RSX-11M/M-PLUS
Executive Reference Manual

Order No. AA-L675A-TC

RSX-11M Version 4.0

RSX-11M-PLUS Version 2.0

digital equipment corporation · maynard, massachusetts

First Printing, May 1979
Revised, November 1981

DEC	DECsystem-10	PDT
DECUS	DECSYSTEM-20	RSTS
DIGITAL	DECwriter	RSX
PDP	DIBOL	VMS
UNIBUS	Edusystem	VT
VAX	IAS	digital
DECnet	MASSBUS	

ZK-2054-81

CONTENTS

CONTENTS

Page

CONTENTS

Page

FIGURES

TABLES

PREFACE

MANUAL OBJECTIVES

The *RSX-11M/M-PLUS Executive Reference Manual* describes the system directives that allow experienced MACRO-11 and FORTRAN programmers to use Executive services to control the execution and interaction of tasks.

INTENDED AUDIENCE

The intended audience for this manual are software developers who are experienced users of MACRO-11 or FORTRAN for user task generation. Information contained in this manual is intended for reference only; no attempt is made to describe the procedures involved in developing user tasks beyond the detailed reference information normally required for directive use. However, Chapters 1 through 4 do contain much information that will aid in better understanding how directives can be effectively used in the RSX-11M/M-PLUS multitasking environment. Convenient quick-reference material is included in appendixes at the end of the manual for use by the more advanced RSX-11M/M-PLUS programmer.

STRUCTURE OF THIS DOCUMENT

A Summary Of Technical Changes provides the experienced RSX-11M /RSX-11M-PLUS user with a quick summary of changes to system software since the previous version of this manual. Comments are general and serve only as a guide to areas of change.

Chapter 1 defines system directives and describes their use in both MACRO-11 and FORTRAN programs.

Chapter 2 defines significant events, event flags, system traps, and stop-bit synchronization, and describes their relationship to system directives.

Chapter 3 introduces the concept of extended logical address space within the framework of memory management directives.

Chapter 4 introduces the concept of parent/offspring tasking, including associated directives, generated data structures, and task communications.

Chapter 5 contains a short summary of all directives, arranged according to their functional categories. The summary is followed by detailed descriptions of each system directive arranged alphabetically according to macro call.

PREFACE

Appendix A contains directives arranged alphabetically according to macro call. Abbreviated specifications include directive name, FORTRAN call, macro call, and parameters only.

Appendix B lists the standard error codes returned by the RSX-11M or RSX-11M-PLUS Executive.

Appendix C lists Directive Identification Codes for all directives in the same octal values that they have in the Directive Parameter Block. A description of how the values are obtained is included.

Appendix D lists all directives, the operating systems where the individual directives are available (RSX-11S, RSX-11M, or RSX-11M-PLUS), and the SYSGEN option required (if applicable) to obtain that directive support.

ASSOCIATED DOCUMENTS

Manuals that are prerequisite sources of information for readers of this manual are: the _RSX-11M/M-PLUS Task Builder Manual_ and the _PDP-11 MACRO-11 Language Reference Manual_, the _IAS/RSX-11 FORTRAN IV User's Guide_, or the _FORTRAN-77 User's Guide_.

Other documents related to the contents of this manual are described briefly in the appropriate documentation directory supplied with the software kit.

CONVENTIONS USED IN THIS DOCUMENT

Whenever necessary, information that is applicable to a specific operating system (RSX-11M or RSX-11M-PLUS) is clearly indicated. In addition, for ease of reference, those portions of text that apply to RSX-11M-PLUS only are indicated by background shading on the printed page.

SUMMARY OF TECHNICAL CHANGES

This revision of the RSX-11M/M-PLUS Executive Reference Manual contains changes and additions to document two operating systems: RSX-11M V4.0 and RSX-11M-PLUS V2.0.

The following new directives have been added to both RSX-11M and RSX-11M-PLUS:

Request and Pass Offspring Information
Send, Request and Pass Offspring Control Block
Set System Time
Unlock Group Global Event Flags

The following directives that were formerly specific to RSX-11M-PLUS are now common to both operating systems:

Send, Request and Connect
Specify Requested Exit AST

The following new directives have been added to RSX-11M-PLUS:

Checkpoint Common Region
Map Supervisor D-Space
Move to/from Supervisor or User I- or D-Space
Read Single Event Flag
Variable Send, Request and Connect
Send Next Command

There are also four new directives that support alternate Command Line Interpreters. These CLI support directives are common to both operating systems:

Get Command for Command Line Interpreter
Get Command Interpreter Information
Set Command Line Interpreter
Specify Command Arrival AST

The description of the Spawn directive now includes an example that shows how to use parent/offspring tasking to provide AST service routines in FORTRAN. Guidelines for coding FORTRAN AST service routines are included in Chapter 1.

CHAPTER 1

USING SYSTEM DIRECTIVES

This chapter describes the use of system directives and the ways in which they are processed. Some of the Executive services described in this manual are optional RSX-11S, RSX-11M, or RSX-11M-PLUS features and may not be present in the system you are currently using. The discussion of the system directives assumes that all possible features are present in your system. See the appropriate system generation manual for a list of optional features.

1.1 INTRODUCTION

When a task requests the Executive to perform an indicated operation, this process is called a system directive. You use the directives to control the execution and interaction of tasks. If you are a MACRO-11 programmer, you usually issue directives in the form of macros defined in the system macro library. If you are a FORTRAN programmer, issue system directives in the form of calls to subroutines contained in the system object module library.

System directives enable tasks to:

- Obtain task and system information
- Measure time intervals
- Perform I/O functions
- Spawn other tasks
- Communicate and synchronize with other tasks
- Manipulate a task's logical and virtual address space
- Suspend and resume execution
- Exit

Directives are implemented by the EMT 377 instruction. EMT 0 through EMT 376 (or 375 for unmapped tasks and mapped privileged tasks) are considered to be non-RSX EMT synchronous system traps. They cause the Executive to abort the task unless the task has specified that it wants to receive control when such traps occur.

If you are a MACRO-11 programmer, use the system directive macros supplied in the system macro library for directive calls, rather than hand-coding calls to directives. Then you need only reassemble the program to incorporate any changes in the directive specifications.

Sections 1.2, 1.3, and 1.6 are intended for all users. Section 1.4 specifically describes the use of macros, while Section 1.5 describes the use of FORTRAN subroutine calls. Programmers using other supported languages should refer to the appropriate language reference manual supplied by DIGITAL.

1.2 DIRECTIVE PROCESSING

Processing a system directive involves four steps:

1. The user task issues a directive with arguments that are only used by the Executive. The directive code and parameters that the task supplies to the system are known as the Directive Parameter Block (DPB). The DPB can be either on the user task's stack or in a user task's data section.

2. The Executive receives an EMT 377 generated by the directive macro (or a DIR$ macro) or FORTRAN subroutine.

3. The Executive processes the directive.

4. The Executive returns directive status information to the task's Directive Status Word (DSW).

Note that the Executive preserves all task registers when a task issues a directive.

The user task issues an EMT 377 (generated by the directive) together with the address of a DPB or a DPB itself, on the top of the issuing task's stack. When the stack contains a DPB address, the Executive removes the address after processing the directive, and the DPB itself remains unchanged. When the stack contains the actual DPB rather than a DPB address, the Executive removes the DPB from the stack after processing the directive.

The first word of each DPB contains a Directive Identification Code (DIC) byte, and a DPB size byte. The DIC indicates which directive is to be performed; the size byte indicates the DPB length in words. The DIC is in the low-order byte of the word, and the size is in the high-order byte.

The DIC is always an odd-numbered value. This allows the Executive to determine whether the word on the top of the stack (before EMT 377 was issued) was the address of the DPB (even-numbered value) or the first word of the DPB (odd-numbered value).

The Executive normally returns control to the instruction following the EMT. Exceptions to this are directives that result in an exit from the task that issued them and an Asynchronous System Trap (AST) exit.

The Executive also clears or sets the Carry bit in the Processor Status Word (PSW) to indicate acceptance or rejection, respectively, of the directive. The DSW, addressed symbolically as $DSW[1], is set to indicate a more specific cause for acceptance or rejection of the directive. The DSW usually has a value of +1 for acceptance and a range of negative values for rejection (exceptions are success return codes for the directives CLEF$, SETF$, and GPRT$, among others). RSX-11M/M-PLUS associate DSW values with symbols, using mnemonics that report either successful completion or the cause of an error (see

1. The Task Builder resolves the address of $DSW. Users addressing the DSW with a physical address are not guaranteed compatibility with IAS and may experience incompatibilities with future RSX-11M releases.

Section 1.3). (The Instrument Society of America (ISA) FORTRAN calls CALL START and CALL WAIT are exceptions, since ISA requires positive numeric error codes. See Sections 5.3.57 and 5.3.41 for details; the specific return values are listed there with each directive.)

In the case of successful Exit directives, the Executive does not, of course, return control to the task. If an Exit directive fails, however, control is returned to the task with an error status in the DSW.

On Exit, the Executive frees task resources as follows:

- Detaches all attached devices
- Flushes the AST queue and despecifies all specified ASTs
- Flushes the receive and receive-by-reference queues
- Flushes the clock queue for outstanding Mark Time requests for the task
- Closes all open files (files open for write access are locked)
- Detaches all attached regions except in the case of a fixed task, where no detaching occurs
- Runs down the task's I/O
- Deaccesses the group global event flags for the task's group
- Disconnects from interrupts
- Flushes all outstanding CLI command buffers for the task
- Breaks the connection with any offspring tasks
- Marks for deallocation all virtual terminal units that the task has created
- Frees the task's memory if the task was not fixed

If the Executive rejects a directive, it usually does not clear or set any specified event flag. Thus, the task may wait indefinitely if it indiscriminately executes a Wait For directive corresponding to a previously issued Mark Time directive that the Executive has rejected. You should always ensure that a directive has been completed successfully.

1.3 ERROR RETURNS

As stated above, RSX-11M/M-PLUS associate the error codes with mnemonics that report the cause of the error. In the text of the manual, the mnemonics are used exclusively. The macro DRERR$, which is expanded in Appendix B, provides a correspondence between each mnemonic and its numeric value.

Appendix B also gives the meaning of each error code. In addition, each directive description in Chapter 5 contains specific, directive-related interpretations of the error codes.

1.4 USING THE DIRECTIVE MACROS

If you are programming in MACRO-11, you must decide how to create the DPB before you issue a directive. The DPB may either be created on the stack at run time (see Section 1.4.1.3, which describes the $S form, of directive) or created in a data section at assembly time (see Sections 1.4.1.1 and 1.4.1.2, which describe the $ form and $C form, respectively). If parameters vary and the code must be reentrant, the DPB must be created on the stack.

Figures 1-1 and 1-2 illustrate the alternative directives and also show the relationship between the stack pointer and the DPB.

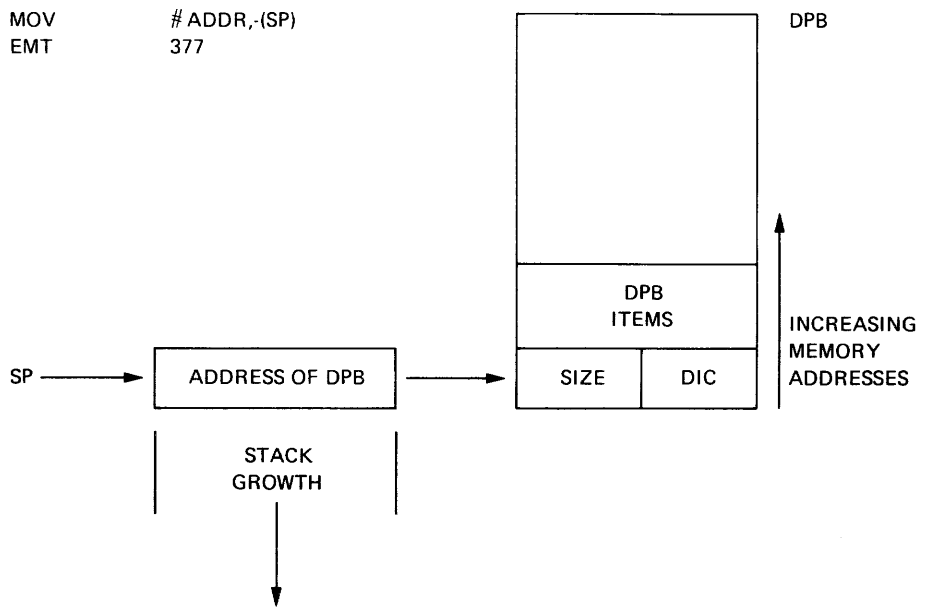


Figure 1-1 Directive Parameter Block (DPB) Pointer on the Stack

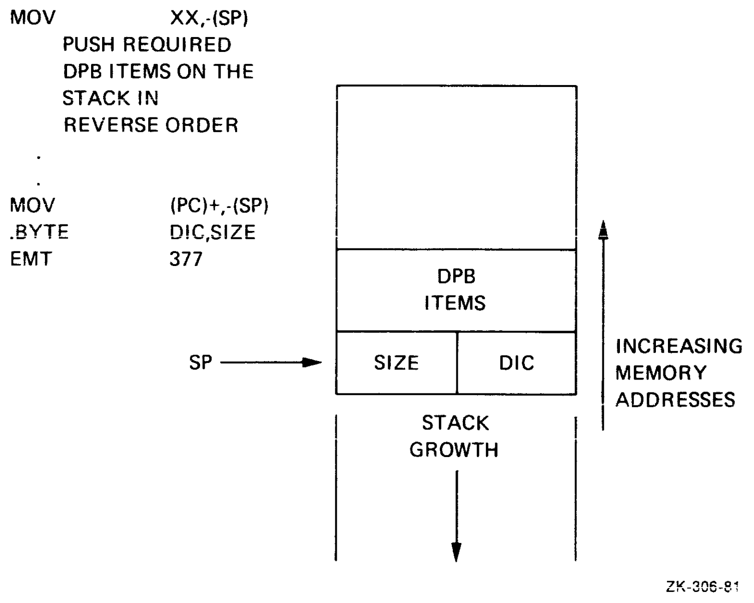


Figure 1-2 Directive Parameter Block (DPB) on the Stack

1.4.1 Macro Name Conventions

When you are programming in MACRO-11, you use system directives by including directive macro calls in your programs. The macros for the RSX-11M directives are contained in the System Macro Library (LB:[1,1]RSXMAC.SML). The .MCALL assembler directive makes these macros available to a program. The .MCALL arguments are the names of all the macros used in the program. For example:

```
;
; CALLING DIRECTIVES FROM THE SYSTEM MACRO LIBRARY
; AND ISSUING THEM.
;

    .MCALL  MRKT$S,WTSE$S
        .
        .
        .
    Additional .MCALLs or code
        .
        .
        .
    MRKT$S  #1,#1,#2,,ERR    ;MARK TIME FOR 1 SECOND
    WTSE$S  #1               ;WAIT FOR MARK TIME TO COMPLETE
        .
        .
        .
```

Macro names consist of up to four letters, followed by a dollar sign ($) and, optionally, a C or an S. The optional letter or its absence specifies which of three possible macro expansions the programmer wants to use.

1.4.1.1 **$ Form** - The $ form is useful for a directive operation that is to be issued several times from different locations in a non-reentrant program segment. The $ form is most useful when the directive is issued several times with varying parameters (one or more but not all parameters change), or in a reentrant program section when a directive is issued several times even though the DPB is not modified. This form produces only the directive's DPB, and must be issued from a data section of the program. The code for actually executing a directive that is in the $ form is produced by a special macro, DIR$ (discussed in Section 1.4.2).

Because execution of the directive is separate from the creation of the directive's DPB:

1. A $ form of a given directive needs to be issued only once (to produce its DPB).

2. A DIR$ macro associated with a given directive can be issued several times without incurring the cost of generating a DPB each time it is issued.

3. It is easy to access and change the directive's parameters by labeling the start of the DPB and using the offsets defined by the directive.

When a program issues the $ form of macro call, the parameters required for DPB construction must be valid expressions for MACRO-11 data storage instructions (such as .bYTE, .WORD, and .RAD50). You can alter individual parameters in the DPB. You might do this if you want to use the directive many times with varying parameters.

1.4.1.2 $C **Form** - Use the $C form when a directive is to be issued only once. The $C form eliminates the need to push the DPB (created at assembly time) onto the stack at run time. Other parts of the program, however, cannot access the DPB because the DPB address is unknown. (Note, in the $C form macro expansion of Section 1.4.5, that the new value of the assembler's location counter redefines the DPB address $$$ each time an additional $C directive is issued.)

The $C form generates a DPB in a separate p-section[1] called $DPB$$. The DPB is first followed by a return to the user-specified p-section, then by an instruction to push the DPB address onto the stack, and finally by an EMT 377. To ensure that the program reenters the correct p-section, you must specify the p-section name in the argument list immediately following the DPB parameters. If the argument is not specified, the program reenters the blank (unnamed) p-section.

This form also accepts an optional final argument that specifies the address of a routine to be called (by a JSR instruction) if an error occurs during the execution of the directive (see Section 1.4.2).

When a program issues the $C form of a macro call, the parameters required for DPB construction must be valid expressions for MACRO-11 data storage instructions (such as .BYTE, .WORD, and .RAD50). (This is not true for the p-section argument and the error routine argument, which are not part of the DPB.)

1.4.1.3 $S **Form** - Program segments that need to be reentrant should use the $S form. Only the $S form produces the DPB at run time. The other two forms produce the DPB at assembly time.

In this form, the macro produces code to push a DPB onto the stack, followed by an EMT 377. In this case, the parameters must be valid source operands for MOV-type instructions. For a 2-word Radix-50 name parameter, the argument must be the address of a 2-word block of memory containing the name. Note that you should not use the Stack Pointer (or any reference to the Stack Pointer) to address directive parameters when the $S form is used.[2] (In the example in Section 1.4.1, the error routine argument ERR is a target address for a JSR instruction; see Section 1.4.3.)

