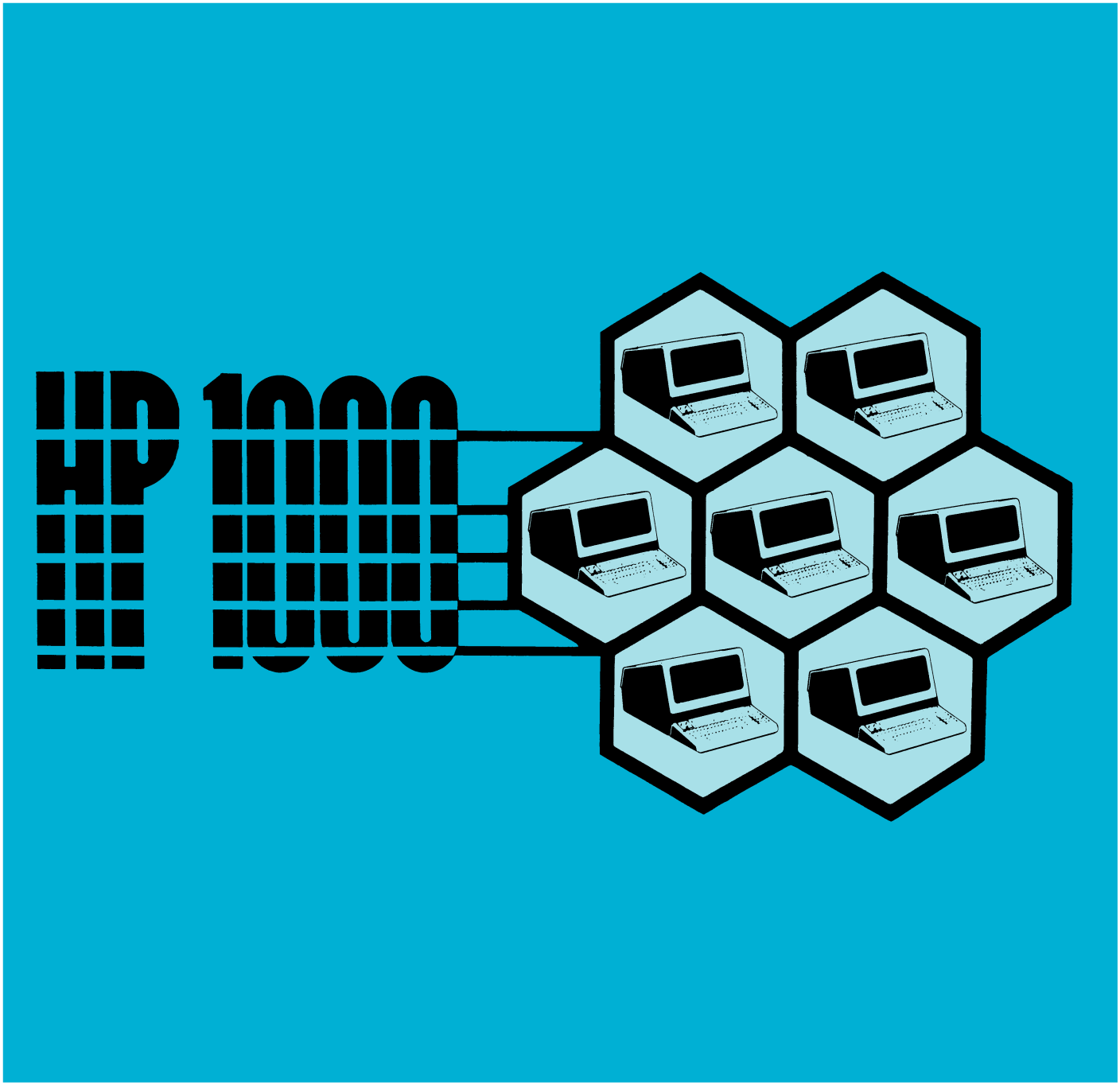
HEWLETT PACKARD

RTE-IVB TECHNICAL SPECIFICATIONS

Reference Manual



RTE-IVB
TECHNICAL SPECIFICATIONS

Reference Manual

HEWLETT-PACKARD COMPANY
Data Systems Division
11000 Wolfe Road
Cupertino, California 95014

Library Index Number
2RTE.340.92068-90013

MANUAL PART NO. 92068-90013
Printed in U.S.A. January 1980

NOTICE

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

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

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

PREFACE

This manual is intended as a tool to the advanced user of the RTE-IVB Operating System who requires a detailed explanation of the internal algorithms and data structures of RTE-IVB and many of its subsystems. It is assumed that the user has a general understanding of RTE-IVB and HP 1000 computers and has access to various manuals, specifically the RTE-IVB Programmer's Reference Manual, the RTE-IVB Terminal User's Reference Manual, the RTE-IVB System Manager's Manual, and the RTE-IVB On-Line Generator Reference Manual. RTE-IVB software is supported by Hewlett-Packard only at the user interface level documented in the manuals supplied with the product. Subroutines or data structures not documented there but which are described or mentioned within this manual or the RTE-IVB sources are mentioned solely for reference and must not be accessed directly. Their existence, calling sequences or structures are subject to change without notice to the user. This theory of operation makes no attempt to provide a line-by-line discussion of the source code. For this level of detail, the user is directed to the appropriate source code with its accompanying comments which can be purchased through any Hewlett-Packard field office.

This document was originally developed during the initial design phase of RTE-IVB development and updated as the operating system was implemented. Hewlett-Packard has made every effort to verify the authenticity of this document versus the actual algorithms; however, the user is cautioned that differences may exist. As modifications occur to the operating system due to corrections or enhancements or new subsystems are added and documented, then new revisions of this manual will be published.

TABLE OF CONTENTS

Chapter 7
EMA MICROCODE SPECIFICATIONS

Chapter 8
ON-LINE GENERATOR

Chapter 11
SWTCH

Chapter 12
RTE-IV ASSEMBLER CHANGES

Chapter 13
SESSION MONITOR ACCOUNT PROGRAM

Chapter 18
FORMT UTILITY

ILLUSTRATIONS

TABLES

Chapter 1

DSP4--RTE-IVB DISPATCHER

DISPATCHER OPERATION OVERVIEW

The Dispatcher module's main function is to control program execution by switching CPU control to the highest priority program in the scheduled list if it is ready to run. The Dispatcher serves as the return point from the operating system back to the user. If there are no programs scheduled, the Dispatcher prepares for execution of the idle loop under the user map and waits for an interrupt to occur. An interrupt is generated by an I/O device, a user program request, or an error condition signal which requires entry into the operating system for a response. The Dispatcher turns control over for program execution until the next interrupt.

In the process of returning control to a user program, the Dispatcher needs to perform a number of smaller tasks. Once it is determined that a program should be executed, the Dispatcher needs to check whether or not the program is present in memory. If the program is not already in memory then it needs to be loaded from the disc. The Dispatcher is responsible for finding an empty partition of the type and size, if one was not previously specified. If there are no free partitions it would be necessary to swap out a dormant program or a lower priority executing program to make the partition available for the load. Then finally the user map and memory protect fence are set up before executing the program.

Other functions of the Dispatcher are to set up the partition list headers at initialization of the operating system and to coordinate the cleaning up of a program's resources and system available memory when a program is aborted.

GENERAL OVERVIEW OF TIMESLICING OPERATION

All programs competing for the central processor access it in an orderly manner, under the direction of RTE-IVB. The system places programs into the scheduled list in order of their priority. When a program completes, terminates or is suspended, the RTE-IVB Dispatcher searches the scheduled list for the next program of highest priority, and transfers control to it.

The scheduled list (see Figure 1-1) is divided into logical areas, each corresponding to a particular type of dispatching and priority level. Scheduling within each priority can be performed in a linear or a circular fashion.

The default priority range for linear scheduling is from 1 to 49 (see QU command). Programs of this type are given processor control until the program is either completed, terminated or suspended to await the availability of a required resource.

Circular scheduling is performed on all program priority levels lower (higher number) than the timeslice limit (see QU command). Programs of this type are given processor control for an interval (Time Quantum) of maximum duration (or until completed, terminated or suspended). Control is then passed to the next program of the same priority (queue), continuing in a round-robin fashion until all programs of the specified priority have completed, terminated of suspended. The RTE-IVB Dispatcher then searches the scheduled list for the next highest priority level that has programs prepared to execute.

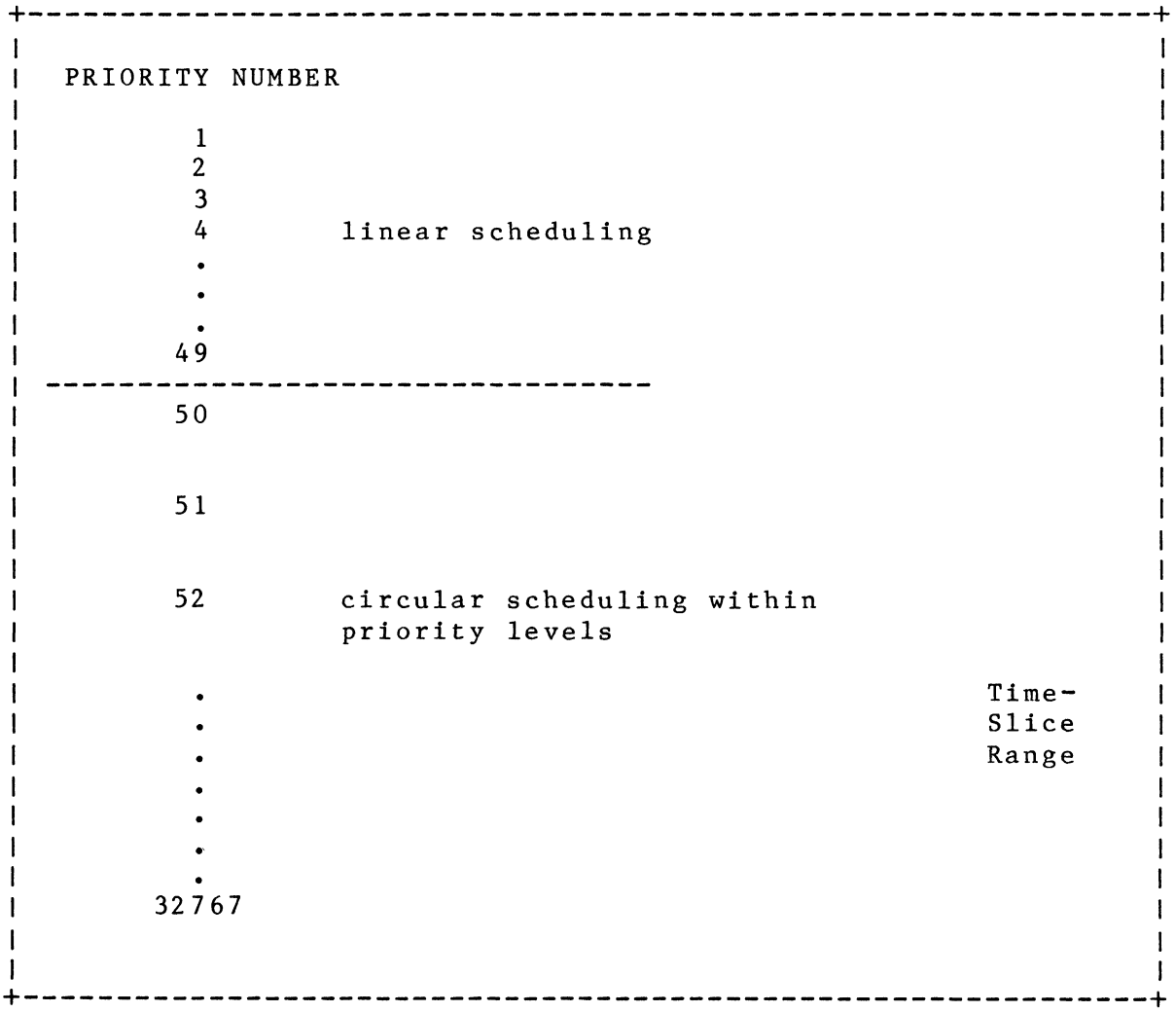


Figure 1-1. Scheduled List

Within the scheduled list, each priority level (in the timeslice range) may be thought of as a circular queue. The program at the head of the queue represents the next program of that priority to be executed. All programs of a higher priority (lower number), that are in the scheduled list will have a chance to execute before a lower priority program is entered. When a timeslice program is entered, a maximum execution slice is set up within the operating system. This program is then allowed to execute until one of the following occurs:

1) The program leaves the scheduled list (such as I/O suspend, memory suspend or dormant).

2) A higher priority program is ready to execute.

3) The program exceeds its timeslice quantum.

If a program leaves the scheduled list, its execution slice is assumed complete. Therefore, when the program is again ready to execute, it is placed at the end of the queue within its priority, in the scheduled list. Also, when the program is again picked to execute, the original timeslice quantum is set up.

If a higher priority program causes your program to stop executing (but you're still scheduled), the remaining execution slice value is saved in your program's ID segment. Then, when your program is again ready to be entered, the remaining count is set up as the timeslice quantum to be used.

When a program exceeds its execution slice, it is moved behind (in the scheduled list) all other programs of the same priority. The program remains scheduled but execution now passes to the new head of the scheduled list (also head of that priority's queue).

The System Manager can control the scheduled list (Timeslicing) in the following ways:

a) Modify the system (multiplier) timeslice quantum (QUantum command).

b) Modify the priority level at which programs are timesliced (QUantum command).

c) Modify a specific program's timeslice level (PRiority command).

EXTERNAL COMMUNICATION

The Dispatcher communicates with the rest of the operating system through the system tables, base page communication area, and subroutine calls to/from the rest of the system. There are no direct paths of communication between the Dispatcher and any user program because there are no functions in the Dispatcher which would be useful to a calling program.

ID SEGMENT (Figure 1-2)

An ID segment is processed by the Dispatcher. Each ID segment is linked into the scheduled list by word 0. The priority, type, and status are checked to determine whether or not the program may execute. The memory protect fence table index (MPFI) and number of pages are set into word 21 by the system generator or by the relocating loader. The number of pages (which does not include a base page) is the number of pages actually occupied by a program and its largest segment (if any) or the override size specified by the user. The size of an EMA program includes the mapping segment size (still excludes the base page).

The program address bounds in words 22-25 are used by the Dispatcher for loading the program into memory. The program's disc address is in word 26. If the program is swapped out, the Dispatcher keeps the address of the swap tracks in word 27.

I/O Segment word 28 is used by the Dispatcher to determine whether or not a program is using an extended memory area (EMA). If this word is zero, no EMA is used. If the word is non-zero, the ID Extension is checked to see if the default EMA size was chosen. If it was not defaulted then the program size (from word 21) minus the MSEG size is added to the EMA size plus base page to determine the partition size required. If the EMA size was defaulted, the largest Mother partition size, $MCHN, is used as the partition size required.

Word 30 contains the timeslice quantum value (addressed by $LICE). This word is setup (Negative 10's of MS) if the program is in the timeslice priority window. Word 32 indicates if the program is running under session control. If it is, the Dispatcher defines the location of the CPU usage counter ($DCPU) in the users SCB.

MEMORY PROTECT FENCE TABLE (Figure 1-3)

The Memory Protect Fence table ($MPFT) a table of addresses used by the Dispatcher to set the correct value for the memory protect fence. This address is stored in the base page word FENCE (1775). Bits 7-9 of word 21 in the ID Segment contain the index into this table.

MEMORY ALLOCATION TABLE (Figure 1-4)

Each partition defined by the user at generation time will have an entry in the Memory Allocation Table. The table starts at the entry point $MATA and extends upwards toward high memory. The word $MNP contains the total number of partitions (set by the generator). Each partition entry (see Figure 1-5) is 7 words long. The MAT Link Word contains -1 if the partition is undefined.

There are three different types of partitions:

1. Real Time Partitions headed by $RTFR at system start-up.
2. Background Partitions headed by $BGFR at system start-up.
3. Mother Partitions headed by $CFR at system start-up.

These three lists are set up by the Generator in order of increasing size. The primary purpose for having RT partitions and BG partitions is to keep the two classes of programs, RT and BG, from contending with each other for memory. There are no differences in the two classes of partitions. Mother Partitions are primarily for EMA programs.

Mother partitions are defined during generation by a "YES" answer to the prompt "SUBPARTITIONS?" when a partition is larger than the maximum addressable space. Although Mother partitions are in a separate list, subpartitions may be linked into either a Mother partition (see Figure 1-6) or linked into a BG or RT (either free or allocated) list. When the subpartitions are part of a Mother partition, the Mother partition's MAT entry word 6 (Subpartition Link Word - SLW) will point to the Link Word (word 0) of the first subpartition whose SLW will point to the link word of the second subpartition and so on. The SLW of the last subpartition will point to the Link Word of the Mother. If no subpartitions were actually defined but the prompt for subpartitions was answered "YES" then the SLW of the Mother partition will point to the Link Word of the same MAT entry.

When a Mother partition is in use, the entire chain of subpartitions is in use and this is indicated by the C bit being set. In this case, all partition status information is kept in the Mother partition MAT entry; i.e.; priority, ID segment address, and Read Completion flag. In addition, the chained partitions are treated as a single entity while the C bit is set. Individual subpartitions are not swappable in this case - the whole Mother partition may be swapped if needed.

The Dispatcher checks for empty partition lists at start-up (see INITIALIZATION Section). If there are no Real Time partitions then the header of the RT partitions list will point to the Background partitions. If there are no BG partitions then the RT partitions are used in the BG list. If there are no Mother partitions BG partitions are used unless there are no BG partitions, in which case the RT partitions are used.

The sizes of the largest partition of each type are kept in three words:

1. $MRTP - size of the largest non-reserved RT partition.
2. $MBGP - size of the largest non-reserved BG partition.
3. &MCHN - size of the largest non-reserved Mother partition.

SYSTEM BASE PAGE COMMUNICATION

*	XIDEX	1645	ID EXT addr of current prog or 0
*	XMATA	1646	MAT address of current program or 0
*	XI	1647	Pointer to current program's X-Y save area
*	SKEDD	1711	Head of scheduled list
*	XEQT	1717-1733	Current program's ID Segment pointers
	SWAP	1736	Swap delay in bits 8-15
	BPA2	1743	Last word user base page (add 1 for BP fence)
	LBORG	1745	Logical address of Resident Library
*	RTDRA	1750	Dynamically set address
*	AVMEM	1751	bounds for partition
*	BGDRA	1754	resident programs
*	TATLG	1755	Dispatcher locks disc tracks for FMGR
	TATSD	1756	#Tracks system disc
	SECT2	1757	#Sectors LU2
	SECT3	1760	#Sectors LU3
*	FENCE	1775	Memory Protect Fence value for user
*	BGLWA	1777	Dynamically set bound

* Set or changed by the Dispatcher.

DISPATCHER ENTRY POINTS

$ALDM Subroutine, used by SCHEDULER to unlink a partition's MAT entry from the allocated list into the dormant list.

The Calling Sequence:

```
<A-reg> has ID Segment Address
JSB $ALDM
```

$BRED Subroutine, used by SCHEDULER to read in program segments.

The Calling Sequence:

```
<B-reg> has ID Segment Address
JSB $BRED
```

$DCPU ENTRY POINT, defines the current sessions CPU usage counter or a dummy system location if the XEQT program is not a session program. Used by RTIME.

$DMAL Subroutine, used by SCHEDULER to unlink a partition's MAT entry from the dormant list into the allocated list.

The Calling Sequence:

```
<A-reg> has ID Segment Address
JSB $DMAL
```

$HIGH Page number of last page of last disc resident program partition.

$LICE Entry point, defines the address of word 30 (Timeslice Quantam) of the XEQT programs ID segment or a dummy system location if the XEQT program is not being timesliced. Used by RTIME.

$LOW Page number of first disc resident program partition.

$MAXP Subroutine, called by the routine $PERR when a partition is undefined or by the SCHEDULER when a partition's "reserved" status is changed by the RS command. This subroutine searches the MAT to determine the values of $MBGP, $MRTP and $MCHN by scanning each MAT entry for the largest partition of the specified type and update the appropriate word. $MAXP may have to update more than one (and maybe all) if the size words if any of the partition lists were initially empty. This is necessary because the DISPATCHER would have changed the list pointers of the empty list(s) to point to a non-empty list.

The Calling Sequence:

```
JSB $MAXP
<return>
```

$PRCN Subroutine, called by DISPATCHER and SCHEDULER to relink a partition in the allocated list by priority.

The Calling Sequence:

```
<A-reg> has ID Segment Address
<B-reg> has new priority
JSB $PRCN
```

$RENT Entry point, JMP there from the DISPATCHER and EXEC for setting up the Resident Library address in the memory protect fence.

$RVAL Entry point, defines the Timeslice reset value for the XEQT program. Used by RTIME if the XEQT program uses a full timeslice and no one else of the same priority is scheduled. In this case, the original timeslice value ($RVAL) is restored to ID 30 and the program is allowed to continue.

$SMAP Subroutine, called by DISPATCHER and RTIOC to set up the user map. Unused pages are write-protected.

The Calling Sequence:

```
<A-reg> has length of program to map in pages
<B-reg> has MAT entry address
JSB $SMAP
```

$UNPE Subroutine, called by $PERR to unlink partition MATA entries and undefine the partition where a parity error is detected in it.

The Calling Sequence:

<B-reg> has MATA address of partition
JSB $UNPE

$XCQ Entry point, actual entry for $XEQ where the main dispatching algorithm is performed.

$XDM Subroutine, called by non-privileged drives to set up user map.

The Calling Sequence:

<A-reg> has ID Segment Addr
JSB $XDMP (Entry via Table Area II entry point)

$ZZZZ Entry point, used by DISPATCHER and SCHEDULER as the head of the program abort list. This entry point is used as a subroutine during the start-up sequence.

DISPATCHER'S EXTERNAL TABLES AND POINTERS

All of these entry points are located in Table Area II unless otherwise specified.

$BGFR Pointer, BG free list initialized by the Generator.

$CFR Pointer, free list header of Mother partitions, initialized by Generator.

$CMST Value, start page number of COMMON area, set up by the Generator

$EMRP Address, last word of memory resident program area, set by the Generator.

$ENDS Value, number of pages occupied by the system, its base page. Both table areas, System Driver Area, driver partition area, Common, and the first 2K of System Available Memory. This word is initialized by the Generator.

$IDEX Pointer, ID Extension list.

TECHNICAL ASPECTS OF OPERATION

This portion of the Technical Specifications is a detailed description of major portions of the DISPATCHER code as outlined in the general overview (see the Dispatcher Operation Overview Section in this Chapter). It is assumed that the reader has a good general understanding of the RTE operating system.

INITIALIZATION

$ZZZZ serves as the entry point of the initialization subroutine which is executed only once during system initialization. The routine first sets up $LOW and $HIGH, the first and last words respectively of the disc resident program partition area. Next it sets up the starting address of SSGA in $SGAF for the EXEC. Then $ZZZZ loads the user map with the memory resident map registers which were built by the Generator in $MRMP. Whenever a memory resident program is dispatched, $MRMP is used to set up the user map.

Next, the $ZZZZ routine will set the base page fence so that all addresses between 16XX and 1777 are not mapped. The fence value minus one is contained in base page location BP2 (1743) which is the address of the last available user link.

A number of system dependent address and sizes are calculated and saved so that they may be reused by the DISPATCHER without being recalculated each time. Some of these "constants" include the address of the memory resident library, the number of pages in the memory resident library the starting register number and the number of registers to be used for mapping chunks of EMA to be swapped.

The partition free lists are also checked by the Dispatcher during the initialization of the system. If the BG free list header ($BGFR) is zero, meaning that it is empty, then the RT free list pointer ($RTFR) is stored into $BGFR. If there were no RT partitions then the RT free list pointer will be set equal $BGFR. The maximum unreserved partition size words, $MRTP and $MBGP, will be updated accordingly. If the Mother partition list is empty the $BGFR pointer will be used, assuming that the BG list is not empty since we've already done the check above. $MCHN will also be updated. However, if it turns out that the BG list is empty because the RT list was empty, then with all three lists empty the SCHEDULER will report an error on any scheduling attempts. This code is in subroutine LSTIN which is also called by $MAXP.

The last thing done during initialization is the scheduling of the File Manager program, FMGR. The Dispatcher first locks all of the disc tracks by saving the number of tracks word (TATLG) in the FMGR's first parameter word and then replacing TATLG with -1.

The DISPATCHER'S initialization code is overlayed by the disc I/O triplets which are built for doing program loads and swaps.

DISPATCHING

$XCQ (user map entry point, $XEQ) is the entry point into the DISPATCHER code which performs the allocation of execution time to programs. The primary objective of $XCQ is to execute the highest priority program in the scheduled list, SKEDD, if possible.

First the DISPATCHER checks to see if there are any programs which were aborted (see CLEANUP Section). If no programs were aborted, then the DISPATCHER checks the scheduled list. If there are no programs in the scheduled list, or for some reason the programs in the list can't be executed at this time, the "idle loop" is executed instead of a user program. The base page point-of-suspension word, XSUSP, is set to the idle loop code address and the base page register save area pointers (XA, XB, XEO and XI) are set up to use a two-word dummy save area. The Timeslice and CPU usage pointers are set up to dummy locations. The idle loop code ($IDLE) and dummy save area are located in Table Area I so the user map may be used. The base page word XEQT is cleared to indicate that no program is executing, the memory protect fence register is set to zero and stored in FENCE on base page. Then it exits the system via $IRT to enable memory protect, the interrupt system and the user map.

If there are programs to be scheduled in the SKEDD list, the DISPATCHER makes the decision to execute a program based on the program's priority, status, type, and address space needs.

If the currently executing program is of higher priority than the program in the scheduled list, execution of the current program is resumed. If the program in the scheduled list is higher, or, if the program in the scheduled list is of an equal priority and the current program has used a full timeslice, a check is made to see if the program (from the scheduled list) is in memory and if it can be executed.

If it can be executed, the user map registers are set up with the program's physical page numbers. The program's logical address bounds are set up in RTDRA, AVMEM, BKDRA and BKLWA. The ID segment pointers are set up at XEQT. The X \ Y registers save area address is also set up at XI. If the program's priority is higher than the timeslice limit, skip the timeslice setup. If ID word 30 is less than zero, use it as the timeslice value. Otherwise, the timeslice value is calculated by using the program's priority in the following equation:

Program Slice Value = Sys Slice * Z + SYS Slice

where SYS Slice = 1500 ns default. This value (1.5 sec) may be increased or decreased via the "QU" command. See Appendix J for an explanation of the default value.

z = bits 8-11 (isolated and shifted right 8 positions) of the program's priority.

Define $LICE to be equal to the address of ID 30. If ID 32 is a positive value, define $DCPU to address the CPU usage location of the session control block.

If the program from the scheduled list is not in memory, see if it can be loaded.

Now that the program is ready to execute, the address of where to begin or resume execution is determined. If the point of suspension address is zero, control is given to the program at the primary entry point. If the point of suspension is non-zero, control is returned to that address. The memory protect fence is set up according to the Memory Protect Fence Table index in the ID Segment. Then control is turned over to the program by exiting through $IRT which enables memory protect, enables the interrupt system, and enables the user map.

The general dispatching procedure described above is slightly different for different types of programs.

MEMORY RESIDENT PROGRAMS

If the scheduled program is a Memory Resident Program, the memory protect fence may be set differently since these programs are the only type which may reference the Resident Library. The Dispatcher will clear the Write Protect bit from the Resident Library pages in the Memory Resident Map by clearing the sign bit from those words when the User Map is being set up. All the other registers would remain the same as in the Memory Resident Map Table ($MRMP) which is never changed. Then the memory protect fence is set at LBORG if the reentrant bit is set in the ID Segment. This code has an entry point of $RENT for access by the EXEC.

DISC RESIDENT PROGRAMS

If the scheduled program is a disc resident program, it needs to be loaded from the disc into a partition which was allocated for it. If a partition was pre-assigned at relocation time, that MAT entry will be checked to see if it is available or if its resident program is swappable. If either case is true, the MAT entry will be set up for the new program and will be linked into the allocated list of the pertinent type. The MAT entry will not be modified or relinked if its current resident is the program which RTE is trying to dispatch.

If a partition was not specified at load time, the MAT entry for the partition in which the program was last resident will be checked to see if the program is still resident. The MAT entry is first checked to see if the partition still exists. The partition may be undefined if a parity error was detected in one of its pages since the program was last resident there. If the partition still exists and the program was the last occupant in there, the partition is set up to be used and the program is dispatched there after the user map is built (see Chapter 2).

If the program is no longer resident in the partition or the partition became undefined, a default partition list will be scanned for a free partition. The default partition types are:

a. RT program (Type 2) - RT partition ($RTFR)
b. Large BG program (Type 4) - BG partition ($BGFR)
c. BG program (Type 3) - BG partition ($BGFR)
d. EMA program - Mother partition ($CFR)

For RT, BG and large BG programs, the appropriate free list will be scanned for the smallest free partition in which the program can fit. EMA programs which have a specific EMA size declared will get the smallest free Mother partition. But, EMA programs which let the EMA size default will take on the maximum Mother partition size ($MCHN). This size minus the program code size is then put into word 29 of the ID segment to prevent the problem where a swapped out program may be reloaded into a smaller partition if $MCHN was changed because of a parity error on the partition or it became reserved.

If a free partition is not available, the appropriate dormant list (RTDM, BGDM or CDM) will be scanned for a partition with a swappable program. The dormant lists are a subset of the allocated lists (see Figure 1-7). The last entry in the dormant list points to the allocated list so if the dormant list is empty, the dormant list just points to the allocated list.

Upon finding a suitable partition, the occupant will be swapped out. The MAT entry will be reset and relinked, and prepared for loading of the disc resident program. If no dormant partition qualifies for the swap, the allocated list (RTPR, BGPR or CPR) will be scanned for the lowest priority program which can be swapped. The same procedure described for the dormant swap will be followed.

DISC RESIDENT MAP SET UP

Once a partition is allocated for a program, the user map is set up for the program. If the program is being scheduled initially (program's first dispatch) the User Map registers must be loaded by the DISPATCHER and a copy of it saved in the user's protected portion of base page (see Figure 1-8). If the program is being re-dispatched, to continue after being suspended or after being "bumped" by a higher priority program, the User Map registers are set up by copying them from the saved copy in the protected portion of the user's base page.

A program's first dispatch is identified by the fact that the point of suspension word (XSUSP) is 0 in the program's ID Segment. The base page register (logical page 0) is loaded with the page value in word 3 of the MAT entry. The next registers are then loaded sequentially with numbers starting at one end incremented by one in each successive register. The number of registers set in this manner depends on the program type or whether or not the program uses COMMON.

If the program type is 2 or 3, the number of registers set sequentially is determined by one less than the value of $SDA added to $SDT2. Actually the number of registers mapped is one less than $SDA. The next registers mapped (number of registers is determined by $SDT2) have the write-protect bit set. This maps into the User Map: Table Area I, the Driver Partition Area, COMMON (including SSGA), write-protected System Driver Area and write-protected Table Area II.

If the program is not type 3, the Memory Protect Fence Table Index (in the ID Segment) is checked to see if the program uses any COMMON or SSGA. If COMMON or SSGA is used, the number of registers set up following the base page register is determined by one less than the value in $CMST. If COMMON or SSGA is needed, the value $SDA -1 is the number of registers to map in Table Area 1, the Driver Partition Area, and COMMON. The user program is mapped in the registers following these registers pointing to the system areas.

The next registers are loaded with the next physical page numbers sequentially following the page used for the user base page. These are loaded into the map registers until the number of registers specified in word 21 of the ID Segment have been set up. The non-standard MSEG bit (bit 15 of word 0 in the ID Segment extension) is set if the program is an EMA program to force the EMA subroutines to remap.

The remaining registers in the user map will be read/write protected to insure that a program cannot access memory outside of its partition. This mapping is done in $SMAP which is the only code which loads the user map to describe a specific program. It is also called by RTIOC.

A copy of the user map is saved in the last 32 words of the user's physical base page (see Figure 1-8). The system's map register for the driver partition ($DVPT) is used to map in the user's base page. This portion of the base page is not used during the program's execution since the system communication area is always mapped in on the top portion of the user base page.

With all of the above done the program is ready for dispatch.

SWAPPING

A program is swapped out of memory to make a partition available for another program to run. The first programs chosen to be swapped are the ones in the dormant list. These programs are the ones which have terminated with either the save resources or serially reusable option or are operator suspended and still in memory. Otherwise, programs with the lowest priority will be checked for swappability.

Programs are not swappable if any of the following are true:

1. The resident memory lock is in effect.

2. If the resident is scheduled (State 1) and has a higher priority than the program to be scheduled.

3. The resident is dormant but has a higher priority and is in the time list to be scheduled in less than the minimum permissible amount of time specified in swap (swap delay).

4. The resident is I/O suspended with the buffer in the program area, i.e., unbuffered I/O.

5. It has the same priority as the program to be scheduled and it has not used a full timeslice.

When a swap is required, the necessary number of tracks needed to swap the program out are computed and a request is made to $DREQ for the contiguous disc tracks. The number of tracks is computed by rounding up the number of sectors (to next even sector) needed for the base page and rounding up the number of sectors (to the next even sector) needed for the Main Code. The number of sectors is then converted to tracks and it is then rounded up to the end of a full track. If tracks are not currently available, swapping cannot take place and the DISPATCHER proceeds to check the next program in the scheduled list (if any). If tracks are available, the necessary $XSIO parameters are computed.

SETUP is the subroutine which creates the parameters for the $XSIO disc calls. SETUP guarantees that all calls to read or write disc tracks are broken down into groups of smaller I/O requests. Each one of these smaller groups of 3 words each (triplets) define an I/O request which will not cross a track boundary. The triplets have (1) starting memory address of the piece of data, (2) the number of words to transfer, and (3) the starting track and sector address. These triplets are built in memory overlaying the DISPATCHER'S initialization code (code following $ZZZZ). There may be up to seven triplets for a $XSIO call (enough for a 32K transfer with 6K words per track). The triplets are terminated by a zero.

There are five separate $XSIO calls, one for each type load/swap I/O so that each call can be started independently and overlap in time. Disc accesses for each type of partition can be completed at different times depending on their sizes. For each of these calls, there are triplets tables. The following table shows the names used by DISPATCHER.

Type of $XSIO CALL	Code Busy Flag	Triplets Area Terminator	Triplets Terminator Address	Triplets Area Pointer
RT	RTSWP	RTRIP	RTRPA	RTRP
BG	BGSWP	BTRIP	BTRPA	BTRP
Mother	CHSWP	CTRIP	CTRPA	CTRP
EMA	CHSW2	CTRIP	CTRPA	CTRP2
Segment	SGSWP	STRIP	STRPA	STRP

When a swap out is completed, the disc logical unit track address and number of tracks are stored in the ID Segment (word 27, SMAN).

When a program is loaded (or swapped back) into memory, $XSIO is called using parameters computed by SETUP. ID Segment value DMAN (word 26) is used for the disc address if the program is not swapped out; and SMAN if the program is swapped out. The program's dispatching status in word 5 of the MAT entry is cleared to indicate that a program read is in progress. The "LOAD IN-PROGRESS" bit is set in the programs ID segment. This bit (bit 8 word 15) indicates that the program may not be dispatched. When the read is complete, any swap tracks are released via $DREL and the program is scheduled via $LIST. A check is made to see if the read was correct. If not, the program is aborted via $ABRT which sets it dormant, releases its tracks, and removes it from the time list. If the read was correct, the "LOAD IN-PROGRESS" bit is cleared. The program is ready to execute.

When an EMA program needs to be swapped, the swap out to the disc is done in two parts. The program's code up to the page where the mapping segment starts (MSEG) and the program's base page are swapped out first using the CHSWP $XSIO call. The number of swap tracks needed is computed by adding the number of integral tracks needed for the program code and base page to the number of integral tracks needed for the EMA area. The program is swapped just like any other disc resident program. Note that in the ID Segment word 27, the number of tracks refer to just those used for the program code.

The EMA area is swapped out next, beginning at the next even sector boundary following the program code's swap tracks. EMA is swapped out in large chunks equal in size to the maximum logical address space in the user map (up to a maximum of 28K words). The User Map registers from $CMST to the end of the map, inclusive, define the number of pages in each chunk. The chunk is mapped in the User Map, the triplets are built and then the chunk is swapped out using the CHSW2 $XSIO call. When the transfer is completed, the next chunk is mapped and swapped. This process repeats until all of the EMA is swapped. A similar process takes place when swapping into memory.

The computation for the number of swap tracks needed must allow an extra sector for each chunk. The number of tracks for the EMA area is saved in word 2 of the program's ID Segment Extension. The number of tracks is needed so that the correct number of tracks can be released when the program terminates or gets aborted.

When the decision to swap out a resident program has been made, that program's current map information (32 user map registers) must be saved. The save area is from words 2 to 34 of the program area. That is, if the program was loaded at 16000B (octal) then locations 16000B and 16001B are used to save the X and Y registers and 16002B to 16041B are used to save the user map registers. The subroutine SWOUT is called to perform this save. SWOUT does this by mapping the first two pages of the partition into the driver map area. SWOUT then moves the register information (words 1740B to 1777B) of the mapped base page (now mapped into the driver partition) into the program area.

Now, since the map information is saved, RTIOC can build a new map for the swap out of the program. This insures that no matter what the resident program did to his own map the whole program will be swapped to disc as opposed to what he has mapped-in being swapped to disc.

Sometime later the dispatcher will decide to bring the swapped out program back into memory. RTIOC builds a map identical to the initial load map for the swap back into memory. This map is not used to execute the program. Rather, after the program is brought into memory a call is made to SWPIN. SWPIN rebuilds the user map and stores this map into words 1740B (octal) to 1777B of the user's base page (i.e., the first page of the partition). SWPIN does this by mapping in the first two pages of the partition into the driver partition and looking at the map information of the old partition compared to the new partition. If the first register saved on swap-out is the same page number as the first page of the current partition (i.e., we swapped the program back into the same partition), then the map information does not need modification. In this case the old map information is moved to words 1740B to 1777B of the first page of the partition. $SMAP will use this to set up the user map registers. If we are bringing the program into a different partition, then the map registers must be modified. The algorithm used to do this is:

For all 32 user map registers,

If:

```
        $LOW  =<   OP#   =<  $HIGH

        then       NP#   =   OP#-OSP# + NSP#

        else       NP#   =   OP#
```

Where:

$LOW	=	Start page number of first program partition in the system.
$HIGH	=	Last page number of last partition in the system.
OP#	=	Old page number of the current register we are working on.
NP#	=	New page number to place into the map register.
OSP#	=	Old partition's starting page number.
NSP#	=	New partition's starting page number.

This algorithm insures that the map set up when the program executes reflects the program's state when the program was last swapped out. If an EMA program is swapped into a different partition, the start page of the EMA word in the ID extension is updated to the new physical page number.

MOTHER PARTITION USAGE

If a program (any type) is assigned to a Mother partition or an EMA program defaults to any Mother partition, there is more handling involved than is the case with RT or BG partitions. If a Mother partition is used when it is in the free list ($CFR), each subpartition must be checked. If a subpartition is either free or is occupied by a swappable program the C bit is set in word 4 of the MAT entry to prevent the subpartition from getting used while the Dispatcher continues to check each subpartition. If all of the subpartitions are either free or swappable, a second pass is made on all of the subpartitions to perform the necessary swaps. The subpartitions are unlinked from any lists they might be in. When all of the subpartitions are free, the Mother partition is unlinked from the free list ($CFR) and linked into the allocated list. The program can then be loaded into the Mother partition.

If any one of the subpartitions has a memory-locked program or a program which is doing I/O in its own program space the subpartition can't be made available by swapping. All of the C bits must be cleared from each one of the previously scanned subpartitions and the dispatch is terminated. The next program in the scheduled list is examined.

When a program terminates and it was using a Mother partition, the Mother partition is relinked in the free list. All of the subparetitions are linked into the free list of the appropriate type (BG or RT).

When the C-bits are set on the subpartitions (set in chained mode), program which are assigned to these subpartition will have to wait if the DISPAT is still in the process of swapping out any subpartitions. If a program is already in the Mother Partition, the normal swappability checks apply.

In the case where a program of lower priority was in the process of loading a Mother Partition and a scheduled program is assigned to a subpartition the loading process is aborted. Then the subpartitions are released from chained mode and relinked into the proper free list. A special check is made (at SMABT) when a Mother Partition load needs to be aborted to free up a specific subpartition. If the partition type is BG and the BGSWP call is busy, the abort is not performed. If the type is RT and the RTSWP is busy, the abort also does not take place. This check prevents a deadlock which could keep the interrupt system off and the busy RT or BG call would not be able to complete. When it is necessary to clear out all the subpartitions for a Mother Partition the CHSWP call is used so that regular RT/BG partitions may continue to be used for other programs.

When a RT or BG program is scheduled and it is not assigned to a partition, a search is made for a partition of the same type which is large enough. If none can be found in the free list, none in the dormant list, and none can be found in the allocated list or it contains non-swappable programs, then the dormant Mother partition list will be searched for one which has a subpartition of the correct type and size. If a suitable subpartition can be found, the dormant program in the Mother partition will be swapped out or overlayed in the case of a serial reusable termination.

CLEAN UP

Whenever a program is terminated, either by an EXEC call 6 or aborted by the system because of an error, the program is put into the dormant state and the list processor adds the program's ID Segment into a list headed by $ZZZZ. The linking is through word 8 of the ID Segment (the point of suspension save area) since the program will begin execution at the primary entry point if it is rescheduled. Everytime the system goes to $XEQ, $ZZZZ is checked. If it is non-zero the DISPATCHER performs five major clean up tasks.

First, if the program is disc resident, any swap tracks it may have are released. This may happen if a program is aborted while it is swapped out. When tracks are released by $DREL in the EXEC module it will also call the list processor at $LIST to reschedule any programs waiting for disc tracks. If the program was an EMA program, it will be necessary to call $DREL twice; once for the program swap tracks, and once for the EMA swap tracks.

Next, $ABRE in the $EXECUTIVE is called to return any reentrant memory the program may have. This may happen if a program terminates or is aborted while in a reentrant subroutine. If $ABRE returns any memory programs waiting for memory will be rescheduled.

Next, the DISPATCHER calls $WATR in the SCHEDULER to reschedule any programs which were waiting to schedule (EXEC 23,24) this program.

Next, the DISPATCHER calls $TRRN which calls $ULLU to unlock any LUs which may have been locked by the program. $TRRN also unlocks any locally locked RNs and deallocates any locally allocated RNs the program may have. Each of these processes may call $SCD3 to reschedule any programs waiting for these resources.

Next, if the Equipment Lock Table is not empty, the DISPATCHER calls $EQCL (in RTI04) to release any EQT locks owned by the programs, if required.

Next, if the program is a disc resident program and it is still resident in the partition, the partition is linked into the free list.

Next, if the program did not terminate with the save resources option, the sequence counter (high 4 bits of ID word 31) is incremented by one. This invalidates any FMP open flags defined for the terminating program.

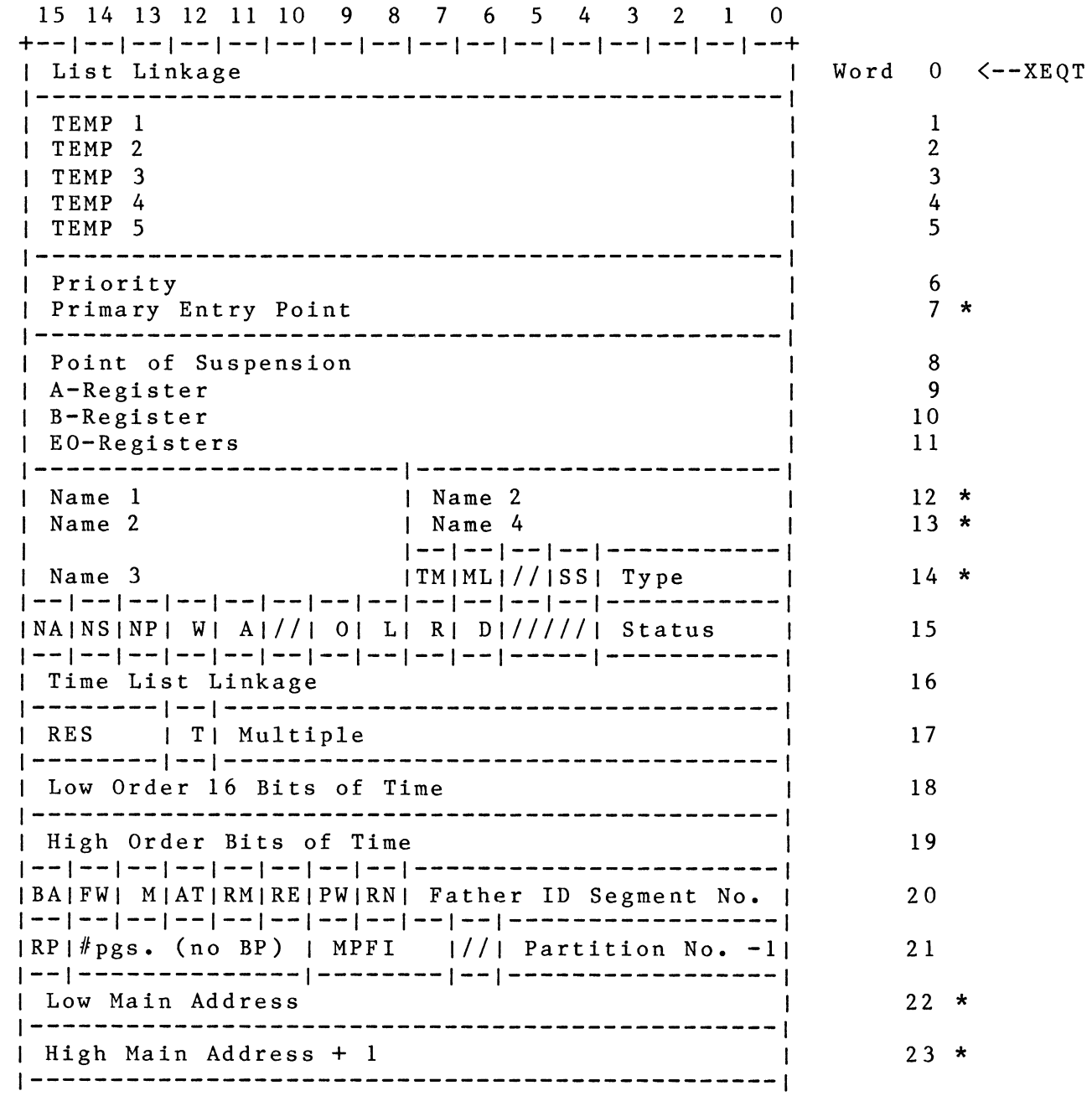
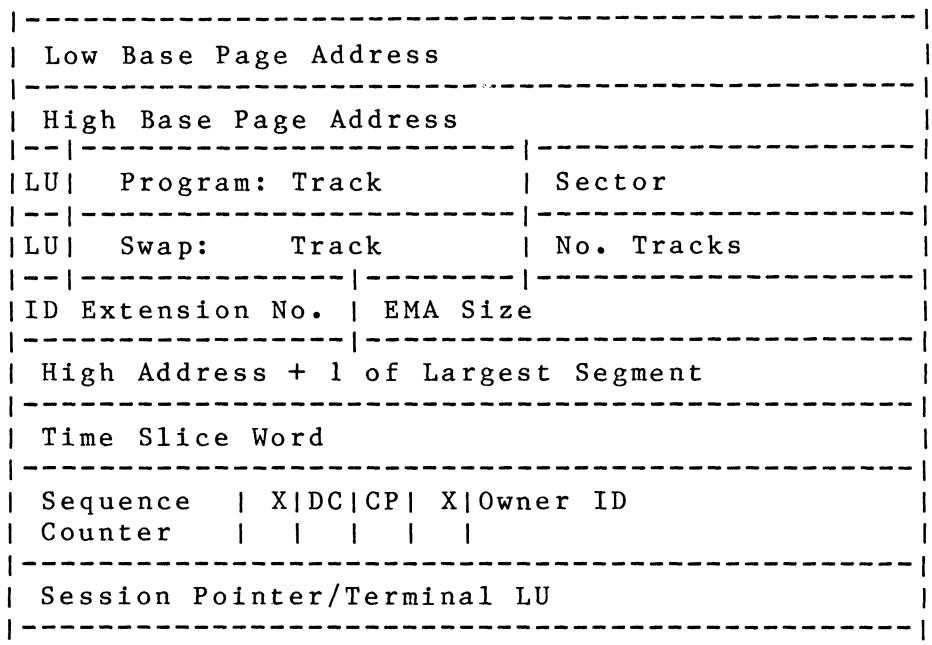


Figure 1-2. ID Segment Format (See Appendix D for a detailed description)



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 (1-6)

NA = no abort (instead, pass abort errors to program)

NP = no parameters allowed on reschedule

NS = no suspend (instead pass I/O suspension reason to program)

W = wait bit (waiting for program whose ID segment address is in word 2)

A = abort on next list entry for this program

O = operator suspend on next schedule attempt

R = resource save (save resources when setting dormant)

D = dormant bit (set dormant on next schedule attempt)

L = load from disc "IN-PROCESS"

Figure 1-2. ID Segment Format (continued)

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 = remaining number of 10's of MS for this slice.

Sequence Counter = incremented each time a program terminates. Used in FMP open Flags.

DC = program may not be copied.

CP = program is a copy of another program.

Owner ID = Session number of Session which created the ID segment.

Session Pointer/Terminal LU :

If positive = pointer to Session control block.

If negative = terminal LU.

Figure 1-2. ID Segment Format (continued)

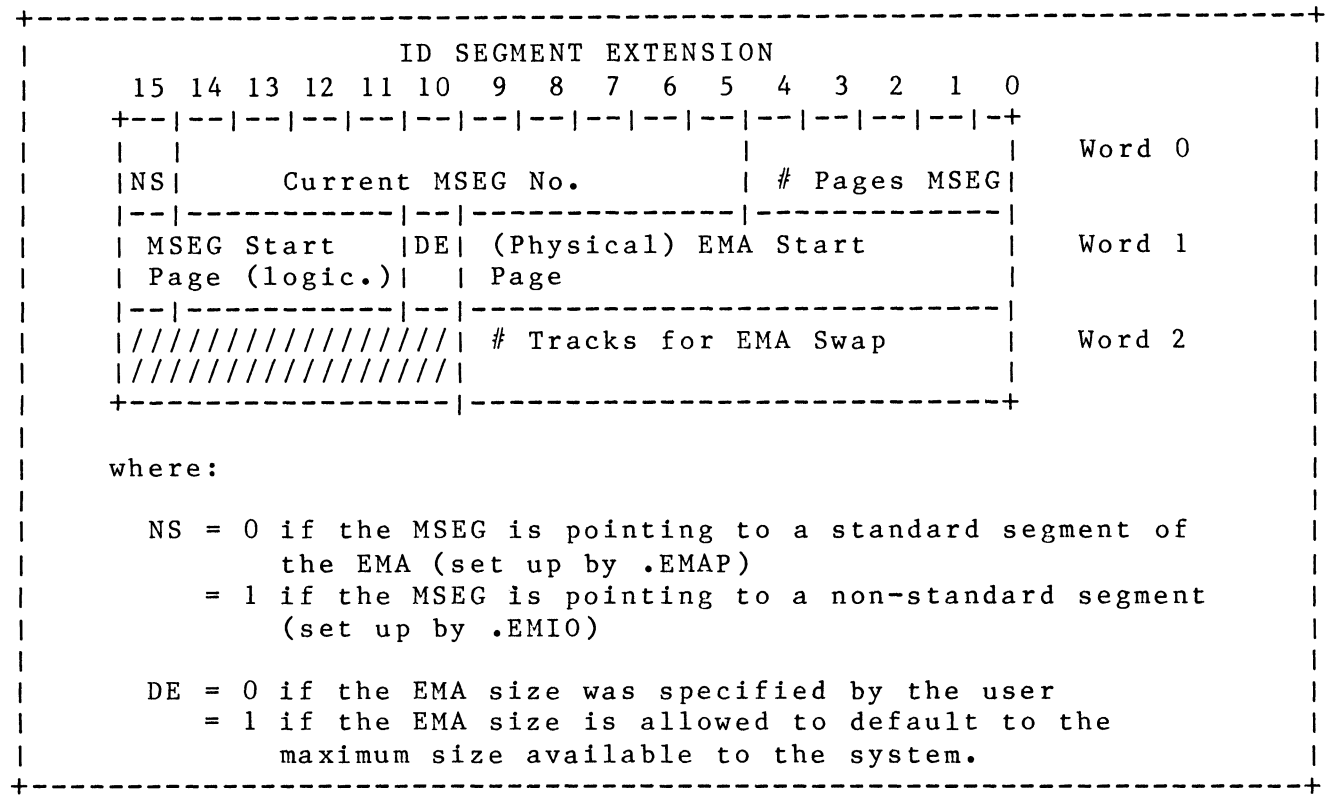


Figure 1-2. ID Segment Format (Continued).

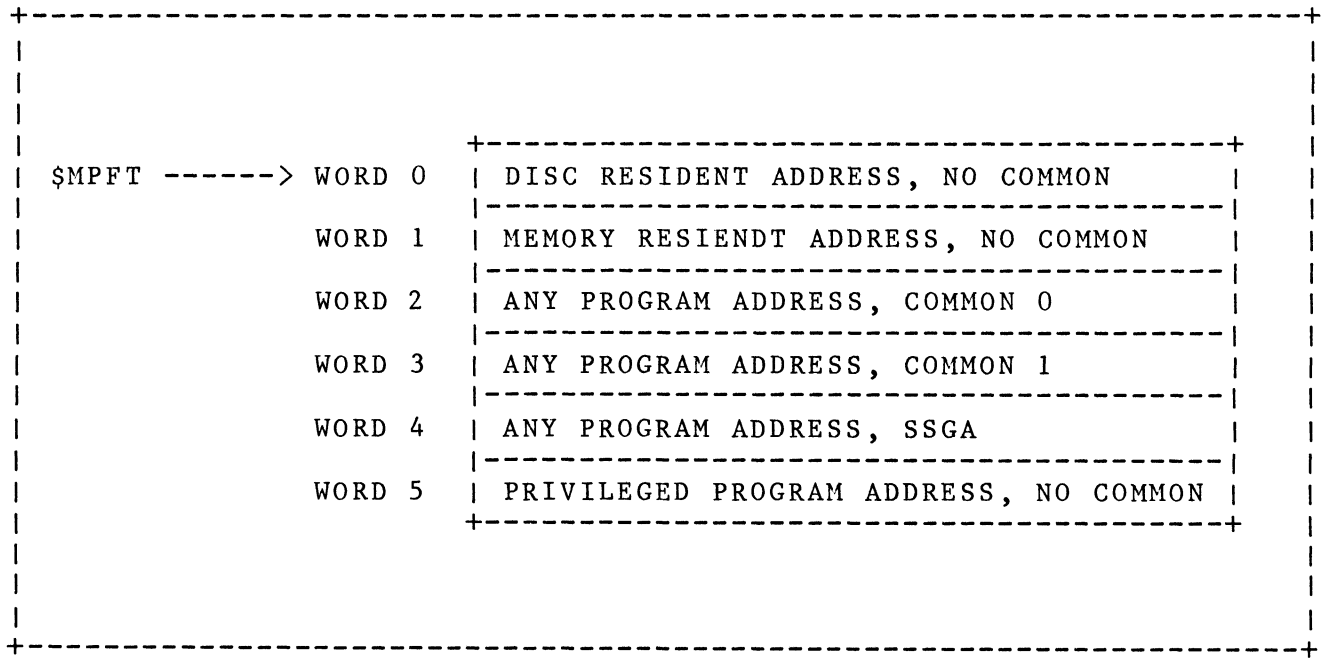


Figure 1-3. RTE-IVB Memory Protect Fence Table

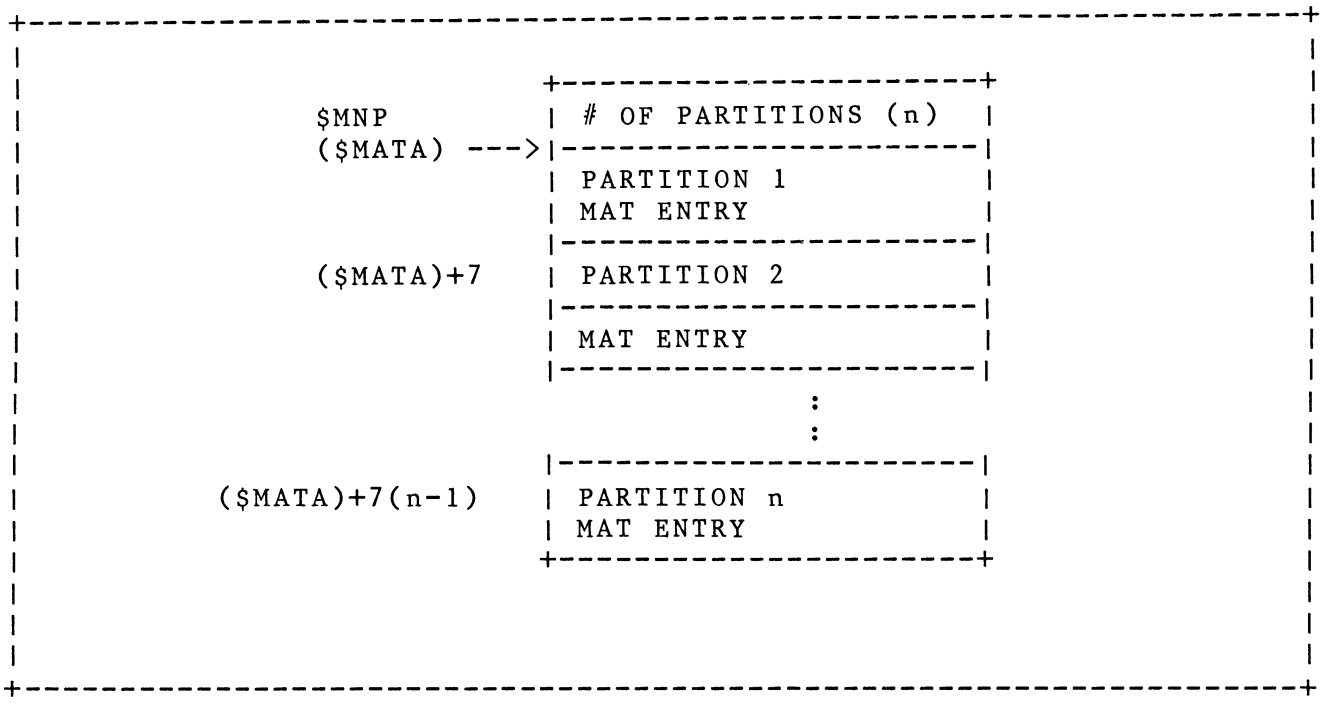


Figure 1-4. RTE-IVB Memory Allocation Table (MAT)

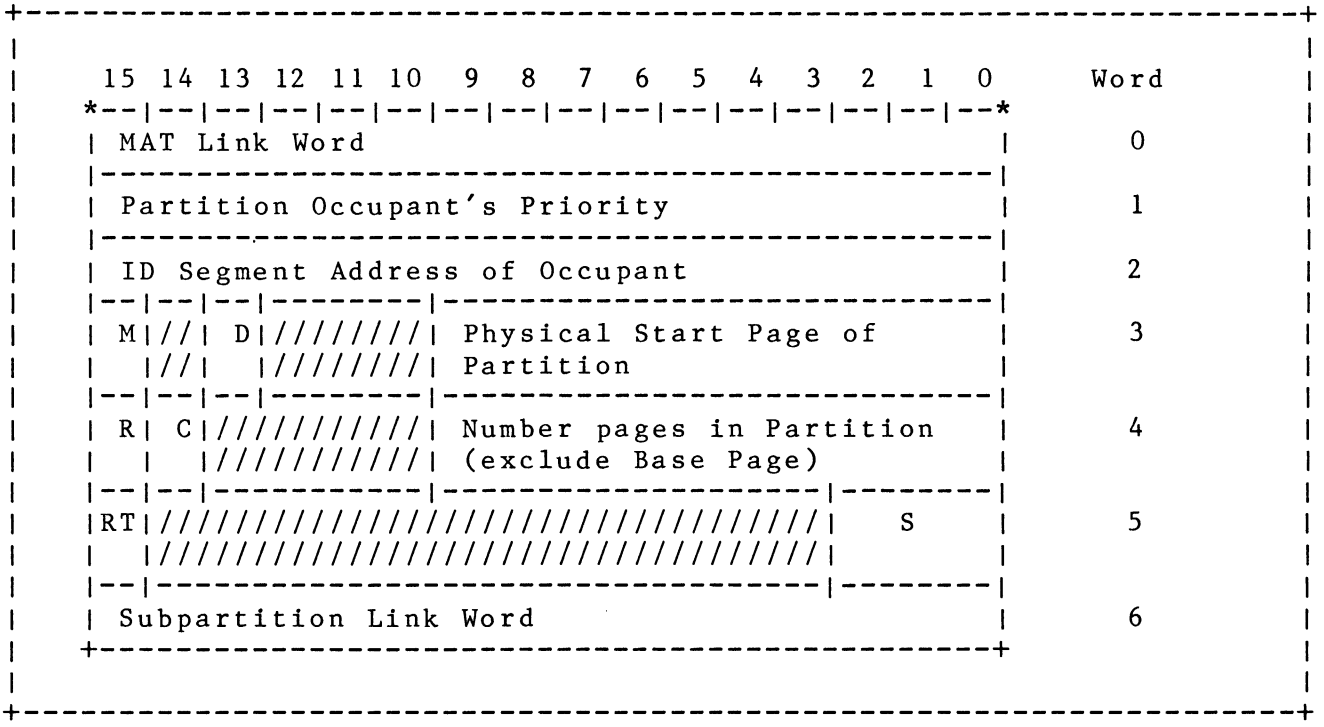


Figure 1-5. Memory Allocation Table Entry Formats

```
where:

MAT Link Word = -1 if partition not defined either during
                  system generation or by parity error

              = 0 if end of list

M             = 1 if MAT entry is for a mother partition

D             = 1 if program is dormant after save-resource
                  or serially reusability termination

R             = 1 if partition is reserved

C             = 1 if partition is in use as part of a chained
                  partition

RT            = 1 if MAT entry is for real-time partition

S             = program's dispatching status

              = 0 - program being loaded
                1 - program is in memory
                2 - segment is being loaded or swapped out
                3 - program is swapped out
                4 - subpartition swap-out started for mother
                    partition
                5 - subpartition completed. Mother partition
                    cleared.

Subpartition Link Word

              = 0 if MAT entry is not a subpartition or a mother
                partition
              = next subpartition address if subpartition
              = mother partition MAT address if this entry
                is the last partition.
```

Figure 1-5. Memory Allocation Table Entry Format (continued)

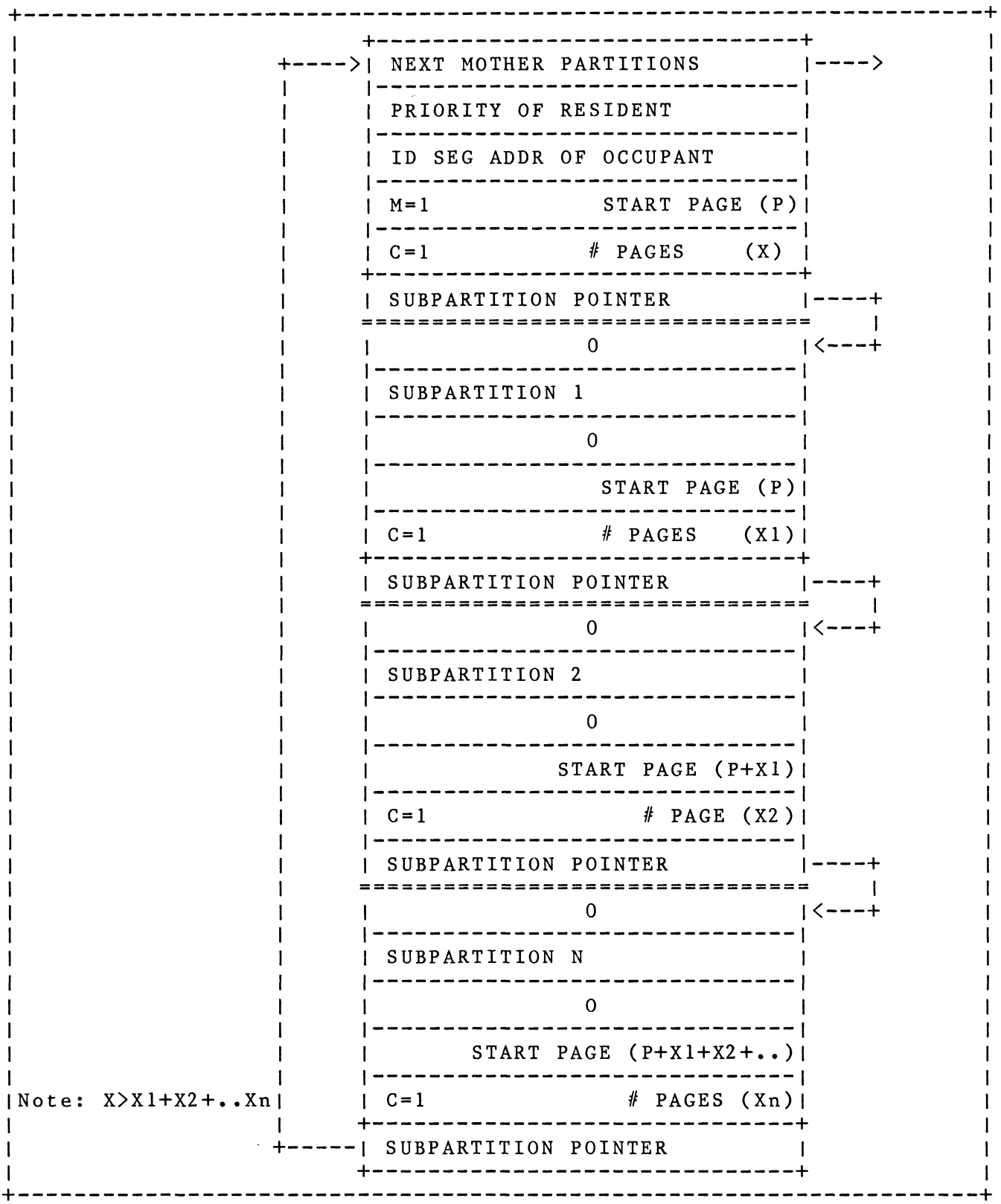


Figure 1-6. RTE-IVB Mother Partitions

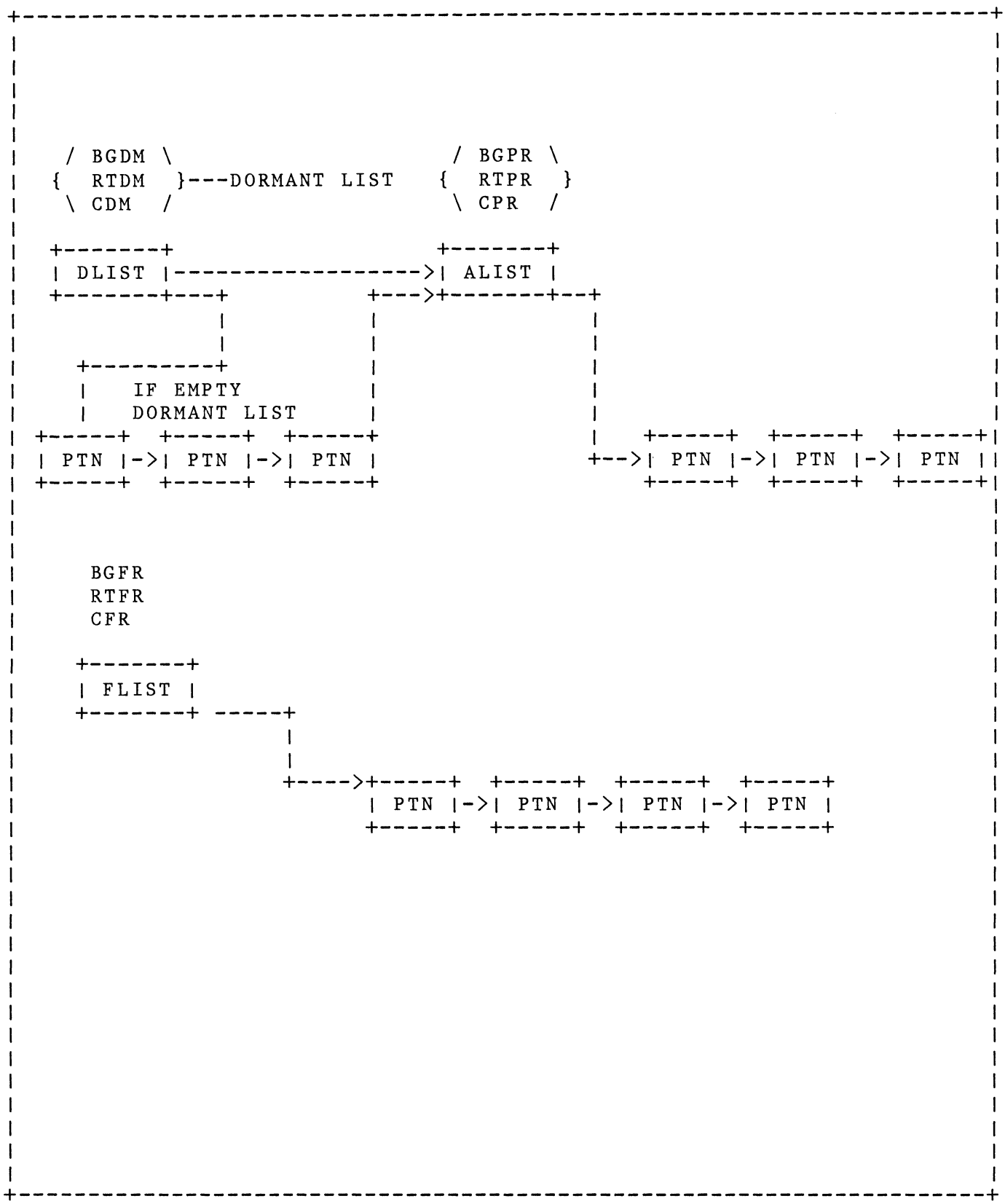


Figure 1-7. RTE-IVB Dispatcher's Allocated Partition Lists

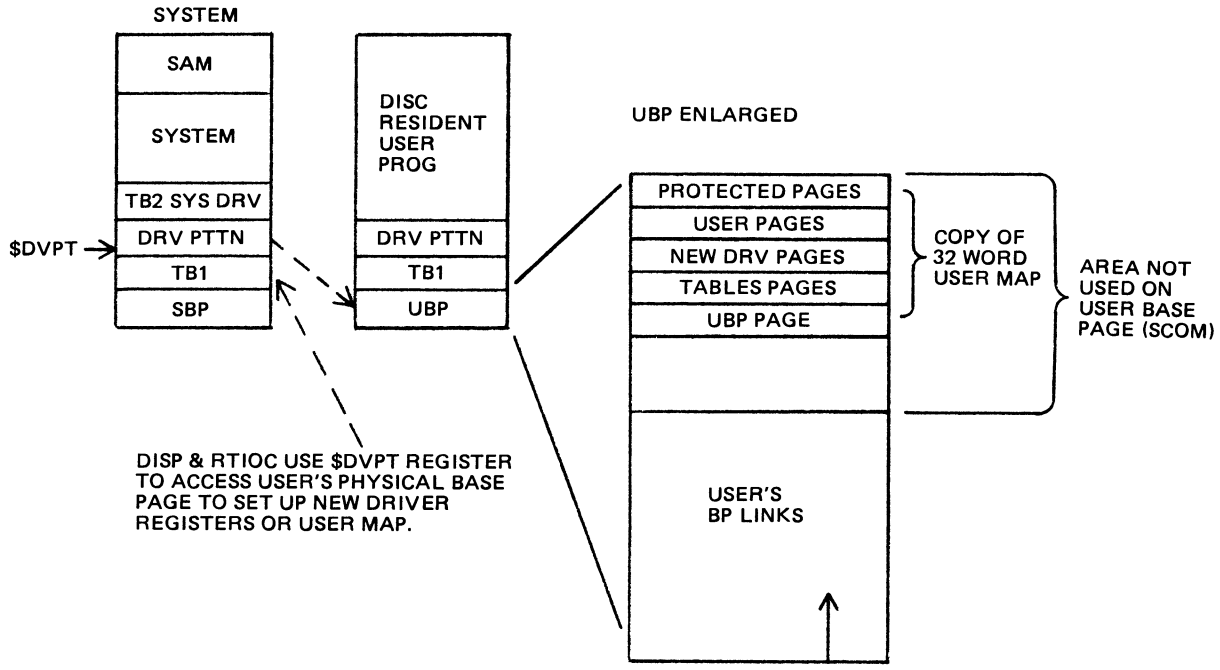


Figure 1-8. RTE-IVB User Base Page

Chapter 2

RTIOC OVERVIEW

GENERAL OVERVIEW OF OPERATION

The RTIOC module controls all aspects of the system's input and output operations. It serves as the centralized I/O interrupt handler which identifies the source of an interrupt and turns control over to the appropriate processor. All I/O requests are made to this module either by EXEC calls from user programs or by $XSIO calls from the other parts of the operating system. The I/O requests are passed directly to the appropriate device driver if the driver is available. The I/O module also queues I/O requests for busy drivers or for buffered requests. All the necessary mapping and base page communication area words needed to perform I/O will be done by RTIOC. Upon completion of I/O, the next request (if any) is started and control is returned to any waiting programs. RTIOC also detects and reports errors at various phases of the process and performs any necessary clean-up.

EXTERNAL COMMUNICATION

RTIOC communicates to the rest of the operating system through table structures, the system's base page communication area, and by subroutines available only to the system modules. The tables used by RTIOC are described below. The table formats are shown in Figures given at the end of this chapter.

EQUIPMENT TABLE (Figures 2-1 AND 2-2)

Each I/O controller and device controlled by the IOC/driver relationship is defined by static and dynamic information in the Equipment Table. Each EQT entry is composed of 15 words. If there is an EQT extension, the address of the extension is in EQT Word 13 and the size of the extension is in EQT Word 12. This table is built by the Generator.

INTERRUPT TABLE (Figure 2-3)

A table, ordered by hardware interrupt priority, designates the associated software processor and the procedure for initiating the processor. This table is constructed by the System Generator on information supplied by the user in configuring the system. The table consists of one entry per interrupt source--each entry contains only one word. The contents of each valid entry is the identifier of the processor. System processors are noted by positive values, user processors by negative values, and a zero denotes an unused entry.

DEVICE REFERENCE TABLE (Figures 2-4 AND 2-5)

The Device Reference Table provides the user for logical addressing of physical units defined in the Equipment Table. "DRT" consists of two-word entries corresponding to the range of user-specified "logical units," 1 to n where n is less than or equal to 254 (decimal). All word 1's are in one table followed immediately by a second table containing all DRT word 2's. The contents of DRT word 1 for a given logical unit is the relative position of the EQT entry defining the assigned physical unit, in bits 0-5, and the subchannel of the EQT entry to be referenced by this logical unit number, in bits 11-15. The LU lock flag (the resource number being used for the lock) is in bits 6-10. An unassigned unit contains entry value of zero.

DRT word 2 contains a flag (bit 15) indicating whether a device (lu) is up or down (0/1). If a device is down, then all I/O associated with the device is stacked on the major LU (first LU for this device in the DRT) in bits 0-14 of DRT word 2. If the downed LU is not the major LU, then bits 0-14 of DRT word 2 will contain the LU number of the device's major LU.

Certain logical unit numbers are permanently assigned to facilitate system, user and system support I/O operations. These are:

```
  0 - Bit Bucket
  1 - System Teletypewriter
  2 - System Disc
  3 - Auxiliary Disc
  4 - "Standard" Output Unit
  5 - "Standard" Input Unit
  6 - "Standard" List Unit
  7
  .
  .    Assigned by user
  .
254
```

NOTE: The RTE EXEC call will not allow direct reference of all 255 logical unit numbers (maximum is 63 decimal). RTIOC will however allow access of logical units greater than 63 when accessed via a switch table. Also note that the XLUEX call permits direct access of all 254 logical units.

TRACK ASSIGNMENT TABLE (Figure 2-6)

The TAT is a table describing the availability of each track on the System Disc and Auxiliary Disc (if included in the configuration). TAT is ordered by track number and consists of a one-word entry per track. The value of an entry defines its availability:

```
     0 - Available for assignment to user or system
100000 - Assigned to system (or not available)
077777 - Assigned globally
077776 - Assigned to file management package
<ID Segment Address> - The ID Segment address of the assigned
                       user program.
```

BATCH SWITCH TABLE

RTIOC will scan a two entry table for each call made for I/O with the BATCH flag set. If session is installed in the system, the session switch table (see below) rather than the BATCH switch table is checked. If the request LU does not refer to a disc and is found, the LU will be switched to the table defined LU. This table will give an LU to LU transform for BATCH programs only. The table format is:

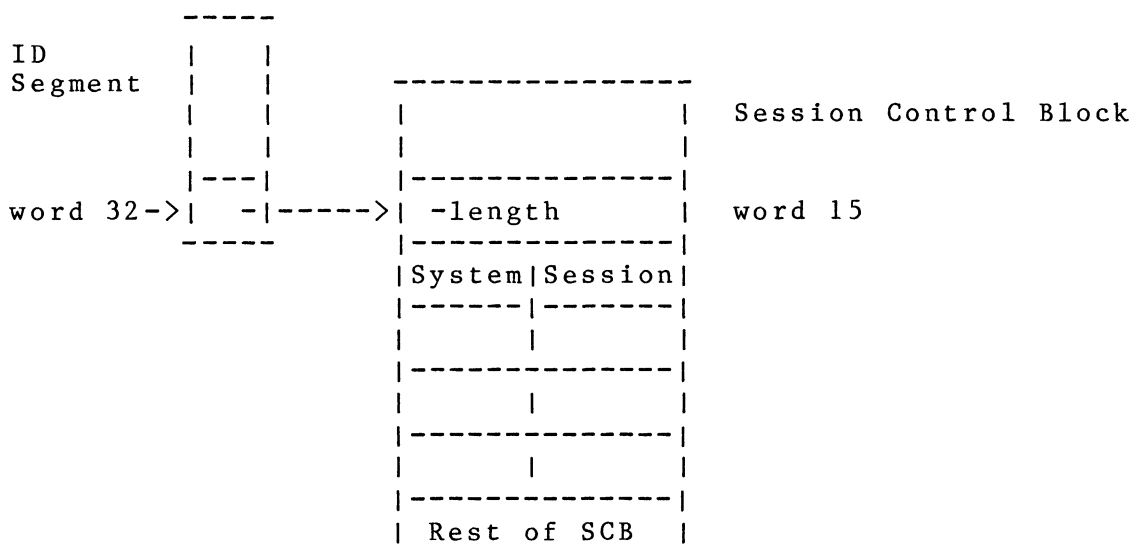
```
       ENT $LULU
$LULU  DEC -N
       REP N
       OCT -1
```

Each active word is set with the address LU in the low 8-Bits and the LU to be used in its place in the high 8-Bits. There may be up to N entries in any order. This table will be generated by the Generator and maintained by the BATCH monitor.

SESSION Switch Table

RTIOC will scan a two entry table for each call made for I/O with a positive value in the session word (ID segment word 32). If the request LU is found, the LU will be switched to the table defined LU. If the LU is not found, the request will be rejected with an IO12 error (requested LU not defined for session use).

The SST is resident in SAM (as part of a session control block) and is addressed by word 32 of the programs ID segment. This word points at the length of the SST defined for this session. The table format is:



Free space (for on-line changes/additions) is reserved with a -1. This table is set up (at user log-on) and maintained by the Session Monitor Package.

NOTE: Both tables (Session Switch and BATCH Switch) use the logical unit -1 (LU-1), rather than the actual logical unit.

DRIVER PARTITION MAP TABLE (Figure 2-7)

Each EQT will have associated with it, a two-word driver map table entry which indicates whether the driver for that EQT is in the System Driver Area (SDA) or is in a driver partition and whether the driver (if it is in SDA) does its own mapping or not. If the driver is in a partition, the entry contains the starting page number of the partition. This page number is put into the appropriate system map or user map registers to map in the driver.

The second word of each entry is used when I/O is started on the corresponding driver. The sign bit of the second word indicates whether or not, I/O is being done for a memory resident program. The word is zero for system I/O. The low 10 bits contain the page number of the user's physical base page if it is a partition resident program. This word is used to save time on setting up the proper map on processing interrupts.

EQUIPMENT LOCKING TABLE

The Equipment Locking Table is a variable length, 3-word-per-entry table with a one-word header. The table resides in Table Area II and has a pointer to the header, $ELTB, also in Table Area II.

The table header contains the table size (number of entries) in bits 0 through 14. Bit 15 (NOT EMPTY bit), when set, indicates that the table has at least one entry. Word 0 of an entry contains the EQT number. Word 1 contains the ID segment address of the locking program in bits 14-0. Bit 15 (LOCK or ABORT bit), when set, indicates that if the locking program aborts, the EQT should remain inaccessible unless another program locks the EQT to itself. Word 3 points to a linked list of $XSIO requests for that EQT issued since the EQT lock. Refer to the EQT Locking Capability section in this chapter for further information.

RTIOC ENTRY POINTS

$BFOT Subroutine - determines the Session (MTM) Terminal and sets up a buffered output request (T field=01) if all the following conditions are true:

1) LU is not zero (LU-1=B377)
2) LU is defined (not greater than max)
3) EQT is not EQT 0
4) Enough SAM is available for the request
5) LU is up

This routine will never cause a request to be initiated (call driver) as any problems will cause the NOTRD routine to be entered. This is illegal as NOTRD calls this routine.

If no other requests are pending, $DLAY is called to cause initiation of the request on the next TBG Tick.

Calling Sequence:
```
LDA BUF (1st word is -LEN, buffer follows)
LDB ID SEG ADDR OF PROG FOR ECHO
JSB $BFOT
```

RETURN (A) is not changed
All other registers modified

The request header is defined as follows:

```
**********************************************************************
      BF.HD DEF BF.CN   DEFINES START OF THE HEADER

LINK WORD IS ADDED IN SAM

      BF.CN NOP         *CONTROL WORD IS PLACED HERE
            NOP         PRIORITY OF REQUEST=0
```

```
BF.TL NOP          *TOTAL BLOCK LENGTH IS PLACED HERE
BF.UL NOP          *MESSAGE LENGTH PLACED HERE
      NOP          TRACK
      NOP          AND SECTOR INFO

*****************************************************************
                OUTPUT BUFFER FOLLOWS IN SAM
```

$BITB Value - non zero if requests are queued on bit bucket

$BLLO Value - low buffer limit

$BLUP Value - high buffer limit

$CICO Entry point - jumps here from $CIC for interrupt processing

$CKLO Subroutine - check if below the buffer limit on the current EQT.

Calling Sequence:

```
JSB $CKLO
```

$CON1 Entry point - driver completion return.

$CON2 Entry point - driver continuation return. The code to enter the driver's continuator section must be in all maps since drivers return via the address resulting from a subroutine call. RTIOC will do a JMP $UCON when ready to enter a driver's continuator section under the user map.

$CON3 Entry point - driver needing/giving up DMA from continuation section. If (A) = 5 , driver needs DMA; (A) = 6, driver wants to give up DMA.

$CVEQ Subroutine - converts an EQT entry number into the actual EQT address and calls $ETEQ to set up the base page EQT pointers.

$CXC Entry point - jumps here from $YCIC to skip the CLF and LIA in RTIOC at $CIC.

$DEVT Entry point - jumps here from the system clock routine when a device times out.

$DLAY Subroutine - used to set up a timeout to delay initiation of an I/O operation on a timed-out EQT.

Calling Sequence:

```
LDA EQT1 JSB $DLAY
```

$DMEQ Value-address of the dummy EQT used for bit bucket operations.

$DMS Value-Dynamic Mapping System status is saved here when $CIC is entered. It is used by $PERR.

$DRVM Subroutine - Sets up the map registers for entry to a driver after the base page EQT pointers are already built in base page. See the $DRVM Subroutine section in this Chapter for details of $DRVM. Called by RTIOC and DVP43.

Calling Sequence:

```
(EQT1-EQT15 already set up)
JSB $DRVM
<returns> A-reg and B-reg same as on entry
E=0 need to enter driver in System Map
E=1 need to enter driver in User Map
```

$ETEQ Subroutine-sets up the base page EQT pointers EQT1-EQT15

$IODN Entry point - to set an LU or EQT down. This processes any operator DN request (from the scheduler's message processor) that a device (LU) or an I/O slot (EQT) be down. It first determines if an LU or EQT is being set down. If an EQT is being set down, it checks the validity of the EQT via the <IODNS> subroutine. It also determines if the EQT (I/O slot) to be set down is associated with the system console. If either error condition exists, it prints an "INPUT ERROR" message and returns to the Scheduler module's message processor. Otherwise it uses <XUPIO> to set all downed LU's on this EQT into the "up" state and uses <$UNLK> to down the EQT by setting its availability indicator (bits 14-15 of EQT 4) to 01. After suspending in the general wait list any programs queued making unbuffered I/O requests, it returns to the Dispatcher.

If an LU (device) is being set down, it first checks the validity of the LU and whether the user is trying to down LU 1 or an LU pointing to the bit bucket. If one of these error conditions exists, the message "INPUT ERROR" is issued and return is made to the Scheduler module's message processor. If the LU's EQT is down, then the LU is simply marked down (set bit 15 DRT word 2). If the EQT is up, then set the LU and all other LU's associated with the device down and relink any I/O on the device's major - LU (first LU for the device in the DRT). Return is made to the Dispatcher.

$IORQ Entry point - All EXEC calls for I/O related requests are processed here.

$IOUP Entry point - To make an EQT available again. This processes any operator request (from the Scheduler's message processor) to set a device "UP". It first checks the validity of the EQT number of the device to be set up via the <IODNS> subroutine. If valid, it next schedules all programs waiting on a downed EQT or LU. Next it uses <XUPIO> to set up all LU's associated with this EQT. <XUPIO> will use <$XXUP> to relink on to the EQT any I/O found on a downed LU. If the EQT was down or available, the "up" processor will reset the EQT "up" and return to a point in <IOCOM> to start the next request. Otherwise, return is made to the Dispatcher.

$IRT Entry point - Common exit point from the system back to the user. $IRT is a routine used in exiting from the system back to the user program. It does nothing except clear the memory protect flag in a non-privileged system and restores the registers. In a privileged system, $IRT clears control on the privileged interrupt card so that when the interrupt system is reenabled for all devices, all devices can interrupt. The exit from $IRT back to the user program is a "user map enable and jump" instruction. Note that $IRT always enables the user map. The map is loaded before $IRT gets control.

$LU?? Subroutine - find session terminal LU in system LU terms. This routine interprets the passed programs ID segment looking for the session or MTM terminal LU it is associated with.

Calling Sequence:

```
LDB ID SEG ADDR
JSB $LU??
```

Return P+1 Not in session or MTM\ or LU 1 not defined (session)
P+2 A=SYS LU (LU-1) of associated terminal
B=Logical address of SST if session PROG
or
B=- Terminal LU if MTM PROG

$PSTE Subroutine - post error to session control block. This routine determines if the specified program is under session and if so, places the 1st four words of the specified error message into the error buffer in the SCB.

Calling Sequence:

```
LDA BUFFER ADDRESS
LDB PROG'S ID ADDRESS
JSB $PSTE
```

(A) and (B) are returned unchanged

$SYMG Subroutine - system error routine (to system console and session terminal). This routine routes the message to the system console (via $YMG) and then performs the following functions:

If the XEQT program is under session control (ID word 32>0), fetch the console LU from the Session Switch Table. Otherwise, use the two's complement of ID word 33 as the console LU (if the program is not under session control, the Scheduler will place the two's compliment of the terminal from which the program was scheduled into this word). If this LU equals 1, then we are all done. Else send the message to the specified user terminal. This is accomplished by calling the buffered system output routine ($BFOT).

NOTE: Processors that can't assume that the XEQT program is responsible for the error, or must perform session specific work, will call $YMG and/or $BFOT directly.

$UNLK Subroutines, used to unlink I/O requests from the current EQT I/O request queue. This is called when an LU is set down and all of the I/O for that LU is moved to the LU's down queue.

$UP Entry point - jumped to by $UPIO from Table Area 1 via SJP. This entry is used by drivers to automatically "UP" the EQT and is essentially the same code as $IOUP.

$XSIO Subroutine - called by the operating system modules to perform I/O.

$XXUP Subroutine - takes an I/O queue and positions the I/O requests (by calling the LINK subroutine) in the current EQT queue according to their priority. It returns a flag if an I/O operation should be initiated.

Calling sequence:

```
A-reg is EQT address of old device
B-reg is address of first stacked I/O requests to
   be linked onto the current EQT
JSB $XXUP
B-reg is 0 on return
A-reg is the address of the head of the current queue
   with an I/O operation to be initiated.
```

$YMG — Subroutine, system error message output. This routine provides for the output of system messages and error diagnostics on the system console. The routine maintains a "rotating" buffer area consisting of fifteen 12-word blocks; i.e., the maximum length of a message is 22 characters (11 words) plus 1 word preceding the message which contains the character count.

$EQCL — Subroutine to handle EQT locks of terminating program. Refer to EQT Locking Capability section in this section for further information.

BASE PAGE COMMUNICATION

XI
EQTA
EQT#
DRT
LUMAX
INTBA
INTLG
TAT
KEYWD
EQT1-EQT15
CHAN
TBG
SYSTY
RQCNT
RQRTN
RQP1-RQP7
XEQT etc
OPATN
DUMMY
TATLG
TATSD
SECT2
SECT3
LGOTK
LGOC
MPTFL

These are all located in the System Communication Area of Base Page.

DETAILED TECHNICAL ASPECTS OF OPERATION

INTERRUPT PROCESSING

During execution of the RTE System Generation program, RTGEN, the user can designate which one of the three basic ways each I/O interrupt may be handled during RTE execution. One way is to designate an instruction to be stored into the I/O address's trap cell.

Note that on an HP 1000 computer, a NOP or I/O instruction in a trap cell will cause trouble - DMS and MP will not be restored to the correct state if an interrupt comes through on that channel.

Another way is to store into the trap cell a JMP to a special interrupt handling routine embedded within the system. The other (and most general) way is to designate that the interrupt be handled by CIC within the Real Time Input/Output (RTIOC) module. The CIC portion of this specification defines the handling of only those interrupts to CIC. CIC is responsible for saving and restoring the current state of the machine, analyzing the source of the interrupt, and activating the appropriate software processor. This sub-module is "table-driven" by the internal "Interrupt Table" (INTBA).

Interrupt acknowledgement by the CPU causes the instruction in the word corresponding to the interrupt source to be executed. For the active I/O channels which have been specified during RTGEN to interrupt to CIC (plus I/O locations 5-7), the instruction set in each interrupt location is a Jump Subroutine (JSB-,I) indirectly to $CIC. The System Generator sets the instruction in the interrupt locations.

After entry at $CIC, CIC performs the following:

1. Disables the interrupt system.

2. Saves status of MEU at interrupt.

3. Saves all registers plus the interrupt return point in the executing ID segment.

4. Clears the flag of the interrupt source.

5. Sets the memory protect flag.

6. Sets control on the privileged interrupt card (to inhibit non-privileged device interrupts) and re-enables the interrupt system if the system has privileged interrupt. If not, step 6 is bypassed entirely.

7. Transfers directly to the interrupt processor for the sources:

   5 - Memory Protect Violation/Parity Error
   (TBG) - Time Base Generator

For other sources, the Interrupt Source Code (set in I/O location 4) is used to index to the corresponding entry in the Interrupt Table. (Refer to Interrupt Table Section in this Chapter).

SCHEDULING PROGRAMS BY INTERRUPT

If the interrupt table entry is negative, the two's complement address is made positive and is set up in a call to $LIST in "SCHED." The address is the ID segment address of the corresponding user program. After the program is scheduled, control is transferred to the $XEQ section in SCHED.

Attempts to schedule a user program by an external interrupt when the program is already scheduled or suspended cause the interrupt to be ignored and a message (diagnostic) to be output to the system TTY. Before calling "LIST" to schedule the program, the current status is examined. If dormant (status=0), the call is made and control transferred to "$XEQ". Otherwise, the program name is stored in the message

"SC03 INT XXXXX"

and a call to "$YMG" is made to perform the output request. Control is then returned to the interrupt sequence.

UNUSED INTERRUPT TABLE ENTRIES

If the interrupt table entry is zero, the undefined or illegal (spurious) interrupt is ignored.

Illegal interrupts are ignored in that the hardware flag is cleared and control is returned to the point of interruption. In addition, a message (diagnostic) recording the occurrence is output on the system console. The illegal (or undefined) interrupt code is stored in the message

"ILL INT XX"

and a call to "$YMG" is made to perform the output request.

$CIC

If the interrupt table entry is positive, the content of the entry is assumed to be the FWA of an EQT entry. The addresses of the 15-word EQT entry are set in <EQT1-EQT15>. The device time-out clock is set from the clock reset value, and control is transferred to the driver "completion" section address (word 3 of the EQT entry).

$DRVM is called to determine which map, user or system, is necessary for processing the I/O call. If the user map is needed the current contents of the user map must be saved and then the map must be reloaded to describe the program associated with I/O call. The necessary map is enabled and the driver is entered at the completion section.

If the user map is needed, entry to the driver's continuator section is entered by a JMP $UCON. If the system map is used, the simple JSB B,I will be done.

I/O REQUESTS

All input/output operations are performed concurrently with program computation in the overall system. A program is I/O suspended until the transmission or operation is completed unless automatic buffering (output only) was specified for the device or the request was a class I/O request. In these two cases the buffer is moved to system available memory and the user program is not suspended.

If a program is I/O suspended with the buffer in the user program area the program is not swappable. If the buffer is in common or system available memory, the program is swappable. A user may call REIO to move the buffer to system available memory and make the I/O call.

The user program making the request is scheduled immediately if return code 4 is used by a driver. The 5 return is made by a driver if it needs DMA to do the current request but the DMA bit is not set in the EQT.

USER I/O REQUESTS

All user I/O requests are channeled to $IORQ after initial request processing by "EXEC". $IORQ performs validity checks on the request parameters and sets the addresses of the referenced EQT entry. (Error conditions and diagnostics are described later.) The buffer address and length is examined for legality for input requests to insure that protected memory is not altered during the transfer. The last page of I/O buffers in the User Map are checked for read/write protect status to insure valid memory accesses.

If the EQT or LU specified in the request is locked to someone else, the user is suspended in General Wait list. If the EQT or LU is locked to the user making the request, the request is linked at priority 7777B to insure that the locking program's requests are queued contiguously (in case there were other requests queued prior to the lock). The suspension is overriden with the No Suspend bit in the I/O request.

DISC I/O REQUESTS

If the referenced I/O device is a disc unit, the request is checked to insure that parameters are supplied. If the disc LU number is either 2 or 3, the request is additionally checked to insure that the disc track and sector numbers are legal and that the transfer does not exceed a track boundary. If the request is output, a referenced track on LU 2 or 3 must belong to the user (i.e., the TAT entry address must equal the ID Segment Address of the user) unless the track is a load-and-go track or a global track.

BUFFERED OUTPUT AND CLASS I/O REQUESTS

If a Write or Control request references a device for which the user designated automatic buffering, a block equal to the buffer size, plus control information (5 words), is requested for allocation in the system available memory area. (Call to $ALC.) A Class I/O request is also moved into system available memory.

If the block cannot be allocated, the user program is suspended and linked into the memory suspension list. (The memory processor ($RTN) will cause the user program to be scheduled as soon as a block is released.)

After a block is allocated, the control information (CONWD and buffer length), priority and buffer (if a Write request) are moved into the block. The first word of the block is used for linking into the device list. (See the I/O Request Types Section in this Chapter).

STANDARD USER REQUESTS AND REIO REQUESTS

The parameters of a user request (which is not buffered as above) are moved into the 5-word temporary area in the ID Segment of the program. Word 1 of the ID Segment is used for linking into the device list. (See the I/O Request Types Section in this Chapter).

The user program is suspended with a suspension code of 2 (I/O suspension). This is also done for an REIO call. The only difference is that the buffer address will point to system available memory instead of the user area. The sign bit of the buffer address in the temporary words of the ID segment is set if the buffer is moved as the result of an $REIO call. This is to tell the system that the driver must process that I/O request under the system map.

ERROR CONDITIONS AND DIAGNOSTICS

Detection of the following error conditions causes a diagnostic identifying the error type, a program name and location of the request to be printed on the system teletype. The program is then aborted ($ABRT in EXEC).

	Code	Meaning
IO	01	Insufficient # of request parameters.
	02	Illegal logical unit #, or less than 5 parameters with X-bit set.
	03	Illegal EQT reference, select code = 0.
	04	User buffer violates system (or Real-Time) boundary.
	05	Illegal disc track or sector # in disc request.
	06	Write reference to protected track.
	07	Driver rejected the request as illegal for the device (unbuffered requests only).
	08	Disc transfer exceeds track boundary.
	09	Overflow of load-and-go area.
	10	Class GET occurred and one GET call outstanding on this class.
	11	Illegal user map request in System Driver Area.
	12	Referred logical unit not defined for session user.
	13	EQT or LU locked to another program (only with the no suspend bit set).

SYSTEM I/O REQUEST PROCESSOR <$XSIO>

A privileged entry is provided at $XSIO to allow modules of Real-Time Executive to call for I/O operations without incurring the overhead and procedures involved with user I/O requests. No error checking is performed, the request is linked into the appropriate I/O list at a priority of zero (highest priority) except that disc request may specify a priority, and control is immediately returned to the first word following the request. If the equipment is locked, the request is linked in the $ELTB table entry for this equipment instead of in the EQT and the driver is not called.

Request Format: A system I/O request differs from the user I/O request in format and power. Specifically, a system disc call can specify a series of transfers to be performed before the next operation is initiated.

A completion address can be specified for operation of an open subroutine at the end of the operation. This facility is only available to system routines and is useful for resetting flags, etc., because an I/O operation is always buffered to the system. A zero completion address indicates absence of a completion routine.

```
Word
          Ext     $XSIO
          .
          .
          .
 1        JSB     $XSIO
 2        OCT     <logical unit number>
 3        DEF     <completion routine address> or 0
 4        NOP     <list pointer word-set by "LINK">
 5        OCT     <control information/request code>
 6        DEF     buffer address, or <location of disc control>
 7        DEC     buffer length or <disc request priority>
 8        OCT     map word
```

Word 5 is in the same field format as the control word in a user request except that the request code replaces the logical unit.

Word 8 is set to zero if the request is to be processed under the system map. If the user map is required, the word 8 must contain the ID segment address of the program to be described. Word 8 is 100000 (octal) if the request is to be processed under the User Map as it is currently-without change. Word 8 is set to an ID segment address with the sign bit set if a modified user map is used (e.g., when the Dispatcher is swapping a portion of EMA).

Word 2 is the LU word but its sign bit is also used to pass along mapping information. If the sign bit is set, then the $XSIO I/O request must go under the current USER map saved in protected base page.

Also, the $XSIO call uses the same routine, DRIVR, to set up for and then enter the driver.

Disc Version of Request: Word 6 points to an array containing "n" sets of triplets designating the buffer control, for each transfer. The array of triplets is open-ended and terminated by a zero word:

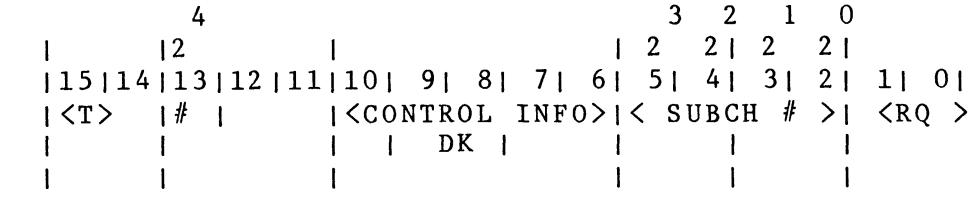
```
Word
-----
 1        DEF     <buffer address>
 2        DEC     <buffer length>
 3        OCT     <track/sector#>
 .
 .
 .
 n        OCT 0
```

Word 7 in this case is used as the priority of the request.

I/O REQUEST INITIATION

I/O REQUEST TYPES

The control word is set up as follows:



RQ - User request (1=READ, 2=WRITE, 3=CONTROL)

SUBCH# - Indicated in powers of 2; contained in a 5-bit binary word (bits 2,3,4,5 and 13).

DK - Set only for a disc request. Indicates if disc is system auxiliary, or peripheral.

T - Request type identifier
00 = User (Normal Operation)
01 = User (Automatic Buffering)
10 = System
11 = Class I/O

After the necessary legality checks are made, the request is linked into the queue for the referenced I/O device. If the request is a normal user request, the parameters are set in the temporary storage area of the ID segment. If the request is class I/O or the device has automatic buffering (output or control only), the request parameters are moved into system available memory.

I/O requests are linked in a list for each device according to priority. The requests are user (normal), user (automatic output buffering), class I/O, or system. Identification of the request type is the code in bits 15-14 of the control word in each request format. This field, the "T" field, identifies the request as:

00 User (normal operation)
01 User (automatic buffering) or buffered system request ($BFOT)
10 System
11 Class I/O

1. User (Normal Operation)

The parameters from the request are stored in the temporary area of the program ID Segment. The link word of the segment is used to contain the linkage for the I/O list.

Word	Contents
1	linkage word
2	T,control information,request code
3	buffer address or control parameter
4	buffer length
5	disc track # or optional parameter or zero
6	disc starting sector # or optional parameter or zero
7	program priority
.	remainder
.	of
.	ID Segment

2. User or System (Automatic Output Buffering)

Requests of this type are constructed in the system available memory area.

Word	Contents
1	linkage word
2	T,control information,request code
3	priority of requesting program (=0 if system request)
4	total block length in words
5	user buffer length
6	word 1 of user buffer
.	
.	
.	
n+5	word n of user buffer

3. User (Class Input/Output)

Requests of this type are constructed in the section of system available memory.

Word	Contents	
1	<linkage word >	
2	<T,control info,code >	
3	<priority of requestor>	(changed to status at comp.)
4	<total block length >	
5	<class ID word >	
6	<user buffer length >	(changed to TLOG at comp.)
7	<track option word >	
8	<sector option word >	
9	<word 1 of user buffer>	
.	. . .	
.	. . .	
N+8	<word N of user buffer>	

4. System Request

The system request is linked into the I/O list by using word 4 of the call as a link word. A system request assumes the priority level of zero (highest priority), except that disc request may specify priority in word 7 of the call.

Word	Contents
1	JSB $XSIO
2	logical unit number
3	completion routine address
4	linkage word
5	T,control information,request code
6	buffer address or disc control word
7	buffer length or disc request priority
8	map word

LINK SUBROUTINES

This subroutine provides the mechanism for linking an I/O request into the suspended list (queue) corresponding to the referenced EQT. The procedure of adding an entry (request) into the list involves only the alteration of the linkage value in the new entry and in the entry preceding the new one in the priority chain.

For all requests with priorities above 40 (larger number), the request is linked on a FIFO basis. For all other priorities (0-40) the new entry is linked according to its priority and on a first-in/first-out basis within the same priority level. The end of a list is marked by a linkage value of zero. The pointer (or head) of the list is in the word 1 of the EQT entry; word 1 = 0 if the list is empty.

If the list is empty, the link word in the EQT entry is set to point to the new entry and an indication is given to the caller of <LINK> that the new request may be initiated--i.e., <DRIVR> may be called.

The first entry in a list is skipped because it is assumed to be the requester for the current I/O operation on the device.

Calling Sequence:

```
"TEMP1"     = Location of the new request
"TEMP2"     = Priority of the new request
"TEMP3"     = Disc QUEUE flag (#0 if disc)
EQT1-EQT15  = Addresses of EQT entry
Registers meaningless

(P)         JSB  LINK
(P+1)       -return-

(A) on return = 0 if the new request is the only entry in
                the I/O list.
```

DRIVR SUBROUTINE

The subroutine DRIVR provides a central point for calling an I/O driver to initiate a new operation. This routine, before calling a driver, sets the request parameters into the appropriate words in the EQT entry corresponding to the referenced device and assigns a DMA channel if required.

If the device or controller is "down" or "busy", no action is taken and return is made to the caller. If a DMA channel is required but no channel is available, the "AV" field in the EQT is set to 3 (waiting for DMA), one is added to "DMACF" for the number of devices waiting for DMA, and return is made to the caller.

If the device is available (and a DMA channel is assigned if required), the device time-out clock is set from the clock reset value (in EQT14), the subchannel is set into EQT4 (from EQT6), and preparations are made for calling the driver. DRVMP is called to do the driver map set up (See the following Section). If a DMA channel is being used, DRIVR also sets up the DMA map by copying either the System Map or the User Map into the correct port map. Note that the DMA map does not need to be reloaded until it is reallocated. If the driver needs the user map, that map must be reloaded before each entry into the driver.

DRVMP

This code is called to set up and enable the necessary maps for the driver. The preparation for setting up the maps for the driver call includes checking word 1 and setting up word 2 of the Driver Mapping Table (Figure 2-7). The first word in the table is found by using the EQT number to index from $DVMP. The second word is found by adding EQT# to the address of the entry's first word.

If the driver is in the System Driver Area and does its own mapping, the driver is always entered under the System Map. NOTE: $XSIO calls must have zero in the eighth parameter, no checks are made.

If the driver is in the System Driver Area but does not do its own mapping, the T-field of the request is examined. If the T-field is zero (normal user) the program must be type 2 or 3 in order to use the unmodified user map. The second word of the driver's Mapping Table entry is set to the physical page number of the program's base page (first page of partition). Any other types of I/O requests or types of programs requesting I/O for a driver which doesn't do mapping and is in the System Driver Area will cause the request to be rejected (a program will be aborted with an IO11 error message). NOTE $XSIO calls may not be used to call a driver in the System Driver Area if the driver does not do its own mapping! The IO11 error message will also be issued in this case, the request will be rejected and returns control to the XSIO caller.

If the driver is in a driver partition, the T-field is checked. If the T-field is 2 (system request), the eighth word is checked to see if it is either a disc program load request or a special disc I/O request. If the eighth word of the $XSIO request is 100000 (octal), it is a special request which specifies that the current User Map is used. The special request is used by the DISPATCHER and by the reconfigurator. If the eighth word is a positive value, it is the ID Segment address of a program to be loaded. The program's map is built by $SMAP. If the eighth word is negative, it is an ID segment address with the sign bit set. This form is used by the Dispatcher for swapping channels of EMA. A special user map is kept in the user's protected portion of base page just below the normal copy of the user's map. The physical page number of the base page is set into the first word of the Driver Mapping Table and the second entry is set to zero. If the eighth word of the $XSIO is a positive ID segment address and the second word has its sign bit set, then this is an I/O request that must be made under the user's existing map. All other disc $XSIO calls use the program map.

All buffered user request and class I/O requests use the modified system map. If the T-field indicates that it is an unbuffered user request, the second word of the Driver Mapping Table (see Figure 2-7) is checked to see if the request was made by a Memory Resident program. If it was, the MR bit of the second word of the Driver Mapping Table Entry is set and the modified Memory Resident map is used. If it was not a Memory Resident program request, the first page number of the program's partition is entered into the second word of the Driver Mapping Table entry and the modified User Map is used.

When a driver is in a driver partition, the map under which the driver is entered must be changed to address the physical pages of the partition. The modified map is saved if it was a user map (other than when I/O is being done by a Memory Resident program) that was modified. The purpose of this is to save set up time on each continuation interrupt. The page number in the second word of the Driver Mapping Table entry is loaded into the System Map's driver partition register ($DVPT) to map in the user's physical base page (see Figure 2-8). The copy of the modified User's Map is then stored in the top portion of the physical base page via a cross-map store through the driver partition register in the system map.

$DRVM SUBROUTINE

When a driver needs to be called as a result of an interrupt (continuation) $DRVM is called to check the Driver Mapping Table entry. The first word of the entry determines whether or not a driver partition's pages need to be set up in a map. The second word indicates which map to use.

If the second word is zero, the System Map is used. If the driver is in a partition (Word 1 has the partition's starting page number) the System Map is modified to address this partition. If the driver System is in the Driver Area, no modification is necessary. $DRVM returns with an indication that the System Map is to be used (E=0).

If the second word has the sign bit set the Memory Resident Map is used. The current user map is saved in a buffer (SVUSR). Then the User Map is set up with the Memory Resident Map and the driver partition registers are set up according to word 1 of the mapping table. $DRVM returns with an indication that the User Map is needed (E=1).

If the second word has a page number, the necessary user map is already set up and is stored in the last 32 words of the indicated page. $DRVM saves the current user map in SVUSR. Regardless of the driver's area of residence, the User Map is specified (E=1).

Note that the user map is saved and set up to the required map only if it is not already mapped in the user map. This saves time in setting up duplicate maps.

$RSM SUBROUTINE

This routine is called on every return from a driver. It checks the flag DVMPS to see if the user map was changed. If it was, RSTUS reloads the user map with its original contents (saved by DRVMP or $DRVM) and clears flag DVMPS.

I/O DRIVER INITIATION RETURN

Upon return from the driver $RSM is called to restore the user map (if it was changed) to its status prior to driver entry. The driver returns a code to DRIVR indicating whether the operation was accepted or rejected and the cause of the reject. This code is in A on return:

```
    0 - operation successfully initiated.
    1 - Read or Write request illegal for device.
    2 - Control request illegal or not defined.
    3 - equipment malfunction or not ready.
    4 - operation successful-immediate completion.
    5 - driver requires a DMA channel for this operation but the
        "D" bit is not set in EQT.
    6 - initiation OK, but driver wants to give up DMA.
 7-59 - Reserved for HP RTE System modules and system drivers.
60-99 - Reserved for user drivers.
```

If the code is 5 a DMA allocation is attempted and if successful, the driver is reentered with the request.

If the operation was otherwise rejected, DRIVR returns to P+2 of the call with the reject code in A.

If the code in A is 3, the device was found to be unavailable for I/O (not ready). The device availability indicator is set to 01. If a DMA channel was allocated, it is released. The "NR" diagnostic is printed and IOCM is exited either back to $XEQ in the Scheduler or to the completion routine specified in a system request. If the code in A is 1, 2, 4, 7 or greater, control is transferred to subroutine <ILLCD>. If a zero is returned, I/O was initiated successfully, with subsequent device interrupt expected, and control is transferred to $XEQ in the Scheduler module to switch to the next lower priority which requires execution time.

DMA CHANNEL ALLOCATION

The two DMA channels are dynamically allocated to the high-speed and synchronous devices identified to RTIOC (bit 15-1 of word 4 in the EQT entry). The assignment process consists of setting the EQT address of the device in the DMA channel entry in the Interrupt Table and setting the channel number in the word "CHAN" in the Communications Area.

A driver with its EQT DMA bit not set may also request a DMA channel by setting A=5 and returning to the system at initialization of the I/O request, or at the (P+3) point from continuation. A driver can also give up DMA channels allocated to it by setting A=6 and returning to the system at initialization return point or at P+3 from continuation.

If more than one device is waiting for a channel, the order of priority for assignment is the order of the Positions in the Equipment Table. There are two exceptions to this scheme:

1. If the first entry in the EQT is waiting for a DMA, the channel is assigned to that device, which is assumed to be the system disc.

2. If the first entry encountered (other than entry #1) just released a DMA channel, then the next lower priority device waiting for DMA is used. This allows for a "switching" operation in the allocation of a DMA channel.

Special processing is required by any I/O driver which uses the interrupt on a DMA channel to perform data transmission with the device. A software flag must be set after a DMA channel is initiated to indicate that the channel is active and that a completion interrupt is expected. The setting of this flag is to set Bit 15=1 in the Interrupt Table word corresponding to the DMA channel:

INTBL(1) - channel #1 (location 6)
INTBL(2) - channel #2 (location 7)

The address of INTBL is contained in the word "INTBA" in the Base Page Communication Area. When Bit 15 is set, the rest of the word must not be altered. This operation must be done only if DUMMY is non-zero. When a system has privileged drivers, i.e., DUMMY = 0, control is cleared on both DMA channels everytime an interrupt is processed through CIC - in order to let the privileged interrupts be the only ones "on". Thus if a driver needs that DMA interrupt, it must set bit 15 in the appropriate word. $IRT checks these words and if the bit is set, it reenables the DMA interrupt.

I/O COMPLETION

The return point by an I/O driver (from a call by CIC) indicates the continuation or completion of the I/O operation. In RTE-IVB, the user map is restored if it was modified for driver entry. RSTUS is the routine called to do this.

1. Return at (P+1) : Completion of the operation. CIC transfers directly to the IOC completion section at "IOCOM".

2. Return at (P+2) : Continuation of the Operation. CIC restores all registers and returns to the point of interruption, with the exception of special processing which must be done for operator attention: If the flag is Base Page Communication Area "OPATN" is set = 0, control is transferred to $TYPE in SCHED: "OPATN" is set = 0.

3. Return at (P+3) : Need (or want to give up) a DMA channel (A=5 or 6 respectively). For A=5, if channel is allocated immediately, then reenter driver at initiator. If no DMA channel is available now, then queue the EQT and do continuation return as at P+2. When the DMA channel is allocated, then the driver will be entered at the initiator. For A=6, any channels allocated are given up, and any programs waiting are resumed. Otherwise, this acts as the (P+2) return. These features are for HP use only and are subject to change without notice.

IOCOM

This section is responsible for the initiation of stacked I/O operations, placing a program back in the scheduled state when its I/O operation is completed, dynamic allocation of the two DMA channels among synchronous devices, and calling for operator notification of equipment errors or malfunction.

<IOCOM> is entered directly from <CIC> when an IO operation is terminated and all error recovery procedures have been attempted. On entry to this section, (B) contains the number of words (or characters) transferred. ((B)=track # on which error occurred if disc.)

The addresses of the Equipment Table entry are in EQT1 to EQT15 in the Communication Area in Base Page from the CIC pre-processing. The device time-out clock is cleared.

After completing the processing for the completed successful operations, IOCOM checks a stacked request for the device. If none, IOCM transfers to "IOCX." The user program for the completed operation has already been rescheduled.

If a request is stacked, the subroutine DRIVR is called to initiate the operation.

The IOCOM exit section "IOCX" transfers control to:

1. <Completion Routine> if the system I/O request specified.

2. L.136 if the bit bucket has I/O stacked on it which must be completed.

3. $TYPE (in SCHED) if the operator attention flag is set (the flag is also cleared by "IOCX").

I/O COMPLETION ERRORS

An I/O driver informs <IOCOM> of an error or malfunction at the end of an operation. This is indicated by a non-zero value in A-register on entry to IOCOM; these values are:

0 - good transmission-no error
1 - not ready/malfunction
2 - end-of-information
3 - transmission parity error
4 - device time-out from driver

A driver which obtained a DMA channel by returning from its initialization section with A=5 must indicate this fact to IOCOM at completion so that the DMA channel is released. This is done by setting bit 15 of the A register on exit from the continuator of the driver at the end of an operation.

In addition, an I/O driver error value of four (device time-out) is simulated by the $DEVT routine in RTIOC if a device times out. In such a case, the device never interrupts and the driver is therefore never entered. When the clock processor in the Scheduler module finds a timed-out device, it gives control to $DEVT. This $DEVT routine sets the time-out bit in EQT4 of the device's EQT entry, if bit 12 of EQT4 is set the driver is entered with A=channel #, otherwise the A register is set to four, the B register to zero (transmission log) and IOCOM entered. Thus, an entry to IOCOM from a driver, with an I/O error of four, is simulated.

At <IOERR> a diagnostic indicating the condition and the device is printed on the system teletype and the LU is declared down; i.e., bit 15 of DRT word 2 is set=1.

