) (M)
     ----------------------------------------------------------------
     XX|14| XX| 12 |XX| 10| 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |0|
     ----------------------------------------------------------------
                       |<--Function Code-->|<----Logical Unit--->|
```

The function of the control word fields are summarized below:

* Logical Unit (bit 0-5). Logical unit number (LU) of the device
  to/from which data is to be transferred. Can be the LU of a disc
  or non-disc device. If the LU is specified as zero, the call is
  executed, but no data is transferred. If LU specifies a disc
  device, IOP1 and IOP2 are required (see Optional Parameters,
  below).

* Function Code (bit 6-10). Specifies control information for the
  driver module associated with the I/O device involved in the data
  transfer. This information is driver dependent and the user should
  reference the appropriate driver reference manual for more
  information. The function code field can be defaulted to zero if
  no special driver action is desired. Some example values and
  corresponding meaning:

  DVA05 Bit 8=1 EXEC 1 (Read from 264x terminal - Echo ON).
  DVR12 Bit 7=0 EXEC 2 (Write to 2607A Line Printer) - First
                         character of output is used for line control.

* Z-bit (bit 12). If the Z-bit is set ("1"), a control buffer
  containing additional information is passed to the driver. The
  control buffer is defined by IOP1 and IOP2 (see Optional
  parameters, below). Note that this technique is only used for
  transfers to non-disc devices.

  The Z-bit is clear ("0"), if a control buffer is not being passed
  to the driver or the transfer is to a disc device. Note that
  this is the default condition.

* UB-bit (bit 14). If the UB-bit is set ("1"), the I/O operation is
  forced to be unbuffered, even if the I/O device is buffered. This
  is useful if I/O status is desired after a transfer or for very
  large I/O requests (equal to or more than 1K words).

* Bit 11 and 13-15. These bits are used by the system and should be
  set to zero by the user.

DATA LENGTH (ILEN)---This variable defines the length of the data record to be transferred. A positive number is used to specify the length of the data record in words. A negative number is used to specify the length of the data record in characters (bytes). If the data record contains REAL data, this must be allowed for in the ILEN parameter. For example, a data record containing 10 REAL values would require ILEN to be +20 or -40 since each REAL value requires two words. If the data record contains double-precision data, three words or four words are required for each value, depending on the option taken for the FORTRAN Compiler at generation time.

OPTIONAL PARAMETERS (IOP1 and IOP2)---These two parameters are required or optional depending on the state of the Z-bit (see above). If the Z-bit is set ("1"), IOP1 specifies the address of a control buffer and IOP2 specifies the length of the buffer (positive number of words or negative number of characters). The contents of the control buffer is driver dependent and the appropriate driver reference manual should be consulted for more information.

If the Z-bit is clear ("0") and the LU specifies a disc device, IOP1 contains the track number and IOP2 contains the sector number necessary for the transfer.

If the Z-bit is clear and the LU specifies a non-disc device, IOP1 and IOP2 are not used.

A- AND B-REGISTER RETURNS---End-of-operation information for successful read and unbuffered write operations is transmitted to the calling program via the A- and B-registers. These register returns are summarized below:

* A-register <-- word 5 (status word) of devices' EQT entry (see Appendix B, "EQT FORMAT").

* B-register <-- positive number indicating number of words transferred (if ILEN in call was positive), or positive number of characters (bytes) transferred (if ILEN in call was negative).

* A- and B-register returns are meaningless for write requests to buffered devices.

For unsuccessful operations, error information is returned in the A- and B-registers. "EXEC CALL ERROR RETURNS", included earlier in this section, describes the considerations associated with error returns.

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

A. The data buffer is not in the calling programs' partition, i.e., it is in system COMMON.

B. The input or output buffer is completely contained in the Temporary Data Block (TDB) of a reentrant routine, and enough SAM was allocated to hold the TDB (refer to Chapter 6, "Reentrant Subroutines section").

C. The device is buffered, the request is for output, and enough SAM was allocated for buffering the data record to be transferred.

Only the first buffer of a double-buffer request (Z-bit set) is checked to determine swappability. It is the user's responsiblity to put the second buffer in an area that implies swappability (COMMON, etc.) if condition "a" or "b" are true. The system handles condition "c".

Only the first buffer of a double-buffer request (Z-bit set) is checked to determine swappability. It is the user's responsiblity to put the second buffer in an area that implies swappability (COMMON, etc.) if condition "a" or "b" are true. The system handles condition "c".

EXAMPLE---Write a message to the system console (LU1) which prompts the operator to input his name. Read the response entered by the operator. Check for EXEC errors.

```
        PROGRAM OPNAM
        DIMENSION IBUF1(5),IBUF2(10)
        DATA IBUF1/2HEN,2HTE,2HR ,2HNA,2HME/
        .
        .
        ICODE=2+100000B
        ICNWD=1
        ILEN=5
        CALL EXEC(ICODE,ICNWD,IBUF1,ILEN)
        GO TO 100
    10  CONTINUE
        .
        .
        CALL EXEC(1+100000B,1,IBUF2,-20)
        GO TO 100
    20  CONTINUE
        .
        .
   100  CALL ABREG(IA,IB)
        error processing
        .
        .
        END
```

I/O Control Call — EXEC 3

Initiates various I/O control operation (backspace, write end-of-file, rewind, etc.) on specified LU. Function is dependent on device to be controlled.

```
+-------------------------------------------------------------+
|                                                             |
|   CALL EXEC(ICODE,ICNWD[,IOP1])                             |
|                                                             |
+-------------------------------------------------------------+
|   ICODE - Request code.  3 = control.                       |
|                                                             |
|   ICNWD - Control word. Specifies the  LU  and  the control |
|           action to be initiated on that LU.                |
|                                                             |
|   IOP1 -  Optional  parameter.  Optional or required depending |
|           on ICNWD.                                         |
+-------------------------------------------------------------+
```

COMMENTS:

CONTROL WORD (ICNWD)---The format of the control word is as follows:
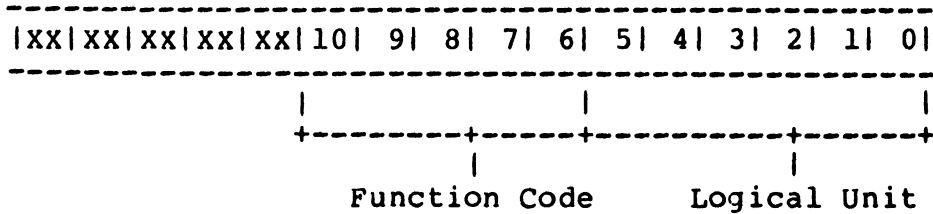
```
    ------------------------------------------------------
    |XX|XX|XX|XX|XX|10| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0|
    ------------------------------------------------------
                     |              |              |
                     +-------+------+----------+------+
                             |                  |
                        Function Code      Logical Unit
```

The function of the control word fields are summarized below:

*   Logical Unit (bit 0-5). Logical unit number (LU) of device that control action is to be initiated on.

*   Function Code (bit 6-10). Specifies the type of control action to be initiated. Shown below is a partial summary of the function codes associated with various devices and drivers. For more details, the user should reference the driver reference manual associated with the device to be controlled.

FUNCTION CODE (OCTAL)	ACTION
00	Clear device (U)
01	Write end-of-file (MT,CTU)
02	Backspace one record (MT,CTU)
03	Forward space one record (MT,CTU)
04	Rewind (MT,CTU)
05	Rewind standby (MT), Rewind (CTU)
06	Dynamic Status (U)
11	List output line spacing, requires use of IOP1. (LP)
12	Write inter-record gap (MT)
13	Forward space one file (MT, CTU)
14	Backspace one file (MT, CTU)
15	Conditional form feed (LP)
20	Enable Terminal (CRT)
21	Disable Terminal (CRT)
22	Set time out, requires IOP1 (CRT)
26	Write end-of-data (CTU)
27	Locate file, requires IOP1 (CTU)

```
U   = Universal
MT  = Magnetic Tape Drive
CTU = Terminal Cartridge Tape Unit
LP  = Line Printer
CRT = 264X Terminal
```

For more information consult the appropriate Driver Manual.

OPTIONAL PARAMETER (IOP1)---IOP1 is optional or required depending on the function code used. For example, if the function code is 11 (octal); indicating line spacing for the line printer, IOP1 would contain the number of lines to be spaced. If the function code is 27 (octal), IOP1 would contain the number of a file to be located on the CTU. For more details on the optional parameter, the user should refer to the driver reference manual associated with the device being controlled.

A- AND B-REGISTER RETURNS---For successful I/O control calls, the values returned into the A- and B-registers are as follows:

* A-register <-- If call was to an unbuffered device, word 5 (status word) of devices' EQT entry is returned (see Appendix B, "EQT FORMAT"). If call was to a buffered device, return is meaningless.

* B-register <-- meaningless.

Register returns for unsuccessful call contain error information (see "EXEC CALL ERROR RETURNS").

EXAMPLE---Cause line printer (LU6) to space 5 lines.

```
        .
        .
        .
ICODE=3
ICNWD=6+1100B
IOP1=5
CALL EXEC(ICODE,ICNWD,IOP1)
        .
        .
        .
```

Disc Track Management EXEC calls

The Disc Track Management EXEC calls allow the user to allocate disc tracks for data storage and release disc tracks when they are no longer required. After allocation, the disc tracks can be accessed using EXEC read and write calls (see "STANDARD EXEC I/O CALLS"). The tracks are allocated on the system or auxiliary cartridges (LU2 or 3).

Disc tracks can be allocated as either local or global tracks. Locally allocated tracks can only be written to, or released by, the program that they are allocated to; they can be read by any program. Globally allocated tracks can be read from, written to, or released by any program.

When tracks are allocated, the system makes an entry in the Track Assignment Table (TAT). For a local allocation, the TAT entries corresponding to the allocated tracks will contain the ID segment address of the "owning" program. For global allocations, the TAT Entries will contain 077777 (octal). TAT entries for available tracks (released or never assigned) will contain zero. The TAT can be examined using the LGTAT utility described in the RTE-IVB Terminal Users' Reference Manual.

When tracks are allocated, they are supplied in units of complete tracks, i.e., one track is the minimum that can be allocated and no fraction of a track can be allocated. Tracks are also allocated on a contiguous basis. If six tracks are requested, six adjacent tracks will be allocated and they will be on the same cartridge.

Automatic track switching is not provided. If the user writes or reads a record that crosses track boundaries, it is the user's responsibility to divide his record access into two operations. One operation to access the data on one track and another operation to access the data on the following track.

The EXEC calls included in this section are listed below in the order of their presentation:

*   EXEC 4 or 15 – Disc Track Allocation

*   EXEC 5 or 16 – Disc Track Release

DISC TRACK ALLOCATION EXEC CALLS - EXEC 4 OR 15.

Allocate contiguous disc tracks for use by a single program (locally) or by multiple programs (globally).

```
+--------------------------------------------------------------+
|                                                              |
|   CALL EXEC(ICODE,ITRAK,ISTRK,IDISC,ISECT)                   |
|                    ----- ----- -----                         |
+--------------------------------------------------------------+
|                                                              |
|   ICODE - Request  code.  4 = local  allocation ,  15 = global |
|           allocation.                                        |
|                                                              |
|   ITRAK - Number  of contiguous tracks requested.   Program is |
|           suspended  if tracks are not available;  rescheduled |
|           when  tracks become  available.   If bit  15  is set |
|           (ITRAK=ITRAK+100000B),   the program is not suspended |
|           if tracks are not available.                       |
|                                                              |
|   ISTRK - Starting  track number; returned by system.   Set to |
|           -1 if tracks not available.                        |
|                                                              |
|   IDISC - LU  number of disc cartridge where tracks were allo- |
|           cated  (2 or 3); returned by system.               |
|                                                              |
|   ISECT - Number  of 64 word sectors  per track;   returned by |
|           system.                                            |
|                                                              |
+--------------------------------------------------------------+
```

COMMENTS:

A- AND B-REGISTER RETURNS---For successful completion of the call, the contents of the A- and B-registers are meaningless. For unsuccessful calls, error information is returned (see, "EXEC CALL ERROR RETURNS").

EXAMPLE---Allocate two contiguous disc tracks globally and write 128 words into the sixth and seventh sector of the second track. If tracks are not available, do not suspend program; forget about write and continue processing. Check for errors after each call. Place location information in COMMON for use by other programs.

```
      PROGRAM DSCWT
      DIMENSION IBUFF(128)
      COMMON IDISC,ISTRK
         .
         .
      ICODE=15+100000B
      ITRAK=2+100000B
      CALL EXEC(ICODE,ITRAK,ISTRK,IDISC,ISECT)
      GO TO 100
 10   IF(ISTRK.EQ.-1) GO TO 200
         .
         .
      place data in IBUFF
         .
      ICODE=2+100000B
      ICNWD=IDISC
      ILEN=128
      IOP1=ISTRK+1
      IOP2=6
      CALL EXEC(ICODE,ICNWD,IBUFF,ILEN,IOP1,IOP2)
      GO TO 100
 20   CONTINUE
         .
         .
100   CALL ABREG(IA,IB)
      error processing
         .
         .
200   CONTINUE
         .
         .
      END
```

DISC TRACK RELEASE EXEC CALL - REQUEST CODE 5 OR 16.

Release contiguous disc tracks that were previously allocated either locally or globally.

```
+---------------------------------------------------------------------+
|                                                                     |
|   CALL EXEC(ICODE,ITRAK[,ISTRK][,IDISC])                            |
|                                                                     |
+---------------------------------------------------------------------+
|                                                                     |
|   ICODE - Request code. 5 = release local tracks, 16 = release     |
|           global tracks.                                            |
|                                                                     |
|   ITRAK - Number  of contiguous disc tracks to be released. If     |
|           ICODE = 5,  ITRAK  can be set  to  -1 and all tracks      |
|           allocated  locally  to the  calling program  will be     |
|           released.                                                 |
|                                                                     |
|   ISTRK - Starting  track number of tracks to be released. Not     |
|           used if ICODE = 5 and ITRAK = -1.                         |
|                                                                     |
|   IDISC - Disc  LU (2 or 3) where tracks to be released reside.    |
|           Not used if ICODE = 5 and ITRAK = -1.                     |
|                                                                     |
+---------------------------------------------------------------------+
```

COMMENTS:

A- AND B-REGISTER RETURNS---For successful calls with ICODE = 5, A- and B-register returns are meaningless. If ICODE = 16, the B-register return is meaningless and the A-register return can be interpreted as follows:

* A = 0.   Tracks have been released.

* A = -1. No tracks have been released. This indicates that at least one of the tracks was in use, i.e., a program was queued to read or write on one of the tracks at the time of the release request.

* A = -2. No tracks have been released. This indicates that at least one of the tracks to be released was not assigned globally.

Error information is returned in the A- and B-registers for unsuccessful calls (see, "EXEC CALL ERROR RETURNS").

If a program aborts with tracks allocated locally to itself, the tracks are released automatically by the system.

The local track release function (ICODE=5) can also be accomplished interactively using the RT command described in the RTE-IVB Terminal Users' Reference Manual.

EXAMPLE---Read the data placed on disc in the previous example (see Chapter 2). Release the tracks and verify that they were released. Check for EXEC errors. Obtain the track number and disc LU from COMMON.

```
      PROGRAM DSCRD
      DIMENSION IBUFF(128)
      COMMON ISTRK,IDISC
         .
         .
      ICODE=1+100000B
      ICNWD=IDISC
      ILEN=128
      IOP1=ISTRK+1
      IOP2=6
      CALL EXEC(ICODE,ICNWD,IBUFF,ILEN,IOP1,IOP2)
      GO TO 100
   10 CONTINUE
         .
         .
      ICODE=16+100000B
      ITRAK=2
      CALL EXEC(ICODE,ITRAK,ISTRK,IDISC)
      GO TO 100
   20 CALL ABREG(IA,IB)
      IF(IA.EQ.0) GO TO 200
      tracks-not-released processing
         .
         .
  100 CALL ABREG(IA,IB)
      error processing
         .
         .
         .
  200 CONTINUE
         .
         .
         .
      END
```

Program Management EXEC calls

The program management EXEC calls allow user written programs to control their own execution or the execution of other programs. This includes scheduling, suspending, or terminating programs, as well as, controlling program swapping and implementing communication between programs. A list of the program management EXEC calls are summarized below in the order of their presentation.

*   EXEC 9,10,23,24 - Program Scheduling

*   EXEC 14 - String Passage

*   EXEC 12 - Timed Program Execution

*   EXEC 22 - Program Swapping Control

*   EXEC 8 - Program Segment Load

*   EXEC 7 - Program Suspension

*   EXEC 6 - Program Completion

Program Scheduling — EXEC 9, 10, 23, and 24

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

```
+----------------------------------------------------------------+
|                                                                |
|    CALL EXEC(ICODE,INAME[,IOP1][,IOP2][,IOP3][,IOP4][,IOP5]    |
|      [,IBUFF][,ILEN])                                          |
|                                                                |
+----------------------------------------------------------------+
|                                                                |
|    ICODE - Request Code.   9=immediate schedule with wait.     |
|                           10=immediate schedule without wait.  |
|                           23=queue schedule with wait.         |
|                           24=queue schedule without wait.      |
|                                                                |
|    INAM - 3-word  array containing ASCII name of program to be |
|           scheduled.                                           |
|                                                                |
|    IOP1  thru  IOP5 - Optional  parameters;  passed to program |
|           specified in INAME.                                  |
|                                                                |
|    IBUFF- Optional  buffer;  passed  to program  specified in  |
|           INAME.                                               |
|                                                                |
|    ILEN - Data length;  positive  number of words  or negative |
|           number of characters  (bytes) to be passed from IBUF |
|           to the program specified in INAME.                   |
|                                                                |
+----------------------------------------------------------------+
```

COMMENTS:

FATHER AND SON---When one program schedules another, the program that did the scheduling is known as the "Father" and the program that was scheduled is known as the "Son". The system places a pointer in word 20 of the son's ID segment that points back to the father's ID segment (see Appendix B, "ID SEGMENT FORMAT"). If a program is not scheduled by another program, e.g., has no father, the pointer is set to zero. This would be the case if a program was scheduled by the operator (RU command) or an interrupt, or scheduled from the time list. If the father/son relationship exists and the son completes or is terminated, the pointer is cleared.

Note that the father will receive an error return (SC11) if it tries to schedule a son that has been placed in the time list from another session.

ICODE=9 OR 10; IMMEDIATE SCHEDULE (WITH OR WITHOUT WAIT)---When a father schedules a son without wait (ICODE=10), the son is scheduled for execution according to its priority. The father continues executing at its priority without waiting for the son to complete.

When a father schedules a son with wait (ICODE=9), the father is placed in the general wait list (state 3). While in this state, the father is swappable. The son executes according to its priority. when the son completes or is terminated, the father resumes execution at the point immediately following the scheduling EXEC call.

In order for the father program to immediately schedule a son (ICODE=9 or 10), the son must be dormant (state 0). If the son is not dormant, it is not scheduled. The father can determine whether the son was scheduled or not by examining the contents of the A-register after the execution of the scheduling call. The A-register contents will be as follows:

* If the son was successfully scheduled (dormant before call) the A-register will contain zero.

  Note that if the son was suspended by a previous EXEC 6 call (see, "PROGRAM COMPLETION - EXEC 6"), only the four least significant bits of the A-register will be zero.

* If the son could not be scheduled (was not dormant) the A-register will contain the status of the son.

ICODE = 23 OR 24; QUEUE SCHEDULE (WITH OR WITHOUT WAIT)---The EXEC 23 and 24 calls are similiar to the EXEC 9 and 10 calls, respectively, except that the system places the father in a queue if the son is not dormant. While in the queue, the father is suspended until the son can be scheduled. When the potential son goes dormant, and can therefore be scheduled, the system reissues the father's request and execution proceeds like the immediate schedule calls described above. That is, for an EXEC 23, when the son becomes available, the father is removed from the queue (rescheduled) and execution continues as an EXEC 9. An EXEC 24 would continue as an EXEC 10.

Note that the father does not have access to the son's status via the A-register as with the immediate schedule calls.

SON TERMINATION CONSIDERATIONS---When a father performs an immediate schedule with wait (ICODE = 9) or a queue schedule with wait (ICODE = 23), it is suspended until the son completes or is terminated. The father can determine if the son terminated normally or not by examining the systems copy of optional parameter 1 (see "OPTIONAL PARAMETER PASSAGE" below). The father does this by calling the RMPAR routine which is described in the DOS/RTE Relocatable Library Reference Manual. The returned contents of IOP1 are summarized below:

* IOP1 = 100000B. Indicates that the son terminated abnormally for one of the following reasons:

    1.   System aborted son

    2.   Operator issued OF command (see RTE-IVB Terminal User's Reference Manual).

    3.   Son performed an EXEC (6,0,2) or EXEC (6,0,3) self-termination call (see, "PROGRAM COMPLETION - EXEC 6").

* IOP1 = Original value or value passed back by son indicates that the son terminated normally.

OPTIONAL PARAMETER PASSAGE---The optional parameters (IOP1 - IOP5) can be used to pass values from the father to the son. The father places the values to be passed into the optional parameters and then makes the scheduling call. When the son begins executing, it can obtain these values by calling the library subroutine RMPAR. Note that the RMPAR call should be the first executable statement in the son program.

If the father performed a schedule with wait call, the son can pass back up to five parameters to the father. To do this, the son would make a call to the library routine PRTN (see Chapter 6, "SYSTEM LIBRARY UTILITIES") to place the values to be returned into a five-word array. The son should then terminate itself using an EXEC 6 call to prevent B-register modification.

The son's termination causes the father to be rescheduled. The father can then obtain the returned values by calling RMPAR. Note that if the son terminates abnormally, the first returned parameter will be overwritten with 100000B (see "SON TERMINATION CONSIDERATIONS", above).

OPTIONAL BUFFER PASSAGE---In addition to five optional parameters, the father can pass an optional buffer to the son. The father places the data to be passed into IBUFF and specifies the length of the data in ILEN. When the father makes the scheduling call, the buffer is moved into System Available Memory (SAM). The son can obtain the buffer by making an EXEC 14 call (see "STRING PASSAGE - EXEC 14"). If there is not enough SAM currently available to hold the buffer, the father is memory suspended (state 4). If there will never be enough SAM available, the father is aborted or, if the "no-abort" bit is set, the error return is taken (see, "EXEC CALL ERROR RETURNS"). Note that the length of the buffer is limited only by the amount of usable SAM available.

If the father performed a schedule with wait call, the son can pass a buffer back to the father by using an EXEC 14 call and the father can recover the buffer with an EXEC 14 call. If a program is scheduled by an operator command, e.g., the RU, ON, or GO command described in the RTE-IVB Terminal User's Reference Manual, an EXEC 14 call can be used to obtain the command string or a call to the library routine GETST can be used to obtain the parameter string. GETST is described in Chapter 6, "SYSTEM LIBRARY ROUTINES". The command string and the parameter string are described with the discussion of the EXEC 14 call later in this section.

A- AND B-REGISTER RETURNS---For successful EXEC scheduling calls, the contents of the A- and B-registers are used to determine status and parameter location information. These register returns have been discussed in the above text with the calls that they pertain to. For convenience they are briefly summarized below:

*   A-register (for immediate schedule). 0 if son was scheduled, son's status if son could not be scheduled.

*   A-register (for queue schedule). Meaningless.

*   B-register (when son is scheduled with wait). Address of parameters, if optional parameters are being used. If optional parameters are not used, B-register is cleared.

For unsuccessful calls, the A- and B-register will contain error information (see "EXEC CALL ERROR RETURNS").

EXAMPLE---Father schedules a son (queue schedule with wait, ICODE=23). Father passes the son five parameters. Son returns five parameters to father. Father checks for EXEC errors.

Table 2-1A. Summary of EXEC Calls 9, 10, 23, 24

| | IMMEDIATE | | QUEUE | |
	SON DORMANT	SON ACTIVE	SON DORMANT	SON ACTIVE
Wait 9, 23	9 Son Scheduled	Son not scheduled	23 Son Scheduled	Father goes into state 3 until son can be scheduled.
	Father goes into state 3 until son completes	Son's state in A-Reg	Father goes into state 3 until son completes	Son scheduled Father stays in state 3 until son completes
No Wait 10, 24	10 Son scheduled	Son not scheduled	24 Son scheduled	Father goes into state 3 until son can be scheduled
	Father stays in state 1	Son's state in A-Reg	Father stays in state 1	Son scheduled Father goes into state 1

```
      PROGRAM FATHR
      DIMENSION IPAR(5),INAM(3)
      DATA INAM/2HSO,2HNN,2HY/
         .
         .
      place values in optional parameters (IOP1-IOP5)
         .
      ICODE=23+100000B
      CALL EXEC(ICODE,INAM,IOP1,IOP2,IOP3,IOP4,IOP5)
      GO TO 100
 10   CALL RMPAR(IPAR)
      IF(IPAR(1).EQ.100000B) GO TO 200
      process five parameters returned by son.
         .
         .
100   CALL ABREG(IA,IB)
      error processing
         .
         .
200   CONTINUE
         .
         .
      END

      PROGRAM SONNY
      DIMENSION IP(5)
      CALL RMPAR(IP)
      process five values passed by father
         .
         .
      place values to be returned to father in IP
         .
      CALL PRTN(IP)
      CALL EXEC(6)
      END
```

String Passage — EXEC 14

Allows a program to retrieve a buffer from the program that scheduled it (father) or retrieve the command string if it was scheduled by an operator command. Allows a son to pass a buffer back to its father.

```
+------------------------------------------------------------------+
|                                                                  |
|   CALL EXEC(ICODE,IRWCD,IBUFF,ILEN)                              |
|              =====                                               |
+------------------------------------------------------------------+
|                                                                  |
|   ICODE  - Request code.   14 = string passage.                  |
|                                                                  |
|   IRWCD  - Retrieve/write code. 1 = retrieve  buffer  or command |
|            string.  2 = write  buffer to father.                 |
|                                                                  |
|   IBUFF  - Buffer into  which  retrieved data or command  string |
|            is  placed  (IRWCD = 1)  or a  buffer into  which son  |
|            places  data  to be returned to father (IRWCD = 2).    |
|                                                                  |
|   ILEN   - Positive  number  of  words  or  negative  number  of |
|            characters  (bytes)  to be transferred.               |
|                                                                  |
+------------------------------------------------------------------+
```

COMMENTS:

COMMAND STRING---When a program is scheduled using an operator command (see RU, ON, or GO commands in the RTE-IVB Terminal Users´ Reference Manual) a command string is placed in a buffer in System Available Memory (SAM). The command string is a copy of the command used to schedule the program. For example, if the RU command with optional parameters was used to schedule a program, the command string would appear as:

    RU,PROG,IP1,IP2,IP3,IP4,IP5,STRING

where:

    PROG = name of program to be scheduled
    IP1 - IP5 = one-word parameters
    STRING = ASCII string

If a program that was scheduled by an operator command performs an EXEC 14 call (IRWCD=1), the command string contained in the SAM buffer is retrieved and placed in IBUFF. It is the program´s responsibility to parse the string into separate parameters.

Note that the program could call the library utility program GETST to retrieve the parameters following the second comma in the command string (known as the parameter string). GETST is described in Chapter 6, "SYSTEM LIBRARY UTILITIES".

FATHER/SON BUFFER PASSAGE---If a father includes the optional buffer as a parameter in the EXEC call used to schedule a son (see "PROGRAM SCHEDULING - EXEC 9, 10, 23 or 24"), the contents of the buffer are placed in SAM when the call is executed. If the son performs an EXEC 14 (IRWCD=1), the buffer in SAM is retrieved, the contents are placed in IBUFF, and the SAM buffer is deallocated. The son program should perform the EXEC 14 prior to any other executable statement which may cause the SAM buffer to be overwritten or deallocated (e.g., on EXEC 23, scheduling D.RTR may overwrite the SAM buffer that was intended to be passed from the father to the son).

If the son was scheduled with wait (EXEC 9 or 23), the son can pass a buffer back to the father. The son would place the data to be returned into IBUFF, specify the length in ILEN, and perform an EXEC 14 call with IRWCD=2. The son should then terminate itself (see "PROGRAM TERMINATION - EXEC 6"), allowing the father to be rescheduled. The father could obtain the returned buffer from SAM by performing an EXEC 14 call with IRWCD=1.

Note that the EXEC 14 write call (IRWCD=2) deallocates any block of SAM associated with the father, which the father created with an EXEC 14 call, and allocates a new block for the father into which the returned buffer will be placed.

ILEN---Only ILEN words or characters are transmitted. If the command string on a retrieve operation, or the buffer on a retrieve or write operation, is longer than ILEN, the excess data will be lost. If an odd number of characters are requested for a retrieve operation, the least significant byte of the last word is undefined.

A- AND B-REGISTER RETURNS---Upon successful return from a retrieve operation (IRWCD=1), the A- and B-registers can be interpreted as follows:

* A-register = 0 if operation was successful

* A-register = 1 if no string could be found

* B-register = positive number of words or characters transferred.

For unsuccessful calls, error information is returned (see, "EXEC CALL ERROR RETURNS").

EXAMPLE---Father schedules a son (immediate schedule without wait, ICODE=9) and passes it two optional parameters and an optional buffer. The son retrieves the buffer and prints its contents. The optional parameters contain the list and log LU to be used by the son.

```
      PROGRAM SENDR
      DIMENSION IBUF1(10),INAME(3)
      DATA INAME/2HGE,2HTE,2HR /
         .
         .
      ICODE=9+100000B
      IOP1=1
      IOP2=6
      ILEN=10
         .
      place data in IBUF1
         .
      CALL EXEC(ICODE,INAME,IOP1,IOP2,,,,IBUF1,ILEN)
      GO TO 100
   10 CONTINUE
         .
         .
  100 error processing
         .
         .
      END

      PROGRAM GETER
      DIMENSION IBUF2(10),IPAR(5)
      CALL RMPAR(IPAR)
      LOGLU=IPAR(1)
      LSTLU=IPAR(2)
      ICODE=14+100000B
      CALL EXEC(ICODE,1,IBUF2,10)
      GO TO 100
   10 CALL ABREG(IA,IB)
      IF(IA.NE.0) GO TO 200
      CALL EXEC(2+100000B,LSTLU,IBUF2,IB)
      GO TO 100
   20 CONTINUE
         .
         .
  100 CALL ABREG(IA,IB)
      WRITE(LOGLU,101)IA,IB
  101 FORMAT("A-REG=",I6,5X,"B-REG=",I6)
      CONTINUE
         .
         .
  200 WRITE(LUGLU,201)
  201 FORMAT("BUFFER NOT FOUND")
      CONTINUE
         .
         .
      END
```

Program Time Scheduling — EXEC 12

Places calling program or another program into the time scheduling list. The EXEC 12 call has two modes of operation:

1. Absolute Time Mode - a program can be scheduled to run once at an absolute time or to run repeatedly at specified intervals with the first run to be at an absolute time.

2. Initial Offset Mode - a program can be scheduled to run once at a time offset from the current time or to run repeatedly at specified intervals with the first run to be at a time offset from the current time.

```
+------------------------------------------------------------------+
|                                                                  |
|    Initial offset:                                               |
|        CALL EXEC(ICODE,INAME,IRESL,IMULT,IOFST)                  |
|                                                                  |
|    Absolute time:                                                |
|        CALL EXEC(ICODE,INAME,IRESL,IMULT,IHRS,IMIN,ISEC,IMSEC)   |
|                                                                  |
+------------------------------------------------------------------+
|                                                                  |
|    ICODE - Request code.  12 = time scheduling.                  |
|                                                                  |
|    INAME - Program name.  3-word array containing ASCII name of  |
|            program  to be scheduled; set to 0 if calling program |
|            is to be scheduled.                                   |
|                                                                  |
|    IRESL - Resolution  code.  Specifies  units  to be used with  |
|            IMULT and IOFST.                                      |
|                                                                  |
|            1 = 10's of milliseconds                              |
|            2 = seconds                                           |
|            3 = minutes                                           |
|            4 = hours                                             |
|                                                                  |
|    IMULT - Execution  multiple.  Integer (0-4095) that specifies |
|            the  time interval between runs for programs that run |
|            repeatedly.  Used  in conjunction with IRESL.  0 in-  |
|            dicates that program is to run once.                  |
|                                                                  |
|    IOFST - Initial offset.  Negative integer that specifies off- |
|            set from  current time that a program will first run. |
|            Used in conjunction with IRESL.                       |
|                                                                  |
|    IHRS,IMIN,ISEC,IMSEC - Specifies the hours, minutes,  seconds |
|            and  milliseconds, respectively,  when a program will |
|            first run.                                            |
|                                                                  |
+------------------------------------------------------------------+
```

COMMENTS:

INITIAL OFFSET---The initial offset version of the EXEC 12 call can be used to schedule programs as follows:

*   Run once - IMULT is set to zero to specify that a program is to run once. IRESL and IOFST are set to indicate the offset from the current time that the program is to be run. For example, if IMULT=0, IRESL=3, and IOFST=-45, the program specified in INAME will run once, 45 minutes from the current time. Note that the program to be scheduled must be dormant at that time or the call is ignored.

*   Run Repeatedly - IMULT is set to specify the interval between run times; IRESL and IOFST indicate the offset from the current time that the program will first run. For example, if IMULT=30, IRESL=3, and IOFST=-60, the program specified in INAME will run every 30 minutes, with the first run being 60 minutes from the current time. The program to be scheduled must be dormant at the indicated scheduling times or the call is ignored for that time period. The future scheduling times are not affected.

ABSOLUTE TIME---The absolute start time version of the EXEC 12 call can be used to schedule programs as follows:

*   Run once - IMULT is set to zero to specify that the program is to run once. IRESL is unused. The time for execution is specified in IHRS, IMIN, ISEC, and IMSEC. For example, if IHRS=18, IMIN=45, ISEC=30, and IMSEC=0, the program specified in INAME would be scheduled to execute at 18:45:30:0. If the program is not dormant at the specified time, the scheduling call is ignored. If a time is specified that has already passed, the program will be scheduled in the next 24-hour period of the clock.

*   Run repeatedly - IMULT is set to indicate the interval between run times. IRESL specifies the units to be used with IMULT. IHRS, IMIN, ISEC, and IMSEC specify the first time that the program will run. For example, if IRESL=3, IMULT=30, IHRS=18, IMIN=30, ISEC=0, and IMSEC=0, the program specified in INAME would be scheduled to execute every 30 minutes with the first run to be at 18:30:0:0. The program must be dormant when a scheduling time arrives, or the call is ignored for that time period. Future scheduling times are not affected. If the specified first-run time has already passed, the programs scheduling cycle will be started in the next 24-hour period of the clock.

LIMITATIONS---For IRESL=3, IMULT and IOFST must be less than 1440. For IRESL=4, IMULT and IOFST must be less than 24.

SELF TIME SCHEDULING---If INAME is set to zero, the calling program time schedules itself.

* Offset - After the calling program executes the EXEC 12 call, the program is set dormant and rescheduled at the specified offset from the current time. The point of suspension (after the EXEC 12 call) is saved, and the execution of the program continues from that point when it is rescheduled.

  The calling program can be scheduled to run repeatedly with the EXEC 12 call. IMULT is set to specify the interval between run times. The calling program is set dormant after the EXEC 12 call and is rescheduled at the specified offset from the current time. The program continues executing at the point of suspension until completion. The program schedules itself for future execution starting at the beginning of the program. The program can be designed to avoid executing the EXEC 12 call again.

* Absolute - The program is placed into the timelist to be rescheduled at the specified starting time. The calling program continues execution, does not go dormant, but schedules itself for future execution starting at the beginning of the program.

A- AND B-REGISTER RETURNS---Upon successful completion of the call, the A-register contents will be meaningless and the B-register contents will be unchanged. For unsuccessful calls, the A- and B-registers will contain error information (see, "EXEC CALL ERROR RETURNS").

Note that the EXEC 12 call performs a function similiar to the IT operator command described in the RTE-IVB Terminal Users´ Reference Manual.

Note also that the calling program will receive an error return (SC11) if it attempts to time schedule a program that is currently scheduled (or in the time list) for another session.

EXAMPLE---Schedule calling program to execute every 30 minutes with first run to be 60 minutes from current time.

```
        PROGRAM SCHED
        :
        IRESL=3
        IMULT=0
        IOFST=-60
        :
    10  CONTINUE
        CALL EXEC(12,0,IRESL,IMULT,IOFST)
        CONTINUE--+
        :         |
        IOFST=-30 | This section of code executes every 30 minutes.
        GO TO 10  |
        :       --+
        END
```

Program Swapping Control — EXEC 22

Allows a program to lock itself into memory, i.e., program performing call will not be swapped out if a higher priority program needs its partition. Allows program to release the lock.

```
+--------------------------------------------------------------------+
|                                                                    |
|   CALL EXEC(ICODE,ILOCK)                                           |
|                                                                    |
+--------------------------------------------------------------------+
|                                                                    |
|   ICODE - Request Code.  22 = swapping control                    |
|                                                                    |
|   ILOCK - Lock control.  1 = program cannot be swapped            |
|                          0 = program can be swapped               |
|                                                                    |
+--------------------------------------------------------------------+
```

COMMENTS:

MEMORY LOCK CONSIDERATIONS - In order for a program to be allowed to lock itself into a memory partition, the memory lock feature must be enabled at generation time (see RTE-IVB On-Line Generator Manual). If the memory lock feature is not enabled and a program attempts to perform a memory lock, a SC07 error results.

When a program performs a memory lock (ILOC=1), bit 6 of word 15 of its ID segment is set (see Appendix B, "ID SEGMENT FORMAT"). When a program performs a memory "unlock" (ILOCK=0), the bit is cleared. The bit is also cleared if the program aborts or terminates except if the program terminated because of an EXEC program completion call saving resources (see "PROGRAM COMPLETION CALL - EXEC 6").

A- AND B-REGISTER RETURNS---Upon successful completion of the call, the A-register will be meaningless and the B-register will be unchanged. If an error occurred, the A- and B-registers can be examined for the cause (see "EXEC CALL ERROR RETURNS").

EXAMPLE---Program locks itself into memory, performs critical processing, and unlocks itself. Checks for EXEC errors.

```
      PROGRAM LOCK
         .
         .
         .
         .
      ICODE=22+100000B
      ILOCK=1
      CALL EXEC(ICODE,ILOCK)
      GO TO 100
  10  CONTINUE
         .            \
         .             > critical code
         .            /
         .           /
      ILOCK=0
      CALL EXEC(ICODE,ILOCK)
      GO TO 100
  20  CONTINUE
         .
         .
 100  CALL ABREG(IA,IB)
      error processing
         .
         .
         .
      END
```

Program Segment Load — EXEC 8

Loads the calling program's segment from disc into memory and transfers control to the segment's entry point (see Chapter 4, "PROGRAM SEGMENTATION").

```
+-----------------------------------------------------------------+
|                                                                 |
|   CALL EXEC(ICODE,INAME[,IOP1][,IOP2][,IOP3][,IOP4][,IOP5])     |
|                                                                 |
+-----------------------------------------------------------------+
|                                                                 |
|   ICODE - Request code.  8 = segment load.                      |
|                                                                 |
|   INAME - Segment  name.   3-word array  containing the ASCII   |
|           name of the segment to be loaded.                     |
|                                                                 |
|   IOP1 thru IOP5 - Optional  parameters;   passed to  segment   |
|           specified in INAME.                                   |
|                                                                 |
+-----------------------------------------------------------------+
```

COMMENTS:

OPTIONAL PARAMETER PASSAGE---The calling program can pass up to five optional parameters to the segment specified in INAME. The calling program places the values to be passed into IOP1 through IOP5. When the segment load call is executed, these values are placed into the temporary values of the program's ID segment and their address is placed in the B-register. When control is transferred to the segment's entry point, the segment can recover these values by calling the library routine RMPAR (see the DOS/RTE Relocatable Library Reference Manual). Note that the call to RMPAR should be the first executable statement in the segment's code to prevent the B-register from being modified by the segment's execution before RMPAR has used it to access the parameters.

A- AND B-REGISTER RETURNS---When control has been successfully transferred to the segment via its entry point, the contents of the A- and B-registers will be as follows:

* A-register will contain the segment's ID segment address.

* B-register will not be changed unless optional parameters are being passed, in which case the B-register will contain the address of the parameters (used by RMPAR).

If an error occurs it is indicated in the A- and B-registers (see "EXEC CALL ERROR RETURNS"). If the segment to be loaded does not exist, an SC05 error is returned.

PROGRAM TYPES---The main program must be a type 2, 3, or 4 program. The segment must be a type 5 program. The programs type is defined in its program definition statement (see Appendix F, "PROGRAM TYPES").

EXAMPLE---Main program loads its segment and passes it three parameters.

```
        PROGRAM MAIN,3
        DIMENSION INAME(3)
        DATA INAME/2HSE,2HGM,2HN/
          .
          .
        place values in IOP1,IOP3,and IOP5
          .
          .
        CALL EXEC(8,INAME,IOP1,,IOP3,,IOP5)
          .
          .
        END

        PROGRAM SEGMN,5
        DIMENSION IPAR(5)
        CALL RMPAR(IPAR)
        process parameters obtained from main
          .
          .
        END
```

Program Suspend — EXEC 7

Suspends execution of the calling program. Program can be rescheduled by using the GO operator command described in the RTE-IVB Terminal User's Reference Manual.

```
+-------------------------------------------------------------------+
|                                                                   |
|   CALL EXEC(ICODE)                                                |
|                                                                   |
+-------------------------------------------------------------------+
|                                                                   |
|   ICODE - Request code.  7 = program suspend                      |
|                                                                   |
+-------------------------------------------------------------------+
```

COMMENTS:

A-REGISTER, B-REGISTER, AND PARAMETER PASSAGE---The execution of the EXEC 7 call causes the system to place the calling program into the operator suspend list (state 6). The contents of the A- and B-registers are saved. When the program is rescheduled via the GO command (without parameters), the registers are restored to their presuspension status and the program resumes execution. If the program is rescheduled with a GO command that includes parameters, the B-register will contain the address of the parameters and they can be recovered by calling the library routine RMPAR (see DOS/RTE Relocatable Library Reference Manual). The RMPAR call should immediately follow the EXEC 7 call to prevent the B-register from being modified before RMPAR has used it to access the parameters.

Note that when a call to RMPAR is used, the optional parameters must be included in the GO command. If they are not included when the program is rescheduled, RMPAR will use the restored B-register contents as the address of parameters that don't exist. If the parameters are included on a periodic basis, the program should be coded to check for optional parameters and by-pass the RMPAR call if they are not included (see Example).

If an EXEC 7 call is unsuccessful, error information will be returned in the A- and B-registers (see "EXEC CALL ERROR RETURNS"). Note that it is illegal to use an EXEC 7 call to suspend a program running in the batch environment (see the Batch and Spooling Reference Manual). If attempted, an SC00 error is returned.

OTHER SUSPENSION OPERATIONS---The SS operator command described in the RTE-IVB Terminal User's Reference Manual performs the same function as the EXEC 7 call. The FORTRAN-IV PAUSE statement performs the same function as the EXEC 7 call and additionally displays an optional "pause-number" on the session console (see RTE FORTRAN-IV Reference Manual).

EXAMPLE---Suspend the calling program. Use FORTRAN-IV function EXEC call technique to implement optional parameter passage check. If parameters are included in GO command, call RMPAR to recover them. If parameters not included, by-pass RMPAR call.

```
        PROGRAM SPEND
        DIMENSION IPAR(5),IAB(2)
        EQUIVALENCE(REG,IAB(1))
        REG=0.0
        REG=EXEC(7)
        IF(IAB(2)) 200,200,100
100     CALL RMPAR(IPAR)
        process parameters
          .
          .
200     CONTINUE
          .
          .
        END
```

Note that REG=0.0 indirectly causes the A- and B-registers to be set to zero.

Program Completion — EXEC 6

Notifies the operating system that the calling program wishes to terminate itself or a subordinate program.

```
+----------------------------------------------------------------+
|                                                                |
|   CALL  EXEC(ICODE[,INAME][,ICMCD][,IOP1][,IOP2][,IOP3][,IOP4] |
|         [,IOP5])                                               |
|                                                                |
+----------------------------------------------------------------+
|                                                                |
|   ICODE - Request code.  6 = program termination               |
|                                                                |
|   INAME - Program  name.   3-word array containing the  ASCII  |
|           name of the program to be terminated  (must be sub-  |
|           ordinate program).   Set to zero if calling program  |
|           wishes to terminate itself.                          |
|                                                                |
|   ICMCD - Completion code.                                     |
|                                                                |
|   IOP1  thru  IOP5 - Optional  parameters.   Only  used  when  |
|           INAME=0  (self-termination).   Parameter values are  |
|           saved in terminating program's  ID segment;  can be  |
|           recovered when program next executes.                |
|                                                                |
+----------------------------------------------------------------+
```

COMMENTS:

COMPLETION CODE (ICMCD)---The completion code specifies the manner in which the program indicated by INAME is to be terminated. The codes are summarized below:

* ICMCD = 0   Normal completion.

* ICMCD = -1   Serially reusable completion. When rescheduled, the program is not reloaded into memory if it is still resident.   ICMCD = -1 should only be used with disc resident programs that can initialize their own buffers or storage locations. The program is reloaded from disc only if it has been overlaid by another program, therefore, the program must be able to maintain the integrity of its data in memory.

* ICMCD = 1   Save resources completion. Makes program dormant; saves suspension point and all resources allocated to the program. When program is rescheduled, it will continue executing from point of suspension and have access to previously allocated resources. If a program does not terminate itself, it can only be rescheduled by its father or by the RU or ON operator command. If a program does terminate itself, it can be rescheduled by any stimulus (EXEC, operator, time, or interrupt scheduling). An EXEC 6 call with ICMCD = 1, is similiar to the EXEC 7 call described earlier or the SS operator command described in the RTE-IVB Terminal User's Reference manual.

* ICMCD = 2   Terminate program and remove it from the time list. If program is I/O suspended, the system waits until the I/O operation completes before setting the program dormant. Disc tracks allocated to the program are not released. An EXEC 6 with ICMCD = 2 is equivalent to the OF,name,0 operator command (see RTE-IVB Terminal User's Reference Manual).

* ICMCD = 3   Immediately terminate program, remove it from time list, and release all disc tracks allocated to it. If program is I/O suspended, a system generated clear request is sent to the driver. An abort message is displayed on the system console. An EXEC 6 with ICMCD = 3 is equivalent to the OF, name,1 operator command (see RTE-IVB Terminal User's Reference Manual).

OPTIONAL PARAMETER PASSAGE---If INAME = 0 (self-termination) and the optional parameters are included in the EXEC 6 call, they are stored in the program's ID segment, word 1 thru 5, (see Appendix B, "ID SEGMENT FORMAT") along with their address which is placed in the B-register save word (word 10). When the program is again scheduled to execute, it can obtain the parameters by calling the library routine RMPAR (see the DOS/RTE Relocatable Library Reference Manual). The RMPAR call should be the first statement executed when the program is rescheduled to prevent the restored B-register from being modified before RMPAR has used it to access the parameters. The use of the optional parameters allows a program being scheduled from the time list to use the same parameters each time it executes. If the program is to be run interactively it must be scheduled by the system RU command.

A- AND B-REGISTER RETURNS---For successful EXEC 6 calls, the contents of the A- and B-registers are summarized below:

* A-register.   Unchanged.

* B-register.   Unchanged (no optional parameters specified) or address of optional parameters.

Note that the RTE FORTRAN-IV Compiler generates an EXEC 6 when it compiles an END statement.

Note also that a father can terminate its son normally (ICMCD=0) or with the son saving resources (ICMCD=1).

EXAMPLE---Before program terminates itself, it obtains the current time (see, "TIME REQUEST - EXEC 11"). The time is placed into the optional parameters used in the EXEC 6 call. When the program is rescheduled it obtains the saved time values and prints them on the session console (LU1).

```
        PROGRAM EXC6
        DIMENSION ITM(5),IPAR(5)
        CALL RMPAR(IPAR)
        IHRS=IPAR(4)
        IMIN=IPAR(3)
        ISEC=IPAR(2)
           .
           .
        WRITE(1,100)IHRS,IMIN,ISEC
100     FORMAT("PROGRAM LAST RUN AT",I2,":",I2,":",I2)
        CONTINUE
           .
           .
        INAME=0
        ICMCD=0
        CALL EXEC(11,ITM)
        CALL EXEC(6,INAME,ICMCD,0,ITM(2),ITM(3),ITM(4))
        END
```

Status EXEC Calls

The status EXEC calls are used to obtain information about the operating environment. This includes obtaining the current time indicated by the 24-hour system clock, obtaining size and status information about memory partitions, and determining the present status of an I/O device.

The status EXEC calls are listed below in the order of their presentation:

* TIME REQUEST - EXEC 11

* PARTITION STATUS - EXEC 25

* MEMORY SIZE - EXEC 26

* I/O STATUS - EXEC 13

Time Request — EXEC 11

Request the current time indicated by the 24-hour real-time clock.

```
+------------------------------------------------------------+
|                                                            |
|   CALL EXEC(ICODE,ITIME,[IYEAR])                           |
|             -----  -----                                   |
|                                                            |
+------------------------------------------------------------+
|                                                            |
|   ICODE - Request code.  11 = time request.                |
|                                                            |
|   ITIME - 5-word array.   Time indicated by a real-time clock |
|           is returned by the system as follows:            |
|                                                            |
|           ITIME(1) = 10's of milliseconds                  |
|           ITIME(2) = Seconds                               |
|           ITIME(3) = Minutes                               |
|           ITIME(4) = Hours                                 |
|           ITIME(5) = Day of the year                       |
|                                                            |
|   IYEAR - Optional one-word variable.   Current year is re- |
|           turned, by the system as a 4-digit number.       |
|                                                            |
+------------------------------------------------------------+
```

COMMENTS:

A- AND B-REGISTER RETURNS---Upon successful completion of this call, the A-register contents will be meaningless and the B-register will be unchanged. If an error occurs, it is indicated in the A- and B-register (see "EXEC CALL ERROR RETURNS").

ALTERNATE METHODS---The EXEC 11 call is similiar to the TI operator command described in the RTE-IVB Terminal Users' Reference Manual. The library routine FTIME can also be used to obtain the current time (see Chapter 6, "SYSTEM LIBRARY ROUTINES").

EXAMPLE---Obtain the current time on the real-time clock as well as the current year. Print out date and time as hrs:mins day/year on printer (LU6).

```
          PROGRAM TMRQT
          DIMENSION ITIME(5)
            .
            .
          ICODE=11
          CALL EXEC(ICODE,ITIME,IYEAR)
          IHR=ITIME(4)
          IMIN=ITIME(3)
          IDAY=ITIME(5)
          WRITE(6,20)IHR,IMIN,IDAY,IYEAR
   20     FORMAT (" CURRENT TIME AND DATE ",I2,":",I2," ",I3,"/",I4)
          CONTINUE
            .
            .
          END
```

Partition Status — EXEC 25

Requests system to return status information about a specified memory partition.

```
+------------------------------------------------------------------+
|                                                                  |
|   CALL EXEC(ICODE,IPART,IPAGE,INPGS,IPST)                        |
|              ----- ----- ----                                    |
|                                                                  |
+------------------------------------------------------------------+
|                                                                  |
|   ICODE - Request code.  25 = partition status.                 |
|                                                                  |
|   IPART - Number  of partition  that information is being re-    |
|           quested on.                                            |
|                                                                  |
|   IPAGE - starting page number of partiton.                     |
|                                                                  |
|   INPGS - Number of pages in partition (including base page).   |
|                                                                  |
|   IPST -  Partition status word.                                |
|                                                                  |
+------------------------------------------------------------------+
```

COMMENTS:

PARTITION STATUS WORD (IPST) - The format of IPST is as follows:

```
        (RS) (RT) (M) (S) (C)
        -----------------------------------------------------------
        | 15| 14| 13| 12| 11| XX| XX| XX| 7| 6| 5| 4| 3| 2| 1| 0|
        -----------------------------------------------------------
                                           |                    |
                                           | --ID segment number-- |
```

The meaning of the IPST fields are summarized below:

* bit 15 (RS).   Set to "1" if partition is reserved.

* bit 14 (RT).   Set to "1" if partition is a real-time partition.

* bit 13 (M).   Set to "1" if partition is a mother partition.

* bit 12 (S).   Set to "1" if partition is a sub-partition of a mother partition.

* bit 11 (C).   Set to "1" if chain-mode is in effect, i.e., subpartition is linked to an active mother partition.

* bit 8-10.   Not used.

* bit 0-7.   ID segment number. Set to the ordinal number (index into keyword block) of the ID segment for the program that occupies the partition. Set to zero if partition is not occupied.

A- AND B-REGISTER RETURNS---For successful calls the A-register contents will be meaningless and the B-register will be unchanged. For unsuccessful calls, error information is returned (see, "EXEC CALL ERROR RETURNS").

If the partition number (IPART) is illegal, -1 will be returned in INPGS and zero will be returned in IPAGE. For more information on partitions, refer to Chapter 1, "MEMORY MANAGEMENT".

Note that the number of pages and the starting page number of all partitions can be obtained by running the utility program WHZAT described in the RTE-IVB Terminal User's Reference Manual.

EXAMPLE---Obtain size information on all the partitions in the system. Print results as a utility report on session console (LU1).

```
      PROGRAM PUTIL
        .
        .
      IPART=1
  50  CALL EXEC(25,IPART,IPAGE,INPGS,IPST)
      IF(INPGS.EQ.-1)GO TO 150
      WRITE(1,100)IPART,IPAGE,INPGS
 100  FORMAT(" PARTITION NUMBER ",I2," STARTS AT PAGE ",I4,
     *" AND IS ",I2," PAGES LONG ")
      IPART=IPART+1
      GO TO 50
 150  CONTINUE
        .
        .
      END
```

Memory Size — EXEC 26

Returns the memory limits of the partition that the calling program is currently executing in.

```
+------------------------------------------------------------------+
|                                                                  |
|   CALL EXEC(ICODE,IFAW,ILMEM,INPGS[,IMAP])                       |
|             ----  -----  -----  ----                             |
|                                                                  |
+------------------------------------------------------------------+
|                                                                  |
|   ICODE - Request code.  26 = memory size.                       |
|                                                                  |
|   IFAW -  Address of first  available word behind the calling    |
|           program, e.g., last word of program, plus length of    |
|           largest segment, plus 1.                               |
|                                                                  |
|   ILMEM - Number of words available between the last word  of    |
|           the program and the last word of the programs  add-    |
|           ress space.                                            |
|                                                                  |
|   INPGS - Length of the partition (including base page)  that    |
|           program currently resides in.                          |
|                                                                  |
|   IMAP -  32-word  array into  which a copy of the  currently    |
|           enabled user map is returned.                          |
|                                                                  |
+------------------------------------------------------------------+
```

COMMENTS:

PARAMETER RELATIONSHIPS---The system calculates ILMEM by subtracting IFAW from the address of the last word of the program's logical address space. A program's logical address space is determined at load-time and can be equal to the program's size (default) or greater than the program's size (using LOADR size-override option). Furthermore, the partition a program resides in may be greater than or equal to the program's logical address space.

For EMA programs, ILMEM corresponds to the number of words between IFAW and the beginning of MSEG (see Chapter 5, "EMA PROGRAMMING").

Figure 2-1 illustrates the relationships between the EXEC 26 returned parameter values.

A- AND B-REGISTER RETURNS---For successful EXEC 26 calls, the returned A-register contents will be meaningless and the B-register contents will be unchanged. For unsuccessful calls, error information is returned (see, "EXEC CALL ERROR RETURNS").
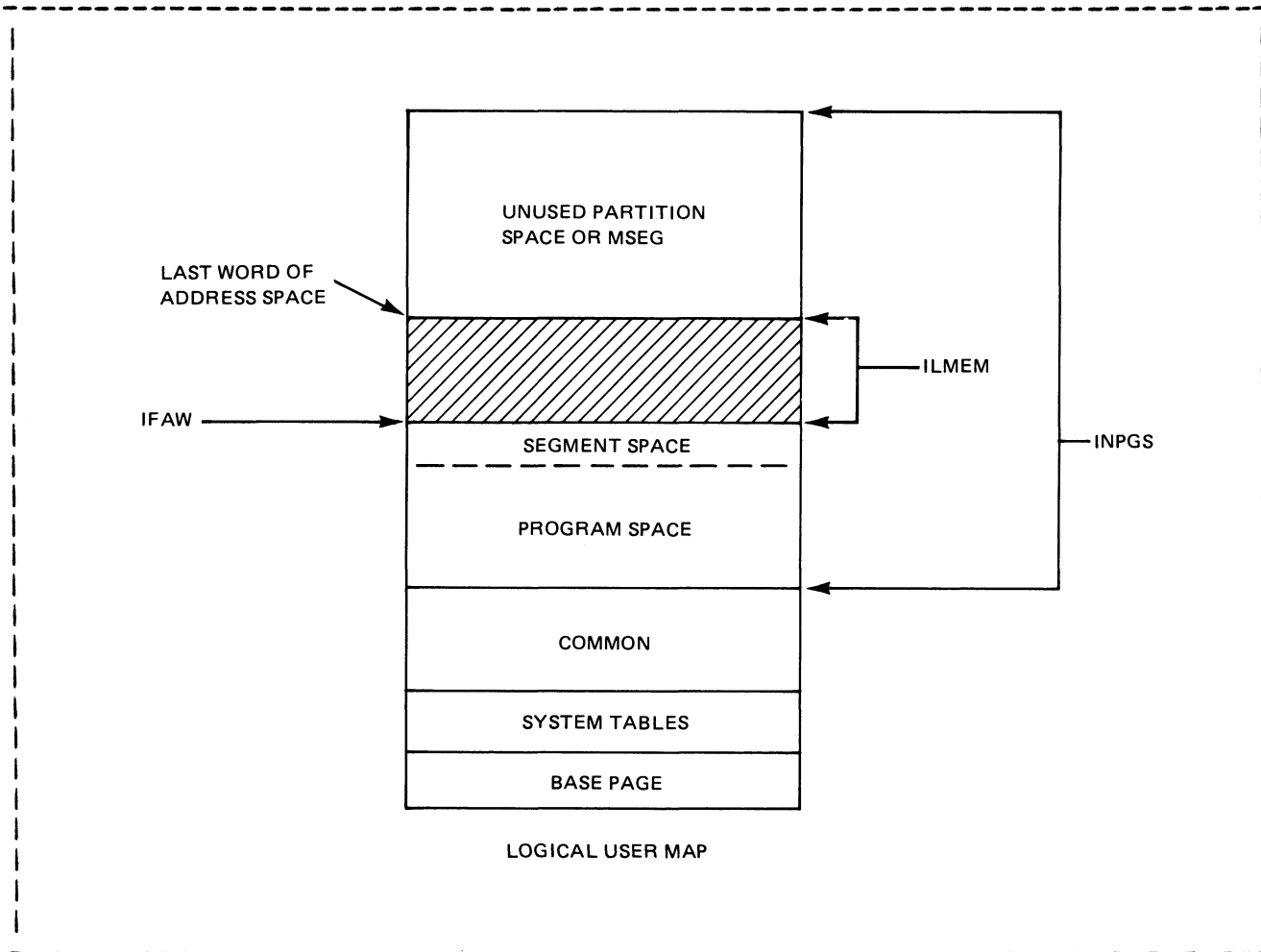
Figure 2-1. EXEC 26 Parameter Relationships.

EXAMPLE---FORTRAN program makes an EXEC 26 call to determine the size
of the unused portion of its logical address space. The size and the
address of the first word of the space are passed to a user-written
Assembler routine (USER) that uses the space for temporary data
storage.

```
        PROGRAM DYALC
          .
          .
        ICODE=26
        CALL EXEC(ICODE,IFWA,ILMEM,INPGS)
          .
          .
        CALL USER(ILMEM,IFWA,..)
          .
          .
        END
```

I/O Status — EXEC 13

Requests information (status and device type) about device associated with a specified Logical Unit number.

```
+-------------------------------------------------------------------+
|                                                                   |
|   CALL EXEC(ICODE,ICNWD,IST1[,IST2][,IST3])                       |
|                                                                   |
+-------------------------------------------------------------------+
|                                                                   |
|   ICODE  - Request code.  13 = I/O status.                        |
|                                                                   |
|   ICNWD  - Session Logical Unit number (0-63) of device that status|
|            is being requested on.                                 |
|                                                                   |
|   IST1   - Status word.    Word 5  of  the devices' EQT entry  (see|
|            COMMENTS).                                              |
|                                                                   |
|   IST2   - Status  word.   Word 4  of  the devices' EQT entry  (see|
|            COMMENTS).                                              |
|                                                                   |
|   IST3   - Status word. Specifies whether device is "up" or "down".|
|            Also indicates subchannel associated with the device.  |
|                                                                   |
+-------------------------------------------------------------------+
```

COMMENTS:

STATUS WORDS---The contents of IST1 and IST2 are obtained from words 5 and 4, respectively, of the devices Equipment Table (EQT) entry. Since the status is obtained from the EQT and not the driver, the status obtained from a buffered device or a device with multiple subchannels, may not reflect the current device status. The format of these words is shown in Table 2-1 and Table 2-2.

The contents of IST3 is obtained from the Device Reference Table (DRT) entry associated with the device (see Appendix C, "DEVICE REFERENCE TABLE FORMAT"). Bit 15 of IST3 will be set to "1" if the device is down. Bit 15 will be cleared to "0" if the device is up. Bits 4 through 0 of IST3, will indicate the subchannel associated with the device.

If an LU greater than 63 is specified in ICNWD, the status of the LU with the value of (ICNWD-63) is returned.

A- AND B-REGISTER RETURNS---The contents of the A- and B-registers for a successful EXEC 13 call are meaningless. For unsuccessful calls, error information is returned (see, "EXEC CALL ERROR RETURNS").

EXAMPLE---Program checks the LU passed to it in IPAR(1) to determine
if it is associated with device type 05 or 07. If it is, execution
continues, if it is not, the program terminates.

```
        PROGRAM INTAC
        DIMENSION IPAR(5)
        CALL RMPAR(IPAR)
        ICNWD=IPAR(1)
        MASK5=2400B
        MASK7=3400B
        CALL EXEC(13,ICNWD,IST1)
        ITEST=IAND(IST1,037400B)
        IF(ITEST.EQ.MASK5) GO TO 100
        IF(ITEST.EQ.MASK7) GO TO 100
        GO TO 200
           .
           .
  100   CONTINUE
           .
           .
  200   CALL EXEC(6)
        END
```

RMPAR and IAND are described in the DOS/RTE Relocatable Library
Reference Manual.

Table 2-1   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 2-2)								

ISTA2	D	=	1 if DMA 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).
	Unit	=	Last sub-channel addressed.
	Channel	=	I/O select code for device (lower number if a multi-board interface).

ISTA1

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.

EQUIP. TYPE CODE = type of device. When this number is linked with "DVR." it identifies the device's software driver routine:

00 to 07_8 = 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
           subchannel 3   = graphics
           subchannel 4   = aux. printer

07   multipoint driver

10 to 17 = unit record devices

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

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.

Table 2-2   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

Class I/O EXEC Calls

The class I/O feature of RTE-IVB is implemented by a special set of I/O EXEC calls. These class I/O calls provide user programs with I/O and communication capabilities not available with the standard I/O EXEC calls discussed earlier in this section. The features provided by class I/O are summarized below:

* I/O without wait - allows a program to continue executing concurrently with its own input operation (class READ) or output operation to an unbuffered device (class WRITE).

* Mailbox I/O - allows cooperating programs to communicate via a controlled access data buffer.

* Data passage synchronization - prevents communicating programs from processing incomplete or non-updated data; a program can suspend itself until it receives a signal indicating that valid data is available from another program.

* I/O control without wait - allows a program to initiate a control operation on an I/O device and continue executing without waiting for the control operation to complete.

Class I/O is based on the use of a buffer in System Available Memory (SAM) with an associated "access key". The access key is known as a class number; the class number and the buffer are collectively known as a class.

Note that the use of class I/O is exclusive of system or local COMMON used in standard program-to-program communication.

Class I/O can be considered "double-call" I/O. One call is necessary to initiate the operation and another call is necessary to complete the operation. The initiation call (class READ, WRITE, WRITE/READ, or CONTROL) places request parameters, plus data if required, in the class buffer in SAM. The completion call (CLASS GET) retrieves any data and releases the request. The class number must be used as a parameter in the GET call, thereby insuring that only authorized programs (programs "knowing" class number) can access the buffer. If a program other than the program that initiated the I/O operation wishes to retrieve the results, the class number must be made available to the retrieving program (via COMMON, in command string, etc.). Note that once a class I/O operation is initiated, the calling program has the option of either continuing with its execution or waiting for the operation to complete.

A class number is allocated when a user program makes an initial class I/O call and requests a class number. The class remains allocated until a program deallocates it. The class should always be deallocated when it is no longer needed; freeing it for use by other programs. The maximum number of classes (1-255) is established at generation time (see the RTE-IV3 On-Line Generator Reference Manual). Note that a program can have more than one class number allocated to itself.

A buffer in SAM is allocated each time a class I/O operation is initiated. The buffer contains the request parameters and, optionally, data. When the operation is completed (using GET call), the buffer can be released or retained at the users option.

When a class I/O request is made (i.e., READ, WRITE, etc.) it is queued on the specified class number. This is known as the "pending class queue". The request remains in the pending class queue until the driver has received the request and processed it accordingly.

When the driver has completed the specified operation, the request is linked to the "completed class queue" associated with the class number. The results of the operation are then available to the calling program (or another program) via a GET call. Note that the queueing technique (pending and complete) allows more than one buffer to be associated with the same class number, i.e., a program can make multiple requests specifying the same class number.

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 in SAM (class READ, WRITE or WRITE/READ). The user program can 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.

A user program recovers the results of a Class I/O call by 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 reissuing 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 class buffer. The calling program (or another 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, as well as, program-to-program communication, is given in Appendix K, "CLASS I/O APPLICATION EXAMPLES".

A more detailed explanation of how the RTE-IVB operating system handles class I/O is given later in this section (see "SYSTEM CLASS I/O CONSIDERATIONS").

The Class I/O EXEC calls are listed below in the order of their presentation:

* CLASS READ, WRITE and WRITE/READ - EXEC 17, 18 and 20.

* CLASS I/O CONTROl - EXEC 19

* CLASS GET - EXEC 21

Class I/O Read, Write, and Write/Read — EXEC 17, 18 and 20

Initiates the transfer of information to/from a non-disc I/O device, or to another program.