Note that in the $S form of the macro, the macro arguments are processed from right to left. Therefore, when using code of the form:

```
     MACRO$S,,(R4)+,(R4)+
```

the result may be obscure.

1.4.2 DIR$ **Macro**

The DIR$ macro allows you to execute a directive with a DPB predefined by the $ form of a directive macro. This macro pushes the DPB address onto the stack and issues an EMT 377 instruction.

---

1. Refer to the PDP-11 MACRO-11 Language Reference Manual for a description of p-sections (program sections).

2. Subroutine or macro calls can use the stack for temporary storage, thereby destroying the positional relationship between SP and the parameters.

The DIR$ macro generates an RSX-11M Executive trap using a predefined DPB:

Macro Call: DIR$ adr,err

adr and err are optional

adr
: The address of the DPB. (The address, if specified, must be a valid source address for a MOV instruction.) If this address is not specified, the DPB or its address must be on the stack.

err
: The address of the error return (see Section 1.4.3). If this error return is not specified, an error simply sets the carry bit in the Processor Status Word.

NOTE

DIR$ is not a $ form macro, and does not behave as one. There are no variations in the spelling of this macro.

1.4.3 Optional Error Routine Address

The $C and $S forms of macro calls and the DIR$ macro can accept an optional final argument; note that the DIR$ macro is not an Executive directive (DIR$C and DIR$S are not valid macro calls). The argument must be a valid assembler destination operand that specifies the address of a user error routine. For example, the DIR$ macro

```
     DIR$        #DPB,ERROR
```

generates the following code:

```
     MOV         #DPB,-(SP)
     EMT         377
     BCC         .+6
     JSR         PC,ERROR
```

Since the $ form of a directive macro does not generate any executable code, it does not accept an error address argument.

1.4.4 Symbolic Offsets

Most system directive macro calls generate local symbolic offsets describing the format of the DPB. The symbols are unique to each directive, and each is assigned an index value corresponding to the offset of a given DPB element.

Because the offsets are defined symbolically, you can refer to or modify DPB elements without knowing the offset values. Symbolic offsets also eliminate the need to rewrite programs if a future release of RSX-11M changes a DPB specification.

All $ and $C forms of macros that generate DBPs longer than one word generate local offsets. All informational directives (see Section 6.1.3), including the $S form, generate local symbolic offsets for the parameter block returned as well.

If the program uses either the $ or $C form and has defined the symbol $$$GLB (for example $$$GLB=0), the macro generates the symbolic offsets as global symbols and does not generate the DPB itself. The purpose of this facility is to enable the use of a DPB defined in a different module. The symbol $$$GLB has no effect on the expansion of $S macros.

When using symbolic offsets, you should use the $ form of directives.

1.4.5 Examples of Macro Calls

The examples below show the expansions of the different macro call forms.

1. The $ form generates a DPB only, in the current p-section.

   ```
   MRKT$     1,5,2,MTRAP
   ```

   generates the following code:

   ```
   .BYTE     23.,5           ; "MARK-TIME" DIC & DPB SIZE
   .WORD     1               ; EVENT FLAG NUMBER
   .WORD     5               ; TIME INTERVAL MAGNITUDE
   .WORD     2               ; TIME INTERVAL UNIT (SECONDS)
   .WORD     MTRAP           ; AST ENTRY POINT
   ```

2. The $C form generates in p-section $DPB$$ a DPB, and in the specified section the code to issue the directive.

   ```
   MRKT$C    1,5,2,MTRAP,PROG1,ERR
   ```

   generates the following code:

   ```
   .PSECT    $DPB$$
   $$$=.                     ; DEFINE TEMPORARY SYMBOL
   .BYTE     23.,5           ; "MARK-TIME" DIC & DPB SIZE
   .WORD     1               ; EVENT FLAG NUMBER
   .WORD     5               ; TIME INTERVAL MAGNITUDE
   .WORD     2               ; TIME INTERVAL UNIT (SECONDS)
   .WORD     MTRAP           ; AST ENTRY POINT ADDRESS
   .PSECT    PROG1           ; RETURN TO THE ORIGINAL PSECT
   MOV       #$$$,-(SP)      ; PUSH DPB ADDRESS ON STACK
   EMT       377             ; TRAP TO THE EXECUTIVE
   BCC       .+6             ; BRANCH ON DIRECTIVE ACCEPTANCE
   JSR       PC,ERR          ; ELSE, CALL ERROR SERVICE ROUTINE
   ```

3. The $S form generates code to push the DPB onto the stack and to issue the directive.

   ```
   MRKT$S #1,#5,#2,R2,ERR
   ```

   generates the following code:

   ```
   MOV       R2,-(SP)        ; PUSH AST ENTRY POINT
   MOV       #2,-(SP)        ; TIME INTERVAL UNIT (SECONDS)
   MOV       #5,-(SP)        ; TIME INTERVAL MAGNITUDE
   MOV       #1,-(SP)        ; EVENT FLAG NUMBER
   MOV       (PC)+,-(SP)     ; AND "MARK-TIME" DIC & DPB SIZE
   .BYTE     23.,5           ; ON THE STACK
   EMT       377             ; TRAP TO THE EXECUTIVE
   BCC       .+6             ; BRANCH ON DIRECTIVE ACCEPTANCE
   JSR       PC,ERR          ; ELSE, CALL ERROR SERVICE ROUTINE
   ```

4. The DIR$ macro issues a directive that has a predefined DPB.

```
DIR$     R1,(R3)          ; DPB ALREADY DEFINED.  DPB ADDRESS IN R1.
```

generates the following code:

```
MOV      R1,-(SP)         ; PUSH DPB ADDRESS ON STACK
EMT      377              ; TRAP TO THE EXECUTIVE
BCC      .+4              ; BRANCH ON DIRECTIVE ACCEPTANCE
JSR      PC,(R3)          ; ELSE, CALL ERROR SERVICE ROUTINE
```

1.5 FORTRAN SUBROUTINES

RSX-11M/M-PLUS provide an extensive set of FORTRAN subroutines to perform system directive operations.

The directive descriptions in Chapter 5 describe the FORTRAN subroutine calls, as well as the macro calls.

The FORTRAN subroutines fall into three basic groups:

- Subroutines based on the Instrument Society of America (ISA) Standard ISA 62.1 -- These subroutines are included in the subroutine descriptions associated with the macro calls (see Chapter 5).
- Subroutines designed to use and control specific process control interface devices supplied by DIGITAL and supported by the RSX-11M/M-PLUS operating systems.
- Subroutines for performing RSX-11M/M-PLUS system directive operations -- In general, one subroutine is available for each directive. (Exceptions are the Mark Time and Run directives. The description of Mark Time includes both CALL MARK and CALL WAIT. The description of Run includes both CALL RUN and CALL START.)

All the subroutines described in this manual can be called by FORTRAN programs compiled by either the FORTRAN IV or FORTRAN IV-PLUS compiler.

These subroutines can also be called from programs written in the MACRO-11 assembly language by using PDP-11 FORTRAN calling sequence conventions. These conventions are described in the _IAS/RSX-11 FORTRAN IV User's Guide_ and in the _FORTRAN IV-PLUS User's Guide_.

1.5.1 Subroutine Usage

All the subroutines described in this manual are added to the RSX-11M system object module library when either FORTRAN compiler is generated for RSX-11M. You call these subroutines by including the appropriate CALL statement in the FORTRAN program. When the program is linked to form a task, the Task Builder first checks to see whether each specified subroutine is user defined. If a subroutine is not user defined, the Task Builder automatically searches for it in the system object module library. If the subroutine is found, it is included in the linked task.

1.5.1.1 **Optional Arguments** - Many of the subroutines described in this manual have optional arguments. In the subroutine descriptions associated with the directives, optional arguments are designated as such by being enclosed in square brackets ([]). An argument of this kind can be omitted if the comma that immediately follows it is retained. If the argument (or string of optional arguments) is last, it can simply be omitted, and no comma need end the argument list. For example, the format of a call to SUB could be the following:

```
     CALL SUB (AA,[BB],[CC],DD[,[EE][,FF]])
```

In that event, you may omit the arguments BB, CC, EE, and FF in one of the following ways:

- CALL SUB (AA,,,DD,,)
- CALL SUB (AA,,,DD)

In some cases, a subroutine will use a default value for an unspecified optional argument. Such default values are noted in each subroutine description in Chapter 5.

1.5.1.2 **Task Names** - In FORTRAN subroutines, task names may be up to six characters long. Characters permitted in a task name are the letters A through Z, the numerals 0 through 9, and the special characters dollar sign ($) and period (.). Task names are stored as Radix-50 code, which permits up to three characters from the set above to be encoded in one PDP-11 word. (Radix-50 is described in detail in the IAS/RSX-11 FORTRAN IV User's Guide and the FORTRAN IV-PLUS User's Guide.)

FORTRAN subroutine calls require that a task name be defined as a 2-word variable or array that contains the task name as Radix-50 code. This variable may be any of the following:

- REAL
- INTEGER*4
- An INTEGER*2 array of 2 elements

This variable may be defined at program compilation time by a DATA statement, which gives the real variable an initial value (a Radix-50 constant).

For example, if a task name CCMF1 is to be used in a system directive call, the task name could be defined and used as follows:

```
     DATA CCMF1/5RCCMF1/
          .
          .
          .
     CALL REQUES (CCMF1)
```

A program may define task names during execution by using the IRAD50 subroutine or the RAD50 function as described in the IAS/RSX-11 FORTRAN IV User's Guide or the FORTRAN IV-PLUS User's Guide.

1.5.1.3 **Integer Arguments**

All the subroutines described in this manual assume that integer arguments are INTEGER*2 type arguments. Both the FORTRAN IV and FORTRAN IV-PLUS systems normally treat an integer variable as one PDP-11 storage word, provided that its value is within the range -32768 to +32767. However, if you specify the /I4 option switch when compiling a program, ensure that all integer array arguments used in these subroutines are explicitly specified as type INTEGER*2.

1.5.1.4 **GETADR Subroutine**

Some subroutine calls include an argument described as an integer array. The integer array contains some values that are the addresses of other variables or arrays. Since the FORTRAN language does not provide a means of assigning such an address as a value, you must use the GETADR subroutine described below.

Calling Sequence:

```
    CALL GETADR(ipm,[arg1],[arg2],...[argn])
```

ipm
    An array of dimension n.

arg1,...argn
    Arguments whose addresses are to be inserted in ipm. Arguments are inserted in the order specified. If a null argument is specified, then the corresponding entry in ipm is left unchanged. When the argument is an array name, the address of the first array element is inserted into ipm.

Example:

```
     DIMENSION IBUF(80),IOSB(2),IPARAM(6)
     .
     .
     .
     CALL GETADR (IPARAM(1),IBUF(1))
     IPARAM(2)=80
     CALL QIO (IREAD,LUN,IEFLAG,IOSB,IPARAM,IDSW)
     .
     .
     .
```

In this example, CALL GETADR enables you to specify a buffer address in the CALL QIO directive (see Section 5.3.44).

1.5.2 **The Subroutine Calls**

Table 1-1 is a list of the FORTRAN subroutine calls (and corresponding macro calls) associated with system directives (see Chapter 5 for detailed descriptions).

For some directives, notably Mark Time (CALL MARK), both the standard FORTRAN-IV subroutine call and the ISA standard call are provided. Other directives, however, are not available to FORTRAN tasks (for example, Specify Floating Point Exception AST [SFPA$] and Specify SST Vector Table For Task [SVTK$]).

Table 1-1
FORTRAN Subroutines and Corresponding Macro Calls

Directive	Macro Call	FORTRAN Subroutine
Abort Task	ABRT$	CALL ABORT
Alter Priority	ALTP$	CALL ALTPRI
Assign LUN	ALUN$	CALL ASNLUN
Attach Region	ATRG$	CALL ATRG
Cancel Time Based Initiation Requests	CRSQ$	CALL CANALL
Cancel Mark Time Requests	CMKT$	CALL CANMT
Checkpoint Common Region	CPCR$	CALL CPCR
Clear Event Flag	CLEF$	CALL CLREF
Connect	CNCT$	CALL CNCT
Create Address Window	CRAW$	CALL CRAW
Create Group Global Event Flags	CRGF$	CALL CRGF
Create Region	CRRG$	CALL CRRG
Create Virtual Terminal	CRVT$	CALL CRVT
Declare Significant Event	DECL$S	CALL DECLAR
Disable AST Recognition	DSAR$S	CALL DSASTR
Disable Checkpointing	DSCP$S	CALL DISCKP
Detach Region	DTRG$	CALL DTRG
Eliminate Address Window	ELAW$	CALL ELAW
Eliminate Group Global Event Flags	ELGF$	CALL ELGF
Eliminate Virtual Terminal	ELVT$	CALL ELVT
Emit Status	EMST$	CALL EMST
Enable AST Recognition	ENAR$S	CALL ENASTR
Enable Checkpointing	ENCP$S	CALL ENACKP
Exit If	EXIF$	CALL EXITIF
Exit With Status	EXST$	CALL EXST
Extend Task	EXTK$	CALL EXTTSK
Get Command for Command Interpreter	GCCI$	CALL GTCMCI

(continued on next page)

Table 1-1 (Cont.)
FORTRAN Subroutines and Corresponding Macro Calls

Directive	Macro Call	FORTRAN Subroutine
Get Command Interpreter Information	GCII$	CALL GETCII
Get LUN Information	GLUN$	CALL GETLUN
Get Mapping Context	GMCX$	CALL GMCX
Get MCR Command Line	GMCR$	CALL GETMCR
Get Partition Parameters	GPRT$	CALL GETPAR
Get Region Parameters	GREG$	CALL GETREG
Get Sense Switches	GSSW$S	CALL READSW CALL SSWTCH
Get Task Parameters	GTSK$	CALL GETTSK
Get Time Parameters	GTIM$	CALL GETTIM
Inhibit AST Recognition	IHAR$S	CALL INASTR
Map Address Window	MAP$	CALL MAP
Mark Time	MRKT$	CALL MARK CALL WAIT (ISA Standard call)
Queue I/O Request	QIO$	CALL QIO
Queue I/O Request And Wait	QIOW$	CALL WTQIO
Read All Event Flags	RDAF$	
	RDXF$	single, local, common, or group-global event flag can be read by a FORTRAN task)
Read Single Event Flag	RDEF$	CALL READEF
Receive By Reference	RREF$	CALL RREF
Receive Data	RCVD$	CALL RECEIV
Receive Data Or Exit	RCVX$	CALL RECOEX
Receive Data Or Stop	RCST$	CALL RCST
Remove Affinity	RMAF$S	CALL RMAF
Request and Pass Offspring Information	RPOI$	CALL RPOI

(continued on next page)

Table 1-1 (Cont.)
FORTRAN Subroutines and Corresponding Macro Calls

Directive	Macro Call	FORTRAN Subroutine
Request	RQST$	CALL REQUES
Resume	RSUM$	CALL RESUME
Run	RUN$	CALL RUN CALL START (ISA Standard call)
Send By Reference	SREF$	CALL SREF
Send Data	SDAT$	CALL SEND
Send Message	SMSG$	CALL SMSG
Send Next Command	SNXC$	CALL SNXC
Send, Request And Connect	SDRC$	CALL SDRC
Set Affinity	STAF$	CALL STAF
Set Command Line Interpreter	SCLI$	CALL SCLI
Send Data Request and Pass OCB	SDRP$	CALL SDRP
Set Event Flag	SETF$	CALL SETEF
Set System Time	STIM$	CALL SETTIM
Spawn	SPWN$	CALL SPAWN
Specify Power Recovery AST	SFPA$	EXTERNAL SUBNAM CALL PWRUP (SUBNAM) (to establish an AST) CALL PWRUP (to remove an AST)
Specify Requested Exit AST	SREA$ SREX$	CALL SREA CALL SREX
Stop	STOP$S	CALL STOP
Stop For Logical OR Of Event Flags	STLO$	CALL STLOR
Stop For Single Event Flag	STSE$	CALL STOPFR
Suspend	SPND$S	CALL SUSPND
Task Exit	EXIT$S	CALL EXIT
Unlock Group Global Event Flags	ULGF$S	CALL ULGF
Unmap Address Window	UMAP$	CALL UNMAP

(continued on next page)

Table 1-1 (Cont.)
FORTRAN Subroutines and Corresponding Macro Calls

Directive	Macro Call	FORTRAN Subroutine
Unstop	USTP$	CALL USTP
Variable Receive Data	VRCD$	CALL VRCD
Variable Receive Data Or Exit	VRCX$	CALL VRCX
Variable Receive Data Or Stop	VRCS$	CALL VRCS
Variable Send Data	VSDA$	CALL VSDA
Variable Send, Request and Connect	VSRC$	CALL VSRC
Wait For Single Event Flag	WTSE$	CALL WAITFR
Wait For Logical OR Of Event Flags	WTLO$	CALL WFLOR
Wait For Significant Event	WSIG$S	CALL WFSNE

NOTE

The following directives are not available as FORTRAN subroutines:

Directive	Macro Call
AST Service Exit	ASTX$S
Connect To Interrupt Vector	CINT$
Map Supervisor D-space	MSDS$
Move to/from Supervisor or User I- or D-space	MVTS$
Specify Command Arrival AST	SCAA$
Specify Floating Point Exception AST	SFPA$
Specify Parity Error AST	SPEA$
Specify Receive By Reference AST	SRRA$
Specify Receive Data AST	SRDA$
Specify SST Vector Table For Debugging Aid	SVDB$
Specify SST Vector Table For Tasks	SVTK$
Supervisor Call	SCAL$S

1.5.3 Error Conditions

Each subroutine call includes an optional argument that specifies the integer to receive the Directive Status Word (ids). When you specify this argument, the subroutine returns a value that indicates whether the directive operation succeeded or failed. If the directive failed, the value indicates the reason for the failure. The possible values are the same as those returned to the Directive Status Word (DSW) in MACRO-11 programs (see Appendix B), except for the two ISA calls, CALL WAIT and CALL START. The ISA calls have positive numeric error codes (see Sections 5.3.41 and 5.3.57).