The format of the diagnostic is:

```
      NR
"IOET Lwww 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/from the device
TO = the device has timed out

Additionally, any other LU pointing to the same device (as defined by EQT # and subchannel) will be set to the down state. The first LU defined in the DRT for this device is called the major LU. Any I/O associated with this device linked on the EQT will be unlinked (at <$UNLK>) and relinked on the major LU in DRT word 2, bits 0-14. All other downed LU's for this device will have the major LU's number in bits 0-14 of DRT word 2.

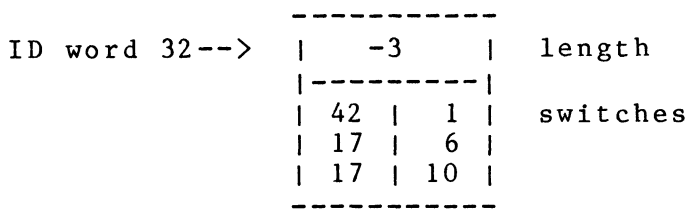
(Exceptions to this are the system teletype which is not "downed" and disc errors which cause special processing.)

We are finished with the error reporting if:

1. The action that generated the error was a buffered completion request.

2. The requesting program (the one that made the I/O request) was initiated from LU 1.

Otherwise, report the error diagnostic to the terminal from which the program was initiated (via $BFOT). If the requesting program is under session control, scan 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), issue the error diagnostic to the session terminal, but use 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:



-3		length
42	1	switches
17	6	
17	10	

The system console will receive the following diagnostic:

"IONR L 17 E10 S 0 004"

while 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 digits at the end of the message represent the device status returned by the driver.

Disc parity Error/Abort Processing:

This error track number and the EQT number and subchannel number of the disc are set in the diagnostic:

"TRnnnnn EQTxx,Uyy S/U"

which is printed on the system teletype. The S or U in the message indicates a system or user request was involved:

System - the (B) = -1 on entry to the completion routine; the result is that the program loading is aborted.

User - to LU2 or LU3: the referenced track is set to "100000" (unavailable) in the TAT and $ABRT is called to abort the user program.

A disc is not declared "down" as a result of a disc parity error.

ILLCD SUBROUTINE

This subroutine is entered primarily if an illegal request is detected by an I/O driver. The reason is a Read or Write operation is illegal for the device or a control request is meaningless for the device. An additional reason for transfer to this section is an "Immediate Completion" (Code 4) return from the driver; it is processed as a control reject.

Additional error messages may be defined by HP system or user drivers as follows:

6-59	Reserved for HP RTE system modules and system drivers.
20-29	Reserved for the Spool Monitor.
30-39	Reserved for Multipoint Driver (DVR07).
60-99	Reserved for user drivers.

Error procedure is:

1. If the request is processed as buffered output, the temporary block is released to available memory.

2. The reject is ignored if a system program generated the request --however, a completion routine, if specified in the request, is operated. (NOTE: this philosophy is based on the assumption that this condition should never occur.)

3. A user control request (A=2 or 4, refer to I/O completion section) which is rejected is treated as if it was performed. The program is linked back into the schedule list.

4. An unbuffered user read or write request reject (A=1) causes a diagnostic to be issued ("IO 07") and the program aborted.

5. Other reject codes (A>5) for unbuffered user read or write requests will be mapped into an IOxx error message where "xx" is the error code and the program will be aborted.

MISCELLANEOUS ROUTINES

<$IOCL> Subroutine

The function of this routine is to remove a program from an I/O hang-up condition resulting from an input request not being completed by the device.

This "clearing" procedure is initiated by the operator in using the I/O Abort version of the "OF,XXXXX,1" command. The "OF" statement processor in "SCHED" calls this section if the referenced program is suspended for an I/O input request.

The list of each EQT, down DRT, and $ELTB entries is searched to find the queued request corresponding to the ID Segment of the referenced program. The entry is removed from the list and the list is appropriately linked to reflect the change. If the entry was the first one in an EQT list (i.e., an active request) and the EQT is not down or in DMA wait then a clear request (100003B) is forced into the initiator. This can be done only after the Driver Mapping Table entry is checked and the driver is mapped in if needed. If the request is accepted then an interrupt is expected and the device set busy with an arbitrary timeout of 1 second. The sign bit of EQT word 1 is set indicating device clearing. The timeout will be trapped in <$DEVT> and routed to <IOCOM>. <IOCOM> recognizes special interrupts on timeouts associated with device clearing by checking the sign bit of EQT word 1. If the request is not accepted, then the timeout is cleared and control is given to <IOCOM> for initiating the next request.

<IODNS> SUBROUTINE

This subroutine checks the legality of an EQT number. If it is valid, it returns to the caller; otherwise, it sets up to print out the diagnostic "INPUT ERROR" and goes to the Scheduler module's message processor.

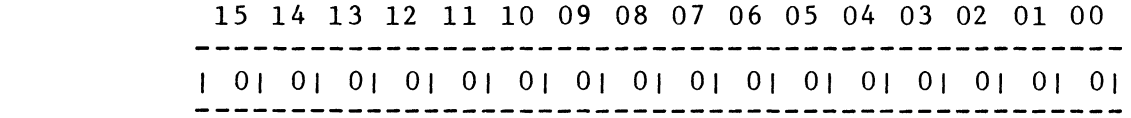
CLASS I/O REQUESTS

Class I/O refers to no-wait I/O in which the user directs the completion information to a "class" by number. The user requests I/O on a class. The RTIOC requests buffer memory for the request, moves the request to the buffer memory, queues the request on the specified EQT, and enter in the class queue that a request is pending. On completion, the completed request is queued in the class queue, and any program waiting for the class is restarted.

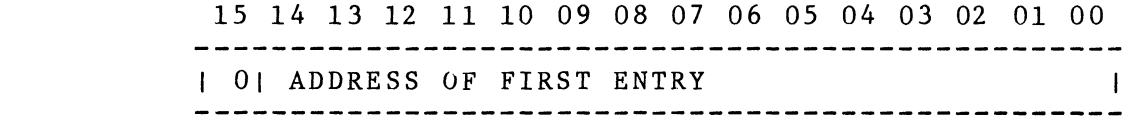
The class table is defined at generation time and is located at $CLAS. The table consists of a length word defining the number of classes, followed by one word for each class.

CLASS I/O "QUEUE FORMAT AND USE"

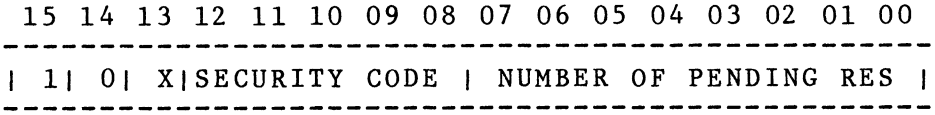
The class queue can be in four different states.



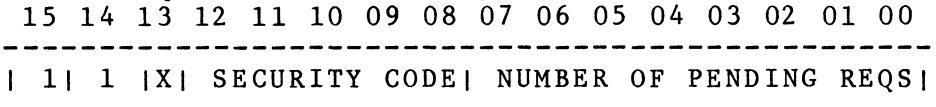
State 1: Class deallocated, available



State 2: Pointer to first entry in class queue



State 3: Class allocated, no one waiting on class. Number of pending requests counter may be 0-255



State 4: Class allocated, someone waiting (suspended). Number of pending requests counter may be 0-255.

Actions to be taken when handling a class I/O or get request depend on the current state of the class queue head.

Get Requests:

State 1. Abort the program I000, no class
State 2. Return the data from class buffer
State 3. Set the some one waiting bit (bit 14), suspend program
State 4. Abort the Program I000, only one program may be suspended per class.

Class I/O Requests:

State 1. State 3 is set up, security code is low 5 bits of program ID Number, counter is set to 1.
State 2. The counter at end of queue is incremented by 1.
State 3. The counter is incremented by 1.
State 4. The counter is incremented by 1.

On Completion of Class I/O Requests:

State 1. Illegal---should never happen---buffer is returned and the completion is ignored.
State 2. The new data is added at the end of the list (FIFO) and the counter is decremented by 1.
State 3. The new data is added at the end of the list (FIFO) and the counter is decremented by 1.
State 4. The waiting program is scheduled and the counter is decremented by 1 and the someone waiting bit (Bit 14) is cleared.

EQUIPMENT LOCKING CAPABILITY

The equipment locking capability allows a program to control access to the device controller associated with a given LU. A program can lock the controller connected to one or more I/O slots and one or more devices for the duration of the program and then unlock it. This capability is needed during on-line system switching (program SWTCH) and on-line equipment diagnostics.

During on-line system switching, SWTCH reads a newly configured operating system from one area of the disc and writes it onto the operating system area of the system disc. In general, the operating system on the disc is invalid during this time. Any other program causing any part of the operating system to be accessed from the disc needs to be prevented.

On-line diagnostic programs may change the characteristics of a controller (e.g., by down-loading a different set of instructions in its RAM).

FUNCTIONAL DESCRIPTION

A program can lock the I/O controller associated with an LU by calling subroutine EQTRQ. If the controller is not locked at the time of the request to another program, and if the equipment locking table is not full, a successful return will be made. The equipment locking table is currently set at two-entries long, up to two controllers can be locked in the system at any instant by one or more programs. After an EQT is locked to a program, no further I/O requests are queued on the EQT; however, there may be some requests already queued on it. This poses no problems for a locking program using the operating system for its I/O, because its post lock I/O requests will be queued after the prelock requests, but it may mix I/O commands issued by a privileged locking program with those for I/O requests queued before the lock. It is up to the privileged program to insure that the EQT is "Not Busy" before issuing I/O instructions. The program may make the call with or without Wait in the usual manner. It may further specify in the call if it wants the equipment to stay locked if the program should be aborted (i.e., terminated in any way other than via an EXEC 6 call). If the program does so specify and then if it should abort, the equipment stays locked, though to no one in particular. Any program can then issue a lock request for that equipment and will succeed in doing so.

A program can also unlock an equipment locked to it by calling the same EQTRQ subroutine. Such a call is always without wait, and no other options are provided. If the equipment specified in the call is locked to the calling program, the unlock request will be successful.

When a diagnostic program aborts, maintaining the lock on an equipment, a recovery program should be scheduled immediately. This program should lock the equipment to itself, reset the system

environment (the controller micro-code memory, etc.) to the standard state for other programs, and unlock the equipment, before exiting. If a diagnostic program aborts after changing the characteristics of the disc controller, a recovery program can be scheduled only if it is in memory. This can be accomplished, for example, by the recovery program scheduling the main diagnostic program as a son. Such circumstances should be taken into consideration when designing the diagnostic procedures.

There is a possibility that, in the short period (window) while the EQT is free,i.e., when a diagnostic program aborts but the recovery program has not yet been dispatched, a user program may access the "abnormal" equipment via an LU associated with it. But this possibility is remote, because the user program would have to know exactly when the window became open, and the program would have to initiate the equipment lock to itself (at assembly time) before being able to access it.

CALLING SEQUENCE

The following calling sequence is used in a program to allow it to lock or unlock the equipment associated with a given LU.

Assembler:

```
                EXT   EQTRQ
                  .
                  .
                JSB   EQTRQ
                DEF   *+3
                DEF   IOPT
                DEF   LU
          Return----------
                  .
                  .
LU      DEC   Logical Unit Number
IOPT    DEC   Option word
```

FORTRAN:

```
CALL  EQTRQ (IOPT, LU)
```

Bit assignments in the option word are as follows:

```
Bit     0    1 = Lock, 0 = Unlock

       14    1 = No Abort on Call Error; return ASCII code in
                 A, B registers
             0 = Abort on error
```

In addition, for the lock request:

```
Bit    13    1 = Keep EQT locked on abortion
             0 = Release on abortion
```

15 1 = Without wait; 0 = with wait

Thus, bit 0 specifies the main request code. Bit 14 is the No Abort bit and has the standard RTE-IVB meaning. If it is set, any EQxx errors defined in abort error response will be passed back in ASCII in A and B registers at the return address, and the program will not be aborted. If it is set, and there are no EQxx errors, return is made at return address + 1. Bit 13 specifies that if the caller, after having successfully locked the specified equipment, should be aborted, the equipment should stay locked, though not to this caller. A subsequent program can use it only after requesting to relock it. Bit 15 is the without-wait bit and has the standard RTE-IVB meaning. If it is set, a return is made whether or not the lock is successful; A and B registers indicate the status of the request as defined by the no-abort error responses.

Further, if a locking program terminates saving resources (EXEC 6 request code with INUMB parameter = 1), then the locked equipment stays locked to that program ID segment address.

RETURN CONDITIONS

Abort Error Responses

There are two abort errors for this call:

EQ01 - System Console (LU #1) specified
EQ00 - Illegal LU# specified (LU # specified is > LUMAX)

No-abort Error Responses

On return from the Lock Without Wait request, content of the A register has the following meaning:

A = 0 if successful or required equipment already locked to this program (locking a bit bucket is always successful and results in an NOP); also the locked EQT # is in register B.

1 if equipment associated with specified LU locked to another program;

-1 if equipment lock table full.

On return from the Lock-With-Wait request, the A and B registers contain:

(A) = 0
(B) = number of locked EQT

For a Lock-With-Wait request, if the equipment associated with the specified LU is locked to another program, or if the equipment lock

table is full, a return is not made. The calling program is put in State 3 (General Wait) until the request can be fulfilled.

On return from the Unlock request, register A contains the following:

A = 0 if successful (unlocking the bit bucket is always successful and results in a NOP);

-1 if the required equipment was not locked;

1 if the required equipment is locked to another program;

2 if the required equipment has not completed I/O for the locking program.

DRIVER INTERFACE SUBROUTINE CHEL

An Assembler subroutine, CHEL, is also provided to be called by drivers to interrogate if the specified equipment is locked. The calling sequence for CHEL is as follows:

Assembler:

```
                    EXT   CHEL
                      .
                      .
                      .
                    LDB   EQT #
                    JSB   CHEL
              Return----
```

On return, if the specified equipment was not locked, content in register A is equal to 0, otherwise, it is the ID segment address of the locking program.

INTERNAL OPERATION

A program requiring to lock the equipment associated with a given LU makes a call to System Library routine EQTRQ, which will be loaded with the program. If the LU is illegal within the present system (i.e., points to the system console or has no EQT associated with it), the program is aborted with an EQXX error or is so informed. If the EQT is not locked, and there is room in the equipment lock table, $ELTB, an entry of the program ID segment address is made in $ELTB for the equipment. A table-empty bit is updated if this is the first entry being made in the table. If the LU specified is a bit bucket (i.e., if it is 0 or points to equipment number 0), the lock request is allowed, but results in a NOP. A bit is set in the entry if the program specifies that the equipment not be released if the program terminates abnormally. If a program is going to bypass the operating system to access the equipment now locked to itself, i.g., via a privileged driver, it must insure that all currently scheduled requests for that equipment, if any, are completed before accessing it. If the table was full and the call was with wait, the program is put in the general wait list indicated in the table header. If the

call was without wait and the table was full, the caller is so informed. If the EQT is locked to someone else, and the call is with wait, the program is put in the general wait list, flagged on that equipment entry in the table. If the call was without wait, and the equipment was locked to someone else, the caller is so informed. If the equipment was locked but without any program ID segment address, the caller is allowed to lock it.

In RTI04, code is added within the request code analysis area to suspend in the general wait list the calling program accessing a locked equipment. If the program wants to access an equipment locked to itself, the priority of the request is reduced to 77777B in order to dispatch single program requests in chronological order, before continuing the analysis.

In the ABORT processor in DISP4 (which processes all terminations, normal or abnormal), a check is made to see if $ELTB is Not-Empty. If so, subroutine $EQCL in RTI04 is called to handle the EQT lock terminating process.

For each EQT locked to the terminating process, $EQCL does the following:

* updates the Not-Empty bit (if need be) in the $ELTB table header,

* clears the $ELTB entry word 1 (ID segment address) for programs that were abnormally terminated/aborted, e.g., due to PE, DMA violation, operator abort, etc.. This is indicated to the ABORT processor through a flag set in the ID segment if the programs had the Lock-on-Abort option set in the EQT lock request.

For all other programs, $EQCL additionally does the following:

* clears the rest of the $ELTB entry.

* elevates the priority of any buffered I/O calls to the locked EQT to zero, to insure they are all flushed out before any new ones. Any pre-lock, unbuffered calls have to contend for the EQT.

* calls scheduler $SCD3 to schedule any programs waiting for this EQT, or for a slot in the table.

* transfers the linked list of $XSIO requests in the entry just cleared back on the EQT, after the previous locking program's zero-priority calls.

* calls DRIVR to initiate the first $XSIO call reentered on the EQT, if the EQT was idle at the time.

* jumps to NOTRD for diagnostic processing, if DRIVR makes an error return, else

* returns to the caller.

$XSIO calls (I/O calls from the operating system) are handled slightly differently. In $XSIO, if the calling request is for a locked EQT, the request is queued in the list in the $ELTB entry for that EQT.

In $IOCL (in RTI04), if the request to be aborted is found neither in any EQT entry nor in any DRT entry, a check is made to see if it is in any $ELTB entry (word 3). If so, it is unlinked, leaving all others intact, before returning to $XEQ.

Further, every time an equipment is unlocked (this happens either when a program issues an unlock request or when a locking program is terminated normally not saving resources or abnormally not locking on abort), the $ELTB empty bit is updated, the list of scheduled requests for that EQT in $ELTB is reentered in the EQT for that device and calls are made to $SCD3 to reschedule all programs waiting in the general wait list flagged either on that device or on the

table itself (i.e., because the table was full).

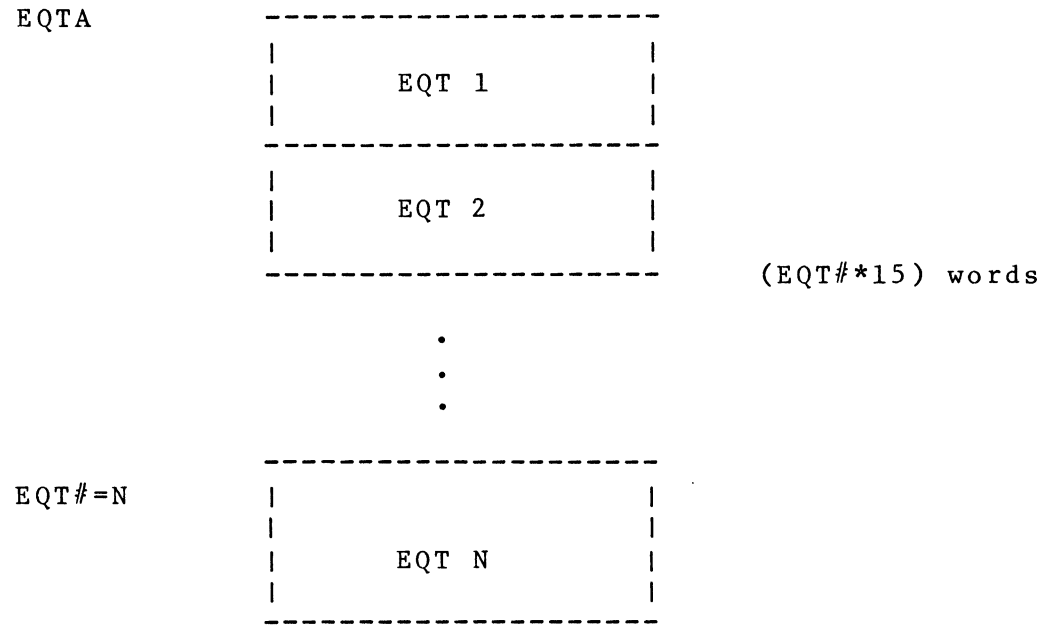


Figure 2-1. RTE-IVB Equipment Table

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														
2	R	Driver "Initiation" Section Address														
3	R	Driver "Completion" Section Address														
4	D	B	P	S	T	Unit #					Channel #					
5	AV	EQ TYPE CODE						STATUS								
6	CONWD (Current I/O Request Word)															
7	Request Buffer Address															
8	Request Buffer Length															
9	Temporary Storage for Optional Parameter															
10	Temporary Storage for Optional Parameter															
11	Temporary Storage for Driver															
12	Temporary Storage for Driver															
13	Temporary Storage for Driver															
14	Device Time-Out Reset Value															
15	Device Time-Out Clock															

Figure 2-2. RTE-IVB Equipment Table Entry

Where:

R = reserved for HP use
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 the I/O controller (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 DMA channel

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.

EQ TYPE CODE = type of device. When this octal number is linked with "DVCx," it identifies the device's software driver routine

CONWD = user control word supplied in the I/O EXEC call (see Chapter 3).

Figure 2-2. RTE-IVB Equipment Table Entry (continued)

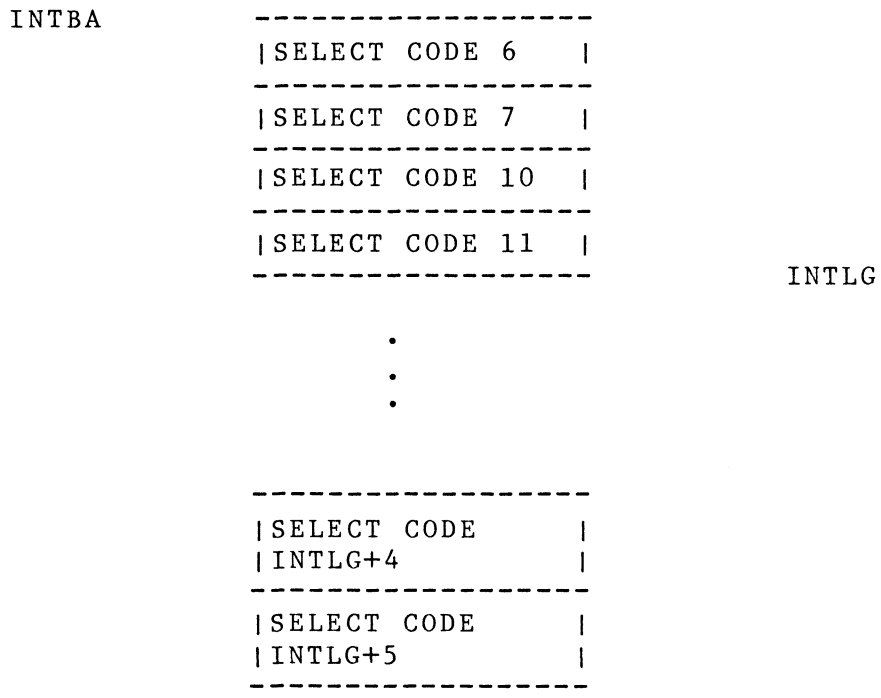


Figure 2-3. RTE-IVB Interrupt Table

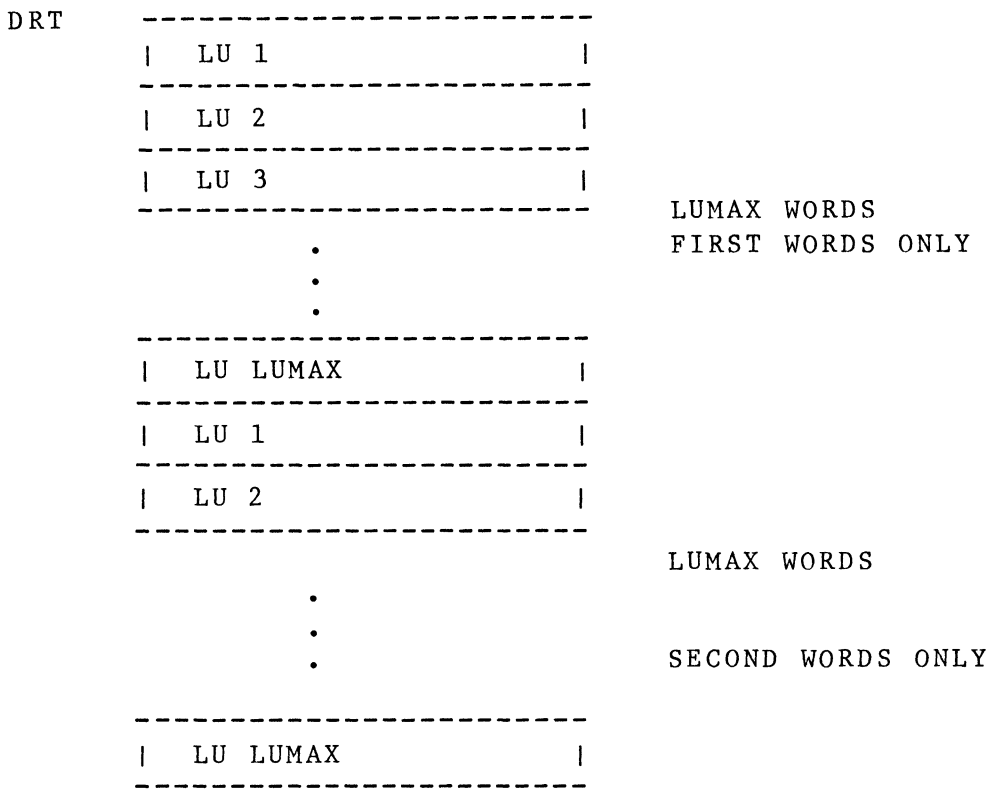


Figure 2-4. RTE-IVB Device Reference Table

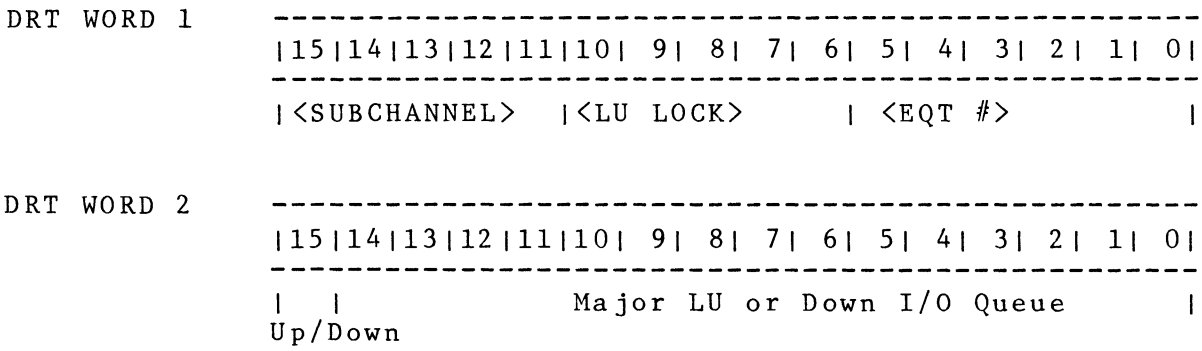


Figure 2-5. RTE-IVB Device Reference Table Entry

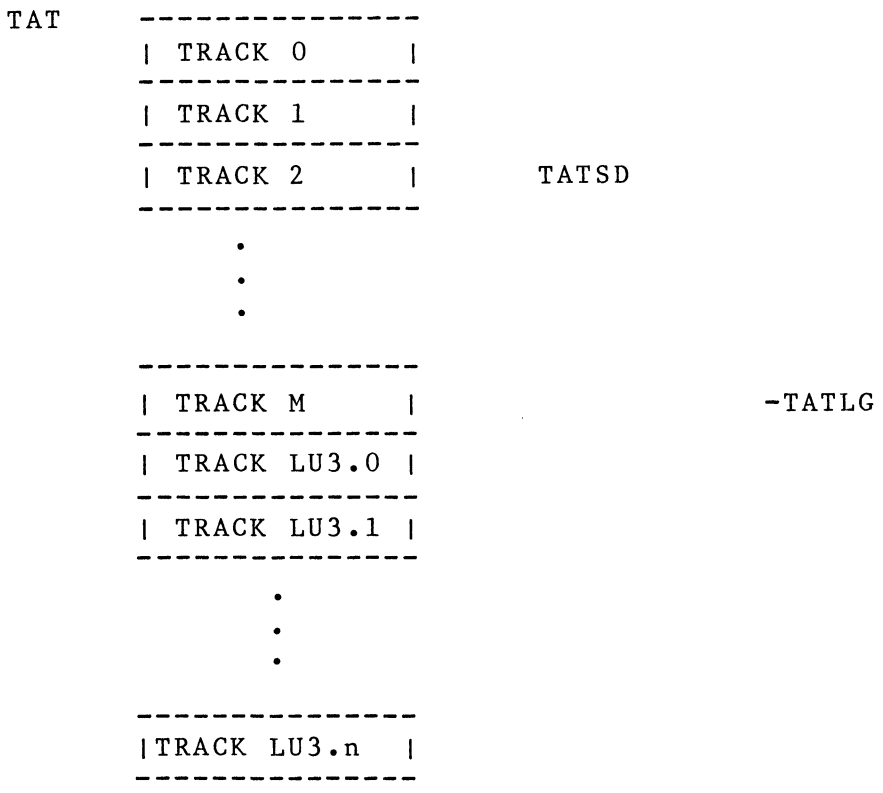


Figure 2-6. RTE-IVB Track Assignment Table

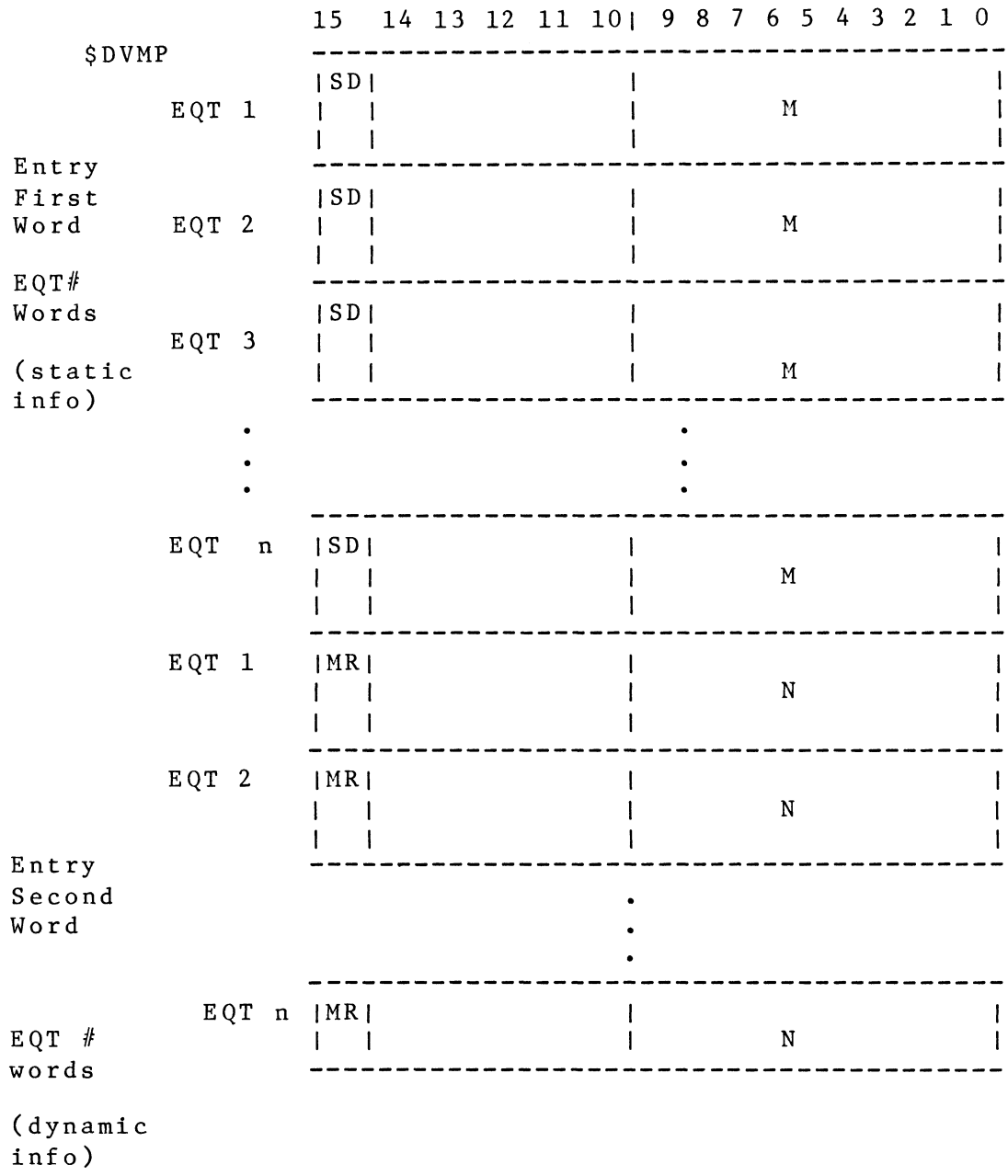


Figure 2-7. RTE-IVB Driver Mapping Table

```
     SD=0 Driver in driver partition and
          M=starting page number in bits 0-9

       =1 Driver in system driver area and
          M=0 not doing own mapping or
          M=1 is doing own mapping

     MR=0 Not memory resident program I/O,
          N=0 System I/O
          N=0 User I/O physical page number of base page
       =1 Memory resident program I/O

$DVPT = logical start page of driver partition
$DLTH = number of pages per driver partition
```

Figure 2-7. RTE-IVB Driver Mapping Table (continued)

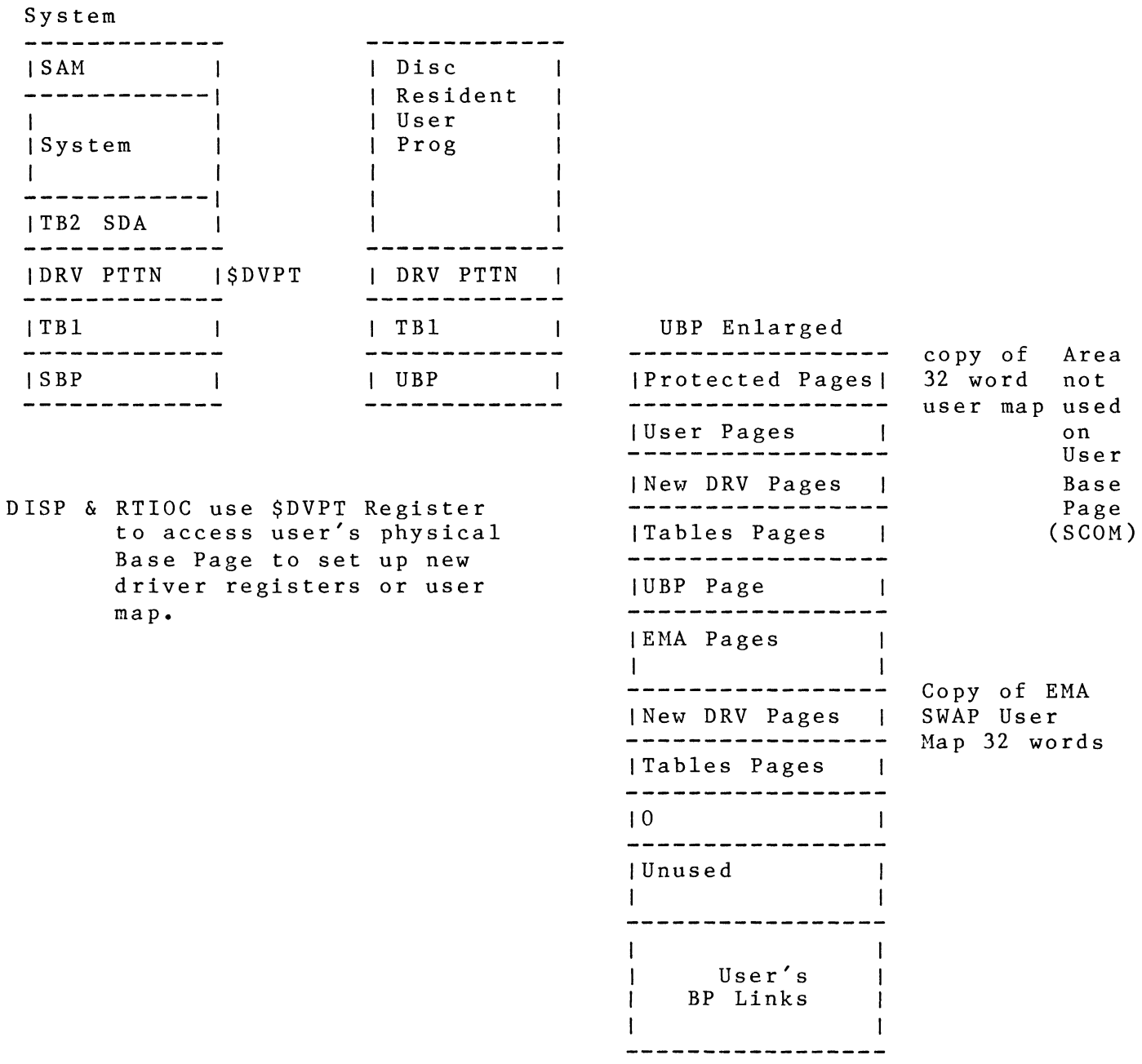


Figure 2-8. RTE-IVB User Base Page

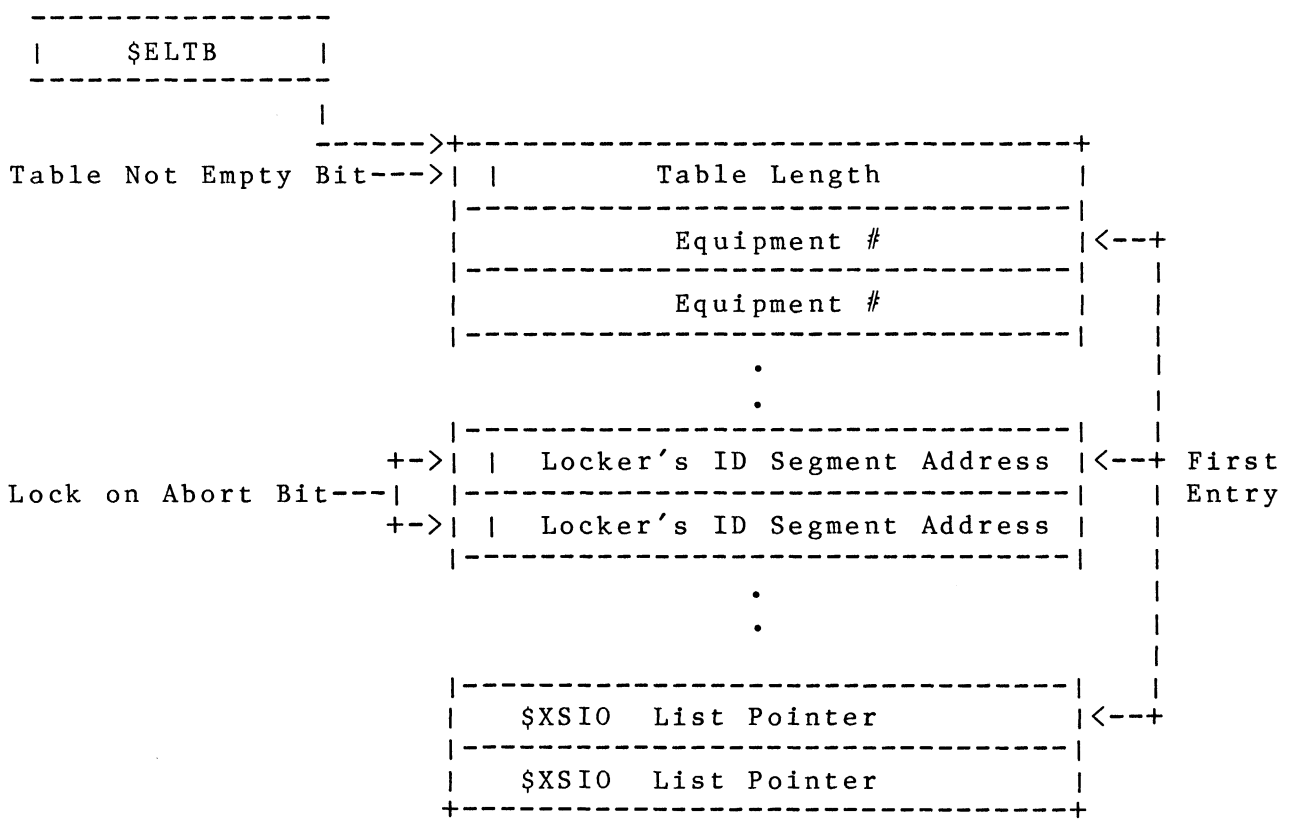


Figure 2-9. Equipment Locking Table

Chapter 3

EXEC AND $ALC

INTRODUCTION

This part of the Technical Specifications Manual deals with the EXEC and system available memory portion of the RTE-IVB Operating System. The EXEC is that portion of the operating system that checks for legality of all user EXEC requests, vectors legal requests to appropriate processors, vectors illegal requests to the abort processors, handles reentrant processing, and allows users to execute with the interrupt system off (privileged subroutines).

The $ALC portion of the system allocates System Available Memory (SAM) to system processors that request memory for buffer, tables, etc.

The EXEC modules contains five major sections:

1. System Request Analyzer (Memory Protect Violation Control)

2. Resident Library Execution Control (Dynamic Mapping Violation Control)

3. Privileged and Reentrant Subroutine Processors

4. Disc Track Allocation and Release Processors

5. General Error Message and Program Abort Processors

In order to understand how the system receives and handles an EXEC request, it is necessary to understand system memory protect and the rudiments of interrupt processing. The discussion below is a very brief description of interrupt processing with memory protect.

Suppose the user wishes to do output to the line printer from a high level language like FORTRAN. He may code something like:

```
CALL EXEC (2,6,IBUFR,IBUFL)
```

where the 2 is a Write Request, the 6 is the LU, IBUFR is the buffer to write, and IBUFL is the buffer length.

The FORTRAN compiler would change this to something like:

```
      JSB EXEC
      DEF RETRN   Return address
      DEF IWRIT   Address of Request Code
      DEF LU      LU to write to
      DEF IBUFR   Buffer Address
      DEF IBUFL   Buffer Length
```

RETURN-

When this code is executed the JSB EXEC will generate a memory protect. In fact any JMP, JSB, ISZ, STA, STB, DST, CBT, JLY, JPY, MVB, MVW, SAX, SAY, SBX, SBY, STX, or STY instruction which would either directly or indirectly affect a memory location below the MP fence will be inhibited and memory protect will force an interrupt to Location 5. The lower bound of protected memory is Location 2 the upper bound is set by the operating system with an OTA 5 (or OTB 5) where A is the address of the highest protected word.

Thus the JSB EXEC was never executed, rather the contents of trap cell 5 (the interrupting location) was executed. The contents of trap cell 5 is a JSB $CIC,I. This now allows us to enter the operating system into a module called RTIOC.

RTIOC is obliged to find out where the interrupt came from and what kind of interrupt it was. By executing a LIA 4 RTIOC will receive the interrupt code # of last interrupt. If the interrupt code corresponds to the Time Base Generator RTIOC jumps to $CLCK in the RTIME module. If the interrupt code is 5 (Dynamic Mapping, Memory Protect or Parity) RTIOC jumps to EXEC. If the interrupt code is anything else RTIOC uses the interrupt table to look up the appropriate processor.

If the interrupt was on interrupt code 5, then a LIA 5 (or LIB 5) will give the violation address; i.e., the address of the JSB EXEC.

Figure 3-1 shows a graphic representation of a JSB EXEC.

We now know how the system enters the EXEC.

The user tries to execute a JSB EXEC, memory protect catches this and instead executes the contents of trap cell 5. This causes an entry into the module RTIOC. RTIOC turns off the interrupt system analyzes where the request is to go and turns control over to the appropriate processor.

EXEC CALL PROCESSOR

The primary function of this section is to provide for general checking and examination of EXEC CALL requests (EXEC requests) and to call the appropriate processing routine.

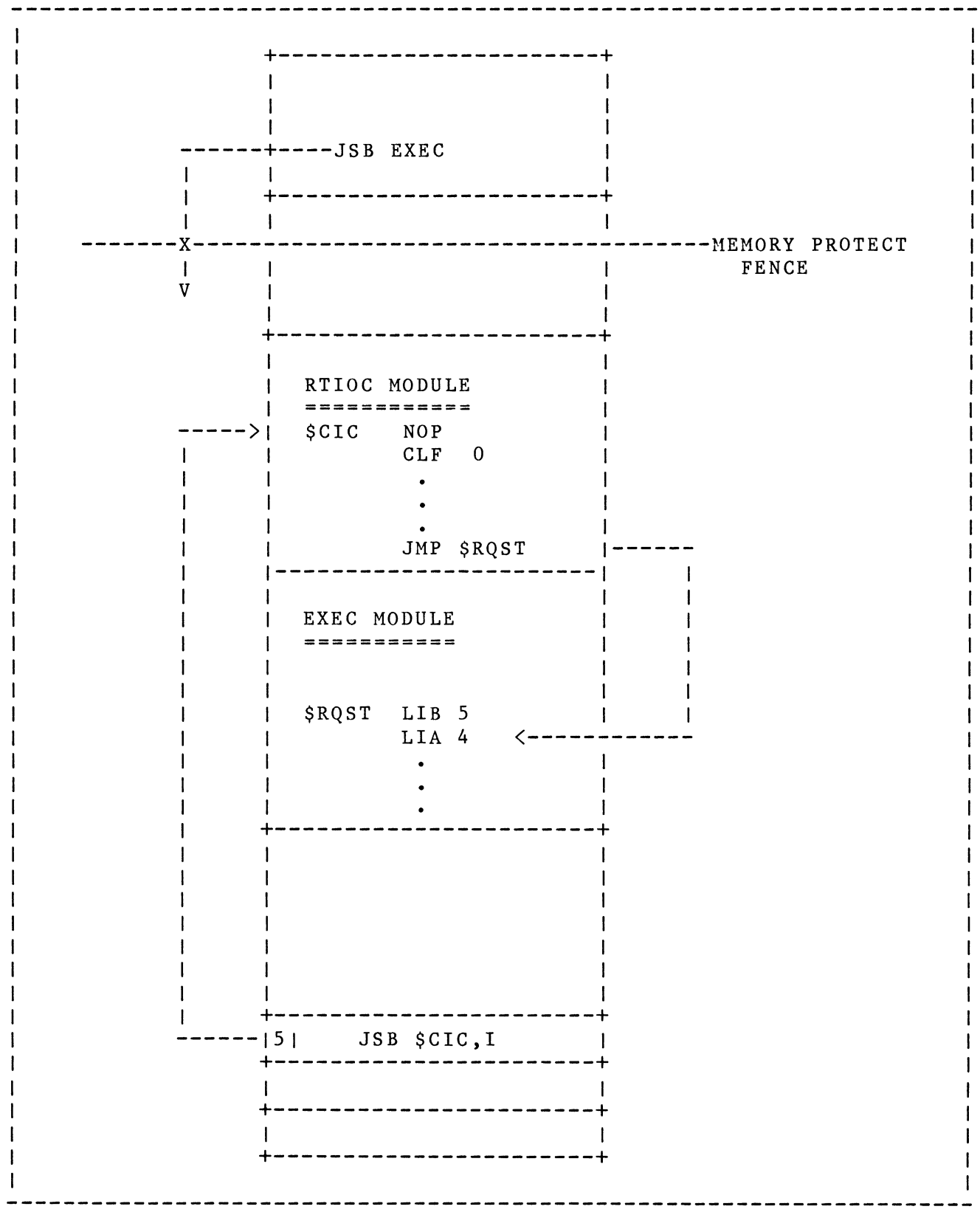


Figure 3-1. JSB EXEC

This section is called directly from the Central Interrupt Control (CIC) section (in RTIOC) when a memory protect (MP) or dynamic mapping violation (DM) is recognized. (All system requests from a user program cause a protect violation.) This section also determines non-legitimate protect violations in user programs such as executing halt or I/O instructions or attempting to write into a non mapped or protected area. It also recognizes user calls for resident library routines, reentrant, or privileged processing.

Upon entry from CIC EXEC must decide whether the violation was a true memory protect, parity error, or mapping violation. The EXEC request analyzer examines all memory protect and Dynamic Mapping violations. If the violation is legal, the EXEC jumps to the appropriate processor.

A DM violation is distinguished from a MP violation by executing a SFS 05 instruction. A DM error will set the flag on channel 05, a MP error will clear the flag.

Since parity error and memory protect share the same interrupt locations, it is necessary to distinguish which type of error is responsible for the interrupt. A parity error is indicated if, after the LIA (or LIB) 05 instruction is executed, bit 15 of the selected register is a logic 1; a memory protect violation is indicated if bit 15 is a logic 0. In either case, the remaining 15 bits of the selected register contains the address of the error location. Note, however, that parity errors are detected in RTIOC not EXEC.

Only one form of DMS violation is legal. This DMS violation will occur when a memory resident program tries to enter the memory resident library. The memory resident library is used only by memory resident programs. The physical address of the library will be above the memory protect fence if the program is using common; however, the pages containing the library are write protected. Thus any JSB, JMP, etc. to the library will cause a DMS violation. EXEC, after determining that the violation is a DMS violation, will check for three conditions. They are:

1. That the call is a JSB

2. That the destination is in the memory resident library

3. That the program is a memory resident program-Type 1

If any condition is not satisfied, EXEC aborts the offending program and issues a DM error message.

If all conditions are met, EXEC will jump to the routine if it is a privileged subroutine or jump to $RENT in the dispatcher if the routine is reentrant. $RENT does further processing for reentrant subroutines (such as resetting the memory protect fence).

If the violation was a true Memory Protect, EXEC looks at the destination. Only EXEC, $LIBR and $LIBX are legal destination addresses. All other destinations are flagged as errors and the offending program will be aborted with an MP error. (The abortion is only partially done within the EXEC.)

If the memory protect destination was EXEC then the EXEC further examines the request to see if the request is legitimate.

The EXEC checks to see if there are too many parameters, too few parameters, if the request itself is defined, or if the return address is illegal. EXEC also checks to see if any returned parameters would cause a store below the memory protect fence. This is done by using the NAMTB Table.

The NAMTB has one byte for each EXEC request. Each bit corresponds to a possible parameter. If the bit is set then the EXEC knows that the parameter is a possible store location and checks the address of the store. If the store address is below the memory protect fence, then the program will be aborted with a memory protect.

For example, consider the EXEC disc track allocation request:

```
JSB EXEC
DEF RETRN
DEF ICODE    ICODE=4 or 15
DEF #TRKS    # of tracks desired
DEF STRAK    Returned start track #
DEF DISC     Returned Disc LU
DEF SECT#    Returned Sectors per track
```

There would be a bit set for STRAK, DISC and SECT# because these are returned values which must not overlay memory below the memory protect fence. No check would be made for #TRKS as this is not a possible store location.

This is also how the EXEC decides if a read operation should be aborted.

That is, the store address (buffer location) would be below the memory protect fence.

The octal contents of the first 3 NAMTB table entries are shown below:

NAMTB Value	Request Code
000002	0/1 not used/Read
000000	2/3 write/control
007000	4/5 disc allocate/Release

Note the upper byte for word 3, the octal 7, it is there because the disc track allocate call returns the starting track number, disc LU, and sectors per track. The lower byte on word 1 has bit 1 set to indicate the buffer location is a storage location for a read request.

When EXEC decides that all the EXEC call parameters are okay, it jumps through the Request Code Table to the appropriate processor.

The request code table is a Table of EXEC request processor addresses. For processes external to EXEC the entry would be:

```
EXT $XXXX
DEF $XXXX+0
```

This gives the direct address of $XXXX for the JMP.

EXEC requests, entry points and the modules called are listed in Table 3-1.

Table 3-1. EXEC Requests and Entry Points

EXEC REQUEST	PURPOSE	ENTRY POINT	MODULE
1	I/O READ	$IORQ	RTIOC
2	I/O WRITE	$IORQ	RTIOC
3	I/O Control	$IORQ	RTIOC
4*	Local Disc Track Allocation	DISC1	EXECD
5*	Local Disc Track Release	DIS2	EXECD
6	Program Completion	$MPT1	SCMEDM
7	Operator Suspension	$MPT2	SCHEDM
8	Load Program Segment	$MPT3	SCHEDM
9	Program Schedule w/wait	$MPT4	SCHEDM
10	Program Schedule w/o wait	$MPT5	SCHEDM
11	System Time & Date	$MPT6	SCHEDM
12	Prog. Schedule after offset or Prog. Schedule at absolute time	$MPT7	SCHEDM
13	I/O Device Status	$IORQ	RTIOC
14	Get/Put String	$MPT9	SCHEDM
15*	Global Disc Track Allocation	DISCA	EXECD
16*	Global Disc Track Release	DISCD	EXECD
17	Class I/O READ	$IORQ	RTIOC
18	Class I/O WRITE	$IORQ	RTIOC
19	Class I/O Control	$IORQ	RTIOC
20	Class I/O Write/Read	$IORQ	RTIOC
21	Class I/O GET	$GTIO	RTIOC
22	Prog. Swapping Control	$MPT8	SCHEDM
23	Prog. Schedule w/WAIT & w/QUEUE	$MPT4	SCHEDM
24	Prog. Schedule w/o WAIT & w/QUEUE	$MPT5	SCHEDM
25**	Partition Status	$PTST	EXECD
26***	Memory Size Status	MEMST	EXECD

* The request is serviced in EXEC.

** This request has changed for RTE-IVB and is serviced in EXEC.

*** This request is new for RTE-IVB and is serviced in EXEC

Before transferring control to the appropriate processors, the EXEC places the address of all the request parameters in the base page as defined below:

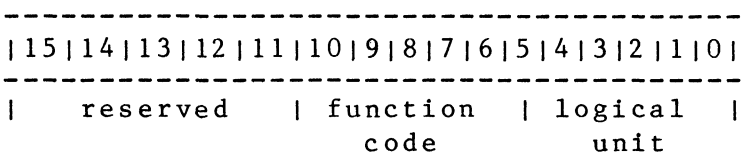
BASE PAGE ADDRESS	LABEL	USE
1676	RQCNT	Request Count=# of EXEC call parameters -1
1677	RQRTN	RETURN address of EXEC call
1700	RQP1	The REQUEST CODE of CALL
1701	RQP2	2nd Request Parameter
1702	RQP3	3rd Request Parameter
1703	RQP4	4th Request Parameter
1704	RQP5	5th Request Parameter
1705	RQP6	6th Request Parameter
1706	RQP7	7th Request Parameter
1707	RQP8	8th Request Parameter
1710	RQP9	9th Request Parameter

These base page locations will always be visable regardless of map. The contents, however, refers to addresses in the user map. The EXEC executes under control of the system map.

Extended EXEC*

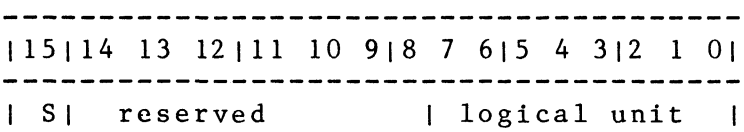
The following set of extended EXEC (XLUEX) calls will provide HP subsystems access to logical units greater than 63 (decimal). The XLUEX calls will have similar calling sequences (with EXEC) and identical functions. The only difference is in the definition of the control word (RQP2). XLUEX will use two words to specify the logical unit and control information while EXEC uses one.

EXEC control word:

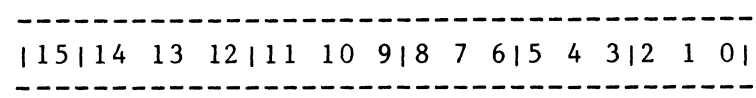


XLUEX expands the control word into Logical unit and function code parameters:

XLUEX - Logical Unit word



- Function Code word



The S bit, if set, will inhibit the session or batch switch table mapping (i.e., the logical unit number supplied is the logical unit number to be used).

NOTE This capability will not be documented for the user until all supported HP subsystems have been modified to enable access of the full range of logical unit numbers. Until that point in time, the user must access logical units > 63 via the session switch table.

CAUTION! This implementation may be a temporary solution as future projects may alter the external characteristics of these calls.

The following functions will be supported by XLUEX calls:

Read, write, control, status, class read, class write, class, write/read, class control.

Calling sequence: (refer to the RTE-IVB Programmer's Reference Manual for Parameter details - other than those previously discussed.

```
READ/WRITE

EXT XLUEX

JSB XLUEX
DEF EXIT
DEF RCODE          (READ=1, WRITE=2)
DEF CONWD          Note:  New or changed (2 word parameter)
DEF BUFFR
DEF BUFFL
DEF DTRAK          optional
DEF DSECT          optional
```

Exit

```
      CONTROL:

      EXT XLUEX

      JSB XLUEX
      DEF RTN
      DEF RCODE          (control =3)
      DEF CONWD          Note:  New or changed (2 word parameter)
      DEF IPRAM          optional
```

Rtn

```
      STATUS:

      EXT XLUEX

      JSB XLUEX
      DEF RTN
      DEF RCODE          (Status=13)
      DEF LU             Note:  This word contains the logical unit only
      DEF ISTA1
      DEF ISTA2          optional
      DEF ISTA3          optional
```

Rtn

```
      Class Read or Write of Write/Read

      EXT XLUEX

      JSB XLUEX
      DEF RTN
      DEF RCODE          (Read=17,write=18,write/read=20)
      DEF CONWD          Note:  New or changed (2 word parameter)
      DEF IBUFR
      DEF IBUFL
      DEF IPRM1          optional
      DEF IPRM2          optional
      DEF Class          optional

      CLASS I/O Control

      EXT XLUEX

      JSB XLUEX
      DEF RTN            (Class Control=19)
      DEF RCODE
      DEF CONWD          Note:  New or changed
      DEF IPRAM
      DEF ICLAS
```

LIBRARY EXECUTION CONTROL

The Relocatable Library contains the set of subroutines required for floating point operations, intrinsic functions, FORTRAN run-time processors and general utility functions.

A program in this library is structured in one of the following formats:

1. Re-entrant: (Type 6) During the execution of the routine, it may be suspended and entered again by a call from a higher priority program. Subroutines in this format may not modify in-line code (i.e., they are "read only" and all temporary variables must be grouped into a block within the program). This block is termed the "Temporary Data Block",TDB. The execution time of a reentrant routine is usually greater than 1 millisecond. In RTE-IVB only those reentrant subroutines loaded into the memory resident library and called from Memory Resident programs or those subroutines loaded into SSGA can be reentered by different programs.

2. Privileged: (Type 6) A routine in this format is permitted to run with the interrupt system and memory protect disabled. A subroutine of this type should have an execution time of less than 1 millisecond. It also may not incorporate input/output calls, nor may it call a reentrant routine.

3. Utility: (Type 7) This classification is used for programs containing I/O functions or other features which do not allow reentrant or privileged structure. Examples of this type are the FORTRAN runtime routines PAUSE and STOP. There are no restrictions on internal program structure or features. The subroutine will always be appended to the end of the user's program.

RESIDENT LIBRARY SUBROUTINES

The resident library consists of those subroutines referenced by memory resident programs. Should that subroutine reference another subroutine, the second subroutine will also become part of the memory resident library. The memory resident library is shared by all memory resident programs. The sharing prevents commonly called subroutines from being appended to each memory resident program that calls it, thus affecting a conservation of memory for memory resident programs. Note that the resident library is created at generation time and that all routines which are loaded into the resident library are also put in the relocatable library for disc resident programs. Since the subroutine is shareable it should be written in a privileged or reentrant format.

UTILITY AND SINGLE-USER LIBRARY PROGRAMS

A "utility" subroutine can be called by only one user program. Therefore, a copy of the utility program is appended to the absolute version of any user program which references it. All programs in reentrant or privileged format are reclassified as utility if they are not included in the Resident Library by RTGEN. A copy of each subroutine is appended to each disc-resident user program which references it. (Thus all type 6 routines put in at generation become type 7 after generation.)

All library type subroutines entered when the system is generated are reclassified as "utility" and stored in packed relocatable format on the disc for use by the LOADR in loading programs on-line.

Users who wish to write subroutines which can be loaded into the memory resident library to be shared by memory resident programs should refer to Appendix C for the required format.

Reentrant and privileged subroutines require special pre and post processing. This processing is done by the routines $LIBR and $LIBX. The format is shown below. The code below the dotted line is needed for reentrant routines only.

```
            EXT    $LIBR,$LIBX
ENTRY       NOP
            JSB    $LIBR
            DEF    TDB (or "NOP" if privileged)
            ---    First program instruction--
             -
             -
             -     body
             -     of
             -     program
             -
EXIT        JSB    $LIBX
            DEF    TDB (or DEF ENTRY if privileged)
-------------------------------------------------------------------
            DEC    N Return adjustment for reentrant
                     (Return=N + ENTRY)
TDB         NOP      Holds linkage to previous block
            DEC    K Total Length of TDB in words
            NOP      Holds return address of call
             -       -Blocks used
             -        for temporary
             -        storage of values
             -         generated by the program
```

The TDB (Temporary Data Block) and return adjustment is only for reentrant format. The return adjustment for reentrant format in the exit call is used to vary the return point to the calling program. The return address and return adjustment are added to determine the final return address.

The parameter following the JSB $LIBR (DEF TDB, or NOP) identifies the subroutine format to the system and the type of processing that is required. A NOP signifies a privileged subroutine.

Reentrant programs may call other reentrant and privileged programs. However, privileged programs may only call privileged programs.

The JSB $LIBR is intercepted by EXEC because it causes a memory protect.

PRIVILEGED and REENTRANT PROCESSING

Privileged or reentrant processing starts whenever the initial memory protect or DMA violation for that service is detected. This can happen in two ways.

Consider the two cases below:

CASE 1 ANY PROGRAM

```
          .
          .
          .
          JSB    SUB
          .
          .
          .
     SUB  NOP
          JSB    $LIBR
          NOP
```

CASE 2 MEMORY RESIDENT PROGRAM

```
          .
          .
          .
        JSB  SUB
          .
          .
          .
```

```
+-------------------------------------------------------------+
|                                        MEMORY RESIDENT LIBRARY |
|                                                             |
|          SUB NOP                                            |
|              JSB $LIBR                                      |
|              NOP                                            |
|                                                             |
|                                                             |
+-------------------------------------------------------------+
```

In Case 1 all code is within the users program. The JSB $LIBR causes the memory protect. As mentioned earlier $LIBR is a valid memory protect and thus the system starts the privileged or reentrant run.

In Case 2, however, a DM violation resulted due to the JSB SUB. This is because SUB resides in the memory resident library. Here the privileged or reentrant run started at the JSB SUB. EXEC places the return address (P+1 of JSB SUB) into SUB, that is, it simulates the JSB instruction and eventually returns control to three words past the SUB NOP (i.e., the target of the JSB). In this case the JSB $LIBR was never executed.

As can be seen from Case 2 all subroutines that are loaded into the memory resident library (type 6 subroutines) must be in the privileged or reentrant format.

EXEC examines the word (P+1) following the JSB $LIBR. If (P+1)=0 (NOP), the called subroutine is "privileged". $LIBR restores the registers, adds 1 to "$PVCN" (privileged subroutine nest count), leaves the interrupt system disabled, (which also means MP disabled) and transfers control to the word following the $LIBR call (i.e., P+2). The return address to the program (P+1) of the JSB SUB is stored in the entry point of the library subroutine if a protect violation occurred on the original call.

If the (P+1) of the JSB $LIBR is non-zero, the value is the address of the Temporary Data Block of the reentrant subroutine. The first word of the TDB is checked. If it is zero, then the subroutine is not being reentered.

The first word is then set up to point to the second word of a 4 word block of memory set up for each JSB $LIBR used in a reentrant run. This block is located in system available memory (SAM). The contents of this second word is the ID address of the program using the TDB. (More discussion on this reentrant list structure will be found in the following sections. Referencing to the list structure in Appendix B at this time should help in understanding the discussion below.)

If the link word is non-zero, the subroutine is being reentered (i.e., two memory resident programs want the same subroutine) and $ALC is called by the EXEC MTDB routine to allocate a block in available memory equal to the length of the TDB (word 2). If $ALC rejects the allocation request, the main user program is suspended and linked into the memory suspend list.

If the block is allocated, the TDB is moved to the new block. If the new block is one word longer than requested (refer to discussion on $ALC), word 2 (word length of TDB) in the new block is set negative as a flag. The first word of the moved TDB in the system map is changed to point to the first word of the original TDB in the user map.

The address of the original program call is set in word 3 of the program TDB as the return address. The reentrant program must not modify the first three words of the TDB. EXEC then calls $RENT in the dispatcher who sets the memory protect fence to the beginning of the Resident Library area, removes DMS write protect, and restores the program registers. The interrupt system is enabled, memory protect turned on, and control transferred to the program.

For privileged subroutines the system saves all registers going into the subroutine and restores them when the subroutine starts to execute. With nested privileged subroutines the system does not save the registers on the 2,3,4, etc., call but neither does the system destroy the registers. That is, the A,B,Y,X,E and 0 registers may be used to pass parameters to and from privileged subroutines (and reentrant subroutines).

The return to the main program at the end of a reentrant or privileged subroutine is performed by a JSB $LIBX. The execution of this instruction is executed directly if a privileged program is executing; it causes a memory protect violation if a reentrant program is executing. In the latter case, EXEC transfers control to $LIBX indirectly after the initial protect violation processing.

If the executing program is privileged (i.e., $PVCN>0), one is subtracted from $PVCN. If $PVCN is still non-zero, control is returned directly, with registers restored, to the return point in the calling privileged program. If now $PVCN=0, control is returned to the caller with the interrupt system enabled and the memory protect fence set to the beginning of the area of the original calling program.

If the executing program was reentrant the return address is calculated by adding the contents of the third word of the TDB which contains the P+1 of the original JSB SUB and the P+2 of the JSB $LIBX which may contain a return adjustment. This address is placed into the ID segments point of suspension. In addition, the necessary adjustments are made to the reentrant list and to system available memory. This structure is discussed below.

All $LIBR calls require an associated $LIBX call.

REENTRANT LIST STRUCTURE

Every reentrant call requires the creation of a 4-word table in system available memory called a reentrant table. All of these tables are connected through a list structure with its head in the EXEC (DHED) (the reentrant list). The list is a two dimensional list. The first dimension is a stack and is one entry per program. The second dimension is for programs that make nested reentrant calls and is a push down stack after the first entry (i.e., the one that got the program in the list in the first place.

The purpose, structure, and content of this reentrant ID list is graphically documented in Appendix B.

FORMAT OF REENTRANT SUBROUTINE LIST

The reentrant Table is a 4 word table in system available memory that is allocated every time a reentrant call is made (i.e., one for every reentrant JSB $LIBR).

Word	Purpose
1	Link to next 4 word block (0=End of List)
2*	ID address of user making this reentrant call
3**	Pointer to TDB buffer in reentrant subroutine
4	Used if one reentrant subroutine calls another. It points to next 4 word entry for this program.

* Sign Bit set if K+1 words of SAM allocated instead of K words asked for.

**Sign Bit of this word is set if TDB has been moved to system available memory. If sign bit set, pointer points to moved TDB in SA

The reentrant structure is also used to allow buffered input and output. The $REIO routine in EXEC is called by RTIOC (is never called by EXEC itself) anytime I/O is done in a reentrant subroutine. For example, the FORTRAN callable REIO routine; i.e., CALL REIO (I,LU, BUFR,BUFRL) does I/O from a reentrant subroutine and causes entry into $REIO.

Consider the case of normal (unbuffered) input. Since the input from the peripheral device is being placed within the program area itself, that program will be I/O suspended and unswappable. The program cannot be swapped because I/O is being done to a particular part of memory. If another program were placed there that area of the other program would be overlayed by the incoming data. Thus the unbuffered input has caused a lock of that partition meaning no other program can use it. The case of normal output is the same, an unusble partition for the length of the I/O.

This problem can be avoided by doing I/O from a reentrant subroutine where the I/O buffer is wholly within the TDB itself. $REIO is called from RTIOC anytime I/O is done from a reentrant subroutine. $REIO looks to see if the program has an ID Reentrant TAG (i.e., is it really reentrant and has done a JSB $LIBR) if so it then looks at the buffer address and length. If the entire buffer is within the TDB then $REIO has MTDB call $ALC for TDB space in system available memory, sets the $MVBF flag in RTIOC to the negative of the TDB address, and returns to RTIOC. RTIOC then knows that the buffer is not in the program area and this then makes the program swappable and frees the partition for other uses.

The process for output is essentially the same. The output buffer within the TDB is moved to S.A.M. and $REIO gives RTIOC the negative new address in $MUFB which will be outside the program area. The program may then continue to execute because all the data is outside the program area.

DISC TRACK ALLOCATION PROCESSORS

DISC TRACK REQUESTS

The system maintains complete control over the allocation and ownership of the system (LU2) and auxiliary disc (LU3) tracks. User programs, through EXEC requests, can allocate tracks to themselves (local) or allocate tracks for general use by anyone (global). User programs can also release the tracks back to the available pool via EXEC requests.

A track, if allocated to a program, is such that only that program which requested it can write on it and/or release it. Any program can read from it.

A global track is such that any program can read from it, write on it, and/or release it.

Track control is maintained via the Track Assignment Table (TAT). Peripheral discs (NOT LU2 or LU3) are not managed through the track assignment table.

Figure 3-2 shows the structure of the system disc (LU2). The system disc has three distinct areas. The first area, from track 0 to approximately track 20 (this area will vary depending on the size of the system, 15 to 40 tracks is typical) is the system area of the disc. The virgin copy of the operating system, drivers and all user programs loaded at generation time are stored in this location.

The second area from approximately track 20 to track 100 is the track pool or scratch area of the disc. The upper boundary of this area is determined the first time a generated system is booted up. The boundary is set by the File Manager initialize command. (IN, Master sec code, -LU, cartridge ref., label, start track, # of tracks).

The Track Pool is used by the system for swapping, text editing, loading permanent program additions, etc. There must be a minimum of 8 track pool tracks on LU2, however, a minimum of 70 track pool tracks is recommended.

If the Extended memory Feature of RTE-IVB is being used more track pool area may be necessary to allow swapping of large arrays. The additional space needed can be gauged by recalling that one disc track contains space for 6144 words.

The third area of the system disc is for user files. The File Manager maintains this area.

An auxiliary disc (LU3), Figure 3-3, can be used with RTE to extend the size of the track pool if desired.

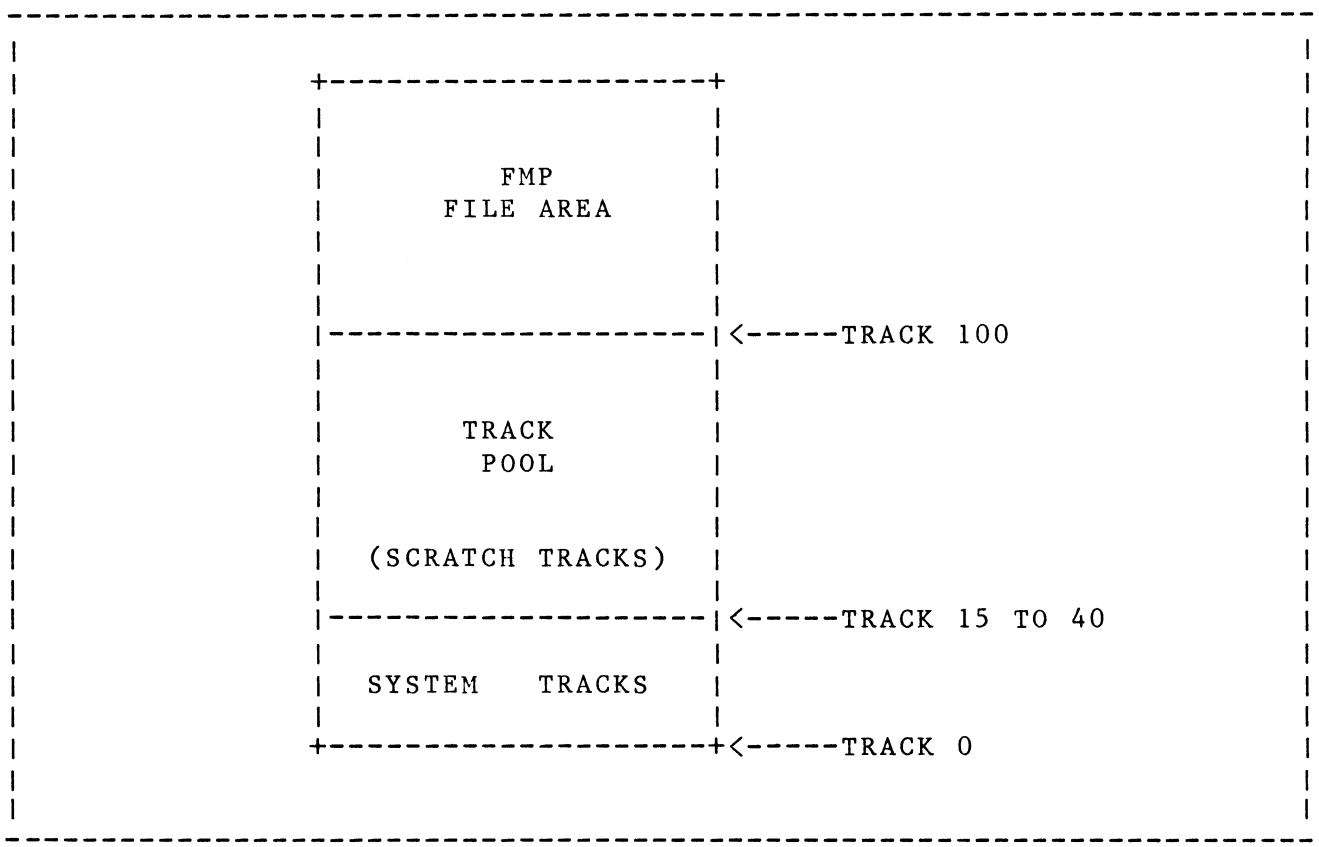


Figure 3-2. LU2

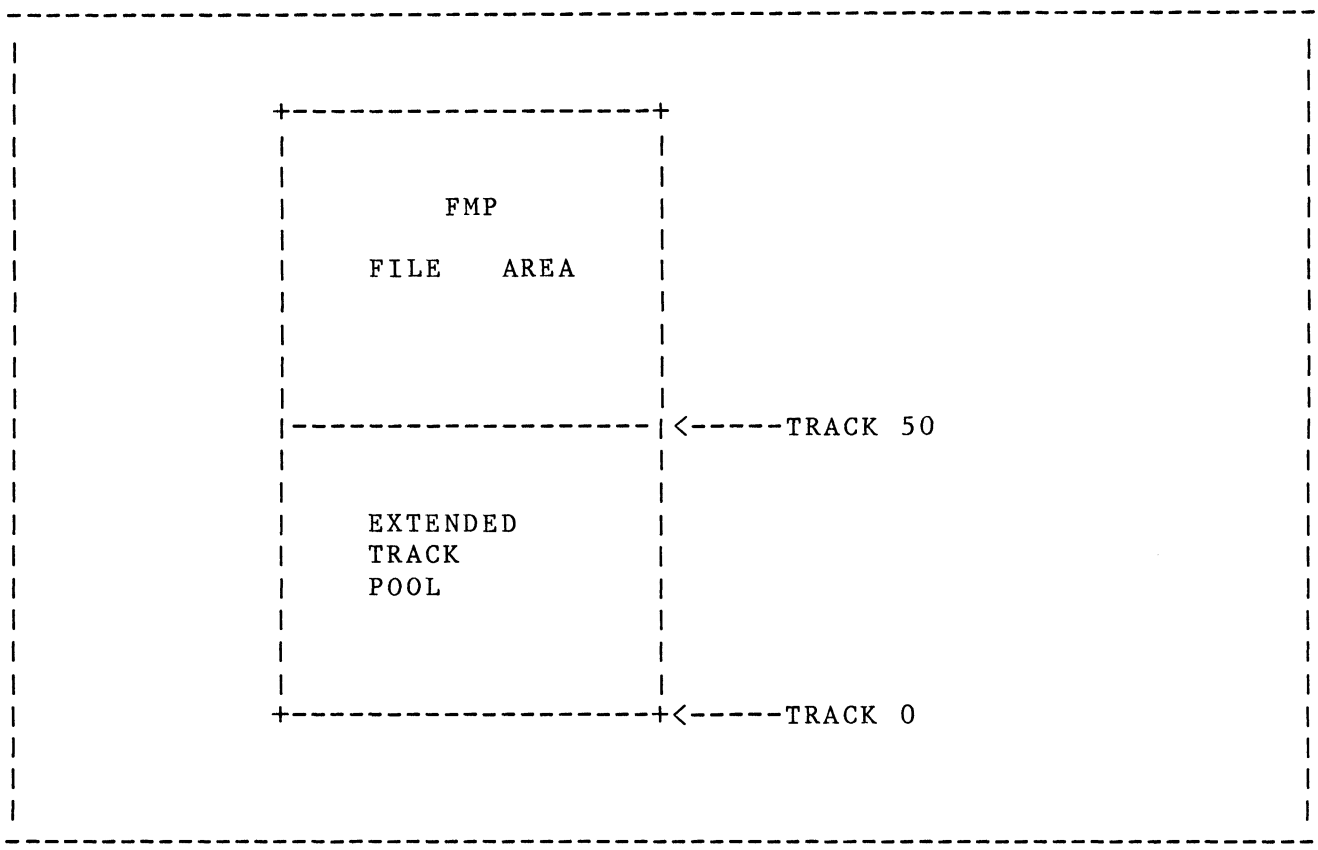


Figure 3-3. LU3

TRACK ASSIGNMENT TABLE (TAT)

The TAT is a variable length table describing the availability of each disc track on the system and auxiliary discs. The TAT is constructed by "RT4GN" based on user parameters declaring the size of the system disc and the availability and size of an auxiliary disc. Each track is represented by a one-word entry. The first words of the table correspond to the "n" tracks of the system disc. The word "TATSD" in the Base Page Communication Area contains the size of the system disc as a positive integer. If an auxiliary disc is included, the rest of the TAT contains one-word entries to describe the tracks on that disc.

"RT4GN" initializes the protected tracks of the system disc to be assigned to the system (permanently unavailable).

The contents of a track assignment entry word may be one of the five values:

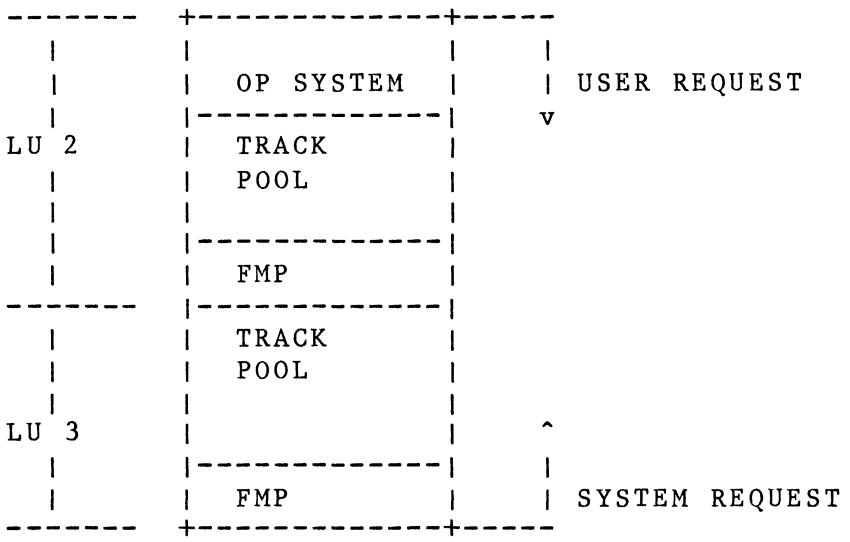
CONTENTS OF TRACK ASSIGNMENT TABLE

Contents	Meaning
0	Available
100000	Assigned to System (or protected)
077777	Assigned globally (anybody can write)
077776	Assigned to FMGR (FMP Package)
XXXXXX	ID segment address of owner

Base Page Words Used for Track Assignment

BP Word	Name	Purpose
1656	TAT	FWA of Track Assignment Table
1755	TATLG	NEGATIVE length of Track Assignment Table
1756	TATSD	# of Tracks of System Disc
1757	SECT2	# of Sectors/Track on System Disc (LU2)
1760	SECT3	# of Sectors/Track on Aux Disc (LU3)

Graphically the TAT is searched as shown below:



From the diagram above, the user can see how to optimize system search time for free tracks. No FMP area (or a very small area) on LU3, 8 tracks of track pool on LU2 (minimum required) will optimize system search time for system tracks. The user can also improve system performance physically by putting LU2 and LU3 on separate physical discs.

LU2 and LU3 are both limited to a maximum of 256 tracks.

ERROR MESSAGE PROCESSOR

The EXEC will detect five classes of errors Memory Protect (MP), Dynamic Mapping (DM), Request Code (RQ), Reentrant Subroutine errors (RE), and Parity ERRORS (PE).

All of these errors will cause program abortion (even if the no abort bit is set). The error message and the error is discussed below:

MEMORY PROTECT

In RTE-IVB the operating system is protected by a hardware memory protect. This means that any 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 it's legality. If the memory protect is illegal, then the program is aborted and the following message is reported to the system console:

```
MP INST = XXXXXX          XXXXXX = OFFENDING OCTAL INSTRUCTION CODE
ABE PPPPPP QQQQQQ R       CONTENTS  OF  A, B & E REGISTERS AT ABORT
XYO PPPPPP QQQQQQ R       CONTENTS  OF  X, Y & O REGISTERS AT ABORT
MP YYYYY   ZZZZZ          YYYYY = PROGRAM NAME
                          ZZZZZ = VIOLATION ADDRESS
YYYYY ABORTED
```

DYNAMIC MAPPING VIOLATION

A dynamic mapping violation occurs when an illegal read or write occurs to a protected page of memory. This may happen when one user tries to write beyond his own address space to non existant memory or someone elses memory. In this case the program is aborted and the following message is printed:

```
DM VIOL = WWWWW           WWWWW = CONTENTS  OF  DMS  VIOL  REGISTER
DM INST = XXXXX
ABE PPPPPP QQQQQQ R
XYO PPPPPP QQQQQQ R
DM YYYYY  ZZZZZ
YYYYY ABORTED
```

EX ERRORS

It is possible to execute in the privileged mode (i.e. interrupt system off) in this case the user may not make EXEC requests because memory protect, which is the access vehicle to EXEC is off. An attempt to make an EXEC call with the interrupt system off will cause the calling program to be aborted and the following message printed:

```
EX   YYYYY ZZZZZ
EX   ABORTED
```

This error is detected in $TBl. The error is detected by virtue of the fact that EXEC was entered directly instead of causing a Memory Protect.

UNEXPECTED DM AND MP ERRORS

The operating system handles all MP and DM violations. Certain of these violations are legal and others are not. In any case the operating system associates these violations with program activity. If a DM or MP error occurs and no program was active then, this is an unexpected MP or DM violation. Since no program is present, there is no program to abort in this case the following message will be printed:

```
DM VIOL = WWWWW
DM INST = XXXXX          OR          MP INST = XXXXX
ABE PPPPPP QQQQQQ R                  ABE PPPPPP QQQQQQ R
XYO PPPPPP QQQQQQ R                  XYO PPPPPP QQQQQQ R
DM <INT>     0                       MP <INT> = 0
```

```
** WARNING  **  WARNING  **  WARNING  **  WARNING  **  WARNING  **
================================================================
```

The above message which specifies <INT> as the program name is a signal to the user that an unexpected memory protect or dynamic mapping violation error has occurred. This is a serious violation of OP system integrity. Most times it means user written software (driver, privileged subroutine) has damaged the operating system integrity or inadequately performed required (driver) system housekeeping. It may also mean that the CPU has failed and that the operating system caught the failure in time to avoid a system crash.

If this error occurs it is suggested that users save whatever they were doing (i.e., finish up editing, etc.) and reboot the system. If only HP system modules are present in the operating system, CPU failure is highly suspected and CPU diagnostics should be run.

SYSTEM AVAILABLE MEMORY (SAM)

Reentrant subroutine ID tags, reentrant I/O, automatic buffering to I/O devices, and many other operating system features require blocks of memory to be made available at any time. In order to satisfy these temporary needs for memory an area of memory was set aside and called system available memory (SAM). Two routines manage SAM. The routine $ALC allocates memory to the requestor and the routine $RTN returns memory no longer needed back to SAM.

SAM is allocated in contiguous chunks of memory and is maintained via a list of available contiguous chunks. Over the course of time memory will be given away and returned many times. Memory that is returned is checked to see if it is contiguous from above or below to any existing free memory. If not it is linked to the currently existing free memory. The link structure uses the first two words of the chunk returned for the linkage. The first word is the number of words in that block and the second word contains the address of the first word of the next available free chunk of memory. If the returned memory is contiguous to an existing block then the returned memory is concatenated by just updating (or creating) the two word linkage at the beginning of the block to reflect the fact that the new block length is greater.

$ALC allocates memory to the caller by giving that caller the amount of memory requested the first time it finds that much memory in a free block. No best fit algorithm is used as it has been found that best fit routines are too slow and wasteful of CPU time. Due to the way $ALC is linked, it can happen that the user will ask $ALC for N words and instead get N+1. This happens when a request for N words would only leave 1 word of system available memory left over in a queue block. Since $ALC requires 2 words for its link structure and only one word would be left, $ALC gives the other word to the user to force him to keep track of it. Appendix B also shows how this one extra word is carried along if the need arises. It is the users responsibility to detect this condition and return the extra word when $RTN is called. As mentioned memory is allocated in contiguous chunks; however, $ALC is written so that SAM need not be contiguous memory. The disconnected blocks of memory are linked through the first two words of each block. A drawing of the linkage for RTE is shown in Figure 3-4 so that the reader will understand how the routine will work in the general case.

If a block size request comes into $ALC and the size requested is larger than any currently contiguous free block, then $ALC returns a flag to this effect. The calling routine is obliged to check for this condition and may place the program, on whose behalf the request was made, into the memory suspend state (state 4) via a $LIST call. If a program does go into the memory suspend list, then the number of words requested must also be posted into the second word of the ID segment.

On all $RTN calls a check is made of the suspend list after the memory has been added to SAM. If enough contiguous memory has become available to satisfy the highest priority program in the list (i.e. the first one in the list), then $LIST is called for every program in the suspend state until the end of the list or until a request length is found that is greater than the currently existing largest block of SAM. For example, if programs A and B are in the suspend list with priorities of 10 and 20 respectively but with block requests of 1000 and 100 respectively B will never be rescheduled until enough memory has been collected for A. The philosophy here is that he who has the highest priority should get resources first. Note, however, that any future $ALC requests that come in will be honored if there is enough memory. This allows programs of lesser or greater priority to continue and hopefully give block memory at a later date.

Calling Sequences:

```
1)  $ALC      (Allocate section)
    (p)       JSB $ALC
    (P+1)     (# words needed)
    (P+2)     -Return-
```

On return:

(A) = FWA of allocated block, or = 0 if reject
(B) = # words allocated (may be 1 greater than # requested)

If no block is large enough to alloctae the requested length,
(A) = 0 on return.

```
2)  $RTN      (Return block section)
    (p)       JSB $RTN
    (P+1)     (FWA of buffer)
    (P+2)     (# words returned)
    (P+3)     -Return:  Registers meaningless-
```

There are no error conditions detected by these sections.

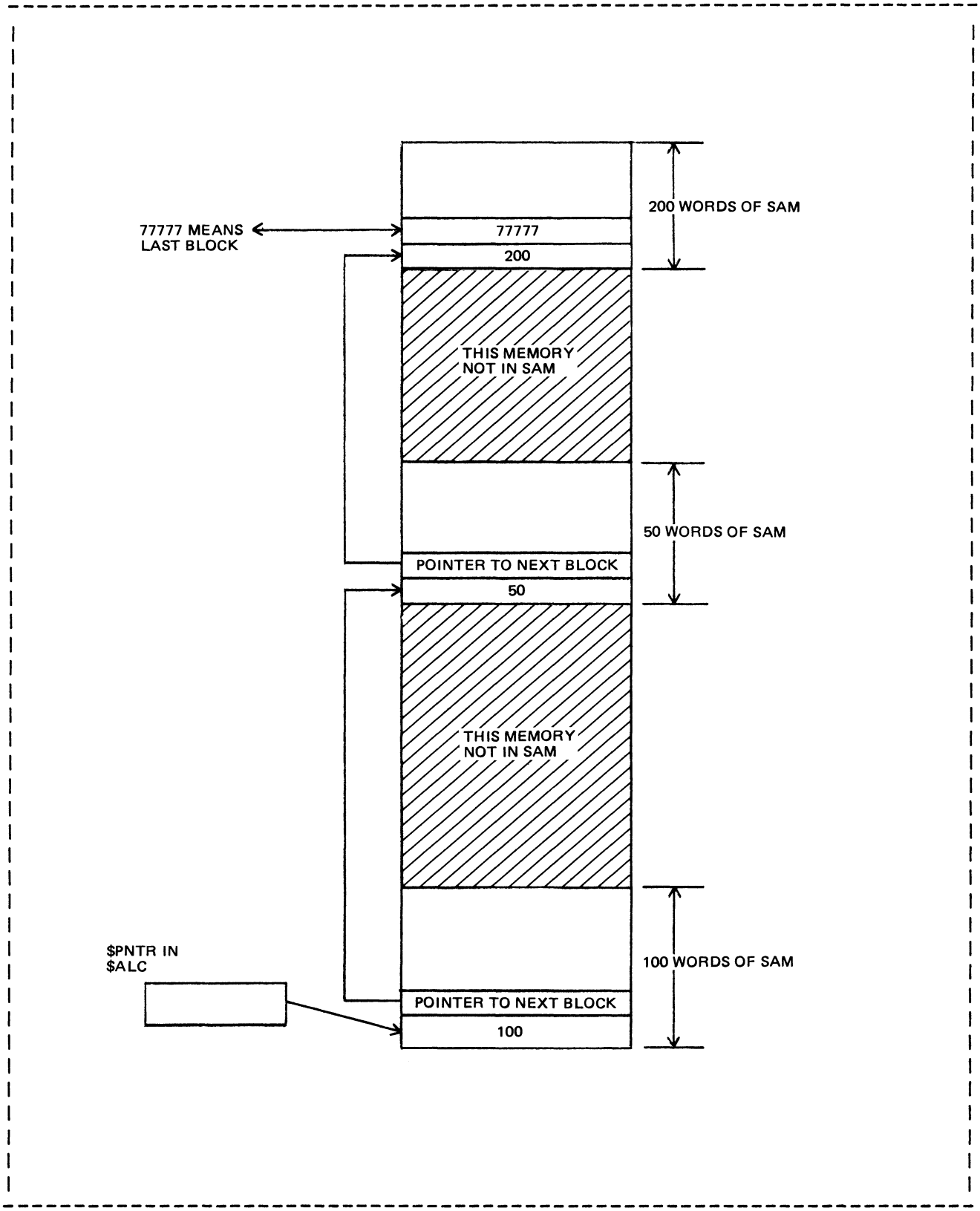


Figure 3-4. Example of SAM Linkage

Now suppose the user returns 35 words. See what SAM now looks like in Figure 3-5.

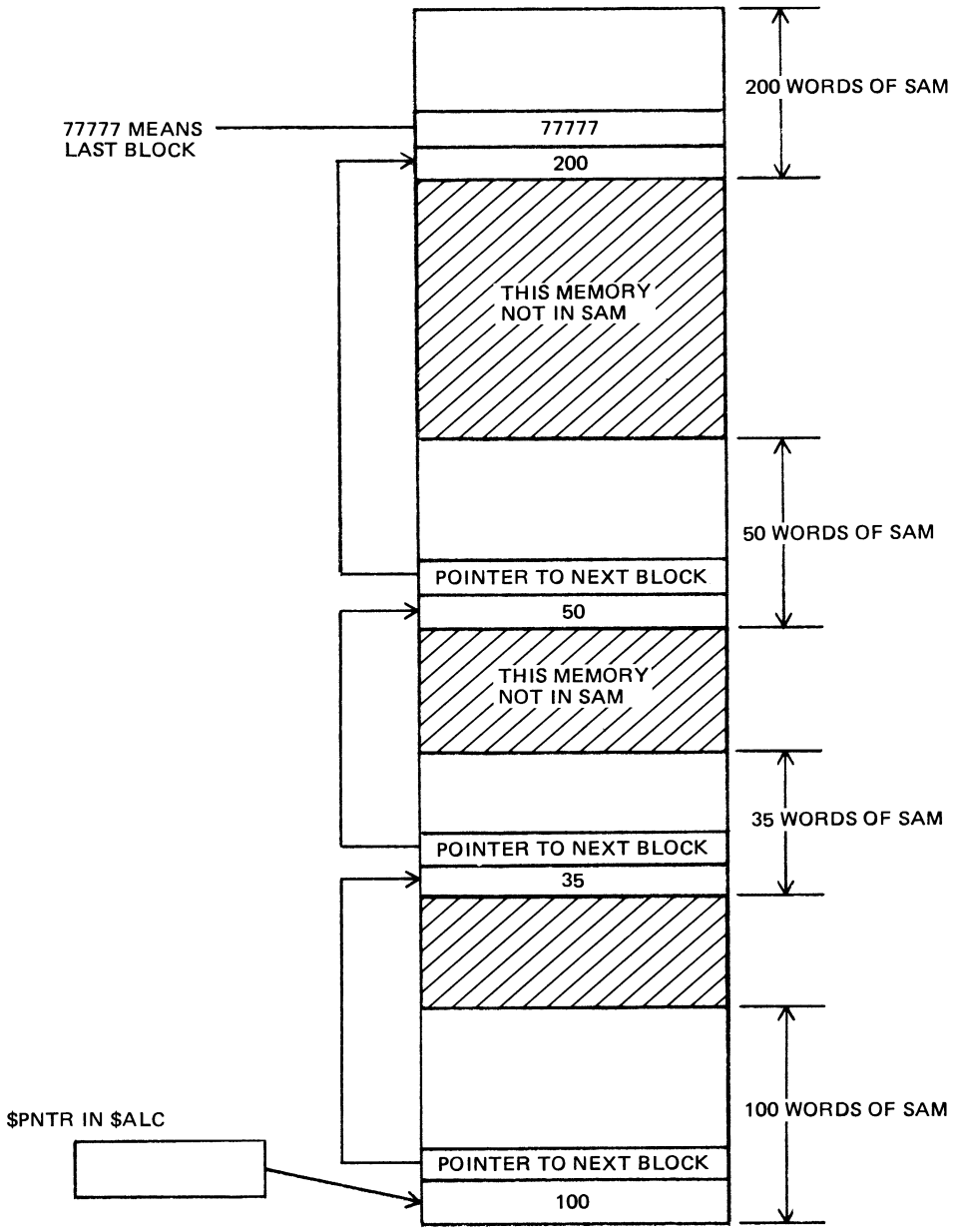


Figure 3-5. Example of SAM Linkage After Returning Memory

Chapter 4

SCHEDULER

INTRODUCTION

The scheduler is the RTE-IVB module which oversees program state transitions, responds to operator input commands, begins system start up at boot up, and satisfies or vectors to other processors eleven EXEC call requests (EXEC 6,7,8,9,10,11,12,14,22,23 and 24). All of this processing is done completely from within the system map.

Calls to the scheduler may come from either the user or other parts of the system itself and thus from either the user map or system map. For this reason a preamble to certain sections of the scheduler are found in Table Area 1 which is in both maps. The entry points that start in the preamble are $LIST, $MESS, $IDNO, and $SCD3. In essence the purpose of this preamble is to get the current DMS status for return purposes, enable the system map, and jump to the appropriate processor. While this code is not specifically part of the scheduler, it is, so to speak, the front door.

The technical discussion on the scheduler which follows assumes that the reader is completely familiar with the 33 word RTE-IVB ID segment and 3 word ID extension. For those who are not, Appendix A at the end of this manual contains a complete description of every word, bit and field.

LIST PROCESSOR

The list processor is a subroutine in the scheduler that is called to move a program from one state to another. In RTE-IVB a program is always said to be in a state. The states are:

STATE NUMBER	STATE
0	DORMANT
1	SCHEDULED
2	I/O SUSPEND
3	GENERAL WAIT SUSPEND
4	MEMORY SUSPEND
5	DISC SUSPEND
6	OPERATOR SUSPEND

The state number is the number used in the status field (word 16) of the ID segment to indicate that a program is in a particular state. For each of these states, except the dormant state, a linearly linked list of all programs in that state is kept. The scheduler manages 5 of these lists. The lists and their heads are:

LOCATION	MAJOR STATE
1711	1 SCHEDULED LIST
1713	3 GENERAL WAIT LIST
1714	4 MEMORY SUSPEND LIST
1715	5 DISC TRACK WAIT SUSPEND
1716	6 OPERATOR SUSPEND

The I/O suspend state has a list headed at each EQT but these lists are managed by RTIOC not the scheduler.

Programs are moved in and out of these lists as their major state changes. The lists are maintained in priority order with the highest priority programs first. Programs of the same priority are added to the list behind the others of same priority. Each list is threaded through ID segment word 1 and is terminated with a zero.

Any number of things can cause a program to move from state to state. For example, suppose FMGR was executing, entering a *SS,FMGR on the system console would cause the system (list processor) to move FMGR from state 1 to state 6. Thus FMGR's status field would change from 1 to 6, word 1 of FMGR's ID segment would be taken out of the scheduled list and put into the operator suspend list.

There is no user interface to the list processor. All calls to the list processor come from other system modules. User requests are first processed in the EXEC or scheduler and then go to the list processor.

LIST PROCESSOR CALLING SEQUENCE

```
     JSB    $LIST
     OCT    (Address Code)(Function Code)
     DEF    (Address) <This word not always required>
ON RETURN
     If A = 0, then no message & B = PROG ID address
     If A not = 0, the A = ASCII error code address
        & B contains decimal error code
```

```
Address codes of 0, 6, & 7 are reserved for drivers.
The only function code allowed with these address
codes is 1 (schedule)
If successful A = 0 ELSE
              B = 3 ILLEGAL STATUS
              B = 5 NO SUCH PROG

For a driver that wants to convert a program name to an
ID address:  JSB $LIST
             OCT 217
             DEF PNAME (Prog Name)
```

This performs a simple list move like changes to priority. (If the program is dormant it's a big NOP). Upon a successful return (A = 0) B will be the ID address of the program. If the program is scheduled many times, doing this removes the search time for the ID segment of the program.

```
       Function Code
          0 = Dormant Request
          1 = Schedule Request
          2 = I/O Suspend Request
          3 = General Wait List Request
          4 = Memory Available Request
          5 = Disc Allocation Request
          6 = Operator Suspend Request
         17 = Relink Program Request
         10 thru 16 are not assigned

          Address Code
               0 = ID segment address (5 parameters passed)
               1 = ID segment address (as next octal value)
               2 = ASCII program name address (a DEF)
               3 = ID segment address in work (no DEF addr.)
               4 = ID segment address in B-Reg(no DEF addr.)
               5 = ID segment address in XEQ1 (no DEF addr.)
               6 = ID segment address (Next parameter is value to
                                       put into B Reg at suspension)
               7 = ASCII program name (passes 5 parameters)
```

For example:

```
---0,7,&6 (Four Drivers)-------  ---1----  ---2----  ----3-----
-                            -  -         -      -  -        -

JSB $LIST  JSB $LIST   JSB $LIST  JSB $LIST JSB $LIST JSB $LIST
OCT 001    OCT 701     OCT 601    OCT 1XX   OCT 2XX   OCT 3XX
DEF RETRN  DEF RETRN   OCT IDADR  OCT IDADR DEF PNAME  ID ADR IN $WORK
OCT IDADR  DEF PNAME   OCT BVAL
DEF PRAM1  DEF PRAM1
DEF PRAM2  DEF PRAM2
DEF PRAM3  DEF PRAM3          (NO INDIRECT DEFS !!)
DEF PRAM4  DEF PRAM4
DEF PRAM5  DEF PRAM5


---4-----       ------5--------
-       -       -              -

JSB $LIST         JSB $LIST
OCT 4XX           OCT 5XX
 ID ADR IN B REG   ID ADR IN XEQT
```

The list processor breaks up the requests shown in the calling sequence into four general cases:

1. Dormant Request

2. Schedule Request

3. Operator suspend request

4. Non-operator suspend request

    a. I/O suspend

    b. Unavailable Memory suspend

    c. Unavailable disc space suspend

In general, before a call to the list processor is made other modules have done a considerable amount of error checking to see if the change is legitimate. These checks are of the nature "Does the program exist"? or "Were the parameters in the proper range"? etc. The list processor performs a "was-will be" check. That is, what was the last state; what will be the next state; are the two compatible? If the compatibility answer is yes, then the requested transition is made. If the answer is no, then the list processor decides on what the proper new state will be. In addition, one other answer can be made. The answer is "yes, but not now". In this case a bit is set to flag an action to be deferred. The R,D and O bits are deferred action bits in the ID segment.

DORMANT REQUEST

The transition processing by the list processor is done as follows:

A. If the abort bit is set then:

   1. The 5 temporary ID segment words are cleared.

   2. The program is placed into a push down stack, linked through word 9 of the ID segment, and headed at $ZZZZ in the dispatcher. (Refer to Appendix E for what the dispatcher does to this stack.)

   3. XEQT is cleared (Base Page word 1717).

   4. The entire status word is cleared and the CL bit.

   5. If this is the currently executing program $PVCN, the privileged rest counter, is cleared.

   6. Link processor is called to do the list move. (Link processor is discussed in the next section.

B. If the abort bit is not set and

   1. Previous status is I/O suspend (state 2) or 0 bit set, then only set D bit and call link processor.

   2. Save resource bit not set then go do A1 through A5 above.

   3. If resource save bit set and 0 bit not set then CLEAR R&D bits, set status to zero; if this is not the currently executing program set the no parameters bit, and call link processor.

SCHEDULE REQUEST

The schedule request portion of the list processor checks actual program status information in the ID segment to see if the program is schedulable.

On a schedule attempt if the program's status is not 0, 2, or 6 then:

1. If dormant bit set jump to dormant request processor.

2. If the W bit is set, change the status field to 3 and call the link processor to put the program in the general wait list.

3. If not 1 or 2 above set entire status word = 1 this clears out all other bits; then

4. Call link processor to schedule program.

If the current status is 6 and ...

1. Dormant bit set too, then set status to 0, clear R&D bits, and call link processor to make dormant.

2. Wait bit set too, then change status to 3 (general wait) and call link processor to put program into general wait state.

3. Else call link processor to put program in scheduled list. That is done A1 through A4.

If the current status is I/O suspend, state 2,

1. If O bit set, and R or O bit set then change status field to a 6 and call link processor to make program operator suspended.

2. If D bit set jump to dormant request processor.

If the current status is 0, that is, first dispatch, then:

1. Perform C1 and C2 in case the program was in the time list and C on SS command set the O bit.

2. Check if the program is disc resident. If so, check if the program terminated saving resources or terminated serially reusable, or was operator suspended. If so, and the program is still in the partition (i.e., has not been swapped out or overlayed), then go to step 3 below, or go do A1-A4 (if the above is not true).

3. If still in partition, then call the dispatcher routine $DMAL to set the partition up to be reused.

4. The final step is to force the programs timeslice word to a 1. This indicates that this is a re-dispatch or a new dispatch so the program is to receive a full timeslice. Then go do A1 through A4.

LIST CALLS BY DRIVERS

Certain $LIST calls have been set aside for use by drivers. These are list calls with function codes of 0, 6, and 7. The form of the call is:

```
JSB $LIST          JSB $LIST          JSB $LIST
OCT 001            OCT 701            OCT 601
DEF RETRN          DEF RETRN          OCT IDADR
OCT IDADR          DEF PNAME          OCT BVAL
DEF PRAM1          DEF PRAM1
DEF PRAM2          DEF PRAM2
DEF PRAM3          DEF PRAM3
DEF PRAM4          DEF PRAM4
DEF PRAM5          DEF PRAM5
```

For function codes of 0 and 7 up to 5 parameters may be passed. At least one parameter must be supplied. The five parameters are put into the XTEMP area of the ID segment and may be picked up by calling RMPAR.

The DEF RETRN must delimit the parameters and no indirect DEF's are allowed. For function code of 1, the ID address (IDADR) must be in the call. For function code of 7 PNAM points to a 3 word array containing the ASCII program name. For function code 6 BVAL is placed in word 11 of the ID segment, the B register at suspension.

Only schedule requests may be made. No other requests are allowed. Note that $LIST does almost no error checking for drivers and none for the op system. It is assumed that if you call $LIST you know what you are doing.

OPERATOR SUSPEND REQUEST

1. If the entire status word is 0 and the program is not in the time list or the status field = 6, then make an "Illegal Status" error return.

2. If current status field = 2, I/O suspend, then set 0 bit.

3. If status field = 0 (i.e. other bits =0) then set R&D bits, make status field = 6, and call link processor to make list move.

4. If not 1,2 or 3 above set status to 6 and call link processor.

NON OPERATOR SUSPEND REQUEST

1. Put requested future status into status field of program's ID segment saving all the upper bits of the same word.

2. Call link processor to make list transition.

On return from $LIST

A = 0 means success
B = ID address of program referenced

else

A = ASCII error code address and
B = numeric error code
= 3 means illegal status (not dormant)
= 5 no such program

LINK PROCESSOR

The REAL TIME EXECUTIVE "LINK PROCESSOR" function is to remove program from one list to add the program to another list.

When removing a program from a list, a check is made of the program status to see if it is in the I/O suspend list. NOTE: The I/O suspend list is not kept in SCHED, but is kept by I/O processor (RTIOC). Thus, if the program is in I/O suspend list, the program removal portion of the routine is bypassed. If program is not in the I/O suspend state, the removal request code value is used to compute the address of the "top of list" word for the particular list. If the program cannot be found in the list, or it is a null list, the program returns as if the action has been performed. This should be an impossible case. Assuming that the program is found in list, the action taken depends on where the program is in the list.

The removal of program from a list consists of:

1. If I/O list (code 2), then this is special case and does not require removal.

2. If NULL list, then error exit taken.

3. If first and only program in list, then list value set to zero.

4. If first program in list, but not the only program in list (linkage not zero), then set list value to the linkage value.

5. If in middle of list, the linkage of the ID segment which points to the program to be removed is set to the linkage value of the program that is removed.

6. If last program in list, the linkage value of previous program in list is set to zero.

After the program has completed the removal portion of the routine, it can then be added to another list. The addition code value is examined to see if it is to be added to I/O suspend list, in which case return is made to calling program. Otherwise, the addition request code value is used to compute the address of the "top of list" word for the particular list. Programs are added to a list according to priority. The program is added to the list just prior to the program of lower priority. The program is added to the list in the following manner:

1. If I/O list (code 2), then this is special case and no addition made to list.

2. If NULL list, then list value set to point to id segment or program to be added and the linkage set to zero.

3. If not null list, the program is inserted into list according to priority level and linkages changed to reflect this insertion.

4. If a lower priority, than any program in list, then last linkage is set to point to the program to be added and the program linkage is cleared.

MESSAGE PROCESSOR

The operator input message processor, $MESS, accepts input commands programatically, generally through the system library routine MESSS or from the system console via the $TYPE routine.

The $TYPE routine is entered by an interrupt created by the operator striking any key on the system teletype. Upon entry, the system teletype ready flag is checked for busy. If the flag is busy, then control is given to $XEQ. If the flag is zero, then check the session mode flag.

If not in session ($ENBL=0), an * (asterisk) is output to the system teletype via $XSIO and a request for teletype input is made via $XSIO with the completion address TYPIO. The system teletype flag is set and control given to $XEQ. When the operator has input his request (signified by LF), the operator message processor routine ($MESS) is called. Upon return from $MESS, the A-register is checked for zero or non-zero. If non-zero, then a message is to be output from $MESS on the system teletype. The A-register contains the address of the buffer which contains the message. The first word of this buffer contains the number of characters to be output and the ASCII message begins at the next word. This message is output via $XSIO and teletype busy flag is cleared and control given to $XEQ. If the A-register is zero upon return from $MESS, the teletype flag is cleared and control given to $XEQ.

If in session ($ENBL not 0 and invoked by the "EN" command), then we look for a session control block defined for LU1. This is done by checking word 3 (session identified) of the first SCB in the list headed by $SHED. If this word is not a "1", we issue the "LOGON" prompt and start the read of the response ($XSIO with a completion address of SESIN). When the read has completed, the user response is sent to a communication program, $YCOM, in a string buffer. $SYCOM then transmits the request to the "LOGON" program to perform the actual log on. The session bit map (!BITM) is updated to indicate that a log-on is in progress for LU1, the system console busy flag is cleared and exit is to $XEQ.

If a session already exists for LU1, the break mode prompt ("S=1 command") is issued. The read of the command is then issued ($XSIO with a completion address of BRKIN), the system console busy flag is set, and exit is to $XEQ. When the input is complete, a check is performed to see if the command entered was an "OP" command. If not an "OPerator" command the command is sent to $YCOM who then sends the request onto RPN for processing. If the command was an "OP" control is transferred to the command processor who processes the command as if it was entered from the system console while not in session mode.

Why, you might ask yourself, do you go to so much trouble in the processing of the system console. The answer is simple -- you should never be locked out of the system console. For example, if the standard PRMPT and RPN processing were to replace $TYPE while the system console was enabled for session use, what would happen if you could not dispatch a program (Disc down, or priority 1 program in a tight loop)? Answer -- nothing! In this example the only course of action would be to reboot the system. With the "OP" command you simply enter a command and the problem is corrected.

NOTE: The following prompt is issued whenever standard processing cannot be performed (no memory for string, log-on started but not complete, etc.).