```
+--------------------------------------------------------------+
|                                                              |
|   CALL EXEC(ICODE,ICNWD,IBUFF,ILEN,IOP1,IOP2,ICLAS)          |
|                                                              |
+--------------------------------------------------------------+
|   ICODE - Request code. 17=Read,  18=Write,  20 = Write/Read.|
|                                                              |
|   ICNWD - Control word. Specifies the LU of device involved  |
|           in data transfer, driver dependent information and |
|           optional parameter (IOP1 and IOP2) considerations. |
|                                                              |
|   IBUFF - Data buffer. For READ operations, a dummy parameter;|
|           for WRITE and WRITE/READ operations, the array     |
|           where the program places the data to be written.   |
|                                                              |
|   ILEN -  Data length. Positive number of words or negative  |
|           number of characters to be read or written.        |
|                                                              |
|   IOP1 -  Optional parameter (see COMMENTS).                 |
|                                                              |
|   IOP2 -  Optional paramter (see COMMENTS).                  |
|                                                              |
|   ICLAS - Class word (see COMMENTS).                         |
+--------------------------------------------------------------+
```

COMMENTS:

CONTROL WORD (ICNWD)---The format of the control word is as follows:

```
      (Z)      (X)(A)(K)(V)(M)
------------------------------------------------------
|XX|XX|XX|12|XX|10| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0|
------------------------------------------------------
                |                        |
                |<--Function-->|<-Logical  Unit->|
                      Code
```

The function of the control word fields are summarized below:

*   Logical Unit (bit 0-5). Logical unit number (LU) of the device that data is to be transferred to or from. Must be LU of non-disc device. If LU is set to zero, the call is executed but no data is transferred to/from an external device.

*   Function Code (bit 6-10). Specifies control information for the driver module associated with the I/O device involved in the data transfer. This information is driver dependent and the appropriate driver reference manual should be consulted for more information. The function code can be defaulted to zero if no special action is required.

*   Z-bit (bit 12). The Z-bit is set to "1" if an additional control buffer is to be passed to the driver module or passed to the program that performs the GET call. The control buffer is defined by the optional parameter (IOP1 and IOP2).

*   Bit 11 and 13-15. These bits are used by the system and should be set to zero by the user.

ILEN---This variable defines the length of the data record to be transferred. A positive number is used to specify the length of the data record in words. A negative number is used to specify the length of the data record in characters (bytes). The data type must be allowed for in ILEN. For example, a data record containing 10 REAL values would require ILEN to be +20 or -40 since each REAL value requires two words. If the data record contains double-precision data, three or four words are required for each value, depending on the options taken at generation time for the FORTRAN compiler (see RTE FORTRAN-IV Reference Manual).

OPTIONAL PARAMETERS (IOP1 AND IOP2)---If the Z-bit is set, IOP1 defines the address of an optional control buffer and IOP2 defines the length of the buffer in positive words or negative characters. The control buffer can contain driver dependent information (the user should refer to the appropriate driver reference manual for more details) or control information to be used by the program that issues the GET call.

If the Z-bit is clear, IOP1, and IOP2 can be used to pass one-word parameters to the GET (EXEC 21) call described later in this section.

```
FTN4,L
      PROGRAM IOPT
      DIMENSION IBUF(10)
      DATA IBUF/2HHI,2H T,2HHE,2HRE,2H!!.PH! /
  100 CONTINUE
      ICLAS=0
      WRITE(1,1)
    1 FORMAT(/5X,"PLEASE ENTER IOP1...<-")
      READ(1,*) IOP1
      IF(IOP1 .EQ. 0) STOP
      WRITE(1,2)
    2 FORMAT(/5X,"PLEASE ENTER IOP2...<-")
      READ(1,*) IOP2
      CALL EXEC(18,1,IBUF,6,IOP1,IOP2,ICLAS)
      WRITE(1,10) IOP1,IOP2
   10 FORMAT(/5X,"CLASS WRITE INITIATED!! - IOP1 = ",I6," IOP2 = ",I6)
      CALL EXEC(21,ICLAS,IBUF,6,IP1,IP2,IP3)
      WRITE(1,20) IP1,IP2,IP3
   20 FORMAT(/5X,"IP1 = ",I6," IP2 = ",I6," IP3 = ",I6)
      GO TO 100
      END
```

FTN4 COMPILER: HP92060-16092 REV. 2001 (791101)

A- AND B-REGISTER RETURNS---When the user's program issues a Class I/O call, the system allocates a buffer from System Available Memory and puts the request in the buffer. 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 either aborted or the "no abort" error return is taken (IO04), depending on the state of the no-abort bit in ICODE.

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.

.The A-register will contain -1 if the "no-wait" bit was set and the program tried to allocate a class number with no class numbers available.

The A-register will contain zero if the request was successful.

The returned content of the B-register is meaningless. Error information is returned to the A- and B-register for unsuccessful calls (see, "EXEC CALL ERROR RETURNS").

CLASS WRITE---The general flow of a Class WRITE operation is as follows:

1. User places data in IBUFF, specifies data length in ILEN, and issues an EXEC 18 call specifying a previously allocated class number in ICLAS (if ICLAS = 0, a class number will be allocated, if available).

2. The system allocates a buffer in SAM (if available) large enough to contain the data plus the request parameters, places the data from IBUFF plus the request parameter into the buffer, and links the buffer into the Pending Class Queue associated with ICLAS. The calling program continues executing or suspends itself with an EXEC 21 call to the class number.

3. The request is queued (according to program priority) on the EQT associated with the LU specified in ICNWD.

4. When the driver completes the WRITE operation, the request portion of the buffer is linked into the completed class queue. The data portion of the buffer is released back to the system. A program suspended by a previous GET call (EXEC 21) on the class number is rescheduled. See, "CLASS GET - EXEC 21", for details associated with the GET call.

CLASS READ---The general flow of a Class Read operation is as follows:

1. User issues an EXEC 17 call specifying a previously allocated class number in ICLAS (if ICLAS = 0, a class number will be allocated, if available). The amount of data to be transferred from the external I/O device to a buffer in SAM is specified in ILEN.

2. The system allocates a buffer in SAM large enough to contain the data plus the request parameters, places the request parameters in the buffer, and links the buffer into the Pending Class Queue associated with ICLAS. The calling program continues executing or suspends itself with an EXEC 21 call to the class number.

3. The request is then queued (according to the calling program's priority) on the EQT associated with the LU specified in ICNWD.

4. When the driver completes the transfer of data from the external I/O device to the buffer in SAM, the request portion, plus the data portion, of the buffer are linked into the completed class queue. A program suspended by a previous GET call (EXEC 21) on the class will be rescheduled. See, "CLASS GET - EXEC 21", for details concerning the GET call.

CLASS WRITE/READ---The general flow of a class WRITE/READ request nas characteristics of both the class WRITE and class READ calls described above. The flow progresses like a class WRITE call until the point of driver completion is reached. In a class WRITE call, only the request portion of the class buffer would be linked into the completed class queue; for a Class WRITE/READ, botn the request portion plus the data portion of the class buffer is linked into tne completed class queue and flow continues like a class READ request. Hence the name WRITE/READ.

The Class WRITE/READ call with LU=0 is used for program-to-program communication. The data is transferred from the user program's buffer into the class buffer and written to LU 0 ("bit-bucket"). The data is retained in the completed class queue to be recovered oy an EXEC·21 call from another program.

CLASS WORD (ICLAS)---The format of the class word is as follows:

(NW)(S)(DA)
```
+-----------------------------------------------+
|15|14|13| 12| 11| 10| 9| 8| 7| 6| 5| 4| 3| 2| 1| 0|
+-----------------------------------------------+
        |                                       |
        | <-----------Class Number----------> |
```
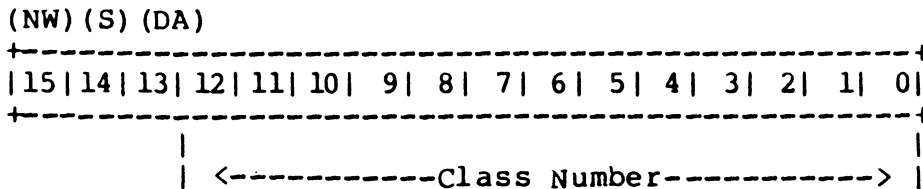
The function of the class word fields are described below:

* CLASS NUMBER (bit 0-12) - To obtain a class number from tne system, the bits are set to zero. This causes the system to allocate a class number (if one is availaole) to the calling program. The number is returned to the ICLAS parameter when tne call completes and the user must specify this parameter (unaltered) when using it for later calls.

* "DE-ALLOCATE" BIT (bit 13) - (see "CLASS GET - EXEC 21")

* "SAVE" BIT (bit 14) - (see "CLASS GET - EXEC 21")

* "NO-WAIT" bit (bit 15) - When set, the calling program does not memory suspend if memory (or a class number) is not availaole.

Class I/O Control — EXEC 19

Performs various I/O control operations such as backspace, write
end-of-file, rewind, etc. The calling program does not wait for the
operation to complete.

```
+-----------------------------------------------------------------+
|                                                                 |
|   CALL EXEC(ICODE,ICNWD,IPRM,ICLAS[,IOP1][,IOP2])               |
|                                                                 |
+-----------------------------------------------------------------+
|                                                                 |
|   ICODE - Request code.  19 = I/O control.                      |
|                                                                 |
|   ICNWD - Control  word.   Specifies  the  LU and the control   |
|           function to be carried out on that LU.                |
|                                                                 |
|   IPRM  -  Optional paramter.  Required for some control func-  |
|           tions. Must be used as a place holder if not required |
|           for the control operation.                            |
|                                                                 |
|   ICLAS - Class word.                                           |
|                                                                 |
|   IOP1 and IOP2 - Optional parameters.                          |
|                                                                 |
+-----------------------------------------------------------------+
```

COMMENTS:

PARAMETER CONSIDERATIONS---ICNWD and IPRM perform the same function
for the class I/O control operation (EXEC 19) as for the standard I/O
control operation (EXEC 3) described earlier.

ICLAS is described earlier in this section with the discussion of the
class READ, WRITE, and WRITE/READ EXEC calls.

IOP1 and IOP2 can be used to pass information to the EXEC 21 call
described later in this section.

A- AND B-REGISTER RETURNS--For successful EXEC 19 calls, the
A-Register contains zero; the contents returned to the B-Register are
meaningless. Error information is returned to the A- and B-Registers
for unsuccessful EXEC 19 calls (see, "EXEC CALL ERROR RETURNS").

GENERAL FLOW---The general flow of the EXEC 19 call is as follows:

1. User program makes EXEC 19 call specifying the LU and operation
   to be performed in ICNWD. A previously allocated class number is
   specified in ICLAS (if ICLAS=0, a class number is allocated, if
   available).

2. The system allocates a buffer in SAM, places the request parameters in the buffer, and links the buffer into the Pending Class Queue associated with ICLAS. The calling program can continue executing or suspend itself with an EXEC 21 call to the class number.

3. The request is queued (according to program priority) on the EQT associated with the LU specified in ICNWD.

4. When the driver completes the control operation, the request is linked into the Completed Class Queue. A program suspended by a previous GET call (EXEC 21) to the class is rescheduled. See, "CLASS GET - EXEC 21", for details associated with the GET call.

Class GET — EXEC 21

Completes the operation previously initiated by a Class READ, WRITE, WRITE/READ or CONTROL request.

```
+-------------------------------------------------------------------+
|                                                                   |
|   CALL EXEC(ICODE,ICLAS,IBUFF,ILEN[,IOP1][,IOP2][,IOP3])          |
|                          -----       ----   ----   ----           |
+-------------------------------------------------------------------+
|                                                                   |
|   ICODE - Request code.  21 = Class GET.                          |
|                                                                   |
|   ICLAS - Class word.                                             |
|                                                                   |
|   IBUFF - Data buffer.  Array (in calling program) where the      |
|           system places retrieved data.                           |
|                                                                   |
|   ILEN -  Data length.  Positive number of words or negative      |
|           number of characters (bytes) to be retrieved.           |
|                                                                   |
|   IOP1, IOP2, and IOP3 - Optional parameters; IOP1 and IOP2       |
|           are obtained from the original class request, IOP3      |
|           is returned by the system.                              |
|                                                                   |
+-------------------------------------------------------------------+
```

COMMENTS:

CLASS WORD (ICLAS)---The format of the class word is described earlier in this section (see, "CLASS READ, WRITE, and WRITE/READ - EXEC 17, 18, and 20"). The function of bits 15, 14, and 13 of the class word are described below:

* "No Wait" Bit (15)---When set, the calling program is not suspended if the results of the operation are not yet available, i.e., the request has not been linked into the Completed Class Queue.

* "Save" Bit (14)---When set, the buffer in the Completed Class Queue is not released; a subsequent GET call will return the same data.

* "Deallocate" Bit (13)---When set, the class number is not released back to the system. If bit 13 is cleared to zero and no requests are left in the Pending Class Queue, and no requests for the class are pending on the driver for processing, the class number is released to the system.

Bits 14 and 13 work in conjunction with each other. If bit 14 is set, the buffer is not released and the class number cannot be released because there will still be an outstanding request against it.

Note that the class number should always be released when it is no longer needed since the system does not automatically release the number when the allocating program terminates.

Only when the GET call retrieves the results of the last request on a class or examines an empty class queue can the class number be released by clearing bit 13 in the class word (ICLAS).

DATA LENGTH (ILEN)---This variable defines the length of the data record to be retrieved; the data type must be allowed for. If the data record contains REAL values, two words per value are required. If the data record contains double-precision data, three or four words per value are required depending on the options taken at generation time for the FORTRAN compiler (see the appropriate FORTRAN Reference Manual).

Note that ILEN must allow for the length of the optional output buffer (Z-bit set in original call) if it is to be retrieved.

OPTIONAL PARAMETERS (IOP1,IOP2, AND IOP3)---Optional parameters from the Class READ, WRITE, WRITE/READ, or CONTROL calls are returned in IOP1 and IOP2. These words are protected from modification by the driver. The original request code received by the driver is returned in IOP3 as follows:

ORIGINAL REQUEST CODE	IOP3 RETURN
17/20 (READ,WRITE/READ)	1
18 (WRITE)	2
19 (CONTROL)	3

GENERAL FLOW---When a program issues a class GET call, the program is telling the system that it is ready to accept data provided by a previous class READ or WRITE/READ call, or remove a completed class WRITE or CONTROL request from the Completed Class Queue. If the driver has not yet completed (GET call was issued before request was linked into completed class queue), the calling program is suspended in the general wait list (state 3) and a marker so indicating 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.

One of the features of the GET call is that a user program waiting for system resources can suspend itself without CPU overhead or program overhead such as polling. A program can issue a GET call on a class number associated with a device or another program and thereby suspend itself. The program will be rescheduled when there is something to process; the requested data will be available. After the data is processed, the program can again suspend itself with another GET call.

BUFFER CONSIDERATIONS---There are several buffer considerations in using the class GET call:

* The number of words returned to IBUFF is the lesser of:

    a. the number requested (ILEN specified in GET call)

    b. the number in the Completed Class Queue element being retrieved. (ILEN specified in original request.)

* If the original request was made with the Z-bit set in the control word (ICNWD), the returned value of IOP1 will be meaningless.

* The "Z-buffer" will be returned only if the original request was a READ or WRITE/READ call; for WRITE requests, no data is returned. Note that ILEN must allow for the length of the Z-buffer e.g. ILEN = length of original request buffer + length of Z-buffer.

* The remaining words in IBUFF (if any) past the number indicated by the transmission log (B-register), are undefined. If a "Z-buffer" is also returned, the words remaining past the end of the Z-buffer are undefined.

A- AND B-REGISTER RETURNS---The A- and B-register contents after the return from a successful EXEC 21 call are as follows:

If a return is made with data, then:

    A-register (bit 15) = 0
    A-register(bit 14-0)=status (word 5 of devices EQT entry)
    B-register=transmission log (positive number of words of
                   characters transferred depending on original request.

If a return is made without data ("no wait" bit set in ICLAS), then:

    A-register = negative number of (requests +1) made to the class
                   but not yet serviced by the driver (pending class
                   requests).

    B-register = meaningless.

For unsucessful calls, error information is returned in the A- and B-registers (see "EXEC CALL ERROR RETURNS").

System Class I/O Consideration

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 (SAM) and puts the request parameters 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 calling program.

b. If this is the only request pending on the EQT, the driver is called immediately; otherwise, the system returns control to the class user and queues the request according to the calling program's priority.

c. If buffer space is not available in SAM, the class user is memory suspended unless bit 15 ("no wait") is set in the class word. If the "no wait" bit is set, control is returned to the calling program 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 IO04 error return.

e. If a 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. If the "no-wait" bit is set, the program is not suspended and the A- register will indicate the condition.

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 request (in the form of a standard I/O call) and completes, the system will:

a. Release the data buffer portion of the request if a WRITE. The header is retained for the GET call.

b. Queue the header portion of the buffer (plus data if required) in the Completed Class Queue.

c. If a GET call is pending on the Class Number, reschedule the calling program. 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 data if applicable). The calling program has the option of leaving the I/O request in the Completed Class Queue so as not to lose the data (a subsequent GET call will obtain the same data), or dequeuing the request and releasing the header and buffer (can also release the Class Number back to the system).

b. If the driver has not yet 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).

Executive Error Messages

When RTE-IVB 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 session terminal if in session environment) 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 Chapter 2 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 2-3 contains a summary of all possible errors associated with EXEC calls.

Memory Protect Violations

The RTE-IVB 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 yyyyy    zzzzz         (yyyyy=program name; zzzzz=violation address)
yyyyy ABORTED
```

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-existant memory or to some other program's memory. In this case, the program is aborted and the following message is issued:

```
DM VIOL = wwwww           (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 yyyyy    zzzzz         (yyyyy=program name; zzzzz=violation address)
yyyyy ABORTED
```

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 ppppp qqqqq r   (contents of A,B, and E registers at abort)
XYO ppppp qqqqq r   (contents of X,Y, and O registers at abort)
DP   yyyyy zzzzz    (yyyyy = program name; zzzzz = violation address)
yyyyy aborted
```

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 ppppp qqqqq r   (contents of A,B and E registers at abort)
XYO ppppp qqqqq r   (contents of X,Y and O registers at abort)
EX   yyyyy zzzzz    (yyyyy=program name; zzzzz=violation address)
yyyyy ABORTED
```

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 occuring 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 ppppp qqqqq r        (contents of A, B and E registers at abort)
XYO PPPPP qqqqq r        (contents of X, Y and O registers at abort)
DM <INT>      0

        or

MP INST = xxxxxx         (offending octal instruction code)
ABE ppppp qqqqq r        (contents of A, B and E registers at abort)
XYO ppppp qqqqq 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.

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 (between 1 and 26)
	An RQ00 error means that the address of a returned parameter is below the memory protect fence.

TRACK ERROR (DISC PARITY)

Upon detecting a disc parity error, the program that encountered the error is aborted and the following message is issued:

```
                          S
     TR nnnnn EQT xx,Uyy < or >
                          U
```

where:

    nnnnn = Track number of track containing error.

       xx = EQT of disc.

       yy = Subchannel of disc.

        S = System request encountered error.

        U = User request encountered error.

Parity Errors

Upon detecting a "hard" parity error (i.e., one that is reproducible), RTE aborts the program that encountered the parity error and the following message is issued in addition to those listed above:

```
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 0-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 is issued in addition to those listed above:

```
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

Alternatively, if xx is a mother partition, then yy is a subpartition that contained the parity error. In either case, partition xx and yy are no longer 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.

Note: If CPU switch A1S1 is set to "HALT" instead of "INT/IGNORE", the CPU halts when a parity error occurs and the parity error is not processed by RTE.

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; see below)

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).

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.

Schedule Call Error Messages

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-IVB ignores the interrupt and returns to the point of interruption.

SC04   = program name specified 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.

SC11   =   Attempt to schedule (or time schedule) a program already in the time list for another session.

I/O Call Error Codes

I000   =   Illegal class number. Outside table, not allocated, or bad security code.

I001   =   Not enough parameters were specified.

I002   =   Illegal logical unit number was specified.

I003   =   Illegal EQT referenced by LU in I/O call (Select code=0).

I004   =   Illegal user buffer. Extends beyond RT/BG area or not enough system available memory to buffer the request.

I005   =   Illegal disc track or sector.

I006   =   Reference to a protected track; or using LG tracks before assigning them (see LG, Chapter 3).

I007   =   Driver has rejected call.

I008   =   Disc transfer longer than track boundary.

I009   =   Overflow of LG area.

I010   =   Class GET call issued while one call already outstanding.

I011   =   Type 4 program made an unbuffered I/O request to a driver that did not do its own mapping.

I012   =   Logical unit not defined for this session.

I014   =   An I/O request was issued with the no-suspend option.

I020   =   Read attempted on write-only spool file.

IO21   =   Read attempted past end-of-file (EOF).

IO22   =   Second attempt to read JCL card from batch input file by
            other than FMGR.

IO23   =   Write attempted on read-only spool file.

IO24   =   Write attempted beyond end-of-file (EOF), usually spool
            file overflow.

IO25   =   Attempted to access spool LU that is not currently set
            up.

IO26   =   Attempted to access LU 255.

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.

Logical Unit 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.

LU04   =   Lock attempted on logical unit not defined for this
            session.

I/O Error Message Format

The following error message format is used to report I/O errors:

```
        NR
"IOET L www Exx Syy zzz"
        PE
        TO
```

where: www = the device's logical unit number
            xx = the device's EQT number
            yy = the subchannel for the device
           zzz = device status returned by driver
            NR = device not ready
            ET = end of tape
            PE = transmission parity error
            TO = the device has timed out

                              NOTE

        If a driver is down when an I/O request is made, the device
        status cannot be assumed to be valid.   Therefore, the
        device status information of the error diagnostic is
        replaced with three asterisks,   indicating that the device
        was already down when the request was issued.

Executive Halt Errors

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, privileged
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 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.

HLT 4   Powerfail occured and one of the following is true:

* Powerfail/auto-restart subsystem was not installed.

* CPU was halted when powerfail occurred.

* Powerfail code did not have time to execute completely.

A summary of EXEC call error messages is provided in Table 2-3.

Error Routing

All EXEC call error messages are reported to the "session" terminal as well as the system console. The term "session" is used to mean the terminal from which the program, making the invalid request, was invoked. This means that the routing of errors to the correct console is not dependent on the session monitor.

Before the message is issued to the session terminal, several system level status checks are performed. Each of the following tests must pass or the echo to the session terminal is not performed.

* The session terminal must not be down.

* Enough System Available Memory is available for the error message.

* The session terminal must not have reached it's buffer limit. If the buffer limit has been reached, only one error message will be allowed until:

    1)   any previous system error message is completed or,

    2)   The buffer limits are no longer exceeded.

Equipment error messages will be sent to the session terminal (in addition to the system console) if any of the following conditions are true:

* The error occured on a non-buffered request.

* The error occured on the initiation of a request.

In either case, the ID segment (word 33) of the requesting program provides the information required to issue the error message to the "session" terminal.

If the error (parity error) occured during a DMA transfer, the request is checked to determine the caller's identity. If a user program can be identified, the ID segment once again identifies the destination of the error message.

If the requesting program is under session control, RTE-IVB scans the program's session switch table for every "system" LU to be put down. For every match found ("system" LU=high byte of switch table), the system issues the error diagnostic to the session terminal, but uses the "session" LU as defined in the SST. For example, say "system" LU 17 goes down (needs to be write enabled) and the user has the following SST:

```
                    -----------------
                    |      -4       |   (length word)
                    -----------------
                    |  42   |   1   |
                    -----------------
(system LU's)       |   2   |   2   |   (session LU's)
                    -----------------
                    |  17   |   6   |
                    -----------------
                    |  17   |  10   |
                    -----------------
```

The system console will receive the following diagnostic:

        "IONR L 17 E10 S 0   004"

The session terminal (system LU 42 in this example) will receive the following diagnostics:

        "IONR L*   6 E10 S 0 004"
        "IONR L* 10 E10 S 0 004"

The three octal numbers at the end of the message represent the device status returned by the driver.

The asterisk before the logical unit indicates that the logical unit reported is the session related value.

                              NOTE

    If the above device was buffered, the calling program would still be scheduled and capable of executing. The next access of the down device would cause:

* The program to be suspended.

* The following diagnostic messages to be issued to the session console.

        "IONR L*   6   E 10   S0 ***"
        "IONR L* 10   E 10   S0 ***"

The system console does not receive any additional message.

If another session, represented by the following SST,

```
-------------------
|      -4         |    length word
-------------------
|  43        1  |    switch
-------------------
|   2        2  |
-------------------
|   6        6  |
-------------------
|  17       21  |
-------------------
```

attempts to access the same down device, the session terminal (system LU 43 in this example) will receive the following diagnostic:

"IONR   L*   21   E10   S0   ***"

Table 2-3. EXEC Call Error Summary

ERROR MEANING	READ	WRITE	CONTROL	PROGRAM TRACK ALLOCATE	PROGRAM TRACK RELEASE	PROGRAM COMPLETION	PROGRAM SUSPEND	PROG. SEG. LOAD	PROG. SCHED. W/WAIT	PROG. SCHED. 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						
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	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								
IO12 LU not defined for this session.	X	X	X								

8300-01

Table 2-3. EXEC Call Error Summary (cont.)

PROG. SCHED. TIME 12	I/O SATAUS 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	PROG. SWAPPING CONTROL 22	PROG. SCHED. QUEUE W/WAIT 23	PROG. 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	1 2	2	1 2	1				
									X				
	X				X	X	X	X					

Table 2-3.   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	PROG. SEG. LOAD 8	PROG. SCHED. W/WAIT 9	PROG. SCHED. 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											
LU04 LU not defined for this session											
RQ00 Return buffer below memory protect fence	X	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 No resource numbers in system											
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.						3		1	2	2	
SC06 Resolution not 1, 2, 3, or 4.											
SC07 Prohibited core memory lock attempted											
SC08 Partition is too small. 1. Assigned partition too small for program. 2. Segment too large for partition. (Segment not relocated with main.)								2	1	1	
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	
SC11 Attempted to schedule a program already scheduled from another session.									X	X	X

Table 2-3. EXEC Call Error Summary (cont.)

PROG. SCHED. TIME 12	I/O SATAUS 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	PROG. SWAPPING CONTROL 22	PROG. SCHED. QUEUE W/WAIT 23	PROG. SCHED. QUEUE WO/WAIT 24	RNRQ	LURQ
														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	
													x	
													x	
2		3												
		2								1				
											2	2		
2											2	2		
										x				
													x	
													x	
											x	x		
		x												

Chapter 3
File Management Via FMP

Introduction

File management is performed through program calls to the File Management Package (FMP) library and by interactive operator commands to the program FMGR. The FMP calls mainly control input to and output from disc files or peripheral devices treated as files. The file management capability is increased by using FMGR for interactive program development, disc cartridge manipulation, and batch job control.

This section describes the FMP calls available under RTE-IVB. For more details on FMGR operator commands the reader should refer to the RTE-IVB Terminal User's Reference Manual.

A summary of the services available to the user via FMP Calls is shown below:

*   FILE DEFINITION
*   FILE ACCESS
*   FILE POSITIONING
*   SPECIAL PURPOSE CALLS

Files

Files are a collection of information logically organized into records. They can be stored on disc or they may reference nondisc peripheral devices by name. The information in files may be programs or the data used by the programs. Data may be in binary or ASCII code. Programs may be in the form of ASCII source code, or binary code in either relocatable or absolute form. Programs may also be in memory-image form, a form used by RTE for programs ready to be executed.

Eight file types are defined by the file system. Additional types may be defined by the user. Only the first four types differ in format; all subsequent types differ only in the type of data the file system expects the file to contain. The file types may be divided into three categories as shown in Table 3-1. The first category contains one type, type zero. This type includes all nondisc devices defined as files and accessible by name. The second category contains two file types, types 1 and 2. These fixed-length-record files are used for quick random access. The remaining file types all belong to the third category of files with variable-length records designed for sequential access. All files can be extended. Types 3 and above are automatically extended; types 1 and 2 can be extended by an option used with open call.

Table 3-1. Categories of File Types

CATEGORY	TYPE	DESCRIPTION
Fixed-length, random access	1	Fixed-length 128-word record files
	2	Fixed user-defined-record-length files
Variable-length, sequential access	3	Variable-length records; any data type
	4	Source program file; ASCII
	5	Object program file; relocatable binary
	6	Executable program file; memory-image code
	7	Absolute binary
	8-32767	User-defined data format

FMP file types are summarized below:

TYPE 0 FILES

Type 0 files are used to reference nondisc devices by name. They afford a measure of device independence in that the standard file commands can be used to control the device. A directory entry is made for the device as if it were a file. The File Directory entry for a file of this type contains special entries that specify logical unit number and the operations allowed on the particular device. A type 0 file is created with a FMGR command, not with an FMP call.

TYPE 1 FILES

Type 1 files have fixed length records of 128 words. Because the File Management Package transfers data to and from disc in 128-word blocks, this file type allows direct access between disc and the user's buffer area in his program, thereby eliminating the need to go through an intermediate packing buffer (the Data Control Block). As a result, type 1 files have the fastest transfer rate. Any other file type, except type 0, may be opened and accessed as a type 1 file in order to take advantage of the faster transfer rate. However, if the files being transferred have less than 128-word logical records, the user must be able to recognize where his records begin and end within the 128 words, or if his records are longer, be able to work with part of a record at a time. The end-of-file is defined to be the last word of the last block.

TYPE 2 FILES

The record lengths of type 2 files are fixed. Like type 1 files, the end-of-file is defined to be the the last word of the last block. Only one logical record is transferred at a time, but unlike type 1 files, the transfer must go through a packing buffer (the Data Control Block). For this reason, files of type 2 and above have a slower transfer rate than type 1 files.

TYPE 3 FILES

These files have variable length records and can contain data, source code, relocatable or absolute binary code. Only one logical record is transferred at a time and the transfer must be made through the packing buffer (Data Control Block). The first and last words of each record as written on disc always contain the number of words in the record (minus the two length words). A zero-length record, consisting of two zero words, can be used to separate groups of records into sub-files. The end-of-file is marked by a -1 as the first length word in the next record. Words following the end-of-file are undefined. However, FMP can write records beyond the end-of-file by replacing the end-of-file with a new record followed by an end-of-file mark.

TYPE 4 FILES

This file type is the same as type 3, except the file system expects these files to contain ASCII data. Typically, source program files are type 4.

TYPE 5 FILES

Same as type 3 files, except the file system assumes type 5 files contain relocatable binary code. Typically, object program files are type 5.

TYPE 6 FILES

This type file is the same as a type 3 file, except the system assumes it contains a program in memory-image format that is ready to run. Type 6 files are created by the Save Program (SP) command which copies a program stored on the system track into a type 6 file on the FMP tracks of LU 2 or LU 3. These files are always accessed by the File Management Package as type 1 files. The first two sectors of a type 6 file are used to record ID segment information for the program. As a result, this file type can be used for programs that do not have a permanent ID segment.

TYPE 7 FILES

Same as type 3 files, except the system expects type 7 files to contain absolute binary code.

FILE TYPES GREATER THAN 7

Same as type 3, but the content is user-defined. FMP does no special processing based on file type for types greater than 7. For instance, any checksums must be specifically requested. Content is also user-defined; it may be source, relocatable binary, memory-image format, etc.

File Access

Type 1 and type 2 files contain fixed length records which makes it possible to calculate the position of a desired record. On the other hand, type 3 files and above contain variable length records, so the system must access the disc at least once, and in some cases several times, in order to position to the desired record location. For this reason, access takes longer for file types greater than type 2. Type 2 files should not be opened using update mode and automatic extendability at the same time. An FMP -12 error will occur when attempting to write to an extent that does not already exist.

All Files can be automatically extended whenever a write request points to a location beyond the range of the currently defined file. The extent is created by FMP with the same name and size as the main file, and access continues. FMP numbers each extent starting with 1. The extent number and location is kept in the file directory entry for the extent. When a file with extents is referenced by its file name, any extents are provided automatically. A file can have up to 255 extents.

For type 3 and above files, all extents are sequential. That is, the main is created first followed by extent 1,2,3 and so on. The end of the file is signified by an EOF mark. At close, the extents may be truncated to provide more disc space or they may be retained.

For type 1 and 2 files, extents may be created sequentially or randomly. That is, the main is created first and sequential writes to the file may cause sequential extents to be created. However, a random write to a record that would fall into an extent will cause that extent to be created whether or not the extent immediately preceding it exists.

For example, assume a type 1 file with a main file size of 10 blocks is created. Writing to record 11 of this file will cause extent 1 to be created. Now the main and only extent 1 exist. A write to record 31 will cause extent 3 to be created, not extent 2. Now the file consists only of the main, extent 1, and extent 3.

External type 1 and type 2 files are treated as logically contiguous. As in the example above, writing 1408 words starting at record 31 causes 10 records (1280 words) to be written in extent 3, and 1 record in extent 4. Also, it is possible to read a record from an extent that does not exist as long as that record is not beyond the last word of the last extent in the file. No disc access is performed; the user's buffer will be zero-filled. For example, a read to record 22 of the file created above will return with no error, the return length will be 128 words, the user's buffer will be zero-filled, and the next record is set to 23.

The end of file for type 1 and type 2 files is defined as the last word of the last extent in the file.

When a type 1 or type 2 file is copied using the FMGR ST, DU, or CO commands, the resultant file may occupy more disc space than the original file occupied. This is because records from non-existant extents are physically read into the destination file. This feature is especially important when compacting the extents of the file with the ST, DU or CO commands. The resultant file must be physically contiguous as well as logically contiguous to guarantee the same displacement.

Files may be opened for access in either update or non-update mode. Update mode is used to access existing files that are to be modified in a random manner. Non-update mode is used to access files in a sequential manner or to enter the original data into a file.

In update mode, the entire block containing the record is automatically read into the Data Control Block before the record is modified. After modification, the entire block is written to the disc. This is done to insure that the Data Control Block always contains the unmodified as well as the modified data, thereby guaranteeing restoration of the block to the disc.

Reading or positioning a file is not affected by the update or non-update modes of access.

Cartridges

Files managed by the File Management Package, whether they are program files, data files, or spool files, are kept on FMP disc cartridges. An FMP disc cartridge is a logical entity that may correspond directly to a disc platter or may be a subdivision of the disc platter. On some discs, a cartridge may cross platter boundaries.

Each cartridge is defined as a contiguous block of tracks, and is assigned a logical unit number. A cartridge reference number or the LU may be used to reference the cartridge. Files on the same cartridge must have unique names. Duplicate names may be used as long as the duplicates are on separate cartridges so the file can be uniquely identified by its name and a cartridge identifier.

Non-extended files are located in blocks of contiguous sectors on an FMP disc cartridge (file extents are non-contiguous). User files begin in the lowest numbered track and work up. Directory entries for user files begin in the highest numbered track and work down. Removable cartridges containing FMP files are interchangeable between drives of the same type within a system, or between drives on different systems provided that logical track 0 refers to the same physical track on every disc unit. (Refer to figure 3-1 for an illustration of disc organization using one cartridge on the system disc starting at the first FMP track, and one on a peripheral disc starting at track 0.)

At cartridge initialization, the number of directory tracks for that cartridge is specified. The first cartridge track must be assigned at initialization; the number of sectors per track may be specified at this time, but is supplied by FMP as a default if not.

Files may cross track boundaries, but may not cross cartridge boundaries. Files are subject to being moved whenever a cartridge is packed. This causes files to be relocatable within a cartridge and no absolute file addresses should be kept in any file or program.

Files always start on even sector boundaries and all accesses are multiples of 128 words addressed to even sectors.

Disc errors are passed back to the user for action. Error codes are printed on the system log device when using the FMGR operator commands, or passed to the user program when calling a File Management Package library routine. You may report bad tracks to the system through the FMGR Initialization command. Bad tracks discovered by the system result in an error return to the calling program.

Cartridge and File Directories

Two directory types are maintained by the file system; the FMP cartridge directory on the system disc, and the file directory on each cartridge. Program D.RTR updates and maintains the directories.

The cartridge directory is a master index to all mounted FMP cartridges. It is maintained in the system area of LU 2 (refer to figure 3-1). Its length is two blocks and it has an entry for all currently mounted cartridges. The directory has room to describe up to 63 cartridges using four words for each.
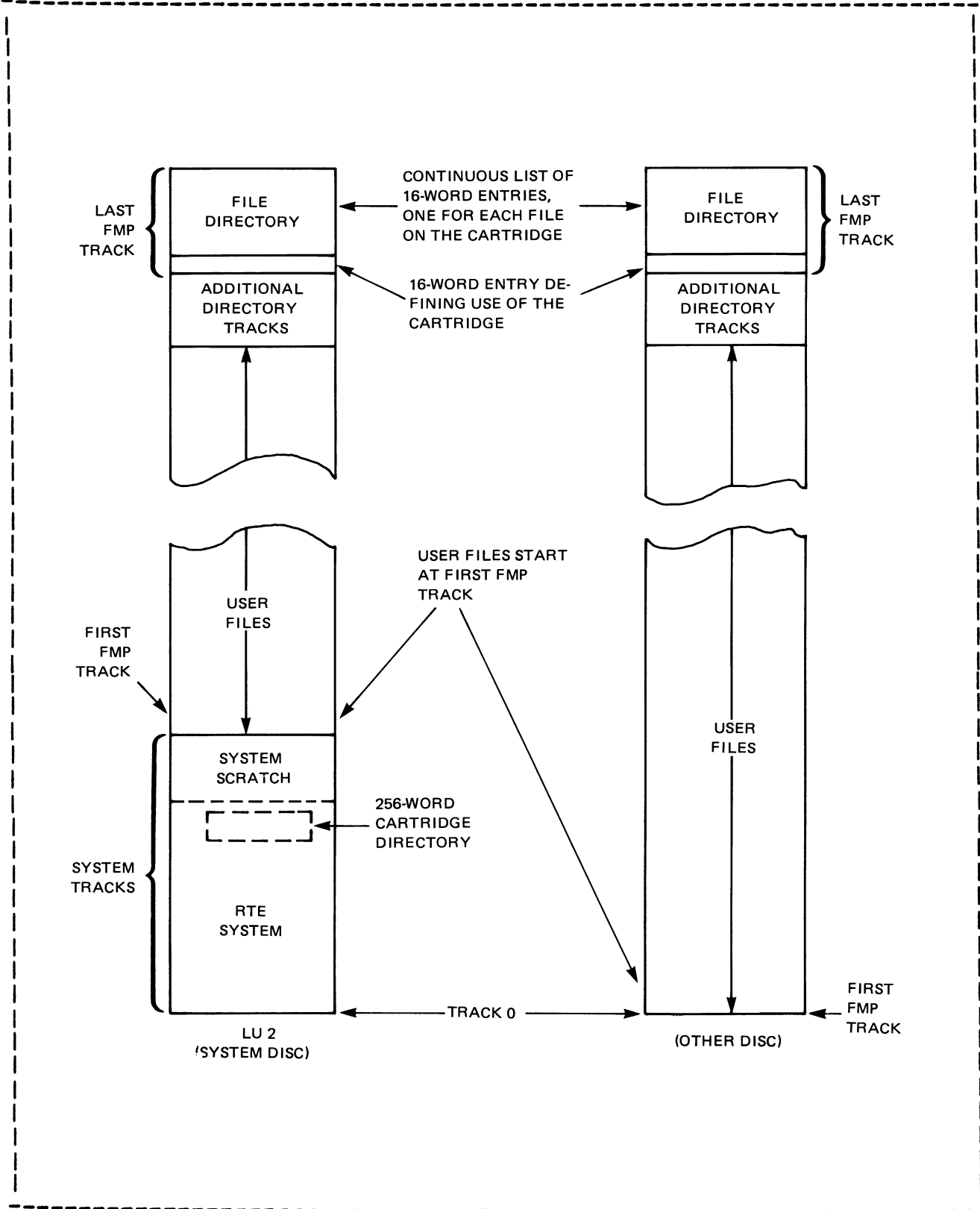
Figure 3-1. Disc Cartridge Organization

A file directory, maintained on each cartridge, contains information on each file on that particular cartridge. Each directory starts in sector 0 of the last track available to FMP. The first 16-word entry in this directory contains label and track information for the cartridge itself. Each subsequent 16-word entry has information on a user file. The last entry is followed by a zero word. When a file is purged, the first word in the directory entry for the file is set to -1 to indicate that the directory entry is to be ignored. When the cartridge is packed, the directory entry for any purged file is cleared and the cartridge area where the file was located is overwritten by non-purged files wherever possible.

The formats of the cartridge and File Directory entries are shown in Appendix H.

File Security

In addition to the security features provided through the Session Monitor (see below), the File Management Package provides two additional security levels: file system security and individual file security.

During FMP initialization, a master security code is assigned to the file system. If the code is zero, no security is provided. If non-zero, the master code must be known in order to get directory listings that include the specific file security codes and also in order to re-initialize an FMP cartridge.

Each file also has a security code. This code may be zero, positive, or negative. A zero code allows the file to be opened to any caller (who has access to the disc cartridge containing that file) with no restrictions; in effect this code provides zero security. A positive code restricts writing on the file but not reading; that is, a user who does not know the code may open the file for read only, but may not write on the file. A negative code restricts all access to the file; this code must be specified in order to open a file protected by it. An attempt to open a file so protected without the correct security code results in an error message.

Cartridges in the Session Environment

In the session environment, cartridge security is provided by identifying them as to whom they are mounted.

When a disc cartridge is mounted to the system, an ID is put in the cartridge list entry which identifies the category of the cartridge and to whom the cartridge was mounted (refer to Appendix H).

There are four categories of cartridges in the session environment as follows:

1. System or global cartridges.

2.   Private cartridges.

3.   Group cartridges.

4.   Non-session cartridges.

System cartridges are those cartridges that only the system manager can mount or dismount. The cartridge where the system tracks reside must be defined during system generation as LU 2. An auxiliary system cartridge, defined as LU 3 at system generation, can also be defined.

LU 2 and LU 3 are read-only cartridges for session users. Only the system manager and non-session users can write on these cartridges. LU 2 contains:

1.   Operating System

2.   System Library

3.   System Scratch Tracks

4.   Cartridge Directory

5.   FMP Tracks

LU 3 can be defined if more room is required for type 6 files than what is available on LU 2 or if additional system tracks are required.

In addition to LU 2 and LU 3, system global cartridges can be defined which provide both read and write access to all users on the system. Global cartridges are cartridges other than LU 2 or 3 that are mounted by the system manager.

Private cartridges are those that are accessible only to the user who mounts them to his session and the system manager (who has access to all cartridges in the system).

Group cartridges are those that are accessible only to members of a group who have it mounted to their sessions and the system manager.

Non-session cartridges are only accessible to users operating in a non-session environment or through the system console. Non-session cartridges are those mounted to a user not operating under session control. The system manager also has access to non-session cartridges.

When the session system is initialized, the system manager can create a spare cartridge pool from which session users can mount additional private or group cartridges to their session. The cartridges "taken" from the pool are typically used for temporary storage and are released back to the pool by the user when they are no longer needed.

The session monitor provides session users with protection for their mounted cartridges by restricting cartridge access to only those cartridges mounted in the user's Session Control Block (SCB). Whenever a cartridge is specified in a call, FMP checks that the cartridge is mounted not only to the system, but is also mounted to the user's SCB (see Appendix J, "SCB FORMAT").

There are some exceptions to the above check:

1. The system manager is allowed access to all cartridges mounted to the system by using the SL command described in the RTE-IVB Terminal User's Reference Manual.

2. Some internal subsystems need to have, and are given, access to all cartridges mounted to the system.

3. Procedure files can be set up by the system manager on LU 2 or LU 3 and may be run by any session user. The commands in these procedure files are not subject to capability checking or cartridge access restrictions.

4. User message files may reside on LU 2 and LU 3. Session users will be able to open, read, and write into these files via the ME and SM commands described in the RTE-IVB Terminal User's Reference Manual.

5. Read only access will be allowed on LU 2 and LU 3, and read/write access will be allowed on system global cartridges when operating under session.

When accessing a file, if a particular cartridge is not specified, the user's private discs are searched in the order that they are mounted in the system cartridge directory. Then the user's group cartridges are searched in the order that they appear in the system cartridge directory. Finally, system cartridges are searched in the order that they appear in the directory.

To summarize, session users have access to their private and group cartridges, system global cartridges, and have restricted access to system LU 2 and 3.

Non-session users have access to non-session cartridges and unrestricted access to all system cartridges. Non-session users do not have access to cartridges mounted to session users.

The system manager has access to all cartridges.

Note that cartridges are mounted and dismounted using the MC, AC, and DC commands described in the RTE-IVB Terminal User's Reference Manual.

FMP Calls

The FMP program calls provide an interface between programs and the File Management utility routines. With these calls, the user can open, close, read from, and write to files. In addition, the calls can be used to create or purge disc files, position either disc or non-disc files and directly control non-disc files.

Table 3-2 lists all the FMP calls according to general function and indicates the status, before and after the call, of the Data Control Block. It also indicates when and if the file directory is accessed by the call.

The Data Control Block

The Data Control Block is a block of words defined within your program that acts as an interface between the program and the File Management Package. You must supply one Data Control Block for each open file. It is an array which contains control information for the file including the file name, type, size, and location on disc if the file is a disc file. In addition, it acts as a buffer for the physical transfer of data between a file and your program. The Data Control Block is used to:

* Avoid directory access for file information

* Keep track of current record position in file

* Provide a buffer for transfer of data between a file and the program.

Once a file is open, the Data Control Block is referenced for file information and the file name is no longer needed or used in FMP calls.

Each Data Control Block is an array with a minimum of 144 words. The first 16 words are a control block to provide all the file information required by the FMP calls. The remaining words are a packing buffer used for the transfer of data in blocks of 128 words. The 16-word control area is maintained and used only by FMP and must not be modified directly. Refer to Appendix H for the format of this area.

A packing buffer of 128 words is the minimum that can be specified. The buffer may be larger, as large as available memory, but any file can be accessed with the minimum 128-word buffer regardless of the buffer size specified at creation.

Table 3-2. FMP Call Summary

CATEGORY	ROUTINE	FUNCTION	DCB STATUS		DIRECTORY ACCESS
			AT ENTRY	AT RETURN	
File Definition	CREAT ECREA	Enter file in directory; open exclusively for update.	CBO	OPNX	YES
	CRETS	Enter scratch file in directory; open exclusively for update.	CBO	OPNX	YES
	PURGE	Close file and remove from directory.	CBO	CLOS	YES
	OPEN OPENF	Open file.	CBO	OPN	YES
	CLOSE ECLOS	Close file.	MBO	CLOS	YES
File Access	READF EREAD	Transfer record from file to user buffer.	MBO	OPN	EXTENTS
	WRITF EWRIT	Transfer record from user buffer to file.	MBO	OPN	EXTENTS
File Position	LOCF ELOCF	Retrieve current position and status of open file.	MBO	OPN	NO
	APOSN EAPOS	Position disc file to a particular record.	MBO	OPN	EXTENTS
	POSNT EPOSN	Position disc or non-disc file relative to current record.	MBO	OPN	EXTENTS
	RWNDF	Position file to first record.	MBO	OPN	EXTENTS
Special Purpose Routines	FCONT	Specify control functions for non-disc file.	MBO	OPN	NO
	FSTAT	Retrieve contents of cartridge directory.	—	—	—
	IDCBS	Retrieve actual size of DCB buffer used by FMP.	MBO	OPN	NO
	NAMF	Rename existing file.	CBO	CLOS	NO
	POST	Write DCB buffer to disc.	MBO	OPN	NO

Legend:	
CBO	Can be open: DCB can be assigned to open file; that file will be closed and, in case of CREAT and OPEN, file specified in call will be opened.
MBO	Must be open: DCB must be assigned to open file.
OPN	Open: File assigned to DCB is opened or is left open.
OPNX	Open: New file is assigned to DCB and is opened exclusively for update.
CLOS	Closed: File assigned to DCB is closed; DCB is available for other use.
EXTENTS	Directory is accessed only if call changed extents.

Data Transfer

In addition to the Data Control Block, another buffer must be defined
in your program for transferring individual records. This buffer, the
user buffer, is where a record to be written is specified and into
which a record is read. The relation between the user buffer, the
Data Control Block, and a disc file is illustrated in Figure 3-2.
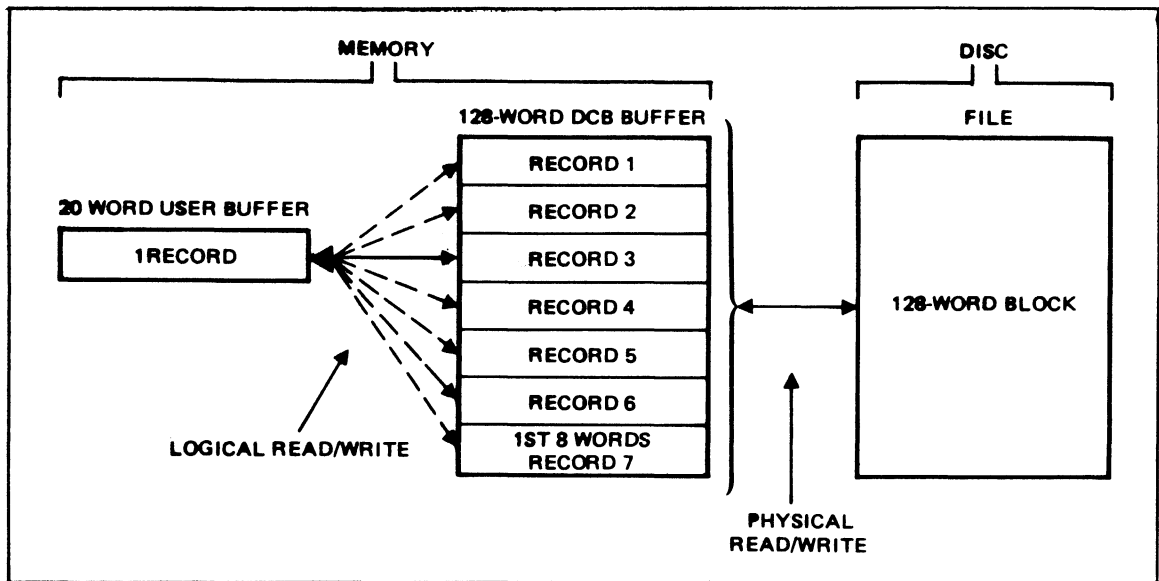
Figure 3-2. Sequential Transfer Between Disc File and Buffers

Each call to read or write a record transfers one record between the
user buffer and the Data Control Block packing buffer. This type of
transfer within memory is known as a logical read or write.

A physical read or write transfers a 128-word block between the disc file and the Data Control Block packing buffer. A physical write is performed automatically when the packing buffer is full, when the file is closed, or when a specific request is made with the POST call. On a read request, a block of data is physically read into the packing buffer from the disc only if the requested record is not already in that buffer. Any time a record being read or written is not wholly contained in the packing buffer (refer to record 7 in Figure 3-2), then the File Management Package reads or writes blocks until the entire record has been transferred.

When type 2 files are accessed randomly, the process is essentially the same as the sequential access illustrated in Figure 3-2 except that physical transfers may be more frequent since successive references are less likely to be to records in the same block in the packing buffer.

Since each record in a type 1 file is 128 words, the intermediate transferred to the Data Control Block packing buffer is omitted and each record is transferred directly between the user buffer and the file as illustrated in Figure 3-3. This type of access is the most efficient. A full 144 word Data Control Block must still be specified in the user program.
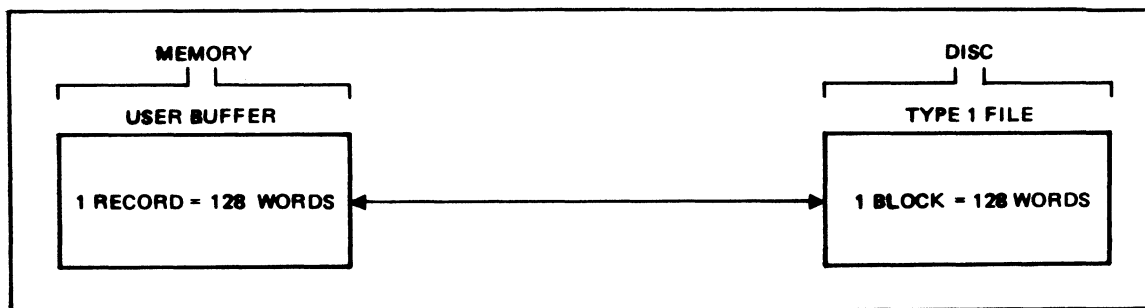
Figure 3-3.   Data Transfer With Type 1 Files

Type 0 (non-disc) files also bypass the Data Control Block during transfers. Records in these files are written or read directly to or from the device identified as a type 0 file. A specific number of words, rather than a record, is the unit transferred by a read or write request to this file type. The transfer can thus be tailored to the particular record length of the device.

FMP Call Formats

When a calling sequence is encountered during execution of a user program, the File Management Package executes the call according to the value of the parameters in the calling sequence, and/or returns information to areas defined in the parameter list. The number, type, and meaning of the parameters depends on the subroutine and are described in detail in this section.

For FORTRAN, tne general form of the calling sequence is:

    CALL name(pl,p2, . . . ,pn)

where:

    name is the subroutine name.

    pl through pn are the parameters; parameters name a real or an
    integer array or variable or, if the parameter is one-word, a
    value. Position determines parameter meaning.

For Assembly Language, the general form of the calling sequence is:

```
        EXT    name
        .
        .
        .
        JSB    name
        DEF    RTN (or * +(n+1) where n is tne number of parameters)
        DEF    pl
        DEF    p2
        .
        .
        .
        DEF    pn
  RTN    (return location)
```

where:

    name is the subroutine name; it must always be defined as an
    external with an EXT statement before it is called.

    RTN is the label of the location to which the subroutine returns
    upon completion; it must always follow the last parameter in the
    calling sequence.

    pl through pn are the parameters; each parameter names an integer
    array or variable.

Common Parameters

Parameters used frequently in FMP calls are described here and under Optional Parameters.

IDCB

This parameter specifies the array used as a Data Control Block. It must be at least 144 words, 16 words for file control information and 128 for the minimum packing buffer. For faster processing, a larger buffer can be specified. In general, the larger the usable buffer, the faster the transfer rate. For example, a usable buffer of 256 words (IDCB=272 words) will nearly double the transfer rate for sequential accesses.

While a file may be created with a large DCB buffer, it may be accessed with any packing buffer that is at least 128 words long (IDCB=144 words). All transfers of data use the full actual buffer size. This size is always 128, or a multiple of 128, words. The actual size used is determined from the size requested for the Data Control Block in conjunction with the file size in the following manner:

* If the requested buffer size is greater than or equal to the file size, FMP allocates an actual Data Control Block equal to the file size +16 control words.

* If the requested buffer size is less than the file size, FMP determines the actual Data Control Block buffer size according to the following rules:

    - It must be a multiple of 128 words.

    - It must be less than or equal to the size specified for IDCB by the user.

    - It can be evenly and exactly divided into the total file size.

To illustrate, Table 3-3 shows the relation between file size, requested packing buffer size and the actual size assigned by FMP for two file sizes, one with factors and the other a prime number. File size is given in 128-word blocks for convenience.

Table 3-3. Relation of Actual to Requested Packing Buffer Size

FILE SIZE IN BLOCKS	REQUESTED DCB BUFFER* BLOCKS	WORDS	ACTUAL DCB BUFFER BLOCKS	WORDS
10	1	128	1	128
	2	256	2	256
	3	384	"	"
	4	512	"	"
	5	640	5	640
	6	768	"	"
	7	896	"	"
	8	1024	"	"
	9	1152	"	"
	10	1280	10	1280
13	1	128	1	128
	2	256	1	128
	.	.	"	"
	.	.	"	"
	.	.	"	"
	12	1536	1	128
	13	1664	13	1664

*16 words must be added to the buffer size when dimensioning the Data Control Block array (IDCB).

A call to routine IDCBS may be made to determine the actual size of the DCB (actual buffer + 16 control words).

IERR

When an error occurs during a subroutine call, a negative error code is returned in this parameter and usually in the A-register. (A list of the FMP error codes and their meaning is included at the end of this section). For successful OPEN or OPENF calls, the file type is returned as a positive integer in IERR; for successful CREAT calls the number of disc sectors used is returned in IERR as a positive integer. When using the CLOSE or ECLOSE calls always check the IERR parameter. In calls where IERR is the last parameter, it is optional. It should be omitted only if errors are checked in the A-register. Negative error codes allow for easy error checking and error checking should not be omitted as a general practice.

INAM

For calls requiring a file name, the name is specified in this 3-word array. File names are six ASCII characters and must conform to the following rules:

*   only printable characters may be used

*   first character must not be blank or a number

*   if less than six characters, must be padded with trailing blanks

*   embedded blanks are not allowed

*   plus (+) minus (-) colon (:) or comma (,) are not allowed

Duplicate file names are not allowed on the same cartridge.

IBUF

This array is the user buffer and is included in the calls that transfer records. It should be as long as the longest record to be transferred. The record to be written must be in this array; when a record is read it is placed in this array.

Optional Parameters

Most subroutines have one or more optional parameters. They always appear at the end of the calling sequence and may be omitted only from the end. If an optional parameter is needed that is preceded by other unnecessary optional parameters, all the parameters up to the desired parameter must be included; unused optional parameters should be set to zero.

ISC

The file security code is specified in ISC. It can be a positive or negative integer or two ASCII characters representing a positive integer. When characters are used, FMP converts them to their integer equivalent. If omitted, ISC is set to zero.

ISC considerations are summarized below:

CREATION	FILE PROTECTION	FILE REFERENCE
ISC = 0	unprotected	ISC = any value for any access
ISC = + n	write protected	ISC = +n or −n to write or purge ISC = any value to open or read
ISC = − n	read/write protected	ISC = −n for any access

ICR

This parameter is specified to restrict the file search to a particular cartridge or logical unit number. The search is for space if the call is CREAT or ECREA, for a file name in other calls. ICR may be a positive or negative integer. If omitted, it is set to zero.

If ICR = 0   The file search is not restricted to a particular cartridge. A CREAT or ECREA call locates the file on the first cartridge with enough room; other calls using ICR search the cartridges in the following order:

       1) Private cartridges as they appear in the cartridge list.

       2) Group cartridges as they appear in the cartridge list.

       3) System cartridges as they appear in the cartridge list.

     >0   File search is restricted to the cartridge identified by the cartridge reference number (ICR), an identifier assigned to all cartridges in the system. A list of cartridge reference numbers can be obtained with the FSTAT call or the :CL FMGR command.

     <0   File search is restricted to the cartridge associated with the logical unit number (-ICR). To illustrate, if ICR is -14, the file search is restricted to logical unit 14.

If ICR specifies a cartridge not mounted to the user's SCB, an error
is returned.

IDCBZ

When a Data Control Block larger than 144 words is specified in
parameter IDCB, then parameter IDCBZ must also be specified. It
informs FMP of the number of words available in the DCB packing
buffer for data transfer. Any positive number can be specified; the
actual usable buffer is always determined by FMP as described in the
discussion of the IDCB parameter. This size is never larger than the
size specified in IDCBZ, but it may be smaller. For example, if IDCBZ
is less than 256 (any value between 1 and 255), then 128 words are
used for the DCB packing buffer (144 for the entire Data Control
Block).

Normally, you will specify IDCBZ as 16 words less than the array size
specified for IDCB.

FMP Call Description Conventions

In the subsections that follow, certain conventions are used to
describe FMP calls. These conventions are summarized below:

* Parameters that are underlined, such as

    CALL OPEN(IP1,IP2,IP3)

    have values returned by the system, e.g., the value is not
    supplied by the user.

* Parameters that are double underlined, such as

    CALL OPEN(IP1,IP2,IP3)

    have values that are supplied by the system in some cases and
    user-supplied in other cases. The comments associated with the
    call description should be consulted for details concerning their
    use.

* Parameters enclosed in square brackets, such as

    CALL OPEN(IP1,IP2[,IP3])

    are optional.

* Parameters enclosed in angle brackets, such as

      CALL OPEN(IP1,IP2<,IP3>)

  are optional in some cases and required in others. The comments associated with the call description should be consulted for details concerning their use.

* Parameters with no qualifiers, i.e., square brackets, angle brackets, or underlines, are required and their value is supplied by the user.

* All FMP call descriptions use the FORTRAN-IV subroutine call format. If desired, the description of FMP call general formats included at the beginning of this section can be consulted to convert the calls to Assembly Language format.

File Definition FMP Calls

A file is defined in terms of its name, size, type, and where it is located. In addition, a security code can be assigned to a file; restricting access by unauthorized programs. A file is defined when it is created using the CREAT or ECREA call. When a file is created, an entry is made for the file in the file directory (see Appendix H, "FILE DIRECTORY FORMAT").

Once defined, the file may be opened for access by any program using the proper security code in an OPEN or OPENF call. When a file is opened, necessary information (location, size, etc.) is transferred from the file directory to the control words of the Data Control Block (DCB) associated with the file, thereby making a logical connection between the file name in the directory and the DCB.

When access to a file is no longer necessary, the logical connection between the file directory and the DCB is removed by closing the file using a CLOSE or ECLOS call. Closing a file frees the DCB for other uses, but the file remains defined in the file directory.

The PURGE call is used to flag the directory entry associated with a file so that the file is no longer defined. Note that the PURGE call closes the file before performing the purge operation.

Scratch files provide temporary disc space. Such files are created using the CRETS call. A number is passed by the calling program to CRETS. When the scratch file is created, it is opened exclusively to the calling program and a unique file name is returned. To eliminate the scratch file, call PURGE. To save a scratch file, either copy it to another file or rename the scratch file with NAMF.

The File Definition FMP calls are listed below in the order of their presentation:

* CREAT,ECREA

* CRETS

* OPEN,OPENF

* CLOSE,ECLOS

* PURGE

CREAT and ECREA Calls

A call to CREAT or ECREA creates a disc file by making an entry in the File Directory and allocating disc space for the file. The ECREA call performs the same function as the CREAT call except that larger files can be created. CREAT can define files up to 16383 blocks in size, ECREA can define files up to 32767x128 blocks in size.

Following the creation call, the file is left open in update mode for exclusive use of the calling program. To open a file in a different mode, or for non-exclusive use, a call must be made the OPEN or OPENF routines described later in this section.