In addition, two types of error are reported by means of the FORTRAN Object Time System (OTS) diagnostic messages. Both of these errors result in the termination of the task. The error conditions are:

- DIRECTIVE: MISSING ARGUMENT(S)
  This message indicates that at least one necessary argument was missing from a call to a system directive subroutine (OTS error number 100).

- DIRECTIVE: INVALID EVENT FLAG NUMBER
  This message indicates that an event flag number in a call to STLOR (Stop for Logical OR of Event Flags), or WFLOR (Wait For Logical OR Of Event Flags) was not in the range 1 to 96 (OTS error number 101).

1.5.4 AST Service Routines

The following FORTRAN callable routines provide support for ASTs in FORTRAN programs:

- CALL CNCT
- CALL CRVT
- CALL PWRUP
- CALL SDRC
- CALL SPAWN
- CALL SREA
- CALL SREX

Use great caution when coding an AST routine in FORTRAN. The following types of FORTRAN operations may not be performed at AST state:

- FORTRAN I/O of any kind (including ENCODE and DECODE statements and internal file I/O)

  FORTRAN I/O is not reentrant, therefore the information in the impure data area may be destroyed.

- Floating-point operations

  The floating-point processor's context is not saved while in AST state. Since the scientific subroutines use floating-point operations, they may not be called at AST state.

- Traceback information in the generated code

  Use of traceback corrupts the error recovery in the FORTRAN run time library. Any FORTRAN modules that will be called at AST state must be compiled without traceback. See the FORTRAN IV or FORTRAN-77 User's Guide for more information.

- Virtual array operations

  Use of virtual arrays at AST state remaps the current array such that any operations at non-AST state will not be executed correctly.

- Subprograms may not be shared between AST processing and normal task processing.

- EXIT or STOP statements with files open

  FORTRAN flushes the task's buffers, which could be in an intermediate state. Therefore, data might be lost if any output files are open when the EXIT or STOP is executed.

  You can EXIT or STOP at AST state if no output files are open.

  Since the message put out by STOP uses a different mechanism from the normal FORTRAN I/O routines, the act of putting out this message does not corrupt impure data in the run time system. Therefore, you can issue a STOP statement at AST state unless there are output files open.

Note also the following:

- Any execution time error at AST state will corrupt the program.

- Use extreme care if the FORTRAN task is overlayed. Both the interface routine and the actual code of the FORTRAN AST routine must be located in the root segment. Any routines that are called at AST state must also be in the root segment.

1.6 TASK STATES

Many system directives cause a task to change from one state to another. There are two basic task states in RSX-11M/M-PLUS: dormant and active. The active state has three substates: ready-to-run, blocked, and stopped.

The Executive recognizes the existence of a task only after it has been successfully installed and has an entry in the System Task Directory (STD). (Task installation is the process whereby a task is made known to the system; see the _RSX-11M/M-PLUS MCR Operations Manual_.) Once a task has been installed, it is either dormant or active. These states are defined as follows:

- Dormant -- Immediately following the processing of an Install command by the Monitor Console Routine, a task is known to the system, but is dormant. A dormant task has an entry in the STD, but no request has been made to activate it.

- Active -- A task is active from the time it is requested until the time it exits. Requesting a task means issuing the RQST$, RUN$, SPWN$, SDRC$, VSRC$, RPOI$, or SDRP$ macro, or an MCR Run command. An active task is eligible for scheduling, whereas a dormant task is not.

  The three substates of an active task are as follows:

  a. Ready-to-run -- A ready-to-run task competes with other tasks for CPU time on the basis of priority. The highest priority ready-to-run task obtains CPU time and thus becomes the current task.

  b. Blocked -- A blocked task is unable to compete for CPU time for synchronization reasons or because a needed resource is not available. Task priority effectively remains unchanged, allowing the task to compete for memory space.

  c. Stopped -- A stopped task is unable to compete for CPU time because of pending I/O completion, event flag(s) not set, or because the task stopped itself. When stopped, a task's priority effectively drops to zero and the task can be checkpointed by any other task, regardless of that task's priority. If an AST occurs for the stopped task, its normal task priority is restored only for the duration of the AST routine execution; once the AST is completed, task priority returns to zero.

1.6.1 Task State Transitions

Dormant to Active - The following commands or directives cause the Executive to activate a dormant task:

- A RUN$ directive
- A RQST$ directive
- A SPWN$ directive
- A SDRC$ directive
- A VSRC$ directive
- A RPOI$ directive
- A SDRP$ directive
- An MCR or DCL Run command

Ready-to-Run to Blocked - The following events cause an active, ready-to-run task to become blocked:

- A SPND$ directive
- An unsatisfied Wait For condition
- Checkpointing of a task out of memory by the Executive

Ready-to-Run to Stopped - The following events cause an active, ready-to-run task to become stopped:

- A STOP$S directive is executed, or an RCST$, SDRP$, GCCI$, or VRCS$ directive is issued when no data packet is available.
- An unsatisfied Stop For condition.
- An unsatisfied Wait For condition while the task has outstanding buffered I/O.[1]

Blocked to Ready-to-Run - The following events return a blocked task to the ready-to-run state:

- A RSUM$ directive issued by another task
- An MCR Resume command or a DCL Continue command
- A Wait For condition is satisfied
- The Executive reads a checkpointed task into memory

Stopped to Ready-to-Run - The following events return a stopped task to the ready-to-run state, depending upon how the task became stopped:

- A task stopped by the STOP$, RCST$, or VRCS$ directive becomes unstopped by USTP$ directive execution, or an MCR Unstop command or DCL START command.
- A Wait For condition is satisified for a task with outstanding buffered I/O.
- A task stopped for an event flag (or flags) becomes unstopped when the specified event flag (or flags) becomes (or become) set.

Active to Dormant - The following events cause an active task to become dormant:

- An EXIT$S, EXIF$, RCVX$, or VRCX$ directive, or a RREF$ or GCCI$ directive that specifies the exit option
- An ABRT$ directive
- An MCR or DCL Abort command
- A Synchronous System Trap (SST) for which a task has not specified a service routine

Blocked to Stopped - The following event causes a task that is blocked due to an unsatisfied Wait For condition to become stopped:

- The task initiates buffered I/O at AST state and then exits from AST state.

1. Only in systems that support the checkpointing of tasks during buffered I/O. An I/O request can be buffered only when the task is checkpointable and the region that I/O is being done to/from is checkpointable.

Stopped to Blocked - The following event causes a task that is stopped due to an unsatisfied Wait For condition and outstanding buffered I/O to return to a blocked state:

- Completion of all outstanding buffered I/O

1.6.2 Removing an Installed Task

You remove an installed task from the system by issuing the MCR or DCL Remove command from a privileged terminal. Refer to the RSX-11M/11M-PLUS MCR Operations Manual or the RSX-11M/M-PLUS Command Language Reference Manual.

1.7 THE GENERAL INFORMATION DIRECTIVE

Some of DIGITAL's software modules use the General Information Directive to obtain information from Executive data structures without being directly mapped to the Executive. Since this directive may change from release to release of RSX-11M-PLUS, it is specifically not documented in this manual. However, advanced users desiring to use this directive can refer to the Executive module [11,10]DRGIN.MAC and macro GIN$ in the Executive macro library. Although the directive may operate in the same manner in future releases, its operation is specifically not guaranteed, and users are cautioned accordingly.

1.8 DIRECTIVE RESTRICTIONS FOR NONPRIVILEGED TASKS

Nonprivileged tasks cannot issue certain Executive directives, except as listed below:

Directive	Macro Call	Comments
Abort Task	ABRT$	In systems that support multiuser protection, a nonprivileged task can only abort tasks with the same TI: as the task issuing the directive.
Alter Priority	ALTP$	In systems that support multiuser protection, a nonprivileged task can only alter its own priority to values less than or equal to the task's installed priority.
Cancel Time Based Initiation Requests	CSRQ$	Cannot be issued by a nonprivileged task in systems that support multiuser protection except for tasks with the same TI: as the issuing task.

Directive	Macro Call	Comments
Connect To interrupt Vector	CINT$	Cannot be issued by a nonprivileged task in mapped systems.
Set Command Line Interpreter	SCLI$	Cannot be issued by a nonprivileged task under any circumstances.

1.9 RSX-11M-PLUS

RSX-11M-PLUS supports multiprocessor PDP-11 system configurations and provides additional Executive services. Executive services include virtual terminal support, supervisor-mode library support, CPU/UNIBUS affinity, parity error AST routine support, and Executive-level dispatching.

CHAPTER 2

SIGNIFICANT EVENTS, SYSTEM TRAPS, AND STOP-BIT SYNCHRONIZATION

This chapter introduces the concept of significant events and describes the ways in which your code can make use of event flags, synchronous and asynchronous system traps, and stop-bit synchronization.

2.1 SIGNIFICANT EVENTS

A significant event is a change in system status that causes the Executive to reevaluate the eligibility of all active tasks to run (For some significant events, specifically those in which the current task becomes ineligible to run, only those tasks of lower priority are examined.) A significant event is usually caused (either directly or indirectly) by a system directive issued from within a task. Significant events include the following:

- An I/O completion
- A task exit
- The execution of a Send Data directive (see Section 5.3.61)
- The execution of a Send Data, Request and Pass OCB directive (see Section 5.3.63)
- The execution of a Send, Request, and Connect directive (see Section 5.3.62)
- The execution of a Send By Reference or a Receive By Reference directive (see Sections 5.3.74 and 5.3.55)
- The execution of an Alter Priority directive (see Section 5.3.2)
- The removal of an entry from the clock queue (for instance, resulting from the execution of a Mark Time directive or the issuance of a rescheduling request)
- The execution of a Declare Significant Event directive (see Section 5.3.16)
- The execution of the round-robin scheduling algorithm at the end of a round-robin scheduling interval
- The execution of an Exit, an Exit With Status, or an Emit Status directive

2.2 EVENT FLAGS

Event flags are a means by which tasks recognize specific events. (Tasks also use Asynchronous System Traps (ASTs) to recognize specific events. See Section 2.3.3.) In requesting a system operation (such as an I/O transfer), a task may associate an event flag with the completion of the operation. When the event occurs, the Executive sets the specified flag. Several examples later in this section describe how tasks can use event flags to coordinate task execution.

Ninety-six event flags are available to enable tasks to distinguish one event from another. Each event flag has a corresponding unique Event Flag Number (EFN) Numbers 1 through 32 form a group of flags that are unique to each task and are set or cleared as a result of that task's operation. Numbers 33 through 64 form a second group of flags that are common to all tasks, hence their name "common flags." Common flags may be set or cleared as a result of any task's operation. The last eight flags in each group, local flags (25-32) and common flags (57-64), are reserved for use by the system. Numbers 65 through 96 form the third group of flags, known as "group-global event flags." You can use these flags in any application where common event flags can be used; however, only tasks running under UICs containing the group code specified when the group-global event flags were created can use them. Four directives (Create Group Global Event Flags, Eliminate Group Global Event Flags, Unlock Group Global Event Flags, and Read Extended Event Flags) provide the Executive support for implementing group-global event flags.

Tasks can use the common or group global flags for intertask communication or their own local event flags internally. They can set, clear, and test event flags by using Set Event Flag (SETF$), Clear Event Flag (CLEF$), and Read All Event Flags (RDAF$) directives. (The Read All Event Flags directive will not return the group-global event flags. When these flags are in use, read all event flags using the Read Extended Event Flags (RDXF$) directive.)

Take great care when setting or clearing event flags, especially common and group global flags. Erroneous or multiple setting and clearing of event flags can result in obscure software faults. A typical application program can be written without explicitly accessing or modifying event flags, since many of the directives can implicitly perform these functions. The Send Data (SDAT$), Mark Time (MRKT$), and the I/O operations directives can all implicitly alter an event flag.

Examples 1 and 2 below illustrate the use of common event flags (33-64) to synchronize task execution. Examples 3 and 4 illustrate the use of local flags (1-32).

Example 1

> Task B clears common event flag 35 and then blocks itself by issuing a Wait For directive that specifies common event flag 35.
>
> Subsequently another task, Task A, specifies event flag 35 in a Set Event Flag directive to inform Task B that it may proceed. Task A then issues a Declare Significant Event directive to ensure that the Executive will schedule Task B.

Example 2

> In order to synchronize the transmission of data between Tasks A and B, Task A specifies Task B and common event flag 42 in a Send Data directive.

Task B has specified flag 42 in a Wait For directive. When Task A's Send Data directive has caused the Executive to set flag 42 and to cause a significant event, Task B proceeds and issues a Receive Data directive because its Wait For condition has been satisfied.

Example 3

A task contains a Queue I/O Request directive and an associated Wait For directive; both directives specify the same local event flag. When the task queues its I/O request, the Executive clears the local flag. If the requested I/O is incomplete when the task issues the Wait For directive, the Executive blocks the task.

When the requested I/O has been completed, the Executive sets the local flag and causes a significant event. The task then resumes its execution at the instruction that follows the Wait For directive. Using the local event flag in this manner ensures that the task does not manipulate incoming data until the transfer is complete.

Example 4

A task specifies the same local event flag in a Mark Time and an associated Wait For directive. When the Mark Time directive is issued, the Executive first clears the local flag and subsequently sets it when the indicated time interval has elapsed.

If the task issues the Wait For directive before the local flag has been set, the Executive blocks the task, which resumes when the flag is set at the end of the proper time interval. If the flag has been set first, the directive is a no-op and the task is not blocked.

Specifying an event flag does not mean that a Wait For directive must be issued. Event flag testing can be performed at any time. The purpose of a Wait For directive is to stop task execution until an indicated event occurs. Hence, it is not necessary to issue a Wait For directive immediately following a Queue I/O Request directive or a Mark Time directive.

If a task issues a Wait For directive that specifies an event flag that is already set, the blocking condition is immediately satisfied and the Executive immediately returns control to the task.

Tasks can issue Stop For directives instead of Wait For directives. When this is done, an event flag condition not satisfied will result in the task's being stopped instead of being blocked until the event flag(s) is set. A task that is blocked still competes for memory resources at its running priority. A task that is stopped competes for memory resources at priority 0.

The simplest way to test a single event flag is to issue the directive CLEF$ or SETF$. Both these directives can cause the following return codes:

IS.CLR - Flag was previously clear

IS.SET - Flag was previously set

For example, if a set common event flag indicates the completion of an operation, a task can issue the CLEF$ directive both to read the event flag and simultaneously to reset it for the next operation. If the event flag was previously clear (the current operation was incomplete), the flag remains clear.

2.3 SYSTEM TRAPS

System traps are transfers of control (also called software interrupts) that provide tasks with a means of monitoring and reacting to events. The Executive initiates system traps when certain events occur. The trap transfers control to the task associated with the event and gives the task the opportunity to service the event by entering a user-written routine.

There are two kinds of system traps:

- Synchronous System Traps (SSTs) -- SSTs detect events directly associated with the execution of program instructions. They are synchronous because they always recur at the same point in the program when trap-causing instructions occur. For example, an illegal instruction causes an SST.

- Asynchronous System Traps (ASTs) -- ASTs detect events that occur asynchronously to the task's execution. That is, the task has no direct control over the precise time that the event -- and therefore the trap -- may occur. The completion of an I/O transfer may cause an AST to occur, for example.

A task that uses the system trap facility issues system directives that establish entry points for user-written service routines. Entry points for SSTs are specified in a single table. AST entry points are set by individual directives for each kind of AST. When a trap condition occurs, the task automatically enters the appropriate routine (if its entry point has been specified).

2.3.1 Synchronous System Traps (SSTs)

SSTs can detect the execution of:

- Illegal instructions

- Instructions with invalid addresses

- Trap instructions (TRAP, EMT, IOT, BPT)

- FIS floating-point exceptions (PDP-11/40 only)

The user can set up an SST vector table, containing one entry per SST type. Each entry is the address of an SST routine that services a particular type of SST (a routine that services illegal instructions, for example). When an SST occurs, the Executive transfers control to the routine for that type of SST. If a corresponding routine is not specified in the table, the task is aborted. The SST routine enables the user to process the failure and then return to the interrupted code. Note that if a debugging aid and the user's task both have an SST vector enabled for a given condition, the debugging aid vector is referenced first to determine the service routine address.

SST routines must always be reentrant if there is a possibility that an SST can occur within the SST routine itself. Although the Executive initiates SSTs, the execution of the related service routines is indistinguishable from the task's normal execution. An AST or another SST can therefore interrupt an SST routine.

2.3.2 SST Service Routines

The Executive initiates SST service routines by pushing the task's Processor Status (PS), Program Counter (PC), and trap-specific parameters onto the task's stack. After removing the trap-specific parameters, the service routine returns control to the task by issuing an RTI or RTT instruction. Note that the task's general purpose registers R0-R5 and SP are not saved. If the SST routine makes use of them, it must save and restore them itself.

To the Executive, SST routine execution is indistinguishable from normal task execution, so that all directive services are available to an SST routine. An SST routine can remove the interrupted PS and PC from the stack and transfer control anywhere in the task; the routine does not have to return control to the point of interruption. Note that any operations performed by the routine (such as the modification of registers or the DSW, or the setting or clearing of event flags) remain in effect when the routine eventually returns control to the task.

A trap vector table within the task contains all the service routine entry points. You can specify the SST vector table by means of the Specify SST Vector Table For Task directive or the Specify SST Vector For Debugging Aid directive. The trap vector table has the following format:

		Associated	
Word	**Offest**	**Vector**	**Trap**
0	S.COAD	4	Odd or nonexistent memory error (Also, on some PDP-11 processors -- for example, PDP 11/45 -- an illegal instruction traps here rather than through word 04).
1	S.CSGF	250	Memory protect violation
2	S.CBPT	14	T-bit trap or execution of a BPT instruction
3	S.CIOT	20	Execution of an IOT instruction
4	S.CILI	10	Execution of a reserved instruction
5	S.CEMT	30	Execution of a non-RSX EMT instruction
6	S.CTRP	34	Execution of a TRAP instruction
7	S.CFLT	244	Synchronous floating-point exception (PDP-11/40 only)

A zero appearing in the table means that no entry point is specified. An odd address in the table causes an SST to occur when another SST tries to use that particular address as an entry point. If an SST occurs and an associated entry point is not specified in the table, the Executive aborts the task.