```
S = ??COMMAND?OP,
```

When this prompt appears, the only command permitted is the "OP" command (note that the prompt contains the first part of the "OP" command as a reminder).

If the log-on or log-off process cannot complete (possibly no programs will run) commands may still be entered via this version of the "OP" command.

The entry point $MESS is in Table Area 1. It is a front end to the actual processing itself. It contains:

```
$MESS NOP
      SSM $MEU
      SJP $MSG
```

The entry point $MEU will then contain the DMS status of the system when the $MESS call was made. This status will be restored when $MESS returns.

$MESS is not a closed subroutine. For example, the OF command will cause a program to be aborted and the associated clean up code to be executed. The return is to the dispatcher not to the caller of $MESS.

The following things are done for calls to $MESS:

1. The command's existence is verified.

2. The command is parsed.

3. The command is dispatched.

The first of these operations is done by checking the transmission log. If zero characters were received, $MESS just exits.

If, upon entry to $MESS, character count is non-zero then the internal parsing routine is called and parses the entire operator input. The output of the parse routine is a 33 word internal buffer. The calling sequence and two examples are shown below:

The Parsing routine scans the ASCII input buffer and stores the data into parameter tables. Commas are used to flag separation of parameters. The character count from teletype driver is assumed to be in the B register upon entry.

A parameter may be up to six ASCII characters in length. There may be up to seven parameters and one operation code input with a maximum of eighty characters. As the input is scanned, a count of parameters and count of characters for each parameter is kept. Characters are stored left justified in the buffer. Word PARAM contains the parameter count and OP,P1,...,P7 contains the ASCII parameter values. The character count for each parameter is kept in word just prior to buffers. PARAM is kept as positive integer and character counts are negative integers.

SYSTEM PARSE ROUTINE

Calling sequence:

```
JSB $PARS
DEF PBUFR     33 word buffer for parsed output

A-REG = input buffer address
B-REG = positive character count
```

The parse routine will accept up to 8 parameters delimited by commas. Each parameter is parsed into 4 words where the first word describes the type of parameter. The format is shown below:

WORD #	CONTENTS
1 (TYPE)	0 if null, 1 if numeric, 2 if ASCII
2	binary # if type = 1, 1st two ASCII char's if type = 2
3	used for ASCII only = 2nd two ASCII characters
4	used for ASCII only = 3rd two ASCII characters

Example:

```
PQ, P Q  RST,55,,10B,556377X,ABCDEFGHIJ
```

Notes:

1. All blanks are ignored.

2. Any ASCII characters past the first 6 are ignored

3. To enter ASCII 77 enter ,77 X, where X is any ASCII character

After the command is parsed its existance must be verified. This is done by a table look up. The Table is at LDOPC and is just a simple list of ASCII opcodes. If the opcode is valid, then a jump is made through table LDJMP. Each entry in LDOPC has a corresponding entry on LDJMP. LDJMP contains the address of the various processors. Note how easy this makes adding new commands. One merely places the ASCII opcode into LDOPC and the address of the processor into LDJMP.

Commands not in the table are dispatched to a routine which returns the proper error.

Errors are returned to the caller of $MESS to be printed in the proper place (or not at all). Recall that $MESS can be called from a program via MESSS (see the library section of your manual).

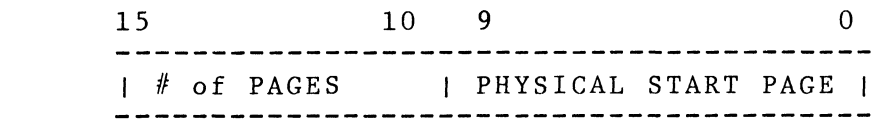
SYSTEM START UP

When the user pushes the run button the final time on system boot up a jump is made to the $STRT routine in the scheduler. $STRT's job is to get the system going. This section of code is executed once and is later overlayed.

The first thing that the start up routine does is to set up the system map.

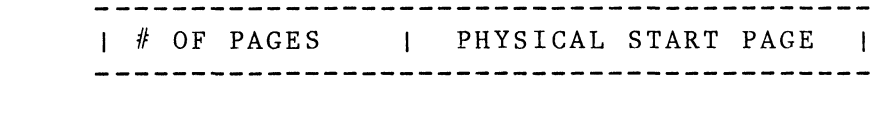
To begin with the first 32K of physical memory will be the system map none of which will be write protected. A JSB is then made to $CNFG, the slow boot routine. This will allow the user to reconfigure system available memory, I/O, and partitions. After this the slow boot returns to $STRT so that set up of the system map can be finished. This mapping routine uses the following information about system available memory.

1st PHYSICAL "CHUNK" of SAM



BP 1660 LOGICAL START ADDRESS
BP 1661 NUMBER OF WORDS

2nd PHYSICAL "CHUNK" OF SAM



BP 1662 LOGICAL START ADDRESS
BP 1663 NUMBER OF WORDS
BP 1664 LOGICAL START ADDRESS
BP 1665 NUMBER OF WORDS
BP 1666 LOGICAL START ADDRESS
BP 1667 NUMBER OF WORDS
BP 1670 LOGICAL START ADDRESS
BP 1671 NUMBER OF WORDS

The first area of SAM, which is a minimum of 2 pages, is set up by the generator and does not change. Physically it is located directly behind the operating system. The second area is set up at generation time but is changable via $CNFG at boot up. It physically resides after the memory resident program area (i.e., before the first program partition).

Note that the second area is divided into four pieces. This allows the user (with the slow boot) to work his way around any bad pages of memory that may exist within SAM.

While the two areas are not physically contigious, they will be made logically contigious. This is done by taking the physical page numbers of both areas of SAM and placing these numbers contigiously into the DMS registers corresponding to their logical address in the system map. $RTN, the system available memory return routine is then called at least twice to fill up SAM with the now contigious memory.

When this calculation is complete the system map is reset. Typically it would look as shown in Figure 4-1.

The $STRT routine also initializes the contents of a few system entry points for later use by other system modules. The following entry points are set.

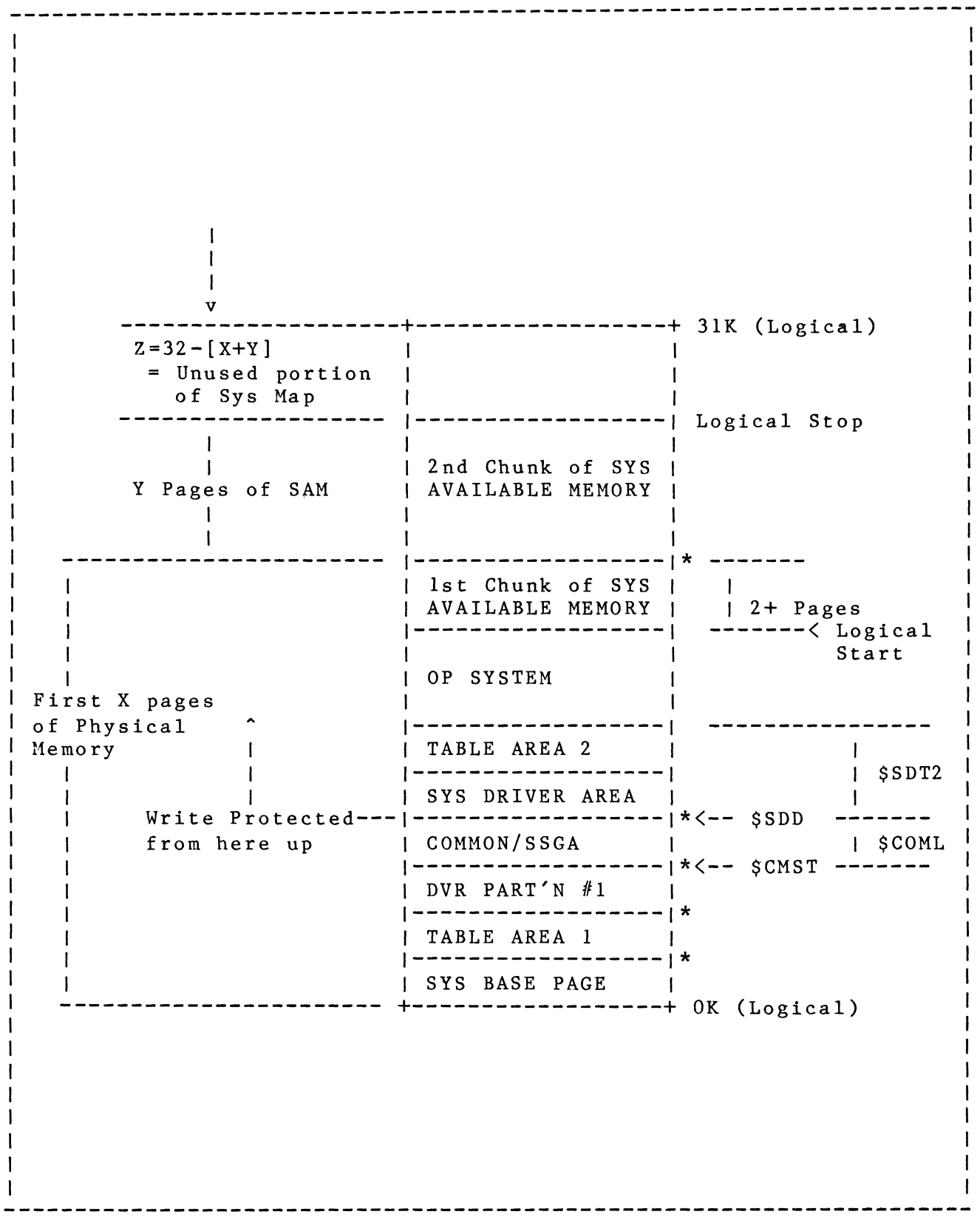


Figure 4-1. System Map for Start Up

$CMST — Starting page of common.
Note that logical and physical pages are the same for $CMST.
$CMST = bits 14-10 of $DLP shifted down
$DLP = disc resident program load point set up by generator

$COML — Number of pages of common
$COML - bits 14-10 shifted down of [MPFI (3) + BGCOM-$DLP]

Where: BGCOM = base page 1753 length of background common
MPFI(3) = fourth entry in memory protect fence table. Start of background common.

$SDA — Starting page of system driver area.
$SDA = $CMST + $COML
Note that logical and physical pages are the same for $SDA.

$SDT2 — Number of pages occupied by the system driver area and table area II.
$SDT2 = Bits 14-10 shifted down of $PLD-$SDA
Where: $PLD is the privileged program load point set up by the generator.

$RLB — Logical starting page of the memory resident library.
$RLB = bits 14-10 shifted down of LBORG (Base Page location 1745) LBORG is the address of the library set by the generator.

$RLN — Number of pages in memory resident library.
$RLN = bits 14-10 shifted down of [MPFI(1) - LBORG]

After initializing these values $STRT calls the $ZZZZ routine in the dispatcher. At this time XEQT is cleared; the interrupt system is cleared; the memory protect fence register is set to 0, swap delay is set up; a check is made to see if there are background, real time, and chained partitions and if not the partition list headers are reset, and lastly FMGR is scheduled. This section of code is only executed once and is later overlayed. A return is made to $STRT.

The last thing $STRT does is pick up the ID address of FMGR, D.RTR, and SMP. These addresses are used later by the system for various types of error checking. $STRT then jumps to the EXEC to finish the system start up.

EXEC also saves D.RTR's address for error checking so that its disc tracks are not released improperly by the user. EXEC then jumps to $CGRN in the $TRRN module to set up the resource number table. A last jump is made then to $SCLK in RTIME to start up the real time clock.

The $SCLK routine starts the time base generator, uses the RTIOC routine $SYMG to print out 'SET TIME' and lastly jumps to $XEQ in the dispatcher. The system is now ready to go.

EXEC REQUEST HANDLERS

Currently there are eleven EXEC REQUESTS involved in the scheduler. They are:

EXEC REQUEST #	PURPOSE	ENTRY POINT
6	Program Completion	$MPT1
7	Program Suspend	$MPT2
8	Load Background Program Segment	$MPT3
9	Schedule w/wait	$MPT4
10	Schedule w/o wait	$MPT5
11	System Time Request	$MPT6*
12	Schedule at absolute time or with time offset	$MPT7*
14	GET or put string	$MPT9
22	Program Swap Control	$MPT8
23	Schedule w/wait and w/queue	$MPT4
24	Schedule w/o wait and w/queue	$MPT5

* The processing of these requests is shared with the system module RTIME.

Control is transferred to the entry points shown above from the EXEC. Briefly, the EXEC call creates a memory protect interrupt which goes to the $CIC routine in the RTIOC module. $CIC transfers control to EXEC after finding that the interrupt was due to memory protect. EXEC checks the parameters for various error conditions (refer to the EXEC Technical Specifications) and if all is well transfers control to the appropriate entry point.

As can be seen from the table above many of the requests ultimately deal with the list processor. In general, the processors pull in the request parameters locally, check them for validity, and if the parameters are valid, a call to the list processor is made.

Four of these requests are briefly discussed here. The other requests are discussed in conjunction with other scheduler functions.

PROGRAM SUSPEND REQUEST

This is an EXEC 7 Request. The processor first checks the program's batch bit. If set, an SC00 error is generated and the program aborted. This is because programs under batch may not be suspended. If clear, $ALDM, which is a dispatcher subroutine that will move the partition out of the allocated list and into the dormant list, is called. Lastly, $LIST is called to operator suspend the program.

SEGMENT LOAD REQUEST

This is an EXEC 8 request. The processor first looks at the request count. If bad, an SC01 error is generated. If OK the system subroutine TNAME is called to get the ID address of the segment. If it is not found, an SC05 error is generated. The entry point address of the segment is then fetched and made the return address of the segment load EXEC call. $BRED in the dispatcher is called to do the actual load. Any parameters that are to be passed are placed in the temporary words of the ID segment. Control is then transferred to $XEQ.

SYSTEM TIME REQUEST

This is an EXEC 11 request. It returns the current system time. The time is kept in two words. ($TIME and $TIME+1) in Table Area II. Each bit corresponds to 10 MSEC with the most significant bits in the upper byte of the second word.

The scheduler checks the input parameters for errors, picks up the time words and turns the rest of the processing over to the $TIMV routine in the RTIME module. $TIMV takes the words and formats them into hours, days, minutes and 10ths of MSECs.

TIME SCHEDULE REQUEST

Only the request count and resolution codes are checked in the scheduler. GETID is called to get the programs ID address. All other processing is turned over to $TIMR in the RTIME module.

PROGRAM TERMINATION

In RTE-IVB there are 9 ways a user may terminate his program. In addition, the system may abort programs too. The user has three variations of the OF command, five variations of the EXEC 6 request, and the EXEC 12 REQUEST. Some of these may be grouped, however, in terms of what the system does.

1. TYPE 1 SOFT ABORT
    a. OF,PROG
    b. CALL EXEC (6,0,2)

2. TYPE 2 HARD ABORT
   a. OF,PROG,1
   b. CALL EXEC (6,0,3)
   c. SYSTEM ABORT

3. TYPE 3 Remove program from System
   a. OF,PROG,8

4. TYPE 4 TERMINATE SAVING RESOURCES
   a. CALL EXEC (6,0,1)
   b. CALL EXEC (12,...)

5. TYPE 5 TERMINATE SERIALLY REUSABLE
   a. Call EXEC (6,0,-1)

6. TYPE 6 NORMAL PROGRAMMATIC COMPLETION
   a. CALL EXEC (6,0,0)

We shall discuss each of these types in the order of increased system processing requirements.

The type 6, normal completion request, requires the least processing and is by far the most common of program terminations. It is mostly done in the scheduler TERM subroutine.

The TERM routine first calls the list processor to put the program dormant. If the father's waiting bit (FW) is set for this program, then the system finds the father and clears his 'W' bit which was set, and if he is in state 3, the list processor is called to schedule him. It is possible that the father is waiting but is not in state 3. This would indicate that he is possibly dormant because his father made him dormant or that he is in another state with the 'W' bit set. For this reason he is rescheduled only if he is in state 3. For other cases the list processor picks up the fact that he should be scheduled by the indication that was left by clearing the 'W' bit. The TERM routine then clears all but the "RM","RE".

"PW and "RN" bits in words 21 of the program being put dormant, and returns. The RN bit of the ID segment indicates that the program has resource numbers. The RM flag indicates that it has re-entrant memory that has been moved. These resources will be released by DISPA when it finds the program linked into the abort list at "$ZZZZ" (refer to Appendix E for a description of this process).

Next, any optional parameters supplied in the termination request are placed in the 5 word temporary word of the ID segment. This allows the original scheduling parameters (or any others) to be picked up with the system subroutine RMPAR.

Next, bit 1 is set in word 20 of the ID segment to indicate normal termination. This flag is checked when the ABORT processor in the Dispatcher calls $EQCL to handle EQT's locked to the program. Refer to the Equipment Locking Capability section in Chapter 2 for details.

This is the minimum processing for program completion.

The Type 1, soft abort termination, requires a little more processing. The soft abort starts with a call to the SABRT subroutine in the scheduler.

The first thing SABRT does is to clear the 'R' and D bit in the status word. This will force the list processor ($LIST) to truly put the program dormant. The system then calls $TREM (in RTIME), which will remove the program from the time list. This clears the ID segments T bit.

The W bit is checked next, if set then this program is a father waiting for a son. (Recall that son's ID address is in word 2 of fathers ID segment.) In this case the sons FW bit is cleared. This insures proper processing when the son terminates.

The TERM subroutine, described earlier is next called.

Lastly the SABRT routine checks to see if this program is the son of another program. If so then a 100000B is placed into word 2 of the fathers ID segment and the address of word 2 is placed into word 11, the B register at suspension word. This allows the father to do a RMPAR call and to get back a word (the first of 5) that indicates that the son program was aborted. This is how FMGR, for example, knows to generate the "ABEND XXXXX ABORTED" message.

Next in order of processing is type 2, the hard abort. The hard abort is performed in the $ABRT subroutine. However, before calling this routine a check is made of the programs current status. If the status is I/O suspend (state 2) a jump is made to the RTIOC routine $IOCL.

Briefly, $IOCL CLEARS out any 'hang up' conditions caused by program input or output. It scans all the EQT's I/O linked lists looking to see if the program is in the list. (Linked through first word in ID segment). If any I/O is found the program is delinked and the I/O cleared. $IOCL then calls $ABRT to finish the abort.

$ABRT sets the abort ("A") bit in the programs status word (recall that we discussed this bit in the $LIST discussion). The "A" bit being set indicates a hard abort to $LIST and forces it to set the program dormant. $ABRT then calls SABRT which we just discussed. $ABRT then calls $SDRL in EXEC which releases any disc tracks the program owns, and, if any are released, calls $LIST to schedule all programs waiting for disc tracks. The exception here is that $SDRL will not release tracks belonging to D.RTR. After $SDRL returns, $ABRT sets up the program abort message and sends it to $SYMG in RTIOC which will send it to the system console.

Next in order of processing is the power abort, type 3. Normally this is not done programmatically (call to $MESS), it is done with the OF command. The power abort calls $ABRT to do the hard abort first. The TM bit is next checked if the TM bit is set, it indicates that the program was loaded temporarily online, and there is no copy of its ID-segment on the disc. Only in this case can the OF processor clear the ID segment. The rest of the OF code computes the number and location of the tracks holding the program (words 23-27 of the ID segment) calls $DREL in EXEC to release the tracks. The OF request assumes an ID segment owns a track only if it references sector 0 on that track. This convention prevents double release of tracks in cases where background segments start in the middle of a track. Furthermore, $DREL will only release the tracks if they are owned by the system (i.e., it will not free FMP tracks). $DREL also reschedules any programs waiting for disc tracks by calling $LIST.

When $DREL returns, the OF routine clears the 3 name words (except for the SS bit, which indicates a short ID-segment, and the track assignment words), it releases any EMA ID extension, and then goes to $XEQ.

The type 4, save resources termination is a special case of the normal termination. In this case the dispatcher subroutine $ALDM is called. This routine unlinks the partition the program executed in from the allocated list and puts the partition into the dormant list. The $MATA entry D bit is also set. Next the R bit in the ID segment is set. This is done so that the list processor will not put the program in the clean up stack headed at $ZZZZ. (Refer to Appendix E) ($LIST will clear the R bit).

Now if this case is a father terminating his son then all that is left to do is a $LIST call to place the program dormant. The more general case, however, is the program terminating itself.

In this case the $WATR routine is called. All $WATR does is check the PW bit to see if any other program wants to schedule this program that is doing the save resources termination. If the bit is set then a search of the general wait list is made to see who is waiting. (Recall word 2 of the waiting program will have the prospective son's ID address. The prospective son is now doing the save resources termination). If the prospective father can be found a $LIST call is made to reschedule him. This allows the schedule request to be reissued. The rest of the processing is done exactly like the normal program termination.

Lastly there is the serially reusable completion. A check is made to make sure a father is not trying to terminate a son as serially reusable. If this is detected a normal termination results. If the program is terminating itself then the TERM subroutine is called. Next the least significant bit of the father ID number word is set as a flag to the dispatcher clean up routine (refer to Appendix E) that the programs partition is not to be put in the free list. $ALDM is then called to take care of the partition. Lastly any optional parameters supplied are placed in the ID segment temporary area.

PROGRAM SCHEDULING

There are four ways to schedule a program in RTE-IVB. The program can be scheduled by time, event, operator command, or another program.

NOTE that if a session program is in the time list, the following access restrictions are enforced:

- -EXEC schedule or program time value requests referencing a program in the time list may only be issued by another program of the same session. An attempt to reference a session's time scheduled program by another session or a non-session program will result in an SC11 error.

- -The session operator commands IT, RU and ON have the same access restrictions as described above. Attempts to reference a time scheduled program belonging to another sssion will result in an "ILLEGAL STATUS" error.

Note that the above restrictions apply to session programs and operator commands only.

To schedule a program by time the program must have been in the time list already. (This would require the operator ON request earlier). Every time the time base generator interrupts control is transferred to the $CLCK routine in the RTIME module. Here every program in the time list (threaded through ID word 17) is checked to see if it is time to execute. If words 19 & 20 of the ID segment equal the system time stored at $TIME & $TIME+1 and if the program is dormant, a call is made to the list processor to schedule the program. Regardless of program state, the next start time is calculated and stored back into the ID segment. (The new time is not computed if the multiple value is 0. This means the program is to be removed from the time list.)

Scheduling by event is typically done by drivers. DVR00 and DVR05 for example, schedule the program PRMPT due to an event, that is, an interrupt. This scheduling is done by a $LIST call.

The ON and RU commands are another way to schedule a program. These two commands differ in that the RU command will schedule a program now regardless of the time list parameters. The ON command is capable of putting a program in the time list and/or scheduling the program immediately. In both cases a call is made to $LIST to do the scheduling.

Before the $LIST call is made the program is checked to see if it is dormant. If not an "illegal status" message is returned. If the 'IH' was not entered in the schedule command and parameters are allowed on schedule (i.e. NP bit Clear), then any parameters supplied with the command are put into a string block in system available memory. The first five of the parameters are placed into the temporary words of the ID segment. (String processing is discussed in the next section.) In the case of the RU command the $LIST call is made next and that's the end of the RU processing.

The ON processor looks at the programs ID segment resolution code to determine the next process. If the resolution code is 0, only a $LIST call is made. If the resolution code is not 0 then the $ONTM processor in RTIME finishes the processing. Basically $ONTM checks for the NO (NOW) in the command. If present then the program is put into the time list and executes at the current system time and 10 milleseconds. If the NO is absent $ONTM places the program into the time list. The program then executes at the time specified in words 19 and 20 of it's ID segment.

The last way to schedule a program is programmatically (EXEC 9, 10, 23 and 24 requests). The processing here is somewhat more involved than the ON or RU commands because a father son relationship is involved. Most of the processing is done in the IDCHK subroutine. The routine does the following:

1. Makes sure the program exists, else generates an SC05 error.

2. Makes sure the name specified is not a segment name, else generates an SC05 error.

3. Makes sure the program will find a partition large enough to execute in, else generates an SC09 or SC08 error.

4. Places perspective son's NP bit and bits 0-3 of status field into the perspective father's A-Register at suspension word.

5. Calls the string passing routines if necessary. (i.e., if RQP9 = 0 no string passing.)

6. Makes sure that the first five optional scheduling parameters are put into the sons ID temporary words.

For exec 9, 10, 23, and 24 requests, the RU, ON, SZ and AS commands, the SIZIT subroutine is called to see if a partition exists that is large enough to execute the program. Thus insuring that a program scheduled is dispatchable. For memory resident programs the check is ignored.

For non EMA programs the check uses the # of pages field, word 22, of the ID segment and compares this against:

# of pages =< $MBGP if the program is background

# of pages =< $MRTP if the program is real time

Alternatively, if the program is assigned to a partition (RP bit in ID segment set) then the partition # field is used as an index into the $MATA table to see if the destination partition is large enough for the program and if the partition is still defined. (Note programs already in memory with an allocated partition may not have their sizes changed. The SZ operator request error check routine guards against this.)

It may also happen that $MBGP or $MRTP is larger than a 32K address space. In this case the check # of pages =< MAX ADDRESS SPACE is used.

If the program is an EMA program, the following check is used.

OF PAGES - MSEG + EMA SIZE =< $MCHN OR ASSIGNED PARTITION SIZE

where MSEG is in word 1 of the ID extension, EMA size is in word 29 of the ID segment, and $MCHAN is the size of the largest Mother Partition.

If the check fails on SC09 or SC08 error (SIZE ERROR) will result. However, if the DE bit (EMA default) is set then the EMA size is reset to 1 and the check is performed again. If the check now passes all is well and the EMA size of 1 will be used by the dispatcher as a flag to give the program the largest possible EMA size.

If the reader has already read the sections on the AS and SZ commands, the question may come up "Why check for size, this is already done in the LOADR and for on line commands?" the reason is that the FMGR 'SP' and 'RP' commands allow the user to save programs whose size or assignment may not match the currently defined partitions. The error checking prevents a mismatch of program and partition from causing system problems.

NOTE that every time a program is scheduled $MCHN, $MBGP or &MRTP (or the destination partition size) is used as a check to see if the program can fit into a partition. If $MCHN, $MRTP or $MBGP = 0, then no partitions of that type is available and the program is not dispatchable. This may happen if a parity error causes a partition or partitions to become undefined. Should the scheduler detect this condition, the program will not be scheduled and an SC08, SC09, or 'SIZE ERROR' will be reported to the system console.

STRING PASSING

Upon scheduling a program with the RU, ON or GO commands, a section of system-available-memory (SAM) will be allocated for storage of any command string and entered in a push down stack linked through the first word of each block (see Figure 4-2). The head of the stack will have the name $STRG and reside in the SCHED module. A command string is defined as everything following the prompt in a scheduling call.

If the program is scheduled by a RUIH,ONIH, or GOIH, then the string storage portion of the command will be inhibited. The first word of each block of memory will contain a pointer to the next memory block. The last block of memory in the stack will contain 0 in its link word. The second word of each block of memory will contain the ID address of the scheduled program. The sign bit, when set, will indicate that the memory block has an additional word (see system description of the memory allocation routine, ($ALC)).

The third word of each block will contain the character count of the command string. The fourth through (N+1+3)/2 words will contain the N characters in the command string.

Upon scheduling a program with the RU, ON or GO command, the following steps will occur at parameter storage time:

1. If there is no parameter string, continue at Step 5.
2. Store parsed parameters into ID segment words 2 to 6 as before.
3. If the command is RUIH, ONIH or GOIH then do not store parameter string and continue at 5.

4. Deallocate any string block(s) associated with the scheduled program.

   Allocate a block from SAM, store the entire command string into the block and enter it into the stack. If SAM is not available, then the request is ignored, the following error message is issued to the operator's terminal:

   CMD IGNORED - NO MEM

   and control is returned to the system at $XEQ.

5. Schedule the program for execution.

The user can retrieve the string by using the EXEC 14 request or the system library routine GETST. Both routines release the string memory back to the system. Alternately, programs can still recover the first five parameters (treated as one computer word each) by using the RMPAR call as the first call in the program.

Any time a program goes dormant, normally or abnormally, any command string block assigned to the program will be returned to SAM. This is accomplished in the ABORT routine of the dispatcher.

SCHEDULER INTERFACE WITH DISPATCHER

Several portions of the scheduler interface to the dispatcher. The list processor portion of the scheduler interfaces on program scheduling. The list processor also interfaces with the dispatcher on program completion as described in Appendix B. In addition, the UR, AS and SZ operator commands affect the dispatchability of a program. The error checking for these commands is discussed below.

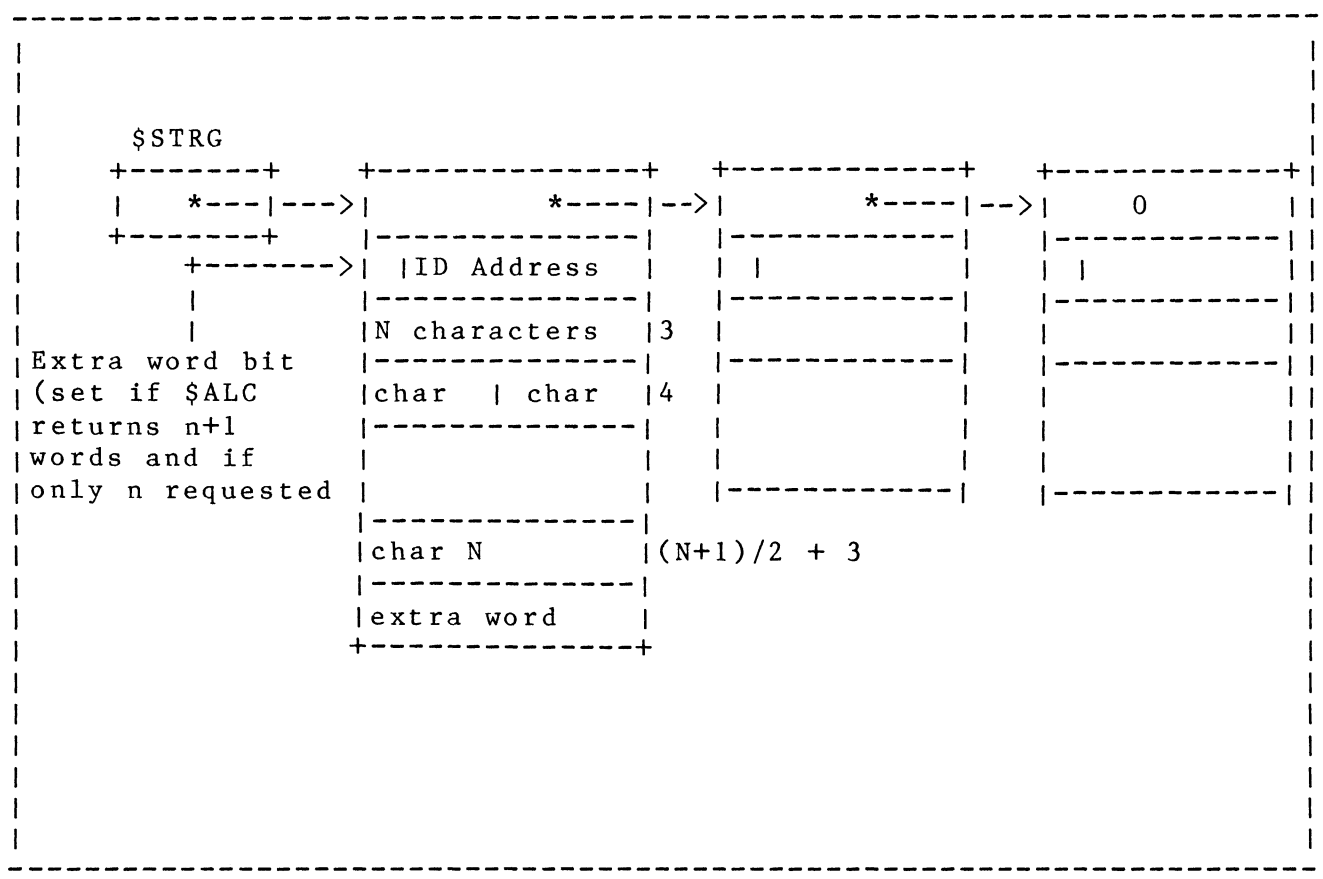


Figure 4-2. Stacking of Memory Blocks

The AS and SZ both require the program referenced to be dormant and not memory resident. Moreover, the program must not still own the last partition in which it executed. (Recall that a serial reusable, save resource termination, operator suspension does not release the partition.) The partition # field of word 22 is used as on index into the $MATA table and the $MATA residency word is checked to make sure the referenced program no longer owns the partition. If any of these conditions are not met the "ILLEGAL STATUS MESSAGE" is output.

Some other error checking is performed for the AS command. The Partition must exist and the size of the program is checked against the size of the referenced partition. For non EMA programs the # of pages field is compared against and must not be greater than the partitions $MATA entry. For EMA programs the formula used is:

# of Pages - MSEG SIZE + EMA SIZE <= MOTHER PARTITION SIZE

where: MSEG SIZE is in word 1 of the ID extension, EMA SIZE is in word 29 of the ID segment, and MOTHER PARTITION SIZE is in the $MATA table.

EMA programs may be assigned to regular partitions in addition to chained ones. The size check formula used in this case is

OF PAGES - MSEG+EMA SIZE <= PARTITION SIZE

If at the end of all the error checking, the AS command is determined to be valid, then the RP bit is set and the partition # is set into partition # field.

(Partitions count from 0. That is; AS,PROGX,7 will result in a 6 being placed into the partition # field.)

The SZ command processor performs the program partition and size checks mentioned earlier plus a few more. Word 30 of the program ID segment for segmented programs or word 24 for non segmented programs is used as the lower limit of the error check. The upper limit is defined by the program type as follows:

new SIZE-1< $MBGP for background programs
new SIZE-1< $MRTP for real time programs

If the program is assigned to a partition

new SIZE-1 <= ASSIGNED PARTITION SIZE

(The minus one is because $MBGP & $MRTP does not include Base Page.)

If the size is found to be valid then the # of pages field is updated to reflect the new size. (Note that the # of pages field does not include base page.)

NOTE also that $MRTP, $MBGP, or the partition size is not used if the MAX address space is smaller than these values. That is, a program plus the associated system tables may not exceed a 32K address space.

EMA programs have a special form of the SZ command (i.e., SZ,PROG,P1, P2). As mentioned earlier checks for partition and program status are made. Other checks are also made. The DE bit, word 1 of the ID extension must be set to change EMA size or the command is invalid. Recall that a set DE bit means default EMA (not necessarily MSEG) was taken.

In this case P1 is the new EMA size and P2 is the new MSEG size. P1 is checked as:

P1 + PROG CODE SIZE <= $MCHN or assigned partition size

P2 is checked as:

P2 + PROG CODE SIZE <= PROG Address space

If both of the above are satisfied P1, the new EMA size is placed into the EMA size field of word 29 of the ID segment and P2 is placed into the MSEG field of the 1st word of the ID extension.

The last operator command that affects partitions is the UR command. This command clears the R bit in the referenced partition's $MATA table entry. This command may affect the system entry points $MCHN, $MBGP and $MRTP. These entry points contain the size of the largest unreserved partition of that type (i.e. Mother, background and real time).

If a partition is being unreserved and it would then be the largest unreserved partition of its type then $MAXP will be called to do the appropriate updating.

Chapter 5

PERR4 - RTE-IVB PARITY ERROR MODULE

PARITY MODULE OVERVIEW

The Parity Error module's main task is to report parity errors detected by the hardware and to continue operation of the RTE-IVB system if possible. PERR4 also tries to reproduce parity errors to identify and warn system users of soft parity errors: errors which may be intermittent or may be generated erroneously.

EXTERNAL COMMUNICATION

The Parity Error module communicates with the rest of the operating system through the system tables, base page communication area, and subroutine calls to other modules in the system.

SYSTEM TABLES REFERENCED

The System tables used by PERR4 are:

a. ID Segment entry for accessing program status.

b. $MATA table for accessing partition configuration information.

c. INT table for determining PORT map status.

SYSTEM BASE PAGE COMMUNICATION

XMATA	1646	Address of current MAP entry
INTBA	1654	Address of interrupt table
EQT1	1660	Address of current EQT entry
XEQT	1717	Address of current program ID Segment entry.

EXTERNAL SUBROUTINES CALLED

$ABXY	-	set up ABE,XYO register reporting messages and print them on system console
$CNV1	-	convert number to ASCII (one word)
$CNV3	-	convert number to ASCII (three words)
$ERMG	-	used by PERR4 to print "PE" error message and abort user program
$MAXP	-	reestablish maximum size words of unreserved partitions
$SYMG	-	print message on system console
$UNPE	-	unlink a partition entry from the proper list and undefine the partition.

OTHER EXTERNAL REFERENCES

$CIC	-	entry point to Central Interrupt Control routine (contains address of last point of interrupt).
$DMS	-	two word save area. word 1 - DMS status at last interrupt word 2 - Interrupt status at last interrupt 0 if ON, 1 if OFF.
$XCQ	-	entry point of Dispatcher. This is used instead of the return point at $CIC when a program is aborted.

DETAILED TECHNICAL ASPECTS OF OPERATIONS

This portion of the Technical Specifications is a detailed description of the major portions of the Parity Error module, PERR4. It is assumed the reader is familiar with the detailed operations of the Dispatcher (DISP4) and the I/O module (RTIO4).

PARITY ERROR DETECTION

Because parity error interrupts can occur even when the interrupt system is off, the code at $CIC must be able to save the complete system status. The major hole in being able to save the complete state is in saving the interrupt system state. In order to do this in both the 21MX and the 21XE the instruction 103300 was used to both test the interrupt system and turn it off.

Parity error interrupts may be generated at almost anytime because DCPC transfers may be stealing memory access cycles. If it occurs while the system is in the idle loop, $CIC can not save the registers in XA, XB, etc. because all of these are actually one location. It was necessary for $CIC to identify the source of interrupt before saving all the registers. Only the A-register needs to be saved temporarily so that LIA 4 and a LIA 5 can be done. PERR4 is entered only when LIA 4 = 5 and LIA 5 = 1xxxxx.

PERR4 saves all registers in local locations. It requires that 2 words be set up at entry point $DMS by $CIC. The first word being the DMS status register contents containing the memory protect status and mapping information. The second word indicates the status of the interrupt system at the last interrupt (the parity error interrupt). The logical parity error address from the violation register is saved. The contents of location 5 are saved and replaced by a JSB indirect through a base page location to a PERR4 routine.

PARITY ERROR VERIFICATION

The routine TRYPE is called to test if the parity error is in the system map. [The DMS status word cannot be used to determine the map under which the parity error occurred because certain DMS instructions change maps in the course of their execution and do not change the DMS status register.] TRYPE saves the map indicator value and then re-enables the parity error system. If the system map is needed, a regular load is done from the logical address of the parity error. The next instruction is executed if there is no parity error at tested location. If the user map is needed, a cross-map load instruction is used to read from the logical address of the parity error. The next instruction is executed if no parity error is detected. CLF 5 is used to turn off parity error until another verification attempt is made. A NOP is needed between the XLA LOGPE,I and the CLF 5 because of timing delays required by the HP 1000 M, E, and F Series computers.

If a parity error cannot be reproduced in the system map an attempt is made in the port maps. The user map is saved before the port Maps are checked. The interrupt table is checked to see DCPC channel 1 is busy. If it is, the Port A map registers are copied into the user map. The TRYPE routine is called to try and reproduce the parity error. If no error is found the next DCPC channel is tested in the same manner.

After both DCPC channels have been tried without success, the user map registers are restored and TRYPE is called once again. The user map is tried last to avoid an erroneous report in the case where a swap out was taking place in one of the port maps. The user map may still contain a copy of the same user (left over from the set up for the port map by RT104).

PARITY ERROR RECOVERY PHILOSOPHY

While it is possible to always detect the occurrence of a parity error, it is not always possible to effect a complete recovery from a parity error. There are a number of reasons why 100% recovery is not possible; these will be explained below. The overriding philosophy is to maintain system Operation whenever possible and eliminate, if feasible, the possibility of future parity errors.

WHO DUNNIT?

When a parity error is detected, the violation register records the logical address of the word containing bad parity. The P-register saved in the interrupt handler's entry point may or may not point to the instruction which caused the bad location to be referenced. This is especially difficult to trace back when the instruction was a multiple word instruction such as XLA, MVW, or DLD. So while we may verify that a location in the system contains bad parity, we cannot determine that a user program caused the reference to the bad location via use of a XLA instruction.

THE SUDDEN BLOW

A parity error detected during a DCPC transfer while the system map was enabled means the operating system was executing and it is a privileged system. Since the system may still be in RTIOC following a DCPC initiation, in the DISPATCHER in the EXEC abort routine, or in the system console driver; these routines would have to be reentered to print parity error messages or abort a program. So these are not recoverable.

IT'S AN INSIDE JOB

A parity error detected within the operating system itself may cause erroneous execution of the system. For example, if a parity error was in a JMP instruction, it is possible the P-register may not get set correctly. This type of error is also not recoverable.

SOFT PARITY ERROR

If a parity error cannot be reproduced (by reading a word at the logical parity error address in the system map, Port A map, Port B map, and user map) then it is considered to be a soft parity error. This type of error usually indicates an equipment problem: There may be intermittent memory parity errors, it may be a memory controller/ backplane problem, or even a firmware error.

Soft parity errors cause a message to be printed which gives the logical parity error address and the DMS status register contents at the time of the interrupt. These messages should help indicate where intermittent failures may be located, especially if these soft parity error messages become more frequently reported.

SYSTEM PARITY ERROR

Parity errors in memory locations in the system itself cannot be recovered as described in Parity Error Recovery Philosophy section. The system is halted (102005) with the A-register containing the physical page number and the B-register containing the logical parity error address. The table areas and system COMMON areas are also considered to be part of the system.

USER PROGRAM PARITY ERROR

Parity errors within the memory resident area will cause the program to be aborted. The physical page number, ABEXY0 register contents and the logical parity error addresses are printed on the system console in addition to the program abort message. The system then continues operating.

Parity errors within a disc resident program require the partition or partitions affected to be undefined. The program's MATA (Memory Allocation Table) entry is examined to see if it is in a regular partition, a subpartition, or a mother partition.

If the parity error is detected in a program in a regular partition or a subpartition, an attempt is made to check if the physical page number of the parity error is actually within the partition's physical page definition. If the page is not in the partition, the error is treated as if it were in the system area and halts (102005). If the page is in fact part of the partition, the partition MATA entry address is saved. The partition is then unlinked from any partition lists and is undefined by a call to $UNPE. If there is a Mother partition, $UNPE is also called to undefine that partition.

If the parity error is in a program which occupies a Mother partiton. The partition MATA entry address is saved. Then a search is made through all of its subpartitions to see which subpartition is also affected. That subpartition's MATA address is then saved and the subpartition is removed from the system by $UNPE. $UNPE also releases all the other subpartitions back into the appropriate regular partition free list.

Finally the partition number or numbers are printed out as being downed. Then the program is aborted along with the parity error messages as in the case for memory resident programs.

DCPC PARITY ERRORS

If a parity error is verified to have occurred under a DCPC transfer, the DMS status register is checked (this is almost the only time when DMS status register can reliably indicate the correct map which was enabled at the time of the parity error interrupt). If the system was enabled at the time of the interrupt, a halt (103005) is necessary because the operating system must not be reentered. If a user or the idle loop was interrupted, the I/O request currently queued on the EQT which had the DCPC channel is examined. If the request was a system or buffered request, a halt (102005) is done. If it was a user request, the parity error is treated as in the case of a user program parity error (see the System Parity Error Section).

Chapter 6

SYSTEM LIBRARY

SYSTEM LIBRARY CHANGES SUMMARY(RTE-III TO RTE-IV)

Changes had to be made in several system library routines to make them compatible with RTE-IV. The system entry points necessary for $ALRN, RNRQ, LURQ, COR.A, EQLU, IFBRK, PRTN and MESSS routines are included in Table Area 1. The only changes made to these routines were to do crossmap loads and stores to entries in the system. Routines KCVT, TMVAL,INPRS, CNUMO, CNUMD and PARSE need to use routines in the system whose entry points are not available to the user. The code for these routines ($CVT1, $CVT3, $TIMV and $PARS) is duplicated in the system library routines that call them.

Five new routines COR.B, .EMAP, MMAP, .EMIO and EMA were added to the system library. COR.B routine returns in the B register, the first word of free available memory of the program if there are no segments. If the program is segmented then COR.B returns the high address + 1 of the largest segment. If the ID segment address passed to COR.B is that of a short ID segment, COR.B makes an error return with a -1 in the A register. .EMAP resolves array addressing for normal arrays and for EMA's. MMAP maps mapping segments for EMA's, .EMIO handles EMA addressing and mapping for special cases to insure the entire buffer needed is mapped into the logical address space and EMA returns information on EMA. These new routines are described in the next section.

.EMAP, MMAP and .EMIO can be RP'ed in an RTE-IVB generation for use on an HP 1000 E-series computer with instructions which will link to microcoded versions of these routines. .EMAP and .EMIO and MMAP are interruptible. The opcodes for the EMA microcode are:

.EMAP	105257
.EMIO	105240
MMAP	105241

TECHNICAL DETAILS FOR EMA ROUTINES

.EMAP SUBROUTINE

This routine is used to resolve addressing of an element in an n-dimensional array. The algorithm used to calculate displacement for an array element (A_1, A_2, A_3,...A_{n-1}, A_n) is:

Displacement=

$$(((...((A_n - L_n)*d_{n-1} + (A_{n-1} - L_{n-1}))*A_{n-2} + ...$$

$$(A_3 - L_3))*D_2 + (A_2 - L_2))*D_1 + (A_1 - L_1))* \#words/element$$

where $A_1,...,A_n$ are subscript values defining an element in an n-dimensional array, $L_1,...,L_n$ are the lower bounds of the dimensions, $D_1,...,D_{n-1}$ are the magnitudes of subscript declarators ($D_i = U_i - L_i + 1$, where U_i is the upper bound of the ith dimension) for dimensions 1 thru n-1, # words/elements is the number of words per element in the array (for e.g., 2 for real constants, 3 for double precision constants, etc.). The leftmost dimension (A_1 is subscript value) is varied the fastest to calculate the displacement.

The user (compiler in the case of the higher level languages) must build a table containing the number of dimensions in the array, the negative of the lower bounds for every dimension, the magnitude of subscript declarators for dimensions 1 thru n-1, and the number of words per element, and two offset words if the array is in EMA. The format of the table is:

Table 6-1. Number of Dimensions in the Array

# dimensions - L n D n-1 - L n-1 D . n-2 . . - L 2 D 1 -L 1 # of words/element offset word 1(low 16 bits) offset word 2(high 16 bits)	used only for EMA

Where Li is the lower bound and Di is the magnitude of the ith dimension.

.EMAP CALLING SEQUENCE

The calling sequence for .EMAP is:

```
    JSB .EMAP
    DEF RTN        Return address
    DEF ARRAY      start of the array
    DEF TABLE      Table containing the array parameters
    DEF A          Subscript value for the nth dimension
         n
    DEF A          Subscript value for the (n-1)st dimension
         n-1
    .
    .
    .
    DEF A          subscript value for the 2nd dimension
         2
    DEF A          subscript value for the 1st dimension
         1
```

RTN error return A Reg = 15 (ASCII), B Reg = EM (ASCII)
RTN+1 normal return B Reg = address of the element

If the XIDEX, base page location in the communication area containing the currently executing program's ID segment extension address is not zero and the start address of the array is greater than or equal to the starting logical page of MSEG for this program, then the array for which addressing has to be resolved is an EMA. The procedure followed by .EMAP to resolve element addresses for EMA and non-EMA arrays is the same except for two-word calculations being performed for the EMA. Following steps describe address evaluation for an element in EMA.

1. Initialize pointer PTABL to point to the first entry in TABLE and set the two summation words SUM1 and SUM2 to 0.

2. Get # of dimensions, if negative then error, if 0 then skip to Step 10.

3. Add -Li to Ai and add it to the current sum of previous terms (SUM1).

4. If bit 15 of the summation word (SUM1) is set, then increment SUM2 by 1. Clear bit 15 in SUM1.

5. Multiply SUM1 by D_{i-1}. Save A & B registers in SUM1 and SUM3 respectively.

6. Shift bit 15 of SUM1 into the bit 0 position of SUM3. Clear bit 15 of SUM1.

7. If SUM2 is not equal to 0 multiply it by D.

8. Add SUM3 to SUM2.

9. All dimensions done? If not, go to step 2 to evaluate displacement for the next dimension.

10. Get MSEG size and the logical start page of EMA from the first and second words of the ID segment extension of the program.

11. Get the two offset words from the table. Adjust them so that the first word contains low 15 bits (bits 0-14) and the second word contains bits 15-31 of the double word offset. Error if second offset word was negative.

12. Add offset word 1 to SUM1 and offset word 2 to SUM2. Error if overflow occurs either into the sign bit or out of the sign bit of SUM2.

13. The array is an EMA. To get the # of pages in the displacement move bits 10-14 of SUM1 into the least significant bits of SUM2. Then SUM2 is the # of pages in the displacement. The remainder in SUM1 is the offset into the last page for the element address.

14. Get EMA size from the ID segment of the program. If # of pages in the displacement from start of EMA +1 >EMA size then error.

15. Divide SUM2 by MSEG size. MSEG# = quotient. The number of words displacement into the mapping segment = Remainder *2000B + SUM1. The address of the element in the mapping segment = displacement + base address.

16. The number of pages displacement from start of the EMA up to the start of the MSEG to be mapped JPGS=(number of pages displacement from the start of the EMA up to the page containing the element -#of pages displacement from the start of the MSEG to be mapped up to the page containing the element).

17. If the mapping segment number in the ID segment extension = MSEG# and bit 15 of word 0 of the ID segment extension is not set, then go to step 24.

18. To determine the highest possible MSEG#, divide EMA size by MSEG size and if the remainder is equal to 0 subtract 1 from the quotient.

19. The highest possible MSEG# is HMSEG = quotient.

20. If the remainder is not 0, the mapping segment size of HMSEG = remainder -1 otherwise it is the standard MSEG size.

21. If the mapping segment to be mapped is HMSEG then its size is that of HMSEG and jump to step 23.

22. The mapping segment to be mapped is not the highest mapping segment therefore the number of pages to be mapped is equivalent to the standard mapping segment size.

23. Call MMAP to map the mapping segment with IPGS, mapping segment size and mapping segment # as parameters.

24. Return with B register = the address of the element calculated in step 15 at location RTN+1.

.EMIO SUBROUTINE

This routine is used when the user wants to access a certain portion of the external memory area and wants to insure that the entire portion will be mapped in the mapping segment. .EMIO checks if the requested length of buffer is contained within the current mapping segment or within any standard MSEG. If not, .EMIO maps the pages that contain this buffer into the MSEG logical memory.

.EMIO CALLING SEQUENCE

```
    JSB .EMIO
    DEF RTN        Return address
    DEF BUFL       # of words in the buffer
    DEF TABLE      Table containing the array parameters.
    DEF A          subscript value for the nth dimension
        n
    DEF A          subscript value for the (n-1)st dimension
        n-1
    .    .
    .    .
    .    .
    DEF A          subscript value for the 2nd dimension
        2
    DEF A          subscript value for the 1st dimension
        1
```

RTN error return A reg = 16 (ASCII), B reg = EM (ASCII)
RTN+1 normal return B reg = address of the element

Where the element is located at $(A_1, A_2, \ldots, A_{n-1}, A_n)$ in the array.
TABLE is the table containing array parameters as described in .EMAP routine and BUFL is the length of the buffer to be accessed. The leftmost dimension (A_1 is the subscript value) is varied the fastest to calculate the displacement.

.EMIO routine follows these steps:

1. Check IDEX location on base page. If contents are 0 then not an EMA program, return with error.

2. Get the buffer length and set up the pointer to the table of array parameters.

3. Calculate element address following steps 1-16 described under .EMAP.

4. Add buffer length and number of words displacement from start of MSEG up to the start of the buffer. Convert to pages.

5. If this number of pages is greater than the standard mapping segment size, go to Step 7 to map a non-standard mapping segment for this buffer.

6. Add the number of pages displacement from the start of the EMA up to start of the MSEG containing the element to the number of pages calculated in Step 4. If this resultant number of pages is greater than EMA size, then the buffer desired overflows beyond the EMA, make an error return. Otherwise the buffer fits inside the standard mapping segment, follow steps 17-23 described in .EMAP to map this standard mapping segment. Make a normal return to RTN+1 location with the address of the element in the B register.

7. The buffer does not fit within a standard mapping segment, therefore a special mapping segment must be mapped to include the entire buffer. The number of pages to be mapped is calculated by converting the number of words between the start of the page containing the element and the buffer length into pages. The number of pages offset will be the # of pages from the start of EMA up to the page containing the element.

8. Set MSEG # to -1. Call MMAP to map the non-standard mapping segment. MMAP will make an error return if the number of pages asked to map is greater than the standard MSEG size or it overflows the EMA.

9. Return to the calling program with the start address of the buffer in the B register.

MMAP SUBROUTINE

This routine is used to map a sequence of physical pages into the mapping segment area within the logical address space of the program.

MMAP CALLING SEQUENCE

```
        JSB MMAP
        DEF RTN              return address
        DEF IPGS             displacement in # of pages from start
                             of EMA up to the start of the mapping
                             segment to be mapped.
        DEF NPGS             # of pages to be mapped.
RTN

Returns:  A reg = 0 if normal return
                =-1 if error return
```

MMAP routine has three entry points:

MMAP is the main entry point into the routine, ..MP is used by .EMIO routine to map the non-standard mapping segment, .MMAP is used by .EMAP and .EMIO to map the standard mapping segment.

MMAP routine follows these steps:

1. If IPGS and NPGS are negative values then error return.

2. If EMA is not defined in the calling program, error return is made. Get EMA size from word 28 of the ID segment, and MSEG size from the first word of the ID segment extension of the program.

3. Divide IPGS by the mapping segment size. If the remainder is 0, the pages to be mapped start at a standard MSEG boundary and the quotient is the MSEG number to be mapped. If the remainder is not 0, a non-standard mapping segment has to be mapped.

4. If IPGS+NPGS is greater than EMA size or the number of pages to be mapped (NPGS) is greater than MSEG size, then an error return.

5. Next, MMAP adjusts the number of pages to be mapped (NPGS) to the standard MSEG size +1 (for the overflow page). If IPGS+MSEG size +1 is greater than EMA size, then NPGS is adjusted to EMA size-IPGS i.e., number of pages up to the end of EMA.

6. MMAP goes privileged at this point. Get the physical start page number of EMA from the second word of the program's ID segment extension. Add the IPGS to this value to get the physical start page number of the mapping segment.

7. The user map residing in the upper 32 locations of the unmapped portion of the user base page has to be changed to point to the new mapping segment. The DMS base page fence is set such that these upper locations cannot be accessed with the normal user map setting. To work around this, MMAP reads DMS register 40B, which is the first DMS register for the user map and contains the user base page number.

8. The system entry point $DVPT contains the logical start page number for the driver partition in the user map. MMAP changes the DMS register in the user map corresponding to $DVPT to point to the user base page.

9. The user map in the upper 32 locations of the user base page can now be addressed through $DVPT. The logical address at which MMAP starts changing MSEG page numbers is:

   MLOC=$DVPT*2000B+1740B+start logical page of MSEG.

10. The user map is changed for NPGS starting at physical start page of the segment to be mapped. If NPGS is less than the MSEG size+1, the rest of the pages in MSEG size+1 are read- write protected.

11. Transfer MSEG size + 1 locations starting at MLOC into the DMS register starting at (40B+logical start page of MSEG).

12. If MSEG number is -1, set bit 15 of the first word of the ID segment extension to indicate a non-standard mapping segment. Otherwise clear bit 15 and set up bits 5-14 of the first word of the ID segment extension to reflect the MSEG # mapped. Return.

EMAST SUBROUTINE

This routine returns the total EMA size, MSEG size and the starting logical page # for MSEG. All of this information is picked up from word 29 of the ID segment and words 0 and 1 of the ID segment extension of the currently executing program.

EMAST CALLING SEQUENCE

```
JSB EMAST                Return address
DEF RTN
DEF NEMA (returned)      Total EMA size
DEF NMSEG (returned)     Total MSEG size
DEF IMSEG (returned)     Starting logical page MSEG
```

RTN

Upon return A register = 0 if normal return and -1 if error return. An error return is made if an EMA does not exist i.e., XIDEX location on base page communication area is 0.

Chapter 7

EMA MICROCODE SPECIFICATIONS

MICROCODED ROUTINES

This specification reflects the following firmware parts:

92067-80001
92067-80002
92067-80003

The major time savings realized by microprogramming the .EMAP, .EMIO and MMAP routines derives from two areas. The first is handling map register modifications from the microcode without the overhead of going privileged in order to execute the appropriate DMS instruction. This will eliminate approximately 300 microseconds. The second area is common to both .EMAP and .EMIO and envolves the arithmetic calculations necessary to determine element address. Every subscript past one will cause an additional iteration through this calculation loop adding an approximate 40 microseconds software (10 microseconds microexecution). The following table summarizes timing differences for the software versus microcoded routines:

Software	Firmware
MMAP	49 + 1.2 x (# pages)
.EMAP no remapping	35 + 10 x (# dim)
.EMAP remapping	(firmware .EMAP always remaps)
.EMIO no remapping	43 + 10 x (# dim)
.EMIO remapping	90 + 10 x (# dim) + 1.2 (# pages)

All times are approximate and worst case in microseconds.

In order to modify maps from the microcode without DMS generating a memory protect violation, the microprogrammed map routines disable memory protect and re-enable it upon completion. This is accomplished by specifying IOG in the special field creating a memory protect violation. IAK is then specified in the special field of the by memory protect and it is disabled (assuming a non-I/O instruction is present in the instruction register IR). Memory Protect is re-enabled by building a STC 5 in the IR and jumping to the I/O group routine in the base set. Because of this, it is not possible to call the firmware EMA routines from privileged subroutines, although the software versions can be called from a privileged subroutine.

It should be noted that the operation of the firmware version of .EMAP is not identical to the software version. The calling sequence is the same and the result is the same (that is, the address of the element is returned in B). However, the firmware version of .EMAP does not use standard MSEGs. Instead, it maps in only the page containing the element and the following page, if possible.

Two procedures are available to verify the proper installation and operation of the firmware. A fourth opcode is provided which, when executed in single step mode, loads 102077 into the S register.

The procedure to run this test follows:

1. Set P to 0
2. Load A with 105242
3. Push Preset
4. Push Single Step

The display indicator should be S and the display register should be 102077. Any other result is an error. The firmware simply builds a 102077, puts it on the display register, sets the display indicator to S, and returns. It does test for single step so the instructions 105242 is a NOP if executed while running.

The second procedure is to run the EMA diagnostic (named #EMA) in RTE-IVB. This does a complete test of all the operations of the firmware.

The opcodes for the EMA microcode are:

.EMAP 105257
.EMIO 105240
MMAP 105241

The following discussion assumes that the EMA microcode is at hand and that the XE microprogramming manual is available. Address references are to micro control store addresses.

EMAP - MICRO CONTROL STORE ADDRESSES

22000-22017

These are the 16 available entry points into this module. EMAP is the last one so that a jump is not required. There is no reason for the entry points not being sequential.

22017

A read is started to fetch the array address. P is incremented to point to the table pointer. The flag is cleared to indicate to MMAP (if it is called) that is was called from a microroutine rather than with a 105241 opcode. Note that the return address is not fetched here. It is fetched in the exit code.

22020

Since M has the address of the array address, M-1 is the address of the return point. This is saved in Y for returning and in case of an interrupt, Y will have the location of the calling opcode +1.

22021-22034

GETPARM is a subroutine which fetches a parameter. The address is left in M. This section of code saves the array address and the table address. The address of the ID extension (1645) is built and put into S3, which is never used for anything else. The contents of 1645 is fetched and if zero, the non-EMA address calculations are used.

22035-22043

Word 1 of the ID.EXT, is fetched and masked to give the logical start of the MSEG. If the start of the array is below the log start MSEG, then it is not on EMA array.

22044

The RESOLVE subroutine calculates a 31-bit address offset into EMA from the subscripts provided.

22045-22051

This section of code saves the address within the page and converts the 31 bit address returned by RESOLVE to a physical page offset into EMA. If bits were shifted out then there was an error.

22051-22061

GETAD28 fetches word 28 of the ID segment. Next we put 1777B in L to do some masking. The LSB part of the 32 bit address is masked to give the address within the page of the desired element. S4 (which was loaded at 22040) is masked to give physical start page EMA. Finally, the EMA size is put into L and subtracted from the required offset. If the answer is positive then OFFSET> EMASIZE which is an error. If the result is -1, then the second page must be protected. This is signaled by setting the flag.

22062-22063

Add the physical start EMA to the offset to give the required physical page.

22064-22066

Form the logical start page of the MSEG in B.

22067

Check interrupts before we crash MP.

22070-22076

The IOG here forces a Memory Protect interrupt. The IAK four lines down acknowledges the interrupt and thus disables MP. On the next line CIR is checked to see who received the IAK. It can only be MP(SC5) or Powerfail(SC4). It is possible that a powerfail occurred between the JMP CNDX HOI and the IOG. If it did, we immediately bail out. Since IOG happens at T2 and IAK happens at T6, there is some dead time which can be used. First the logical start page of the MSEG is masked out and saved in X. Then a 20040B is built in S2. This will be used for setting the MEM address register.

22077

Since an interrupt always switches to the system map, MEM must be set back to the user map.

22100-22107

Since this revision of EMAP doesn't use standard MSEGs, the nonstandard MSEG bit must be set. The word is fetched and if that bit is not set, it is set and written back.

22110-22121

Now set the MEAR to map 40B which is the users base page. Get the users physical base page and put it in the log start of MSEG map register. Base page can then be updated by writing into location (log start address MSEG + 1740B + Log start Page MSEG). This was done so that it wasn't necessary to save a different map register and restore it when finished.

22122-22131

Now write into BP the two physical pages which will be mapped in. If the flag is set, the protect bits are set on the second page.

22132-22140

MEAR is set to the MAP at the logical start of MSEG. The physical pages required are in S9 and S10. While loading the map registers, there is some dead time due to the requirements for a READ, WRITE, or RJ30 exactly two micro-instructions before a Q_0, Q_1, or Q_2 micro-order. A STC5 will be required, so it is built now.

22141-22147

The address of the return address is in Y. This must be retrieved to do a proper return. The address which must be returned in the B register is the address of the element in the page plus the logical start of the MSEG. The return address is incremented by one because the normal return goes to rtn+1. The JMP IOG with STC5 in the instruction register is required to turn MP back on, since it was turned off by the IOG...IAK combination.

22150-22215

This section does the subscript calculation for non-EMA arrays. There is no magic here.

22216-22245

This is the error return section. The error code is dependent upon which EMA routine was called. However, they all take the same error exit. It is not possible to just check the Counter (which is the low 8 bits of the IR) because it may have been used. Therefore, the opcode must be fetched to determine the proper error return. The error return exits by doing a jump to fetch (location 0) rather than a return because the error routine may have been jumped to by a subroutine (such as EMAS). A RTN would return to the calling microprogram rather than terminating the microprogram.

EMIO - MICRO CONTROL STORE ADDRESSES

22246-22260

The buffer length is fetched and saved in S9. The address of the return address is saved in Y. The address of the table is put in X. 1645B (the address of the ID extension address) is built in S3.

22261-22267

The address of the ID ext.. is fetch. If it is zero, then error. Word 1 of the ID ext is retrieved via XLOAD and masked to give the log start address of the MSEG in S.

22270

EMAS will do the required subscript calculations and compute the MSEG required.

22271-22301

The address of the element in the page plus the buffer length gives the number of words which must be mapped. This is converted to pages and saved in S11.

22302-22305

If the number of pages for the buffer plus the offset is bigger than EMA size then error (buffer extends beyond end of EMA).

22306-22322

S1 is the offset in the MSEG to the element. S2 + S9 converted to pages gives the standard MSEG size required to hold the buffer. If this will fit in a standard MSEG, then we can map a standard MSEG as calculated by EMAS.

22323-22331

Must calculate the nonstandard MSEG necessary to hold the buffer. S1 is the logical address of the buffer. S5 is the number of pages needed to hold the buffer. S6 is set to the page displacement into EMA to the page containing the element. The flag is set to tell MMAP that it was called from a micro routine. MMAP02 is an entry point in MMAP. B is set to the address of the buffer and a normal exit taken.

22332

Come here if the buffer will fit in a standard MSEG. EMAT will map the proper MSEG if required.

22333-22342

This section of code gets the return address and sets P to rtn+1. It also checks to see if a STC5 is in the IR. If not, then a normal return is done. If there is a STC5 in the IR, then mapping occurred and we must return through the base set to turn MP back on.

MMAP - MICRO CONTROL STORE ADDRESSES

22343-22350

The flag is cleared so that the mapping routine will not return to a calling routine but will exit. The page displacement is fetched. During the dead time 1645 is built in S3.

22351-22361

Fetch the number of pages and put into S5. Get the ID extension address and put into SP.

22362-22402

The number of pages in a standard MSEG is retrieved from ID.EXT. word zero and saved in P. The logical start of the MSEG is fetched from word 1 of ID.eXT. and put in S9. The physical start of EMA is put in S. The displacement is added to physical start page to give physical start of MSEG in S7. The displacement is divided by MSEG size to see if a standard MSEG is being mapped (which is true if the remainder is zero) and to get the MSEG number.

22403-22407

The MSEG number (or all 1's if a nonstandard MSEG is being mapped) is put in S4. Subtract the number of pages to be mapped from the MSEG size. If the result is negative then error.

22410-22426

Get the EMA size and put in L. Add the Start page of EMA and save in B. This gives the last physical page which can be mapped. Next, the physical start page of the MSEG is added to the requested number of pages to give the last page requested. If (SPMSEG+REQSIZE) > (SPEMA+EMASZ), the request was for too many pages. If (SPMSEG+STAN.MSEG.SIZE) > (SPEMA+EMASIZE), then map only up to end of EMA, otherwise map a standard MSEG number of pages.

In other words map to the end of EMA or the MSEG, whichever comes first.

22427-22436

The map routines will always map the MSEG size +1 number of pages, if possible. This section checks to see if that last page is with EMA. If it is not then it must be protected. Then a check for a pending halt or interrupt is done.

22437-22442

Memory protect is disabled by forcing and acknowledging a MP interrupt. This is necessary in order to load base page and the DMS registers. The Central Interrupt Register is checked to see if the powerfail fail interrupt received the IAK. This is possible if a powerfail happened after the last interrupt check. Return to the base set if powerfail was pending.

22443

The interrupt forced a switch to the system map. DMS must be set back to the user map.

22444-22456

The constant 20040B will be used to set the Memory Expansion Address Register to the users physical base page. The page number is stored in S6. The content of user map register 1 is saved so that the base page can be accessed through map register 1. Map register 1 is then set to the users physical base page. Because of the requirement for a READ, RJ30 or WRITE exactly two instructions before some of the DMS instructions, there is some dead time. This is used to build some of the constants which will be required later.

22457-22463

The logical start page of the MSEG is computed to determine the first of the map registers to be changed. S3 is loaded with STANSIZE+1 because we always load that many map registers, although some may be protected.

22465-22471

The location 3740B is the start of the user maps in BP (remember the physical base page was loaded into user map register 1). The logical start page of MSEG is added to 3740B to give the place to start changing the map registers. This address is saved in S6. The counter is loaded with the number of maps to be done. L is cleared so no read/write protect bits are set.

22472-22503

This heroic loop writes the required map register contents into base page. The counter (previously S5) starts with the number of map register to be loaded but not read/write protected. The rest of the registers, up to MSEGSIZE+1, are read write protected. If cntr=MSEGSIZE+1 no pages are protected.

22504-22514

This section reads the map register contents from base page and loads the DMS registers. The counter must be decremented in the loop because ICNT followed immediately by JMP CNDX CNT8 does not always work. Note that the bottom 8 bits of the address is the negative of the number of times through the loop. In other words, if the counter counted up we stop when it gets to 000 (the last map to be loaded is 1777 in BP). Since the counter must count down, we use the negative of the bottom 8 bits of the address.

22515-22531

Restore user map register 1. Form ID EXT word 0 (MSEG# etc.) and cross store. In the meantime, build a STC 5 in the IR.

22532

If the flag was set, then we were called by a microroutine and must return. Otherwise return through IOG to turn on MP.

EMAS - MICRO CONTROL STORE ADDRESSES

22540

Resolve computes a 31-bit address from the array subscripts.

22541-22561

The standard MSEG size is fetched and put in S5. The log start of the MSEG is saved in S8. The required page is computed and saved in S2 and A. If the page displacement is greater than EMA size then error.

22562-22575

The offset to the element is divided by MSEG size to give MSEG number. MSEG number is saved in S4. The remainder is subtracted from the page offset to element to give page offset to MSEG. The remainder is converted to words and added to the bottom 10 bits of the 31-bit address to give the word offset in the MSEG. B is set to logical start MSEG. (Note: B+S1 gives address of element.)

22576-22660

This routine converts the array subscripts to a 31-bit address.

EMAT - MICRO CONTROL STORE ADDRESSES

Check to see if mapping is necessary.

22661-22667

The logical address of the element is formed in S1. Bit 15 of the ID EXT word zero is checked to see if a nonstandard MSEG is mapped. If bit 15 is set, then must remap.

22670-22674

The MSEG number in ID EXT word 0 is compared to the required MSEG number. If equal, then don't have to remap.

22675-22723

The Physical Start EMA is fetched. Physical start EMA + offset to MSEG gives Start Page MSEG. Next, the EMA size is divided by MSEG to give maximum MSEG number. If this is the MSEG required we use the remainder as MSEGSIZE, otherwise use the standard MSEGSIZE. An entry into MMAP is used to do the mapping. (Note the flag is set so MMAP returns.)

GETPARM - MICRO CONTROL STORE ADDRESSES

22724

Start a read in case this address is indirect. Get the results and load M.

22725-22730

If the last address was not indirect, return with a read on the parameter initiated. If it was indirect, fetch its contents and try again. If halt or int. is pending, go service it, unless it is single stepped.

22731-22732

The address of the EMA opcode+1 was saved in Y. P is reset to this value and the interrupt handled. The EMA microcode will be restarted.

22733-22742

This code used when interrupts are checked at various places.

POSDIV - MICRO CONTROL STORE ADDRESSES

22743-22747

Does a positive divide of A by L. Quotient in A, remainder in B.

GETAD28 - MICRO CONTROL STORE ADDRESSES

22750-22754

Get word 28 of the ID segment. Flows into XLOAD.

XLOAD - MICRO CONTROL STORE ADDRESSES

22755-22761

Does a cross load. Can't return on the same line as TAB because word Type II may not work right after TAB. Must have S11 in S-bus field so we can do conditional jumps based on retrieved value.

22762-22763

If powerfail occurred before the IAK then come here, reset P go to base set after IAK.

22764-22771

This section of code builds a 102077 in the S register. This can be executed from the front panel to verify proper installation of the ROMs.

Chapter 8

ON-LINE GENERATOR

INTRODUCTION

This section is intended to serve as an aid to the programmer when modifying the on-line generator program, RT4GN. It should be used in conjunction with the generator source listings, as it assumes familiarity with RTE-IVB and its system generation process. It assumes familiarity with the RTE-IVB On-Line Generator Reference Manual outlining the generation process.

The modularity of the RTE software makes it easy to configure a real-time operating system tailored to particular application requirements for input/output peripherals, instrumentation, program development, and user software. With the on-line generator a configuration can be achieved under control of the present RTE system, concurrent with other system activities. The on-line generation process utilizes the file management features of BSM for the retrieval of the generation parameters and software modules, for the output of the system bootstrap loader and for the actual storage of the absolute system code and its associated generation map. The special utility program SWTCH performs the switchover from the present system configuration to that of the new.

OPERATION

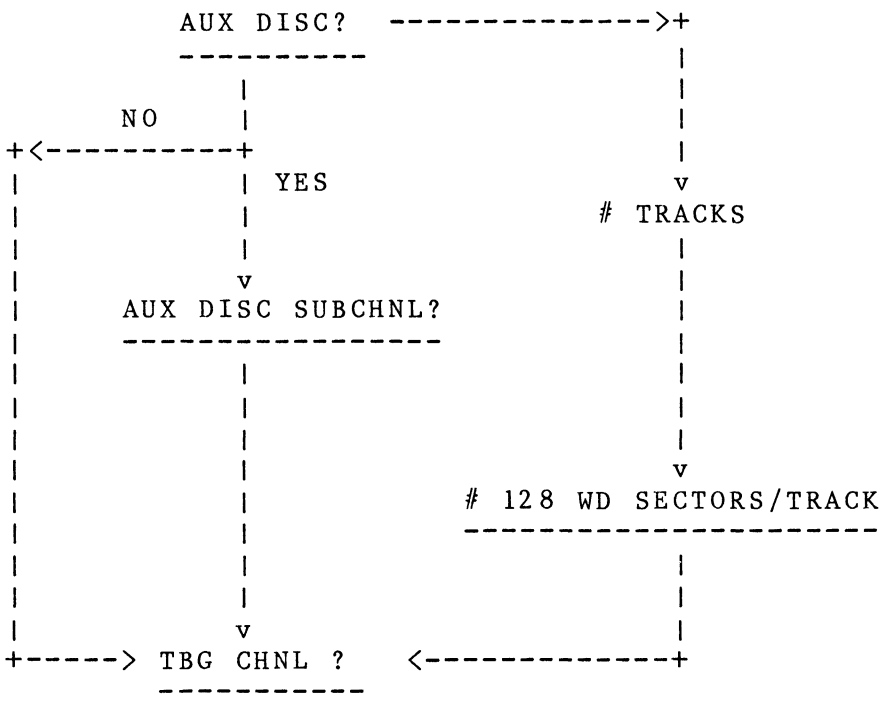
RT4GN is a type 2 or 3 segmented program which may be run as a type 2, 3 or 4 program in RTE-IVA/B (It will run in RTE-II/III, as type 2 or 3). The generator accepts its command input from an "answer" file located on disc, a logical unit, or a combination of the two. These parameters direct the generator in building and defining the system tables and values, the logical memory layout, the physical memory layout, and in relocating the software modules to be included in the system. All relocatable modules must exist in FMP file format and are specified by file name to be included in the system. The absolute, memory-image system being built is itself stored in a type 1 FMP file, which is then transferred by SWTCH.

ON-LINE GENERATOR

GENERATION SEQUENCE

```
LIST FILE NAMR?
--------------

ECHO?
--------------

OUTPUT FILE NAMR?
-----------------
------------------


for HP 7900/7901 Disc Only:

CONTROLLER SELECT CODE?
------------

#TRKS,FIRST TRK ON SUBCHNL?
 0?
--------,--------
.
.
.
or for HP 7905/06/20/25 Disc Only:

CONTROLLER SELECT CODE?
------------

MODEL,#TRKS,FIRST CYL,HEAD,# SURFACES,UNIT,# SPARES FOR SUBCHNL:
 00?
--------,--------,--------,--------,--------,--------
.
.
.
or for HP 7906H/20H/25H/9895 Discs Only:

MODEL,#TRKS,FIRST CYL,HEAD,#SURFACES,ADDRESS,#SPARES(,UNIT) FOR SUBCHNL:
--------,--------,--------,--------,--------,--------
```

SYSTEM SUBCHNL?



TBG SELECT CODE?

PRIV. INT. SELECT CODE?

MEM. RES. ACCESS TABLE AREA II?

RT MEMORY LOCK?

BG MEMORY LOCK?

SWAP DELAY?

MEM SIZE?

BOOT FILE NAMR?

ON-LINE GENERATOR

```
PROG INPUT PHASE:
-----------
-----------
-----------
-----------
.
.
.
-----------
-----------
/E

PARAMETERS
-----------
-----------
-----------
.
.
.
-----------
-----------
/E

CHANGE ENTS?
-----------
-----------
-----------
.
.
.
-----------
-----------
/E
```

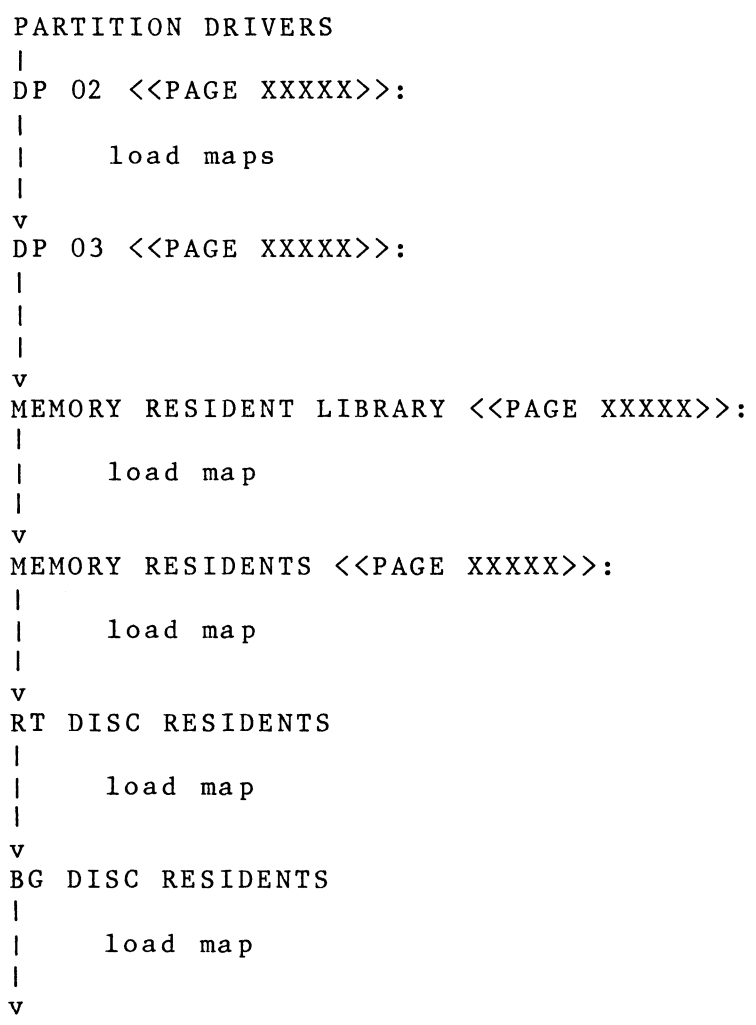
```
TABLE AREA I <<PAGE XXXXX>>:

EQUIPMENT TABLE ENTRY

EQT 01?
-------,-------,-------,-------,-------,-------,-------,-------

EQT 02?
-------,-------,-------,-------,-------,-------,-------,-------

EQT 03?
-------,-------,-------,-------,-------,-------,-------,-------

EQT 04?
-------,-------,-------,-------,-------,-------,-------,-------

EQT 05?
-------,-------,-------,-------,-------,-------,-------,-------
.
.
.
/E

DEVICE REFERENCE TABLE

001 = EQT #?
-------,-------

002 = EQT #?
-------,-------

003 = EQT #?
-------,-------

004 = EQT #?
-------,-------
.
.
.
/E
```

INTERRUPT TABLE

```
   4   ,   ENT   ,  $POWR
-------,-------,-------
-------,-------,-------
-------,-------,-------
-------,-------,-------
-------,-------,-------
-------,-------,-------
-------,-------,-------
-------,-------,-------
-------,-------,-------
-------,-------,-------
-------,-------,-------
-------,-------,-------
.
.
.
/E
```

```
TABLE AREA I MODULES
|
|
|   load map
|
v
DRIVR PART 00002
CHANGE DRIVR PART?
--
DP 01 <<PAGE XXXXX>>:
|
|
|   load map
|
v
SUBSYSTEM GLOBAL AREA <<PAGE XXXXX>>:
|
|
|   load map
|
v
RT COMMON XXXXX
CHANGE RT COMMON?
---------

RT COM ADD YYYYY

BG COMMON XXXXX
CHANGE BG COMMON?
BG COM ADD YYYYY
------

BG COMMON XXXXX
```

```
SYSTEM DRIVER AREA <<PAGE XXXXX>>:
|
|
|     load map
|
v
TABLE AREA II <<PAGE XXXXX>>:

# OF I/O CLASSES?
-------------

# OF LU MAPPINGS?
-------------

# OF RESOURCE NUMBERS?
-------------

BUFFER LIMITS (LOW,HIGH)?
-------,-------

XXXX LONG ID SEGMENTS USED
-------,-------

# OF BLANK LONG ID SEGMENTS?
-------

XXXX SHORT ID SEGMENTS USED
-------,------

# OF BLANK SHORT ID SEGMENTS?
-------

XXXX ID EXTENSIONS USED
-------,------

# OF BLANK ID EXTENSIONS?
-------

MAXIMUM # OF PARTITIONS?
-------

TABLE AREA II MODULES
|
|
|     load map
|
v
SYSTEM <<PAGE XXXXX>>:
|
|
|     load map
|
v
```




```
RT PARTITION REQMTS:
PNAME XX PAGES E
.
.
.
BG PARTITION REQMTS:
PNAME XX PAGES *E
.
.
.
```

```
MAXIMUM PROGRAM SIZE:
W/O COM YY PAGES
W/  COM ZZ PAGES
W/  TA2 XX PAGES

SYS AV MEM XXXXX WORDS

ENTER 1ST PARTITION PAGE:  XXXXX (DEFAULT) TO YYYYY:
-------

SYS AV MEM XXXXX WORDS

PAGES REMAINING:  XXXXX

DEFINE PARTITIONS
PART 01, XXXX PAGE?
.
.
.
SUBPARTITIONS?
.
PART 02, XXXX, (YYYY) PAGES?
.
.
/E

MODIFY PROGRAM PAGE REQUIREMENTS?
-------
-------
.
.
.
/E

ASSIGN PROGRAM PARTITIONS?
-------
-------
.
.
.
/E

SYSTEM STORED IN FILE
SYS SIZE: XXX TRKS, YYY SECS (ZZ SECTORS/TRACK)
          =TTTTT BLOCKS (128 WORDS/BLOCK)

RT4GN FINISHED

ZZZZ ERRORS
```

FILE INTERFACE

All I/O within the generator is handled through FMP calls, be it to answer, list, boot, echo, relocatable, absolute, or scratch files. Where I/O to a specific lu is allowed (answer file, list file, boot file, or echo) a dummy type 0 file DCB is created so that the same READF, WRITF, and CLOSE calls are used throughout. Six DCB's are set up and used (and sometimes reused) for file I/O:

\ADCB Absolute output file - always open to a file

\LDCB List file - always open to a file or lu

\IDCB Input file - always open to a file or lu (changes as TR's and errors occur)

\EDCB Echo - always open to lu of operator console but not necessarily used if \IDCB or \LDCB are used to same lu, or if option denied.

\RDCB Relocatable input file - used to reference all relocatable files during generation, open to only one file at a time

\BDCB Boot file - created only when boot file is output by PTBOT routine.

\NDCB Modified NAM records file (@@NM@A) - scratch file open when being built and when referenced during relocation

All files except the answer file(s) and relocatable input files are created by the generator. The above two file categories cannot be actual type 0 files, as the generator may reference them by record number. In the case of the relocatable files, the generator actually opens and closes each file many times.

INTERFACE ROUTINES

\CRET - is passed a DCB address and creates a file whose name is at PARS2+1,+2,+3, security code is at PARS3+1, and crn is at PARS4+1, and size is at PARS6+1.

\CRET first calls FOPEN which calls TYP0 - if a type 0 dummy DCB was built then that is sufficient and \CRET returns. If it was a file, then that file is closed (no error check done here on file since may never have existed), and then created by a CREAT call.

A CREAT call is made with the assumption that whoever called \CRET checks the FMP error parameter FMRR.

\CLOS - is passed the DCB address and truncate option. For a file, a simple CLOSE call is made, leaving the \CLOS caller the responsibility of checking \FMRR (not usually done).

For a dummy type 0 file, however, word 9 is merely set to 0. If the type 0 file being closed is the list file, then a page-eject control request is made to it. The no-abort bit is set on the control request to prevent abortion of the generator to a device with no EOF code (like the console).

\OPEN - is passed a DCB address, and attempts to open a file whose name is in PARS2+1,+2,+3, security code in PARS3+1, and crn in PARS4+1.

A call to TYP0 determines if a lu was specified in the first parameter and TYP0 sets up the dummy DCB \FMRR is always cleared).

For a file, an OPEN call is made leaving the check of FMRR up to the caller of \OPEN.

TYP0 - is passed the DCB address in the A-register. It determines whether a numeric parameter was specified as a file name, in which case it will continue with the building of a dummy DCB. LU's are allowed by the generator for answer, list and boot files; echo is always to the lu of the operator console (ERRLU).

The dummy DCB format and initial values are:

```
Word 0    0     directory entry address
Word 1    0        of file
Word 2    0     type
Word 3    read/write subfunction, lu
Word 4    EOF control subfunction, lu
Word 5    0 no spacing legal
Word 6    100001 read/write legal
Word 7    100030 security codes agree; update open
Word 8    -----
Word 9    ID segment address of generator (from 1717)
Word 10   -----
Word 11   -----
Word 12   -----
Word 13   1 (initial value) Record Number (Low Word)
Word 14   ecord Number (High Word)
```

Special checks are made in determining the EOF control of subfunctions. For driver types >=17 and for DVR05 exclusive of subchannel 0, a 0100 is merged with the lu. For DVR00, DVR02, DVR05, and DVR07 (subchannel 0 only), the EOF control subfunction 1000 is merged with the lu. For all other driver types between 1 and 16, 1100 is the merged subfunction. For non-type 05 or 23 devices, an EOF will be sent immediately-causing leader or a page eject, respectively.

SCRATCH FILE

The generator creates a temporary file of its own for storage of modified NAM records, @@NM@A. Modified NAM records result when the program length of a compiled program has been determined (during the Program Input Phase), or when a program's priority or execution interval are changed during the Parameter Phase. If such a modified NAM record does exist for a program, then bit 14 of ID5 in its IDENT entry is set so that the correct values may be retrieved during relocation.

The generator purges this scratch file during final clean-up or its own abortion clean-up. The file will still remain, however, if the generation is aborted by some other means. When the generator tries to create the scratch file during initialization and finds that it already exists, it will increment the last character of the name (e.g., A>B) and create a new one. It gets confused if there exist old entries in a file left over from a previous generation, so a new file is always created.

RELOCATABLE INPUT

All relocatable input is handled through the routines \RNAM and \RBIN (both in the main). \RNAM sets up the parse buffer to open the file specified in the current IDENT entry (words \ID9 through \ID13). A non-zero B-register on entry to \RNAM lets us assume that the file is still open. Otherwise, the relocatable file currently open to \RDCB is closed. \RBIN is called to (possibly) open the file, and to read the record specified by \ID14 through \ID16. \RBIN may also be called to merely read the next relocatable record of a file, and optionally to get its position.

ANSWER FILE

Upon start-up, the generator determines through RMPAR and GETST calls whether an answer file name or lu was specified via the turn-on parameters. If the first parameter was 0, lu 1 will become the default command (answer) lu. If parameter 1 was numeric, that lu will be used for command input (in an MTM environment, this would be the operator console's lu provided no parameters were specified). A dummy DCB will be created in TYP0 for the lu, or the answer file specified in the Namr parameters will be opened via FMP. If an error occurs on the answer file open call, the appropriate error message will be displayed on the console via an EXEC call, and control will be transferred to lu 1.

An "error lu" is also defined at start-up. If an lu was obtained from either the turn-on parameter or the default command lu 1, then that lu becomes the error lu provided it represented an interactive device. If it was not interactive, the photoreader for example, then the error lu would default to lu 1.

When an error occurs, the error message(s) is sent to both the list file and the error lu. For many errors, control will be transferred to the error lu for corrective action by the operator. This is done by stuffing a "TR,ERRLU" into the command buffer, where ERRLU represents the two digit error lu. The error processor \GNER then calls TRCHK which processes the TR command. If the command input was already from an interactive lu then the control is not transferred from it.

All command input is handled by the \PRMT routine, which also issues the prompting message. \PRMT filters the input looking for a !! starting in Column 1 - indicating the operator wishes to abort the generator; or for a , : or TR - indicating that a transfer is to be done. An EOF encountered in an answer file/lu results in the simulation of a "TR" command which pops the input stack.

The parse routine \PARS is called with the input buffer address, and returns the parameters in the following format. Parameter 2 is the file name or lu, Parameter 3 the security code, Parameter 4 the CRN, Parameter 5 the file type, and Parameter 6 the file size.

PARS2,3,4,5,6,	Type: 0=null,	1=numeric,	2=ASCII
PARS2+1,3+1,4+1,5+1,6+1,	0	number	char 1 & 2
PARS2+2,3+2,4+2,5+2,6+2,	0	0	char 3 & 4
PARS2+3,3+3,4+3,5+3,6+3,	0	0	char 5 & 6

Asterisks (*) are not allowed within filenames, security codes, file size or cartridge labels. As soon as an * is encountered the beginning of a comment is assumed and \PARS returns.

\PRMT does some checks to determine whether or not to send the response just received to the list file. If the list file is to the lu of the operator console and if that's the current command input lu (CMDLU) then the response is not sent; otherwise \LOUT is called (\LOUT does more checks for echoing).

TRCHK determines if the command input stack is to be pushed or popped. If the current command buffer contains a TR (or : or,) with no parameter, then the stack is popped to the previous source of command input; otherwise the stack is pushed with the new element. Ten entries may be placed on the stack by the user. (GEN ERR 19 on overflow or underflow) with each entry of the form:

Word 0 entry type: 1=Type 0(lu), 2=file
Word 1 lu, else CH1 and CH2
Word 2 0, else CH3 & CH4
Word 3 0, else CH5 & CH6
Word 4 Security Code
Word 5 Crn
Word 6 0,else record count for next record to read

An eleventh entry to lu 1 is hard-coded at the bottom of the stack.

On a transfer, the current file is first closed. The routine PUSH will then save the next record number of that file in its stack entry, for repositioning when the file is later reopened. PUSH then picks up the file name/lu from the parse buffer and builds the new stack entry. If overflow results, no push is done (recovery handled in TRCHK). POP on the other hand merely decrements the STACK pointers to the previous entry - on underflow no pop is done and TRCHK handles the recovery. Before returning to TRCHK, both PUSH and POP call the routine STATE which performs status checks on the new source of command input setting CMDLU (0, else input lu) and IACOM (1 if an interactive lu, 0 if a file name or non-interactive lu). IACOM is used in determining the echo of input/output to the list file or console. STATE also checks the validity of an lu specified as the new command input source. If invalid, STATE does an error return, as does PUSH, and TRCHK issues a GEN ERR 20 and handles the recovery. This error will not occur on a POP as the command input source we are returning to would have already been checked at the original transfer.

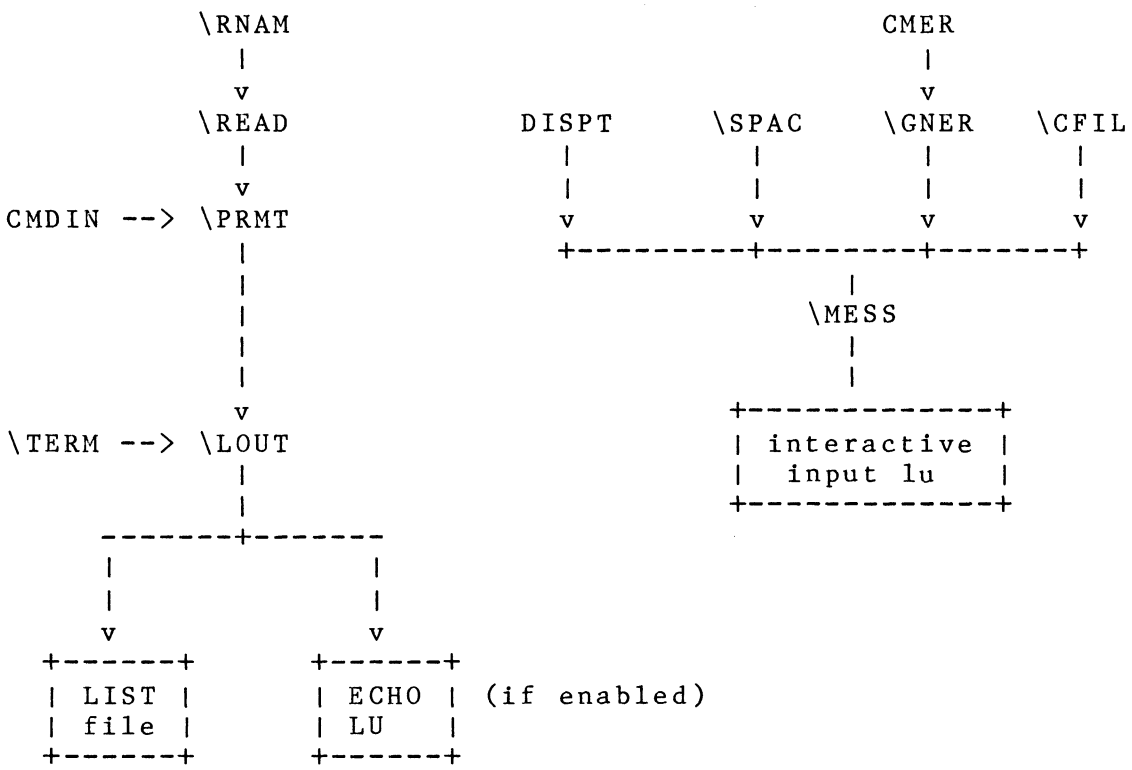


Figure 8-1. Tree Structure for Generator Command Input and Echo/List Output Routines

LIST FILE

The operator chooses where they wish to send the generation listed output. If a file name was specified without a file size, then a file of size 64 blocks (default size) is created by the generator (extents are created as needed by FMP). Since a CREAT assumes an exclusive open, the file is then re-OPENED with the non-exclusive option. This permits examination of the list file concurrent with generation.

For list output to an lu, a dummy DCB is created in the routine TYPO. If the lu specified is to a non-interactive device, then an attempt is made to lock it. If unsuccessful, the generator issues the appropriate message (not in the form of an error) and reissues the lu lock call with the wait bit set. The generator is suspended until that time when the resource becomes available.

If the lu was interactive then the flag IALST is set to 1. IALST and IACOM are then used in the list output routines \MESS and \LOUT to prevent duplicate output to the operator console. A line is always sent to the list file (using \LDCB) via \LOUT. If the list file was not an interactive lu (IALST=0) but the command input/answer file was (IACOM=1), then the line is sent to the operator console (using \IDCB) as well. The status of the command input mode reflected in IACOM changes as TR's are encountered or errors are detected. Therefore, it is necessary to perform these checks every time list output is done.

See the Error processing section for the handling of list file errors.

ECHO

The operator must always answer the ECHO? prompt even where not applicable (as when list file is lu of operator console, or generator is to be directed interactively). If the operator requests an echo, then checks are made in \LOUT to see if both IALST and IACOM are equal to 0, meaning neither the listed output or command input are an interactive device. If IALST or IACOM indicate an interactive lu, then one further check is made with either LSTLU or CMDLU against ERRLU to see if they represent the same interactive lu, in which case no ECHO is done. If more than one lu points to a particular interactive device, no checks are done to determine if they reference the same EQT. Note that because echo is dependent on the command input mode it may change as TR's are done to and from the operator console or when error mode is enabled.

BOOTSTRAP FILE

At \BOT0,\BOT5 the moving head bootstrap loader may be sent to an FMP file or to an lu; a 0 response by the operator implies no boot is desired. The "0" must be specifically checked for as the TYP0 routine defaults to Lu 1 when a 0 lu is given - which is where the boot file would go in this case, and garbage (actually absolute code) would appear on the system console. If an lu was specified for the boot file output, an EOF will be written (e.g. trailer for a paper tape bootstrap). On an abortive termination \TERM purges the boot file if one was created, otherwise \TERM simply closes the file.

ABSOLUTE OUTPUT

The absolute output file for the system being generated must be a type 1 file because of its fixed length, quick random access features. Because type 1 files are not extendible, the user must over-estimate the number of blocks in the new system, with the routine \TRUN truncating any excess file space at the end of the generation. The file size specified is checked for a 1000 block minimum, as is a size so large that it results in a negative number of blocks. If the size specified is less than 1000 blocks, a GEN ERR 17 is issued. If the file created is too small, and overflows, a GEN ERR 17 will occur at that point where the end-of-file is passed, usually a considerable way into the generation. Because the generator has to abort, the user usually has to learn this lesson only once!

Since RTE requires locations of items on the disc, disc addresses within the generated system are relative to the start of the disc subchannel and thus are relative to the start of the absolute output file (exclusive of header records).

ABSOLUTE OUTPUT FILE TRUNCATION

The routine \TRUN is called at the end of the generation during the cleanup processing. It closes the absolute output file, truncating any unused file area. Immediately preceding the call is a forced access to the last record +1 (using \DSKI) to make the truncate work. A special situation results when the area used by the new system is exactly equal to the created size of the file. When the forced access is done, an FMP-12 error occurs because we've overflowed the file. The record was obviously not read in (makes no difference because it was a dummy read anyway), and most important, the track and sector addresses in \ADCB were not set. So \TRUN must check the FMP error code still in FMRR from DISKD's READF call - and if it's negative, no truncation is to be done.

The truncation is done by determining the number of unused blocks in the file, and deleting them in the \CLOS call:

(# blocks in file) - (current block #) = TMP

where the current block # is obtained as follows:

(last file track - first file track)*#sectors per track - (first file sector + last file sector) *2

HEADER RECORDS

The absolute output file contains two, 128-word header records. These are used as a means of passing pertinent information to SWTCH, without SWTCH having to go searching through the absolute output file for it. The track map table (TMT) buffer at TB30 is used for storage of the header record information as it's being built.

The header records have a different format, depending on the type of the system disc. For 7900 based systems, refer to Figure 8-3. For MAC/ICD (7905/06(H)/20(H)/25(H)) systems, refer to Figure 8-4.

Header Record #1 contains the system disc track map table, and is sent to the disc just prior to loading Table Area I modules. This occurs at \TB32 in Segment 7 for MAC/ICD Discs (\TB31 in Segment 1 for 7900 Discs).

> Note that for MAC/ICD systems, the TMT is 160 words long, and therefore extends into Header Record #2. After the TMT is loaded into Table Area I, the information in the 160 word TMT buffer at TB30 is reorganized at \TB32 so that TB30 becomes a Header Record #2 image area.

Header Record #2 is built in bits and pieces.

The EQT information is obtained while the EQT's are being built. IOADD, the channel, is placed in the high byte, and IOTYP, the EQT type, is placed in the low byte of word pointed to by HEADR (the next header record entry). When the lu of the system console has been assigned, the corresponding channel is retrieved. Header Record #2 is sent to the disc at the very end of the generation in Segment 3, which calls \FSC5 in Segment 7 for MAC/ICD Discs (\FSC0 in Segment 1 for 7900 Discs).

This is used so the RTE-IVB version of SWTCH can check against RT2GN/RT3GN-produced output files.

HEADER RECORD #1 (128 words)

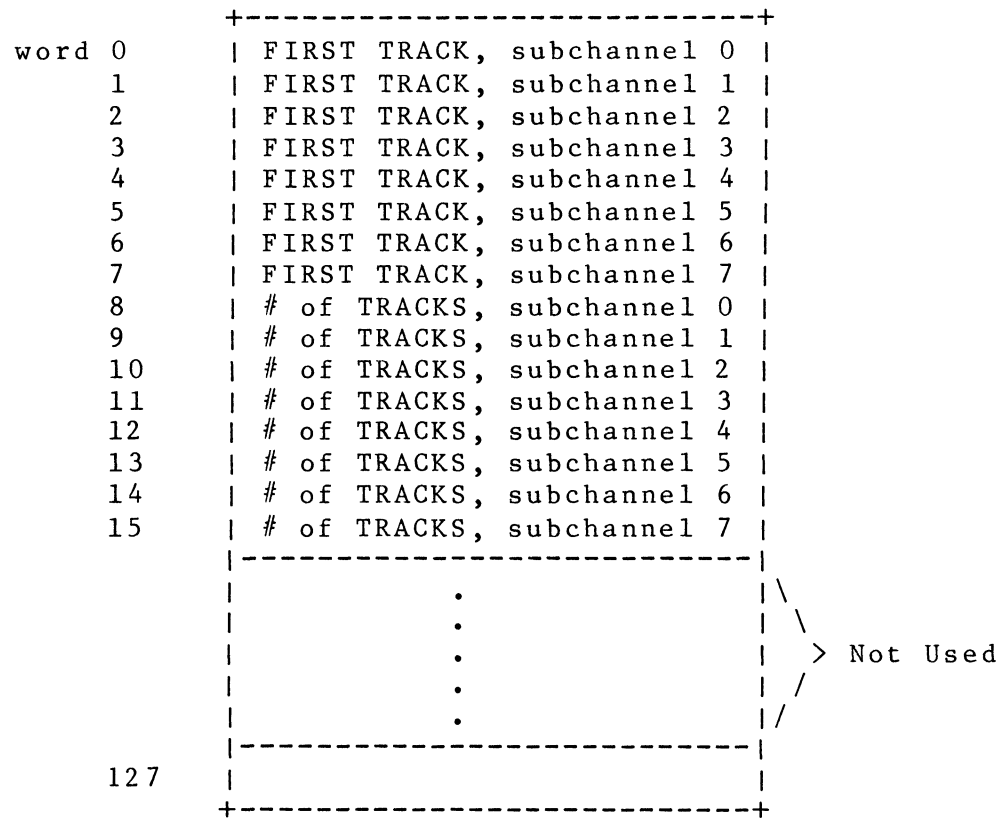


Figure 8-2. Header Record Format - 7900 Disc

HEADER RECORD #2 (128 words)

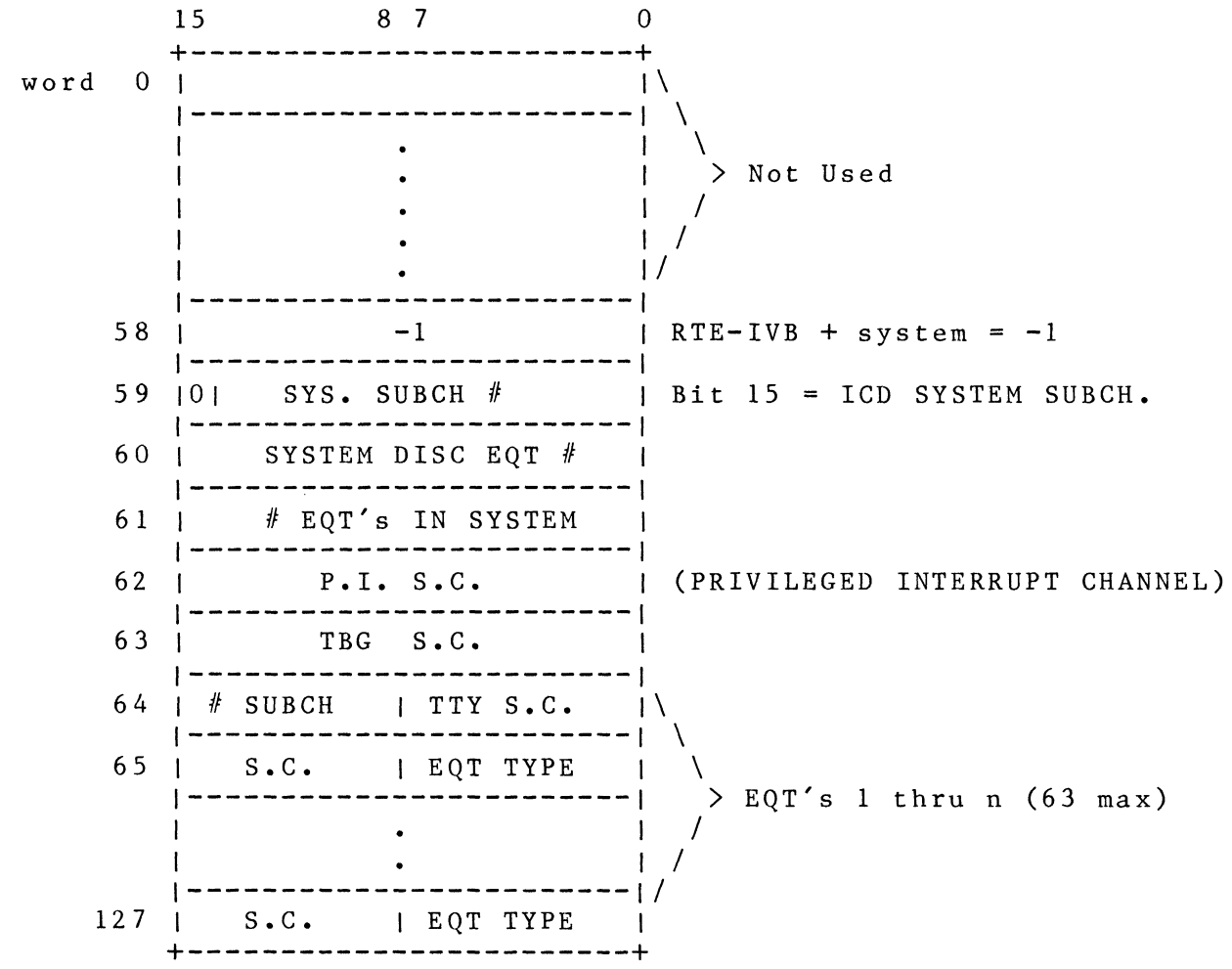


Figure 8-2. Header Record Format - 7900 Disc (Cont.)

HEADER RECORD #1 FORMAT

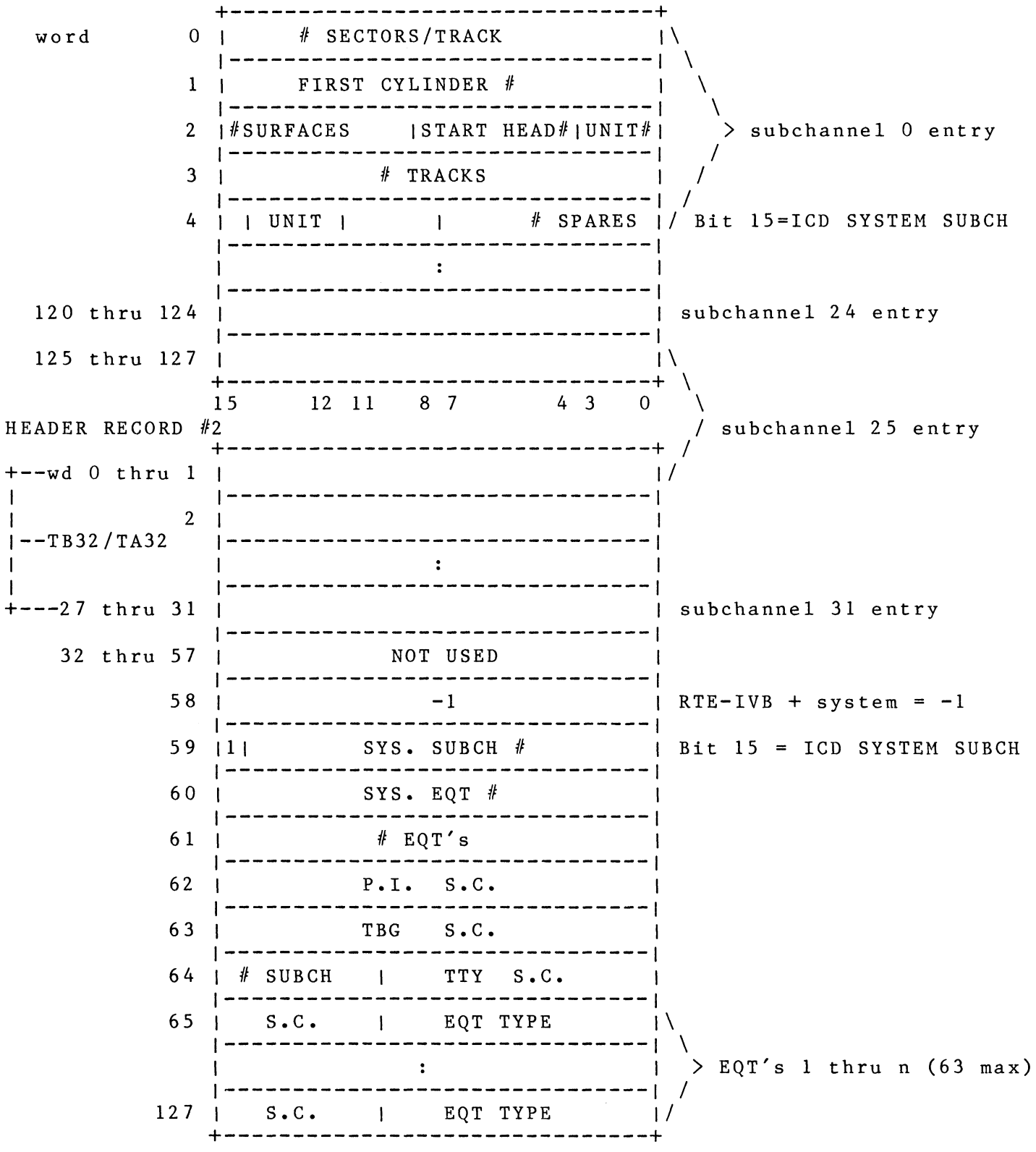


Figure 8-3. Header Record Format for 7905/06(H)/20(H)/25(H) Discs

OUTPUT ROUTINES

The following 5 routines control the output of code to the core-image output file.

\ABDO May be the most useful routine in the generator. It is used to read or write words from the absolute output file, using the memory address of the word in the target system. \ABDO operates using a 3-word map giving the base disc location, the memory address referencing that location, and the highest memory address referenced using this map. \ABDO puts out the current absolute code word (in A-register) at the memory address (in B-register) under the current map. Gaps are filled with zero codes if the current word falls beyond the highest previously generated word. For convenience, three "maps" are automatically maintained. These are for the system, a driver partition or a user program, and a user program segment. Subroutines \SYS, \USER, and \SEGS are used to activate these respective maps. Other maps could be constructed and activated by SETDS.

\DSKD Translates a disc address to a record number in the Type 1 absolute output file, thus satisfying the file's random access nature. FMP READF and WRITE calls are used to access the output file in 128-word (sector-sized) chunks. Note that an FMP-12 error (EOF sensed) is ignored on READF. The reason behind this is that during clean-up, there is a forced access to the last record used, +1, before \TRUN is called. This is to set up the proper track addresses in the DCB \ADCB so \TRUN can truncate the absolute output file to the last referenced record. \DSKI is used to read in that record but it is never rewritten (where the error would be sensed).

In the special case when the header records are written, the memory address in the B-register is set to negative to indicate record #1. For record #2 the A-register containing the disc address as well as the B-register are negative. See Header Record Section for the record formats.

\DSKA Increments the current disc address to that of the succeeding sector. SDS# is used in determining track crossings and equals the number of 64 word sectors per track.

\DSKI Controls input from the disc (at specified disc and core addresses) and uses a buffer making the disc appear to have 64 word sectors.

\DSKO Controls output to the disc as \DSKI controls input.

PROGRAM INPUT PHASE

During the Program Input Phase (PIP), the generator will accept one of many responses to its prompt. Because of this, segment 2 contains its own command scanner and branch mnemonic table. At PRCMD/NXTCM the routine CMDIN is called to retrieve the next command, SCAN is called to determine the correct command processor, and control is sent to that processor (who returns to NXTCM unless a /E was entered). CMDIN issues the prompt and receives the command via \PRMT (which filters and handles all TR's and etc.).

SCAN is used by PRCMD/NXTCM to search for the command keyword, returning a processor index in the A-register. Scan is also called by some of the individual command processors to search only their portion of the command table such as MODULES, GLOBALS, LINKS, OFF or ALL for the MAP command. PTABL is the branch table for the seven keyword commands (note the existence of entries for both RELOCATE and REL). CMDIN catches blank or comment (*) line, QGETC catches all * following the PIP commands (on the same line), while GETAL catches all those following non-PIP commands - in both cases the remainder of the line is ignored for processing.

The command mnemonic table CTABL contains an entry for each allowable keyword - if an abbreviation is exactly the same as the beginning of a longer keyword (e.g. REL and RELOCATE) the longer must appear first. Each entry in the table contains two pieces of information: bits 15-8 indicate the number of characters in the ASCII keyword, and bits 7-0 contain the offset into table CMTBL of the ASCII value for that keyword. The ordering of entries in both PTABL and CTABL must not be changed, whereas that of CMTBL is of no importance.

When called by the individual processors (of DISPLAY, MAP or LINKS) SCAN operates on their individual keyword tables (LTABS, MTABS and ITAB respectively), again returning the matched keyword index in the A-register.

DISPLAY COMMAND PROCESSOR

DSPST makes use of the error return (P+1) from SCAN, when neither UNDEFS or TABLE were indicated, and consequently searches the LST for an entry point. If not found, the entry point name followed by an "UNDEFINED" is printed.

DSPST will always send its output to both the operator console and list file. Therefore it is necessary for RT4GN to simulate a TR to the operator console (unless already in the interactive mode), do the display, and then pop that TR from the stack. The operator never realizes that this even happens. If either TABLE or UNDEFS was specified, then the routine EPL is utilized (A-register on entry = 1 means list LST entry points, = 0 means list LST undefined externals). externals). If a "DISPLAY UNDEFS,TR" is requested, the 'pop' will not be done if there existed any undefined externals (A-register is non-zero on return from EPL).

REL(OCATE) COMMAND PROCESSOR

RELST is the most complicated PIP command processor because of the optional module name specification allowed before the file name. Because an lu cannot be specified for relocatable input, special checks are done at CHFNM to insure that one was not entered. \RNAM and \RBIN will catch an invalid file name. Note that because of the special format of the RELOCATE command, no comma is specified until immediately before the filename. \PARS will place the filename, security code and crn in the usual parse buffer locations (see File Interface). If a module name was specified, RELST stores the name in buffer XNAM; if none specified, XNAM's value is 0. When a NAM record is read, checks are performed at LDRC3 to determine whether or not to load the module. If no module name was specified in XNAM, then the entire file is unconditionally loaded (note that "loaded" here refers to module entries being placed in the LST and IDENT tables; actual relocation is done later). If a module name was specified, then it is determined if there is a match between the names of the XNAM module and the current NAM. If no match resulted then those relocatable records through the next END record are skipped, otherwise that module only is loaded. Two variables are used to control loading: SERFG = 0 indicates a module is to be loaded, -1 that it is to be skipped; NAMR = 0 when a NAM record is expected (either at beginning of file, after an END record, or in record skipping mode); = 1 if no record is expected.

MAP COMMAND PROCESSOR

MAPST controls the memory map listing during the relocation phases. The value of MAPMD is stored in bits 3-0 of ID5 of all IDENT entries. MAPMD settings (bit 0 for globals, bit 1 for modules, bit 2 for links) are in effect for all IDENTS entered through subsequent RELOCATE commands. Options can be turned off only by entering a MAP OFF command, and then entering another MAP command listing the desired options (more than one can be specified). A MAP statement is processed left to right - therefore MAP MODULES,

ALL,LINKS,OFF,GLOBALS would result in GLOBALS only being mapped (MAPMD = 000001). The initial (default) value of MAPMD is off.

LINKS COMMAND PROCESSOR

LNKST controls the linkage mode during relocation. As for the MAP command, the option specified is in effect for all IDENTS entered through subsequent RELOCATE commands. The initial (default) value for LNKMD is 0 for base page mode; 1 indicates current page mode.

The value of LNKMD is stored in bit 15 of all IDENT entries. Current page linking is never done on assembled type 3,4 or 5 programs (and their variations).

/E COMMAND PROCESSOR

EOL terminates the Program Input Phase by exiting through PRCMD's success return. Before returning however, it calls EPL indicating a listing of the undefined EXT's (if any) - this listing goes only to the list file (not to operator console, as did the DISPLAY UNDEFS command).

IDENT, LST, AND FIXUP TABLE STRUCTURES

The IDENT table contains an entry for each relocatable module which is specified by RELOCATE commands during the Program Input Phase. There is one entry in the LST for each entry point defined in each relocatable module; entries are also created by the generator (e.g. for $CLAS, $LUSW, $RNTB, $LUAV, $TB31/2) and for those ENT's supplied by the user during the Parameter Phase. FIXUP entries are used during relocation when an entry point is accessed before it has been defined (no address in the LST).

These three tables are stored in the available memory space starting at the first word following the end of Segment 3, and ending at LWAM. Note that the assumption is made that Segment 3 is the largest segment.

There must exist at least 512 words of undeclared memory in order to insure at least one sector's worth of words for each table. Initially the space is allocated: 1/4th for the FIXUP table and 3/8th's each for the LST and IDENT tables. Once a block of space is allocated, it is truncated to a sector-multiple number of words. The block size must also be divisible into the track size (so that when many blocks are swapped out none will cross a track boundary); thus a block may be truncated further by one or more sectors. All truncated words are collected and added to the LST block, as it usually needs the greatest space and is accessed the most. Its block size must still fulfill the above two restrictions, however. The maximum block size is WDTK words (i.e., one track's worth of information).

Six tracks are obtained for the swapping of these tables: 1 for the FIXUP, 2 for the LST, and 3 for the IDENT. If these tracks cannot be obtained, the generator issues the appropriate message and suspends itself (by re-issuing the call without the no-suspend bit) until the tracks become available.

POINTERS AND INDICES

The following variables are set up for manipulating the three tables:

Table 8-1. FIXUP, IDENT, LST

	FIXUP	IDENT	LST
# sectors per block	FX.#S	ID.#S	LS.#S
# words per block	LFIX	LIDNT	LLST
# entries per block	EFIX	EIDNT	ELST
core address of block	BFIX	BIDNT	BLST
# entries used	\PFIX	\PIDN*	\PLST
current entry index	\TFIX	\TIDN*	\TLST
index of first entry in core	B.F	B.I*	B.L
# swap tracks	FIX#T	IDT#T	LST#T
starting swap track	FX.BT	ID.BT	LS.BT
last swap track + 1	FX.ET	ID.ET	LS.ET
last swap track read	FX.LT	ID.LT	LS.LT
last swap sector read	FX.LS	ID.LS	LS.LS
current swap track	FX.CT	ID.CT	LS.CT
current swap sector	FX.CS	ID.CS	LS.CS

*IDENT indices start at 10 (rather than 0 as for the FIXUP & LST indices) so as not to be confused with values <10 in \LST4.

For each table there exists a set of address pointers, one for each word in the current table entry. \FIX1 through \FIX4 point to the FIXUP entry whose index is \TFIX-1; \ID1 through \ID16 to the IDENT entry whose index is \TIDN-1 (includes the +10 offset); and \LST1 through \LST5 to the LST entry whose index is \TLST-1.

TABLE PROCESS ROUTINES

There exists a set of similar routines that process the three tables:

\INID,\ILST,\IFIX sets the entry index \TIDN,\TLST,\TFIX to the first entry in the current core block - 10 for \TIDN and 0 for TLST and TFIX.

\IDXS and \LSTS search the table for an entry name matching that in the B-register address. The address pointers are set to either the matching entry, or to the next free entry if no match was found.

\IDX,\LSTX,\FIXX insure that the proper block (containing the entry \TIDN,\TLST,\TFIX) is in core. A check is made against B.I,B.L,B.F (which are the indices of the first entry in the current core block), and against the last entry index (determined by adding EIDNT,ENST, EFIX respectively). If the block containing the desired entry is not in core, the track and sector address of its swap area is determined, and it is brought in via RDIDN,RDSMB,RDFIX. The address pointers for the desired entry are also set.

RDIDN,RDSMB,RDFIX write the current block out to its swap track area, and brings the desired block into the memory location allocated to that particular table.

\LSTE searches the LST (via the routine \LSTS) for a particular symbol, entering it in the LST if not already there.

Table 8-2. IDENT Table Entry Format

```
word 1:   ID1 - NAME 1,2
word 2:   ID2 - NAME 3,4
word 3:   ID3 - (15-8) NAME 5
                ( 2-0) USAGE FLAGS:
                       2 this module was loaded as part of a segment
                       1 must load this module (ext defined by it)
                       0 this module was loaded
word 4:   ID4 - (15)   set if main program module
                (14-0) COMMON LENGTH
word 5:   ID5 - (15)   BASE/CURRENT PAGE LINKING FLAG=1/0
                (14)   NEW NAM RECORD FLAG
                (13-4) EMA SIZE
                (3-0)  MAP OPTIONS:
                       2 links
                       1 modules
                       0 globals
word 6:   ID6 - (15)   EMA DECLARED
                (14-10)MSEG SIZE
                (9-8)  NOT USED
                (7)    "DON'T DUPLICATE" FLAG
                (6-0)  TYPE
                       4 SSGA
                       3 REVERSE COMMON
word 7:   ID7 - LOWEST DBL ADDRESS
word 8:   ID8 - DISK LENGTH FOR UTILITY RELOCATABLES
             or MAIN IDENT INDEX FOR SEGMENTS
             or (15-8) PAGE REQUIREMENTS
                (7-0)  KEYWORD INDEX
             or (15)   EQT defined
                (14)   SDA declared
                (13)   SDA own mapping
                (13-0) DRIVER LENGTH
word 9:    ID9  - FILE NAME 1,2
word 10:   ID10 - FILE NAME 3,4
word 11:   ID11 - FILE NAME 5,6
word 12:   ID12 - SECURITY CODE
word 13:   ID13 - CARTRIDGE LABEL
word 14:   ID14 - RECORD NUMBER
word 15:   ID15 - RELATIVE BLOCK
word 16:   ID16 - BLOCK OFFSET
```

Table 8-3. LST Entry Format

```
word 1:   LST1 - NAME 1,2
word 2:   LST2 - NAME 3,4
word 3:   LST3 - (15-8) NAME 5
                 (7-0)  ORDINAL
word 4:   LST4 - IDENT INDEX
                 or 2 if COMMON
                 or 3 if ABSOLUTE
                 or 4 if REPLACE
                 or 5 if UNDEFINED
                 or 6 if EMA
word 5:   LST5 - SYMBOL VALUE
                 or IDENT INDEX if EMA
```

Table 8-4. Fixup Table Entry Format

```
word 1:  FIX1 - MEMORY ADDRESS
word 2:  FIX2 - (15-11) instr. code
                (10) byte instr. indicator
                (9) upper BP link indicator
                (2-0) DBL record type
word 3:  FIX3 - OFFSET
word 4:  FIX4 - INDEX OF LST ENTRY REFERENCED
                0 if local symbol
                -1 if .ZRNT
```

LST INDEX FOR .ZRNT

Since the indices to the LST entries begin with zero (unfortunately) there may be confusion with the value of FIX4,I whose value is either the index of the LST entry referenced, or zero for no reference. It turns out the .ZRNT is always the first entry in the LST because the generator places it there, so it always has the 0 index. Therefore, during the DBL relocation processing in segment 4 (at DBL45 to be exact) when a .ZRNT reference is detected (a special case as it is), then the corresonding FIX4,I entry is set to -1. DFIX will later check this value (where here 0 means to use a zero value), but since .ZRNT is a replace operation (\LST4 = 4) then the bogus -1 value of FIX4,I is never used.

Table 8-5. Program Types

```
 0:  SYSTEM
 1:  MEMORY RESIDENT
 2:  RT DISK RESIDENT
 3:  BG DISK RESIDENT W/ TAII ACCESS
 4:  BG DISK RESIDENT W/O TAII ACCESS
 5:  SEGMENT
 6:  LIBRARY/UTILITY
 7:  UTILITY
 8:  UTILITY LOAD ONLY TO SATISFY EXTERNAL REFERENCES
 9:  MEMORY RESIDENT USING BACKGROUND COMMON
10:  RT DISK RESIDENT USING BACKGROUND COMMON
11:  BG DISK RESIDENT USING REAL TIME COMMON W/ TAII ACCESS
12:  BG DISK RESIDENT USING REAL TIME COMMON W/O TAII ACCESS
13:  TABLE AREA II
14:  TYPE 6 BEING FORCE-LOADED INTO RESIDENT LIBRARY
15:  TABLE AREA I
16:  SLOW BOOT
17,18,19,20,25,26,27,28:  TYPES 1,2,3,4,9,10,11,12 (RESPECT.)
                          with access to SSGA
30:  SUBSYSTEM GLOBAL AREA
21-24,29,31-99:  UNUSED
                 (TYPE +80 IS USED TO DESIGNATE AUTO SCHEDULE AT
                  STARTUP, BUT MAY ONLY BE ENTERED IN THE PARAMETER
                  PHASE.  +80 IS JUST A FLAG TO THE GENERATOR AND IS
                  NOT STORED IN THE ID-SEGMENT.)