Note that the CREAT and ECREA calls can only create disc files. To create non-disc files (type 0), the FMGR CR command must be used (see the RTE-IVB Terminal User's Reference Manual).

```
+------------------------------------------------------------------+
|                                                                  |
|   CALL CREAT (IDCB,IERR,INAM,ISIZE,ITYPE[,ISC][,ICR][,IDCBZ])     |
|      or                                                           |
|   CALL ECREA (IDCB,IERR,INAM,ISIZE,ITYPE[,ISC][,ICR][,IDCBZ]      |
|            [,JSIZE])                                              |
|            -----                                                 |
|                                                                  |
+------------------------------------------------------------------+
|   IDCB  --  Data Control Block. An array of 144 +n words where   |
|             n is positive or zero.                               |
|                                                                  |
|   IERR  --  Error return.  A one-word  variable  in which a      |
|             negative error code  is returned.  If no error       |
|             occurs, IERR is set to the  number of 64-word         |
|             sectors in the created file (for CREAT); for ECREA    |
|             calls, IERR is indeterminant.                        |
|                                                                  |
|   INAM  --  File name. 3-word array containing ASCII file name.  |
|                                                                  |
|   ISIZE --  File size. For CREAT call, a 2-word array with the   |
|             number of  blocks  in  the  first  word  (must be    |
|             <=16,383);  record length (in words) in the second   |
|             word (used only for type 2 files).                   |
|                                                                  |
|             For  ECREA  call,  a 2-entry array with each entry   |
|             being a double word integer. The first entry speci-  |
|             fies  the file size in blocks  (up to  32767 x 128   |
|             blocks).   The second  entry specifies  the record   |
|             length in words (used for type 2 files only).        |
|                                                                  |
+ - - - - - C O N T I N U E D   N E X T   P A G E - - - - - - - +
```

```
+ - - C O N T I N U E D   F R O M   P R E V I O U S   P A G E - -+
|                                                                 |
|   ITYPE  --   File type. A one-word variable between 1 and 32767; |
|               types  1 - 7  are  FMP-defined, the rest are user- |
|               defined special purpose files.                     |
|                                                                 |
|   ISC    --   Security  code.  An  optional  one-word  variable  |
|               between 0 and +/-32767. If 0 (or omitted),the file |
|               is not protected.  If positive, the file is write- |
|               protected only.   If negative,  the file is write- |
|               and read-protected.                                |
|                                                                 |
|   ICR    --   Cartridge reference. An optional one-word variable. |
|               If  positive,  specifies  a  cartridge  reference  |
|               number.  If negative,  specifies  a  logical  unit |
|               number.  If 0 or -32768 (or omitted),  first avail- |
|               able cartridge  with enough room will be used.  If |
|               in  session environment,  the ICR  must  specify a |
|               cartridge mounted to the user's session.           |
|                                                                 |
|   IDCBZ  --   Packing buffer size. An optional one-word variable |
|               set  to  the number of words in the packing buffer |
|               if more  than 128 are  requested.   If omitted, FMP. |
|               assumes  DCB size (control words + packing buffer) |
|               is 144 words regardless of the IDCB dimension.     |
|                                                                 |
|   JSIZE  --   Created file size.  Used with ECREA call; optional |
|               double-word  parameter  in  which  the actual file |
|               size created (in sectors) is returned if the call  |
|               was successful.                                    |
|                                                                 |
+-----------------------------------------------------------------+
```

COMMENTS:

FILE SIZE---Records are addressed by number; therefore, the number of records contained in a file must not exceed the maximum accessible using the file access calls described later in this section.

If a file is to be accessed with the standard access calls (READF, WRITF, etc), the number of records contained in a file must not exceed $(2**15)-1$. If a file is to be accessed with the extended access calls (EREAD, EWRIT, etc), the number of records must not exceed $(2**31)-1$.

The number of records contained in a file can be determined as follows:

$$\text{number of records} = \frac{\text{words in file (blocks in ISIZE x 128)}}{\text{record length in words}}$$

When the exact size is not known, the rest of the cartridge may be allocated by setting ISIZE to -1 (valid only for files of type 3 or greater). The unused area may be returned to the system by truncating the file when it is closed (refer to the CLOSE and ECLOS calls described later). For CREAT call, the rest of the cartridge (but not more than 16383 blocks) will be allocated. For ECREA the rest of the cartridge (but not more than 32767 x 128 blocks) will be allocated.

The ECREA call allocates files in block increments. After 16383 blocks have been allocated, then ECREA begins allocating files in chunk increments (1 chunk = 128 blocks). If a file size (ISIZE) is specified that is not evenly divisible by 128, a full 128-block chunk is allocated to accommodate the remainder.

FILE TYPES -- FMP defined file types are as follows:

    type 1 - Fixed length, 128-word records; random access.
    type 2 - Fixed length, user-defined record lengths; random access.
    type 3 (and greater). Variable-length records; sequential access;
                          automatic extents provided.
    type 4 - source program
    type 5 - relocatable program
    type 6 - memory-image program
    type 7 - absolute binary program

File types greater than 7 are user-defined but are treated by FMP as type 3. Any special processing based on file type is not provided as a default; it must be specified.

When any file of type 3 or greater is created, FMP writes an EOF mark at the beginning of the file. As records are written to the file, the EOF is moved automatically to follow the last record.

When a file of type 1 or 2 is created, extents are not automatically created unless the file is opened via the OPEN FMP call with the EX bit (bit 5) set.

Further details on file types are given at the beginning of this section.

EXAMPLE 1 -- Create a type 2 file called FIX with 100 blocks,62 words per record, security code AB, and a DCB packing buffer of 128 words:

```
    DIMENSION NAM1(3),ISIZ(2),IDCB1(144)  <---- 16 control words +
                                                 128-word buffer
    DATA NAM1/2HFI,2HX ,2H  /
    ISIZ=100 <--------------------------------- number of blocks
    ISIZ(2)=62 <------------------------------- record size
    :                                    v---------file type
    CALL CREAT(IDCB1,IERR,NAM1,ISIZ,2,2HAB)
    IF (IERR .LT. 0) GO TO 900          ^-----security code
    CONTINUE                     ^
    :                            +----------------process any errors at 900
```

EXAMPLE 2 -- Create a type 3 file called PROG1, use rest of cartridge, no security code, using a 256 word DCB packing buffer, and locate it on logical unit 14.

```
     DIMENSION NAM2(3),ISIZE(2),IDCB2(272) <----  16-control words
                                                  + 256-word buffer
     DATA NAM2/2HPR,2HOG,2H1 /
     ISIZE=-1 <-------- file uses rest of cartridge, record length
                        unspecified
     ICR=-14 <---------- file located on logical unit 14
     ITYPE=3 <---------- file type is 3
     IDCBZ=256 <-------- DCB buffer size
     .
     .
     .
     CALL CREAT(IDCB2,IERR,NAM2,ISIZE,ITYPE,0,ICR,IDCBZ)
     IF (IERR .LT. 0) GO TO 900  <---------------- error check
     CONTINUE
     .
     .
     .
```

Another method is to use literals for all one-word variables:

```
     DIMENSION NAME2(3),ISIZE(2),IDCB2(272)
     DATA NAME2/2HPR,2HOG,2H  /
     .
     .
     CALL CREAT(IDCB2,IERR,NAM2,-1,3,0,-14,256)
     IF (IERR .LT. 0) GO TO 900
     CONTINUE
     .
     .
```

Care should be taken when literals are used as parameters since this practice can result in problems if the values are changed by the call routines.

EXAMPLE 3 -- Create a type 2 file (PROGD) that is 76,864 blocks in size (76,864/128 = 600.5 chunks; 601 are allocated). Each record is 64 words long; DCB packing buffer is 128 words.

```
    DIMENSION NAM3(3),ISIZE(4),IDCB(144)
    DATA NAM3/2HPR,2HOG,2HD /
    ISIZE(1) = 1
    ISIZE(2) = 11328
    ISIZE(3) = 0
    ISIZE(4) = 64
    ITYPE = 2
    .
    .
    CALL ECREA(IDCB,IERR,NAM3,ISIZE,ITYPE,0,0,JSIZE)
    IF (IERR .LT. 0) GO TO 900
    CONTINUE
    .
    .
```

The double-integer parameter in the above example is manipulated "by-hand". Note that 76864 - 65536 = 11328. Since $65536 = 2**16$, bit 16 of the double-word integer must be set. This is equivalent to bit 0 of ISIZE(1). ISIZE(2) is then set to 11328 with the net result being a double-integer representation of 76864 in ISIZE(1) and ISIZE(2).

CRETS Call

A call to CRETS creates a temporary or scratch disc file by making an entry in the File Directory and allocating disc space for the file. CRETS can define files up to 32767 x 128 blocks in size.

Following the CRETS call, the file is left open in update mode for exclusive use of the calling program. The file is also given a unique file name.

Upon terminating access to the scratch file, you must use the PURGE call to delete the file. If the user program aborts after file creation and prior to the call to PURGE, the file must be purged using FMGR. If the file name consists of six numeric ASCII characters the FMGR purge command will not succeed unless an alphabetic character is appended to the end of the file name, e.g., a file named 077033 is purged as :PU,077033X. To save the file, either copy it to another file or change its name using NAMF.

```
+-------------------------------------------------------------------+
|   CALL CRETS (IDCB,IERR,NUM,INAM[,ISIZE][,ITYPE][,ISC][,ICR]      |
|                              ----                                  |
|           [,IDCBZ][,JSIZE])                                       |
|                    -----                                          |
|-------------------------------------------------------------------|
|   IDCB    -- Data Control Block.  An array of 144+n words where n |
|              is positive or zero.                                 |
|                                                                   |
|   IERR    -- Error return.   A  one-word  variable  in  which  a  |
|              negative error code is returned.                     |
|                                                                   |
|   NUM     -- Scratch file number.  A one-word integer  between 0  |
|              and 99.                                              |
|                                                                   |
|   INAM    -- File name created 3-word array containing ASCII file |
|              name.                                                |
|                                                                   |
|   ISIZE  -- File size.  A  2-entry array (or 4-word array) with   |
|              each entry being a double-word integer.  The first   |
|              entry specifies the file size in blocks  (up  to     |
|              32767 x 128 blocks).  The second entry specifies the |
|              record length in words (used for type 2 files only). |
|              Default is 24 blocks.                                |
|                                                                   |
|   ITYPE  -- File type.  A one-word variable between 1 and 32767;  |
|              types 1-7 are FMP-defined, the rest are user-defined |
|              special purpose files.  Default is type 3.           |
+----------------CONTINUED NEXT PAGE--------------------------------+
```

```
+-----------------CONTINUED FROM PREVIOUS PAGE----------------+
|                                                             |
|   ISC    -- Security code. An optional one-word variable between  |
|             0 and +/-32767.  If 0 (or omitted),  the file is not  |
|             protected.  If positive, the file is write-protected  |
|             only.  If negative,  the file is  write-  and  read-  |
|             protected.                                            |
|                                                             |
|   ICR    -- Cartridge reference.  An optional one-word variable.  |
|             If positive, specifies a cartridge reference number.  |
|             If negative, specifies a logical unit number.  If 0   |
|             or -32768 (or omitted),  first available  cartridge   |
|             with  enough  room  will  be used.  If  in  session   |
|             environment,   the  ICR  must  specify  a  cartridge   |
|             mounted to the user's session.                        |
|                                                             |
|   IDCBZ  -- Packing buffer size.   An optional one-word variable  |
|             set to the number of words  in the packing buffer if  |
|             more than 128 are requested. If omitted, FMP assumes  |
|             DCB  size  (control words + packing buffer)  is  144  |
|             words regardless of the IDCB dimension.               |
|                                                             |
|   JSIZE  -- Created file size. Optional double-word parameter in  |
|             which  the actual file size created  (in sectors) is  |
|             returned if the call was successful.                  |
|                                                             |
+-------------------------------------------------------------+
```

COMMENTS

FILE SIZE---Records are addressed by number; therefore, the number of
records contained in a file must not exceed the maximum accessible
using the file access calls described later in this section.

If a file is to be accessed with the standard access calls (READF,
WRITF, etc), the number of records contained in a file must not
exceed (2**15)-1.   If a file is to be accessed with the extended
access calls (EREAD, EWRIT, etc), the number of records must not
exceed (2**31)-1.

The number of records contained in a file can be determined as
follows:

$$\text{number of records} = \frac{\text{words in file (blocks in ISIZE x 128)}}{\text{record length in words (ISIZE(3) and ISIZE(4))}}$$

When the exact size is not known, the rest of the cartridge may by allocated by setting ISIZE to -1 (valid only for files of type 3 or greater). The unused area may be returned to the system by truncating the file when it is closed (refer to the CLOSE and ECLOS call described later). The rest of the cartridge but not more than 32767 x 128 blocks will be allocated.

The CRETS call allocates files in block increments. After 16,383 blocks have been allocated, then CRETS starts allocating files in chunk increments (1 chunk = 128 blocks). If a file size (ISIZE) is specified that is not evenly divisible by 128, a full 128 block chunk is allocated to accommodate the remainder.

Users of the scratch files are encouraged to use the default size. The reason for this is that when a file is created requesting a specific size, a check is made in the directory for a purged file of exactly the same size. If such a file is found, the new file replaces it; if such a file is not found, the new file is placed after all other files on the cartridge. Therefore, if most programs request the same size (that is, the default size) scratch files, the space on the disc is used more efficiently.

FILE TYPES -- FMP defined file types are as follows:

   Type 1. Fixed length, 128-word records; random access.

   Type 2. Fixed length, user-defined record lengths; random access.

   Type 3 (and greater). Variable length records; sequential access; automatic extents provided.

          type 4 -- source program
          type 5 -- relocatable program
          type 6 -- memory-image program
          type 7 -- absolute binary program

File types greater than 7 are user defined but are treated by FMP as type 3. Any special processing based on file type is not provided as a default, it must be specified.

When any file of type 3 or greater is created, FMP writes an EOF mark at the beginning of the file. As records are written to the file, the EOF is moved automatically to follow the last record.

Further details on file types are given at the beginning of this section.

EXAMPLE 1 -- Create a type 3 scratch file with 24 blocks, security code HP, cartridge A3, and a DCB packing buffer of 128 words. At the end of the program, the file will be renamed to DATA05.

```
        .
        .
      IMPLICIT INTEGER (A-Z)
      DIMENSION DCB (144), NAME (3), NUNAME (3), SIZE (4)
      DATA SIZE /0,24,0,0/, NUNAME/2HDA,2HTA,2H05/
      TYPE = 3
      SC = 2HHP
      CRN = 2HAB
      NUM = 23
        .
        .
      CALL CRETS (DCB,ERROR,NUM,NAME,SIZE,TYPE,SC,CRN)
      IF (ERROR.LT.0) GO TO 500
        .
        .
        .
      CALL NAMF (DCB,ERROR,NAME,NUNAME,SC,CRN)
      IF (ERROR.LT.0) GO TO 700
        .
        .
        .
```

OPEN and OPENF Calls

These routines open a file for access and position it at the first record. The file must have been created prior to the OPENF or OPEN call. The file opened may be a disc or non-disc (type 0) file. Type 0 files may be opened with a function code specified at creation or the original function code may be overridden.

Files may be opened for exclusive use of the calling program or for non-exclusive use of up to seven programs. A file may be opened for update or for standard sequential write operations.

By using the OPENF call, a logical unit number can be passed in the first word of the INAM parameter causing a DCB to be created to allow type 0 access to the logical unit. No type 0 file is necessary and none is created by the call.

```
+-----------------------------------------------------------------+
|                                                                 |
|   CALL OPEN(IDCB,IERR,INAM[,IOPTN][,ISC][,ICR][,IDCBZ])         |
|         or                                                      |
|   CALL OPENF(IDCB,IERR,INAM[,IOPTN][,ISC][,ICR][,IDCBZ])        |
|-----------------------------------------------------------------+
|                                                                 |
|   IDCB ---   Data Control Block.  An array of 144+n words where |
|              n is positive or zero.                             |
|                                                                 |
|   IERR ---   Error return.  A one-word variable that a negative |
|              error code is  returned to for unsuccessful calls. |
|              File type is returned for successful calls.        |
|                                                                 |
|   INAM ---   File name or LU. This is either a three-word array |
|              containing the ASCII file name (for OPEN or OPENF) |
|              or an integer containing a binary LU (OPENF only). |
|                                                                 |
+---------------CONTINUED ON NEXT PAGE ---------------------------+
```

```
+------------------------------------------------------------------+
| IOPTN --- Open options;  optional 1-word variable  set to        |
|           octal value to specify non-standard opens. If omit-    |
|           ted or set to zero,  the file is opened by default     |
|           as follows:                                            |
|                                                                  |
|           *  Exclusive  use  -  only the calling program can     |
|              access the file.                                    |
|                                                                  |
|           *  Standard  sequential  output  -  each record is     |
|              written following the last, destroying any data     |
|              beyond the record being written.                    |
|                                                                  |
|           *  File type defined for file  at creation is used     |
|              for access.                                         |
|                                                                  |
|           *  Type 1 and 2 files are not extendable.              |
|                                                                  |
|           To open  a file  with other options,  set IOPTN as     |
|           described below under OPEN OPTIONS.                     |
|                                                                  |
| ISC   --- Security code.  An optional one-word variable be-      |
|           tween 0 and +/-32767.  If 0 (or omitted), the file     |
|           is not protected.  If positive, the file is write      |
|           protected only.  If negative, the file is write and    |
|           read protected.                                        |
|                                                                  |
| ICR   --- Cartridge reference;  optional 1-word variable; if     |
|           set, FMP searches that cartridge for file; if omit-    |
|           ted, it searches cartridges in the caller disc add-    |
|           ressing space and opens first file found with spec-    |
|           ified name.                                            |
|                                                                  |
| IDCBZ --- DCB buffer size;  optional 1-word variable; set to     |
|           number of words in DCB packing buffer if larger than|
|           128; if omitted, FMP assumes DCB size (control words|
|           + buffer) is 144 words regardless of IDCB dimensions|
+------------------------------------------------------------------+
```

COMMENTS:

OPEN OPTIONS -- The IOPTN parameter is defined as follows:

```
                    (X)(A)(K)(V)(M)(EX)   (F)(T)(U)(E)
           ------------------------------------------------
           |XX|XX|XX|XX|XX|10| 9| 8| 7| 6| 5|XX| 3| 2| 1| 0|
           ------------------------------------------------
                           |             |          |
                           +---Function---+          |
                           |     Code                |
                           |                         |
                           +-----Type 0 Options----+
```
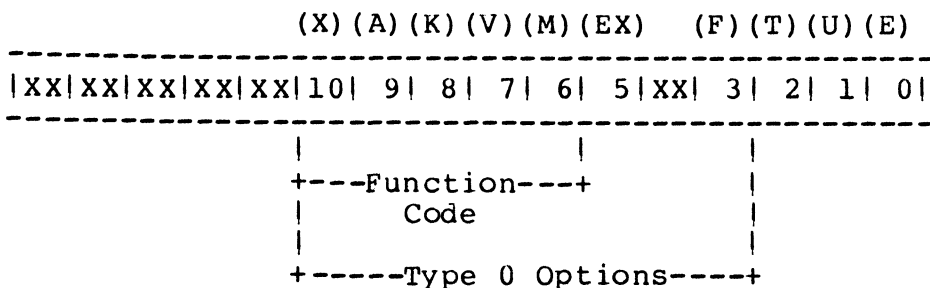
The following bits can be set for any file type:

E (bit 0) = 0 File opened exclusively for this program. In an
                 OPENF of an LU or OPEN of a type 0 file, if the open
                 is exclusive and the device is not interactive, the
                 device is locked. The device becomes unlocked when
                 the DCB is closed.

           1 File may be shared by up to seven programs

U (bit 1) = 0 File opened for standard (non-update) write
(disc files only)

           1 File opened for update

T (bit 2) = 0 Use file type defined at creation
(disc files only)

           1 File type is forced to type 1

The following bits are used for type 0 files only (they are ignored
when opening other file types):

F (bit 3) = 0 Use function code defined at creation (see FMGR :CR
                 command described in the RTE-IVB Terminal User's
                 Reference Manual). If an LU is used in an OPENF
                 call, the defaults are as shown in Table 3-4.

           1 Use function code defined in bits 6-10 of IOPTN.

The following bit is used for type 1 and 2 files only (it is ignored
when opening other file types):

EX (bit 5) = 0 File is not extendable.

           1 File extents are to be created automatically.

Bits 6-10 correspond exactly to the function code used for the READ
or WRITE EXEC call. These are driver dependent and the appropriate
driver reference manual should be consulted for more information.

Table 3-4. OPENF Defaults

DEVICE	DEVICE TYPE	EOF CODE	SPACING	READ/WRITE
Reader or Punch	2	LE	BO	BO
Minicartridge	5 subchannel 1, 2	EO	BO	BO
Mag/Tape, mass storage, other	>17	EO	BO	BO
All others	—	PA	BO	BO
Bit Bucket	—	PA	BO	WR

```
    EOF CODE                   SPACING                READ/WRITE
    --------                   -------                ----------

EO=subfunction 0100       FS=forward space          RE=read
LE=subfunction 1000       BS=back space             WR=write
PA=subfunction 1100       BO=both                   BO=both
```

EXCLUSIVE OPEN---By default, a file is opened for exclusive use of the calling program. An exclusive open is granted to only one program at a time. If the call is rejected because the file is open to another program, the call must be made again; it is not stacked by FMP. Exclusive open is useful in order to prevent one or more programs from destructively interfering with each other.

NON-EXCLUSIVE OPEN---If more than one program needs to access the file, it should be opened non-exclusively by setting the IOPTN E bit. A non-exclusive open may be granted to as many as seven programs per file at one time. A non-exclusive open will not be granted if the file is already opened exclusively. Each time an open is requested for the file, all programs currently having the file open are checked. If any program is dormant, the file is closed to that program. That type of close does not free the DCB and does not post the contents of the DCB buffer to the file. Any open flag will also be cleared if it does not point to a valid ID segment.

UPDATE OPEN---In update mode, IOPTN U bit set, the block containing the record to be written is automatically read into the DCB before it is modified. This insures that existing records in the block will not be destroyed. This mode of open has no effect on reading or positioning, it is only necessary when writing to a site that already contains valid data, or when building a site in a random manner.

Update mode should be used to write to type 2 files. A type 2 file should be opened in standard mode only when originally writing the file or adding to the end of the file, and then only if it is to be written sequentially.

Type 1 files should not be opened in update mode. Although, like type 2, they are designed for random access with fixed length records and the end-of-file in the last word of the last block, each record is the same length as the block transferred so that there is no danger of writing over existing records.

For type 3 and above files, update mode is not generally used; most writes are sequential with an end-of-file mark written after each record. These files should be opened for update only if a record previously written to the file is being modified. In this case, care must be taken not to change the length of the modified record. If it is changed, a -005 error is issued. Regardless of the mode of open (update or standard) a record written beyond the end-of-file replaces the end-of-file and is followed by a new end-of-file.

TYPE 1 ACCESS---Any file may be forced to type 1 access by setting the IOPTN T bit. Type 1 access is faster because it bypasses the Data Control Block buffer and transfers a block of data directly to the user buffer defined as IBUF. The file type defined at creation is not affected; the file is treated as type 1 only for the duration of this open. The program is responsible for any packing or unpacking of records in files forced to type 1.

OPEN FLAGS---Occasionally files are left open to a program upon termination. The operating system and D.RTR cooperate to close these files when necessary. A termination sequence counter is kept in the ID segment. Each time a program is terminated or removed from an ID segment, the termination counter in that ID segment is incremented. The open flag placed in a file's directory entry whenever a file is opened or created contains both the ID segment number and the termination sequence counter. Whenever D.RTR finds an open flag whose termination counter is not current with the ID segment, or the program occupying the ID segment is dormant, it removes the flag.

EXAMPLE 1 -- Open a type 2 file named FIX for update in non-exclusive
mode. The security code at creation was AB.

```
   DIMENSION NAME(3),IDCB1(144)
   DATA NAME/2HFI,2HX 2H /
   .
   .                          +----------set bit 1 for update and
   .                          v          bit 0 for non-exclusive open
   CALL OPEN(IDCB1,IERR,NAME,3B,2HAB)
   IF (IERR .NE. 2) GO TO 900 <-------- test for error; file type expected
   CONTINUE
   .
   .
   .
```

EXAMPLE 2 -- Open type 3 file (PROG1) with default options. The file
is located on logical unit 14:

```
   DIMENSION NAM(3),IDCB2(144)
   DATA NAM/2HPR,2HOG,2H1 /
   .                          +-------default options
   .                          |
   .                          v v-----no security code
   CALL OPEN(IDCB2,IERR,NAM,0,0,-14)
   IF (IERR .NE. 3) GO TO 900
   CONTINUE
   .
```

Although PROG1 was created with a DCB buffer of 256 words (refer to
CREAT examples), it can be opened and accessed with the minimum DCB
buffer of 128 words.

EXAMPLE 3 -- Open type 0 file PTAPE (created with :CR command) and
set options so that binary data is punched on paper type without
leader.

```
   DIMENSION NAME(3),IDCB3(144)
   DATA NAME/2HPT,2HAP,2HE /
   IOPTN=2110B <------------------- set M, V, and X for "honesty mode"
   .
   .
   .
   CALL OPEN(IDCB3,IERR,NAME,IOPTN)
   IF (IERR .NE. 0) GO TO 900
   .
   .
   .
```

For a description of "honesty mode", refer to the DVR00 Programming
and Reference Manual.

EXAMPLE 4 -- Use OPENF call to allow programmatic type 0 access to LU
6 (line printer). Use defaults for options (IOPTN).

```
DIMENSION NAM(3),IDCB(144)
NAM(1)=6
  .
  .
  .
CALL OPENF(IDCB,IERR,NAM)
IF (IERR .LT. 0) GO TO 900
CONTINUE
  .
  .
```

CLOSE and ECLOS Calls

A call to CLOSE or ECLOS removes the logical connection between the
Data Control Block (DCB) and the File Directory entry associated
with the file. The DCB is freed for association with other files; the
entry for the file is maintained in the File Directory.

A disc file opened for exclusive use of the calling program may be
truncated freeing unused disc space. Up to 32767 blocks may be
truncated with the CLOSE call. Up to 32767 x 128 blocks may be
truncated with the ECLOS call.

```
+--------------------------------------------------------------------+
|                                                                    |
|   CALL CLOSE(IDCB<,IERR>[,ITRUN])                                   |
|       or                                                           |
|   CALL ECLOS(IDCB<,IERR>[,ITRUN])                                   |
|--------------------------------------------------------------------+
|                                                                    |
|   IDCB---Data Control Block.  An array of 144+n words where n      |
|          is positive or zero;  previously associated with a        |
|          file in a create or open FMP call.                        |
|                                                                    |
|   IERR---Error return. A one-word variable in whicn a negative     |
|          error code is returned if a truncation operation was      |
|          unsuccessful; only required when ITRUN is specified.      |
|                                                                    |
|   ITRUN--Truncation word.  For CLOSE call, optional one-word       |
|          variaole containing integer number of blocks to be        |
|          deleted from the file at closing; if zero or omitted,     |
|          the file is closed without truncation; if negative,       |
|          only extents are truncated.  ITRUN must be </= 32767      |
|          for CLOSE call.                                           |
|                                                                    |
|          For ECLOS call, ITRUN is an optional double-word vari-    |
|          able specifying the number of blocks to be deleted        |
|          from the file at closing;  if zero or omitted,  the       |
|          file is closed without truncation. If negative, only      |
|          extents are truncated.   ITRUN must be </= (2**31)-1      |
|          for ECLOS call.                                           |
|                                                                    |
+--------------------------------------------------------------------+
```

COMMENTS:

FILE TRUNCATION---When a file has been created with more blocks than
are actually needed to accommodate the data in it, it can be
truncated at closing to save disc space. A file may be truncated only
if:

*   the file is a disc file

*   the current position is in the main file, not in an extent

* the file was opened with the correct security code

* the file was opened for exclusive use of the calling program

* the number of blocks deleted are less than or equal to the total number of blocks in the file.

If all these conditions are met, the value of ITRUN can be a:

* positive integer - to specify the number of blocks to be deleted from the end of the main file; any extents are automatically truncated: if equal to the total number of blocks in the file, the file is purged.

* negative integer - to specify that any extents be deleted from the file; the main file is not affected.

If a main file larger than 16383 blocks is to be truncated (previously created with an ECREA call), it is done on a "chunk" basis (chunk = 128 blocks). For example, if 1 </= ITRUN </= 127, the file size is not changed. If 128 </= ITRUN </= 255, 1 chunk is deleted. If the file size is less than or equal to 16,383 blocks, the actual number of blocks specified in ITRUN are truncated.

The value of ITRUN (when positive) can be determined from information returned by a previous call to LOCF or ELOCF. Assuming the current position is at the end of the data in the file, the block number of the block containing the next record to be written or read (IRB+1) is subtracted from the total blocks with which the file was created (JSEC/2) and assigned to ITRUN. When negative, ITRUN can be any value. The number of extents need not be known. If the file is currently positioned in an extent, it can be re-positioned to the main file with a RWNDF call.

A zero value for ITRUN is exactly the same as omitting this parameter; a standard close is performed with no truncation.

EXAMPLE 1---Close file FIX (IDCB1) with no truncation; assume FIX has been opened:

```
DIMENSION IDCB1(144) <------------ file name associated with IDCB1
    .
    .
    .
CALL CLOSE(IDCB1,IERR)
IF (IERR .LT. 0) GO TO 900 <------ error return
CONTINUE
    .
    .
    .
```

IDCB1 is freed for association with other files.

EXAMPLE 2---Close type 3 file PROG1 and truncate it to exactly the
number of blocks used. Assume the file was written sequentially and
the current location is at the end of the data in the file.

```
    DIMENSION IDCB2(144)
    .
    .
    CALL LOCF(IDCB2,IERR,I,IRB,I,JSEC) <--call LOCF for file length and
    ITRUN=(JSEC/2)-(IRB+1)                       next block
    CALL CLOSE(IDCB2,IERR,ITRUN) <--------IERR required with truncation
    IF (IERR .LT. 0) GO TO 900
    .
    .
    .
```

Since JSEC contains the number of sectors it must be divided be 2 for
the number of blocks. IRB is the next block number to be written to;
a 1 must be added since blocks are numbered starting with 0.

EXAMPLE 3---Close the same file (PROG1), but delete only the extents;
use the RWNDF call to insure that current position is in the main
file.

```
    DIMENSION IDCB2(144)
    .
    .
    .
    IF(RWNDF(IDCB2))900,10,10
 10 CALL CLOSE(IDCB2,IERR,-1) <-------- ITRUN negative to delete
    IF (IERR .LT. 0) GO TO 900        all extents
    CONTINUE
    .
    .
    .
```

PURGE Call

A call to PURGE removes the named file from the system along with any extents associated with the file. Only disc files can be purged through a program call; non-disc files may be purged with the :PU FMGR command (see RTE-IVB Terminal User's Reference Manual). When a file is purged, the file directory entry is no longer available. If the file was open, it is closed freeing the Data Control Block. A file that is open to any program other than the calling program cannot be purged until it is closed by the other program(s).

```
+-------------------------------------------------------------------+
|                                                                   |
|   CALL PURGE(IDCB,IERR,INAM<,ISC><,ICR>)                           |
|                 ----                                               |
|                                                                   |
|-------------------------------------------------------------------|
|                                                                   |
|   IDCB---Data Control Block.  An array of 144+n  words where n    |
|          positive or zero.                                        |
|                                                                   |
|   IERR---Error return.  A one-word  variable  that a negative     |
|          error code is returned to for unsuccessful calls.        |
|          Zero is returned for successful calls.                   |
|                                                                   |
|   INAM---File name.  A  three-word  array containing the ASCII    |
|          name of the file to be purged.                           |
|                                                                   |
|   ISC----Security code. An optional one-word variable; must be    |
|          specified if file to be purged has a security code;      |
|          may be omitted if file does not have a security code.    |
|                                                                   |
|   ICR----Cartridge reference. An optional one-word variable; if   |
|          specified  FMP purges the named file on the specified    |
|          cartridge;  if omitted  FMP  searches  the  cartridge    |
|          directory  and  purges  the  first  file found in the    |
|          users disc addressing space with the specified name.     |
|                                                                   |
+-------------------------------------------------------------------+
```

COMMENTS:

PURGING FILES OPENED NON-EXCLUSIVELY---When a file is opened to more than one user, each user should first close the file before it is purged to insure a successful purge. Only the calling program may purge the file without first closing it, and then, only if it is closed to all other programs.

A type 6 file cannot be purged if an ID segment pointing to the disc space occupied by the type 6 file exists (the file has been RP'ed).

RECOVERY OF FILE AREA FOLLOWING PURGE---The area on disc occupied by a purged file is returned to the file system automatically only if the purged file is the last file on the cartridge. If the file is the last file, all the area from the beginning of the file to the end of the cartridge is returned to the file system. If the file is followed by other files, the cartridge must be packed in order to recover the file space. Packing is accomplished with the :PK FMGR command (refer to the RTE-IVB Terminal User's Reference Manual).

Note that space from a purged file in the middle of the cartridge will be reused if a file is created with exactly the same size as the file that was purged.

EXAMPLE 1---Purge the type 2 file named FIX created with security code AB.

```
    DIMENSION NAM1(3),IDCB1(144)
    DATA NAM1/2HFI,2HX ,2H /
    .
    .
    CALL PURGE(IDCB1,IERR,NAM1,2HAB)
    IF (IERR .LT. 0) GO TO 900 <---------- process errors
    CONTINUE
    .
    .
    .
```


EXAMPLE 2---Purge type 3 file (PROG1) created without a security code on logical unit 14.

```
    DIMENSION NAM2(3),IDCB2(144)
    DATA NAM2/2HPR,2HOG,2H1 /
    .
    .
    .
    CALL PURGE(IDCB2,IERR,NAM2,0,-14)
    IF (IERR .LT. 0) GO TO 900
    CONTINUE
    .
    .
    .
```

File Access

Information in files is accessed with the READF, EREAD, WRITF, and EWRIT routines. Calls to these routines are basically the same whether the file is a device (type 0) or a disc file (type 1 and above). Whether reading or writing, you can specify that exactly one record be transferred or you may specify a particular number of words. In general, it is good practice to specify the number of words since this permits generality among file types.

The normal mode of access for files of type 3 and above is sequential. Such files are created with an end-of-file in the first record. The first record written overrides the end-of-file and a new end-of-file is written immediately following the record. As each subsequent record is written, the process is repeated so that the end-of-file always follows the last record written.

Variable length records are assumed for these file types. For this reason it is necessary to specify the number of words when the record is written. On a read, if the number of words is not specified, FMP determines its length and reads exactly one record.

For file types 1 and 2, random access is the normal mode. No end-of-file is written. The end-of-file is calculated according to the file size defined at creation. Since each record is a fixed length determined at creation, the file is easily positioned to a particular record. Generally, one record is written or read at a time, although more may be transferred when accessing a type 1 file.

When accessing a type 0 file, the number of words should be specified unless a zero-length record is to be read or written or a record skipped. End-of-file marks are not written automatically to type 0 files; you must specify the end-of-file.

The record format for fixed and variable length records is shown in Appendix D.

READF and EREAD Calls

These routines read a record from an open file into a user programs buffer. One full record or a specified number of words can be read. The record to be read may be the record at which the file is currently positioned or, for type 1 and 2 files, it may be any specified record. The READF call can specify a record number up to $(2**15)-1$. The EREAD call can specify a record number up to $(2**31)-1$.

```
+------------------------------------------------------------------+
|                                                                  |
|   CALL READF(IDCB,IERR,IBUF[,IL][,LEN][,NUM])                     |
|       or              ----  ----                                 |
|   CALL EREAD(IDCB,IERR,IBUF[,IL][,LEN][,NUM])                     |
|                       ----  ----                                 |
|                                                                  |
+------------------------------------------------------------------+
|                                                                  |
|   IDCB---Data Control Block.  An array of 144+n  words where n   |
|          is positive or zero; previously specified in a create   |
|          or open operation.                                      |
|                                                                  |
|   IERR---Error return. A one-word variable in which a negative   |
|          error code is returned  for unsuccessful calls.  Zero   |
|          is returned for successful calls.                       |
|                                                                  |
|   IBUF---Data buffer.  An array into which the  requested data   |
|          is placed by the system. It should be large enough to   |
|          contain the largest record expected.                    |
|                                                                  |
|   IL-----Data length requested.  An optional one-word variable   |
|          that specifies the number of words to be read.  Refer   |
|          to Table 3-5 for details.                               |
|                                                                  |
|   LEN----Data length read.  An optional  one-word  variable in   |
|          which actual number of words read is returned. Set to   |
|          -1 if end-of-file is read.                              |
|                                                                  |
|   NUM----Record number. An optional one-word  (for READF)   or   |
|          double word variable  (for EREAD) used to specify the   |
|          record number to be read  (if positive) or the number   |
|          records to  backspace  (if negative).  Used only for    |
|          type  1 or 2  files.  If omitted,  the record  at the   |
|          current position is read.                               |
|                                                                  |
+------------------------------------------------------------------+
```

COMMENTS:

RELATION OF IL TO FILE TYPE---It is a good idea to specify IL for file type 0 and it doesn't hurt to specify it for other file types. If you do not know the length of a disc file record, IL can be specified as the user buffer length to prevent reads beyond this area. If the record is shorter than IL, the exact record length is read for file types greater than type 1. Table 3-5 illustrates the effect of IL depending on file type.

Table 3-5. Effect of IL Parameter in READF

IL VALUE	FILE TYPE 0	FILE TYPE 1	FILE TYPE > 1
IL > 0	Up to IL words are read; if less than IL, file defined record length is read.	Exactly IL words are read; IL may be more or less than 128-word record.	Up to IL words are read; if less than IL, actual record length is read.
IL = 0 (not recommended)	Zero length record is read; usually record is skipped and counted as read.	No action. (Zero-length read, no position change.)	Record is skipped and counted as read.
IL omitted	Zero-length record is read; usually record is skipped and counted as read. (Not recommended.)	128-word record is read.	Actual record length is read.
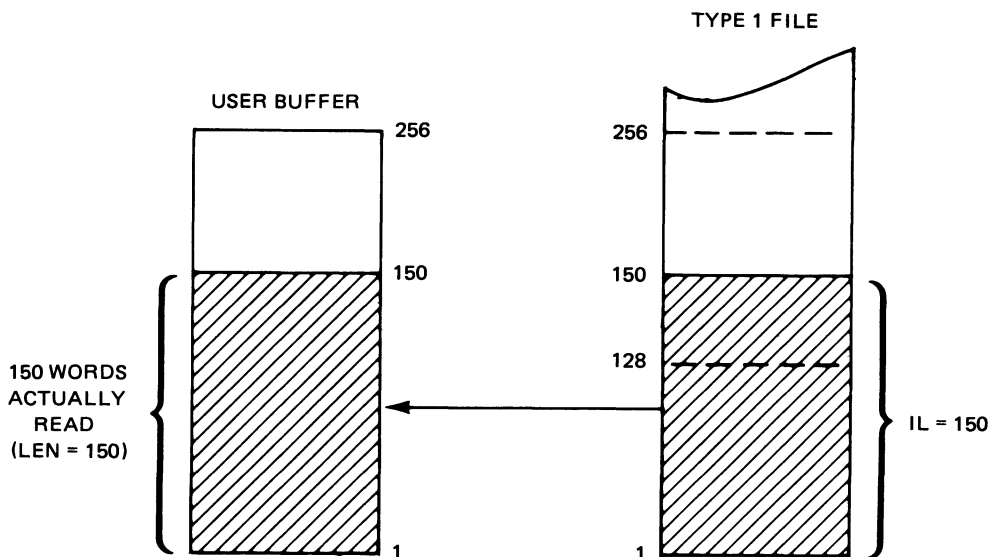
Figure 3-4. Read Type 1 File When IL Greater Than 128

Figure 3-4 illustrates a type 1 file read. The file is read directly to the user buffer when the number of words specified in IL is greater than the 128 words expected for a type 1 file. Other file types may be forced to type 1 at open in order to benefit from this type of transfer.

USING LEN -- Upon completion of a read, the actual number of words transferred to the user buffer is returned in LEN. If, however, the number of words in LEN is equal to IL, more words may actually have been in the disc record. This is because LEN is never set to a value greater than IL.

LEN can be used to test for possible overflow of the user buffer. Except for type 1 files, the user buffer and IL can be specified one word larger than the largest expected record. If, when tested, LEN equals this size, it is a good indication that the record read was too large for the buffer. Do not use this test for type 1 files since exactly IL words are read for this file type.

Another use of LEN is to test for end-of-file in all file types except 1 and 2. For types 1 and 2, an end-of-file is reported as an error in IERR. Depending on file type, reading an end-of-file results in the following:

* Type 0   LEN is set to -1 when EOF is read; no error occurs and access may continue beyond the end-of-file.

* Type 1 & 2   IERR is set to -12 indicating an error. Access is not permitted beyond the end-of-file.

* Type 3 & up LEN is set to -1 for the first EOF read; no error
occurs out an attempt to read past this EOF causes an
error (IERR = -12); you may not read past the
end-of-file, but you may write beyond it.

Note that length words in variable-length records (file types 3 and
above) are not transferred to the user buffer and are not counted in
LEN.

POSITIONING WITH NUM---NUM is used only to position file types 1 and
2; it may be specified for other file types but is ignored. If
positive, NUM specifies the record number of the record to be read;
records are numbered from the first record in the file starting with
1 and proceeding sequentially upward. If negative, NUM specifies the
number of records to backspace from the current position in the file.

To illustrate, assume the file is positioned at the beginning of
record 4:

1. If NUM=0 or is omitted, read record 4.

2. If NUM=6, read record 6.

3. If NUM=-3, read record 1.

```
        SOF                                            EOF

         | |  1  |  2  |  3  |  4  |  5  |  6  |  8  | |
      ---+ +-----+-----+-----+-----+-----+-----+-----+ +---
         | |     |     |     |     |     |     |     | |
                             ^
                             |
                             |
                   +--CURRENT POSITION
```
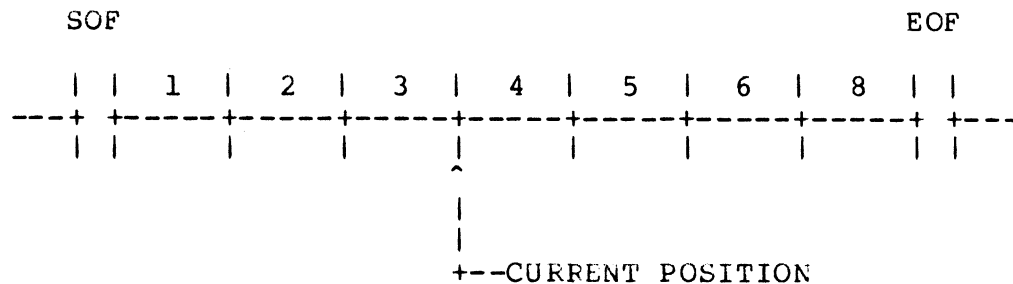
EXAMPLE 1---Read records in a type 0 file until an end-of-file is reached; the record length is 80 words but 81 words are assigned to the buffer in order to test LEN; assume the file is positioned at the first record.

```
          DIMENSION IDCB0(656),IBUF(81)
          .
          .
          .
100       CALL READF(IDCB0,IERR,IBUF,81,LEN)
          IF(IERR .LT. 0) GO TO 900
          IF(LEN .EQ. -1) GO TO 500 <--- test for end-of-file

          IF(LEN .GT. 80) GO TO 550 <--- test for record greater than
          .                                    80 words
          . <----------------------- process record
          .
          GO TO 100 <------------------ read next record
          .
          .
          .
500       CALL CLOSE(IDCB0,IERR) <------ close file at end-of-file
          IF(IERR .LT. 0) GO TO 910
          END
```

EXAMPLE 2---Read record 24 from a type 2 file having 256 word records; use 257 word buffer.

```
          DIMENSION IDCB1(272),IBUF1(257)
          .
          .
          .
          CALL READF(IDCB1,IERR,IBUF1,257,LEN,24)
          IF(IERR .LT. 0) GO TO 900 <---- test for error or EOF
          IF(LEN .EQ. 257) GO TO 550 <--- test for too long record
          .
          . <-------------------------- process record
          .
```

EXAMPLE 3---Read file with variable-length records until first
end-of-file is reached. Assume the file is positioned at the first
record and that no record exceeds 128 words.

```
          DIMENSION IDCB2(144),IBUF2(129)
          .
          .
          .
100       CALL READF(IDCB2,IERR,IBUF2,129,LEN)
          IF(IERR .LT. 0) GO TO 900
          IF(LEN .EQ. -1) GO TO 500 <----- test for end-of-file
          IF(LEN .EQ. 129) GO TO 150 <---- test for buffer overflow
          .
          . <---------------------------- process record
          .
          GO TO 100 <-------------------- read next record
          .
          .
          .
          .
          .
500       CALL CLOSE(IDCB2,IERR) <-------- close file at end-of-file
          .
          .
          .
```

WRITF and EWRIT Calls

These routines transfer a record from the user's buffer to an open file. For type 0 files or type 3 (and above) files, the specified number of words is written. For type 1 files, records are transferred in blocks of 128 words. For type 2 files, records are transferred in the lengths specified when the file was created. Type 2 files should be opened in update mode in order to write to the file (refer to the OPEN FMP call).

For type 1 and 2 files, a specific or relative record number can be specified in NUM. For the WRITF call, the record number specified must be less than, or equal to, $(2**15)-1$. For the EWRIT call, the record number can be up to $(2**31)-1$.

```
+----------------------------------------------------------------+
|                                                                |
|    CALL WRITF(IDCB,IERR,IBUF[,IL][,NUM])                        |
|       or           ----                                        |
|    CALL EWRIT(IDCB,IERR,IBUF,[IL],[NUM])                        |
|                    ----                                        |
|                                                                |
|----------------------------------------------------------------|
|                                                                |
|    IDCB---Data Control Block.  An array of 144+n  words where n |
|           is positive or zero; previously specified in a create |
|           or open call.                                        |
|                                                                |
|    IERR---Error return. A one-word variable in which a negative |
|           error code is  returned if the call was unsuccessful. |
|           Zero is returned for successful calls.               |
|                                                                |
|                                                                |
|    IBUF---Data buffer.  An array where the user places the data |
|           to be written.  It should be  large enough to contain |
|           the largest expected record.                         |
|                                                                |
|    IL-----Buffer length.  An optional one-word variable used to |
|           specify the number of words to be written. If omitted, |
|           one  record is  written  to type 1 and 2 files; zero- |
|           length  record is  written  for the other file types. |
|           Refer to Table 3-6 for details.                      |
|                                                                |
|    NUM----Record number. An optional  one-word  variable  (for |
|           WRITF)  or double-word variable (for EWRIT) used to   |
|           specify the record to be written to, (if positive) or |
|           the  number of records  to backspace (if negative).  |
|           Used only for type 1 and 2 files.  If omitted, record |
|           at current file position is written to.              |
|                                                                |
+----------------------------------------------------------------+
```

COMMENTS:

RELATION OF IL TO FILE TYPE---IL should be specified for all but type 2 files and may be specified for all files. It is ignored by type 2 files but can be used with type 1 files to write more than one 128-word record at a time. For files of type 3 and above, it is essential to specify record length in IL. To omit IL for these file types is the same as setting IL=0, a zero-length record is written. Refer to Table 3-6.

IL can also be used to write an end-of-file on files of type 0, type 3 or greater. An attempt to write an end-of-file to a type 1 or 2 file is ignored; no error is indicated.

Table 3-6. Effect of IL Parameter in WRITF

IL VALUE	FILE TYPE 0	FILE TYPE 1	FILE TYPE 2	FILE TYPE > 2
IL > 0	Exactly IL words are written.	IL is rounded up to 128 or a multiple of 128.	IL is ignored; file defined record length is written.	Exactly IL words are written.
IL = 0	Zero length record is written.	No action.	IL is ignored; file defined record length is written.	Zero length record is written.
IL omitted	Zero length record is written.	128 words are written.	IL is ignored; file defined record length is written.	Zero length record is written.
IL=−1	End-of-file is written.	No action.	No action.	End-of-file is written.
IL <−1 not recommended	IL is treated as a character count.	No action.	No action.	Undefined.

When IL is not 128 or a multiple of 128 for a type 1 file, it is rounded up so that 128 or a multiple of 128 words are always transferred. The user buffer need be no larger than the size specified in IL. If the exact size is always read, no problems result from the transfer of words beyond the buffer. Figure 3-5 illustrates a write to a type 1 file with IL = 150 words. In this case, 256 words (the shaded area) are actually transferred.
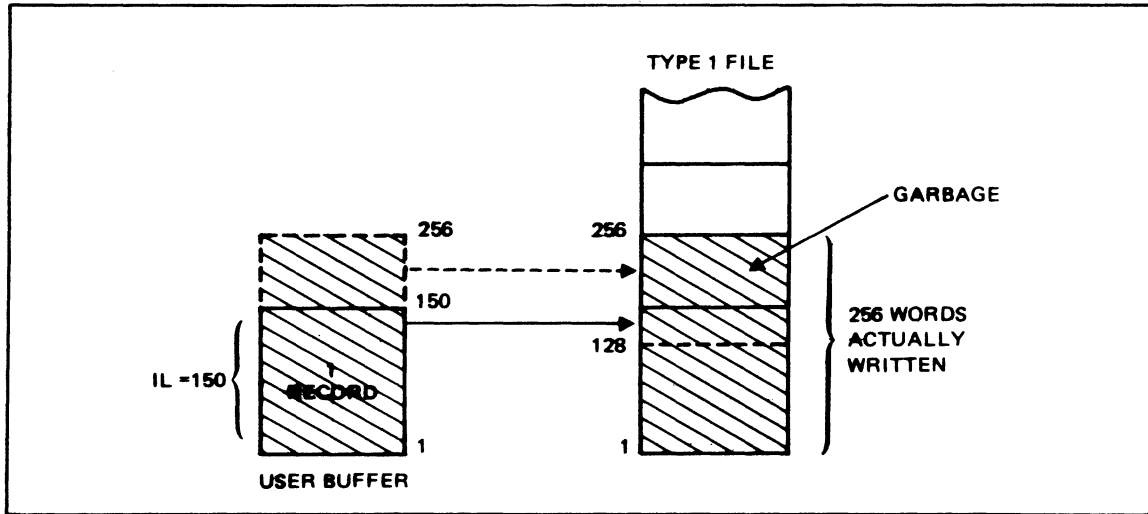
Figure 3-5. Sample Write to Type 1 File

POSITIONING WITH NUM---NUM is used only to position file types 1 and 2; if specified for other file types, it is ignored. A positive value causes a write to the specified record number; records are numbered relative to the start of the file beginning with 1. When negative, NUM specifies the number of records to backspace from the current file position.

To illustrate, assume the file is positioned at the beginning of record 5:

1. If NUM=0 or is omitted, write is to record 5.

2. If NUM=6, record number 6 is written.

3. If NUM=-3, record number 2 is written.
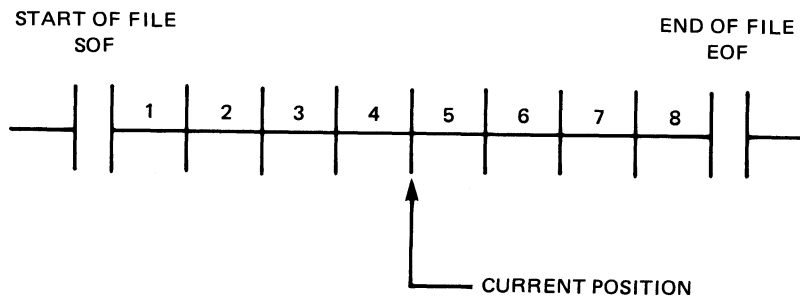
NOTE

Although it is possible to rewrite specific records in files of type 3 and above, great care must be taken. If the length of the existing record and that of the replacing record are not identical, the integrity of the file is destroyed.

EXAMPLE 1---Write records sequentially to a file starting at record 1; when all records are written, write an end-of-file; set IL to the exact record length of each record using a maximum record length of 100. File could be type 0 or a type 3 or above:

```
          DIMENSION IDCB0(144),IBUF0(100)
          .
          .
          .
    100 <----------------------------------- move record to IBUF0
          .
          .
          .                                   number of words in record
          IL = <--------------------------   (after last record,
          .                                   set IL = -1)
          .
          CALL WRITF(IDCB0,IERR,IBUF0,IL)
          IF(IERR .LT. 0) GO TO 900 <--------- check for error
          IF(IL) 110,100
    110   CALL CLOSE(IDCB0,IERR) <------------ close file after EOF
          .
          .
          .
```

EXAMPLE 2---write record number 24 to a type 2 file with a record length of 256 words:

```
          DIMENSION IDCB2(272),IBUF2(256)
          .
          .
          . <---------------------------------- move record to IBUF2
          CALL WRITF(IDCB2,IERR,IBUF2,0,24)
          IF(IERR .LT. 0) GO TO 900
          .
          .
          .
```

File Positioning

The FMP positioning routines provide a variety of ways to position a file. The methods are summerized below:

*   APOSN and EAPOS -- Can be used to position all file types except type 0. Used to position a file to specific record; if file contains variable-length records, LOCF or ELOCF must be used in conjunction with APOSN or EAPOS, respectively.

*   POSNT and EPOSN -- Can be used to position all file types. Used to position a file forward or backward (if backspace legal) from the current file position. Can also be used to position a file to a specific record.

*   RWNDF -- Can be used to position a file at the first record for any file that permits backspacing.

Disc files with fixed-length records (type 1 and 2) can also be positioned to a particular record or backspaced with the NUM parameter in the READF, EREAD, WRITF, or EWRIT calls.

All disc files and magnetic tape files can be backspaced. Non-disc files, other than magnetic tape, usually have fixed-length records but cannot be backspaced.

In addition to being used in conjunction with the APOSN and EAPOS routines for file positioning, LOCF and ELOCF can also be used to return status information (size, position, etc.) on any open file.

Note that when a file is opened, it is automatically positioned at the first record.

LOCF and ELOCF Calls

A call to these routines retrieves status and location information on an open file. The information is obtained from the Data Control Block (DCB) associated with the file. The minimum information returned is the next record number; the other information is optional.

LOCF can only be used with files that are up to 16383 blocks in size and contains up to (2**15)-1 records. ELOCF can be used with files up to 32767 x 128 blocks in size that contain up to (2**31)-1 records.

When calling these routines on a file that has extents, the file size is returned as the number of sectors in the main only. This information is carried in the DCB. Each file extent is the same size as the file main.

```
+-----------------------------------------------------------------+
|                                                                 |
|CALL LOCF(IDCB,IERR,IREC[,IRB][,IOFF][,JSEC][,JLU][,JTY][,JREC])|
|       or      ----  ----   ---   ----    ----   ---   ---   ----|
|CALL ELOCF(IDCB,IERR,IREC[,IRB][,IOFF][,JSEC][,JLU][,JTY][,JREC])|
|               ----  ----   ---   ----    ----   ---   ---   ----|
+-----------------------------------------------------------------+
|                                                                 |
|    IDCB--Data Control Block.  An array of  144+n  words where n |
|          is positive or zero;  previously specified in a create |
|          or open call.                                          |
|                                                                 |
|    IERR--Error return.   A  one-word  variable  that a negative |
|          error code is returned to if the call is unsuccessful. |
|          Zero is returned for successful calls.                 |
|                                                                 |
|    IREC--Next record.   A  one-word  variable  (for LOCF)  or a |
|          double-word variable  (for ELOCF)  that the number of  |
|          the next sequential  record to be accessed is returned |
|          to.                                                    |
|                                                                 |
|    IRB---Next block.  An optional  one-word variable (for LOCF) |
|          or double-word  variable (for ELOCF) that the relative |
|          block number containing the next record to be accessed |
|          is returned to.                                        |
|                                                                 |
|    IOFF--Next word.  An  optional  one-word  variable that the  |
|          offset of the next record in the block is returned to; |
|          not returned for type 0 files.                         |
|                                                                 |
|    JSEC--File size. An optional one-word variable (for LOCF) or |
|          double-word variable (for ELOCF) that the file size in |
|          sectors is returned to; not returned for type 0 files. |
|          JSEC/2 equals number of blocks in file.                |
|                                                                 |
|    JLU---Logical unit.   An optional one-word variable that the |
|          LU to which the file is allocated is returned to.      |
|                                                                 |
|    JTY---File type. An optional one-word variable that the file |
|          type defined when the file was opened is returned to.  |
|                                                                 |
|    JREC--Record length. Optional one-word variable that the rec-|
|          ord length (for type 1 or 2 files)  or the read/write  |
|          code (for type 0 files)  is returned to.  Meaningless  |
|          for type 3 (and above) files.                          |
|                                                                 |
+-----------------------------------------------------------------+
```

COMMENTS:

LOCATION INFORMATION---Together, IREC, IRB, and IOFF provide the
current position within a disc file; they are not set for non-disc
files. The values in these parameters may be passed directly to APOSN
(or EAPOS) to position the file to this location. The values returned
in IRB and IOFF give the exact physical location of the record
pointer in the file. The values of IRB and IOFF are based on a Data
Control Block buffer size of 128 words. If the actual Data Control
Block buffer size is greater than 128, these values are adjusted
automatically by APOSN or EAPOS. Note that ELOCF and EAPOS must be
used together because for each, the record number and relative block
position are double-word variables.

IREC numbers records starting with 1 for the first record, 2 for the
second, and so forth. IREC alone is sufficient to find the location
in type 1 files. IRB numbers blocks starting with 0 for the first
block in the file, 1 for the second, and so forth. If the file is
extendable (type 3 and above), IRB includes extent information and is
specified as:

    (blocks in main file) x (current extent number) + (block number in |
    current extent)

IOFF numbers the words in a block beginning with zero. Since the DCB
packing buffer is assumed to be 128 words the range of IOFF is 0
through 127.

STATUS INFORMATION---JSEC is always an even number, with two 64-word
sectors for each 128-word block in a disc file. It is not applicable
to non-disc files.

JLU is the logical unit to which a file, disc or non-disc, is
allocated.

JTY is the file type of the file; if forced to type 1 at open, then 1
is returned.

JREC as a record length is meaningful for type 2 files only; it is
the length specified at creation. For type 1 files, whether actual
or forced to type 1 at open, the record length is set to 128 on the
first read or write access.

For type 0 files, JREC specifies the read/write access code as
follows:

*   bit 15 = 1   read legal

*   bit   0 = 1   write legal

EXAMPLE 1---Determine the actual location of the record pointer in the open file PROG1 defined in IDCB2:

```
        DIMENSION IDCB2(144)
        .
        .
        .
        CALL LOCF(IDCB2,IERR,IREC,IRB,IOFF)
        IF(IERR .LT. 0) GO TO 900 <-------------- process errors
        .
        .
        .
```

EXAMPLE 2---Open an existing file (DATA) and create a new file (NEW) with the same file type and size; transfer all data from DATA to NEW.

```
        DIMENSION IDATA(272),INEW(272),NDATA(3),NNEW(3)
        DIMENSION IBUF(256),ISIZ(2)
        DATA NDATA/2HDA,2HTA,2H /,NNEW/2HNE,2HW ,2H /
        CALL OPEN(IDATA,IERR,NDATA,0,0,0,272)
        IF(IERR .LE. 0) GO TO 900 <------------ test for error or type 0
10      CALL LOCF(IDATA,IERR,I,I,I,ISIZ(1),I,ITYP,ISIZ(2))
        IF(IERR .LT. 0) GO TO 920
20      ISIZ(1)=ISIZ(1)/2 <--------------- set ISIZ to number of blocks
        CALL CREAT(INEW,IERR,NNEW,ISIZ,ITYP,0,0,256)
        IF(IERR .LT. 0) GO TO 920
30      CALL READF(IDATA,IERR,IBUF,256,L)
        IF(IERR .LT. 0) GO TO 920
40      CALL WRITF(INEW,IERR,IBUF,L)
        IF(IERR .LT. 0) GO TO 920
        IF(L)950,30 <---------- return for next record if not end-of-file
```

PROCESS ERRORS AND CLOSE FILES

```
900     IF(IERR .NE. 0) GO TO 920
        WRITE(1,910) <--------type 0 file cannot be created from program
910     FORMAT("IDATA IS TYPE 0 FILE")
        GO TO 950
920     WRITE(1,930)
930     FORMAT("ERROR ATTEMPTING TO COPY IDATA TO NEW")
950     CALL CLOSE(IDATA,IERR)
1000    CALL CLOSE(INEW,IERR) <-close files following end-of-file or error
        END
```

APOSN and EAPOS Calls

These routines are called to position any disc file to a specific record. For the APOSN routine, the record location can be determined by a prior call to LOCF. ELOCF would be used with the EAPOS call.

APOSN can be used to position files up to (2**15)-1 blocks in length and containing up to (2**15)-1 records. EAPOS can position files up to 32767 x 128 blocks in size and contain up to (2**31)-1 records.

APOSN and EAPOS are intended to position sequential files with variable-length records prior to a read or write request. They may also be used to position random files with fixed-length records (type 1 and 2) but cannot be used to position type 0 files (non-disc files). The POSNT and EPOSN routines described later can be used to position type 0 files.

```
+-----------------------------------------------------------------+
|   CALL APOSN(IDCB,IERR,IREC[,IRB[,IOFF]])                       |
|        or      ---                                              |
|   CALL EAPOS(IDCB,IERR,IREC[,IRB[,IOFF]])                       |
|                ----                                             |
|-----------------------------------------------------------------|
|                                                                 |
|   IDCB--Data Control Block. An array of 144+n words where n is  |
|         positive or zero;   previously specified in an open or  |
|         create call.                                            |
|                                                                 |
|   IERR--Error return.  A one-word variable in which a negative  |
|         error  code is returned  if the call  is unsuccessful;  |
|         zero returned for successful calls.                     |
|                                                                 |
|   IREC--Next record. A one-word variable (for APOSN) a double-  |
|         word variable (for EAPOS) that specifies the number of  |
|         the next sequential  record;  can be  determined by a   |
|         prior call to LOCF or ELOCF.                            |
|                                                                 |
|   IRB---Next block.  An optional one-word variable (for APOSN)  |
|         or double-word variable (for EAPOS) that specifies the  |
|         block number containing the next record; can be deter-  |
|         mined by a prior call to LOCF or ELOCF.   Required for   |
|         files with variable length records;  omitted for files  |
|         with fixed length records.                              |
|                                                                 |
|   IOFF--Next word.  An optional  one-word variable that speci-  |
|         fies  the offset in the block of the next record;  can  |
|         be  determined  by  a  prior  call  to  LOCF or ELOCF.  |
|         Required  for  files  with  variable  length  records;  |
|         omitted for files with fixed length record.            |
|                                                                 |
+-----------------------------------------------------------------+
```

COMMENTS:

PARAMETER CONSIDERATIONS---APOSN and EAPOS assume the value entered for the record address (IREC) is based on a 128-word Data Control Block packing buffer. The routines adjust the values according to the buffer size currently in use if it is greater than 128-words.

IREC must be specified; the value can be determined by the user or obtained from a prior call to LOCF or ELOCF. In addition, IRB and IOFF can be retrieved by LOCF or ELOCF, permitting a file to be reset to a previous record position.

Note that when a file with variable-length records is positioned, the two parameters, IRB and IOFF, must be included.

EXAMPLE 1---Call LOCF to retrieve and save the current position parameters. After reading more of the file, re-position the file associated with IDCB2 to the position whose location was saved.

```
        DIMENSION IDCB2(144)
        .
        .
        .
        CALL LOCF(IDCB2,IERR,IREC,IRB,IOFF) <----- retrieve location at
        IF(IERR .LT. 0) GO TO 900                   start of access
        .
        . <-------------------- read and process records in the file
        .
        CALL APOSN(IDCB2,IERR,IREC,IRB,IOFF) <-- position to previously
        IF(IERR .LT. 0) GO TO 900                  saved location
        .
        .
        .
```

The values of IREC and IOFF retrieved by the call to LOCF are simply passed to the call APOSN by using the same variable names in both calls.

EXAMPLE   2---Position type 2 file defined in Data Control Block IDCB1
to the 6th record in the file and then read next 8 records in
sequence.

```
          DIMENSION IDCB1(144),IBUF(128)
          INTEGER COUNT
          .
          .
          CALL APOSN(IDCB1,IERR,6) <-------------- position file
          IF(IERR .LT. 0) GO TO 900
    10    COUNT=0
          CALL READF(IDCB1,IERR,IBUF1,128,LEN)
          IF(IERR .LT. 0) GO TO 900
    20    IF(LEN .EQ. -1) GO TO 50 <------------- test for end-of-file
          .
          . <--------------------------------- process record
          .
          COUNT=COUNT+1
          IF(COUNT .LT. 8) GO TO 10
          .
          .
          .
```

POSNT and EPOSN Calls

These routines position files relative to the current file position or to a specific record number. They can be used to position all file types.