An even vector entry causes the SST routine to be executed in the same mode (either user or supervisor) that the processor was in when the SST vector was specified. An odd vector entry causes the SST routine to be executed in the other mode. For example, if the processor was in supervisor mode and the vector entry was odd, the SST routine is executed in user mode.

Depending on the reason for the SST, the task's stack may also contain additional information, as follows:

Memory protect violation (complete stack)

```
SP+10  --  PS
SP+06  --  PC
SP+04  --  Memory protect status register (SR0)[1]
SP+02  --  Virtual PC of the faulting instruction (SR2)[1]
SP+00  --  Instruction backup register (SR1)[1]
```

TRAP instruction or EMT other than 377 (and 376 in the case of unmapped tasks and mapped privileged tasks) (complete stack)

```
SP+04  --  PS
SP+02  --  PC
SP+00  --  Instruction operand (low-order byte) multiplied by 2,
           non-sign-extended
```

All items except the PS and PC must be removed from the stack before the SST service routine exits.

2.3.3 Asynchronous System Traps (ASTs)

The primary purpose of an AST is to inform the task that a certain event has occurred-for example, the completion of an I/O operation. As soon as the task has serviced the event, it can return to the interrupted code.

Some directives can specify both an event flag and an AST; with these directives, ASTs can be used as an alternative to event flags or the two can be used together. Therefore, you can specify the same AST routine for several directives, each with a different event flag. Thus, when the Executive passes control to the AST routine, the event flag can determine the action required.

AST service routines must save and restore all registers used. If the registers are not restored after an AST has occurred, the task's subsequent execution may be unpredictable.

Although not able to distinguish execution of an SST routine from task execution, the Executive is aware that a task is executing an AST routine. An AST routine can be interrupted by an SST routine, but not by another AST routine.

---

1. For details of SR0, SR1, and SR2, see the section on the memory management unit in the appropriate PDP-11 processor handbook.

The following notes describe general characteristics and uses of ASTs:

- If an AST occurs while the related task is executing, the task is interrupted so that the AST service routine can be executed.
- If an AST occurs while another AST is being processed, the Executive queues the latest AST (First-In-First-Out or FIFO). The task then processes the next AST in the queue when the current AST service is complete (unless AST recognition was disabled by the AST service routine).
- If a task is suspended or stopped when an associated AST occurs, the task remains suspended or stopped after the AST routine has been executed, unless it is explicitly resumed or unstopped either by the AST service routine itself, or by another task (the MCR and DCL Resume or UNSTOP command, for example).
- If an AST occurs while the related task is waiting (or stopped) for an event flag to be set (a Wait For (Stop For) directive), the task continues to wait after execution of the AST service routine unless the event flag is set upon AST exit.
- If an AST occurs for a checkpointed task, the Executive queues the AST (FIFO), brings the task into memory, and then activates the AST when the task returns to memory.
- The Executive allocates the necessary dynamic memory when an AST is specified. Thus, no AST condition lacks dynamic memory for data storage when it actually occurs. The AST re-uses the storage allocated for I/O and Mark Time directives. Therefore, no additional dynamic storage is required.
- Two directives, Disable AST Recognition and Enable AST Recognition, allow a program to queue ASTs for subsequent execution during critical sections of code. (A critical section might be one that accesses data bases also accessed by AST service routines, for example.) If ASTs occur while AST recognition is disabled, they are queued (FIFO) and then processed when AST recognition is enabled.

2.3.4 AST Service Routines

When an AST occurs, the Executive pushes the task's Wait For mask word, the DSW, the PS, and the PC onto the task's stack. This information saves the state of the task so that the AST service routine has access to all the available Executive services. The preserved Wait For mask word allows the AST routines to establish the conditions necessary to unblock the waiting task. Depending on the reason for the AST, the stack may also contain additional parameters. Note that the task's general purpose registers R0-R5 and SP are not saved. If the routine makes use of them, it must save and restore them itself.

On RSX-11M systems that support stop-bit synchronization or checkpointing during buffered I/O and all RSX-11M-PLUS systems, the Wait For mask word comes from the offset T.EFLM in the task's Task Control Block (TCB). On systems that do not support those features, the Wait For mask word comes from the offset H.EFLM in the task's header. Its value and the event flag range to which it corresponds depend on the last Wait For or Stop For directive issued by the task. For example, if the last such directive issued was Wait For Single Event Flag 42, the mask word has a value of 1000(8) and the event flag range is from 33 to 48. Bit 0 of the mask word represents flag 33, bit 1 represents flag 34, and so on.

The Wait For mask word is meaningless if the task has not issued any type of Wait For or Stop For directive.

Your code should not attempt to modify the Wait For mask while in the AST routine. For example, putting a zero in the Wait For mask results in an unclearable Wait For state.

After processing an AST, the task must remove the trap-dependent parameters from its stack; that is, everything from the top of the stack down to, but not including, the task's Directive Status Word. It must then issue an AST Service Exit directive with the stack set as indicated in the description of that directive (see Section 5.3.4). When the AST service routine exits, it returns control to one of two places: another AST, or the original task.

There are 13 variations on the format of the task's stack, as follows:

- If a task needs to be notified when a Floating Point Processor exception trap occurs, it issues a Specify Floating Point Processor Exception AST directive. If the task specifies this directive, an AST will occur when a Floating Point Processor exception trap occurs. The stack will contain the following values:

  ```
  SP+12  --  Event flag mask word
  SP+10  --  PS of task prior to AST
  SP+06  --  PC of task prior to AST
  SP+04  --  Task's Directive Status Word
  SP+02  --  Floating exception code
  SP+00  --  Floating exception address
  ```

  NOTE

  Refer to the appropriate processor handbook for a description of the FPU exception code values.

- If the task needs to be notified of power failure recoveries, it issues a Specify Power Recovery AST directive. An AST will then occur when the power is restored if the task is not checkpointed. The stack will contain the following values:

  ```
  SP+06  --  Event flag mask word
  SP+04  --  PS of task prior to AST
  SP+02  --  PC of task prior to AST
  SP+00  --  Task's Directive Status Word
  ```

- If a task needs to be notified when it receives either a message or a reference to a common area, it issues either a Specify Receive Data AST or a Specify Receive By Reference AST directive. An AST will occur when the message or common reference is sent to the task. The stack will contain the following values:

```
SP+06  --  Event flag mask word
SP+04  --  PS of task prior to AST
SP+02  --  PC of task prior to AST
SP+00  --  Task's Directive Status Word
```

- When a task queues an I/O request and specifies an appropriate AST service entry point, an AST will occur upon completion of the I/O request. The task's stack will contain the following values:

```
SP+10  --  Event flag mask word
SP+06  --  PS of task prior to AST
SP+04  --  PC of task prior to AST
SP+02  --  Task's Directive Status Word
SP+00  --  Address of I/O status block for I/O request (or
           zero if none was specified)
```

- When a task issues a Mark Time directive and specifies an appropriate AST service entry point, an AST will occur when the indicated time interval has elapsed. The task's stack will contain the following values:

```
SP+10  --  Event flag mask word
SP+06  --  PS of task prior to AST
SP+04  --  PC of task prior to AST
SP+02  --  Task's Directive Status Word
SP+00  --  Event flag number (or zero if none was
           specified)
```

- An offspring task, connected by a Spawn, Connect, or Send, Request And Connect directive, returns status to the connected (parent) task(s) upon exiting by the Exit AST. The parent task's stack contains the following values:

```
SP+10  --  Event flag mask word
SP+06  --  PS of task prior to AST
SP+04  --  PC of task prior to AST
SP+02  --  Task's Directive Status Word
SP+00  --  Address of exit status block
```

- If a command arrives for a CLI, the Command Arrival AST routine is entered. The stack contains:

```
SP+10  --  Event flag mask word
SP+06  --  PS of task prior to AST
SP+04  --  PC of task prior to AST
SP+02  --  Task's Directive Status Word
SP+00  --  Command buffer address
```

- If a parent task issues a Create Virtual Terminal directive, the input and output AST routines are entered. The task's stack contains the following values:

```
SP+14  --  Event flag mask word
SP+12  --  PS of task prior to AST
SP+10  --  PC of task prior to AST
SP+06  --  Task's Directive Status Word

SP+04  --  Third parameter word (Vertical Format
           Control - VFC) of the offspring request
SP+02  --  Byte count of offspring request
SP+00  --  Virtual terminal unit number (low byte); I/O
           subfunction code of offspring request (high
           byte)
```

- If the Attach/Detach AST routine is entered for a virtual terminal attach, the task's stack contains the following values:

SP+14	-- Event flag mask word
SP+12	-- PS of task prior to AST
SP+10	-- PC of task prior to AST
SP+06	-- Task's Directive Status Word
SP+04	-- Second word of offspring task name
SP+02	-- First word of offspring task name
SP+00	-- Virtual terminal unit number (low byte); I/O subfunction code of offspring request (high byte)

- If the Attach/Detach AST routine is entered for a virtual terminal detach, the task's stack contains the following values:

SP+14	-- Event flag mask word
SP+12	-- PS of task prior to AST
SP+10	-- PC of task prior to AST
SP+06	-- Task's Directive Status Word
SP+04	-- Second word of offspring task name = 0
SP+02	-- First word of offspring task name = 0
SP+00	-- Virtual terminal unit number (low byte); I/O subfunction code of offspring request (high byte)

- If a task issues a Specify Parity Error AST directive, the parity error AST service routine is entered. The task's stack contains the following values:

SP+62	-- Event flag mask word
SP+60	-- PS of task prior to AST
SP+56	-- PC of task prior to AST
SP+54	-- Task's Directive Status Word
SP+52	--
SP+50	--
SP+46	--
SP+44	--
SP+42	--
SP+40	--
SP+36	--
SP+34	-- Contents of memory parity CSRs
SP+32	-- (hardware-dependent information)
SP+30	--
SP+26	--
SP+24	--
SP+22	--
SP+20	--
SP+16	--
SP+14	--
SP+12	-- Contents of cache control register
SP+10	-- Contents of memory system error register
SP+06	-- Contents of high error address register
SP+04	-- Contents of low error address register
SP+02	-- Processor identification (single processor system=0)
SP+00	-- Number of bytes to add to SP to clean the stack (52)

- If a task becomes aborted via directive, DCL or MCR when the Specify Requested Exit AST (SREA$) is in effect, the abort AST is entered with the task's stack containing the following values:

```
SP+06  --  Event flag mask word
SP+04  --  PS of task prior to AST
SP+02  --  PC of task prior to AST
SP+00  --  Task's Directive Status Word
```

- If a task becomes aborted by directive, DCL, or MCR when the Extended Specify Requested Exit AST (SREX$) is in effect, the abort AST is entered. The task's stack contains the following values:

```
SP+12  --  Event flag mask word
SP+10  --  PS of task prior to AST
SP+06  --  PC of task prior to AST
SP+04  --  DSW of task prior to AST
SP+02  --  Trap dependent parameter
SP+00  --  Number of bytes to add to SP to clean the stack
```

- If a task issues a QIO IO.ATA function to the full-duplex terminal driver, unsolicited terminal input will cause entry into the AST service routine. Upon entry into the routine, the task's stack containing the following values:

```
SP+10  --  Event flag mask word
SP+06  --  PS of task prior to AST
SP+04  --  PC of task prior to AST
SP+02  --  Task's Directive Status Word
SP+00  --  Unsolicited character in low byte; parameter 2
           in the high byte
```

2.4 STOP-BIT SYNCHRONIZATION

Stop-bit synchronization allows tasks to be checkpointed during terminal (buffered) I/O or while waiting for an event to occur (for example, an event flag to become set or an Unstop directive to become issued). You can control synchronization between tasks by the setting of the task's Task Control Block (TCB) stop bit.

When the task's stop bit is set, the task is blocked from further execution, its priority for memory allocation effectively drops to zero, and it may be checkpointed by any other task in the system, regardless of priority. If checkpointed, the task remains out of memory until its stop bit is cleared, at which time the task becomes unstopped, its normal priority for memory allocation becomes restored, and it is considered for memory allocation based on the restored priority.

If the stopped task receives an AST, it becomes unstopped until it exits the AST routine. Memory allocation for the task during the AST routine is based on the task's priority prior to the stopped state. Note that a task cannot be stopped when an AST is in progress, but the AST routine can issue either an Unstop or Set Event Flag directive to reference the task. This causes it to remain unstopped after it issues the AST Service Exit directive.

There are three ways in which a nonprivileged task can become stopped and three corresponding ways to become unstopped. Only one method for stopping a task can be applied at a time.

- A task is stopped whenever it is in a Wait For state and has outstanding buffered I/O. A task is unstopped when the buffered I/O is completed or when the Wait For condition is satisfied.

- You can stop a task for event flag(s) by issuing the Stop For Single Event Flag directive or the Stop For Logical OR Of Event Flags directive. In this case, the task can only be unstopped by setting the specified event flag(s).

- You can stop a task by issuing a Stop; the Receive Data Or Stop directive, or the Get Command Command for Command Interpreter directive. In this case, the task can only be unstopped by issuing the Unstop directive or by the MCR or DCL Unstop command.

You cannot stop a task when an AST is in progress (AST state). Any directives that can cause a task to become stopped are illegal at the AST state.

When a task is stopped for any reason at the task state, it can still receive ASTs. If the task has been checkpointed, it becomes eligible for entrance back into memory when an AST is queued for it. The task retains its normal priority in memory while it is at the AST state or has ASTs queued. Once it has exited the AST routine with no other ASTs queued, the task is again stopped and effectively has zero priority for memory allocation.

You can use six directives for stop-bit synchronization:

- Stop - This directive stops the issuing task and cannot be issued at the AST state.

- Receive Data Or Stop and Variable Receive Data Or Stop - These directives attempt to dequeue send data packets from the specified task (or any task if none is specified). If there is no such packet to be dequeued, the issuing task is stopped. These directives cannot be issued at the AST state.

- Stop For Logical OR Of Event Flags - This directive stops the issuing task until the specified flags in the specified group of local event flags become set. If any of the specified event flags are already set, the task does not become stopped. This directive cannot be issued at the AST state.

- Stop For Single Event Flag - This directive stops the issuing task until the indicated local event flag becomes set. If the specified event flag is already set, the task does not become stopped. This directive cannot be issued at the AST state.

- Unstop - This directive unstops a task that has become stopped by the Stop or Receive Data Or Stop directive.

- Get Command for Command Interpreter - This directive stops a CLI task when there is no command queued for it. The GC.CST option must be specified to force the task to stop. See Section 5.3.30. This directive cannot be issued at the AST state.

CHAPTER 3

MEMORY MANAGEMENT DIRECTIVES

Within the framework of memory management directives, this chapter discusses the concepts of extended logical address space, regions, and virtual address windows.

3.1 ADDRESSING CAPABILITIES OF AN RSX-11M TASK

Without overlaying of tasks, an RSX-11M task cannot explicitly refer to a location with an address greater than 177777 (32K words). The 16-bit word size of the PDP-11 imposes this restriction on a task's addressing capability. Overlaying a task means that it must first be divided into segments: a single root segment, which is always in memory; and any number of overlay segments, which can be loaded into memory as required. Unless an RSX-11M task uses the memory management directives described in this chapter, the combined size of the task segments concurrently in memory cannot exceed 32K words.

When resident task segments cannot exceed a total of 32K words, a task requiring large amounts of data must access data that reside on disk. Data are disk based not only because of limited memory space but also because transmission of large amounts of data between tasks is only practical by means of disk. An overlaid task, or a task that needs to access or transfer large amounts of data, incurs a considerable amount of transfer activity over and above that caused by the task's function.

Task execution could obviously be faster if all or a greater portion of the task were resident in memory at run time. RSX-11M includes a group of memory management directives that provide the task with this capability. The directives overcome the 32K-word addressing restriction by allowing the task to dynamically change the physical locations that are referred to by a given range of addresses. With these directives, a task can increase its execution speed by reducing its disk I/O requirements at the expense of increased physical memory requirements.

On RSX-11M-PLUS systems you can effectively triple the memory available for tasks on PDP-11 systems that are capable of operating in supervisor mode through the use of supervisor-mode library routines and separate user-mode I- and D-space. Supervisor-mode library routines are instruction-only routines that are mapped into supervisor-mode I-space (up to 32K words maximum). User task parameters, stack, and any locations that may be written are mapped into supervisor-mode D-space (up to 32K words maximum). User tasks that use I- and D-space may consist of up to 32K words of instructions and 32K words of data.

3.1.1 Address Mapping

In a mapped system, the user does not need to know where a task resides in physical memory. Mapping, the process of associating task addresses with available physical memory, is transparent to the user and is accomplished by the KT11 memory management hardware. (See the appropriate PDP-11 processor handbook for a description of the KT11.) When a task references a location (virtual address), the KT11 determines the physical address in memory. The memory management directives use the KT11 to perform address mapping at a level that is visible to and controlled by the user.

3.1.2 Virtual and Logical Address Space

The three concepts defined below, physical address space, logical address space, and virtual address space, provide a basis for understanding the functions performed by the memory management directives:

- Physical Address Space -- A task's physical address space is the entire set of physical memory addresses.
- Logical Address Space -- A task's logical address space is the total amount of physical memory to which the task has access rights. This includes various areas called regions (see Section 3.3). Each region occupies a contiguous block of memory.
- Virtual Address Space -- A task's virtual address space corresponds to the 32K-word address range imposed by the PDP-11's 16-bit word length. The task can divide its virtual address space into segments called virtual address windows (see Section 3.2).

If the capabilities supplied by the RSX-11M memory management directives were not available, a task's virtual address space and logical address space would directly correspond; a single virtual address would always point to the same logical location. Both types of address space would have a maximum size of 32K words. However, the ability of the memory management directives to assign or map a range of virtual addresses (a window) to different logical areas (regions) enables the user to extend a task's logical address space beyond 32K words.