                  (TYPE +128 IS USED TO INDICATE THAT THE PROGRAM
                  CANNOT BE DUPLICATED.)
```

ENTRY POINT AVAILABILITY PER PROGRAM TYPE

Because system entry points are not available to every program type, and entry points in the memory resident library are available to memory resident programs only, certain checks must be made to prevent illegal references. (See Table 8-6.)

System entry points defined in type 0 and 16 modules can be referenced only by the system and its tables (types 0,13,15 respectively), the slow boot configurator (type 16), by SSGA (type 30) and by type 3 (11,19,27) programs.

Library entry points defined by type 6 and 14 modules can be referenced only by the memory resident library and memory resident programs (types 1,9,17,25). Outside the memory resident area all type 6 and 14 modules are treated as type 7 modules so they will be appended to all programs referencing them.

For all programs, Type 7 modules will be appended to each module referencing them.

Table 8-6. Program Type References

"REFERENCER"	SYS 0	MR 1	RTDR 2	BGDR 3	BGDR 4	SEG 5	LIB 6	UTIL 7	UTIL 8	MR 9	RTDR 10	BGDR 11	BGDR 12	TA II 13	LIB 14	TA I 15	CONFG 16	MR 17	RTDR 18	BGDR 19	BGDR 20	•	MR 25	RTDR 26	BGDR 27	BGDR 28	•	SSGA 30
	"REFERENCEE"																											
0	✓	-	-	-	-	-	✓2	✓	✓	-	-	-	-	✓	✓2	✓	✓	-	-	-	-		-	-	-	-		X
1	X	-	-	-	-	-	✓	✓	✓	-	-	-	-	✓	✓	✓	-	-	-	-	-		-	-	-	-		X
2	X	-	-	-	-	✓	✓2	✓	✓	-	-	-	-	✓	✓2	✓	-	-	-	-	-		-	-	-	-		X
3	✓	-	-	-	-	✓	✓2	✓	✓	-	-	-	-	✓	✓2	✓		-	-	-	-		-	-	-	-		X
4	X	-	-	-	-	✓	✓2	✓	✓	-	-	-	-	✓	✓2	✓	-	-	-	-	-		-	-	-	-		X
5	✓	-	✓	✓	✓	X	✓2	✓	✓	-	✓	✓	✓	✓	✓2	✓	-	-	✓	✓	✓		-	✓	✓	✓		X
6	X	-	-	-	-	-	✓	X	X	-	-	-	-	✓	✓	✓	-	-	-	-	-		-	-	-	-		X
7	X	-	-	-	-	-	✓2	✓	✓	-	-	-	-	✓	✓2	✓	-	-	-	-	-		-	-	-	-		X
8	X	-	-	-	-	-	✓	✓	✓	-	-	-	-	✓	✓2	✓	-	-	-	-	-		-	-	-	-		X
9	X	-	-	-	-	-	✓	✓	✓	-	-	-	-	✓	✓	✓	-	-	-	-	-		-	-	-	-		X
10	X	-	-	-	-	✓	✓2	✓	✓	-	-	-	-	✓	✓2	✓	-	-	-	-	-		-	-	-	-		X
11	✓	-	-	-	-	✓	✓2	✓	✓	-	-	-	-	✓	✓2	✓	✓	-	-	-	-		-	-	-	-		X
12	X	-	-	-	-	✓	✓2	✓	✓	-	-	-	-	✓	✓2	✓	-	-	-	-	-		-	-	-	-		X
13	✓	-	-	-	-	-	✓2	✓	✓	-	-	-	-	✓	✓2	✓	✓	-	-	-	-		-	-	-	-		X
14	X	-	-	-	-	-	✓	X	X	-	-	-	-	✓	✓	✓	-	-	-	-	-		-	-	-	-		X
15	✓	-	-	-	-	-	✓2	✓	✓	-	-	-	-	✓	✓2	✓	✓	-	-	-	-		-	-	-	-		X
16	✓	-	-	-	-	-	✓2	✓	✓	-	-	-	-	✓	✓2	✓	✓	-	-	-	-		-	-	-	-		X
17	X	-	-	-	-	-	✓	✓	✓	-	-	-	-	✓	✓	✓	-	-	-	-	-		-	-	-	-		✓
18	X	-	-	-	-	✓	✓2	✓	✓	-	-	-	-	✓	✓2	✓	-	-	-	-	-		-	-	-	-		✓
19	✓	-	-	-	-	✓	✓2	✓	✓	-	-	-	-	✓	✓2	✓		-	-	-	-		-	-	-	-		✓
20	X	-	-	-	-	✓	✓2	✓	✓	-	-	-	-	✓	✓2	✓	-	-	-	-	-		-	-	-	-		✓
•																												
25	X	-	-	-	-	-	✓	✓	✓	-	-	-	-	✓	✓	✓	-	-	-	-	-		-	-	-	-		✓
26	X	-	-	-	-	✓	✓2	✓	✓	-	-	-	-	✓	✓2	✓	-	-	-	-	-		-	-	-	-		✓
27	✓	-	-	-	-	✓	✓2	✓	✓	-	-	-	-	✓	✓2	✓		-	-	-	-		-	-	-	-		✓
28	X	-	-	-	-	✓	✓2	✓	✓	-	-	-	-	✓	✓2	✓	-	-	-	-	-		-	-	-	-		✓
•																												
30	✓	-	-	-	-	-	✓2	✓	✓	-	-	-	-	✓	✓2	✓	-	-	-	-	-		-	-	-	-		✓

✓ allowed references

X disallowed reference

- don't care references

Note 1 - allowed only when loaded with a type 3, 11, 19 or 27 main.

Note 2 - all reference to type 6 and 14 library routines treated as type 7 utility references when outside the memory resident area

System Table Area entry points defined by type 13 and 15 modules can be referenced by anyone. If Table Area II was not included in the user map, cross-map loads must be used to reference it (note that it's write-protected).

SSGA modules (type 30) can be referenced only by modules specifically declaring SSGA access, types 17-20 and 25-28 and another type 30 module.

Checks must be done at load time rather than during relocatable input because of the parameter change capability.

RELOCATION BY TYPES

type 15's in Table Area I

type 0 drivers in partition #1

type 30's in SSGA

type 0 system drivers

type 13's in Table Area II

type 0 SYSTEM

type 16 configurator

type 0 drivers in partitions #2 onward

type 6 & 14 memory resident library

memory resident's: type 1,(9,17,25)

Real Time DR's: type 2,(10,18,26) plus any type 5 segments

Background DR's: type 3,(11,19,27) plus any type 5 segments

Background DR's: type 4, (12,20,28) plus any type 5 segments

LIBRARIES

MEMORY RESIDENT LIBRARY (MRL)

The memory resident library contains all type-14 force-loaded modules, and all type 6 modules referenced by type 14 modules or memory resident programs. In order to pick up the type 6 modules, a pseudo-load of all memory resident programs is done.

Because library routines can be referenced only when in the memory resident map, they must be made available to disc resident programs. Therefore, after memory resident loading, all type-6 and 14 modules are demoted by /DEML to type 7 (utility) modules, so they will be appended to all DR programs referencing them. Type 30 SSGA modules are not demoted. If referenced before the MRL is relocated, they are treated as type 7's.

RELOCATABLE DISC RESIDENT LIBRARY

The relocatable library contains all type 7 modules. Note that these modules include all demoted type 6 and 14 modules, some of which may have been included in the memory resident library.

LIBRARY ENTRY POINTS LIST

The entry points available to the user are sent to the disc in three passes. See Appendix H for the physical disc layout and the entry formats.

PASS 1: all entry points defined by type 0 & 16 (system) modules are sent.

PASS 2: all entry points defined by type-30 modules and by type 15 and 13 modules (Table Areas I and II respectively) are sent, in addition to LST types 2 (common), 3 (absolute), 4 (replace).

PASS 3: All entry points defined by type-7 modules (includes all type 6 and 14's).

The output of pass 1 starts on a sector boundary = DSCLB, and contains SYSLN entries. The output of passes 2 and 3 contains DSCLN entries.

UNDEFINED ENTRY POINTS DURING GENERATION

To recover from an undefined entry point, the user must enter a "DISPLAY UNDEFS,TR" before exiting from the Program Input Phase. These undefined externals will be listed on both the operator console (regardless of echo) and the list file. If undefineds existed, a TR will automatically be done to the console for optional recovery.

The LST type for an undefined external will be 5 until that point where it becomes defined. (Note that a CHANGE ENT will do it even after exiting from the PIP). Once exiting from the PIP, it will be treated like a type 4 during program relocation; the value will be zero (a NOP). No error diagnostics will be printed when an undefined is referenced during generation loading.

SIZE RESTRICTIONS

The following limits for an RTE-IVB System must be enforced due to the 32K (15 bit) logical address space of HP 1000 computers, base page ignored. Extended memory areas are not included. P (Area X) is defined as the smallest number of pages which completely contains area X.

SYSTEM:

p(Table Area I) + p(Driver Partition) + p(Common) +
p(System Driver Area + Table Area II + System + Configurator)=< 31 pages

MEMORY RESIDENTS:

p(TAI) + p(DP) + p(COM) + p(SDA+ TAII) +
p(Memory Resident Library + Memory Resident Programs)=< 31 pages

where p(COM) and p(SDA + TAII) are optional

RT AND BG DISK RESIDENTS:

p(TAII) + p(DP) + p(COM) + p(SDA + TAII) +

p(a RT or BG Disk Resident program)=< 31 pages

LARGE BG DISK RESIDENTS:

p(TAI) + p(DP) + p(COM) + p(a large BG Disk Resident Program)=< 31 pages

where p(COM) is optional

PAGE ALIGNMENTS

The following areas are automatically aligned by the generator to start on a page boundary:

Base Page
Table Area I
Driver Partition
Common
System Driver Area
Resident Library
Memory Resident Programs (first one only)
Disk Resident programs

MISC AREAS

BASE PAGE

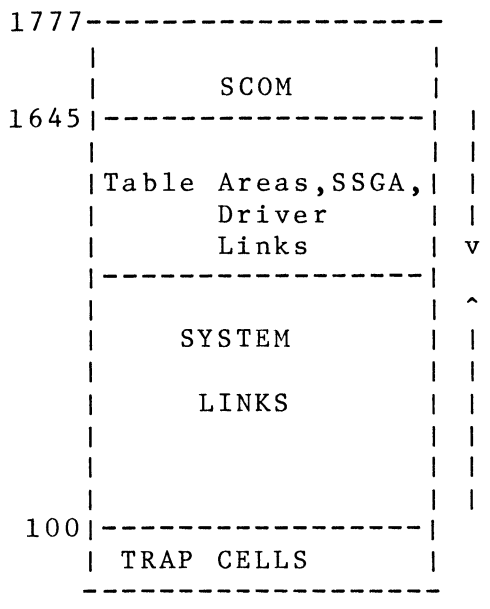
Table 8-7 describes the various base page formats. Only one (each) system and memory resident base page exists, but each disk resident program has its own copy of base page. The base page links used by a disk resident program are stored in the next disk sector following the program's code. The system base page is both logically and physically page 0 and is stored starting at track 0 sector 2 of the system. The memory resident base page (MRBP) resides in physical memory after the last driver partition page, and the memory resident library (MRL) starts on the physcial page after that. Physically the MRBP links are stored on the next disk sector following the last memory resident program's code.

The System Communication Area (SCOM) and all Table Area I, SSGA, Table Area II and drive links are resident in both system and user maps, the SCOM residing in BP locations 1777-1645 and the upper BP links from 1644 downward. After the track 0 sector 0 boot extension has been sent to the disk, the dummy base page (it resides in core overlaying the initialization code of the generator MAIN) is written for the sole purpose of reserving its disk space. The system links (including configurator) always start at location 100 and grow upward toward the SCOM. The partition drivers links are not allocated until the PRD's have been relocated, so checks are done for overflow of these driver links into the system links. The system base page on disk is updated at the end of the system's relocation for the trap cells and system links. Note that trap cells referencing programs are fixed-up as the programs are relocated. The BP driver links are updated on disc after all the PRD's have been relocated, and the SCOM is updated during the final generation cleanup.

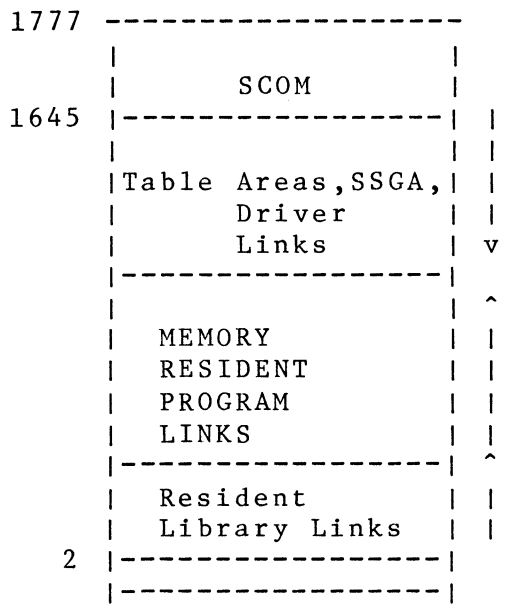
Memory resident and disk resident program links start at BP location 2 and grow upward. A GEN ERR 16 is issued on each overflow into the upper link area. In MRBP the memory resident library links are allocated first, followed by those links necessary for all the memory resident programs.

Table 8-7. Base Page Formats

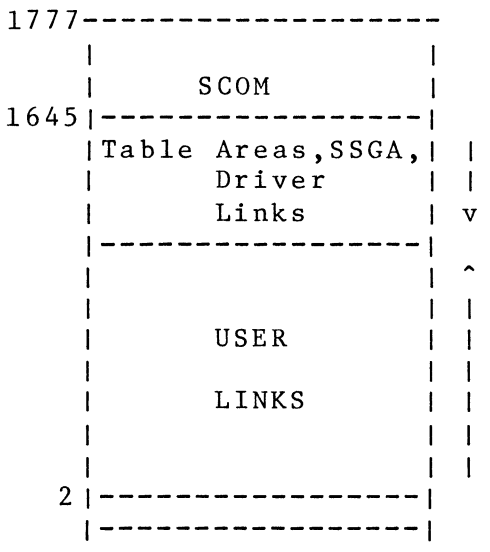
System BP



Memory Res. BP



Disk Res. BP



SYSTEM COMMUNICATION AREA (SCOM)

The SCOM is built at the end of generation during final clean-up. 133 octal words below the label USRTR are initialized to 0 and overlaid as SCOM is built, transferred to the dummy base page, and then sent to the disk using /ABDO. The base page locations are set by the generator variables as follows:

1645	XIDEX	ID extension address of currently executing program	<-0
1646	XMATA	MAT entry address of currently executing program	<-0
1647	XI	address of index register save area	<-0
1650	EQTA	first word address of equipment table	<-AEQT
1651	EQT#	number of EQT entries	<-CEQT
1652	DRT	first word address of device reference table	<-ASQT
1653	LUMAX	number of logical units in DRT	<-CSQT
1654	INTBA	first word address of interrupt table	<-AINT
1655	INTLG	number of interrupt table entires	<-CINT
1656	TAT	first word address of track allocation table	<-ADICT
1657	KEYWD	first word address of keyword block	<-KEYAD
1660	EQT1	address of SAM#1	<-LWSYS+1
1661	EQT2	# words	<-SAM#1
1662	EQT3	address of SAM#2	<-LWSYS+1+SAM#1
1663	EQT4	# words	<-SAM#2
1664	EQT5	address of SAM#0	<-LWTAI+1
1665	EQT6	# words	<-DPADD-(LWTAI+1)
1666	EQT7		<-0
1667	EQT8		<-0
1670	EQT9		<-0
1671	EQT10		<-0
1672	EQT11		<-0
1673	CHAN	current DMA channel number	<-0

```
1674 TBG    I/O address of time base card                    <-TBCHN
1675 SYSTY  EQT entry address of system TTY                  <-SYSTY
1676 RQCNT  number of request parameters, less 1               <-0
1677 RQRTN  return point address                              <-0
1700 RQP1             Addresses                               <-0
1701 RQP2             of request                              <-0
1702 RQP3             parameters                              <-0
1703 RQP4             (set                                    <-0
1704 RQP5             for a                                   <-0
1705 RQP6             maximum                                 <-0
1706 RQP7             of                                      <-0
1707 RQP8             nine                                    <-0
1710 RQP9             parameters)                             <-0
1711 SKEDD  address of system 'schedule' list                <-SCH4
1712                                                          <-0
1713 SUSP2  address of 'wait suspend' list                    <-0
1714 SUSP3  address of 'available memory' list                <-0
1715 SUSP4  address of 'disc allocation' list                 <-0
1716 SUSP5  address of 'operator suspend' list                <-0
1717 XEQT   ID segment addr. of current program              <-0
1720 XLINK  ID segment linkage                               <-0
1721 XTEMP  ID segment temporary                             <-0
1722 XTEMP  ID segment temporary                             <-0
1723 XTEMP  ID segment temporary                             <-0
1724 XTEMP  ID segment temporary                             <-0
1725 XTEMP  ID segment temporary                             <-0
1726 XPRIO  ID segment priority word                         <-0
```

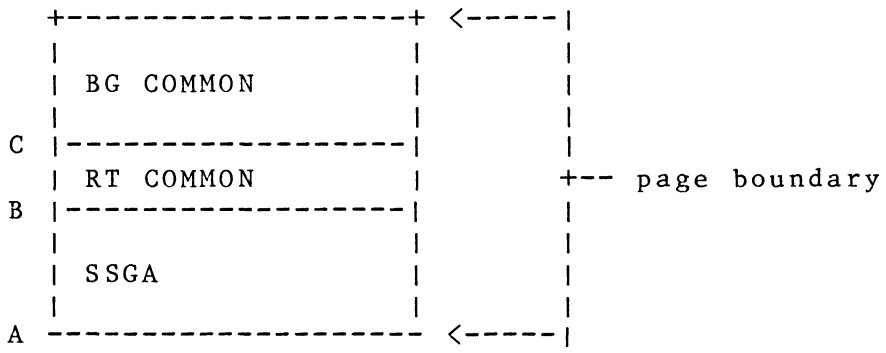
1727	XPENT	ID segment primary entry point	<-0
1730	XSUSP	ID segment point of suspension	<-0
1731	XA	ID segment A-Register at suspension	<-0
1732	XB	ID segment B-Register at suspension	<-0
1733	XEO	ID segment E and overflow reg. at suspension	<-0
1734	OPATN	operator/keyboard attention flag	<-0
1735	OPFLG	operator communication flag	<-0
1736	SWAP	RT disc resident swapping flag	<-SWAPF
1737	DUMMY	I/O address of dummy interface card (PI)	<-PIOC
1740	IDSDA	disc address of first ID segment	<-DSKSY
1741	IDSDP	position within sector of first ID segment	<-IDSP
1742	BPA1	FWA user base page link area	<-2
1743	BPA2	LWA user base page link area	<-LOLNK-1
1744	BPA3	FWA user base page link	<-2
1745	LBORG	FWA of resident library area	<-LBCAD
1746	RTORG	FWA of real-time common	<-RTCAD
1747	RTCOM	length of real-time common	<-COMRT
1750	RTDRA	FWA of real-time partition	<-MEM6
1751	AVMEM	LWA+1 of real time partition	<-SYMAD
1752	BGORG	FWA of background common	<-BGCAD
1753	BGCOM	length of background common	<-COMBG
1754	BGDRA	FWA of background partition	<-MEM12
1755	TATLG	negative length of track assignment table	<- -(DSIZE+DAUXN)
1756	TATSD	number of system disc tracks	<-DSIZE
1757	SECT2	number of sectors per track on lu 2 (system)	<-SDS#
1760	SECT3	number of sectors per track on lu 3 (aux.)	<-ADS#
1761	DSCLB	disc address of entry point library	<-DSKLB

1762	DSCLN # of user available entry points in library	<-LBCNT
1763	DSCUT disc address of reloc. disk resident library	<-DSKUT
1764	SYSLN # of system entry points in library	<-SYCNT
1765	LGOTK load and go: lu, starting track, # of tracks	<-0
1766	LGOC current load and go track/sector address	<-0
1767	SFCUN log source: lu, disc address	<-0
1770	MPTFL memory protect on/off flag (0/1)	<-1
1771	EQT12	<-0
1772	EQT13	<-0
1773	EQT14	<-0
1774	EQT15	<-0
1775	FENCE memory potect fence address	<-0
1776		<-0
1777	BGLWA last word memory address of BG partition	<-LWASM

COMMON

The RT and BG commons along with the Subsystem Global Area (SSGA, type 30 module) occupy a single area collectively known as "COMMON". Since any program using any of the three areas can "see" (has map entries for) the others, only the memory protect fence table can provide any protection (see its format under MPFT writeup).

The order of the three areas was chosen such that a hierarchical protection is preserved:



The memory protect fence will be placed at A, B, or C if a program is using COMMON.

When the IDENT's are scanned for ID segment allocation at the end of the PIP, the common sizes of each program stored in \ID4 bits (14-0) is used to set the maximum RT and BG common sizes, COMRT and COMBG respectively.

Starting on a page boundary after the driver partition all SSGA modules are loaded first, followed by the allocation of the RT and BG common area.

The RT common size is displayed, the user is given the option of increasing it, and the starting address is displayed:

```
RT COMMON  XXXXX <------- decimal words
CHANGE RT COMMON?
NNNNN      0 means no change; GEN ERR 14 issued on an invalid response
RT COM ADD XXXXX <------- octal address
```

RTCAD is set to the RT common starting address from PPREL, and COMRT, the number of words, maybe updated; BGBND is set to the starting address of BG Common, PPREL+COMRT. Before COMBG is displayed, it is updated to include that area from BGBND to the end of the page (because the SDA is automatically aligned on the next page boundary after BG common). The following sequence occurs for BG common determination:

```
BG COMMON   XXXXX<--------decimal words (new size)
CHANGE BG COMMON?
NN         0 means no change; GEN ERR 14 issued on an invalid response
BG COM ADD  XXXXX
BG COMMON   XXXXX<----decimal words
```

BG Common size is increased in page multiples so COMBG has NN*1024 added to it, and is redisplayed.

CONFIGURATOR PROGRAM

The configurator program is a special system module of type 16. It has access to all the system entry points and is loaded immediately after the system in what will later be the beginning of System Available Memory (SAM #1). Its base page links are included with those of the system. The last word must not be greater than 77577B (77377B for ICD Systems) or a GEN ERR 18 will result and the generation will be aborted. The memory above this address must be reserved for the boot extension, which loads the system. The last word of both the system code and configurator code must be saved (in LWSYS and LWSLB respectively) in order to compute the size of SAM #1 at the beginning of the Partition Definition Phase. SAM #1 will include that specific memory area covered by the configurator plus any remaining area left on the last page occupied by the configurator.

The configurator references the Table Area II entry point $SBTB which is the first word of the following 6 word table:

```
$SBTB  disc address of driver partitions
       # of pages for all driver partitions
       disc address of memory resident base page
       # of pages for memory resident base page
       disc address of memory resident library and programs
       # of pages for all memory resident library and programs
```

The values are stuffed into $SBTB when the generator sets the values of all the Table Area II entry points specified in $$TB2 at the end of the partition definition phase (see the Memory Protect Fence Table later in this section).

BOOTSTRAP AND EXTENSION

The generator builds both the track 0 sector 0 boot extension and moving head bootstrap loaders for either the 7900, MAC or ICD discs. Generator segment 1 builds the 7900 bootstrap loader whereas segment 7 builds it for the ICD and MAC discs. The generator stores the system subchannel disk specifications in the bootstrap loader (i.e., first track, # of tracks, starting head, # surfaces, etc.), and for the moving head bootstrap loader, configures the disk I/O instructions to the select code of the system disc. The high address of the configurator is stored in the track 0 sector 0 boot extension in HIGH so the first chunk of memory can be read in from the disc starting at track 0 sector 2 (track 0 sector 4 for ICD discs). The generator also sets the following values in the boot extension:

	BOOT EXTENSION VALUES	BOOT STRAP VALUES
7900 Disc:		
	TBASE	UN#IT
	U#NIT	H#AD
	B#MSK	S#EKC
	SKCMD	R#DCM
	R#CMD	DSKDR
	HIGH	T#ACO
7905/06/20/25 Disc:		
	TBASE	PT#TR
	BHD#	PT#T2
	#HDS	H#AD
	WAK	PT#H2
	SKCMD	WA#KE
	AD#RC	PT#SK
	R#CMD	PT#AD
	S#TAC	R#DCM
	HIGH	P#EN

```
     ------------------------------------------------
7906H/20H/25H Disc.
          HTBAS                                !CYLL
          BHED#                                !CYLH
          #HDS                                 !HEAD
          AD1                                  #HEDS
          AD2                                  !UNIT
          AD3                                  !AD1
          AD4                                  !AD2
          AD5                                  !AD3
          HHIGH                                !AD4
                                               !AD5
```

TABLE AREAS I AND II

The generator-built track map table ($TB31, $TB32 or $TA32), EQT's and extensions, DVMAP table, DRT, INT and all type 15 modules will be loaded and/or stored in Table Area I (in that order). Table Area I exists in all maps. The user-available entry points to system code will be loaded into Table Area I from the type 15 module $$TB1. Note that all user-defined track map tables will have to be type 15 modules in order to exist in all maps. The space left on the last page occupied by Table Area I is allocated to SAM (SAM #0).

Table Area II contains (in the following order):

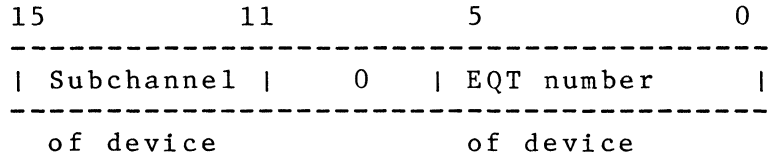
$CLAS table
$LUSW table
$RNTB table
$LUAV table
$IDEX table
ID extensions
keyword table
ID segments
$MATA table
$MRMP map
$MPFT table
Track Allocation Table
$$TB2 entry points
type 13 modules

Type 13 module $$TB2 contains the entry points to system tables most of whose values are set when the $MATA, $MRMP, and $MPFT tables are built during the Partition Definition Phase. Since Table Area II is included only in the system, memory resident (optional), and type 2 RT and type 3 BG program address spaces, those type 4 BG programs wanting to access any Table Area II entry points must do so via cross-map loads.

All external references from the Table Areas are resolved through fixups once the system and all drivers are relocated. The Table Areas can reference each other, the system, and types 6, 7 and 8 utility modules. Their links are included with those of the system. Table Area I starts on a page boundary, following the base page. Table Area II immediately follows the System Driver Area in memory, so both are mapped in when either is referenced.

EQUIPMENT TABLE (EQT), DEVICE REFERENCE TABLE (DRT), and INTERRUPT TABLE (INT) SIZES

The maximum number of EQT and DRT entries is 63 and 254, respectively. Since both the EQT and DRT entries are sequentially prompted, the generator will issue a GEN ERR 35 for all entries past the 63rd or 254th until a /E is encountered. The size of the DRT is always twice the number of LU's defined (CSQT), with the second zero-filled chunk of size CSQT following the first. The first CSQT words of the DRT are set as follows by the generator:



The INT contains entries for each channel from 6B through 77B, even though the user may not have defined up to the maximum. The entire channel spectrum must be present for possible I/O channel reconfiguration at slow boot time. This also implies that base page location 100B will always be the first SYSTEM base page link. All I/O locations from 2B through 77B are initialized to the absolute code for "JSB $CIC,I", except that location 4 is initialized to "HLT 04".

The INT records are processed as follows:

1. N1,EQT,N2 - The address of the EQT entry specified by N2 is set into the INT entry designated by N1. The INT location contains "JSB $CIC,I".

2. N1,PRG,PNAME - The 2's complement of the ID Segment ADDR for contains "JSB $CIC,I".

3. N1,ENT,ENTRY - The INT entry specified by N1 is set = 0 and the interrupt location N1 is set to contain "JSB X,I", where X is the BP link address containing the address of ENTRY.

4. N1,ABS,XXXXXX - The INT entry specified by N1 is set = 0 and the interrupt location N1 is set to contain "XXXXXX".

All locations in the Interrupt Table from 6B to 77B which are not specified by INT records are set = 0. For N1 = 4 the only legal entries are types 3 and 4. All INT records must be entered in increasing N order, with the exception of 4.

For ENT type entries, the entry point referenced must be contained in a type 0 module. If that type 0 module is a driver (IDENT word 8 bit 15 is set) then that driver must be in the System Driver Area (IDENT word 8 bit 14 is set).

DRIVERS and DVMAP

Drivers will be relocated to reside in either a driver partition or the System Driver Area (SDA). The I/O tables (EQT's, DVMAP, DRT and INT) are stored in Table Area I, and are therefore built before any drivers have been relocated. Fixups are then resolved for EQT words 3 and 4 once a driver's initiation and completion sections are relocated. The two FIXUP table entries will automatically be allocated when the EQT is built. The fixup entries are built as follows:

```
word 1:  memory location (in EQT) where address of I.XX or C.XX
         to be stored
word 2:  instr. code = 0, DBL record type=5
word 3:  offset = 0
word 4:  LST index of I.XX or C.XX
```

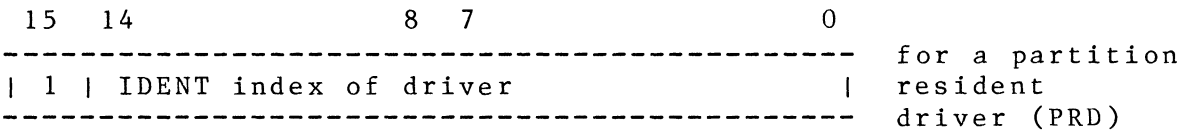
Setting the DBL record type in word 2 equal to 5 simulates an external reference with offset. With the instruction code equal to 0 this indicates a DEF to an external with offset (of 0) at fixup time, therefore making it direct.

All drivers (identified as type 0 modules beginning with "DV") will be sent to driver partitions unless so specified by an S or M in their EQT definitions as an SDA type. Those drivers without an EQT and possibly not beginning with "DV" will be relocated with the system. If an SDA driver is to do its own mapping, then an M in addition to or in place of the S may be specified.

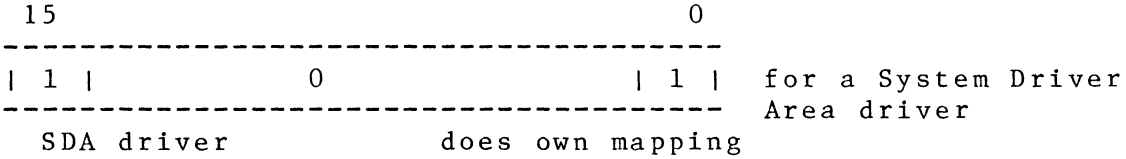
When an EQT is defined, the IDENT table entry for the named driver is retrieved (a GEN ERR 25 is issued if not found). After the EQT is built the driver's IDENT word 8, bit 15 is set to indicate that a valid EQT existed, bit 14 is set if SDA was declared, and bit 13 is set if the SDA driver is to do its own mapping. If an M is specified without an S, then an S is assumed and both bits 14 and 13 are set. If bit 15 indicates that a driver had already been specified in a previous EQT, then the new type must match that of the old. i.e., bits 14 and 13 of the current entry must match the values to be set by the new entry, otherwise a GEN ERR 23 would be issued and the EQT would have to be redefined.

The system disc driver cannot reside in SDA. When an EQT's select code matches the "CONTROLLER SELECT CODE?" response, the system disc EQT is assumed and a check is made to make sure that SDA was not declared for this driver - or a GEN ERR 23 occurs.

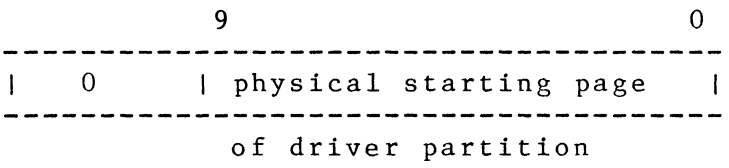
The first half of the driver map table DVMAP is dynamically built in a buffer as the EQT's are defined. DVMAP consists of two consecutive chunks of size CEQT (the number of EQT's). After all the EQT's and EQT extensions have been built, space is reserved for the DVMAP and it is sent to the disc. Table Area II entry point $DVMP is set (later) to its address. The first CEQT chunk has values stored in it by the generator, while the second CEQT chunk is zero-filled for user by RTIOC. A 64-word buffer (maximum # of EQT's) is used for building the first part of DVMAP. The dummy entries are built as follows, with word 0 corresponding to EQT1,...word CEQT-1 corresponding to EQT CEQT:



or:



The PRD entries in DVMAP are updated on disc when those drivers are relocated; the SDA entries are left as defined. The final PRD form is:



When a PRD is relocated into a partition, all the EQT entries in DVMAP must be scanned for an IDENT index matching that of the driver. All matching DVMAP entries are then replaced with the driver partition starting page.

SYSTEM DRIVER AREA (SDA)

All drivers going into the System Driver Area are relocated following the construction of Common. Since Common always ends on a page boundary, the SDA always begins on one.

DRIVER PARTITIONS

The defaulted driver partition size is two pages - large enough to hold any HP partition-resident driver. As many drivers are relocated into a DP as will fit, so increasing the DP size will allow more drivers to fit into a particular partition - possibly saving physical pages if a lot of leftover page space can be used. For partition-resident drivers greater than 2 pages, the DP size must be overridden in order to accommodate it. Otherwise, if the driver overflows a DP at relocation time the generation will be aborted with a GEN ERR 59.

The current DP size is displayed in decimal number of words, and the user is given the option of increasing it:

```
DRIVR PART 00002 PAGES
CHANGE DRIVR PART?
```

An zero (0) response implies no change, otherwise the new size must be >= DPLN and less than 17 (a blue-sky estimate). If an invalid response is entered then a GEN ERR 01 is issued and the query re-prompted. The new value of DPLN is used to set the Table Area II entry point $DLTH. The last word occupied by Table Area I is founded up to the next page boundary and stored in DPADD (the skipped memory is later allocated to SAM). DPADD converted to a logical page number is used to set Table Area II entry point $DVPT (starting logical page of driver partition). Memory skipped in the page alignment is "sent" to the disc by updating the relocation address, PPREL. When \ABDO is called the next time, that disc space will be zero-filled since PPREL will be greater than the address of the highest previously generated word in the system map (MXABC,I of the \ABDO specification table for the system).

After the relocation of Table Area I, the first driver partition is relocated. The system disc driver must be relocated into this partition for use by the configurator program; this driver is determined by using DRT2 (the system disc EQT #) to offset into the temporary DVMAP table in order to pick up its IDENT index.

Once a driver is relocated, a check is made to see if the logical address space used for a driver partition has been overflowed. If not, the IDENT table is scanned for a driver that will fit into the remaining space of the DP. The scan always begins at the beginning of the IDENT table stopping when a driver's size specified in \ID8 of its entry indicates that it will fit. In addition, the routine CPL? is called to check for the memory requirements when current page links are in effect. If the above two checks pass, the driver is relocated, otherwise the scan through the IDENT table is continued.

Note however, that a driver may still overflow a partition. This may happen when referenced subroutines are appended to the driver during relocation. Upon overflow, the violating driver is 'backed-up' over, an error (actually a warning) is issued, and the IDENT table scan is continued, searching for another driver that will fit. The DVMAP entries are not updated for the overflowed driver. When Fixups to driver entry points are resolved during relocation, the entries are not deleted from the FIXUP table. Thus in the case where a driver is relocated more than once, the references are simply re-fixed up to the final value. The violating driver will be relocated into a subsequent partition.

When no more drivers can fit into a partition, the remainder is zero-filled. For driver partition #1 zero-fill is done to the last word of a DP, but for the other DP's zero-fill is done only to the last word of the last page used. With this feature pages may be saved where one or more complete pages of a DP are unused.

For each new DP, the scan is then started at the beginning of the IDENT table for the next unrelocated partition-resident driver. If none exist, the driver partitions are done. The fixup table is cleared before the memory resident library is built.

For driver partitions #2 onward the \ABDO specification map is changed from that of the system to that of driver partitions. This is done because these driver partitions reside logically in the system area, but physically on the disc and physically in pages above the system area. From then on, when each new DP is started, the DP map's disc address ABDSK,I is updated but ABCOR,I and MXABC,I are reset to DPADD.

After a driver is loaded, the physical starting page of that driver partition is stored in all the DVMAP entries referencing that driver. The fixup entries pertaining to EQT words 3 and 4 are also resolved. Note that the \ABDO map must be changed to that of the system in order to perform these Table Area I updates. When a new driver partition is started it's starting physical page is set: PAGE#<---PAGE# + number of pages required by previous driver partition (DPLN). PAGE# is initially set, for driver partition #2, to $ENDS which is the physical page immediately following SAM#1. (See the physical memory map in Appendix I.) The physical page for DP#1 is the same as its logical page, $DVPT.

Partition driver links start at BP location 1644 and grow downward. Since the system usually has already been relocated, checks must be make during PRD relocation for overflow of links into those occupied by the system, a GEN ERR 16 results and the generation is aborted.

Note, a user-entered disc Track Map Table (e.g. $TB31 or $TB32) must be defined as a type 15 module only if it is to be compatible with the RTE-IVA Disc Backup Utilities. Otherwise, they may be typed as subroutines, and appended to the driver in its driver partition.

ID SEGMENTS AND EXTENSIONS

During the construction of Table Area II, space is reserved for long ID segments (33 words long), memory resident ID segments (33), short ID segments (9), and ID extensions (3). Long ID segments are allocated to real-time and background disk resident programs; memory resident ID segments to memory resident programs; short ID segments for each program segment; and ID extensions for each long ID segment of an EMA program. The minimum number of each type necessary is obtained by scanning the IDENTS keying off the program type and EMA flag in \ID6. The user is given the opportunity to have blank ID segments and extensions allocated through the queries:

```
# of BLANK LONG ID SEGMENTS?
# of BLANK SHORT ID SEGMENTS?
# of BLANK ID EXTENSIONS?
```

A GEN ERR 60 is issued if the total number of long and memory resident ID segments is >254. If more than 254 long ID segments are required before any blanks are requested, then the generator aborts after giving the GEN ERR 60. A GEN ERR 01 is given if the number of ID EXTENSIONS exceeds the number of long ID segments.

The keyword table and ID extension table ($IDEX) have one word allocated for each ID segment and ID extension, respectively - plus one "stop" word equal to zero. The keyword table entries are set to the ID segment addresses as the ID segments are being built, or during final cleanup for the blank ID segments generated. The ID extension table and the ID extensions precede the keyword table and ID segments. When built, the ID extension table ($IDEX) is initialized to the addresses of the ID extension entries (3 words each). See the EMA section for the description of the values stored in the ID extension entry.

ID segment entries are built as follows:

Table 8-8. ID Segment Words Set During Generation

0	0	
1-5	0	
6	PRIORITY from NAM record	
7*	PRIMARY ENTRY POINT	
8	0	
9	0	
10	address of ID segment word 1	
11	0	
12*	ID1 gives NAME1 & 2	
13*	ID2 gives NAME3 & 4	
14*	ID3 (15-8) gives NAME5; ID6 (2-0) gives TYPE	ID
15	optionally set bit 0 if the scheduled program	SEGMENT
16	0	
17	resolution code and execution multiple from NAM record	
18	Time word from NAM record	
19	Time word from NAM record	
20	0	plus
21	** (see below)	words
22*	low main address from PPREL	30,31,32
23*	high main address from TPREL	
24*	low BP address from PBREL	
25*	high BP address from TBREL	
26	main disk address from DSKMN	
27*	0	
28	ID EXT# & EMA SIZE - see EMA section	
29	high main of largest segment = TPMAX, else 0	
30	0 (session monitor word 1)	
31	0 (session monitor word 2)	
32	0 (session monitor word 3)	

* short ID segments

** RP bit 15 may be set during the Partition Definition Phase. # pages required (14-10) set at end of main program load by IDFIX; for EMA programs includes MSEG size; may be changed for non-EMA programs during the Partition Definition Phase. MPFI (9-7) set at end of main program load by IDFIX. Partition # (5-0) may be set if assigned during PDP.

EXTENDED MEMORY AREAS (EMA)

When an EMA declaration (relocatable record type 6) is encountered in a module during the Program Input Phase, the EMA bit (15) is set in word 6 of the current module's IDENT entry. The EMA size is retrieved from word 2 of the relocatable record (bits 9-0) and is stored in IDENT word 5 (bits 13-4). The MSEG size is retrieved from word 7 of the record (bits 4-0) and is stored in IDENT word 6 (bits 14-10). A zero value for either of these two sizes indicates that the default values are to be determined. The default MSEG size is determined at load time by the generator, and the default EMA size at system dispatch time. If more than one EMA declaration occurs in a module (\ID6 bit 15 has already been set) then a GEN ERR 41 results and the program's IDENT entry and all its LST entries are deleted (table pointers are backed up).

The EMA program type is picked up from IDENT word 6 (bits 6-0) to determine if it is a legal EMA type. EMA programs must be disc resident, either real-time or background - types (2,10,18,26), (3,11,19,27) & (4,12,20,28) respectively. The EMA declaration must be in the main program so any declarations in subroutines or segments are not allowed. If the type check fails, the program's type in IDENT word 6 is set to 8, and a warning is issued to the user (GEN ERR 40). A main program of type 8 will not be relocated; recovery is possible by changing the program's type to a valid EMA type during the Parameter Phase. The EMA type check is also performed in the Parameter Phase when the type of an EMA program is changed (bit 15 of IDENT word 6 indicates EMA). If the new type is invalid, the warning GEN ERR 40 is issued, and the program type is not changed. ID segments with extensions will be allocated only for those EMA programs of the correct type (plus any user indicated spares).

The EMA label is stored in the loader symbol table (LST) word 4 as a type 6 entry. The IDENT index of the defining EMA module is stored in word 5. The symbol type is used to prevent incorrect references to EMA symbols - i.e., modules of type 0,1,(9,17,25), 6,13,14,15,16,17 and 30 referencing an LST type 6 symbol will cause a GEN ERR 42 when the EXT is encountered at load time. A NOP replaces the referencing instructions. The same thing happens when a non-EMA, but valid program type references an EMA symbol. By checking the IDENT index stored in word 5, a GEN ERR 42 will also result if an EMA program references the EMA symbol belonging to another EMA program. The referencing instructions will also be NOP'ed. Note that the EMA label declarations are also subject to the duplicate entry points restriction, with the second definition overriding the first. Being a special type of symbol, the value of an EMA label cannot be changed during the CHANGE ENT's Phase.

During the relocation of an EMA program, its EMA label is always treated as a forward reference to an external, even in the module declaring the EMA. The declaration of the EMA label forces an immediate base page link to be allocated. All references to the EMA label will then use this link. The link is allocated before the first reference is encountered to avoid the situation where an EMA program segment contained the first reference, in which case a base page link and fixup entry would be allocated in the segment's BP area; but the segment would be long gone before the fixup could be resolved. Because the logical MSEG address is the relocated address of the EMA label, that address cannot be determined until the main, its segments, and all appended subroutines have been relocated.

The highest relocation address for a program (stored in TPREL or TPMAX) is rounded up to the next page address to produce the "effective high main" (EHM). EHM becomes the logical MSEG address which resolves the EMA references. Converting EHM to a logical page address (EHMP) and subtracting it from 31 gives the maximum MSEG size. If the maximum MSEG size is <=0, a GEN ERR 43 results and the program "disappears" (i.e., no ID segment is built for it and its disc storage space is reused). If an MSEG size was declared in the program (\ID6 (14-10) is non-zero) then it is checked against the maximum MSEG size. If greater, a GEN ERR 43 also results and the program again disappears. If defaulted, the MSEG size becomes the maximum MSEG size.

For an EMA main program's ID segment, the # of pages required for execution stored in word 21 bits (14-10) is the # pages occupied by the entire program plus the MSEG size. The # of pages for an EMA program cannot be overriden during the Partition Definition Phase. Word 28 is set up as follows during the relocation of an EMA program: bits 15-10 contain the index of the next available ID extension entry starting at zero; bits 9-0 contain the declared EMA size retrieved from ID5 (13-4) or is set to 1 for a default EMA declaration. After getting the address of the ID extension entry by indexing into $IDEX table, the entry is set up as follows:

word 0: bits 4-0 are set to the MSEG size (declared or defaulted)
word 1: bits 15-11 are set to the starting logical page of the MSEG (gotten from EHMP)
bit 10 is set if the EMA size was defaulted
word 2: 0

The ID extension index is then bumped to that of the next (free) entry.

PARTITION DEFINITION PHASE

PROGRAM PAGE REQUIREMENTS AND MAXIMUM SIZES

The Partition Definition Phase begins with the display of the page requirements of all real-time and background disk resident programs. Type 4 BG programs will have a "*" appended to their display line, and EMA programs will have an "E" appended to their display line. IDFIX stored a program's page requirements in word 8 of the program's IDENT entry (bits 15-8) at the same time it built word 21 of the ID segment - so a scan of the IDENT table is done based on the program type in \ID6 (bits 2-0). One page is added in the displayed value, however, to include base page. For EMA programs, this value is calculated as follows:

The MSEG size in \ID6 bits (14-10) is subtracted from the program size in \ID8 bits (15-8) - and then the EMA size in \ID5 (13-4) is added. If the EMA was defaulted a 1 is added. In both cases, one page is added in the displayed value to include the base page.

The maximum program address space is displayed in three categories:

W/O COM XX PAGES
W/ COM XX PAGES
W/ TA2 XX PAGES

The without-common size MAXPG would be the # of logical pages left after the driver partition. The with-common size would be the # pages left after the entire common chunk. And the W/ TA2 would be the # pages left after Table Area II. MAXPG is used during partition definition to determine if a partition qualifies as a mother partition (# of defined pages is >MAXPG).

SYSTEM AVAILABLE MEMORY (SAM)

System available memory can exist in three chunks - the size of the first two chunks determined by the generator, the size of the third chunk specified by the user (and may be zero). The first chunk (which will be referred to as SAM#0) lies in the area between the end of Table Area I and the start of the driver partition. The second chunk (SAM#1) covers the total area occupied by the configurator plus any remaining area left on the last logical page the configurator occupies. The size calculation is performed as follows:

word size of SAM#1 = (last word of configurator rounded up to next page)
- (last word occupied by SYSTEM code)

$MPSA bits (15-10) are set to the # of pages occupied by SAM#1 (including the page shared with the system, if that's the case), and bits (9-0) are set to the starting page of SAM#1. SCOM word EQT1 is set to the logical starting address of SAM#1 which is actually the last word occupied by the SYSTEM, plus 1. SCOM word EQT2 is set to the # of words in SAM#1. The Table Area II entry point $ENDS is equivalent to $MPSA(9-0) + $MPSA (15-10) which amounts to the physical page # immediately following SAM#1. SCOM word EQT5 is set to the starting address of SAM#0, and EQT6 is set to its size.

The logical combination of SAM#1 and SAM#2 must appear in the first 32K of logical address space, where SAM#2 is "relocated" so as to appear logically contiguous with SAM#1. Physically, the pages containing the driver partitions and memory resident area (base page, library and programs) will separate the two chunks. If $MPSA (9-0) + $MPSA (15-10) is equal to 32 then SAM#1 occupies the rest of the logical address space and SAM#2 will not exist (it's descriptors $MPS2 and EQT4 will all be zero). Since the last word of SAM cannot be 77777 (see $ALC routine in SYSTEM for explanation), the word count in EQT2 must be two less in order to force 77775 as the last word.

The present size of SAM (i.e., of SAM#0 + SAM#1) is displayed in decimal number of words to the user. Then the user is given a range for the first physical page for user partitions and is asked to enter a page number in that range. If a value greater than the default value is entered, the user is allocating the skipped pages to SAM (thereby defining SAM#2). If the new first partition/page "equals the default value" then SAM#2 doesn't exist and it's descriptors are set to zero. The size of SAM#2 is calculated as: (new first page - old first page)* 1024. $MPS2 bits (15-10) are set to the # of pages occupied by SAM#2 (new first page - old first page); and $MPS2 bits (9-0) are set to the physical starting page of SAM#2, which is the old first page. Since SAM must still reside in the first 32K logical address space, ($MPSA (9-0) + $MPSA (15-10) + $MPS2 (15-10)) must be < 32. If not, a GEN ERR 44 will be issued and the user will be re-prompted. If equal to 32, then the size of SAM#2 stored in EQT4 is decremented by 2, making the last word of SAM equal too 77775. EQT3 is set to the logical starting address of SAM#2 - by setting it to (EQT1 + EQT2) SAM#2 is "relocated" to immediately follow SAM#1. The total size of SAM, EQT2 + EQT4 + EQT6, is displayed in decimal number of words before going on to partition definition.

MEMORY ALLOCATION DEFINITION

The memory allocation table (MAT) and the entry points describing it are located in Table Area II. When the maximum number of partitions $MNP is set by the user, the space for that number of MAT entries is reserved with $MATA pointing to the first entry.

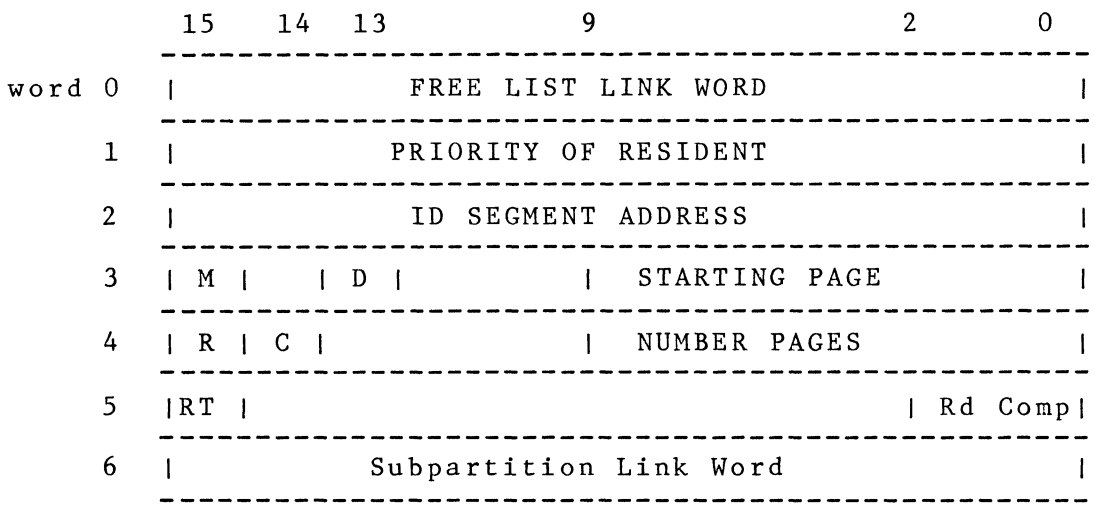
The number of remaining physical pages (DPARE, the memory size stored in NUMPG minus the first partition page PAGE#) is next displayed to the user for partition definitions. The link word (word 0) of each MAT entry is initialized to -1 to indicate an undefined partition, whereas words 1-6 are set to 0. Note that since the MAT is already on the disc it must be referenced thru its absolute memory address, updating the code on the disk via \ABDO. See Table 9 for the format of a MAT entry.

The user is prompted for the definition of each partition starting with "PART 01,XXXX PAGES?", and stopping when a "/E" is entered by the user. The physical pages will be sequentially allocated to the MAT entries and the first -1 link will thus indicate the end of the defined partitions. The user enters the number of pages, partition type (either RT, BG, RTM, BGM or S) and optionally the reserved flag.

The number of pages, less 1 to exclude the base page, is stored in MAT word 4 bits (9-0). If a RT partition, bit 15 is set in MAT word 5 (cleared for BG partitions). If a reserved partition bit 15 is set in MAT word 4. If the partition size is greater than the maximum addressable size MAXPG, then the user is asked if they want to define subpartitions. A NO response simply results in a large unchained partition being defined. If the user responds YES, or if the user entered "RTM" or "BGM", then that partition becomes a mother partition with bit 15 set in word 3 of its MAT entry and the MAT subpartition link word (6) of the mother partition is initialized to point to itself. At this time only can subpartitions for a mother partition be defined. The user has the option of responding YES and still not defining any subpartitions; this would result in a chained partition with the mother partition the only element in the chain. The generator prompts for the next partition definition. If the type code is S then this is a subpartition for the current mother partition. The partition type (either RT or BG) is carried from the mother to the subpartitions. The size of the subpartition cannot be greater than that of the mother, or a GEN ERR 56 will be issued and the partition must be redefined, it can however be larger than MAXPG, but further subpartitioning is not allowed.

The sum of the subpartition sizes cannot exceed the size of the mother, but may be less - a GEN ERR 46 results on that subpartition definition causing the overflow, and that partition must be redefined. If the type is RT or BG, then we've left the subpartition mode and are proceeding as normal.

Table 8-9. Mat Entry Format



M = Mother partition bit
D = Dormant bit
R = Reserved bit
C = Chain in effect bit
RT= Real-time partition bit
Rd= Read completion

PARTITION DEFINITIONS

The following sequence occurs for a partition definition:

A. Clear Subpartition flag, Subpartition prompt flag, and Mother partition flag: SUBS?<--DPMOM<--SUBP?<--0; If NEXTP=MAXPT then set the XX in the prompt to blanks. If in subpartition mode (SUBMD=1) then prompt: PART XX, XXXX (XXXX) PAGES?

B. Else (SUBMD=0) prompt: PART XX, XXXX PAGES? Get the response. If "/E" is entered, the generator proceeds to the partition cleanup at Q. If NEXTP > $MNP then no more MAT entries can be entered and the generator issues a GEN ERR 49 and goes back to B.

C. Retrieve the partition size, subtract 1 for the base page, and store in DPSIZ. DPSIZ must be >= 1 page, else issue GEN ERR 45 and go to A.

D. Retrieve the partition type (RT,RTM,BG,BGM or S); if first two characters are neither RT, BG, or S then issue a GEN ERR 46 and go to A.

E. If S then SUBMD must = 1 indicating subpartitioning enabled, else issue a GEN ERR 46 and go to A.

F. If DPSIZ+1> MLEFT (# Pages left in mother partition) then issue a GEN ERR 56 and go to A.

G. Set current definition flag to indicate a subpartition: SUBP?<--SUBP?+1;

H. Set type of subpartition to that of mother, DPTY<--MOMTY. Go to J.

I. If RT or BG then check upper range: Require (DPSIZ+1<=PLEFT (total pages left)) else issue GEN ERR 45 and go to A. At this point, we know we're defining a RT, RTM, BG or BGM partition, so SUBMD is set to 0 (may have already been in regular mode). Set DPTY to 1 for RT or RTM so bit 15 of word 5 can be set; else set DPTY to 0 for BG or BGM.

J. Check the third character in the response for "M". If "M" is found (RTM or BGM input) then set SUBS?<--2; to indicate a mother partition, but turn off "SUBPARTITIONS?" prompt. If "M" is not found, this still may be a mother partition if DPSIZ > MAXPG (largest logical partition size).

K. Retrieve reserved flag. If one entered, set bit 15 for word 4 (DPRSV<--0, -1 otherwise).

L. If SUBS?=0 or =2, then go to N, else prompt the user "SUBPARTITIONS?".

M. If NO then go to N. If YES: enable subpartition mode SUBMD<--1; store address of current (mother) MAT address in MOMAD; save mother partition size for subpartition checking, MLEFT<--DPSIZ+1; save mother partition type for its subpartitions, MOMTY<--DPTY; and set bit 15 for word 3 of current MAT entry making it a mother partition (DPMOM = -1, 0 otherwise).

N. Build the new MAT entry (words 0 & 3 are completed during partition cleanup).

word 0: set to 0 to indicate a defined entry.
word 3: DPMOM is used to (optionally) set bit 15 if a mother partition
word 4: DPRSV is used to (optionally) set bit 15 if a reserved partition, and DPSIZ is stored in bits 9-0.
word 5: DPTY is used to (optionally) set bit 15 if a RT partition
word 6: if SUBMD=1 then set to MOMAD, else 0. This will set the SLW (Subpartition Link Word) to point to itself if the mother partition, or to the mother MAT entry if a new subpartition at the end of the chain.

O. If SUBMD=1 and SUBS?=0 then at least one subpartition has been defined. The current subpartition must then be linked to the previous MAT entry (which is either the previous subpartition in the chain or the mother partition). Since CURMT is the memory address of the current MAT entry, then (CURMT-1)<--CURMT.

P. If current partition is a subpartition (SUBP?=1) then MLEFT (DPSIZ+1); else PLEFT<--PLEFT-(DPSIZ+1); bump NEXTP. Go to A.

Q. Partition Definition Cleanup. The MAT is scanned, summing up the individual partition sizes, until the first undefined entry is found (link word 0 = -1) or the end of the table is reached. Only the regular and mother partition sizes are included in the total, and 1 is added to each of these sizes because the base page was not included in the size stored in word 4. Subpartition sizes are not included, their pages having already been included in the mother partition; a subpartition MAT entry is detected by word 6 (SLW) being nonzero and the mother bit (15) not being set in word 3. If the total number of pages occupied by the defined partitions (DPTOT) does not equal the number available (DPARE), then a GEN ERR 53 is issued and all the partitions must be redefined.

FREE LISTS

The memory allocation table (resident on the disc) is sorted into three free lists each based on increasing partition sizes, by setting the link addresses in word 0 of each MAT entry. The lists, separating real-time, background and chained (mother) partitions, are referenced thru the Table Area II entry points $RTFR, $BGFR and $CFR respectively.

The generator starts scanning the MAT with the first partition's entry and stops when the end is encountered ($MNP entries have been threaded) or when the first undefined entry is found (link word = -1). The three list headers DPRTL, DPBGL and DPCL are initialized to 0, and are pointed to by DPRT., DPBG., and DPC., respectively. The list headers (and their lists) are accessed and updated by setting DPLH.,I where DPLH. is set to one of the header pointers, depending on the partition type. The partition type is determined as follows:

> If the mother bit of word 3 is set then both RT & BG mother partitions go into $CFR, or if the RT bit of word 5 is set then it's the $RTFR list, and the remaining go into the $BGFR list. The type of list being threaded is irrelevant once the particular header address has been set.

As a particular MAT entry is linked into a list, its starting physical page is stored in word 3 bits 9-0. DPORG is initially set to the first physical page for partitions from PAGE#, and is updated as pages are allocated to a partition. When a mother partition is encountered, MORG is set before DPORG is updated to the start of the next partition. When MORG is non-zero, the next set of subpartition entires in the MAT have their starting physical page set by MORG (which is incremented after each subpartition). When the next non-subpartition is encountered, MORG is cleared and starting pages are set by DPORG again.

When the threading is completed, the last element in each list is retrieved and the Table Area II entry points $MRTP, $MBGP and $MCHN are set to the page sizes of the largest non-reserved partition in the real-time, background, and mother free lists, respectively.

MODIFY PROGRAM PAGE REQUIREMENTS

The IDENT entry for the named program is retrieved; a GEN ERR 48 is issued if the program name can't be found or if it is of incorrect type. Only disk resident programs (masked types 2,3, and 4) executing in user partitions can have their page requirements increased. The page requirements of an EMA program cannot be overidden, so if bit 15 of \ID6 is set, a GEN ERR 55 is issued.

When the program's ID segment is built, the keyword offset is stored in the program's IDENT entry word 8 bits (7-0). The routine IDFND retrieves the program's ID segment address by going thru \ID8 and the keyword's value stored on disc. Before the program's ID segment word 21 can be updated, the new page size must be verified. The program's low main is retrieved from ID segment word 22 and is converted to its starting page to which is added the new page requirements (less 1, stored in DPSIZ). If overflow occurs (>32) then a GEN ERR 51 results. A program's page requirements, stored in its IDENT entry word 8 bits (15-8) when the ID segment was built, are compared against the override in DPSIZ - if DPSIZ is less than a GEN ERR 51 again is issued. Otherwise DPSIZ is stored in ID segment word 21 bits (14-10) of the named program. The page requirement in \ID8 is not updated, however to allow a re-override.

ASSIGN PROGRAM PARTITIONS

The IDENT entry and ID segment address for the named program are retrieved as when modifying a program's page requirements. Only disk resident programs may be assigned to partitions, provided the partition is large enough to hold the program. A GEN ERR 49 is issued if the partition number specified in DPNUM is greater than the maximum allocated (MAXPT) or if the partition is undefined (link word 0 = -1). The size of the partition is retrieved from its MAT entry word 4 bits (9-0) and stored in DPSIZ. The page requirements of a non-EMA program are retrieved from ID segment word 21 compared against DPSIZ. A GEN ERR 50 is issued if the program is too large for the specified partition, otherwise word 21 bits (5-0) of the program's ID segment are set to DPNUM -1 and the RP bit 15 is set.

For EMA programs (bit 15 of \ID6 is set) the page requirements stored in \ID8 bits (15-8) include the MSEG size, but it is the EMA size that must be included when considering whether or not the program will fit in a partition. The program code size is determined by subtracting the MSEG size in \ID6 bits (14-10) from the page requirements in ID8, and adding the EMA size in ID5 (13-4) - adding 1 if the EMA was defaulted-and storing the result in DPORG. If the resulting page size <DPSIZ then ID segment word 21 is updated as mentioned above to reflect the partition assignment, otherwise a GEN ERR 50 results.

MEMORY PROTECT FENCE TABLE

The 6 word MPFT stored in Table Area II on the disk is updated to reflect the logical fence addresses for the following program categories:

Table 8-10. MPFT

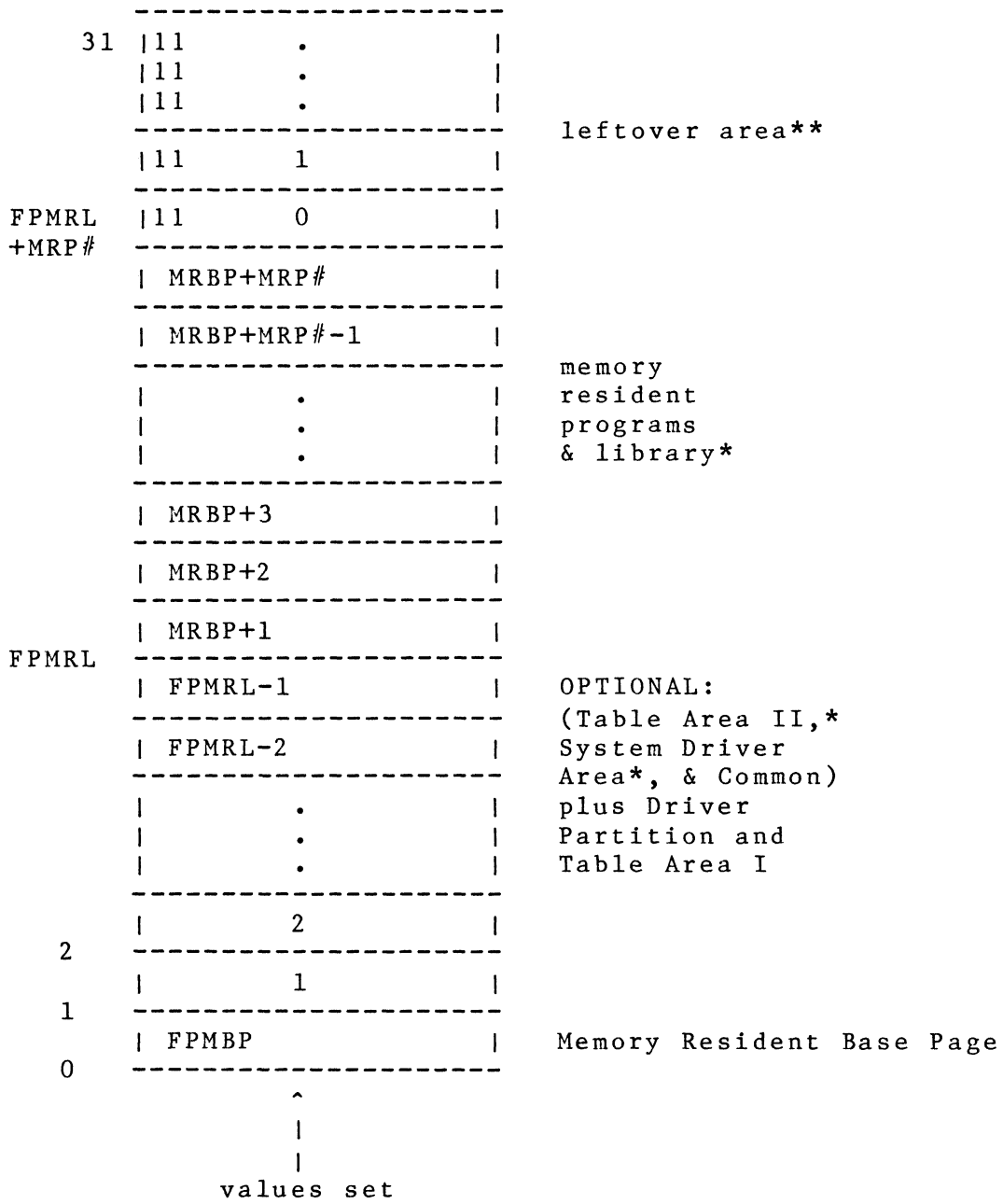
word 0 type 4 BG disk resident without common
1 memory resident
2 any program using RT common
3 any program using BG common
4 any program using SSGA
5 RT or type 3 BG disk resident without common

- Table Area II entry point $DPL (load point for disc resident program) sets word 0.
- The variable FWMRP (first word of memory resident program) sets word 1.
- The variable RTCAD (real-time common address) sets word 2.
- The variable BGBND (background common address) sets word 3.
- The variable SSGA. (SSGA starting address) sets word 4.
- Table Area II entry point $PLP (load point for privileged programs) sets word 5.
- Table Area II entry point $MPFT will contain the address of the MPFT.

MEMORY RESIDENT PROGRAM MAP

The DMS map for memory resident programs, MRMP stored on the disc in Table Area II, is updated for use by the Dispatcher. The MRMP, addressed by Table Area II entry point $MRMP, is 32 words long having one word per physical register. The map is built as follows:

Table 8-11. Memory Resident Map



* System Driver Area, Table Area II, and the Memory Resident Library are write-protected (bit 14 is set).

** Both read and write-protected.

Word 0 is set to the physical memory resident base page FPMBP. The first word of the memory resident library is converted to its logical page address and is stored in FPMRL. Words 1 thru FPMRL-1 are thus set to their logical and physical page addresses, 1 thru FPMRL-1. If the System Driver Area and Table Area II are to be included in the map (MRTA2=1) then their pages are write-protected. MRP# contains the number of pages occupied by the memory resident library and programs. The map words FPMRL thru (FPMRL + MRPGS-1) are thus set to their corresponding physical pages, MRBP+1 thru (MRBP+MRPGS). The library pages (FPMBP+1 to FPMRP-1) are always write-protected. The remaining map words (FPMRL+MRPGS) thru 31 are set starting over at page 0 - this area corresponds to the logical address space above the memory resident area and each page is therefore read- and write-protected (bits 15 and 14 are set in its MRMP entry).

SETTING SYSTEM ENTRY POINTS

Crucial values are passed to the system from the generator. This is done by stuffing values into locations defined as entry points in Table Area II. The code to update these values on disc is table-driven, with a table entry consisting of these 5 words:

```
label DEF *+2
      <value to be stored>
      ASC 3,<entry point name>

or    DEC 0     (last entry)
```

Before updating the entry points, the values in the table are filled in. The following entry point values are set as indicated:

$MRMP, memory address of memory resident map

$ENDS, physical page following SAM#1

$MATA, memory address of memory allocation table

$MPSA, # pages/starting page of SAM#1

$MPS2, # pages/starting page of SAM#2

$MPFT, memory address of memory protect fence table

$RTFR, MAT entry address of real-time free list header

$BGFR, MAT entry address of background free list header

$CFR, MAT entry address of chained free list header

$EMRP, last word address of memory resident program area

$DVMP, memory address of Driver Map Table

$DVPT, logical starting page of driver partition

$DLTH, number of pages per driver partition

$MNP, maximum number of partitions

$MCHN, page size of largest mother partition

$MBGP, page size of largest background partition

$MRTP, page size of largest real-time partition

$IDEX, memory address of ID extension table

$DLP, load point address for RT/BG DR programs without common

$PLP, load point address for privileged DR programs

$LEND, last word +1 address of memory resident library

$BLLO, negative lower buffer limit

$BLUP, negative upper buffer limit

$CL1, system disc track number where cartridge list begins.

$CL2, system disc sector number where cartridge list begins.

$STRK, Disc track number where system (base page) begins.

$SSCT, Disc sector number where system (base page) begins.

When done setting the above values there are six values to be stored in the following table (for use by the configurator):

starting at $SBTB: disc address of driver partitions #2 onward
# of pages for driver partitions #2 onward
disc address of memory resident base page
# of pages for memory resident base page
disc address of memory resident lib/programs
# of pages for memory resident lib/programs

ERROR PROCESSING

There are two classes of errors that occur during generation: FMP ERR's resulting from files being accessed through FMP calls, and GEN ERR's resulting from an illegal generator response or an erroneous condition detected during the generation. In most cases an FMP error will cause a GEN error as well. A count ERCNT is kept for the number of errors occurring during a generation, and is displayed after both normal and abortive generator terminations, in the form: XXXX ERRORS.

On many errors control will be passed to the operator console by calling TRCHK with a "TR,LU" stuffed in the input buffer, LU being the lu of the operator console ERRLU. The current input source is pushed down the stack, so after the operator corrects the error (probably by re-entering the response), a simple TR will return them to the next response in that answer file.

List file errors encountered after the list file has been created are detected in /LOUT. The error that occurs most frequently results when an extent to the list file cannot be created due to lack of disc space on the same subchannel. Because this error can occur anytime during generation, the status of the input/output buffers LBUF and TBUF must be maintained as they may contain relocatable or absolute code. The FMP and GEN errors reported upon the occurrence of a list file error are therefore, issued via EXEC call writes. (Using the normal error reporting routines would result in an eventual call to /LOUT - but recursion doesn't work!) The user is then prompted with an "OK TO CONTINUE?" On a NO response the generator aborts via \TERM call. On a YES response, LFERR is cleared to indicate that all future list file errors encountered in /LOUT are to be ignored. The ECHO option must then be turned on (if not already on).

GENERATION ERRORS

\GNER outputs all errors of the form "GEN ERR XX" where XX is the two digit ASCII error code passed in the A-register. If the A-register is negative, then it implies that we have an error type for which no TR to the ERRLU is to be done (these codes typically pertain to duplicate names or entry points). Otherwise \GNER checks as did \CFIL to determine if control is to be transferred to the operator console; it also saves/restores the return address when calling TRCHK. The flag EOFFL is set in \PRMT to signal that an EOF had been encountered in the answer file. Thus when the POP is done on the answer file stack only to find nothing there, a GEN ERR 19 will not be printed - control will simply be transferred to the console as intended. Since calling \GNER is the realization of an actual error, it is up to the caller to take corrective action.

FILE ERRORS

All FMP errors are detected and processed in the routine \CFIL. \CFIL is called after each FMP call is made (i.e., all READF, WRITF, CREAT, CLOSE, OPEN, LOCF, APOSN and RWNDF calls) and checks the error parameter \FMRR. CNUMD is called to convert the error code to ASCII and stuff it into the message "FMP ERR-XX FLNAME". The DCB address is passed to \CFIL in the A-Register. From the DCB words 0 and 1, the file directory entry address or the lu is retrieved. An EXEC call read is done to that track and sector and the file name is transferred from words 0-2 of the directory entry to the error message buffer. If word 0 of the DCB is 0 then it was a type 0 file and the file name in the error message is set to the lu, "LU XX". Since \CFIL issues error messages, an error can occur on an OPEN or CREAT call in which case the DCB is not set up correctly. Therefore if the A-Register DCB address is zero indicating a check following an OPEN or CREAT call, then the file name is picked up from PARS2+1,+2,+3 since it always contains the file to be opened. An error never occurs on the OPEN/CREAT of a type 0 file since the generator routine TYP0 builds the actual DCB, so this combination is not encountered.

\CFIL also determines whether or not a transfer of control to the operator is necessary, in which case TRCHK is done. Some return addresses are saved and restored in case it was TRCHK who originally called \CFIL. \CFIL has two returns with the error return being at (P+1). It is up to the caller of \CFIL to determine the course of action when a file error occurred.

ABORTIVE TERMINATION

\ABOR

\ABOR issues its own error of the form "GEN ERR 00 XXXXX" where XXXXX is the octal address of the caller of \ABOR. Because \ABOR is called from several places, the address helps in tracing down the problem. After outputting the message, \TERM is called for clean-up before termination. The abort may result if there exists a problem with the generator's LST,IDENT, or FIXUP table or its scratch file (@@NM@A) - such that an entry is no longer there. The loss of a table entry would result from an incomplete disc swap of a table block - this could be an actual generator problem or it could be a hardware problem. Past experience recommends one to check the hardware first on all GEN ERR 00's. Obviously this error should never occur - so only strange conditions will cause it.

\TERM

\TERM is called when the operator aborts the generator with a !! command, when a GEN ERR 00, 02, 07, 17, 18, 21, 38, 57, 59, 60, 61 occur, or after file errors to \NDCB, \EDCB, \RDCB, \IDCB or \ADCB. The absolute output file, boot file and modified NAM record file are purged (using a CLOSE call with truncate option), and the list file, relocatable input file, and answer file are closed. The abort message is printed, the generator releases the scratch tracks allocated to it, and the generator terminates.

MISC. ERROR PROCESSORS

\INER and \IRER:

\INER is called from several places in the main and segments 1, 5, and 7 to issue the initialization response error GEN ERR 01. It merely calls \GNER where the transfer of control is done to the console. The caller of \INER then reissues the questions for the corrected response from the operator.

\IRER calls \GNER for the irrecoverable errors 07, 12 and 21, followed by a call to \TERM to perform clean-up and abortion.

NROOM and CMER:

NROOM issues errors 02 (not enough space for tables, 512 word minimum) and 38 (ID segment of segment 3 cannot be found) by calling \GNER, then aborts the generation with a \TERM call.

CMER in Segment 2 issues a GEN ERR 06 when an invalid Program Input Phase command was entered, or when an FMP ERR-XX FNAME occurred on a file referenced in a RELOCATE command. NXTCM then prompts (-) for the next command.

ERROR SUSPENSIONS

The generator detects two error conditions which result in a message sent to the console (only) and the suspension of the generator until the situation is resolved. When the generator requests its 6 scratch tracks and they are not available, then it issues the message "GENERATOR WAITING FOR TRACKS", and reissues the EXEC 4 call with the wait bit set. The same sequence of operation occurs when an attempt is made to lock the list file (provided it was to a non-interactive lu) where "GENERATOR WAITING ON LIST LU LOCK" is displayed on the console.

ANSWER FILE ERRORS

When doing transfers within TRCHK, special processing must be done for PUSH/POP errors. At POPRR, which results from TR stack underflow, a GEN ERR 19 is issued with a forced "TR,ERRLU". At PUSHR, resulting from TR stack overflow, the stack address is decremented by one to point to word 6 if the previous entry (actually current since the PUSH was never done) and RECOV is called. RECOV pops the stack to the previous entry, thus enabling a "TR,ERRLU" to be done on return in some cases. When an invalid lu was specified on a PUSH, at TR3 RECOV is again called before issuing the GEN ERR 20; the same holds true at TR4 when an error occurred on the new input file, only here we have to save the error code while RECOV is being called.

When an invalid lu or file was specified in the turn-on parameters, STRT2 issues its own errors rather than call \GNER or \CFIL before the answer file and IACOM have been established. Once the lu of the operator console has been determined (default is 1) the ASCII of that lu is stored in the "TR,XX" message to be used later with all "TR,ERRLU" calls to TRCHK.

DRIVER PARTITION OVERFLOW

When multiple drivers are being relocated into a driver partition and a driver overflows the logical memory space reserved for the DP, warning message of the form:

'DRIVER PARTITION OVERFLOW'

is issued. This does not constitute an error condition and no TR,ERRLU is done. The message is informative only, essentially telling the user to ignore the load map printed for the driver just relocated. That driver will be re-relocated into a subsequent driver partition.

Table 8-12. Generator Error Codes

|$	0:	HARDWARE/GENERATOR ERROR (SEND IN BUG REPORT)
|	1:	INVALID REPLY TO INITIALIZATION PARAMETERS
|$	2:	INSUFFICIENT AMOUNT OF AVAILABLE MEMORY FOR TABLES
|-	3:	RECORD OUT OF SEQUENCE
|-	4:	INVALID RECORD TYPE
|-	5:	DUPLICATE ENTRY POINTS
|	6:	COMMAND ERROR - PROGRAM INPUT PHASE
|$	7:	LST,IDENT,FIXUP TABLE OVERFLOW
|-	8:	DUPLICATE PROGRAM NAMES
|	9:	PARAMETER NAME ERROR
|	10:	PARAMETER TYPE ERROR
|	11:	PARAMETER PRIORITY ERROR
|	12:	PARAMETER EXECUTION INTERVAL ERROR
|	13:	SEGMENT PRECEDES DISC RESIDENT MAIN
|-	14:	CHECKSUM ERROR
|-	15:	ILLEGAL CALL BY A TYPE 6 OR 14 PROGRAM TO A TYPE 7
|-	16:	BP LINKAGE AREA OVERFLOW
|$	17:	TYPE 1 OUTPUT FILE OVERFLOW/OR SIZE BELOW 1000 BLOCK MINIMUM
|-($)	18:	MEMORY OVERFLOW
|	19:	TR STACK UNDERFLOW/OVERFLOW
|	20:	INVALID COMMAND INPUT LU
|$	21:	'$CIC' NOT FOUND IN LOADER SYMBOL TABLE
|	22:	LIST FILE ERROR
|	23:	INVALID S OR M OPERANDS, SDA SPECIFIED FOR SYSTEMS DISC DRIVER
|	24:	INVALID SELECT CODE IN EQT ENTRY
|	25:	INVALID DRIVER NAME IN EQT ENTRY
|	26:	INVALID D, B, U, T, X, S, M OPERANDS IN EQT ENTRY
|	27:	INVALID DEVICE REFERENCE NO.
|	28:	INVALID INTERRUPT SELECT CODE
|	29:	INVALID INTERRUPT SELECT CODE ORDER
|	30:	INVALID INT ENTRY MNEMONIC
|	31:	INVALID EQT NO. IN INT ENTRY
|	32:	INVALID PROGRAM NAME IN INT ENTRY
|	33:	INVALID ENTRY POINT IN INT ENTRY
|	34:	INVALID ABSOLUTE VALUE IN INT ENTRY
|-	35:	MORE THAN 63 EQT OR 255 DRT ENTRIES DEFINED
|	36:	INVALID TERMINATING OPERAND IN INT ENTRY
|-	37:	INVALID COMMON LENGTH IN SYS, LIB, OR SSGA MODULE.....
|$	38:	ID-SEGMENT OF SEGMENT 3 NOT FOUND
|	39:	NOT USED
|	40:	INVALID EMA PROGRAM TYPE
|	41:	MULTIPLE EMA DECLARATIONS
|	42:	INVALID REFERENCE TO EMA SYMBOL

CONTINUED NEXT PAGE

Table 8-12. Generator Error Codes (continued)

	43:	INVALID MSEG SIZE
	44:	SAM EXCEEDS 32K LOGICAL ADDRESS SPACE
	45:	INVALID PARTITION SIZE
	46:	INVLAID PARTITION TYPE
	47:	INVALID PARTITION RESERVATION
	48:	INVALID OR UNKNOWN ASSIGNED PROGRAM NAME
	49:	INVALID PARTITION NUMBER
	50:	PROGRAM TOO LARGE FOR PARTITION SPECIFIED
	51:	INVALID PAGE OVERRIDE SIZE
-	52:	ILLEGAL REFERENCE TO SSGA ENTRY POINT
	53:	SUM OF PARTITION SIZES DOESN'T EQUAL # PAGES LEFT
-	54:	SUBROUTINE OR SEGMENT DECLARED MORE COMMON THAN MAIN
	55:	PAGE REQM'TS OF AN EMA PROGRAM CAN'T BE OVERRIDDEN
	56:	SUBPARTITION SIZE OR SUM OF SIZES > THAN MOTHER PART'N SIZE
$	57:	MISSING SYSTEM ENTRY POINT
-	58:	ILLEGAL REF TO TYPE 0 SYSTEM ENTRY POINT BY NON-TYPE 3 MODULE
$	59:	DRIVER PARTITION OVERFLOW
($)	60:	LONG ID SEGMENT LIMIT OF 254 EXCEEDED
$	61:	PHYSICAL MEMORY OVERFLOW
-	62:	INVALID INSTRUCTION REFERENCE TO AN EMA SYMBOL
	-	NO TRANSFER TO OPERATOR CONSOLE
	$	ABORTIVE ERROR

Chapter 9

MTM

This section discusses the flow of control between PRMPT and RPN for the MTM subsystem.

MESSS

MESSS is a system library routine that allows users to interface to $MESS in the scheduler. MESSS is used by many programs to schedule programs when the 'ON' or 'RU' command is issued at an MTM terminal. MESSS now performs the same operation as $TYPE, that is, if the 'ON' or 'RU' was entered and was successful (no returned message from $MESS) then the program to be scheduled will have its session word 3 set to the negative MTM LU number.

PRMPT - RPN

The guts of MTM is really two programs, PRMPT and RPN. PRMPT is a program that is scheduled by interrupt. This means that PRMPT must be relocated at generation time and then one entry XX,PRG,PRMPT be made in the Interrupt Table phase of generation for every select code address, XX, where the user wishes MTM terminal handling.

PRMPT is the program that issues the XX> prompt and issues a class I/O read on the terminal. RPN does the class get and handles any input typed into the terminal. The one exception to this is for MTM scheduling a copy of FMGR. This is PRMPT's responsibility.

The flow of control in MTM actually starts before the interrupt. As mentioned PRMPT must be set up at generation time. The terminal must also be enabled before MTM will do any processing. Typically a terminal is enabled with the "CN,LU#,20B" command. It may be disabled with the "CN,LU#,21B" command.

Enabling the terminal allows the driver to place PRMPT's ID address into the associated EQT. The driver takes the ID address out of the interrupt table (it is in the twos complement form) and places it into a temporary word in the EQT or EQT extension (as a positive address). The interrupt table entry is then replaced with the first word address of the referenced EQT. MTM is now ready to handle the terminal.

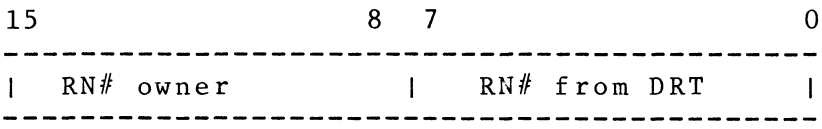
An interrupt from the CRT (TTY) device is generated by hitting any key. $CIC in RTIOC is entered and vectors the interrupt to the appropriate driver. The driver then determines if the terminal is enabled, if not the interrupt is ignored. If the terminal is enabled, the driver schedules PRMPT via the system routine $LIST and passes the address of the fourth word of the appropriate EQT.

PRMPT

PRMPT takes the address of word four of the EQT and calls TRMLU. TRMLU is a new system library routine. Its responsibility is to match up the EQT address to an interactive LU number. This is done by comparing the passed EQT address to the contents of the device reference table (DRT). Recall that the lower six bits of the DRT has the ordinal number of the EQT associated with that LU. TRMLU also insures that the LU number returned is an interactive LU. This is done by checking for driver type. If the driver is DVR00, DVR05 and subchannel zero, or DVR07 then the device is interactive.

When TRMLU returns PRMPT has both the LU number and the EQT. Checks are made to insure that the EQT and LU are up. The availability bits are checked in the EQT and the sign bit is checked in part two of the DRT to insure that the LU is up. A check is also made to see if the CRT LU is locked. If the LU is locked bits 10-5 of the DRT will have the resource number (RN#) of the lock. If that field is non-zero, then the LU is locked.

It is possible to write through an LU lock. Parameter nine (RQP9) of all I/O EXEC requests has been reserved as an LU lock bypass word. The word is configured as:



The RN# owner can be retrieved by indexing into the RN# Table and isolating the lower byte. If the above word is configured and a DEF is made to it in RQP9, then the system will not suspend the executing program and will honor the I/O request.

Next a check is made to see if a FMGXX exists, if so the prompt XX>FMGXX is sent to the terminal and FMGXX is scheduled with the list device set to XX. The schedule request uses the string pass feature to send FMGXX to the HI file before control is transferred to the terminal. Lastly PRMPT reenables the terminal and terminates saving resources.

If FMGXX does not exist or is busy, then the prompt XX> is issued, a class read is performed on terminal LU XX, RPN is scheduled passing the class number. In the case of DVR00 and DVR05, the terminal is disabled to avoid multiple prompts from being written. For DVR07, an edit mode control request is made.

PRMPT does perform one other task. After the very first successful class read, which also requests the class number, PRMPT saves the returned class number in $MTM in Table Area 1. This insures that aborts of PRMPT or RPN do not also lose the class number.

RPN

RPN is the MTM module that does class gets with wait. That is, it receives all input to the terminal, screens it, and passes it on to the operating system via direct call to the operating system routine $MESS.

When RPN is first scheduled, it picks up the class number and tracks down the ID address of FMGR, SMP, and D.RTR for later use.

RPN then performs a class get to get the input data. This get is performed over and over again so that RPN is always get suspended.

The system reschedules RPN whenever there are any input operator commands to process. RPN, on reschedule performs the same EQT, LU, and LU lock checks as does PRMPT. If I/O to the CRT is possible, execution continues. If the EQT or LU is down, the request is ignored and RPN goes back to the class get to suspend itself.

If I/O is possible then three commands are screened. BR, AB, and FL commands are handled locally in RPN. However, if no FMGXX exists only the FL command is honored. In this case, BR and AB are just passed on to the system.

If the input command was not AB,BR, or FL, then the command is sent to the operating system module $MESS. When $MESS returns any messages returned from the system are set to terminal XX with a class write call. This insures buffered writes to unbuffered devices. Lastly, the terminal is reenabled if the driver was DVR00 or DVR05 and a class get is performed so that RPN may suspend itself.

NOTE that RPN does class gets to retrieve its own class write information and PRMPT's class read information.

Chapter 10

CONFIGURATOR

GENERAL OVERVIEW OF THE SYSTEM BOOT-UP OPERATION

The ROM or boot-strap loader loads the boot-extension into memory from disc. The boot-extension loads the system into memory up to the top of the first part of the configurator and then transfers control to the system. The system startup routine $STRT builds the system map and immediately after that makes a subroutine call to the configurator program.

The configurator program loads the memory resident programs, library, base page and the driver partitions. If the user has requested for memory or I/O reconfiguration by setting the switch register bits as described in the following two sections, the configurator will perform the reconfiguration by interacting with the user. For I/O reconfiguration the configurator will allow assigning the I/O device from any select code whose number is 10 octal or greater to any other select code up to 77 octal. The configurator makes the I/O reconfiguration permanent on disc if the user opts for it.

The configurator then asks the user if memory reconfiguration is desired If the response is yes, the user has the option to change the size of the system available memory (SAM) extension, the partition definitions, modify program size and change program assignments to partitions. The SAM extension and user partition definitions are allowed to be defined to avoid bad pages. The configurator makes the memory reconfiguration permanent on disc if the user chooses the option.

Upon completion, the configurator restores the system to its initial state and transfers control back to the system initialization routine.

DISC BOOT EXTENSION

USING THE ROM LOADER

After the ROM loader loads the track 0 sector 0 boot-extensions, the disc boot-extensions examines bit 5 of the switch register. If this bit is set, it means that the user wants I/O or memory reconfiguration. A HLT 77B is issued so that the user can reset the switch register with new disc and console select codes. If bit 5 was not set, no halts are issued and the switch register is cleared.

The disc boot-extension communicates with the configurator via the switch register. Therefore the switch register contents should not be changed until after the completion of the boot-up procedure.

The disc boot-extension then relocates itself to the top of the first 32K. The select code for the disc is extracted from bits 6-11 of the switch register.

USING THE BOOT-STRAP LOADER

The boot-strap loader configures itself to a new disc select code if it was entered in the switch register when the HLT 77B occurred. The boot-strap then loads the track 0 Sector 0 boot-extension in the top of the first 32K of the memory. The disc select code is passed to the boot-extension by setting it in bits 6-11 of the switch register.

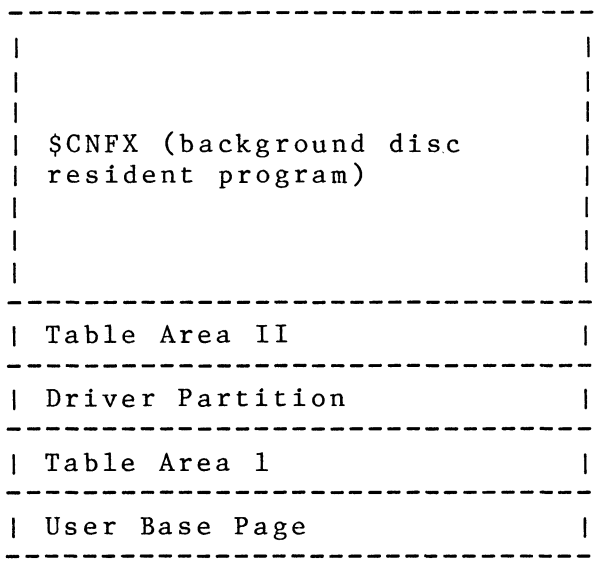
The disc I/O instructions are then configured. Disc boot-extension now loads the RTE-IVB system into memory as one block up to the top of first part of configurator ($CNFG). The high address had been set up in a variable HIGH in the disc boot-extension by the on-line generator. Control is then transferred to the system startup routine.

If the size of the disc boot-extension has to be changed in the future, the generator and some unreleased utilities (CMM4, etc.) will be affected.

TECHNICAL DETAILS OF THE CONFIGURATOR OPERATION

STRUCTURE OF THE CONFIGURATOR PROGRAM

The configurator program is divided into two parts which are relocated as separate programs $CNFG and $CNFX by the generator. The first part of configurator $CNFG is relocated as a type 16 program. Type 16 is a special type used only for the configurator which signals the generator to load $CNFG as a system module overlaying the default SAM. The second part of configurator $CNFX is relocated by the generator as a type 3 background disc resident program. It is loaded into memory by $CNFG under a background disc resident program map.



MAP USED BY $CNFG TO LOAD $CNFX

The disc address for $CNFX is taken from the 26th word of the ID segment for $CNFX. $CNFX is loaded into the first contiguous block of memory that is large enough, starting from the end page of memory resident program +1 if there is no SAM extension; otherwise, it is the last page of SAM extension +1. The base page is loaded starting at logical address contained in the 24th word of the ID segment for $CNFX. The next sequence of physical pages up to the end of Table Area II is copied from the system map. The # of DMS registers to be copied is: page # extracted from $PLP mimus 1. The $CNFX program is then loaded starting at logical address contained in the 22nd word of the ID segment and physical page is physical page # for user base page. The high address up to which $CNFX is loaded is taken from the 24th word of the ID segment.

The configurator program is divided into two parts to make more memory available to it since it cannot fit in the first part of SAM. By making $CNFX a type 3 program, it can access all the system entry points and thus communicate with $CNFG. $CNFG has to reside in the first part of SAM as a system module, so that is can use the $XSIO routine in RTIOC for input and output. EXEC calls cannot be used since the system has not been initialized. The work load divided between $CNFG and $CNFX is based only upon how much code can fit in the first block of SAM. $CNFG will load the memory resident and driver partitions, reconfigure I/O and contain the I/O subroutines also to be used by $CNFX. $CNFX will handle memory reconfiguration.

The two configurator programs communicate with each other through external entry points defined in $CNFG. Everytime I/O has to be performed during execution of $CNFX, an SJS instruction is issued by $CNFX to jump to the I/O subroutine in $CNFG. To obtain the parameter values defined in the calling sequence, $CNFG does a cross-map load. To return to $CNFX from the $CNFG I/O subroutines, $CNFG performs a UJP return addr,I instruction.

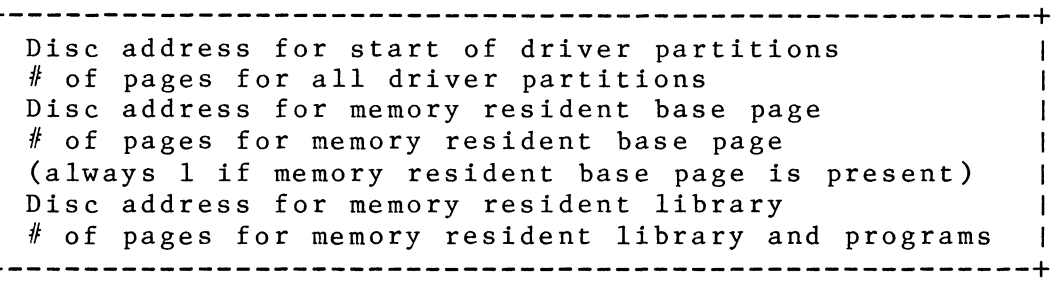
When the configuration procedure is completed, the $CNFG program will be overlayed by buffers using the system available memory. These tables show an image of physical memory after the disc boot-strap load and after the configurator load.

INITIALIZATION PROCEDURE FOR $CNFG

The $CNFG program clears all interrupts as soon as it is given control. It then saves the base page locations SYSTY (EQT entry address of system TTY), DUMMY (privileged I/O card location) and EQT1-EQT4 and clears them. Clearing SYSTY prevents the user from gaining control of the system by striking a key on the keyboard of the system console and getting a prompt. DUMMY is cleared to prevent any privileged interrupts. SKEDD is cleared and $LIST is set to 1, to prevent any scheduled programs from running. These locations will be restored just before $CNFG returns control to the system start-up routine. If the console or list device is buffered, $CNFG will clear the B bit in word 4 of the device's EQT to make it unbuffered so that system available memory is not needed for I/O. The original buffered or unbuffered status will be restored before returning control to the system. I/O errors cannot be handled due to the fact that operator console capability is taken away from the user. If an I/O error does occur, the boot-up procedure will have to restart. The $CNFG program will pickup the new select code (if any) for system disc from the switch register and reconfigure it (see replaced with what the entries were before any the I/O Reconfiguration Section for details) in memory so that disc accesses can be made to load memory resident and driver partitions.

LOADING THE MEMORY RESIDENT PROGRAMS AND DRIVER PARTITIONS

The on-line generator passes a table $SBTB of six parameters to $CNFG.



Each of the # of pages and starting disc address couplets in $SBTB are broken up into several triplets of the form:

starting memory address
number of words to transfer
starting track/sector address

These triplets are setup to avoid crossing track boundaries. They are passed to the $XSIO routine in RTIOC module for reading the corresponding code from the system disc.

The driver partitions are loaded into memory as one big piece of code. The on-line generator stores the relocated drivers in partition size chunks on the system disc. If the # of pages taken up by the driver partitions is greater than maximum addressable partition size, $CNFG breaks them up into chunks large enough to fill up the logical address space and loads the driver partitions a chunk at a time. The maximum logical address space is determined by 31-$CMST ($CMST is the starting page of common). The first chunk of driver partitions is loaded under a user map whose starting page is contained in $ENDS entry point. Starting physical page # to build user map to load subsequent blocks of driver partitions is determined by adding the size of maximum logical address space to the starting page # used to build the previous map. Starting disc address for subsequent chunks is determined by picking up the # of words from the last set of triplets, divide it by 64 to get # of sectors and add it to the track/sector address in the triplet.

If # of pages for memory resident base page is 0, then the system has no memory resident programs. In this case, the configurator will proceed to configure I/O. In the event that the memory resident programs have been generated into the system, $CNFG loads the memory resident partition map $MRMP into the mapping registers. Memory resident base page is then loaded into memory starting at logical address 2. The memory resident library and programs are loaded into memory under $MRMP map starting at logical address contained in base page location LBORG.

Next step is I/O Reconfiguration.

I/O RECONFIGURATION

I/O reconfiguration is performed in $CNFG by assigning the current select code's trap cell and interrupt table entry to the new select code. The equipment table entry pointing to the current select code is changed to point to the new select code. Initially, the changes needed to be made for I/O reconfiguration are recorded in tables in $CNFG's area of memory. At the end of the I/O reconfiguration, these changes are transferred to the trap cell, interrupt table and equipment tables in the system in memory. To enable the configurator to load the driver partitions and memory resident programs and also to perform I/O and memory reconfiguration interactively, the system disc, system console and the list device select code configurations have to be changed in the actual tables in the system in memory before the reconfiguration process can begin

I/O RECONFIGURATION TABLES

Several tables are used to record changes made to trap cells, interrupt table and equipment table during the I/O reconfiguration process. All tables are initialized to -1. Following is a description of these table

TRPCL - is 70 words long with one word per entry. Each entry correspond to the actual select code numbers varying from 10 octal to 77 octal. TRPCL is used to hold all changes made to trap cells during I/O reconfiguration.

INTBL - has the same structure as TRPCL. This table is used to record all changes made to the interrupt table.

EQTBL - has the same structure as TRPCL. This table is used to contain address of EQT word 4, the select code entry which has to be changed to point to the new select code. An entry can be made in this table only if an EQT pointing to the corresponding current select code can be found.

OLSTB - has the same structure as TRPCL. This table is used to contain the current select code number corresponding to the new select code. No entry is made if the new select code is assigned a privileged I/O card by the entry

```
x,PI
```

where x is the new select code.

OLSTB is used for the following situations:

1) If the sequence of responses is:
   x,y
   x,z

   where x is the current select code and y and z are the new select codes.

   OLSTB is scanned to check if select code x has been previously assigned to another select code y. If so, the changes made to y are erased from TRPCL, INTBl and EQTBL. x is now assigned to z.

2) If the sequence of responses is:
   x,y
   z,y

   OLSTB is used to get the previously assigned select code x. TRPCL and INTBL entries of x are erased.

SVTBL - has four entries containing information for up to two new system disc select codes, one new select code for the system console and a new select code for the list device. The format for each entry is:

word 1 - new select code number
word 2 - original trap cell contents for the new select code
word 3 - original interrupt table contents for the new select code
word 4 - address of the EQT word 4 that originally contained the new select code number

The interrupt table and trap cell entries in the system tables are changed to the reconfigured system disc, system console and list device select codes. SVTBL is used to remember original values for the new select codes to which these devices are assigned.

RSTBL - has four entries containing information for the system disc select codes, system console and the list device select codes. Each entry is six words long and the format is:

word 1 - old (current) select code number for the device
word 2 - INTBL entry for old select code
word 3 - TRPCL entry for old select code
word 4 - INTBL entry for new select code
word 5 - TRPCL entry for new select code
word 6 - EQTBL entry for new select code

RSTBL is used to set up the TRPCL, INTBL, EQTBL and OLSTB tables to the state they were in before starting the I/O re-configuration phase. This is done to re-start I/O re-configuration when /R is entered when responding to the "CURRENT SELECT CODE, NEW SELECT CODE?" query.

I/O RECONFIGURATION PROCEDURES

I/O re-configuration is done mainly by two procedures, INENT and IPROC. INENT is used to fill entries in TRPCL, INTBL, EQTBL and OLSTB as the responses for the current and new select code pairs are accepted. IPROC is used to transfer the changes made due to these responses during I/O re-configuration to the appropriate tables in the system. A description of these two procedures follows:

INENT - Say the current and new select code pair response just accepted

```
x,y
```

where x is the current select code and y is the new select code. OLSTB is scanned to find out if x had been assigned to another select code z previously, i.e., if one of the previous responses was

```
x,z
```

If such a response was made these steps are followed:

1) Scan SVTBL to see if select code z is the new select code for the system disc, system console or the list device. If so then an error has been made in the x,y response since the assignment of new system disc, system console or list device select code cannot be changed.

2) Erase the assignment of x to z. If one of the previous responses was z, w, then the TRPCL and INTBL entries for select code z get JSB LINK,I (where LINK is a base page address containing the address of $CIC routine) and 0 respectively. If z was not assigned to another select code previously, TRPCL and INTBL entries for z are assigned -1 to signify that no changes were made for select code z. OLSTB and EQTBL entries for z are also changed to -1.

   If z used to be the select code for TBG or the privileged I/O card originally, restore it.

If the new select code y is also a new select code for the system disc, system console or the list device, then an error.

If y is currently the select code for the TBG or privileged I/O card, unassign it.

If x is currently the select code for the TBG or privileged I/O card, then y is now the new select code for one of these cards.

If select code x is a new select code for the system disc, system console or the list device, use SVTBL entries for x to fill in TRPCL, INTBL and EQTBL entries for select code y. Otherwise use the entries from system trap cell and interrupt table for select code x to fill TRPCL and INTBL entries for y. Scan the equipment table to find an EQT pointing to select code x. If such an EQT is found enter the address of its fourth word in EQTBL entry for y.

If the assignment of current select code x has not been changed during the I/O re-configuration process, assign JSB LINK,I (where LINK is a base page location containing address of $CIC routine) to TRPCL entry for x and a 0 to INTBL entry for x. Return to the calling routine.

IPROC - This routine is used to transfer the changes made as a result of responses of current and new select code pairs to the appropriate tables in the system. Say select code y has been passed to IPROC. If there was no change made in the assignment for select code y, then return.

Transfer TRPCL and INTBL entries for select code y to the appropriate trap cell and interrupt table entries. If there is an EQTBL entry for y, then get the OLSTB entry for y, say x. Then x was the current select code and y the new select code. Scan all EQT's and for every EQT that points to select code x these steps are performed.

1) If the EQT is for the system disc, system console or the list device, then make no changes and look for the next EQT pointing to select code x.

2) If the first word of the EQT is a -1, it indicates that the EQT has been changed to point to select code x, then make no further changes and look for the next EQT pointing to select code x.

3) Change the fourth word of the EQT to point to select code y and set the first word to -1, indicating a change has been made in this EQT. Look for next EQT pointing to select code x.

After all the EQT's have been searched, scan the OLSTB table to find out if an entry for the select code y has been made. If select code y was never used as a current select code, then scan the EQT's to clear out any unchanged EQT's that may be pointing to select code y. For every EQT these steps are followed:

1) If the first word of EQT is a -1, make no changes.

2) If the EQT is for the system disc, system console or the list device make no changes.

3) If the EQT points to select code y then clear the select code entry in the fourth word of the EQT.

Return from the IPROC routine after all the EQT's have been examined.

Making I/O reconfiguration permanent:

To make I/O re-configuration permanent the following tables and base page locations must be written out on disc: Interrupt table, Trap cells, Fourth word of all EQT's, first word of the device reference table (DRT) which is the entry for the system console, base page locations TBG (1674B), SYSTY (1675B) and DUMMY (1737B).

Prior to writing the interrupt table on the disc, its entries for the new console and list device select codes are saved. These are replaced with what the entries were before I/O was performed to these two devices. The saved entries are restored after the interrupt table is written on disc. This is done because some of the console and list device drivers, when executed for the first time, change the interrupt table entries.

MEMORY RECONFIGURATION

For the most part memory reconfiguration is straight forward. $CNFG accepts a list of up to 100 bad pages in an increasing order. The end of the list is marked by a -1. In $CNFX program System Available Memory (SAM) extension and user partitions can be re-defined to avoid the bad pages in memory. To modify program size, assign and unassign programs from partitions, changes are made in the program's ID segment. This section describes the definition of SAM extension and user partitions.

DEFINING SAM EXTENSION

The number of pages in the SAM extension as it is currently defined are determined from the system entry point $MPS2. The physical starting page number for SAM extension is determined by adding contents of $ENDS, # of pages taken up by the driver partitions, # of pages for memory resident base page, library and programs. These last three values were passed to the configurator by the generator in $SBTB table. $CNFX then checks to see if this resulting starting page of SAM extension is included in the list of bad pages. If so, start page of SAM extension is incremented to avoid the bad page (and any other consecutive bad pages). If $MPS2 is not zero, this start page is compared with it. If they do not match, $MPS2 is changed to match this newly evaluated starting page. This case may happen if some previously bad pages at the start of SAM extension were replaced with a new memory module or some pages at the start of SAM extension went bad. Maximum number of pages available for SAM extension is:

```
say a=32-$SENDS
    b=# of pages in physical memory-start page of SAM extension
```

If a>b then # of pages in SAM extension is b; otherwise, it is a. If a change in SAM extension is desired, $CNFX sets up $MPS2 word with start page and # of pages for SAM extension. A scan of the list of bad pages is performed next. The number of pages in SAM extension are divided up into blocks of memory between bad pages. For every block of memory on which SAM extension is defined, two entries are made in the 10 word long $SMTB table (initialized to 0) in $CNFG. The first entry indicates the starting page number of the block of memory. The second entry contains the number of pages included in this block of memory. If the size of SAM extension is such that more than five blocks of good memory are required then an error. The system map is set up to reflect the new page numbers used by SAM extension. SAM extension is write protected.

$SMTB table is written out on disc in $CNFG's area if this memory reconfiguration is made permanent. After this point, every time the system is booted-up, $CNFG checks the $SMTB table. If SAM extension was divided up into more than one block of contiguous memory, the system map is set up for the physical page numbers recorded in $SMTB.

If SAM extension ends on page 31, the number of words in SAM extension must be decreased by 2 so that the last address for SAM extension is 77775B. The last two words of page 31 are reserved for system use.

DEFINING USER PARTITIONS

$CNFX picks up blocks of memory between bad pages or pages remaining if there are no more bad pages and prompts user to define partitions for it. Partitions defined for a particular block of memory must use up all the pages in the block, otherwise $CNFX asks the user to redefine partitions for that particular block. $CNFX fills the start page of partition, # of pages, reserved bit R and RT or BG values into the MAT entry for the partition. If the partition is defined to be a mother partition, the M-bit in the MAT entry is set. Every partition following with an 'S' as the fourth parameter is a subpartition. The block of memory to be allocated to subpartitions is the same as that used by the mother partition. The subpartition link word (SLW) of the mother partition is made to point to the first subpartition, SLW of the first subpartition points to the second and so on. SLW of last subpartition points back to the mother partition.

After all the partitions have been defined, $CNFX threads them into either real time, background or chain partition (mother) free lists. Several passes have to be made through the MAT table to thread each list. $MBGP, $MRTP, $BGFR, $RTFR and $CFR entry points are set up by $CNFX while threading these lists.

Making Memory Reconfiguration Permanent:

The tables to be written out on disc to make memory reconfiguration permanent are:

$SMTB in $CNFG's area
Memory Allocation table (MAT)
ID segments

The system entry points to be written out on discs are:

$MCHN, $MBGP, $MRTP, $BGFR, $RTFR and $CFR

PROCEDURES TO TRANSFER DATA FROM MEMORY TO DISC

Following is a description of two of the procedures used to make memory or I/O reconfiguration permanent.

MEMDS: This procedure is used to convert the given memory location in system code into a corresponding disc location. The disc location is determined in terms of track #, 128 word sector #, and # of words offset within the sector. Following steps are performed:

1. Divide memory location by the # of words in a track on the system disc. The quotient is the track #.

2. Divide the remainder by 64. The remainder is the # of words offset into the sector.

3. Add the number contained in $SSCT entry point sectors to the quotient (to account for the boot-extension) to get the sector number.

4. If sector # is greater than the number of sectors per track, increment track number by one and the sector number is 0.

$TRTB: This procedure is used to transfer a table from memory to a corresponding disc address.

1. Use procedure A to determine track #, sector # and # of words of offset into the sector.

2. Read the 128 word sector into a buffer.

3. Move the first (128-# of words offset) or # of words in the if <(128-# of words offset) words of the table into the 128 word sector buffer starting at the offset word and overlaying the contents of the rest of the sector buffer.

4. Write sector buffer back on disc.

5. Divide the remaining number of words in the table by 64. The remainder is the number of words that fall in the last sector of the table.

6. Transfer the remaining number of words not yet written on disc minus the remainder obtained in step 5 to the disc.

7. Read next sector from disc.

8. Move the remaining # of words from the table into the sector starting at word 0.

9. Write sector buffer back on to the disc.

Chapter 11

SWTCH

INTRODUCTION

This Chapter is intended to serve as an aid to the programmer when modifying the SWTCH program. It assumes that the reader is familiar with the RTE-IVB generation process, and has run both the on-line generator and SWTCH. It assumes familiarity with the RTE-IVB System Manager's Manual, especially the section on the operation of SWTCH.

This Chapter should be used in conjunction with the SWTCH listings, as it in no way attempts to provide the level of detail given by the code itself and comments. The technical specifications will aid in using the listings by discussing overall structure and flow, some data structure and complex algorithms/techniques.

OVERVIEW OF SWTCH ORGANIZATION

SWTCH consists of a main program and two disc interface segments - one which is a self-contained 7900 disc driver, the other, a segment which calls disc primitive routines in the Disc Utility Library ($DSCLB) in order to interface to 7905/06(H)/10H/20(H)/25(H) discs. Because SWTCH will transfer either a 7900 or MAC/ICD RTE system, both segments are present, but only one segment is rolled in and used per execution of SWTCH.

Before looking at the layout of SWTCH, a few words must be said about the internal buffering scheme. SWTCH maintains one buffer, BUFR, for use in reading from and writing to the new system area. BUFR is declared to be 128 words long - the size of a type 1 FMP record or one disc block. As SWTCH progresses, the buffer area covered by BUFR increases to 512 words, and finally to 8192 words, (the size of one track on the largest disc). What happens is that as BUFR "grows" the data stored in it overlays SWTCH code that will not be referenced again. When the point in the SWTCH code is reached where no more no-longer-necessary code can be overlaid, then there will be a BSS to fill out the rest of the needed buffer space. For example, the BSS 8192+BUFR-* reserves space for the full track of information necessary to be read in by means of the DISKD call of the next line.

LAYOUT OF SWTCH CODE

```
MAIN PROG: IBBUF-16 words (command buffer for $DSCLB)
 (SWTCH)   BUFR- 128 words                512-wd
           messages                       buffer

           messages, constants
           GTLEN,READD
           VFYSY,VTOSO                    overlayable
           STDSK                          subroutines              8192-wd
           OK?,YE?NO,TARGT                                         buffer
           PARMP,PYN
-------------------------------------------------------------------------

Main Entry Pt:

           SWTCH
           VERIF - verify validity of system file.
           \SWTM - display destination I/O Configuration.
           OKAY
           SAVE?
           SUBI?
           SUBI5
           INIT?
           AUTO?                          end of 8192-wd buffer
           PURGF                          (overlaid code)
-------------------------------------------------------------------------

           BFULL
           PUR6                           Non-overlaid code
           XFER
           DDONE
           ISUBS
           UPTAT
           BOOT?
           \XOUT
           SWAPD
           ULDSK
           FINSH
           DISKD - Subroutine which calls the correct segment to do
           SPINT   Disc I/O
           CLRBF
           UPDAT
           PURGT                          subroutines
           \BLIN,\DSPL,\RDIN,\DFLT,LOOP     & etc.
           \CVAS,GETD,GET#,CHKSM
           variables,messages constants
```

```
SEGMENT1:
 (SWSG1)
(7900 driver)
            constants
            \STD0
            \GDMA
            \RDMA
            INIER
            I/OTB & I/OTC
            DISK0,INTON
            SEEK,STATC
            ESUB
            messages

SEGMENT 2:
(SWSG2-ICD/
 MAC interface)
            \SETD
            \DSK5
            DSG0
            ENDBR
            ENDOK \
            FAULT  \
            RECAL   \
            EOCYL    \
            DSKER     > Branch Table processing blocks
            DEFTR    /
            ILSPR   /
            ST2ER  /
            UWAIT /

            XFER
              SEEK?
              ADRC?
              FMSK?
              READ?
              RDFS?
              WRIT?
              INIT?
              VRFY?
              ENDX?
              XEXIT
            SKIP?
            CKST1
            STFIX
            REQST
            NIXSP
            FMTR?
            DADTR
            CYLOG
            RPORT
            ESUB
            SPECR
            NOSPR
```

PARER
NRDER
FRMER
PROTR
DCYLR
INBLK

constants
messages

TURN-ON PARAMETERS

SWTCH allows the following turn-on sequence:

RU,SWTCH,namr,scB/disc LU,addr/unit/pltr,autoboot,filesave,type-6,init

where:

namr — is the name of the FMP file that contains your generated system. This may be specified in the following form:

filename[:security code[:cartridge label]]

This file must exist on a standard host system subchannel. If a target cartridge is to be inserted for the SWTCH process, the file must not exist on the cartridge that is to be swapped out for the target.

scB/disc LU — sc: for the 7900 disc, sc is the select code of the target disc controller (octal value with a B as the terminating character). This target select code does not need to be configured into either the host or the destination RTE system. It is used as a means of specifying the correct controller I/O card for the transfer. SWTCH configures its own driver to this select code.

disc LU: for switching MAC or ICD based systems the target disc LU is the logical unit number of any disc subchannel on the target disc. The LU is not affected by SWTCH. It is a reference for SWTCH to find the select code of the target disc driver. The target disc driver, DVR32 for MAC discs, or DVA32 for ICD discs must be present in the host system.

Neither LU2 or LU3 should be specified as the target disc LU because the system does special checks to protect these LU's. If the target disc being initialized contains more sectors per track than the host systems LU2 or LU3, SWTCH will be aborted with an I007 error.

addr/unit/platter

address: for ICD discs, enter the target ICD address number (0-7) where the new system will be stored.

unit: for MAC discs, enter the hardware unit number (0-7) where the new system will be stored.

platter: for 7900 discs, enter the logical surface number where the new system will be stored (0, 2, 4, or 6 for the fixed platter; 1, 3, 5, or 7 for the removable platter).

The disc system will be transferred to the subchannel that was defined as LU2 during system generation.

autoboot

is the automatic boot-up option.

Specify Y (yes) to attempt an automatic boot-up following the transfer of the new system. The host configuration must match the destination configuration. See the paragraph titled AUTOBOOT SPECIFICATION for more detail on this match.

Specify N (no) to deny automatic boot-up.

filesave

is the filesave option.

Specify Y (yes) to attempt saving the target disc's current file structure during the transfer.

Specify N (no) to deny saving the target disc's current file structure.

type-6

is the option to purge Type 6 files.

Specify Y (yes) to purge the target disc's Type 6 files during the transfer.

Specify N (no) to deny purging the target disc's Type 6 files.

init — is the subchannel initialization option.

Specify Y (yes) to request initialization of destination disc subchannels other than the system subchannel. SWTCH will prompt you for each subchannel that was defined to be on the same disc controller (MAC discs) or interface card (ICD discs) as the system subchannel.

Note that SWTCH will not initialize subchannels defined on the 9895 floppy disc. This must be done with the FORMT utility.

Specify N (no) to deny additional subchannel initializations.

The variables capable of being specified by the SWTCH turn-on parameters (\TDLU,\TSUB,\TUNT,AUTO,\SAVE,TYP6,\SUBI), are initialized to -1 to indicate an unspecified state. When a parameter is skipped or erroneous (to the extent that it should have been ASCII/numeric) its value remains at -1 so that the parameter value will be prompted for when the proper time comes. Note that once a MAC/ICD system has been determined, \TUNT is set to value of \TSUB.

The variable BATCH is initialized to -6. During turn-on parameter retrieval (PARS thru CP3), every time a valid parameter type is obtained, BATCH is incremented. So when BATCH is 0, SWTCH runs in an automatic, non-interactive mode. SWTCH then proceeds without intervention except when an error occurs where an operator response or decision is needed.

NAMING CONVENTIONS

Four naming conventions are being used throughout this spec:

HOST = System subchannel definition of the system under which SWTCH is executing.

DESTINATION = System subchannel definition of the new system as defined during generation.

TARGET = Temporary specification of disc channel, subchannel, and unit for use by SWTCH during the transfer.

SOURCE = Subchannel definition of that subchannel containing the system file.

There exists a set of similar variables used by SWTCH.

HCH	host system disc channel
HEQT	host system disc type
HSBCH	host system disc subchannel
HUNIT	host system subchannel unit (ICD/MAC)
HNHD	host system subchannel starting head # (ICD/MAC)
HNSU	host system subchannel # surfaces (ICD/MAC)
HFTR	host system subchannel first track
H#ST	host system subchannel #sectors/track
HTTY	host system operator console channel
PI	host system Privileged Interrupt Channel
TBG	host system TBG channel
\DCH	destination system disc channel
\DSUB	destination system disc subchannel
DEQT	destination system disc EQT type (31 or 32 octal)
\DUNT	destination system subchannel unit/address (ICD/MAC discs)
\DFTR	destination system subchannel first track
\DNTR	destination system subchannel number of tracks
\DSHD	destination system subchannel starting head # (ICD/MAC discs)
\DNSU	destination system subchannel # surfaces (ICD/MAC discs)
\DNSP	destination system subchannel # spares (ICD/MAC discs)
DTTY	destination system console channel
DPI	destination Privileged Interrupt channel
DTBG	destination TBG channel
\TDLU	target system disc LU (for ICD/MAC discs) or target select code (for 7900 discs)
\T32C	target system disc channel for (ICD/MAC) discs
\TSUB	target system disc subchannel (7900)
\TUNT	target system disc unit (ICD/MAC discs)

MAJOR PROCESSING BLOCKS

OUTPUT FILE TEST

SWTCH attempts to open the FMP file that contains the generated system and prints the FMP error number if the open fails. Next SWTCH reads the first 4 records of the file and checks to make sure that the file contains a valid RTE-IVB system via the VT0S0 routine. SWTCH also computes a 256 word checksum for later use (SWTCH will verify that the operator did not accidentally remove the disc media containing the FMP file when given the opportunity to insert the target cartridge).

SEGMENT LOAD

SWTCH tests the $DATC entry point (in Table Area I) to verify that the system software is Rev. 2001 or later. This is necessary because the EQT lock software and special drivers (DVR32 & DVA32) were not introduced until this time. At this point, the proper disc interface segment (SWSG1 for 7900 discs or SWSG2 for ICD/MAC discs) is loaded and remains in memory for the duration of SWTCH.

NEW SYSTEM I/O CONFIGURATION

SWTCH displays all the select codes and driver types in the destination system based on the information contained in the header records. The subchannel organization of LU2 is also displayed.

TARGET DISC INFORMATION

SWTCH provides a large amount of flexibility when installing the new system on a target disc drive. The 7900 disc interface segment has its own internal disc driver which turns off the interrupt system and bypasses the RTE I/O system. This is necessary so that privileged disc commands can be sent to the disc controller in order to initialize the disc and the RTE on-line driver does not have this capability (e.g. WRITE a track with the "protect" bit set). The 7900 segment requires only a target select code so that the driver may configure all of its I/O instructions to this select code. This select code does not have to exist in the host system since the transfer will be done without the aid of the operating system. Note that SWTCH does have to acquire a DMA channel for its use and prevent the system from allocating it to another driver. The \GDMA routine handles this and also sets up the port map for use by SWTCH.

If installing an ICD/MAC system, SWTCH uses SWSG2 to interface with the Disc Utility Library ($DSCLB), which uses the RTE on-line driver in a special mode. $DSCLB requires a target disc LU instead of a select code. The subchannel information for this target LU is never used; the LU is used only as tool for the library to determine which driver to call (DVA32 or DVR32). Note that the target disc address/unit is not derived from the LU specified, but is specified by the SWTCH user and does not have to be a valid address/unit in the host system. The $DSCLB routines perform special EXEC calls to the on-line driver to perform privileged disc operations that are not possible with standard user EXEC calls. SWTCH locks the EQT of the target disc for the entire duration of the SWTCH. This is necessary for two reasons. The user is given an opportunity to remove a host system disc cartridge and replace it with a temporary target cartridge. The EQT lock is used to suspend all I/O to the host cartridge (e.g. edits, FMP operations) before removing it. The EQT must remain locked while the target cartridge is in the disc drive so that no host I/O will be accidentally performed on the target disc. The second reason for the EQT lock is that SWTCH performs individual subroutine calls to $DSCLB to SEEK and WRITE, for example. The lock guarantees that no other I/O request will move the heads on the disc between the two subroutine calls.

TARGET CARTRIDGE INSERTION

The operator is given a chance to insert a temporary target cartridge in the drive at OKAYY. Before the "NOW IS THE TIME.." message is issued, SWTCH does an EXEC call to lock itself into memory and then calls EQTRQ to lock the EQT. The memory lock and EQT lock must always be done in this order to prevent a deadlock situation where the system may try to swap out SWTCH after the EQT is locked (the XSIO request will be blocked by the lock). Next SWTCH does a dummy I/O request to the target disc LU so that all pending I/O requests on the EQT are completed before the operator is told to remove the host disc media. (SWTCH's dummy I/O request will be linked behind all requests currently waiting on that EQT.)

SAVING TARGET FILE STRUCTURE

If the user requested the filesave option, SWTCH calls VFYSY, which determines the expected location of the directory track according to the destination subchannel definition for LU2. If a valid directory track is found there, SWTCH allows the user to save all the files above (last track of new system + 9 (scratch tracks)). If some FMP tracks must be overlaid (because the new system is larger than the old), SWTCH warns the user "NEW SYSTEM WILL DESTROY SOME FMP FILES". If a valid directory is not found, the user is told that the information will be destroyed.

SUBCHANNEL INITIALIZATION PROMPTS.

7900 Initialization

For 7900 discs each subchannel whose # of tracks word is non-zero is prompted for initialization. If a /E is entered, the initialization prompting is terminatd and AUTO? is transferred to. On YES responses, the TARGET PLATTER is requested. On all responses (even defaults), the target platter is checked against \TSUB, the target platter of the system subchannel. If it is not the same, that subchannel's entry in \TMT is updated as follows:

\TMT+SUBIA: bit 15 is set to indicate initialization
\TMT+SUBIA+16: set to the target platter

ICD/MAC Initialization

For ICD/MAC discs, SUBI5 divides the subchannels into two groups depending on whether or not their destination unit is the same as the destination unit of the system subchannel.

\TMT is scanned first for all those subchannels defined on \DUNT. Each subchannel is prompted for initialization by INIT?. INIT/ matches on \DUNT and always sets the target platter to \TUNT for YES responses to these subchannels.

All remaining subchannels (except 9895 floppy subchannels) are prompted for according to their defined address/unit, from 0 through 7 but not equal to \DUNT. If the TARGET UNIT? response to a group was not a /E, then a check is made to insure that it does not equal \TUNT of the system subchannel. INIT? will then prompt for the initialization of each individual subchannel in that group-matching on the current scan unit # and setting the target unit # to the TARGET UNIT? response just entered.

INIT? scans the ICD/MAC version of \TMT searching for a destination unit matching that in TEMP3 (parameter in A-Register on entry to INIT?). The system subchannel (\DSUB) is skipped, as are all subchannels with their number of tracks equal to 0, those already marked for initialization. If the unit specified in the entry matches TEMP3, then that subchannel is prompted. If a /E is entered, then INIT? returns to the caller and no more subchannels in this group are prompted for initialization. On a YES response the subchannel's entry in \TMT is updated as follows:

\TMT+(subch")*5+2: has its unit field replaced with the target unit specified in TEMP4 (B-Register on entry to INIT?).

\TMT+(subch#)*5+3: has bit 15 set to indicate initialization.

AUTO BOOT OPTION

SWTCH requires 6 conditions for autoboot, and tests for these by using the information obtained from the Header Records.