POSNT can be used to position files containing up to (2**15)-1 records. EPOSN can be used to position files containing up to (2**31)-1 records.

```
+-----------------------------------------------------------------+
|     CALL POSNT(IDCB,IERR,NUR[,IR])                              |
|        or              ----                                     |
|     CALL EPOSN(IDCB,IERR,NUR[,IR])                             |
|                        ----                                     |
|-----------+-----------------------------------------------------+
|                                                                 |
|   IDCB--Data Control Block. An array of 144+n words where n is  |
|         positive or zero; previously specified in a create or   |
|         open call.                                              |
|                                                                 |
|   IERR--Error return.  A one-word variable in which a negative  |
|         error  code is returned  if the call  is unsuccessful;  |
|         zero is returned for successful calls.                  |
|                                                                 |
|   NUR---Record position.  A  one-word variable  (for POSNT) or  |
|         double-word variable (for EPOSN) that specifies:        |
|                                                                 |
|         *  number of records  to position  forward if positive |
|            (IR=0)                                               |
|                                                                 |
|         *  number of records to backspace if negative (IR=0)    |
|                                                                 |
|         *  absolute record number to position to (IR does not   |
|            equal 0)                                             |
|                                                                 |
|         Refer to Table 3-7  for details.                        |
|                                                                 |
|   IR----Position mode flag. An optional one-word variable that  |
|         performs the following function:                        |
|                                                                 |
|         *  if  non-zero,  indicates  that  NUR is to be inter-  |
|            preted as specific record number.                    |
|                                                                 |
|         *  if zero,  indicates that NUR is to be used for rel-  |
|            ative positioning (forward or backspace).            |
|                                                                 |
|         Refer to Table  3-7 for details.                        |
+-----------------------------------------------------------------+
```

COMMENTS:

The relationship between NUR and IR are shown in Table 3-7 below:

Table 3-7. Relationship Between NUR and IR

NUR	IR = 0 OR OMITTED RELATIVE POSITION	IR ≠ 0 ABSOLUTE POSITION
NUR > 0	Position forward number of records specified.	Position to record number specified.
NUR = 0	No operation.	No operation
NUR < 0	Position backward number of records specified.	Error

POSITIONING TYPE 0 FILES---When the file is on a non-disc device, the forward or backward positioning specified by NUR must be legal for the device.

To forward position a type 0 file, the records are read until one less than the specified number of records is read or an EOF is encountered. In every case, an EOF terminates positioning.

When backspacing a type 0 file, the first record backspaced over may be an EOF. If an EOF is encountered not as the first record backspaced over, an error (-12) is returned and the call terminates after forward spacing to position the file immediately after the EOF.

POSITIONING TYPE 1 AND 2 FILES---EPOSN or POSNT may be used to position these file types, however, file positioning for files with fixed length records can be specified in the read or write requests and POSTN or EPOSN may not be necessary.

POSITIONING TYPE 3 AND ABOVE FILES---These files are treated as magnetic tape files. To be correct, a backspace should be issued after an EOF is read and before continuing to write on the file. This action writes the next record over the EOF allowing a correct extension of the file for either disc or magnetic tape files. If the file is on disc, it can be extended without backspacing.

EXAMPLE 1---Current file position immediately follows EOF2. Backspace fourrecords to final position immediately following EOF1 (no error). Note that EOF2 is counted as a record:

```
         EOF1                    EOF2

          | |  1A  |  1B  |  1C  | |  2A  |  2B  |
      ---| |-----|-----|-----| |-----|-----|---
          | |       |       | |     |     |
             ^                    ^
             |                    |
             |                    |
             |                    +-POSITION BEFORE CALL
          +-POSITION AFTER CALL
```

```
      DIMENSION IDCB3(144)
      .
      .
      .
      NUR=-4
      CALL POSNT(ICB3,IERR,NUR)
      IF(IERR .LT. 0) GO TO 900
      .
      .
      .
```

EXAMPLE 2---Current file position follows record 2A. Backspace five records resulting in a -12 error because the end-of-file was not the first record backspaced over. The call is terminated and the file is left positioned immediately after EOF2.

```
        DIMENSION IDCB3(144)
        .
        .
        .
        NUR=-5
        CALL POSNT(IDCB3,IERR,NUR)
        IF(IERR .LT. 0) GO TO 900
        .
        .
        .
900     WRITE(1,910)IERR
910     FORMAT("POSITION ERROR = ",I5)
```

```
        EOF1                    EOF2

        | |  1A  |  1B  |  1C  | |  2A  |  2B  |
    ---| |-----|-----|-----| |-----|-----|---
        | |     |     | |     |     |
                            ^       ^
                            |       |
                            |       +-POSITION BEFORE CALL
                            |
                            +-POSITION AFTER CALL
```

EXAMPLE 3---Position file with Data Control Block JDCB to record number 10:

```
        DIMENSION JDCB(144)
        .
        .
        .
        NUR=10
        IR=1
        CALL POSNT(JDCB,IERR,NUR,IR)
        IF(IERR .LT. 0) GO TO 900
        .
        .
        .
```

RWNDF Call

A call to this routine rewinds a magnetic tape or positions a disc file to the first record in the file.

```
+-------------------------------------------------------------------+
|   CALL RWNDF(IDCB [,IERR])                                        |
|            ----                                                   |
|-------------------------------------------------------------------|
|                                                                   |
|   IDCB--Data Control Block. An array of 144+n words where n is    |
|         positive or zero; previously specified in a create or     |
|         open call.                                                |
|                                                                   |
|   IERR--Error return. An optional one-word variable that a neg-   |
|         ative error code is returned to if the call is unsuc-     |
|         cessful. IERR can be omitted when the A-register is       |
|         checked for error conditions.                             |
|                                                                   |
+-------------------------------------------------------------------+
```

COMMENTS:

If the rewind cannot take place, the call is completed with the file position unchanged; no error is indicated. The rewind will not take place if, for instance, the file being rewound is a paper tape punch, the line printer, or some other device that cannot be backspaced.

EXAMPLE---Position a disc file to the first record:

```
        DIMENSION IDCB2(144)
        .
        .
        .
        CALL RWNDF(IDCB2,IERR)
        IF(IERR .LT. 0) GO TO 900
        .
        .
        .
```

Special Purpose FMP Calls

The Special Purpose FMP calls provide functions not directly related to the standard I/O functions of defining, accessing, and positioning files. The special FMP routines are summarized below:

* FCONT -- Performs control operations on non-disc type 0 files. The control functions are identical to those provided by the I/O control EXEC call, except that the device is identified by a file DCB, instead of a logical unit number.

* FSTAT -- Returns the status of all cartridges in the FMP cartridge directory or, if under session control, returns status of cartridges in users cartridge addressing space.

* IDCBS -- Retrieves the actual size of the DCB buffer for a currently open file; the usable buffer size allocated by FMP is determined by the total file size and the requested buffer size.

* NAMF -- Renames a created file; the file itself is not changed, but can no longer be opened or purged using the old name.

* POST -- Writes the contents of the DCB buffer to the disc; since FMP normally performs this function when the file is closed or the buffer is full, POST is useful mainly to enable modification of records in a file opened for non-exclusive use.

FCONT Call

This routine performs control functions on a peripheral device. The device must have been created and opened as a type 0 file. The call has no effect on other file types. The FCONT call performs the same functions as the RTE I/O CONTROL EXEC call, i.e., rewind, write end-of-file to magnetic tape, etc.

```
+-----------------------------------------------------------------+
|   CALL FCONT(IDCB,IERR,ICON1[,ICON2])                           |
|                   ----          =====                           |
|-----------------------------------------------------------------+
|                                                                 |
|   IDCB---Data Control Block.  An array of  144+n words where n  |
|          is positive or zero;  previously linked  to a type  0  |
|          file by an open call.                                  |
|                                                                 |
|   IERR---Error return.  A  one-word  variable  that a negative  |
|          error code is returned to if the call is unsuccessful. |
|          Zero is returned for successful calls.                 |
|                                                                 |
|   ICON1--Function code.  A  one-word  variable set to an octal  |
|          code defining the function to be performed. See Table  |
|          3-8 for specific codes.                                |
|                                                                 |
|   ICON2--Used  in  conjunction  with  certain  function  codes  |
|          (ICON1).  If  ICON1 = 11 (line spacing), ICON2 speci-  |
|          fies the number of lines to be spaced.  If ICON1 = 27  |
|          (find file) ICON2 specifies a file number. If ICON1=6  |
|          (device status), the status is returned in ICON2.      |
|                                                                 |
+-----------------------------------------------------------------+
```

COMMENTS:

FUNCTION CODE---Bits 6 through 10 of ICON1 are used for the function code.

```
 15 14 13 12 11 10    9    8    7    6    5    4    3    2    1    0
+--+--+--+--+--+--+---+---+---+---+---+---+---+---+---+---+---+
| 0| 0  0  0| 0           |         | 0   0   0| 0   0   0 |
+--+--+--+--+--+--+---+---+---+---+---+---+---+---+---+---+---+
            +                       +
            |                       |
            +-------+---------+
                    |
              FUNCTION CODE
```

The function codes are defined in Table 3-8.

ICON1 AND ICON2---If the function code is 11 octal, then FCONT expects a value in ICON2. This value controls output line spacing on the line printer or a keyboard display device:

   0 to suppress line spacing on the next line
 >0 to indicate the number of lines to space before the next line
 <0 to page eject on line printer; space specified lines on keyboard device.

If the function code is 27 octal, then FCONT excepts a value in ICON2. This value declares the absolute file number to be located, in the range 1 through 255.

Table 3-8. FCONT Function Codes

FUNCTION CODE (OCTAL)	FUNCTION	DEVICE
00	Unused	
01	Write end-of-file	Magnetic tape Cartridge Tape Unit
02	Backspace one record	Magnetic tape Cartridge Tape Unit
03	Forward space one record	Magnetic tape Cartridge Tape Unit
04	Rewind	Magnetic tape Cartridge Tape Unit
05	Rewind standby Rewind	Magnetic tape Cartridge Tape Unit
06	Actual device status	Magnetic tape Cartridge Tape Unit
07	Set end-of-tape	Paper tape/TTY
10	Generate leader	Paper tape/TTY
	Write end-of-file if not just previously written or not at load point	Cartridge Tape Unit
*11	List output line spacing	Line printer
12	Write 3 inch interrecord gap	Magnetic tape
13	Forward space on file	Magnetic tape Cartridge Tape Unit
14	Backspace one file	Magnetic tape Cartridge Tape Unit
15	Conditional top-of-form	Line printer/display device

Table 3-8. FCONT Function Codes (Continued)

FUNCTION CODE (OCTAL)	FUNCTION	DEVICE
20	Enable terminal — allows terminal to schedule its program when any key is struck	Codes 20-27 are defined for a keyboard terminal (DVR00). Refer to the DVR00 manual 29029-60001 for other uses.
21	Disable terminal — inhibits scheduling of terminal program	
22	Set time-out — sets new time-out interval	
23	Ignore all further requests until: • the request queue is empty • an input request is received • a restore control request is received	
24	Restore output processing (this request is usually not needed)	
26	Write end-of-data	Cartridge Tape Unit
** 27	Locate file number	Cartridge Tape Unit

*When function code 11 is specified in ICON1, then ICON2 must be included in the parameter list to specify the particular line spacing.

**When function code 27 is specified in ICON1, then ICON2 must be included in the parameter list to specify the particular file number.

EXAMPLE 1---Backspace one record on file (MT) previously created and opened with Data Control Block MTDCB.

```
DIMENSION MTBUF(144)
   .
   .
   .
ICONT=200B  <------------------- function code 2
CALL FCONT(MTBUF,IERR,ICONT)
IF(IERR .LT. 0) GO TO 900
   .
   .
   .
```

EXAMPLE 2---Page eject to line printer   created and opened as a file
with Data Control Block LPDCB.

```
DIMENSION LPDCB(144)
.
.
.
ICONT=1100B    page eject on line printer (would
LPSP=-1        space 1 line on keyboard device)
CALL FCONT(LPDCB,IERR,ICONT,LPSP)
IF(IERR .LT. 0) GO TO 900
.
.
```

FSTAT Call

This routine returns the status of mounted cartridges in the
cartridge directory. When under session control, ISTAT will contain
information about only those discs mounted to the session's SCB and
system discs. The caller may get information on all discs mounted to
the system by specifying IOP non-zero. FSTAT gives the option of
providing the information in one of two formats. The first format
provides for each cartridge the logical unit number, the last FMP
track, the cartridge reference number, and if a program has locked
the cartridge, its ID segment address. The second format provides
the above information plus the identifier of who mounted the
cartridge.

Note that in the second format, a locking program is represented by
its ID segment number instead of its ID segment address.

```
+-----------------------------------------------------------------+
|   CALL FSTAT(ISTAT,[ILEN],[IFORM],[IOP],[IADD])                  |
|-----------------------------------------------------------------|
|                                                                 |
|   ISTAT--Status buffer.  An array that the status of the cart-  |
|         ridges is returned to (see Table 3-9 and Table 3-10).   |
|                                                                 |
|   ILEN---Buffer length.  An optional  one-word  variable  that  |
|         specifies the length in words of  ISTAT.  If omitted,   |
|         ISTAT is assumed to be 125 words.                       |
|                                                                 |
|   IFORM--Format.  An optional one-word variable that specifies  |
|         whether  Format I or Format II will be used to return   |
|         the cartridge directory (see Table 3-9 and Table 3-10). |
|         If zero, Format I is used;  if non-zero, Format II is   |
|         used.                                                   |
|                                                                 |
|   IOP----Option.  An optional one-word variable that specifies  |
|         the  type  of  cartridges  that  information is to be   |
|         returned about.  If:                                    |
|                                                                 |
|         IOP=1; the status on all cartridges mounted to the      |
|         system is returned in ISTAT;  whether under session     |
|         contol or not.                                          |
|                                                                 |
|         IOP=0 and  under  session control;   the status of      |
|         private and group  cartridges mounted to the session    |
|         and system cartridges are returned (in that order).     |
|                                                                 |
|         IOP=0 and  not  under  session control;  status on      |
|         system cartridges  and  non-session  cartridges  are    |
|         returned (in that order).                               |
|                                                                 |
|   IADD---System sets IADD  to non-zero if not all the cart-     |
|         ridge list could be returned in ISTAT  (e.g.,  ISTAT    |
|         not large enough). IADD is set to 0 if all cartridges   |
|         were returned.                                          |
|                                                                 |
+-----------------------------------------------------------------+
```

COMMENTS:

The two formats that can be used with the FSTAT call are shown in
Table 3-9 and Table 3-10, below:

Table 3-9. ISTAT Format I

ISTAT		
WORD	**CONTENTS**	**CARTRIDGE**
1 2 3 4	Logical Unit Number Last FMP track Cartridge Reference Number Lock Word	First cartridge
5 6 7 8	Logical Unit Number Last FMP track Cartridge Reference Number Lock Word	Second cartridge
9 • • • •	Logical Unit Number • • • •	• • • •
	0 no more discs	

where: Lock word is ID segment address of locking program or 0 (not locked).

Table 3-10. ISTAT Format II

ISTAT		
WORD	**CONTENTS**	**CARTRIDGE**
1 2 3 4	Lock word Logical unit # Last FMP track Cartridge Reference Number ID	First cartridge
5 6 7 8	Lock word Logical unit # Last FMP track Cartridge Reference Number ID	Second cartridge
9 . . .	Lock word Logical unit # . . .	. . .
	0 no more discs	

where: Lock word is the offset of the ID segment in the Keyword Table or 0 (not locked).

ID identifies who mounted the cartridge.

EXAMPLE---Find the cartridge reference number of the cartridge associated with logical unit 13 in order to create file XX on that cartridge. Use Format II and only search cartridges mounted to the current session.

```
      DIMENSION idcb(144), istat(253), isize(2)
      DATA isize /100,128/
      CALL FSTAT (istat,253,1,0)        ! Get cartridge directory
      DO 10 i = 1,248,4                 ! Check for LU 13 (bits 0-5)
          IF (iand(istat(i),77B) .eq. 13) GOTO 100

10    CONTINUE
          :
      <error, LU not found>
          :
100   icr = istat(i+2)                 ! Get the CRN
      CALL CREAT(idcb, ierr, 6HXX, isize, 3, 0, icr) ! Create file
          :
      <etc>
```

IDCBS Call

This function returns the number of words in a Data Control Block actually used by the File Management Package for data transfer and file control.

```
+--------------------------------------------------------------+
|    ISIZE=IDCBS(IDCB)                                         |
+--------------------------------------------------------------+
|                                                              |
|    IDCB--Data Control Block. An array of 144+n words where n is |
|          positive or zero; previously specified in a create or |
|          open call.                                          |
+--------------------------------------------------------------+
```

COMMENTS:

When a Data Control Block larger than 144 words is specified for the file at open or creation, the File Management Package may not use the entire DCB buffer area. The actual size used depends on the file size as well as the requested buffer size. This call returns the actual Data Control Block size; the buffer used plus 16 control words. See the description of the parameter at the beginning of this section for more details.

EXAMPLE--- A file has been opened using the Data Control Block MBUF with a size of 5000 words. Use IDCBS to determine how much of MBUF is being used by FMP. If 144 or more words words remain, create KFIL using the remainder of MBUF as a Data Control Block.

```
      DIMENSION NAM(3),NAM1(3)
      DIMENSION ISZ1(2),MBUF(5000)
      DATA   NAM/2HMY,2HF1,2HLE/,NAM1/2HKE,2HIL,2H  /
      ICR=-14
      CALL OPEN (MBUF,IERR,NAM,0,ICR,5000-144)
      IF(IERR.LT.0) GO TO 150
      ISZ1(1)=5
      ITYPE=3
      ISIZE=IDCBS(MBUF)
      CALL CREAT(MBUF(ISIZE),IERR,NAM1,ISZ1,ITYPE,0,ICR,5000-ISIZE)
      IF(IERR.LT.0) GO TO 150
```

NAMF Call

This routine renames an existing file. If the code was created with a security code, this code must be specified. If the file is open, it is closed and then returned.

```
+--------------------------------------------------------------------+
|    CALL NAMF(IDCB,IERR,INAM,MNAM[,ISC][,ICR])                      |
|                 ----                                               |
|--------------------------------------------------------------------+
|                                                                    |
|    IDCB---Data Control Block.  An  array of 144+n words when n     |
|           is positive or zero.                                     |
|                                                                    |
|    IERR---Error return.  A one-word  variable that a negative      |
|           error code is returned to if the call was unsuccess-     |
|           ful.  Zero is returned for successful calls.             |
|                                                                    |
|    INAM---File name.  A three-word  array containing the ASCII     |
|           file name of original file.                             |
|                                                                    |
|    MNAM---File name.   A three-word array containing the ASCII     |
|           file name to replace INAM.                              |
|                                                                    |
|    ISC----Security code. An optional one-word variable (0 thru     |
|           +/-32767).  Omitted  if INAM  was  created without a     |
|           security coae (or 0); must match if INAM has security    |
|           code.                                                   |
|                                                                    |
|    ICR----Cartridge reference.  An  optional one-word variable     |
|           (0 thru 32767)  that  specifies  the  cartridge that     |
|           INAM  resides on.  If  omitted or zero,  first file      |
|           found that matches the original file name  (INAM) is     |
|           renamed.                                                |
|                                                                    |
+--------------------------------------------------------------------+
```

COMMENTS:

EXAMPLE---Rename file PROG1 as MYFILE. Since a cartridge reference is specified,   the system will look for PROG1 on logical unit 14 only. No security code is needed.

```
    DIMENSION IDCB2(144),NAME(3),NNAME(3)
    DATA NAME/2HPR,2HOG,2H1 /,NNAME/2HMY,2HFI,2HLE/
    ICR=-14
    CALL NAMF(IDCB2,IERR,NAME,NNAME,0,ICR)
    IF(IERR .LT. 0) GO TO 900
    .
    .
    .
```

POST Call

This routine is called to post (write) the contents of the Data Control Block buffer to a disc file (type 2 or above). Normally, this is done by the system when the buffer is full or the file is closed. POST provides direct control over the physical write to disc, assures that the next read is from disc, and can be used in a special case to save records in a file opened for non-exclusive use.

```
+-------------------------------------------------------------------+
|    CALL POST(IDCB,[IERR])                                          |
|             ----                                                  |
|-------------------------------------------------------------------|
|                                                                   |
|    IDCB---Data Control Block.  An array of 144+n words where n     |
|           is positive or zero;  previously specified in a open     |
|           or create call.                                          |
|                                                                   |
|    IERR---Error return.  An optional  one-word variable that a     |
|           negative  error  code is  returned to if the call is     |
|           unsuccessful. Zero is returned for successful calls.     |
+-------------------------------------------------------------------+
```

COMMENTS:

This call is ignored for files of type 0 or 1 since transfers to these files are always direct (bypass the Data Control Block buffer).

USING POST FOR MODIFICATION---In conjunction with the RTE Resource Numbering call RNRQ (see Chapter 6), POST allows several programs to modify a file without requiring an exclusive open. RNRQ is used to lock the file for exclusive use of the calling program, and POST then clears the DCB buffer before modifying the record and again after modification.

The sequence to be followed is:

1. Open the file.

2. Read the file to retrieve the resource number (RN).

3. Call POST to clear the DCB buffer (no data is posted since none was written). This insures that the next read accesses the disc.

4. Call RNRQ to lock the file for exclusive use of the calling program.

5. Call READF to read the record to be modified.

6. Modify the record and call WRITF to write the record.

7. Call POST to transfer the updated record to the file from the DBC buffer.

8. Call RNRQ to unlock the file for use by other programs.

It is possible that WRITF in step 6 above causes the buffer to be posted to the disc, but POST should be called to insure the transfer.

EXAMPLE---Assume the resource number is in location IRN; modify record number 5 in a type 2 file opened non-exclusively.

```
          DIMENSION IDCB(144),IBUF(128)
30        CALL POST(IDCB,IERR) <-------------- clear DCB buffer
          IF(IERR .LT. 0) GO TO 150
          ICODE=1B <------------------- set code to local lock
40        CALL RNRQ(ICODE,IRN,ISTAT) <--lock file for exclusive use
          IF(ISTAT .NE. 2) GO TO 150      of program
50        IL=80
          CALL READF(IDCB,IERR,IBUF,IL,LEN,5) <----- read record 5
          IF(IERR .LT. 0) GO TO 150
          IF(LEN .GT. 80) GO TO 150
          .
          . <---------------------------------- modify record in IBUF
          .
60        CALL WRITF(IDCB,IERR,IBUF,IL,5) <--write modified record
          IF(IERR .LT. 0) GO TO 150
70        CALL POST(IDCB,IERR) <----------- clear buffer again
          IF(IERR .LT. 0) GO TO 150
80        ICODE=4B <----------------------- set code to unlock file
          .
          .
          .
          CALL RNRQ(ICODE,IRN,ISTAT)
          .
          .
          .
150       process errors here and unlock Resource Number
          .
          .
          .
```

Examples Using FMP Calls

The following short sample program creates up to ten data files from one control file. It then reads and checks the data in each data file against control information in the control file.

Program FMPEX

```
C
      PROGRAM FMPEX,3,99
C
      DIMENSION LU(5),IBUF(40),IPBUF(33),IHEADR(40),IREC(10)
      DIMENSION IRB(10),IOFF(10),IOPEN(10),ILOCF(10),ISCHK(16),IREG(2)
      DIMENSION IDCB(144),JDCB(1440),DATA(5)
C
      EQUIVALENCE (IA,IREG),(IB,IREG(2)),(X,IREG)
      EQUIVALENCE (FNAM,IPBUF(2)),(ISECU,IPBUF(6)),(ICR,IPBUF(10))
      EQUIVALENCE (ITYPE,IPBUF(14)),(ISIZE,IPBUF(18)),(IRLNT,IPBUF(22))
C
      DATA ISCHK/0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15/
C
C
C     THIS PROGRAM IS INTENDED TO MANAGE THE DATA-FILE SCHEME FOR A
C     DATA ACQUISITION SYSTEM. NOT INCLUDED HERE IS THE
C     CODE THAT ACTUALLY OBTAINS THE DATA, SINCE THIS IS
C     SIMPLY AN EXAMPLE OF FILE USAGE.
C
C     THE OPERATION OF THIS ROUTINE FOLLOWS:
C
C     A TYPE 4 'CONTROL' FILE IS CREATED BY THE TEST OPERATOR THAT
C     CONTAINS INFORMATION ABOUT THE TESTS TO BE PERFORMED AND THE
C     FILES INTO WHICH THE DATA MUST GO. THIS ROUTINE REQUESTS
C     THE NAME OF THE FILE, OPENS IT, AND USES INFORMATION
C     IN IT TO CREATE AND USE OTHER DATA FILES.
C
C     THE FORMAT OF THIS CONTROL FILE IS:
C
C         RECORD #          CONTENTS
C
C            1              TEST LABEL : A HEADER FOR DATA FILES
C            2              'NAMR' OF DATA FILE #1
C            3              'NAMR' OF DATA FILE #2
C            4                ETC.
C                          END WITH "/E" AS FILE NAMR
C            N              OPERATOR SECURITY CODE (E.G. FOR TEST ACCESS)
C                              AS CHECKED AGAINST THE 'ISCHK' VALUES, ABOVE
C
C         NOTE: ANY LINE BEGINNING WITH "**" IS CONSIDERED A COMMENT
C
C
C     THE DATA, AS IT COMES IN, HAS ASSOCIATED WITH IT A
C     'DATA CODE' INDICATING THE DISPOSITION OF THE DATA. THE
C     CODES ARE LISTED BELOW:
```

```
C
C         CODE          FUNCTION
C          0            DATA O.K.    WRITE IT TO THE APPROPRIATE FILE
C          1            THIS DATA IS TO BE IGNORED
C          2            IGNORE THE PREVIOUS DATA ITEM (ERASE IT FROM THE FILE)
C          3            REMEMBER THE FILE POINTERS, WE MAY WANT TO GO BACK
C          4            REWIND THE DATA FILE - START OVER AGAIN
C          5            GO BACK TO 'REMEMBERED' POINT IN DATA FILE
C          6             *
C          7             *    SAME AS 0, ABOVE
C          8             *
C          9             *
C         10            END OF TEST - CLOSE OF THE DATA FILE
C
C
C         NOTE: DATA IS WRITTEN INTO THE FILES IN ASCII
C
C
C
C
C
C
C
C     1)  SET UP TERMINAL LU
C
          LU=1
          ILU=LU+400B
C
C     2)  ASK OPERATOR FOR CONTROL FILE NAME
C           USE ROUTINE 'COLON' TO REPLACE COLONS IN FILE 'NAMR'
C           WITH COMMAS SO WE CAN USE SYSTEM PARSE ROUTINE 'PARSE'
C
10        WRITE(LU,100)
100       FORMAT("/FMPEX: ENTER CONTROL FILE 'NAMR': <--")
          X=REIO(1,ILU,IBUF,20)
          CALL COLON(IBUF,IB)
          CALL PARSE(IBUF,IB*2,IPBUF)
          CALL OPEN(IDCB,IERR,FNAM,0,ISECU,ICR)
          IF(IERR.GE.0) GO TO 12
          WRITE(LU,101)IERR
101       FORMAT("/FMPEX: FILE ERROR"I6".   TRY AGAIN!"/)
          GO TO 10
C
C     3)  READ CONTROL FILE AND:
C           CREATE EACH DATA FILE
C           WRITE DATA FILE HEADER FROM CONTROL FILE
C           CHECK OPERATOR SECURITY CODE
C
12        N=1
          CALL READF(IDCB,IERR,IBUF,20)
14        CALL READF(IDCB,IERR,IBUF,16,IB)
          IF(IERR.LT.0) GO TO 80
          IF(IB.NE.-1) GO TO 13
          WRITE(LU,112)
112       FORMAT("/FMPEX: FOUND EOF IN CONTROL FILE. ERROR!")
```

```
               GO TO 80
C
13             IF(IBUF.EQ.2H**) GO TO 14
               IF(IBUF.EQ.2H/E) GO TO 16
               IF(N.GT.10) GO TO 14
C
               CALL COLON(IBUF,IB)
               CALL PARSE(IBUF,IB*2,IPBUF)
               IPBUF(19)=IRLNT
               CALL CREAT(JDCB(144*(N-1)+1),IERR,FNAM,ISIZE,4,ISECU,ICR)
               IF(IERR.LT.0) GO TO 80
               N=N+1
               GO TO 14
C
16             N=N-1
               CALL READF(IDCB,IERR,IBUF,20,IL)
               IBUF(IL+1)=2H,,
               CALL CODE
               READ(IBUF,*)ISECU
               DO 17 I=1,16
               IF(ISECU.EQ.ISCHK(I)) GO TO 18
17             CONTINUE
               WRITE(LU,117)ISECU
117            FORMAT("/FMPEX: "I6" IS NOT A VALID OPERATOR SECURITY CODE!")
               GO TO 82
C
18             CALL RWNDF(IDCB,IERR)
               IF(IERR.LT.0) GO TO 80
               CALL READF(IDCB,IERR,IHEADR,40,IB)
               IF(IERR.LT.0) GO TO 80
               DO 19 I=1,N
               CALL WRITF(JDCB(144*(I-1)+1),IERR,IHEADR,IB)
               IF(IERR.LT.0) GO TO 80
19             CONTINUE
C
C
C    4)   THIS SECTION WOULD CONTAIN SOME DATA ACQUISITION ROUTINES.
C         FOR THE PURPOSE OF THIS EXAMPLE WE WILL SIMPLY INPUT SOME
C         DATA FROM A TELETYPE.
C
C         THE DATA WILL CONSIST OF 7 ITEMS:
C             ITEM        VALUE
C              1          DATA FILE TO BE USED FOR STORAGE
C              2          DISPOSITION FLAG, AS DISCUSSED ABOVE
C            3 THRU 7     DATA VALUES
C
C         IN A MORE GENERALIZED CASE, THIS SECTION WOULD ALSO
C         INCLUDE DATA PROCESSING OR CORRECTION, DATA CHECKING, AND
C         DETERMINATION OF DISPOSITION FLAG VALUE.
C
C
20             WRITE(LU,120)
120            FORMAT("/FMPEX: ENTER FILE #,FLAG, AND 5 DATA ITEMS: <--")
               READ(LU,*)I,IFLG,(DATA(J),J=1,5)
```

```
C
C          CHECK DISPOSITION FOR 'IGNORE', ERROR IN FLAG, OR END OF TEST
C

           IF(IFLG.EQ.1) GO TO 20
           IF(IFLG.EQ.10) GO TO 30
           IF((IFLG.LT.1).OR.(IFLG.GT.10)) GO TO 38
C
C    RE-FORMAT THE DATA TO THE FORMAT OF THE DATA FILE
C
           CALL CODE
           WRITE(IBUF,122)I,IFLG,(DATA(J),J=1,5)
122        FORMAT(I4", "I4,5(", ",F8,2))
C
C          DATA IS MERELY WRITTEN TO THE FILE FOR IFLG=0,6-9
C
           IF((I.NE.0).AND.(I.LE.5))GO TO 22
21         CALL WRITF(JDCB(144*(I-1)+1),IERR,IBUF,30)
           IF(IERR.LT.0)GO TO 80
           GO TO 20
C
C    WE BACKSPACE THE FILE TO OVER-WRITE PREVIOUS DATA IF 'IFLG' = 2
C
22         IF(I.NE.2)GO TO 24
           CALL POSNT(JDCB(144*(I-1)++1),IERR,-1)
           IF(IERR.LT.0)GO TO 80
           GO TO 21
C
C          'REMEMBER' THE FILE POINTERS BEFORE WRITING IF 'IFLG' = 3
C          A 10-WORD ARRAY IS USED IN ORDER TO MAINTAIN ONE SET OF POINTERS
C          FOR EACH DATA FILE
C
24         IF(I.NE.3)GO TO 26
           CALL LOCF(JDCB(144*(I-1)+1),IERR,IREC(I),IRB(I),IOFF(I))
           IF(IERR.LT.0)GO TO 80
           GO TO 21
C
C    IF !IFLG! = 4, WE START THE DATA FILE ALL OVER AGAIN... REWIND
C
26         IF(I.NE.4)GO TO 28
           CALL RWNDF(JDCB(144*(I-1)+1),IERR)
           IF(IERR.LT.0)GO TO 80
           GO TO 21
C
C    THAT DATA TAKEN SINCE WE NOTED THE FILE POINTERS IS NOW DETERMINED
C    TO BE OF NO VALUE. RESTORE THE DATA FILE TO THAT POSITION AND
C    OVERWRITE OLD DATA WITH NEW.
C
28         IF(IERR.NE.5)GO TO 35
           CALL APOSN(JDCB(144*(I-1)+1),IERR,IREC(I),IRB(I),IOFF(I))
           IF(IERR.LT.0)GO TO 80
           GO TO 21
C
30         CALL CLOSE(JDCB(144*(I-1)+1),IERR)
           IF(IERR.LT.0)GO TO 80
```

```
          N=N-1
          IF(N.EQ.0)90,20
C
35        WRITE(LU,135)IFLG
135       FORMAT("/FMPEX: "I6" IS AN ILLEGAL DISPOSITION FLAG!")
          GO TO 20
C
38        WRITE(LU,138)I
138       FORMAT("/FMPEX: "I6" IS AN ILLEGAL DATA FILE NUMBER!")
          GO TO 20
C
C
C     FILE ERROR SECTION
C
80        WRITE(LU,180)IERR
180       FORMAT("/FMPEX: FILE ERROR"I6". ABORT!")
          DO 82 I=1,6
          CALL CLOSE(JDCB(144*(I-1)+1),IERR)
82        CONTINUE
C
          CALL CLOSE(IDCB,IERR)
C
C
C     END
C
90        WRITE(LU,190)
190       FORMAT("/FMPEX: DONE!"/)
C
          END

          END$
```

Subroutine COLON

COLON is used to convert the colons (:) in namr to commas for the PARSE
routine. PARSE is in the RTE system library.

```
C
          SUBROUTINE COLON(IBUF,N)
C
          DIMENSION IBUF(2)
C
C     THIS SUBROUTINE ACCEPTS A BUFFER CONTAINING AN ASCII STRING
C     (USUALLY AN FMP 'NAMR') AND CONVERTS ALL COLONS (:) TO COMMAS (,)
C     THIS ALLOWS THE USE OF THE SYSTEM PARSE ROUTINE (WHICH REQUIRES
C     COMMAS AS DELIMITERS) TO PARSE OUT FMP 'NAMR'S WHICH USE COLONS
C     AS DELIMITERS
C
C
C
          DO 10 I=1,N
            IF(IAND(IBUF(I),77600B).NE.35000B)GO TO 5
            IBUF(I)=IOR(IAND(IBUF(I),177B),26000B)
```

```
5         IF(IAND(IBUF(I),177B).NE.72B)GO TO 10
          IBUF(I)=IOR(IAND(IBUF(I),77600B),54B)
10        CONTINUE
C
          RETURN
C
          END

          END$
```

FMP Error Codes

Table 3-11. FMP Error Codes

ERROR	MESSAGE	MEANING & CORRECTIVE ACTION
000	(no error)	none
-001	DISC ERROR	The disc is down: try again and then report it to the system manager of facility.
-002	DUPLICATE FILE NAME	A file already exists with specified name: repeat with new name or purge existing file.
-003	BACKSPACE ILLEGAL	Attempt was made to backspace a device (or type 0 file) that cannot be back-spaced: check device type.
-004	RECORD SIZE OF TYPE 2 FILE IS 0 OR UNDEFINED.	Attempt was made to create a type 2 file without specifying record size or specifying it to be zero. Check size parameter.
-005	RECORD LENGTH ILLEGAL	Attempt to read or position to a not written, or on update to write an illegal record length; check position or size parameters.
-006	FILE NOT FOUND	Attempt to access a file that cannot be found; check the file name.
-007	ILLEGAL SECURITY CODE OR ILLEGAL WRITE ON LU 2 OR 3	Attempt to access a file with no SC or the wrong code; use correct code or do not access file. Or attempt to write on LU2 or 3.
-008	FILE OPEN OR LOCK REJECTED	Attempt to open file already open exclusively or open to eight programs or cartridge containing file is locked: use CL or DL to locate lock: if file being packed, check if spool shut down
-009	ATTEMPT TO USE APOSN OR FORCE TO 1 A TYPE 0 FILE	Type 0 files cannot be positioned APOSN or be forced to type 1; check file type.
-010	NOT ENOUGH PARAMETERS	Required parameters omitted from call enter the parameters.

Table 3-11. FMP Error Codes (Continued)

ERROR	MESSAGE	MEANING AND CORRECTIVE ACTION
-011	DCB NOT OPEN	Attempt to access an unopened DCB; use CREAT, or OPEN to open DCB; check for errors.
-012	EOF OR SOF ERROR	Attempt to read or write or position beyond the file boundaries; check record position parameters, result depends on file type and call.
-013	CARTRIDGE LOCKED	Cartridge is locked;initialize it if needed; otherwise keep trying.
-014	DIRECTORY FULL	No more room in file directory; purge files and pack directory if possible; or try another cartridge.
-015	ILLEGAL NAME	File name does not conform to syntax rules; correct name.
-016	ILLEGAL TYPE OR SIZE=0	Wrong type code supplied; attempt to create or purge type 0 file or create 0-length file; check size and type parameters.
-017	ILLEGAL READ/ WRITE ON TYPE 0 FILE	Attempt to read/write or position type 0 file that does not support the operation; check file parameters, from FMGR check namr.
-018	ILLEGAL LU	Attempted to access an LU that is not assigned to the system.
-019	ILLEGAL ACCESS ON A SYSTEM DISC	Attempted to write the LU 2 or LU 3 while under session control.
-020	ILLEGAL ACCESS LU	LU number specified in SU or CS command or for spool file set up through SPOP11 call unit number. It may not be a disc LU or LU1. Correct command entry or the set up buffer for SPOPN call.
-021	ILLEGAL DESTIN- ATION LU	LU number specified must be a logical unit number that was allocated by GASP

Table 3-11. FMP Error Codes (Continued)

ERROR CODE	MESSAGE	MEANING & CORRECTIVE ACTION
-022	NO AVAILABLE SPOOL LUs	No spool logical units currently available. Re-run after spool LU becomes available.
-023	NO AVAILABLE SPOOL FILES	No spool files currently available. Re-run after spool file becomes available.
-024	NO MORE BATCH SWITCHES	LU switch table full; size of switch table created at system generation not adequate.
-025	NO SPLCON ROOM	SPLCON full. May occur when spool system is competing with programs running outside batch and using their own spooling files and SMP.
-026	QUEUE FULL OR MAX PENDING SPOOLS EXCEEDED	Self-explanatory. Re-run when space becomes available.
-27	SPOOL SYSTEM IS NOT INITIALIZED OR SMP PROGRAM NOT FOUND	Self explanatory. Initialize spool system using GASP program or load SMP program.
-30	VALUE TOO LARGE FOR PARAMETER	The value supplied in the parameter is beyond the defined range of information to be returned in the parameter is too large to fit in a single word.
-032	CARTRIDGE NOT FOUND	Attempt to access a cartridge that cannot be found. Check the cartridge number.
-033	NO ROOM ON CARTRIDGE	Attempt to create a file on a cartridge that has no more room. Try another cartridge or decrease file size.
-034	DISC ALREADY MOUNTED	Attempt to mount a disc that is already mounted on the CL.
-035	ALREADY 63 DISCS MOUNTED ON CL	Attempt to mount a disc when there are already 63 mounted.

Table 3-11. FMP Error Codes (Continued)

ERROR CODE	MESSAGE	MEANING & CORRECTIVE ACTION
-36	LOCK ERROR ON DEVICE	Call to OPENF caused an attempted lock on a device and that lock was unsuccessful.
-38	ILLEGAL SCRATCH FILE NUMBER	Call to CRETS with an invalid File number. Range of 0-99.
-39	SPOOL LU NOT MAPPED TO THE SPOOL DRIVER	Spool LU must point to a spool EQT. Switch all spool LU's to point to spool EQT's.
-040	DISC NOT IN SST	Accessing a disc LU that does not have an entry in your sessions SST.
-41	NO ROOM IN SST	No spare entries in session switch table. Free up an entry using SL,LU,-
-46	GREATER THAN 255 EXTENTS	Attempt to create extent 256. Make file size of main larger.
-47	NO AVAILABLE SESSION LU FOR SPOOL FILE	Attempt by SMP to allocate a session LU less than 64 that is not already used in the SST. Use :SL,LU,- to release a session LU in the SST.
-48	SPOOL NOT INITIAL-IZED OR SMP CANNOT BE SCHEDULED	SMP does not exist; either it was purged or was not loaded properly into a large enough partition.
-99	DIRECTORY MANAGER EXEC REQUEST ABORTED	An EXEC request made by D.RTR was aborted. Make sure all discs are up. Notify system manager.
-101	ILLEGAL PARAMETER IN D.RTR CALL	Possible operator error; recheck previous entries for illegal or mis-placed parameters.
-102	ILLEGAL D.RTR CALL SEQUENCE	Lock not requested first or file not opened exclusively; possible operator error such as removal of cartridge without DC command.

Chapter 4
Program Segmentation

Introduction

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 4-1). 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.
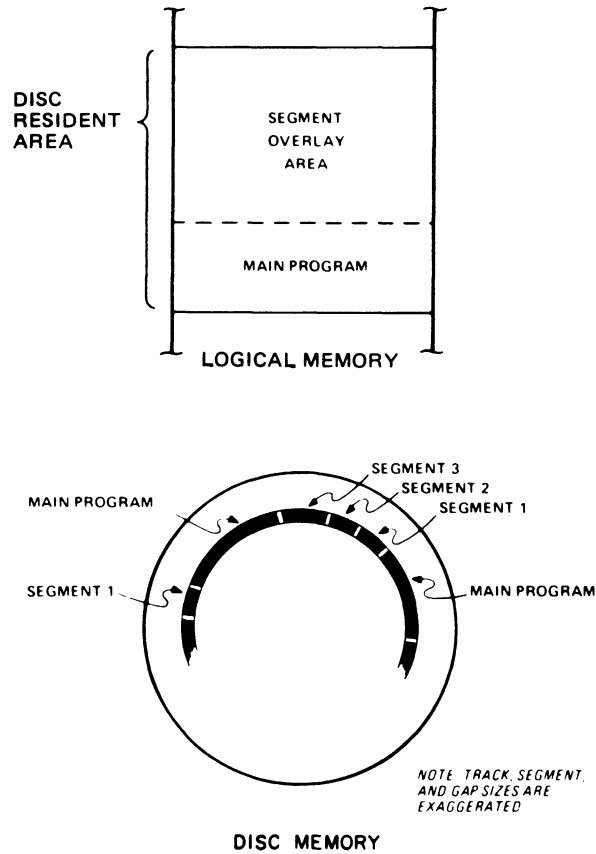
Figure 4-1. Segmented Programs

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 largest segment.

RTE FORTRAN-IV Segmentation

For RTE FORTRAN-IV programs to be segmented, certain conventions must
be 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.

Return from a segment to the main program is effected through the FORTRAN ASSIGN statement which may be passed through COMMON or parameter passage. Since it is possible to place several return addresses in COMMON, segments may return to any desired location in MAIN and MAIN may call the same segments as often as desired.

For example:

```
      PROGRAM MAIN (4)
      COMMON INSTN
       :
      ASSIGN 100 TO INSTN
      CALL EXEC (8,6HSEGM01)
  100 return instruction
       :
      END

      PROGRAM SEGM01 (5)
      COMMON INSTN
       :
      GO TO INSTN
      END
```

FORTRAN-IV Segmentation Example

The following example consists of a main program with an appended subroutine and two segments. Communication between the main program (MAIN) and the first segment (SEG1) is implemented via local COMMON. Communication between SEG1 and the second segment (SEG2) is implemented via parameter passage; a feature of the EXEC 8 segment load call (see Chapter 2, "Executive Communication".). SEG2 calls a user-written subroutine (SUBR) appended to the main program.

```
PROGRAM MAIN(3)<---main has to be type 2,3,or 4
DIMENSION INAM(3)
COMMON ICOM(2)
DATA INAM/2HSE,2HG1,2Hbb/
     .
     .
N1=value<----------place value in N1 &N2 that are
N2=value                to be passed to SUBR.
     .
     .
CALL SUBR(N1,N2,J1,J2)
     .
     .
ICOM(1)=J1<--------place values returned from
ICOM(2)=J2          SUBR into COMMON.
     .
     .
CALL EXEC(8,INAM)<--------Load segment (SEG1) and
                          transfer control to it.
END
SUBROUTINE SUBR(I1,I2,I3,I4)
     .
     .
process I1 and I2 and place
   results in I3 and I4, respectively.
     .
     .
RETURN
END

PROGRAM SEG1(5)<----segment must be type 5.
DIMENSION INAM(3)
COMMON ICOM(2)
DATA INAM/2HSE,2HG2,2Hbb/
     .
     .
process values in ICOM and place
   results in IOP1 and IOP2
     .
     .
CALL EXEC(8,INAM,IOP1,IOP2)<-----load segment (SEG2) and
END                              transfer control to it,
                                 passing parameters.
```

```
PROGRAM SEG2(5)<-------------------type 5 program
DIMENSION IPAR(5)
CALL RMPAR(IPAR)<---------------obtain values passed by SEG1.
IOP1=IPAR(1)
IOP2=IPAR(2)
    .
    .
CALL SUBR(IOP1,IOP2,IRT3,IRT4)<---pass parameters obtained from
CONTINUE                           SEG1 to SUBR for processing.
    .
    .
process IRT3 and IRT4 values
    returned by SUBR.
    .
    .
END
```

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. If a main program has a subroutine with an external reference defined in a segment then only the segment in which the external is defined may call the subroutine.

Figure 4-2 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.
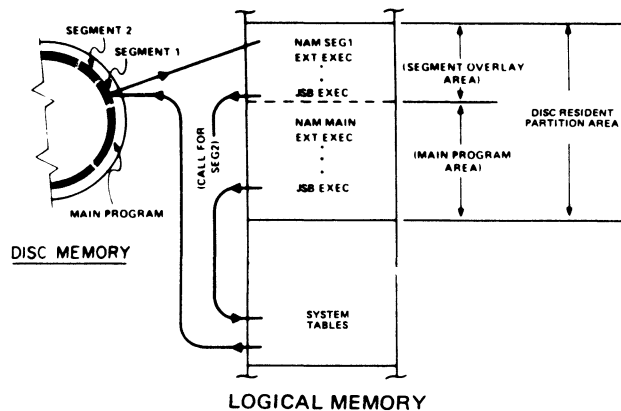
Figure 4-2. Main Calling Segment

An executing segment may itself call in another of the main program's segments by using the same "JSB EXEC" request (see Figure 4-3).

LOGICAL MEMORY
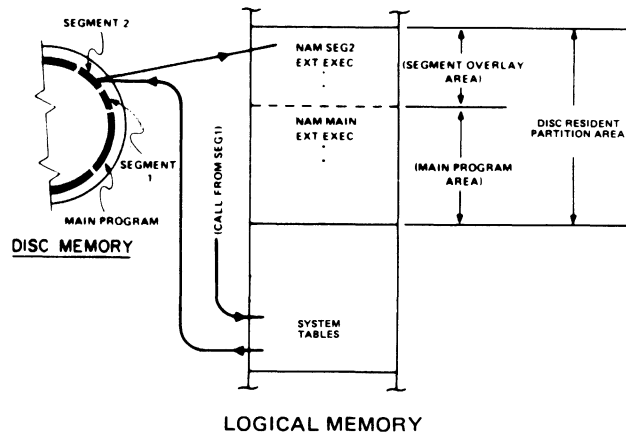
Figure 4-3. Segment Calling Segment

A main program and segment operate as a single program when they are
resident in memory. Jumps from a segment to the main program (or
vice-versa) can be programmed by declaring an external symbol and
referencing it via a JMP instruction (see Figure 4-4). A matching
entry symbol must be defined at the destination in the the other
program.

The generator and the loader associate the main program and its
segments by replacing the symbolic linkage with actual absolute
addresses (i.e., a jump into a segment is executed as a jump to a
specific address). The programmer should be sure that the correct
segment is in memory before the JMP into it is executed.
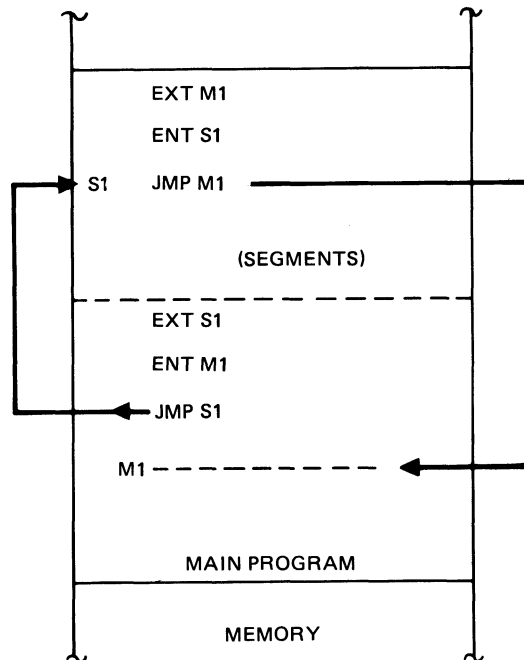
Figure 4-4. Main to Segment Jumps

Chapter 5
EMA Programming

Introduction

The RTE-IVB Operating System has provisions that allow user written programs to manage large data arrays contained in an Extended Memory Area (EMA). The management of the EMA is implemented by routines contained in the System Library. To accommodate large EMA programs (program code plus EMA), RTE-IVB provides for the linking of Real-Time (RT) or Background (BG) partitions (subpartitions) to form a "mother partition" (see, "PARTITION CONSIDERATIONS", below). The EMA variable cannot be used as the buffer parameter in an EXEC or FMP call.

Extended Memory Area (EMA)

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 address 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.

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 can 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 (the buffer crosses a mapping segment boundary), .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, .ERES, .EMIO, MMAP and EMAST subroutine description later in this section for more detailed information.)

Figure 5-1 illustrates the structure of EMA's and MSEG's.
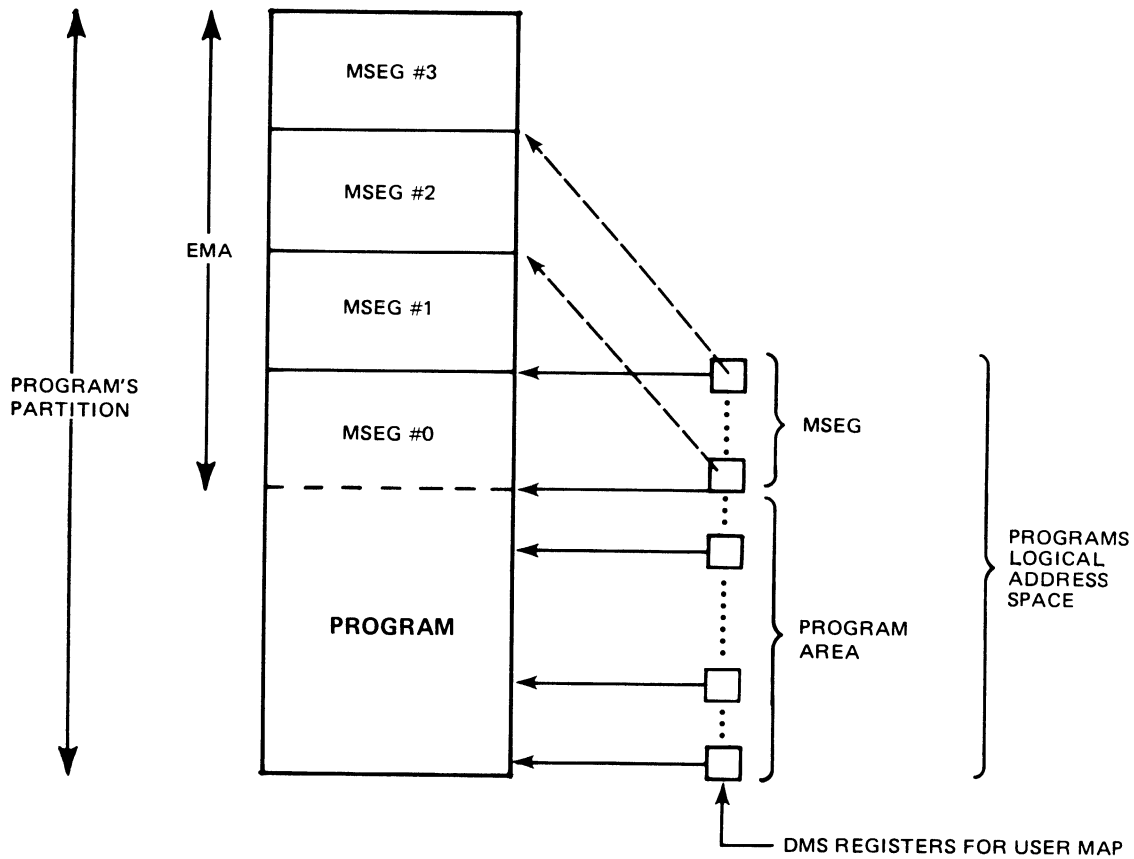
Figure 5-1. EMA and MSEG Structure

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 ml,m2

where label is the EMA label and must be defined, ml 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 5-2.
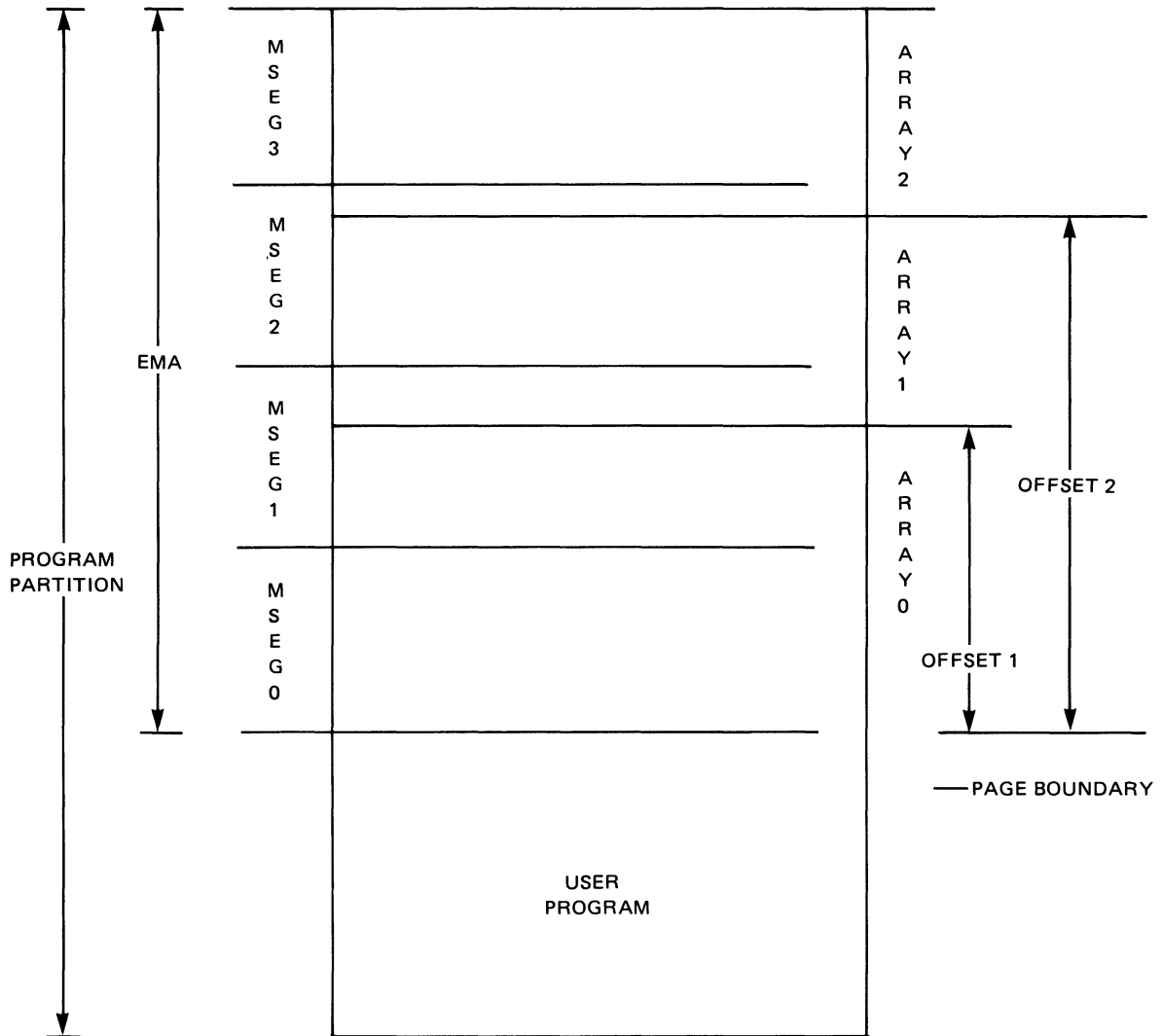
Figure 5-2. Multiple Data Arrays Organization

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-IVB 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.

Partition Considerations

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, 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 into the mother partition. A subpartition cannot be made available by swapping if it contains a memory-locked program, contains a program that is performing I/O in its own area, or contains a scheduled program of higher priority.

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. Note that sufficient swap tracks must be available or the swap of the program does not take place, preventing the dispatching of a new program. 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. Also, if several EMA programs must contend for the same mother partition, overall throughput will be reduced because one program must be swapped to make room for the other.

Subpartitions are not available for dispatching programs when the mother partition is in use by an active program. When a program in a mother partition terminates normally or is aborted, the subpartitions are released and again become available. The mother partition occupant is swapped only under the following conditions:

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 a 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, or 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 dormat program in the mother partition will be swapped out.

EMA Management Subroutines

Five subroutines implement the Extended Memory Area (EMA) capability in the RTE-IVB Operating System. These are: .EMAP, .ERES, .EMIO, MMAP, and EMAST. Although the software versions of these subroutines are actually part of the system library described in Chapter 6 of this manual, they are described here because they are an integral part of EMA programming.

Firmware versions of .EMAP, .EMIO, and MMAP exist for use on the HP/1000 series computer. The firmware version of .EMAP operates slightly different than the software version, as described in the discussion of .EMAP.

Note that the firmware versions of .EMAP, .EMIO, and MMAP should not be called from a program running in privileged mode. To do so could cause the calling program to abort or destroy the integrity of the system.

.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.

NOTE

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.

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 sent to LU6:

        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 that contains the element whos 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).

.ERES Subroutine   (RESOLVES ARRAY ELEMENT ADDRESS; DOES NO MAPPING)

The .ERES subroutine resolves adresses for elements in EMA (and only EMA) arrays. This routine returns a two-word address that represents the offset of the element from the start of EMA.

The calling sequence is:

```
    EXT     .ERES
    •
    •
    JSB     .ERES
    DEF     RTN
    DEF     DUMY        dummy parameter
    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
```

ON RETURN---for a normal return, the offset will be in the A- and B-registers (most significant bits in the B-register). On an error return, the A-register will contain 20 (ASCII) and the B-register will contain EM (ASCII). If the relocatable library subroutine ERR0 is called to handle the error, the following message will be sent to LU6:

                name   20-EM@address

where:

        name=name of program executing when error occured.

        address=address from which ERR0 was called.

The .ERES subroutine is similiar in function to the .EMAP subroutine except that it cannot be used with non-EMA arrays. For more details on error returns and call parameters, refer to the discussion of the .EMAP routine.

.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 ERR0 is called to handle the error, the following message is sent to LU6:

        name 16-EM @ address

where name is the name of the program and address is the location from which ERR0 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.

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
```

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.

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.

EMA Error Message Summary

The following errors can be generated by programs using EMA:

        <name> EM-15 @ <address>

(.EMAP Subroutine) <name> is the name of the program executing, and
<address> is the address of the unresolved array element. The array
(EMA or non-EMA) was specified with incorrect subscripts - negative
subscripts, negative dimensions, or subscript less than the lower
bound.

        <name> EM-16 @ <address>

(.EMIO Subroutine) <name> is the name of the program executing, and
<address> is the location from which the error routine was called.
The buffer length is negative, does not fall within EMA bounds, or the
buffer length plus offset is greater than the mapping segment size.
Or, the array was specified by TABLE with incorrect subscripts -
negative subscripts, negative dimensions, subscript less than the
lower bound.

        <name> EM-20 @ <address>

(.ERES Subroutine) <name> is the name of the program executing, and
<address> is the address of the unresolved array element. The array
(EMA only) was specified with incorrect subscripts - negative
subscripts, negative dimensions, subscript less than the lower bound.

Chapter 6
RTE-IVB Library Subroutines

Introduction

RTE-IVB operating systems are delivered with a collection of relocatable subroutines that comprise the system library. This group of subroutines is specific to RTE-IVB operating systems and is 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.

In addition, many RTE subsystems (i.e., Spooling) include subroutines that may be of general use. See the appropriate subsystem manual for more information.

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 are 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 are created by the generator so they can be appended to disc-resident programs.

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 to 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 does incur the overhead associated with reentrant or privileged subroutines.

When a library subroutine is called, there is no guarantee that the contents of the registers will be preserved.

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-IVB through the entry point $LIBR in order to save the data. At the end of execution, the re-entrant subroutine again calls RTE-IVB 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.

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-returned address to calling
                              program
```

```
         BSS   K-3
          -
          -
          -                  Temporary data (K-3 words)
          -
```

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 buffers 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 Chapter 2).

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
```

Memory Resident Library

The memory resident library area in RTE-IVB 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 memory resident library and all memory resident programs, all Type 6 routines are converted to Type 7 routines (making them available to disc resident programs).

Not all subroutines referenced by memory resident programs are necessarily 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 and $LIBX, the subroutine will execute faster since the system does not need to do the reentrant or privileged processing prior to executing the subroutine.

Utility Subroutine Structure

Utility subroutines are subroutines that cannot be shared by several programs because of internal design or I/O operations. A copy of a utility subroutine is appended to every program that calls 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.

System Library Subroutine

In the subsections that follow, certain conventions are used to desribe System Library Subroutine calls.

*   Parameters that are underlined, such as

    CALL RNRQ(IP1,IP2,IP3)
    ___

    have values returned by the system, e.g., the valued is not supplied by the user.

*   Parameters that are double underlined, such as

    CALL REIO(IP1,IP2,IP3,IP4)
    ===

    have values that are system-supplied in some cases and user-supplied in other cases. The comments associated with the call description should be consulted for details concerning their use.

*   Parameters enclosed in square brackets, such as

    CALL PTERR(IP1[,IP2])

    are optional.

*   Parameters with no qualifiers, i.e., square brackets, angle brackets, or underlines, are required and their value is supplied by the user.

REIO - REENTRANT I/O SUBROUTINE

The REIO subroutine permits user programs to perform reentrant I/O and disc resident programs to be swappable while performing I/O. REIO is a utility type library subroutine that is appended to each program that calls it (Type 7).

```
+-----------------------------------------------------------------+
|                                                                 |
|    CALL REIO (ICODE, ICNWD, IBUFF, ILEN)                        |
|                  =====                                          |
+-----------------------------------------------------------------+
|                                                                 |
|    ICODE - Request code. 1 = read;  2 = write.                  |
|                                                                 |
|    ICNWD - Control word. Specifies the LU (must be non-disc)    |
|            involved in the I/O operation. Can also specify      |
|            driver dependent information.                        |
|                                                                 |
|    IBUFF - Data buffer. Contains the data to be written in      |
|            a write operation or data returned from a read       |
|            operation.                                           |
|                                                                 |
|    ILEN  - Data length. Positive number of words or negative    |
|            number of characters to be read or written.          |
|                                                                 |
+-----------------------------------------------------------------+
```

COMMENTS:

ICODE, IBUFF, and ILEN are identical to those used in the EXEC 1 and EXEC 2 calls described in Chapter 2. ICNWD is similiar except that the Z-bit is not used (no control buffer passage) and the LU cannot specify a disc device.

REIO will always perform the requested I/O operation, however, it will perform the operation on a reentrant basis only if the buffer is less than 130 words (to save System Available Memory), and the buffer address is at least three words beyond the beginning of the program. Note that FORTRAN puts equivalenced arrays at the beginning of the program. Therefore, a dummy array (of at least three words) should be equivalenced prior to equivalencing the buffer to be used in the REIO call.

Error processing and the A- and B-register returns are the same as for the EXEC 1 and 2 calls described in Chapter 2.

BINRY - DISC READ OR WRITE

The BINRY subroutine is called from a FORTRAN program to transfer information to or from a disc device. BINRY has two entry points; BREAD for read operations and BWRIT for write operations.

```
+-------------------------------------------------------------------+
|                                                                   |
|   CALL BWRIT (IBUFF, ILEN, IDISC, ITRAK, ISECT, IOFST)            |
|                                                                   |
|   CALL BREAD (IBUFF, ILEN, IDISC, ITRAK, ISECT, IOFST)            |
|             -----                                                 |
+-------------------------------------------------------------------+
|                                                                   |
|   IBUFF - Data buffer. Contains data to be written for a write    |
|           operation or data returned from a read operation (must  |
|           be a non-EMA buffer).                                   |
|                                                                   |
|   ILEN  - Data length. The number of words to be read or written. |
|                                                                   |
|   IDISC - Disc LU. Logical unit number of disc device involved in |
|           the data transfer.                                      |
|                                                                   |
|   ITRAK - Disc track number.                                      |
|                                                                   |
|   ISECT - Disc sector number.                                     |
|                                                                   |
|   IOFST - Sector offset. Offset (in words) within the sector.     |
|           If IOFST = O, the transfer starts at the sector bound-  |
|           ary.  If IOFST = n, the transfer starts n words past    |
|           the beginning of the sector.                            |
|                                                                   |
+-------------------------------------------------------------------+
```

COMMENTS:

Since data transfer between a users program and a disc device are buffered through the driver module on a sector basis, certain sector offset considerations must be allowed for. These sector offset considerations are summarized below:

1. Offset=n (transfer begins within a sector), and less than a sector is written, or the data transfer ends on a sector boundary. The entire first sector is initially read into the driver's 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 (transfer begins on a sector boundary), and less than a sector is written. The remaining data in the sector will be lost unless the entire existing sector on the disc is first 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 unless the entire final sector (of the data transfer) on the disc is read into a user's buffer, modified to reflect the desired changes, and then rewritten on the disc as a full sector.

Note   that   it   is   the   users   responsibility   to   initiate   the "read-before-write" operation in case 2 and 3 above, the system automatically handles case 1.

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 in Appendix L.