3.1.3 Supervisor-Mode Addressing

RSX-11M-PLUS supports PDP-11 processors capable of operating in supervisor mode. The supervisor mode is one of three possible modes (user, kernel, and supervisor) in which those systems can operate. In user mode, eight active page registers (APRs) are available for address mapping of user tasks. Note that only I-space APRs are employed in user mode for both instructions and data.

Supervisor-mode support doubles the instruction space available to tasks because 16 APRs (8 User-mode I-space and 8 supervisor-mode I-space) are available for address mapping. The contents of user-mode D-space APRs (I-space APRs on systems that do not support user data space) are copied into supervisor-mode D-space APRs to allow supervisor-mode routines to access user-mode data. (Refer to the appropriate PDP-11 processor handbook for a complete description of address mapping, memory management, and the various APR registers).

3.1.4 Mapping Structure of I- and D-Space Tasks

RSX-11M-PLUS supports user-mode I- and D-space. Tasks that do not use D-space execute with I- and D-space overmapped. However, these tasks may create D-space windows. This allows tasks to increase the total virtual size without a full implementation of I- and D-space.

Tasks in which the Task Builder has separated the I-space and D-space structures are mapped separately (I- and D-space are not overmapped). The overall mapping structure for these tasks is as follows:

Window 0	Root I-space.
Window 1	Task Header, stack and root D-space.
Window 2	I-space of the read-only section if a multi-user task. Memory resident overlays if not a multi-user task.
Window 3	D-space of the read-only section if a multi-user task. Memory resident overlays if not a multi-user task.
Window 4	Memory resident overlays.
.	.
.	.
.	.

The multi-user section of a multi-user task is also separated into I- and D-space areas. Memory resident libraries are not separated and are normally mapped by both I- and D-space. Common regions are also normally mapped through D-space only. However, the memory management directives can be used to attach to and map a data common with an explicit D-space window.

3.2 VIRTUAL ADDRESS WINDOWS

In order to manipulate the mapping of virtual addresses to various logical areas, you must first divide a task's 32K of virtual address space into segments. These segments are called virtual address windows. Each window encompasses a contiguous range of virtual addresses, which must begin on a 4K-word boundary (that is, the first address must be a multiple of 4K). The number of windows defined by a task can vary from 1 to 7 for RSX-11M tasks and from 1 to 23 for RSX-11M-PLUS tasks. For all tasks, window 0 is not available to the user. For tasks using I- and D-space, windows 0 and 1 are not available to the user. The size of each window can range from a minimum of 32 words to a maximum of 32K words.

A task that includes directives to manipulate address windows dynamically must have window blocks set up in its task header. The Executive uses window blocks to identify and describe each currently existing window. You specify the required number of additional window blocks (the number used for windows created by the memory management directives) to be set up by the Task Builder when linking the task (see the *RSX-11M/M-PLUS Task Builder Reference Manual*). The number of blocks that you specify should equal the maximum number of windows that will exist at any one time when the task is running.

A window's identification is a number from 0 to 15. for either user or supervisor windows on systems that support supervisor-mode libraries (0 to 23 for systems with user and supervisor I- and D-space), which is an index to the window's corresponding window block. The address window identified by 0 is the window that maps the task's header and root segment. In tasks using I- and D-space, window 0 maps the task's root instruction segment. Window 1 maps the task's header, stack, and root data segment. The Task Builder automatically creates window 0, which is mapped by the Executive and cannot be specified in any directive.

Figure 3-1 shows the virtual address space of a task divided into four address windows (windows 0, 1, 2, and 3). The shaded areas indicate portions of the address space that are not included in any window (9K to 12K and 23K to 24K). Addresses that fall within the ranges corresponding to the shaded areas cannot be used.

When a task uses memory management directives, the Executive views the relationship between the task's virtual and logical address space in terms of windows and regions. Unless a virtual address is part of an existing address window, reference to that address will cause an illegal address trap to occur. Similarly, a window can be mapped only to an area that is all or part of an existing region within the task's logical address space (see Section 3.3).

Once a task has defined the necessary windows and regions, it can issue memory management directives to perform operations such as the following:

- Map a window to all or part of a region.
- Unmap a window from one region in order to map it to another region.
- Unmap a window from one part of a region in order to map it to another part of the same region.

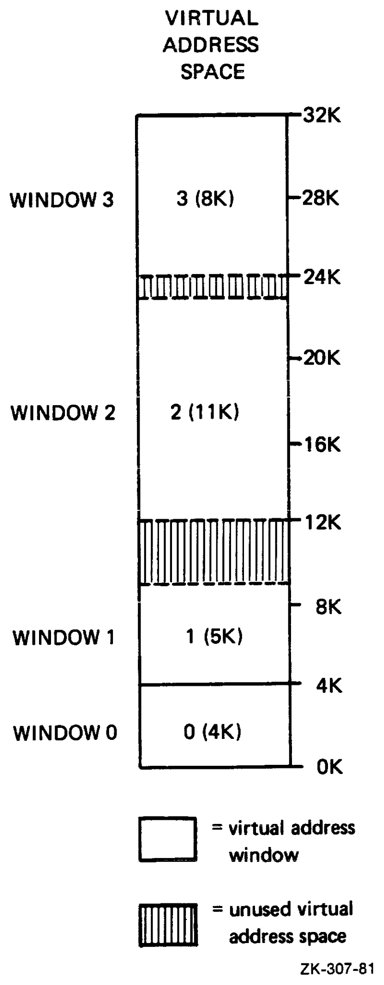


Figure 3-1 Virtual Address Windows

3.3 REGIONS

A region is a portion of a physical memory to which a task has (or potentially may have) access. The current window-to-region mapping context determines that part of a task's logical address space that the task can access at one time. A task's logical address space can consist of various types of regions:

- Task region -- A contiguous block of memory in which the task runs
- Static common region -- An area, such as a global common area, defined by an operator at run time or at system generation time

NOTE

On RSX-11M systems, static common regions occupy physical memory from the time they are created. On RSX-11M-PLUS systems, they are dynamically loaded whenever needed.

- Dynamic region -- A region created dynamically at run time by issuing the memory management directives
- Shareable region -- A read-only portion of multiuser tasks that are in shareable regions (RSX-11M-PLUS only)

Tasks refer to a region by means of a region ID returned to the task by the Executive. A region ID from 0 to 23 refers to a task's static attachment. Region ID 0 always refers to a task's task region. Region ID 1 always refers to the read-only (pure code) portion of multiuser tasks. All other region IDs are actually addresses of the attachment descriptor maintained by the Executive in the system dynamic storage area.

Figure 3-2 shows a sample collection of regions that could make up a task's logical address space at some given time. The header and root segment are always part of the task region. Since a region occupies a contiguous area of memory, each region is shown as a separate block.

Figure 3-3 illustrates a possible mapping relationship between the windows and regions shown in Figures 3-1 and 3-2.

3.3.1 Shared Regions

Address mapping not only extends a task's logical address space beyond 32K words, it also allows the space to extend to regions that have not been linked to the task at task-build time. One result is an increased potential for task interaction by means of shared regions. For example, a task can create a dynamic region to accommodate large amounts of data. Any number of tasks can then access that data by mapping to the region. Another result is the ability of tasks to use a greater number of common routines. Thus, tasks can map to required routines at run time, rather than linking to them at task-build time.

3.3.2 Attaching to Regions

Attaching is the process by which a region becomes part of a task's logical address space. A task can map only a region that is part of the task's logical address space. There are three ways to attach a task to a region:

- All tasks are automatically attached to regions that are linked to them at task-build time.
- A task can issue a directive to attach itself to a named static common region or a named dynamic region.
- A task can request the Executive to attach another specified task to any region within the logical address space of the requesting task.

Attaching identifies a task as a user of a region and prevents the system from deleting a region until all user tasks have been detached from it. (It should be noted that fixed tasks do not automatically become detached from regions upon exiting.)

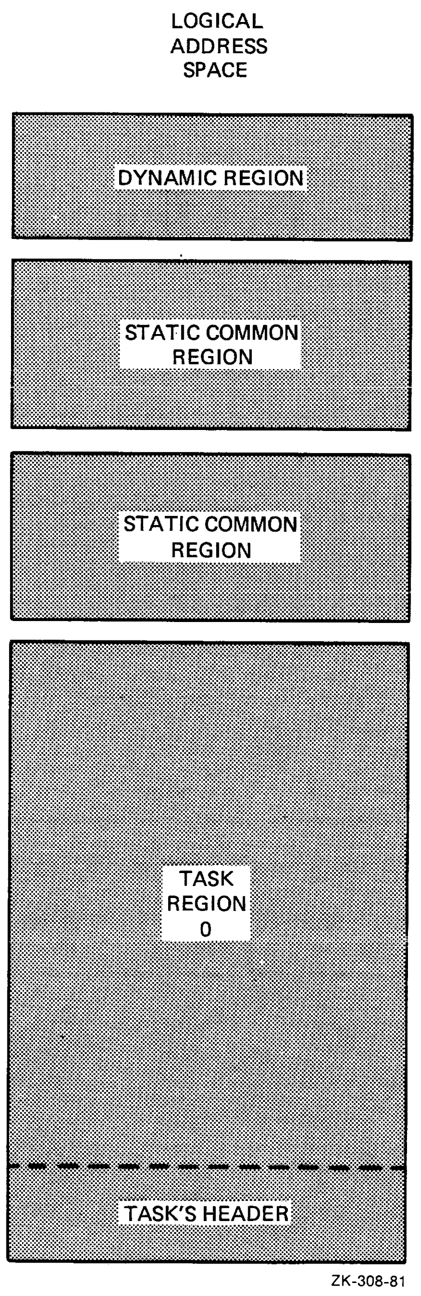


Figure 3-2 Region Definition Block

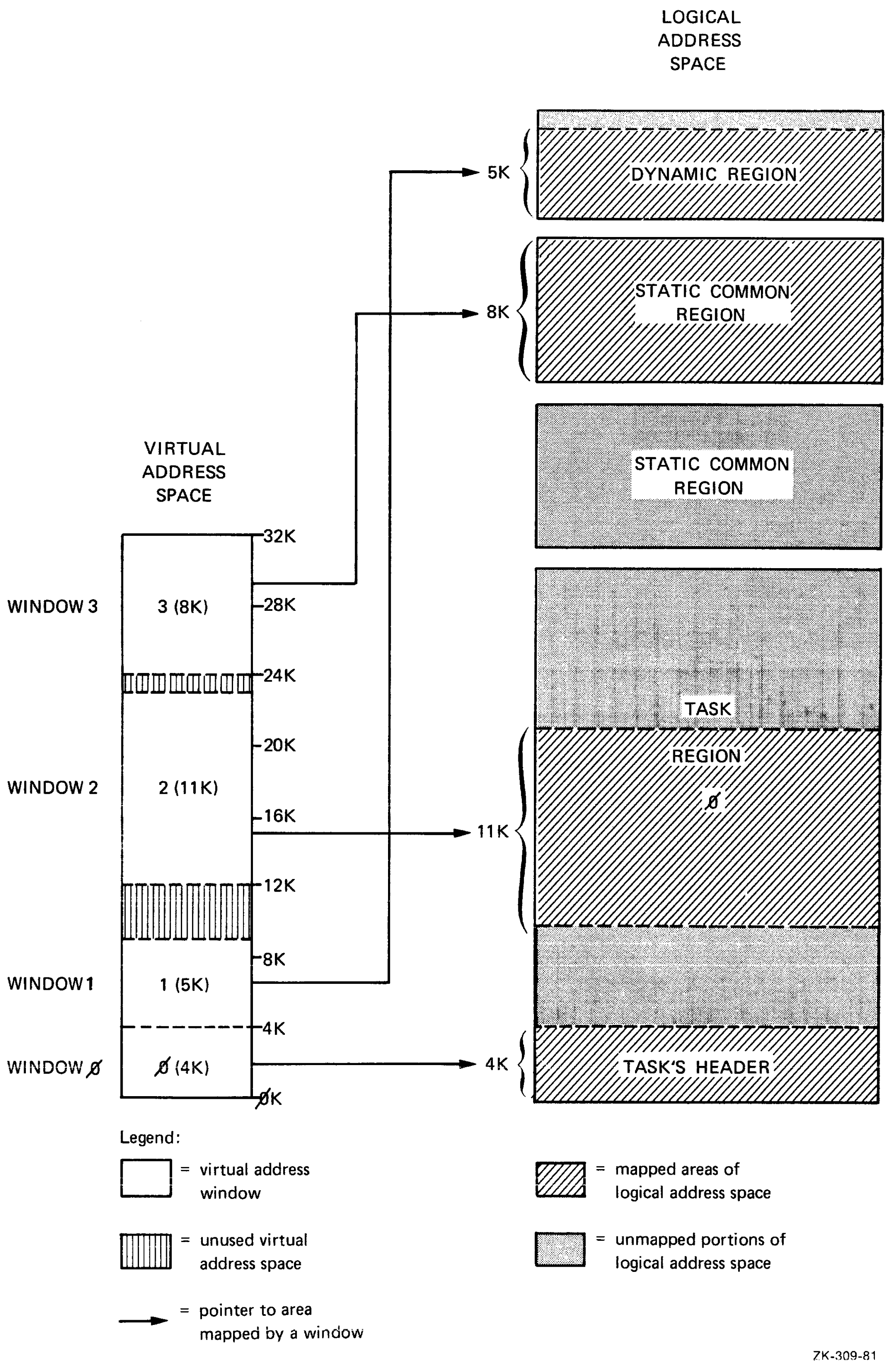


Figure 3-3 Mapping Windows to Regions

NOTE

Each Send By Reference directive issued by a sending task creates a new attachment descriptor for the receiving task. However, multiple Send By Reference directives referencing the same region require only one attachment descriptor. After the receiving task issues a series of Receive By Reference directives and all pending data requests have been received, the task should detach the region in order to return the attachment descriptors to the pool.

In RSX-11M-PLUS systems, it is possible to avoid multiple attachment descriptors when sending and receiving data by reference. Setting the WS.NAT bit in the Window Descriptor Block (see Section 3.5.2) causes the Executive to create a new attachment descriptor for that region only if necessary (that is, if the task is currently not attached to the region).

3.3.3 Region Protection

A task cannot indiscriminately attach to any region. Each region has a protection mask to prevent unauthorized access. The mask indicates the types of access (read, write, extend, delete) allowed for each category of user (system, owner, group, world). The Executive checks that the requesting task's User Identification Code (UIC) allows it to make the attempted access. The attempt fails if the protection mask denies that task the access it wants.

To determine when tasks may add to their logical address space by attaching regions, the following points must be considered. (Note that all considerations presume there is no protection violation.):

- Any task can attach to a named dynamic region, provided it knows the name. In the case of an unnamed dynamic region, a task can attach to the region only after receiving a Send By Reference directive from the task that created the region.
- Any RSX-11M-PLUS task can issue a Send By Reference directive to attach another task to any region. On RSX-11M, the task region itself may not be one of the regions involved. The reference sent includes the access rights with which the receiving task attaches to the region. The sending task can only grant access rights that it has itself.
- Any task can map to a named static common region.

3.4 DIRECTIVE SUMMARY

This section briefly describes the function of each memory management directive.

3.4.1 Create Region Directive (CRRG$)

The Create Region directive creates a dynamic region in a designated system-controlled partition and optionally attaches the issuing task to it (see Section 5.3.13).

3.4.2 Attach Region Directive (ATRG$)

The Attach Region directive attaches the issuing task to a static common region or to a named dynamic region (see Section 5.3.5).

3.4.3 Detach Region Directive (DTRG$)

The Detach Region directive detaches the issuing task from a specified region. Any of the task's address windows that are mapped to the region are automatically unmapped (see Section 5.3.19).

3.4.4 Create Address Window Directive (CRAW$)

The Create Address Window directive creates an address window, establishes its virtual address base and size, and optionally maps the window. Any other windows that overlap with the range of addresses of the new window are first unmapped and then eliminated (see Section 5.3.11).

3.4.5 Eliminate Address Window Directive (ELAW$)

The Eliminate Address Window directive eliminates an existing address window, unmapping it first if necessary (see Section 5.3.20).

3.4.6 Map Address Window Directive (MAP$)

The Map Address Window directive maps an existing window to an attached region. The mapping begins at a specified offset from the start of the region and goes to a specified length. If the window is already mapped elsewhere, the Executive unmaps it before carrying out the map assignment described in the directive (see Section 5.3.40).

3.4.7 Unmap Address Window Directive (UMAP$)

The Unmap Address Window directive unmaps a specified window. After the window has been unmapped, its virtual address range cannot be referenced until the task issues another mapping directive (see Section 5.3.85).

3.4.8 Send By Reference Directive (SREF$)

The Send By Reference directive inserts a packet containing a reference to a region into the receive queue of a specified task. The receiver task is automatically attached to the region referred to (see Section 5.3.74).

3.4.9 Receive By Reference Directive (RREF$)

The Receive By Reference directive requests the Executive first to select the next packet from the receive-by-reference queue of the issuing task, and then to make the information in the packet available to the task. Optionally the directive can map a window to the referenced region or cause the task to exit if the queue does not contain a receive-by-reference packet (see Section 5.3.55).

3.4.10 Get Mapping Context Directive (GMCX$)

The Get Mapping Context directive causes the Executive to return to the issuing task a description of the current window-to-region mapping assignments. The description is in a form that enables the user to restore the mapping context through a series of Create Address Window directives (see Section 5.3.34).

3.4.11 Get Region Parameters Directive (GREG$)

The Get Region Parameters directive causes the Executive to supply the issuing task with information about either its task region (if no region ID is given) or an explicitly specified region (see Section 5.3.36).