```
Destination Disc Channel (\DCH) = Target Channel (\T32C or \TDLU(7900))
Destination addr/unit/subchannel= Target addr/unit/subchannel
Destination TBG channel         = Host TBG channel
Destination TTY channel         = Host TTY channel
Destination Priv. Int. Chan     = Host Priv. Int. Channel(if they exist)
Destination Disc type (ICD/MAC) = Target Disc type
```

OVERLAY CONDITIONS

If the host addr/unit/subch is the same as the target SWTCH assumes that the host's LU2 will be overlaid and warns the user, "DISC IN HOST SYSTEM DRIVE WILL BE OVERLAID". A flag (OVLAY=1) is set for later reference. Note that this is a very general test. No attempt is made to determine whether the new system will overlay the area on the disc which contains the FMP file. The message should be adequate to alert the user that disc is being overlaid. It is his responsibility to make sure that the data has been saved, and that the FMP file containing the new system will not be written over during the SWTCH process.

FILE PURGE

At this point, SWTCH has gathered all the needed information from the user, so the user is given one more opportunity to abort the process (if in the interactive mode). If the user is saving files, SWTCH must now purge all files which are to be overlaid. The first full track read is done at BFULL which uses the track size buffer. All code up to BFULL will be overlaid by data at this point. SWTCH now calls PURGT to purge all overlaid files on LU2 and lists them for the user. The same is done for type 6 files (if requested).

SYSTEM INSTALLATION

At the XFER label, SWTCH prints "INSTALLING SUBCHANNEL XX", and prepares to install LU2 for the new system. A full track is now read from the FMP file containing the system, and the checksum (the first 256 words) of the system is recomputed. This checksum is then compared with the previous value saved in CKSUM. This check will determine whether the operator has physically removed the disc media containing the system file by mistake. SWTCH now installs the system a track at a time by reading from the FMP file at RDISK and writing it out at WDISK, until the entire file has been written to the disc. The remainder of LU2 is then initialized without the write protect bit set.

SUBCHANNEL INITIALIZATION

ILOOP is called for each subchannel that the user asked to initialize. For ICD/MAC discs, SPINT cleans up the spare pool by removing any "spare" or "protected" bits from the preamble, and flags all defective spares with the "defective" bit so that they won't be used. The data tracks of the subchannel are then initialized by writing zeroes and setting the address of defective tracks to reference spare tracks in the spare pool. Bad tracks are reported and spared to tracks in the pool as long as spares are available. Note that the 7900 disc controller is not capable of sparing, so defective tracks are merely reported to the user.

AUTO BOOT UP

SWTCH is now ready to boot up the newly installed system, if requested. The autoboot section in SWTCH is designed to imitate the function of the disc ROM loader as closely as possible. The ROM is responsible for loading the boot extension (128 to 256 words) into memory at location 2011 (octal).

For 7900 autoboots, the actual bootup is done at NVRFY in SWSG1. Note that the S-register and DMA control word 1 are set up as the ROM loader would set them.

For ICD/MAC discs, the bootup is done in the main program at BOT32. The boot extension is read back off the new system subchannel into SWTCH's local buffer. Interrupts are then turned off so that the privileged code can be executed. The S-register and DMA are set up for the boot extension at BOT32+6. The boot extension is then moved from the local SWTCH buffer to location 2011 in physical memory. The base page fence is cleared, mapping is turned off, and SWTCH jumps through location 2055 as does the loader ROM.

TERMINATION

If autoboot is not enabled, and the overlay conditions were met (OVLAY=1), SWTCH warns the user that the new system must be booted. The user is now given the opportunity to remove the temporary target cartridge (if used), and replace the host cartridge before control of the disc is given back to the host system. At this point, the EQT is unlocked (ICD/MAC switches), and all the host system I/O that was held off by the EQT lock is allowed to resume where it left off.

MAJOR SUBROUTINES

VFYSY

Uses DISKD to read from the target subchannel in order to verify or negate the existence of a cartridge director CD and file directory FD, which are the necessary conditions to assure that the target file structure can in fact be saved.

An RTE-IVA (or previous RTE) A CD is verified if:

a) there exists an lu 2 entry whose last FMP track, D.LT, is >0

b) word 124 = 0

c) for all other non-zero entries, words 1-3 are > 0.

An RTE-IVB (or later) CD is assumed if:

a) the LU word is negative of > 63

b) words 1-3 of any entry are < 0

A FD is verified if:

a) word 0 and word 8 are < 0

b) words 1-7 and 9-15 are >=0

c) word 6 >= word 5

d) word 7 - (word 8 + 1) = D.LT obtained from CD

e) D.LT = last logical track according to system subchannel defined at generation time.

If a system cannot be verified to exist, in which case the file structure cannot be saved, the user is given the option to abort SWTCH or continue. If continuation is desired, the flags are set to disallow saving the files and purging type 6 files.

VT0S0

VT0S0 verifies the existance of a track 0 sector 0 boot extension in record 3 (and 4 for ICD Systems) of the Type 1 system file. This is accomplished by computing a 5-word checksum and comparing this with a pre-computed "magic number" which is constant for the 7900, ICD and MAC boot extensions. The checksum is computed by XORing words 109-113 for MAC and 7900 Boot, and words 191-195 for ICD Boot.

Return to:(P+1) Not a valid boot extension
(P+2) Valid boot extension

OK? - Asks the user "OK TO PROCEED?"; calls YE?NO to decipher the answer, and transfers to \XOUT on a N response, doing a simple return on a Y response.

YE?NO - reads the operator's response, looking for a Y, N, or /E in the first 2 characters (first word).

returns to (P+1) if invalid response
(P+2) if /E
(P+3) if N
(P+4) if Y

TARGT - reads a response and reissues the EXEC call in case of time-out at the console.

\DFLT - checks for a single space followed by a carriage return as an operator response - this indicates the default value for whatever value SWTCH was trying to obtain, or else a signal to SWTCH that it's to proceed (as after the "NOW IS THE TIME TO INSERT CORRECT CARTRIDGE ...").

returns to: (P+1) if not a single space
(P+2) if one single space

PARMP & SCAN

PARMP is the parameter parsing routine used for the SWTCH turn-on parameters or when the filename is entered in the interactive SWTCH mode. This routine is a modified version of the NAMR routine present in the FORTRAN library. A GETST call will return the string:

FLNAME: P1: P2, P3, P4, P5, P6, P7, P8

which corresponds to the parameters in:

RU,SWTCH,namr,scB/disc LU,addr/unit/pltr,autoboot,filesave,type-6,init

PARMP takes the string and produces a twelve word parameter buffer:

```
WORD 1 = 0 IF TYPE = 0 (SEE BELOW)
       = 16 BIT TWO'S COMPLEMENT NUMBER IF TYPE = 1
       = CHARS 1 & 2 IF TYPE = 3
WORD 2 = 0 IF TYPE = 0 OR 1, CHARS 2 & 3 OR TRAILING SPACE(S)
         IF 3
WORD 3 = SAME AS WORD 2 (TYPE 3 PARAM IS LEFT-JUSTIFIED)
WORD 4 = PARAMETER TYPE OF ALL 8 PARAMETERS IN 2 BIT PAIRS
           0 = NULL PARAMETER
           1 = INTEGER NUMERIC PARAMETER
           2 = NOT IMPLEMENTED YET (FMGR?)
           3 = LEFT JUSTIFIED 6 ASCII CHARACTER PARAMETER
      BITS FOR P1 : P2 : P3 : P4 : P5 : P6  : P7  : P8
                 0,1 2,3  4,5  6,7  8,9  10,11 12,13 14,15
WORD 5 = 1ST SUB-PARAMETER AND HAS CHARACTERISTICS OF WORD 1.
WORD 6 = 2ND SUB-PARAMETER DELIMITED BY COLONS AS IN WORD 5.
WORD 7 = 3RD SUB-PARAMETER (MAY BE 0, NUMBER OR 2 CHARS).
WORD 8 = 4TH SUB-PARAMETER (MAY BE 0, NUMBER OR 2 CHARS).
WORD 9 = 5TH SUB-PARAMETER (MAY BE 0, NUMBER OR 2 CHARS).
WORD 10= 6TH SUB-PARAMETER (MAY BE 0, NUMBER OR 2 CHARS).
WORD 11= 7TH SUB-PARAMETER (MAY BE 0, NUMBER OR 2 CHARS).
WORD 12= 8TH SUB-PARAMETER (MAY BE 0, NUMBER OR 2 CHARS).
```

The namr parameter specification is not included in the above buffer. After scanning for the first parameter (namr), its type is simply stored in FILEW, so a zero value would indicate the absence of the parameter.

SCAN is called by PARMP to fetch the next parameter. The end of a parameter is delimited by either a comma, colon, or end-of-buffer. Colons indicate parameters P1 and P2 only and are ignored if they occur after the first comma. The first comma always marks the beginning of parameter P3.

PYN

PYN is called after PARMP has placed the parsed parameters into the 12-word buffer. On entry to PYN, the A-Register contains the parameter word while the B-Register contains the value of word 4 with the bit positions for the previous parameter rotated to bits 1 & 0. PYN checks only for parameters 5-8, whose values should be Y or N. If the next parameter position in word 4 indicates that the parameter was not specified, or if it indicates that it was not ASCII, or if the ASCII value wasn't Y or N, then PYN returns to (P+1), otherwise PYN returns to (P+2) with A-register = 0 for a N response, =1 for a Y response. Note that the B-register, containing word 4 is maintained between successive calls to PYN.

PURGT

PURGT purges a file by setting word 0 of its file directory entry to -1 and size word 6 to 0, and displays that file name on the console. PURGT is called once it has already been determined that the entry pointed to by BPTR is to be purged.

When a file overlaid by the new system is purged (indicated by CURCH not equal to 0), then that file name is entered in the list of files whose extent file directory entries (if any) are to be purged. This list is built from the top of core (or partition) downward having three words per entry with the values filled in upward. PENT is the word 0 address of the next entry and has an initial value of LWAM-3.

UPDAT

This routine updates the file directory pointers when looking for entries of either overlaid files or Type 6 files that are to be purged.

UPDAT depends on these temporary variables being set:

```
TEMP4 = # of directory tracks (from D.#).
TCNT  = # entries left to search on current disc track.
CURCH = -1 to purge extents of overlaid files
      = 0 when purging type 6 files which have no extents.
BPTR  = buffer address of next directory entry.
REWRT = 0 current directory track has not been changed
      > 0 at least one entry of the current directory
          track has been changed, so track must be rewritten.
#PF   = number of overlaid files (not Type 6's) purged so far.
```

UPDAT has three returns:

(P+1) continue search on same directory track with B-Register containing the buffer address of next entry (The caller of UPDAT then stores it in BPTR.)

(P+2) the entire file directory has been searched and all updated tracks rewritten in the process; continue with next SWTCH step.

(P+3) start searching a new directory track with B-Register containing buffer address of next (actually first) entry to search, and A-Register containing the number of entries left on this track to search (TCNT is re-initialized with this value).

At label UPDTT, REWRT is checked to determine if the current track is to be rewritten to disc (because something was changed) before a new directory track is read in. If this was not the last directory track, then the next one is read in and UPDAT returns to (P+3).

When purging overlaid files, before (possibly) rewriting the current directory track and moving on to the next, the directory is rescanned in order to purge any extents belonging to the overlaid files. For each entry in the PENT list (number = #PF) the entire directory is scanned for a matching file name. When a match is found the extent entry is purged and REWRT is set (if not already done so). When done scanning for each PENT extent on that directory track, we drop through to UPDTT - where we proceed as usual until we're done with the next directory track.

\BLIN - sends a blank line to the console using \DSPL.

\DSPL - sends the message starting at the address in the B-Register, of length A-Register words to the lu of the operator console. Note the call to LOOP, which loops until the EQT5 status word indicates that the device is no longer busy, whereupon it returns.

LOOP - is not needed for all \DSPL calls. The need arose in those situations where an I/O call via SWTCH's own driver was made shortly after \DSPL sent a message. Because DISKD turns off the interrupt system while doing its thing the messages came out in chunks (often a character at a time!) rather than one continuous stream.

SWSG1 ROUTINES FOR 7900 DISCS

\STDO,

This routine configures the I/O instructions in the 7900 driver, DISK0, to the target channel \TDLU. The data channel instructions specified by I/OTB are configured to \TDLU and the command channel instructions specified by I/OTC to \TDLU+1.

DISKD

Calls the correct disc interface segment, either \DSK0 for a 7900 target system or \DSK5 for an ICD/MAC target system.

\GDMA is a 7900 routine that allocates DMA channel 2 for the 7900 driver. It is called once by the main program, and DMA channel 2 remains allocated for the duration of the SWTCH process. \GDMA checks if channel 2 is available (with the interrupt system on) and loops on this check until the DMA channel appears to be available. \GDMA then turns off interrupts and checks again to make sure that the system did not allocate it to someone else (in case SWTCH was interrupted). If this check succeeds, SWTCH stores 777 octal at INTBA+1 to tell the system that the channel is in use, and turns the interrupt system back on.

\RDMA is the 7900 routine which releases DMA channel 2 by storing 0 at INTBA+1 if and only if SWTCH's ownership key of 777 octal was stored there.

\DSK0 - is the 7900 disc driver routine.

\DSK0 references the current target disc subchannel specified by \TSUB, and defined by \DFTR (first physical track) and \DNTR (number of tracks). The subchannel (i.e. platter) further determines the disc unit UN#IT and head H#AD. These values are combined with the logical track and sector passed in \TRAK and \SECT, respectively, to form an absolute track and sector address. Immediately before initiating the transfer, \DSK0 turns off the interrupt system via a $LIBR call. INTON turns the interrupts back on before \DSK0 returns.

The variable \INIT is set differently, depending on whether SWTCH is:

a) doing a normal read or write.

b) doing a write initialize.

c) doing a write initialize, with write protect on (i.e., operating system code).

d) flagging a track defective from the error recovery routine.

This setting helps \DSK0 to figure out what it is to do in error situations because it knows approximately where it was in the SWTCH transfer. Check each use of \INIT because it depends on the situation or error encountered.

ERRCH deciphers the error status in the A-Register and branches to the appropriate error recovery code. For write protect and not ready errors a message is sent and a halt is done, waiting for a restart of the disc I/O operation at RTRY. For defective cylinders, DISBM determines whether a bad track can even be flagged defective. For SEEK errors 10 tries are made before further error recovery is attempted.

For irrecoverable errors, the driver jumps to SWTCH's abort exit \XOUT, after issuing the proper message containing the guilty track and subchannel #'s.

\BOOT is checked before a normal return to see if we just read in the TOSO boot extension and are therefore ready to jump into it. In addition, we must clear the base page fence register by doing a DJP to disable the user map. In order for the configurator program to access the disc when starting up the new system, it retrieves the disc select code from bits 11-6 of the switch register. Since this is done by the ROM loader during normal boot-up SWTCH sets this value for auto-boot. Also note that SWTCH set up DMA control word 1 with the select code of the 7900 disc. This is necessary because the boot extension uses DMA channel 1, and assumes it is set by the ROM.

INIER (SWSG1) - is entered after 10 tries have been made to initialize a 7900 disc track. This routine is not branched to when a disc error occurs and the write protect bit in \INIT is on because that indicates that a system track is being written - and defective tracks are not allowed there.

If the seek check or end-of-cylinder bits were set in the status word, then INIER assumes an invalid subchannel definition, sends the message, and aborts SWTCH through \XOUT.

\DSK0 is jumped to in order to flag the track defective. On return, INIER reports the defective track, enters it in the bad track table (FLGTR) if \LU2 indicates the system subchannel, and does a JMP \DSK0,I.

SWSG2 ROUTINES

OVERVIEW

SWSG2 is the segment which interfaces to all ICD/MAC discs (includes 7905/06(H)/10H/20(H)/25(H)). It performs primitive disc operations (e.g. SEEK,FILEMASK,READ,WRITE), by calling subroutines in the Disc Utility Library, $DSCLB. This library, in turn, calls the appropriate driver (DVA32 or DVR32) with a special EXEC call. The driver then operates in a passive mode, where the command buffer is sent to the disc controller without any error checking. \DSK5 - is the primary subroutine in SWSG2 and handles all I/O calls to ICD/MAC discs for SWTCH. It has 4 modes of operation for reads and writes, where the \MODE parameter selects which type of operation is to be performed.

\MODE=1 - for standard reads/writes to the disc, in order to update the directory track on the target disc, for example.

\MODE=2 - for initializing and writing system tracks to the disc with the protect bit set in the preamble, doing sparing as needed.

\MODE=3 - for initializing (and writing zeroes to) non-system tracks. The "protect bit" is not set, but sparing is done for all tracks found to be defective. (MODE 3 is also used for the 7910H system tracks since there is no format switch on the 7910H, and thus tracks may not be "protected", if they are to be written later.)

\MODE=4 - Initializes and writes zeroes to tracks in the spare pool. Clears "spare" or "protected" bits and sets the "defective" bit if a bad spare track is found. \DSK5 is called in \MODE 4 for all tracks in the spare pool before initializing or writing any of the tracks in the data portion of the subchannnel.

TBL02 is the status word jump table. The value of status word 1 from the disc controller determines which processing block in the table is branched to. ENDBR is the common return point for all processing blocks in the branch table. The processing blocks determine the proper action to take based on the current state variables \MODE and PHASE.

Note that a single request to \DSK5 causes the code section from DSGO to ENDBR to be executed multiple times until the value of \RET indicates that the operation is complete and we should return from \DSK5. Each iteration through this loop will update PHASE to indicate the current state of the operation. PHASE may have the following values:

PHASE =1 - Read Full Sector to get the track's status from preamble.
=2 - Write Initialize to the track, rewriting track & sector addresses.
=3 - Read Full Sector to get a spare's status.
=4 - Write Initialize to a spare to point it at the defective track.
=5 - Write Initialize to a data track to point it to a spare.
=6 - Write Initialize to a spare, flagging it defective.

BRANCH TABLE PROCESSING BLOCKS

ENDOK is branched to when the last command to the disc controller succeeded as expected (stat code=0). The next phase is entered.

FAULT catches all unexpected error codes (1, 2, 3, 4, 5, 6, 12, 13, 15, 24, 25). It sends a "DEFECTIVE CYLINDER ..." message and sets \RET to abort the current \DSK5 request.

RECAL is branched to on a cylinder miscompare (stat code=7). It issues a RECALIBRATE command to the disc and allows the operation to be retried up to 10 times before aborting the current request. Since the 7910H does not support RECALIBRATE, a seek to cylinder 0 is done instead.

DSKER is branched to on stat codes 10 (uncorrectable data error), 16 (overrun) and 17 (possibly correctable data error). The operation is retried up to 10 times before aborting the current request.

DEFTR is branched to when a track is found to be defective (stat code =21) or DSKER has retried the operation 10 times without success.

EOCYL is branched to on stat codes 11 (Head/sector miscompare) and 14 (End of cylinder). The "INVALID DISC SPECIFICATIONS.." message is sent, and \RET is set to abort the current operation.

ILSPR is branched to on stat code 20 (Illegal Access to spare). If a standard read/write (\MODE=1), the operation is aborted, because a direct reference to a spare track is not permissible. When initializing tracks, this status may occur when seeking to a spare track to clear the "spare" and "protected" bits, and is normal for MODE 2 or 3.

ST2ER is branched to on stat codes 22 (Access not ready), 23 (status 2 error), and 26 (Illegal write). The following conditions are checked by ST2ER:

1. Seek check-send "INVALID DISC SPECIFICATIONS.."
2. Disc not ready-send "READY DISC.." message and retry the operation.
3. Format switch off-send "TURN ON FORMAT SWITCH..", and retry.
4. Protect switch on-send "TURN OFF DISC PROTECT..", and retry.
5. Unknown status 2 error-retry the operation up to 10 times.

UWAIT is invoked when the disc is not available (multi-CPU environment). The operation is retried 10 times before "INVALID DISC SPECIFICATIONS.." is sent.

MAJOR SWSG2 SUBROUTINES

XFER provides the interface between SWSG2 and the Disc Utility Library, $DSCLB. \DSK5 uses XFER to carry out all the primitive disc operations by setting up the action parameter, \ACTN and calling XFER. Each bit in \ACTN is associated with a disc primitive routine, so XFER calls the corresponding library routine when a bit is set.

The \ACTN word is interpreted as follows:

BIT	MEANING
0	CALL XSEEK TO ISSUE A SEEK COMMAND TO CONTROLLER.
1	CALL XADRC TO ISSUE AN ADDRESS RECORD COMMAND TO CONTROLLER.
2	CALL XFMSK TO ISSUE A FILE MASK COMMAND TO CONTROLLER.
3	CALL XDRED TO ISSUE A READ COMMAND TO CONTROLLER.
4	CALL XRDFS TO ISSUE A READ FULL SECTOR COMMAND TO CONTROLLER.
5	CALL STFIX TO ADJUST THE STAT CODE AFTER A READ FULL SECTOR.
6	CALL XDWRT TO ISSUE A WRITE COMMAND TO CONTROLLER.
7	CALL XINIT TO ISSUE A WRITE INITIALIZE COMMAND TO CONTROLLER.
8	CALL XVRFY TO ISSUE A VERIFY COMMAND TO CONTROLLER.
9	CALL XEND TO ISSUE AN END COMMAND TO CONTROLLER.
10	UNUSED
11	UNUSED
12	UNUSED
13	UNUSED
14	UNUSED
15	UNUSED

XFER begins with bit 0 and continues through bit 15, executing every primitive whose action bit is set. For example an \ACTN value of 1017 (octal) would be used to perform a standard disc read operation. In this case, the disc controller would receive the SEEK,ADDRESS RECORD, FILE MASK, READ, and END commands (in that order). XFER checks status after each primitive is sent to the controller, and immediately returns if an abnormal status is detected, so that the appropriate action can be taken.

CKST1 analyzes the disc status returned from the controller. It distinguishes a power fail/timeout condition from an abnormal status condition and returns as follows:

```
Return to (P+1)-Power fail or timeout-restart operation.
          (P+2)-Abnormal status-return with the stat code.
          (P+3)-Normal status=0-return and continue.
```

NIXSP finds the next available spare track from the spare pool for the current subchannel. The physical (Cylinder and Head) address of the spare is computed and set in CYL# and HEAD#.

```
Return to (P+1)-if out of spares for this subchannel.
          (P+2)-if next spare found.
```

FMTR? is called when SWTCH is attempting to save files on LU2, and \DSK5 is in MODE 4 cleaning up the spare pool. Normally, all tracks in the spare pool would have their "spare" and "protected" bits removed and be made available for use. When files are being saved, however, FMTR? must check to see if the current track is a spare being used by an FMP file in the save area. This is determined by doing a READ FULL SECTOR to get the address of the data track which is currently using the spare. If this address is in the LU2 FMP area being preserved, the spare track is not reclaimed for use, but left intact with all the original data, and the "spare" bit set. FMTR? determines whether the spare is currently being used by an FMP file by examining the following conditions. If condition 1 AND either condition 2,3, or 4 is met, the spare track is not reclaimed for the spare pool, but left intact.

1. The defective track's head# is within the head# range of the system subchannnel (\DSHD to \DSHD+\DNSU).

2. First FMP cylinder < defective cylinder < last FMP cylinder.

3. Defective cylinder = First FMP cylinder AND defective track head# >= head# of first FMP track.

4. Defective cylinder = First FMP cylinder AND defective track head# <= head# of last FMP track.

\SETD determines the subchannel specification for the current subchannel (\DSUB), and sets the appropriate values in \D#ST, \D#WT, \DFTR, \DUNT, \DHSD, \DNSU, \DNTR, and \DNSP.

DADTR translates a logical track number on the current subchannel into a physical disc address (Cylinder and Head number) and stores the result in CYL# and HEAD#. The translation is based on the current subchannel definition set by \SETD.

CVLOG is the inverse of DADTR and translates a physical disc address into a logical track number using the subchannel definition of the current subchannel.

RPORT is the bad track and spare track reporting routine. It sends the bad track header message for the first bad track on each subchannel, and reports BAD TRACK, BAD SPARE, or, SPARED TO and the logical and physical address of the track concerned.

ESUB sets the current subchannel number in error messages.

Chapter 12

RTE-IV ASSEMBLER CHANGES

ASSEMBLER

There are two major changes made in the RTE-Assembler going from RTE-III to RTE-IV. A pseudo opcode EMA was added to the assembler. This opcode allows the user to declare an external memory array. Next, I/O from and to LS/LG areas is replaced by I/O from and to file manager files using the compiler library. List output can also be directed to a file manager file. The cross-reference generator XREF is also changed to do file I/O.

EMA PSEUDO OPCODE

The EMA instruction is defined as:

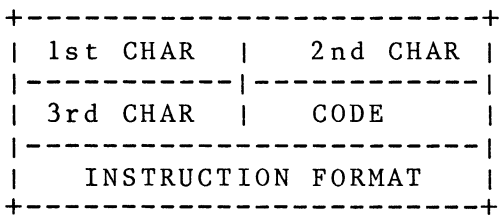
Label EMA size,MSEG Size.

EMA is assigned 5 as an opcode identifier. In pass 1 of the assembler, the processing for the EMA instruction is done in the EMP processor. Here EMCNT flag is checked to determine if it is 0. If this flag is non-zero, then another EMA instruction was encountered previously. Since only one EMA instruction per program is allowed, an 'IL', illegal instruction error message is printed for the second 'EMA' instruction encountered. The CHOP routine is called next to evaluate the two operand values. EMP checks to make sure the two values returned by CHOP are absolute, a 'UN' - undefined symbol error is printed otherwise. The EMA size must be positive and less than 1024. The MSEG size must be positive, less than EMA size (unless EMA size is 0) and less than 32. If any of the above conditions are not met, 'M'- illegal operand error is printed. An EMA instruction must have a label. If a label is not present an 'LB', label not present error message is printed. The symbol type assigned to the EMA label is 4 - the same as that for an external symbol. The undefined bit, bit 15 of word 1 of the symbol table entry is set to distinguish between an external symbol and an EMA label whenever necessary. The label for an EQU to EMA label is given the symbol type 5 with the undefined bit set. The starting address of the external memory array is the beginning of the first page in free available memory and can be defined only at load time.

The assembler creates a special 7-word EMA binary record with the record identification number as 6. Refer to the RTE manual for a description of the EMA binary record. This binary record is set up and output at the beginning of pass 2, just after all the EXT binary records are output. The relocation indicators in the DBL records for instructions using EMA label are: 4 for instructions using EMA label and 5 for EMA label with offset. The assembler does not make a distinction between EXT and EMA symbols while processing the memory reference instructions.

OPCODE TABLE FORMAT

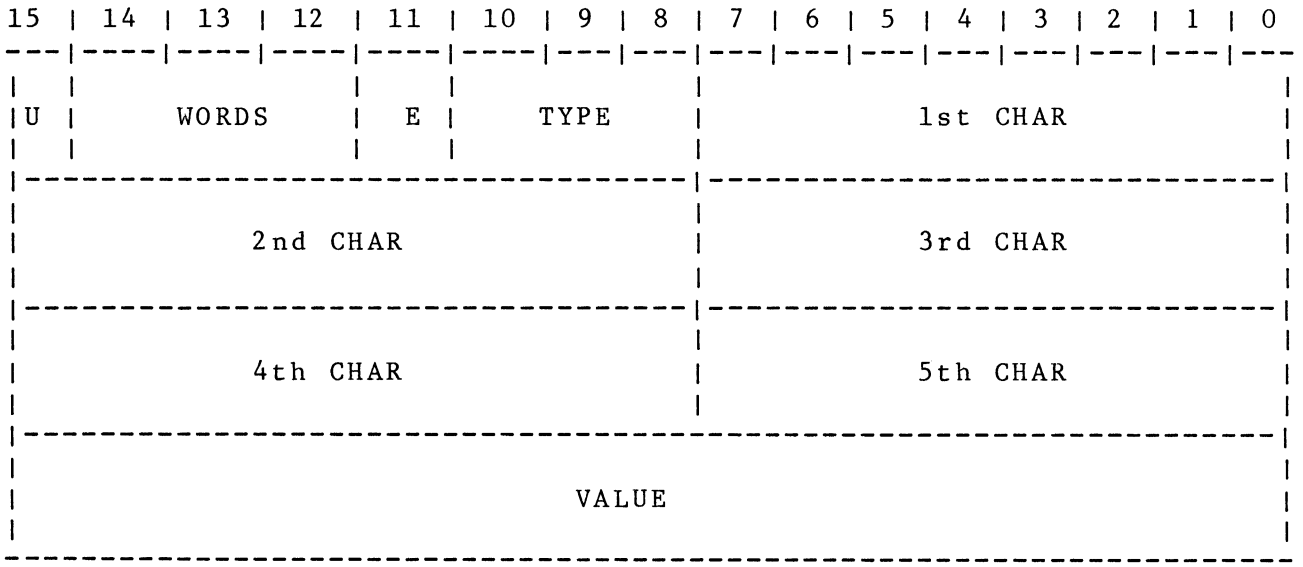
Each entry has the following format:



Code is one of 64 (8-bit) identifiers used to process the OPCODE during assembly. Instruction Format is the instruction format or the location of the processor for the opcode.

SYMBOL TABLE contains:

a. Labels
b. External symbols
c. COMMON names



U = Undefined bit:
- 0 = Symbol defined
- 1 = Symbol undefined or EMA (symbol type 4 or 5)

WORDS = Number of words in entry (2-4)

E = Entry point bit:
- 1 = Symbol has been declared an entry point
- 0 = Not an entry point

TYPE = Symbol Type:
- 0 = Absolute
- 1 = Relocatable
- 2 = Base Page Relocatable
- 3 = COMMON name
- 4 = External or EMA
- 5 = Label equated to External symbol or EMA label
- 6 = Code replacement (ENT)
- 7 = Literal

VALUE = Symbol value:

a. Absolute Value (type 0)
   Value relative to relocation base for types 1, 2, 3
   External or EMA symbol ordinal (types 4, 5)

b. The value will contain the location of the literal relative to the end of the main program at object time.

Chapter 13

SESSION MONITOR ACCOUNT PROGRAM

INTRODUCTION

The Account Program Technical Specifications are intended to serve as an aid to the programmer when modifying the RTE Session Monitor account program. This document should be used in conjunction with the account program source listings, as it does not attempt to provide the level of detail given by the code itself. It is assumed that the reader is familiar with the operation of the account program as described in the Session Monitor System Manager Reference Manual.

General Overview

The Session Monitor Account Program is provided for the System Manager's use in initializing and maintaining the account file. It allows the System Manager to create accounts for new users or groups of users, to remove existing accounts, to modify specific user or group attributes, or to perform particular account file utility functions.

Operation

The account program (ACCTS) is a background, segmented program. ACCTS accepts input either interactively or from a disc file. These command inputs instruct ACCTS to perform specific operations on the account file. The account file itself is a type 1 FMP file.

ACCOUNT FILE STRUCTURE

Figures 13-1 through 13-9 detail the structure of the account file. The first record is the account file header, which contains specific global Session Monitor information (see Figure 13-2).

Account File Header

Location pointers - the first six words in the headers are pointers to the beginning of each part of the account file.

WORD	
1.	Active Session Table
2.	Configuration Table
3.	Disc Pool
4.	User/group ID Map
5.	Account Directory
6.	Account Entries

System Message File - the name (namr) of the message file which is first listed to the user's terminal when the user logs on. The file name may be changed with the ALTER,ACCT command.

Prompt String - the prompt is a 0-20 character string which is output to a terminal when any key is entered on session terminal which is not currently active. The default string is PLEASE LOG-ON: the first word is number of characters in the string.

Lowest Private ID used - the lowest number (1-4095) which has been assigned as a private ID by the Session Monitor. This number is always greater than the highest group ID and is initially set to 4096.

Highest Group ID Used - the highest number (1-4095) which has been assigned as a group ID by the Session Monitor. This number is initially set to 1 and is always less than the lowest private ID used.

Resource Number - Used for cooperative updating of the account file by ACCTS and the log-on and log-off routines.

LU of MSG Files - this is the logical unit of the disc on which all message files will be stored.

Memory Allocation - this is the number of words of SAM to be allocated to session monitor. If negative it is minus the number words to allocate and the number is computed at boot up based on the session limit.

Session Limit - the maximum number of users allowed to be logged-on at any given time. It's current value is checked before LOGON allows a user to log on.

Active Sessions - the current number of active sessions.

Shut Down Flag - this is a flag to tell ACCTS whether the Session system is shut down or the Accounts file is to be purged or an individual account needs to be purged.

1	Account entry to be purged
0	Session system active
-1	Session shut down
-2	Accounts file to be purged
-3	Session shut down momentarily
-4	Session shut down and session memory released

Session Limit - this is a copy of session limit after a shut down. When start up is entered, this is copied back into the session limit above.

Class Number - this is a class number used by accounts to send messages via the TELL command.

Length of Configuration Table - the length of the configuration table is kept here for easy expansion of accounts file.

Resource Number - this resource number is used so that various copies of ACCTS can update file.

Disc Pool Length - this is the length actually returned by $SALC. It is kept so that the entire block can be returned.

Active Session Table

The second record (Figure 13-3) contains 4-word Active Session Blocks, one per active session (i.e., one per user currently logged-on).

Logical Unit - the logical unit of the terminal at which the user is logged on.

Log-on Time - a doubleword value indicating the time at which the user logged on. This value is used at log-off to update the connect time clocks.

Directory Entry # - the directory entry # (entry offset) of the directory entry for this account.

Configuration Table

The Configuration Table follows the Active Session Table, which describes the default logical units to be used for specific device logical units. Each station logical unit 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. The default logical unit associated with the station itself is always 1. At log on, these default values are written into the user's Session Control Block (SCB), unless overridden by entries in this particular user's Session Switch Table (SST).

The first word in the Configuration Table is the length word (the number of devices to which default logical units will be assigned and associated with this station, plus 1). Following this word is the first station logical unit with which default LU's will be associated. Next are the one word entries for each system logical unit and its associated session (default) logical unit. Following is the logical unit of the next station with which default logical units will be associated. The entire table is terminated with a zero.

Example: Associated with station LU30 are to be the left and right cartridge tape units (CTU's) which can always be accessed by a session user at this terminal as LU4 and LU5, respectively. Associated with station LU40 are to be its left and right CTU's which are also to be accessed by session users at this sttion as LU4 and LU5. Also associated with station LU40 is to be a dedicated line printer (actually LU57), to be accessed by session users at this station as LU6. The Configuration Table would look as follows:

3		length of entry
30		station LU
34	4	default left CTU (LU34) to LU4
35	5	default right CTU (LU35) to LU5
4		length of entry
40		station LU
44	4	default left CTU (LU44) to LU4
45	5	default right CTU (LU45) to LU5
57	6	default printer (LU57) to LU6
0		end of table

Disc Allocation Pool

The Disc Allocation Pool is a list of disc logical units (LU's) to be allocated to session and non-session users. The record contains one disc LU per word, with zeroes filling out the unused words of the 128-word record. Figure 13-5 shows the structure of this record.

Non-zero entries in the disc allocation pool are written to system available memory during initialization of the Session Monitor. The first entry in the memory-resident disc pool is pointed to by $DSCS. Figure 13-1 indicates the end of this disc pool.

User/Group ID Map

The ID Map is 256 words long containing a map of the 4096 User or Group ID numbers. Word 1 contains the bits identifying ID numbers 1-16, word 2 is used for ID numbers 17-32, etc. A bit set indicates that the associated ID number is currently assigned to an account. Thus word 1, bit 15, if set, indicates that ID number 16 has been assigned, and word 2, bit 0, if set, indicates that ID number 17 has already been assigned. Words 6 and 7 in the account file header can be used to determine if the ID is a user ID or a group ID. The user/group ID map is used when allocating ID numbers (at account setup) and when releasing ID numbers (at account purge).

Account file Directory

Each directory entry is a 16-word record containing the ASCII name of the user or group and the relative address of the record containing this user's or this group's account entry (Figure 13-7). The length of the directory is determined by the response, at account setup, to the prompts:

```
NUMBER OF USER ACCOUNTS?
NUMBER OF GROUP ACCOUNTS?
```

The directory size is computed assuming 20% of User accounts will require more than 64 words. The final result is rounded up to the next block.

Word 1 of each directory entry indicates whether the directory entry is for a user account or a group account. If word 1 is -1, this directory entry is free (resulting from the purging of a user or group account). A zero in word 1 indicates the end of the directory. If word 1 is -2, this indicates that a block is reserved for the second half of a user account.

User Account Entries

Following the directory in the Account File are the user and group account entries, interspersed, with one record per account entry. A user account entry is distinguished from a group account entry by the first word of the record. If negative or zero the record contains a user account and the value is the negative of the number of characters in the user's password. If the first word of the record is positive, the record contains a group account entry and the value is the group ID.

Figure 13-8 shows the structure of user account entries. The following is a brief description of the fields in a user account entry:

Contents	Comments
# CHARS IN PASSWORD	(If bit 15 is set, SST extends into second block.)
PASSWORD	0-10 ASCII CHARACTERS, CANNOT BE "_".
USER HELLO FILE	Namr of file to be transferred to upon log-on.
USER MESSAGE FILE	Namr of file to which mail is sent. Generated by the SM command processor.
CAPABILITY	Integer, 1-63 (63=most capable).
LAST LOG-OFF TIME	2-word entry.
CUMULATIVE TIME	Cumulative connect time in minutes (1-word entry).
CPU USAGE	Cumulative CPU usage in seconds (1-word entry).
USER ID	Integer, 1-4095.
GROUP ID	Integer, 1-4095.
DISC LIMIT	Integer.
GROUP SST & SPARE	Length of group SST in upper byte and number of spare SST entries in lower byte.
SST LENGTH	SST length in words, including spares. If positive, group SST is also to be mapped to user SCB.
SST ENTRIES	Session LU (less one) in bits 0-7, System LU (less one) in bits 8-15. (If SST entends into second block word 64 contains record number of extensions.)

Group Account Entries

Figure 13-9 shows the structure of group account entries. A few comments on the fields in a group account entry:

Contents	Comments
GROUP ID	Integer, 1-4095.
CUMULATIVE TIME	2-word entry.
CUMULATIVE CPU USAGE	2-word entry.
-GROUP SST LENGTH	Negative of SST length in words.
SST ENTRIES	Session LU (less one) in bits 0-7, System LU (less one) in bits 8-15.

MEMORY COMMUNICATION

Several variables are stored in memory for fast communication between modules and to prevent the loss of system resources when a program is aborted or the accounts file is purged. The following describes these variables:

1. $DSCS

	$DSCS + 1		
$DSCS	>=0	-1	-2
>0 (disc pool)	Session ready	No new sessions allowed but allow current sessions to log off.	No logging on or off allowed. Close +@CCT! and terminate.
0 no (disc pool)			
-1	Session not Initialized (boot up)		
-2	Session not initialized but RN's and Class #'s allocated.		

2. $ACFL — LU on which accounts file is located.

3. $SPCR — LU for spool control files.

4. $LMES — Location of prompt string.

5. $CES — System session console disable.

6. $LGOF,$LGON,$STH — Class numbers for mail box communication of session system.

7. $SMEM
   $SMEM+1 — Description of memory allocated to session system.

INITIALIZATION

The initialization module is entered whenever the program ACCTS is run. If the account system has already been initialized, ACCTS exits the initialization module and enters into its command processing module. If not initialized, ACCTS enters into a dialog with the user (if run interactively) or terminates if run non-interactively.

1. If ACCTS scheduled to clean up call DTACH.
2. GET RUN STRING and parameters and set up Transfer Stack.
3. If account file exists go to 16.
4. Prompt to load or initialize. If load go to 15.
5. If /A terminate.
6. Prompt for disc LU, session limit, memory allocation number of user and group accounts.
7. Create accounts file large enough to accommodate all accounts.
8. Prompt for prompt string.
9. Define configuration table.
10. Define disc pool.
11. Initialize Account Directory.
12. Prompt for MANAGER.SYS password.
13. Create Accounts
    SYS
    SUPPORT
    GENERAL
    MANAGER.SYS
    ENGINEER.SUPPORT
14. Go to 16.
15. Load accounts file.
16. If memory allocated go to 18.
17. Allocate memory and set up disc pool by calling $BALC, $RTRN, and $SALC.

18. If first run parameter =-1, go to 26.

19. If under session, go to 22.

20. Prompt and verify system manager's password.

21. Set ID to 7777B (System managers), go to 23.

22. Get Users ID and capability.

23. Set ID and capability for command processors.

24. CALL COMMAND processor.

25. If not terminated, go to 24.

26. If clean up required call ACACP to clean up.

27. Call ACTRM to terminate.

LOG-ON Interface

The following steps are performed during any log-on sequence which is attempted before the Session Monitor has been initialized.

1. If account file is not found, report error (user can then either mount the cartridge containing the account file if one exists, or run ACCTS to create one).

2. If Session Monitor is not initialized ($DSCS=0) then schedule ACCTS to perform initialization.

COMMAND PROCESSING

Once the account system has been initialized, the ACCTS program enters a command processing loop which does the following:

a. Prompts for a command NEXT?

b. Retrieves and parses the command name and parameters.

c. Verify that user has capability to execute command.

d. Transfers control to the appropriate command subroutine.

ALTER,ACCT

Sequence of Operations

1. Prompt for Session limit and validate.
2. Prompt for memory allocation.
3. Prompt for new prompt string.
4. Prompt for message file name.
5. UPDATE accounts file header.
6. Put prompt string in memory.
7. Read Disc Pool.
8. Prompt for additions to disc pool.
9. Prompt for deletions to disc pool.
10. Update disc pool in file.
11. Update disc pool in memory.
12. Prompt to ADd, MOdify or DElete, or NO change.
13. IF not "NO" change, go to 23.
14. Prompt for station LU.
15. If DElete remove entry and go to 12.
16. Search for entry and remove.
17. If Add and found, report error.
18. If modify and not found, report error.
19. Prompt for SST definition.
20. If not /E or /A, go to 19.
21. If /A, go to 14.
22. Add station to table, go to 12.
23. Post station table and return.

ALTER,GROUP
ALTER,USER

Sequence of Operations

1. Parse user.group name.
2. If ALTER,GROUP make user name = 0.
3. Save account name in IU and IG.
4. Find account using ACFDA, If not found report error and return.
5. If not MANAGER.SYS and not his group or group specified was @, report error and return.
6. If ALTER,USER, go to 12.
7. If group not GENERAL or @, prompt for new group name.
8. Prompt for SST.
9. Update group account(s) and directory.
10. Set user parameters to no change.
11. Set user and IU name to @ and go to 20.
12. If user or group = @, go to 15.
13. Prompt for new user name.
14. Prompt to use new group.
15. Prompt to use group SST.
16. Prompt for password, hello file, capability, disc cartridges.
17. Prompt SST definition and SST spares.
18. Prompt to relink to existing account, if no, go to 20.
19. Find account and verify password, if no verify, go to 18.
20. Get group account.
21. Get user first account in group.
22. If not found, go to 27.

23. Update directory.

24. Update account and merge group SST.

25. If IU is @ get next entry in group and go to 23.

26. If IG is @ get next group.

27. If found, go to 21.

28. Return.

NOTE: Change of name and linking are bypassed if either user or group are specified by @.

EXIT

Sequence of Operations

1. Close all files.

2. Terminate ACCTS.

HELP

Sequence of Operations

1. If number supplied schedule HELP with ACCT #.

2. If keyword supplied, scan HELP and dump message to list LU.

3. Else, dump entire help file, first lines only.

4. Return.

LIST,ACCT

Sequence of Operations

1. Set up list LU and check optional parameter.

2. Read account file header.

3. Read account file directory for user.group names of active sessions.

4. Read cartridge list for current state of disc pool.

5. If AC or AL, write header information and active session table.

6. If PO or AL, write disc pool status.

7. If CO or AL, write configuration table.

8. Return.

LIST,GROUP

Sequence of Operations

1. Get parameters.

2. If group name = @ go to 6.

3. Search account file for group name using FINDG; return error is not found.

4. List group account to list LU.

5. Return.

6. Search directory for next group entry; at end, go to 8.

7. List corresponding group account and go to 6.

8. Return.

LIST,USER

Sequence of Operations

1. Get parameters.

2. If KEY supplied, set KEY flag.

3. If user not equal @ then if group not equal @ call FINDU to find user.group account.

4. If @, search all user accounts.

5. If @.@ search all user and group accounts.

6. List the account(s) found and keywords if KEY flag.

7. Return.

LOAD

Sequence of Operations

1. Get parameters.
2. If ACCTS option and account file does not exist, force entire load.
3. If account file exists with accounts, issue request to verify purge.
4. If not verified, return.
5. Prompt for station table size and number of accounts.
6. Read and verify NAMR (backup file) header record.
7. Read and backup into scratch file.
8. Purge old account file.
9. Rename accounts file.
10. Return.

NEW,GROUP

Sequence of Operations

1. Get group name, SST entries and verify before writing to buffer.
2. Search account file for duplicate name.
3. Search for spare entries in account file.
4. Update highest group ID used.
5. Write buffer to record.
6. Return.

NEW,USER

Sequence of Operations

1. Prompt for user name.
2. If name is "@" or "/E", or if name has imbedded comma(s) or period(s), report invalid name and go to 1.
3. Prompt for group name.

4. If name is "@" or if name has imbedded comma(s) or period(s), report invalid name and go to 3.

5. If name is not "/E", go to 7.

6. If no group has yet been defined for user, report expecting valid group name and go to 3, else go to 12.

7. Search account file directory for group name using FINDG.

8. If not found, report group account does not exist and go to 3.

9. Search account file directory for user.group name using FINDU.

10. If found, report duplicate user.group name and go to 3.

11. Prompt for whether to use group SST and save group SST yes/no flag with group name and group ID. Go to 3.

12. Prompt for password until " " or a valid password is entered.

13. Prompt for hello file until " " or a valid NAMR is entered.

14. Prompt for capability until a valid capability is entered.

15. Prompt for disc limit until a valid limit is entered (positive integer not greater than MAXD).

16. Prompt for SST entries until "/E" is entered. Validate system LU.

17. Prompt for SST spares and validate (positive integer =< MAXST-total SST entries defined).

18. Prompt for whether to link user. If no link, go to 22.

19. Search account file for user.group name and verify password. If not found or password does not match, report error and go to 18.

20. Read existing user ID and post to buffer.

21. Go to 23.

22. Generate user ID and update lowest user ID used.

23. For each user.group account, call FREEU to get free account file space and write record, updating directory.

24. Return.

PURGE,ACCT

Sequence of Operations

1. Verify request to purge.
2. Shut down session system and purge +@CCT!!-31778:$ACFL.
3. Return.

PURGE,GROUP

Sequence of Operations

1. Get parameter.
2. If group name not equal @, go to 6.
3. Verify user's request to purge all accounts; if not, return.
4. Scan account file directory and flag all entries except SYS, SUPPORT and GENERAL with -1.
5. Return.
6. If group name = SYS, SUPPORT or GENERAL, error return.
7. Search account file directory for all user accounts with matching group name or group account with matching group name.
8. If found, verify that group is inactive and has no discs mounted, then purge account directory entry.
9. If none found, error return.
10. Return.

PURGE,USER

Sequence of Operations

1. Get parameter.
2. If user name not equal @, go to 5.
3. Verify user's request to purge all of this group's account; if not, return.
4. Go to 6.

5. If user.group = MANAGER.SYS or ENGINEER.SUPPORT, error return.

6. Search account file directory for user account(s) with matching group name.

7. If found, verify that user is inactive and has no discs mounted, then purge account directory entry.

8. If none found, error return.

9. Return.

RESET

Sequence of Operations

1. Get parameters.

2. If user or group name = @, go to 5.

3. Call FINDU to find user account and offset to CPU and CONNECT words and zero.

4. Return (error return if not found).

5. If @.@, go to 9.

6. @.group; search for all user accounts with matching group name, offset to CPU and CONNECT words and zero.

7. Find group account and zero.

8. Return.

9. For every account, if group then zero clock words, or, if user, also zero clock words.

10. Return.

TELL

Sequence of Operations

1. Get parameters.

2. If user or group name = @, go to 7.

3. Find user.group account by searching active session table, getting record # and comparing name.

4. Get terminal logical unit or error return if zero (not logged-on).
5. Send message using class I/O.
6. Return.
7. If @.@, go to 10.
8. @.group; find all active users in group by searching active session table, getting record # and comparing on group part of name.
9. Send message to each and return.
10. Search SCB list for terminal logical units and
11. Send message and return.

UNLOAD

Sequence of Operations

1. Get parameter and verify NAMR.
2. Read file +@CCT!:-31178:-2 and write NAMR removing all unused space.
3. Return.

PASSWORD

Sequence of Operations

1. Prompt for current password.
2. Verify against current users.
3. If not valid report error and terminate.
4. Else prompt for new password.
5. Update entry.
6. Return.

INTERNAL SUBROUTINES

This section describes those subroutines which are used internally by the Account Program in initialization and command processing. Included in the description of each routine are:

1. brief description
2. entry point
3. external references
4. calling sequence
5. where routine is used
6. sequence of operations

ACCRE - Creates Accounts File

Entry point: ACCRE

External References: OVRD.,.ENTR,CREAT,PURGE,$ACFL,ACOMD,$LIBR,$LIBX

Calling Sequence: CALL ACCRE (NDCB,NAME,ISIZE,IERR)

Parameters:

NDCB	Data control block for file access
NAME	File name to be created
ISIZE	Size of file in 128 word blocks
IERR	ERROR parameter returned

Used in: ACCT1, ACLOA, ACACP

Sequence of Operations

1. Set disc Lu is $ACFL.
2. Set override bit.
3. Purge file of same name on all mounted cartridges.
4. Create file on disc LU.
5. Reset override bit.
6. Return

ACOPN - Opens Accounts File

Entry point; ACOPN

External References: $SMID,OVRD.,.ENTR,ACOM1,ACOMD,OPEN,ISMVE,
$ACFL,$LIBR,$LIBX,LOCF

Calling Sequences: Call ACOPN (IERR,IDSES)

Parameters:

IERR	ERROR RETURN PARAMETER
IDSES	SESSION ID used to determine capability

Used in: ACCT1, ACLOA, ACACP, ACOPL

Sequence of Operations

1. Call ISMVE to get session ID.
2. Set override bit.
3. Get $ACFL.
4. Open accounts file on $ACFL or first disc which it is found.
5. Clear override bit.
6. If $ACFL set up return.
7. Call LOCF to find LU.
8. Set $ACFL.
9. Return.

ACWRH - Writes Syntax Messages For Help

Entry point: ACWRH

External References: ACWRL, .ENTR

Calling Sequences: CALL ACWRH (KEYWD,IERR,JERR)

Parameters:

KEYWD	First 2 characters of keyword
IERR	ERROR returned from ACWRL
JERR	ERROR if keyword not found

Used In: ACCT5

Sequence of Operations

1. Get first table entry.
2. If keyword equal to zero, go to 4.
3. If entry does not match keyword, go to 6.
4. Print syntax of command.
5. If keyword not equal to zero, print explanation.
6. Get next table entry.
7. If not end of table, go to 2.
8. Else return.

ACINT - Retrieves $DSCS and $DSCS+1

Entry Point: ACINT

External References: $DSCS, .ENTR

Calling Sequence: CALL ACINT (ISTAT,JSTAT)

Parameters:

```
     ISTAT      = $DSCS
     JSTAT      = $DSCS+1
```

Used in: ACCT1

Sequence of operation

1. Retrieve $DSCS.
2. Retrieve $DSCS+1.
3. Return.

ACPAS - Verifies MANAGER.SYS Password

Entry Point: ACPAS

External References: ACOM1,.ENTR,ACOM6,ACOM7,ACOM0,ACOMC,
ACOMD,READF,ACPSN,ACERR,ACTRM

Calling Sequence: CALL ACPAS

Used In: ACCTS

Sequence of Operations

1. Find account with ID=7777B.
2. If no password for account return.
3. Else prompt for password.
4. If password is verified, return.
5. Else print error.
6. Terminate call ACTRM.

ACPSN - Inputs and Parses Password

Entry Point: ACPSN

External References: .DIV,.ENTR,ACOM7,ACOM9,ACOMC,ACPRM,ACREI,XLUEX,
PARSN,ACERR,ACTRM

Calling Sequence: CALL ACPSN (MESS,LENGTH,JPASS,IERR)

Parameters:

MESS	IS	PROMPT
LENGTH	IS	LENGTH OF PROMPT
JPASS	IS	BUFF FOR PARSED PASSWORD
IERR	IS	RETURN ERR

Used In: ACCT1, ACPAS, ACAPA

Sequence of Operations

1. Print message.
2. Read password no echo.
3. If not read from DVR07, go to 5.
4. Back up and clear previous line.
5. Parse password.
6. If wrong format print error.
7. If no password make it default.
8. Return.

ACSDN - Shuts Down an Active Session

Entry Point: ACSDN

External References: EXEC,.ENTR,$LGOF,XLUEX,XFTTY,LUSES,
ACOMD,$CES,$LIBR,$LIBX

Calling Sequence: CALL ACSDN (LU,IERR)

Parameters:

LU	Session LU to be shut down
IERR	Returned error

Used In: ACPUA

Sequence of Operations

1. If not LU=0, go to 4.
2. Clear $CES.
3. Return.
4. Get SCD ADDRESS.
5. Send message to $LGOF to log off session.
6. Schedule LGOFF
7. Return.

ACAST - Retrieves Entry in Active Session Block

Entry Point: ACAST

External References: .ENTR, ACOM3, LDCB

Calling Sequence: CALL ACAST (JBUF)

Parameters:

JBUF	CURRENT SST

Used In: ACALU

Sequence of Operations

1. Get first change in LDCB.
2. Search for session LU match in JBUF.
3. If delete, remove it and compress.
4. If modify, change system LU.
5. If add, add entry to SST.
6. If more changes get next change and go to 2.
7. Else update number of entries.
8. Return.

ACSTR - Prints Stars

Entry Point: ACSTR

External References: .ENTR

Calling Sequence: CALL ACSTR

Used In: ACLIV, ACLIA

Sequence of Operations

1. Print 45 *'s
2. Return.

ACACP - Does File Cleanup and Completes Shut Down

Entry Point: ACACP

External References: .MPY,.ENTR,EXEC,IFBRK,ACOM1,ACOM4,ACOM6,
ACOM9,READF,ACGSP,ACINM,RLMEM,ACERR,ACWRI,
RNRQ,CLOSE,ACCRE,MESSS,ACOPN,ACTRM,IVBOF,
ACDIR,ACFST,ACPGA,ACSID,WRITF

Calling Sequence: CALL ACACP

Used In: ACCT1

Sequence of Operations

1. If shut down or purge accounts, go to 11.
2. If not purge an account, go to 14.
3. Find account that is flaged to be purged.
4. If none flaged, go to 14.
5. If active session, go to 9.
6. If spool file, go to 9.
7. If disc mounted, go to 9.
8. Purge account.
9. Find next account that is flaged to be purged.
10. Go to 4.
11. Release memory.
12. Release class numbers and resource number.
13. If purge account, purge +@CCT!
14. Return.

ACNVS - Converses to Terminal

Entry Point: ACNVS

External References: .ENTR,ACOM7,ACOM4,ACPRM,ACREI,NAME,PARSN

Calling Sequence: CALL ACNVS (IOUT,NWORDS,MODE)

Parameters:

IOUT	OUTPUT STRING
NWORDS	NO OF WORDS IN STRING
MODE	PARSING MODE

Used In: ACCT1,ACALT,ACLOA,ACPUA,ACPUC

Sequence of Operations

1. Output string.
2. Input string.
3. If mode = 0, go to 6.
4. Call PARSN to parse first parameter of input string.
5. Return.
6. Call NAMR to parse first parameter of input string.
7. Return.

ACTIM - Prints Connect and CPU Times

Entry Point - ACTIM

External References: .MPY,.DIV,.ENTR,ACDDV,ACFMT

Calling Sequence: Call ACTIM (ITIME,IERR)

Parameters:

ITIME	Words 1&2 Connect Time Words 3&4 CPU Time
IERR	Error returned.

Used In: ACLIV

Sequence of Operations

1. Convert connect time.
2. Print connect time.
3. Convert CPU time.
4. Print CPU time.
5. Return.

ACSID - Set ID Bit Map

Entry Point: ACSID

External References: .MPY,.ENTR,ACOM6,ACOM5,ACOM1,ACSBT,
RNRQ,IVBVF,WRITF,READF

Calling Sequence: CALL ACSID

Used In: ACALU, ACACP

Sequence of Operations

1. Get first account.
2. If group account, go to 6.
3. Call ACSBT (IDU,MBUF).
4. If IDU < LOWUS then set LOWUS to IDU.
5. Go to 7.
6. If IDG>HIGR then get HIGR to IDG.
7. Call ACSBT (IDG,MBUF).
8. Get next account and if any, go to 2.
9. Post ID bit map.
10. Return.

ACNFG - Retrieves Entry From Configuration Table

Entry Point: ACNFG

External References: .MPY,.DLD,.DST,.ENTR,FLOAT,ACOM1,
ACOM6,MBYTE,READF,LBYTE

Calling Sequence: CALL ACNFG (IERR,IDX)

Parameters:

IERR	Returned ERROR
IDX	Index into Configuration Table

Used In: ACLIA

Sequence of Operations

1. Read record which contains IDX.
2. Put value in registers.
3. Increment IDX.
4. Return.

ACFDF - Finds Free Account Entry

Entry Point: ACFDF

External References: .MPX,.DIV,.ENTR,ACOM6,ACOM1,READT,MOD

Calling Sequence: CALL ACFDF (IDIRN,IRECM,IOFST,JERR,K)

Parameters:

IDIRN	=	DIRECTORY ENTRY NUMBER of free account
IRECM	=	RECORD NO. of free account
IOFST	=	Offset 0 or 64
JERR	=	ACERR return word
K	=	1 for normal request
K	=	2 for extension request (start on sector boundary)

Used In: ACALV, ACNWG, ACNWU

Sequence of Operation

1. If K=1 search every 64 word block for free entry.
2. Else search every 128 word block for free entry.
3. Set up return parameters.
4. Return.

ACGSP - Schedules GASP for Spool Information

Entry Point: ACGSP

External References: .DLD,.DST,.ENTR,EXEC,KSPCR,RMPAR

Calling Sequence: CALL ACGSP (NAME,IERR,TYPE)

Parameters:

NAME	USER.GROUP NAME
IERR	ERROR RETURNED
TYPE	FUNCTION FOR GASP

Used In: ACPUA, ACACP

Sequence of Operations

1. Check $SPCR if 0 return.
2. Build run string.
3. Schedule GASP.
4. Call RMPAR.
5. Set IERR.
6. Return.

ACGTG - Gets Group Account

Entry Point: ACGTG

External References: .ENTR,ACOM1,ACFDA,READF

Calling Sequence: CALL ACGTG (IGRP,IBUF,IOFST,IERR)

Parameter:

IGRP	5-WORD BUFFER CONTAINING GROUP NAME
IBUF	128-WORD BUFFER WHERE ACCOUNT ENTRY IS RETURNED
IOFST	OFFSET INTO BUFFER (0 or 64)
IERR	ERROR RETURN WORD (-200) FMP ERROR

Used In: ACLIV

Sequence of Operations

1. Set IUSER = 0.
2. Call ACFDA.
3. Read account into buffer.
4. Return.

ACGTU - Gets User Account

Entry Point: ACGTU

External References: .ENTR,ACOM1,ACFDA,READF

Calling Sequence: CALL ACGTU (IUSER,IGRP,IBUF,IOFST,IERR)

Parameters:

IUSER	5-WORD BUFFER USER NAME
IGRP	5-WORD BUFFER GROUP NAME
IBUF	128-WORD BUFFER WHERE ACCOUNT IS RETURNED
IOFST	OFFSET INTO BUFFER (0 or 64)
IERR	ERROR RETURNED (-200 of FMP error)

Used In: ACALU, ACLIU, ACNWU

Sequence of Operations

1. CALL ACFAA to find account.
2. Read account into IBUF.
3. Return.

ACGID - Gets Free ID Number

Entry Point: ACGID

External References: .ENTR,IABS,ACOM5,ACOM6,ACOM1,RNRQ,
READF,ACGBT,WRITF

Calling Sequence: CALL ACGID (ITYPE,ID,IERR)

Parameters:

ITYPE	1	Get user ID
	-1	Get group ID
ID	ID number returned	
IERR	ERRORS	-1 invalid parameter
		-2 No ID available
		FMP error

Used In: ACNWG, ACNWU

Sequence of Operations

1. If ITYPE = 1, go to 6.
2. Search up for ID.
3. If ID >=LOWUS, go to 10.
4. Post ID.
5. Return.
6. Search down for ID.
7. If I<=IHIGR, go to 10.
8. Post ID.
9. Return.
10. IERR = -2.
11. Return.

ACGBT - Gets Bit Out of ID Map

Entry Point: ACGBT

External References: .ENTR

Calling Sequence: CALL ACGBT (NWRD,IDIR,BITNO)

Parameters:

NWRD	WORD IN WHICH BIT IS SEARCH
IDIR	DIRECTION OF SEARCH -1 means 0 to 15 1 means 15 to 0
BITNO	BIT NO of available bit

Used In: ACGID

Sequence of Operations

1. Get word.
2. IF IDIR = -1, go to 8.
3. Set count to 15.
4. Rotate left through E.
5. If E zero, go to 13.
6. Decrement count.
7. Go to 4.
8. Set count to 0.
9. Rotate right through E.
10. If E zero, go to 13.
11. Increment count.
12. Go to 9.
13. Set bit in original word.
14. Return.

ACSBT - Sets Bit in ID Map

Entry Point: ACSBT

External References: .ENTR

Calling Sequence: CALL ACSBT (ID,NBUF)

Parameters:

ID	ID number of which corresponding bit in bit map must be set.
NBUF	BUFFER which contains Bit Map.

Used In: ACSID

Sequence of Operations

1. Compute word which must be updated.
2. Compute bit.
3. Inclusive "OR" bit into existing word.
4. Return.

ACASB - Searchs for Active Session

Entry Point: ACASB

External References: .MPY,.ENTR,ACOM6,IVBUF

Calling Sequence: CALL ACASB (IDIRN,LU,I)

Parameters:

IDIRN	IS DIRECTORY ENTRY NUMBER
LU	IS THE STATION LU WHERE ACCOUNT IS ACTIVE
I	IS THE INDEX INTO THE ACTIVE SESSION TABLE

Used In: ACTEL

Sequence of Operations

1. Search for entry with IDIRN.
2. If not found return.
3. Read LU and return.

IVBUF - Treats File as Large Array

Entry Point: IVBUF

External References: READF, WRITF, ACOM1

Calling Sequence:
```
READ    I=IVBUF (INDEX,IREC)
WRITE   CALL IVBUF (INDEX,IREC,IVAL)
POST    CALL IVBUF
```

Parameters:

INDEX	IS INDEX INTO THE LARGE ARRAY
IREC	IS THE STARTING RECORD NUMBER OF THE ARRAY
IVAL	IS THE VALUE TO BE WRITTEN

Used In: ACALT, ACPUA, ACACP, ACSID, ACASB

Sequence of Operations

1. If post request post both records and return.
2. Else compute record number.
3. If in memory, go to 6.
4. Post oldest buffer to disc.
5. Read in new buffer.
6. If read, read value.
7. If write, write value.
8. Return.

ACINM - Initialize and Release Session Memory

Entry Points: ACINM, RLMEM

External References: .ENTR,EXEC,$LIBR,$LIBX,$LGOF,$LGON,$STM,
$DSCS,$SMVE,$SRTN,$SALC,$BALC,$BRTN,$SMEM

Calling Sequence: CALL ACINM (ISIZE,MAXEV,IBUF,LNGTH,OLDLN)

Parameter:

ISIZE	AMOUNT OF MEMORY REQUESTED
MAXEV	LARGEST BLOCK POSSIBLE
IBUF	BUFFER CONTAINING DISC POOL
LNGTH	LENGTH OF DISC POOL
OLDLN	OLD LENGTH OF DISC POOL

Used In: ACCT1, ACALT, ACACP

Sequence of Operations

1. If memory allocated, go to 5.
2. Allocate memory.
3. Give it to $SALC.
4. Allocate class numbers.
5. Allocate memory for disc pool.
6. Transfer disc pool.
7. Set up $DSCS.
8. Return.

Calling Sequence: CALL RLMEM (IDSCS,ICLAS)

Parameters:

IDSCS	New value for $DSCS
ICLAS	CLASS numbers used by ACCTS

Used In: ACACP

Sequence of Operations

1. Release class numbers.
2. Give disc pool back.
3. Take memory from session.
4. Return it to $BRTN.
5. Return.

ACLNK - Links to Subroutines in Other Segments

Entry Point: ACLNK

External References: .ENTR,EXEC,ACOM2,SEGLD,ACERR,ACWRI

Calling Sequence:
```
ASSIGN 100 TO LRTRN
ASSIGN 200 TO LRTR2
CALL ACLNK (ISEG,IGOTO)
```

Parameters:

ISEG	IS THE ASCII (1H) OF THE LAST CHARACTER OF THE SEGMENT NAME
IGOTO	IS THE INDEX TO BE USED BY COMPUTED GO TO IN THE SEGMENT

Used In: ACCTS,ACCT1,ACMND,ACHLP

Sequence of Operations

1. If in memory, jump to start.
2. Else build segment name.
3. Call SEGLD.

ACLTM - Prints Last Log of Time.

Entry Point: ACLTM

External References: .ENTR

Calling Sequence: CALL ACLTM (ITIME,IBUF)

Parameters:

ITIME	2-wd Array of the Time
IBUF	17-wd Buffer for ASCII of Time

Used In: ACLIU, ACLIA

Sequence of Operations:

1. Get seconds, minutes, and year.
2. Separate seconds, minutes, and year.
3. Adjust year.
4. Get hours and days.
5. Compute AM or PM.
6. Compute month, day of month, and day of week.
7. Convert and put ASCII in buffer.
8. Return.

ACOPL - Opens List File

Entry Point: ACOPL

External References: .ENTR,IABS,IFBNR,ACOM2,ACOM3,ACOMC,ACOPN,
IXOR,LUTRU,ACLCK,ACERR,LURQ,ACTIN,ACROP

Calling Sequence: CALL ACOPL (IERR,ITYPE,JSIZE)

Parameters:

IERR	ERROR return FMP ERROR.
ITYPE	TYPE of OPEN LIST or BINARY.
JSIZE	Size of file if created.

Used In: ACCT1,ACMND,ACLIV,ACLIA,ACLOA,ACUNL,ACHLP

Sequence of Operations:

1. If ITYPE.GE.0, go to 4.
2. Open accounts file.
3. Go to 8.
4. If list = LLIST, use LLIST and return.
5. If LU then lock it and return.
6. Else check file not in input stack.
7. If in input stack report error and return.
8. Open file.
9. Return.

ACLCK - Locks List LU

Entry Point: ACLCK

External References: .ENTR,EXEC,IFBRK,XFTTY,LURQ,ABREG,ACWRI

Calling Sequence: CALL ACLCK (LU,IERR)

Parameters:

LU	LU which is to be locked.
IERR	ERROR returned
	10 Break
	12 LU not in switch table

Used In: ACOPL,ACHLP,ACXFR

Sequence of Operations:

1. If TTY return.
2. Lock LU no wait.
3. If reject set error 12 word return.
4. If not previously locked return.
5. Else print messages.
6. Test Break Flag. If set set error = 10 and return.
7. Suspend 50 milliseconds
8. Go to 2.

ACROP - Opens or Creates File

Entry Point: ACROP

External References: .ENTR,OPEN,POSTN,CREAT

Calling Sequence: CALL ACROP (IDCB,IERR,NAME,IOPT,ISC,
ICRN,ISIZE,ITYPE)

Parameters:

IDCB	FMP Data Control Block.
IERR	FMP Error return.
NAME	NAME of file.
IOPT	Open option.
ISC	Security code of file.
ICRN	Cartridge reference # of file.
ISIZE	File size.
ITYPE	File type.

Used In: ACOPL,ACXFR

Sequence of Operations:

1. Try to open file.
2. If error - 6, go to 6.
3. If an error, return.
4. If TYPE = 3, POSTN file to end.
5. Return.
6. Create file.
7. Return.

IFBNR - Determines if Device has Binary Mode.

Entry Point: IFBNR

External References: .DIV,.ENTR,XLUEX

Calling Sequence: IF (IFBNR(IRW,LU))...

Parameters:

IRW	MODE OF OPERATION	
	=	0 Both Read and Write.
		1 Read
		2 Write
		3 Binary
LU	Logical Unit of Device.	

Used In: ACOPL,ACWRL

Sequence of Operations:

1. Read Status EXEC (15,...)
2. If device can handle transfer, go to 5.
3. Else set IFBNR = .FALSE.
4. Return.
5. Set IFBNR = .TRUE.
6. Return.

ACNXA - Gets Next Account Directory Entry and Compresses Directory

Entry Point: ACNXA

External References: .MPY,.ENTR,ACOM1,ACOM6,ACOM9,READF,MOD

Calling Sequence: CALL ACNXA (J,IREC,IDEL,KOUNT,IDIR,IDELX)

Parameters:

J	IS OFFSET INTO JBUF OF DIRECTORY ENTRY J is set to -1 to start at beginning of directory.
IREC	Is REC NUMBER OF DIRECTORY ENTRY
IDEL	Is MINUS # holes in directory.
KOUNT	Is the count of directory entries.
IDIR	Is directory entry number.
IDELX	Is index into delta table.

Used In: ACUNL

Sequence of Operations:

1. Get first directory entry.
2. If an extent align to even directory.
3. If empty adjust delta.
4. Else return.

ACFID - Fix Message, User and Group Pointers

Entry Point: ACFID

External References: .ENTR

Calling Sequence: CALL ACFID (IVAL,IDELI,KDEL)

Parameters:

IVAL	Location of pointer in account.
IDELI	Initial
KDEL	Point in delta table.

Used In: ACUNL

Sequence of Operations:

1. Search for delta.
2. Add delta and initial offset to IVAL.
3. Restore IVAL.
4. Return.

ACPGA - Clears Directory Entry

Entry Point: ACPGA

External References: .ENTR, ACDIR

Calling Sequence: CALL ACPGA (I,IDIRN,ID)

Parameters:

I	Value for first word of directory -1 = free, -2 = extent
IDIRN	DIRECTORY ENTRY NUMBER
ID	ID number of account.

Used In: ACALU, ACNWU, ACACP

Sequence of Operations:

1. Set IBUF (1) to I.
2. Write directory entry.
3. Return.

ACTRM - Terminates ACCTS

Entry Point: ACTRM

External References: .ENTR,EXEC,ACOMC,ACOM2,ACOM1,ACOM3,ACOM4,
ACOM6,CLOSE,ACCLS,LURQ,XLUEX,DTACH

Calling Seqeunce: CALL ACTRM

Used In: ACCTS,ACCT1,ACPAS,ACAPA,ACPSN,ACACP

Sequence of Operations:

1. Close files.
2. Unlock LU's.
3. Print END ACCTS.
4. If no clean up, go to 7.
5. If PROG is "ACCTS", go to 8.
6. Schedule "ACCTS" to do clean up.
7. Terminate (EXEC(6)).
8. Schedule myself ACCTS to clean up in 30 seconds.
9. Terminate.

ACDDV - Double Word DIVIDE

Entry Point: ACDDV

External References: .ENTR

Calling Sequence: CALL ACDDV (IDBL,IDIVR,IQUOT,IREMN)

Parameters:

IDBL	DOUBLE WORD INTEGER
IDIVR	DIVISOR
IQUOT	QUOTIENT
IREMN	REMAINDER

Used In: ACTIM

Sequence of Operations:

1. Load dividend.
2. Swap A and B registers.
3. If sign bit set, go to 6.
4. Divide.
5. If no overflow, go to 8.
6. Set quotient MAX.
7. Set remainder to max mod value.
8. Store quotient.
9. Store remainder.
10. Return.

ACDIR - Reads and Writes Directory Entries

Entry Point: ACDIR

External References: .MPY,.DIV,.ENTR,ACOM6,ACOM1,READF,WRITF,ACERR

Calling Sequence: CALL ACDIR (ICODE,IDIRN,IBUF,IERR)

Parameters:

ICODE	1 READ
	2 WRITE
IDIRN	DIRECTORY ENTRY NUMBER
IBUF	16 WD BUFFER TO READ OR WRITE
IERR	ERROR RETURN WORD

Used In: ACCT1,ACALU,ACNWG,ACNWU,ACACP,ACPUU,ACAPA,ACPGA

Sequence of Operations:

1. Verify parameters, if ok, go to 3.
2. Get error, return.
3. Compute record number and offset.
4. Read record.
5. If write, go to 8.
6. Move from record to buffer.
7. Return.
8. Move buffer to record.
9. Post to disc.
10. Return.

ACFDA - Finds Account Entry

Entry Point: ACFDA

External References: .MPY,.DIV,.ENTR,ACOM6,ACOM1,ACOMA,READF

Calling Sequence: CALL ACFDA (IUSER,IGRP,IDIRN,IRECU,IRECG,JERR)

Parameters:

IUSER	5-WD BUFFER containing user name.
IGRP	5-WD BUFFER containing group name.
IDIRN	DIRECTORY ENTRY NO. (Returned)
IRECU	2-WD ARRAY Record number and offset of user account
IRECG	2-WD ARRAY Record number and offset of group account
JERR	ERROR returned -200 not found or FMP ERROR.

Used In: ACALU,ACNWG,ACNWU,ACPUU,ACTEL,ACGTG,ACGTU

Sequence of Operations:

1. If searching, go to 4.
2. Initialize indexes to start of directory.
3. Go to 8.
4. If group search, go to 7.
5. Set indexes to last found user account.
6. Go to 8.
7. Set indexes to last found group.
8. Scan for match (@ matches anything).
9. Save indexes (for future searches).
10. Return directory number record numbers and offset(s).

ACFMT - Formats and Outputs Data

Entry Point: ACFMT

External References: ACWRL,.ENTP,ACOM2,NAM..

Calling Sequence: CALL ACFMT (IERR,F1,IBUF1,-N,F2,IBUF2,F3,...,FN,IBUFN)

Parameters:

F1,F2,...FN are function codes

N is number of blanks (next PARM is a function code)

Function codes are:

where:	0<FN<"I0"	PRINT N ASCII CHARACTERS
	FN=0	PRINT ASCII CHARACTERS UNTIL BLANK IS ENCOUNTERED (MAXIMUM NO OF CHARS FOLLOWS)
	FN="I0"	PRINT DECIMAL NUMBER WITHOUT LEADING BLANKS
	"I0"<FN<"I6"	PRINT IN "In" FORMAT
	FN="CR"	PRINT ASCII IF LEGAL NAME ELSE PRINT INTEGER

IBUF1,IBUF2,IBUF3,... ARE EITHER ASCII STRINGS OR NUMERIC DATA

Used In: ACLIV,ACLIA,ACTIM

Sequence of Operations:

1. Clear parameter addresses.
2. Call .ENTP to get parameters.
3. Get first function code.
4. If positive, go to 7.
5. Put -N number of blanks in output buffer.
6. Go to 18.
7. If function code =,"CR", go to 11.
8. If function code ="I0", "I1",...,"I6", go to 16.

9. Transfer string to output buffer.

10. Go to 18.

11. Call NAM..

12. If not ASCII, go to 15.

13. Put 2 ASCII characters in output buffer.

14. Go to 18.

15. Get code to "IO".

16. Convert decimal number to ASCII.

17. If "IO" suppress leading blanks else use full field width.

18. If another function code, go to 4.

19. Return.

ACCLL - Closes List File or Unlocks LU.

Entry Point: ACCLL

External References: .ENTR,ACOM3,XLUEX,LURQ,ACCLS

Calling Sequence: CALL ACCLL

Used In: ACLIV,ACLIA,ACLOA,ACUNL,AHLP,ACWRL

Sequence of Operations:

1. If file, go to 5.
2. Send top of form.
3. Unlock LU.
4. Go to 6.
5. Call ACCLS to close file.
6. Set &IST =-1.
7. Return.

ACCLS - Closes and Truncates File

Entry Point: ACCLS

External References: .DIV,.ENTR,LOCF,CLOSE

Calling Sequence: CALL ACCLS (IDCB,ITYPE)

Parameters:

IDCB	DCB of file to be closed
ITYPE	Type of file to be closed

Used In: ACTRM,ACCLL,ACSFR

Sequence of Operations:

1. Set truncate to zero.
2. If type 1, go to 5.
3. Call locf to find position.
4. Compute truncate parameter.
5. Close and truncate.
6. Return.

ACPRM - Prompts Interactive Device

Entry Point: ACPRM

External References: .ENTR,IFBRK,ACOM8,ACERR,ACWRI

Calling Sequence: CALL ACPRM (MSG,MSGLNG)

Parameters:

MSG	ASCII STRING TO BE WRITTEN
MSGLNG	LENGTH of string in words

Used In: ACCT1,ACMND,ACALT,ACALU,ACNWG,ACNWU,ACPSN,ACNVS,ACREI

Sequence of Operations:

1. If MSGLNG = -1, go to 4.
2. Save buffer in local buffer.
3. Output string.
4. Return.
5. Output string from local buffer.
6. Return.

ACREI - Inputs Commands From Device, File, or Memory

Entry Point: ACREI

External References: .MPY,.DIV,.ENTR,ACOM3,ACOMC,XLUEX,ABREG,ACWRI,
NAMR,MBYTE,ACXFR,ACERR,ACHLP,ACPRM,READF

Calling Sequence: CALL ACREI (IBUF,IERR)

Parameters:

IBUF	INPUT BUFFER
IERR	ERROR RETURN

Used In: ACCT1,ACMND,ACALT,ACALU,ACNWG,ACNWU,ACPSN,ACNVS

Sequence of Operations:

1. Reading from memory, go to 13.
2. Reading from file, go to 11.
3. Read from LU.
4. If echo set write to log device.
5. Fill with blanks.
6. If not "/TR" or "/HE" return.
7. If "/TR" call ACXFR.
8. If "/HE" call ACHLP.
9. Call ACPRM with MSGLNG = -1.
10. Go to 1.
11. Call READF.
12. Go to 4.
13. Transfer data from LDCB.
14. Go to 4.

ACHLP - Process HELP Commands

Entry Point: ACHLP

External References: .ENTR,EXEC,ACOM2,ACOM3,ACOMC,ACOMD,NAMR,ACOPL,ACLNK, ACWRL,ACERR,ACCLL,ACITA,LUTRU,LURQ,ACLCK,ACWRI

Calling Sequence: CALL ACHLP (ICMND,ISTRC)

Parameters:

ICMND	Command string
ISTRC	String pointer

Used In: ACCTS,ACMND,ACREI

Sequence of Operations:

1. If scheduled from ACREI or parameter is numeric, go to 7.
2. OPEN list file.
3. Call ACLNK to call ACWRL.
4. If not found, write not found.
5. Close list file.
6. Return.
7. Convert list device.
8. If same as list already open unlock LU.
9. Schedule HELP.
10. Relock LU if necessary.
11. Return.

ACERR - Posts and Prints Errors

Entry Point: ACERR

External References: .ENTR,ACOMC,ACITA,PTERR,ACXFR,ACWRI

Calling Sequence: CALL ACERR (IERR)

Parameters:

IERR	Error number to be posted

Used In: ACCT1,ACCT2,ACMND,ACALT,ACALU,ACLIU,ACLIA,ACLOA,ACNWG,ACNWU, ACPAS,ACAPA,ACPSN,ACPUA,ACPUU,ACTEL,ACUNL,ACACP,ACLNK,ACOPL, ACDIR,ACPRM,ACREI,ACHLP,ACWRL

Sequence of Operations:

1. Convert error code to ASCII.
2. Post error to DCB.
3. Transfer to log device.
4. Write error.
5. Return.

ACWRL - Writes to List File or Device

Entry Point: ACWRL

External References: .ENTR,IFBNR,XFTTY,IFBRK,ACOM2 ,ACOM3 ,XLUEX,
WRITF,ACWRI,ACERR,ACCLL

Calling Sequence: CALL ACWRL (IBUF,NO,IERR)

Parameters:

IBUF	is output buffer
NO	is number of words in buffer
IERR	is error returned

Used In: ACWRH,ACLIV,ACLIA,ACUNL,ACSTR,ACFMT,ACHLP

Sequence of Operations:

1. If list file, go to 5.
2. If list default, go to 10.
3. Write to device.
4. Return.
5. If logical list, go to 8.
6. Write to list file.
7. Return.
8. Write to logical list file.
9. Return.
10. Write to input device.
11. Return.

ACREL - Read From List Device or File

Entry Point: ACREL

External References: .ENTR,ACOM1,ACOM3,XLUEX,ABREG,READF

Calling Sequence: CALL ACREL (IBUF,NO,LEN,IERR)

Parameters:

IBUF	is input buffer
NO	is buffer size
LEN	is words transmitted
IERR	is error returned

Used In: ACLOA

Sequence of Operations:

1. If read from +@ACCT! go to 7.
2. If read from file, go to 5.
3. Read from device.
4. Return.
5. Read from file (LDCB).
6. Return.
7. Read accounts file.
8. Return.

ACITA - Converts Integer to ASCII

Entry Point: ACITA

External Reference: .ENTR

Calling Sequence: CALL ACITA (INT,IBUF,NOWDS)

Parameters:

INT	Integer to be converted
IBUF	Buffer where ASCII if put
NOWDS	No of words in buffer

Used In: ACCT1,ACALT,ACALU,ACLOA,ACNWU,ACMSN,ACHLP,ACERR

Sequence of Operations:

1. If negative generate minus sign.
2. Change to positive.
3. Convert number with leading zeros.
4. Return.

MBYTE,LBYTE - Retrieves Upper or Lower Byte of Word

Entry Point: MBYTE,LBTYE

Calling Sequence:
```
IUP = MBYTE (INT)
ILW = LBYTE (INT)
```

Parameters:

INT is integer

Used In: ACCTl,ACALT,ACALU,ACLIV,ACLIA,ACNWU,ACPUU,ACTEL,ACNFG,ACREI

Sequence of Operations:

MBYTE

1. Get INT.
2. Swap bytes.
3. Go to 5.

LBYTE

4. Get INT.
5. Mask off upper byte.
6. Return.

ACXFR - Transfer Control to Device or Command File

Entry Point: ACXFR

External References: .MPY,.ENTR,ACOMB,ACOM3,ACOMC,LOCF,CLOSE,NAMR,LUTRU,
OPEN,APOSN,LURQ,ACLCK,ACCLS,ACTEN,ACROP

Calling Sequence: CALL ACXFR (ICMND,ISTRC,IERR)

Parameters:

ICMND	Command string
ISTRC	No of words in string
IERR	Error returned

Used In: ACCTS,ACMND,ACREI,ACERR

Sequence of Operations:

1. If current input is not file, go to 4.
2. Save current position.
3. Close file.
4. Parse to get next.
5. If null, set to -1.
6. If negative back up transfer stack and go to 8.
7. Advance transfer stack and put new file or LU on stack.
8. If file, go to 12.
9. If error, go to 14.
10. Lock LU.
11. Go to 15.
12. Open and position file.
13. Go to 15.

14. Set LU to log LU.
15. Reset list LU if specified.
16. Reset Echo bit if specified.
17. Return.

ACTIN - Test List Files Against Transfer Stack File Names.

Entry Point: ACTIN

External References: .ENTR,ACOMB

Calling Sequence: CALL ACTIN (IPBUF,IERR)

Parameters:

IPBUF	Buffer with namr of list file
IERR	Error which is returned.

Used In: ACOPL, ACXFR

Sequence of Operations:

1. Compare IPBUF with first entry in stack.
2. If equal, go to 7.
3. Compare IPBUF with next entry in stack.
4. If equal, go to 7.
5. If more entries in stack, go to 3.
6. Return.
7. Set error.
8. Return.

ACWRI - Writes to Input Device

Entry Point: ACWRI

External References: .ENTR,ACOM3,ACOMC,XFTTY,XLUEX,WRITF

Calling Sequence: CALL ACWRI (IBUF,ILEN)

Parameters:

IBUF is buffer to write

ILEN is length of buffer in words IF ILEN<0 CALL FROM ACREI

Used In: ACCT1,ACMND,ACALT,ACALU,ACLOA,ACNWU,ACACP,ACPUA,ACAPA,ACLNK, ACLCK,ACPRM,ACREI,ACHLP,ACERR,ACWRL

Sequence of Operations:

1. If echo call from ACREI, go to 6.
2. If not interactive LU, go to 4.
3. Print buffer.
4. If there is not a list of file or LU, go to 6.
5. Write to list file or LU.
6. If echoing and input is not the same aa the log unit then write to log unit.
7. Return.

ACSES - Utilities (1) ACSES Session Shut Down
(2) Store Prompt String
(3) Retrieve Spool Disc LU
(4) Read Cartridge List

Entry Point: ACSES,LMES,KSPCR,ACFST

External References: EXEC,.ENTR,$LIBR,$LIBY,$DSCS,$LGOF,$LGON,$LMES,
$SPCR,$CL1,$CL2

Calling Sequences:
```
CALL ACSES (-2) SHUT DOWN
CALL ACSES (0) START UP
CALL LMESS (JBCNT,JBUF,IDSC1)
I = KSPCR (IDUM)
CALL ACFST (IBUF)
```

Parameters:

JBCNT	LNGTH of log on string
JBUF	LOGON string
IDSC1	VALUE to put in $DSCS+1
IDUM	DUMMY parameter
IBUF	Buffer for cartridge list

Used In: ACCT1,ACALT,ACLIA,ACACP,ACLOA,ACPUA,*GSP

Sequence of Operations:

ACSES

1. Stuff parameter into $DSCS+1.
2. If not restart, return.
3. Tell LOGON to shut down.
4. Tell LGOFF to shut down.
5. Return.

LMESS

1. Put long string in memory.
2. Update $DSCS+1.
3. Return.

KSPCR

1. Retrieve $SPCR.
2. Return.

ACFST

1. Read cartridge list.
2. Mask all ID words.
3. Return.

Chapter 14

SESSION TERMINAL HANDLERS

OVERVIEW

The Session Terminal Handlers provide for the initiation of the log-on sequence, the actual log-on process, after logged-on the processing of "break" mode requests and finally the log-off process.

Figure 14-1 shows the four programs which provide the above defined processes. The PRMPT program gets things started and is the main controlling processor. PRMPT is scheduled by interrupt (from each terminal requesting service) and then communicates with either the log-on processor (LOGON) to perform a log-on, or the command response processor (RPN) if a session does not exist for the interrupting terminal.

The log-off processor (LGOFF) receives its control from either a session progenitor (copy of FMGR) or the account program (ACCTS). Note that RPN, LOGON and LGOFF receive all their control via class I/O.

By using class I/O, the prompt program can initiate several log-on requests before the first is completed. This method avoids the problem of the log-on program being busy when another log-on request is recognized by the program prompt. Remember, the prompt program is scheduled by interrupt, it cannot wait around for the log-on program to complete.

OPERATING ENVIRONMENT

1. SESSION TERMINALS must be configured with an interrupt table entry of:

   ```
   XX,PRG,PRMPT
   ```

2. At least three class numbers must be available and the ACCTS program must have performed its initialization process. This initialization includes the allocation of memory for control blocks and the allocation of the three class numbers required by the session terminal handlers. These class numbers are saved in Table Area I to prevent their loss should one of the terminal handlers be aborted.

   ```
   $LGON  -  LOGON PROGRAM CLASS #
   $LGOF  -  LGOFF PROGRAM CLASS #
   $STH   -  R$PN$ PROGRAM CLASS #
   ```

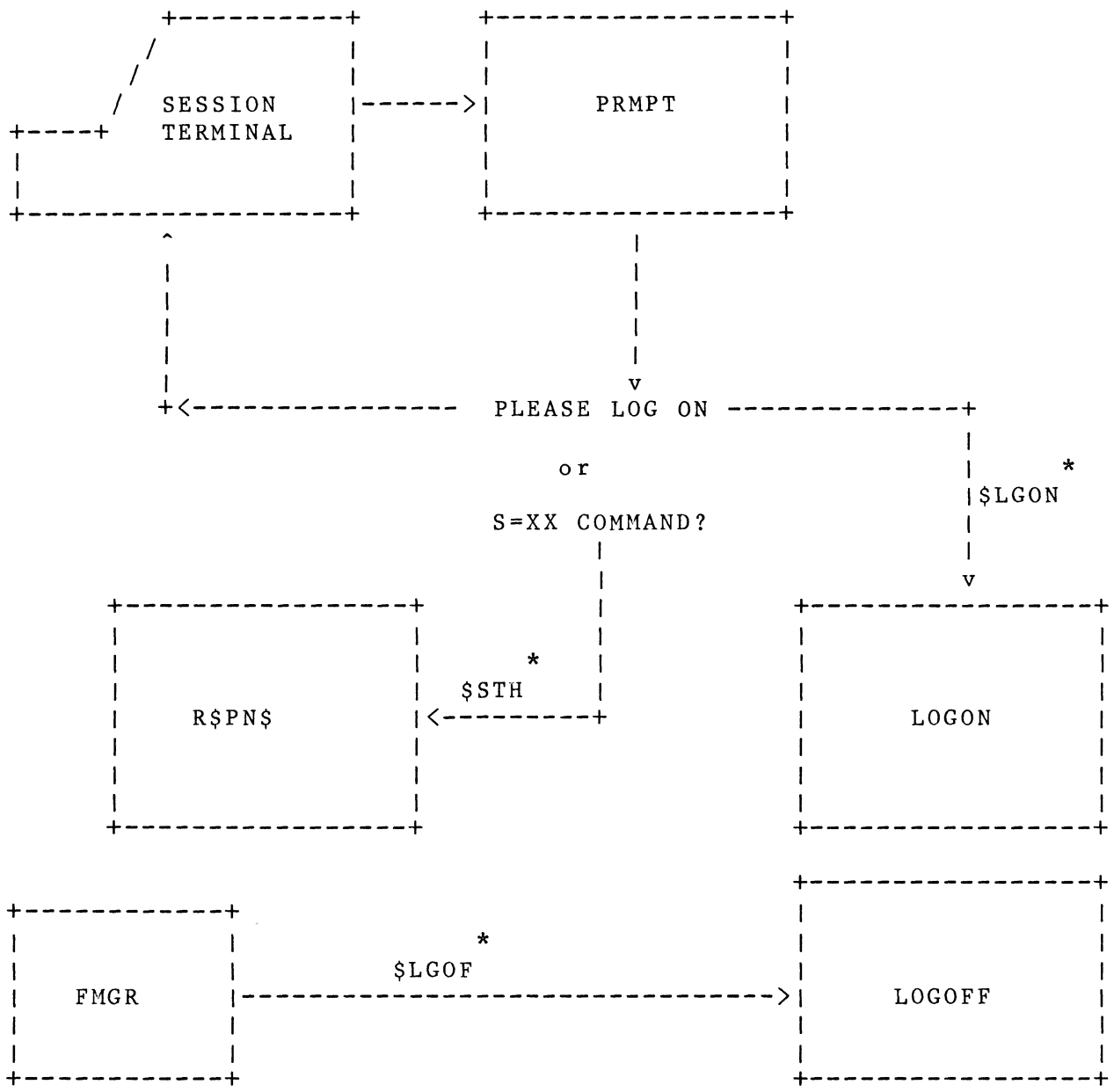


Figure 14-1. Session Terminal Handling Process

3. The terminal must have been enabled to permit the driver to schedule the PRMPT program whenever an asynchronous interrupt is received.

Typically, a terminal is enabled with the :CT,LU# command. It may be disabled with the :CT,LU#,21B command.

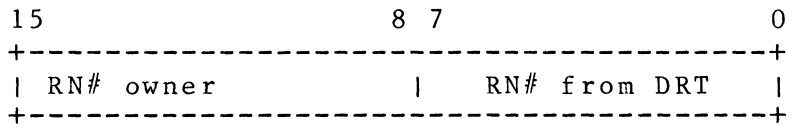
Enabling the terminal allows the driver to place PRMPT's ID address into the associated EQT. The driver takes the ID address out of the interrupt table (it is in two's complement form) and places it into a temporary word in the EQT or EQT extension (as a positive address). The interrupt table entry is then replaced with the first word address of the referenced EQT. SESSION is now ready to handle the terminal.

An interrupt from the CRT (TTY) device is generated by hitting any key. $CIC in RTIOC is entered and vectors the interrupt to the appropriate driver. The driver then determines if the terminal is enabled, if not the interrupt is ignored. If the terminal is enabled, the driver schedules PRMPT via the system routine $LIST and passes the address of the fourth word of the appropriate EQT.

Prompt Processing:

Prompt is given the address of EQT 4, of the interrupting device when it is scheduled. From the EQT address, the routine FNDLU calculates the LU, the device status, device type and possible RN bypass word. If the LU or the EQT is down, the request is ignored. The RN bypass word is returned if the interrupting device is locked.

It is possible to write through an LU lock. Parameter nine (RQP9) of all I/O EXEC requests has been reserved as an LU lock bypass word. The word is configured as:



The RN# owner can be retrieved by indexing into the RN# Table and isolating the lower byte. If the above word is configured and a DEF is made to it in RQP9, then the system will not suspend the executing program and will honor the I/O request. Only "BREAK" mode requests will be processed if the terminal is locked.

If a log-on is required, an error is reported and the request is ignored.

If the interrupting device is a multipoint terminal, a control request to set the EDIT-MODE flags is issued.

Next, a check is made to insure that the session environment has been correctly enabled (class number defined). If the environment is OK, the terminal is disabled to prevent multiple log-on or break mode requests to be initiated before the first one has completed. The disable is performed by setting a bit in the session bit map, !BITM. The disable bit is set by the PRMPT program and cleared by RPN or LOGON. A "soft" disable rather than a "hard" disable (CN,LU,21B) is used to prevent ever "losing" a terminal. A terminal can be "lost" if the program responsible for enabling the terminal has been aborted. In this case, the terminals interrupts would be ignored until a control request to enable it was received.

With the "soft" disable, if the reenable program should be aborted, pressing a key would get PRMPT scheduled. Upon finding the bit map entry set, PRMPT will make sure that LOGON, LGOFF and RPN are all scheduled and then terminate.

Next, PRMPT checks to see if a session is defined for the terminal. This is performed by scanning the list of active session, looking for a match with the session ID (SESSION ID = TERM LU).

If a session exists, the "break" mode prompt, S=XX COMMAND?, is issued. The class read of the response is initiated (class # = $STH) and the response program is scheduled. PRMPT then terminates, waiting for the next interrupt to be received.

If a session does not exist, the log-on or shut-down message is issued. This prompt string is defined externally and is updated by the ACCTS program. If we are in a shut down mode ($DSCS+1 <0), do not initiate the read of the response, just clear the disable bit (!BITM) and terminate. Otherwise, start the class read of the response (class # = $LGON), schedule the logon processor (LOGON) and terminate.

RPN Processing:

RPN is the program responsible for processing all session operator commands. After some initialization (session enabled and class # defined), a class "GET" request is issued on the class # $STH. R$PN$ waits for a completion of a command input from any of the active session terminals.

Next, the same checks performed by PRMPT are made to assure that the device may be written to. If the terminal is OK, a check is made to verify that the session issuing the command still exists (could have logged off) and the soft disable on the terminal is cleared (!BITM).

All operator commands are checked to verify that the user has sufficient capability to execute it. The system library routine CAPCK, performs the capability check and returns a pass-fail status. If the user has the ability to perform the command, we next check to see if any special processing must be performed. The following commands are processed by the RPN program:

FL: Make control request to flush all requests to this terminals driver until all buffered requests have been cleared or a read request is found.

WH: Schedule the WHZAT program for this session.

HE: Schedule the HELP program for this session.

TE: Send a message to the system console.

BR: If no parameters, break the current session program.

SS: If no parameters, suspend the current session program.

GO: If no parameters, issue a "GO,PROG" against the current session program.

OF: If no parameters, abort the current session program (if FMGXX, do not abort but set the break flag).

UP: If no parameters, and if the current session program is waiting for a down device, issue a "UP,EQT" against the down device.

SL: Display the session-system LU mapping.

RS: If FMGXX is active (not dormant) do a OF,FMGXX,1 then schedule FMGXX for this session. If FMGXX can not be found, create one, then schedule it for this session.

The current session program is determined by finding the last son of FMGXX or FMGXX itself. Note: except for the "UP" command the programs D.RTR and SMP are exempted from the son test as you should never abort or suspend these programs.

If the command was not one of the aforesaid "special" commands, pass the command to the message processor for handling by the operating system.

Issue any response from the operating system and finally, check for the next command to process by making another class "GET" request.

LOGON Processing:

All interactive processing performed by LOGON is performed via class I/O. This is done to aid in the reduction of the contention for the log-on program.

Once scheduled, LOGON never terminates, but is suspended on a class "GET" request, waiting for the next log-on request.

Calling Sequence

The LOG-ON process is initiated by the program PRMPT. The PRMPT program is scheduled as a result of striking any key on a terminal configured as a session terminal.

It (PRMPT) has the following responsibilities:

a. Set a bit map flag (in Table Area I) to indicate that log-on for this terminal is in progress.

b. Prompt the user for his USER.GROUP NAME. This prompt has the following format:

```
"PLEASE LOG-ON:"
```

c. Initiate a class read of the user response (using the communication class number, $LGON). The format of this input is as follows:

```
USER[.GROUP][/PASSWORD]
```

NOTES: "." and "/" are delimiters
: blanks are ignored
: maximum field size for user, group and password is ten characters each
: see definition of optional parameters passed in class request.

d. Schedule the program LOGON (without wait or queue), passing following information in the first two optional parameters:

```
WD1=0
WD2=Terminal logical unit number (1 to 99).
```

NOTE: The class I/O request allows optional parameters to be saved with the class request and then returned with the class "GET" request.

These optional parameters define the request to the log-on processor as follows:

a. INTERACTIVE log-on request

```
IOP1=0
IOP2=Terminal LU (1 to 99), used as session identifier
```

b. NON-INTERACTIVE log-on request

IOP1=0 or class # for communication from LOGON back to the calling program.

IOP2=Number to be used as session identifier (100 to 254).

c. PASSWORD reply

IOP1=NOT used

IOP2=-Character count of second buffer. The class read or write/read of the password must be a double buffered request. Refer to the password discussion for details on the required contents of the second buffer.

d. SHUT-DOWN request

IOP1=NOT used
IOP2=-1

e. Batch (JOB) LOG-ON request

IOP1=NOT used
IOP2=-2
Class Buffer=Directory entry # of account to be logged onto.

f. Account Name request

IOP1=NOT used
IOP2=-3
Class Buffer=Account directory entry #.

g. Account Directory Entry # request

IOP1=NOT used
IOP2=-4
Class Buffer="USER.GROUP/PASSWORD" string.

LOG-ON FLOW

START-UP WORK (only performed when LOGON is first scheduled).

a. Verify that the session environment has been initialized.

Any problems cause an error diagnostic to be issued and the log-on program will terminate.

b. Fetch SCB offsets and communication class number $LGON.

The SCB offsets are the locations of information in the SCB to be built by LOGON. The offsets are defined externally to allow ease of modification to the SCB structure.

c. OPEN the ACCOUNT file and check its validity.

The following conditions must hold for the account file or an error diagnostic is issued and LOGON terminates.

The file must be found.

The file must be type 1.

The low 8 bits of word 25, record 1 must contain a resource number lockable by LOGON.

1. Get a LOG-ON request.

Release possible RN lock and clear no-parse flag.

A class I/O "GET" request is issued against the class defined by $LGON. This request will cause the program LOGON to be suspended until a class read (or write/read) has completed.

NOTE: The SAVE buffer option is used with this request to prevent the loss of a log-on request in the event of LOGON being aborted. The buffer is released by another "GET" request before going on to the next log-on request.

2. Determine type of call.

If the original request was not a class read or write/read, release the current class bufffer and continue at 1.

If log-on request, continue at 3.

This is a shutdown request. The following functions are performed:

Close account file
Release class request buffer
Terminate

If this is a special request (IOP2 <0) continue at 6-1.

3. This is a log-on request.

3-1. Setup for log-on

Set interactive/non-interactive flags.
Request lock on account file resource number.
Get account file header into memory.

3-2. Check Session Limit

The session limit, along with the number of currently active sessions reside in the account file header.

If the session limit has been reached, 1) issue an error 2) continue at 5-1.

3-3. Parse the Input Buffer.

The password processor (refer to 6.0) verifies that a password was entered and then sets some flags and continues as a regular log-on request. One of the flags set is the no-parse flag. This indicates that the parse of the user group name has already taken place.

If the no-parse flag is true, continue at 3-4.

Call the parse subroutine to produce the following:

a. Byte counts of user, group and password.
b. Isolated user, group and password names.

NOTE: The user name is required but the group and password are optional. If a group name is not provided, the group name is filled with the default, "GENERAL".

If a user name was not entered, 1) issue an error 2) continue at 5-1.

3-4. Scan Account File Directory for User

The location (record) of the account file directory is defined by word 5 of the account file header. The number of records in the directory is defined as the difference between the location of the directory (word 5 of header) and the location of the first account entry (word 6 of header).

The first word of a directory entry (16 words each) defines the following:

0 = end of directory
<0 = free entry
>0 = user or group directory entry

Scan the account file directory until:

a. word 1 of an entry = 0

or

b. last directory record read

or

c. the user account directory entry is found.

If the user is not found, 1) Issue an error, 2) continue at 5-1.

3-5. User Identified

Word 15 of the user's directory entry defines the record number of the actual account entry. The sign bit of this word indicates where in the record the account entry resides. If clear, the entry begins on word 1 of the record. If set, the entry starts at word 64 of the record.

READ the user account into memory.

If a password is not required, continue at 3-7 (word 1 of the account file entry contains the character count of the password).

3-6. Password Required

If a password was passed, compare it to the required password. If it matches, continue at 3-7. Else, 1) issue an error, 2) continue at 5-1.

Password not passed, prompt the user for the required password:

"PASSWORD?"

3-6.1 If this is an interactive call, continue at 3-6.2. Else issue an error and continue at 5-1.

3-6.2 Interactive request of password

Setup and issue a double buffered class read (without echo) to the session terminal, on class number $LGON.

The contents of the second buffer, which must be returned with the "GET" of the response, is defined below:

WD1 = 0 if interactive
= class # for communication response

WD2 = session identifier

WD3 = Byte counts of user + group names (user byte count is in the high byte).

WD4-8 = user name (blank filled)

WD9-13 = group name (blank filled)

NOTE: Word 1 currently has no use. It is used here on the chance that non-interactive log-on requests may want to be able to prompt for and then return a password if it was not provided in the original request. By changing the above set up for the class write/read, the double buffered request could be sent back to the requestor.

3-7. User Identified

At this point, the user has been identified and verified. The user's account entry is in memory.

Check word 64 of the user's account entry. If the sign bit is set, the low 15 bits of this word define the record number of an extension for this account. This extension will always begin on word 1 of the sector.

Initialize SCB to zero and move in: Identifier; capability; user ID; Group ID, Disc Limit.

The SST spares (-1's) are placed in the SST first. These are the only entries which may be modified on-line. All the entries from the console definition (LU1) to the end of the table are defined by LOGON and are not altered thereafter.

Define first SST entry past end of spares for LU1. If interactive request, map LU1 to the terminal LU. If non-interactive, map session LU1 to system LU1.

LU2 is always mapped to LU2.

LU3, if defined in the system, is mapped to LU3.

Move SST entries from user account into the SST being built.

The system disc LU's are now defined by reading the cartridge directory (defined by $CL1 and $CL2) and then making an entry in the SST for each disc mounted to the MANAGER.SYS account (ID=7777B).

NOTE: The LU's in the SST are allowed 1 byte each with the system LU residing in the left hand byte. The LU's are saved as LU-1 to match internal operating system formats.

3-8. Check for a Configuration Table Entry

Save required information (from account file entry) before the input buffer is used for the configuration table search.

If this is a non-interactive request, continue at 4.

Scan the Configuration Table for an entry defined for this terminal. (NOTE: The SST entry for LU1 defines the station.)

If not found, continue at 4.

3-9. Configuration Table Entry Found

Move the entries in the Configuration Table to the SST being built.

If any entry causes a conflict in the definition of a session LU (duplicate session LU's, different system LU's), reject the Configuration Table entry and issue a diagnostic message. Continue with the next Configuration Table entry.

NOTE: Conflicts in the definition of SST entries do not terminate the log-on sequence.

4-0. Update the SST Length Word

The SST length word is the sum of the following:

Account file SST entries + 2 (or 3 if LU3 is defined) + Configuration Table entries + the number of spare entries requested (low byte, word 32, user account entry) + the number of system discs (other than LU's 2 + 3).

NOTE: As LU's are stored as LU-1, the spare entries are defined as -1's (LU-1).

4-1. Complete the Construction of the SCB

Place the two's complement of disc limit in the first word past the last SST entry.

Total size of SCB includes the number of words specified by the disc limit.

4-2. Build the Session Progenitor

If this is a non-interactive log-on request, continue at 4-3.

The progenitor is merely a copy of FMGR with the name FMGLU, where LU is the session terminal logical unit.

If no free ID segments exist: 1) issue an error, 2) continue at 5-1.

4-3. Move & Link the SCB

Move the SCB to free memory and link it into the SCB list headed at $SHED. This is accomplished by the MKSCB subroutine.

If no room or duplicate session identifier error: 1) issue an error, 2) remove the session progenitor, 3) continue at 5-1.

4-4. Post Log-On Information to Account File

Update the active session counter

Add an entry to the active session table as follows:

```
    Wd 1 = session identifier
wds 2-3 = log-on time
    wd 4 = Directory entry number
```

The format of the log-on time is:

```
WD1 = year(offset from 1978, 15-13) min (12-7) sec(6-0)
WD2 = day(13-5) hour(4-0).
```

4-5. Session Created and Account File Updated

Release the resource number lock to permit account file alteration.

Scan the list of mounted discs (FMP cartridge directory) looking for a match with this session's user and/or group ID's.

For every match found, call the FMP mount cartridge routine to make the disc addressable by this session.

4-6. Issue "LOG-ON COMPLETE" Messages

List system message file (defined in account file header record).

If the user has mail waiting, let him know about it.

4-7. If this is a non-interactive request, we are all done. Continue at 5.

4-8. Start-Up FMGLU

FMGLU is started by building the following string:

RU,FMGLU,name:security code:cartridge

where name:security code:cartridge is the user's HI (command) file namr.

This string is then sent to the MESSS routine, passing the SCB address in the second optional parameter.

The MESSS routine then causes FMGLU to become scheduled and tags it as belonging to the new session.

5-0. LOG-ON COMPLETE

Issue a "TIE-OFF" request to the message processor. This causes the user terminal to be enabled, if interactive. If non-interactive, the calling program is informed of the completion via a class write/read request.

5-1. Release the current log-on request and continue at 1.

6-0. This is a password response

Set up required flags (so message processor will operate if required) PARSE the password.

If password not supplied: issue an error and continue at 5-1.

Set the no-parse flag (NOPAR=1). Continue at 3-5.

6-1. Summary of Special Requests to the LOGON Processor

1. NON-INTERACTIVE log-on request. The log-on processor will communicate with other programs requesting a session creation. The communication is performed via a class number supplied by the caller. All messages normally returned to the session terminal are returned in this class number. Refer to the message processor section (MESSP) for details on returned messages and status flags. Once we determine that the request is not interactive, the communication class number (remember, this is not $LGON) is saved away and the log-on process continues the same as if the request was interactive.

   The special processing for non-interactive requests follows:

   a. Password (if required by user account) must be supplied or the request is rejected.

      NOTE: This restriction was requested by the Batch Processor and D.S.. The LOGON processor has the ability to request the password from the calling program and then continue with this request when it is returned.

   b. The session terminal is defined as the system console.

   c. A session progenitor is not created for non-interactive sessions.

2. PASSWORD reply. If a password is required but is not supplied, LOGON saves all the required information about the current process in the second buffer of a double-buffered class I/O write/read of the requested password. When the user completes the input of the password, both buffers are returned to the LOGON processor via the class get request (both the password buffer and the control buffer). This control buffer permits LOGON to continue processing on a previously initiated request. This processing merges back with the standard log-on processing at the point where the directory is searched for the specified user.

   NOTE: that we have already found this user once and the directory entry number could have been saved in the control buffer of the password read. This index would then directly point you at the requested directory entry. The reason for going back and searching the entire directory is that the directory may have been reworked or the user could have been purged between the point in time where the read of the password is initiated and completed.

3. SHUT DOWN REQUEST. The ACCTS program sometimes has the requirement of purging the account file. To do this, LOGON must close the file and then terminate.

4. BATCH (JOB) LOG-ON REQUEST. LOGON provides for the creation of a special session (ID=255) for the BATCH system. This request operates on directory entry numbers, not on a log-on string (see the discussion on the directory entry number request). After decoding the directory number and verifying that this user still exists, the standard log-on processing continues, treating the request as a standard non-interactive request with ID=255.

5. ACCOUNT NAME REQUEST. LOGON will translate a directory entry number into a user and group string (directory format) and return it to the calling program in a user supplied class #.

6. ACCOUNT DIRECTORY ENTRY #. LOGON will accept a user group and password string and scan the account file directory looking for a match. If a match is found, the directory entry number is returned in the user specified class number. This function is used by the JOB processor and the spool system as it allows the compression of a USER.GROUP/PASSWORD string into one word. This number is the count of directory entries scanned, counting from zero, until the requested entry is found.

LGOFF PROCESSING

All LGOFF processing is performed via class I/O. Once scheduled, LGOFF never terminates but is suspended on a class "GET" request, waiting for the next log-off request.

Calling Sequence

The log-off process is initiated by a copy of FMGR or the ACCTS program sending a log-off request in the class number defined by Table Area I entry point $LGOF. The two optional parameters (IOP1 & IOP2) of the Class I/O request define the type of request:

IOP1 >0 then:

IOP1= SESSION ID of session to be logged off in bits 0-8.

Bit 15 = 1 Dismount private discs
Bit 14 = 1 Dismount group discs
Bit 13 = 1 Kill active programs

IOP2= SCB pointer of the session logging off.

class buffer = possible communication class #.

IOP2 <0 then:

IOP2= -1 Shut down

or

= -length of second buffer (response to kill program PRMPT).

LGOFF FLOW

The log-off processor consists of three major sections: active program work; disc dismounting work; and account updating.

Before a user can be logged off all programs linked to the session must be aborted and possibly purged from the system. This is accomplished by scanning the list of ID segments looking for all session words (ID word 32) equal to the SCB address (SST length word) defined for this session. For all programs found, perform the following work:

1. Is the program D.RTR or SMP? If not continue with 2. Else, set a flag (OOPS) to indicate that D.RTR or SMP is currently active for this session. These programs must be allowed to clean themselves up as they may be modifying the disc. D.RTR is cleaned up via a session clean-up schedule request. This request is currently treated as a NOP (by D.RTR). Its function is to prevent the destruction of a session control block while D.RTR is still linked to it. SMP is cleared via a session clean-up request to it. This request will cause all pending spool LU's (files) associated with the session to be closed and printed (if required).

   NOTE: that these clean-up requests will be issued after all other currently active session programs have been aborted. Therefore, after our schedule request of these programs, there is no way they could be currently operating for our session.

2. Active program found and it is not D.RTR or SMP. Before we can abort this program, we must have permission from whoever requested the log-off. Permission may be passed in the request (bit 13 IOP1=2) or we have to prompt for and fetch permission. If permission has been given, continue at 4. Else, set a flag (FND=+1) to indicate that an active program exists, then return the name of the program to the caller (session terminal or class buffer). Continue searching ID segments at 1.

3. End of ID segments. If any programs have been aborted (FND<0), go back (1) and make another pass to verify that no one came in behind us. If any active programs were found and we did not have permission to kill (FND>0), prompt the caller and start the class read (on $LGOF) of the response.

   NOTE: this is a double buffered request with the following information saved in the control buffer:

   WORD 1 = Bit 15 = 1 if dismount private discs requested.

   Bit 14 = 1 if dismount group discs requested.

   Bits 8-0 = Session ID.

   WORD 2 = SCB pointer of the session logging off.

   WORD 3 = Possible communication class # (non-interactive).

   Go make next class "GET" for next log-off request.

   If no active programs were found (FND=0), continue at 5.

4. Active program found and we have permission to abort the program. Issue a "OF,PROG,1" request (via MESSS) to abort the program. Set FND=-1 to indicate that a program was aborted. Continue searching the list of ID segments (1).

5. At this point, all programs running for this session have been aborted. If D.RTR was found active to this session (see step 1 above), issue the session clean-up request to it.

6. Next, release all ID segments (longs only) allocated to this session. The list of ID segments is again scanned, this time checking the owner flag (bits 8-0 of word 31) with the session ID of this session. The owner flag is defined by LOADR or the "RP" FMGR command. It's value is set equal to the session ID of the requesting session. The exception to this is the MANAGER.SYS session. Programs loaded or RP'ed by this session have a zero for their session ID, allowing the system manager to leave programs in the system after he logs-off. For each match found (owner ID = SESSION ID) and the program is dormant, issue a "OF,PROG,8" command (via MESSS). If an ID segment was built for this session and someone else is using the program (remember, all programs related to this session must be dormant or we would not be here), give the ID to the session currently running the program. This is performed by changing the programs owner flag to reference the session it is currently active to. This completes the program management portion of LGOFF.

7. The next function performed is disc cartridge management. The log-off request defines which discs (if any) are to be removed. The dismount is performed by calling the FMP subroutine DCMC. The cartridge list provides the information (disc ID and LU) required to decide which discs are to be dismounted. Each disc which matches the specified private or group ID is removed from the system. Each disc removed has an informational message issued to the session to inform the caller that the disc was really removed.

8. Finally, the spool monitor program is called to clean up all spool files associated with this session and the following information is updated in the account file: active session block for this entry is cleared; active session counter is decremented; connect and CPU usage information is updated to both private and group entries. The session control block is now released (via RLSCB) and a tie-off request is sent to the LOGON/LGOFF message processor. This request will clear the disable bit in the session bit map and return a completion status to the caller if the request was not interactive.

9. Go get next log-off request.

LOGON/LGOFF MESSAGE PROCESSOR - MESSP

MESSP provides the diagnostic message interface for the LOGON and LGOFF processor. This routine will send a message to the session terminal and/or the system console. MESSP also provides a programmatic interface by returning diagnostics in a user supplied class number.

Calling Sequence:

```
Call MESSP (where,what,how much)

where = Bit 15    = Termination call
        Bit 12    = Don't append session # to message
        Bit 11-6  = ERROR code flag
        Bit 1     = Print on System console
        Bit 0     = Print on Session console

what  = message buffer address

how much = +word, -byte count of buffer
```

NOTES: If "where" is zero or has bit 15 set (and this is an interactive request), this routine will enable the terminal logging-on (off) by clearing the corresponding bit set back when the log-on (off) request was initiated.

The "where" parameter is passed to the calling program if the current request is non-interactive. Communication is via a class I/O write/read to the communication class number passed in the original call (not $LGON ($LGOF) but another class # supplied by the caller).

The status of the request passed in the class buffer is defined by the "where" parameter. This status is returned in the second optional parameter of the class write/read request.

IOP2=0 = Successful log-on/off, nothing in class buffer - last message.

>0 = LOGON/OFF message or diagnostic is in the class buffer. At least one more message will follow.

<0 = Terminating error is in class buffer. This is the last message for this log-on/off request.

Bits 11-6 contain the error code or 0 (if this is not an error). For example, LGON 06 error would have 000 110 in bits 11-6 of the where parameter.

The following special subroutines are used by the Session Terminal Handlers:

DTACH

Purpose: To remove a program from session.

NOTE: If the calling program is not a session program, this routine does nothing more than return.

Calling Sequence:

```
Call DTACH  \ removes prog from session by changing session word to
              contain - terminal LU of it's session.
```

or

```
Call DTACH(IDUMMY)  \ removes prog from session by changing session
                      word  to contain -1 (makes  it appear to have
                      been run from the system console).
```

In either case, the owner flag is changed to indicate that the system owns this ID.

CAPCK

Purpose: To provide command validation and capability level checking of RTE-IVB session operator commands.

Calling sequence: REG=CAPCK(IBUF,LEN,ISCB,ICAP)

Where:
- IBUF = command buffer to be checked.
- LEN = -bytes or + words
- ISCB = optional SCB address to be used.
- ICAP = optional capability level to check the command against.

returns:

(ok return)
- (A) = ASCII command
- (B) = parameter count
- (X) = address of parameter #1 in CAPCK's buffer.

(capability error)
- (A) = ASCII command
- (B) = -1
- (X) = addr of parm #1

(command undefined)
- (A) = -1
- (B) = parameter count
- (X) = addr of parm #1

$SALC + $SRTN

Purpose: To manage a predefined block of memory for use by the Session Monitor.

Calling Sequence

1. Allocate memory from session block:

```
(P)     JSB $SALC
(P+1)   (# of words needed)
(P+2)   -Return no memory ever (A)=-1,  (B)=max ever
(P+3)   -Return no memory now  (A)=0.   (B)=max now
(P+4)   -Return OK             (A)=addr (B)=size or size+1
```

2. Release buffer to session memory

```
(P)     JSB $SRTN
(P+1)   (FWA of buffer)
(P+2)   (# of words returned)
(P+3)   -return- (all registers destroyed)
```

If a request for a buffer of length X cannot be filled during a given call, return is made with:

(A) = 0

If, when buffer requested, - (AVMEM) - shows insufficient memory available to contain a buffer of the length requested, then return is made with:

(A) = -1
(B) = maximum length buffer that the program may allocate.

To find out how large a buffer may be allocated, use the call:

```
JSB $SALC
DEC 32767
```

Blocks of memory available for output buffering are linked through the first two words of each block -

WORD1 - length of block
WORD2 - address of next block (or 77777 if this is last block)

The allocator "transfers" the upper end of a block to caller and shortens the length of the block by the amount "transferred".

Registers are not preserved

To initialize the session memory block, the following call must be made.

```
(P)      JSB $SRTN
(P+1)    (-FWA of buffer)
(P+2)    (-# words in session block [one's complement])
(P+3)    -return- all registers modified
```

To release the block (indicate to session that no memory is available for use) make the same call as defined above. The status of the release call is returned in the (A) register.

A=0 Did not reset pointers as the # words specified did not match the current number of words available (i.e., active SBC still exists so do not return memory).

A=1 Did not reset pointers as the start address of the block did not match the current block address.

A=-1 Session pointers are reset to indicate no memory is available - OK to return memory to SAM.

MKSCB

Purpose: To move and link an SCB from the user map to the session available memory in the system map.

Calling Sequence: CALL MKSCB (IBUFA,IBUFL,IADDR,IERR)

IBUFA = Buffer address of SCB in user map

IBUFL = Length of SCB in user area (SCB in user space begins with word 3, the session identifier).

IADDR = Address of the SCB in session available memory (system map) is returned here.