```
+-----------------------------------------------------------------------+
|                                                                       |
|    CALL RNRQ (ICON, IRN, ISTAT)                                        |
|                 -----                                                  |
|                                                                       |
+-----------------------------------------------------------------------+
|                                                                       |
|    ICON   - Control option. Defines how the resource number is to     |
|             be used (see COMMENTS).                                    |
|                                                                       |
|    IRN    - Resource number.                                          |
|                                                                       |
|    ISTAT  - Status word. The status word returns are as follows:      |
|                                                                       |
|             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          |
|                                                                       |
+-----------------------------------------------------------------------+
```

COMMENTS:

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 or the system itself.

All programs must agree that a certain RN will be used as a lock or busy indicator for a given device.

Figure 6-1 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 OPTION		
NO W A I T	NO A B O R T	C L E A R	G L O B A L	L O C A L	C L E A R	G L O B A L	L O C A L

Figure 6-1. Control Word Format (ICON)

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 set quantity of resource numbers (RNs) that are specified during generation. If a resource number is not available when a program requests one, the program is suspended until one is free, unless the `no wait´ bit is set. If the `no wait´ bit is set, the RN 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 RN is cleared or deallocated.

The no abort bit is used to alter the error return point of the call as shown in the following example:

```
                                      •
                                      •
                          ICON = ICON + 40000B
                          CALL RNRQ (ICON,...)
error routine         ----->  GO TO 100
normal return point ----->    •
                              •
                              •
                          100 error processing
                              •
```

The above special error return is established by setting bit 14 to 1 in the request code word (ICON). 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. If the error return is taken, error information will be available in the A- and B-registers.

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.

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 - Unlock the specified RN.

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
```

LURQ - LOGICAL UNIT LOCK

Allows a program to exclusively dominate (lock) an input/output device.

```
+-------------------------------------------------------------------+
|                                                                   |
| CALL LURQ (ICON, LURAY, NUM)                                      |
|                                                                   |
+-------------------------------------------------------------------+
| ICON - Control option. An octal number that specifies the locking |
|        or unlocking action to be performed (see COMMENTS).        |
|                                                                   |
| LURAY - LU array. An array of LU numbers to be locked or unlocked.|
|                                                                   |
| NUM - The number of LU's to be locked or unlocked.                |
+-------------------------------------------------------------------+
```

COMMENTS:

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 LURQ routine request allows up to 31 programs to exclusively dominate (lock) input/output devices. Any other program attempting to use or lock a locked LU is suspended until the original program unlocks the LU or terminates.

The functions of the control option (ICON) are summarized below:

* ICON = 000000B - unlock LUs specified in LURAY.

* ICON = 100000B - unlock all LUs the program currently has locked.

* ICON = 000001B - lock (with wait) the specified LUs.

* ICON = 100001B - lock (without wait) the specified LUs.

NO ABORT BIT (bit 14) - The no-abort bit is used to alter the error return point of this call as shown in the following example:

```
                                         .
                              ICON = ICON + 40000B
                              CALL LURQ (ICON,..)
   error routine     ----->   GO TO 100
   normal return point ----->          :
                              100 error processing
                                         .
```

The above special error return is established by setting bit 14 in
ICON (ICON = ICON + 40000B). 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. An error can occur if the
specified LU is not in the SST, or if the specified control word is
illegal. If the program takes the error return, error information
will be available in the A-Register.

DISC ALSO BIT (bit 11)---The disc also bit is used to allow LU locks
on discs. Failure to set this bit when specifying a disc LU lock
request results in the abort error LU02.

UNLOCK---To unlock all owned LUs, the LURAY array is not used but
still must be coded; the program does not abort.

Any LUs the program has locked are unlocked when the program:

* Performs a standard termination.

* Performs a serial reusability termination.

* Aborts.

Note that LUs are not 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 contains 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 to some other
       program.

Note that the calling program cannot 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.

PARSE ($PARS) – ASCII PARSE SUBROUTINE

Allows a program to parse a string into separate parameters.

```
+------------------------------------------------------------------+
|                                                                  |
|   CALL PARSE (IBUFA,ICON,IRBUF)                                   |
|                        -----                                     |
+------------------------------------------------------------------+
|                                                                  |
|   IBUFA - Source buffer. Contains the ASCII string to be parsed. |
|                                                                  |
|   ICON  - Character count. Number of characters in the ASCII     |
|           string.                                                |
|                                                                  |
|   IRBUF - Receiving buffer. A 33-word buffer that contains the   |
|           results of the parse operation.                        |
|                                                                  |
+------------------------------------------------------------------+
```

COMMENTS:

The results of the parse operation are stored in IRBUF using four words of IRBUF to represent each parameter found in IBUFA. The function of these four words is summarized below:

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 (ASCII).
4	VALUE(3)	0 If NULL or numeric else the 5th and 6th characters (ASCII).

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 by the system before parsing the string contained in IBUFA.

Word 33 of IRBUF will be set to the number of parameters in the string.

The PARSE routine ignores all blanks and expects commas to delimit the parameters. ASCII parameters are padded to six characters with blanks or if more than 6 characters, the left most 6 are used. Numbers may be negative (leading "-") and/or octal (trailing "B").

The format for the Assembly language version of the PARSE subroutine is as follows:

```
        .
        .
        EXT     $PARS
        .
        .
        LDA     IBUFA
        LDB     ICON
        JSB     $PARS
        DEF     IRBUF
        .
        .
```

INPRS - BUFFER CONVERSION

Converts a buffer of parsed data (produced by the PARSE subroutine) back to its original form.

```
+-----------------------------------------------------------------+
|                                                                 |
|   CALL INPRS (IRBUF, NUPAR)                                      |
|                                                                 |
+-----------------------------------------------------------------+
|                                                                 |
|    IRBUF  - Parameter buffer. Contains the parameters to be con-|
|             verted back to an ASCII string; previously produced by |
|             the PARSE routine.                                   |
|                                                                 |
|    NUPAR  - Number of parameters contained in IRBUF; obtained from |
|             word 33 of IRBUF returned by PARSE (NUPAR = IRBUF(33)).|
|                                                                 |
+-----------------------------------------------------------------+
```

COMMENTS:

IRBUF and NUPAR are obtained from a previous call to the PARSE routine or are formatted by the user as if they were; the function of INPRS is to reverse the action performed by PARSE.

The results of the INPRS operation are stored in IRBUF. The length of the resultant ASCII string will be eight times the number of parameters (8 X NUPAR).

$CVT3 (CNUMD,CNUMO), $CVT1 (KCVT)   — BINARY TO ASCII CONVERSION SUBROUTINES.

These routines convert a positive integer binary number to ASCII.

```
+-----------------------------------------------------------------+
|   CALL CNUMD ( n ,IBUF)                                         |
|   CALL CNUMO ( n ,IBUF)                                         |
|   I= KCVT (n)                                                   |
+-----------------------------------------------------------------+
|                                                                 |
|   n    - Actual binary number that is to be converted to ASCII; |
|          must be positive.                                       |
|                                                                 |
|   IBUF - Three-word array that the ASCII representation         |
|          (6 characters) of n is returned to. For CNUMD, a decimal|
|          representation is returned; for CNUMO, an octal        |
|          representation is returned. Leading zeros are suppressed|
|                                                                 |
|   I    - Least significant two digits of the ASCII decimal      |
|          representation of n.                                   |
|                                                                 |
+-----------------------------------------------------------------+
```

COMMENTS:

The Assembler Language formats for the above routines are shown below:

```
        .
        .
        .
        EXT       $CVT3 (or $CVT1)
        .
        .
        .
        LDA       n
        .
        CLE                 (for octal results; E=0)
        or
        CCE                 (for decimal results; E=1)
        .
        JSB       $CVT3 (or $CVT1)
        return
        .
        .
```

Upon return the register contents will be as follows:

E-register = 1

A-register = (for $CVT3) address of 3-word ASCII result.
             (for $CVT1) the 2 least significant characters of the
             converted number.

B-register = unchanged.

MESSS- MESSAGE PROCESSOR INTERFACE

Provides programmatic access (limited by the user capability if under session control) to all system commands.

```
+------------------------------------------------------------------+
|                                                                  |
|   IA = MESSS (IBUF,INUM [,LU])                                    |
|                                                                  |
+------------------------------------------------------------------+
|                                                                  |
|   IA    - 0 if no message is returned from the system; negative  |
|           character count if message is returned (same as        |
|           A-register return).                                     |
|                                                                  |
|   IBUF  - Command buffer. Contains the ASCII command to be       |
|           passed to the RTE-IVB operating system.                |
|                                                                  |
|   INUM  - Character count. Number of characters (bytes)          |
|           contained in the ASCII command.                        |
|                                                                  |
|   LU    - Replacement LU. If the command in IBUF is a RU or an   |
|           ON, and the first parameter in the parameter string is |
|           zero or absent, then LU will be inserted as the first  |
|           parameter.                                             |
|                                                                  |
+------------------------------------------------------------------+
```

COMMENTS:

If the operating system returns information, it will be placed in IBUF and the character count (negative) is placed in IA (and the A-register). The command buffer (IBUF) should be at least 14 words long to allow for the largest possible system return.

If the command was a RU or ON command, the father ID segment word 33 will be propagated to the son's ID segment word 33.

The formats of the program ID segment and of the Session Control Block (SCB) are shown in Appendix B and J, respectively.

The Assembly Language format of the MESSS routine is as follows:

```
            .
            .
            .
        JSB   MESSS
        DEF   RTN
        DEF   IBUF  ---+
        DEF   INUM     |-parameter addresses
        DEF   LU    ---+
RTN         .
            .
            .
IBUF    BSS   ---+
INUM    DEC      |-parameter values
LU      DEC   ---+
```

On return the A-register will be 0 if no information is returned or
will be the negative character count if information is returned.

COR.A, COR.B - FIND FIRST WORD OF AVAILABLE MEMORY.

COR.A returns the address of the first word of available memory (high address +1) for a main program or program segment given the address of the main or segments ID segment.

COR.B returns the first word of available memory for a main program given the address of its ID segment. For segmented programs the address returned is equal to:

largest segment high address +1.

```
+--------------------------------------------------------------------+
|                                                                    |
|   Assembly Language calling sequence:                              |
|                                                                    |
|                                                                    |
|          .                        .                                |
|          .                        .                                |
|         EXT   COR.A              EXT   COR.B                        |
|          .                        .                                |
|          .                        .                                |
|         LDA   IDSEG              LDA   IDSEG                        |
|         JSB   COR.A              JSB   COR.B                        |
|         -return                  -return-                          |
|          .                        .                                |
|          .                        .                                |
|                                                                    |
|   IDSEG - ID segment address; must be long ID segment address      |
|           for COR.B.                                               |
|                                                                    |
+--------------------------------------------------------------------+
```

COMMENTS:

For non-segmented programs, the address returned by COR.A and COR.B are the same.

On return from COR.A, the A-register contains the high address +1 of the main program or segment associated with the ID segment specified. The B-register is not used.

On successful returns from COR.B, the B-Register contains the high address +1 of the main program associated with the specified ID segment. If the program is segmented, the B-Register contains the high address +1 of the largest segment. The A-Register contains 0.

On unsuccessful returns from COR.B, the A-Register contains -1, and the contents of the B-Register are meaningless. COR.B makes an error return if it is passed the address of a short ID segment.

.DRCT - INDIRECT ADDRESS SUBROUTINE

This routine resolves an indirect address within the calling program's map.

```
+--------------------------------------------------------------+
|                                                              |
|    Assembly Language calling sequence:                       |
|                                                              |
|                      EXT   .DRCT                             |
|                        .                                     |
|                        .                                     |
|                      JSB   .DRCT                             |
|                      DEF   ADDR                              |
|                      -return-                                |
|                                                              |
|    ADDR - Address to be resolved.                            |
|                                                              |
+--------------------------------------------------------------+
```

COMMENTS:

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.

FTIME - CURRENT TIME FORMATTED.

This subroutine returns the date and time as an ASCII string.

```
+----------------------------------------------------------------+
|                                                                |
|   CALL FTIME (IBUF)                                             |
|              ----                                              |
|                                                                |
+----------------------------------------------------------------+
|                                                                |
|   IBUF - 15-word array that receives the ASCII string.         |
|                                                                |
+----------------------------------------------------------------+
```

COMMENTS:

The format of the returned string is illustrated by the following examples:

    12:42 PM WED., 16 AUG., 1978

The month will be returned in the following format:

    xxxx or xxx.

GETST - RECOVER PARAMETER STRING.

This routine 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).

```
+------------------------------------------------------------------+
|                                                                  |
|   CALL GETST (IBUF, ILEN, ILOG)                                  |
|                ----        ----                                  |
+------------------------------------------------------------------+
|                                                                  |
|    IBUF - String buffer. Array that the parameter string is      |
|           returned to.                                           |
|                                                                  |
|    ILEN - String length. Requested number of words (if positive) |
|           or characters (if negative) to be returned.            |
|                                                                  |
|    ILOG - Transmission log. Actual number of words or characters |
|           returned.                                              |
+------------------------------------------------------------------+
```

COMMENTS:

The Assembly Language calling sequence for GETST is as follows:

```
            EXT    GETST
            .
            .
            JSB    GETST
            DEF    RTN
            DEF    IBUF
            DEF    ILEN
            DEF    ILOG
    RTN     .
            .
            .
    IBUF    BSS    n
    ILEN    DEC    n
    ILOG    NOP
            .
```

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 lower byte of the last word.

Note that the an EXEC 14 call described in Chapter 2 can be used to recover the entire command string (includes the parameter string).

PRTN, PRTM - PARAMETER RETURN

These two routines are used by the calling program to pass parameters back to its father (the program that scheduled it). The father can recover the returned parameters by calling the RMPAR routine described in the DOS/RTE Relocatable Library Reference Manual.

```
+-----------------------------------------------------------------+
|                                                                 |
|   CALL PRTN (IPRAM)                                             |
|                                                                 |
|   CALL PRTM (IPRAM)                                             |
|                                                                 |
+-----------------------------------------------------------------+
|                                                                 |
|   IPRAM - Parameter buffer. A 5-word array (for PRTN) or a 4-word|
|           array (for PRTM) that contains the parameters to be   |
|           returned to the father program.                       |
|                                                                 |
+-----------------------------------------------------------------+
```

COMMENTS:

The PRTN routine passes five parameters and clears the wait flag. Since the wait flag is cleared, the calling program should terminate immediately after the call. For example;

```
        DIMENSION IPRAM (5)
           .
           .
        CALL PRTN (IPRAM)
        CALL EXEC (6)
```

The PRTM routine passes four parameters and does not clear the wait flag; an immediate termination (EXEC 6) is not necessary. When the parameters are recovered with RMPAR, the first parameter will be meaningless.

The Assembly Language calling sequence for the PRTN and PRTM routines are shown below:

```
        EXT   EXEC, PRTN
          .
          .
        place values in IPRAM
          .
        JSB   PRTN
        DEF   *+2
        DEF   IPRAM
        JSB   EXEC
        DEF   *+2
        DEF   SIX
          .
          .
IPRAM   BSS   5
SIX     DEC   6


        EXT   PRTM
          .
          .
        place values in IPRAM
          .
        JSB   PRTM
        DEF   *+2
        DEF   IPRAM
          .
          .
IPRAM   BSS   4
```

IFBRK - BREAK FLAG TEST.

This routine tests the break flag and clears it if it is set. The
break flag is set with the BR command described in the RTE-IVB
Terminal User's Reference Manual.

```
+-------------------------------------------------------------------+
|                                                                   |
|    IF (IFBRK (IDMY)) n, m                                          |
|                                                                   |
+-------------------------------------------------------------------+
|                                                                   |
|    IDMY - A dummy variable used to inform the FORTRAN compiler     |
|           that an external function is being called.              |
|                                                                   |
|    n    - Statement number that control branches to if the break  |
|           flag was set; the flag will be cleared.                 |
|                                                                   |
|    m    - Statement number that control branches to if the break  |
|           flag is not set.                                        |
|                                                                   |
+-------------------------------------------------------------------+
```

COMMENTS:

The Assembly Language calling sequence for IFBRK is as follows:

```
         .
         EXT  IFBRK
         .
         .
         JSB  IFBRK
         DEF  *+1
         .
         .
```

On return, the A-register will contain -1 if the break flag was set
or will contain 0 if it was not set. The flag will be cleared if it
was set.

IDGET - RETRIEVE PROGRAM'S ID SEGMENT ADDRESS.

This routine retrieves the ID Segment address of a specified program.

```
+-------------------------------------------------------------------+
|                                                                   |
|    IDSEG = IDGET (INAM)                                            |
|    -----                                                          |
|                                                                   |
+-------------------------------------------------------------------+
|                                                                   |
|    IDSEG - ID segment address. Contains the returned ID segment   |
|            address of the specified program; set to 0 if the      |
|            program does not exist.                                 |
|                                                                   |
|    INAM  - Program name. 3-word array used to contain the         |
|            5-character ASCII name of the program that the ID      |
|            segment address is being requested for.                |
|                                                                   |
+-------------------------------------------------------------------+
```

COMMENTS:

The Assembly Language calling sequence for the IDGET routine is as follows:

```
              .
              .
         EXT     IDGET
              .
              .
         JSB     IDGET
         DEF     *+2
         DEF     INAM
              .
              .
    INAM ASC 3,PROGY
              .
```

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

TMVAL - CURRENT TIME REFORMAT

This routine reformats the system millisecond time format (double word negative integer) into an array of time parameters.

```
+-----------------------------------------------------------------+
|                                                                 |
|   CALL TMVAL (ITIM, ITMR)                                        |
|                 ----                                             |
|                                                                 |
+-----------------------------------------------------------------+
|                                                                 |
|   ITIM - Two-word negative time value in tens of milliseconds.  |
|          This double-word integer can be obtained from the system|
|          entry point $TIME (real-time clock) or the time value  |
|          stored in a program's ID segment.                      |
|                                                                 |
|   ITMR - Time array. 5-word array that the system returns the   |
|          reformatted time to. The array is set up as:           |
|                                                                 |
|          ITMR(1) = tens of milliseconds                         |
|          ITMR(2) = seconds                                      |
|          ITMR(3) = minutes                                      |
|          ITMR(4) = hours                                        |
|          ITMR(5) = day of year (julian) -                       |
|                    not related to call values.                  |
|                                                                 |
+-----------------------------------------------------------------+
```

COMMENTS:

The next scheduled execution time of a program currently in the time list can be obtained from the program's ID segment and then formatted into an array of time parameters by the routine TMVAL.

```
      DIMENSION   INAM(3),ITMR(5),IARRAY(2)
      DATA INAM/2HPR,2H06,2H1/
         .
         .
         .
      IDSEG=IDGET (INAM)
      ITAD=IDSEG+18
         .
         .
         .
      IARRAY(1)=IGET(ITAD)
      IARRAY(2)=IGET(ITAD+1)
         .
         .
      CALL TMVAL (IARRAY,ITMR)
```

EQLU - INTERRUPTING LU QUERY.

This routine finds the logical unit number of an interrupting device given the address of word 4 of the device's Equipment Table entry.

```
+----------------------------------------------------------------+
|                                                                |
|   CALL EQLU (LU)                                               |
|              --                                                |
|                                                                |
+----------------------------------------------------------------+
|                                                                |
|   LU - Logical unit number of interrupting device (same as     |
|        A-register return).                                     |
|                                                                |
+----------------------------------------------------------------+
```

COMMENTS:

The EQLU routine expects the address of EQT word 4 of the interrupting device to be in the B-register. This is done by another program/subroutine (using LDB EQT4) or by the driver associated with the interrupting device.

Note that the routine will function correctly only if the LU number to be returned is less than or equal to 99.

The Assembly Language format is as follows:

```
          EXT   EQLU
            .
            .
          JSB   EQLU
          DEF   RTN
          DEF   LU
    RTN     .
            .
```

On return:

A-register = 0 if an LU referring to the EQT was not found.
           = LU if LU was found.

B-register = ASCII "00" (if LU not found) or the ASCII LU number.

LU   = Same as A-register.

TRMLU - TERMINAL LU QUERY.

This routine finds the logical unit number of an interrupting device given the address of word 4 of its Equipment Table entry and checks that it is an interactive device.

```
+------------------------------------------------------------------+
|                                                                  |
|   CALL TRMLU (LU)                                                |
|              --                                                  |
|                                                                  |
+------------------------------------------------------------------+
|                                                                  |
|   LU - Logical unit number of interrupting device. (Same as     |
|        A-register return).                                       |
|                                                                  |
+------------------------------------------------------------------+
```

COMMENTS:

The TRMLU routine expects the address of EQT word 4 of the interrupting device to be in the B-register. This is done by another program/subroutine (using LDB EQT4) or by the driver associated with the interrupting device.

Note that this routine will function correctly only if the LU number to be returned is less than or equal to 99.

The Assembly Language format is as follows:

```
        EXT   TRMLU
          .
          .
        JSB   TRMLU
        DEF   RTN
        DEF   LU
    RTN   .
          .
```

On return:

A-register = 0 if LU not found.
            = LU number if found.

B-register = ASCII "00" if LU not found, ASCII LU number if found.

LU   = Same as A-register. (optional).

IFTTY - INTERACTIVE LU QUERY.

This routine determines whether a logical unit is interactive or not.

```
+-----------------------------------------------------------------+
|                                                                 |
|    INT = IFTTY (LU)                                              |
|    ---                                                          |
|                                                                 |
+-----------------------------------------------------------------+
|                                                                 |
|    INT - Set to -1 if LU is interactive; Set to 0 if LU is not  |
|          interactive  (same as A-register return).              |
|                                                                 |
|    LU  - Logical unit number of device being tested.            |
|                                                                 |
+-----------------------------------------------------------------+
```

COMMENTS:

The Assembly Language calling sequence for IFTTY is as follows:

```
          EXT  IFTTY
                .
                .
          JSB  IFTTY
          DEF  RTN
          DEF  LU
   RTN     .
                .
   LU   DEC  n        Logical unit being tested.
```

On return, the following registers are set as indicated:

A-register = -1 if LU is interactive, 0 if it is not interactive.

B-register = Upper byte is the driver type (word 5 of EQT table
             entry, bits 8-13). Lower byte is the subchannel number.

LOGLU - RETURNS LU OF SCHEDULING PROGRAM.

This routine returns the logical unit numbers of the terminal that
the currently executing program was scheduled from.

```
+----------------------------------------------------------------+
|                                                                |
|    LU = LOGLU (LUSYS)                                           |
|    --          -------                                          |
|                                                                |
+----------------------------------------------------------------+
|                                                                |
|    LU      - Logical unit number of device from which the calling |
|              program was scheduled  (same as A-register); 1(if in |
|              session) or positive log-LU (if not in session)   |
|                                                                |
|    LUSYS - The system LU of the session terminal (in session) or |
|              the negative log-LU (not in session).             |
|                                                                |
+----------------------------------------------------------------+
```

COMMENTS:

LOGLU will return the LU numbers 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.

The Assembly Language calling sequence is as follows:

```
        EXT   LOGLU
          .
          .
        JSB   LOGLU
        DEF   RTN
        DEF   LUSYS
  RTN   .
          .
```

On return:

A-register = LU number of device from which program was scheduled.

B-register = ASCII LU number.

LUTRU - TRUE SYSTEM LU.

This routine returns the true system logical unit number associated with a session or batch LU.

```
+------------------------------------------------------------------+
|                                                                  |
|   CALL LUTRU (ITEST, LU)                                         |
|                      --                                          |
|      or                                                          |
|                                                                  |
|   LU = LUTRU (ITEST)                                            |
|   --                                                             |
|                                                                  |
+------------------------------------------------------------------+
|                                                                  |
|   LU    - True system LU is returned here.                      |
|                                                                  |
|   ITEST - The session logical unit number to be checked.        |
|                                                                  |
+------------------------------------------------------------------+
```

COMMENTS:

If the calling program is not a session or batch program, LU is set equal to ITEST.

If the calling program is a session program and ITEST is not defined for the caller's session, LU is set to -1.

If the calling program is a batch program and ITEST is not defined in the Batch Switch Table, LU is set equal to ITEST.

LUSES - FIND SCB.

This routine scans the list of Session Control Blocks (SCBs) looking
for a SCB defined for the session identifier passed in the call.

```
+----------------------------------------------------------------+
|                                                                |
|    ISCB = LUSES (IDBNT)                                         |
|    ----                                                        |
|                                                                |
+----------------------------------------------------------------+
|                                                                |
|    ISCB   = Address of SST length word in SCB if found, 0 in not |
|             found (same as A-register return).                 |
|                                                                |
|    IDBNT = Session identifier (word 3 of SCB).                 |
|                                                                |
+----------------------------------------------------------------+
```

COMMENTS:

The Assembly Language format is as follows:

```
          EXT   LUSES
            .
            .
          JSB   LUSES
          DEF   RTN
          DEF   IDENT
    RTN     .
            .
```

On return:

A-register = 0 if SCB not found.

           = SCB address if found.

GTERR - GET SCB ERROR MNEMONIC.

This routine returns the SCB error mnemonic from the current Session Control Block.

```
+--------------------------------------------------------------------+
|                                                                    |
|   CALL GTERR (INMIC [,IERR])                                       |
|                 -----   ----                                       |
|                                                                    |
+--------------------------------------------------------------------+
|                                                                    |
|    INMIC - 4-word buffer that error mnemonic is returned to.       |
|                                                                    |
|    IERR  - Error return. 0 indicates that the retrieval was        |
|            successful; -1 indicates that the calling program       |
|            was not in session  (optional).                         |
|                                                                    |
+--------------------------------------------------------------------+
```

COMMENTS:

The error mnemonic is an 8 ASCII character message that represents the last error encounted for the current session. The error mnemonic is posted by the RTE-IVB operating system and some HP supported subsystems. The error mnemonic can also be updated by a user application program by calling the library routine PTERR.

Note the HELP command (see RTE-IVB Terminal User´s Reference Manual) uses the 8 character mnemonic as the implicit keyword to search the HELP file and return expanded information on the last error posted in the current session.

The error mnemonic is stored in words 5 through 8 of the SCB. The format of the SCB is shown in Appendix J.

Note that if the "no-abort" bit is set when an EXEC call is made and an error occurs, this error program is not put in the SCB because the program did not abort.

PTERR - UPDATE SCB ERROR MNEMONIC.

This routine updates the error mnemonic in the current Session Control Block (SCB).

```
+----------------------------------------------------------------+
|                                                                |
|   CALL PTERR (INMIC [,IERR])                                   |
|                  ----                                          |
|                                                                |
+----------------------------------------------------------------+
|                                                                |
|   INMIC - 4-word buffer that contains the error mnemonic to be |
|           posted to the SCB.                                   |
|                                                                |
|   IERR  - Error return. 0 indicates a successful posting oper- |
|           ation;  -1 indicates that the calling program was not|
|           in session (optional).                              |
|                                                                |
+----------------------------------------------------------------+
```

COMMENTS:

The PTERR routine place an 8 ASCII character (4-words) message in the current SCB. This message can be retrieved by calling the library routine GTERR. The PTERR/GTERR calls can be used to implement error and data communication schemes between programs running in the current session.

SESSN - IN SESSION QUERY

This routine determines if the calling program is in session.

```
+----------------------------------------------------------------+
|                                                                |
|    Assembly Language calling sequence:                         |
|                                                                |
|                       .                                        |
|                       .                                        |
|                       .                                        |
|               JSB    SESSN                                     |
|               DEF    RTN                                       |
|               DEF    ID                                        |
|         RTN         -return-                                   |
|                                                                |
|    ID - ID segment address of program that is to be checked.   |
|                                                                |
+----------------------------------------------------------------+
```

COMMENTS:

On return from SESSN:

E-register = 0 if calling program was in session.
             1 if calling program was not in session.

B-register = ID segment session word (address of SST length word in
             SCB) if calling program was in session.

The ID segment address of the calling program can be obtained by
using the library routine IDGET.

The formats of the ID segment and the Session Control Block (SCB) are
shown in Appendix B and J, respectively.

ICAPS - GET SESSION CAPABILITY.

This routine returns the current session's capability level.

```
+----------------------------------------------------------------+
|                                                                |
|   ICPSE = ICAPS (IDUMY)                                         |
|                                                                |
+----------------------------------------------------------------+
|                                                                |
|                                                                |
|   IDUMY - Dummy variable.                                      |
|                                                                |
|   ICPSE - Session capability; 0 if calling program was not in  |
|           session.                                             |
|                                                                |
+----------------------------------------------------------------+
```

SYCON - MESSAGE ROUTE

This routine writes a message to the system console (system LU1).

```
+-----------------------------------------------------------------+
|                                                                 |
|   CALL SYCON (IBUF, ILEN)                                        |
|                                                                 |
+-----------------------------------------------------------------+
|                                                                 |
|   IBUF - Buffer that contains the message to be written.        |
|                                                                 |
|   ILEN - Length of IBUF, positive value indicates the number of |
|          words and a negative value indicates the number of     |
|          characters.                                            |
|                                                                 |
+-----------------------------------------------------------------+
```

COMMENTS:

This routine bypasses the Session Switch Table (SST) and writes directly to system LU1.

SEGLD - SEGMENT LOAD

This routine loads a program's segment into memory from disc and transfers control to the segment's entry point.

```
+-----------------------------------------------------------------+
|                                                                 |
|    CALL SEGLD (INAM,IERR[,IP1][,IP2][,IP3][,IP4][,IP5])         |
|                     ----                                        |
|-----------------------------------------------------------------|
|                                                                 |
|    INAM - Segment name.  3-word array containing the ASCII      |
|           name of the segment to be loaded.                     |
|                                                                 |
|    IERR - Error return.  05 if load could not be performed.     |
|                                                                 |
|    IP1 thru IP5 - Optional parameters; passed to segment in     |
|                   INAM.                                          |
|                                                                 |
+-----------------------------------------------------------------+
```

COMMENTS:

If the first attempt to load the specified segment is unsuccessful, SEGLD schedules program T5IDM. T5IDM is a program that manages ID segments for type 5 programs. T5IDM then builds an ID segment for the specified program segment (if it was saved as a type 6 file) and SEGLD attempts to load the segment again. If the segment was not saved as a type 6 file or the load is unsuccessful, an error is returned.

To be accessed by T5IDM, the main program and all of its segments must be saved as type 6 files (:SP command) on LU 2 or LU 3. The main and all of its segments must be on the same LU.

GTSCB-Get SCB

This routine returns the contents of the current SCB.

+--+
| |
| CALL GTSCB (IBUF,ILEN,IERR[,ADSCB]) |
| ---- ---- |
IBUF - Buffer that the contents of the SCB are returned to,
beginning with word 3.
ILEN - Length of IBUF.
IERR - If positive, indicates the number of words in the
SCB (not counting words 0, 1, and 2). If -1,
indicates the calling program was not in session.
If -2, indicates the SCB address passed to GTSCB was
not a valid address. If a negative number (besides
-1 or -2), indicates that the buffer size passed to
GTSCB (IBUF) was too small; the negative number
indicates the actual SCB size (not counting words 0,
1, and 2).
ADSCB - Address of SCB (optional).
+--+

COMMENTS:

The format of the Session Control Block (SCB) is shown in Appendix J.

LIMEM- MEMORY PARTITION LIMITS

Finds and returns limits of available memory of the partition in which the calling program is currently executing.

```
+---------------------------------------------------------------+
|                                                               |
|    CALL LIMEM (IWHCH,IFWAM,IWRDS)                              |
|                  -----  -----                                 |
|                                                               |
|---------------------------------------------------------------|
|                                                               |
|    IWHCH - must be >=0 for call to succeed.                    |
|                                                               |
|    IFWAM - returns the address of the first word of available |
|            memory in the partition.                           |
|                                                               |
|    IWRDS - returns the number of words available.             |
|                                                               |
+---------------------------------------------------------------+
```

COMMENTS:

LIMEM returns the first word of available memory and the number of available words in the program partition. If called from the main of a segmented program, LIMEM returns the address of the first word of available memory of the main plus its largest segment.

A call to LIMEM generates an EXEC 26 call, described earlier in this manual. Refer to the Library Subroutine COR.A to determine how much memory is available when a program's segment is in the program's partition.

SEGRT- RETURN TO MAIN FROM SEGMENT

SEGRT allows a segment to return to the instruction following the SEGLD call in the main program.

```
+-------------------------------------------------------------+
| CALL SEGRT (Z)                                              |
+-------------------------------------------------------------+
|     Z is a dummy parameter, supplied to the FORTRAN compiler, |
|     indicating a subroutine call.  This parameter is not     |
|     necessary in Pascal, Assembler, or Macro versions of the |
|     call.                                                    |
+-------------------------------------------------------------+
```

SEGRT allows any segment that was called from the main by a SEGLD request to return to the instruction following the SEGLD request in the main program.

There are restrictions on the use of SEGRT:

1. SEGRT can only be used if the segment was loaded by a SEGLD request.

2. The segment must have been loaded from the main, not another segment.

3. SEGRT can only return to the main, not another segment.

DFCHI, FCHI, DFCIH, FCIH — Convert Floating Point

The floating point conversion routines convert between HP 1000 and IEEE standard floating point formats. These subroutines are part of the $MATH library.

DFCHI- CONVERT DOUBLE PRECISION TO IEEE

DFCHI is a function that converts from HP 1000 format double precision floating point to IEEE standard format.

```
+--------------------------------------------------------------+
| IERR = DFCHI(hpfp, i3efp)                                     |
+--------------------------------------------------------------+
| hpfp  - is the double precision floating point (64-bit) real |
|         to be converted to IEEE standard format.             |
|                                                              |
| i3efp - is the 64-bit variable in which the IEEE standard    |
|         double precision floating point real will be         |
|         stored.                                              |
+--------------------------------------------------------------+
```

IERR Returns:

    0   Successful conversion
    -3   Value of hpfp is denormalized number; i3efp unchanged

FCHI- CONVERT SINGLE PRECISION TO IEEE

FCHI is a function that converts from HP 1000 format single precision floating point to IEEE standard format.

```
+--------------------------------------------------------------+
| IERR = FCHI(hpfp, i3efp)                                      |
+--------------------------------------------------------------+
| hpfp  - is the single precision floating point (32-bit) real |
|         to be converted to IEEE standard format.             |
|                                                              |
| i3efp - is the 32-bit standard single precision floating     |
|         point variable in which the IEEE standard single     |
|         precision floating point real will be stored.        |
+--------------------------------------------------------------+
```

IERR Returns:

    0   Successful conversion
    -3   Value of hpfp is denormalized number; i3efp unchanged
    -5   Underflow occurred; i3efp unchanged

DFCIH- CONVERT DOUBLE PRECISION TO HP 1000

DFCIH is a function that converts from IEEE standard format double precision floating point to HP 1000 format.

```
+----------------------------------------------------------------+
| IERR = DFCIH(i3efp, hpfp)                                      |
+----------------------------------------------------------------+
| hpfp   - is the IEEE standard format double precision          |
|          floating point (64-bit) real to be converted to       |
|          HP 1000 format.                                       |
|                                                                |
| i3efp - is the address 64-bit standard single precision        |
|          floating point variable in which the HP 1000 double   |
|          precision floating point real will be stored.         |
+----------------------------------------------------------------+
```

IERR Returns:

```
    0     Successful conversion
   -1     Value of i3efp is not a number; hpfp unchanged
   -2     Value of i3efp is signed infinity; hpfp unchanged
   -3     Value of i3efp is denormalized number; hpfp unchanged
   -4     Overflow occurred; hpfp unchanged
```

FCHI- CONVERT SINGLE PRECISION TO HP 1000

FCIH is a function that converts from IEEE standard format single precision floating points to HP 1000 format.

```
+----------------------------------------------------------------+
| IERR FCIH(i3efp, hpfp)                                         |
+----------------------------------------------------------------+
| hpfp   - is the IEEE standard format single precision          |
|          floating point (32-bit) real to be converted to       |
|          HP 1000 format.                                       |
|                                                                |
| i3efp - is the 32-bit standard single precision floating       |
|          point variable in which the HP 1000 single            |
|          precision floating point real will be stored.         |
+----------------------------------------------------------------+
```

IERR Returns:

```
    0     Successful conversion
   -1     Value of i3efp is not a number; hpfp unchanged
   -2     Value of i3efp is signed infinity; hpfp unchanged
   -3     Value of i3efp is denormalized number; hpfp unchanged
   -4     Overflow occurred; hpfp unchanged
   -5     Underflow occurred; hpfp unchanged
```

Appendix A
HP Character Set

Effect of Control key *

				000-037B		040-077B		100-137B		140-177B	

BITS				COLUMN →	$^0{}0_0$	$^0{}0_1$	$^0{}1_0$	$^0{}1_1$	$^1{}0_0$	$^1{}0_1$	$^1{}1_0$	$^1{}1_1$
b_4	b_3	b_2	b_1	ROW ↓	0	1	2	3	4	5	6	7
0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	2	STX	DC2	"	2	B	R	b	r
0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
1	0	0	0	8	BS	CAN	(	8	H	X	h	x
1	0	0	1	9	HT	EM	)	9	I	Y	i	y
1	0	1	0	10	LF	SUB	*	:	J	Z	j	z
1	0	1	1	11	VT	ESC	+	;	K	[	k	{
1	1	0	0	12	FF	FS	,	<	L	\	l	¦
1	1	0	1	13	CR	GS	−	=	M	]	m	}
1	1	1	0	14	SO	RS	.	>	N	∧	n	~
1	1	1	1	15	SI	US	/	?	O	_	o	DEL

32 CONTROL CODES

Upshifted Lower Case

|◄── 64 CHARACTER SET ──►|
|◄── 96 CHARACTER SET ──►|
|◄── 128 CHARACTER SET ──►|

EXAMPLE: The representation for the character "K" (column 4, row 11) is.

	b_7	b_6	b_5	b_4	b_3	b_2	b_1
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 Scandanavian 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[1]	Meaning
	Left Byte	Right Byte			
0	000000	000000	NUL	^N_U	Null
1	000400	000001	SOH	^S_H	Start of Heading
2	001000	000002	STX	^S_X	Start of Text
3	001400	000003	ETX	^E_X	End of Text
4	002000	000004	EOT	^E_T	End of Transmission
5	002400	000005	ENQ	^E_Q	Enquiry
6	003000	000006	ACK	^A_K	Acknowledge
7	003400	000007	BEL	Q	Bell, Attention Signal
8	004000	000010	BS	^B_S	Backspace
9	004400	000011	HT	^H_T	Horizontal Tabulation
10	005000	000012	LF	^L_F	Line Feed
11	005400	000013	VT	^V_T	Vertical Tabulation
12	006000	000014	FF	^F_F	Form Feed
13	006400	000015	CR	^C_R	Carriage Return
14	007000	000016	SO	^S_O	Shift Out ⎫ Alternate
15	007400	000017	SI	^S_I	Shift In ⎬ Character Set
16	010000	000020	DLE	^D_L	Data Link Escape
17	010400	000021	DC1	$^D_1$	Device Control 1 (X-ON)
18	011000	000022	DC2	$^D_2$	Device Control 2 (TAPE)
19	011400	000023	DC3	$^D_3$	Device Control 3 (X-OFF)
20	012000	000024	DC4	$^D_4$	Device Control 4 (TAPE)
21	012400	000025	NAK	^N_K	Negative Acknowledge
22	013000	000026	SYN	^S_Y	Synchronous Idle
23	013400	000027	ETB	^E_B	End of Transmission Block
24	014000	000030	CAN	^C_N	Cancel
25	014400	000031	EM	^E_M	End of Medium
26	015000	000032	SUB	^S_B	Substitute
27	015400	000033	ESC	^E_C	Escape[2]
28	016000	000034	FS	^F_S	File Separator
29	016400	000035	GS	^G_S	Group Separator
30	017000	000036	RS	^R_S	Record Separator
31	017400	000037	US	^U_S	Unit Separator
127	077400	000177	DEL	▓	Delete, Rubout[3]

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	⎫
49	030400	000061	1	
50	031000	000062	2	
51	031400	000063	3	
52	032000	000064	4	
53	032400	000065	5	⎬ Digits, Numbers
54	033000	000066	6	
55	033400	000067	7	
56	034000	000070	8	
57	034400	000071	9	⎭
58	035000	000072	:	Colon
59	035400	000073	;	Semicolon
60	036000	000074	<	Less Than
61	036400	000075	=	Equals
62	037000	000076	>	Greater Than
63	037400	000077	?	Question Mark

9206- 1B

Decimal Value	Octal Values		Character	Meaning
	Left Byte	Right Byte		
64	040000	000100	@	Commercial At
65	040400	000101	A	
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	Upper Case Alphabet.
78	047000	000116	N	Capital Letters
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[4]
95	057400	000137	_ ←	Underline, Back Arrow[4]

Decimal Value	Octal Values		Character	Meaning
	Left Byte	Right Byte		
96	060000	000140	`	Grave Accent[5]
97	060400	000141	a	
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	Lower Case Letters[5]
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[5]
124	076000	000174	¦	Vertical Line[5]
125	076400	000175	}	Right (closing) Brace[5]
126	077000	000176	~	Tilde, Overline[5]

9206- 1C

Notes [1]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

[2]Escape is the first character of a special control sequence For example, ESC followed by "J" clears the display on a 2640 terminal

[3]Delete may be displayed as "_", "@", or space

[4]Normally, the caret and underline are displayed Some devices substitute the up arrow and back arrow

[5]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

A-3

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

A-4

Appendix B
System Communication Area and System Tables

This apppendix 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.

* DISC LAYOUT - Allocation of disc space for an RTE-IVB system.

* TABLE AREA I AND II ENTRY POINTS

Other system tables relating to I/O considerations, such as the Equipment Table, Device Reference Table and Driver Mapping Table are described in Appendix C, "I/O TABLES AND PROCESSING".

System Communication Area

This area is a block of storage in the system base page, starting at location 1645, that is used by RTE-IVB to define request parameters, I/O tables, scheduling lists, operating parameters, memory bounds, etc. The RTE-IVB 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.

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, word 1
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	
01661	EQT2	
01662	EQT3	
01663	EQT4	
01664	EQT5	Addresses of first 11 words of current EQT entry
01665	EQT6	(see 01771 for last four words)
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 TTY
SYSTEM REQUEST PROCESSOR/EXEC COMMUNICATION		
01676	RQCNT	Number of request parameters −1
01677	RQRTN	Return point address
01700	RQP1	
01701	RQP2	
01702	RQP3	
01703	RQP4	Addresses of request parameters (set for a
01704	RQP5	maximum of nine parameters)
01705	RQP6	
01706	RQP7	
01707	RQP8	
01710	RQP9	

Table B-1. System Communications Area Locations (Continued)

OCTAL LOCATION	CONTENTS	DESCRIPTION
SYSTEM LISTS ADDRESSES		
01711	SKEDD	Schedule list
01713	SUSP2	Wait Suspend list
01714	SUSP3	Available Memory list
01715	SUSP4	Disc Allocation list
01716	SUSP5	Operator Suspend list
PROGRAM ID SEGMENT DEFINITION		
01717	XEQT	ID segment address of current program
01720	XLINK	Linkage
01721	XTEMP	Temporary (five words)
01726	XPRIO	Priority word
01727	XPENT	Primary entry point
01730	XSUSP	Point of suspension
01731	XA	A-register at suspension
01732	XB	B-register at suspension
01733	XEO	E and overflow register suspension
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 dummy interface flag
01740	IDSDA	Disc address of first ID segment
01741	IDSDP	Position within disc sector
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

B-3

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 library entry points
01762	DSCLN	Number of user available library entry points.
01763	DSCUT	Disc address of relocatable disc resident library.
01764	SYSLN	Number of system library entry points
01765	LGOTK	LGO: LU #, starting track, number of tracks (same format as ID segment word 28)
01766	LGOC	Current LGO 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	
01772	EQT13	Address of last four words of current EQT
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.

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.

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 segment's address is located in the Keyword Table (see location 01657).
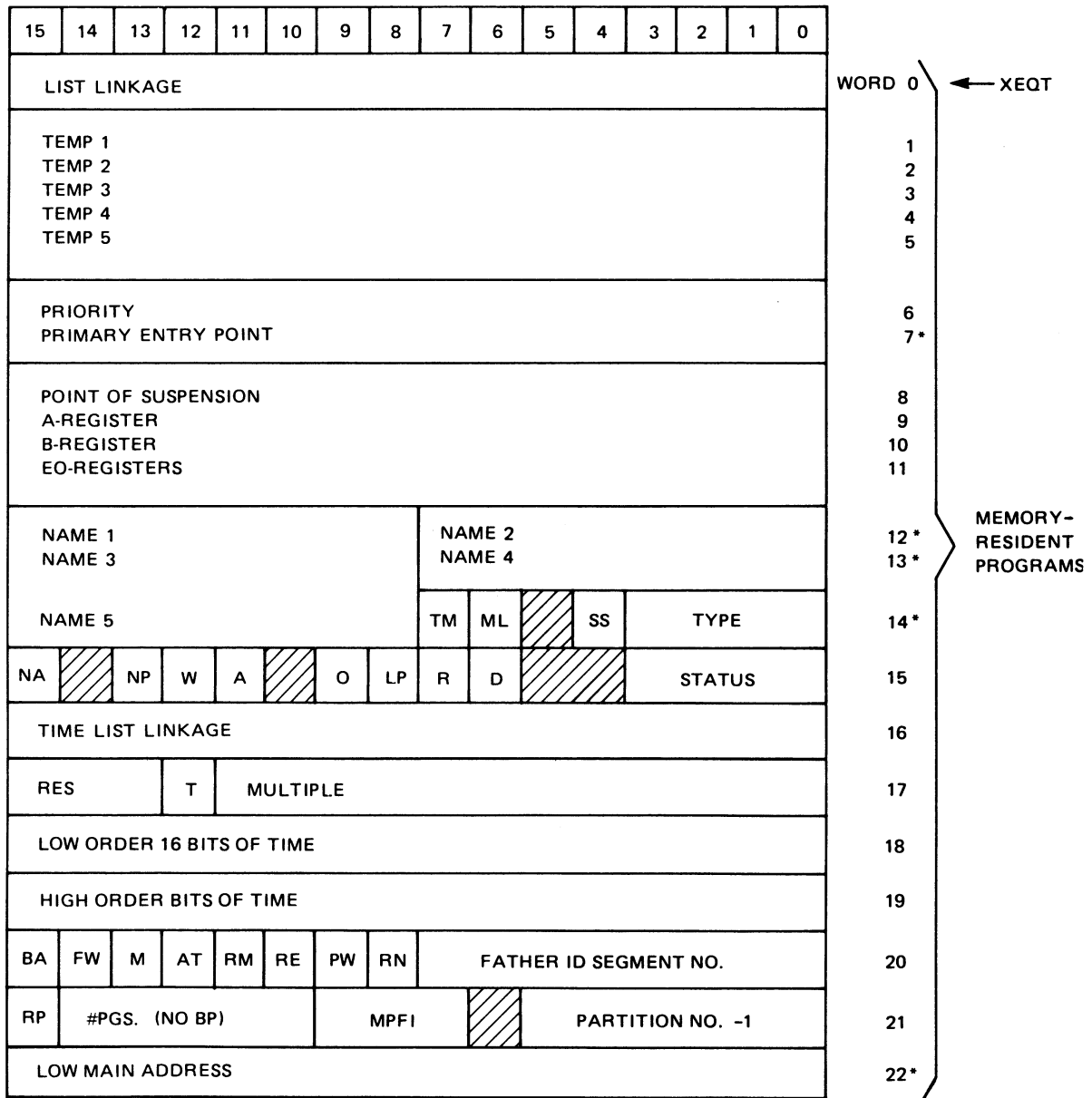
Figure B-1. ID Segment Format

Figure B-1.   ID Segment Format (Continued)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	

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
TIMESLICE WORD	30
SEQCNT	DC	CF	SESSION ID	31
SESSION WORD	32

MEMORY RESIDENTS
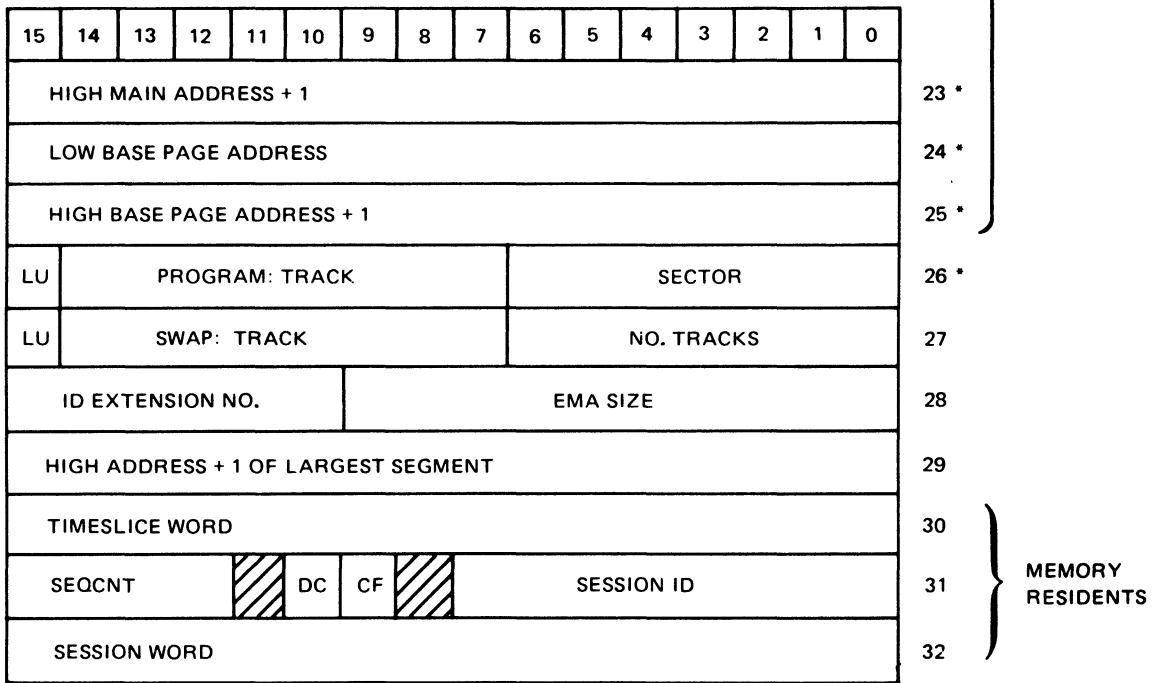
8300-236

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 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 1)

    A = abort on next list entry for this program

    O = operator suspend on next schedule attempt

   LP = load in progress; program is being dispatched from disc.

R = resource save (save resources when setting dormant)

D = dormant bit (set dormant on next schedule attempt)

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

TIMESLICE WORD (30):

The timeslice word defines the timeslicing status of a program. This word is defined as follows:

1 = This program has just been rescheduled or is not timesliced.

0 = This program has used a full timeslice or program is not scheduled.

<0 = This program was running (under timeslice control) and was "bumped" from execution by a higher priority program. This word represents the remaining timeslice for this program.

OPEN FLAG WORD(31):

SEQCNT   = sequence counter. Each time a program is aborted or
              terminates (unless saving resources) the counter is
              incremented. The counter value is used to build FMP open
              Flags.

DC   = don't copy flag. Set by the generator (if 128 is added
              to program type) or the loader (using Don't Copy
              op-code). Indicates that the program is not to be
              renamed by FMGR (no duplication).

CP   = copy flag. Indicates that the program is a copy.

Session ID = System LU of terminal that program was loaded from. For
              programs permanently loaded or temporarily loaded by the
              system manager, a zero is shown here.

SESSION WORD(32):

The session word identifies the user of a program.

A negative value represents the logical unit number of the terminal
from which the program was invoked (not under session).

A positive value represents the address of the SST length word of the
session control block for the session currently using this program
(under session).

Programs scheduled by interrupt have a zero in this word.

ID Segment Extension

Each EMA program requires a 3-word ID segment extension in addition to its 33-word ID segment. The number of ID extensions contained in the system is also 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.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1.	0	
NS	CURRENT MSEG NO.										# PAGES MSEG					WORD 0
MSEG START PAGE (LOGIC.)			DE	(PHYSICAL) EMA START PAGE												WORD 1
/////////				# TRACKS FOR EMA SWAP												WORD 2

WHERE:

NS = 0 IF THE MSEG IS POINTING TO A STANDARD SEGMENT OF THE EMA
(SET UP BY .EMAP)

= 1 IF THE MSEG IS POINTING TO A NON-STANDARD SEGMENT
(SET UP BY .EMIO)

DE = 0 IF THE EMA SIZE WAS SPECIFIED BY THE USER

= 1 IF THE EMA SIZE IS ALLOWED TO DEFAULT TO THE MAXIMUM SIZE
AVAILABLE TO THE SYSTEM.

Figure B-2.   ID Segment Extension

Short ID Segment

Short ID segments requiring nine words are used only for 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. The information contained in a short ID segment is illustrated in Figure B-1.

RTE-IVB System Disc Layout

Figure B-3 illustrates how disc space is allocated when a RTE-IVB system is generated.
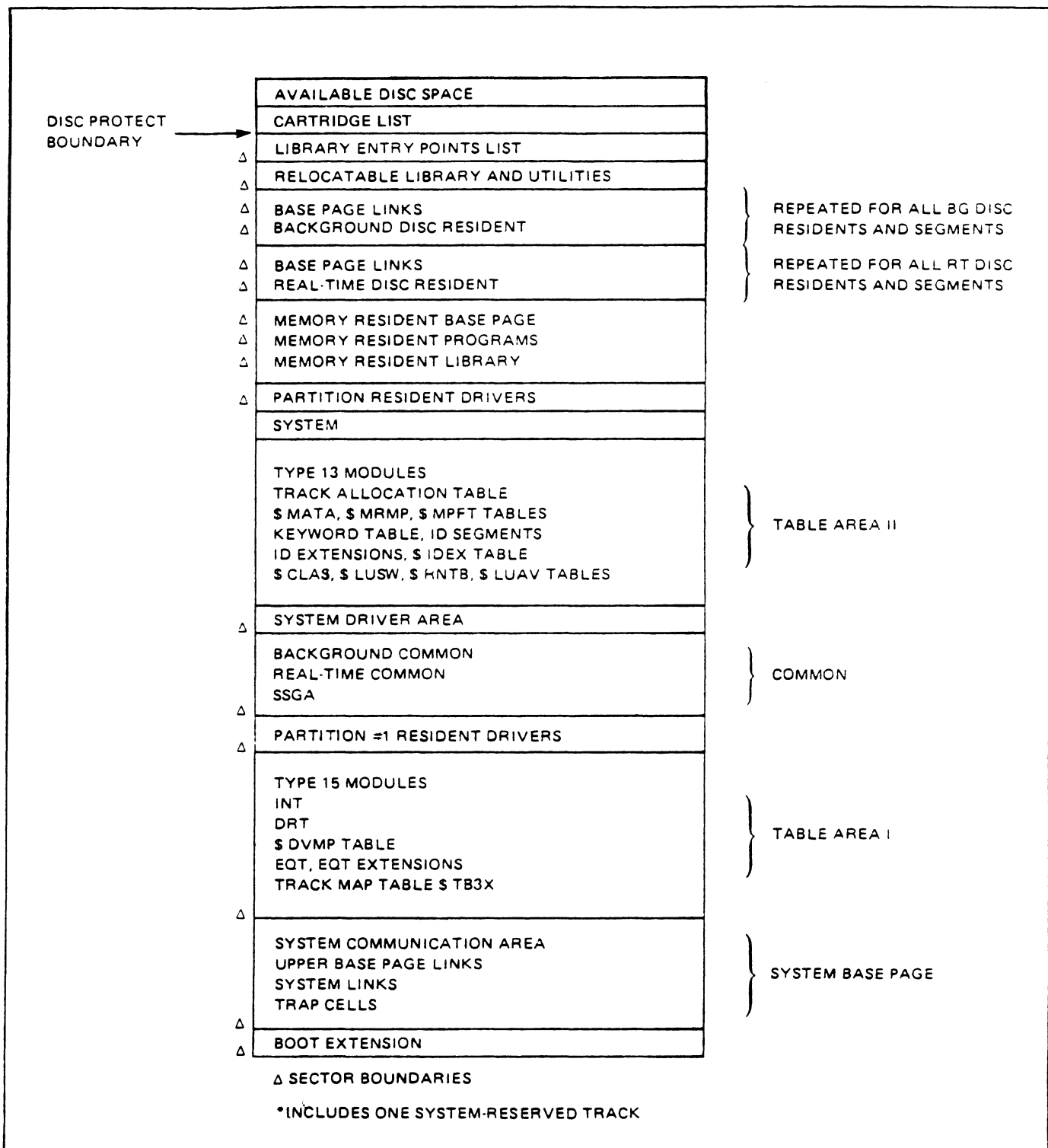
Figure B-3. RTE-IVB System Disc Layout

Table Area I and II Entry Points

TABLE AREA I entry points are as follows:

$ERAB	$PVCN	EXEC	XLUEX
$LIBR	$LIBX	$PVST	$UPIO
$XCIC	$YCIC	$CIC	$UIN
$UCON	$XEQ	$XDMP	$IDLE
$SCD3	$IDNO	$MEU	$LIST
$MESS	$WORK	$$OP	$ULLU
$CGRN	$MTM	$OPSY	$DATC
$CL1	$CL2	$$CPU	$CMAD
$ACFL	$LGON	$LGOF	$STH
LME	$DSCS	$SHED	$BITM
$MCON	$SMLK	$SMLN	$SMEX
$SMCA	$SMER	$SMCP	$SMID
$SMGP	$SMST	$SMDL	$SMII
$SMD#	#SPLU		

TABLE AREA II entry points are as follows:

$MATA	$MCHN	$MBGP	$MRTP
$DLTH	$DVPT	$TIME	$BATM
$DLP	$PLP	$SSCT	$STRK
$ENDS	$MPFT	$BGFR	$RTFR
$IDEX	$MRMP	$MPS2	$EMRP
$MPSA	$SDA	$SDT2	$CMST
$COML	$CFR	$MNP	$DVMP
$RLB	$RLN	$SBTB	$OTAT
$OPRI	$SPCR	$ELTB	$PNTI
$MAXI	$SALI	$SRTI	$CES
$LMES	$SMEM	PVM00	PM00
$SUB2	$DIGL	BRB	$$DLS
$SPCL	$SPOK	$IS43	

Appendix C
I/O Tables and Processing

This Appendix contains information on the following topics:

*   EQUIPMENT TABLE (EQT) - EQT entry format

*   DEVICE REFERENCE TABLE (DRT) - DRT table format and entry format.

*   DRIVER MAPPING TABLE (DMT) - DMT use and entry format.

*   INTERRUPT TABLE AND TRAP CELLS - Uses and contents.

*   POWER FAIL/AUTO RESTART - Function and general flow.

*   STANDARD I/O REQUEST FLOW - General flow of standard I/O request and general flow diagram.

Equipment Table (EQT)

The Equipment Table (EQT) maintains a list of all the I/O equipment in the system. The table consists of 15-word entries, with one 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.

Some information contained in the EQT entry is static, i.e., fixed at generation time or I/O reconfiguration and not changed during on-line operation. Other information is dynamic and can be changed on-line or is modified by the system to reflect various I/O conditions. Word 1 of the EQT entry contains a pointer to the linked list of I/O request buffers pending on the EQT (see Figure C-1 and C-2 below).

All Equipment Table entries are located sequentially in Table Area I, 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 System Base Page Communications area, location 1650B and 1651B, respectively.

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 the 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 reset to m by entering (T=) at generation time (see RTE-IVB On-Line Generator Reference Manual).

2. By use of the TO operator command (see RTE-IVB Terminal User's Reference Manual).

3. By the driver (see RTE Driver Writing Reference Manual).

For privileged drivers, the time-out parameter must be long enough to cover the period from I/O initiation to transfer completion.