3.5 USER DATA STRUCTURES

Most memory management directives are individually capable of performing a number of separate actions. For example, a single Create Address Window directive can unmap and eliminate up to seven conflicting address windows, create a new window, and map the new window to a specified region. The complexity of the directives requires a special means of communication between the user task and the Executive. The communication is achieved through data structures that:

- Allow the task to specify which directive options it wants the Executive to perform

- Permit the Executive to provide the task with details about the outcome of the requested actions

There are two types of user data structures that correspond to the two key elements (regions and address windows) manipulated by the directives. The structures are called:

- The Region Definition Block (RDB)

- The Window Definition Block (WDB)

Every memory management directive, except Get Region Parameters, uses one of these structures as its communications area between the task and the Executive. Each directive issued includes in the Directive Parameter Block (DPB) a pointer to the appropriate definition block. Symbolic address offset values are assigned by the task, pointing to locations within an RDB or a WDB. The task can change the contents of these locations to define or modify the directive operation. After the Executive has carried out the specified operation, it assigns values to various locations within the block to describe the actions taken and to provide the task with information useful for subsequent operations.

3.5.1 Region Definition Block (RDB)

Figure 3-4 illustrates the format of an RDB. In addition to the symbolic offsets defined in the diagram, the region status word R.GSTS contains defined bits that may be set or cleared by the Executive or the task. (RSX-11M reserves undefined bits for future expansion.) The bits and their definitions follow.

Bit	Definition
RS.CRR=100000	Region was successfully created.
RS.UNM=40000	At least one window was unmapped on a detach.
RS.MDL=200	Mark region for deletion on last detach. When a region is created by means of a CRRG$ directive it is normally marked for deletion on last detach. However, if RS.NDL is set when the CRRG$ directive is executed, the region is not marked for deletion. Subsequent execution of a DTRG$ directive with RS.MDL set marks the region for deletion.
RS.NDL=100	Created region is not to be marked for deletion on last detach.
RS.ATT=40	Attach to created region.
RS.NEX=20	Created region is not extendable.
RS.DEL=10	Delete access desired on attach.
RS.EXT=4	Extend access desired on attach.
RS.WRT=2	Write access desired on attach.
RS.RED=1	Read access desired on attach.

These symbols are defined by the RDBDF$ macro, as described in section 3.5.1.1.

The three memory management directives that require a pointer to an RDB are:

Create Region (CRRG$)
Attach Region (ATRG$)
Detach Region (DTRG$)

When a task issues one of these directives, the Executive clears the four high-order bits in the region status word of the appropriate RDB. After completing the directive operation, the Executive sets the RS.CRR or RS.UNM bit to indicate to the task what actions were taken. The Executive never modifies the other bits.

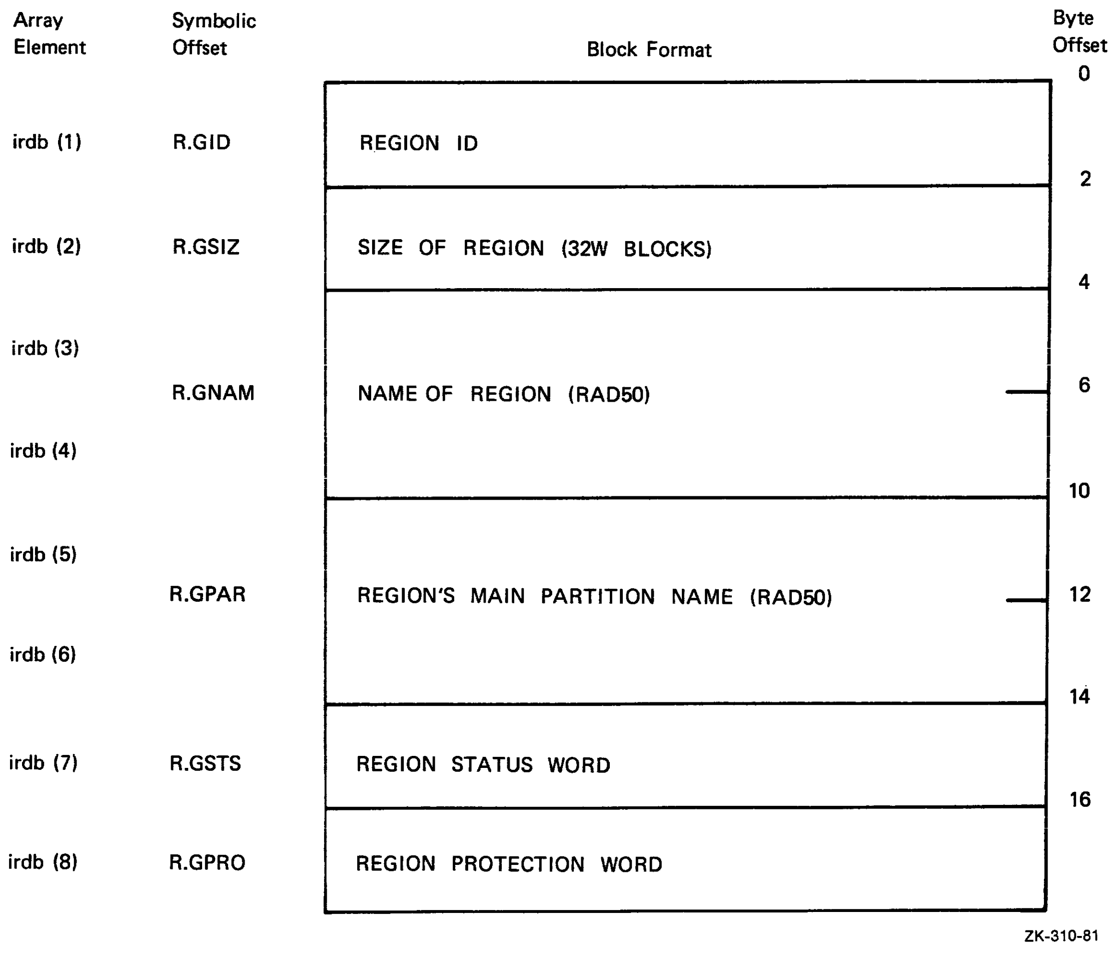


Figure 3-4 Region Definition Block

3.5.1.1 **Using Macros to Generate an RDB** - RSX-11M provides two macros, RDBDF$ and RDBBK$, to generate and define an RDB. RDBDF$ defines the offsets and status word bits for a region definition block; RDBBK$ then creates the actual region definition block. The format of RDBDF$ is:

```
RDBDF$
```

Since RDBBK$ automatically invokes RDBDF$, you need only specify RDBBK$ in a module that creates an RDB. The format of the call to RDBBK$ is:

```
RDBBK$   siz,nam,par,sts,pro
```

siz
: The region size in 32-word blocks.

nam
: The region name (RAD50).

par
: The name of the partition in which to create the region (RAD50).

sts
: Bit definitions of the region status word.

pro
: The region's default protection word.

The sts argument sets specified bits in the status word R.GSTS. The argument normally has the following format:

```
<bit1[!...!bitn]>
```

bit
: A defined bit to be set.

The argument pro is an octal number. The 16-bit binary equivalent specifies the region's default protection as follows:

Bits 15 12	11 8	7 4	3 0
WORLD	GROUP	OWNER	SYSTEM

Each of the four categories above has four bits, with each bit representing a type of access:

Bit 3	2	1	0
DELETE	EXTEND	WRITE	READ

A bit value of 0 indicates that the specified type of access is to be allowed; a bit value of 1 indicates that the specified type of access is to be denied.

The macro call

```
RDBBK$    102.,ALPHA,GEN,<RS.NDL!RS.ATT!RS.WRT!RS.RED>,167000
```

expands to:

```
.WORD     0
.WORD     102.
.RAD50    /ALPHA/
.RAD50    /GEN/
.WORD     0
.WORD     RS.NDL!RS.ATT!RS.WRT!RS.RED
.WORD     167000
```

If a Create Region directive pointed to the RDB defined by this expanded macro call, the Executive would create a region 102 (decimal) 32-word blocks in length, named ALPHA, in a partition named GEN. The defined bits specified in the sts argument tell the Executive:

- Not to mark the region for deletion on the last detach
- To attach region ALPHA to the task issuing the directive macro call
- To grant read and write access to the attached task

The protection word specified as 167000 (octal) assigns a default protection mask to the region. The octal number, which has a binary equivalent of 1110 1110 0000 0000, grants access as follows:

```
System (1110) -- All access
Owner  (1110) -- All access
Group  (0000) -- Read access only
World  (0000) -- Read access only
```

If the Create Region directive is successful, the Executive will first return to the issuing task a region ID value in the location accessed by symbolic offset R.GID, and then will set the defined bit RS.CRR in the status word R.GSTS.

3.5.1.2 Using FORTRAN to Generate an RDB

When programming in FORTRAN, you must create an 8-word, single-precision integer array as the RDB to be supplied in the subroutine calls:

CALL ATRG	(Attach Region directive)
CALL CRRG	(Create Region directive)
CALL DTRG	(Detach Region directive)

(See the PDP-11 FORTRAN Language Reference Manual for information on the creation of arrays.) An RDB array has the following format:

Word	Contents
irdb(1)	Region ID
irdb(2)	Size of the region in 32-word blocks

irdb(3)
irdb(4)
	Region name (2 words in Radix-50 format)

irdb(5)
irdb(6)
	Name of the partition that contains the region (2 words in Radix-50 format)

irdb(7)	Region status word (see the paragraph following this list)
irdb(8)	Region protection code

You can modify the region status word irdb(7) by setting or clearing the appropriate bits. See the list in Section 3.5.1 that describes the defined bits. The bit values are listed alongside the symbolic offsets.

Note that Hollerith text strings can be converted to Radix-50 values by calls to the FORTRAN library routine IRAD50 (see the appropriate FORTRAN User's Guide).

3.5.2 Window Definition Block (WDB)

Figure 3-5 illustrates the format of a WDB. The block consists of a number of symbolic address offsets to specific WDB locations. One of the locations is the window status word W.NSTS, which contains defined bits that can be set or cleared by the Executive or the task. (RSX-11M reserves all undefined bits for future expansion.) The bits and their definitions follow.

Bit	Definition
WS.CRW=100000	Address window was successfully created.
WS.UNM=40000	At least one window was unmapped by a Create Address Window, Map Address Window, or Unmap Address Window directive.
WS.ELW=20000	At least one window was eliminated in a Create Address Window or Eliminate Address Window directive.

WS.RRF=10000	Reference was successfully received.
WS.NBP=4000	Do not bypass cache for CRAW$ directives.
WS.BPS=4000	Always bypass cache for MAP$ directives.
WS.RES=2000	Map only if resident.
WS.NAT=1000	Create attachment descriptor only if necessary (for Send By Reference directives).
WS.64B=400	Defines the task's permitted alignment boundaries -- 0 for 256-word (512-byte) alignment, 1 for 32-word (64-byte) alignment.
WS.MAP=200	Window is to be mapped in a Create Address Window or Receive By Reference directive.
WS.RCX=100	Exit if no references to receive.
WS.SIS=40	Create window in supervisor I-space.
WS.UDS=20	Create this window in user-mode D-space.
WS.DEL=10	Send with delete access.
WS.EXT=4	Send with extend access.
WS.WRT=2	Send with write access. or Map with write access.
WS.RED=1	Send with read access.

Array Element	Symbolic Offset	Block Format		Byte Offset
iwdb (1)	W.NID W.NAPR	BASE APR	WINDOW ID	0
iwdb (2)	W.NBAS	VIRTUAL BASE ADDRESS (BYTES)		2
iwdb (3)	W.NSIZ	WINDOW SIZE (32W BLOCKS)		4
iwdb (4)	W.NRID	REGION ID		6
iwdb (5)	W.NOFF	OFFSET IN REGION (32W BLOCKS)		10
iwdb (6)	W.NLEN	LENGTH TO MAP (32W BLOCKS)		12
iwdb (7)	W.NSTS	WINDOW STATUS WORD		14
iwdb (8)	W.NSRB	SEND/RECEIVE BUFFER ADDRESS (BYTES)		16

ZK-311-81

Figure 3-5 Window Definition Block

These symbols are defined by the WDBDF$ macro, as described in Section 3.5.2.1.

The following directives require a pointer to a WDB:

Create Address Window (CRAW$)
Eliminate Address Window (ELAW$)
Map Address Window (MAP$)
Unmap Address Window (UMAP$)
Send By Reference (SREF$)
Receive By Reference (RREF$)

When a task issues one of these directives, the Executive clears the four high-order bits in the window status word of the appropriate WDB. After completing the directive operation, the Executive can then set any of these bits to tell the task what actions were taken. The Executive never modifies the other bits.

3.5.2.1 **Using Macros to Generate a WDB** - RSX-11M provides two macros, WDBDF$ and WDBBK$, to generate and define a WDB. WDBDF$ defines the offsets and status word bits for a window definition block; WDBBK$ then creates the actual window definition block. The format of WDBDF$ is:

```
     WDBDF$
```

Since WDBBK$ automatically invokes WDBDF$, you need only specify WDBBK$ in a module that generates a WDB. The format of the call to WDBBK$ is:

```
    WDBBK$   apr,siz,rid,off,len,sts,srb
```

apr
: A number from 0 to 7 that specifies the window's base Active Page Register (APR). The APR determines the 4K boundary on which the window is to begin. APR 0 corresponds to virtual address 0, APR 1 to 4K, APR 2 to 8K, and so on.

siz
: The size of the window in 32-word blocks.

rid
: A region ID.

off
: The offset within the region to be mapped, in 32-word blocks.

len
: The length within the region to be mapped, in 32-word blocks (defaults to the value of siz above).

sts
: The bit definitions of the window status word.

srb
: A send/receive buffer virtual address.

The argument sts sets specified bits in the status word W.NSTS. The argument normally has the following format:

```
    <bit1[!...!bitn]>
```

bit
: A defined bit to be set.

The macro call

```
    WDBBK$   5,76.,0,50.,,<WS.MAP!WS.WRT>
```

expands to:

```
    .BYTE    0,5              (Window ID returned in low-order byte)
    .WORD    0                (Base virtual address returned here)
    .WORD    76.
    .WORD    0
    .WORD    50.
    .WORD    0
    .WORD    WS.MAP!WS.WRT
    .WORD    0
```

If a Create Address Window directive pointed to the WDB defined by the macro call expanded above, the Executive would:

- Create a window 76 (decimal) blocks long beginning at APR 5 (virtual address 20K or 120000 octal)
- Map the window with write access (<WS.MAP!WS.WRT>) to the issuing task's task region (because the macro call specified 0 for the region ID)

- Start the map 50 (decimal) blocks from the base of the region, and map an area either equal to the length of the window (76 [decimal] blocks) or to the length remaining in the region, whichever is smaller (because the macro call defaulted the len argument)
- Return values to the symbolic W.NID (the window's ID) and W.NBAS (the window's virtual base address)

3.5.2.2 Using FORTRAN to Generate a WDB

3.5.2.2 **Using FORTRAN to Generate a WDB** - You must create an 8-word, single-precision integer array as the WDB to be supplied in the subroutine calls:

CALL CRAW	(Create Address Window directive)
CALL ELAW	(Eliminate Address Window directive)
CALL MAP	(Map Address Window directive)
CALL UNMAP	(Unmap Address Window directive)
CALL SREF	(Send By Reference directive)
CALL RREF	(Receive By Reference directive)

(See the PDP-11 FORTRAN Language Reference Manual for information on the creation of arrays.) A WDB array has the following format:

Word	Contents
iwdb(1)	Bits 0 to 7 contain the window ID; bits 8 to 15 contain the window's base APR.
iwdb(2)	Base virtual address of the window.
iwdb(3)	Size of the window in 32-word blocks.
iwdb(4)	Region ID.
iwdb(5)	Offset length within the region at which map begins, in 32-word blocks.
iwdb(6)	Length mapped within the region in 32-word blocks.
iwdb(7)	Window status word (see the paragraph following this list).
iwdb(8)	Address of send/receive buffer.

You can modify the window status word iwdb(7) by setting or clearing the appropriate bits. See the list in Section 3.5.2 that describes the defined bits. The bit values are listed alongside the symbolic offsets.

Note that:

- The contents of bits 8 to 15 of iwdb(1) must normally be set without destroying the value in bits 0 to 7 for any directive other than Create Address Window.
- A call to GETADR (see Section 1.5.1.4) can be used to set up the address of the send/receive buffer. For example:

  ```
  CALL GETADR(IWDB,,,,,,,,IRCVB)
  ```

  This call places the address of buffer IRCVB in array element 8. The remaining elements are unchanged. The subroutines SREF and RREF also set up this value. If you use the SREF and RREF routines, you do not need to use GETADR.

3.5.3 Assigned Values or Settings

The exact values or settings assigned to individual fields within the RDB or the WDB vary according to each directive. Fields that are not required as input can have any value when the directive is issued. Chapter 5 describes which offsets and settings are relevant for each memory management directive. The values assigned by the task are called input parameters, whereas those assigned by the Executive are called output parameters.

3.6 PRIVILEGED TASKS

When a privileged task maps to the Executive and the I/O page, the system normally dedicates five or six APRs to this mapping. A privileged task can issue memory management directives to remap any number of these APRs to regions. Take great care when using the directives in this way, because such remapping can cause obscure bugs to occur. When a directive unmaps a window that formerly mapped the Executive or the I/O page, the Executive restores the former mapping.

NOTE

> Tasks should not remap APR0. If APR0 is remapped, information such as the DSW, overlay structures, or language runtime systems will become inaccessible.

CHAPTER 4

PARENT/OFFSPRING TASKING

4.1 PARENT/OFFSPRING TASKING SUPPORT OVERVIEW

Parent/offspring tasking has many real-time applications in establishing and controlling complex interrelationships between parent and offspring tasks. A parent task is one that starts or connects to another task, called an offspring task. A major application for the parent-offspring task relationship is batch processing. When running tasks in this manner, you can set up task relationships and parameters on line to control the processing of a batch job (or jobs) that run off line.

Starting (or activating) offspring tasks is called "spawning." Spawning also includes the ability to establish task communications; a parent task can be notified when an offspring task exits and can receive status information from the offspring task.

Status returned from an offspring task to a parent task indicates successful completion of the offspring task or identifies specific error conditions.

4.2 DIRECTIVE SUMMARY

This section summarizes the directives for parent/offspring tasking and inter-task communication.