```
IERR  = 0    OK - SCB created
      = 1    No memory now
      = 2    No memory ever
      = 3    Duplicate session identifier
```

RLSCB

Purpose: To remove an active SCB from the system.

Calling Sequence: CALL RLSCB (SESID,IERR)

SESID = Session identifier of SCB to be released.

IERR = 0 OK

= -1 SCB not found

Glossary of Terms Used
Session Monitor

System Logical Unit (LU) - A number ($1\text{-}255_{10}$) assigned at system generation to define an I/O device. This number appears on the left hand side of the session switch table and is used to reference a specific I/O device.

Session Logical Unit (LU) - A number ($1\text{-}63_{10}$) assigned:

1) during Account File setup

or

2) at log-on (via Configuration Table entries)

or

3) by the operator (via the SL command)

The Session Logical Unit provides a constant mapping of devices into a common set of user accessible logical units. For example, Session LU 1 always refers to your session console, regardless of which specific device (System LU) you are using.

Session Switch Table (SST) - A table which defines a session's total I/O addressing range. This table, placed in System Available Memory by the LOG-ON processor, consists of System LU's and Session LU's. For every session LU, there is a system LU which defines the actual device which may be referenced. This may be a direct map (Session LU25 and System LU25), or an indirect map (Session LU30 and System LU125).

NOTE: The only way a user program may access a System LU greater than 63_{10} is under session (via the SST).

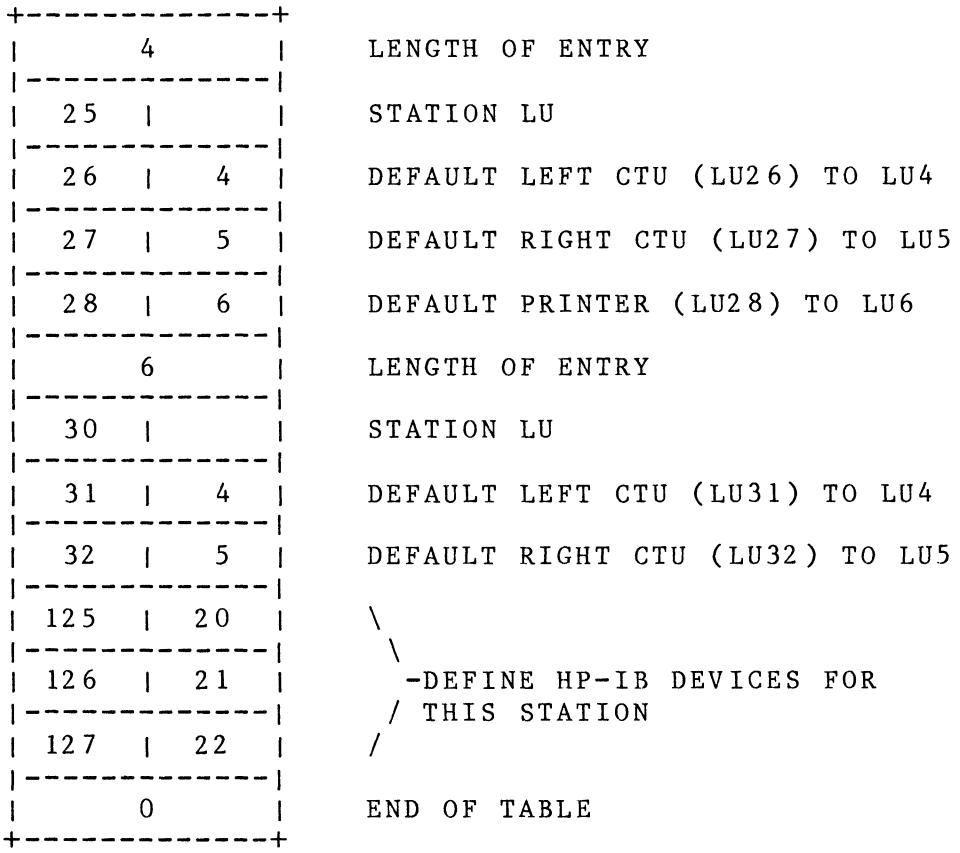
For example:

SYSTEM LU	SESSION LU	
-5		NEGATIVE LENGTH
41	1	SESSION CONSOLE DEFINITION
2	2	SYSTEM DISC
3	3	AUX SYSTEM DISC
42	42	"DIRECT MAP"
179	45	"INDIRECT MAP"

Station - A set of devices available to a session user. For example, a station might consist of a session terminal, cartridge tape units (2), and printer.

Configuration Table - A set of default logical units (added to your SST) defined for a specific station's devices. For example, this table would allow you to reference your terminals left cartridge tape unit as Session LU4, regardless of the actual System LU definition.

For example:



Session Identifier - This value is used in linking and identifying each specific session (word 3 of the SCB). The common contents is the system logical unit for the session console of the session. The session ID need not correspond to an interactive device, but it must be unique within the set of active sessions.

Session Control Block (SCB) - A variable-length table built by the log-on process for each session. It contains information unique to each session (i.e., session switch table).

Account file - A disc file, set up and maintained by the account set-up program (ACCTS). This file defines the following session information: Session Welcome File; active session information; Configuration Table; Disc Allocation Pool; Account File Directory; Group and User Account File Definitions.

Chapter 15

LOADER LIBRARY

INITIALIZING THE LIBRARY

The loader library is initialized by calling the L.INT routine as follows:

```
        JSB L.INT
        DEF RETRN
        DEF START    first word of free space
        DEF END      last word of free space
        DEF BPFWA    first word of real base page
        DEF COMAD    addr of sys com, 0 if local common
        DEF COMLG    length of sys com, 0 if local common
        DEF LOAD PT  load point of PROG
        DEF LWAPG    last point of PROG ADDR SPACE
        DEF TABLE    table of addresses containing subroutine
                     addresses required by the library.
```

From START and END the entry points SYM.L and FXS.L will be set up (refer to the appendix A). These two words define the beginning of the symbol table and the beginning fixup table respectively. Refer to appendix for the format of these two tables.

From the words LOAD PT,BPFWA and COMAD a three word relocation table, RBT.L is initialized. The table is reset at every NAM record and is used by the DBL processor to relocate code in referencing program area, base page, and common respectively.

COMLG and LWAPG are used in determining the loader library -6 and -3 error returns.

TABLE is the address of a three word table which contains the addresses of routines to be called by the library. The table is set up as follows:

1. ADDR of ALLOCATE BP LINK ROUTINE
2. ADDR OF SCAN for BP LINK ROUTINE
3. ADDR OF OUTPUT AN INSTRUCTION ROUTINE

Pointers used in the dummy current page link area are initialized also.

RECORD RELOCATION

Before a call is made to L.REL to relocate a record, the L.CLS routine is called to perform a checksum and to classify the record. The record is assumed to be in L.BUF, a 64 word BSS. L.CLS is called as follows:

```
    JSB L.CLS
    DEF RETRN  return address
    DEF TYPE   record type (1-7)
               -1 check sum error
               -2 illegal record

    DEF SBFLD  >0 subfield type
               0 record has no subfield or illegal record
```

If the record is a legal one, then processing is turned over to the appropriate record processor. These processors are described below. L.REL is called as follows:

```
    JSB L.REL
    DEF RETRN return address
    DEF FLAG  0 no error
             <0 error code
```

and branches to the appropriate record processor.

NAM RECORD PROCESSING

Upon encountering a NAM record the NAM record processor will be called. The first thing done will be to reset the relocation base table for this module. The table is:

```
    TH1.L      Program relocation base address
    BPR.L      Start base page addrress
    CAD.L      Common relocation base address
```

These are the base addresses that will be used by the DBL processor to relocate code. BPR.L is set up from the entry point CBP.L, TH1.L is set up from TH2.L, and CAD.L is set up at the first NAM record read and may never be changed.

The program length is set into PGL.L and the program type is set in PGT.L.

Also set up upon encountering a NAM record are the current page linking pointers. If current page linking is not possible these pointers are set to point to a zero length link area. The dummy current page link area is packed. All link areas that are above CPLS (the top of the link areas to save) and refer to an area on a page below the given page address are deleted. All zero length areas are also deleted.

Note that if local common is declared in the module to be relocated, the largest common declaration must be found in the first NAM record encountered. This is because the storage for local common is always at the beginning of the program. If local common is declared the address used will be found in CAD.L and the length declared in COM.L.

The NAM processor next clears NM1.L, the NAM has been read flag, this flag is checked by all other processors because the NAM record must be the first record read in any module and the first record found after every end record. The pointers to the last entry +1 into the symbol table, LSY.L and base page, CBP.L are saved locally in case the scan mode is in effect and no modules are loaded. If scan mode is in effect, i.e., LBS.L flag is set to -1, then the IGN.L flag is set to -1. If an external is satisfied by the ENT processor during the scan IGN.L is set to 0. When DBL records are encountered with IGN.L =-1 all processing is skipped, that is, ignore this module as it satisfies no undefs.

Next, the module name is saved at NM2.L (the first word is the number of words) and the extended NAM record field is saved at NM3.L (first word is # of words) and a 7 word buffer containing priority, resolution code, execution multiple, hours, minutes, seconds and 10's of milliseconds is saved at NM4.L. Lastly, any base page allocation declared in the NAM is set aside by calling the main's base page allocation routine.

It is also important to note what the NAM record processor does not do. It does not process records on the basis of type, that is, nothing special is done for segment NAM records. It is the loader writer's responsibility to trap segment NAMS, if he so desires, through the L.CLS routine.

ENT RECORD PROCESSOR

ENT records define the address of the referenced symbol. They are always found after NAM records and before EXT records no matter where the ENT is declared. The language develops this construct and the loader enforces it.

Whenever an entry point is processed whether scanning or relocating, the address of the symbol is calculated and then a search of the existing symbol table is made. Refer to appendix for symbol table format. If the symbol is not in the table then it is placed into the table and set as defined (S field = 0 or 1). If it turns out that a scan mode was in effect and the ENT was not useful (i.e. did not satisfy an undef) then the entry in the symbol table will be wiped during END record processing.

If the search of the symbol table finds the entry, then there are four possible conditions to be accounted for as shown in the table below:

	SYMBOL IS ALREADY DEFINED	SYMBOL IS UNDEFINED (i.e. PROCESSED EARLIER AS EXT)
SCAN MODE IN EFFECT LBS.L =-1	IGNORE THIS ENTRY IT HAS ALREADY BEEN CORRECTLY PROCESSED	SET TYPE FROM ENT WORD 6 INTO LST4. SET UP S FIELD =0. PLACE DEFINING ADDRESS INTO LST5. GO DO ALL FIXUPS.
RELOCATE MODE IN EFFECT LBS.L =0	DUPLICATE ENTRY POINT. HE TRIED TO LOAD THIS TWICE, ABORT THE LOAD. RETURN TO L.REL WITH ERROR CODE OF -7. RESET LSY.L POINTER INTO LST.	SET TYPE FROM ENT WORD 6 INTO LST4. SET UP S FIELD =1. PLACE DEFINING ADDRESS INTO LST5. GO DO FIXUPS.

The defining address for a program relocatable ENT is calculated by adding TH1.L+ word 7 of the ENT record.

This address is placed into LST5 of the symbol table and all references to the symbol via EXT records will be directed to that address.

If the symbol was found as undefined in the LST (loader symbol table), then there may already have been references to it.

Therefore, the fix up processor is called to find all previous references to the symbol and to correctly output the reference now that the defining address has been calculated. Fix up processing will be described later.

Two important flags are also set up in the ENT record processor during a scan. They are IGN.L and NOR.L. Recall that in the NAM record processor IGN.L was set to -1 if the scan mode was in effect. If an ENT is found that matches an (EXT) undefined symbol in the LST then IGN.L is set to 0. This will be a flag to the DBL processor to process this record. In addition NOR.L will be set to 0. NOR.L is set to -1 before entry to L.REL and thus if the user wishes to know if any subroutines were loaded during the last scan NOR.L will give him this information, that is, -1/0 no/yes a routine was loaded. This flag is typically used in deciding whether a file should be rescanned to look for more undefs. If a file is scanned over and over until a scan is completed and no routines were loaded then that file is no longer useful in resolving undefined externals and the next file should be looked at. This method of continually rescanning the same file is the only way subroutine backward references can be satisfied.

EXT RECORD PROCESSOR

EXT records are found after ENT records in a relocatable module. EXT records are the means by which one module references another module. Each EXT has an ordinal number assigned to it in the lower byte of word 6 of the particular EXT record. Any references to the EXT via a DBL record uses the EXT ordinal number. These references would be type 4 DBL records.

EXT processing is actually rather simple. First a search of the symbol table is made. If the symbol is found in the LST then regardless of mode (scan or relocate) the ordinal number is taken out of word 6 of the EXT record and placed into the lower byte of LST3. All future DBL records in this module will reference the EXT via the ordinal number.

If the symbol is not found during the LST search then the EXT record and ordinal number are entered at the end of the symbol table. All of LST4 is also set to 2, that is, the symbol is undefined.

Note that if the scan mode is in effect and this modules ENT records do not satisfy any undefs then this modules ENT's and EXT's will be removed from the symbol table during END record processing.

DBL RECORD PROCESSING

Data Block Records (DBL) are the records where the actual program instructions are stored. Since the record format is complex some of the fields are described below.

The Z/C field (word 2, bits 7 and 6) gives the type of DBL record. Currently only two types are legal. They are base page (0) and program (1). This field tells whether the code to follow is to reside on base page (ORB instruction) or in the program area.

Word 2 bits 5-0 gives the number of instructions contained in the DBL record. Word 4 gives the unrelocated load address of the first instruction word contained in this record. Thus, for example, if the first instruction was a LDA and word 4 was 3000B, and the load address at the NAM record (TH1.L) was 46000B, then the LDA would be relocated to 51000B.

Word 5 has five identical three bit fields. Each field describes a program instruction. Bits 15-13 describe the first word, bits 12-10 describe the second word and so on. Seven DBL types are possible. All types describe one word references except DBL types 5 and 6 which are the memory reference and external reference with offset type and byte reference. Thus if all five types in word 5 were memory reference 10 words would follow and then the next five field DBL type word would be found. If all five types were type 1 (15 bit relocatable), then only five words would follow and then the next five field DBL type word would be found.

A type 4 DBL word is an external reference instruction. Recall that when the EXT record was processed, an ordinal number was taken from the EXT record and placed in the symbol table. Bits 7-0 of the external reference instruction word has that ordinal number. Thus external references reference the ordinal number and the ordinal number is in the symbol table with the defining address. Type 5 records, memory reference, are also used for external reference with offset instructions. Bits 9-2 have the ordinal number and the second word has the offset.

DBL types 1,2, and 3 are all 15 bit relocatable references and are easily relocated. The word out of the DBL record is just added to TH1.L, BPR.L, or CAD.L respectively and the word is output. Likewise, type 0, absolute, is also easy to relocate. No processing is done; it is just output.

The type 4 DBL record is the external reference record; the type 5 DBL record is also an external reference if the ordinal field is non zero. In both cases the DBLEX routine configures the instruction. For type 4, the the offset is 0, for type 5 the offset is whatever is found in the second word of the instruction.

The DBLEX routine scans the symbol table for a matching ordinal to the ordinal in the instruction. A match must be found or the assembler made an error. When the match is found the symbol is either defined or undefined. If the symbol is defined, then the FIXIT routine is called to configure the instruction. If the symbol is still undefined a fixup is built for the instruction.

A fixup is a 4 word entity that is built and put into the fixup table such that when the symbol is defined all prior references to it may be output. The four word table is defined in the appendix. Note that word 2 of the fixup table entry is the symbol table address. This has definite implications for future constructs. That is, no symbol table entry should be paged out and another put in its address because the fixup would point to two different symbols. If memory becomes a problem, the fixup table should go to the disc first. If the symbol table entry does go to the disc, then fixups pointing to it must be changed to reflect disc addresses.

Before the new fixup entry is added a check is made to insure that symbol table area is not being overlayed. If so, a -4 error is returned to L.REL. As a future enhancement fixup table entries could be pushed to the disc at this point.

The FIXIT subroutine is called by DBLEX if the symbol is defined. If the symbol is an ENT type 4 (microcode), then it is at this point that the RP replaces the instruction. For all other types an instruction is configured.

The instruction is configured by looking at the address of the instruction LDA, for example, and the target address of the instruction. If both are on the same page no link is needed. If the instruction address and target address are on different pages then a link is required. The contents of the link will be the target address or target address indirect. Consider the example below:

INSTRUCTION ADDRESS (8)	INSTRUCTION	LOADER OUTPUT	OCTAL
TARGET 4060	X DEC 17	DEC 17	21
SAME PAGE 5001	LDA X	LDA X	62060
5002	LDA X, I	LDA X,I	162060
DIFFERENT PAGE			
6001	LDA X	LDA BP1, I	160003
		@ BP1 DEF X	@3 4060
6002	LDA X, I	LDA BP2, I	160004
		@ BP2 DEF X,I	@4 104060
DIFFERENT PAGE (current page link available			
6001	LDA X	LDA CP1, I	163124
		@ CP1 DEF X	@ 7124 4060
6002	LDA X, I	LDA CP2, I	163125
		@ CP2 DEF X,I	@ 7125 104060

When FIXIT discovers that a link is necesssary, it first scans the base page to see if one is already allocated with the correct value, if not, it scans the links on the current page. If not found there, it tries to allocate a link on the current page and if that is not possible, it allocates a base page link.

When the instruction is configured, the absolute output routine is called to output the word. The absolute output routine is given the configured instruction and the program address at which the instruction is to be placed.

FIXIT is also called by a routine called FIXAL. FIXAL is called for every ENT record processed where the symbol name has already been entered into the LST due to being a EXT reference earlier. That is, every time a symbol is defined FIXAL is called to fix up any earlier references to the symbol. Recall that those references are contained in the fixup table.

FIXAL scans the entire fixup table and compares the 2nd word of the fixup table entry to the given 1st word symbol table address. If there is a match then this entry is a reference to the symbol encountered before the symbol was defined and this must be fixed up. FIXAL calls FIXIT to output the instruction.

FIXAL is called from other routines as well. Anytime a symbol is defined, FIXAL must be called. Thus the L.SYE and L.MAT routines also call FIXAL (and post a current page link if one was created). The L.IFX routine also calls FIXAL for forced loads where undefined externals remain in a segment of a segmented program. In this case, the entry will be defined by a HLTO to force memory protect errors. (Again, any current page links created are posted).

FIXAL sets the first word of every entry in the fixup table that was fixed to a -1 and at the end of all the fix ups; it packs the table to insure that the table is always as small as possible. Note that if in later enhancements the fixup table is pushed to the disc, FIXAL must be changed to search disc and memory for fixup entries and to pack disc and memory at the completion of processing.

As mentioned earlier, the type 5 DBL format is two words long. It is used for external reference with offset instructions and memory reference instructions. If the instruction is memory reference (ordinal # =0) then the MREF subroutine is called to configure the instruction. This routine is used most often as memory reference instructions are far more common than external reference instructions.

MREF basically does the same job as FIXIT except that there is no fixup table or external references to worry about. MREF calculates the instruction address and target address, looks to see if they are on the same page, calls the users base page scan, the library's current page link area scan, the library's current page link allocate and the user's base page allocate routines if necessary, and configures the instruction. After the return, the DBL processor calls the absolute output routine to pass the configured instruction to the user.

The last DBL word type is the byte reference. This DBL type also uses two words. The type field currently is always zero. The MR field is used to determine the relocation base to be used for relocation (i.e. the TH1.L, BPR.L, CAD.L table). The address from this table is multiplied by two to get a byte address and then added to the second word of the two word byte format. This is the configured instruction. As might be guessed, this is merely a DEF to a byte address.

END RECORD PROCESSING

Not much processing is done for END records by the library. However, quite a bit of processing will most likely be done by the user. Checks for DBUG, segmentation, library scan or load, and was the module loaded (NOR.L flag) are all user responsibilities.

The library checks to see if the routine was loaded or ignored. If it was ignored, the current load address, current page link, base page and symbol table pointers saved locally during the NAM record processing are set back to the point where the NAM record was encountered. If the module was loaded a check for any common overflow is made. If a common allocation error is made, a -6 error is returned to L.REL. Lastly, all external reference ordinal numbers in the entire loader symbol table are cleared. Any current page links allocated for the module that just ended are written to the disk.

Note that primary entry point information is fetched and the address of the primary entry point of the program is placed into PRI.L. It is the user's responsibility to decide which END record primary point to use as the start address of a program. PRI.L will be overlayed everytime an END record with a primary entry point is encountered.

EMA RECORD PROCESSING

EMA records are processed like ENT records except that no previous EMA declarations may have been made and the record must not appear in a segment (-10 error). SEG.L is tested. If it equals two then a segment is being loaded. If it equals one, the program is segmented but the main is being loaded. If it equals zero, the program is not segmented. If no errors exist, the symbol name goes into the symbol table and a base page link is immediately allocated. The V bit is set in the table and the base page address is placed into the last word of the LST entry. All future references are treated as external references and are sent through the allocated link. The link address used will be placed into EMA.L. Note that it is the user's responsibility to fill in the defining address for this link at the end of the load. The MSEG size will be placed into MSG.L. The EMA size is placed in EMS.L.

LOD AND GEN RECORD PROCESSING

LOD and GEN records contain information specific to a loader or generator. The loader library classifies these records, but L.REL just ignores them with the assumption that the specific loader or generator will do the correct processing.

The Dummy Current Page Link Area

A dummy current page link area is used to avoid disk accesses. The dummy area can be searched for existing links instead of having to search on disk. A dummy area entry is composed of a four word header followed by the link area image. Header word 1 is the start address of the actual current page link area. Header word 2 is the address of the last word allocated +1 of the actual current page link area. Header word 3 is the address of the last word +1 for which dummy area is set aside. For a post link area (a link area following a module) word 2 and word 3 are equal. For a prelink (a link area preceding a module) word 2, is always less than or equal to word 3. Header word 4 is the address of the dummy image of the area, and the image follows the four word definition of the area.

Two utility routines, LNK and LNKS are used to access the dummy current page link area. Given the address of header word 1 in the A register, LNKS sets up LNK1-LNK4 to point to header word 1 through header word 4. Given that LNK1-LNK4 is already set up, LNK takes those values to set up LNK1-LNK4 for the next dummy entry.

NAM Record Processing

When a NAM record is encountered in the process of relocating, CCPLK is called to pack the dummy current page link area to get rid of link areas which are no longer active. The A register is set to the current relocate address. CPLS is set to the top of the fixed current page link area. All link areas that are above CPLS and refer to an area on a page below the page address in the A register are deleted. Zero length areas are also deleted.

GETCP is called to set up the next current page link area. LNK1-LNK4 are set to point to word 1-word 4 of the new area's 4 word header (the next 4 words of the dummy current page link area).

SETPL is called to set up the new area to point to the prelink area (links preceding a module) for the new module. If the module fits on the current page, no links are set aside. If it doesn't fit, the number of links set aside is calculated to be:

$$\frac{\min (x/2 , Y)}{16}$$

CURRENT PAGE LINKING

Where X is the number of words of the module on the current page and Y is the number of words of the module that fall beyond the current page (note that this value may be zero).

PLEND is set to the address of the last word available for prelinks +1 (program load address) and CPL1 is set to point to the first word of the four word header for the current prelink area.

CLRCP is called to zero out the links that have been set aside in the dummy area.

GETCP is called again to set up an area to be used for the new module's post links (links following a module).

SETCP is called to calculate the length of the module so that the actual current page link addresses can be set up. If the module ends on a page boundary, then no links are available. Otherwise, the area left on the page can be used. CPEND is set to the address of the last word available for post links +1 (beginning of next page) and CPL2 is set to point to the first word of the four word header of the current post link area.

CPL1H and CPL2H, the number of links assigned in the current prelink and postlink areas, are set to zero.

DBL Record Processing

Both DBL external references and memory reference instructions, may possibly need links. If the EXT referenced is not on the same page as the instruction, or if the memory reference is outside the current page, an indirect link through base page, or current page if possible, is set up. First base page is scanned to see if a link exists already. If not, then all links on the current page are scanned using the routine SCNCP to scan the dummy current page link area. If no link is found, then ALLCP is called to try and allocate a link on the current page. If that is not possible, a base page link is allocated. If there are no more base page links, an OV BSE base page overflow error occurs.

SCNCP looks through the dummy current page link area to see if the needed link has already been allocated and is usable by the current instruction address, OPPAG must be set up to be the page number of the instruction address and OPRND must be set to the operand to scan for.

SCNCP looks at each 4 word header to see if the links in that area are on the same page as the instruction being processed. If the page is the same, the area is scanned for the required link, otherwise the next 4 word header is set up to be checked.

If the link is found, the actual address to use is returned in the B register. If no link is found, the B register contains a -1.

ALLCP checks to see if the instruction is on the same page as the next available prelink area or postlink area link. If it is, a check is made to see if there is room in the dummy area to allocate another link.

If there is, a link is set up and header word 2 for the area (address of word allocated +1) is incremented. CPL1H or CPL2H is incremented depending on if a prelink or a postlink was allocated. And, if a postlink was allocated header word 3 is incremented since the dummy area for postlink areas is allocated dynamically. The dummy current page link address is returned in the A register and the real current page link address is returned in the B register. If no link was allocated, both registers are set to -1.

END Record Processing

Upon encountering an END record, the pre and post current page links for the module that just ended, are written to the disk using the PSTCP routine. The values are taken from the dummmy area and written to the real current page link locations. PSTCP is called twice, once with the A-register set to CPL1 to write out the prelink area, and once with the A-register set to CPL2 to write out the postlink area.

On a force load, current page links may also be created and posted to the disk in the process of doing fix-ups.

Fixing-up previous references to a memory resident, an absolute, or an RPL microcode symbol can also cause current page links to be created and posted.

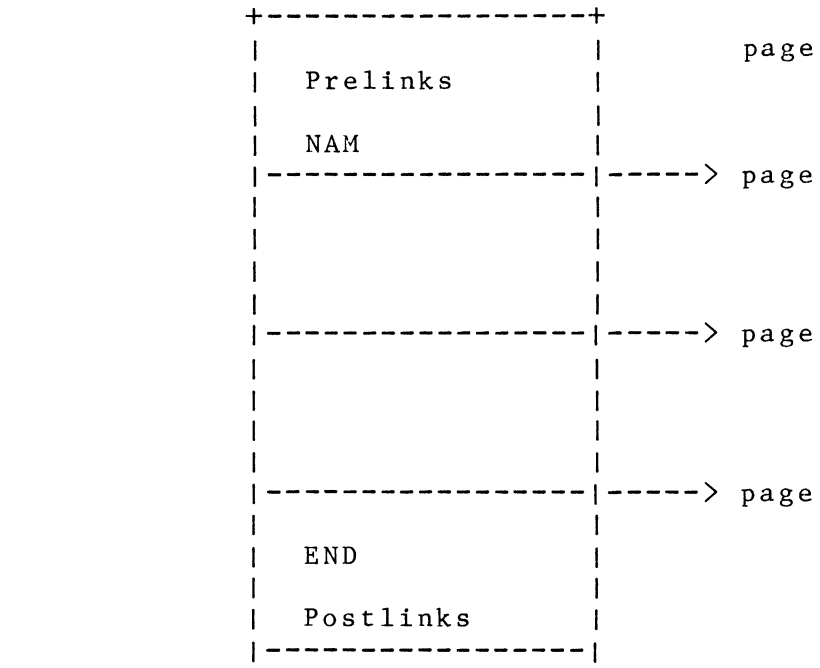
Special Processing for Segments

L.SGO sets up conditions for all future segments. It must be called once at the end of loading the main and before loading the first segment. One of the things it does, is pack the current page link area, by calling CCPLK, then it saves CPL2, the highest valid current page link area, in CPLS, the top of the images to keep. Thus, valid link areas associated with the main will be kept for all future segments to use.

L.SGN set up conditions for all future segments. It takes the value of CPLS and uses it to reset CPL2. In this way, any links created for a particular segment are no longer accessible (which is as it should be) when another segment is encountered.

CURRENT PAGE LINKING
To Optimize Current Page Linking

The key to optimizing current page linking is that the instruction and link must be on the same page. Potentially, a module many pages long will not be able to use many current page links. Consider:



All the code on the middle two pages cannot use any current page links at all, since links are only allocated in front of and behind a module. If the code were broken into smaller modules, more current page links could be allocated.

Relocating a number of small modules together if they share common EXT's will mean that they will also share the same current page links. For example, an external reference to .ENTR might cause a current page link to be created. If the next module is on the same page and again references .ENTR, it can use the link already created by the first.

In general, writing smaller modules (less than a page) will allow more code to use current page links. Also, if modules which reference common externals are loaded on the same page, they can share links. And finally, if modules which reference each other are loaded on the same page, no links at all will be required for those references.

SYMBOL TABLE ACCESS

A number of routines are available in the loader library that provide a main program access to the symbol table. TSY.L is used to point to the current temporary symbol table entry.

L.ADD

L.ADD retrieves the value of a symbol from its entry in the symbol table. If the value in the symbol table is a base page link address, the actual address is retrieved and returned (this will be the case for an EMA array name). The L.ADD routine is called as follows;

```
JSB L.ADD

DEF RETRN          return address

DEF ASYMA          address of symbol name array

DEF VALUE          retrieved value of symbol

DEF SADDER         retrieved symbol table address

DEF RESLT          0 = symbol found and defined
                   1 = symbol not found
                   2 = symbol found and undefined
```

If the symbol is not found, VALUE and SADDER are not defined. If the symbol is found but not defined, VALUE is not defined.

L.LDF

L.LDF may be called repeatedly to list all defined entries in the symbol table. On first call, the user sets 'PNTR' to zero and the symbol table search is done from the first entry. The address of the first defined symbol is returned in the 'ADDRS' parameter. On following calls to L.LDF, the unmodified 'PNTR' parameter is used to begin the next symbol table scan. When the end of the symbol table is reached, 'ADDRS' is returned with a value of zero.

L.LDF marks entries after they are listed. Parameter IGND gives the user the option of listing previously listed entries a second time or not listing entries which have been listed already. The L.LDF routine is called as follows;

```
JSB L.LDF

DEF RETRN          return address

DEF ADDRS          address of symbol name B-reg

DEF PNTR           address to be carried

DEF IGND           -1 do not relist marked entries
                   1 relist all entries
```

L.LUN

L.LUN enables the user to list the current main or segment's undefined externals or the undefined externals in the entire symbol table. L.LUN is called repeatedly with 'PNTR' preserved. On first call the user sets 'PNTR' to zero and the symbol table search is done from either the first entry of the symbol table or the first entry for the current module. On return, 'ADDRS' points to the symbol name array of the next undefined external or has the value zero if the end of the symbol table was reached. The L.LUN routine is called as follows:

```
JSB L.LUN

DEF RETRN          return address

DEF ADDRS          address of symbol name B-reg

DEF PNTR           address to be carried

DEF MNSEG          0 = scan current module's, entries only (main or segment)
                   1 = scan entry symbol table
```

L.SYE

L.SYE is used to enter or update an entry in the symbol table and set the type and value of the symbol. The table below shows the possible actions taken by L.SYE under various conditions:

current state	user requested state: undefined	user requested state: defined
not found	New entry added error = 0, -5	New entry added error = 0, -5
symbol undefined	No action error = 0	Symbol defined fixups processed error = 0
symbol defined	If user wants override symbol undefined else no action error = 0	If user wants override symbol redefined else no action error = 0

The L.SYE routine is called as follows:

```
JSB L.SYE

DEF RETRN          return address

DEF SYMBL          symbol name array

DEF TYPE           type of symbol entry

DEF VALUE          value of symbol

DEF OVERR          >=0, do not override current definition
                   < 0 override current definition

DEF ERROR          0 no error
                   -5 symbol table overflow
```

L.IFX

L.IFX is called to clean up the fixup table if a segment is to be force loaded. The symbol table is searched for undefined externals starting from the first entry for the current segment. If an undefined external is found, its type is set to 4 and its value is set to zero. Any previous references are fixed. The search continues until the end of the table is reached at which point any current page links created are posted. The L.IFX routine is called as follows:

```
JSB L.IFX
DEF RETRN          return address
```

L.MAT

L.MAT fixes up previous references to a memory resident, absolute or RPL microcode symbol. The symbol table is searched for the symbol (ASYMB). If it is found and is undefined, the symbol table entry type and value are set. All previous references are fixed and any current page links created are posted. The L.MAT routine is called as follows:

```
JSB L.MAT

DEF RETRN          return address

DEF ASYMB          symbol name array

DEF TYPE           symbol type

DEF VALUE          symbol value

DEF RESLT          0 used
                   1 not used
```

RESLT is return with a value of 1 if the symbol was not found, or if it was found but already defined.

SETTING SEGMENT CONDITIONS

Two routines are available for setting conditions for segments.

L.SG0

L.SG0 sets up conditions for all future segments. It should be called once at the end of loading the main and before loading the first segment. The current base page location, CBP.L is saved as the first segment base page location, SGB.L. The next available word of the symbol table, LSY.L is saved as the first symbol table entry for the segment, SSG.L. The high address +1, TH2.L, is saved as the segment base address, SGM.L. The current page link dummy area is packed and the last area associated with the main is saved as the last area to save when resetting for new segments. The L.SG0 routine is called as follows:

```
JSB L.SG0

DEF RETRN          return address

DEF DUMMY          0 is for now
                   reserved for future use
```

L.SGN

L.SGN sets up conditions for all segments. L.SG0 must have been called previous to the call to L.SGN. L.SGN resets the proper base page location for a segment, the first segment symbol table entry, the last valid current page link area and the segment base address. This routine should be called before relocation of every segment. The L.SGN routine is called as follows:

```
JSB L.SGN

DEF RETRN          return address

DEF DUMMY          0 for now reserved
```

L.FLG - LOADER LIBRARY GLOBAL FLAG AND POINTER VARIABLES

Module L.FLG contains the loader global flag and pointer variables and the relocatable input buffer. The pointers and flags are divided into four basic areas, control flags which determine flow of control and load options, table pointers into the symbol table and fixup table, program relocation base addresses and lengths, and pointers into the NAM record information.

I. CONTROL FLAGS 0/-1

```
LBS.L NOP          0/1, NO/YES SCAN THIS MODULE FOR UNDEFS.
                               SET BY USER BEFORE CALLING L.REL.
NOR.L NOP          -1/0, NO/YES A SUBROUTINE WAS LOADED IN THIS SCAN.
                               SHOULD BE RESET TO -1 AT EVERY ENTRY TO
                               L.REL.  SET BY LIBRARY.
NM1.L NOP          -1/0, YES/NO NAM MUST BE FIRST.  SET TO -1 AT EVERY END
                               RECORD AND CLEARED AT EVERY NAM RECORD
                               BY THE LIBRARY.
IGN.L NOP          -1/0, IGNORE/DON'T IGNORE THIS MODULE.  SET BY LIBRARY
                               WHEN IN SCAN MODE.  IF ENTS DON'T MATCH
                               EXISTING UNDEFS, THE MODULE IS IGNORED.
CPL.L NOP          0/1, DON'T USE/USE CURRENT PAGE LINKING.  SET BY USER.
SEG.L NOP          0/1/2, PROGRAM NOT SEGMENTED/SEGMENTED AND LOADING MAIN/
                               SEGMENTED AND LOADING SEGMENT.
EBP.L NOP          0/-1 NO/YES EXTERNALS FORCED TO USE BASE PAGE LINKS.
```

II. TABLE POINTERS

```
FXN.L NOP          WORD 1 OF LAST ENTRY IN FIXUP TABLE.
FXS.L NOP          FIRST WORD AVAILABLE OF FIXUP TABLE.
SYM.L NOP          ADDRESS OF FIRST WORD OF LOADER LIBRARY SYMBOL TABLE.
LSY.L NOP          ADDRESS OF LAST WORD +1 OF SYMBOL TABLE.
TSY.L NOP          CURRENT TEMPORARY SYMBOL TABLE ENTRY.
SSG.L NOP          FIRST SYMBOL ENTRY FOR CURRENT SEGMENT.
```

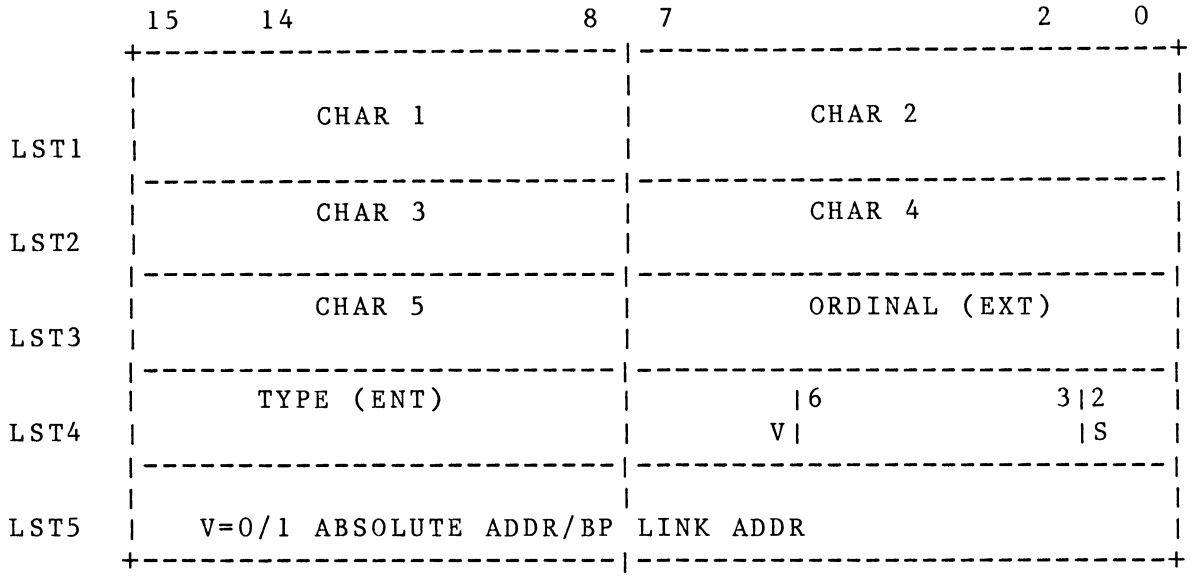
III. NAM RECORD POINTERS

```
EMS.L NOP          EMA SIZE DECLARED IN PROGRAM.
MSG.L NOP          EMA MSEG SIZE IN NUMBER OF PAGES (MINIMUM OF TWO IF EMA
                               DECLARED).
NM3.L DEF  *+1     ADDRESS OF EXTENDED NAM COMMENT. FIRST WORD IS NUMBER
      NOP                      OF WORDS.
      BSS 43
NM4.L DEF  *+1     ADDRESS OF PROGRAM INFORMATION OF THE NAM RECORD.
      BSS 7                    THIS A SEVEN WORD ARRAY RESET FOR EACH NAM
                               CONTAINING THE PRIORITY, RESOLUTION CODE,
                               EXECUTION MULTIPLE, HOURS, MINUTES, SECONDS
                               AND TENS OF MILLISECONDS.
NM2.L DEF  *+1     ADDRESS OF MODULE NAME, NUMBER OF WORDS FOLLOWED BY
      BSS 4                    NAME.
PGT.L NOP          CURRENT MODULE TYPE FROM NAM RECORD.  SET BY LIBRARY.
                               NOTE THIS IS TO THE PROGRAM TYPE THAT WOULD
                               GO INTO AN ID SEGMENT.
PGL.L NOP          PROGRAM LENGTH OUT OF NAM RECORD
                               IF -1, WAS FTN2 OR ALGOL
PRI.L NOP          PRIMARY ENTRY POINT ADDRESS OF THIS MODULE.
                               EQUALS ZERO IF NO PRIMARY ENTRY POINT.
```

IV. PROGRAM RELOCATION ADDRESSES, LENGTHS

```
CBP.L NOP          ADDRESS OF NEXT AVAILABLE BASE PAGE LINK.
                                 SET BY USER.
COM.L NOP          LENGTH OF LOCAL COMMON, = 0 IF NO COMMON DE-
                                 CLARED, OR SYSTEM COMMON BEING USED.
EMA.L NOP          BASE PAGE LINK ADDRESS FOR EMA ARRAYS.
TH2.L NOP          HIGH ADDRESS OF THIS LOAD.  SET BY LIBRARY.
RBT.L DEF   *+1    ADDRESS OF RELOCATION BASE TABLE.
      NOP
TH1.L NOP          BASE ADDRESS OF THIS MODULE.  SET BY LIBRARY
BPR.L NOP          BASE PAGE RELOCATION BASE ADDRESS.
CAD.L NOP          COMMON RELOCATION BASE ADDRESS.
      NOP          ABSOLUTE BASE ADDRESS.
SGM.L NOP          SEGMENT BASE ADDRESS.
SGB.L NOP          SEGMENT BASE PAGE BASE ADDRESS.
L.BUF BSS  64      RELOCATABLE INPUT BUFFER
```

LOADER SYMBOL TABLE



WORD 1: Characters 1 and 2 of symbol name
WORD 2: Characters 3 and 4 of symbol name

WORD 3: Character 5 of symbol name Bits 7-0 is the ordinal number of the symbol if it is currently being referenced as an external. The ordinal comes from the last byte of word 6 of the EXT record.

WORD 4: Bits 10-8 is the entry point type. When the entry is defined by the ENT record its type is set in the type field as it is received from bits 2-0 of word 6 of the ENT record.

Bit 7-the V bit is used to specify the contents of LST5. If V=0 LST5 is an absolute address, RP value, or memory address. If V=1 LST5 is a base page link address that may be used to reference the ENT or EXT. This is generally used when the EXT can not be defined until the end of the load. EMA is an example of this.

Bits 2-0-the S field indicates the status of the symbol

S=0, ENT read during a library scan i.e. defined.

S=1, ENT read during a load operation i.e. defined.

S=2, EXT ENTRY symbol is still undefined.

S=3, EMA entry. The symbol is considered to be defined. V will = 1.

WORD 5: If V=1, the contents will be a base page link address to use when referencing the EMA variable. If V=0, word 5 is the defining address of the ENT.

LOADER FIXUP TABLE

Each entry in the fixup table is four words long as follows:

1) Memory address to be fixed (-1 indicates an empty entry).

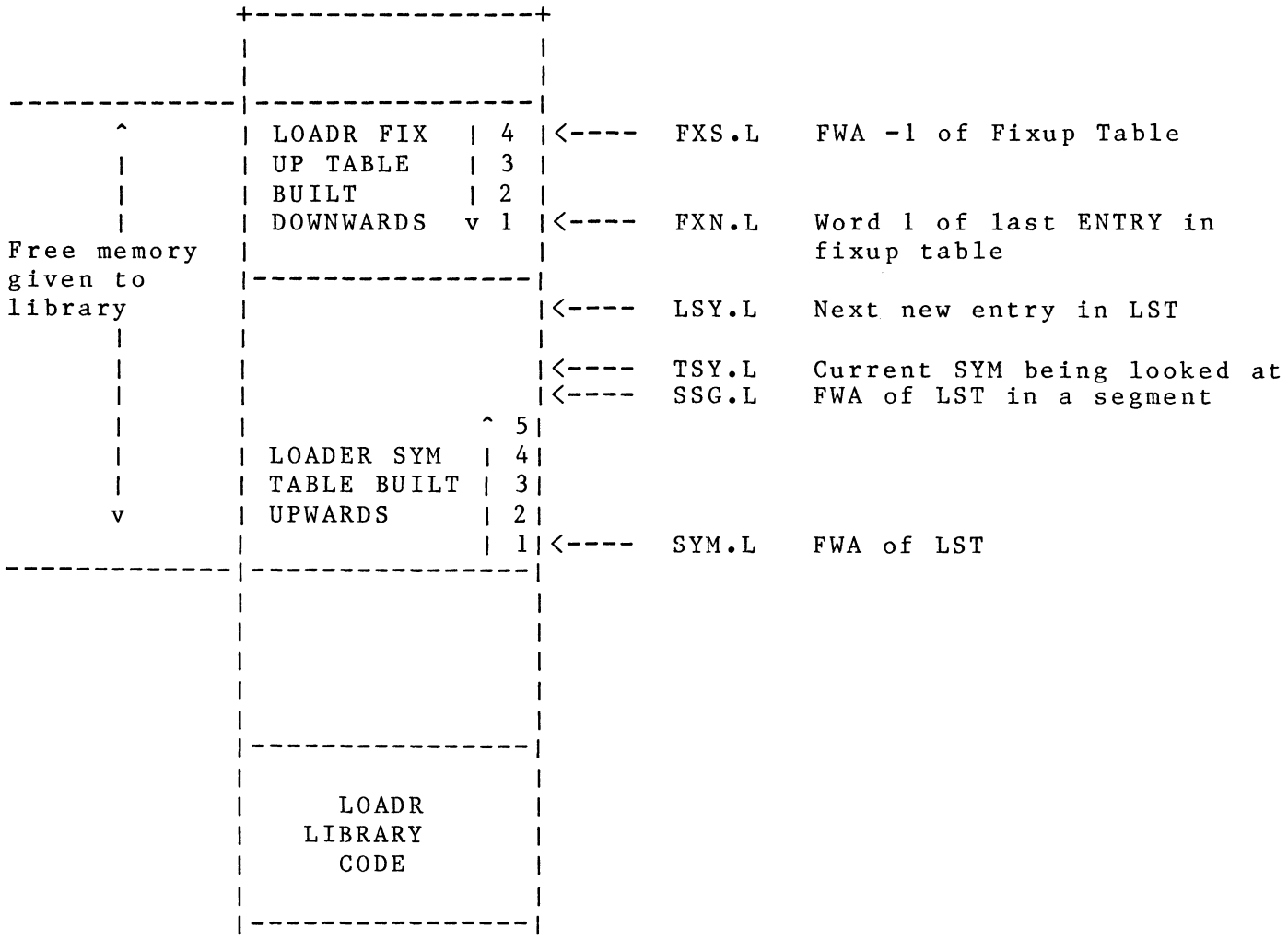
2) Symbol table address of the EXT for this instruction (thus symbol table entries should not be paged out although fixup table entries could be).

3) Bits 15-11 op code from DBL record bits 2-0 DBL type (relocation indicator).

   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

4) Offset from DBL record i.e. LDA $XYZ+15
   Offset=15 used for externals with offset

LOADER MEMORY ORGANIZATION



ERROR CODE RETURNS
(FROM THE L.REL ROUTINE)

-1 CHECKSUM ERROR

-2 ILLEGAL RECORD

-3 MEMORY OVERFLOW (prog too large)

-4 FIXUP TABLE OVERFLOW

-5 SYMBOL TABLE OVERFLOW

-6 COMMON BLOCK ERROR

-7 DUPLICATE ENTRY POINT

-8 RECORD OUT OF SEQUENCE

-9 ASMB PRODUCED ILLEGAL RELOCATABLE

-10 EMA DECLARED TWICE OR IN SEGMENT

-11 ATTEMPT TO REFERENCE EMA EXTERNAL WITH OFFSET OR INDIRECT

Chapter 16

FMGR

INTRODUCTION

The following section of the FMGR Technical Specifications describes the basic operation of the background program FMGR (92067-16185). It is recommended that the reader study the FMGR source listings (mostly written in SPL 2100) together with this document.

ORGANIZATION

FMGR is organized as a main program with 11 segments (FMGR0, FMGR1, ..., FMGR8, FMGR9, FMGRA). It may be thought of as a collection of action routines which are driven by a control routine. In addition, there are several utility modules that are called by the action routines.

CALLING SEQUENCES

Almost all calls use the standard FTN calling sequence, however there is an extensive list of global integers and arrays which are used to pass information between the modules.

Where the calls are not FTN calling sequences, it is to interface from SPL (which usually uses the FTN sequences) to library routines that do not use the FTN sequence. These interface routines are written in ASMB.

ACTION ROUTINE CALLING SEQUENCE

The standard calling sequence for action routines is:

```
ER=0
CALL <routine> <NOCM., P.RAM, ER>   where:
```

NOCM. is the number of parameters converted by the parse routine. (NOCM. is also an entry point in the main).

P.RAM is a parameter list which contains all the parameters after conversion by the parse routine. The format of this array is as follows:

1. There are four words per parameter.

2. The maximum number of parameters (NOCM.) is 16.

3. Any unsupplied parameters are zeroed.

4. The first word of a parameter set is:

   a. 0 - if no parameter
   b. 1 - if a numeric parameter
   c. 3 - if an ASCII parameter

5. The second, third and fourth words are the parameter's value:

   If word 1 = 0, then all are zeroes.
   If word 1 = 1, then word 2 is the value and words 3 and 4 are zero.

   If word 1 = 3, then words 2, 3 and 4 are the six character ASCII value, padded with blanks on the right, if needed.

ER is an error flag. If non-zero on return, an error number is printed as follows:

If ABS(ER) < 1000 then print "FMGRxxxx" where xxxx is the signed number (+ sign suppressed), and control is transferred to the log device. The thousands digit of the error flag has the following meanings:

0 - Only one message printed. Transfer control to log device.
1 - Two messages printed. Transfer control to log device.
2 - One message printed. Control NOT transferred to log device.
3 - Two messages printed. Control NOT transferred to log device.

In addition, all globals (available in the main and in the action routine segment) are available to the action routine.

FROM ONE SEGMENT TO ANOTHER

This section describes how FMGR is segmented and how it gets from one segment to another. It is recommended that the reader have in front of him listings for FMGR (main), FMGR1 (segment 1 main), FMGR2 (segment 2 main), and C.TAB (the command dispatch table).

Initialization: Let's assume that FMGR has already been initialized on the system. (A detailed description of this process can be found in the section on system initialization.) The following events happen whenever FMGR is started:

a. (Entry in the main of FMGR) The first two characters of the current program's name are put into the segment calling sequence. This is done to allow development wherein the loader changes the name of the main and each segment to ..GR, ..GR0, etc.

b. A call to routine zero (CAD. = 0) in segment 2 (AS2BL) is set up.

c. The EXEC is called to load segment 2.

d. The segment loads its address table address and returns to the main at SEG.R, the common segment return.

e. The main adds CAD. to the table address and gets the action routine's address (in the initial case, IN.IT) and calls this routine.

f. IN.IT will (in this case) set up the input, log device, list device and severity code using the scheduling parameters and then return.

g. FMGR (main) will find both MS and IFLG. set to zero and will thus go to FM.AB. FM.AB is where all commands start. The command loop follows.

h. Close, using the internal close routine, the two global DCB's, I.BUF and O.BUF.

i. Set up to call, and call the EXEC to load segment 1, unless it is already in memory (in which case, go to step k.)

j. Segment 1 sets PARS. (in the main) to point to its local routine and then returns to P.SEG in the main. (For a discussion of when IFLG. is non-zero, see the section on system initialization.)

k. At P.SEG in the main, the error flag is cleared and the parse routine is called via the address set in step j above.

l. The parse routine in the main of segment 1 checks if an operator BReak is pending (if so, it sends a FMGR 000 message) and then to see if a FMGR 000 was sent (BRKF.). If so, checks are made to see if the program is in batch mode and has timed out. If so, ABort (at AB..) is called to flush the job.

m. The parse routine then checks the NO.RD (no read) flag, and then calls REA.C. The NO.RD flag can be zero (normal), or greater than or less than zero. If greater than zero, then the previous command is trying to get to TR (at TR..) or the internal abort routine (at ABT..) and has set CAD. to 1 or 6. This is done by making a return to SEG.R. If the NO.RD flag is less than zero, then a command has been built in the command buffer and is to be parsed, so the read is skipped.

n. REA.C prompts if input is from a TTY and in any case, reads a command and then returns.

o. .PARS is then called to parse the command. .PARS sets up the parameter list (P.RAM), number of command parameters (NOCM.), etc. It also searches C.TAB for the command name. At this point, command capability checking is performed (if this copy of FMGR is running under session control). When the name is found (assuming the capability check has succeeded), the next word is used to set up the segment call. C.TAB has two basic entry formats for action routine addresses. If the address is indirect or greater than 8K, the routine is in the current segment (or the main) and no segment need be loaded. In this case, CAD. is set to the address and .PARS exits with the E-register cleared. If the address is positive and less than 8K, then the low 8 bits are the last character of the segment name and the high 8 bits are the routine number in that segment. In this case, CAD. is set to the routine number, the last character is stored at CUSE. in the main (where it fills out the segment name), and .PARS exits with the E-register set.

p. .PARS exits to FMGR1 (main), which exits back to FMGR (main) where the E-register is checked.

q. If E is set, then the EXEC is called to load the new segment, the segment returns with its table address in the A-register at SEG.R where the routine number is added (from CAD.) and the address is picked up indirect and saved in CAD..

r. The action routine is called.

s. If MS is set non-zero on returning from the action routine, MSS. is called to output the error message.

t. IFLG. will be zero, so control returns to FM.AB (step h).

In addition to the normal return from an action routine, the entry point FM.AB is available to abort the action. This entry point is used by IER. and by the TR routine, among others.

FMGR GLOBAL COMMUNICATION

Global Integers and Arrays

The following is an index of FMGR global integers and arrays:

```
.E.R.
.IDAD
.R.E.
ACTV.
BRKF.
BUF. (129)
C.BUF (40)
C.DLM
CAD.
CAM.I (144)
CAM.O
CAMS. (60)
CUSE.
D.
D.SDR (128)
DS.DF
DS.F1
DS.LU
D.LT
D.LB
ECH.
ECHF.
GO. (56) (P7..,S.TTY,S.CAP)
IFLG.
J.NAM
J.REC
JRN.
N.OPL (10)
NO.RD
NOCM.
O.BUF (144), I.BUF (144)
P.RAM (64)
P.TR
PARS.
PK.DR (128)
TL.P
TM.VL
TMP. (5)
TTY.
```

ENTRY POINT	MODULE	IN MAIN
.E.R.	FM.CM	yes

Function: Used in most FMP interface calls as the error return word.

ENTRY POINT	MODULE	IN MAIN
.IDAD	FMGR	yes

Function: Each segment sets its ID segment address here. Currently used only by the pack action routine (PK..).

ENTRY POINT	MODULE	IN MAIN
.R.E.	FM.CM	yes

Function: Contains number of last error printed for ??.. routine.

ENTRY POINT	MODULE	IN MAIN
ACTV.	FMGR	yes

Function: Contains the TR stack pointer in effect when the JOB card was read, or 0 if no active JOB.

ENTRY POINT	MODULE	IN MAIN
BRKF.	FM.CM	yes

Function: Set non-zero whenever FMGR 000 is sent.

ENTRY POINT	MODULE	IN MAIN
BUF.(129)	FM.CM	yes

Function: General file I/O buffer.

ENTRY POINT	MODULE	IN MAIN
C.BUF(40)	FM.CM	yes

Function: Command input buffer.

ENTRY POINT	MODULE	IN MAIN
C.DLM	FM.CM	yes

Function: Contains the character address of the first delimiter in the current command.

ENTRY POINT	MODULE	IN MAIN
CAD.	FMGR	yes

Function: Contains the current action routine address.

ENTRY POINT	MODULE	IN MAIN
CAM.I(144)	FM.CM	yes

Function: Command input DCB.

ENTRY POINT	MODULE	IN MAIN
CAM.O	FM.CM	yes

Function: Log device logical unit.

ENTRY POINT	MODULE	IN MAIN
CAMS.(60)	FM.CM	yes

Function: Contains the command file stack. Entries are made and removed by the TR command.

ENTRY POINT	MODULE	IN MAIN
CUSE.	FMGR	yes

Function: Contains the last character of the current segment name. May be set to zero to force segment reload (only useful if segment 1 in memory).

ENTRY POINT	MODULE	IN MAIN
D.	FMGR	yes

Function: D. is an array of three words containing "D.RTR" and is used to call the EXEC to schedule D.RTR.

ENTRY POINT	MODULE	IN MAIN
D.SDR	FM.UT	no

Function: Disc directory buffer.

ENTRY POINT	MODULE	IN MAIN
DS.DF	FM.UT,8	no

Function: If zero, forces D.RIO to read even if in memory. (Cleared by LOCK.)

ENTRY POINT	MODULE	IN MAIN
DS.F1	FM.UT,8	no

Function: If zero, forces DR.RD to read even if in memory.

ENTRY POINT	MODULE	IN MAIN
DS.LU D.LT D.LB	FM.UT,8	no

Function: Contains the address of location of the LU/last track/label in D.SDR of the last disc referenced with DR.RD.

ENTRY POINT	MODULE	IN MAIN
ECH.	FM.CM	yes

Function: Contains the number of words in the command.

ENTRY POINT	MODULE	IN MAIN
ECHF.	FM.CM	yes

Function: Contains echoed flag. If zero, then echoed. If non-zero, then not echoed.

ENTRY POINT	MODULE	IN MAIN
G0..	FMGR	yes

Function: Global array entry point. Points at 0G (1S and 0S precede 0G in memory). P7, S.TTY and S.CAP are also entry points in the array.

ENTRY POINT	MODULE	IN MAIN
IFLG.	FMGR	yes

Function: Indicates system initialization is in progress.

If IFLG. = 0, system is not being initialized.
= 2, system disc is being initialized.
= 3, auxiliary disc being initialized.

IFLG. is set by IN.IT when, at FMGR turn on, it is found that the system has not been initialized.

ENTRY POINT	MODULE	IN MAIN
J.NAM	FMGR	yes

Function: Contains name of current job.

ENTRY POINT	MODULE	IN MAIN
J.REC	FMGR	yes

Function: Contains the JOBFIL record number of the current spool JOB, or 0 if not spooling.

ENTRY POINT	MODULE	IN MAIN
JRN.	FMGR	yes

Function: Contains the JOBFIL RN lock word whenever JOBFIL is open.

ENTRY POINT	MODULE	IN MAIN
N.OPL(10)	FMGR	yes

Function: N.OPL is set up by the parse routine to contain the 10 subparameters for the first two parameters. Unsupplied parameters are zeroed. Parameters are in the order of entry.

ENTRY POINT	MODULE	IN MAIN
NO.RD	FMGR	yes

Function: If set negative, the parse routine will parse the current contents of C.BUF instead of reading a new command. If positive, the parse routine assumes the new command is already parsed (used to get to TR.. and ABT..).

ENTRY POINT	MODULE	IN MAIN
NOCM.	FMGR	yes

Function: Contains the number of the highest numbered supplied parameter in the input.

ENTRY POINT	MODULE	IN MAIN
O.BUF(144),I.BUF(144)	FMGR	yes

Function: Input and output DCB's of 144 words each. These DCB's are contiguous with O.BUF first. This fact is used by the pack routine to get a 256-word buffer.

ENTRY POINT	MODULE	IN MAIN
P.RAM(64)	FMGR	yes

Function: Contains the parameter list for the current command.

ENTRY POINT	MODULE	IN MAIN
P.TR	FM.CM	yes

Function: Pointer to current command stack (CAMS.) position.

ENTRY POINT	MODULE	IN MAIN
PARS.	FMGR	yes

Function: Set up by segment 1 to be the address of the read and parse routine.

ENTRY POINT	MODULE	IN MAIN
PK.DR(128)	FM.UT	no

Function: File directory buffer.

ENTRY POINT	MODULE	IN MAIN
TL.P	FMGR	yes

Function: Contains program time limit if one, else JOB time limit.

ENTRY POINT	MODULE	IN MAIN
TM.VL	FMGR	yes

Function: Contains JOB time limit.

ENTRY POINT	MODULE	IN MAIN
TMP.(5)	FM.CM	yes

Function: Contains name or LU (first 3 words), security code (4th word) and negative disc LU of current list file (5th word).

ENTRY POINT	MODULE	IN MAIN
TTY.	FMGR	yes

Function: Contains a zero if current command input is not from a TTY, else non-zero.

GLOBAL LABELS

The following are global labels in FMGR. These are labels which may be/are jumped to.

FM.AB - abort current routine
GT.JB - gets first spool job
INI1. - initialize system return 1
INI2. - initialize system return 2
L.SEG - load new segment (assumes CAD. and CUSE. are set up)
P.SEG - segment 1 return
SEG.R - normal segment return

Global Subroutines and Functions (Main)

The following is an index of global subroutines and functions in the FMGR main program:

.DRCT
CLOS.
CONV.
EC.HO
FM.ER
IER.
JER.
MSS.
OPEN.

In addition, the following system library or FMP routines are also in the FMGR main:

.ENTR
$OPEN
CLOSE
GETAD
IFBRK
OPEN
R/W$
RMPAR
RWND$

ENTRY POINT	MODULE,TYPE
.DRCT	.DRCT,7

Function: To return direct addresses.
Called whenever SPL needs a direct address.
No explicit calls are made by FMGR.

ENTRY POINT	MODULE,TYPE
CLOS.	FM.CM,8

Function: To close one or all DCB's.

Calling sequence: CALL CLOS.(IDCB)

where:

IDCB is the DCB to be closed. If the DCB address is zero, then all DCB's are closed.

Checks are made to see if the DCB was faked open by OPEN. and if so, CLOS. just clears the open flag. Otherwise, it calls CLOSE.

ENTRY POINT	MODULE,TYPE
CONV.	FM.CM,8

Function: To convert a positive integer to ASCII.

Calling sequence: CALL CONV.(NO,IBUF,ND)

where:

NO is the positive integer to be converted

IBUF is the buffer for the result and is, in fact, the word where the last digit is to go, right justified.
ND is the number of digits to convert.

If NO is odd, the high half of the lowest used word is not modified. Leading zeroes are not blanked.

ENTRY POINT	MODULE,TYPE
EC.HO	FM.CM,8

Function: To echo the last command.

Calling sequence: CALL EC.HO

EC.HO expects the command length in ECH. and the command image in C.BUF. ECHF. is tested to see if it is zero, in which case nothing is done. After the echo, ECHF. is set to zero. This convention prevents double echos. If input was from the log device, no echo is done.

ENTRY POINT	MODULE,TYPE
FM.ER	FM.CM,8

Function: To print messages on the log device and optionally to transfer control to the log device.

Calling sequence: CALL FM.ER(SVCOD,IMSS,LN)

where:

SVCOD is the severity of the message as follows:

0,1 - no transfer to log
2 - transfer to log

Also, if SVCOD is less than the current severity code, the whole call is ignored.

IMSS is the message array

LN is its length in +words or -chars.

If the message is to be printed, FM.ER first calls EC.HO to echo the offending command.

ENTRY POINT	MODULE,TYPE
IER.	FM.CM,8

Function: To test .E.R. for FMP errors (.E.R. < 0) and if errors, to call MSS. to report same.

Calling sequence: CALL IER.

If an error is detected, IER. jumps to FM.AB to abort the command.

ENTRY POINT	MODULE,TYPE
JER.	FM.CM,8

Function: Same as IER., plus, if the break flag is set in FMGR's ID segment, a message is sent and FM.AB is exited to. Should only be used if no cleanup is required prior to acting on the break.

ENTRY POINT	MODULE,TYPE
MSS.	FM.CM,8

Function: To print an error message on the log device and optionally transfer control to the log device.

Calling sequence: CALL MSS. (ER,NX)

where:

ER is a decimal error number, formatted as follows:
sYXXX, where:

- s = + or -
- Y = 1 or 3 if NX is coded
  - = 0 or 1 if control is to be transferred to the log device
  - = 2 to print one message without transfer

NX is an optional parameter to print a second message if Y = 1 or 3. NX must be between 0 and 999, inclusive.

MSS. prints either
```
FMGR XXX
```
or
```
FMGR XXX
FMGR ZZZ
```
where

ZZZ = NX

ENTRY POINT	MODULE,TYPE
OPEN.	FM.CM,8

Function: To open or fake open a file to a given DCB.

Calling sequence: CALL OPEN. (IDCB,NAME,IOPL,ISOPT)

where:

IDCB is the DCB array

NAME is a file name array or an LU

IOPL is a 2-word array of open parameters as follows:

IOPL: security code
disc id
(normally supplied by referencing N.OPL)

ISOPT is the open option word with the addition that if the sign bit is set and the file is type 0 with an EOF of leader and IDCB refers to 0.BUF, then an EOF call is made (i.e., if punch output, then punch leader).

If NAME is less than 20000B, then it is assumed to be an LU which causes OPEN. to set up a dummy DCB entry for it. In this case, ISOPT defines the read/write subfunction and the EOF subfunction is set as follows:

If DVR05 subchannel 2 or 3, then 1 (EOF).
If DVR17 or higher, then 1 (EOF).
If DVR02 or ISOPT bits 6 and 3 set, then 10B (leader)
If none of the above, then 11B (page).

Special processing:

If IDCB refers to CAM.I, then the transfer stack is updated to indicate a new command file. Also the TTY. flag is updated. If IDCB refers to CAM.I or I.BUF, and a type 0 file then a set EOT call is made.

Global Subroutines and Functions (Segments)

The following is an index of global subroutines and functions in FMGR segments:

.PARS
AVAIL
B.FLG
BUMP.
C.TAB
CK.ID
CK.SM
CREA.
D.RIO
DR.RD
EX.TM
FID.
FREE.
ID.A
IPUT
J.PUT
LOCK.
LU.CL
LULU.
MSC.
ONOFF
RANGE
REA.C
READ.
SET.T
ST.TM
TL.
WRLG.,EFLG.

ENTRY POINT	MODULE,TYPE
.PARS	.PARS,8

Function: To construct the parameter list passed to the action routines from the input command, while trapping the following errors:

1. Leading colon on TTY input.
2. No leading colon on non-TTY input.
3. Illegal command name.
4. Subparameter 3,4 or 5 not numeric.
5. More than 5 subparameters.
6. Subparameters on other than first two parameters.
7. Too many parameters or too many characters in a parameter. (These are buffer limit checks unrelated to the actual command).
8. Illegal global reference.

All of these errors cause the FMGR 010 message followed by the portion of the line up to and including the problem character, followed by a "?".

Calling sequence: CALL .PARS

.PARS refers to the following external integers and arrays:

C.BUF	CUSE.	NOCM.
C.TAB	GO..	P.RAM
CAD.	N.OPL	TTY.

.PARS provides a reconstructed command image with all globals replaced and echoes the command unless it is an SV command. Since .PARS does the global substitution, command processors need not be aware of globals.

Exceptions: Commands with their sign bit set in the command table (C.TAB) will not be error checked except for the leading ":" and for illegal global references.

ENTRY POINT	MODULE,TYPE
AVAIL	AVAIL,8

Function: To scan the 3 words of availability bits for an available spool pool file.

Calling sequence: CALL AVAIL (ADDR,MASK,FNUM)

where:

ADDR is the 3 words of availability bits

MASK will be returned with a bit set corresponding to the bit set in the 3 words

FNUM will be the bits number, i.e., to 80. If FNUM is returned as zero, no file was found. FNUM may be returned greater than the highest created file.

ENTRY POINT	MODULE,TYPE
B.FLG	B.FLG,8

Function: To set/clear the batch flag in the ID segment.

Calling sequence: CALL B.FLG (OP)

where:

OP is 0 to clear and non-zero to set the flag

This routine only sets the flag if the calling program is FMGR.

ENTRY POINT	MODULE,TYPE
BUMP.	BUMP.,8

Function: To calculate P1 = P1 - P2 + $BATM, where P1 and P2 are call parameters and all are double-word integers.

Calling sequence: CALL BUMP. (P1,P2)

This routine is used to calculate the remaining job time after running a program under the program time limit.

If P1 = initial program limit and
P2 = remaining batch time when program was started,

then

$BATM - P2 is the program run time and
P1 - P2 + $BATM is the remaining batch time.

ENTRY POINT	MODULE,TYPE
C.TAB	C.TAB,8

Function: To define the legal commands and their locations.

The format of C.TAB is shown in the section on the Command Dispatch Table. Note that the final NOP defines the end of the table. The first NOP defines the null command (TR). It must be first, otherwise it will be interpreted as the end of the table.

ENTRY POINT	MODULE,TYPE
CK.ID	CK.ID,7

Function: To validate an ID segment address.

Calling sequence:
```
JSB CK.ID
DEF *+2
DEF ID      ID segment address
```

Return:
```
E = 0       Valid ID segment address
E = 1       Invalid ID segment address
```

ENTRY POINT	MODULE,TYPE
CK.SM	CK.SM,7

Function: To check a checksum.

Calling sequence:
```
CALL CK.SM (IBF,ITYP)
```

where:

```
IBF  = buffer containing the record

ITYP = 1 for relocatable
     = 0 for absolute
```

CK.SM checks that the word count is less than 377B and that the checksum is right. If not, an FRETURN (see SPL manual) is made.

ENTRY POINT	MODULE,TYPE
CREA.	CREA.,8

Function: To create a file (not type 0)

Calling sequence: CALL CREA. (IDCB,NAME,IPLST)

where:

IDCB = Data Control Block (144 words)

NAME = File name

IPLST = 5-word array as follows:

Word 1 = Security code
Word 2 = Disc id
Word 3 = File type
Word 4 = File size
Word 5 = Record size

(IPLST is usually specified by referencing N.OPL)

If the first word of the NAME is less than 64, then an FRETURN is made (see SPL manual). Also, if the file size is -1, then the true size is set into IPLST (helps when truncating).

ENTRY POINT	MODULE,TYPE
D.RIO	FM.UT,8

Function: To read/write the cartridge directory to/from D.SDR.

Calling sequence: CALL D.RIO (RCODE)

where:

RCODE = 1 for read
= 2 for write

If a read request, the disc transfer is not done if DS.DF is non-zero. DS.DF is set non-zero after a successful transfer. Write requests are direct if IFLG. is non-zero, else D.RTR is passed the buffer via a disc track obtained from the system.

ENTRY POINT	MODULE,TYPE
DR.RD	FM.UT,8

Function: To read/write a given directory block to/from PK.DR.

Calling sequence: CALL DR.RD (RCODE,DISID,BLK)

where:

RCODE = 1 for read
= 2 for write

DISID = disc label (+) or LU (-)

BLK = relative directory block to be transferred

DR.RD first checks to see if the addressed disc (DISID) is the same as the last disc accessed (DS.F1). If so, DS.LU, D.LT and D.LB are already set up. If not, DR.RD calls D.RIO to read the disc directory into memory and then searches for the addressed disc. If it is not found, an FRETURN (see SPL manual) is made. If it is found, DS.LU, D.LT and D.LB are set up as address pointers to the cartridge directory entries and may be used to examine and/or modify the LU, last track and label therein.

DR.RD sets up its access parameters by reading block zero and saving the parameters or if writing block zero, then the read is skipped. It is thus illegal to write some block other than zero prior to reading some block.

DR.RD makes an FRETURN if an access is attempted beyond the end of the directory. Writes are either direct if IFLG. is non-zero, or are passed to D.RTR via a system track.

ENTRY POINT	MODULE,TYPE
EX.TM	EX.TM,8

Function: To compute the job execute time and print it on the current list device.

Calling sequence: CALL EX.TM

ENTRY POINT	MODULE,TYPE
FID.	FID.,8

Function: Returns false if a reasonable file system header exists on the referenced disc.

Calling sequence: NEW = FID. (IDS)

where:

IDS = +label or -LU of disc in question

FID. checks the cartridge specification entry of the disc to see if:

1. The sign bit is set on word 0.
2. Words 0, 1 and 2 are a legal file name (exclusive of the sign bit on word 0).
3. The label word (word 3) is greater than zero.
4. Word 7 (last file track + 1) - word 8 (-number of directory tracks) - 1 equals the last track for the disc.
5. Word 6 (number of sectors per track) is less than word 5 (next available sector).

If any of these tests fail, FID. returns 1. Otherwise a zero is returned.

ENTRY POINT	MODULE,TYPE
FREE.	FREE.,8

Function: To set or clear an allocate bit in the 3-word spool pool allocate field.

Calling sequence: CALL FREE. (NAM,BITAD)

where:

NUM = last 2 characters of the spool pool file name (should be an ASCII number between 01 and 80)

BITAD = address of the 3 bit flag words

ENTRY POINT	MODULE,TYPE
ID.A	ID.A,8

Function: To find a given ID segment.

Calling sequence: IDAD = ID.A (NAME)

IDAD will be set to NAME's ID segment address. If NAME is all zeroes, then a blank ID segment is found, if possible without tracks. If no ID segment satisfies the request, an FRETURN (see SPL manual) is made.

ENTRY POINT	MODULE,TYPE
IPUT	IPUT,6

Function: To store into a memory location below the MP fence.

Calling sequence: CALL IPUT (ADDRESS,VALUE)

where:

ADDRESS = location to be stored into

VALUE = word to be stored there

ENTRY POINT	MODULE,TYPE
J.PUT	J.PUT,8

Function: To assign and release system tracks for file usage. Used only by the IN processor.

Calling sequence: CALL J.PUT (ITAT,ICD,IER)

where:

ITAT = track assignment table address

ICD = new code to be set at ITAT

IER = returned 0 if assignment done
returned non-zero if not done

The assignment will only be done if the current entry is zero or FMP (i.e., 77776B).

ENTRY POINT	MODULE,TYPE
LOCK.	LOCK.,8

Function: To lock or unlock a given disc.

Calling sequence: CALL LOCK. (IDIS,IRQ)

where:

IDIS = disc ID (+label or -LU)

IRQ = 3 to lock, 5 to unlock

LOCK. calls MSS. if D.RTR returns an error. LOCK. clears DS.DF, the disc directory in memory flag.

ENTRY POINT	MODULE,TYPE
LU.CL	LU.CL,8

Function: To close all spools.

Calling sequence: CALL LU.CL

This routine calls SMP to close each LU in the LU switch table. It then calls LULU. to clear the switch table.

ENTRY POINT	MODULE,TYPE
LULU.	LULU.,8

Function: To set/clear entries in the LU switch table.

Calling sequence:
```
CALL LULU. (LU1,LU2)
go to ERROR
normal exit
```

where:

LU1 = LU to be switched. If zero, then all transforms are cleared.

LU2 = LU to be switched to. If zero, then any existing switch for LU1 is cleared.

The error exit is taken only on a set where no room exists in the table. If LU1 is already in the table, its entry is rewritten.

ENTRY POINT	MODULE,TYPE
MSC.	MSC.,8

Function: To check the master security code.

Calling sequence:
```
IOK = MSC. (LST)
```

where:

LST = the 4-word P.RAM entry for the master security code.

IOK = set to 1 if the master security code is zero, or if the supplied code matches. Otherwise, set to zero.

ENTRY POINT	MODULE,TYPE
ONOFF	ONOFF,8

Function: To set up and print the JOB ON and OFF statements. Also sets up J.NAM.

Calling sequence: CALL ONOFF (NAME,TIME)

where:

NAME = job name (3 words) for ON and 0 (1 word) for OFF

TIME = 6-word time array as follows:

- Word 1 = 10's milliseconds
- Word 2 = Seconds
- Word 3 = Minutes
- Word 4 = Hours
- Word 5 = Days
- Word 6 = Year

ENTRY POINT	MODULE,TYPE
RANGE	RANGE,8

Function: To search JOBFIL record 18 and return the cartridge reference number of the disk containing the spool pool file denoted by FNUM.

Calling sequence: CALL RANGE (FNUM,BUFR)

where:

FNUM = file number of a spool pool file of the form SPOLxx.

BUFR = first word of 16-word buffer containing record 18 of JOBFIL

ENTRY POINT	MODULE,TYPE
REA.C	REA.C,8

Function: To send colon prompt if TTY. is non-zero and then read a command.

Calling sequence: CALL REA.C

READF is called to read in to C.BUF with length into ECH. If a read error or a zero-length record, REA.C will try again. A zero word is set after the last word read.

ENTRY POINT	MODULE,TYPE
READ.	READ.,7

Function: READ. is an interface to the READS library routine to read LS source.

Calling sequence: CALL READ. (LU,BUF,RQLN,L)

where:

LU	is the source LU (2 = LS area)
BUF	is the user's buffer
RQLN	is the requested length in words
L	is the return length in words, or -1 if end of file.

ENTRY POINT	MODULE,TYPE
SET.T	SET.T,8

Function: To set $BATM.

Calling sequence: CALL SET.T (NT,OT)

where:

NT is the new double-word batch time

OT is a double-word into which the old time is stored

SET.T first saves the current $BATM in OT, then resets $BATM to NT.

ENTRY POINT	MODULE,TYPE
ST.TM	ST.TM,8

Function: To convert hours, minutes, seconds to a negative double-word integer in 10's of MS.

Calling sequence: TIME = ST.TM(HR,MIN.S)

where:

HR is the hours to be converted

MIN.S is a two-word array containing the minutes (word 1) and seconds (word 2)

TIME is a real variable which will receive the double-word time

ENTRY POINT	MODULE,TYPE
TL.	TL.,8

Function: This function returns with A = -1 (true) if TL.P < $BATM and $BATM < 0. If not true, A = 0.

Calling sequence:
```
JSB TL.
DEF *+1
```

$BATM is system batch time. This function is used by the RU command processor to determine if it must use program time limit to run the program (yes if true).

ENTRY POINT	MODULE,TYPE
WRLG. EFLG.	WRLG.,7

Function: To interface to the library write load-and-go subroutine.

Calling sequence:

To write a record: CALL WRLG. (IBUF,L)

where:

IBUF = buffer containing the record

L = record length

To flush the end record: CALL EFLG.

(Must be done prior to a subsequent NAM record)

SYSTEM INITIALIZATION

INTRODUCTION

The following discussion covers the initialization of the file management system the first time the system is booted. The modules involved are IN.IT, IN.., FMGR1 (main) and FMGR (main).

System Action on Booting

The RTE system, when booted, does the following for FMGR:

a) Schedules FMGR.

b) Sets TATLG (1755B) on the base page to show one track (-1) and priority to 0.

c) Puts the old contents of TATLG in $OTAT and the old priority in $OPRI.

d) Insures that FMGR is first in the schedule list, regardless of its relative priority with respect to the other possible turn-on programs.

e) FMGR is thus assured of first use of the background and will be located and executed.

INITIALIZATION SEQUENCE

FMGR will run as described in the section FROM ONE SEGMENT TO ANOTHER down through step e. Refer to the referenced listings for the following steps.

a) If IFLG. is non-zero, go to e. (On first entry, IFLG. will be zero.)

b) Get the parameters using RMPAR and the EXEC string fetch.

c) Set the location of .E.R.+1 to the severity code. (The severity code is usually addressed this way.)

d) Set up CAM.0 for the system TTY logical unit.

e) Search the id segments for D.RTR and save its id segment address.

f) If not found, issue the FMGR 008 message, if TATLG = -1 then reset it from $OTAT and terminate.

g) If the last system track is assigned to D.RTR, set up the input, output and list files and go to FM.AB.

h) Assign all available tracks to the current executing program (FMGR). Also assign the track containing the cartridge directory to FMGR.

i) Reset TATLG if it is -1 now. Reset priority if it is now 0.

j) Read the directory of discs and open CAM.I to the system TTY.

k) Compute the sum of base page words 1650B through 1657B, 1742B through 1747B and 1755B through 1764B. If RTE-II or RTE-III, also add in the sum of base page words 1750B through 1754B.

l) If IFLG. is not zero, go to step w.

m) If the sum computed in step k is the same as the one in word 253 (initialization code word) of the directory of discs, go to step y. Set up the cartridge directory in the track and sector defined by $CL1 and $CL2. If a valid cartridge specification entry exists in the first sector, last track of LU 2, then no information from a previous system's cartridge directory is available since it was kept in the system area of the disc. However, if a cartridge directory is found in the first sector, last track of LU 2, then such information is available since this implies a pre-RTE-IVB file structure. In this case, information is taken from the old format cartridge directory and used to build the new format cartridge directory at the track and sector specified by $CL1 and $CL2. The old cartridge specification entry is then moved to the first 16 words of the last track and -1 is written over the remainder of the old cartridge directory and the old cartridge specification entry. If files were saved from an old system, then open flags are all zeroed on LU 2 and control proceeds to step y.

n) Set IFLG. to 2 (this indicates we are initializing LU 2).

o) Go to INI1 in the FMGR main.

p) Call MSS. to output IFLG. as a message (either FMGR 002 or FMGR 003), then load segment 1.

q) Since IFLG. is set, segment 1 calls its routine PAR which is described in FROM ONE SEGMENT TO ANOTHER steps 1 through n, and then jumps to the main entry point, INI2..

r) At this point, .PARS will have left the command name in the B-register. A check is made to see if "IN" was entered and if so, transfers to this action routine.

s) If "??" was entered, then set MS to 4 (to force FMGR 004 at step u below), else go to step p.

t) The segment is loaded and the routine is called. Both ??.. and IN.. have special code for the case where IFLG. is set.

u) On return from the action routine, any error is printed (including the one set at step s above) and is there is an error, control transfers to step p above.

v) Reload the IN.IT segment and go to step a above. Note that IFLG. is still set, so IN.IT will do different things. Some things must be redone, however, because the segment is reloaded.

w) If IFLG. equals 2, then if there is an LU 3, set IFLG. to 3 and go to step o. Note that a bit in word 255 of the cartridge directory is set to indicate that the FMGR 003 message has been sent (it is sent only once).

x) Read the directory of discs, set the computed sum in word 253 and write it back out.

y) Initialize LU 2 (from subroutine TATUP).

z) If not a valid file system disc (FID.), then if LU = 2 then go to step n, else go to step dd.

aa) Set each track assignment word for file tracks to 77776B if the current program (FMGR) owns the track. Otherwise report FMGR 005 followed by the relative track assignment location.

bb) Assign the directory track to D.RTR under the same conditions as above.

cc) If LU = 2, then set to 3 and go to step z.

dd) Assign the track containing the cartridge directory to D.RTR. Release all owned tracks.

ee) Schedule GASP, then schedule ACCTS to initialize the Session Monitor system.

ff) Set up the input, log, list and severity code and enter the command loop.

COMMAND CAPABILITY CHECKING

File manager command capability checking is performed for users operating in the Session Monitor environment. The capability level defined in the user's Session Control Block (SCB), together with the File Manager Command Dispatch Table determine the set of commands which the user is permitted to execute. The user's copy of FMGR (FMGxx) compares the user's capability level with the capability level defined for the specific command. If the user's capability level is greater than or equal to the capability level defined for the command, processing of the command is permitted. Otherwise, an error condition (FMGR 046 INSUFFICIENT CAPABILITY) is reported.

SEQUENCE OF OPERATIONS

The routine .PARS is called to parse the command, constructing the parameter list which is to be passed to the appropriate action routine. It searches the Command Dispatch Table (C.TAB), searching for a matching command name. C.TAB is searched for the specified command from the lowest capability through the user's capability. The user's capability level is determined using the function ICAPS. If a match is found, the next word in C.TAB is used to set up the segment call for the action routine. If a match is not found by the time we reach the user's capability in the Command Dispatch Table, we must determine if:

a) the user has insufficient capability
b) the command is undefined
c) the command is a special session command (e.g., a break-mode command equivalent)

We continue to search C.TAB through the last command in the highest capability group of commands. If a match is found, the user has insufficient capability to execute the command (a). If a match is not found, we search the remainder of C.TAB until a match is found or the end of the table is reached. If a match is found now, a call is made to the library capability check routine, CAPCK, which determines whether the user has sufficient capability. If a match was not found, the command is undefined. The use of a common capability check routine by FMGxx and the break mode command processor insures that for those commands defined in both session and break modes, capability checking will be exactly the same regardless of the operating mode.

Notes: 1) If the override flag (OVRD.) is set (bit 14), the command is allowed, even if the user's capability level is lower than the level required for the command.

2) The variable C.LVL is the pointer used, when searching the command table, to indicate the stopping point for the current search.

COMMAND CAPABILITY LEVEL ASSIGNMENTS

FMGR			SYSTEM and			Capability Level				BREAK-MODE	
EX	SY	TR	HE	OP		1					
HE											
AC	MC	TE	*BL	RS	*TO		10				
CL	ME	WH	+BR	*SL	UP						
DC	*SL	??	EQ	ST	WH						
DL	SM	**	FL	TE							
LI			*QU	TI							
AN	DP	RN						20			
CN	DU	ST									
CO	LL	SV									
CR	PK										
CT	PU										
AB	+OF	SL	+GO	RT	*SS				30		
CS	RP	SP	+OF	RU	SZ						
EO	RT	TL									
JO	RU										
CA	PA	SE								40	
IF											
LO	SL		AS	ON	UR						50
			IT	PR							
IN			BL	GO	SS						60
OF			BR	LU	TM						
			DN	OF	TO						
			EQ	QU							

* Single Parameter Only

\+ Program must be under session's control

COMMAND DISPATCH TABLE

```
0001                ASMB,R,L,C
0002*     NAME:   C.TAB
0003*     SOURCE: 92067-18201
0004*     RELOC:  92067-16185
0007*  ************************************************************
0008*  * (C) COPYRIGHT HEWLETT-PACKARD COMPANY 1979.  ALL RIGHTS     *
0009*  * RESERVED.  NO PART OF THIS PROGRAM MAY BE PHOTOCOPIED,      *
0010*  * REPRODUCED OR TRANSLATED TO ANOTHER PROGRAM LANGUAGE WITHOUT*
0011*  * THE PRIOR WRITTEN CONSENT OF HEWLETT-PACKARD COMPANY.       *
0012*  ************************************************************
0013*
0014  00000             NAM C.TAB,8 92067-16185 REV.1903 790207
0015                    ENT C.TAB
0016*
0017*     SET UP SEGMENT AND ROUTINE NUMBERS.
0018*
0019  00000       R0    EQU 0
0020  00400       R1    EQU 400B
0021  01000       R2    EQU R1+R1
0022  01400       R3    EQU R2+R1
0023  02000       R4    EQU R3+R1
0024  02400       R5    EQU R4+R1
0025  03000       R6    EQU R5+R1
0026  03400       R7    EQU R6+R1
0027  04000       R8    EQU R7+R1
0028  04400       R9    EQU R8+R1
0029  05000       R10   EQU R9+R1
0031  00060       S0    EQU 60B
0032  00061       S1    EQU S0+1
0033  00062       S2    EQU S0+2
0034  00063       S3    EQU S0+3
0035  00064       S4    EQU S0+4
0036  00065       S5    EQU S0+5
0037  00066       S6    EQU S0+6
0038  00067       S7    EQU S0+7
0039  00070       S8    EQU S0+8
0040  00071       S9    EQU S0+9
0041  00101       SA    EQU 101B
0042*
0043*     THIS IS THE COMMAND DISPATCH TABLE FOR THE FMGR PROGRAM.
0044*     EACH COMMAND ID IS FOLLOWED BY ITS ADDRESS.
0045*     FOR ROUTINES IN THE HOME SEGMENT THIS IS AN ADDRESS (DEF XX).

0046*     FOR ROUTINES IN OTHER SEGMENTS IT IS THE ASCII SEGMENT
0047*     SUFFIX IN THE LOW HALF OF THE WORD AND THE ROUTINE
0048*     NUMBER IN THAT SEGMENT IN THE HIGH HALF OF THE WORD.
0049*     .PARS BREAKS THESE APART BY THE ADDRESS BEING 0< ADD < 10000B
0050*     FOR SEGMENT ADDRESS.
0051*
0052*     COMMANDS WITH THE SIGN BIT SET INDICATE THAT THE COMMAND
0053*     NEED NOT SATISFY ALL THE SYNTAX RESTRICTIONS IMPOSED ON
0054*     OTHER COMMANDS.
```

```
0055*
0057*
0058*      SESSION MONITOR COMMAND CAPABILITY LEVELS
0059*
0060  00000 000023R C.TAB DEF BEGIN
0061  00001 000021R       DEF ENDS
0062  00002 000157R       DEF SCMD
0063  00003 000001  L1     DEC 1
0064  00004 000033R L1A    DEF LV10
0065  00005 000012  L10    DEC 10
0066  00006 000065R L10A   DEF LV20
0067  00007 000024  L20    DEC 20
0068  00010 000117R L20A   DEF LV30
0069  00011 000036  L30    DEC 30
0070  00012 000143R L30A   DEF LV40
0071  00013 000050  L40    DEC 40
0072  00014 000153R L40A   DEF LV50
0073  00015 000062  L50    DEC 50
0074  00016 000155R L50A   DEF LV60
0075  00017 000074  L60    DEC 60
0076  00020 000157R L60A   DEF SCMD
0077  00021 000165R ENDS   DEF NONSM
0078  00022 000201R ENDT   DEF END
0080*
0081*      STRUCTURE CHECKS
0082*
0083  00000                ORG C.TAB
0084  00000 000000         BSS ENDT-ENDS
0085  00001 000000         BSS ENDS-L60A
0086  00002 000000         BSS L60A-L50A
0087  00004 000000         BSS L50A-L40A
0088  00006 000000         BSS L40A-L30A
0089  00010 000000         BSS L30A-L20A
0090  00012 000000         BSS L20A-L10A
0091  00014 000000         BSS L10A-L1A
0092  00023                ORR
0094  00023          BEGIN EQU *
0095  00023 000000         NOP            NULL COMMAND (TR)
0096  00024 000001X        DEF TR..
0097  00025 052122         ASC 1,TR
0098                       EXT TR..
0099  00026 000001X        DEF TR..
0100  00027 042530         ASC 1,EX
0101                       EXT EE..
0102  00030 000002X        DEF EE..
0103  00031 151531         OCT 151531     "SY" WITH SIGN BIT SET
0104  00032 001067         ABS S7+R2
0105  00033 037477  LV10   ASC 1,??       <<CAPABILITY LEVEL 10 COMMANDS>>
0106  00034 000467         ABS S7+R1
0107  00035 125052         OCT 125052     "**" WITH SIGN BIT SET
0108  00036 000202R        DEF COMM
0109  00037 125000         OCT 125000     "*<NULL>" WITH SIGN BIT SET
0110  00040 000202R        DEF COMM
0111  00041 125040         OCT 125040     "*<BLANK>" WITH SIGN BIT SET
```

```
0112   00042  000202R         DEF  COMM
0113   00043  046111          ASC  1,LI
0114   00044  000471          ABS  S9+R1
0115   00045  041514          ASC  1,CL
0116   00046  000071          ABS  S9+R0
0117   00047  042114          ASC  1,DL
0118   00050  000463          ABS  S3+R1
0119   00051  046503          ASC  1,MC
0120   00052  001464          ABS  S4+R3
0121   00053  042103          ASC  1,DC
0122   00054  002064          ABS  S4+R4
0123   00055  053510          ASC  1,WH
0124   00056  004065          ABS  S5+R8
0125   00057  151515          OCT  151515     "SM" WITH SIGN BIT SET
0126   00060  000101          ABS  SA+R0
0127   00061  046505          ASC  1,ME
0128   00062  000501          ABS  SA+R1
0129   00063  040503          ASC  1,AC
0130   00064  002464          ABS  S4+R5
0131   00065  041522  LV20    ASC  1,CR       <<CAPABILITY LEVEL 20 COMMANDS>>
0132   00066  000470          ABS  S8+R1
0133   00067  051524          ASC  1,ST
0134   00070  001060          ABS  S0+R2
0135   00071  042125          ASC  1,DU
0136   00072  001460          ABS  S0+R3
0137   00073  050125          ASC  1,PU
0138   00074  001062          ABS  S2+R2
0139   00075  051116          ASC  1,RN
0140   00076  002066          ABS  S6+R4
0141   00077  041517          ASC  1,CO
0142   00100  000460          ABS  S0+R1
0143   00101  050113          ASC  1,PK
0144   00102  000060          ABS  S0+R0
0145   00103  041516          ASC  1,CN
0146   00104  003065          ABS  S5+R6
0147   00105  046114          ASC  1,LL
0148   00106  000064          ABS  S4+R0
0149   00107  051526          ASC  1,SV
0150   00110  001064          ABS  S4+R2
0151   00111  142120          OCT  142120     "DP" WITH SIGN BIT SET
0152                          EXT  DP..
0153   00112  000003X         DEF  DP..
0154   00113  140516          OCT  140516     "AN" WITH SIGN BIT SET
0155   00114  002465          ABS  S5+R5
0156   00115  141524          OCT  141524     "CT" WITH SIGN BIT SET
0157   00116  004465          ABS  S5+R9
0158   00117  051520  LV30    ASC  1,SP       <<CAPABILITY LEVEL 30 COMMANDS>>
0159   00120  000070          ABS  S8+R0
0160   00121  151125          OCT  151125     "RU" WITH SIGN BIT SET
0161   00122  000465          ABS  S5+R1
0162   00123  051120          ASC  1,RP
0163   00124  000065          ABS  S5+R0
0164   00125  047506          ASC  1,OF
0165   00126  001466          ABS  S6+R3
```

```
0166  00127 051124         ASC 1,RT
0167  00130 001066         ABS S6+R2
0168  00131 045117         ASC 1,JO
0169  00132 000066         ABS S6+R0
0170  00133 042517         ASC 1,EO
0171  00134 000466         ABS S6+R1
0172  00135 041523         ASC 1,CS
0173  00136 000063         ABS S3+R0
0174  00137 040502         ASC 1,AB
0175                       EXT AB..
0176  00140 000004X        DEF AB..
0177  00141 052114         ASC 1,TL
0178  00142 001065         ABS S5+R2
0179  00143 051505  LV40   ASC 1,SE      <<CAPABILITY LEVEL 40 COMMANDS>>
0180                       EXT SE..
0181  00144 000005X        DEF SE..
0182  00145 044506         ASC 1,IF
0183                       EXT IF..
0184  00146 000006X        DEF IF..
0185  00147 041501         ASC 1,CA
0186                       EXT CA..
0187  00150 000007X        DEF CA..
0188  00151 150101         OCT 150101    "PA" WITH SIGN BIT SET
0189  00152 001465         ABS S5+R3
0190  00153 046117  LV50   ASC 1,LO      <<CAPABILITY LEVEL 50 COMMANDS>>
0191  00154 000464         ABS S4+R1
0192  00155 044516  LV60   ASC 1,IN      <<CAPABILITY LEVEL 60 COMMANDS>>
0193  00156 000462         ABS S2+R1
0194  00157 051514  SCMD   ASC 1,SL      <<SPECIAL SESSION COMMANDS>>
0195  00160 002466         ABS S6+R5
0196  00161 144105         OCT 144105    "HE" WITH SIGN BIT SET
0197  00162 003465         ABS S5+R7
0198  00163 152105         OCT 152105    "TE" WITH SIGN BIT SET
0199  00164 002065         ABS S5+R4
0200  00165 046125  NONSM  ASC 1,LU      <<NON-SESSION COMMANDS>>
0201  00166 002466         ABS S6+R5
0202  00167 046123         ASC 1,LS
0203  00170 001066         ABS S6+R2
0204  00171 046107         ASC 1,LG
0205  00172 001066         ABS S6+R2
0206  00173 046523         ASC 1,MS
0207  00174 003064         ABS S4+R6
0208  00175 046522         ASC 1,MR
0209                       EXT MR..
0210  00176 000010X        DEF MR..
0211  00177 051501         ASC 1,SA
0212  00200 001070         ABS S8+R2
0213  00201 000000  END    NOP           <<END OF COMMAND TABLE>>
0214*
0215*
0216  00202 000000  COMM   NOP
0217  00203 162202R        LDA COMM,I
0218  00204 124000         JMP 0,I
0219                       END
```

CAPCK SUBROUTINE

The CAPCK routine verifies that the requestor has sufficient capability to perform the specified command.

```
+-----------------------------------------------------------------+
|                                                                 |
|  CALL CAPCK (IBUF, ILEN [,ISES])                                |
|                                                                 |
+-----------------------------------------------------------------+
|                                                                 |
|  IBUF - Command buffer.  IBUF contains the ASCII command        |
|         string that is to be checked.                           |
|                                                                 |
|  ILEN - Command length.  ILEN contains the length (characters)  |
|         of the command string.                                  |
|                                                                 |
|  ISES - Session Control Block (SCB) address.  (see COMMENTS)    |
|                                                                 |
+-----------------------------------------------------------------+
```

COMMENTS:

If the calling program is under session control, the ISES parameter has no meaning; the SCB address of the calling program is used. If the calling program is not under session control, ISES must match the address of an existing SCB.

The A- and B-register returns from the CAPCK routine are defined as follows:

* If the command is not defined, then the A-register will contain -1 and the B-register contents will be meaningless.

* If the command is defined, the A-register will contain the number of parameters in the command (if the capability check was passed) or -1 (if the capability check failed).

The Assembly Language calling sequence is:

```
      JSB CAPCK
      DEF RTN
      DEF IBUF
      DEF ILEN
      DEF ISES
RTN   .
      .
```

The A- and B-register returns are described above.

Capability Check Overriding

A command file stack (CAMS.) is used to save up to ten transfer file names or logical units, enabling the file system to transfer back to nested transfer files. The TR.. action routine in conjunction with OPEN. (in FM.CM) adds and removes entries from the command stack. The pointer to the current command stack position is given by P.TR. For systems with the Session Monitor, a check is made of the location of the currently operating transfer file. If the transfer file is on a system cartridge (LU 2 or 3), a flag is set, allowing the capability checks for commands from that transfer file to be overridden.

Sequence of Operations

OPEN., which is called by the TR.. action routine, sets bit 14 of the global cartridge checking override flag, OVRD., then calls the FMP OPEN routine to open the transfer file. This enables system cartridges to be searched in addition to the normal searching of private and group cartridges.

OVRD. - FMGxx Cartridge Search Override Flag
= 0 indicates search only the user's private and group cartridges
Bit 15 = 1 indicates search all cartridges
Bit 14 = 1 indicates search user's private and group cartridges, then system cartridges
Bit 13 = 1 indicates search system cartridges only (even if bit 14 is set)
OVRD. is initialized to 0, set and/or reset by TR.., OPEN., SP.., SM.. and ME.., and examined by .PARS and SY.. (for capability checking) and by FMP library routines (pass flag to D.RTR)

On return from the transfer file OPEN call (in OPEN.), the DCB is examined to determine if the file was found on a system cartridge (Bit 4 of DCB word 7 set). If not (Bit 4 clear), OVRD. bit 14 is reset to 0. When removing a transfer file name from the command stack, OVRD. bit 14 is reset to 0 also.

The OVRD. flag is used by .PARS in searching the Command Dispatch Table when performing command capability checking. If OVRD. bit 14 is set, capability checking is ignored. Thus any File Manager command which originated from a transfer file on a system cartridge is allowed. Note that such a command can reference the user's private and group cartridges and the system cartridges, but cannot reference or access another session user's cartridges or non-session cartridges.

FMGR ACTION ROUTINES

Action Routine Index

COMMAND	ENTRY POINT	MODULE NAME, TYPE
AB	AB..	AB..,8
AN	AN..	OPMES,8
CA	CA..	CA..,8
CL	CL..	CL..,8
CN	CNT.	CNT.,8
CO	CO..	CO..,8 (calls DU..)
CR	CR..	CR..,8
CS	CS..	CS..,8
CT	CT..	CT..,8
DC	RC..	RC..,8
DL	DL..	DL..,8
DP	DP..	DP..,8
DU	DU..	ST.DU,8
EO	EO..	EO..,8
EX	EE..	EE..,8
HE	HE..	HE..,8
IF	IF..	IF..,8
IN	IN..	IN..,8
JO	JO..	JO..,8
LG	LG..	LG..,8
LI	LI..	LI..,8
LL	LL..	F.UTM,8
LO	LO..	F.UTM,8
LS	LG..	LG..,8
LU	LU..	LU..,8
MC	MC..	MC..,8
ME	ME..	ME..,8
MR	MR..	MR..,8
OF	OF..	OF..,8
PA	PA..	OPMES,8
PK	PK..	PK..,8
PU	PU..	PU..,8
RN	CN..	CN..,8
RP	RP..	RP..,8
RT	LG..	LG..,8
RU	RU..	RU..,8

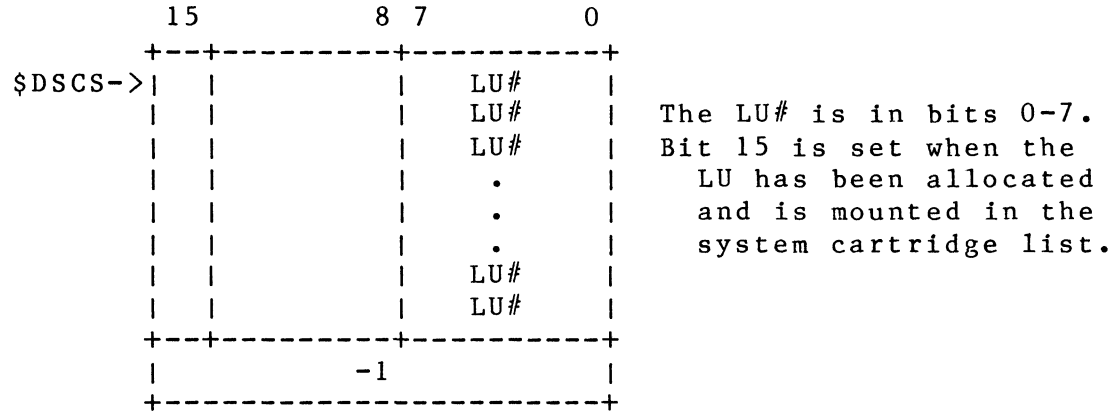
SA	SA..	SA..,8
SE	SE..	SE..,8
SL	LU..	LU..,8
SM	SM..	SM..,8
SP	SP..	SP..,8
ST	ST..	ST.DU,8
SV	SV..	F.UTM,8
SY	SY..	SY..,8
TE	TE..	OPMES,8
TL	TL..	TL..,8
TR	TR..	TR..,8
WH	WH..	WH..,8
??	??..	??..,8
**	----	C.TAB,8
*^	----	C.TAB,8
*<null>	----	C.TAB,8

DISC ALLOCATION AND CARTRIDGE ADDRESSING

DISC POOL

Systems with a Session Monitor may use a disc LU pool scheme for disc space allocation. The System Manager specifies in the account file, the logical unit numbers of the disc cartridges to be included in this pool. At system initialization, the disc pool is set up in the block of memory (System Available Memory or a Table partition) allocated to the Session Monitor. The table consists of a one-word entry for each logical unit in the pool. Each entry contains the LU number in bits 0-7 and a flag in bit 15 to indicate if the disc is allocated. The last entry will be a -1 indicating the end of the pool. The address of the disc pool is in the entry point $DSCS.

DISC POOL



If session is not initialized, $DSCS will have a value of -1. Session monitor systems that do not use the disc pool will have the value of $DSCS set to 0.

Systems with the disc pool scheme allocate disc space in the following ways:

1) AC,crn,P/G: An available LU from the disc pool is allocated and marked busy. It is mounted in the cartridge list as a private or group disc with crn as its cartridge reference number. (This command is valid only for Session Monitor systems with a disc pool).

2) MC,lu,P/G: Whether or not LU is in the disc pool, it will be mounted as a private or group cartridge (assuming it is not already mounted). If the LU is in the disc pool, it's entry is marked busy.

3) RU,READT: An available LU from the disc pool may be automatically allocated and restored to from magnetic tape using the READT utility program.

When a disc is mounted for a session user, the disc LU is posted in the list of mounted discs in that user's SCB. If an entry for the disc LU does not already exist in the SST (Session Switch Table in the SCB), one is added.

The DCMC subroutine is the only routine that modifies the disc pool. Whenever a session user allocates disc space with one of the above 3 methods, DCMC performs the search on the disc pool for an available LU or for the specific LU requested. If found, the busy bit on that entry is set. Similarly, any dismount causes the pool to be searched and if an entry is found, the busy bit is cleared.

A third function performed by DCMC on the disc pool is an "honesty" check. If DCMC receives a request for disc space, and there doesn't appear to be any available, it checks the validity of the busy bits against discs mounted in the cartridge directory. The disc pool is corrected if necessary.

PRIVATE, GROUP, SYSTEM AND NON-SESSION DISCS

In systems with a Session Monitor, there are four types of discs; private, group, system and non-session. When a disc is mounted, an ID is put in the fourth word of the cartridge list entry. This ID identifies what type of disc it is and to whom it is mounted.

Non-session discs:

When a user who is not operating under session control mounts a disc, the ID 0000 is placed in the cartridge directory entry. This disc may be accessed by any non-session user.

Private discs:

When a user who is under session control mounts a disc, he may specify whether that disc is to be mounted as a private or group disc. If he specifies private, the user ID from his SCB is used as the ID in the cartridge directory entry. This way the disc is identified as belonging to a specific user, who is the only one who can access that disc. That is, a private disc is one that belongs to a particular user and is protected from all other users on the system. (see note below)

Group discs:

When a user who is under session control mounts a disc and specifies that it is to be a group disc, the group ID from his SCB is used as the ID in the cartridge directory entry. The disc is thus marked as belonging to a group of users (all those who have that group ID in their SCB's). No one outside of that group may access that disc. A group disc is thus a disc that belongs only to an identified group of users and is protected from other users on the system.

System discs:

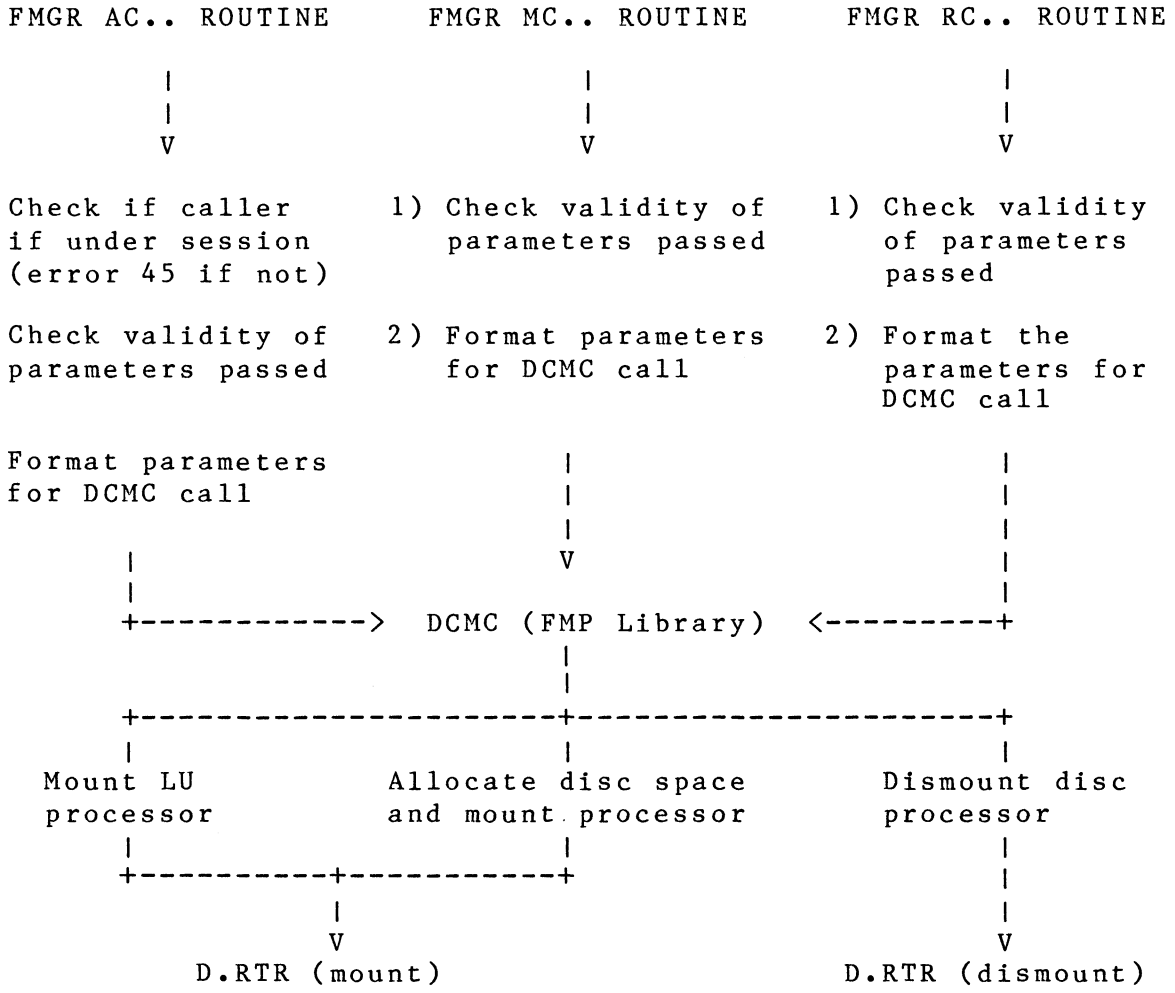
The term system discs really denotes two types of discs. First, system disc means LU 2 or 3, the discs on which reside the system, system tracks, etc. Second, system disc is used to mean Session Monitor "global" disc. The entry in the cartridge directory for a system disc (either of the above types) contains the ID 7777B. This ID is the user ID in the System Manager's SCB. This implies that only the System Manager can mount or dismount a system disc. LU 2 or 3 may be read by any user in the system. However, they may be modified only by the System Manager or non-session users. (See note 2 below). "Global" discs may be read from or written on by any user on the system.

NOTES:
1) This is complicated by the fact that the Session Monitor allows more than one person to be logged on to the same account or linked accounts at one time. This allows more than one user to be logged on with the same user ID at the same time. The effect of this is that a private disc may be mounted to and accessed by more than one user, given that the users have the same user ID.

2) There is one exception to this. Special privileges may be given internally for access to LU 2 and 3. This is done by setting an override flag (OVRD.). Among the routines that make use of this feature are SP.., SM.., ME.. and TR...

3) See the D.RTR Technical Specifications for a description of the use of ID's in file protection and cartridge access checking.

MOUNT/DISMOUNT INTERNAL STRUCTURE



DCMC PROCESSES

A. MOUNT a specified disc LU

1. Make a status request to make sure the logical unit requested is really a disc LU.

2. Search system cartridge directory to see if LU is already mounted.

   a. If LU is mounted to caller's ID (user ID from Session Control Block if disc is to be private, else group ID from SCB), then mount LU to caller's session (ADD) and return.

   b. If LU is mounted to caller's ID but set inactive, then activate LU (ACTIV) and return.

   c. If LU is not mounted in system cartridge directory, continue processing (step 3).

   d. All other cases, return error 12.

3. Do read (IMPRD) to get physical number of tracks on the subchannel. Use this for size if defaulted. Otherwise, make sure the size specified is less than or equal to that available.

4. Call FD.CK to see if there is a valid directory on the last track. If not, see that a CRN was specified and number of directory tracks is reasonable. (#dir trks < (total trks-dir trks) 8 + 1).

5. Search system cartridge directory for CRN to be assigned to the disc. Return error -12 if:

   a. System manager is mounting system disc with CRN the same as any CRN in the cartridge directory.

   b. Caller is mounting a disc with CRN the same as another disc mounted with his private or group ID or a system disc.

6. Check disc pool and mark LU busy if found (ALLOC).

7. Mount LU to caller's SCB (ADD).

8. Schedule D.RTR to mount LU to system cartridge directory.

9. If error return from D.RTR, free LU in disc pool (FREE), remove from caller's SCB (REMOV) and return error.

B. ALLOCATE a free disc and MOUNT it.

1. Scan cartridge directory to make sure CRN supplied can be mounted:

   (Note: If CRN is to be mounted as a private disc, ID1 = user ID from SCB and ID2 = group ID from SCB. Vice versa if CRN is to be mounted as a group disc.)

   a. If CRN is mounted in system CL (cartridge directory) with ID2, return error 12.

   b. If CRN is mounted in system CL with ID1 and to the caller's SCB, return error 12.

   c. If CRN is mounted as a system disc, return error 12.

   d. If caller is system manager and CRN is already mounted in CL to any ID, return error 12.

   e. If CRN is mounted to the system CL with ID1, but not to the caller's SCB, mount it to his SCB (ADD) and return.

   f. If CRN is mounted to the system CL with ID1 and to caller's SCB, but is inactive, activate it (ACTIV) and return.

2. If system CL is already full, return error 62.

3. Read disc pool into external buffer SM.BF (RDSPL).

4. Search disc pool for an available disc:

   a. If next entry is the end of the pool and haven't already done honesty check, do it (POLCK) and begin search at beginning again.

   b. If next entry is the end of the pool and have already done 2 passes, return error 64.

   c. Do XLUEX status request on disc LU. Skip this entry if bad LU or not a disc.

   d. Do read on disc LU to get number of sectors per track and size of subchannel (IMPRD).

   e. If disc does not have at least as many sectors per track or as many tracks (SIZE) as requested, skip this entry.

   f. Make sure number of directory tracks requested is reasonable with respect to size. The test is:

      #dir trks < (total trks-dir trks)

If not, skip this entry. If caller specified both size and number of directory tracks, and this test failed, return error 56.

g. Note: If this is a call from the READT utility, size data is kept and if an entry cannot be found that fits the requested size, the next best fit for size is used. This is done only for READT.

5. Mount disc to caller's SCB by adding entry to discs mounted list, SST if necessary and incrementing dynamic discs mounted counter (ADD). In case of a SST conflict, try another disc pool entry.

6. Set busy bit on disc pool entry (ALLOC).

7. Schedule D.RTR to mount the LU to the system cartridge directory.

8. If an error is returned from D.RTR, free the LU in disc pool (FREE), remove from caller's SCB (REMOV) and try another disc pool entry.

C. DISMOUNT a specified disc

1. Search system cartridge directory for LU or CRN. Make sure entry contains caller's private or group ID. If not there, return error 54. If caller is system manager, make 2 passes in search of CL and let him dismount any disc on the second pass.

2. Make sure disc is mounted to caller's SCB (SCBCK). If not, continue search of system CL. If disc is inactive, set active bit so D.RTR can find it.

3. If "RR" option not specified, just set disc inactive in caller's SCB (ACTIV) and return.

4. Check to see if this disc is mounted to any other active session (SCAN). If it is, just dismount it from caller's SCB (REMOV) and return.

5. Schedule D.RTR to remove entry from system cartridge directory.

6. If dismounting LU 2 or 3, remove disc from SCB (REMOV) and then remount by entering the MOUNT a specified LU process.

7. Remove busy flag on this disc's entry in disc pool (FREE). Dismount disc from caller's session (REMOV) and return.

DCMC INTERNAL ROUTINES

1. SERCH - Searches cartridge directory for a CRN or LU.

   Cartridge directory must already have been read into a buffer. Must specify address of start of search.

2. SCBCK - Checks to see if a disc is mounted to a caller's Session Control Block.

   Scans discs mounted list in SCB for disc PENLU. Returns information on whether or not it is found, and whether active.

3. POLCK - Scans disc pool and cartridge directory and frees any entries in disc pool that are marked as busy but aren't really mounted.

4. IMPRD - Makes a read to a disc unit to get number of tracks on the subchannel and the number of sectors per track. If disc is LU 2 or 3, these values are calculated from base page and no read is made.

5. SPECF - Sets up the 16- word cartridge specification entry for the pending disc. SPECF does an impossible read (IMPRD) to get the number of sectors per track. If this read fails, SPECF does not return normally. It jumps to the CLNUP section.

6. ACTIV - Marks a logical unit active or inactive in the discs mounted list.

   ACTIV reads a "clean" copy of the SCB, marks it and posts it back using the system routine, $SMVE.

7. SCAN - Scans the discs mounted list of each active SCB to make sure the LU to be dismounted is not mounted to any other SCB.

8. RESLV - Resolves an indirect address.

9. RDSPL - Reads disc pool into external buffer SM.BF. Uses the system routine $SMVE to read the disc pool. If there is not disc pool, RDSPL puts a -1 in the first word of the buffer so that the first entry looks like the end.

10. WDSPL - Updates an entry (1 word) in the disc pool. Uses $SMVE to write into the disc pool.

    Note: WDSPL should never be called unless there really is a disc pool.

11. REMOV - Removes a disc LU from the discs mounted list in the SCB. If bit 15 is set on the entry in the discs mounted list, the LU is also removed from the SST. The dynamic discs mounted counter is decremented.

12. ADD - Adds a disc LU to the discs mounted list in the SCB. If the LU is not already in the SST, an entry is made for it there and bit 15 is set on the entry in the discs mounted list (to indicate that the SST entry should be removed later). The dynamic discs mounted counter is incremented.

13. ALLOC - Allocate an entry in the disc pool by setting the busy bit on the entry (bit 15).

14. FREE - Frees an entry in the disc pool by clearing the busy bit (bit 15) on the entry.

Error Returns

AC..	45	Session command only
	50	Not enough parameters
	53	Illegal label
	56	Bad parameter
MC..	43	LU not in SST
	50	Not enough parameters
	53	Illegal label
	56	Bad parameter
RC..	50	Not enough parameters
	56	Bad parameter
DCMC	-1	Disc error
	12	Disc already mounted Lu is already mounted in CL to someone else, or CRN is already mounted with caller's private or group ID, or with system ID
	50	Not enough parameters
	53	Bad label (6-character disc label)
	54	Disc not mounted
	56	Bad parameter
	62	Already 63 discs mounted in CL
	63	Discs mounted list in SCB is full. Caller has already mounted his limit of discs.
	64	No discs available from disc pool
	65	Session LU conflict in SST. Disc LU to be added to SST is already a session LU in this SST.
	66	SST is full - no entry available to add disc

D.RTR INTERFACE

MOUNT PROCESSOR CALLING SEQUENCE

P1 ID (Program's ID segment address)
P2 13 + override bits from OVRD.
P3 -LU, 0 illegal
P4 SCB Address if mounting to session other than the one currently operating under.
P5 ID to which disc is to be mounted
(Bit 15 = 1, initialize the directory)

STRING: first 9 words of cartridge specification entry. This is passed only if disc's file system is to be initialized.

DISMOUNT PROCESSOR CALLING SEQUENCE

P1 ID (Program's ID segment address)
P2 11 + override bits from OVRD.
P3 -LU, 0 illegal
P4 SCB Address if dismounting from session other than the one currently operating under.

DCMC does most of the processing for disc allocation. D.RTR merely acts as a final bottleneck by searching for the entry or an empty spot for the entry, and rewriting the cartridge directory. All information needed to set up the entry is passed to D.RTR.

CL - CARTRIDGE LIST

CL.. calls the FMP routine FSTAT to read the cartridge directory. If the CLAL option is specified, the call to FSTAT is made to return the entire system cartridge directory. For each entry returned in the buffer from FSTAT, CL.. extracts the ID (4th word of each entry) and passes it to the subroutine PGS. to determine if the disc is mounted as a private, group or system disc. If the CLAL option was specified, the ID is passed to the subroutine ACNAM as well. ACNAM returns the name of the user or group associated with the ID. ACNAM will also return the directory entry number of the next account file directory entry containing a matching ID, or 0 if the end of the directory is reached without finding another matching ID. This allows CL.. to print multiple names if the ID is associated with more than one account. As each name is found, CL.. formats the line of cartridge directory entry information and outputs it.

If not run under session and the CLAL option is not specified, CL.. does not print P/G/S information, nor does it print user and group names.

PGS.

This Session Monitor subroutine determines whether a disc ID represents a private, group or system disc.

Calling Sequence:

```
JSB PGS.
DEF *+4
DEF IDCB
DEF ID
DEF PGS
```

Parameters:

IDCB — Open Account File Data Control Block

ID — The ID number originally assigned to a user or group account by the Session Monitor ACCTS program and posted to the cartridge directory when a disc is mounted by the user or group.

PGS — On return, = 1 if private ID
= 2 if group ID
= 3 if system ID
= 0 otherwise

Method:

1. If ID = 0, return PGS = 0.
2. If ID = 7777B, disc mounted as system disc. Return PGS = 3.
3. If ID greater than or equal to lowest private ID used (word 23 of account file header), disc mounted as private. Return PGS = 1.
4. If ID less than or equal to highest group ID used (word 24 of account file header), disc mounted as group disc. Return PGS = 2.
5. Otherwise, return PGS = 0.

ACNAM

This is the Session Monitor subroutine to return the account name (either user.group or group) associated with an ID.

Calling Sequence:

```
JSB ACNAM
DEF *+7
DEF IDCB
DEF ID
DEF PGS
DEF IREC
DEF BUF
DEF BUFL
```

Parameters:

IDCB — Open Account File Data Control Block

ID — The ID number originally assigned to a user or group account by the Session Monitor ACCTS program and posted to the cartridge directory when a disc is mounted by the user or group.

PGS — Set by the caller to 1 if private, 2 if group, 3 if system (see PGS. routine).

IREC — Set to 1 by the caller on the first call for a given ID. ACNAM returns the next directory entry number (2nd, 3rd, etc.) of the account file record containing a matching ID. The caller should continue to call ACNAM until ENTND is set to 0 by the subroutine.

BUF — An 11-word buffer into which ACNAM stores the name of the next user or group whose ID matches ID.

BUFL — Number of characters in account name in BUF.

Method:

1. If PGS is system or private, search the account file directory for user entries (Word 1 of directory entry negative) beginning with IREC. If PGS is group, search the account file directory for group entries (Word 1 of directory entry positive).

2. If a matching ID is found, write the user or group name to BUF.

3. Continue searching for another matching ID. If one is found, return the corresponding directory entry number in IREC. Otherwise, return IREC = 0.

DL - DIRECTORY LIST

Sequence of Operations

1. Call SESSN to determine if in session.
2. If in session, call ISMVE to read the user ID from the SCB.
3. If user ID = 7777B (System Manager), call FSTAT to read the entire cartridge directory. Otherwise, call FSTAT to return only the user's private and group cartridges, and the system cartridges.
4. If the disc LU or CRN was specified, call CKDID to check that is in the FSTAT buffer.
5. Lock and unlock the disc to clear any invalid open flags.
6. Call DR.RD (in FM.UT) to read the next directory block.
7. Format and list the non-purged entries in the directory.
8. Continue reading the directory until the end. If no disc LU or CRN was specified, get the next entry from the FSTAT buffer and continue.

PROGRAM RENAMING

Sequence of Operations

The following section describes the sequence of operations which take place in renaming and scheduling a program as a result of the :RU command. The reader is directed to the source listing of the FMGR action routine, RU.. for more details.

1. If not under session and not MTM, skip renaming.

2. If N.OPL is "IH", skip renaming.

3. Call .RENM to rename the module.

4. Call ID.A to determine if generic ID segment exists.

5. Check the "don't copy" bit (bit 11) in ID segment word 33 and if set, use original name.

6. Call EXEC to schedule the program. On a successful return from EXEC, and if RPSW is set, call IDRPD to remove the ID segment and then release any tracks.

7. If error return from EXEC, then if an ID segment was built by .RENM (RPSW = 1), then 1) return error 49 - can't run RP'ed program or partition too small, 2) call IDRPD to remove the ID segment and 3) release any tracks.

8. Call OPEN to find the type 6 file with the generic name. If the disc is not specified, search LU 2 first, then LU 3 (if necessary and if it exists).

9. If the type 6 file is not found, return error 67 - program not found.

10. If found, then check the "don't copy" bit in the type 6 memory image and if set, use the generic name of the program.

11. Call IDRPL to RP the program with the appropriate name.

12. Close the type 6 file.

13. Set RPSW to 1 to indicate an ID segment was built and schedule the program.

A number of routines are included in the File Management Package allowing for the automatic program renaming feature of FMGR. These routines are SESSN (checks if running under the Session Monitor), .RENM (to rename programs), IDDUP (duplicates an ID segment), IDRPL (does an RP,program) and IDRPD (does an RP,,program).

SESSN

Purpose: To determine if a program is in session

Program Type: 7

Entry Point: SESSN

Externals: None

Calling Sequence:

```
        JSB SESSN
        DEF *+2
        DEF ID          ID segment address of pgm

    <RETURN>    E=1 indicates not in session
                =0 indicates in session
                B=  ID segment session word
                    (address into SCB)
```

Sequence of Operations:

1. Offset to session word in ID segment (word 33).

2. If session word is negative or 0, return not in session (E=1).

3. Return ID segment session word in B-register and E=0.

.RENM

Purpose: Renames a program run with FMGR :RU,program

Program Type: 8

Entry Point: .RENM

Externals: .DFER,.ENTR,CONV.,IDDUP,IDSGA,S.TTY

Calling Sequence: CALL .RENM (NNAM,IERR,RPSW)

where:

NNAM = 3-word buffer containing generic name, returned with new name.

IERR = non-zero if unsuccessful rename.

RPSW = 1 on return if ID segment produced.

Sequence of Operations:

1. Convert the terminal logical unit number in FMGR global S.TTY to ASCII and merge into new name.
2. Replace blanks with ASCII dots.
3. Call IDSGA (ID.A) to determine if the generic (old) name exists in an ID segment.
4. If not, return the new name and set RPSW = 0.
5. Call IDDUP to reproduce the ID segment. If successful, return new name and set RPSW = 1.
6. If ID already exists, then compare disc track/sector/lu word in its ID segment to the generic ID and exit ok if they match. Otherwise, return duplicate name.

IDDUP

Purpose: Duplicates an ID segment already in an RTE system, giving it another name at the same time.

Program Type: 6

Entry Point: IDDUP

Externals: $LIBR,$LIBX,$IDEX,$OPSY,.ENTP,.OWNR,IDSGA,NAM..

Calling Sequence: Callable as a FORTRAN function or call

```
CALL IDDUP(IDNAM,NWNAM,IERR,OID,NID)
or
IF(IDDUP(IDNAM,NWNAM,IERR,OID,NID).NE.0)
   GO TO IERROR
```

where:

IDNAM = An existing program name in the system. Must have been ":RP,IDNAM" or be a permanent program in the system.

NWNAM = New name for the newly created ID segment. Must be unused name in the system.

IERR = Return error code. (Optional)

OID = Return address of existing ID segment. (Optional)

NID = Return address of new ID segment (Optional).

Error codes:

-15 = Illegal name. NWNAM does not conform to namr syntax rules.

14 = Required ID segment not found. Either IDNAM does not exist, or no blank ID segment is available for NWNAM.

17 = ID segment not set up by RP. The program type is not 2 or 3. (Type 4 allowed if RTE-IV).

23 = Duplicate program name with different disc address. NWNAM already exists.

IDRPL

Purpose: Subroutine equivalent of FMGR ":RU,program"

Program Type: 7

Entry Point: IDRPL

Externals: $LIBR,$LIBX,$IDEX,$OPSY,.ENTR,.OWNR,EXEC,IDSGA,NAM..

Calling Sequence: Called as FORTRAN function or call

```
CALL IDRPL (IDCB,IERR,NAME,NID)
or
IF (IDRPL(IDCB,IERR,NAME,NID).NE.0) GO TO IERROR
```

where:

IDCB = An open Data Control Block of the Type 6 file on LU 2 or 3.

IERR = Return error code.

NAME = 3-word buffer containing program name to put in ID segment.

NID = Return address of new ID segment (Optional).

Comments: IDRPL does not close the RP'ed file.
IDRPL uses only the first 10 words of the DCB.

Error codes:

-11 = Data Control Block (IDCB) not open
-15 = Illegal NAME. File name does not conform to namr syntax rules.
14 = Required ID segment not found. No blank ID segments are available.
16 = File must be and is not on LU 2 or 3.
19 = File was not set up by :SP on current system
23 = Duplicate program name.

Comments: Available ID segments are searched for in the following precedence:

A. If program type is 2 or 3:

1. Long ID without tracks
2. Long ID and don't care if it owns tracks

B. If program type is 5 (segment):

1. Short ID without tracks.
2. Long ID without tracks.
3. Short ID and don't care if it owns tracks.
4. Long ID and don't care if it owns tracks.

IDRPD

Purpose: Subroutine equivalent of FMGR ":RP,,program"

Program Type: 7

Entry Point: IDRPD

Externals: $LIBR, $LIBX, $IDEX, $OPSY, .DFER, .ENTP, IDSGA, MESSS, SESSN

Calling Sequence: Callable as FORTRAN function or call

```
CALL IDRPD (NAME,IERR)
or
IF (IDRPD(NAME,IERR).NE.0) GO TO IERROR
or
IERR=IDRPD(NAME)
```

where:

NAME = 3-word buffer containing program name to be deleted from system

IERR = Return error code.

Error Codes:

9 = ID segment not found
17 = ID segment was not set up by RP
18 = Program is not dormant

Comments: The program to be released must be dormant. Any disc tracks belonging to the released ID segment are given to the calling program which must release them. To make the system aware of their release, a CALL EXEC(5,-1) must be used to release the program-owned tracks.

ERROR PROCESSING

Normal FMGR error processing involves passing an error number from the FMGR action routine, through the main, to the MSS. routine in FM.CM. When running under session, the following events occur as part of MSS.:

A call is made to the Session Monitor subroutine, PTERR, which writes the eight-character error mnemonic (FMGRxxxx) to the session's Session Control Block. This error can then be reported and an expanded explanation retrieved with the session help (HE) command. Error message expansion with the ??.. routine also remains in effect.

Chapter 17

FMP AND D.RTR

GENERAL

This chapter describes the internal functions of the Real-Time Executive System File Management Package. This package maintains files on discs and provides an access method to all standard I/O devices.

The File Management Package is divided into three sections:

1. D.RTR, the directory management module.

2. FMGR, the operator interface.

3. FMP Library, a library of utility routines that provide file access from user routines.

D.RTR is a memory resident or disc resident program. It is the only program that may write on the cartridge or file directories. It searches for, creates, and purges files in response to requests from FMGR program or from the library routines. D.RTR also performs other functions required to maintain the directory. D.RTR may be scheduled only by another program; it is never called directly by the user.

FMGR is a segmented, privileged disc resident program. It provides FMP track assignment, system functions (initialization, packing, etc.) and an operator interface for file related functions.

FMP library is for use by user programs and provides program calls to perform file management.

REFERENCES

This document is intended to be used in conjunction with the D.RTR and FMP source listings. It also assumes that the reader is familiar with the SPL/2100 programming language and the RTE-IV and RTE-IVB operating systems. Some documents that may be useful are:

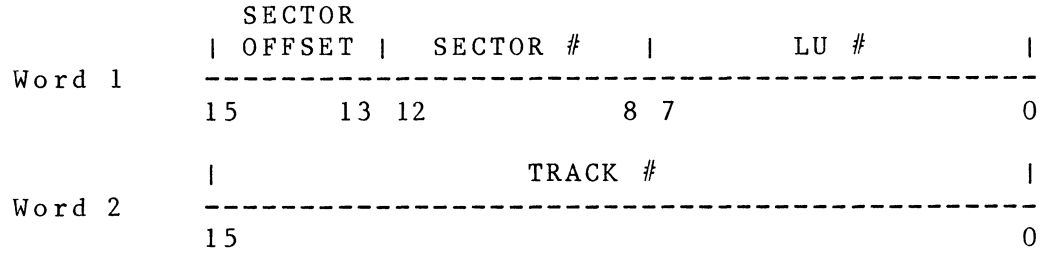
RTE-IVB Programmer's Reference manual	92068-90004
SPL/2100 manual	92100-93015
FMP listings	
D.RTR listings	

SUMMARY OF CHANGES TO FMP

A summary of changes to FMP is given in the following paragraphs.

NUMBER OF TRACKS PER SUBCHANNEL

Previously, the File Management Package used 10 bits in word 1 of the DCB to address disc tracks. This restricted the number of addressable tracks per subchannel to 1024. In order to be able to support larger capacity discs when necessary, the number of addressable tracks per subchannel has been increased to 32767 by reshuffling words 0 and 1 of the DCB. The format for the directory entry address is as follows:



These two words give the lu, track, and sector address of the directory entry with the sector offset indicating the number of the entry within the sector. [SECTOR OFFSET * 16 gives the word offset].

CARTRIDGE ACCESS CHECK FOR SESSION MONITOR

The Session Monitor provides the session user with protection for the cartridges he has mounted. To do this, it is necessary to restrict the cartridges that a session user may access to only those mounted to his session control block (lu must be in the SCB discs mounted list) and to system "global" discs. D.RTR performs most of the checking to make sure a cartridge access is legal. However, FMP shares some of the responsibility. Some D.RTR calls require information from the FMGR global "$OVRD.". This global contains information that causes some of the access checks to be overridden. (See the FMGR chapter.) FMP routines that schedule D.RTR for these calls must pick up the information from OVRD. and pass it in the D.RTR schedule call.

See the D.RTR section of this chapter for the rules of cartridge access and some discussion of its implementation in D.RTR.

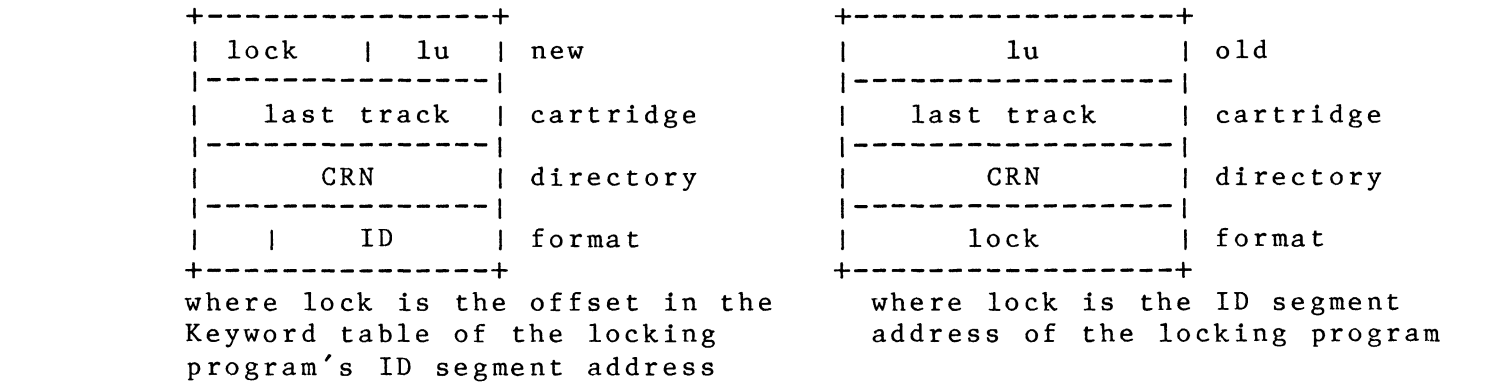
CARTRIDGE DIRECTORY CHANGES

FSTAT ROUTINE

The format and location of the cartridge directory has been modified to increase the number of discs that can be mounted at once. The cartridge directory is located in the system area on LU 2 and is 2 blocks in length. Externals $CL1 and $CL2 will contain its track and sector address. They are retrieved by D.RTR and FMP as follows:

```
       EXT $CL1,$CL2
TRACK  DEF $CL1      contains the track address of the directory
SECT   DEF $CL2      contains the sector address of the directory in
                     the lower byte
```

A 12-bit ID and a 4-bit CPU number will be in each entry in addition to the information in the old format. The ID is used by the Session Monitor to identify which discs are mounted to specific SCB's.



where lock is the offset in the Keyword table of the locking program's ID segment address

where lock is the ID segment address of the locking program

FMP routine FSTAT returns the cartridge directory to a user buffer. Additional parameters are added to the FSTAT call to indicate which format is desired and how large a buffer is being supplied. The caller may also indicate if he wants all discs or just private, group and system discs reported.