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 <A>	B <B>	P <E>	S <E>	T <C>	SUBCHANNEL <C>					I/O SELECT CODE # <A>					
5	AV <F>		EQUIPMENT TYPE CODE <A>						STATUS <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 C-1. Equipment Table Entry Format

where:

R = reserved for system use.

```
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 Figure C-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

```
+---|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 15| 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|

| Status |                 |   Subfunction     |              Function
|---|----|                 |----------|----------|              |---|---|
    |                                 |                              |
```

00 - standard call	00000 = clear controller	01-READ call
01 - buffered call	(if function = 11 =	10-WRITE call
10 - System	CONTROL call)	11-CONTROL call
11 - Class call		
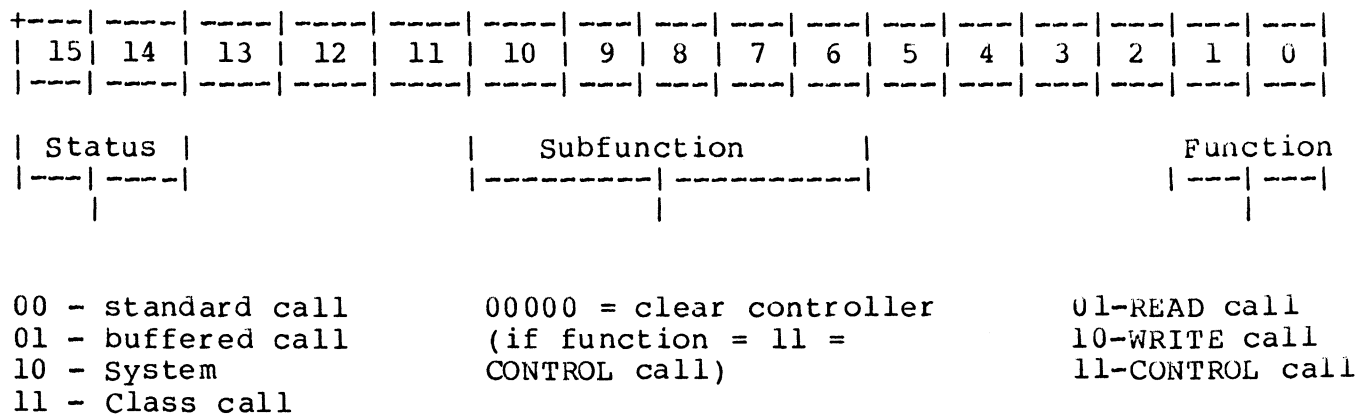
Other subfunctions are
driver specific and may
or may not be defined

Figure C-2.   CONWD   Word (EQT Entry Word 6) Expanded

Device Reference Table (DRT)

The Device Reference Table (DRT) is used by the I/O processor (RTIOC) to relate LU numbers to EQT entries. When a user makes an I/O request specifying an LU number, RTIOC translates the LU into an EQT entry via the DRT. Since the EQT entry contains all the information necessary to operate the I/O device, the transfer can then be initiated.

Each DRT entry is two words long. 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.

There are separate tables for words 1 and 2. The word 2 table is located in memory immediately following the word 1 table. The starting address and length of the word 1 table are recorded in the System Base Page (1652B and 1653B, respectively). The format of the Device Reference Table is illustrated in Figure C-3 and C-4, below.
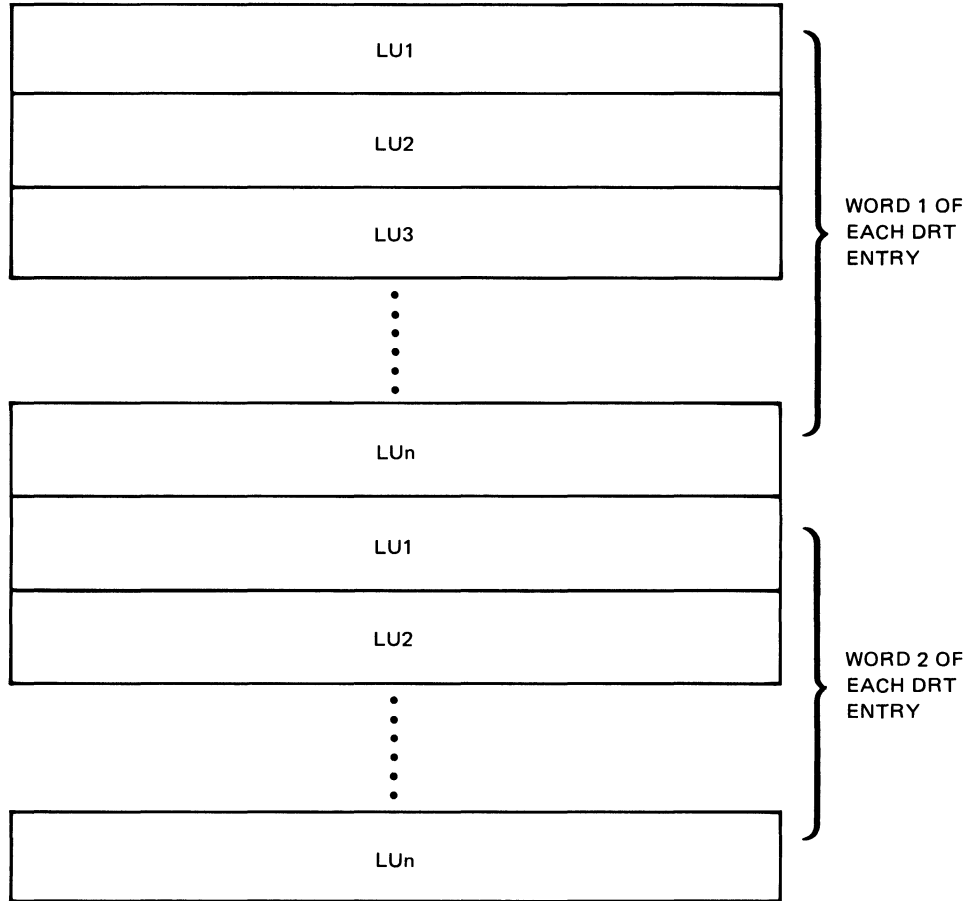
WHERE:

N = NUMBER OF LOGICAL UNITS IN SYSTEM

Figure C-3. Device Reference Table

SUBCHANNEL NO.					(RESERVED)					EQT ENTRY NUMBER						WORD 1
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
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 C-4.   Device Reference Table Entry Format

Driver Mapping Table (DMT)

In the RTE-IVB Operating System, driver modules can be placed in one of two areas; the System Driver Area (SDA) or in one of the driver partitions. Most drivers are placed in driver partitions. The SDA is primarily used for privileged drivers, drivers that do their own mapping, and very large drivers.

The Driver Mapping Table (DMT) is used to record where a driver resides in physical memory, static and dynamic information about the driver, and the location of the I/O request buffer.

There is one DMT entry associated with each EQT entry defined at generation time. Each entry is two words long. Word 1 is set up at generation time and its contents are never changed. It indicates whether the driver resides in the System Driver Area (SDA) or in a driver partition. If it is in the SDA, it also indicates whether or not the driver is doing its own memory mapping. If the driver is in a partition, word 1 also indicates the starting physical memory page number of the driver partition in which it is located.

Word 2 of the DMT entry is dynamic in nature and is set up at each I/O initialization of the associated EQT entry. This word indicates whether the I/O request buffer is located within a disc resident program, memory resident program, or system area. If a disc resident program is making the request and the I/O request buffer is located within the program (i.e., an unbuffered request), word 2 also indicates the physical memory page number of the disc resident program's base page. This information is used to save time on setting up the program map when processing interrupts handled by the driver. The format of the DMT is shown in Figure C-5, below.
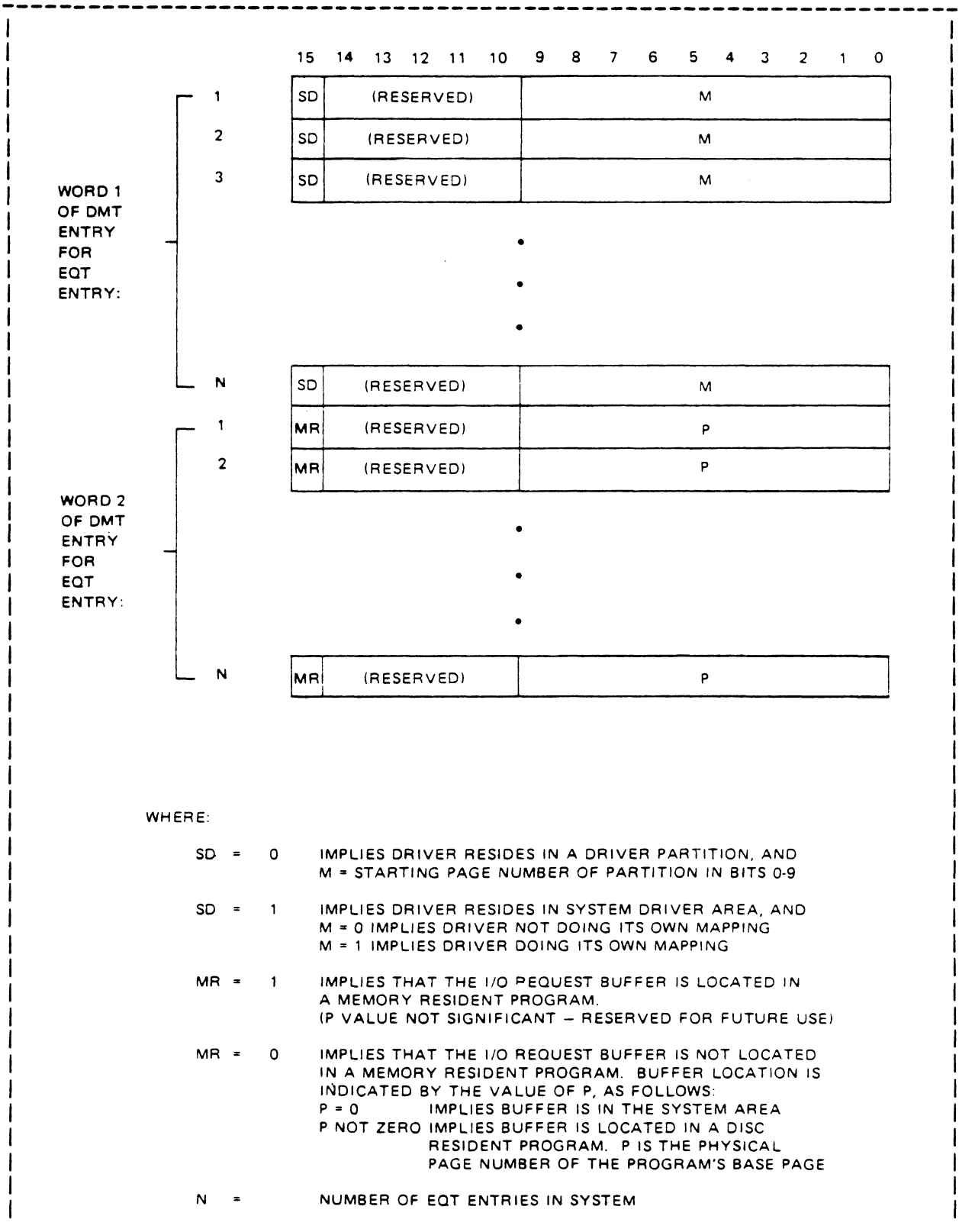
		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

WORD 1 OF DMT ENTRY FOR EQT ENTRY:

```
          15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0
       1 | SD |  (RESERVED)  |            M              |
       2 | SD |  (RESERVED)  |            M              |
       3 | SD |  (RESERVED)  |            M              |
                              •
                              •
                              •
       N | SD |  (RESERVED)  |            M              |
```

WORD 2 OF DMT ENTRY FOR EQT ENTRY:

```
       1 | MR |  (RESERVED)  |            P              |
       2 | MR |  (RESERVED)  |            P              |
                              •
                              •
                              •
       N | MR |  (RESERVED)  |            P              |
```

WHERE:

SD = 0   IMPLIES DRIVER RESIDES IN A DRIVER PARTITION, AND
          M = STARTING PAGE NUMBER OF PARTITION IN BITS 0-9

SD = 1   IMPLIES DRIVER RESIDES IN SYSTEM DRIVER AREA, AND
          M = 0 IMPLIES DRIVER NOT DOING ITS OWN MAPPING
          M = 1 IMPLIES DRIVER DOING ITS OWN MAPPING

MR = 1   IMPLIES THAT THE I/O REQUEST BUFFER IS LOCATED IN
          A MEMORY RESIDENT PROGRAM.
          (P VALUE NOT SIGNIFICANT — RESERVED FOR FUTURE USE)

MR = 0   IMPLIES THAT THE I/O REQUEST BUFFER IS NOT LOCATED
          IN A MEMORY RESIDENT PROGRAM. BUFFER LOCATION IS
          INDICATED BY THE VALUE OF P, AS FOLLOWS:
          P = 0   IMPLIES BUFFER IS IN THE SYSTEM AREA
          P NOT ZERO IMPLIES BUFFER IS LOCATED IN A DISC
                     RESIDENT PROGRAM. P IS THE PHYSICAL
                     PAGE NUMBER OF THE PROGRAM'S BASE PAGE

N =   NUMBER OF EQT ENTRIES IN SYSTEM

Figure C-5. Driver Mapping Table

Interrupt Table and Trap Cells

After an I/O request has been routed through the EXEC processor, the actual data transfer is accomplished by processing interrupts from the device's controller. The I/O controller generates an interrupt after each word transferred.

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, i.e., interrupts from select code 12 cause a transfer to memory location 12; interrupts from select code 13 cause a transfer to location 13, etc. Memory locations from octal 4-77 comprise the entire set of the 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 RTIOC, this instruction is a JSB LINK,I where LINK contains the address of the entry point to the Central Interrupt Controller (CIC). This instruction is initially set up at generation time.

After the CIC has been entered via the execution of a trap cell instruction, it checks the contents of the Interrupt Table entry associated with the select code that the interrupt occured on.

The Interrupt Table contains an entry, established at generation time, for each I/O select code in the computer. The entries can be positive, negative, or zero.

If the contents of the entry is positive, the entry contains the address of the EQT entry associated with the I/O controller in that select code.

If the contents of the entry is negative, the entry contains the negated ID segment address of the program to be scheduled when an interrupt occurs on that select code.

When the CIC has obtained the contents of the Interrupt Table entry associated with the interrupting select code, it initiates the appropriate action. For positive Interrupt Table entries, CIC obtains the driver continuation/completion entry point and transfers control to that point. For negative Interrupt Table entries, CIC transfers control to the scheduling module which schedules the program indicated by the ID segment address.

Interrupt Table entries that contain zero, indicate that CIC and the Interrupt Table are by-passed when an interrupt occurs on that select code. In this case, the trap cell contains the direct entry point to an interrupt processing routine. This technique would be used with time-critical events such as processing privileged interrupts or power fail interrupts. Table C-1, shows the three possible relationships between an Interrupt Table entry and the trap cell instruction associated with a specific select code.

Several conditions can occur during interrupt processing that cause a message to be displayed on the system console. These conditions are summarized at the end of Chapter 2.

Table C-1. Interrupt Table Example

GENERATION ENTRY (for SC 12)	INTERRUPT TABLE CONTENTS	TRAP CELL CONTENTS
12,EQT,1	EQT entry address	JSB LINK,I
12,PRG,name	Negative ID segment	JSB LINK,I
12,ENT,entry	0	JSB entry,I

The beginning of the Interrupt Table and the number of entries are stored in System Base Page locations 1654B and 1655B, respectively. The first entry in the table is for Select Code 6, (DCPC channel 1) and the second entry is for select code 7 (DCPC channel 2). These first two locations are dynamic in nature; they are changed each time a DCPC channel is assigned for a DMA transfer.

Power Fail/Auto Restart

The optional Power Fail/Auto Restart feature available with RTE-IVB is designed to save the computer status when the line voltage drops below a predetermined level (power fail). When a power fail occurs, the machine's status indicators are moved from their volatile hardware registers to memory locations where they can be maintained by a back-up battery power supply.

The Power Fail/Auto Restart routine is generated as a driver (DVP43) into the System Driver Area (SDA). DVP43 is a privileged driver that does its own mapping. The primary entry point to DVP43 is $POWR. The initiator and continuation/completion entry points are IP43 and CP43, respectively. Note that the CPU switch (A1S2) also must be set properly in order for the Power Fail/Auto Restart routine to execute when a power fail occurs.

The following steps illustrate the action taken by the system when a power fail occurs:

1. When power drops below a predetermined level, (or is restored), the condition is detected by the hardware power-sensing circuits of the HP-1000 computers. The hardware circuits cause an interrupt to occur on select code 4 which cause the instruction in Trap Cell 4 to be executed (indirect jump to $POWR).

2. When DVP43 is entered via $POWR, a check is made to determine if power is going down or coming up and the appropriate section of code is jumped to accordingly.

DOWN ROUTINES:

a) DMA transfers are stopped and the word counts are saved.

b) The registers (for example, A, B, O, E, X, Y, memory status register) and the point of power fail are saved.

c) The four memory maps are saved.

d) The power fail logic is reset so the next interrupt on select code 4 will be considered a power-up.

e) Halt.

UP ROUTINES:

a) Check if power down routine had time to complete. If it did, continue; if it didn't, halt.

b) Restore memory maps and memory status register.

c) Search the Equipment Table (EQT) for the DVP43 entry; set its time out counter to -1 (time-out on first tick of system clock).

d) Save the system time (the clock has not been restarted yet, so this is the time of powerfail).

e) Determine state of interrupt system when power failed, and set up to restore when DVP43 was exited.

f) Restore registers and start system clock.

g) Exit to point of power-fail.

3. After the exit from DVP43, the first tick of the system clock will time out DVP43s EQT entry, and DVP43 is entered at CP43 (continuation/completion entry point). The code at CP43 does the following:

a) Sets up DVP43 to time out in one tick of system clock.

b) Searches the Equipment Table to determine the state of the interrupt system at the time of power fail and takes appropriate action as follows:

* If the EQT entry was busy (AV=2) and its power fail bit ("P") set, enter the driver at InXX. The driver's initiator section notes that AV=2 and thus the request is for a power-failure and the driver takes appropriate action.

* If the EQT entry was waiting for a DCPC channel (AV=3), no action is taken.

* $UPIO is called for all other EQT entries to restart requests that were in progress or were pending by calling each driver at InXX.

c) After an EQT entry has been serviced, CP43 (or $UPIO) "idles", waiting for the system clock to tick and time out DVP43 again (reenter DVP43 at CP43).

4. When all the EQT entries have been serviced, a FORTRAN program (AUTOR) is aborted (could have been scheduled from a previous power-fail) and then scheduled. AUTOR is written in FORTRAN to allow for easy user modification to meet site-dependent requirements. AUTOR does the following:

a) Does an EXEC read on the power fail LU (enter at IP43) to get the power fail time.

b) Formats the time into a message and writes it to each interactive terminal.

c) Reenables all terminals.

d) Does a second EXEC read to the power fail LU to signal that the recovery process is complete.

Standard I/O Request Flow

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 program is suspended until the transmission is completed. The next lower priority program is allocated execution time during the suspension of a higher priority program.

An I/O request (READ, WRITE, CONTROL) is channeled to the I/O processor (RTIOC) by the EXEC processor. After the necessary legality checks are made, the request is linked into the request list associated with the I/O controller's EQT entry.

If the device's controller is available (i.e., no prior requests 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 since their I/O buffers are located in SAM.

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 on return from the initiation section or 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) and a diagnostic message is sent to the users teminal. The program is swappable while in this list. If a down LU or EQT entry is set UP, the program is automatically rescheduled.

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. At the end of the operation, the driver returns to RTIOC.

RTIOC causes the requesting program to be placed back into the scheduled list and checks for additional requests queued on the EQT. If there are no queued requests RTIOC exits to the dispatching module; otherwise, the initiation section is called to begin the next operation before returning.

Figure C-6 shows the various tables and control modules involved in standard I/O processing.

Figure C-6. Unbuffered EXEC Read Request Flow

7700-140
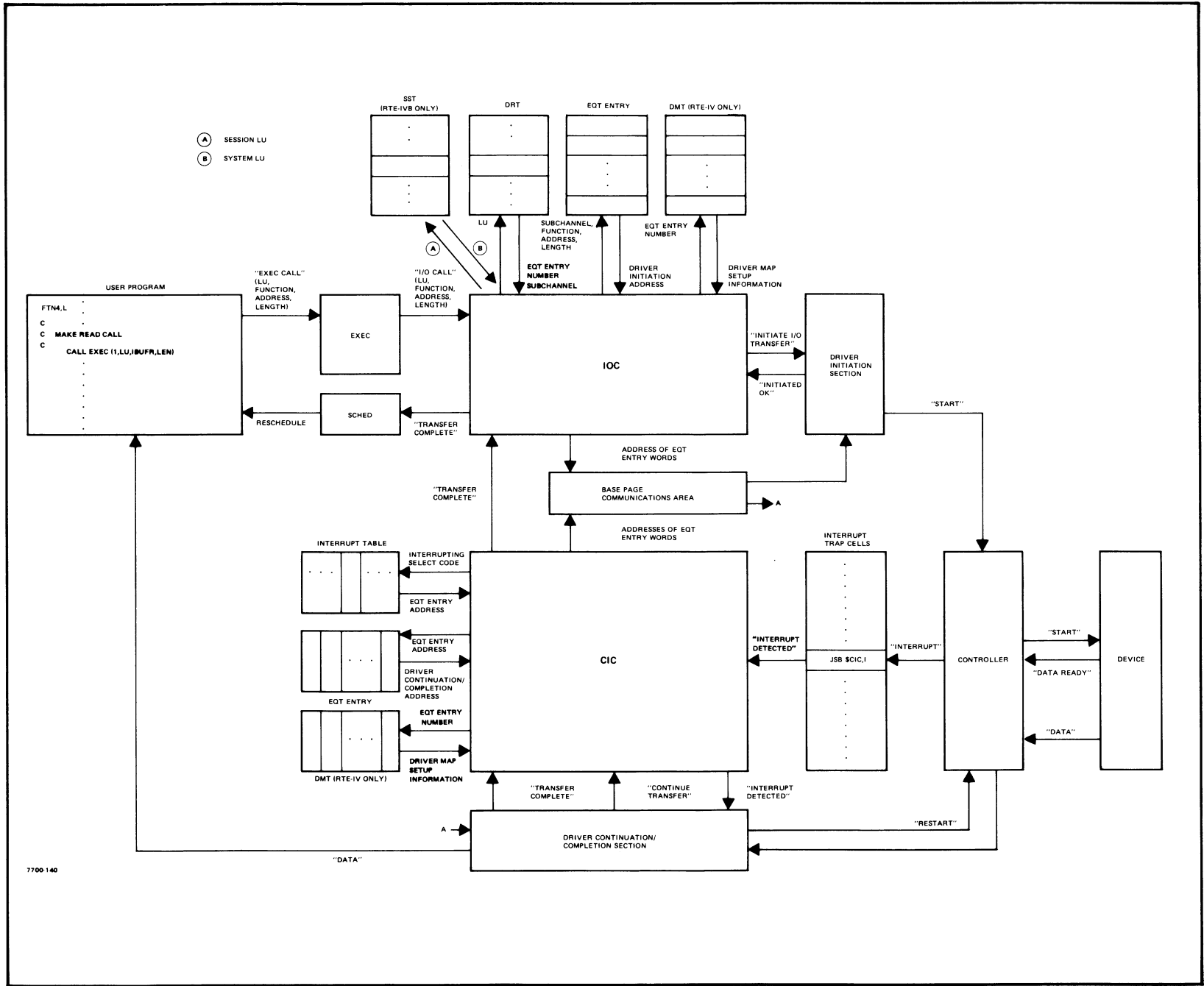
Appendix D
Record Formats

This Appendix contains information on the following:

*   SOURCE RECORD FORMAT

*   RELOCATABLE AND ABSOLUTE RECORD FORMATS

*   ABSOLUTE TAPE FORMAT

*   DISC FILE RECORD FORMATS

*   SIO TAPE RECORD FORMATS

*   MEMORY-IMAGE PROGRAM FILE FORMAT (TYPE 6)

Source Record Format

The source format used for the disc records by the system program
EDITR and FMGR is given in Figure D-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
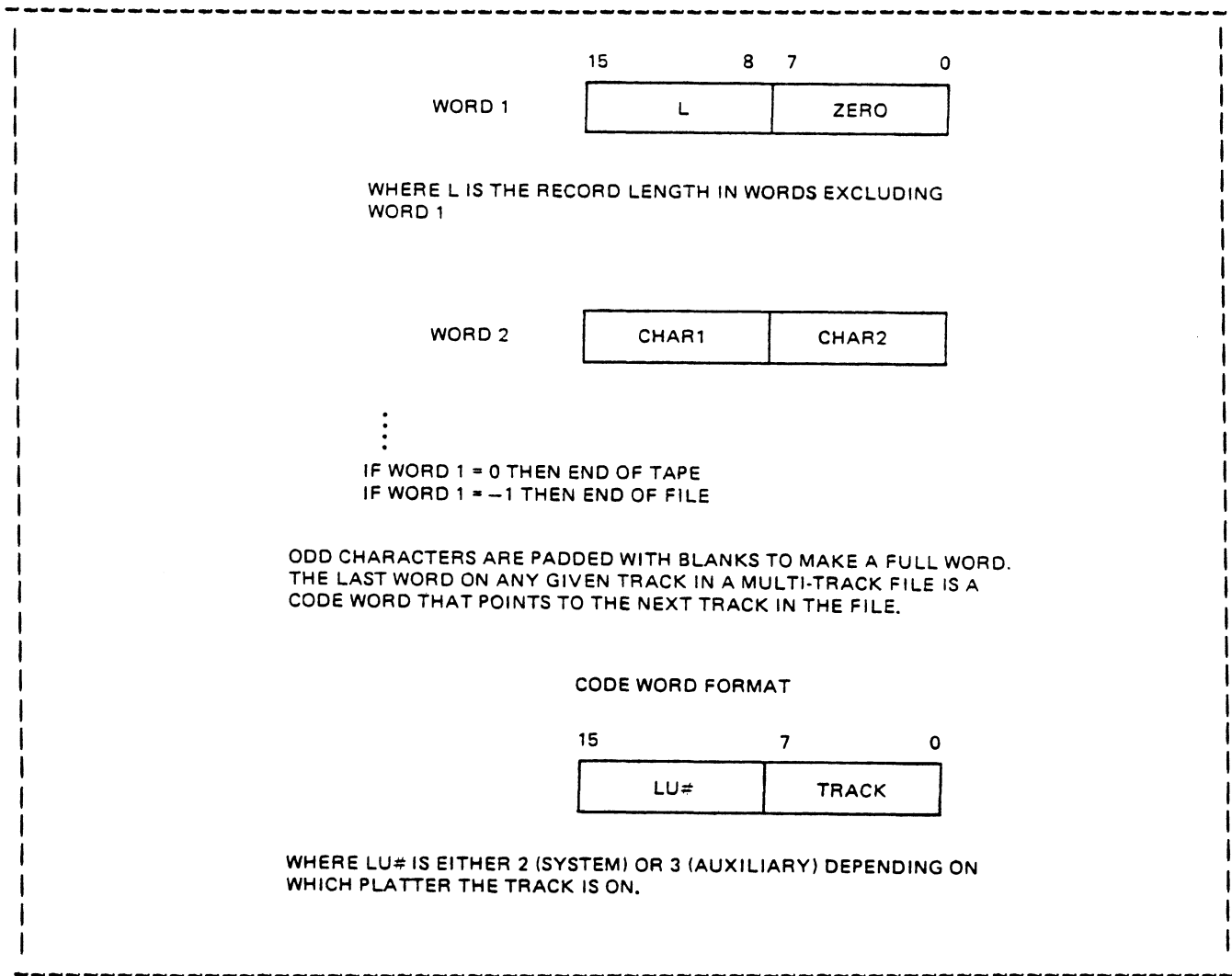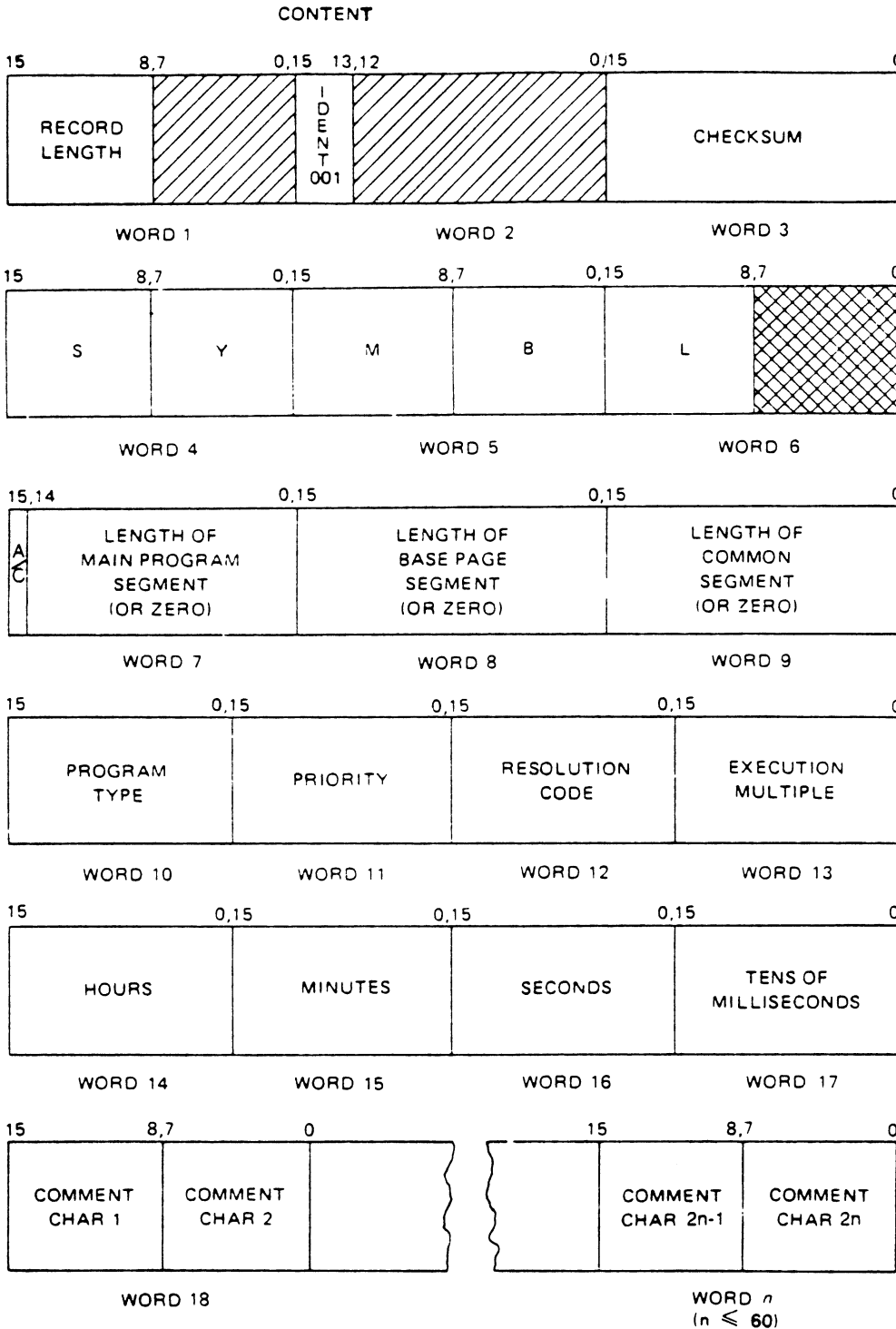code word is not required.

```
 15          8  7              0
         ┌──────────┬──────────┐
WORD 1   │    L     │   ZERO   │
         └──────────┴──────────┘
```

WHERE L IS THE RECORD LENGTH IN WORDS EXCLUDING
WORD 1

```
         ┌──────────┬──────────┐
WORD 2   │  CHAR1   │  CHAR2   │
         └──────────┴──────────┘
```

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

```
 15              7              0
         ┌──────────┬──────────┐
         │   LU≠    │  TRACK   │
         └──────────┴──────────┘
```

WHERE LU≠ IS EITHER 2 (SYSTEM) OR 3 (AUXILIARY) DEPENDING ON
WHICH PLATTER THE TRACK IS ON.

Figure D-1. Source Record Formats

Relocatable and Absolute 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

```
15        8,7        0,15  13,12              0/15              0
┌──────────┬─────────┬─────┬───────────────────┬──────────────────┐
│          │/////////│ I   │///////////////////│                  │
│ RECORD   │/////////│ D   │///////////////////│    CHECKSUM      │
│ LENGTH   │/////////│ E   │///////////////////│                  │
│          │/////////│ N   │///////////////////│                  │
│          │/////////│ T   │///////////////////│                  │
│          │/////////│ 001 │///////////////////│                  │
└──────────┴─────────┴─────┴───────────────────┴──────────────────┘
    WORD 1              WORD 2                     WORD 3
```

RECORD LENGTH = 9-60 WORDS

IDENT = 001

CHECKSUM ARITHMETIC
TOTAL OF ALL WORDS
IN RECORD EXCLUDING
WORDS 1 AND 3.

```
15        8,7        0,15        8,7        0,15        8,7        0
┌──────────┬─────────┬──────────┬──────────┬──────────┬──────────┐
│          │         │          │          │          │XXXXXXXXXX│
│    S     │    Y    │    M     │    B     │    L     │XXXXXXXXXX│
│          │         │          │          │          │XXXXXXXXXX│
└──────────┴─────────┴──────────┴──────────┴──────────┴──────────┘
     WORD 4              WORD 5                 WORD 6
```

SYMBL FIVE CHARACTER
NAME OF PROGRAM

```
15,14                0,15                    0,15                  0
┌─┬─────────────┬──────────────────┬──────────────────┐
│A│  LENGTH OF  │    LENGTH OF     │    LENGTH OF      │
│C│ MAIN PROGRAM│    BASE PAGE     │     COMMON        │
│ │  SEGMENT    │    SEGMENT       │    SEGMENT        │
│ │  (OR ZERO)  │    (OR ZERO)     │    (OR ZERO)      │
└─┴─────────────┴──────────────────┴──────────────────┘
     WORD 7            WORD 8              WORD 9
```

A/C: BINARY TAPE PRECESSION
  = 0 IF ASSEMBLER
    PRODUCED OR LENGTH
    IS EXACT
  = 1 IF COMPILER
    PRODUCED, AND
    LENGTH IS UNKNOWN

```
15            0,15            0,15            0,15            0
┌──────────────┬───────────────┬───────────────┬──────────────┐
│              │               │  RESOLUTION   │  EXECUTION   │
│ PROGRAM      │   PRIORITY    │     CODE      │  MULTIPLE    │
│ TYPE         │               │               │              │
└──────────────┴───────────────┴───────────────┴──────────────┘
   WORD 10         WORD 11          WORD 12          WORD 13
```

```
15            0,15            0,15            0,15            0
┌──────────────┬───────────────┬───────────────┬──────────────┐
│              │               │               │   TENS OF    │
│  HOURS       │   MINUTES     │   SECONDS     │ MILLISECONDS │
│              │               │               │              │
└──────────────┴───────────────┴───────────────┴──────────────┘
   WORD 14         WORD 15          WORD 16          WORD 17
```

```
15        8,7        0                    15        8,7        0
┌──────────┬──────────┬────┐        ┌────┬──────────┬──────────┐
│ COMMENT  │ COMMENT  │    │        │    │ COMMENT  │ COMMENT  │
│ CHAR 1   │ CHAR 2   │    │        │    │ CHAR 2n-1│ CHAR 2n  │
└──────────┴──────────┴────┘        └────┴──────────┴──────────┘
         WORD 18                            WORD n
                                           (n ≤ 60)
```

▨ 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

Figure D-2. Record Formats

ENT RECORD

CONTENT   EXPLANATION

15	8,7	0,15 13,12	4,3	0,15	0

```
RECORD          //////////  I   //////////  E   CHECKSUM
LENGTH                      D               N
                           E               T
                           N               R
                           T               I
                          010              E
                                           S
```

WORD 1   WORD 2   WORD 3

RECORD LENGTH = 7-59 WORDS

IDENT = 010

ENTRIES: 1 TO 14 ENTRIES
    PER RECORD. EACH ENTRY
    IS FOUR WORDS LONG

15	8,7	0,15	8,7	0,15	8,7	3,2	0

```
  S      Y        M      B        L    //////// R
```

WORD 4   WORD 5   WORD 6

SYMBL 5 CHARACTER ENTRY
    POINT SYMBOL

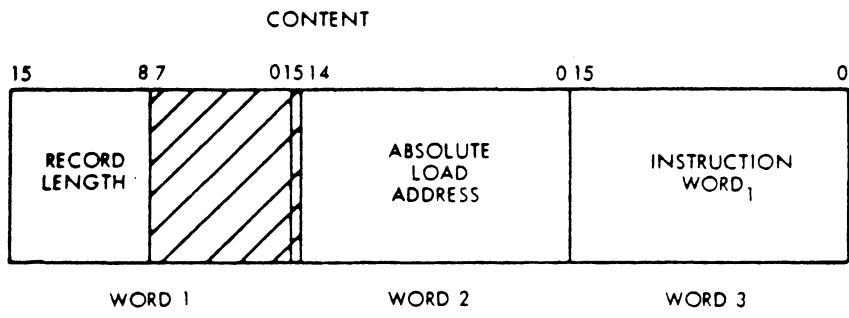R RELOCATION INDICATOR
  = 0 IN PROGRAM RELOCATABLE
  = 1 IF BASE PAGE RELOCATABLE
  = 2 IF COMMON RELOCATABLE
  = 3 IF ABSOLUTE
  = 4 INSTRUCTION REPLACEMENT

15	0,15	8,7	0,15	8,7	0

```
UNRELOCATED
ADDRESS              S      Y        M      B
FOR SYMBL OR
REPLACEMENT INSTRUCTION
VALUE
```

WORD 7   WORD 8   WORD 9

WORDS 4 THROUGH 7 ARE
REPEATED FOR EACH
ENTRY POINT SYMBOL.

15	8,7	3,2 0 15	0,15	0

```
  L    //////// R           UNRELOCATED
                            ADDRESS
                            FOR SYMBL OR
                            REPLACEMENT INSTRUCTION
                            VALUE
```

WORD 10   WORD 59

Figure D-2. Record Formats (continued)

EXT RECORD

CONTENT   EXPLANATION

RECORD LENGTH = 6-60 WORDS

IDENT = 100

ENTRIES. 1 TO 19 PER
RECORD, EACH ENTRY
IS THREE WORDS LONG

RECORD LENGTH	/////	IDENT 100	/////	ENTRIES	CHECKSUM

WORD 1   WORD 2   WORD 3

S	Y	M	B	L	SYMBOL I.D NO.

WORD 4   WORD 5   WORD 6

SYMBL. 5 CHARACTER
EXTERNAL SYMBOL

SYMBOL ID. NO.: NUMBER
ASSIGNED TO SYMBL FOR
USE IN LOCATING
REFERENCE IN BODY
OF PROGRAM.

S	Y		L	SYMBOL I.D. NO.

WORD 7   WORD 60

WORDS 4 THROUGH 6 REPEATED
FOR EACH EXTERNAL
SYMBOL (MAXIMUM OF
19 PER RECORD).

Figure D-2. Record Formats (continued)

DBL RECORD

CONTENT

EXPLANATION

15   8,7   0,15, 13,12   8,7 6,5   0,15   0

| RECORD LENGTH | | I D E N T | | Z C | NO. OF INST. WORDS | CHECKSUM |

WORD 1   WORD 2   WORD 3

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

15   0,15 13,12 10,9 7,6 4,3 1,0 15   0

| UNRELOCATED LOAD ADDRESS | R | R | R | R | R | | ABSOLUTE VALUE |

WORD 4   WORD 5   INSTRUCTION WORD
                                        R = 000
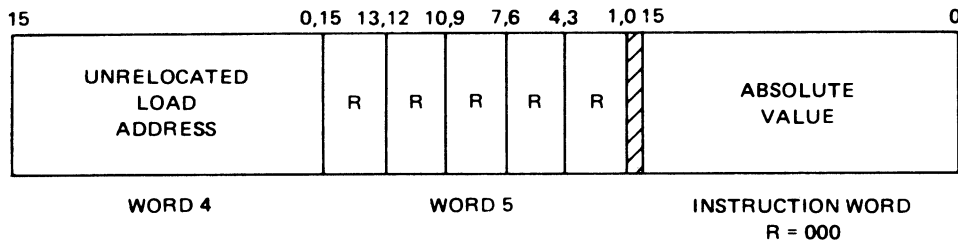
RELOCATABLE LOAD ADDRESS:
STARTING ADDRESS FOR
LOADING THE INSTRUCTIONS
WHICH FOLLOW;

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

15,14   0,15,14   0,15,14   0

| 15-BIT PROGRAM RELOCATABLE VALUE | 15-BIT BASE PAGE RELOCATABLE VALUE | 15-BIT COMMON RELOCATABLE VALUE |

D/I   D/I   D/I
INSTRUCTION WORD   INSTRUCTION WORD   INSTRUCTION WORD
R = 001   R = 010   R = 011
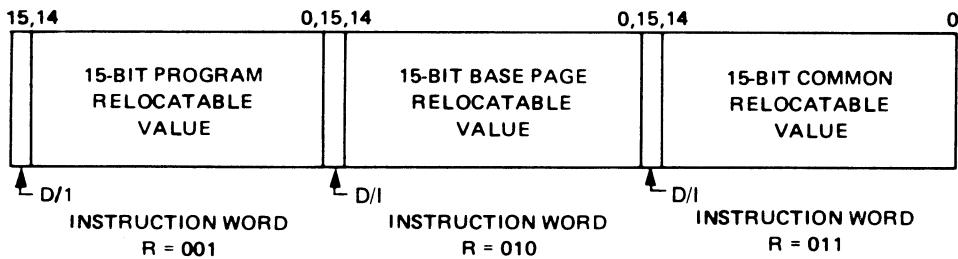
R_1 IS RELOCATION INDICATOR
FOR INSTRUCTION $WORD_1$; R_2,
FOR INSTRUCTION $WORD_2$; ETC.

15,14 11,10 8,7   0,15,14 11,10,9   2,1,0, 15   0

| I N S T R U C T | C O D E | | EXTERNAL SYMBOL I.D. NO. | I N S T R U C T | C O D E | | EXTERNAL SYMBOL I.D. NO. —OR— ZERO | M R | UNRELOCATED VALUE —OR— OFFSET |

D/I   D/I
INSTRUCTION WORD   INSTRUCTION WORD
R = 100   R = 101
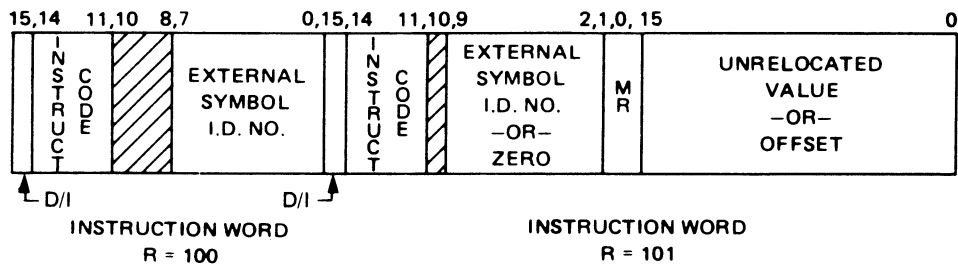
D/I: INDIRECT ADDRESSING

0 = DIRECT
1 = INDIRECT

MEMORY REFERENCE INSTRUC-
TIONS 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

15   12 11   2 10 15   0

| TYPE | | | M R | RELOCATABLE BYTE ADDRESS |

INSTRUCTION WORD   R = 110
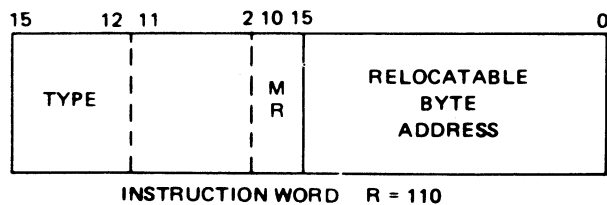
Figure D-2.   Record Formats   (continued)

EMA RECORD

WORD 1 — WORD 2 — WORD 3
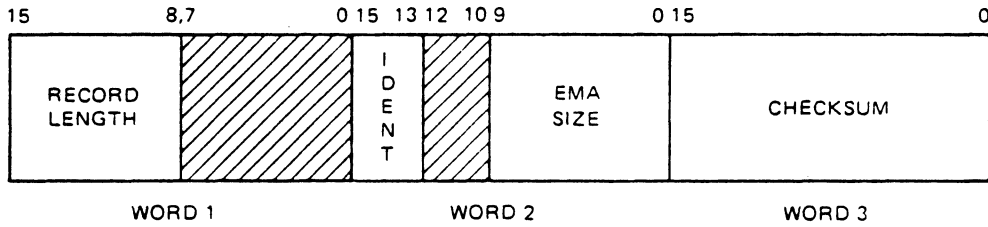
EXPLANATION

RECORD LENGTH = 7 WORDS
IDENT = 110
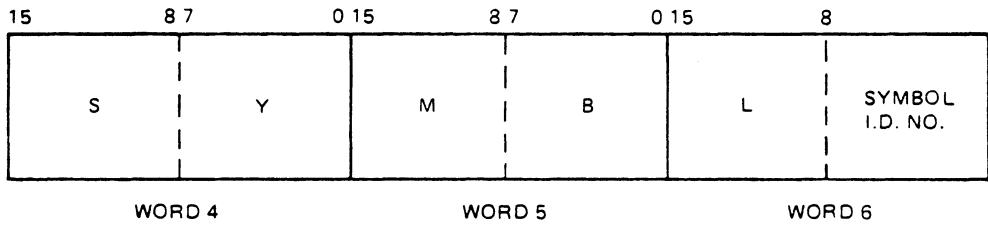
SYMBOL I.D. NO.: NUMBER
ASSIGNED TO SYMBOL FOR
USE IN LOCATING REFERENCE
IN COPY OF PROGRAM.

WORD 4 — WORD 5 — WORD 6

WORD 4

END RECORD

CONTENT

WORD 1 — WORD 2 — WORD 3
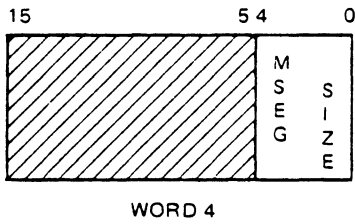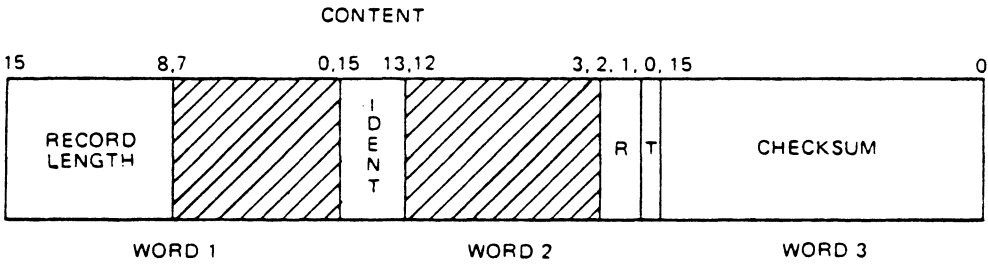
EXPLANATION

RECORD LENGTH = 4 WORDS
IDENT = 101
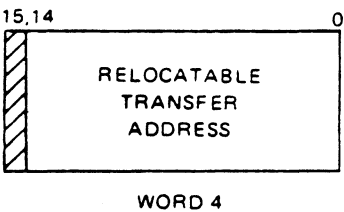
R: RELOCATION INDICATOR
FOR TRANSFER ADDRESS

= 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

RELOCATABLE
TRANSFER
ADDRESS

WORD 4

Figure D-2.   Record Formats   (continued)

Absolute Tape Format

CONTENT

EXPLANATION

15	8 7	0 15 14	0 15	0
RECORD LENGTH	////////	ABSOLUTE LOAD ADDRESS	INSTRUCTION WORD₁	
WORD 1		WORD 2	WORD 3	

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

15	0 15	0 15	0
	INSTRUCTION WORD$_i$	CHECKSUM	
	WORD n - 1	WORD n	
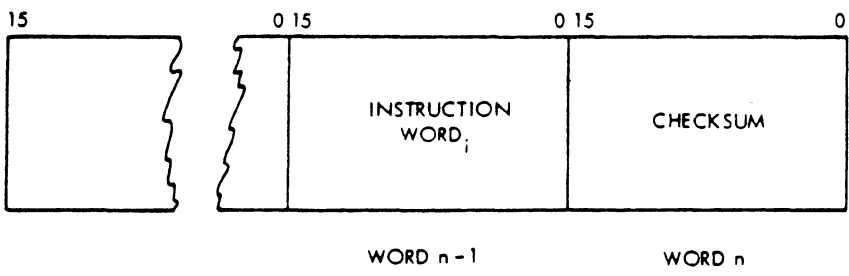
INSTRUCTION WORDS: ABSOLUTE INSTRUCTIONS OR DATA

CHECKSUM: ARITHMETIC TOTAL OF ALL WORDS EXCEPT FIRST AND LAST

---

† On paper tape, each word represents two frames arranged as follows:

Bit 8 → ▸ Bit 0

→ Feed Holes

Bit 15 → ▸ Bit 7
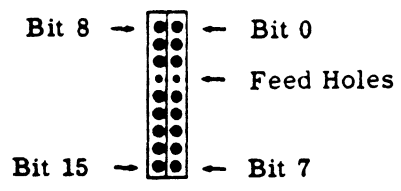
Disc File Record Formats

Fixed Length Formats (Types 1 and 2)

1st 127 WORDS OF LAST BLOCK

| BLOCK 1 | BLOCK 2 | BLOCK 3 | BLOCK 4 | BLOCK N |

128 WORDS

EOF FOLLOWS LAST
WORD IN LAST BLOCK
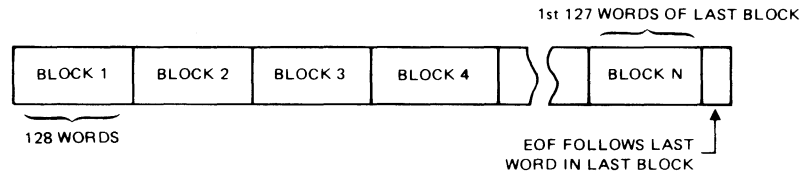
Type 1 Record length = Block length = 128 words

Type 2 Record length is user defined; may cross block boundaries but not past EOF

Variable Length Formats (Types 3 and Above)

RECORD 1   RECORD 2   RECORD 3   RECORD 4   RECORD 5   RECORD N

| L | DATA | L | L | DATA | L | L | DATA | L | 0 | 0 | L | DATA | L | | L | DATA | L | -1 |

LENGTH WORDS

ZERO-LENGTH
RECORDS
(SUB-FILE MARK)

EOF = -1
IN FIRST LENGTH
WORD OF NEXT
RECORD
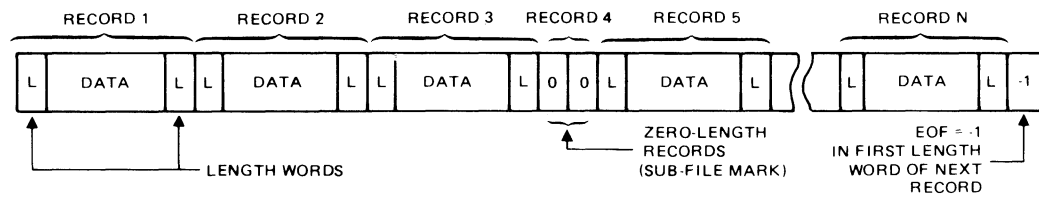
SIO Record Format

Magnetic tape SIO binary records have the following format:
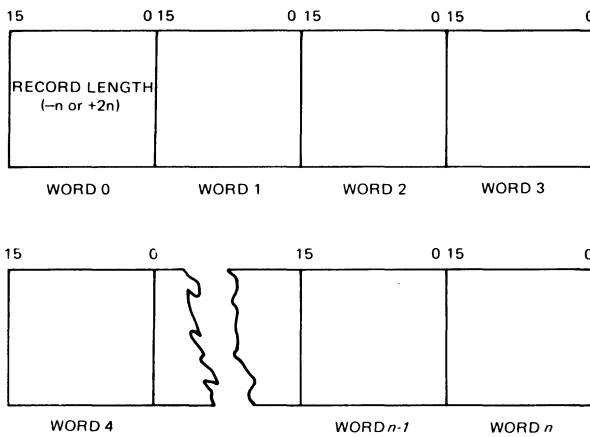
Record length = number of words or characters in record, excluding word 0; negative value denotes words, positive value denotes characters.

NOTE

The length (word 0) is not considered part of the data record. When written with the MS option of the DU command, the length is supplied by FMGR. When read with the MS option of the ST command, the length is removed (in this case, the length word is used instead of the length supplied by the driver).

Memory-Image Program File Format (Type 6)

Files created by the SP command as memory-image program files are always accessed as type 1 files (fixed length, 128-words per record).

WORD	CONTENT	
0	−1	EOF UNLESS FORCED TO TYPE 1
1-5	NOT USED	
6	PRIORITY	
7	PRIMARY ENTRY POINT	
8-11	NOT USED	
12-13	ORIGINAL PROGRAM NAME	
14	PROGRAM TYPE	
15-16	NOT USED	
17-19	TIME PARAMETERS	
20	SUBSTATUS 1 - WORD 20 OF ID SEGMENT	
21	SUBSTATUS 2 - WORD 21 OF ID SEGMENT	
22	LOW MAIN ADDRESS	
23	HIGH MAIN ADDRESS + 1	
24	LOW BASE-PAGE ADDRESS	
25	HIGH BASE-PAGE ADDRESS + 1	
26	PROGRAM TRACK	
27	SWAP TRACK	
28	ID EXT. #/EMA SIZE	
29	HIGH ADDRESS + 1 OF LARGEST SEGMENT	
30	NOT USED	
31	OPEN FLAG WORD	
32	NOT USED	
33	MLS WORD 1	
34	MLS WORD 2	
35	SECTOR/LU OF PROGRAM	
36	CHECKSUM OF WORDS 0 - 32	SUM OF CONTENTS OF WORDS 1650 THRU 1657 AND WORDS 1742 THRU 1747 AND 1755 THRU 1764 IN BASE PAGE
37	SETUP CODE WORD	
38	ID EXTENSION - WORD 0	
39	ID EXTENSION - WORD 1	
40	ID EXTENSION - WORD 2	
41	ID EXTENSION - WORD 3	
42	ID EXTENSION - WORD 4	
43-45	SHARED EMA NAME	
46	OWNER ID	IF SIGN BIT SET, PROGRAM FILE PROTECTED TO THIS USER ID
47	OWNER'S GROUP ID	IF SIGN BIT SET, PROGRAM FILE PROTECTED TO THIS GROUP ID
48	CAPABILITY LEVEL REQUIRED	MINIMUM CAPABILITY REQUIRED TO RU OR RP THIS PROGRAM.
49-112	NOT USED	
113-123	TIME TYPE-6 FILE CREATED	
124-127	NOT USED	

WORDS 0-35 AND 38-42 CONTAIN PROGRAM'S ID-SEGMENT INFORMATION

REMAINDER OF FILE IS AN EXACT COPY OF THE PROGRAM BEING SAVED.

8200-162

```
                    ( RTE-II )    ( RTE-III )  ( RTE-IV )     ( RTE-IVB )

Maximum number                  ┌───┐                      ┌────┐
of partitions          ─ ─ ─ ─ ─│ 2 │─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─│ 64 │
                                └───┘                      └────┘

Maximum physical          ┌───────────┐         ┌───────────┐
memory allowed      ─ ─ ─·│ 64 K bytes│─ ─ ─ ─ ─│ 2 M bytes │
                         └───────────┘         └───────────┘
                                                            IV,IVB

                                             III
Maximum partition        ┌──────┐  ┌─────────────────┐  ┌─────────────────┐
size (without EMA) ─ ─ ─│ N.A. │─ │ 64 K bytes minus │  │ 64 K bytes minus │
                        └──────┘  │ Opertaion System,│  │ Table Area I,    │
                                  │ System Base Page, │  │ Driver Partition,│
                                  │ and Memory Resident│ │ and program's base│
                                  │ Library.         │  │ page.            │
                                  └─────────────────┘  └─────────────────┘

Maximum partition        ┌──────┐          ┌────────────────────┐
size (with EMA)    ─ ─ ─│ N.A. │─ ─ ─ ─ ─ │ Physical memory minus│
                        └──────┘          │ items listed above.  │
                                          └────────────────────┘
                                                         IVB
                            III,IV
Timeslicing             ┌────┐                        ┌─────┐
available          ─ ─ ─│ NO │─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─│ YES │
                        └────┘                        └─────┘
                                    IV                      IVB
                          II,III
Memory and I/O          ┌────┐                        ┌─────┐
reconfiguration    ─ ─ ─│ NO │─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─│ YES │
available               └────┘                        └─────┘

                          II,III                       IV,IVB
EMA capabilities        ┌────┐                        ┌─────┐
available          ─ ─ ─│ NO │─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─│ YES │
                        └────┘                        └─────┘
```

```
                        II,III                        IV,IVB
                          |                              |
                          v                              v
Type 4 programs  >- - - - [  NO  ]- - - - - - - - - - -[  YES  ]
available
                          |                              |
                          v                              v
Program segmentation >- - +------------------+    +------------------------+
capabilities              | for Type 3 programs|- -| for Type 2,3, and 4   |
available                 | only              |    | programs              |
                          +------------------+    +------------------------+
                          |                              |
                          v                              v
Memory Resident  >- - +----------------------+    +----------------------+
Library routines      | callable from disc   |    | callable from only   |
                      | and memory resident  |- - | memory resident      |
                      | prgroams and other M.R.|  | programs and other   |
                      | routines             |    | M.R. routines        |
                      +----------------------+    +----------------------+
                          |                              |
                          v                              v
Action taken by  >- - - [ HALT ]- - - - - -  +--------------------------------+
system when parity                           | If user map enabled:           |
error occurs in a                            |   1. report error              |
partition                                    |   2. remove partition          |
                                             |   3. HALT                      |
                                             | If system map enabled:         |
                                             |   1. HALT                      |
                                             +--------------------------------+
                          |                              |
                          v                              v
Driver Partitions                                       
and System Driver  >- - - [  NO  ]- - - - - - - - - - -[  YES  ]
Area (SDA) available                                  /       \
                          |                      IV /          \
                          v                        /            v
Routing of error   >- - - [  NO  ]<---------------+          [  YES  ]
messages to user's         /  \                                  |
terminal                  /    \ IV                              |
                     II,III    \                          IVB    v
                          v      \                                
Disc source and      +---------+  +------------------------+
destination areas    | LS and LG| | FMP disc files         |
for LOADR and language>- -| tracks  |- - - - - - - - - - -|                        |
translators          +---------+  +------------------------+
                          |                              |
                          v                              v
```

E-2

```
                            ↓ II              ↓ III              ↓ IV,IVB
                    ┌──────────────┐   ┌──────────────┐   ┌──────────────┐
Contents of logical │ System Base  │   │ System Base  │   │ Program Base │
memory with a disc  │ Page         │   │ Page         │   │ Page         │
resident program ⟩──│ System Com-  │   │ System Modules│  │ Table Area I │
executing           │ untication   │   │ M.R. Library │   │ Driver Partiton│
                    │ Area         │─ ─│ COMMON (opt.)│   │ COMMON (opt.)│
                    │ System code  │   │ Disc Resident│   │ SDA (opt.)   │
                    │ M.R. Library │   │ Program      │   │ Table Area II(opt.)│
                    │ RT COMMON    │   └──────────────┘   │ Disc Resident│
                    │ RT M.R. area │           ↓          │ Program      │
                    │ RT disc res- │   ┌──────────────┐   └──────────────┘
                    │ ident area   │   │ System Base  │           ↓
Contents of logical │ SAM          │   │ Page         │   ┌──────────────┐
memory with a memory│ BG COMMON    │   │ System Modules│  │ Program Base │
resident program ⟩─ │ BG M.R. area │─ ─│ M.R. Library │─ │ Page         │
executing           │ BG disc res- │   │ COMMON (opt.)│   │ Table Area I │
                    │ ident area   │   │ Memory Resident│ │ Driver Partition│
                    │ Basic Binary │   │ program      │   │ COMMON (opt.)│
                    │ Loader       │   └──────────────┘   │ SDA (opt.)   │
                    └──────────────┘           ↓          │ Table Area II (opt│
                            ╲              ┌──────────────┐│ M.R. Library │
Contents of logical          ╲            │ System Base  ││ Memory Resident│
memory with the  ⟩─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─ ─│ Page         ││ program      │
System executing             ╱            │ System Modules│└──────────────┘
                            ╱             │ M.R. Library │         ↓
                                          │ COMMON (opt.)│   ┌──────────────┐
                                          │ SAM          │─ ─│ System Base  │
                                          └──────────────┘   │ Table Area I │
                                                             │ Driver Part- │
                                                             │   ition      │
                                                             │ COMMON       │
                                                             │ SDA          │
                                                             │ Table Area II │
                                                             │ System Modules│
                                                             │ SAM          │
                                                             │ SAM extension │
                                                             └──────────────┘
```

```
            ↓ II                              ↓ III            ↓ IV,IVB
Saved on    ┌────────────────────────┐   ┌────────────────────────┐
interrupt   │ A,B,E,O registers      │   │ A,B,E,O,X,Y registers  │
by Priviledged⟩ Point of interrupt   │─ ─│ DMS status             │
Drivers     │ Memory Protect status  │   │ User map               │
            └────────────────────────┘   └────────────────────────┘
                        ↓                            ↓
Program types⟩─ ─┌──────────────┐      ┌────────────────────┐
                 │ 0 thru 14    │─ ─ ─ │ 0 thru 3           │
                 └──────────────┘      │ 5 thru 11          │
                                       │ 14,17,18,19        │
                             III       │ 25,26,27,30        │   IV,IVB
                          ↙            └────────────────────┘            ↘
              ┌────────────────────┐              ↓          ┌────────────────────┐
              │ plus the above types│             │          │ plus the above     │
              │ with 80 added       │             │          │ types with 80      │
              └────────────────────┘              │          │ or 128 added       │
                        │                          │          └────────────────────┘
Support of              │                          │                    │
fixed-head ⟩─ ─┌──────┐ │                ┌──────┐  │                     │
discs          │ YES  │─ ─ ─ ─ ─ ─ ─ ─ ─ │ NO   │ ←─────────────────────┘
               └──────┘                  └──────┘
                  ↓                          ↓
```

E-3