4.2.1 Parent/Offspring Tasking Directives

There are two classes of parent/offspring tasking directives:

- Spawning -- directives that create a connection between tasks
- Chaining -- directives that transfer a connection

Three directives can connect a parent task to an offspring task:

- Spawn - This directive requests activation of, and connects to, a specific offspring task.

  An offspring task spawned by a parent task has the following three task functions that are not provided by the Request or Run directive.

1. A spawned offspring task can be a command line interpreter (CLI).

2. A spawned offspring task in an RSX-11M-PLUS system can have a virtual terminal as its terminal input device (TI:).

3. A spawned offspring task can return current status information or exit status information to a connected parent task or tasks.

Spawn directive options include:

1. Queuing a command line for the offspring task (which may be a command line interpreter).

2. Establishing the offspring task's TI: as a physical terminal, or, in RSX-11M-PLUS systems, as a previously created virtual terminal unit.

3. For privileged or CLI tasks, designating any terminal as the offspring TI:

- Connect - This directive establishes task communications for synchronizing with the exit status or emit status issued by a task that is already active.

- Send, Request, and Connect - This directive sends data to the specified task, requests activation of the task if it is not already active, and connects to the task.

There are also two directives that support task chaining:

- Request and Pass Offspring Information -- This directive allows an offspring task to pass its parent connection to another task thus making the new task the offspring of the original parent. The RPOI$ directive offers all the options of the Spawn directive.

- Send Data, Request and Pass Offspring Control Block -- This directive sends a data packet for a specified task, passes its parent connection to that task, and requests it if it is not already active.

A parent task can connect to more than one offspring task using the Spawn and Connect directives, as appropriate. In addition, the parent task can use the directives in any combination to multiply connect to offspring tasks.

An offspring task can be connected to multiple parent tasks. An appropriate data structure, the Offspring Control Block, is produced (in addition to those already present) each time a parent task connects to the offspring task.

4.2.2 Task Communication Directives

Two directives in an offspring task return status to connected parent tasks:

- Exit With Status - This directive in an offspring task causes the offspring task to exit, passing status words to all connected parent tasks (one or more) that have been previously connected by a Spawn, Connect, or Send, Request, and Connect directive.

- Emit Status - This directive causes the offspring task to pass status words to either the specified connected task or all connected parent tasks if no task is explicitly specified.

When status is passed to tasks in this manner, the parent task(s) no longer remains connected.

Standard offspring task status values that can be returned to parent tasks are listed as follows:

EX$WAR	0	Warning - task succeeded, but irregularities are possible
EX$SUC	1	Success - results should be as expected
EX$ERR	2	Error - results are unlikely to be as expected
EX$SEV	4	Severe Error - one or more fatal errors detected, or task aborted

These symbols are defined in DIRSYM.MAC. They become defined locally when the EXST$ macro is invoked. However, the exit status may be any 16-bit value.

4.3 CONNECTING AND PASSING STATUS

Offspring task exit status can be returned to connected (parent) task(s) by issuing the Exit With Status directive. Offspring tasks can return status to one or more connected parent tasks at any time by issuing the Emit Status Directive. Note that only connected parent-offspring tasks can pass status.

The means by which a task connects to another task are indistinguishable once the connect process is complete. For example, Task A can become connected to Task B in one of the four ways shown below.

- Task A spawned Task B when Task B was inactive.

- Task A connected to Task B when Task B was active.

- Task A issued a Send, Request, And Connect to Task B when Task B was either active or inactive.

- Task A either spawned or connected to Task C, which then chained to Task B by means of either an RPOI$ directive or an SDRP$ directive.

Regardless of the way in which Task A became connected to Task B, Task B can pass status information back to Task A, set the event flag specified by Task A, or cause the AST specified by Task A to occur in any of the five ways shown below. Note that once offspring task status is returned to one or more parent tasks, the parent tasks become disconnected.

- Task B issues a normal (successful) exit directive. Task A receives a status of EX$SUC.
- Task B is aborted. Task A receives a severe error status of EX$SEV.
- Task B issues an Exit With Status directive, returning status to Task A upon completion of Task B.
- Task B issues an Emit Status directive specifying Task A. If Task A is multiply connected to Task B, the OCBs that contain information about these multiple connections are stored in a FIFO queue. The first OCB is used to determine which event flag, AST address, and exit status block to use.
- Task B issues an Emit Status directive to all connected tasks (no task name specified).

When a task has previously specified another task in a Spawn, Connect, or Send, Request, and Connect directive and then exits, and if status has not yet been returned, the OCB representing this connect remains queued. However, the OCB is marked to indicate that the parent task has exited. When this OCB is subsequently dequeued due to an Emit Status directive, or any type of exit, no action is taken since the parent task has exited. This procedure is followed to help a multiply-connected task to remain synchronized when parent tasks unexpectedly exit.

Examples of using directives for intertask synchronization are provided below (macro call form for directives are shown). Task A is the parent task and Task B is the offspring task.

Task A	**Task B**	**Action**
SPWN$	EXST$	Task A spawns Task B. Upon Task B completion, Task B returns status to Task A.
CNCT$	EXST$	Task A connects to active Task B. Upon Task B completion, Task B returns status to Task A.
SDRC$	RCVX$, EMST$	Task A sends data to Task B, requests Task B if it is presently not active, and connects to Task B. Task B receives the data, does some processing based on the data, returns status to Task A (possibly setting an event flag or declaring an AST), and becomes disconnected from Task A.
SDRC$, USTP$	RCST$, EMST$	Task A sends data to Task B, requests Task B if it is presently not active, connects to Task B, and unstops Task B. Task B becomes unstopped (if Task B previously could not dequeue the data packet), receives the data, does some processing based on the data, and returns status to Task A (possibly setting an event flag or declaring an AST).
SDAT$, USTP$	RCST$	Task A queues a data packet for Task B and unstops Task B; Task B receives the data.
SPWN$	RPOI$ SDRP$	Task A spawns Task B. Task B chains to Task C by issuing an RPOI$ or an SDRP$ directive. Task A is now Task C's parent. Task A is no longer connected to Task B.

4.4 SPAWNING SYSTEM TASKS

One special use of the Spawn directive is to pass a command line to a system task. You may use the Spawn directive to pass a command line to a command line interpreter, or to an installed utility.

4.4.1 Spawning a Command Line Interpreter

Command line interpreters can be broken into three classes: MCR, the CLI that is active from TI:, and all others.

- To pass a command line to MCR, use the task name MCR... .
- To pass a command line to the CLI that is currently active from TI:, use the task name CLI... . You can determine which CLI is active from your TI: by issuing the GCII$ directive.
- To pass a command to a specific CLI other than MCR or the CLI active from TI:, simply use that CLI's task name in your Spawn directive. The task name of DCL is ...DCL. Check with your system manager for the task names of any user-written CLI's.
- On RSX-11M systems, you may pass a command to a specific CLI only if the specified task name is not already active. If the task name is already active, the Spawn directive will fail.

4.4.2 Spawning a Utility

Utilities are generally installed under task names of the form ...tsk. This convention allows the utilities to be invoked as MCR commands. (See the RSX-11M/M-PLUS MCR Operations Manual.)

You can pass commands to a utility in one of two ways. You can spawn the utility directly, using the task name ...tsk. Or, you can spawn MCR and pass it a command line that begins with three-character task name.

4.4.2.1 Spawning a Utility under RSX-11M - If you attempt to spawn ...tsk directly on an RSX-11M system, the operation behaves as follows:

- If that task is not yet active, the executive will activate it, under the name ...tsk.
- If the task ...tsk is already active, your Spawn directive will fail, regardless of which terminal has activated that task.

If you pass MCR a command line beginning with "tsk", MCR will:

- Attempt to activate the task ...tsk.
- If that task name is already active, MCR will attempt to activate the task under the name tskTnn, where nn is the unit number of your TI:.
- If both ...tsk and tskTnn are already active, MCR will report failure to your task.

Unless you are certain that the utility you desire is not yet active in the system, direct spawning of the utility offers a greater likelihood of failure than requesting the utility through MCR. For this reason, it is recommended that on RSX-11M systems you request utilities through MCR.

4.4.2.2 **Spawning a Utility under RSX-11M-PLUS** - On RSX-11M-PLUS systems, whenever, you spawn a task using a name of the form ...tsk, the executive activates the task as tskTnn. This is the same renaming procedure used by MCR under RSX-11M-PLUS.

On RSX-11M-PLUS systems, invoking a utility through MCR offers no advantage over spawning the utility directly. Further, invoking the utility through MCR incurs extra system overhead. Therefore it is recommended that you spawn utilities directly on RSX-11M-PLUS systems.

4.4.2.3 **Passing Command Lines to Utilities** - Even when you spawn a utility directly, pass a command line that includes the entire command as you would type it at the terminal or pass it to MCR. Include the 3-character task name followed by a space. This maintains compatibility with the format used by MCR to pass commands to utilities. (See the description of the GMCR$ directive in Chapter 5.)

CHAPTER 5

DIRECTIVE DESCRIPTIONS

Each directive description consists of an explanation of the directive's function and use, the names of the corresponding macro and FORTRAN calls, the associated parameters, and possible return values of the Directive Status Word (DSW). The descriptions generally show the $ form of the macro call (for instance, QIO$), although the $C and $S forms are also available. Where the $S form of a macro requires less space and performs as fast as a DIR$ (because of a small DPB), it is recommended. For these macros, the expansion for the $S form is shown, rather than that for the $ form.

In addition to the directive macros themselves, you can use the DIR$ macro to execute a directive if the directive has a predefined DPB. See Sections 1.4.1.1 and 1.4.2 for further details.

5.1 DIRECTIVE CATEGORIES

For ease of reference, the directive descriptions are presented alphabetically in Section 5.3 according to the directive macro calls. This section, however, groups the directives by function. The directives are grouped into the following ten categories:

- Task execution control directives
- Task status control directives
- Informational directives
- Event-associated directives
- Trap-associated directives
- I/O- and intertask communications-related directives
- Memory management directives
- Parent/offspring tasking directives
- RSX-11M-PLUS directives
- Command line interpreter (CLI) support directives

5.1.1 Task Execution Control Directives

The task execution control directives deal principally with starting and stopping tasks. Each of these directives (except Extend Task) results in a change of the task's state (unless the task is already in the state being requested). These directives are:

Macro	Directive Name
ABRT$	Abort Task
CSRQ$	Cancel Time Based Initiation Requests
EXIT$S	Task Exit ($S form recommended)
EXTK$	Extend Task
RQST$	Request Task
RSUM$	Resume Task
RUN$	Run Task
SPND$S	Suspend ($S form recommended)

5.1.2 Task Status Control Directives

Two task status control directives alter the checkpointable attribute of a task. A third directive changes the running priority of an active task. These directives are:

Macro	Directive Name
ALTP$	Alter Priority
DSCP$S	Disable Checkpointing ($S form recommended)
ENCP$S	Enable Checkpointing ($S form recommended)

5.1.3 Informational Directives

Several directives provide the issuing task with system information and parameters such as: the time of day, the task parameters, the console switch settings, and partition or region parameters. These directives are:

Macro	Directive Name
GPRT$	Get Partition Parameters
GREG$	Get Region Parameters
GSSW$S	Get Sense Switches ($S form recommended)
GTIM$	Get Time Parameters
GTSK$	Get Task Parameters

5.1.4 Event-Associated Directives

The event and event flag directives provide inter- and intratask synchronization and signaling and the means to set the system time. You must use these directives carefully since software faults resulting from erroneous signaling and synchronization are often obscure and difficult to isolate. The directives are:

Macro	Directive Name
CLEF$	Clear Event Flag
CMKT$	Cancel Mark Time Requests
CRGF$	Create Group Global Event Flags
DECL$S	Declare Significant Event ($S form recommended)
ELGF$	Eliminate Group Global Event Flags
EXIF$	Exit If
MRKT$	Mark Time
RDAF$	Read All Event Flags
RDXF$	Read Extended Event Flags
SETF$	Set Event Flag
STIM$	Set System Time

Macro	Directive Name
STLO$	Stop For Logical 'OR' of Event Flags
STOP$S	Stop ($S form recommended)
STSE$	Stop For Single Event Flag
ULGF$S	Unlock Group Global Event Flags ($S form recommended)
USTP$	Unstop
WSIG$S	Wait For Significant Event ($S form recommended)
WTLO$	Wait For Logical OR Of Event Flags
WTSE$	Wait For Single Event Flag

5.1.5 Trap-Associated Directives

The trap-associated directives provide trap facilities that allow transfer of control (software interrupts) to the executing tasks. These directives are:

Macro	Directive Name
ASTX$S	AST Service Exit ($S form recommended)
DSAR$S	Disable AST Recognition ($S form recommended)
ENAR$S	Enable AST Recognition ($S form recommended)
IHAR$S	Inhibit AST Recognition ($S form recommended)
SCAA$	Specify Command Arrival AST
SFPA$	Specify Floating Point Processor Exception AST
SPRA$	Specify Power Recovery AST
SRDA$	Specify Receive Data AST
SREA$	Specify Requested Exit AST
SREX$	Specify Requested Exit AST (extended)
SRRA$	Specify Receive-By-Reference AST
SVDB$	Specify SST Vector Table For Debugging Aid
SVTK$	Specify SST Vector Table For Task

5.1.6 I/O- and Intertask Communications-Related Directives

The I/O- and intertask communications-related directives allow tasks to access I/O devices at the driver interface level or interrupt level, to communicate with other tasks in the system, and to retrieve the MCR command line used to start the task. These directives are:

Macro	Directive Name
ALUN$	Assign LUN
CINT$	Connect To Interrupt Vector
GLUN$	Get LUN Information
GMCR$	Get MCR Command Line
QIO$	Queue I/O Request
QIOW$	Queue I/O Request And Wait
RCVD$	Receive Data
RCVX$	Receive Data Or Exit
SDAT$	Send Data
SMSG$	Send Message

5.1.7 Memory Management Directives

The memory management directives allow a task to manipulate its virtual and logical address space, and to set up and control dynamically the window-to-region mapping assignments. The directives also provide the means by which tasks can share and pass references to data and routines. These directives are:

Macro	Directive Name
ATRG$	Attach Region
CRAW$	Create Address Window
CRRG$	Create Region
DTRG$	Detach Region
ELAW$	Eliminate Address Window
GMCX$	Get Mapping Context
MAP$	Map Address Window
RREF$	Receive By Reference
SREF$	Send By Reference
UMAP$	Unmap Address Window

5.1.8 Parent/Offspring Tasking Directives

Parent/offspring tasking directives permit tasks to start other tasks, and to connect to other tasks in order to receive status information. These directives are:

Macro	Directive Name
CNCT$	Connect
EMST$	Emit Status
EXST$	Exit With Status
RPOI$	Request and Pass Offspring Information
SDRC$	Send, Request And Connect
SDRP$	Send Data, Request and Pass OCB
SPWN$	Spawn

5.1.9 RSX-11M-PLUS Directives

In addition to the directives just listed, RSX-11M-PLUS includes directives that support virtual terminals, CPU/UNIBUS affinity, supervisor-mode library routines, variable-length send/receive data buffers, and parity error AST routine support. These directives are:

Macro	Directive Name
CPCR$	Checkpoint Common Region
CRVT$	Create Virtual Terminal
ELVT$	Eliminate Virtual Terminal
MSDS$	Map Supervisor D-Space
MVTS$	Move to/from I/D-Space
RDEF$	Read Single Event Flag
RMAF$S	Remove Affinity ($S form only)
SCAL$S	Supervisor Call ($S form only)
SPEA$	Specify Parity Error AST
SNXC$	Send Next Command
STAF$	Set Affinity
VRCD$	Variable Receive Data
VRCS$	Variable Receive Data Or Stop
VRCX$	Variable Receive Data Or Exit
VSRC$	Variable Send, Request, and Connect
VSDA$	Variable Send Data

These functions provide for the dispatching of multiuser tasks and can enhance the interface to slave tasks.

The dispatching algorithm used by the Executive is identical to the algorithm used by MCR. Thus, the ability to dispatch copies of multiuser tasks is available at both the MCR command and Executive directive level. A consistent scheme for communication and synchronization between multiuser tasks is made available at the Executive level.

Executive-level dispatching uses the same naming scheme as is used in the RSX-11M-PLUS MCR dispatching algorithm. A single copy of the multiuser task must be installed with a name of the form ...mmm. When a task issues a directive specifying a task name of the form ...mmm, the Executive first forms the task name mmmtnn, where t is the first character of the device name of the TI: of the issuing task, and nn is the unit number. The Executive then attempts to perform the directive as if the task name mmmtnn has been specified. If the directive is one that could activate the task (Request, Spawn, or Send, Request And Connect), a TCB may be dynamically created and filled in from the ...mmm TCB. If the directive is a send user-type directive, and the TCB mmmtnn does not exist, the send packet is queued to the ...mmm TCB until mmmtnn is activated. At that time any send packets for mmmtnn that are queued to the ...mmm TCB are moved to the mmmtnn TCB.

This allows for the specification of a specific copy of a multi-user task in a directive whose TI: is different from that of the issuing task. If the TI: of the target task is known, the task's name can be calculated and explicitly specified in a directive.

5.1.10 CLI Support Directives

The CLI support directives allow CLI tasks to get command lines, request and pass offspring information, get command interpreter information, and set a specified CLI for a terminal. These directives are:

Macro	Directive Name
GCCI$	Get Command for Command Interpreter
GCII$	Get Command Interpreter Information
SCLI$	Set Command Line Interpreter

5.2 DIRECTIVE CONVENTIONS

The following are conventions for using system directives:

1. In MACRO-11 programs, unless a number is followed by a decimal point (.), the system assumes the number to be octal.

   In FORTRAN programs, use INTEGER*2 type unless the directive description states otherwise.

2. In MACRO-11 programs, task and partition names can be from one to six characters long and should be represented as two words in Radix-50 form.

   In FORTRAN programs, specify task and partition names by a variable of type REAL (single precision) that contains the task or partition name in Radix-50 form. To establish Radix-50 representation, either use the DATA statement at compile time, or use the IRAD50 subprogram or RAD50 function at run time.