```
CALL FSTAT (ISTAT,ILEN,IFORM,IOP,IADD)
```

In order to preserve backward compatibility, if IFORM is zero or not supplied, the old cartridge directory format is used. If IFORM is non-zero, the new format is used. If ILEN is not specified, FSTAT will assume the buffer is 125 words long. However, the cartridge list will be truncated if the buffer supplied is not large enough. IADD will be returned non-zero to indicate not all the directory was written into the user buffer. When under session control only those private and group discs mounted to the session and system discs will be returned if IOP is zero. If IOP is non-zero, all discs in the system cartridge list will be returned.

EXTENDED FILE ADDRESSABILITY

The Data Control Block has been modified to expand file addressability to:

a) maximum (2**31)-1 records per file
b) maximum size of a non-extendable file is 32767 X 128 blocks

File size is expressed in positive number of sectors or in negative number of 128 block multiples. Word 5 of the DCB and Word 6 of the file directory entry contain this file size. Whenever possible, file size will be expressed in sectors rather than negative 128 block multiples. If size is requested in negative 128 block multiples, it will be converted to sectors when small enough and size in sectors will only be converted to negative 128 block multiples when it surpasses 32767 blocks.

The number of records per file has been increased by using a double word integer in the DCB for the current record number. Words 13 and 14 now contain the record number. The three bits previously used in Word 13 have been moved to Word 7.

Refer to Appendix M for the Data Control Block and the file directory entry formats.

NEW FMP ROUTINES

Because of the extended file addressing capabilities, there are some new FMP calls that use double word parameters. APOSN, POSNT, READF, WRITF, CREAT, CLOSE, and LOCF have single word parameters for specifying record number and size in sectors. These calls may be used whenever record, sector, or block number specified is less than 32767.

EAPOS, EPOSN, EREAD, EWRIT, ECREA, ECLOS, and ELOCF use double word parameters when dealing with record number or size in sectors. They may be used whenever record number is 0 through (2**31)-1 or when the size specified is 0 through 128 X 32767 blocks.

NOTE: One FMP subroutine handles both the regular calls and extended calls for a function. For example, CREAT and ECREA are simply 2 entry points in the same routine. READF, EREAD, WRITF, and EWRIT are all entry points into the same routine. All arithmetic in the FMP subroutines is done to handle maximum file size and record numbers. A preprocessor converts the incoming parameters to the correct format.

CALLING SEQUENCES FOR USER INTERFACE ROUTINES

The user interface routines are indexed by entry point rather than name. All these routines are either type 6 or 7 and are thus available to the user on line.

The calling sequences are in FTN where FTN calls are possible. Underlined parameters are optional.

Checking to see if enough parameters have been supplied and if optional parameters are being used are handled as follows:

The entry sequence in the FMP routine is coded as:

```
     P1   DEF DUM
     P2   DEF DUM
     P3   DEF DUM
          .
          .
          .
     PN   DEF DUM
     ENT  NOP
          JSB .ENTR
          DEF P1
```

where DUM is defined as:

```
     DUM  NOP
```

.ENTR will set up parameter addresses for all supplied parameters but not for unsupplied parameters; thus if P1 through P3 are required, the check to see if enough parameters have been supplied is:

```
               LDB P3
               CPB DFDUM
               JMP ERROR
```

where

```
         DFDUM DEF DUM
```

For optional parameters, the default is usually zero, thus

```
               LDA PN,I
```

will return either the parameter or the local zero. Likewise, if the routine is to return a parameter, the return is either to his location or to the local location. If the default is more complicated (for example, the READF buffer length option), the check described for insufficient parameters is used to determine if the parameter was supplied.

NOTE: A requirement of this scheme is that the entry sequence must be restored prior to returning from the call. That is, the values of P1 through PN must be restored to contain the address of DUM before returning.

Internal routines which require addresses require direct address unless otherwise indicated.

FMP ROUTINES

CREAT

CREAT routine creates and opens files of type 1 or above. It can create files up to 32767 sectors in size. CREAT sets up a skeleton directory entry from incoming parameters. It then schedules D.RTR, passing the directory entry, to actually create the file and the file directory entry on the disc. $OPEN is called to set up DCB parameters from the directory entry and open the DCB. To create larger files use ECREA call.

Refer to Appendix L for D.RTR calling sequence and skeleton directory format.

ENTRY POINT: CREAT,ECREA TYPE: 7

where Used: FMGR, user interface

CALLING SEQUENCE:

```
CALL CREAT(IDCB,IERR,NAME,ISIZE,ITYPE,ISECU,ICR,IDCBS,DUM,DUM,ILNAM)
                                      ----------------------------
```

where ISIZE - two word array. Word 1 is the size in 128 word double sectors (up to 16383) if positive. If -1, the rest of the cartridge up to 16383 blocks is allocated to the file. Word 2 is used only for type 2 files and is the record length.

DUM - undocumented parameter. Only function is as a place holder.

ILNAM - undocumented parameter. For internal use only. If set to 70707B, CREAT will skip the NAM.. call and allow a file with an illegal name to be created.

Refer to RTE-IVB Programmer's Reference Manual for other parameter descriptions.

External FMP References: CLOSE, $OPEN, NAM.., D.RTR, OVRD.

External System References: EXEC, RMPAR, .ENTR

Detailed Flow

A. Check for enough parameters.

B. Close DCB (CLOSE) in case it's open. Ignore not open error.

C. If ILNAM isn't set to 70707B, call NAM.. to check that NAME is a legal FMGR name.

D. Make sure ITYPE is not type 0. If so return error -16.

E. Check legality of SIZE. For CREAT, SIZE must be less than 16384 blocks. If not, return error -30. If type 1, force record length to 128.

F. Add global OVRD. to D.RTR request code and set in P2. (OVRD. tells D.RTR that certain session monitor restrictions should be overridden).

G. Schedule D.RTR to create the file and set up the directory entry. Pass the skeleton directory entry using string passage.

H. Retrieve return string from D.RTR and exit if any D.RTR errors.

I. Call $OPEN to open the file to the DCB. $OPEN uses information returned from D.RTR in string to set up the DCB. Current position pointers are set up in the DCB and the update mode bit is set.

J. If type >3, write EOF in buffer and set DCB "written-on" flag.

K. If CREAT call, return file size in sectors in error return. If ECREA call, return file size in sectors in double word JSIZE.

L. Return.

ECREA

ECREA is the create routine used for files whose size is up to 32767 x 128 blocks. ECREA is another entry point in routine CREAT. However, ECREA uses a double integer parameter for file size.

ENTRY POINT: ECREA TYPE: 7

WHERE USED: user interface, FMGR

CALLING SEQUENCE:

```
CALL ECREA(IDCB,IERR,NAME,ISIZE,ITYPE,ISECU,ICR,IDCBS,JSIZE,DUM,ILNAM)
                                      ------------------------------
```

where IERR - return parameter for negative error code. If ECREA was successful, IERR contains 0.

ISIZE - 2 entry array. Each entry is a double word. The first entry contains the number of blocks to be created. If -1, the rest of the cartridge is allocated to the file. If <-1 it is -double word number of 128 block multiples to be created in the file. Entry 2 is used only for type 2 files and contains the record length in words.

JSIZE - double word variable in which the actual file size created in +sectors or -tracks is returned if the creation is successful.

DUM - undocumented parameter. Used only as a place holder.

ILNAM - undocumented parameter. For internal use only. If set to 70707B, ECREA will skip NAM.. call and allow a file with an illegal name to be created.

NOTE: The option of ISIZE(1) < -1 is for internal use only. It is not documented for outside users.

Refer to RTE-IVB Programmer's Reference Manual for other parameter descriptions.

See CREAT routine for detailed flow.

CRETS

CRETS routine creates and opens scratch files given a scratch file number. It calls routine ECREA to do this. If the scratch file already exists, CRETS purges it through a call to D.RTR and then trys the create again.

ENTRY POINT: CRETS TYPE: 7

WHERE USED: user interface

CALLING SEQUENCE:

```
CALL CRETS(IDCB,IERR,NUM,NAME,ISIZE,ITYPE,IS,ILU,IBUF,JSIZE)
                              ----------------------------
```

where NUM - the scratch file number to create. 0 through 99

NAME - 3-word array containing the new file's name which CRETS has created. NOTE: this is a returned parameter.

ISIZE - a two entry array. Each entry is a double word. The first entry is the file size in 128-word double sectors. The second entry is used only for type 2 files and is the record length (double word). The default size is 24 blocks.

JSIZE - double word return parameter for the size of the file created if successful.

Refer to RTE-IVB Programmer's Reference Manual for other parameter descriptions.

Detailed Flow

A. Check for enough parameters.

B. Calculate ID segment number of calling program. Convert to ASCII.

C. Combine ASCII ID segment number, CPU flag, and scratch file number (NUM) to form scratch file name.

D. Test bounds of NUM. If too large or small return error -38.

E. Call ECREA to create the file.

F. If -2 error (duplicate file name) is returned, schedule D.RTR to purge the existing scratch file and retry the create.

G. Return.

OPEN

OPEN is the file open routine of the file management package. OPEN schedules D.RTR, passing the file name to be opened, to find the file directory and set an open flag in it. $OPEN is called to set up the DCB parameters from the directory entry to open the file to the DCB.

ENTRY POINT: OPEN TYPE: 7

WHERE USED: user interface, FMGR, PURGE, NAMF

CALLING SEQUENCE:

```
CALL OPEN (IDCB,IERR,NAME,IOPTN,ISECU,ICR,IDCBS)
                          ---------------------
```

Refer to RTE-IVB Programmer's Reference Manual for parameter descriptions.

External FMP References: CLOSE, $OPEN, D.RTR, OVRD.

External System References: EXEC, RMPAR, .ENTR, SESSN, $SMID, ISMVE

Detailed Flow

A. Check for enough parameters.

B. Close DCB (CLOSE) in case it's open. Ignore not open error.

C. Set up to send OVRD. information to D.RTR in P2 by adding it to the request code. (OVRD. tells D.RTR that certain session monitor restrictions should be overridden.)

D. Schedule D.RTR to open the file. Send the file name to D.RTR using string passage.

E. Retrieve return string from D.RTR and exit if any D.RTR errors.

F. Call $OPEN to open the file to the DCB. $OPEN uses information returned from D.RTR in the string to set up the DCB. Current position pointers are set up in the DCB.

G. If OPEN protected (security code is negative) and security code specified doesn't match file security code, close file and exit.

H. If under session control and not the system manager and opening a file on LU 2 or 3 allow read access only.

I. If type=0 and subfunction option present (bit 3 in IOP set) replace subfunction in DCB.

J. Return

OPENF

OPENF routine opens a file or an lu. If a file name is passed, OPENF simply calls routine OPEN to open the file. If an LU is passed, OPENF sets up a type 0 DCB for the lu. A type 0 file is never created.

Entry Point: OPENF Type: 7

WHERE USED: user interface

CALLING SEQUENCE:

```
CALL OPENF(IDCB,IERR,NAME,IOP,IS,ILU,IBLK)
```

Where NAME - 6-character name array or LU to open. LU is in first word.

IOP - open option flag word. Options are:

Bit	Meaning if set
0	non-exclusive open
1	update open
2	force to type 1 open
3	use subfunction in bits 6-11 if type 0

Refer to RTE-IVB Programmer's Reference Manual for other parameter descriptions.

External FMP references: OPEN, CLOSE

External system references: XLUEX, .ENTR, LURQ

Detailed Flow

A. Check for enough parameters.

B. Close DCB (CLOSE) in case its open. Ignore not open error.

C. If NAME is a file name, call OPEN routine to open the file and return.

D. Make an XLUEX status check on the LU passed in NAME to get the device type.

E. If LU is a disc, return error -17.

F. If device is interactive, default subfunction to echo. Set LU and subfunction in DCB.

G. If tape reader or punch set EOF code to Leader. If mini-cartridge or type > 17 set EOF code to write EOF. Set EOF code for all others to page eject.

H. Set spacing code = both. Read/Write flag = both. Open mode = update open. Security code = match. Set open flag. Set record count to 1.

I. If open was exclusive and device is not interactive, lock it (LURQ).

J. If lock is successful, set DCB word 15 to 0. Otherwise set it to 1 = don't unlock.

K. Return.

PURGE

PURGE deletes a file from the file directory by opening it exclusively (OPEN), getting the file length, and closing the file truncating it to zero length (CLOSE).

ENTRY POINT: PURGE TYPE: 7

WHERE USED: FMGR, user interface

CALLING SEQUENCE:

```
CALL PURGE (IDCB,IERR,NAME,ISECU,ICR)
           ---------
```

Refer to RTE-IVB Programmer's Reference Manual for parameter descriptions.

External FMP References: OPEN, ECLOS, OVRD.

External System References: EXEC, .ENTR, SESSN, ISMVE, $SMID

Detailed Flow

A. Check for enough parameters.

B. Open file exclusively using OPEN.

C. If this is a type 0 file, or if DCB security code indicates mismatch, close the file and error exit.

D. If caller is under session control, allow purge of type 6 file only if caller is the one who SP'ed it. Word 39 of record 1 contains the private ID of caller who SP'ed the file. Get private ID from SCB using ISMVE.

E. If this is a type 6 file and caller is allowed to purge it, re-open the file with OVRD. bit 15 set to bypass cartridge access checks.

F. Get file length from DCB. Close the file and truncate to zero length, (ECLOS) and return.

NAMF

NAMF routine renames a file manager file. NAMF calls OPEN to do an exclusive open on the file in order to find the directory entry and assure no one else has the file open. D.RTR is scheduled to change the file name on the directory entry on the disc. The renamed file is then closed using CLOSE. Refer to Appendix L for D.RTR calling sequence.

ENTRY POINT: NAMF TYPE: 7

WHERE USED: FMGR, user interface

CALLING SEQUENCE:

```
CALL NAMF (IDCB,IERR,NAME,NNAME,ISECU,ICR,DUM,ILNAM)
                                 ------------------
```

where DUM - undocumented parameter. Its only function is as a place holder.

ILNAM - undocumented parameter. For internal use only. If set to 70707B, NAMF will skip the NAM.. call and allow the file to be renamed with an illegal name.

External FMP References: CLOSE, NAM.., OPEN, D.RTR, OVRD.

External System References: EXEC, .ENTR, RMPAR, SESSN, ISMVE, $SMID

Detailed Flow

A. Check for enough parameters.

B. If ILNAM is not set to 70707B, use NAM... to make sure NNAME is a legal FMGR name.

C. Open file exclusively using OPEN. Exit if open errors.

D. If caller is under session control, allow rename of type 6 file only if caller is the one who SP'ed it. Word 39 of record 1 contains the private ID of caller who SP'ed the file. Get caller's private ID from his SCB using ISMVE.

E. If this is a type 6 file and caller is allowed to rename it, reopen the file with bit 15 of override set to bypass the cartridge access checks.

F. Check the security code in DCB. If mismatch, close and exit.

G. Schedule D.RTR to change the file name, sending the new name to D.RTR using string passage.

H. Close the file using CLOSE and return.

READF/WRITF - READ FUNCTION

READF routine reads a record from an open file to a user buffer. File types 1 and 2 are random access positioned to a specific record if the position option is used. Because NUM is a single word parameter, READF can position up to record number 32767. Type 0 files are read to the user buffer using REIO. Type 1 files are read directly to the user buffer with an EXEC read request. Type >=2 files are read to the DCB buffer with R/W$ and then read from the DCB buffer to the user buffer using RW$UB. To position beyond record 32767, use EREAD call.

Entry Points: READF,EREAD
WRITF,EWRIT
Type: 7

Where Used: FMGR, user interface
POSNT, ONOFF
EX.TM

CALLING SEQUENCE:

```
CALL READF (IDCB,IERR,IBUF,IL,LEN,NUM)
                          ----------
```

where NUM - the record number to be read if type 1 or 2. NUM must be <=32767.

Refer to RTE-IVB Programmer's Reference Manual for other parameter descriptions.

External FMP References: RFLG$, R/W$, P.PAS, RW$UB, KIP, DXFR

External System References: EXEC, .ENTR, REIO, .DAD, .DMP, .DDI, .DIN, .DSB, .DDE

Detailed Flow

A. WRITF, EWRIT = -1 will be used as "this is a READ type call" flag. Fetch parameters using P.PAS and .ENTR.

B. For READF and WRITF calls, convert NUM to a double word integer.

C. Check for enough parameters.

D. Make sure DCB is open to the current program. If not, error
E. exit. Set reading flag (RFLG$) for reading or update.

F. If type 1, force the record length in DCB to 128.

G. For type 1 or 2: do random access positioning (implies EOF if original position is not in the file). If new position is not in the currently resident block, then write the DCB buffer if it was written on (R/W$).

H. For type 0:

1) Check read legality bits (word 6) in DCB.
2) Read the record using REIO.
3) Set return length.
4) Check status word for EOF.
5) Return

I. For type 1:

1. Round up request length to even 128 words and save in record count (in blocks).
2. Make sure the request is within the file.
3. If track switch, compute maximum words to be read on this access.
4. Call EXEC to read the record. Check for disc errors.
5. Update buffer pointer for the rest of the record.
6. If there is more to transfer, go to I-3.
7. Update the record count in DCB and return.

J. For Type 2:

1. If reading flag (RFLG$) is set and DCB buffer is empty, read to DCB using R/W$.
2. If request length is too short (less than record length), set skip count so rest of record will be spaced over.
3. Read the record to user buffer using RW$UB
4. Step record count in DCB and return

K. For Type >=3:

1. If the reading flag (RFLG$) is set and the DCB buffer is empty, read to DCB using R/W$.
2. If current position is at EOF, set EOF encountered flag in DCB. If already set, take error -12 exit; else step the record count. Set return length to -1 and return.
3. Read record length word using RW$UB.
4. If request length is less than record length, set skip for the difference.
5. Read the record using RW$UB.
6. Skip the rest of the record ($KIP) if skip count is set.
7. Read second length word (RW$UB).
8. Take error exit if twin length words don't match.
9. If reading flag is not set, write EOF in buffer and set "written on" flag in DCB (word 7, bit 0).
10. Step record count in DCB and return.

EREAD/EWRIT - READ FUNCTION

READ reads a record from an open file to a user buffer. EREAD is another entry in the READF routine. However, EREAD uses a double word parameter when handling record number so the position option may specify a record number 0 through (2**31)-1.

ENTRY POINTS:	EREAD,READF EWRIT,WRITF	WHERE USED:	FMGR, user interface POSNT, ONOFF
TYPE: 7			EX.TM

CALLING SEQUENCE:

```
CALL EREAD(IDCB,IERR,IBUF,IL,LEN,NUM)
                          ----------
```

where NUM - double word integer containing the record number to be read if type 1 or 2.

Refer to RTE-IVB Programmer's Reference Manual for other parameter descriptions.

See READF routine for detailed flow.

READF/WRITF - WRITE FUNCTION

WRITF routine writes a record from a user buffer to an open file. File types 1 and 2 are random access positioned to a specific record if the position option is used. WRITF can only position up to record 32767 because NUM is a single word parameter. Type 0 files are written from the user buffer using REIO. Type 1 files are written directly from the user buffer to the disc using an EXEC write request. Type >=2 files are written from the user buffer to the DCB buffer using RW$UB and then from the DCB to the disc using R/W$. To position beyond record 32767, call EWRIT.

ENTRY POINTS: READF, WRITF TYPE: 7
EREAD, EWRIT

WHERE USED: FMGR, user interface

CALLING SEQUENCE:

```
CALL WRITF (IDCB,IERR,IBUF,IL,NUM)
                          ------
```

where NUM - record number to be written if type 1 or 2. NUM must be <=32767.

Refer to RTE-IVB Programmer's Reference Manual for other parameter descriptions.

External FMP References: RFLG$, R/W$, P.PAS, RW$UB, KIP, DXFR

External System References: EXEC, .ENTR

Detailed Flow

A. READF, EREAD = -1 will be used as "this is a WRITE type call" flag. Fetch parameters using P.PAS and .ENTR.

B. For READF and WRITF calls convert NUM to double word integer.

C. Check for enough parameters.

D. Make sure the DCB is open to the calling program. If not error exit.

E. Check security code. Error if mismatch.

F. If this is an update, set reading flag RFLG$.

G. If type 1 file, force record length in DCB to 128.

H. For type 1 or 2: If EOF write, then exit; else do random access positioning (implies EOF if initial position not in file). If the new position is not in the currently resident block, then write the DCB buffer if it was written on (R/W$).

I. For type 0:

1. Check the write legality code (word 6) in the DCB.
2. If write EOF, send control request and return.
3. Write the record using REIO.
4. Increment the record count in the DCB and return.

For type 1:

1. Round up request length to even 128 words and save in the record count (in blocks).
2. Check that the request in within the file.
3. If track switch, compute maximum words to be written on this disc access.
4. Write the record using the rounded up record length (EXEC).
5. Check for disc errors.
6. If there is more to transfer for this record, go to J-3.
7. Update the record count and return.

For Type 2:

1. If the reading flag (RFLG$) is set and the DCB buffer is empty, read a block to the DCB (R/W$).
2. Write the record from the user buffer to the DCB using RW$UB.
3. Increment the record count and return.

For Type >=3.

1. If the reading flag (RFLG$) is set and the DCB buffer is empty, read a block to the DCB (R/W$).
2. If current position is at EOF, clear the reading flag (RFLG$).
3. If EOF write, write EOF in the buffer, set the "written on" flag in the DCB, increment the record count, and return.
4. If reading (update write) check that the request length matches the old record length. If not, error -5 exit.
5. Write record length word in the DCB buffer (RW$UB).
6. Write the record and then the twin length word in the DCB buffer (RW$UB).
7. If reading flag is not set, set EOF in buffer, set the "written on" flag in the DCB, increment the record count, and return.

EREAD/EWRIT - WRITE FUNCTION

EWRIT writes a record from a user buffer to an open file. EWRIT is another entry into the WRITF routine. However, EWRIT uses a double word integer parameter when handling record number so the position option may specify a record number 0 through (2**31)-1.

ENTRY POINTS: EREAD, EWRIT TYPE: 7
READF, WRITF

WHERE USED: FMGR, user interface

CALLING SEQUENCE:

```
CALL EWRIT(IDCB,IERR,IBUF,IL,NUM)
                          ------
```

where NUM - double word integer containing the record number to be written if type 1 or 2.

Refer to the RTE-IVB Programmer's Reference manual for other parameter descriptions.

See WRITF routine for detailed flow.

FSTAT

FSTAT reads the disc directory into a user supplied buffer. The user can optionally specify in what format he wants to see the directory. Default will be the old disc directory format. If the buffer supplied is not large enough, the CL will be truncated and IADD will return a nonzero value. 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.

ENTRY POINT: FSTAT TYPE: 7

WHERE USED: FMGR, user interface

CALLING SEQUENCE:

```
CALL FSTAT (ISTAT,ILEN,IFORM,IOP,IADD)
                  ------------------
```

where ISTAT - is the buffer into which the directory will be read.

ILEN - length of the user buffer. Default assumes 125 words.

IFORM - if 0, the disc directory will be written into the buffer in the old format (default).

- if non-zero, the disc directory will be written into the buffer exactly as it appears on the disc (new format).

IOP - if non-zero, when under session control all discs mounted to the system are to be returned in ISTAT.

- if zero (or default) when under session control, only those private and group discs mounted to the session and system discs are returned (in that order).

IADD - set nonzero if not all of the cartridge list could be returned in ISTAT.

Old Format:
WORD 1: LU#
WORD 2: last track
WORD 3: CRN
WORD 4: 0 if not locked, or
ID segment address of locking program

Refer to Appendix N for the new disc cartridge directory format.

External FMP References: SM.BF, UT.BF, GTSCB

External System References: EXEC, .ENTR, $CL1, $CL2, $SMID, $SMGP $SMDL, $SMST, $SMLK

Detailed Flow

A. Read the cartridge directory into external buffer UT.BF.

B. Clear out user's buffer ISTAT.

C. If IOP = 0 (report private, group, and system), go to F.

D. Get address of next CL entry. Make sure another 4-word CL entry will fit in user's buffer. If not set IADD non-zero and return.

E. Move 4-word CL entry from UT.BF to user's buffer. Reformat to old format if FORM is non-zero.

F. Repeat D-E until find end of cartridge directory. Then return.

G. Read caller's SCB (GTSCB). Put caller's private and group ID's and system ID (7777B) into ID table. If caller is system manager, put 7777B in table only once. If caller isn't under session control, put 0 and 7777B into ID table.

H. Scan CL once for each entry in the ID table.

I. Get next entry from CL. Make sure ID matches current ID in ID table and LU is mounted to caller's session (or is a system disc or nonsession disc when not under session control.)

J. Make sure another 4-word CL entry will fit in user's buffer. If not, set IADD non-zero and return.

K. Move 4-word CL entry from UT.BF to user's buffer. Reformat to old format if FORM is non-zero.

L. Repeat H-I-J-K until all scans of CL are complete. Return.

RWNDF

RWNDF performs the reset or rewind function. For type 0 files, an EXEC control request is issued to rewind the device. For type >0 files the DCB current position pointers are reset to the beginning of the file.

ENTRY POINT: RWNDF TYPE: 7

WHERE USED: FMGR, user interface

CALLING SEQUENCE:

```
CALL RWNDF (IDCB,IERR)
            ----
```

Refer to RTE-IVB Programmer's Reference Manual for parameter descriptions.

External FMP References: R/W$, RWND$

External System References: .ENTR, EXEC

Detailed Flow

A. Make sure the DCB is open to the calling program. If not, exit.

B. Set the record number in the DCB to 1.

C. If type 0 file, do an EXEC rewind and return.

D. Write current sector block if it has been written on (R/W$).

E. If the current position is not in the main file (i.e., in an extent), schedule D.RTR to do an extension open on extent 0 (the main file).

F. Reset the current position pointers in the DCB to the beginning of the file (RWND$).

G. Return.

POSNT

POSNT is the file position routine. Random access files are forward spaced using READF and back spaced using $KIP routine. Since NUR is a single word variable, POSNT will forward or back space only up to 32767 records. EPOSN will perform this function for larger number of records.

ENTRY POINTS: POSNT,EPOSN TYPE: 7

WHERE USED: user interface

CALLING SEQUENCE:

CALL POSNT (IDCB,IERR,NUR,IR)
--

where NUR - number of records to be forward spaced or back spaced. If 0, no operation. NUR must be <=32767.

Refer to RTE-IVB Programmer's Reference Manual for other parameter descriptions.

External FMP References: RFLG$, P.PAS, READF, $KIP

External System References: .ENTR, EXEC, .DNG, .DAD, .DDE, .DIS

Detailed Flow

A. If POSNT call convert NUR to a double word integer.

B. If NUR is zero, just exit no error.

C. Check for enough parameters.

D. Set reading flag (RFLG$-global parameter to indicate if reading or writing) and fetch DCB parameters (P.PAS).

E. Make sure the DCB is open to the calling program. If not, error exit.

F. Calculate the relative record number of the new position. If no change, return.

G. For Types 1 and 2: If the absolute record position is less than one, error exit. Otherwise set the record number in the DCB and return.

H. For Type 0, 3, and above forward space: call READF to read the required number of records (thus forward spacing) and return. Error exit if EOF is read.

I. For type 0 backspace:

1. Check spacing code (word 5) in DCB to make sure backspace is legal for this file.
2. Use an EXEC control call to backspace 1 record.
3. If an EOF is encountered (and not the 1st backspace), forward space 1 record and return EOF. This will position at the beginning of the file.
4. Decrement the record number.
5. If finished, return, else go to I2.

J. For Type 3 and above backspace:

1. If the current position is at EOF, clear the EOF bit in the DCB (word 7), decrement the record number in the DCB, and return if done.
2. Using $KIP, backspace over the record length word.
3. Using $KIP, backspace over the record and the twin record length word.
4. If lengths don't match, error -5 exit.
5. Decrement the record number in the DCB. Return if done, else go to J-2.

EPOSN

EPOSN is the file position call used when forward or back spacing 0 to (2**31)-1 records. EPOSN is another entry into the POSNT routine. With EPOSN, a double word integer parameter is used for specifying record number.

ENTRY POINTS: EPOSN, POSNT TYPE: 7

WHERE USED: user interface

CALLING SEQUENCE:

```
CALL EPOSN(IDCB,IERR,NUR,IR)
                         --
```

where NUR - double word integer containing the number of records to be forward spaced or backspaced. If 0, no operation.

Refer to RTE-IVB Programmer's Reference Manual for other parameter descriptions.

See POSNT routine for detailed flow.

APOSN

APOSN does absolute file positioning for files (type > 0). For random access files, LOCF is called to get the current file position information. After calculating relative block change, NX$EC is called to skip to the new position. Current position pointers in the DCB are updated. APOSN uses single word parameters so next record and next block values may be up to 32767. For larger values of next record and next block, use EPOSN call.

ENTRY POINTS: APOSN, EAPOS TYPE: 7

WHERE USED: user interface

CALLING SEQUENCE:

```
CALL APOSN (IDCB,IERR,IREC,IRB,IOFF)
                           --------
```

Where IREC - a 1-word variable containing the next record. IREC must be <=32767.

IRB - a 1-word variable for next block. IRB must be <=32767.

Refer to RTE-IVB Programmer's Reference Manual for other parameter descriptions.

External FMP References: KIP, NXEC, RFLG$, ELOCF

External System References: .ENTR, .DNG, .DAD, .DDI

Detailed Flow

A. If APOSN call, convert IRB and IREC to double word integers.

B. Set RFLG$ (global parameter set to indicating reading).

C. If DCB is not open to the calling program, reject the call.

D. If the file is type 0, reject the call.

E. Make sure enough parameters were passed.

F. If type 1 or 2 go to I.

G. Call LOCF to get the current record number and sector number.

H. Position to new block using $KIP and NX$EC. ($KIP is called only because it sets up DCB parameters by calling P.PAS and that must happen before calling NX$EC. NX$EC is called to position to the correct block and read it in case the absolute position is outside the current block.)

1. NX$EC may have to make an extent change using D.RTR

2. R/W$ is called to write the current block if it is written on.

I. Calculate offset and set the buffer pointer and the record number in the DCB (words 12,13,14).

J. Unconditionally clear EOF read flag.

K. Return.

EAPOS

EAPOS does absolute file positioning. EAPOS is another entry into the APOSN routine. With EAPOS, double word parameters are used when handling block numbers and record numbers. Next record and next block may be 0 through (2**31)-1.

ENTRY POINTS: EAPOS, APOSN TYPE: 7

WHERE USED: user interface

CALLING SEQUENCE:

```
CALL EAPOS(IDCB,IERR,IREC,IRB,IOFF)
                          --------
```

where IREC - 2-word variable containing the next record.

IRB - 2-word variable containing the next block.

Refer to RTE-IVB Programmer's Reference Manual for other parameter descriptions.

See APOSN for detailed flow.

FCONT

FCONT controls I/O devices via type zero files. The control requests are issued to the device by making an EXEC control call.

ENTRY POINT: FCONT TYPE: 7

WHERE USED: user interface, CNT.

CALLING SEQUENCE:

```
CALL FCONT (IDCB,IERR,ICNWD,IPRAM)
                            -----
```

Refer to RTE-IVB Programmer's Reference Manual for parameter descriptions.

External FMP References: (none)

External System References: EXEC, .ENTR

Detailed Flow

A. Reject call if the DCB is not open to the calling program or if this is not a type 0 file.

B. Make the EXEC control call.

C. If an EOF is encountered, error 12 exit.

D. Return

LOCF

LOCF returns file status and current position for file manager files with <=32767 records and that are <=32767 logical sectors in size. For larger files, use the ELOCF call for this function.

ENTRY POINTS: LOCF, ELOCF TYPE: 7

WHERE USED: FMGR, APOSN, user interface

CALLING SEQUENCE:

```
CALL LOCF (IDCB,IERR,IREC,IRB,IOFF,JSEC,JLU,JTY,JREC)
                          -------------------------
```

Where IREC - is the next record. This is a single word variable so IREC must be <=32767.

IRB - is the relative block of the next record. This is a 1-word variable too, so IRB will be <=16383.

JSEC - is the # of sectors in the file. File size must be <=32767 logical sectors.

Refer to RTE-IVB Programmer's Reference Manual for other parameter descriptions.

External FMP References: P.PAS

External System References: .ENTR, .DDE, .DMP, .DAD

Detailed Flow

A. Check for enough parameters.

B. Fetch DCB parameters using P.PAS.

C. If the file is not open to the current program, reject the call.

D. Calculate file size in double word sectors.

E. Set IREC and JSEC from DCB words. Go through PRMRT so that double or single word returns are made depending on whether call was to LOCF or ELOCF.

F. For file types 1 and 2, calculate the relative sector (IRB) and word offset (IOFF) from the record number.

G. For random access files, calculate IRB and IOFF.

1. IOFF=buffer offset (BUFPT-DCB) adjusted for control words (-16) and adjusted to 128-word block base (modulo 128).

2. IRB=size in blocks times number of previous extents plus the sector offset in this extent plus block overflow (# blocks left in this DCB buffer when word offset was adjusted to 128-word block base).

H. Set JTY,JLU and JREC from DCB words using PRMRT.

I. Return

ELOCF

ELOCF returns file status and current position for file manager files whose size is up to 32767 tracks or that have up to (2**31)-1 records. ELOCF is another entry into the LOCF routine. With ELOCF, double word integer parameters are used for returning file size, record number and block numbers.

ENTRY POINTS: ELOCF, LOCF TYPE: 7

WHERE USED: FMGR, EAPOS, user interface

CALLING SEQUENCE:

```
CALL ELOCF(IDCB,IERR,IREC,IRB,IOFF,JSEC,JLU,JTY,JREC)
                          --------------------------
```

where IREC - double word variable containing the next record.

IRB - relative block of the next record. This is also a double word variable.

JSEC - double word variable containing the file size in sectors.

Refer to RTE-IVB Programmer's Reference Manual for other parameter descriptions.

See LOCF routine for detailed flow.

CLOSE

CLOSE closes a file manager file and optionally truncates blocks from the main file or purges extents. CLOSE calls D.RTR to clear the open flag in the disc directory and calculate the new size if the truncate option was used. CLOSE clears the open flag in the DCB. This routine may be used to truncate up to 32767 blocks. To truncate more than 32767 blocks, use ECLOS call.

ENTRY POINTS: CLOSE, ECLOS TYPE: 7

WHERE USED: OPEN, CREAT, NAMF, PURGE, user interface

CALLING SEQUENCE:

```
CALL CLOSE (IDCB,IERR,ITRUN)
            ----------
```

where ITRUN - optional 1-word variable containing integer number of blocks to be deleted from the file at closing; if negative, extents are truncated only; if omitted or zero, the file is closed without truncation. ITRUN must be <=32767 blocks.

Refer to RTE-IVB Programmer's Reference Manual for other parameter descriptions.

External FMP References: R/W$, D.RTR

External System References: EXEC, .ENTR, RMPAR, .DNG, LURQ

Detailed Flow

A. If CLOSE call, convert ITRUN to a double word integer.

B. Check for enough parameters.

C. If this is a type 0 DCB set up by OPENF and the LU was locked, call LURQ to unlock it. Go to G.

D. Reject call if the DCB is not open to the calling program.

E. Check file type, current extent number and security code for truncate option. File must be type >0, must be in main extent, and security code must be correct to truncate. If not, the file is closed without truncation.

F. Schedule D.RTR to close and truncate the file. D.RTR removes the open flag from the file's directory entry and recalculates size for the truncation option.

G. Clear open flag in the DCB word 9.

H. Pick up D.RTR error code if there is one and return.

ECLOS

ECLOS closes a file manager file and optionally truncates blocks from the main file or purges extents. ECLOS is another entry into the CLOSE routine. With ECLOS, a double word integer parameter is used to specify number of blocks truncated.

ENTRY POINTS: ECLOS, CLOSE TYPE: 7

WHERE USED: PURGE, CREAT, user interface

CALLING SEQUENCE:

```
CALL ECLOS(IDCB,IERR,ITRUN)
           ----------
```

where ITRUN - optional double word variable containing integer number of blocks to be deleted from the main file at closing; if negative, extents are truncated only; if omitted or zero, the file is closed without truncation.

Refer to RTE-IVB Programmer's Reference Manual for other parameter descriptions.

For detailed flow, see CLOSE routine.

POST

The POST routine clears the DCB buffer by posting any data that needs to be written on the disc. It will in all cases clear the in-core flag (word 7 in DCB) so that the next file access will force a disc read.

ENTRY POINT: POST TYPE: 7

WHERE USED: FMGR, user interface

CALLING SEQUENCE:

```
CALL POST (IDCB,IERR)
          ----
```

Refer to RTE-IVB Programmer's Reference Manual for parameter descriptions.

External FMP References: R/W$

External System References: .ENTR

Detailed Flow

A. Reject call if DCB is not open to the calling program.

B. Call R/W$ to write the DCB buffer on the disc if it has been written on (modified).

C. Return

NAM..

NAM.. verifies that a 6 character ASCII string is a legal file manager file name. A legal name consists of characters from the 64 ASCII set exclusive of +,-, and : and must not start with a blank or a number or contain imbedded blanks.

ENTRY POINT: NAM.. TYPE: 6

WHERE USED: FMGR, CREAT, NAMF, FID.

CALLING SEQUENCE:

```
     DIMENSION  NAME(3)
              .
              .
              .
     IER=NAM..(NAME)
```

on return IER=0 if name is legal and -15 if name is illegal

External FMP References: (none)

External System References: $LIBR, $LIBX, .ENTP

Detailed Flow

A. If first character is a blank or numeric, error exit.

B. If character is colon (:), error exit.

C. If character is a blank, set blank flag and go to F.

D. If character is not between (blank) and (*) or (-) and (*) then error exit.

E. If the blank flag is set, error exit.

F. If more characters to check, go to B.

G. Exit-good name.

IDCBS

IDCBS returns the number of words in the DCB buffer actually used by FMP for data transfer and control. IDCBS looks in the DCB for the buffer length and returns buffer length + control words.

ENTRY POINT: IDCBS TYPE: 7

WHERE USED: user interface

CALLING SEQUENCE:

ISIZE = IDCBS(IDCB)

Refer to RTe-IVB Programmer's Reference Manual for parameter descriptions.

External FMP References: (none)

External System References: .ENTR

Detailed Flow

A. Reject call if DCB is not open to the calling program.

B. For type 0 and 1 files, return the control block length 16.

C. For type >=2, get DCB buffer size from word 7. Return size=buffer size + 16 (control block length).

$OPEN

$OPEN sets up the DCB control words. It uses information returned via string from D.RTR OPEN or CREAT process to set up the DCB. RWND$ is called to set up the current position pointers in the DCB (words 10-15). Refer to Appendix M for the DCB and file directory entry formats.

ENTRY POINT: $OPEN TYPE: 7

WHERE USED: CREAT, OPEN

CALLING SEQUENCE:

Set up A,B,E,O as follows:

A=DCB address

B=address of buffer containing words 4-8 of directory entry returned from D.RTR.

E=1 if type Type 1 override

O=1 if an update open

Words 0 and 1 of the DCB are set to the directory address.

```
JSB $OPEN
DEF IBLK
DEF #SECT
JMP ERROR          error return
normal return
```

where IBLK - length of DCB or 0

#SECT - word passed from D.RTR containing number of sectors per track

On a normal return:

A and B = garbage

External FMP References: RWND$

External System References: EXEC

Detailed Flow

A. Set update open bit if update open mode requested.

B. If type 1 override, force DCB type word to 1.

C. Set the extent # in DCB to 0.

D. If this is a type 0 file and type 1 override was specified, error -9 exit.

E. Move file type (except type 1 override), starting track, starting sector, file size and record length from the buffer to the DCB.

F. If the DCB buffer size specified is less that 1 block, force it to 1 block.

G. Calculate buffer size in words and set it in the DCB. (This is the size that will be used by FMP; it may be smaller than the user specified area.)

H. Set the number of sectors per track in the DCB.

I. Call RWND$ to set up the current position pointers in the DCB, words 10-15.

J. Set the record number in the DCB to 1.

K. Set the open flag word in the DCB to the ID segment address of the current program.

L. Return

P.PAS

P.PAS is used to set up parameter addresses or to move information from the call area.

ENTRY POINT: P.PAS TYPE: 7

WHERE USED: D.RTR, READF, RW$UB, LOCK, $KIP, POSNT

CALLING SEQUENCE:

Set E, B, and A registers as follows:

```
 E = 0       To set up call area
 E = 1       To set up other area; move from call area
 B = 0       To set up addresses only
 B = 100000  To move parameters
 A = first word address of other area

     JSB P.PAS
     DEC  N          N = number of words to be set up
     BSS  N          Call area buffer
```

Return: A = original A + N
B undefined

If P.PAS is called with B, E = 0 and A = address of the DCB, it will set up the call area to contain the address of the first N words of the DCB. If B = 100000, E = 0, the call area will contain the first N words of the DCB.

If B = 100000, E = 1, the call area is moved to the first N words of the DCB. If B = 0, E = 1, the first N words of the DCB would contain the addresses of the N-word call area. (FMP never uses this last sequence.)

Detailed Flow

A. Configure code to be a load or a load indirect according to the B register.

B. Get the count.

C. Get the call area address (destination).

D. Set up the call area using a load or a load indirect.

E. Return

RW$UB

RW$UB routine reads or writes one or more words to/from the DCB buffer. It uses NX$EC to position to the next block if necessary. $KIP forward or backward skips over words in the file. NX$EC positions the DCB to the relative block desired. This includes calling D.RTR to open the appropriate extent when necessary. NX$EC updates the current position pointers in the DCB.

ENTRY POINTS: RW$UB, NX$EC, $KIP

RW$UB entry point

CALLING SEQUENCE:

```
E = 1 for read
E = 0 for write
B = +number of words to transfer
A = DCB address

JSB RW$UB
DEF BUFR    buffer address - may be indirect
JMP ERROR   error return A = error code
normal return
```

On return the A-register will contain the last word transferred.

External FMP References: P.PAS, RWND$, R/W$, RFLG$

External System References: EXEC, .DAD, .DMP, .DDI, .DNG

Detailed Flow

A. If the transfer count = 0, take normal return, else set the count negative and save.

B. Set the read/write switch.

C. Set up the DCB addresses using P.PAS.

D. Calculate the # of words remaining in the DCB buffer (from the word offset to the end of the buffer).

E. Save the user buffer address - resolving any indirects.

F. Transfer a word and step the buffer pointers:

   For read, get a word from the DCB and put it in the user buffer. For write, get a word from the user buffer and put it in the DCB.

G. If the DCB is exhausted, go to I.

H. If all words have been transferred, update the word offset pointer in the DCB, set the "written-on" flag in the DCB (word 7, bit 0) if writing, and returno else go to F andg continue transferring.

I. If type 2 file and done transferring, go to H.

J. Set the "written-on" flag in the DCB (word 7, bit 0) if writing.

K. Call NX$EC to write the current block if the "written-on" flag is set, compute the next track and sector addresses, and read the new block if this is a read operation.

L. Reset the word offset pointer in the DCB and go to H.

$KIP entry point

CALLING SEQUENCE:

```
A = DCB address
B = number of words to skip (may be positive, negative or zero)

JSB $KIP
JMP ERROR     (A = error code)
normal return
```

External FMP References: NX$EC, P.PAS

External System References: .DAD

Detailed Flow

A. Save the skip count. Fetch the DCB addresses using P.PAS.

B. Compute the relative word and sector addresses.

C. If the relative sector address is not zero (new position is not in this DCB buffer), call NX$EC to get the correct block.

D. Set the new buffer pointer in the DCB and return.

NX$EC entry point

CALLING SEQUENCE:

A,B = double word relative block number (maybe negative)

```
JSB NX$EC
JMP ERROR     Error return
normal return
```

External FMP References: RWND$, R/W$

External System References: .DMP, .DAD, .DDI, .DNG

Detailed Flow

A. Convert relative block to change to sectors and save.

B. Call R/W$ to write the current block if it has been written on.

C. Calculate the relative sector and extent.

D. If the relative extent is not zero,

   1. If the file is type = 2 or <0, return -12 error (SOF EOF).

   2. Call RWND$ to open the desired extent and reset the current position pointers in the DCB.

E. Calculate the current track and current sector address and set in the DCB.

F. If RFLG$ (read flag) is non-zero, call R/W$ to read the new block.

G. Normal return.

NOTE: NX$EC shares DCB addresses with $KIP and RW$UB but does not fetch them itself. Therefore $KIP or RW$UB must be called prior to calling NX$EC. $KIP has a do nothing call for this reason.

RWND$

RWND$ sets or resets the current position pointers in words 10-15 of the DCB. If the extent specified is not the current extent, D.RTR is scheduled to open the desired extent and return its address. Refer to Appendix L for the calling sequence to D.RTR.

ENTRY POINT: RWND$ TYPE: 7

WHERE USED: RW$UB, $OPEN, RWNDF, READF, POSNT, APOSN

CALLING SEQUENCE:

```
A = the desired extent
B = DCB address
```

It is assumed that word 15 of the DCB (current extent) is set and that DCB words 0,1 (file directory address) and 3,4 (track and sector address of the file) are correct.

```
JSB RWND$
JMP ERROR     error return        A = error code
normal return
```

External FMP References: RFLG$

External System References: EXEC

Detailed Flow

A. If current position is in the extent we want, go to E.

B. Call D.RTR to do an extension open on the desired extent. D.RTR will return track and sector address of the extent.

C. If D.RTR error, exit with the error code in the A-register.

D. Put the track and sector address returned from D.RTR into the DCB in words 3 and 4.

E. Set the track and sector words (3 and 4) into the current track and sector address words (10 and 11) in the DCB.

F. Set the buffer pointer (location of the next word in the file; word 12) to the first word in the DCB buffer. Clear the read/write flags (word 7).

G. Set the extent word.

H. Return

R/W$

R/W$ writes the current DCB buffer if it has been written on (modified) or reads it. R/W$ is often called before reading into the DCB buffer or terminating a routine to assure that any information that has been updated in the DCB gets written to the disc. Routine D$XFR does the actual transfer from the buffer to the current disc address. D$XFR handles track overflow.

ENTRY POINTS: R/W$
D$XFR
D.R (ASCII D.RTR)

TYPE: 7
WHERE USED: CLOSE, READF, RW$UB, CREAT, OPEN, RWNDF, POST

R/W$

CALLING SEQUENCE:

```
E = 0 for write (conditional)
E = 1 for read (unconditional)
B = DCB address

JSB R/W$
JMP ERROR      Disc error (A = -1)
normal return
```

Detailed Flow

A. If request is to write the block and the DCB "written-on" flag (word 7, bit 0) is not set, the return, no error.

B. Set up for the read or write: A = length, B = DCB address, and BUFA = DCB buffer address.

C. Call D$XFR to do the read or write.

D. If error, exit A = -1.

E. If this was a call to read, set the in-core flag (word 7, bit 2) in the DCB to indicate the current block is in core. Clear the "written-on" and "EOF-read" flags in word 7. If read call, clear EOF read flag.

F. Return

D$XFR - reads/writes N words to/from the current disc address.

CALLING SEQUENCE:

```
     E = 0 for read
     E = 1 for write
     B = DCB address
     A = length in words to be read or written

     JSB D$XFR
     DEF BUFR     Buffer address (must be direct)
     JMP ERR      disc error A = -1
     normal return
```

Detailed Flow

A. Get the current track and sector address from the DCB.

B. Check for track overflow; that is, if there are more words to transfer than left on the rest of this track. If so use the rest of the track length for the transfer.

C. Call EXEC to do the read/write.

D. If disc error, take error exit A = -1.

E. If there was track overflow determined in B, reset the track/sector address (track = track + 1, sector = 0) and go to B.

F. Return.

SUMMARY OF D.RTR CHANGES

NEW CALLING SEQUENCES

D.RTR calling sequences take advantage of string passage to transfer information to and from calling programs. This eliminates the use of a system track to pass data in all but one case, the GEN (replace the whole directory) call. In this call large portions of the directory, up to one track, are passed and rewritten at a time. The strings passed are retrieved by the individual processors when they are needed. Refer to Appendix L for descriptions of the calling sequences and return parameters.

CARTRIDGE ACCESS CHECK FOR SESSION MONITOR

The Session Monitor provides the session user with protection for the cartridges he has mounted. To do this it is necessary to restrict the cartridges that a session user may access to only those mounted in his session control block and to system "global" discs. There are some exceptions to this general rule.

A. The system manager has access to all discs on the system.

B. Some internal subsystems (SMP, etc.) also need to access any disc on the system.

C. Transfer files may be set up by the system manager on LU 2 or LU 3. These may be run by any session user. Also any command in the transfer file may have read and write access to LU 2 and LU 3.

D. Message files may reside on system discs. A session user must be able to open those files to write a message to another user and to read his own messages.

These exceptions are communicated to D.RTR via certain bits set in the FMGR global OVRD.. Information from this global is picked up by FMP routines and passed to D.RTR in the schedule parameters. Global OVRD. has the following format:

bit 15 set - allow user access of all discs in system CL.

bit 14 set - allow user access to his private and group discs and read and write access to all system discs.

bit 13 set - allow user access to only the system discs (read and write).

D.RTR has the responsibility of checking that a cartridge access is legal. Whenever D.RTR is scheduled with a call that specifies a disc ID (+CRN or -LU), the disc access is checked according to the following rules.

A. If the caller is the system manager he may talk to any disc on the system.

B. If the caller is not under session control he may access system discs and non-session discs.

C. If the caller is under session control and no override is specified he may access his private and group discs and system "global" discs. He may also have read only access to LU 2 and LU 3.

D. If the caller is under session control and OVRD. has bit 14 set, he may access his private and group discs and all system discs (including write access to LU 2 and LU 3).

E. If bit 15 is set on OVRD., the caller is treated like the system manager and may have access to any disc in the system.

F. If bit 13 on OVRD. is set, the caller may access only system discs.

The discs in the CL are identified by LU, CRN, and an ID that is written in the CL entry when a disc is mounted. Non-session discs have an ID of 0000, system discs have an ID of 7777. Private and group session ID's are defined in each Session Control Block.

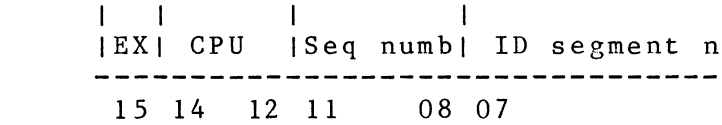
D.RTR searches the CL for a disc, checking the appropriate ID to be sure the disc access is legal. When the call to D.RTR specifies a disc, +CRN or -LU, D.RTR searches the CL for the CRN or LU and then checks the ID. When the disc is not specified, default case, D.RTR searches the CL for an ID that matches the caller's ID. On some calls to D.RTR more than one ID may be valid. If this is the case, D.RTR uses the private ID, group ID, and finally the system ID. A complete pass is made on the CL for each valid ID.

After a private or group disc entry is found, if under session control D.RTR checks that the LU is mounted not only in the system but also in the caller's SCB discs mounted list. If it is, the preprocessor continues with this CL entry.

Open Flag Clean-up

When files are left open to a program upon termination, their open flags in the files' directory entries must be removed so the files are not open to subsequent programs that get assigned the same ID segment.

OPEN FLAG FORMAT



where EX - if set means file is opened exclusively.

CPU - reserved to hold CPU number of processor of system that opened the file.

Seq numb - sequence counter from word 32 of ID segment of program opening the file.

ID seg num - ID segment number of the program opening the file.

Invalid open flags get cleaned up in two ways. First, whenever a disc is mounted either through D.RTR or READT, all open flag words are unconditionally zeroed out. Therefore, any disc newly mounted to the system should be free of invalid open flags. Second, whenever a file is accessed through D.RTR (open, rename, etc.) all open flags for that file are examined for validity. If not valid, they are cleared. Also, whenever a disc is locked through D.RTR (for PK, DL, IN, etc.) all open flags in the entire directory are examined for validity and cleared if invalid.

The validity check (in routine DORM) makes these tests.

1. If the flag doesn't belong to our CPU, leave it as is.

2. If the ID segment number is beyond the Keyword table length, clear the flag.

3. If the ID segment number is the same as the current caller to D.RTR, clear the flag. (This is to allow the same program to open the same file twice with only one open flag.)

4. If the sequence counter in the open flag does not match the current sequence counter in the ID segment from the open flag, clear the flag.

5. If the point of suspension in the ID segment indicates that the program is dormant, clear the flag.

Note: Even though this solution takes care of most of the invalid open flags, it is not a 100% solution. Some invalid open flags may still show up.

LARGER BUFFER

The buffer in D.RTR for the directories, BUF, is declared 6144 words in length - one 96 sector track. However it is structured to work with any declared buffer size >= 128 words. It will also handle any track size. For optimal efficiency, length of the declared buffer should be the track size of the disc with the largest track that will be accessed.

The amount of declared buffer that is used by D.RTR is determined by the following conditions:

1. If buffer >= track size, the track size is used.

2. Otherwise BUFFER SIZE = 128 * (n * 7 + 1)

   where n is a positive integer

   BUFFER SIZE <= supplied buffer size
   = maximum read length

that is, buffer size is exactly the track size or 128, 1024, 1920 2817, 3712, 4608, 5504, 6400, 7296, etc.

D.RTR MOUNT AND DISMOUNT PROCESSORS

Some of the responsibility for mounting and removing cartridges in the cartridge directory has been moved from FMGR to D.RTR. This is a result of a more complicated disc allocation scheme for Session Monitor users. It also eliminates the possibility of a race condition if more than one mount or dismount occur at the same time.

Specifically, the mount processor in D.RTR does the following:

A. D.RTR reads the cartridge directory and searches it to make sure the LU to be mounted is not there already.

B. Searches for the end of the directory to find the spot to add an entry and makes sure the directory is not already full.

C. Sets up the entry from information passed to D.RTR from FMGR subroutine DCMC and writes the new CL to the disc.

D. If a string containing the cartridge specification entry was passed to D.RTR, it clears the file directory on the disc being mounted.

E. Unconditionally clears all open flags in the directory.

The dismount processor in D.RTR is also called by FMGR subroutine DCMC and does the following:

A. D.RTR reads the cartridge directory and searches for the LU of the cartridge to be dismounted.

B. The entry is removed by moving all following entries in the cartridge directory up four words.

Refer to Appendix L for the format of mount and dismount calls to D.RTR and Appendix N for the cartridge directory format. Chapter 16 describes the disc allocation scheme and the functions performed by the FMGR mount and remove cartridge routines.

D.RTR OPERATION

DEFINITIONS

Set written-on flag implies setting location WCS non-zero. This means the buffer contents have been modified and need to be posted to the disc before the buffer is read into again.

Computing the next track and sector is always done by using the current file's address and size.

The notation (SUBR) implies SUBR is a subroutine that is used to perform the indicated function.

D.RTR operates by function. Each type of call to D.RTR (OPEN,CLOSE, PACK, etc.) performs its own function. These functions, one for every calling sequence to D.RTR, are described below. In addition, each time D.RTR is invoked it does several things that are common to each invocation. These actions are divided into pre and post functions.

PRE-ACTION PROCESSOR

Code begins at label BEGIN.

A. Fetch call parameters with cross map loads from the ID segment.

B. Get ID segment address from P1. Fetch sequence counter from ID segment.

C. Make sure D.RTR was scheduled with wait and that P1 contains the correct ID segment address.

D. Calculate SBFLN = length of BUF in 64-word sectors
SBFLM = largest 2 + (N * 7) <= SBFLM
KEYLN = length of Keyword table
IDSSW = address in BUF of SCB SST length word

E. If this is a mount or dismount call, get SCB address from P4. Otherwise force to 0 to get SCB of caller's session.

F. Read the SCB (GTSCB).

G. Set up ID table according to cartridge access requirements. See the Cartridge Access Check for Session Monitor section. ID table contains the ID's that discs that may be accessed by this caller may be mounted to. NOTE: For mount call want to search the entire CL to make sure the LU is not mounted.

For calls where a directory address is passed instead of an LU or CRN, it's already been through the cartridge access checks, so just search the entire CL for the LU.

H. Read the cartridge directory. (RDPS)

I. The function code (P2) is divided into even or odd or open and the disc ID is extracted from the appropriate parameter; i.e.,

open	P3	(-LU or +CRN)
odd	P3	(-LU or +CRN)
even	P3&P4	(directory entry address)

J. Search the cartridge directory:

1) If CRN was specified use section CRNSH. Scan CL for CRN. (SCAND) If one is found, the ID in the entry must match the current entry in the ID table. CRNSH can make as many passes through the CL as there are entries in the ID table. Entry must pass the SCBCK test (be mounted to SCB).

2) If LU was specified use section LU.SH. Scan CL for LU. (SCAND) If found, the ID in the entry must match one of the IDs in the ID table. Entry must also pass SCBCK test.

3) If no disc was specified and caller is system manager, just give him the next disc in the CL.

4) If no disc was specified and caller isn't the system manager, scan CL for an entry with the same ID as the current entry in the ID table - ID.SH section (SCAND). ID.SH can make as many passes through the CL as there are entries in the ID table. Entry must pass SCBCK test.

If not found, go to P.

K. See if the "found" disc is mounted to someone else (CK.LK) If it is, go look for another disc (default case) or return -13 error.

Further checks in the pre-processor or searches of the file directory for file names or room in the other processors may cause the disc found above to be rejected. This is essentially done by going back to (J) above (NEXT). This causes either a -6, -32, or -33 error if the disc was specified, or if the disc was not specified, the next available disc is set up.

L. If "found" disc is LU 2 or 3, can only use if caller is non-session, or if this is an open call, or if OVRD. is set, or if caller is being treated as system manager (includes even function codes).

M. Make sure LU is in caller's SST (SSTCK).

N. Do XLUEX status request to make sure LU points to a disc.

O. The request is decoded by trapping the OPEN request, and using a jump table for the rest of the requests.

P. If P2 = 13, go to the MOUNT processor.

Q. If this is not a generate call (P2 = 7) to update the disc directory (bit 15 on P4 is set), error -32 exit.

R. Read the new cartridge directory passed via string, set the "written-on" flag and post the first block (WCSR). Set pointers to the 2nd block, set "written-on" flag and exit to the Post-Processor.

NOTE: This form of the type 7 call is entered only after searching the directory for a disc, thus parameter P3 must not be findable. This is usually done by specifying -65 which is not currently a legal lu.

POST ACTION FUNCTIONS

The post processor passes normal completion parameters to the caller. The post processor's code begins at label CREX.

A. Set up the 5 return parameters (RPRM).

B. Write to the disc any buffers that are written on (WCSR).

C. Pass the return parameters to the caller (PRTN).

D. Terminate D.RTR serially reusable using an EXEC 6 call.

OPEN PROCESSOR

Code begins at label OPEN.

A. Read the passed string containing the file name into STRNG (RDPAS). Move file name into NAME (MOVE1).

B. Set up to read the first block of the file directory (SETDR).

C. Read the directory and search for the file to be opened (N.SHR).

D. If file is not found, reject this disc and go to the pre-processor to get the next one.

E. Set up the addresses for the directory entry (SETAD).

F. If security code from directory is negative and does not exactly match security code passed in, return error -7.

G. Check the open flags (FLAG).

H. If this is a scratch file purge and there is more than one open flag, return error -101. Otherwise jump to PURGE to purge it.

I. If there are already 7 open flags, error -8 exit.

J. If exclusive open request and this file already has an open flag, error -8 exit.

K. If the file is already exclusively open, error -8 exit.

L. Find an empty slot and set the open flag. Set the "written-on" flag.

M. Get zero to be returned as the first parameter and exit to the post processor.

CLOSE PROCESSOR

Code begins at label CLOSE.

A. Read the specified file directory entry (DIRCK).

B. Search the directory for the callers open flag. If not found, error exit -11.

C. Clear the open flag and set the "written-on" flag.

D. If the truncate code = 0 or if the file wasn't exclusively open, then exit to the post processor.

E. If the truncate code is positive (indicating purge extents only), search for an extent (EXSH) and go to J to purge it.

F. Calculate the new file size using double integer routines. If the new size is less than zero or an odd number of sectors, ignore the truncate request and exit. (Set "written-on" flag and exit to the post processor.)

G. If the new file size is zero, go to J to purge it.

H. Set the new size in the directory entry. (If size was >32767 sectors, divide by 128 blocks (256 sectors), round up 1 chunk if necessary, and make negative.)

I. If this is the last entry in the directory (LAST?), calculate the next available track and sector address (NXT/S). Go to CREAT-L to update next track and sector and exit. If this is not the last entry, search for any extents (EXSH) so they can be purged.

J. PURGE the entry by setting the purged flag (-1) in the first word of the directory entry.

K. If this is a type 6 file, treat it as not the last file (don't want to reuse type 6 file space.

L. If this is not the last entry in the directory (LAST?) continue searching for more extents (EXSH) and go to J to purge them.

M. If this is the last entry, reset the first word to zero to indicate end of directory and set the "written-on" flag.

N. Back up to previous entry (BACK).

O. If the entry is type 6 or the first (i.e. directory/specification entry) calculate the next track and sector addresses (NXT/S) and go to CREAT-L to update next track and sector and exit.

P. If the entry is type 0, set pointers back one more entry (BACK) (until a non-type zero file is found) calculate next available track and sector address and go to CREAT-L to update the track and sector and exit.

Q. If the entry is purged, set the first word to 0 and go to M.

R. Otherwise, calculate the next available track and sector address and go to CREAT-L to update the track and sector and exit.

CREATE PROCESSOR

Code begins at label CREAT.

A. Call RDPAS to read the skeleton directory in the passed string to STRNG. Call MOVE1 to move information to NAME.

B. If this is create of a type 0 file, save words 5,6,7,8 of the directory entry and replace with zero's. This is to get a file with length zero created, and then patch up the directory entry later.

C. Add sign bit to ID for exclusive open and set the extent # to 0.

D. Take size specified in words 6, 7 of the string and calculate size in +sectors or -128 block multiples and put it into the directory entry size word.

E. Read the file directory and search for a duplicate name. (SETDR, N.SHR). If one is found exit -2 error.

F. If a reusable entry is found (purged entry exactly the right size) RUSE sets up the directory entry as follows:

   1) Read the directory block containing the reusable entry into the file buffer (RWSUB) and set up addresses (SETAD).

   2) Set track and sector addresses, extent flag, and size in the skeleton directory.

   3) Move the directory entry into the buffer (MOVE1). Set the "written-on" flag and put open flag in the directory.

   4) Exit to the Post Processor.

G. If the directory is full, exit -14 error.

H. Call SETAD to set up the address for the file entry in the buffer. Move in the skeleton directory (MOVE1).

I. Set exclusive open flag in the file directory entry.

J. Insure the file is wholly above or below (for -1 option) any bad tracks.

K. If zero size on a "rest-of-the-disc" size request or not enough room on a specified size request, clear the first word of the directory entry and reject the disc. If this is an extension create, exit error -33. If not, return to the preprocessor.

L. Update next available track and sector. Set "written-on" flag.

M. If this was a type 0, replace words 5,6,7,8 that were removed earlier from the directory entry.

N. Set up return parameters with R1 containing the file size.

O. Read first directory block (SETDR, RDNXB), write the new track and sector address (DPMM) in the disc parameters.

P. Set "written-on" flag and exit to the post processor at B.

RENAME PROCESSOR

Code begins at label CNAM.

A. Call RDPAS to read the new name from the passed string into STRNG. Move name to NAME using MOVE1. Save 2nd copy using MOVE2.

B. Read the file directory and search for a duplicate name (SETDR, N.SHR). If one is found, error -2 exit.

C. Read the specified file directory entry (DIRCK).

D. If the file is not open exclusively to the caller, error -102 exit.

E. Move the new name into the directory entry (MOVE2).

F. Search for any extents and repeat E for each one found.

G. Exit to the Post Processor.

LOCK REQUEST PROCESSING

Code begins at label RLOCK.

A. If disc address is not specified, return error -101.

B. Search for any open flags on the disc (OPNCK). Exit with error -8 if any valid open flags are found.

C. Read the cartridge directory (RDPS).

D. Set the lock word in the CL entry. (DIRAD was set by preprocessor to the correct address of the CL entry.)

E. Post the cartridge directory (PSTCL) and exit to the post processor.

CLEAR LOCK PROCESSING

Code begins at label ULOCK.

A. Clear the lock word in the CL entry (DIRAD was set by the preprocessor to the address of the correct CL entry).

B. Post the cartridge directory (PSTCL) and exit to the post processor.

EXTENSION OPEN PROCESSOR

Code begins at label EXOPN.

A. Read the master directory entry (DIRCK) and set the open flag word to 0.

B. If the request is for extent 0 (i.e. the main), get the file type and exit to the post processor. (It is already set up.)

C. If extent number requested is >255, exit error -46.

D. Save the extent number for a possible create.

E. Search for the extent (EXSHR).

F. If not found and this was a read extent request, then error -5 exit. If it was a write extent request go create an extent CREAT-D.

G. If the required extent was found, get the file type (to be returned in the first parameter) and exit to the post processor.

H. Go search for another extent (at NSH4 - returns to F).

CHANGE THE WHOLE DIRECTORY PROCESSOR

Code begins at label GEN.

A. If the disc is not locked or not locked to the caller, error -102 exit. (TESTL)

B. Set up to access the first block of the file directory on the specified disc (SETDR).

C. Read the new directory block from the system data track (RDPAS) into the buffer.

D. Update the directory addresses (UDAD).

E. If end of the directory exit to the Post Processor.

F. If this is the first block, set up parameters (DPMM).

G. Set "written on" flag and write the block (WCSR).

H. Continue reading blocks of the new directory (go to RDPA2 - returns to D above).

REPLACE ONE DIRECTORY BLOCK PROCESSING

Code begins at label PACK.

A. If the disc is not locked or not locked to the caller, error -102 exit. (TESTL)

B. Set up to access the first block of the file directory on the specified disc (SETPR).

C. If the relative block to be replaced is zero, go to G.

D. Update the directory addresses (UDAD).

E. If end of directory, error -101 exit.

F. Repeat D&E until desired directory block is found.

G. Read the passed string into the buffer (RDPAS).

H. Set the track and sector address for the write (ADDR), "written-on" flag and exit to the Post Processor.

MOUNT PROCESSING

Code begins at label MOUNT.

A. Read the 9-word passed string in to BUF. (RDPAS)

B. Search the cartridge list for an empty entry. Return error -35 if the cartridge directory is already full.

C. Put lock word, LU, LTR, CRN, and ID from parameters and string passed to D.RTR into the new cartridge entry.

D. Post the new cartridge directory to the disc. (PSTCL)

E. If the initialize disc flag isn't set, clear off any open flags (CLOPF) and exit to the post processor.

F. Zero out words in BUF following the 9-word specification entry to write to the disc to clear out the directory.

G. Set up to access the beginning of the file directory (SETAD) and save the disc parameters from the passed in specification entry.

H. Update track and sector addresses (UDAD). Exit to post processor when end of the directory is found.

I. Set WCS and post the block (WCSR).

J. After the first block, start writing an entire block of zero's. Go to H.

DISMOUNT PROCESSING

Code begins at label DISMT.

A. If a specific disc was not requested, exit error -101.

B. Search for any open flags on the disc to be dismounted (OPNCK). Exit with error -8 if an open flag is found.

C. Re-read CL (RDPS), find CL entry, and clear it by moving all the following entries up one.

D. Post the new cartridge directory to the disc (PSTCL) and exit to the post processor.

CHANGE CL ENTRY PROCESSING

Code begins at label CHGCL

A. If a specific disc was not requested, exit error -101.

B. Read passed string (RDSTR) into the CL at location DIRAD (from pre-processor).

C. Post new cartridge directory to disc (PSTCL) and exit to the post processor.

D.RTR INTERNAL ROUTINES

This section discusses each routine internal to D.RTR giving the function, calling sequence, and a brief flow. None of these routines are global; they are all called only by D.RTR.

Index:

BAKUP

Function: Back up to the previous directory entry and set up addresses.

Calling Sequence

```
DIRA - address of current directory entry
JSB BAKUP
return
A,B,E undefined
```

Calls WCSR, RWSUB, SETAD

A. Check to see if current directory entry is the first in the block. If not, go to E.

B. Write the current block (WCSR).

C. Back up sector by 14. Take track boundary into account.

D. Read the block and position to one word beyond the end of it.

E. Subtract 16 words to position to beginning of previous entry.

F. Set up addresses into the directory entry (SETAD) and return.

CK.LK

Function: Check the lock in the cartridge directory entry. If invalid clear it.

Calling Sequence:

```
A = address of lock word in CL entry
JSB CK.LK
return P+1 - not locked or locked to caller or invalid lock
             that was cleared.
return P+2 - Valid lock belonging to someone else
```

A. Get lock word from CL and mask. If not locked, or locked to this caller, return.

B. If lock > keyword table length, clear it and return.

C. Get ID segment address from keyword table for the ID segment number. Position to point of suspension in ID segment. If zero, clear flag and return.

D. Return P+2 valid lock belonging to someone else.

CLOPF

Function: Clear any open flags found in the file directory of the pending disc.

Calling sequence:

```
ATRAK - last track
ALU   - lu of disc
JSB CLOPF
return
```

Calls SETDR, RDNXB, SETAD, WCSR

A. Set up to read beginning of the file directory (SETDR).

B. Read the next block (RDNXB). If end of directory, return.

C. Set up addresses for next entry (SETAD).

D. Zero out any open flags and set "written-on" flag.

E. Write the block (WCSR).

F. Go to B.

DIRCK

Function: Reads an addressed directory entry to the local buffer and sets up its parameters using MOVE1.

Calling Sequence:

```
JSB DIRCK
Normal Return
```

Calls SETPR, SETAD, RWSUB, MOVE1

A. If already set up for this disc, go to C).

B. Set up and read first block of the file directory (SETPR).

C. Get track, sector, offset from parameters passed in.

D. Read the directory block (RWSUB).

E. Set address to the directory (SETAD).

F. Move first 9 words of directory entry to local buffer (MOVE1).

G. Return.

DORM

Function: Checks an ID segment address to see if a program is dormant.

Calling Sequence:

```
Set A = open flag
Set B = buffer address of the flag in directory
JSB DORM
return P+1 - not dormant
return P+2 - dormant

On return,    A,E are undefined
              B as on entry
              B,I set to 0 if an invalid flag found
              WCS set # 0 if an invalid flag found
```

A. If open flag = 0 go to F).

B. If flag doesn't belong to our CPU return not dormant (P+1).

C. If ID segment number from flag is beyond table length, go to G.

D. If ID segment number from flag is the same as caller's ID segment number, go to G.

E. Get sequence counter from ID segment. If it doesn't match sequence number in open flag, go to G.

F. Get point of suspension from ID segment. If non-zero go to I (return P+1 dormant).

G. Clear flag and set WCS non-zero.

H. Set for dormant return (P+2).

I. Return.

DPMM

Function: Sets up/resets access parameters from the first directory block.

Calling Sequence:

```
Read the first directory block
E = 0 to move from the buffer to local
E = 1 to move from local to the buffer

JSB DPMM
Normal return
A,B,E undefined
```

Calls P.PAS

A. Set up to move words.

B. Move words (P.PAS).

C. Set the last file track address for UDAD.

D. Set the LU for core resident present checks.

E. Return.

EXSH

Function: Controls extent search.

Calling Sequence:

```
    Set name as per MOVE1
    JSB EXSH
    Normal return
    A,B,E undefined
    WCS set on entry, may be set on exit
```

If an extent is not found a normal return from D.RTR is taken.

Calls EXSHR

A. Set WCS flag.

B. Search for extent (EXSHR)

C. If not found, exit from D.RTR.

D. Return.

EXSHR

Function: Search for extents.

Calling Sequence:

```
    JSB EXSHR
    return P+1 - not found
    return P+2 - found - the entry's address will have been set
                 up by SETAD.

    on return A,B undefined
              E = 1
```

Calls N.SHR, SETAD

A. This routine jumps into the middle of N.SHR, so set return address for N.SHR to C) below.

B. Search for file (N.SHR). Entered at NSHR0.

C. If not found return P+1.

D. Set up addresses to directory entry (SETAD).

E. If main (i.e. not extent) return to N.SHR and continue search.

F. Return.

FLAG

Function: Counts the number of valid open flags in a directory entry. Clears any invalid entries.

Calling Sequence:

```
Set up directory entry addresses (SETAD)
JSB FLAG
Return
COUN2 = # of VALID flags
```

Calls DORM

A. Clear number of open flags counter and set loop counter to -7.

B. Get open flag.

C. Check if dormant (DORM).

D. If not dormant step count.

E. If done with all 7 open flags in this entry return.

F. Go to B.

FORWD

Function: Forward spaces to next directory entry.

Calling Sequence:

```
DIRA - address of current directory entry  (changes to next
       directory entry address).

JSB DIRA

return P+1 - end of directory
return P+2 - normal return
```

Calls RDNXB, SETAD

A. Add 16 to the address of current entry to position to next entry.

B. If this address is not the start of the next block go to E.

C. Read the next block (RDNXB).

D. Return P+1 if ran out of directory.

E. Set up addresses to directory entry (SETAD).

F. Step to P+2 and return.

LAST?

Function: Test if current entry is the last one in the directory.

Calling Sequence:

```
JSB LAST?
return P+1 - not last entry
return P+2 - last entry
A,B,E undefined
```

Calls NXT/S, FORWD, BACK

A. Compute next file address (NXT/S).

B. If next track, sector are not the same as next track and sector from the directory, return not last file (P+1).

C. Position to next entry in directory (FORWD). If ran out of directory this must be the last file so reset value in SECT and return last file (P+2).

D. If first word of "next" entry is 0 set to return at P+2 (last file).

E. Position back to original entry (BACK).

F. Return.

MOVE1/MOVE2

Function: To save or restore a 9/3 word block on/from a local save area.

Calling Sequence:

```
Set A = address of non-local area
Set E = 0 to save it
      = 1 to restore it
JSB MOVE1
return A,B undefined
         E as set on call
```

Calls P.PAS

MOVE1 moves 9 words

MOVE2 moves 3 words

A. Set for move words.

B. Call P.PAS to move words.

C. Return.

N.SHR

Function: Searches the directory for a file name.

Calling Sequence:

```
Set up name buffer (MOVE1)
Set up to access directory (SETDR)
JSB N.SHR
return P+1 - end of directory return
return P+2 - found
```

On EOD return

```
A = 0 if physical, i.e. no disc space left
A = Buffer address of next entry if there is room.
Directory block will be in core.
B,E undefined
WCS = 0
```

On found return

```
A = Address of first word of entry.
Directory block will be in core.
B,E undefined
WCS = 0
```

Calls RDNXB

NOTE: 1) This routine may be re-entered after a found exit by not changing A and jumping to NSHR4. The search will continue with the next entry and return will be to the original EOD/FOUND exit points.

2) This routine sets up pointers to the first found purged file with the same size as the requested entry for use in reusing file space.

A. Read next block (RDNXB).

B. If end of directory, return.

C. Search for name in block.

D. If found, compute found exit and return.

E. Go to A.

NXT/S

Function: Computes next track and sector address.

Calling Sequence:

```
JSB NXT/S
return P+1 - next track >32767
return P+2 - normal
A = next track
B = next sector
```

Calls .DAD

A. Convert starting sector to double word integer.

B. Get file size. If negative convert to sectors. Make into a double word.

C. Compute next track and sector address by adding (.DAD) current file address and size.

D. Divide by number of sectors per track to take care of track overflow.

E. If overflow on divide, return P+1.

F. Return P+2.

OPNCK

Function: Scan all file directories to make sure there are no open files on the current disc.

Calling Sequence:

```
JSB OPNCK
return P+1 - open flag found
return P+2 - not open flags on disc
A,B,E undefined
```

Calls SETDR, RDNXB, SETAD, FLAG

A. Set up to read file directory (SETDR).

B. Read next block of directory (RDNXB).

C. If end of directory, return no open flags found.

D. Look at each entry in the block. If a purged entry, skip to the next one. If end of directory, return no flags found. Check open flag section in each entry. If one is found, return open flag found.

E. Go to B.

PSTCL

Function: Posts the 2 block cartridge directory list from BUF to LU 2, track $CL1, sector $CL2.

Calling sequence:

```
JSB PSTCL return
```

Calls EXEC

A. Get LU 2 + privileged bits.

B. Call EXEC to write CL to LU 2, track $CL1, sector $CL2.

C. If transmission log isn't 256, return disc error -1.

D. Return.

RDNXB

Function: Read the next logical directory block.

Calling Sequence:

```
Set up to read the directory (one time only) (SETDR)

JSB RDNXB

End of directory return
Normal return

A,B,E undefined
WCS = 0
```

Calls UDAD, RWSUB, DPMM

A. Update addresses (UDAD).

B. If EOD - return.

C. Read the block (RWSUB).

D. If FIRST flag is not set, return OK.

E. Set up parameters (DPMM).

F. Clear FIRST flag - return OK.

RDPAS

Function: Read the passed string or a passed 128 word block from the disc.

Calling Sequence:

```
    JSB RDPAS normal return If disc read error exit is to EX1.
```

Calls EXEC

Note: RDPAS will read the next contiguous block if re-entered at RDPA2. Return is to the original call's return point.

A. Retrieve string with data track address (RDSTR).

B. Set up the LU, track, and sector addresses.

C. Read the block (EXEC).

D. If disc error exit to EX1.

E. Update the track and sector address for possible re-entry at C.

F. Return.

RDPS

Function: Read the disc directory.

Calling Sequence:

```
JSB RDPS
return
A,B undefined
E,WCS = 0
```

Calls WCSR, EXEC

A. Write the current block (WCSR).

B. Read the cartridge directory (EXEC) from LU 2, track $CL1, sector $CL2.

C. If disc error, return error -1.

D. Clear in-core flag and return.

RDSTR

Function: retrieves the string passed to D.RTR in the schedule.

Calling sequence:

```
A = address of buffer
B = length of buffer
JSB RDSTR
return
   A = 0 successful
   A = 1 no string found
```

Calls EXEC

A. Call EXEC to retrieve string (EXEC 14).

B. Return.

RPRM

Function: Sets up the 5 return parameters for D.RTR.

Calling Sequence:

```
Set the directory addresses (SETAD)
Set A to the first word to be returned.
JSB RPRM
return
A,B,E undefined
```

A. Set the first parameter.

B. If call was for create or open, call EXEC to retrieve string containing 6 words of directory entry.

C. Assemble the remaining 4 parameters.

D. Return.

RWSUB

Function: Read the addressed block if it is not in core or write the addressed block.

Calling Sequence:

```
 DRLU = disc lu
TRACK = disc track of requested read
 SECT = disc sector of requested read
#SECT = number of sectors per track if known, else 0
 ABUF = address of buffer BUF
SBFLN = sector length of BUF
SBFLM = largest 2 + (N * 14) <= SBFLN
```

```
      WSC # 0 means last block is dirty and will be posted before
            next read

    JSB RWSUB

    return

    LTRAC = last track read; same as TRACK when exit
    LSECT = last sector read; same as SECT when exit
    LDRLU = last lu read; same as DRLU + PRC when exit
     SBUF = A-reg = address of requested block
     NSEC = negative value of sector beyond buffer -(end of buffer+1)
    C.BFL = current # sectors in buffer
    WORD# = current # words in buffer

   Exits to error -1 if disc error
```

Calls WCSR, EXEC

4 cases of how much will be read

CASE	SEC/TR	SECT	SBFLN	READ LNG	COMMENTS
1	unknown	2-254	2-254	2	used for normal close
		0	2	2	not requiring a search.
2	unknown	0	16-256	16	
3	16-256	0-256	=>SCT/TR	TRACK	
4	16-256	0-254	<SECT/TF	largest SBLFM - (N*14) that won't overflow the track	

A. Write the current block if dirty (WCSR).

B. If requesting same lu and track as least read, and required sector is in the buffer, calculate address of sector in BUF, save current position and return.

C. Calculate number of sectors that can be read. See above 4 cases.

D. Save Starting sector of disc read. Calculate NSEC, C.BFL and WORD#.

E. Read from directory on the disc (EXEC). Return error -1 if disc error.

F. Return.

SCAND

Function: Searches the cartridge directory for the value in TMP2.

Calling sequence:

```
    TMP2  - value we're searching for

    OFFST - offset in entry = 0 for LU
                            = 2 for CRN
                            = 3 for ID

    MASK  - mask if needed  = 177777B for CRN
                            = 377B for LU
                            = 7777B for ID

    DIRAD - starting position in CL

    JSB SCAND

    return P+1 - end of CL and not found
    return P+2 - found

        DIRAD - points to next entry in CL
        B     - points to lst word of "found" entry
        ALU   - LU # from "found" entry
        ATRAK - last track info from "found" entry
```

A. Calculate EXTRA = 4 - OFFST

B. If end of directory, return not found.

C. Save LU # in ALU

D. Position to appropriate word in entry and mask.

E. If it matches TMP2, set DIRAD and ATRAK and return (found).
   Otherwise go to B.

SCBCK

Function: See if LU is mounted in caller's SCB disc's mounted list. Just returns found if caller is not under session control or if ALU is a system disc.

Calling sequence:

```
DISCL - # entries in discs mounted list
MTDSC - address of discs mounted list in SCB copy in BUF
ALU   - LU # to be searched for
PTR1,I- current entry in ID table

JSB SCBCK

return P+1 - found ALU in SCB
return P+2 - not there
```

A. If caller is not under session control or ALU is a system disc, just return P+1.

B. Get next entry in discs mounted list and mask to just the LU and the active bit.

C. If LU is same as ALU and is active then return found (P+2).

D. If end of list, return not found (P+2) else go to B.

SETAD

Function: Sets addresses to point to a directory entry contained in local buffer.

Calling sequence:

```
A - address of the 1st word in the directory entry (see N.SHR)
JSB SETAD
return                  Addresses are set through the
A,B undefined           first flag address
E = 0
```

Calls P.PAS

A. Clear B,E.

B. Call P.PAS to set up 10 addresses in the directory entry.

C. Return

SETDR

Function: Set up to read a file directory.

Calling Sequence:

```
JSB SETDR
Return
A,B,E undefined
WCS = 0
```

Calls WCSR

A. Set FIRST flag.

B. Set the last track and LU. Set SECT to -14 to compensate for the way UDAD handles sector skipping.

C. Return

SETPR

Function: Set up and read the 1st block of a file directory.

Calling Sequence:

```
JSB SETPR
return
```

Calls SETDR, RDNXB

A. Set up to access the disc (SETDR).

B. Read the first block (RDNXB) from the requested disc.

C. Return.

SSTCK

Function: See if LU is in caller's SST. If caller is not under session control, just returns found.

Calling sequence:

```
ALU   - LU # to be searched for
IDSSW - pointer to SST length word in SCB copy in BUF

JSB SSTCK

return P+1 - found it
return P+2 - not there
```

A. If caller is not under session control just return P+1.

B. Get next entry and mask to system LU. Add 1 to it to compensate for SST format.

C. If same as ALU return P+1.

D. If end of list return P+2, else go to B.

TESTL

Function: To insure the disc is locked to the caller.

Calling Sequence:

B - address of the lock word.

JSB TESTL

normal return

If the disc is not locked, TESTL does not return but exits to EX102 causing an error -102.

A. If locked to the current caller, return.

B. Exit to EX102.

UDAD

Function: Update the directory addresses to read the next block.

Calling Sequence:

Set up the directory addresses one time (SETDR).

JSB UDAD

return P+1 - end of directory return
return P+2 - normal return

A = 0 if EOD return

otherwise A,B,E undefined

WCS = 0

Calls WSCR

A. Write the current block (WCSR).

B. Compute the new track/sector address [sector = (sector + sector skip value) mod #sectors/track].

C. If delta track = 0, return.

D. If sector # 0, return.

E. Subtract 1 from track.

F. If EOD take EOD return.

G. Normal return

WSCR

Function: Write the current directory block if WCS = 0.

Calling Sequence:

```
Set WCS # 0 to write
        = 0 for NOP

JSB WCSR

return  A = 1
        E,B undefined
        WCS = 0
```

Calls EXEC

A. Check WCS (write current sector) flag. If not set, return.

B. Clear WCS.

C. Call EXEC to write 128-word block.

D. Exit error -1 if disc error.

E. Return.

Chapter 18

FORMT UTILITY

INTRODUCTION

This section describes the FORMT utility and assumes that you are familiar both with its operation and the description contained in the RTE-IVB Utility Programs Reference Manual. The technical specification should be used in conjunction with the FORMT listings, as it in no way attempts to provide the level of detail given by the code itself and the comments. The technical specification will aid in using the listings by discussing overall structure and flow, some data structures, and complex algorithms/techniques.

The technical discussions that follow also assume that you are familiar with the various "hard discs" supported in RTE (7906(H), 7910H,7920(H), 7925(H)), as well as the 9895 floppy disc. Reference to the technical specifications for the on-line disc support library and for DVx32 should also be helpful.

FUNCTIONAL OVERVIEW

The FORMT program performs the following functions:

a) Format a 9895 floppy disc-all new floppy media must be formatted by writing track and sector addresses. Another function is to mark all defective tracks on a new or previously-formatted, floppy disc as "invisible" so they will not be accessed. No data is preserved.

b) Initialize or re-format a MAC/ICD hard disc subchannel-a process of removing the protected status of all tracks on the lu, of cleaning up the spare track pool for that subchannel, and of sparing any old or new defective tracks on that subchannel. No data is preserved. This process should be performed on a disc subchannel before valid data is stored on it, and should be repeated every time the track map table configuration is changed.

c) Spare an individual defective track on a MAC/ICD hard disc subchannel-the process of recovering and replacing a defective track by the next available spare track on a disc subchannel. The disc controller would then access the good spare on all future references to the physical address of the defective track. This version (on-line but not automatic) of sparing attempts to recover every word of data on the defective track and transfers it for storage on the spare track, where future references will go for the data. Note that NONE of the other tracks on the subchannel will be disturbed, so the spare option would be most useful when a disc subchannel full of valid data developed a media defect, which would cause an RTE track error message.

OPERATING ENVIRONMENT

FORMT runs in RTE-IVB (92068A) revision code 2001 or later, having both the EQT-lock capability and DVR32 or DVA32 that supports FORMT's operations. FORMT is unsegmented; it does not require Table Area II access and therefore may be a Type 2, 3 or 4 program. FORMT requires a buffer equal to the track size of the disc when it accesses hard discs, (4096 words for the 7910, 6144 for the 7905/06/20, and 8192 for the 7925). The floppy disc routines require only a one-sector buffer (128 words), since they are never required to issue a full track transfer. This buffer is retrieved from the free logical memory at the end of FORMT (and its appended subroutines) so the page requirements of FORMT must be overridden at load time. Once a task is chosen, routine CLRBF checks to make sure that the buffer space is large enough, and issues an error message and terminates if it is not. This means that for one schedule of FORMT, it may, for example, have enough buffer space to format a floppy, but not enough space to initialize a 7925 track.

THE DISC UTILITY LIBRARY

In order to accomplish track sparing, initializing, and formatting, FORMT must issue privileged command sequences to the disc controller. Many of these command sequences are unique to the disc maintenance function and are never issued by the driver when servicing standard disc read/write requests by RTE. In order to provide these functions (without resorting to an off-line, stand-alone program), the driver supports a subfunction mode via a special EXEC call. When the driver operates in this special subfunction mode, it outputs a 16-word command protocol buffer to the disc controller in a passive mode (without any error checking). These command protocol sequences are controller dependent, and therefore the disc utility library ($DSCLB), was written to supply the appropriate command protocols to the controller being addressed. The library enables a high-level program like FORMT to call a routine, XSEEK, for example, which will insert the appropriate protocol in the command buffer (\BUFI to \BUFA) and cause the disc to execute a single SEEK operation without doing any extra operations. The program calling the disc library does not need to be concerned whether it is talking to a MAC or ICD type of disc.

THE EQT LOCK FUNCTION

While FORMT is performing privileged sequences of commands to the disc, it must have exclusive access to the controller, so that other disc I/O operations do not move the heads, or change the status of the disc between its calls to the Disc Library. This function is provided by the EQT lock mechanism in RTE-IVB. The lock holds off all other disc requests to the EQT (including RTE loads and swaps), and allows only FORMT to access the EQT.

Since the EQT lock may severely degrade system performance (because loads and swaps will not be permitted), the lock is issued only when necessary. The SP (spare) command locks the EQT for the entire process of moving data from the defective track to the spare track (which is done sector by sector). The IN (initialize) command locks and unlocks the EQT on a track by track basis (which allows a small amount of I/O to occur between track accesses). The FO (format) command for floppy discs may take anywhere from 5 to 20 minutes depending on the "fill" value specified. Therefore, it is not reasonable to allow the EQT to remain locked for this amount of time in RTE-IVB, since this EQT may also support the system swap disc. FORMT solves this problem by calling the LKLUS routine instead of the EQT lock. LKLUS searches the DRT and locks all LU's in the system which reference the floppy disc controller. This safeguard prevents any other disc I/O to the floppy from destroying status information in the controller registers, or moving the heads on the drive.

Memory Lock

FORMT locks itself into memory for the entire duration of its execution. Any program using the EQT lock should follow this policy to avoid a deadlock situation. This could occur if the EQT were locked by a program, and the system then tried to swap the program out to disc. The XSIO request would be held off by the lock, but yet the system would not let the program execute because it would think that the program was no longer in memory. Thus, any program wishing to use the lock should use it in the following manner: 1) lock itself into memory; 2) lock the EQT; 3) perform the necessary disc operations; 4) unlock the EQT; 5) unlock itself from memory.

DISC SPECIFICATIONS

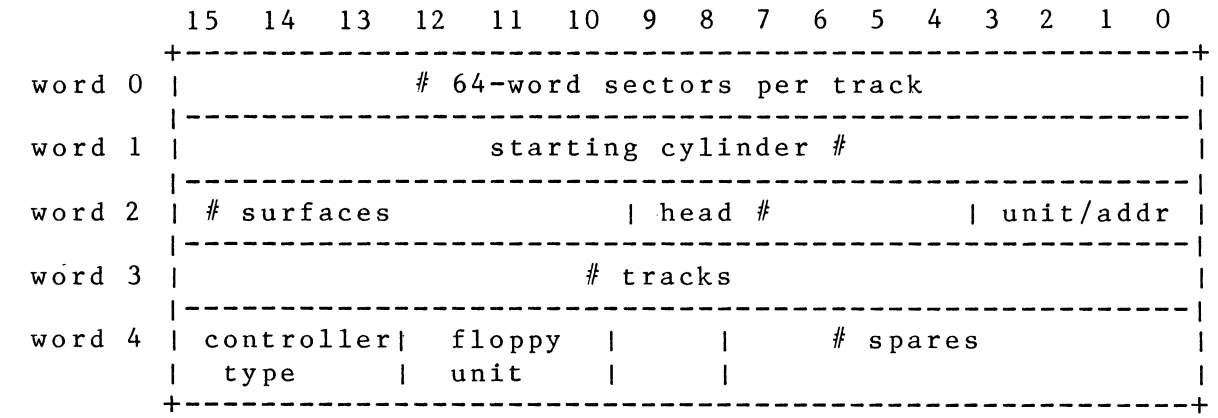
The following disc models are supported by FORMT. A table containing their specifications is included so you can see the differences between models:

MODEL	#128-wd sectors per track	#CYLINDERS*	#SURFACES	CAPACITY	TYPE
7905	48	411	3	15MB	MAC
7906	48	411	4	20MB	MAC/ICD
7910	32	748	2	12MB	ICD Winchester
7920	48	823	5	50MB	MAC/ICD
7925	64	823	9	121MB	MAC/ICD
9895	30	77	2**	1MB per drive	ICD floppy

*Includes tracks which should be allocated to the spare pool and those which are not recommended for use on the floppy. A maximum of 67 tracks per floppy surface is recommended, where the remaining 10 may be used to keep the number of data tracks at the level of 67 per surface, as defective tracks are marked "invisible".

**A single-sided floppy may be used with the 9895 as long as the track map table reflects this subchannel definition.

The above discs are broken down into subchannels for accessing via lu's (FORMT requests lu's, not subchannel numbers). A track map table is built into RTE at generation time, ($TB32 for DVR32 and $TA32 for DVA32). The format of each 5-word entry for a subchannel (maximum of 32 subchannels) is:



The \SETD subroutine gets the 5-word definition into the buffer \TMT by calling the XTTBL subroutine with the disc lu \DLU. \SETD sets up the parameters \D#ST (128 word sectors now), \D#WT, \DFTR, \DUNT, \DSHD, \DNSU, \DNTR, and \DNSP for use by FORMT's pseudo disc driver DISKD. The disc lu had already been verified by the subroutine LU?

OPERATING MODES

FORMT can execute in one of two modes: "BATCH" mode means that all necesssary parameters were entered via the RUN string and that no operator intervention is required. If a parameter in invalid, or an error condition occurs during the performance of a task, then the appropriate error is displayed and a response requested. None of the customary cautions with "OK TO PROCEED?" messages are done if the flag BATCH=7 (meaning all 3 parameters were specified) or BATCH=6 (for IN where only 2 parameters are needed, but set to 7 in IPROC). "Interactive" mode implies that not all of the RUN string parameters for the task were specified. The user must therefore answer all the skipped parameters as well as all the invalid ones, and respond to the "OK TO PROCEED?" queries. FORMT will therefore terminate only with a /E, EX, or EN response to the TASK? query.

At the beginning of each task in FORMT, before branching to the code for the task, all appropriate global-like variables are initialized at ITASK. Then the second RUN parameter or TASK? response is parsed, and the branch to the appropriate task processor is done. All task processors return to DONE? when finished, where a check is made for batch mode (so EXIT) or interactive mode (go back to ITASK and start a new, entirely interactive process). On irrecoverable errors in interactive mode, the task (but not FORMT) is aborted - a transfer to DONE? after the appropriate error message is displayed.

RESPONSE PARSING and PROMPTING

RMPAR is used to return the actual values of parameters from the run string. The sole purpose of the PARSE call is to check the presence and the type (ASCII vs. numeric) of the run-string parameters 2 (task), 3 (lu), and 4 (fill or track#). The actual parameter or any response entered interactively to the TASK? query is parsed and partially processed by the subroutine PTASK, which assumes that the response to be parsed is in RBUFR. When in interactive mode, a TARGT call places the response there. However, in run-string mode, the PARM2 must be explicitly stores in RBUFR (at SET2) and the TARGT call is skipped. PTASK has no return point for the ?? or a non-FO/IN/SP response, so it simply calls EXPLN to display the response options and then assumes interactive mode by starting over again, this time with the TARGT call enabled.

Subroutine PRESP parses all remaining responses, breaking them into the following categories: possibly numeric (no ASCII to numeric conversions by BIDEC are done yet), EN or /E, NO, YE, blank followed by a carriage return, and ??. The first return point is a catch-all for numeric and invalid responses. Run-string parameters 3 and 4 are not parsed by PRESP or converted by BIDEC, since they had already been converted by RMPAR. Note that all callers of PRESP force a JUMP to DONE? if an EN, EX, or /E was entered, returning you to TASK? mode, not exiting FORMT.

All three task processors call LU? to retrieve the disc subchannel lu upon which the task will be performed. LU? also insures that it corresponds to a type 32 disc subchannel (i.e., DVA32 or DVR32) and obtains the TMT definition by calling \SETD. It assures that FO will be done on a floppy disc subchannel because it has 30 sectors per track, whereas IN or SP will be performed only on those subchannels with 32, 48 or 64 sectors per track. This is only a software check, not a hardware check.

All prompts and messages are output by routine PROMT which indexes through a table of message lengths and a table of message addresses based on the index stored in the B-Register on entry. Routine EXPLN displays response options to a class of queries whenever a "??" or erroneous response is entered: EXPLN uses its own tables of lengths and addresses, based on a response class number stored in INDEX.

TASK PROCESSORS

FPROC, IPROC and SPROC set the stage for the actual disc I/O operations which are performed by DISKD and its subroutines. These processors make sure that all the required parameters have been defined and that the track buffers are large enough, call DISKD to perform the operation(s) once or many times, and then do a little cleanup before returning to DONE? The processors set up the modes in which DISKD is to operate. IPROC first defines mode 4 to cleanup the spare track pool by calling subroutine SPINT. It then proceeds to mode 3 for initializing each track on the subchannel. SPROC defines mode 5 for sparing one track on a subchannel, and FPROC defines mode 6 for formatting a floppy. DISKD is state driven and depends on these modes and their sub-phases to direct the pseudo-disc driver. DISKD returns to the task processor only when an operation has been completed (initializing one track, sparing one track, cleaning up a spare track, or formatting a floppy subchannel), or an irrecoverable error has been encountered.

DISKD MODES

The modes and their phases for the various tasks to be performed are indicated below. Note that a phase may exist in more than one mode but its operation is still the same.

MODE 3 - Initialize a hard-disc subchannel.

Phase 1- Read Full Sector to a track to determine if its status bits (S, P, or D) are set/clear.

Phase 2- Write initialize the track without the D or P bit set, and verify.

Phase 3- Read Full Sector to a potential spare to get the S, P, D bits and determine if it is a useable spare track.

Phase 4- Write initialize the available track with the S bit set and its address pointing to the defective track.

Phase 5- Write initialize the defective track with the D bit set and its address pointing to the spare track. Verify the sparing operation done in Phases 4/5 and report it.

MODE 4 - Cleanup the spare track pool for a hard-disc subchannel.

Phase 3- Read Full Sector to a spare track to determine if its status bits (S, P, D) are set/clear.

Phase 4- Write initialize the spare track without the S,P or D bits set and its address pointing to itself (i.e., its now a regular track but in the spare pool). Verify the operation.

Phase 6- Write initialize the spare track with its D bit set so that it is now defective and will be unavailable for use as a spare track. Report the operation.

MODE 5- Spare a track on a hard-disc subchannel, attempting to preserve its data.

Phase 0- Read with offset each sector from the defective track until all sectors are read. Report the defective track, and get address of next available spare on subchannel.

Phase 4- Write initialize the available spare with the S-bit set and data retrieved from Phase 0. Its address points to the defective track.

Phase 5- Write initialize the defective track with the D-bit set and its address pointing to the spare track. Verify the sparing operation done in Phase 4/5 and report it.

Phase 6- A potential spare turned up defective, so do a write initialize to it with the D bit set. Get the next available spare and return to phase 4.

MODE 6- Format a floppy disc subchannel.

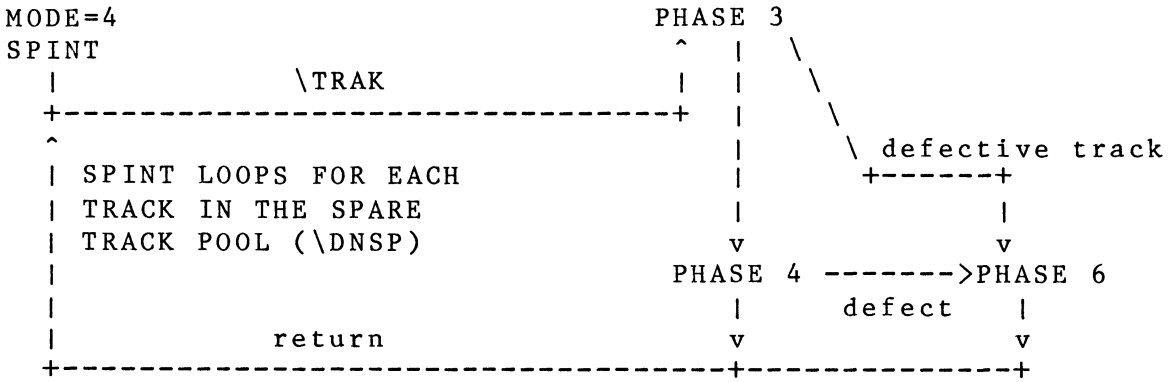
Phase 7- Format the floppy disc with one of 5 bit patterns, the appropriate sector interleave value and format type.

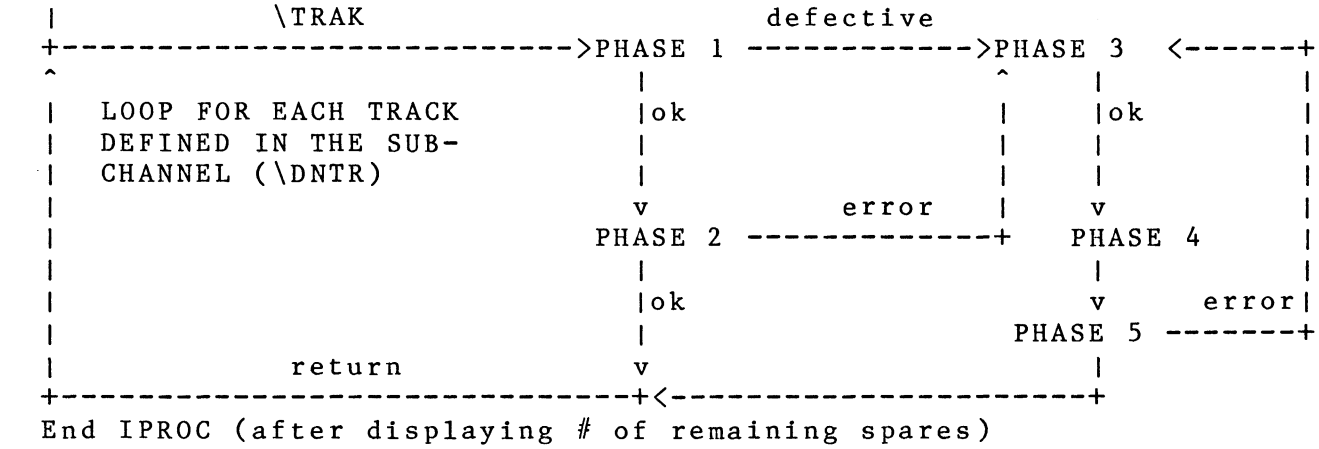
Phase 8- Verify the floppy disc to retrieve any bad tracks and enter them in BDTBL (unless 5th and final pass), and continue verifying starting at the next track.

Phase 9- Write initialize the first sector on each defective track specified in BDTBL with its D-bit set. These "defective" tracks will be made "invisible" on the 5th and final format pass.

MODES 3 AND 4 OPERATION

The flow through a subchannel's initialization is described as follows:



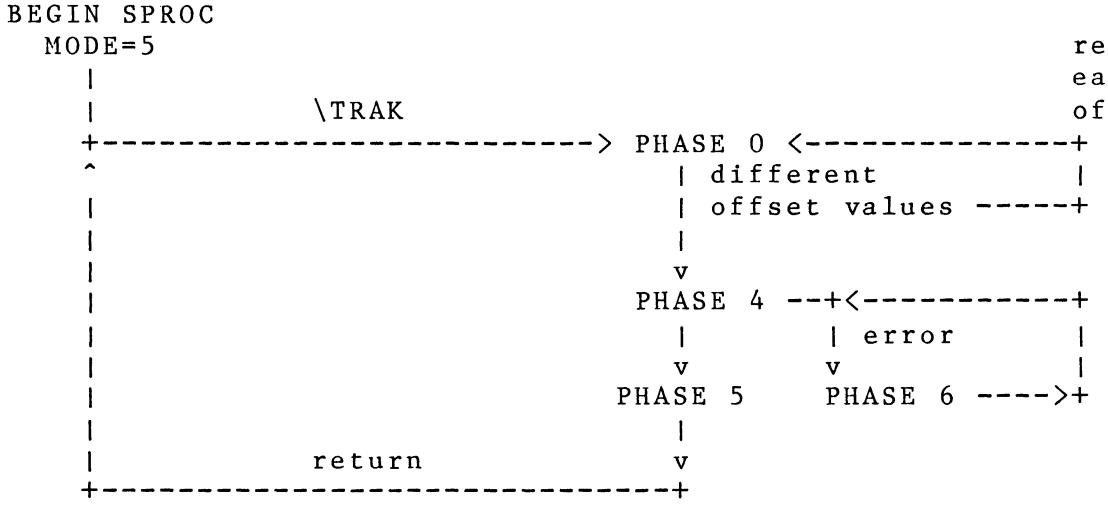


Note that MODE 3 assumes that all bad spares have been detected during MODE 4 so that MODE 3 contains no Phase 6.

NIXSP is the subroutine that is called when in MODE 3 to get the physical address (based on the current subchannel's definition and the value of CRSP#) of the next available spare, either after a defective track or a defective spare is encountered. This is different from the call to XSPAR for MODE 5 (below) since in that case an entire subchannel has already been initialized and a physical scan of the media itself must be done. XSPAR looks for a track in that subchannel's spare pool that does not have its S or D bit set. Obviously, the latter operation in MODE 5 requires that a MODE 4 be done when the subchannel was first initialized.

MODE 5 OPERATION

The flow for sparing one track on a previously initialized subchannel is illustrated as follows:



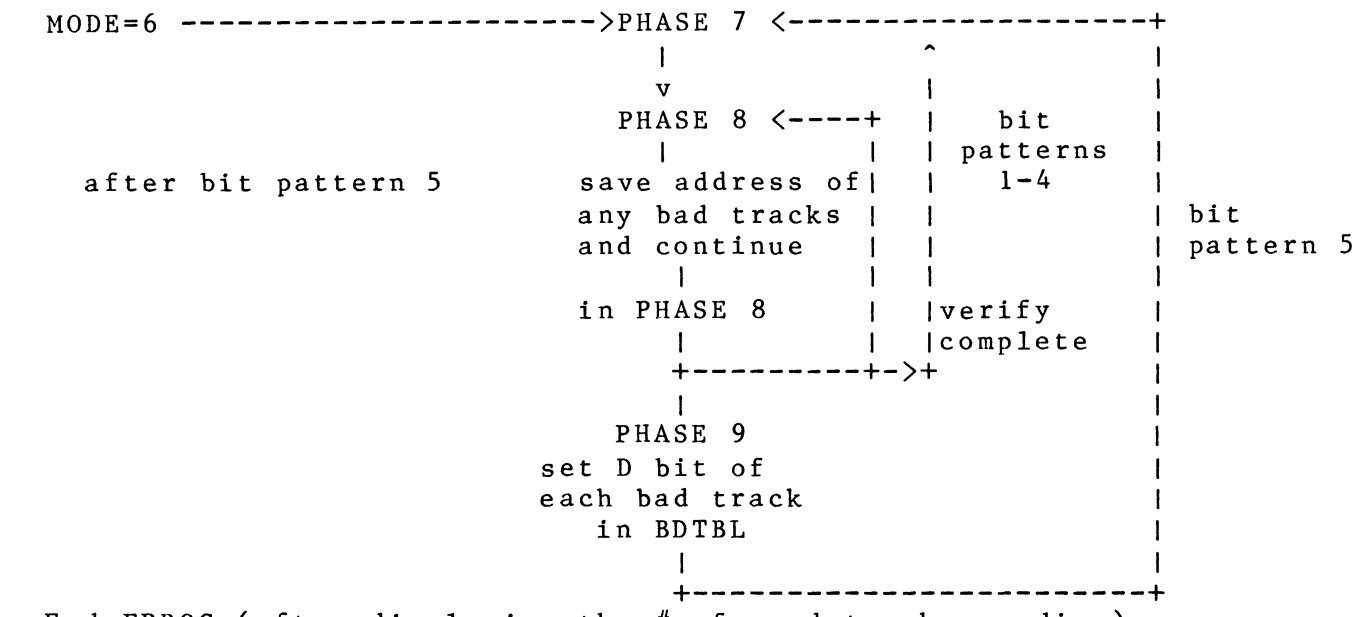
In an attempt to retrieve data from the defective track, READ with OFFSET commands are sent (via XRDOF) for each sector on the track. The subroutine NEWOF sets the offset values (in order) of 0, +10, +20, +40, +50, +60, -10, -20, -30, -40, -50, -60, until a non-erroneous read has been performed or until all offsets listed have been exhausted, in which case warning message MES11 is displayed.

Since the XRDOF call assumes that the first 16 words of the buffer passed to it are for storage of the disc protocol, and that the next 128 words are for the data, the standard buffers decribed by \BUFI cannot be used.

This is because in MODE 5 we are incrementally reading from the defective track, one sector at a time rather than an entire track in one operation. Therefore, the response buffer RBUFR is temporarily used for getting the 128 words of data. Before moving on to the next sector after one is completed, the data is transferred to the appropriate position within \BUFA based on the pointer BPTR. Later then, when the data is to be stored in the spare track during a write initialize, the entire defective track's data will be resident in that buffer.

MODE 6 OPERATION

The flow for formatting an entire floppy media is illustrated as follows:



Before the first FORMT command is actually issued in Phase 7, a status request to the disc controller is done to determine the actual physical type of media in the driver (single-sided vs. double-sided vs. no disc at all regardless of the subchannel definition for \DLU).

Five passes are actually made thru Phases 7 & 8, the first four being identical except for a different bit pattern in the data byte word. The first four passes have an interleave value of 1 (fastest), the override old-format bit set to yes, and type = 2 for HP format. After each Phase 7, Phase 8 is executed to verify all tracks formatted by the current bit pattern in Phase 7. If any errors are found during the verify operation, an XLGAD call is done to retrieve the address of the bad track, and the address is stored in the table BDTBL, unless it was already there because of conflict with some previous bit pattern. The verify operation then continues where it left off, verifying the number of remaining sectors on the disc starting at the next track after the bad one (a seek has to be issued).

After all 4 passes are complete, Phase 9 is initiated to scan BDTBL and doing a write initialize with the D bit set to every track found there. Only the first sector of the defective tracks are initialized. This is adequate to mark the whole track defective and saves buffer space in FORMT.

The final Phase 7/8 pass is done after Phase 9. This time with a bit pattern of 0. The sector interleave value is set to that specified by the user, type again is 2 for HP format. The override old format bit is turned off so we skip those defective tracks marked during Phase 9 and during the last FORMT pass in Phase 7. Now those tracks will be made "invisible". It is assumed that no errors will occur during the final verify in Phase 8. If errors occur, the user is warned that they probably have a bad floppy and that they best dispose of it.

SPECIAL TIMEOUT PROCESSING

Formatting a floppy requires 5 to 20 minutes in RTE-IVB, depending on the number of surfaces (one or two) and the user specified "fill" (sector interleave-1) value. Some disc library calls to the floppy take a very long time to execute. For example, a double-sided verify operation with a fill value of 28 would take approximately 13 minutes. This is true because XVRFY is called to verify all 154 tracks with one I/O request to the disc controller. A fill value of 28 implies that the disc must make 30 revolutions to verify one track.

154 tracks x 30 revolutions/track x 1 second/6 revolutions=770 seconds

Obviously, the disc driver DVA32 will timeout unless an unusually large timeout value is set. This alternative was judged unacceptable since hard discs (high performance) may be sharing the same driver. This would imply that the system may have to wait 13 minutes before it could swap a program! In order to minimize the 9895's apparent incompatibility with the "real-time" concepts, the time-out value is set to two seconds. Long disc library operations are then expected to time-out by FORMT. The library returns the IER parameter which takes on the following values:

IER = 0 if operation successful (and completed)
IER = 1 if disc error (and operation completed)
IER = 2 if disc had a power fail
IER = 4 if operation timed out (not completed)

FORMT checks the IER parameter to determine whether an operation timed out. There are 3 operations where a time-out is anticipated in FORMT: 1) when issuing the first status to a floppy after power on (at REQ6); this may take up to 10 seconds for the hardware to determine the type of media the user inserted; 2) the XFRMT call to format a floppy disc is expected to take about 45 seconds for a single-sided, and 80 seconds for a double-sided floppy; 3) the XVRFY call may take up to 13 minutes to verify every sector on a double-sided floppy. Note that the verify time is unpredictable because a media defect may be encountered at any random time during the 13 minutes, and in this case, the controller interrupts the driver and returns with the defective track status.

There is a problem with allowing timeouts to be expected. How does FORMT distinguish an expected time-out from a time-out because the disc is not ready or is non-existent? The SLEEP routine takes care of these unusually long waits, and detects the "disc not ready" condition. In the 3 instances above, FORMT calls SLEEP to suspend FORMT, putting it in the system time list, until the operation is expected to be complete. For maximum efficiency, SLEEP suspends FORMT for an initial length of time (equal to the best case execution time), and then polls the disc library with an XDSJ call every 5 seconds to see if the operation is complete. The caller of SLEEP is responsible for setting the maximum tolerated execution time (in terms f the number of 5 second polls). If this time is exceeded, SLEEP takes the "disc not ready" return.

SPECIAL HANDLING OF THE 7910H DISC

The 7910H makes use of a slightly different disc controller than the 7906H/20H/25H discs. The primary difference is that the 7910H performs address verification on SEEK commands (instead of waiting until the data transfer operation READ or WRITE). The result of this is that a status 7 or 11 may occur unexpectedly after a SEEK operation. The SK10H flag indicates that the current operation is a SEEK, so that the status testing routine CKST1 will ignore Status 1 errors on 7910 seek. If a status 2 error occurs on a 7910H seek, CKST1 forces the status 1 word (= 23) to indicate this.

The 7910H does not support the FILEMASK command or the RECALIBRATE command. FORMT skips the FILEMASK command, and the RECALIBRATE is replaced with a seek to Cylinder 0 head 0. A standard read operation is substituted for a READ WITH OFFSET on the 7910H, since this command is not supported.

Appendix A

DVP43 POWER-FAIL AUTO-RESTART DRIVER

The power-fail auto-restart driver must reside in the System Driver Area because it is entered directly in the System Map from the trap cell when power-fail or power-up occurs. The entry point for the interrupt is $POWR.

On power failure, DVP43 stops DMA transfers on both ports, saves all the programmable registers and S-registers, and saves all of the map registers. DVP43 also saves the location of the last memory protection violation if the system was in the process of registers. With all this done, the driver does a JMP* to wait for power to fade out completely.

When power returns, $POWR is entered and control is transferred to the "UP" section of code. A switch is set so that if another power failure occurs, while DVP43 is in the process of restoring the system, none of the "DOWN" code is executed. This preserves the saved information from the first power failure in case it could not be restored before a subsequent failure occurred.

In the "UP" code, all of the map registers are restored and the base page fence is set up again. Then a search is made for DVP43's EQT entry. If it is not found, a halt will be simulated. The S-register will contain 103004 octal while a JMP* is executed. Once the power-fail driver's EQT is found, the driver's entry in the Driver Mapping Table is modified to indicate that the driver does its own mapping. The time-out handling bit is set in the power-fail driver's EQT and the EQT is set up to time-out on the next clock tick.

The EQT count is set up for scanning all of the EQT's later. If this is another restart attempt (after being interrupted from the "UP" process by another power failure), the current EQT which was being processed for restart is set busy so another restart try will be done during the scan of all EQT's.

The current time-of-day is saved, if it was already saved on a previous restart attempt. This preserves the time of the first power failure if there happens to be a number of successive failures. Then the clock is restarted by a call to $SCLK.

The privileged I/O terminator card is set up, if there is one, before the registers are restored and return is made to the point of interrupt. The switch is reset to allow another power-fail to be processed.

Each time-out entry into DVP43 causes an EQT to be checked for an I/O request in progress. If an EQT is busy and it has the power-fail handling bit set, the driver map is set up by a call to $DRVM, and the driver is entered at the initiator entry point. If the driver is busy but does not do its own power-fail recovery, the driver is set "down" and $UPIO is called to restart the last I/O request.

When all the EQT's have been checked, AUTOR is scheduled. AUTOR is aborted before it is scheduled because it may still be scheduled from a previous power failure. Finally, the time-out counter is cleared in DVP43's EQT and control is returned to the system via $XEQ.

The first word, TDB, will be used by the system as follows:

```
 0 - subroutine is available
=0 - points at 4 word block describing current "owner";
     i.e. the program currently executing in the subroutine.
```

When the TDB is moved to system memory, the first word is changed to point to the location the TDB must be moved back to.

The sign bit of the third word of the block indicates if the block was moved or not. The sign bit of the ID-address indicates if the fourth word block is four words (0) or five words (1) long. (This is caused by a one word imprecision in memory allocation.)

The ID Extension List is a two dimensional one way linked list. The HEAD of the list points to all programs processing reentrant subroutines. They are added to the head of the list as each JSB $LIBR is processed. The other dimension is a list of all reentrant subroutines being processed by one program; that is, on reentrant subroutine calling another. The general reentrant list structure is illustrated in the example given in Figure B-1. Expansions of the structure are given in Figure B-2 through B-5 respectively. Figure B-6 provides a detailed illustration of the total process.

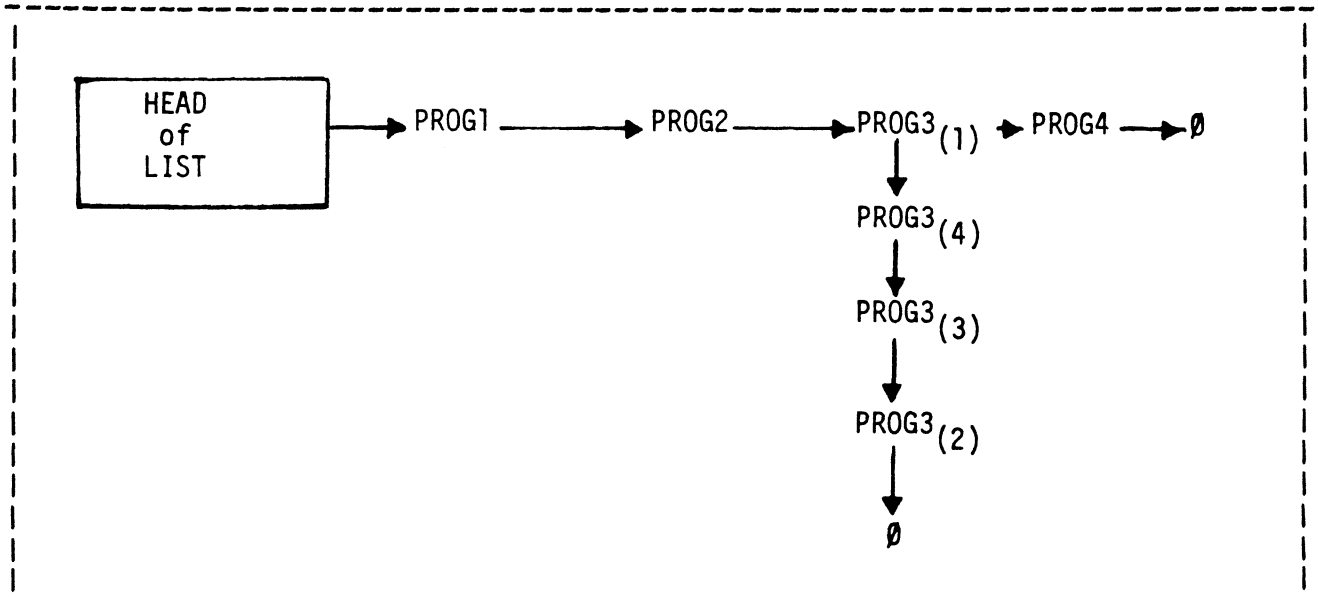


Figure B-1. General Reentrant List Structures

In Figure B-1, subscripts of PROG3 refer to the order in which the reentrant subroutines were called. PROG4 was the first to enter a reentrant routine; PROG1 was the last.

In Figure B-2 one four-word block is created each time a reentrant routine is entered. Programs A and B are both in reentrant subroutines. A entered its routine first.

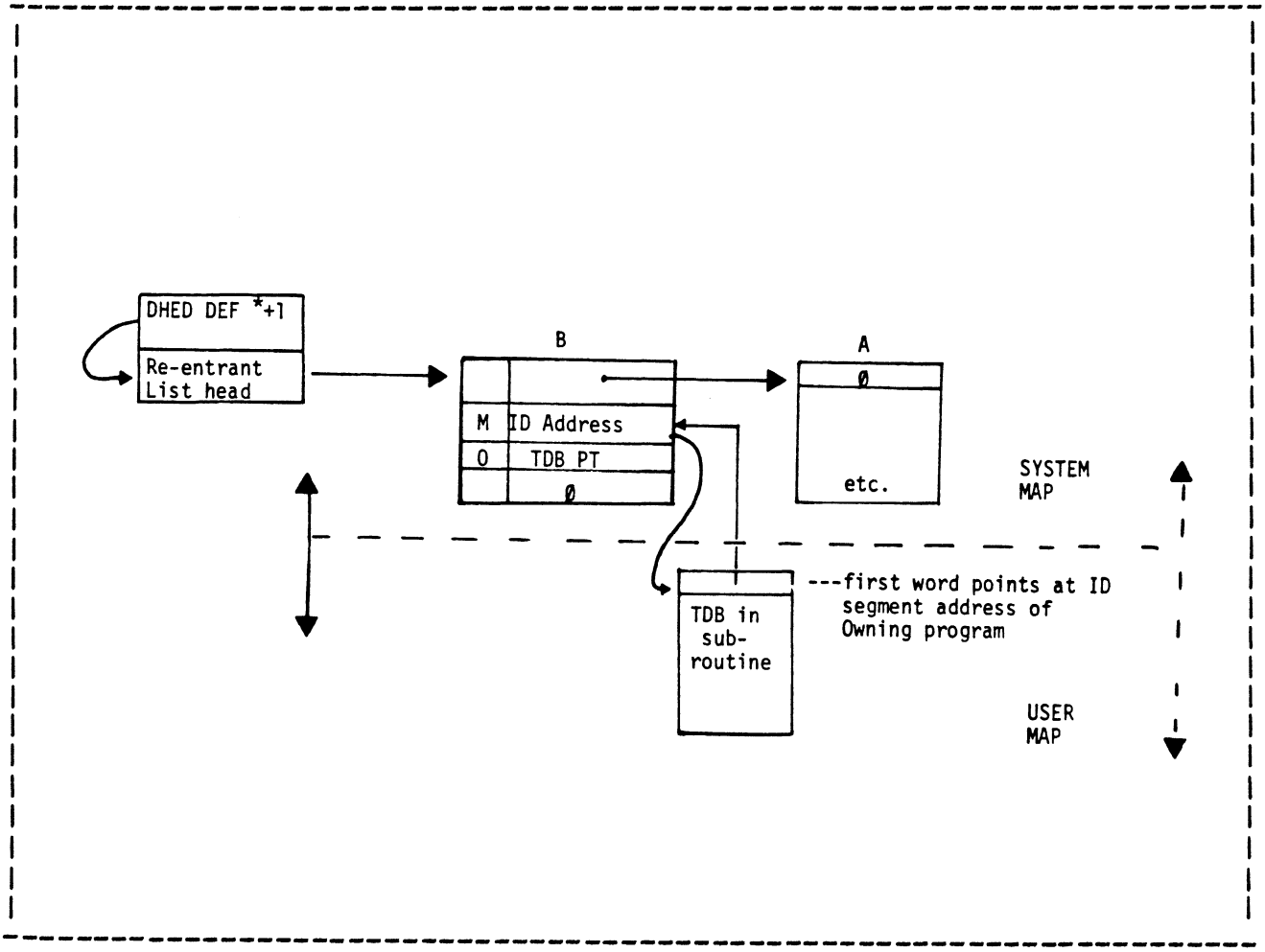


Figure B-2. Reentant List Structure (Expanded)

In Figure B-3, program "B" is suspended -- program "C" reenters "B's" subroutine.

NOTE: The moved status is indicated by "B's" TDB pointer not pointing in turn to B's ID segment address.

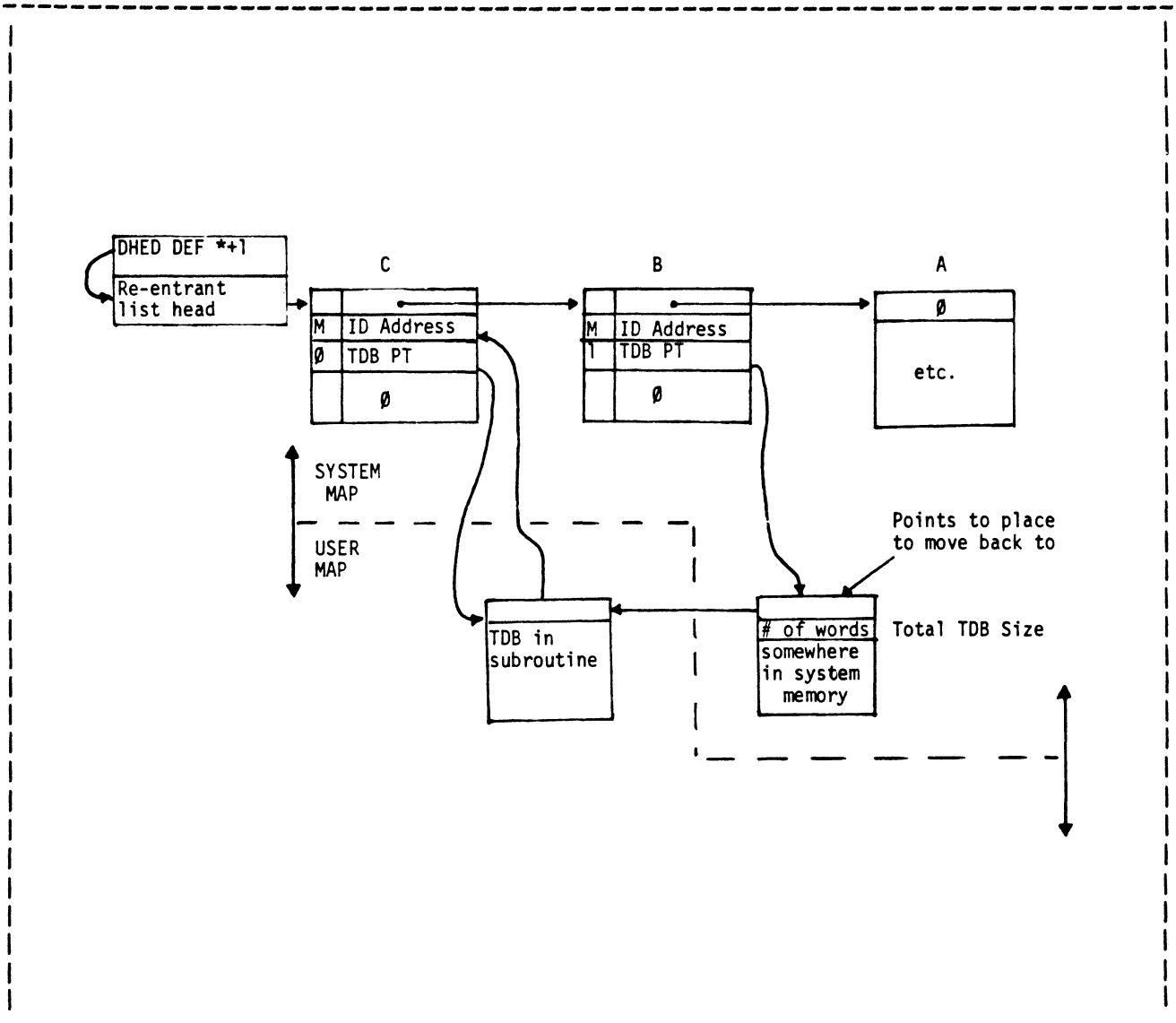


Figure B-3. Reentrant List Structure (Expanded)

In Figure B-4, program "C" exits the routine.

The routine is available - B's memory will be moved back when the dispatcher is committed to run it.

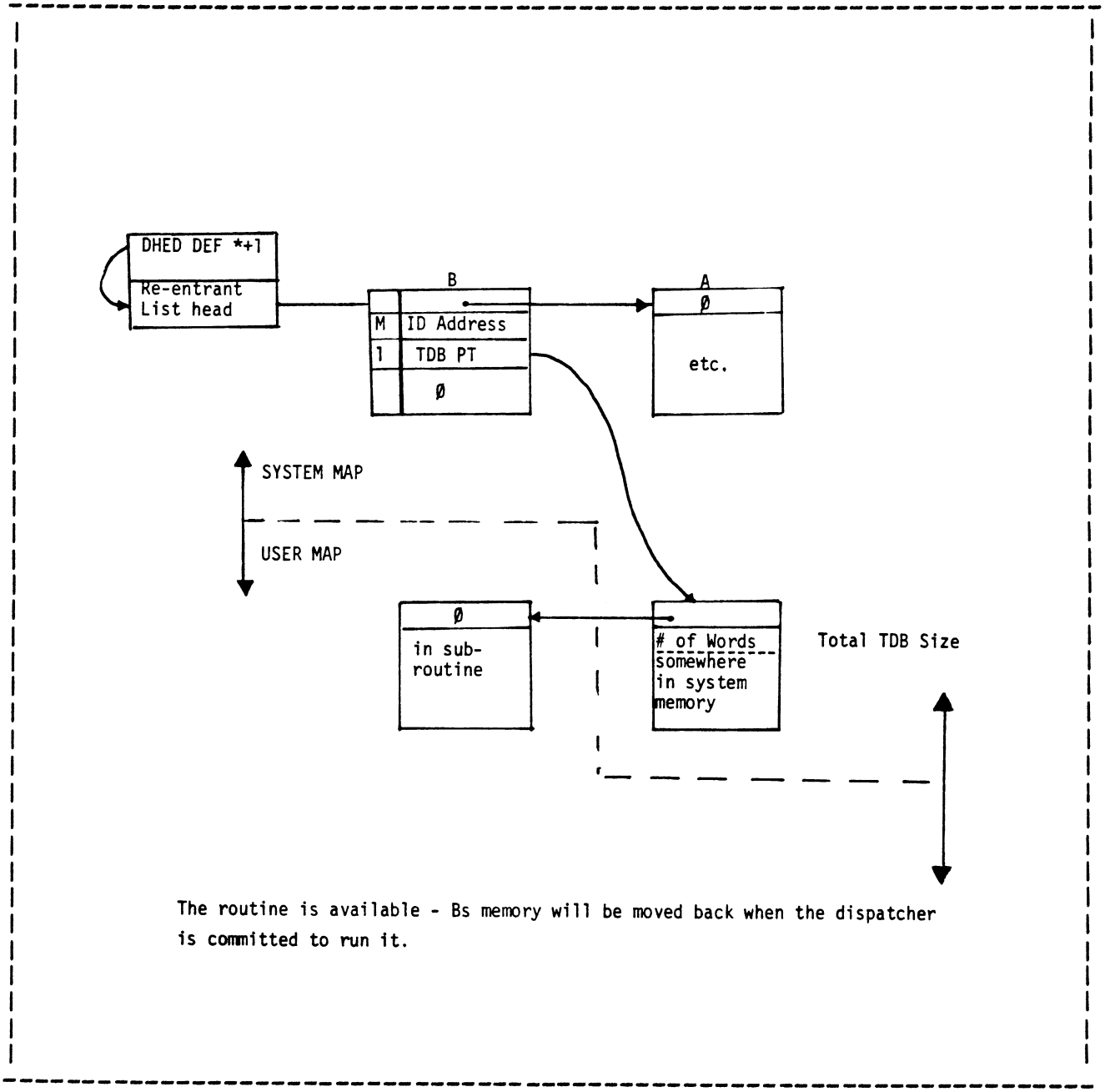


Figure B-4. Reentrant List Structure (Expanded)

Assume that in Figure B-3, program "B" was to be executed prior to "C's" exit from the reentrant routine. Then "C's" memory must be saved and "B's" moved back in as illustrated in Figure B-5.

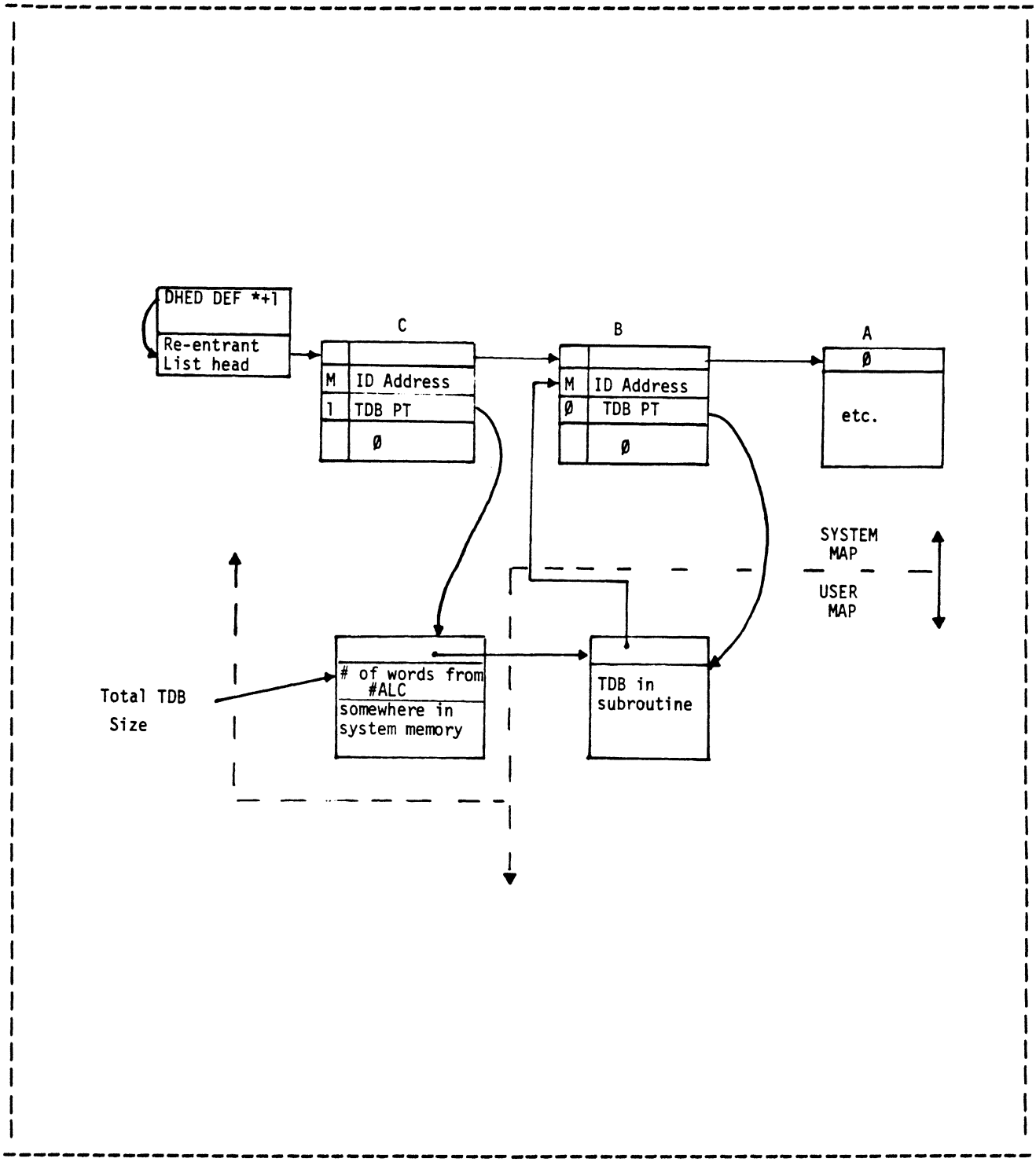


Figure B-5. Reentrant List Structure (Expanded)

Suppose starting at Figure B-3, routine "C" now calls another reentrant subroutine -- the list structure is now as illustrated in Figure B-6.

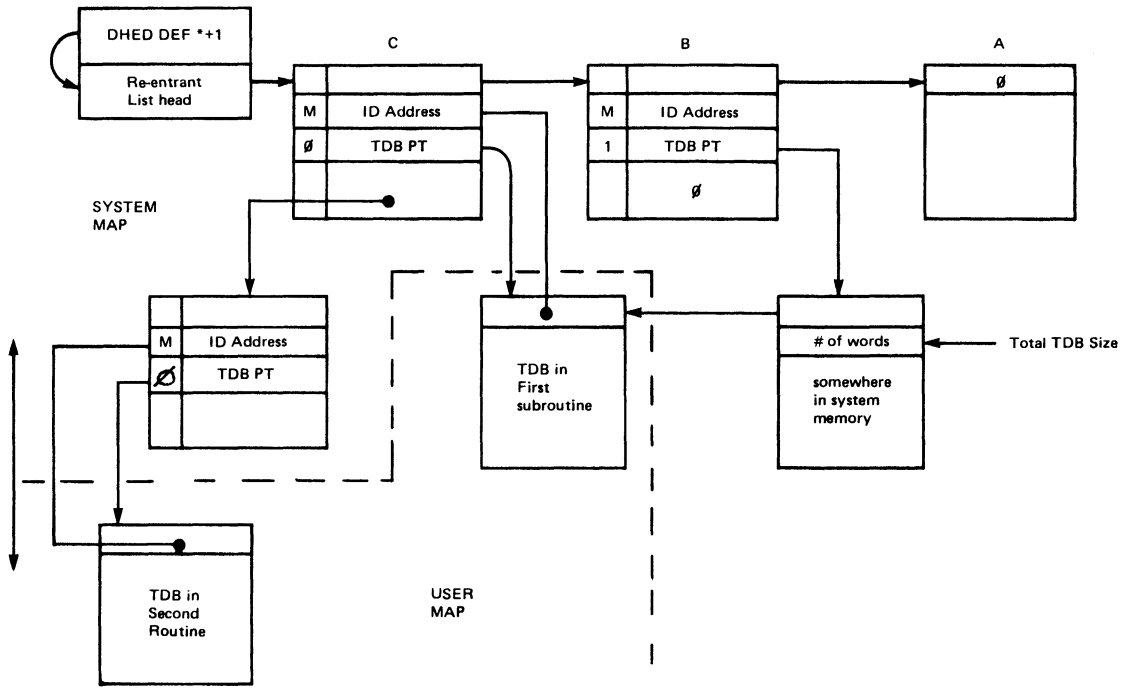


Figure B-6. Reentrant List Structure (Expanded)

Figure B-7 shows programs PROGD, PROGC, PROGB, and PROGA, all of which are executing reentrant subroutines. PROGC is the program currently running.

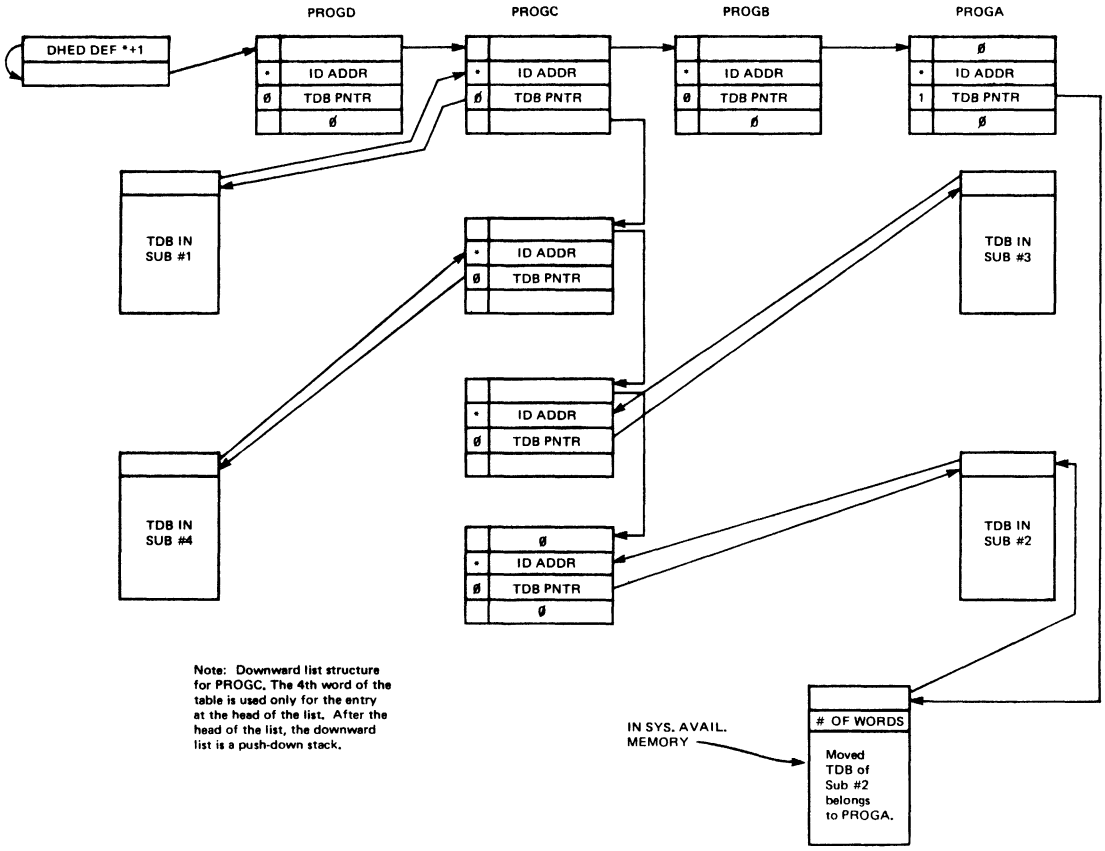


Figure B-7. Detailed Reentrant List Structure

Appendix C

.ZPRV/.ZRNT CALLING SEQUENCES

The externals .ZPRV and .ZRNT are treated as "special" entry points in the RTE-IVB Operating System. The RTE Generator modifies the code that is loaded for subroutines that reference these externals. The changes made depend on whether or not the code is loaded into the core resident library (and hence may be shareable) or if the code is loaded with the program (not shareable), in the latter case the externals are satisfied by replacing the calls to .ZPRV or .ZRNT with an RSS (i.e., .ZPRV,RP,2001). These RP's are passed to the on-line loader in the same manner as an operator's RP command at RTGEN time, thus, the on-line loader can perform the same functions as the RTE-Generator with respect to the externals .ZPRV, .ZRNT, $LIBR, and $LIBX. The following examples should help to illustrate how an assembled subroutine is modified.

AS ASSEMBLED	WHEN "SUB" IN CORE RESIDENT LIBRARY	WHEN "SUB" NOT IN CORE RESIDENT LIBRARY
NORMAL PRIVILEGED ROUTINE		
SUB NOP	SUB NOP	SUB NOP
JSB .ZPRV	JSB $LIBR	RSS
DEF LIBX	NOP	DEF LIBX
...	...	...
...	...	...
LIBX JMP SUB,I	LIBX JSB $LIBX	LIBX JMP SUB,I
DEF SUB	DEF SUB	DEF SUB

P R I V I L E G E D W I T H " . E N T R "

```
PARM1 NOP                PARM1 NOP              PARM1 NOP
PARM2 NOP                PARM2 NOP              PARM2 NOP
SUB   NOP                SUB   NOP              SUB   NOP
      JSB .ZPRV                JSB $LIBR              RSS
      DEF LIBX                 NOP                    DEF LIBX
      JSB .ENTP                JSB .ENTP              JSB .ENTP
      DEF PARM1                DEF PRAM1              DEF PRAM1
      ...                      ...                    ...
      ...                      ...                    ...
LIBX  JMP SUB,I          LIBX  JSB $LIBX      LIBX JMP SUB,I
      DEF SUB                  DEF SUB                DEF SUB
```

N O R M A L R E - E N T R A N T R O U T I N E

```
SUB   NOP                SUB   NOP              SUB   NOP
      JSB .ZRNT                JSB $LIBR              RSS
      DEF LIBX                 DEF TDB                DEF LIBX
      ...                      ...                    ...
       ...                      ...                    ...
      ISZ SUB                  ISZ SUB                ISZ SUB
      ISZ TDB+2                ISZ TDB+2              ISZ TDB+2
      NOP                      NOP                    NOP
      ...                      ...                    ...
LIBX  JMP SUB,I          LIBX  JSB $LIBX      LIBX  JMP SUB,I
      DEF TDB                  DEF TDB                DEF TDB
      DEC 0                    DEC 0                  DEC 0
```

R E - E N T R A N T W I T H " . E N T R "

```
PRAM1 NOP                PRAM1 NOP              PRAM1 NOP
PRAM2 NOP                PRAM2 NOP              PRAM2 NOP
SUB   NOP                SUB   NOP              SUB   NOP
      JSB .ZRNT                JSB $LIBR              RSS
      DEF LIBX                 DEF TDB                DEF LIBX
      JSB .ENTP                JSB .ENTP              JSB .ENTP
      DEF PRAM1                DEF PRAM1              DEF PRAM1
      STA TDB+2                STA TDB+2              STA TDB+2
      ...                      ...                    ...
      ...                      ...                    ...
LIBX  JMP TDB+2,I        LIBX  JSB $LIBX      LIBX  JMP TDB+2,I
      DEF TDB                  DEF TDB                DEF TDB
      DEC 0                    DEC 0                  DEC 0
```

Appendix D

RTE-IVB ID SEGMENT TABLE

The RTE-IVB ID segment for programs is 33 words long (see Figures 1-1 and 1-1A). In addition, all EMA type programs have a three word ID extension. Program segments have a 9 word ID segment. The format for the ID segment and ID extension is shown on the next page. A description of the various words, fields and bits follows.

Word 0 is the linkage word for the program. Whenever the program is put into a state (scheduled operator suspend, etc.) the program is put into a linked list threaded through word 0. This word is also used to queue the program up on EQT's for I/O processing.

Words 1-5, called XTEMP, are used dynamically in the ID segment for operating system information regarding the program. Initially at program schedule, the scheduler places the schedule parameters into this 5 word area. For example, a RU,PROGX,1,2,3 would cause words 1-5 in the ID segment to contain 1,2,3,0 and 0 respectively. The scheduler also takes the address of Word 1 and places this into word 10 of the ID segment, the B-Register at suspension word. When the program starts executing the system library subroutine RMPAR can be called; it uses word 10 to pick up the run parameters. The words are also used for unbuffered I/O.

Word 1 of the ID segment is also used to specify why a program is in the general wait state. A program can get into the general wait state in eight ways. The reason for being in a state is specified in ID segment word 1 by the following rules:

REASON	CONTENTS OF ID WORD 1
Waiting for Resource # allocation	Address of $RNTB
LU# locked	Bits 6-10 of DRT for LU reference = RN#
Resource # locked	Address of referenced RN #
Waiting for class # allocation	Address of $CLAS
Waiting for Class Get Competition	Address of $CLAS entry referenced
Device (LU or EQT) down	4
Waiting for Son to complete	Son's ID address
Buffer Limited	EQT address

Word 1 is also used by the $ALC routine anytime a program needs more system available memory than is currently available, assuming enough memory can ever be available. In this case $ALC places the number of words requested in Word 1 of the requesting programs ID segment. Every time memory is returned through $RTN, Word 1 of the highest priority memory suspended program is checked to see if the memory suspended program can be rescheduled. No lower priority memory suspended programs are suspended until the highest priority memory suspended program is rescheduled.

Word 6 is the priority word. This has the priority of the program. Priorities range from 1 to 32767 for user programs. Occasionally, systems programs give themselves a priority of 0. FMGR does this at Boot up. This allows the program to run at the highest possible priority.

Word 7 is the primary entry point of the program or program segment. It is the relocated address of the first instruction in the program to be executed.

Word 8 is the point of suspension. Everytime a program is suspended or interrupted Word 8 is the address within the program to start the continuation of that program when it is rescheduled. Whenever a program terminates this word is set to zero. However, if the program terminates saving resources (NOT SERIALLY REUSABLE) word 8 is not reset because to terminate saving resources is to save the point of suspension. Then whenever the program is rescheduled, execution will begin at the address specified in word 8.

This word does have one other use which is not generally known. The word can also be used as a debug tool for the systems level programmer. Since the word always defines the point of suspension, it always defines the area of a program which is in an infinite loop. This is especially useful for the assembly language programmer because the infinite loop location can be quickly pinpointed.

Words 9,10,11 contain the A,B and E/O registers at suspension. Words 12,13 and the upper byte of word 14 contain the 5 ASCII characters of the program name.

The lower byte of word 14 contains the TM,CL,AM and SS bits plus the type field. Word 15 contains the NA,NP,W,A,O,R and D bits plus the status field. These bits and fields are used as follows:

TM This bit is set if the program is temporary. That is, there is no permanent copy of the ID segment in the system area of the disc. If the bit is clear the program is a permanent one.

CL Memory lock (core lock) bit. This bit is set by the EXEC 22 request if the user wishes to lock the program into memory and thus prevent swapping.

SS Short segment bit. If set, then the ID segment is a 9 word ID segment used for segments in a segmented program. This is set up by the generator and never changed.

Type FIELD This field of word 15 specifies the program type Memory Resident = 1, Real Time = 2, Background = 3, Large Background = 4, Segment = 5 (refer to summary of types in user manual).

NA No abort bit. This bit is set if the sign bit of the current EXEC call is set. It informs the system that certain errors are in this request to be handled by the program itself and should not cause the program to be aborted. (MP,RQ,RE,PE,DP, and DM errors will abort the program regardless). Note that setting the sign bit of the EXEC request also increments the normal return address of the EXEC request by one.

NP No parameters allowed on reschedule. This bit is set if no parameters should be passed to the program on reschedule. This bit is set if the program is operator suspended or if a father suspends a son.

W The wait (W) bit is set whenever a program (father) has scheduled another program (son) with wait (EXEC 9 or 23). The son's ID address will be found in word 1 of the father's ID segment.

A This is the abort bit. This bit is set when a program is to be aborted. If the A bit is ever set on a LIST processor entry, no matter what the request, the program is immediately put dormant. The A bit is set by the system on detection of certain errors.

O The operator suspend (O) bit. This bit is set when an operator suspension is attempted at a time when it is not feasible to do it directly. The bit indicates that the system should do it at some later time. This is what is meant by deferred action. The system tried to do something, found out, for one reason or another, that it wasn't feasible so it wrote a note to itself (set a bit) to remind it to do the requested action as soon as it is feasible. Uncompleted DISC I/O would be one reason for deferred action.

R R bit, for the most part, is also a deferred action bit in that it indicates how a program is to be set dormant when it is set dormant. (This bit has nothing to do with a serially reusable program termination.) When the program is set dormant the bit is cleared. Word 8 = 0 is the flag by which the system knows that the program terminated saving resources.

L Load "IN-PROGRESS" bit, is set while program is being brought into memory from the disc. This inhibits the dispatch of the program until the load is complete. This permits a program to stay in the scheduled list and maintains its timeslice position with programs of the same level.

D The dormant (D) bit is a deferred action bit which is set if a program cannot be set dormant on request. It indicates that the program is to be set dormant as soon as feasible.

Status
FIELD This is the current state of the program. States are 0,1,2,3,4,5,6 - dormant, scheduled, I/O suspend, general wait, memory suspend, disc suspend, and I/O suspend respectively.

Words 16,17,18 and 19 contain time scheduling information about the program. The four words are used in the operator command *ST,PROGX to give time information about the program.

Word 16 is the time list linkage word. All programs in the time list will be linked together through this word.

Resolution (bits 15-13) in word 17 contains the resolution code. Multiplier (bits 11-0) contain the multiplier for the resolution. The T bit is set if the program is in the time list. Words 18 and 19 contain the system time in 10's of milliseconds of when the program is to execute next. The two words give a 10 millisecond resolution for a 24 hour period. Word 19 contains the high order bits of the time.

Word 20 of the ID segment contains the BA,FW,MTM,AT,RM,RE,PW and RN bits plus the father ID segment number field. They are used as follows:

BA Batch bit. This bit is set if the program is running under batch Program JOB or the FMGR :JO command set this bit. The batch bit is propagated from father to son. That is, if the father is under batch and schedules a son the son's BA bit will be set.

FW Father waiting bit. If the father is scheduled with wait (EXEC 9 or 25), the son's FW bit is set. If the father is scheduled W/O wait,the bit is clear.

MTM Multi-Terminal Monitor Bit. This bit is set if the program is operating under the multi-terminal monitor mode. Like the BA bit, this bit is propagated from father to son.

AT Attention bit also called the break bit. This bit is set by the BR operator command and cleared by the IFBRK system library routine and program termination.

RM Reentrant memory moved bit. This bit is set if the program has information (Temporary Data Block) in system available memory that must be moved into the program area before the program can continue to execute.

RE Reentrant routine in control now. This bit is set anytime a reentrant subroutine of this program is executing.

PW Program wait bit. This bit is set when some program wishes to schedule this program with wait (EXEC 9 or 25) but this program is currently active. The perspective father will be in the general wait state with the prospective son's ID address in Word 1 of the prospective fathers ID.

RN This bit is set when a resource number is either owned or locked by this program.

Father FIELD The field is used if this program is a son. The field will have the ordinal number of the father's ID segment. The number will be there regardless of the type of schedule i.e., EXEC 9,10,23 or 24. The least significant bit is also set if the program terminates serially reusable. This bit is a flag for the dispatcher to avoid certain program clean up procedures. The bit is cleared later.

Word 21 contains the RP bit, the # of pages field, memory protect fence index field, and the partition number field.

RP Reserved partition bit. This bit is set if the program is assigned to a partition. The partition number will be in the partition number field. The numbers start counting from 0.

# of PAGES FIELD This field contains the number of pages the program takes up not counting base page. For segmented programs it is the # of pages of the main, subroutines and largest segment. For EMA programs the size includes main, subroutines, largest segment and the MSEG size.

For non EMA Programs

$$1 < \#\ \text{of pages} < \text{MAXIMUM LOGICAL ADDRESS SPACE (in pages)}$$

For EMA PROGRAMS

$$2 < \#\ \text{of PAGES} < \text{MAXIMUM LOGICAL ADDRESS SPACE} + \text{MSEG SIZE}$$

MPFI FIELD This is the memory protect fence index field. This field contains an index (0-5) which when added to the start of the memory protect fence table gives a location containing the proper memory protect address for this program. This is set by the LOADR or generator and does not change.

PART'N
#
FIELD This field contains the partition number that the program last executed in. [Counts from 0.]

Words 22 and 23 contain the high main +1 and low main address respectively of this program. Word 23 address does not include the high main +1 of program segments.

Words 24 and 25 contain the low and high base page address of the program. High +1 does not include link address for any program segments.

Word 26 contains the disc address of the virgin copy of the program on the disc. The program may only reside on LU2 or LU3. If on LU2 then bit 15 is clear, if on LU3 bit 15 is set. Bits 14-7 contain the track # (0-255) and bits 0-6 contain the sector number. Word 27 is formatted as word 26 but is the swap address in the track pool for the program.

Word 28 is used only for EMA programs and is zero for non EMA programs. Bits 15-10 contain the original number of the 3 word ID extension associated with this program. Bits 9-0 contain the EMA size of this program. The value here will be 1 if a default EMA size is taken and the program has not yet run. Else the value will be a minimum of 2 to the maximum size of the largest partition minus the program size.

Word 29 is used only for segmented programs and is the High Main +1 address of the program, subroutines, and largest segment. If the program is not segmented it is 0.

Word 30 is the timeslice word. This word defines the timeslice of the program and has the following states:

=1: The program has just been placed into the scheduled list and has not been dispatched (or redispatched) or the program is not being timesliced.

=0: The program is not scheduled or has used a full timeslice.

L0: The program is currently running under timeslice control or was bumped from execution by a higher priority program. This word represents the remaining timeslice for this program (in 10's of milliseconds).

Word 31. The Termination Sequence counter is incremented each time a program completes or aborts (except Save Resources termination). This value is used by the FMP in the definition of a valid open file. The counter is the property of the ID segment, not the program currently resident in the ID. Therefore, these bits are not altered when LOADR or FMGR builds an ID for a new program.

The "DC" bit is used to indicate that the program may not be duplicated. Since permanent Programs are now copied, (D.RTR,SMP,GASP). This is needed as certain programs must not be copied. The generator sets the DC bit whenever the primary type code is expanded by adding 128 to it. The LOADR also provides an op-code ("NC") to provide for the setting of the "DON'T COPY" bit.

The "CP" bit is used to indicate that this program is a copy of another program. It is set by the FMP routine IDDUP and is used in the killing of ID segments.

LOADR checks the "DC" and "CP" bits during purge operations. If the "CP", bit is set, the ID is just killed. If the "DC" bit is set, the standard purge operation is performed. Otherwise, (i.e., the program is not a copy, but can be copied) LOADR checks to see that no copies exist of the program. If copies exist, LOADR rejects the purge and an error is issued.

The owner ID is used by program LGOFF to give back ID segments at session termination. It identifies the creator of this ID segment. All ID segments created by the session are killed at log off. The one exception here is that if the program occupying the ID segment is currently active and was run by another session, then the ID segment is given to that session. The session identifier in the ID segment is changed to that of the other session. Note that programs created under the "MANAGER.SYS" account or programs not created under session will have an ID = 0. Therefore, the LGOFF program will not remove programs thus created.

Word 32. In session-systems ID word 33 is the SST address of the session this program is operating under.

In MTM systems ID 33 is the -LU of the terminal associated with the program.

When operating in the non-session environments, ID 33 is set to 0.

Note that this word is passed down from father to son programs.

ID EXTENSION:

Word 0 and 1 of the ID extension contains the I/O bit, DE bit, the current MSEG field, the MSEG Size field, starting prog page MSEG field, and start page EMA field.

I/O This bit is set whenever the MSEG is modified in order to do I/O which crosses an MSEG boundary. It indicates the current MSEG number is not valid.

DE This bit is set if the default EMA size is taken, the bit is not effected by MSEG.

MSEG This is the size in pages of the current mapping segment. It can be declared by the user program or defaulted. For default:

MSEG = Maximum Logical Address Space -[PROGRAM SIZE
+LARGEST SEGMENT
+SUBROUTINES]

Current MSEG. Number of the currently mapped MSEG.

START
PROG
PAGE
MSEG This is the logical starting page of the mapping segment.

START
PAGE
DMA Physical starting page # of the EMA-i.e., first mapping segment directly behind the program.

Word 2 of the ID extension is the swap address of the EMA area of the program. It is the number of tracks of EMA array swapped to the disc. The swap location of the EMA array will begin on the first track following the swapped program.

KEYWORD BLOCK:

Because ID segment sizes vary (memory resident size = 28, disc resident size = 33, program segment size = 9), some method of indexing to the first word or the name word of an ID segment is necessary. The keyword block is used for this purpose.

Location 1657B on base page specifies the first entry in the keyword block. The keyword block in turn contains 1-word entries, each pointing to an ID segment, last entry=0. Keyword Block entries are ordered at generation by program type; memory resident, real time disc resident, background disc resident, available ID segments, and the program segment ID segments.

The keyword block entry (ID address) + 12 always points to the name-word. Thus, keyword entries for short ID's don't point to the first word of the ID.

NOTE: Keyword +12 always points to name.

ID EXTENSION KEYWORD BLOCK:

Like the keyword block for ID segments, the ID EXTENSIONS have a table of pointers terminated by a 0. Word $IDEX points to the ID extension keyword block (see Figure 1-1A). Each word in that block points to an ID EXTENSION.

Appendix E

DISPATCHER INTERFACE TO LIST PROCESSOR

As mentioned in Chapter 1, $LIST pushes programs that terminate into a stack through word 8 of the ID segment headed at $ZZZZ in the dispatcher. The dispatcher uses this stack to do program clean up. Every time the system has nothing else to do, it jumps to $XEQ in the dispatcher. Every time the system goes to $XEQ it first checks $ZZZZ. If it is non-zero it does the following to clean up every program:

First it sets the program's point of suspension (ID word 8) to 0 in case the program is to be run later. Next if the program is disc resident, any swap tracks it may have are released. This may happen if a program that is swapped out is aborted. A father may also abort his son causing this condition. The tracks are released by $DREL in the EXEC. $DREL also makes a call to $LIST in the scheduler to reschedule any programs that may have been waiting for disc space.

The dispatcher then calls $ABRE in EXEC to return any reentrant memory the program may have. This may happen if a program terminates or is aborted while in a reentrant subroutine. If the $ABRE routine returns any memory via the $RTN subroutine in $ALC, programs waiting for memory may be rescheduled by calls to the list processor.

Next a call is made to the $RTST routine in the scheduler. This routine returns any string memory the program may own. If any memory is returned, programs waiting for memory may be scheduled by a call to the list processor. The system then calls $WATR in the scheduler to schedule any programs that made SCHEDULE WITH QUEUE requests (EXEC 23,24) for the program. $WATR calls $SCD3 which calls $LIST for any such programs. $SCD3 scans the general wait list (major state=3) looking for entries which have word 1 of their ID segment equal to the ID segment address of the terminating program. Programs in the general wait list will have Word 1 of their ID segment set as follows:

REASONS	CONTENTS OF ID(1)
WAIT TO SCHEDULE A PROGRAM	The programs ID segment address
WAIT FOR COMPLETION OF A "SON"	The "sons" ID segment address
RN ALLOCATE WAIT	Address of the RN table
RN LOCK WAIT	Address of the RN number
LU LOCK WAIT	Address of the RN number associated with the LU LOCK
DOWN DEVICE	4
BUFFER LIMIT EXCEEDED	Address of the EQT on which the limitation was exceeded

NOTE: This call also handles the programs that scheduled with QUEUE.

After $WATR returns, the system calls $TRRN which calls $ULLU to unlock any lock LU's the program has. $TRRN also unlocks any local RN locks and deallocates any local RN allocates the program may have. Each of these processes may call $SCD3 to pick up and schedule waiting programs. If there are any such programs $SCD3 will call $LIST.

Lastly, if the program is a disc resident program and the program still owns the partition but did not make a serial reusable termination, then that partition is released and made available to other programs.

$LIST CALLS USED IN DVR00, DVR05, DVR37

DVR00 and DVR05 both schedule the system program PRMPT with a $LIST call. In addition, the B register at suspension (word 11) is set to point to the EQT word 4 of the EQT of the interrupting device. Both of these functions were separate but now will be condensed into one $LIST call. The calling sequence is:

```
JSB $LIST
OCT 601
OCT IDADR <ID ADDRESS>
OCT BVAL  <THIS VALUE PLACED INTO ID WORD 4>
```

The HP-IB driver (DVR37) will schedule a program on interrupt and pass three words into the temporary area of that programs ID segment. The $LIST calling sequence to do this is:

```
JSB $LIST
OCT 1
DEF RTN
DEF IDADR
DEF P1
DEF P2
DEF P3

RTN
```

Appendix F

MEMORY ADDRESSING SPACES

Figure F-1 shows Memory Addressing Spaces:

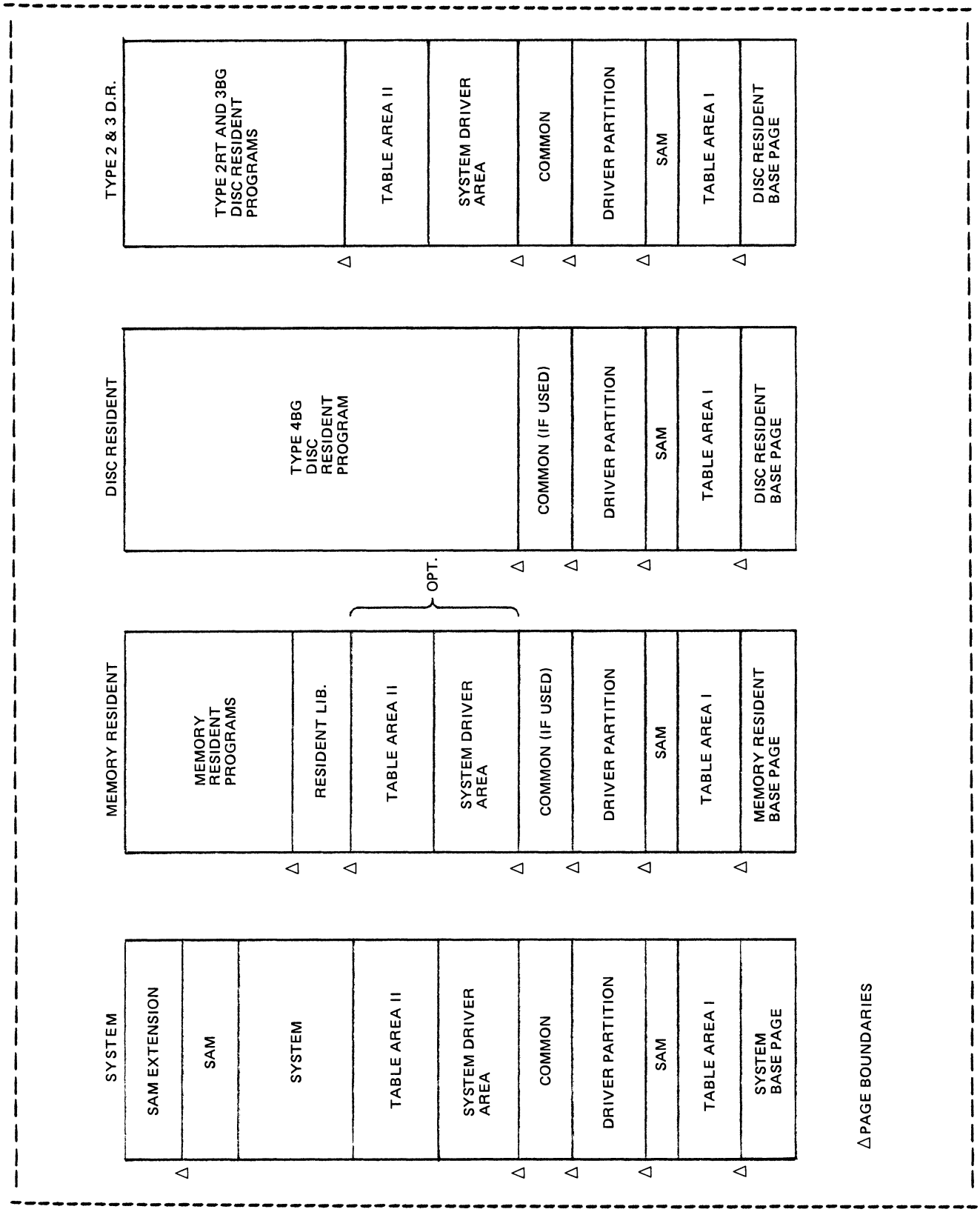


Figure F-1. Memory Addressing Spaces

Appendix G
RTE-IVB SYSTEM DISC LAYOUT

Figure G-1 shows the RTE-IVB System Disc Layout.

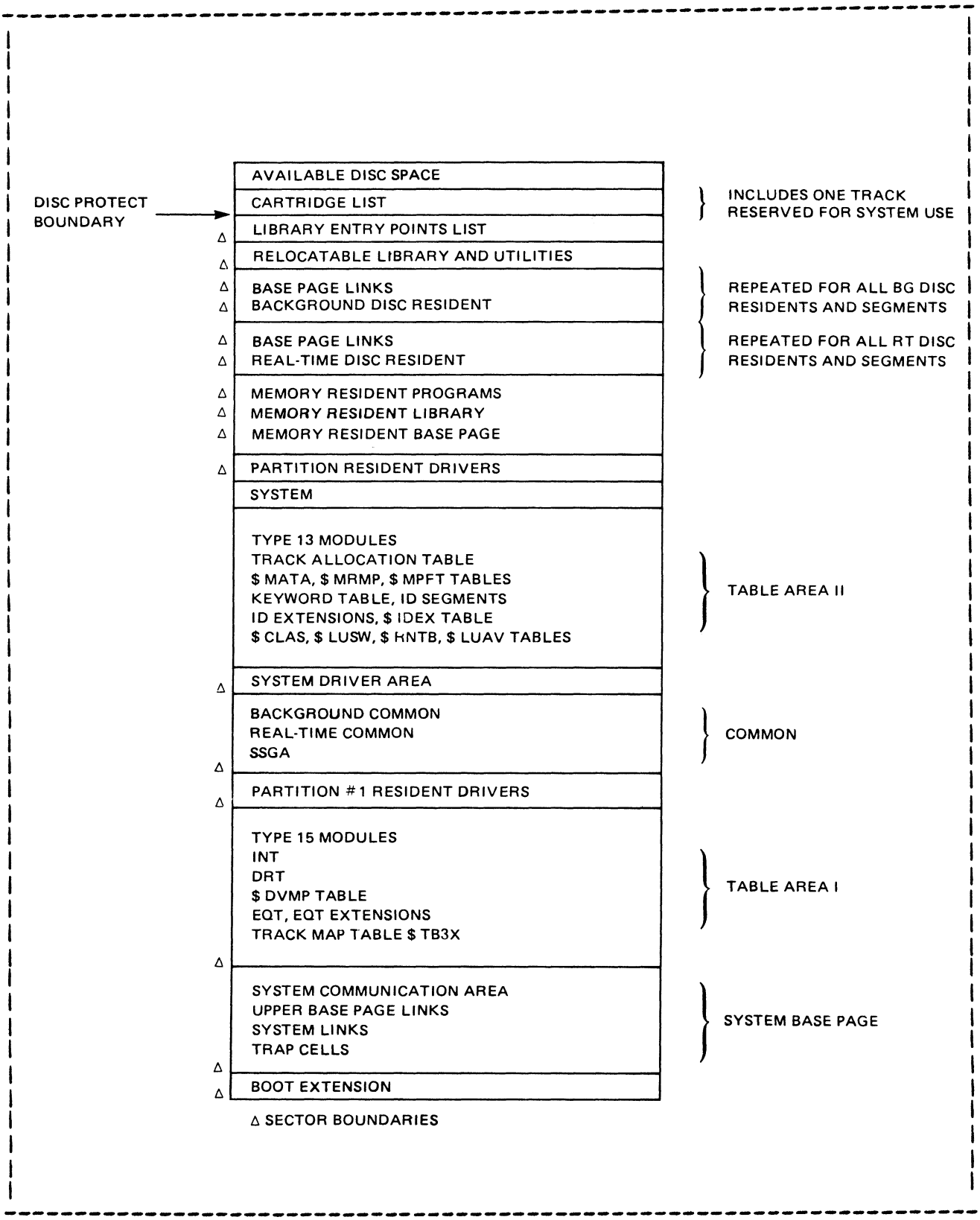
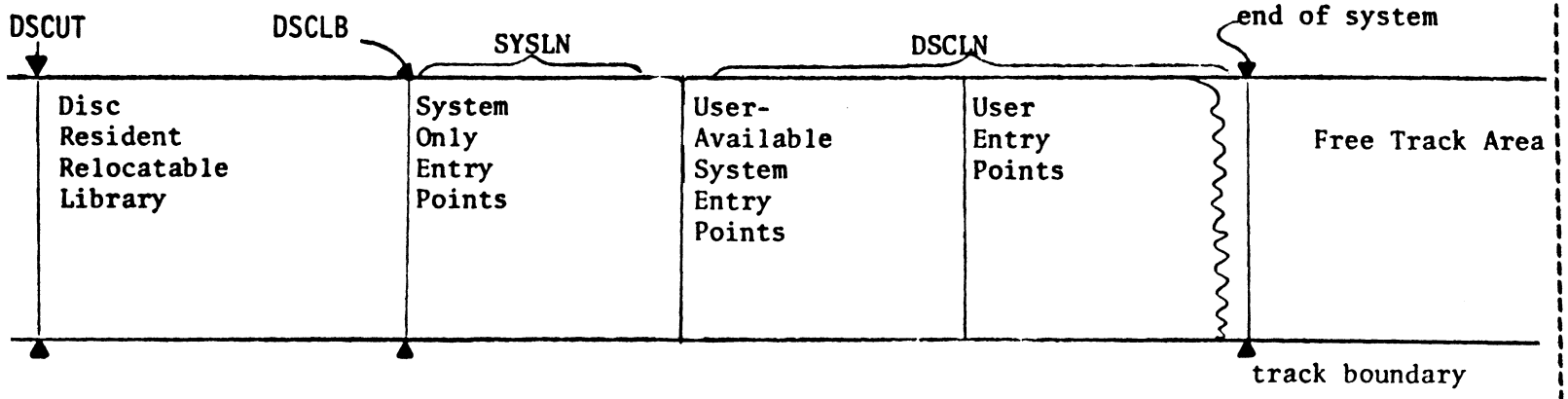


Figure G-1. RTE-IVB System Disc Layout

Appendix H

ENTRY POINT LAYOUT ON THE DISC

Figure H-1 shows Entry Point Layout on the Disc.



LIBRARY ENTRY POINTS LIST

FORMAT:
- word 1 — name 1,2
- word 2 — name 3,4
- word 3 — name 5, flag bits
- word 4 — value

flag bits:

000	memory resident
001	disc resident
010	common
011	absolute
100	replace

NOTE: SYSLN & DSCLN contain the number of 4-word entries

▲ sector boundaries

Appendix I
RTE-IVB PHYSICAL MEMORY ALLOCATION

Figure I-1 shows RTE-IVB Physical Memory Allocation.

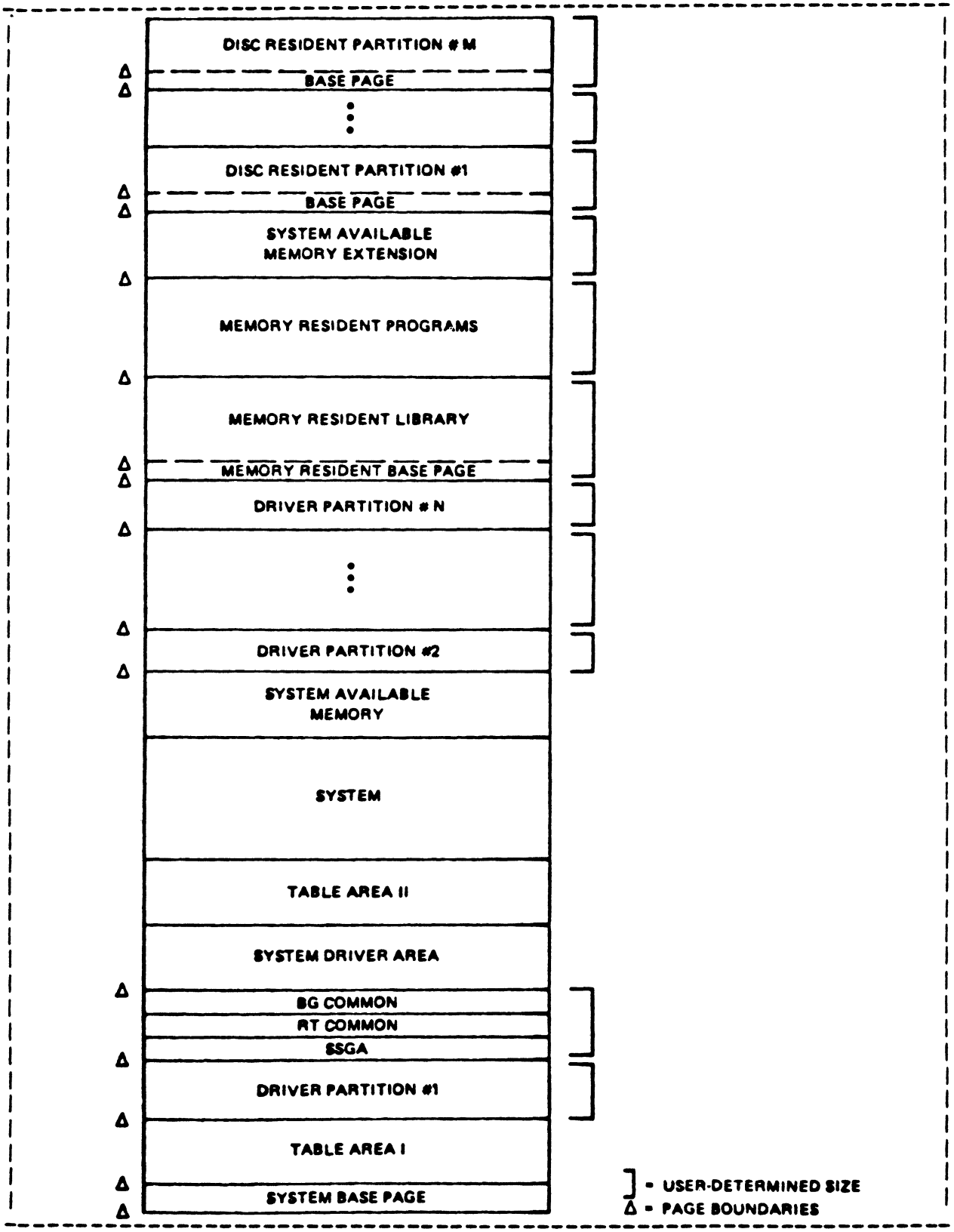


Figure I-1. RTE-IV Physical Memory Allocation

Appendix J

TIMESLICE QUANTUM DEFINITION

TIMESLICE QUANTUM DEFAULT

The default timeslice quantum is defined as 1.5 seconds. The major factor considered in determining the default slice was the percentage of time spent swapping the timeslice programs as compared to the execution slice allowed.

To arrive at a slice value (x), the following assumptions were made:

```
program size = 16K (swap time (y) = 300 ms).
y/x          = .20 (swap time in relation to execution slice).
```

Plug in the swap time to get .300/x = .20 or 1500 ms (1.5 sec).

The swap overhead is defined as follows:

	7925	7920	7905
Worst case seek time:	48.5ms	45ms	45ms
Average latency	11.1ms	8.33ms	8.33ms
Total access time =	59.6ms	53.33ms	53.33ms

Using an average access time of 55ms for the above discs, the time to swap a program (16K) would be:

```
 90ms of data transfer
+55ms of access time
-----
=145ms
 x 2
---
290ms = worst case swap of a 16K program.
```

The following section is intended for those people familiar with the RTE-III and RTE-IV Dispatcher, Scheduler and RTIME modules.

TIMESLICING AND THE DISPATCHER

The following routines make special checks to provide for the implementation of the timeslicing function.

The Switching section (X0010) - If the program from the scheduled list has equal priority and current program (XEQT) have used a full timeslice, then attempt execution switching.

The partition search routine (FNDSG) - When searching the allocated list, if the resident is of equal priority (with program in scheduled list) and has used a full timeslice (indicated by ID word 30 containing a zero value), allow the allocation of the partition to the new program.

The swap check routine (SWPCK) - If the residents priority is equal to the contender and the resident is flagged as having used a full slice allow the swap.

Program execution setup (X0042) - Note that this section is entered only when a new program is to be dispatched. If the current program (XEQT) is to be reentered, set-up is at $RENT.

If priority of new program is <timeslice limit (default is 41 but this value may be changed via the "QU" operator command), setup a dummy timeslice location and continue with the dispatch.

If the priority of new program is > timeslice limit, set the address of ID 30 as the timeslice location ($LICE). If the contents of ID word 30 is negative (remaining timeslice count), continue with the dispatch.

If the remaining count is > 0 (from ID word 30), calculate a full timeslice value using the following equation:

Program slice value = SYS Slice * Z + SYS Slice

where: SYS Slice = 1500ms default. This value (1.5 seconds) may be increased or decreased via the "QU" operator command.

If the new program is under session control, set the CPU usage word ($CPU) to point at the programs session control block word 10.

If not under session, set the CPU usage pointer to point at a dummy system location.

TIMESLICING AND RTIME

Before checking for time scheduled programs (CL010), RTIME performs the following functions:

1. Increment the slice counter (indirect through $LICE). If not zero, continue. Else, if the currently executing program's priority is > timeslice limit and another program of the same priority is scheduled, relink (VIA $RLNK) the currently executing program and force a new dispatch (i.e., put it behind all other programs of the same priority. Refer to the section on Timeslicing and the Dispatcher for details on the dispatch of the new program.

   Note that a zero value in ID word 30 indicates the usage of a full timeslice.

2. Increment the double word CPU usage word (indirect through $CPU).

TIMESLICING AND SCHEDULING

A new subroutine was created to provide the RTIME module with a very fast method for relinking a program in the scheduled list (within its own priority level). All it does is call the link processor to remove the XEQT programs from the scheduled list and then insert it into the scheduled list.

Appendix K

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 K-1.

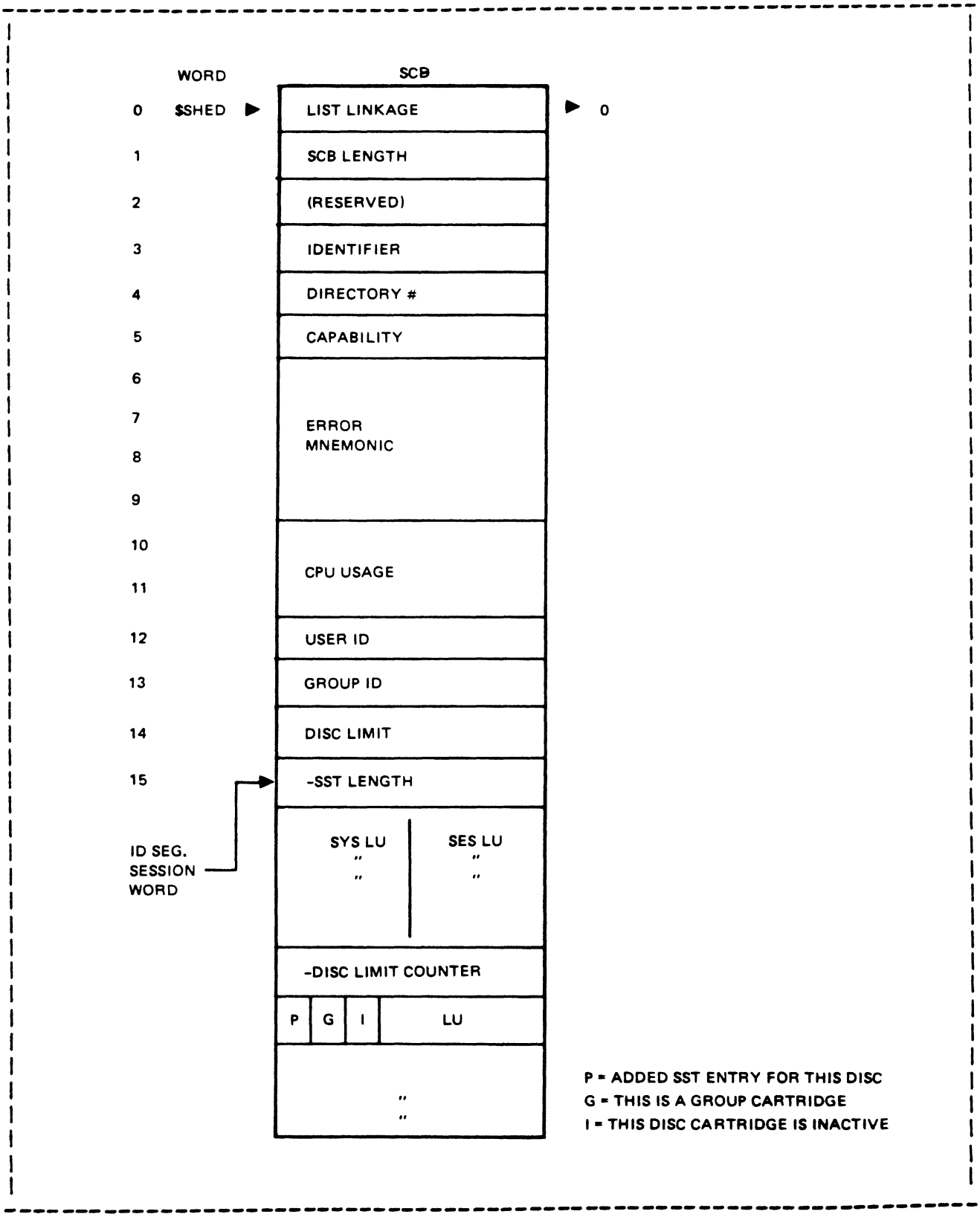


Figure K-1. SCB

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 LU's 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. 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 Session Control Block (SCB). The format of the SST is shown in Figure K-2.

System LUs can be integer numbers between 1 and 254. 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

* at log-on, via Configuration Table entries, or

* on-line using SL command (refer to RTE-IVB Terminal User's Reference Manual).

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 K-3, below.

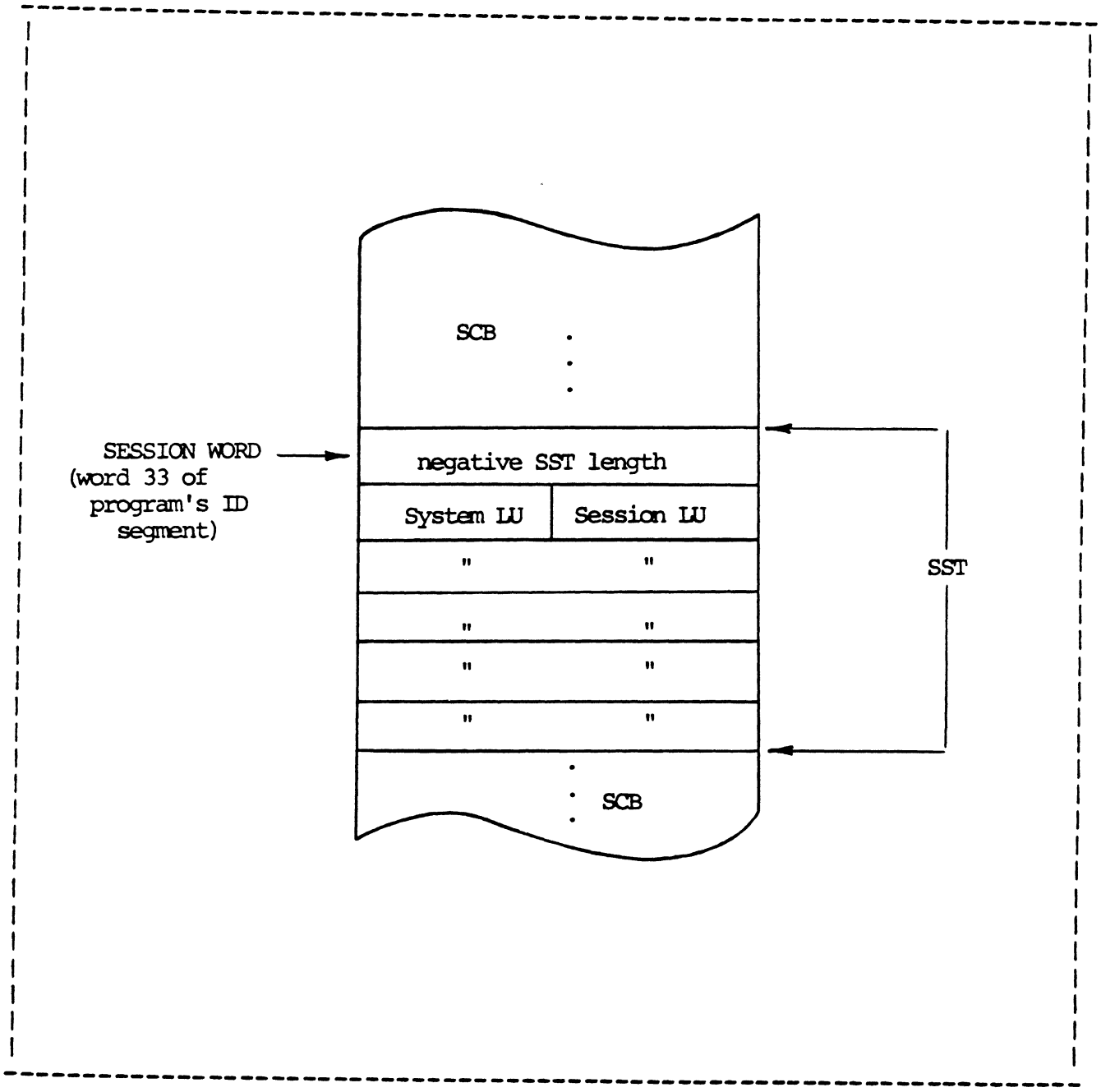


Figure K-2. Session Switch Table (SST) Format

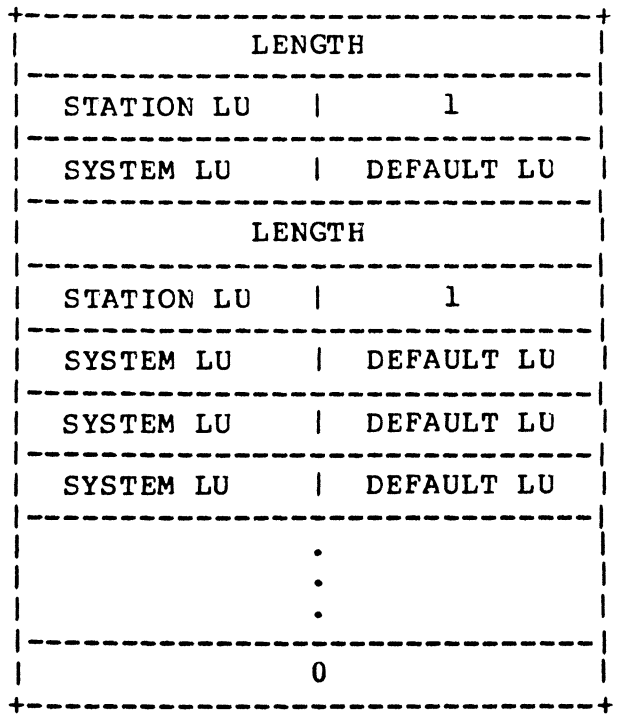
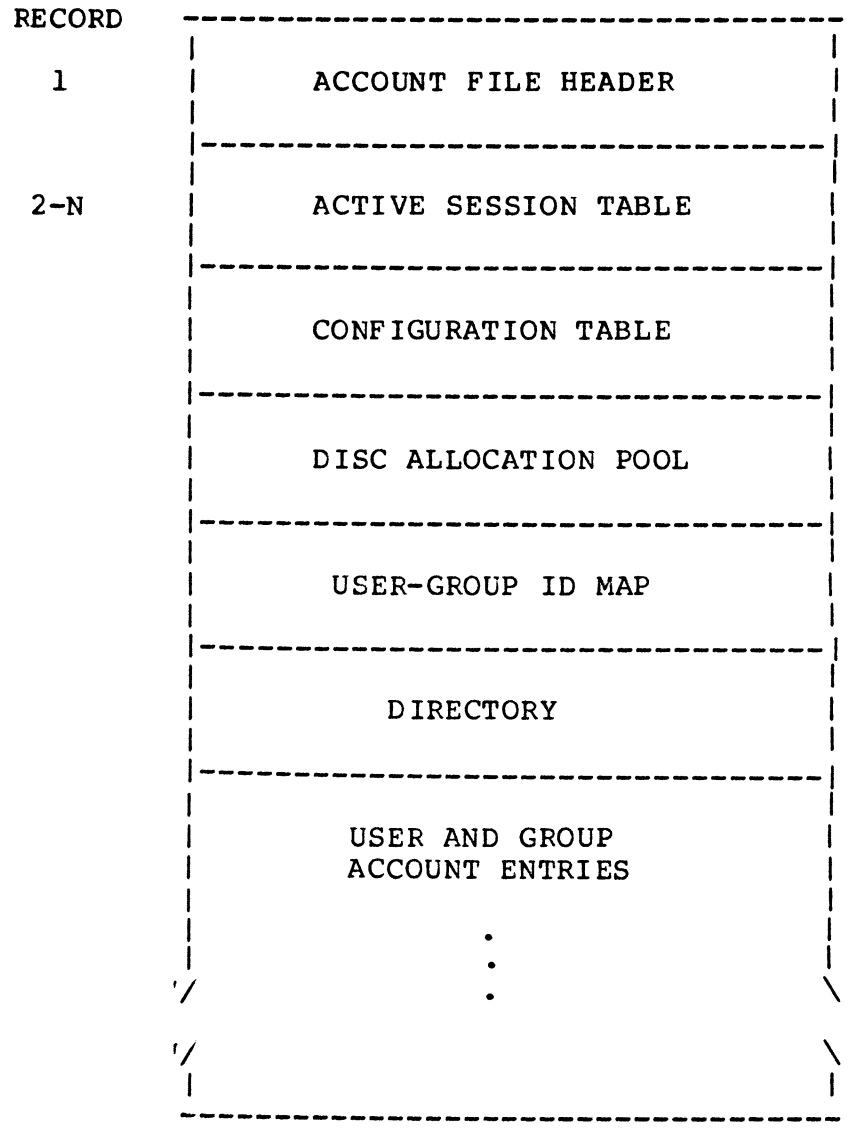


Figure K-3. Configuration Table

The account file structure is shown as follows:

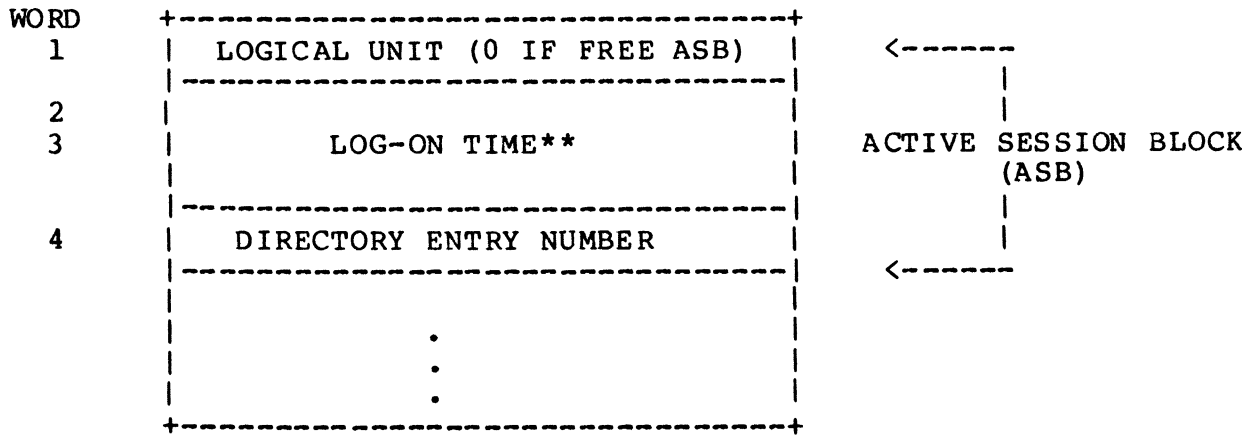
ACCOUNT FILE STRUCTURE



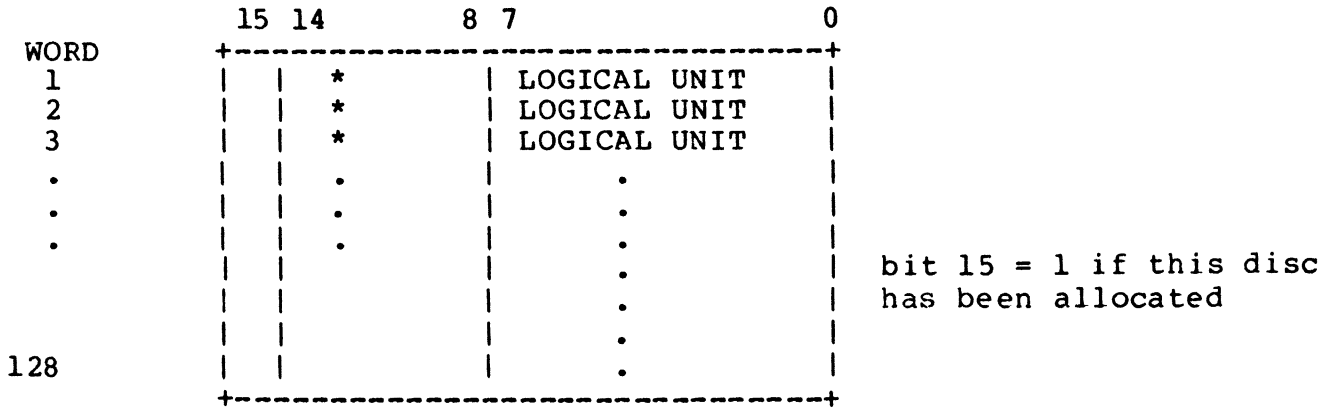
ACCOUNT FILE HEADER

WORD		
1	LOCATION OF ACTIVE SESSN TABLE	
2	LOCATION OF CONFIGURATION TBL	
3	LOCATION OF DISC POOL	
4	LOCATION OF USER/GROUP ID MAP	
5	LOCATION OF DIRECTORY	
6	LOCATION OF 1ST ACCOUNT ENTRY	
7-9	SYSTEM MESSAGE FILE	
10	SECURITY CODE	
11	CARTRIDGE	
12	# OF CHARS IN PROMPT STRING	<--0 if using default prompt
13-22	PROMPT STRING	
23	LOWEST PRIVATE ID USED	
24	HIGHEST GROUP ID USED	
25	RESOURCE NO.	
26	LU # OF MSG. FILES	
27	I MEMORY ALLOCATION SIZE (WDS)	If bit 15=1, use session monitor memory allocation
28	- SESSION LIMIT	
29	NUMBER OF ACTIVE SESSIONS	
30	SHUT DOWN FLAG	
31	COPY OF SESSION LIMIT	
32	CLASS NUMBER	
33	LENGTH OF CONFIG TABLE	
34	IRN2	
35	DISC POOL LENGTH	
36-128		

ACTIVE SESSION TABLE



DISC ALLOCATION POOL

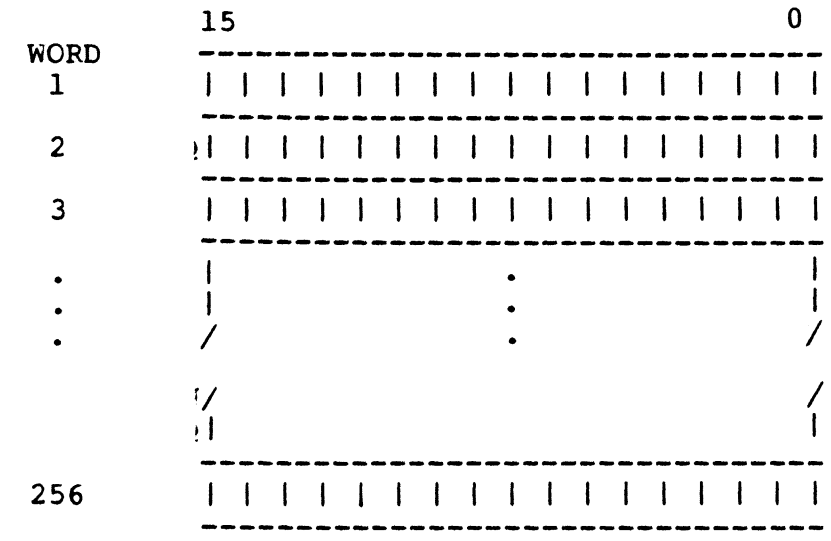


* RESERVED FOR FUTURE USE

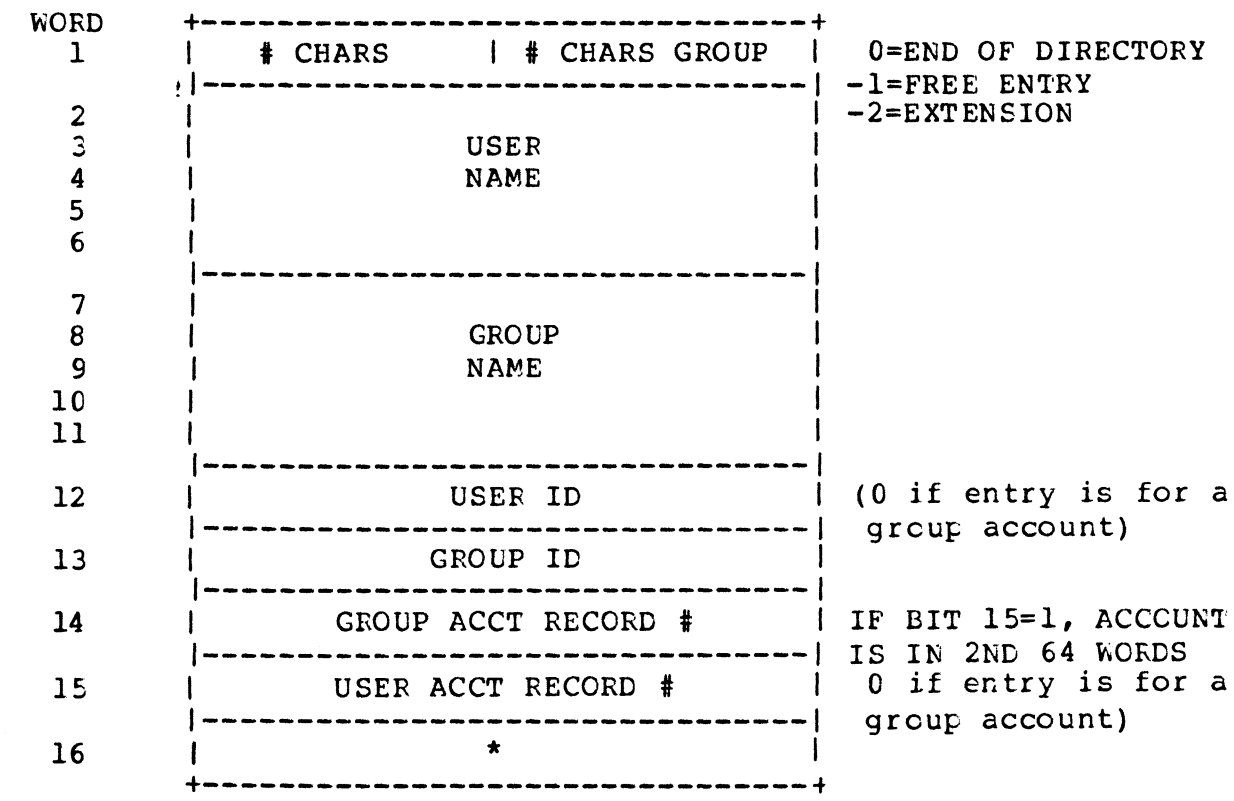
** LOG-ON TIME FORMAT

	15-13	12-7	6-0
WD1	-----	----	---
WD1	year offset from 1978	min	sec
	15-14	13-5	4-0
	-----	----	---
WD2	reserved	day	hour

USER/GROUP ID MAP

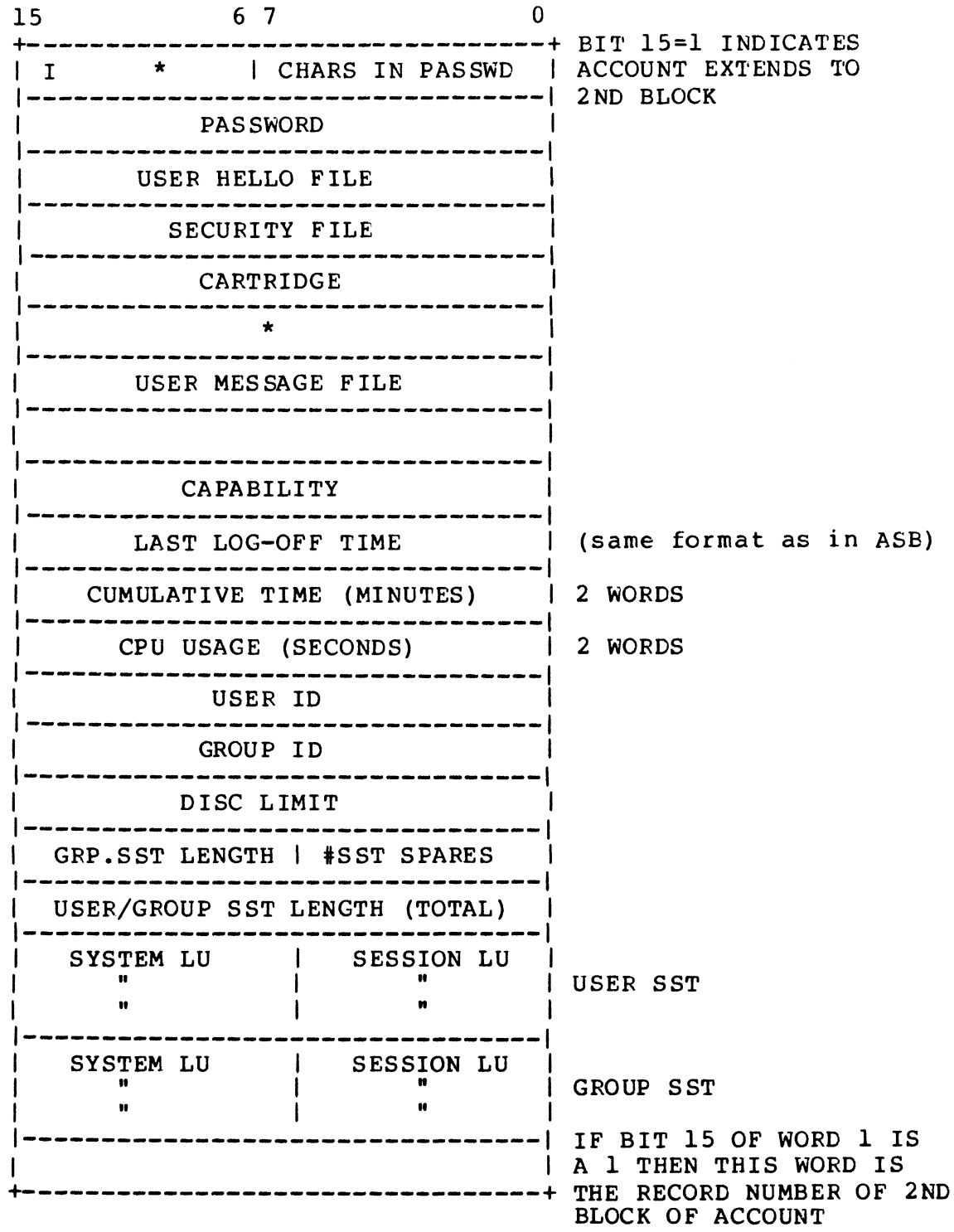


ACCOUNT FILE DIRECTORY

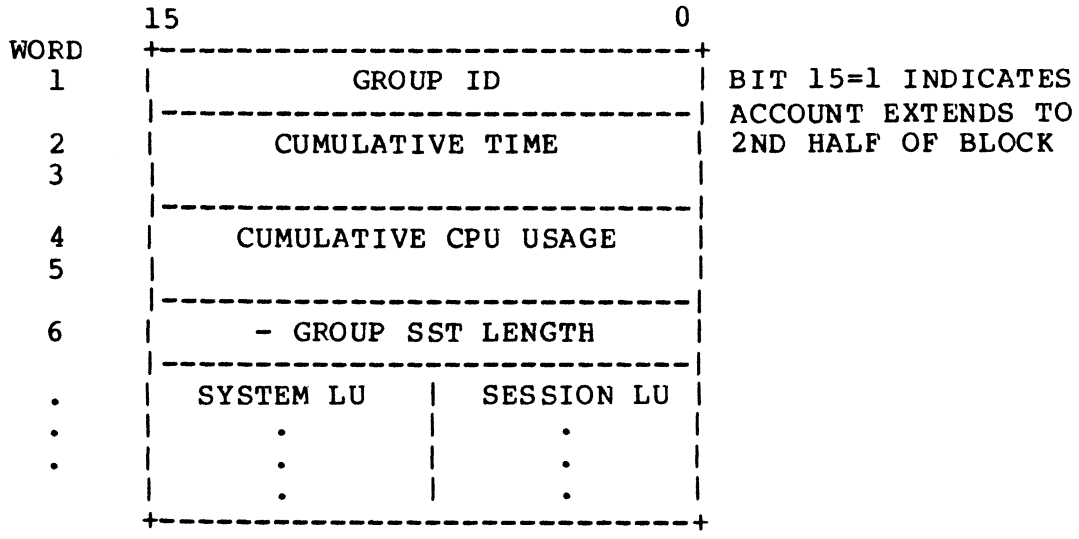


* RESERVED FOR FUTURE USE

USER ACCOUNT ENTRY

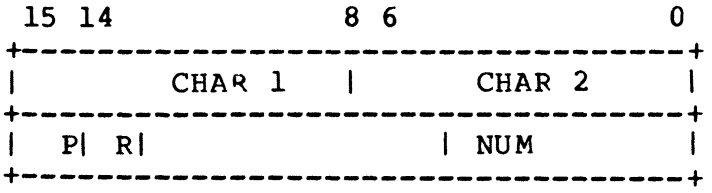


GROUP ACCOUNT ENTRY



COMMAND TABLE

A listing of the operating system command capability table, $CMND appears on the following pages. Each command is defined by a two-word entry of the form:



Where: CHAR1 and CHAR2 = the two character ASCII command

P = 0 If any number of parameters allowed.

= 1 If a limitation is placed on the number of parameters allowed.

NUM = The maximum number of parameters allowed with this command (specified when P=1).

R = 0 No reference check required.

= 1 Program specified for first parameter of command must be attached to this session (ID segment word 33 of program must equal word 33 of caller) or program must be non-session (word 33 equals zero).

The command capability level associated with a command will be determined by the position of the command entry relative to level pointers located at the head of the table. Refer to the listing for details.

If you wish to substitute your own command table for the HP supplied table, it must be specified AFTER the operating system relocatables during generation.

The op system command capability table follows on subsequent pages. The FMGR command table is listed in Chapter 16. The capability levels assigned to various commands depends on their position whitin the table relative to table pointers located at the front of the command table. Each command is defined by a two-word entry. To change the capability level of a command, relocate the two-word entry to the appropriate table section for the desired capability level. Do not modify the two-word entry.

Then reassemble the modified capability table and relocate it after the file manager modules (i.e., %BMPG1, etc.) during generation. You can ignore GEN05 and GEN08 errors here.

NOTE:

Hewlett-Packard does not support modified command capability table.

```
0007  *  ********************************************************************
0008  *  * (C) COPYRIGHT HEWLETT-PACKARD COMPANY 1978.  ALL RIGHTS     *
0009  *  * RESERVED.  NO PART OF THIS PROGRAM MAY BE PHOTOCOPIED,       *
0010  *  * REPRODUCED OR TRANSLATED TO ANOTHER PROGRAM LANGUAGE WITHOUT*
0011  *  * THE PRIOR WRITTEN CONSENT OF HEWLETT-PACKARD COMPANY.       *
0012  *  ********************************************************************
0013  *
0014        NAM $CMND,0 92067-16261 REV.1903 790506
0015        ENT $CMND
0016  *
0017  *
0018  $CMND DEF EINDX     DEFINE THE ADDRESS OF HIGHEST CAPABILITY
0019        DEF BEGIN     DEFINE BEGINNING OF TABLE
0020        DEF END       DEFINE END OF TABLE
0021  *
0022  L60   DEC -60       LEVEL 60
0023  L60A  DEF BEGIN     DEFINE START OF THIS CAPABILITY
0024  *
0025  L50   DEC -50
0026  L50A  DEF L.50
0027  *
0028  L30   DEC -30
0029  L30A  DEF L.30
0030  *
0031  L10   DEC -10
0032  L10A  DEF L.10
0033  *
0034  L00   NOP
0035  L00A  DEF L.00
0036  *
0037  EINDX EQU *-2
0038  *
0039  *
0040  *
0041        ORG $CMND
0042        BSS L10A-L30A
0043        BSS L30A-L50A
0044        BSS L50A-L60A
0045        ORR
0046  *
0047        SKP
0048  L.60  EQU *
0049  BEGIN ASC 1,QU
```

```
0050        OCT 0
0051        ASC 1,DN
0052        OCT 0
0053        ASC 1,LU
0054        OCT 0
0055        ASC 1,EQ
0056        OCT 0
0057        ASC 1,TO
0058        OCT 0
0059        ASC 1,BL
0060        OCT 0
0061        ASC 1,TM
0062        OCT 0
0063        ASC 1,OF
0064        OCT 0
0065        ASC 1,BR
0066        OCT 0
0067        ASC 1,GO
0068        OCT 0
0069        ASC 1,SS
0070        OCT 0
0071        ASC 1,RT
0072        OCT 0
0073  *
0074  L.50  ASC 1,IT
0075        OCT 0
0076        ASC 1,L3      ABILITY TO ADD AN ENTRY IN SST -- SL CMND
0077        OCT 0
0078        ASC 1,AS
0079        OCT 0
0080        ASC 1,UR
0081        OCT 0
0082        ASC 1,DN
0083        OCT 0
0084        ASC 1,PR
0085        OCT 0
0086  *
0087  L.30  ASC 1,RU
0088        OCT 0
0089        ASC 1,OF
0090        OCT 40000
0091        ASC 1,SS
0092        OCT 40000
0093        ASC 1,GO
0094        OCT 40000
0095        ASC 1,RT
0096        OCT 0
0097        ASC 1,SZ
0098        OCT 0
0099        ASC 1,L2      LEVEL 2 SL CMND -- SPOOL AN LU
0100        OCT 0
0101  *
0102  L.10  ASC 1,FL
0103        OCT 0
0104        ASC 1,RS
0105        OCT 0
0106        ASC 1,QU
0107        OCT 100000
0108        ASC 1,BL
0109        OCT 100000
0110        ASC 1,ST
0111        OCT 0
0112        ASC 1,BR
0113        OCT 40000
0114        ASC 1,EQ
0115        OCT 100001
0116        ASC 1,SL
0117        OCT 0
0118        ASC 1,TO
0119        OCT 100001
0120        ASC 1,TE
0121        OCT 0
0122        ASC 1,WH
0123        OCT 0
0124        ASC 1,TI
0125        OCT 0
0126        ASC 1,UP
0127        OCT 0
0128        ASC 1,EN
0129        OCT 0
0130  *
0131  L.00  ASC 1,OP
0132        OCT 0
0133        ASC 1,HE
0134        OCT 0
0135  *
0136  END   EQU *-2
0137        END $CMND
```

OPEN

```
P1.  1,ID              (bit 15 set)

P2.  bit 15 set = session monitor override   (all discs)
     bit 14 set = session monitor override   (priv, group, system only)
     bit 13 set = session monitor override   (system discs only)

P3.  -LU, +CRN, 0

P4.  security code

STRING:  1.  E,NAME(1,2)          E - bit 15 = set if exclusive open
         2.  S,NAME(3,4)          S - bit 15 = set if scratch file purge

         3.  NAME(5,6)
```

CLOSE

```
P1.  ID

P2.  0

P3.  -\
       > Directory address
P4.  -/

STRING:  1.  -\  Negative - Double word # of sectors to be truncated
         2.  -/  Positive - purge extents only
```

CREAT

P1. ID

P2. 1 bit 15 set = sess monit. override (all discs)
bit 14 set = sess monit. override (priv, group, system only)

bit 13 set = sess monit. override (system discs only)

P3. -LU, +CRN, 0

STRING: 1. NAME(1,2)
2. NAME(3,4)
3. NAME(5,6)
4. Type
5.
6. Double word size in +(sectors) or -(128 block multiples)
7. OR - double word -1 = allocate rest of disc (<=32767 x 128)

OR - single word -1 = allocate rest of disc (<=16383 blocks

8. Record length
9. Security code

CHANGE NAME

P1. ID

P2. 2

P3. -\
> Directory address
P4. -/

STRING: 1. -\ 6-character
2. -> new
3. -/ name

SET, CLEAR LOCK

P1. ID

P2. 3 for set, 5 for clear
bit 15 set = session monitor override (all discs)
bit 14 set = session monitor override (priv, group, system only)
bit 13 set = session monitor override (system discs only)

P3. -LU, +CRN 0 not legal

EXTENSION OPEN

P1. ID

P2. 6 for read, 8 for write

P3. -\
  > Directory address of main file
P4. -/

P5. Extent number

GENERATE, PACK, UPDATE

P1. ID

P2. 7 bit 15 set = sess monit. override (all discs)
bit 14 set = sess monit. override (priv, group, system only)

bit 13 set = sess monit. override (system discs only)

P3. -LU, +CRN 0 not legal

P4. S, #sectors/track S - bit 15 = set if disc directory update

STRING: 1. -\ Data track
2. -/ address

PACK

P1. ID

P2. 9 bit 15 set = sess monit. override (all discs)
bit 14 set = sess monit. override (priv, group, system only)

bit 13 set = sess monit. override (system discs only)

P3. -LU, +CRN 0 not legal

P4. Relative directory sector

STRING: -\ 128-word
-\ directory
-/ sector to
-/ be written

MOUNT CARTRIDGE

P1. ID

P2. 13 bit 15 set = sess monit. override (all discs)
bit 14 set = sess monit. override (priv, group, system only)

bit 13 set = sess monit. override (system discs only)

P3. -LU 0 not legal

P4. SCB Address if mounting to session other than one operating under

P5. ID to which disc is to be mounted.
bit 15 = 1 initialize directory

STRING: -\ 1st 9 words String is passed only if disc's file
-\ of cartridge directory is to be initialized. If no
-/ specif string is passed, adding the CL entry
-/ entry to the directory is all that is done.

REMOVE CARTRIDGE

P1. ID

P2. 11 bit 15 set = sess monit. override (all discs)
bit 14 set = sess monit. override (priv, group, system only
bit 13 set = sess monit. override (system discs only)

P3. -LU 0 not legal

ALTER CL ENTRY

P1. ID

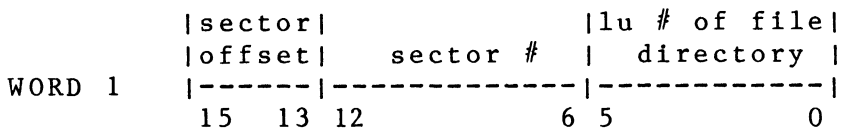
P2. 13 bit 15 set = sess monit. override (all discs)
bit 14 set = sess monit. override (priv, group, system only)
bit 13 set = sess monit. override (system discs only)

P3. -LU, +CRN 0 not legal

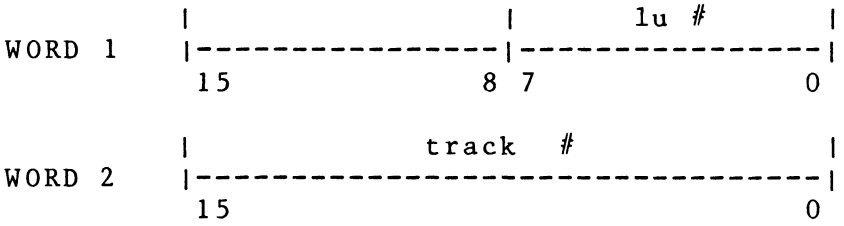
STRING: -\ 4 word
-\ cartridge
-/ directory
-/ entry

ID = ID segment address of calling program

Directory Address Format



Data Track Address Format



RETURN PARAMETERS

R1. Error code OR 0

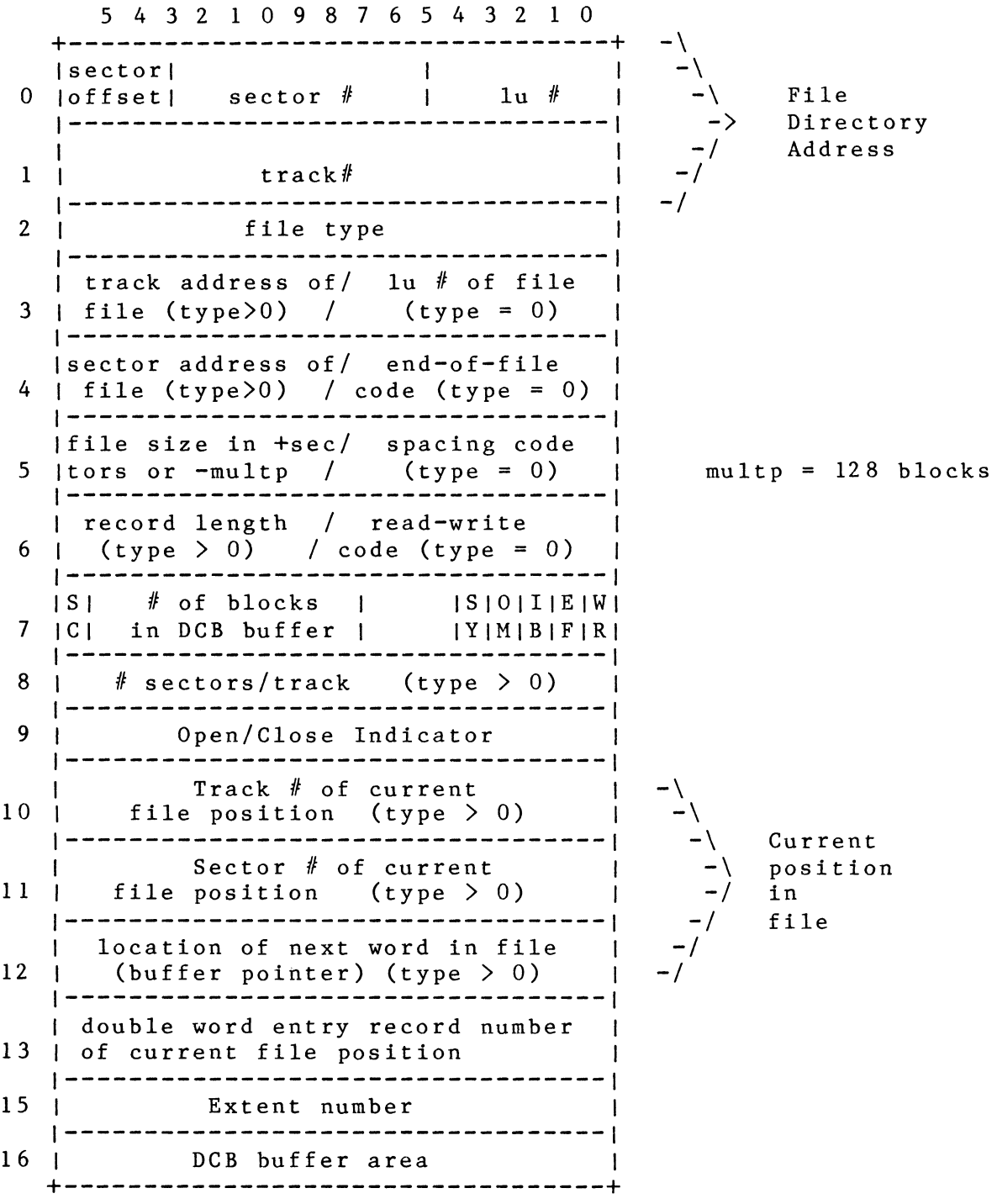
R2. -\
> Directory address
R3. -/

R4. Starting track # of file
LU # if type = 0

R5. #sectors/track (bits 8-15) Starting sector (bits 0-7)

String is returned only for OPEN and CREAT calls.
STRING: 1. File type bit 15 set indicates this is LU 2 or 3
2. Starting track
3. Extent # (bits 8-15) Starting sector (bits 0-7)
4. Size in +sectors or -128 block multiples
5. Record length
6. Security code

DATA CONTROL BLOCK AND FILE DIRECTORY FORMATS

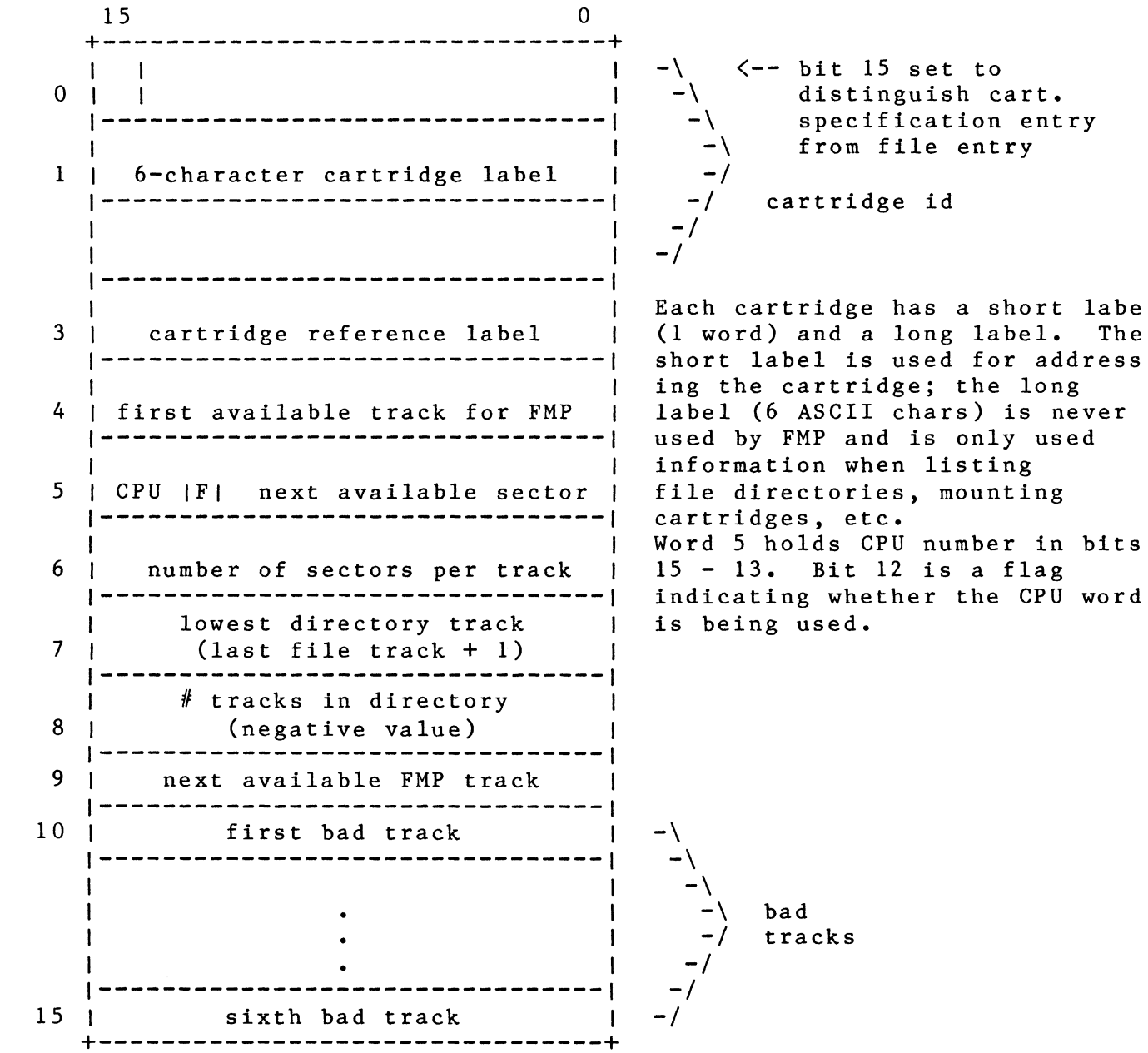


LEGEND FOR DATA CONTROL BLOCK

WORD		CONTENT
4	End-of File Code, type 0 file:	011u = EOF on Magnetic Tape 101u = EOF on Paper Tape 111u = 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 don't agree
	DCB Buffer:	bits 14-7 = # blocks in DCB buffer
	(SY) System Disc	bit 4 = 1 - file 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
	EOF Read Flag:	bit 1 = 1 - EOF has been read = 0 - EOF has not been read
	(WR) To Be Written Flag:	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.

FILE DIRECTORY FORMAT

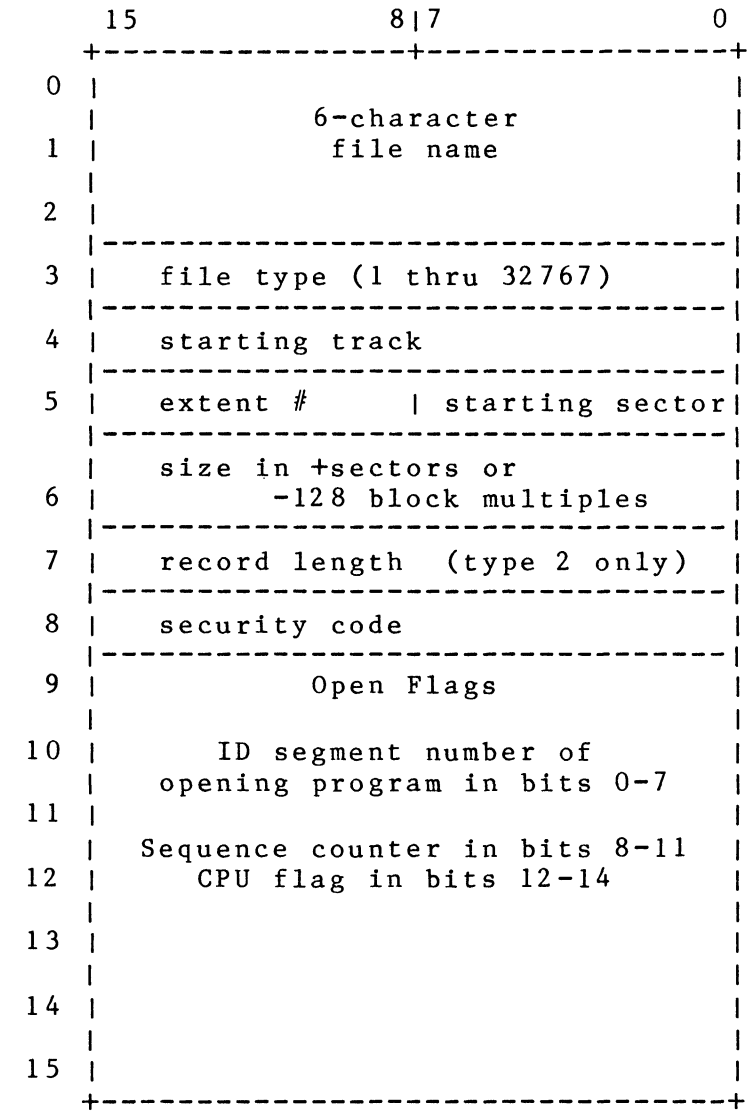
The file directory starts in sector 0 of the last track on all disc lu's. The first entry in each File Directory is the specification entry for the cartridge itself:



Each cartridge has a short label (1 word) and a long label. The short label is used for addressing the cartridge; the long label (6 ASCII chars) is never used by FMP and is only used information when listing file directories, mounting cartridges, etc.

Word 5 holds CPU number in bits 15 - 13. Bit 12 is a flag indicating whether the CPU word is being used.

The 16-word cartridge entry is followed by an entry for each created file.

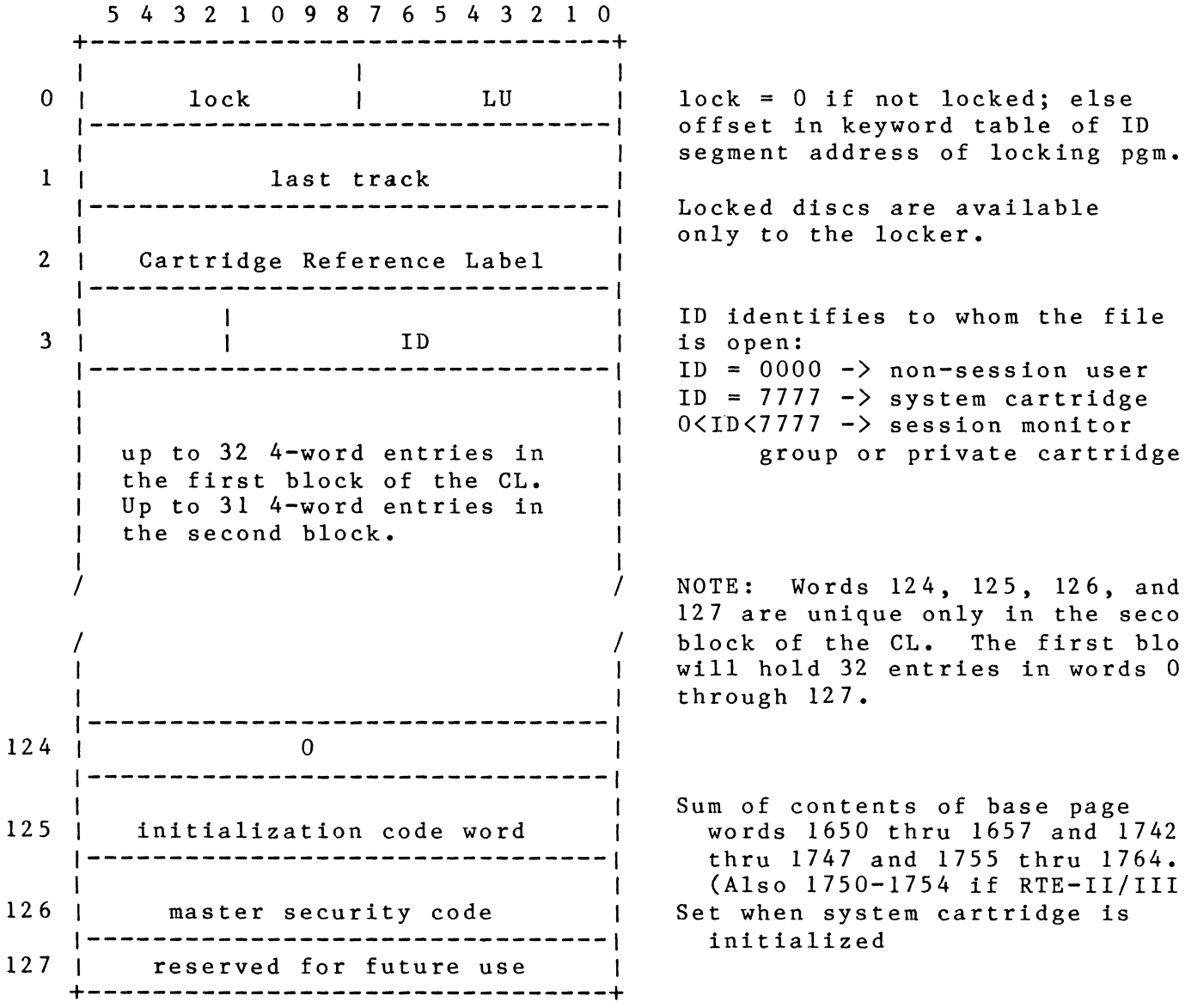


An exclusive open is indicate by the sign bit being set on the only open flag.

Word 0 = 0 if the last entry in directory
= -1 if file is purged

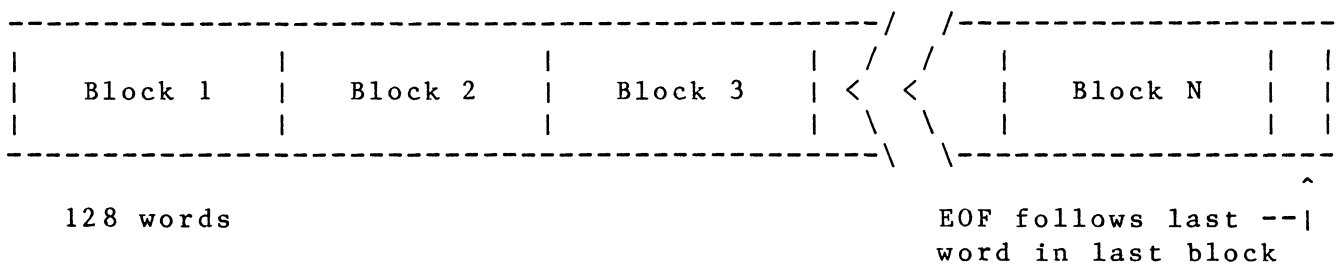
CARTRIDGE DIRECTORY AND FILE RECORD FORMATS

The cartridge directory is two blocks long and is located in the system area on LU 2. The track and sector address are defined in entry points $CL1 and $CL2.



DISC FILE RECORD FORMATS

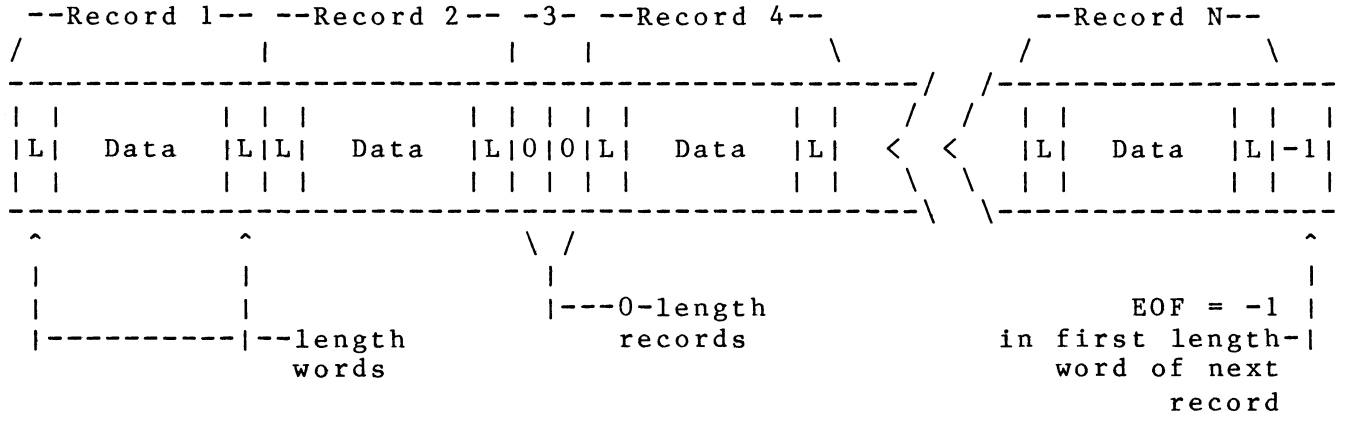
Fixed Length Formats (Types 1 and 2)



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)



READER COMMENT SHEET

RTE-IVB TECHNICAL SPECIFICATIONS
Reference Manual

92068-90013 **January 1980**

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 _____

_____

_____

FOLD FOLD

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.

FOLD FOLD

SALES & SUPPORT OFFICES

Arranged alphabetically by country

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 Australia 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*

SALES & SUPPORT OFFICES

Arranged alphabetically by country

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 Cédex**ORSAY**
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.
Jin. 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

SALES & SUPPORT OFFICES

Arranged alphabetically by country

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 OBY*
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

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

6

SALES & SUPPORT OFFICES

Arranged alphabetically by country

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

MANUAL PART NO. 92068-90013
Printed in U.S.A. January 1980

HEWLETT-PACKARD COMPANY
Data Systems Division
11000 Wolfe Road
Cupertino, California 95014