```
                              II                    III,IV,IVB
                               │                        │
                               ▼                        ▼
Interrupt trap  ╲___┌─────────────────┐___┌─────────────────┐
cell contents   ╱   │ no restrictions │   │  JMP or JSB only │
                    └─────────────────┘   └─────────────────┘
                             │                    │       │
                             │         ┌──────────┘       │ IVB
                             ▼         ▼   III,IV          ▼
Session Monitor ╲_ _ _ _ ┌──────┐_ _ _ _ _ _ _ _ _ ┌──────┐
                ╱        │  NO  │                    │ YES  │
                         └──────┘                    └──────┘
                             │                           │
                             ▼                           ▼
Maximum file    ╲_ _ _ _ ┌──────────────┐_ _ _ ┌──────────────────┐
size            ╱        │ 16383 blocks │      │ 32767 X 128 blocks│
                         └──────────────┘      └──────────────────┘
                             │                           │
                             ▼                           ▼
Maximum number of ╲_ _ ┌─────────┐_ _ _ _ _ _ ┌──────────────┐
records per file  ╱    │  32767  │            │  (2**31)-1   │
                       └─────────┘            └──────────────┘
                             │                           │
                             ▼                           ▼
D.RTR size ╲_ _ _ _ _ ┌──────────────────┐_ _ _ ┌──────────────────┐
           ╱          │ approx.  2 K words│      │ approx. 8 K words │
                      └──────────────────┘      └──────────────────┘
                             │          │               │
                             │          │ III,IV        │
                             ▼          ▼               ▼
COMMON usage ╲_ _ _ ┌──────────────────┐_ _ ┌──────────────────┐
             ╱      │ Main and Segments │    │ Main and Segments use│
                    │ can use different │    │ same COMMON       │
                    │ COMMON            │    │                   │
                    └──────────────────┘    └──────────────────┘
```

E-4

Table F-1. Summary of RTE-IVB Program Types

PROGRAM CATEGORY / EXECUTABLE PROGRAMS	PROGRAM TYPE	REAL-TIME COMMON	BACKGROUND COMMON	SSGA	RT COMMON & SSGA	BG COMMON & SSGA	EMA ALLOWED	LOAD POINT — NO COMMON DECLARED	LOAD POINT — SOME COMMON DECLARED	MEMORY PROTECT FENCE — NO COMMON DECLARED	MEMORY PROTECT FENCE — SOME COMMON DECLARED
MEMORY RESIDENT*	1	✓						L_1	L_1	F_5	F_3
	9		✓					L_1	L_1	F_5	F_4
	17			✓				L_1	L_1	F_1	F_1
	17				✓			L_1	L_1	F_1	F_1
	25					✓		L_1	L_1	F_1	F_1
REAL TIME DISC RESIDENT*	2	✓					✓	L_4	L_4	F_6	F_3
	10		✓				✓	L_4	L_4	F_6	F_4
	18			✓			✓	L_4	L_4	F_1	F_1
	18				✓		✓	L_4	L_4	F_1	F_1
	26					✓	✓	L_4	L_4	F_1	F_1
BACKGROUND DISC RESIDENT*††	3		✓				✓	L_4	L_4	F_6	F_4
	11	✓					✓	L_4	L_4	F_6	F_3
	19			✓			✓	L_4	L_4	F_1	F_1
	19					✓	✓	L_4	L_4	F_1	F_1
	27				✓		✓	L_4	L_4	F_1	F_1
BACKGROUND DISC RESIDENT WITHOUT TABLE AREA II ACCESS*††	4	✓					✓	L_3	L_2	F_2	F_4
	12	✓					✓	L_3	L_2	F_2	F_3
	20			✓			✓	L_2	L_2	F_1	F_1
	20					✓	✓	L_2	L_2	F_1	F_1
	28				✓		✓	L_2	L_2	F_1	F_1

*ADD 80 TO ANY OF THESE TYPES TO SPECIFY AUTOMATIC SCHEDULING AT SYSTEM STARTUP.

††ADD 128 TO ANY OF THESE TYPES TO SPECIFY THAT THE PROGRAM CANNOT BE DUPLICATED.

Table F-1.   Summary of RTE-IVB Program Types   (continued)

SPECIAL PROGRAMS	TYPE	DESCRIPTION
SYSTEM MODULE	0	MODULE TO BE LOADED WITH RESIDENT SYSTEM. PART OF HP-SUPPLIED SYSTEM, USER-WRITTEN DRIVER, ETC.
PROGRAM SEGMENT	5	OVERLAYABLE MODULE USED WITH DISC RESIDENT MAIN. COMMON TYPE, MEMORY-PROTECT FENCE ADDR. AND LOAD PT. DETERMINED BY MAIN.
SUBROUTINE	6	RELOCATED INTO RESIDENT LIBRARY IF CALLED BY ANY MEMORY RESIDENT PROGRAM (ALWAYS BECOME 7'S).
SUBROUTINE	7	STORED ON DISC IN RELOCATABLE FORM. ANY PROGRAM CALLING A TYPE 7 HAS A COPY APPENDED TO IT.
SUBROUTINE	8	APPENDED TO CALLING PROGRAM. ALL TYPE 8 RELOCATABLES ARE DISCARDED AFTER GENERATION.
TABLE AREA II	13	MODULE TO BE LOADED WITH RESIDENT SYSTEM IN TABLE AREA II. PART OF HP-SUPPLIED SYSTEM, USER-WRITTEN TABLES, ETC.
SUBROUTINE	14	RELOCATED INTO RESIDENT LIBRARY, WHETHER CALLED OR NOT (ALWAYS BECOME TYPE 7).
TABLE AREA I	15	MODULE TO BE LOADED WITH RESIDENT SYSTEM IN TABLE AREA I. PART OF HP-SUPPLIED SYSTEM, USER-WRITTEN TABLES, ETC.
SSGA MODULE	30	RELOCATED INTO SUBSYSTEM GLOBAL AREA OF SYSTEM. ACCESSIBLE ONLY TO PROGRAMS OF PROPER TYPE (ABOVE).

LOAD POINT & FENCE DEFINITIONS

L_1 — NEXT AVAILABLE LOCATION DURING LOAD OF RESIDENTS PLUS 2

L_2 — 35TH WORD OF NEXT PAGE AFTER COMMON AREAS

L_3 — 35TH WORD OF NEXT PAGE AFTER DRIVER PARTITION

L_4 — 35TH WORD OF NEXT PAGE AFTER TABLE AREA II

F_1 FIRST WORD OF SSGA

F_2 FIRST WORD OF PAGE FOLLOWING DRIVER PARTITION

F_3 FIRST WORD OF RT COMMON

F_4 FIRST WORD OF BG COMMON

F_5 FIRST WORD OF RESIDENT PROGRAM AREA

F_6 FIRST WORD OF PAGE FOLLOWING TABLE AREA II

With the RTE-IVB operating environment, programs can exist in 7 states. The state of a program can be changed by the operator (RTE-IVB operator command), by another program (EXEC call), or by the system to reflect an environmental change, i.e., program requested memory or disc space that was not available, requested I/O on a down device, etc. At any given time, the state of a program indicates its relationship to the RTE-IVB operating environment.

State 0 - Dormant. This state indicates that a program is not scheduled to execute. A program can be in this state if it has never been scheduled or if it has been placed in this state from a previous state by an operator command or by another program via an EXEC call.

State 1 - Scheduled. This state indicates that a program is scheduled to execute; it has been placed in the scheduled list.

State 2 - I/O Suspended. This state indicates that a program has requested I/O servicing and the system is currently performing the I/O operation or the request is queued to be processed. This condition occurs with any input operation, or with an output operation to an unbuffered device. If the class I/O technique discussed in Chapter 2 is used, or if the output device is buffered, the program will not enter this state and will be allowed to continue executing while the I/O operation is being performed.

State 3 - General Wait. A program is placed in this state if it has requested system resources that are temporarily unavailable or services that temporarily cannot be performed. For example, a program would be placed in this state if it were waiting for a data buffer to be supplied by another program, or if it requested I/O on a device that was allocated and locked to another program.

State 4 - Memory Suspended. A program is placed in this state if it has requested an operation that requires the use of System Available Memory (SAM) and an insufficient amount of SAM is available. This is a temporary state and when enough SAM becomes available, the program will be placed back into the scheduled list.

State 5 - Disc Suspended. A program is placed in this state if it has requested disc space that is unavailable. This is a temporary condition and when disc space becomes available the program will be placed back into the scheduled list.

State 6 - Operator Suspended. A program is placed into this state by an operator command or by an EXEC call from within a program. A GO operator command is necessary to remove the program from this state.

Program States

Figure G-1 shows the various states a program can exist in and the
"transition paths" between each state.

USER PROGRAM STATE DIAGRAM

States: EXECUTING 1* · SCHEDULED 1 · OPERATOR 6 · DISC 5 · MEMORY 4 · WAIT 3 · I/O 2 · DORMANT 0

TRANSITION * [EXECUTING] TO 1[SCHEDULED]

I.] ENVIRONMENT
I.1] NO LONGER HIGHEST PRIORITY PROGRAM IN SCHEDULED LIST

TRANSITION 1[SCHEDULED] TO * [EXECUTING]

I.] ENVIRONMENT
I.1] HIGHEST PRIORITY PROGRAM IN SCHEDULED LIST AND IN A MEMORY PARTITION

TRANSITION * [EXECUTING] TO 6[OPERATOR SUSPEND]

I.] OPERATOR COMMANDS
I.1] (SY)SS,PROG

II.] EXEC CALLS
II.1] EXEC(7)

TRANSITION * [EXECUTING] TO 5[DISC SUSPEND]

I.] EXEC CALLS
I.1] EXEC(4, # OF TRACKS, 1ST TRACK, DISC LU,SECTORS/TRACK)
I.2] EXEC(15, # OF TRACKS, 1ST TRACK, DISC LU,SECTORS/TRACK)

II.] SYSTEM
II.1] SWAP TRACK ALLOCATION

TRANSITION * [EXECUTING] TO 4[MEMORY SUSPEND]

I. EXEC CALLS
I.1 EXEC(2, ...) BUFFERED DEVICE
I.2 EXEC(17, ...) CLASS READ
I.3 EXEC(18, ...) CLASS WRITE
I.4 EXEC(19, ...) CLASS CONTROL
I.5 EXEC(20, ...) CLASS WRITE/READ
I.6 EXEC(ICODE,PROG,IBUFR,IBUFL) ICODE = 9,10,23, OR 24
I.7 EXEC(14,2,IBUFR,IBUFL)
I.8 CALL REIO(...)
I.9 RE-ENTRANT SUBROUTINE

(INSUFFICIENT SAM)

TRANSITION * [EXECUTING] TO 3[GENERAL WAIT]

I. EXEC CALLS
I.1 EXEC(ICODE, ...) UPPER BUFFER LIMIT EXCEEDED ICODE = 2, 17, 18, 19 OR 20
I.2 EXEC I/O REQUEST ON DOWN DEVICE
I.3 EXEC(ICODE,PROG,IBUFR,IBUFL) ICODE = 9, 23, OR 24
I.4 EXEC(ICODE, ...) NO CLASS NUMBERS ICODE = 17, 18, 19, OR 20
I.5 EXEC(21, ...)
I.6 CALL RNRQ (...) ALLOCATE RN
I.7 CALL RNRQ (...) LOCK RN
I.8 CALL LURQ (...) LOCK LU

TRANSITION * [EXECUTING] TO 2[I/O SUSPEND]

I. EXEC CALLS
I.1 EXEC(1,ICNWD,IBFR,IBFL)
I.2 EXEC(2,ICNWD,IBFR,IBFL) UNBUFFERED
I.3 EXEC(3,ICNWD) UNBUFFERED

II. ENVIRONMENT
II.1 LOADING DISC RESIDENT PROGRAM
II.2 SWAPPING IN OR OUT

TRANSITION * [EXECUTING] TO 0[DORMANT]

I.] OPERATOR COMMANDS
I.1] (SY)OF,PROG[,1]
I.2] (SY)AB,PROG[,1]

II.] EXEC CALLS
II.1] EXEC(6,PROG[,0])
II.2] EXEC(6,PROG,2)
II.3] EXEC(6,PROG,3)

III.] SYSTEM
III.1] MP PROGRAM ADDRESS
III.2] RQ PROGRAM ADDRESS
III.3] DM PROGRAM ADDRESS
III.4] TI PROGRAM ADDRESS
III.5] RE PROGRAM ADDRESS
III.6] DISC ALLOCATION ERRORS
III.7] SCHEDULING ERRORS
III.8] I/O ERRORS
III.9] RN ERRORS
III.10] LU LOCK ERRORS

TRANSITION * [SCHEDULED] TO 6[OPERATOR SUSPEND]

I.] OPERATOR COMMANDS
I.1] (SY)SS,PROG

II.] EXEC CALLS
II.1] EXEC(6,PROG,1)

TRANSITION 6[OPERATOR SUSPEND] TO 1[SCHEDULED]

I.] OPERATOR COMMANDS
I.1] (SY)GO,PROG[,P1[,P2[,P3[,P4[,P5]]]]]

II.] EXEC CALLS
II.1] EXEC(ICODE,PROG[,P1[,P2[,P3[,P4[,P5 [,ARRAY,LENGTH]]]]]])
WHERE: ICODE =10 ::= IMMEDIATE SCHEDULE WITHOUT WAIT
=24 ::= SCHEDULE IN QUEUE WITHOUT WAIT

TRANSITION 5[DISC SUSPEND] TO 1[SCHEDULED]

I.] OPERATOR COMMANDS
I.1] (SY)LG,0
I.2] (SY)LS,0
I.3] (SY)OF,PROG,8, A TEMPORARY PROGRAM
I.4] (SY)OF,PROG[,1], A SWAPPED PROGRAM OR A PROGRAM WITH 'OWN' TRACKS
I.5] (SY)RT,PROG, RELEASE PROGRAM'S TRACKS

II.] EXEC CALLS
II.1] EXEC(5,# OF TRACKS, 1ST TRACK,DISC LU)
II.2] EXEC(16,# OF TRACKS, 1ST TRACK,DISC LU)
II.3] EXEC(6)
II.4] EXEC(6,PROG,2)
II.5] EXEC(6,PROG,3)

TRANSITION 4[MEMORY SUSPEND] TO 1[SCHEDULED]

I. OPERATOR COMMANDS
I.1 (SY)IOF,PROG[,1]

II. EXEC CALLS
II.1 EXEC(5,# OF TRACKS, ...) COMPLETION
II.2 EXEC(21, ...) WITH RELEASE BUFFER
II.3 EXEC(14,1,IBUFR,IBUFL)
II.4 EXEC(6)
II.5 CALL REIO(...) COMPLETION
II.6 RE-ENTRANT SUBROUTINE EXIT

TRANSITION 3[GENERAL WAIT] TO 1[SCHEDULED]

I. OPERATOR COMMANDS
I.1 (SY)IUP,EQT
I.2 (SY)OF,PROG[,1]

II. EXEC CALLS
II.1 EXEC(6) EXCEPT SAVING RESOURCES
II.2 EXEC(21, ...) DEALLOCATE CLASS NUMBER
II.3 EXEC(ICODE, ...) WITH I/O COMPLETION ICODE = 17, 18, 19 OR 20
II.4 CALL RNRQ(...) DEALLOCATE RN
II.5 CALL RNRQ(...) UNLOCK RN
II.6 CALL LURQ(...) UNLOCK LU

III. ENVIRONMENT
III.1 LOWER LIMIT ON A BUFFERED EQT IS PASSED

TRANSITION 2[I/O SUSPEND] TO 3[GENERAL WAIT]

I. OPERATOR COMMANDS
I.1 (SY)DN,EQT
I.2 (SY)DN,,LU

II. ENVIRONMENT
II.1 DEVICE GOES DOWN

TRANSITION 2[I/O SUSPEND] TO 1[SCHEDULED]

I. ENVIRONMENT
I.1 I/O COMPLETION
I.2 LOADING COMPLETE
I.3 SWAPPING COMPLETE

TRANSITION 1[SCHEDULED] TO 0[DORMANT]

I.] OPERATOR COMMANDS
I.1] (SY)OF,PROG[,1]
I.2] (SY)AB,PROG[,1]

II.] EXEC CALLS
II.1] EXEC(6,PROG,2)
II.2] EXEC(6,PROG,3)

TRANSITION 0[DORMANT] TO 1[SCHEDULED]

I.] OPERATOR COMMANDS
I.1] (SY)RU,PROG[,P1[,P2[,P3[,P4 [,P5[,STRING]]]]]]
I.2] (SY)ON,PROG[,NOW] [,P1[,P2[,P3 [,P4[,P5[,STRING]]]]]]

II.] EXEC CALLS
II.1] EXEC(ICODE,PROG[,P1[,P2[,P3[,P4[,P5 [,ARRAY,LENGTH]]]]]])
II.2] WHERE: ICODE = 9 ::= IMMEDIATE SCHEDULE WITH WAIT
=10 ::= IMMEDIATE SCHEDULE WITHOUT WAIT
=23 ::= SCHEDULE IN QUEUE WITH WAIT
=24 ::= SCHEDULE IN QUEUE WITHOUT WAIT

III.] SYSTEM
III.1] TIME OF DAY SCHEDULE (TIME LIST)
III.2] INTERRUPT SCHEDULE (INTERRUPT TABLE, INTAB)

TRANSITION 6[OPERATOR SUSPEND] TO 0[DORMANT]

I.] OPERATOR COMMANDS
I.1] (SY)OF,PROG[,1]

II.] EXEC CALLS
II.1] EXEC(6,PROG,2)
II.2] EXEC(6,PROG,3)

TRANSITION 5[DISC SUSPEND] TO 0[DORMANT]

I.] OPERATOR COMMANDS
I.1] (SY)OF,PROG[,1]

II.] EXEC CALLS
II.1] EXEC(6,PROG,2)
II.2] EXEC(6,PROG,3)

TRANSITION 4[MEMORY SUSPEND] TO 0[DORMANT]

I.] OPERATOR COMMANDS
I.1] (SY)OF,PROG[,1]

II.] EXEC CALLS
II.1] EXEC(6,PROG,2)
II.2] EXEC(6,PROG,3)

TRANSITION 3[GENERAL WAIT SUSPEND] TO 0[DORMANT]

I.] OPERATOR COMMANDS
I.1] (SY)OF,PROG[,1]

II.] EXEC CALLS
II.1] EXEC(6,PROG,2)
II.2] EXEC(6,PROG,3)

TRANSITION 2[I/O SUSPEND] TO 0[DORMANT]

I.] OPERATOR COMMANDS
I.1] (SY)OF,PROG,1

II.] EXEC CALLS
II.1] EXEC(6,PROG,3)

Appendix H
DCB and Directory Formats

This Appendix contains information on the following:

*   DATA CONTROL BLOCK (DCB) FORMAT

*   CARTRIDGE DIRECTORY FORMAT

*   FILE DIRECTORY FORMAT

Data Control Block Format

WORD	BIT 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0	
0	SECTOR OFFSET / SECTOR # OF FILE DIRECTORY / LU # OF FILE DIRECTORY OR OF FILE IF ON DISC	FILE DIRECTORY ADDRESS
1	TRACK # OF FILE DIRECTORY	
2	FILE TYPE (MAY BE OVERRIDDEN AT OPEN, UNLESS TYPE 0)	
3	TRACK ADDRESS OF FILE (TYPE > = 1) / LU # OF FILE (TYPE = 0)	
4	SECTOR ADDRESS OF FILE (TYPE > = 1) / END-OF-FILE CODE (TYPE = 0)	
5	FILE SIZE IN –CHUNKS +SECTORS (TYPE > = 1) / SPACING CODE (TYPE = 0)	
6	RECORD LENGTH (TYPE = 2) / READ/WRITE CODE (TYPE = 0)	
7	SC / NUMBER OF BLOCKS IN DCB BUFFER / SY OM IB EF WR	
8	NUMBER OF SECTORS PER TRACK (TYPE > = 1)	
9	OPEN/CLOSE INDICATOR	
10	TRACK # OF CURRENT FILE POSITION (TYPE > = 1)	CURRENT POSITION IN FILE
11	SECTOR # OF CURRENT FILE POSITION (TYPE > = 1)	
12	LOCATION OF NEXT WORD IN FILE (TYPE > = 1)	
13	RECORD # OF CURRENT FILE	
14	POSITION (DOUBLE WORD INTEGER)	
15	EXTENT NUMBER (TYPE > = 3)	
16	DCB BUFFER AREA	

16-WORD CARTRIDGE ENTRY (words 0–15)

BUFFER 128+n

Legend for Data Control Block

Word		Content

0 File Directory Address:
                          bits 6-12 = physical sector # (block) of the directory.

                          bits 13-15 = entry offset from the beginning of the block (org 0).

4 End-of-File Code, type 0 file: 01 lu = EOF on Magnetic Tape
                          10 lu = EOF on Paper Tape
                          11 lu = EOF on Line Printer

5 Spacing Code, type 0 file:
                          bit 15 = 1 - backspace legal
                          bit 0 = 1 - forward space legal

6 Read/Write Code, type 0 file:
                          bit 15 = 1 - input legal
                          bit 0 = 1 - output legal

7 Security Code Check/Open Mode/Buffer Size/In Buffer/To Be Written/ EOF Read Flag, all file types

    (SC) Security Code Check
                          bit 15 = 1 - security codes agree
                                  = 0 - security codes do not agree

    DCB Buffer:
                          bits 14-7 = Number of blocks in DCB buffer

    (SY) System Disc:
                          bit 4 = 1 file is on a system disc
                             = 0 not on a system disc

    (OM) Open Mode:
                          bit 3 = 1 - update open
                              0 - standard open

    (IB) In Buffer Flag:
                          bit 2 = 1 - data in DCB buffer
                             = 0 - data not in DCB buffer

    (EF) EOF Read Flag:
                          bit 0 = 1 - EOF has been read
                           = 0 - EOF has not been read

    (WR) To Be Written:
                          bit 0 = 1 - data in DCB buffer to be written
                          = 0 - data in DCB buffer not to be written

9 Open/Close Indicator: if open, contains ID segment location of program performing open. If closed, set to zero.

Cartridge Directory Format

The cartridge directory is located in the system area on LU 2. Its length is two blocks.

```
        5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0
      ┌──────────────────┬──────────────────┐
   0  │       LOCK       │        LU        │
      ├──────────────────┴──────────────────┤
   1  │            LAST TRACK                │
      ├─────────────────────────────────────┤
   2  │       CARTRIDGE REFERENCE LABEL      │
      ├──────────┬──────────────────────────┤
   3  │          │            ID             │
      ├──────────┴──────────────────────────┤
      │                                     │
      │     UP TO 32 4-WORD ENTRIES IN      │
      │      THE FIRST BLOCK OF THE CL.     │
      │     UP TO 31 4-WORD ENTRIES IN      │
      │      THE SECOND BLOCK.              │
      ~                                     ~
```

LOCK = 0 IF NOT LOCKED; ELSE IS KEYWORD TABLE OFFSET OF ID SEGMENT ADDRESS OF LOCKING PROGRAM

LOCKED DISCS ARE AVAILABLE ONLY TO THE LOCKER.

ID IDENTIFIES TO WHOM THE CARTRIDGE IS MOUNTED.

ID = 0000 → NON-SESSION
ID = 7777 → SYSTEM CARTRIDGE
0<ID<7777 → SESSION MONITOR GROUP OR PRIVATE CARTRIDGE

NOTE: WORDS 124, 125, 126, AND 127 ARE UNIQUE ONLY IN THE SECOND BLOCK OF THE CL. THE FIRST BLOCK WILL HOLD 32 ENTRIES IN WORDS 0 THROUGH 127.

```
      ~                                     ~
      ├─────────────────────────────────────┤
 252  │                 0                   │
      ├─────────────────────────────────────┤
 253  │      INITIALIZATION CODE WORD       │
      ├─────────────────────────────────────┤
 254  │        MASTER SECURITY CODE         │
      ├─────────────────────────────────────┤
 255  │       RESERVED FOR FUTURE USE       │
      └─────────────────────────────────────┘
```

SUM OF CONTENTS OF BASE PAGE WORDS ← 1650 THRU 1657 AND 1742 THRU 1747 AND 1755 THRU 1764.

← SET WHEN SYSTEM CARTRIDGE IS INITIALIZED.

File Directory

The first entry in each File Directory is the specification entry for the cartridge itself. The directory starts on the last FMP track of each cartridge in sector zero on all discs. The directory blocks are written using sector skipping. The directory sector address can be obtained from the block address by the following formula:

    sector address = (block *14) modulo S/T |

where S/T is the number of sectors per track. Directory blocks are 128 words long. Each Directory entry is 16 words long.

```
     Word              Content                     IN Parameters

     15                              0      bit 15 set to distin-
    -------------------------------\        guish cartridge entry
 0 |  |                          | \<---from file entry
   |---------------------------|  \
   |                           |   \
 1 |                           |    \
   |---------------------------|     6 character cartridge
   |                           |   /        label.
 2 |                           |  /
   |---------------------------| /
 3 | cartridge reference number |
   |---------------------------|
 4 | first available track for FMP |
   |---------------------------|  CPU number in bits 15-13.
 5 | CPU| F |  next available sector |  Bit 12 is a flag indicating
   |---------------------------|  whether the CPU word is
   |                           |  being used.
 6 | number of sectors per track |
   |---------------------------|
   | lowest directory track (last file |
 7 | track + 1)                |
   |---------------------------|
   |                           |
 8 | number of tracks in directory (neg- |
   | ative value)              |
   |---------------------------|
 9 | next available FMP track  |
   |---------------------------|
10| first bad track           |
   |---------------------------|
   |                           |
   |                           |
   |              .            |
   |              .            |
   |              .            |
   |              .            |
   |                           |
   |---------------------------|
15| sixth bad track           |
   |---------------------------|
```

Disc File Directory

```
BIT      15 |           | 8 | 7 |         | 0        CR PARAMETERS:
```

WORD	
0	
1	6-CHARACTER FILE NAME	} FILE NAME
2	
3	FILE TYPE (1 THRU 32767)
4	STARTING TRACK
5	EXTENT # STARTING SECTOR
6	FILE SIZE IN + SECTOR OR - CHUNKS
7	RECORD LENGTH (TYPE 2 ONLY)
8	SECURITY CODE
9	
10	
11	
12	
13	
14	
15	

OPEN FLAGS
15 = 1 FOR EXCLUSIVE OPEN
14-12 = RESERVED
11-8 = SEQUENCE COUNTER
7-0 = KEYWORD OFFSET OF
        OPENING PROGRAM'S
        ID SEGMENT

WORD 0 = 0 IF THE LAST ENTRY IN DIRECTORY; = –1 IF FILE IS PURGED

Type 0 File Directory Entry

The entries for non-disc (type 0) files differ from those for disc
files in words 3 through 7:

```
        bit    15|                         |  0        CR parameters:
               _____
              |                           |
word    3     |     0 (file type default) |
              |---------------------------|
        4     |     logical unit number   |
              |---------------------------|
        5     |   end of file subfunction |<---EO,LE,PA or control
              |---------------------------|
        6     |        spacing code       |<----------BS,FS, or BO
              |---------------------------|
        7     |      input-output code    |<----------RE,WR, or BO
               _____
```

Words 5-7 are octal codes:

    end-of-file subfunction = 01LU for MT(EO)
                              10LU for paper tape (LE)
                              11LU for line printer (PA)
                              or subfunction code

    spacing code = bit 15 = 1 backspace legal (BS)
                     bit 0 = 1 forward space legal (FS)

    input/output code = bit 15 = 1 input legal (RE)
                         bit 0 = 1 output legal (WR)

Appendix I
Memory Management and Related Tables

This appendix contains information on the following:

* ADDRESS TRANSLATION

* LOGICAL MEMORY AND BASE PAGE

* MEMORY ALLOCATION TABLE (MAT)

Address Translation

There are four memory maps used by the DMS. Each map consists of 32, 12-bit hardware registers. The contents of the first 10 bits (bit 0 - bit 9) of each map register points to a 1024 word section (one page) of physical memory. DMS breaks the basic 15-bit address into a 5-bit page index (bit 10 - bit 14) and a 10-bit page off-set (bit 0 - bit 9). The page index points to one of the 32 map registers in the currently enabled map (only one map is enabled at any given instant). The contents of this map register (pointer to page in physical memory) is then concatenated with the 10-bit page off-set obtained from the original address (pointer to word within page) to form the final 20-bit address necessary to access 1024K word of physical memory (see Figure I-1).

BASIC ADDRESS

| 14 ——— 10 | 9 ——————— 0 |

OFFSET
WITHIN
PAGE
(1-1024)

POINTER
TO MAP
REGISTER
(1-32)

20-BIT ADDRESS BUS

| 9 ——— 0 |

POINTER
TO PHYSICAL
PAGE
(1-1024)

CURRENTLY ENABLED
MEMORY MAP

PAGE
BOUNDRY
(TYPICAL)
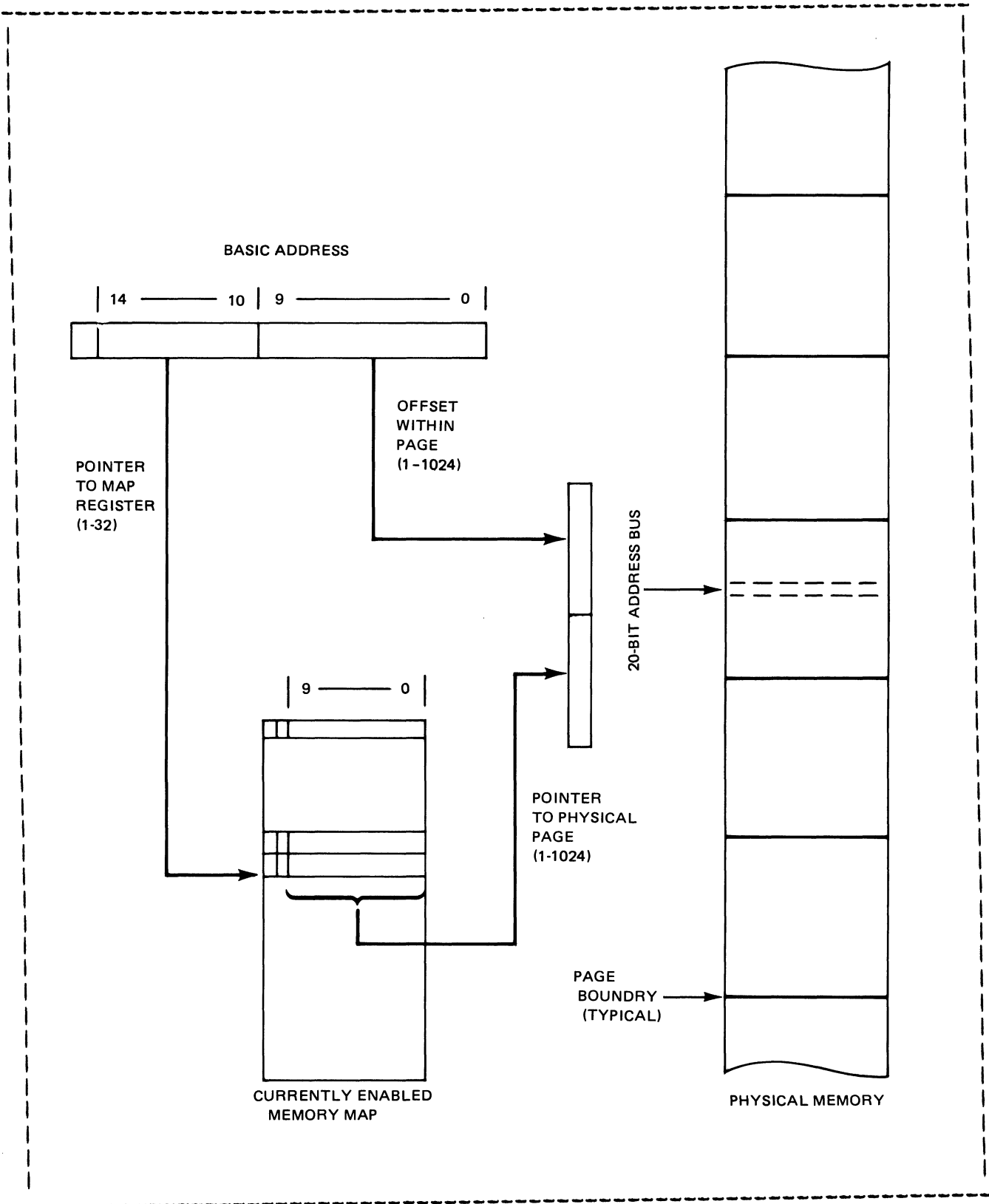
PHYSICAL MEMORY

Figure I-1. Address Translation via Memory Mapping

Logical Memory and Base Page

Logical memory is the 32K-word address space described by the currently enabled memory map. Figure I-2 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 configured differently.

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 program 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 I-2, configuration 3.

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, tables 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 System Communications area (see Appendix B), driver links, SSGA and table area links located in physical page 0 are common to all base pages. Base page structures are illustrated in Figure I-3.

The Base Page Fence (refer to the HP/1000 M-, E-, F-Series Computer 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 I-2. RTE-IVB 32K Word Logical Memory Configurations
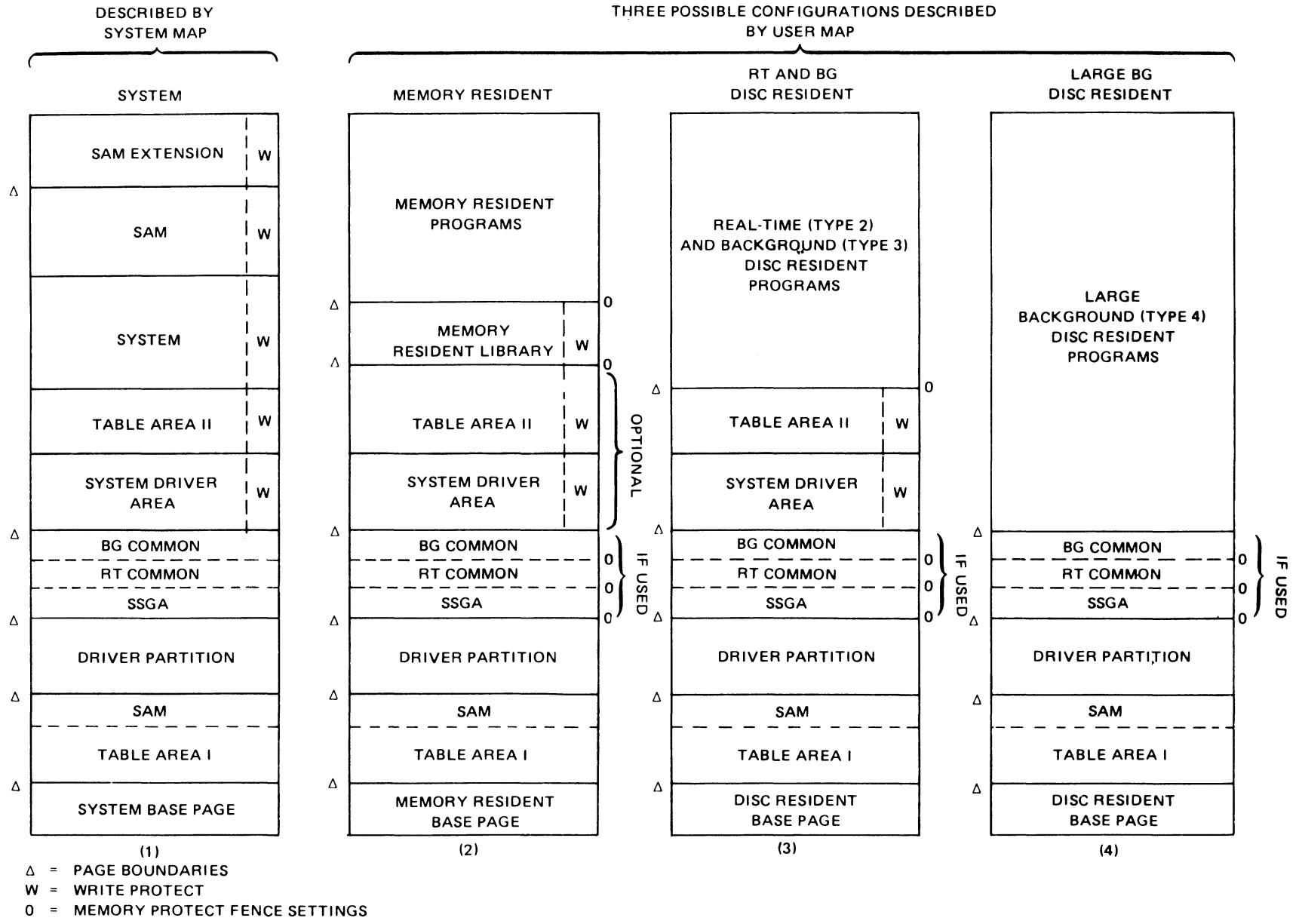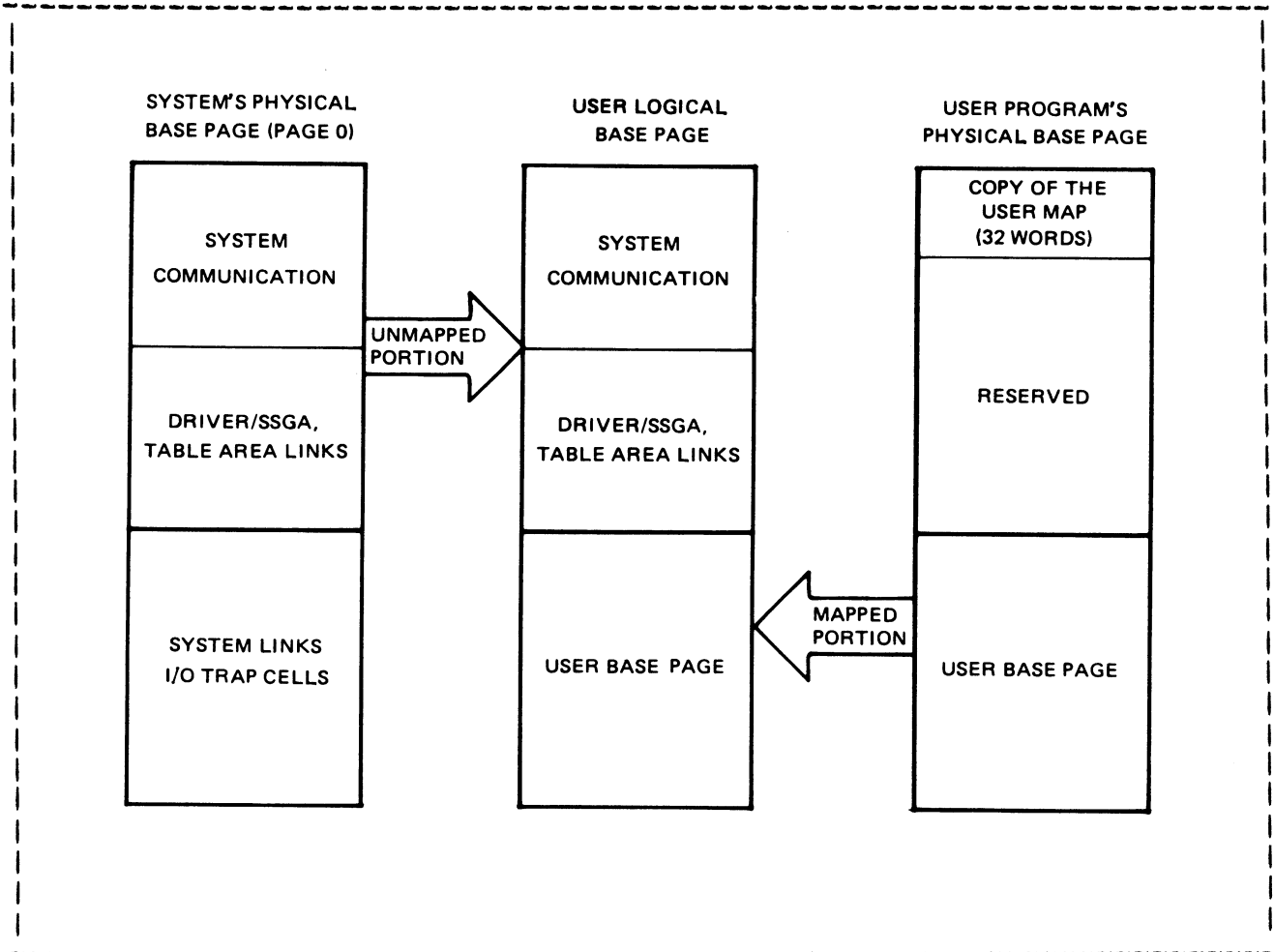
Figure I-3. Base Page Structure

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 I-4.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	WORD
MAT LINK WORD																0
PARTITION OCCUPANT'S PRIORITY																1
ID SEGMENT ADDRESS OF OCCUPANT																2
M		D	/////			PHYSICAL START PAGE OF PARTITION										3
R	C	/////				NUMBER PAGES IN PARTITION (EXCLUDE BASE PAGE)										4
RT	/////													S		5
SUBPARTITION LINK WORD																6

Figure I-4. Memory Allocation Table Entry Format

WHERE:

MAT LINK WORD = -1 IF PARTITION NOT DEFINED EITHER DURING SYSTEM
GENERATION MEMORY RECONFIGURATION, OR BY
PARITY ERROR

= 0 IF END OF LIST

= POSITIVE = NEXT PARTITION ADDRESS IN 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
MOTHER PARTITION

RT = 1 IF MAT ENTRY IS FOR REAL-TIME PARTITION

= 0 IF MAT ENTRY IS FOR BACKGROUND 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 SWAP-OU T COMPLETED.
MOTHER CLEARED.

SUBPARTITION LINK WORD

= 0 IF MAT ENTRY IS NOT A SUBPARTITION

= NEXT SUBPARTITION ADDRESS IF THIS IS A
SUBPARTITION

= MOTHER PARTITION MAT ADDRESS IF THIS ENTRY
IS THE LAST SUBPARTITION.

Figure I-4.   Memory Allocation Table Entry Format

Appendix J
Session Monitor Tables

This appendix contains information on the following:

* SESSION CONTROL BLOCK (SCB)

* SESSION SWITCH TABLE (SST) AND CONFIGURATION TABLE

* SESSION TABLE RELATIONSHIP

Session Control Block (SCB)

A Session Control Block (SCB) is established for each user who has successfully "logged-on" to the system. The SCB contains the information necessary to identify the user to the system and describe his capabilities in terms of command processing and I/O addressing space.

The format of the SCB is shown in Figure J-1.
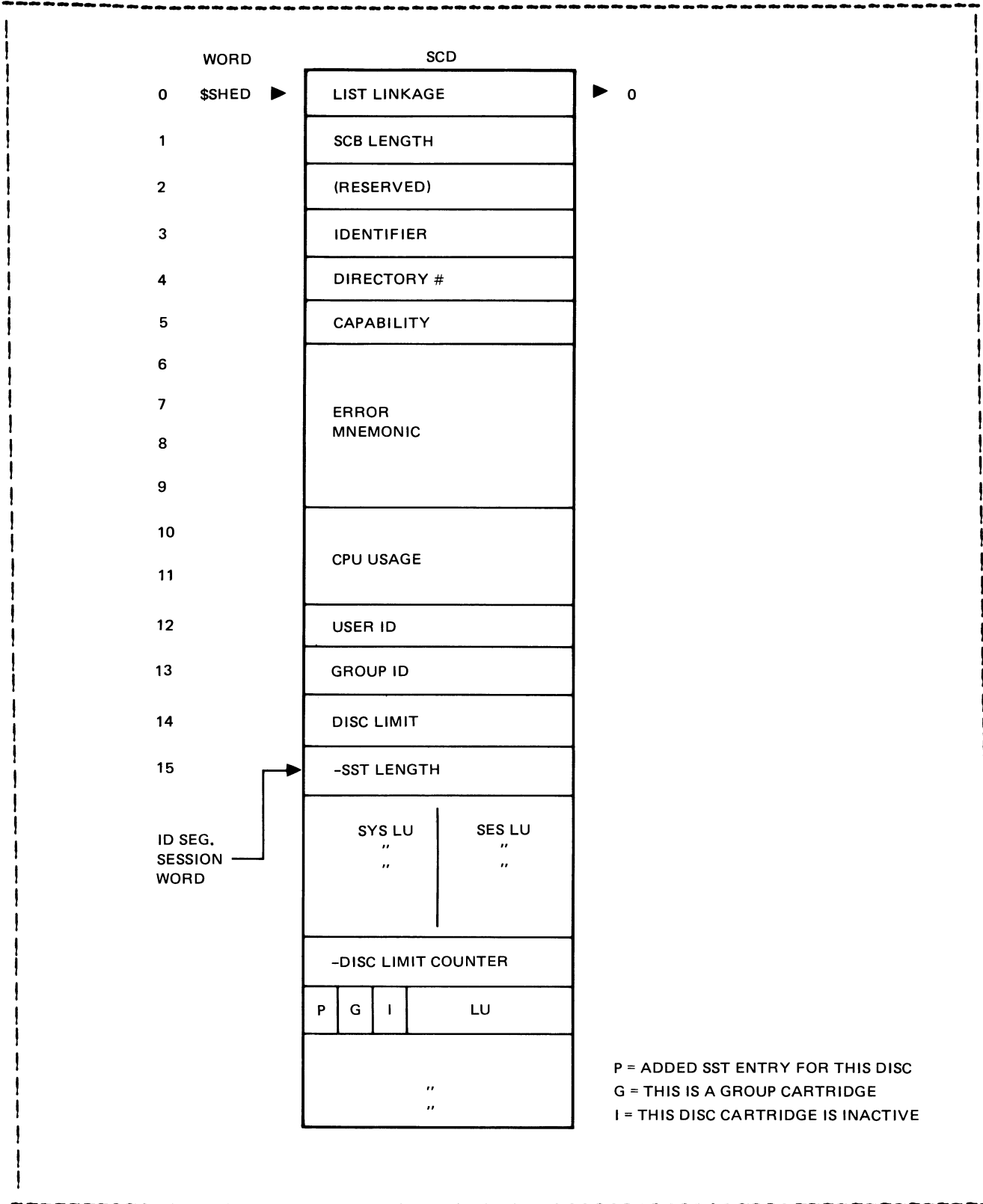
Figure J-1. SCB

Session Switch Table (SST) and Configuration Table

When operating in the session environment every I/O request is routed to the appropriate I/O device via the Session Switch Table (SST). Each SST entry describes a session LU, which the user addresses, and associated system LU where the I/O request will actually be directed. The SST describes the session user's I/O addressing capabilities by defining the system LUs the user has access to and the associated session LUs by which the user accesses them.

When the user makes an I/O request the SST is searched for the specified session LU. If the requested LU is found, it is switched to the associated system LU as specified in the SST entry and the I/O request is processed. Zero-offset addressing makes the internal representation of the LU in the Session Control Block (SCB) one less than the assigned session LU number. If the requested LU is not found, an error is returned (IO12-LU not defined for this session).

The Session Switch Table is maintained in memory as part of the SCB. The format of the SST is shown in Figure J-2 below:

System LUs can be integer numbers between 1 and 255. Session LUs can be integer numbers between 1 and 63. Session LUs are assigned:

*   at log-on, via user and group account file entries, or

*   on-line using SL command (see RTE-IVB Terminal User's Reference Manual), or

*   at log-on, via Configuration Table entries.

The Configuration Table describes the default logical units to be used for specific device logical units. Each station (terminal) logical unit defined in the Configuration Table has associated with it a set of device logical units which are assigned default logical units to be used when a user logs on at this station (terminal). The default logical unit associated with the station itself is always 1.

At log-on, these default values are written from the Configuration Table in the account file into the user's Session Control Block (SCB), unless overridden by entries in this particular user's SST. The format of the Configuration Table is shown in Figure J-3, below.

Session Table Relationships

The interaction among the session tables is shown in Figure J-4.

Figure J-2. Session Switch Table (SST) Format

LENGTH	
STATION LU	1
SYSTEM LU	DEFAULT LU
SYSTEM LU	DEFAULT LU
LENGTH	
STATION LU	1
SYSTEM LU	DEFAULT LU
SYSTEM LU	DEFAULT LU
SYSTEM LU	DEFAULT LU
⋮	
0	

Figure J-3. Configuration Table
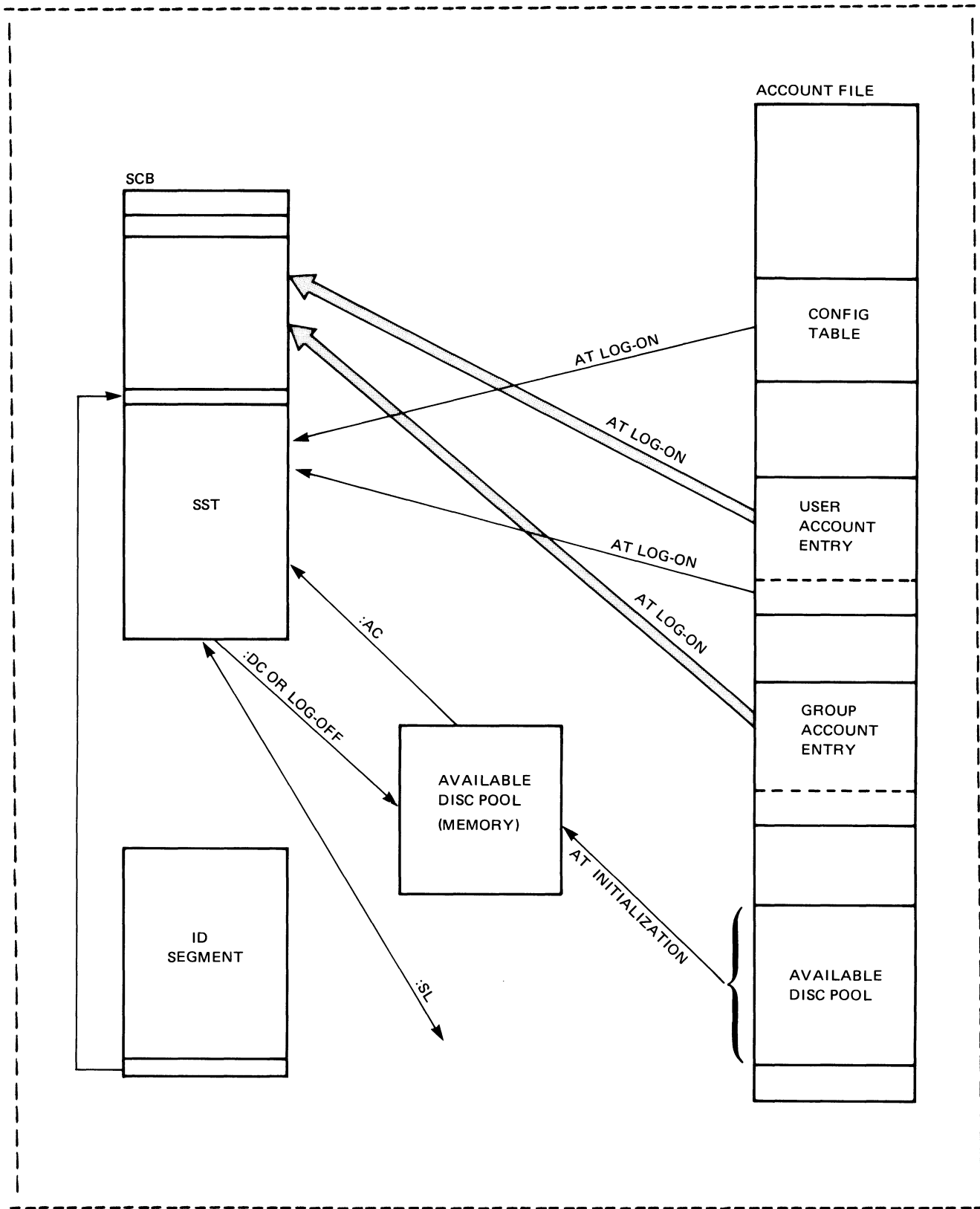
Figure J-4.

Appendix K
Class I/O Application 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 K-1.

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:
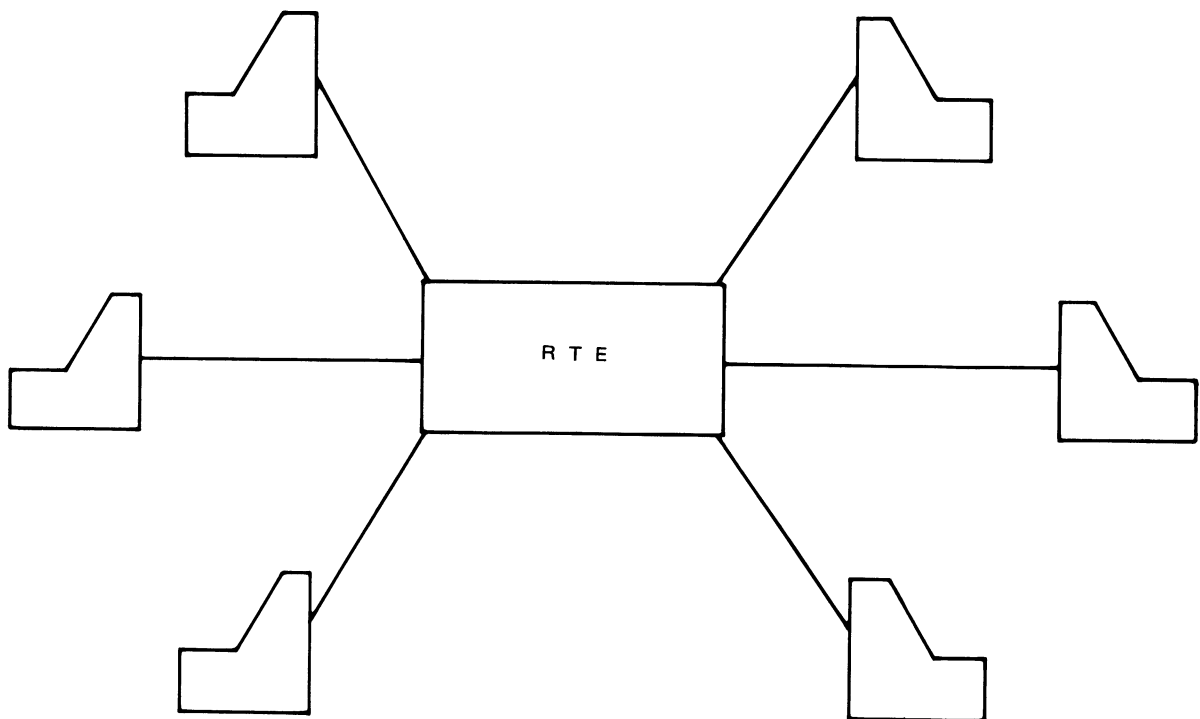
Assume an order-entry situation in which there are several operators but only one program. If standard I/O was used, it would be possible to read from only one terminal at a time. However, by using Class I/O, the program permits all operators to enter data seemingly at once. RTE handles all queueing 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. The flowchart for such an application is illustrated in Figure K-1. Note that although operators and terminal devices are shown, the input could be received from any one of a series of indentical devices.
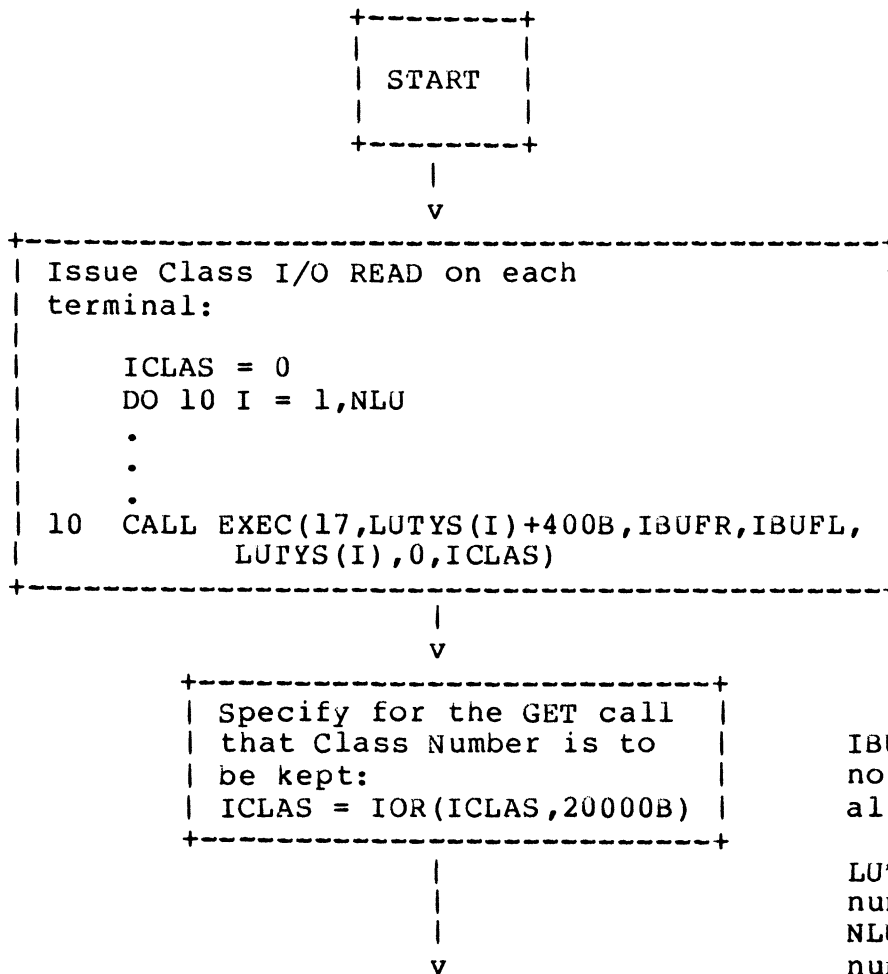
```
                        +--------+
                        |        |
                        | START  |
                        |        |
                        +--------+
                             |
                             v
+-----------------------------------------------------+
| Issue Class I/O READ on each                        |
| terminal:                                           |
|                                                     |
|       ICLAS = 0                                     |
|       DO 10 I = 1,NLU                               |
|          .                                          |
|          .                                          |
|          .                                          |
| 10    CALL EXEC(17,LUTYS(I)+400B,IBUFR,IBUFL,       |
|          LUTYS(I),0,ICLAS)                          |
+-----------------------------------------------------+
                             |
                             v
            +-----------------------------+         Notes:
            | Specify for the GET call    |
            | that Class Number is to     |         IBUFL contains negative
            | be kept:                    |         no. of characters
            | ICLAS = IOR(ICLAS,20000B)   |         allowed for input.
            +-----------------------------+
                             |
                             |                       LUTYS is an array of LU
                             |                       numbers.
                             |                       NLU is the total
                             v                       number of terminals.
```
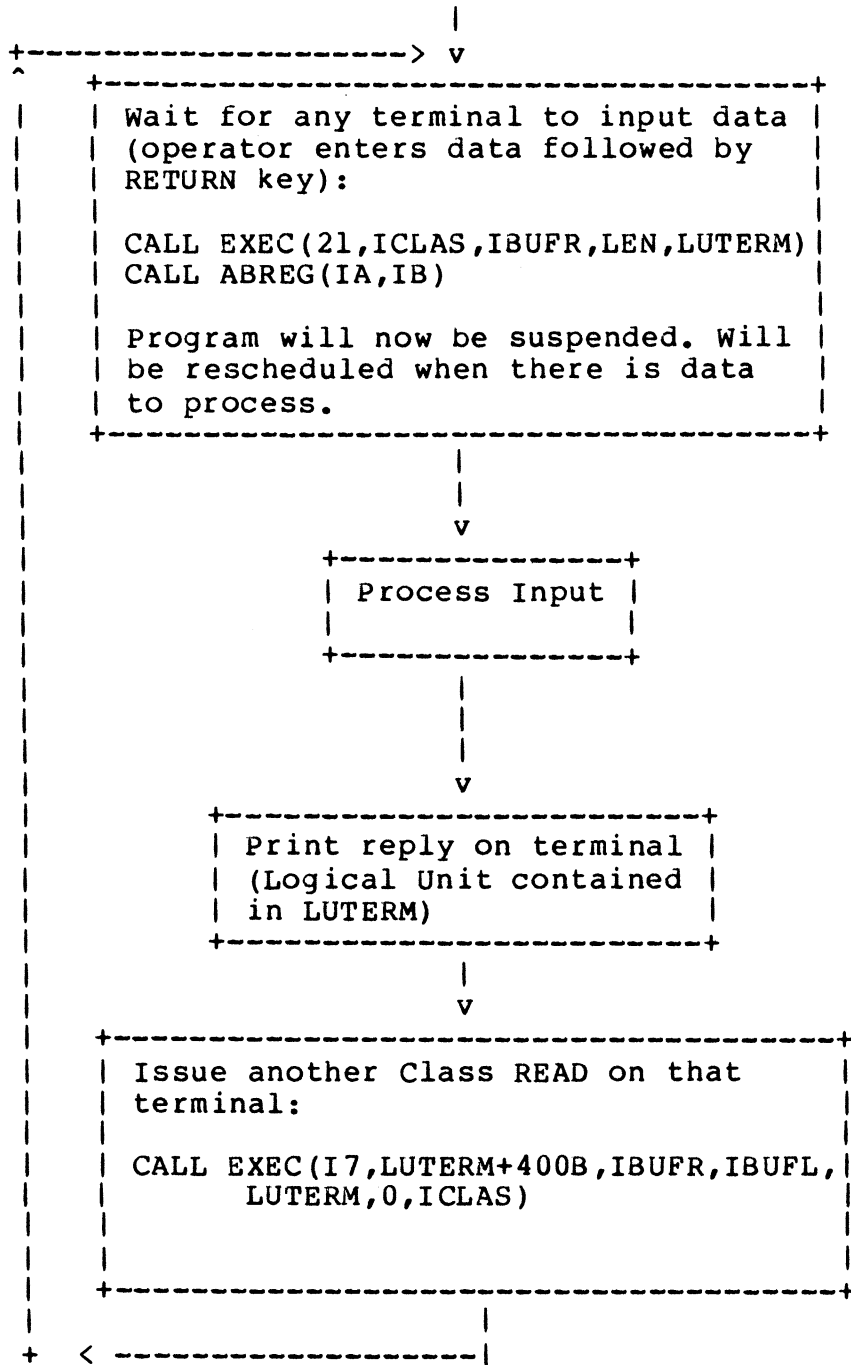
```
                        |                              Notes:
 +----------------------> v
 ^       +-----------------------------------------+
 |       | Wait for any terminal to input data |
 |       | (operator enters data followed by   |    LEN contains maximum
 |       | RETURN key):                        |    length of IBUFR.
 |       |                                     |
 |       | CALL EXEC(21,ICLAS,IBUFR,LEN,LUTERM)|    LUTERM will be the LU
 |       | CALL ABREG(IA,IB)                   |    that responded (obtained
 |       |                                     |    from EXEC 17).
 |       | Program will now be suspended. Will |
 |       | be rescheduled when there is data   |    On return, IA = status
 |       | to process.                         |    (e.g., bit 7 or bit 5
 |       +-----------------------------------------+    will be set for EOF or
 |                        |                         EOT respectively).
 |                        |                         IB = no. of characters
 |                        v                         input (will be positive).
 |            +-----------------+
 |            | Process Input |
 |            |                 |
 |            +-----------------+
 |                        |
 |                        |
 |                        |
 |                        v
 |            +---------------------------+
 |            | Print reply on terminal |
 |            | (Logical Unit contained |
 |            | in LUTERM)              |
 |            +---------------------------+
 |                        |
 |                        v
 |       +---------------------------------------+
 |       | Issue another Class READ on that    |
 |       | terminal:                           |
 |       |                                     |
 |       | CALL EXEC(I7,LUTERM+400B,IBUFR,IBUFL,|
 |       |      LUTERM,0,ICLAS)                 |
 |       |                                     |
 |       |                                     |
 |       +---------------------------------------+
 |                        |
 +    < --------------------|
```

Figure K-1. 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.

EXAMPLE 2. MAILBOX COMMUNCIATION 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 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 K-1 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 K-1 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 K-2 provides a revised version of the sample program given in Figure K-1. 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. The system does NOT release a Class Number when the allocating program terminates.

The "Process Input" box previously illustrated in Figure K-1 can then
be expanded as illustrated in Figure K-2 below:

```
                              |
                              v
              +---------------------+
              | Determine input     |
              | command legality    |
              | and type            |
              +---------------------+
                              |
                              v
          +-------------------------------+
          | Determine Class Number from   |
          | input command type (JCLAS)    |
          +-------------------------------+
                              |
                              v
      +-----------------------------------------------+
      | Send input buffer to processing program       |
      | for that command, with terminal's LU          |
      | and Class Number for use by the               |
      | processing program:                           |
      |                                               |
      | CALL EXEC(20,0,IBUFR,IBUFL,LUTERM,ICLAS,       |
      |      JCLAS)                                    |
      +-----------------------------------------------+
                              |
                              v
                      +-----------+
                      | continue  |
                      +-----------+
```
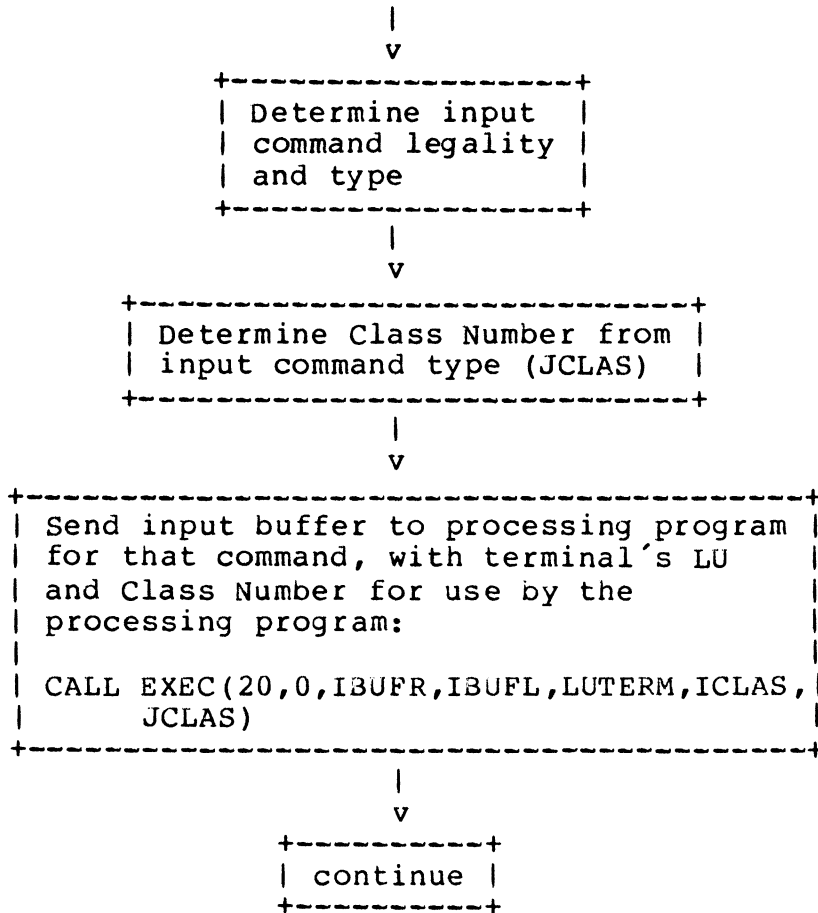
Figure K-2. Dispatching Input to Subtasks for Processing

Since no devices are involved in mailbox I/O, the ICNWD (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 K-1),
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, suspending themselves until other transactions are available for the programs to process.

Appendix L
Resource Numbers and Logical Unit Lock Examples

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.

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. An RN may be saved anywhere that the cooperating programs can find it (SSGA and COMMON are typical) and the cooperating programs must lock the RN locally before accessing the associated resource 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 neccessary 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).

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 File Management 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" 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 L-1A through L-1C 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 L-1C shows a printout by the WHZAT program of this lock-up condition.

L-1A. "DEADLY EMBRACE" Example

PAGE 0001 FTN.   8:07 AM WED., 13 JUN., 1979

```
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. 1926   (780113)

** NO WARNINGS ** NO ERRORS **   PROGRAM = 0081   COMMON = 00002

L-1B. "DEADLY EMBRACE" Example

PAGE 0001 FTN.   8:07 AM WED., 13 JUN., 1979

```
0001   FTN4,L
0002        PROGRAM LOCKB(3,90)
0003        COMMON LOCK1,LOCK2
0004        ICODE=11B
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. 1926   (780113)

** NO WARNINGS ** NO ERRORS **   PROGRAM = 0081   COMMON = 00002

L-1C. "DEADLY EMBRACE" WHZAT Printout

```
17:21:51:520
--------------------------------------------------------------------
  PRGRM T PRIOR PT SZ DO.SC.IO.WT.ME.DS.OP.      .PRG CNTR.  .NEXT TIME.
--------------------------------------------------------------------
**FMG24 3 00090  7 10 * * * *  3,LOCKA  * * * * * P:50352SWP
  LOCKA 3 00090 14  4 . . . .  3,RN  002,LKPRG=LOCKB
                                                   P:42075SWP
** BLOCK **
  LOCKB 3 00090 13  4 . . . .  3,RN  003,LKPRG=LOCKA
                                                   P:42075SWP
** BLOCK **
********** DEAD LOCK **
*** SEE ABOVE FOR REPORT ON FMG24

  WHZAT 3 00001  7  4 . 1, . . . . . . . . . . . P:43453
--------------------------------------------------------------------
ALL LU'S OK
ALL EQT'S OK
LOCKED LU'S (PROG NAME)  28(EDT28),
MAX CONT. FREE TRKS :    26,  LU   2
--------------------------------------------------------------------
17:21:51:630
```

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.

Appendix M
Scheduling FMGR Programmatically

To request FMGR from a program (optionally, a command string may be passed to FMGR), use the following call:

```
+-------------------------------------------------------------------+
| CALL EXEC(ICODE,6HFMGR   ,INP,LIST,ISV,LOG,IDUM,IBUFR,IBUFL)      |
|                           --- ---- --- --- ---- ----- -----       |
| CALL EXEC(ICODE,6HFMGR   ,2HXX,LIST,ISV,LOG,IDUM,IBUFR,IBUFL)     |
|                           ---- ---- --- --- ---- ----- -----      |
|-------------------------------------------------------------------|
| ICODE = 23 to schedule FMGR with wait                            |
|       = 24 to schedule FMGR without wait                         |
|                                                                   |
| INP   = input;  if 2 ASCII characters,  input is entered from the |
|         file whose name appears  in IBUFR;  otherwise  INP is the |
|         logical unit number  of the input device  where  the FMGR |
|         commands will be entered.  If omitted, LU1 is assumed; in |
|         a multi-terminal  environment,  default input  is logical |
|         unit number  of device  where FMGR was scheduled.   IBUFR |
|         must  contain  a  FMGR command  which is  executed before |
|         control is passed to that LU (for example, :TR,TEST where |
|         TEST holds a number of commands.                         |
|                                                                   |
| LIST,ISV,LOG=corresponds to the parameters in the RU,FMGR command.|
|                                                                   |
| IDUM  = dummy placeholder for parameter 5.                       |
|                                                                   |
| IBUFR,IBUFL = optional  buffer  containing  command string  to be |
|         passed  to  FMGR  or  the name  of a file  in which  FMGR |
|         commands are stored (file HELLOO in example shown below). |
|                                                                   |
| example: DATA IBUFR/2H,,,2HHE,2HLL,2HOO/                         |
|          DATA LIST/1/,ISV/0/,LOG/1/,IBUFL/4/                     |
|          CALL EXEC(23,6HFMGR   ,2HXX,LIST,ISV,LOG,IDUM,IBUFR,IBUFL)|
+-------------------------------------------------------------------+
```

The command string to be passed to FMGR via IBUFR must begin with a colon (:). The buffer length specified in IBUFL is a positive value for words, or a negative value for characters. The command string is ignored if:

a. The input device specified is not an interactive device.

b. The command string does not start with a colon.

NOTE: For information pertaining to interactively scheduling FMGR, refer to the RTE-IVB Terminal User's Reference Manual (92068-90002).

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).

ACCOUNT FILE - A disc resident file created and maintained by the System Manager. It contains information on all authorized session users and other session related information.

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-IVB 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.

BIT BUCKET - Logical unit number pointing to Equipment Table entry number zero, which in turn, does not point to any existing device. I/O directed to the bit bucket is lost.

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).

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 initializatrion process.

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

CAPABILITY LEVEL - An integer from 1 to 63 which defines the FMGR, System, and Break-Mode commands which a session user may execute.

CARTRIDGE - A set of contiguous cylinders on a disc unit. Cartridges contain disc files with a directory of the files stored on each cartridge. All files on the same FMP cartridge must have unique names. The system disc on logical unit 2 contains the RTE operating system, and may contain FMP files.

CHAINING - A technique for coordinating sequential execution of independent programs in the same portion of main memory.

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.

CONFIGURATION TABLE - A set of default logical units associated with the station that a session user logs on at.

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.

CYLINDER - The area that passes under all heads during one revolution of the disc surfaces.

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.

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.

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 a 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.

LOG-OFF - The process by which a session is terminated.

LOG-ON - The process by which a session is initiated. The Log-On process involves checking the session user's identification to allow access to the system, and the creation of the user's operating environment through the User HELLO File and Session Control Block.

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 accesory 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.

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, acnieved 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-IVB 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-IVB 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 subchannnel 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.

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.

SCB - See SESSION CONTROL BLOCK.

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.

SESSION CONTROL BLOCK (SCB) - A variable-length table built in physical memory by the log-on processor for each session.

SESSION IDENTIFIER - A number by which the system identifies each session. Typically it is the system logical unit number of the terminal on which the session user has logged on.

SESSION SWITCH TABLE (SST) - A table which defines a session's total I/O addressing range. It provides a mapping between Session Logical Unit numbers which the user addresses and System Logical Unit numbers which is where the system processes the call.

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.

SPARE CARTRIDGE POOL — A set of cartridges defined by the System Manager as being available to session users for temporary disc space.

SST — See SESSION SWITCH TABLE.

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.

STATION — A terminal and its associated peripheral devices.

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.

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-IVB 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 exectuion.

SYSTEM TRACKS - All subchannel tracks assigned to RTE-IVB 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.

TIMESLICING - A means by which compute bound programs can be prevented from monopolizing CPU time. A timesliced program is placed in a round-robin queue with other programs of the same priority. Each program in the queue gets a quantum of CPU time to execute before another program gets its turn. Higher priority programs can interrupt any timesliced program at any time.

UP - See DEVICE UP.

USER HELLO FILE - A procedure file that control is automatically transferred to when the session user first logs on to the system.

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

Index

Index

Index

Index

READER COMMENT SHEET

**RTE-IVB Programmer's
Reference Manual**

92068-90004   **October 1981**

Update No. _____

(If Applicable)

We welcome your evaluation of this manual. Your comments and suggestions help us improve our publications. Please use additional pages if necessary.

FROM:

Name _____

Company _____

Address _____

_____

_____

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 141 CUPERTINO, CA.

— POSTAGE WILL BE PAID BY —

Hewlett-Packard Company
Data Systems Division
11000 Wolfe Road
Cupertino, California 95014
ATTN: Technical Marketing Dept.

Product Line Sales/Support Key

Key	Product Line
A	Analytical
CM	Components
C	Computer Systems
CP	Computer Systems Primary Service Responsible Office (SRO)
CS	Computer Systems Secondary SRO
E	Electronic Instruments & Measurement Systems
M	Medical Products
MP	Medical Products Primary SRO
MS	Medical Products Secondary SRO
P	Personal Computing Products
*	Sales only for specific product line
**	Support only for specific product line

IMPORTANT: These symbols designate general product line capability. They do not insure sales or support availability for all products within a line, at all locations. Contact your local sales office for information regarding locations where HP support is available for specific products.

HP distributors are printed in italics.

ANGOLA
Telectra
Empresa Técnica de Equipamentos
Eléctricos, S.A.R.L.
R. Barbosa Rodrigues, 41-I DT.
Caixa Postal 6487
LUANDA
Tel: 35515,35516
A*,E,M,P

ARGENTINA
Hewlett-Packard Argentina S.A.
Avenida Santa Fe 2035
Martinez 1640 BUENOS AIRES
Tel: 798-5735, 792-1293
Telex: 122443 AR CIGY
Cable: HEWPACKARG
A,E,CP,P

Biotron S.A.C.I.y.M
Avenida Paseo Colon 221
9 Piso
1399 BUENOS AIRES
Tel: 30-4846, 30-1851, 30-8384
Telex: (33)17595 BIONAR
Cable: BIOTRON Argentina
M

Fate S.A. Electronica
Bartolomeu Mitre 833
1036 BUENOS AIRES
Tel: 74-41011, 74-49277,
74-43459
Telex: 18137, 22754
P

AUSTRALIA
**Adelaide, South Australia
Pty. Ltd.**
Hewlett-Packard Australia Pty.Ltd.
153 Greenhill Road
PARKSIDE, S.A. 5063
Tel: 272-5911
Telex: 82536
Cable: HEWPARD Adelaide
A*,CM,CS,E,MS,P

**Brisbane, Queensland
Office**
Hewlett-Packard Australia Pty.Ltd.
5th Floor
Teachers Union Building
495-499 Boundary Street
SPRING HILL, Queensland 4000
Tel: 229-1544
Telex: 42133
Cable: HEWPARD Brisbane
A,CM,CS,E,MS,P

**Canberra, Australia Capital
Office**
Hewlett-Packard Australia Pty.Ltd.
121 Wollongong Street
FYSHWICK, A.C.T. 2069
Tel: 804244
Telex: 62650
Cable: HEWPARD Canberra
A*,CM,CS,E,MS,P

Melbourne, Victoria Office
Hewlett-Packard Australia Pty.Ltd.
31-41 Joseph Street
BLACKBURN, Victoria 3130
Tel: 89-6351
Telex: 31-024
Cable: HEWPARD Melbourne
A,CM,CP,E,MS,P

**Perth, Western Australia
Office**
Hewlett-Packard Austalia Pty.Ltd.
141 Stirling Highway
NEDLANDS, W.A. 6009
Tel: 386-5455
Telex: 93859
Cable: HEWPARD Perth
A,CM,CS,E,MS,P

**Sydney, New South Wales
Office**
Hewlett-Packard Australia Pty.Ltd.
17-23 Talavera Road
NORTH RYDE, N.S.W. 2113
P.O. Box 308
Tel: 887-1611
Telex: 21561
Cable: HEWPARD Sydney
A,CM,CP,E,MS,P

AUSTRIA
Hewlett-Packard Ges.m.b.h.
Grottenhofstrasse 94
Verkaufsburo Graz
8052 GRAZ
Tel: 21-5-66
Telex: 32375
CM,C*,E*

Hewlett-Packard Ges.m.b.h.
Wehlistrasse 29
P.O. Box 7
A-1205 VIENNA
Tel: (222) 35-16-210
Telex: 135823/135066
A,CM,CP,E,MS,P

BAHRAIN
Green Salon
P.O. Box 557
BAHRAIN
Tel: 5503
Telex: 88419
P

Wael Pharmacy
P.O. Box 648
BAHRAIN
Tel: 54886, 56123
Telex: 8550 WAEL GJ
M

BELGIUM
Hewlett-Packard Belgium S.A./N.V.
Blvd de la Woluwe, 100
Woluwedal
B-1200 BRUSSELS
Tel: (02) 762-32-00
Telex: 23-494 paloben bru
A,CM,CP,E,MP,P

BRAZIL
Hewlett-Packard do Brasil I.e.C.
Ltda.
Alameda Rio Negro, 750
ALPHAVILLE 06400 Barueri SP
Tel: 421-1311
Telex: 011 23602 HPBR-BR
Cable: HEWPACK Sao Paulo
A,CM,CP,E,MS

Hewlett-Packard do Brasil I.e.C.
Ltda.
Avenida Epitacio Pessoa, 4664
22471 RIO DE JANEIRO-RJ
Tel: 286-0237
Telex: 021-21905 HPBR-BR
Cable: HEWPACK Rio de Janeiro
A,CM,E,MS,P*

BURUNDI
Typomeca S.P.R.L.
B.P. 553
BUJUMBURA
Tel: 2659
P

CANADA
Alberta
Hewlett-Packard (Canada) Ltd.
210, 7220 Fisher Street S.E.
CALGARY, Alberta T2H 2H8
Tel: (403) 253-2713
Telex: 610-821-6141
A,CM,CP,E*,MS,P*

Hewlett-Packard (Canada) Ltd.
11620A-168th Street
EDMONTON, Alberta T5M 3T9
Tel: (403) 452-3670
Telex: 610-831-2431
A,CM,CP,E,MS,P*

British Columbia
Hewlett-Packard (Canada) Ltd.
10691 Shellbridge Way
RICHMOND, British Columbia
V6X 2W7
Tel: (604) 270-2277
Telex: 610-922-5059
A,CM,CP,E*,MS,P*

Manitoba
Hewlett-Packard (Canada) Ltd.
380-550 Century Street
WINNIPEG, Manitoba R3H 0Y1
Tel: (204) 786-6701
A,CM,CS,E,MS,P*

Nova Scotia
Hewlett-Packard (Canada) Ltd.
P.O. Box 931
900 Windmill Road
DARTMOUTH, Nova Scotia B2Y 3Z6
Tel: (902) 469-7820
Telex: 610-271-4482
CM,CP,E*,MS,P*

Ontario
Hewlett-Packard (Canada) Ltd.
552 Newbold Street
LONDON, Ontario N6E 2S5
Tel: (519) 686-9181
Telex: 610-352-1201
A,CM,CS,E*,MS,P*

Hewlett-Packard (Canada) Ltd.
6877 Goreway Drive
MISSISSAUGA, Ontario L4V 1M8
Tel: (416) 678-9430
Telex: 610-492-4246
A,CM,CP,E,MP,P

Hewlett-Packard (Canada) Ltd.
1020 Morrison Drive
OTTAWA, Ontario K2H 8K7
Tel: (613) 820-6483
Telex: 610-563-1636
A,CM,CP,E*,MS,P*

Quebec
Hewlett-Packard (Canada) Ltd.
17500 South Service Road
Trans-Canada Highway
KIRKLAND, Quebec H9J 2M5
Tel: (514) 697-4232
Telex: 610-422-3022
A,CM,CP,E,MP,P*

CHILE
Jorge Calcagni y Cia. Ltda.
Arturo Burhle 065
Casilla 16475
SANTIAGO 9
Tel: 220222
Telex: JCALCAGNI
A,CM,E,M

Olympia (Chile) Ltd.
Rodrico de Araya 1045
Casilla 256-V
SANTIAGO 21
Tel: 25-50-44
Telex: 40-565
C,P

COLOMBIA
Instrumentación
H. A. Langebaek & Kier S.A.
Apartado Aéreo 6287
BOGOTA 1, D.E.
Carrera 7 No. 48-75
BOGOTA, 2 D.E.
Tel: 287-8877
Telex: 44400
Cable: AARIS Bogota
A,CM,E,M,P

COSTA RICA
Cientifica Costarricense S.A.
Avenida 2, Calle 5
San Pedro de Montes de Oca
Apartado 10159
SAN JOSE
Tel: 24-38-20, 24-08-19
Telex: 2367 GALGUR
Cable: GALGUR
CM,E,M

CYPRUS
Telerexa Ltd.
P.O. Box 4809
14C Stassinos Avenue
NICOSIA
Tel: 45628
Telex: 2894
E,M,P

CZECHOSLOVAKIA
Hewlett-Packard
Obchodni Zastupitelstvi v CSSR
Post. schranka 27
CS-118 01 PRAHA 011
Tel: 66-296
Telex: 121353 IHC

DENMARK
Hewlett-Packard A/S
Datavej 52
DK-3460 BIRKEROD
Tel: (02) 81-66-40
Telex: 37409 hpas dk
A,CM,CP,E,MS,P

Hewlett-Packard A/S
Navervej 1
DK-8600 SILKEBORG
Tel: (06) 82-71-66
Telex: 37409 hpas dk
CM,CS,E

ECUADOR
CYEDE Cia. Ltda.
P.O. Box 6423 CCI
Avenida Eloy Alfaro 1749
QUITO
Tel: 450-975, 243-052
Telex: 2548 CYEDE ED
Cable: CYEDE-Quito
A,CM,E,P

Hospitalar S.A.
Casilla 3590
Robles 625
QUITO
Tel: 545-250, 545-122
Cable: HOSPITALAR-Quito
M

EGYPT
Samitro
Sami Amin Trading Office
18 Abdel Aziz Gawish
ABDINE-CAIRO
Tel: 24-932
P

International Engineering Associates
24 Hussein Hegazi Street
Kasr-el-Aini
CAIRO
Tel: 23-829
Telex: 93830
E,M

Informatic For Computer Systems
22 Talaat Harb Street
CAIRO
Tel: 759006
Telex: 93938 FRANK UN
C

EL SALVADOR
IPESA
Boulevard de los Heroes
Edificio Sarah 1148
SAN SALVADOR
Tel: 252787
A,C,CM,E,P

FINLAND
Hewlett-Packard Oy
Revontulentie 7
SF-02100 ESPOO 10
Tel: (90) 455-0211
Telex: 121563 hewpa sf
A,CM,CP,E,MS,P

FRANCE
Hewlett-Packard France
Le Ligoures
Bureau de Vente de
Aix-en-Provence
Place Romée de Villeneuve
F-13090 AIX-EN-PROVENCE
Tel: (42) 59-41-02
Telex: 410770F
A,CM,CS,E,MS,P*

FRANCE (Cont.)
Hewlett-Packard France
Boite Postale No. 503
F-25026 BESANCON
28 Rue de la Republique
F-25000 BESANCON
Tel: (81) 83-16-22
C,M

Hewlett-Packard France
Bureau de Vente de Lyon
Chemin des Mouilles
Boite Postale No. 162
F-69130 ECULLY Cédex
Tel: (78) 33-81-25
Telex: 310617F
A,CM,CP,E,MP

Hewlett-Packard France
Immeuble France Evry
Tour Lorraine
Boulevard de France
F-91035 EVRY Cédex
Tel: (60) 77-96-60
Telex: 692315F
CM,E

Hewlett-Packard France
5th Avenue Raymond Chanas
F-38320 EYBENS
Tel: (76) 25-81-41
Telex: 980124 HP GRENOB EYBE
CM,CS

Hewlett-Packard France
Bâtiment Ampère
Rue de la Commune de Paris
Boite Postale 300
F-93153 LE BLANC MESNIL
Tel: (01) 865-44-52
Telex: 211032F
CM,CP,E,MS

Hewlett-Packard France
Le Montesquieu
Avenue du President JF Kennedy
F-33700 MERIGNAC
Tel: (56) 34-00-84
Telex: 550105F
CM,CP,E,MS

Hewlett-Packard France
32 Rue Lothaire
F-57000 METZ
Tel: (87) 65-53-50
CM,CS

Hewlett-Packard France
F-91947 Les Ulis CédexORSAY
Tel: (1) 907-78-25
Telex: 600048F
A,CM,CP,E,MP,P

Hewlett-Packard France
Paris Porte-Maillot 13, 15 25
Boulevard De L'Amiral Bruix
F-75782 PARIS Cédex 16
Tel: (01) 502-12-20
Telex: 613663F
CM,CP,MS,P

Hewlett-Packard France
2 Allee de la Bourgonette
F-35100 RENNES
Tel: (99) 51-42-44
Telex: 740912F
CM,CS,E,MS,P*

Hewlett-Packard France
4 Rue Thomas Mann
Boite Postale 56
F-67200 STRASBOURG
Tel: (88) 28-56-46
Telex: 890141F
CM,CS,E,MS,P*

Hewlett-Packard France
20 Chemin de la Cépière
F-31081 TOULOUSE Cédex
Tel: (61) 40-11-12
Telex: 531639F
A,CM,CS,E,P*

Hewlett-Packard France
Bureau de Vente de Lille
Immeuble Péricentre
Rue Van Gogh
F-59650 VILLENEUVE D'ASQ
Tel: (20) 91-41-25
Telex: 160124F
CM,CS,E,MS,P*

GERMAN FEDERAL REPUBLIC
Hewlett-Packard GmbH
Technisches Büro Berlin
Keithstrasse 2-4
D-1000 BERLIN 30
Tel: (030) 24-90-86
Telex: 018 3405 hpbln d
A,CM,CS,E,X,M,P

Hewlett-Packard GmbH
Technisches Büro Böblingen
Herrenberger Strasse 110
D-7030 BÖBLINGEN
Tel: (07031) 667-1
Telex: 07265739 bbn or 07265743
A,CM,CP,E,MP,P

Hewlett-Packard GmbH
Technisches Büro Dusseldorf
Emanuel-Leutze-Strasse 1
D-4000 DUSSELDORF
Tel: (0211) 5971-1
Telex: 085/86 533 hpdd d
A,CM,CP,E,MS,P

Hewlett-Packard GmbH
Vertriebszentrale Frankfurt
Berner Strasse 117
Postfach 560 140
D-6000 FRANKFURT 56
Tel: (0611) 50-04-1
Telex: 04 13249 hpffm d
A,CM,CP,E,MP,P

Hewlett-Packard GmbH
Technisches Büro Hamburg
Kapstadtring 5
D-2000 HAMBURG 60
Tel: (040) 63804-1
Telex: 21 63 032 hphh d
A,CM,CP,E,MS,P

Hewlett-Packard GmbH
Technisches Büro Hannover
Am Grossmarkt 6
D-3000 HANNOVER 91
Tel: (0511) 46-60-01
Telex: 092 3259
A,CM,CS,E,MS,P

Hewlett-Packard GmbH
Technisches Büro Mannheim
Rosslauer Weg 2-4
D-6800 MANNHEIM
Tel: (621) 70050
Telex: 0462105
A,C,E

Hewlett-Packard GmbH
Technisches Büro Neu Ulm
Messerschmittstrasse 7
D-7910 NEU ULM
Tel:
Telex:
C,E

Hewlett-Packard GmbH
Technisches Büro Nürnberg
Neumeyerstrasse 90
D-8500 NÜRNBERG
Tel: (0911) 56-30-83
Telex: 0623 860
CM,CS,E,MS,P

Hewlett-Packard GmbH
Technisches Büro München
Eschenstrasse 5
D-8021 TAUFKIRCHEN
Tel: (089) 6117-1
Telex: 0524985
A,CM,CP,E,MS,P

GREAT BRITAIN
Hewlett-Packard Ltd.
Trafalgar House
Navigation Road
ALTRINCHAM
Chesire WA14 1NU
Tel: (061) 928-6422
Telex: 668068
A,C,E,M

Hewlett-Packard Ltd.
Oakfield House, Oakfield Grove
Clifton
BRISTOL BS8 2BN
Tel: 36806
Telex: 444302
P

Hewlett-Packard Ltd.
14 Wesley Street
CASTLEFORD
Yorkshire WF10 1AE
Tel: (0977) 550016
Telex: 5557355
C

Hewlett-Packard Ltd.
Fourier House
257-263 High Street
LONDON COLNEY
Herts., AL2 1HA
Tel: (0727) 24400
Telex: 1-8952716
C,E

Hewlett-Packard Ltd
Tradax House, St. Mary's Walk
MAIDENHEAD
Berkshire, SL6 1ST
Tel: (0628) 39151
E,P

Hewlett-Packard Ltd.
308/314 Kings Road
READING, Berkshire
Tel: 61022
Telex: 84-80-68
CM,P

Hewlett-Packard Ltd.
Quadrangle
106-118 Station Road
REDHILL, Surrey
Tel: (0737) 68655
Telex: 947234 C,E

Hewlett-Packard Ltd.
Westminster House
190 Stratford Road
SHIRLEY, Solihull
West Midlands B90 3BJ
Tel: (021) 7458800
Telex: 339105
C

Hewlett-Packard Ltd.
King Street Lane
WINNERSH, Wokingham
Berkshire RG11 5AR
Tel: (0734) 784774
Telex: 847178
A,C,E,M

GREECE
Kostas Karaynnis
8 Omirou Street
ATHENS 133
Tel: 32-30-303, 32-37-371
Telex: 21 59 62 RKAR GR
E,M,P

"Plaiso"
G. Gerados
24 Stournara Street
ATHENS
Tel: 36-11-160
Telex: 21 9492
P

GUATEMALA
IPESA
Avenida Reforma 3-48
Zona 9
GUATEMALA CITY
Tel: 316627, 314786, 664715
Telex: 4192 Teletro Gu
A,C,CM,E,M,P

HONG KONG
Hewlett-Packard Hong Kong, Ltd.
G.P.O. Box 795
5th Floor, Sun Hung Kai Centre
30 Harbour Road
HONG KONG
Tel: 5-8323211
Telex: 66678 HEWPA HX
Cable: HP ASIA LTD Hong Kong
E,CP,P

Schmidt & Co. (Hong Kong) Ltd.
Wing On Centre, 28th Floor
Connaught Road, C.
HONG KONG
Tel: 5-455644
Telex: 74766 SCHMX HX
A,M

ICELAND
Elding Trading Company Inc.
Hafnarnvoli-Tryggvagotu
P.O. Box 895
IS-REYKJAVIK
Tel: 1-58-20, 1-63-03
M

INDIA
Blue Star Ltd.
Bhavdeep
Stadium Road
AHMEDABAD 380 014
Tel: 42932
Telex: 012-234
Cable: BLUEFROST
E

Blue Star Ltd.
11 Magarath Road
BANGALORE 560 025
Tel: 55668
Telex: 0845-430
Cable: BLUESTAR
A,CM,C,E

Blue Star Ltd.
Band Box House
Prabhadevi
BOMBAY 400 025
Tel: 422-3101
Telex: 011-3751
Cable: BLUESTAR
A,M

Blue Star Ltd.
Sahas
414/2 Vir Savarkar Marg
Prabhadevi
BOMBAY 400 025
Tel: 422-6155
Telex: 011-4093
Cable: FROSTBLUE
A,CM,C,E,M

Blue Star Ltd.
7 Hare Street
CALCUTTA 700 001
Tel: 12-01-31
Telex: 021-7655
Cable: BLUESTAR
A,M

Blue Star Ltd.
Meenakshi Mandiram
XXXXV/1379-2 M. G. Road
COCHIN 682-016
Tel: 32069
Telex: 085-514
Cable: BLUESTAR
A*

Blue Star Ltd.
133 Kodambakkam High Road
MADRAS 600 034
Tel: 82057
Telex: 041-379
Cable: BLUESTAR
A,M

Blue Star Ltd.
Bhandari House, 7th/8th Floors
91 Nehru Place
NEW DELHI 110 024
Tel: 682547
Telex: 031-2463
Cable: BLUESTAR
A,CM,C,E,M

Blue Star Ltd.
1-1-117/1 Sarojini Devi Road
SECUNDERABAD 500 033
Tel: 70126
Telex: 0155-459
Cable: BLUEFROST
A,E

Blue Star Ltd.
T.C. 7/603 Poornima
Maruthankuzhi
TRIVANDRUM 695 013
Tel: 65799
Telex: 0884-259
Cable: BLUESTAR
E

INDONESIA
BERCA Indonesia P.T.
P.O.Box 496/Jkt.
Jln. Abdul Muis 62
JAKARTA
Tel: 373009
Telex: 31146 BERSAL IA
Cable: BERSAL JAKARTA
A,C,E,M,P

BERCA Indonesia P.T.
P.O. Box 174/Sby.
J.L. Kutei No. 11
SUBAEE-SURABAYA
Tel: 68172
Telex: 31146 BERSAL SD
Cable: BERSAL-SURABAYA
A*,E,M,P

IRAQ
Hewlett-Packard Trading S.A.
Mansoor City 9B/3/7
BAGHDAD
Tel: 551-49-73
Telex: 2455 HEPAIRAQ IK
CP

IRELAND
Hewlett-Packard Ireland Ltd.
Kestrel House
Clanwilliam Court
Lower Mount Street
DUBLIN 2, Eire
Tel: 680424, 680426
Telex: 30439
A,C,CM,E,M,P

Cardiac Services Ltd.
Kilmore Road
Artane
DUBLIN 5, Eire
Tel: (01) 351820
Telex: 30439
M

ISRAEL
Electronics Engineering Division
Motorola Israel Ltd.
16 Kremenetski Street
P.O. Box 25016
TEL-AVIV 67899
Tel: 338973
Telex: 33569 Motil IL
Cable: BASTEL Tel-Aviv
A,CM,C,E,M,P

ITALY
Hewlett-Packard Italiana S.p.A.
Traversa 99C
Giulio Petrone, 19
I-70124 **BARI**
Tel: (080) 41-07-44
M

Hewlett-Packard Italiana S.p.A.
Via Martin Luther King, 38/111
I-40132 **BOLOGNA**
Tel: (051) 402394
Telex: 511630
CM,CS,E,MS

Hewlett-Packard Italiana S.p.A.
Via Principe Nicola 43G/C
I-95126 **CATANIA**
Tel: (095) 37-10-87
Telex: 970291
C,P

Hewlett-Packard Italiana S.p.A.
Via G. Di Vittorio 9
I-20063 **CERNUSCO SUL NAVIGLIO**
Tel: (2) 903691
Telex: 334632
A,CM,CP,E,MP,P

Hewlett-Packard Italiana S.p.A.
Via Nuova san Rocco A
Capodimonte, 62/A
I-80131 **NAPOLI**
Tel: (081) 7413544
A,CM,CS,E

Hewlett-Packard Italiana S.p.A.
Viale G. Modugno 33
I-16156 **GENOVA PEGLI**
Tel: (010) 68-37-07 E,C

Hewlett-Packard Italiana S.p.A.
Via Turazza 14
I-35100 **PADOVA**
Tel: (49) 664888
Telex: 430315
A,CM,CS,E,MS

Hewlett-Packard Italiana S.p.A.
Viale C. Pavese 340
I-00144 **ROMA**
Tel: (06) 54831
Telex: 610514
A,CM,CS,E,MS,P*

Hewlett-Packard Italiana S.p.A.
Corso Giovanni Lanza 94
I-10133 **TORINO**
Tel: (011) 682245, 659308
Telex: 221079
CM,CS,E

JAPAN
Yokogawa-Hewlett-Packard Ltd.
Inoue Building
1348-3, Asahi-cho
ATSUGI, Kanagawa 243
Tel: (0462) 24-0451
CM,C*,E

Yokogawa-Hewlett-Packard Ltd.
3-30-18 Tsuruya-cho
Kanagawa-ku, Yokohama-Shi
KANAGAWA, 221
Tel: (045) 312-1252
Telex: 382-3204 YHP YOK
CM,CS,E

Yokogawa-Hewlett-Packard Ltd.
Sannomiya-Daiichi Seimei-Bldg. 5F
69 Kyo-Machi Ikuta-Ku
KOBE CITY 650 Japan
Tel: (078) 392-4791
C,E

Yokogawa-Hewlett-Packard Ltd.
Kumagaya Asahi Yasoji Bldg 4F
4-3 Chome Tsukuba
KUMAGAYA, Saitama 360
Tel: (0485) 24-6563
CM,CS,E

Yokogawa-Hewlett-Packard Ltd.
Mito Mitsui Building
4-73, San-no-maru, 1-chome
MITO, Ibaragi 310
Tel: (0292) 25-7470
CM,CS,E

Yokogawa-Hewlett-Packard Ltd.
Sumitomo Seimei Bldg.
11-2 Shimo-sasajima-cho
Nakamura-ku
NAGOYA, Aichi 450
Tel: (052) 581-1850
CM,CS,E,MS

Yokogawa-Hewlett-Packard Ltd.
Chuo Bldg., 4th Floor
5-4-20 Nishinakajima, 5-chome
Yodogawa-ku, Osaka-shi
OSAKA, 532
Tel: (06) 304-6021
Telex: YHPOSA 523-3624
A,CM,CP,E,MP,P*

Yokogawa-Hewlett-Packard Ltd.
29-21 Takaido-Higashi 3-chome
Suginami-ku **TOKYO** 168
Tel: (03) 331-6111
Telex: 232-2024 YHPTOK
A,CM,CP,E,MP,P*

JORDAN
Mouasher Cousins Company
P.O. Box 1387
AMMAN
Tel: 24907, 39907
Telex: 21456 SABCO JO
E,M,P

KOREA
Samsung Electronics
4759 Shinkil, 6 Dong
Youngdeungpo-Ku,
SEOUL
Tel: 8334311, 8334312
Telex: SAMSAN 27364
A,C,E,M,P

KUWAIT
Al-Khalidya Trading & Contracting
P.O. Box 830 Safat
KUWAIT
Tel: 42-4910, 41-1726
Telex: 2481 Areeg kt
A,E,M

Photo & Cine Equipment
P.O. Box 270 Safat
KUWAIT
Tel: 42-2846, 42-3801
Telex: 2247 Matin
P

LUXEMBOURG
Hewlett-Packard Belgium S.A./N.V.
Blvd de la Woluwe, 100
Woluwedal
B-1200 **BRUSSELS**
Tel: (02) 762-32-00
Telex: 23-494 paloben bru
A,CM,CP,E,MP,P

MALAYSIA
Hewlett-Packard Sales (Malaysia)
Sdn. Bhd.
Suite 2.21/2.22
Bangunan Angkasa Raya
Jalan Ampang
KUALA LUMPUR
Tel: 483544
Telex: MA31011
A,CP,E,M,P*

Protel Engineering
Lot 319, Satok Rd.
P.O. Box 1917
KUCHING, SARAWAK
Tel: 535-44
Telex: MA 70904 Promal
Cable: Proteleng
A,E,M

MEXICO
Hewlett-Packard Mexicana, S.A. de
C.V.
Avenida Periferico Sur No. 6501
Tepepan, Xochimilco
MEXICO CITY 23, D.F.
Tel: (905) 676-4600
Telex: 017-74-507
A,CP,E,MS,P

Hewlett-Packard Mexicana, S.A. de
C.V.
Rio Volga 600
Colonia del Valle
MONTERREY, N.L.
Tel: 78-42-93, 78-42-40, 78-42-41
Telex: 038-410
CS

MOROCCO
Dolbeau
81 rue Karatchi
CASABLANCA
Tel: 3041-82, 3068-38
Telex: 23051, 22822
E

Gerep
2 rue d'Agadir
Boite Postale 156
CASABLANCA
Tel: 272093, 272095
Telex: 23 739
P

NETHERLANDS
Hewlett-Packard Nederland B.V.
Van Heuven Goedhartlaan 121
NL 1181KK **AMSTELVEEN**
P.O. Box 667
NL1080 AR **AMSTELVEEN**
Tel: (20) 47-20-21
Telex: 13 216
A,CM,CP,E,MP,P

Hewlett-Packard Nederland B.V.
Bongerd 2
NL 2906VK **CAPPELLE**, A/D Ijessel
P.O. Box 41
NL2900 AA **CAPELLE**, Ijssel
Tel: (10) 51-64-44
Telex: 21261 HEPAC NL
A,CM,CP

NEW ZEALAND
Hewlett-Packard (N.Z.) Ltd.
169 Manukau Road
P.O. Box 26-189
Epsom, **AUCKLAND**
Tel: 68-7159
Cable: HEWPACK Auckland
CM,CS,E,P*

Hewlett-Packard (N.Z.) Ltd.
4-12 Cruickshank Street
P.O. Box 9443
Kilbirnie, **WELLINGTON** 3
Tel: 877-199
Cable: HEWPACK Wellington
CM,CP,E,P

Northrop Instruments & Systems
Ltd.
Eden House, 44 Khyber Pass Road
P.O. Box 9682
Newmarket, AUCKLAND
Tel: 794-091
A,M

Northrop Instruments & Systems
Ltd.
Terrace House, 4 Oxford Terrace
P.O. Box 8388
CHRISTCHURCH
Tel: 64-165
A,M

Northrop Instruments & Systems
Ltd.
Sturdee House
85-87 Ghuznee Street
P.O. Box 2406
WELLINGTON
Tel: 850-091
Telex: NZ 3380
A,M

NIGERIA
The Electronics Instrumentations
Ltd.
N6B/S70 Oyo Road
Oluseun House
P.M.B. 5402
IBADAN
Tel: 461577
Telex: 31231 TEIL NG
A,E,M,P

The Electronics Instrumentations
Ltd.
144 Agege Motor Road, Mushin
P.O. Box 6645
Mushin, LAGOS
A,E,M,P

NORTHERN IRELAND
Cardiac Services Company
95A Finaghy Road South
BELFAST BT 10 0BY
Tel: (0232) 625-566
Telex: 747626
M

NORWAY
Hewlett-Packard Norge A/S
Folke Bernadottesvei 50
P.O. Box 3558
N-5033 **FYLLINGSDALEN (BERGEN)**
Tel: (05) 16-55-40
Telex: 16621 hpnas n
CM,CS,E

Hewlett-Packard Norge A/S
Oesterndalen 18
P.O. Box 34
N-1345 **OESTERAAS**
Tel: (02) 17-11-80
Telex: 16621 hpnas n
A*,CM,CP,E,MS,P

OMAN
Khimji Ramdas
P.O. Box 19
MUSCAT
Tel: 72-22-17, 72-22-25
Telex: 3289 BROKER MB MUSCAT
P

PAKISTAN
Mushko & Company Ltd.
10, Bazar Road
Sector G-6/4
ISLAMABAD
Tel: 28624
Cable: FEMUS Rawalpindi
A,E,M

Mushko & Company Ltd.
Oosman Chambers
Abdullah Haroon Road
KARACHI 0302
Tel: 511027, 512927
Telex: 2894 MUSHKO PW
Cable: COOPERATOR Karachi
*A,E,M,P**

PANAMA
Electrónico Balboa, S.A.
Apartado 4929
Panama 5
Calle Samuel Lewis
Edificio "Alfa" No. 2
CIUDAD DE PANAMA
Tel: 64-2700
Telex: 3480380
Cable: ELECTRON Panama
A,CM,E,M,P

Foto Internacional, S.A.
P.O. Box 2068
Free Zone of Colon
COLON 3
Tel: 45-2333
Telex: 3485126
Cable: IMPORT COLON/Panama
P

PERU
Cómpania Electro Médica S.A.
Los Flamencos 145, San Isidro
Casilla 1030
LIMA 1
Tel: 41-4325
Telex: Pub. Booth 25424 SISIDRO
Cable: ELMED Lima
A,CM,E,M,P

PHILIPPINES
The Online Advanced Systems
Corporation
Rico House, Amorsolo Cor. Herrera
Street
Legaspi Village, Makati
P.O. Box 1510
Metro MANILA
Tel: 85-35-81, 85-34-91, 85-32-21
Telex: 3274 ONLINE
A,C,E,M

Electronic Specialists and
Proponents Inc.
690-B Epifanio de los Santos
Avenue
Cubao, QUEZON CITY
P.O. Box 2649 Manila
Tel: 98-96-81, 98-96-82, 98-96-83
Telex: 742-40287
P

POLAND
Buro Informasji Technicznej
Hewlett-Packard
Ul Stawki 2, 6P
PL00-950 WARSZAWA
Tel: 39-59-62, 39-67-43
Telex: 812453 hepa pl

SALES & SUPPORT OFFICES
Arranged alphabetically by country

PORTUGAL
Telectra-Empresa Técnica de
Equipmentos Eléctricos S.a.r.l.
Rua Rodrigo da Fonseca 103
P.O. Box 2531
P-LISBON 1
Tel: (19) 68-60-72
Telex: 12598
A,C,E,P

Mundinter
Intercambio Mundial de Comércio
S.a.r.l
P.O. Box 2761
Avenida Antonio Augusto de Aguiar
138
P-LISBON
Tel: (19) 53-21-31, 53-21-37
Telex: 16691 munter p
M

PUERTO RICO
Hewlett-Packard Puerto Rico
P.O. Box 4407
CAROLINA, Puerto Rico 00630
Calle 272 Edificio 203
Urb. Country Club
RIO PIEDRAS, Puerto Rico 00924
Tel: (809) 762-7255
Telex: 345 0514
A,CP

QATAR
Nasser Trading & Contracting
P.O. Box 1563
DOHA
Tel: 22170
Telex: 4439 NASSER
M

Scitecharabia
P.O. Box 2750
!'DOHA
Tel: 329515
Telex: 4806 CMPARB
P

ROMANIA
Hewlett-Packard Reprezentanta
Boulevard Nicolae Balcescu 16
BUCURESTI
Tel: 130725
Telex: 10440

SAUDI ARABIA
Modern Electronic Establishment
P.O. Box 193
AL-KHOBAR
Tel: 44-678, 44-813
Telex: 670136
Cable: ELECTA AL-KHOBAR
C,E,M,P

Modern Electronic Establishment
P.O. Box 1228, Baghdadiah Street
JEDDAH
Tel: 27-798
Telex: 401035
Cable: ELECTA JEDDAH
C,E,M,P

Modern Electronic Establishment
P.O. Box 2728
RIYADH
Tel: 62-596, 66-232
Telex: 202049
C,E,M,P

SCOTLAND
Hewlett-Packard Ltd.
Royal Bank Buildings
Swan Street
BRECHIN, Angus, Scotland
Tel: 3101, 3102
CM,CS

Hewlett-Packard Ltd.
SOUTH QUEENSFERRY
West Lothian, EH30 9TG
GB-Scotland
Tel: (031) 3311000
Telex: 72682
A,CM,E,M

SINGAPORE
Hewlett-Packard Singapore (Pty.)
Ltd.
P.O. Box 58 Alexandra Post Office
SINGAPORE, 9115
6th Floor, Inchcape House
450-452 Alexandra Road
SINGAPORE 0511
Tel: 631788
Telex: HPSGSO RS 34209
Cable: HEWPACK, Singapore
A,CP,E,MS,P

SOUTH AFRICA
Hewlett-Packard South Africa (Pty.)
Ltd.
P.O. Box 120
Howard Place
Pine Park Center, Forest Drive,
Pinelands
CAPE PROVINCE 7450
Tel: 53-7955, 53-7956, 53-7957
Telex: 57-0006
A,CM,CS,E,MS,P

Hewlett-Packard South Africa (Pty.)
Ltd.
P.O. Box 37066
Overport
DURBAN 4067
Tel: 28-4178, 28-4179, 28-4110
CM,CS

Hewlett-Packard South Africa (Pty.)
Ltd.
P.O. Box 33345
Glenstantia 0010 TRANSVAAL
1st Floor East
Constantia Park Ridge Shopping
Centre
Constantia Park
PRETORIA Tel: 98-1126 or 98-1220
Telex: 32163
C,E

Hewlett-Packard South Africa (Pty.)
Ltd.
Daphny Street
Private Bag Wendywood
SANDTON 2144
Tel: 802-5111, 802-5125
Telex: 89-84782
Cable: HEWPACK Johannesburg
A,CM,CP,E,MS,P

SPAIN
Hewlett-Packard Española S.A.
c/Entenza, 321
E-BARCELONA 29
Tel: (3) 322-24-51, 321-73-54
Telex: 52603 hpbee
A,CM,CP,E,MS,P

Hewlett-Packard Española S.A.
c/San Vicente S/N
Edificio Albia II,7 B
E-BILBAO 1
Tel: (944) 423-8306, 423-8206
A,CM,E,MS

Hewlett-Packard Española S.A.
Calle Jerez 3
E-MADRID 16
Tel: 458-2600
Telex: 23515 hpe
A,CM,E,MP,P

Hewlett-Packard Española S.A.
Colonia Mirasierra
Edificio Juban
c/o Costa Brava 13, 2.
E-MADRID 34
Tel: 734-8061, 734-1162
CM,CP

Hewlett-Packard Española S.A.
Av Ramón y Cajal 1-9
Edificio Sevilla 1,
E-SEVILLA 5
Tel: 64-44-54, 64-44-58
Telex: 72933
A,CM,CS,MS,P

Hewlett-Packard Española S.A.
C/Ramon Gordillo, 1 (Entlo.3)
E-VALENCIA 10
Tel: 361-1354, 361-1358
CM,CS,P

SWEDEN
Hewlett-Packard Sverige AB
Enighetsvägen 3, Fack
P.O. Box 20502
S-16120 BROMMA
Tel: (08) 730-0550
Telex: (854) 10721 MESSAGES
Cable: MEASUREMENTS
STOCKHOLM
A,CM,CP,E,MS,P

Hewlett-Packard Sverige AB
Sunnanvagen 14K
S-22226 LUND
Tel: (46) 13-69-79
Telex: (854) 10721 (via BROMMA
office)
CM,CS

Hewlett-Packard Sverige AB
Vastra Vintergatan 9
S-70344 OREBRO
Tel: (19) 10-48-80
Telex: (854) 10721 (via BROMMA
office)
CM,CS

Hewlett-Packard Sverige AB
Frötallisgatan 30
S-42132 VÄSTRA-FRÖLUNDA
Tel: (031) 49-09-50
Telex: (854) 10721 (via BROMMA
office)
CM,CS,E,P

SWITZERLAND
Hewlett-Packard (Schweiz) AG
Clarastrasse 12
CH-4058 BASLE
Tel: (61) 33-59-20
A,CM

Hewlett-Packard (Schweiz) AG
47 Avenue Blanc
CH-1202 GENEVA
Tel: (022) 32-30-05, 32-48-00
CM,CP

Hewlett-Packard (Schweiz) AG
29 Chemin Château Bloc
CH-1219 LE LIGNON-Geneva
Tel: (022) 96-03-22
Telex: 27333 hpag ch
Cable: HEWPACKAG Geneva
A,CM,E,MS,P

Hewlett-Packard (Schweiz) AG
Zürcherstrasse 20
Allmend 2
CH-8967 WIDEN
Tel: (57) 50-111
Telex: 59933 hpag ch
Cable: HPAG CH
A,CM,CP,E,MS,P

SYRIA
General Electronic Inc.
Nuri Basha-Ahnat Ebn Kays Street
P.O. Box 5781
DAMASCUS
Tel: 33-24-87
Telex: 11215 ITIKAL
Cable: ELECTROBOR DAMASCUS
E

Sawah & Co.
Place Azmé
Boite Postale 2308
DAMASCUS
Tel: 16-367, 19-697, 14-268
Telex: 11304 SATACO SY
Cable: SAWAH, DAMASCUS
M

TAIWAN
Hewlett-Packard Far East Ltd.
Kaohsiung Branch
68-2, Chung Cheng 3rd Road
Shin Shin, Chu
KAOHSIUNG
Tel: 24-2318, 26-3253
CS,E,MS,P

Hewlett-Packard Far East Ltd.
Taiwan Branch
5th Floor
205 Tun Hwa North Road
TAIPEI
Tel:(02) 751-0404
Cable:HEWPACK Taipei
A,CP,E,MS,P

Hewlett-Packard Far East Ltd.
Taichung Branch
#33, Cheng Yih Street
10th Floor, Room 5
TAICHUNG
Tel: 289274

Ing Lih Trading Co.
3rd Floor 18, Po-la Road
TAIPEI
Tel:
Telex:
Cable: INGLIH TAIPEI
A

THAILAND
UNIMESA Co. Ltd.
Elcom Research Building
2538 Sukhumvit Ave.
Bangchak, BANGKOK
Tel: 393-2387, 393-0338
Telex: TH81160, 82938, 81038
Cable: UNIMESA Bangkok
A,C,E,M

Bangkok Business Equipment Ltd.
5/5-6 Dejo Road
BANGKOK
Tel: 234-8670, 234-8671,
234-8672
Cable: BUSIQUIPT Bangkok
P

TRINIDAD & TOBAGO
Caribbean Telecoms Ltd.
P.O. Box 732
50/A Jerningham Avenue
PORT-OF-SPAIN
Tel: 624-4213, 624-4214
A,CM,E,M,P

TUNISIA
Tunisie Electronique
31 Avenue de la Liberte
TUNIS
Tel: 280-144
E,P

Corema
1 ter. Av. de Carthage
TUNIS
Tel: 253-821
Telex: 12319 CABAM TN
M

TURKEY
Teknim Company Ltd.
Riza Sah Pehievi
Caddesi No. 7
Kavaklidere, ANKARA
Tel: 275800
Telex: 42155
E

EMA, Muhendislik Kollektif Sirketi
Mediha Eldem
Sokak 41/6
Yüksel Caddesi, ANKARA
Tel: 17-56-22
Cable: Ematrade
M

UNITED ARAB EMIRATES
Emitac Ltd.
P.O. Box 1641
SHARJAH
Tel: 354121, 354123
Telex: 68136
E,M,P,C

UNITED KINGDOM
see: GREAT BRITAIN
NORTHERN IRELAND
SCOTLAND

UNITED STATES
Alabama
Hewlett-Packard Co.
700 Century Park South
Suite 128
BIRMINGHAM, AL 35226
Tel: (205) 822-6802
CM,CS,MP

Hewlett-Packard Co.
P.O. Box 4207
8290 Whitesburg Drive, S.E.
HUNTSVILLE, AL 35802
Tel: (205) 881-4591
CM,CP,E,M*

Alaska
Hewlett-Packard Co.
1577 "C" Street, Suite 252
ANCHORAGE, AK 99510
Tel: (206) 454-3971
CM,CS**

Arizona
Hewlett-Packard Co.
2336 East Magnolia Street
PHOENIX, AZ 85034
Tel: (602) 273-8000
A,CM,CP,E,MS

Hewlett-Packard Co.
2424 East Aragon Road
TUCSON, AZ 85702
Tel: (602) 889-4631
CM,CS,E,MS**

Arkansas
Hewlett-Packard Co.
P.O. Box 5646
Brady Station
LITTLE ROCK, AR 72215
Tel: (501) 376-1844, (501)
664-8773
CM,MS

UNITED STATES (Cont.)

California
Hewlett-Packard Co.
7621 Canoga Avenue
CANOGA PARK, CA 91304
Tel: (213) 702-8300
A,CM,CP,E,P

Hewlett-Packard Co.
1579 W. Shaw Avenue
FRESNO, CA 93771
Tel: (209) 224-0582
CM,MS

Hewlett-Packard Co.
1430 East Orangethorpe
FULLERTON, CA 92631
Tel: (714) 870-1000
CM,CP,E,MP

Hewlett-Packard Co.
5400 W. Rosecrans Boulevard
LAWNDALE, CA 90260
P.O. Box 92105
LOS ANGELES, CA 90009
Tel: (213) 970-7500
CM,CP,MP

Hewlett-Packard Co.
3939 Lankershim Blvd.
NORTH HOLLYWOOD, CA 91604
Tel: (213) 877-1282
Regional Headquarters

Hewlett-Packard Co.
3200 Hillview Avenue
PALO ALTO, CA 94304
Tel: (415) 857-8000
CM,CP,E

Hewlett-Packard Co.
646 W. North Market Boulevard
SACRAMENTO, CA 95834
Tel: (916) 929-7222
A*,CM,CP,E,MS

Hewlett-Packard Co.
9606 Aero Drive
P.O. Box 23333
SAN DIEGO, CA 92123
Tel: (714) 279-3200
CM,CP,E,MP

Hewlett-Packard Co.
3003 Scott Boulevard
SANTA CLARA, CA 95050
Tel: (408) 988-7000
A,CM,CP,E,MP

Hewlett-Packard Co.
454 Carlton Court
SO. SAN FRANCISCO, CA 94080
Tel: (415) 877-0772
CM,CP

Colorado
Hewlett-Packard Co.
24 Inverness Place, East
ENGLEWOOD, CO 80112
Tel: (303) 771-3455
A,CM,CP,E,MS

Connecticut
Hewlett-Packard Co.
47 Barnes Industrial Road South
P.O. Box 5007
WALLINGFORD, CT 06492
Tel: (203) 265-7801
A,CM,CP,E,MS

Florida
Hewlett-Packard Co.
P.O. Box 24210
2727 N.W. 62nd Street
FORT LAUDERDALE, FL 33309
Tel: (305) 973-2600
CM,CP,E,MP

Hewlett-Packard Co.
4080 Woodcock Drive, #132
Brownett Building
JACKSONVILLE, FL 32207
Tel: (904) 398-0663
CM,C*,E*,MS**

Hewlett-Packard Co.
P.O. Box 13910
6177 Lake Ellenor Drive
ORLANDO, FL 32809
Tel: (305) 859-2900
A,CM,CP,E,MS

Hewlett-Packard Co.
6425 N. Pensacola Blvd.
Suite 4, Building 1
PENSACOLA, FL 32575
Tel: (904) 476-8422
A,CM,MS

Hewlett-Packard Co.
110 South Hoover, Suite 120
Vanguard Bldg.
TAMPA, FL 33609
Tel: (813) 872-0900
A*,CM,CS,E*,M*

Georgia
Hewlett-Packard Co.
P.O. Box 105005
2000 South Park Place
ATLANTA, GA 30339
Tel: (404) 955-1500
Telex: 810-766-4890
A,CM,CP,E,MP

Hewlett-Packard Co.
Executive Park Suite 306
P.O. Box 816
AUGUSTA, GA 30907
Tel: (404) 736-0592
CM,MS

Hewlett-Packard Co.
P.O. Box 2103
1172 N. Davis Drive
WARNER ROBINS, GA 31098
Tel: (912) 922-0449
CM,E

Hawaii
Hewlett-Packard Co.
Kawaiahao Plaza, Suite 190
567 South King Street
HONOLULU, HI 96813
Tel: (808) 526-1555
A,CM,CS,E,MS

Idaho
Hewlett-Packard Co.
11311 Chinden Boulevard
BOISE, ID 83707
Tel: (208) 376-6000
CM,CS,M*

Illinois
Hewlett-Packard Co.
211 Prospect Road
BLOOMINGTON, IL 61701
Tel: (309) 663-0383
CM,CS,MS**

Hewlett-Packard Co.
1100 31st Street
DOWNERS GROVE, IL 60515
Tel: (312) 960-5760
CM,CP

Hewlett-Packard Co.
5201 Tollview Drive
ROLLING MEADOWS, IL 60008
Tel: (312) 255-9800
A,CM,CP,E,MP

Indiana
Hewlett-Packard Co.
P.O. Box 50807
7301 No. Shadeland Avenue
INDIANAPOLIS, IN 46250
Tel: (317) 842-1000
A,CM,CS,E,MS

Iowa
Hewlett-Packard Co.
2415 Heinz Road
IOWA CITY, IA 52240
Tel: (319) 351-1020
CM,CS,E*,MS

Kansas
Hewlett-Packard Co.
1644 S. Rock
WICHITA, KA 67207
Tel: (316) 265-5200
CM,CS

Kentucky
Hewlett-Packard Co.
10170 Linn Station Road
Suite 525
LOUISVILLE, KY 40223
Tel: (502) 426-0100
A,CM,CS,MS

Louisiana
Hewlett-Packard Co.
P.O. Box 1449
3229 Williams Boulevard
KENNER, LA 70062
Tel: (504) 443-6201
A,CM,CS,E,MS

Maryland
Hewlett-Packard Co.
7121 Standard Drive
HANOVER, MD 21076
Tel: (301) 796-7700
A,CM,CP,E,MS

Hewlett-Packard Co.
2 Choke Cherry Road
ROCKVILLE, MD 20850
Tel: (301) 948-6370
Telex: 710-828-9685
A,CM,CP,E,MP

Massachusetts
Hewlett-Packard Co.
32 Hartwell Avenue
LEXINGTON, MA 02173
Tel: (617) 861-8960
A,CM,CP,E,MP

Michigan
Hewlett-Packard Co.
23855 Research Drive
FARMINGTON HILLS, MI 48024
Tel: (313) 476-6400
A,CM,CP,E,MP

Hewlett-Packard Co.
4326 Cascade Road S.E.
GRAND RAPIDS, MI 49506
Tel: (616) 957-1970
CM,CS,MS

Minnesota
Hewlett-Packard Co.
2025 W. Larpenteur Ave.
ST. PAUL, MN 55113
Tel: (612) 644-1100
A,CM,CP,E,MP

Mississippi
Hewlett-Packard Co.
P.O. Box 5028
322 N. Mart Plaza
JACKSON, MS 39216
Tel: (601) 982-9363
CM,MS

Missouri
Hewlett-Packard Co.
11131 Colorado Avenue
KANSAS CITY, MO 64137
Tel: (816) 763-8000
Telex: 910-771-2087
A,CM,CS,E,MS

Hewlett-Packard Co.
1024 Executive Parkway
ST. LOUIS, MO 63141
Tel: (314) 878-0200
A,CM,CP,E,MP

Nebraska
Hewlett-Packard
7101 Mercy Road
Suite 101, IBX Building
OMAHA, NE 68106
Tel: (402) 392-0948
CM,MS

Nevada
Hewlett-Packard Co.
Suite D-130
5030 Paradise Blvd.
LAS VEGAS, NV 89119
Tel: (702) 736-6610
CM,MS**

New Jersey
Hewlett-Packard Co.
Crystal Brook Professional Building
Route 35
EATONTOWN, NJ 07724
Tel: (201) 542-1384
A*,CM,C*,E*,P*

Hewlett-Packard Co.
W120 Century Road
PARAMUS, NJ 07652
Tel: (201) 265-5000
A,CM,CP,E,MP

Hewlett-Packard Co.
60 New England Avenue West
PISCATAWAY, NJ 08854
Tel: (201) 981-1199
A,CM,CP,E

New Mexico
Hewlett-Packard Co.
P.O. Box 11634
11300 Lomas Blvd.,N.E.
ALBUQUERQUE, NM 87123
Tel: (505) 292-1330
Telex: 910-989-1185
CM,CP,E,MS

New York
Hewlett-Packard Co.
5 Computer Drive South
ALBANY, NY 12205
Tel: (518) 458-1550
Telex: 710-444-4691
A,CM,CS,E,MS

Hewlett-Packard Co.
9600 Main Street
CLARENCE, NY 14031
Tel: (716) 759-8621
Telex: 710-523-1893

Hewlett-Packard Co.
200 Cross Keys Office
FAIRPORT, NY 14450
Tel: (716) 223-9950
Telex: 510-253-0092
CM,CP,E,MS

Hewlett-Packard Co.
No. 1 Pennsylvania Plaza
55th Floor
34th Street & 8th Avenue
NEW YORK, NY 10119
Tel: (212) 971-0800
CM,CP,E*,M*

Hewlett-Packard Co.
5858 East Molloy Road
SYRACUSE NY 13211
Tel: (315) 455-2486
A,CM,CS,E,MS

Hewlett-Packard Co.
3 Crossways Park West
WOODBURY, NY 11797
Tel: (516) 921-0300
Telex: 510-221-2183
A,CM,CP,E,MS

North Carolina
Hewlett-Packard Co.
P.O. Box 15579
2905 Guess Road (27705)
DURHAM, NC 27704
Tel: (919) 471-8466
C,M

Nebraska
Hewlett-Packard Co.
5605 Roanne Way
GREENSBORO, NC 27409
Tel: (919) 852-1800
A,CM,CP,E,MS

Ohio
Hewlett-Packard Co.
9920 Carver Road
CINCINNATI, OH 45242
Tel: (513) 891-9870
CM,CP,MS

Hewlett-Packard Co.
16500 Sprague Road
CLEVELAND, OH 44130
Tel: (216) 243-7300
Telex: 810-423-9430
A,CM,CP,E,MS

Hewlett-Packard Co.
962 Crupper Ave.
COLUMBUS, OH 43229
Tel: (614) 436-1041
CM,CP,E*

Hewlett-Packard Co.
330 Progress Rd.
DAYTON, OH 45449
Tel: (513) 859-8202
A,CM,CP,E*,MS

Oklahoma
Hewlett-Packard Co.
P.O. Box 366
1503 W. Gore Blvd., Suite #2
LAWTON, OK 73502
Tel: (405) 248-4248
C

Hewlett-Packard Co.
P.O. Box 32008
304 N. Meridan Avenue, Suite A
OKLAHOMA CITY, OK 73107
Tel: (405) 946-9499
A*,CM,CP,E*,MS

Hewlett-Packard Co.
Suite 121
9920 E. 42nd Street
TULSA, OK 74145
Tel: (918) 665-3300
A**,CM,CS,M*

Oregon
Hewlett-Packard Co.
1500 Valley River Drive, Suite 330
EUGENE, OR 97401
Tel: (503) 683-8075
C

Hewlett-Packard Co.
9255 S. W. Pioneer Court
WILSONVILLE, OR 97070
Tel: (503) 682-8000
A,CM,CP,E*,MS

Pennsylvania
Hewlett-Packard Co.
1021 8th Avenue
King of Prussia Industrial Park
KING OF PRUSSIA, PA 19406
Tel: (215) 265-7000
Telex: 510-660-2670
A,CM,CP,E,MP

Hewlett-Packard Co.
111 Zeta Drive
PITTSBURGH, PA 15238
Tel: (412) 782-0400
A,CM,CP,E,MP

South Carolina
Hewlett-Packard Co.
P.O. Box 6442
6941-0 N. Trenholm Road
COLUMBIA, SC 29260
Tel: (803) 782-6493
CM,CS,E,MS

UNITED STATES (Cont.)

South Carolina (Cont.)
Hewlett-Packard Co.
814 Wade Hampton Blvd.
Suite 10
GREENVILLE, SC 29609
Tel: (803) 232-0917
C

Tennessee
Hewlett-Packard Co.
P.O. Box 22490
224 Peters Road
Suite 102
KNOXVILLE, TN 37922
Tel: (615) 691-2371
A*,CM,MS

Hewlett-Packard Co.
3070 Directors Row
MEMPHIS, TN 38131
Tel: (901) 346-8370
A,CM,CS,MS

Hewlett-Packard Co.
Suite 103
478 Craighead Street
NASHVILLE, TN 37204
Tel: (615) 383-9136
CM,MS**

Texas
Hewlett-Packard Co.
Suite 310W
7800 Shoalcreek Blvd.
AUSTIN, TX 78757
Tel: (512) 459-3143
CM,E

Hewlett-Packard Co.
Suite C-110
4171 North Mesa
EL PASO, TX 79902
Tel: (915) 533-3555
CM,CS,E*,MS**

Hewlett-Packard Co.
5020 Mark IV Parkway
FORT WORTH, TX 76106
Tel: (817) 625-6361
CM,C*

Hewlett-Packard Co.
P.O. Box 42816
10535 Harwin Street
HOUSTON, TX 77036
Tel: (713) 776-6400
A,CM,CP,E,MP

Hewlett-Packard Co.
3309 67th Street
Suite 24
LUBBOCK, TX 79413
Tel: (806) 799-4472
M

Hewlett-Packard Co.
P.O. Box 1270
930 E. Campbell Rd.
RICHARDSON, TX 75081
Tel: (214) 231-6101
A,CM,CP,E,MP

Hewlett-Packard Co.
205 Billy Mitchell Road
SAN ANTONIO, TX 78226
Tel: (512) 434-8241
CM,CS,E,MS

Utah
Hewlett-Packard Co.
3530 W. 2100 South Street
SALT LAKE CITY, UT 84119
Tel: (801) 974-1700
A,CM,CP,E,MS

Virginia
Hewlett-Packard Co.
P.O. Box 9669
2914 Hungary Spring Road
RICHMOND, VA 23228
Tel: (804) 285-3431
A,CM,CP,E,MS

Hewlett-Packard Co.
P.O. Box 4786
3110 Peters Creek Road, N.W.
ROANOKE, VA 24015
Tel: (703) 563-2205
CM,CS,E**

Hewlett-Packard Co.
P.O. Box 12778
5700 Thurston Avenue
Suite 111
VIRGINIA BEACH, VA 23455
Tel: (804) 460-2471
CM,CS,MS

Washington
Hewlett-Packard Co.
15815 S.E. 37th Street
BELLEVUE, WA 98006
Tel: (206) 643-4000
A,CM,CP,E,MP

Hewlett-Packard Co.
Suite A
708 North Argonne Road
SPOKANE, WA 99206
Tel: (509) 922-7000
CM,CS

West Virginia
Hewlett-Packard Co.
4604 MacCorkle Ave., S.E.
CHARLESTON, WV 25304
Tel: (304) 925-0492
A,CM,MS

Wisconsin
Hewlett-Packard Co.
150 S. Sunny Slope Road
BROOKFIELD, WI 53005
Tel: (414) 784-8800
A,CM,CS,E*,MP

URUGUAY
Pablo Ferrando S.A.C. e.I.
Avenida Italia 2877
Casilla de Correo 370
MONTEVIDEO
Tel: 403102
Telex: 901 Public Booth Para Pablo
Ferrando 919520
Cable: RADIUM Montevideo
A,CM,E,M

Guillermo Kraft del Uruguay S.A.
Avda. Libertador Brig. Gral.
Lavalleja 2083
MONTEVIDEO
Tel: 234588, 234808, 208830
Telex: 6245 ACTOUR UY
P

U.S.S.R.
Hewlett-Packard Co.
Representative Office
Pokrovsky Blvd. 4/17 KV12
MOSCOW 101000 Tel: 294-2024
Telex: 7825 HEWPACK SU

VENEZUELA
Hewlett-Packard de Venezuela C.A.
Apartado 50933
3A Transversal Los Ruices Norte
Edificio Segre 2Y3
CARACAS 1071
Tel: 239-4133, 239-4777,
239-4244
Telex: 25146 HEWPACK
Cable: HEWPACK Caracas
A,CP,E,MS,P

YUGOSLAVIA
Iskra-Commerce-Representation of
Hewlett-Packard
Sava Centar Delegacija 30
Milentija Popovica 9
11170 BEOGRAD
Tel: 638-762
Telex: 12042, 12322 YU SAV CEN

Iskra-Commerce-Representation of
Hewlett-Packard
Koprska 46
61000 LJUBLJANA
Tel: 321674, 315879
Telex:

ZAMBIA
R. J. Tilbury (Zambia) Ltd.
P.O. Box 2792
LUSAKA
Tel: 81243
A,E,M,P

ZIMBABWE
Field Technical Sales
45 Kelvin Road, North
P.B. 3458
SALISBURY
Tel:
C,E,M,P

FOR COUNTRIES AND AREAS NOT LISTED:

CANADA

Ontario
Hewlett-Packard (Canada) Ltd.
6877 Goreway Drive
MISSISAUGA, Ontario L4V 1M8
Tel: (416) 678-9430
Telex: 610-492-4246

EASTERN USA

Maryland
Hewlett-Packard Co.
4 Choke Cherry Road
Rockville, MD 20850
Tel: (301) 258-2000

MIDWESTERN USA

Illinois
Hewlett-Packard Co.
5201 Tollview Drive
ROLLING MEADOWS, IL 60008
Tel: (312) 255-9800

SOUTHERN USA

Georgia
Hewlett-Packard Co.
P.O. Box 105005
450 Interstate N. Parkway
ATLANTA, GA 30339
Tel: (404) 955-1500

WESTERN USA

California
Hewlett-Packard Co.
3939 Lankersim Blvd.
LOS ANGELES, CA 91604
Tel: (213) 877-1282

EUROPEAN AREAS NOT LISTED, CONTACT
SWITZERLAND
Hewlett-Packard S.A.
7 Rue du Bois-du-Lan
CH-1217 MEYRIN 2, Switzerland
Tel: (022) 83-81-11
Telex: 27835 hpse
Cable: HEWPACKSA Geneve

EAST EUROPEAN AREAS NOT LISTED, CONTACT

AUSTRIA
Hewlett-Packard Ges.m.b.h.
Wehlistrasse 29
P.O. Box 7
A-1205 VIENNA
Tel: (222) 35-16-210
Telex: 135823/135066

MEDITERRANEAN AND MIDDLE EAST AREAS NOT LISTED, CONTACT

GREECE
Hewlett-Packard S.A.
Mediterranean & Middle East
Operations
35, Kolokotroni Street
Platia Kefallariou
GR-Kifissia, ATHENS, Greece
Tel: 808-0359, 808-0429
Telex: 21-6588
Cable: HEWPACKSA Athens

INTERNATIONAL AREAS NOT LISTED, CONTACT

OTHER AREAS
Hewlett-Packard Co.
Intercontinental Headquarters
3495 Deer Creek Road
PALO ALTO, CA 94304
Tel: (415) 857-1501
Telex: 034-8300
Cable: HEWPACK