3. Device names are two characters long and are represented by one word of ASCII code.

4. Some directive descriptions state that a certain parameter must be provided even though the system ignores it. Such parameters are included to maintain compatibility between RSX-11M, RSX-11M-PLUS, and IAS.

5. In the directive descriptions, square brackets ([ ]) enclose optional parameters or arguments. To omit optional items, either use an empty (null) field in the parameter list or omit a trailing optional parameter.

6. Logical Unit Numbers (LUNs) can range from 1 to 255(10).

7. Event flag numbers range from 1 to 96(10). Numbers from 1 to 32(10) denote local flags. Numbers from 33 to 64 denote common flags. Numbers 65 to 96 denote group-global event flags.

Note that the Executive preserves all task registers when a task issues a directive.

5.3 SYSTEM DIRECTIVE DESCRIPTIONS

Each directive description includes most or all of the following elements:

Name:

This describes the function of the directive.

FORTRAN Call:

This shows the FORTRAN subroutine call, and defines each parameter.

Macro Call:

This shows the macro call, defines each parameter, and gives the defaults for optional parameters in parentheses following the definition of the parameter. Since zero is supplied for most defaulted parameters, only nonzero default values are shown. Parameters ignored by RSX-11M/M-PLUS are required for compatibility with IAS.

Macro Expansion:

Most of the directive descriptions expand the $S form of the macro. Where the $S form is recommended for a directive, the expansion for that form is shown instead. Section 1.4.5 illustrates expansions for all three forms and for the DIR$ macro.

Definition Block Parameters:

Only the memory management directive descriptions include these parameters. This section describes all the relevant input and output parameters in the Region or Window Definition Block (see Section 3.5).

Local Symbol Definitions:

Macro expansions usually generate local symbol definitions with an assigned value equal to the byte offset from the start of the DPB to the corresponding DPB element. This section lists these symbols. The length in bytes of the element pointed to by the symbol appears in parentheses following the symbol's description. Thus:

```
A.BTTN  --  Task name (4)
```

defines A.BTTN as pointing to a task name in the Abort Task DPB; the task name has a length of four bytes.

DSW Return Code:

This section lists all valid return codes.

Notes:

The notes presented with some directive descriptions expand on the function, use, and/or consequences of using the directives. Always read the notes carefully.

ABRT$

5.3.1 Abort Task

The Abort Task directive instructs the system to terminate the execution of the indicated task. ABRT$ is intended for use as an emergency or fault exit. ABRT$ displays a termination notification based on the described condition, at one of the following terminals:

1. The terminal from which the aborted task was requested
2. The originating terminal of the task that requested the aborted task
3. The operator's console (CO:) if the task was started internally from another task by a Run directive, or by an MCR or DCL Run command that specified one or more time parameters

On systems without multiuser protection, a task may abort any task, including itself. When a task is aborted, its state changes from active to dormant. Therefore, to reactivate an aborted task, a task or an operator must request it.

In systems that support multiuser protection, a task must be privileged to issue the Abort Task directive (unless it is aborting a task with the same TI:).

FORTRAN Call:

```
     CALL ABORT (tsk[,ids])

       tsk  =  Name of the task to be aborted (RAD50)

       ids  =  Directive status
```

Macro Call:

```
     ABRT$   tsk

       tsk  =  Name of the task to be aborted (RAD50)
```

Macro Expansion:

```
     ABRT$   ALPHA
     .BYTE   83.,3          ;ABRT$ MACRO DIC, DPB SIZE=3 WORDS
     .RAD50  /ALPHA/        ;TASK "ALPHA"
```

Local Symbol Definitions:

A.BTTN	--	Task name (4)

DSW Return Codes:

IS.SUC	--	Successful completion
IE.INS	--	Task not installed
IE.ACT	--	Task not active

IE.PRI	--	Issuing task is not privileged (multiuser protection systems only)
IE.ADP	--	Part of the DPB is out of the issuing task's address space
IE.SDP	--	DIC or DPB size is invalid

Notes:

1. When a task is aborted, the Executive frees all the task's resources. In particular, the Executive:

   - Detaches all attached devices.
   - Flushes the AST queue and despecifies all specified ASTs.
   - Flushes the receive and receive-by-reference queue.
   - Flushes the clock queue for outstanding Mark Time requests for the task.
   - Closes all open files (files open for write access are locked).
   - Detaches all attached regions except in the case of a fixed task, where no detaching occurs.
   - Runs down the task's I/O.
   - Deaccesses the group global event flags for the task's group.
   - Disconnects from interrupts.
   - Flushes all outstanding CLI command buffers for the task.
   - Breaks the connection with any offspring tasks.
   - Returns a severe error status (EX$SEV) to the parent task when a connected task is aborted.
   - Marks virtual terminals created by the aborted task for deallocation. The virtual terminals actually become deallocated when all tasks using the virtual terminal(s) are aborted or exit (see Section 4.2). Nonprivileged tasks using virtual terminal units that are marked for deallocation as TI: are also aborted.
   - Frees the task's memory if the aborted task was not fixed.

2. If the aborted task had a requested exit AST specified, the task will receive that AST instead of being aborted. No indication that this has occurred is returned to the task that issued the abort request.

3. When the aborted task actually exits, the Executive declares a significant event.

ALTP$

5.3.2 Alter Priority

The Alter Priority directive instructs the system to change the running priority of a specified active task to either a new priority indicated in the directive call, or the task's default (installed) priority if the call does not specify a new priority.

The specified task must be installed and active. The Executive resets the task's priority to its installed priority when the task exits.

If the directive call omits a task name, the Executive defaults to the issuing task.

The Executive reorders any outstanding I/O requests for the task in the I/O queue and reallocates the task's partition. The partition reallocation may cause the task to be checkpointed.

In systems that support multiuser protection, a non-privileged task can issue ALTP$ only for itself, and only for a priority equal to or lower than its installed priority. A privileged task can change the priority of any task to any value less than 250.

FORTRAN Call:

```
     CALL ALTPRI ([tsk],[ipri][,ids])

       tsk   =  Active task name

       ipri  =  A 1-word integer value equal to the new priority,
                a number from 1 to 250 (decimal)

       ids   =  Directive status
```

Macro Call:

```
     ALTP$  [tsk][,pri]

       tsk  =  Active task name
       pri  =  New priority, a number from 1 to 250 (decimal)
```

Macro Expansion:

```
     ALTP$     ALPHA, 75.
     .BYTE     9.,4         ;ALTP$ MACRO DIC, DPB SIZE=4 WORDS
     .RAD50    /ALPHA/      ;TASK ALPHA
     .WORD     75.          ;NEW PRIORITY
```

Local Symbol Definitions:

```
     A.LTTN  --  Task name (4)

     A.LTPR  --  Priority (2)
```

DSW Return Codes:

IS.SUC	--	Successful completion.
IE.INS	--	Task not installed.
IE.ACT	--	Task not active.
IE.PRI	--	Issuing task is not privileged (multiuser protection systems only).
IE.IPR	--	Invalid priority.
IE.ADP	--	Part of the DPB is out of the issuing task's address space.
IE.SDP	--	DIC or DPB size is invalid.

ALUN$

5.3.3 Assign LUN

The Assign LUN directive instructs the system to assign a physical device unit to a logical unit number (LUN). It does not indicate that the task has attached itself to the device.

The actual physical device assigned to the logical unit is dependent on the logical assignment table (see the Assign command in the _RSX-11M/M-PLUS MCR Operations Manual_ or the _RSX-11M/M-PLUS Command Language Reference Manual_). The Executive first searches the logical assignment table for a device name match. If it finds a match, the Executive assigns the physical device unit associated with the matching entry to the logical unit. Otherwise, the Executive searches the physical device tables and assigns the actual physical device unit named to the logical unit. In systems that support multiuser protection, the Executive does not search the logical assignment table if the task has been installed with the slave option (/SLV=YES).

When a task reassigns a LUN from one device to another, the Executive cancels all I/O requests for the issuing task in the previous device queue.

FORTRAN Call:

```
     CALL ASNLUN (lun,dev,unt[,ids])

       lun  =  Logical unit number

       dev  =  Device name (format:  1A2)

       unt  =  Device unit number

       ids  =  Directive status
```

Macro Call:

```
     ALUN$ lun,dev,unt

       lun  =  Logical unit number

       dev  =  Device name (two characters)

       unt  =  Device unit number
```

Macro Expansion:

```
     ALUN$     7,TT,0          ;ASSIGN LOGICAL UNIT NUMBER
     .BYTE     7,4             ;ALUN$ MACRO DIC, DPB SIZE=4 WORDS
     .WORD     7               ;LOGICAL UNIT NUMBER 7
     .ASCII    /TT/            ;DEVICE NAME IS TT (TERMINAL)
     .WORD     0               ;DEVICE UNIT NUMBER=0
```

Local Symbol Definitions:

```
     A.LULU  --  Logical unit number (2)

     A.LUNA  --  Physical device name (2)

     A.LUNU  --  Physical device unit number (2)
```

DSW Return Codes:

IS.SUC	-- Successful completion.
IE.LNL	-- LUN usage is interlocked (see Note 1 below).
IE.IDU	-- Invalid device and/or unit.
IE.ILU	-- Invalid logical unit number.
IE.ADP	-- Part of the DPB is out of the issuing task's address space.
IE.SDP	-- DIC or DPB size is invalid.

Notes:

1. A return code of IE.LNL indicates that the specified LUN cannot be assigned as directed. Either the LUN is already assigned to a device with a file open for that LUN, or the LUN is currently assigned to a device attached to the task, and the directive attempted to change the LUN assignment. If a task has a LUN assigned to a device and the task has attached the device, the LUN can be reassigned, provided that the task has another LUN assigned to the same device.

2. In RSX-11M-PLUS systems, physical I/O (output) operations should not be executed with spooled devices. Output should be performed using File Control Services (FCS).

ASTX$

5.3.4 AST Service Exit ($S form recommended)

The AST Service Exit directive instructs the system to terminate execution of an AST service routine.

If another AST is queued and ASTs are not disabled, then the Executive immediately effects the next AST. Otherwise, the Executive restores the task's pre-AST state. See Notes below.

FORTRAN Call:

Neither the FORTRAN language nor the ISA standard permits direct linking to system trapping mechanisms; therefore, this directive is not available to FORTRAN tasks.

Macro Call:

```
ASTX$S [err]

  err  =  Error routine address
```

Macro Expansion:

```
ASTX$S  ERR
MOV     (PC)+,-(SP)        ;PUSH DPB ONTO THE STACK
.BYTE   115.,1             ;ASTX$S MACRO DIC, DPB SIZE=1 WORD
EMT     377                ;TRAP TO THE EXECUTIVE
JSR     PC,ERR             ;CALL ROUTINE "ERR" IF DIRECTIVE
                           ;UNSUCCESSFUL
```

Local Symbol Definitions:

None

DSW Return Codes:

IS.SUC	--	Successful completion.
IE.AST	--	Directive not issued from an AST service routine.
IE.ADP	--	Part of the DPB or stack is out of the issuing task's address space.
IE.SDP	--	DIC or DPB size is invalid.

Notes:

1. A return to the AST service routine occurs if, and only if, the directive is rejected. Therefore, no Branch on Carry Clear instruction is generated if an error routine address is given. (The return occurs only when the Carry bit is set.)

2. When an AST occurs, the Executive pushes, at minimum, the following information onto the task's stack:

SP+06	--	Event flag mask word
SP+04	--	PS of task prior to AST
SP+02	--	PC of task prior to AST
SP+00	--	DSW of task prior to AST

The task stack must be in this state when the AST Service Exit directive is executed.

In addition to the data parameters, the Executive pushes supplemental information onto the task stack for certain ASTs. For I/O completion, the stack contains the address of the I/O status block; for Mark Time, the stack contains the Event Flag Number; for a floating-point processor exception, the stack contains the exception code and address.

These AST parameters must be removed from the task's stack prior to issuing an AST exit directive. The following example shows how to remove AST parameters when a task uses an AST routine on I/O completion:

Example:

```
;
; EXAMPLE PROGRAM
;
; LOCAL DATA
;

IOSB:   .BLKW   2                  ;I/O STATUS DOUBLEWORD
BUFFER: .BLKW   30.                ;I/O BUFFER

;
; START OF MAIN PROGRAM
;

START:  .                          ;PROCESS DATA
        .
        .
        QIOW$C   IO.WVB,2,1,,IOSB,ASTSER,<BUFFER,60.,40>
        .
        .                          ;PROCESS & WAIT
        .
        EXIT$S                     ;EXIT TO EXECUTIVE

;
; AST SERVICE ROUTINE
;

ASTSER:                            ;PROCESS AST
        .
        .
        .
        TST     (SP)+              ;REMOVE ADDRESS OF I/O STATUS BLOCK
        ASTX$S                     ;AST EXIT
```

3. The task can alter its return address by manipulating the information on its stack prior to executing an AST exit directive. For example, to return to task state at an address other than the pre-AST address indicated on the stack, the task can simply replace the PC word on the stack. This procedure may be useful in those cases in which error conditions are discovered in the AST routine; but you should use extreme caution when doing this alteration since AST service routine bugs are difficult to isolate.

4. Because this directive requires only a 1-word DPB, the $S form of the macro is recommended. It requires less space and executes with the same speed as the DIR$ macro.

ATRG$

5.3.5 Attach Region

The Attach Region directive attaches the issuing task to a static common region or to a named dynamic region. (No other type of region can be attached to the task by means of this directive.) The Executive checks the desired access specified in the region status word against the owner UIC and the protection word of the region. If there is no protection violation, the Executive grants the desired access. If the region is successfully attached to the task, the Executive returns a 16-bit region ID (in R.GID), which the task uses in subsequent mapping directives.

You can also use the directive to determine the ID of a region already attached to the task. In this case, the task specifies the name of the attached region in R.GNAM and clears all four bits described below in the region status word R.GSTS. When the Executive processes the directive, it checks that the named region is attached. If the region is attached to the issuing task, the Executive returns the region ID, as well as the region size, for the task's first attachment to the region. You may want to use the Attach Region directive in this way to determine the region ID of a common block attached to the task at task-build time.

FORTRAN Call:

```
CALL ATRG (irdb[,ids])
```

irdb	=	An 8-word integer array containing a Region Definition Block (see Section 3.5.1.2)
ids	=	Directive status

Macro Call:

```
ATRG$   rdb
```

rdb = Region Definition Block address

Macro Expansion:

```
ATRG$   RDBADR
.BYTE   57.,2       ;ATRG$ MACRO DIC, DPB SIZE=2 WORDS
.WORD   RDBADR      ;RDB ADDRESS
```

Region Definition Block Parameters:

Input parameters:

Array Element	Offset		
irdb(3)(4)	R.GNAM	--	Name of the region to be attached
irdb(7)	R.GSTS	--	Bit settings[1] in the region status word (specifying desired access to the region):

1. If you are a FORTRAN programmer, refer to Section 3.5.1 to determine the bit values represented by the symbolic names described.

Bit		Definition
RS.RED	--	1 if read access is desired
RS.WRT	--	1 if write access is desired
RS.EXT	--	1 if extend access is desired
RS.DEL	--	1 if delete access is desired

Clear all four bits to request the region ID of the named region if it is already attached to the issuing task.

Output parameters

irdb(1)	R.GID	--	ID assigned to the region
irdb(2)	R.GSIZ	--	Size in 32-word blocks of the attached region

Local Symbol Definition:

A.TRBA	--	Region definition block address (2)

DSW Return Codes:

IS.SUC	--	Successful completion.
IE.UPN	--	An attachment descriptor cannot be allocated.
IE.PRI	--	Privilege violation.
IE.NVR	--	Invalid region ID.
IE.PNS	--	Specified region name does not exist.
IE.HWR	--	Region had parity error or load failure.
IE.ADP	--	Part of the DPB or RDB is out of the issuing task's address space.
IE.SDP	--	DIC or DPB size is invalid.

CINT$

5.3.6 Connect to Interrupt Vector

The Connect to Interrupt Vector directive enables a task to process hardware interrupts through a specified vector. The Interrupt Service Routine (ISR) is included in the task's own space. In a mapped system, the issuing task must be privileged.

The overhead entails the execution of about 10 instructions before entry into the ISR, and 10 instructions after exit from the ISR. The Executive provides a mechanism for transfer of control from the ISR to task-level code, using either an AST or a local event flag.

After a task has connected to an interrupt vector, it can process interrupts on three different levels: interrupt, fork, and task. The task level may be subdivided into: AST level and non-AST level.

1. Interrupt Level

   When an interrupt occurs, control is transferred, with the Interrupt Transfer Block (ITB) that has been allocated by the CINT$ directive, to the Executive subroutine $INTSC. From there control goes to the ISR specified in the directive.

   The ISR processes the interrupt and either dismisses the interrupt directly or enters fork level through a call to the Executive routine $FORK2.

2. Fork Level

   The fork-level routine executes at priority 0, the lowest processor priority, allowing interrupts and more time-dependent tasks to be serviced promptly. If required, the fork routine sets a local event flag for the task and/or queues an AST to an AST routine specified in the directive.

3. Task Level

   At task level, entered as the result of a local event flag or an AST, the task does final interrupt processing and has access to Executive directives.

Typically, the ISR does the minimal processing required for an interrupt and stores information for the fork routine or task-level routine in a ring buffer. The fork routine is entered after a number of interrupts have occurred as deemed necessary by the ISR, and further condenses the information. Finally, the fork routine wakes up the task-level code for ultimate processing that requires access to Executive directives. The fork level may, however, be a transient stage from ISR to task-level code without doing any processing.

In a mapped system, a task must be built privileged in order to be able to use the CINT$ directive. However, it is legal to use the /PR:0 switch to the Task Builder to have "unprivileged mapping," that is, up to 32K words of virtual address space available. This precludes use of the Executive subroutines from task-level code; however, the ISR and fork-level routines are always mapped to the Executive when they are executed. In any case, the Executive symbol table file (RSX11M.STB) should be included as input to the Task Builder.

As will be described later, in a mapped system, special considerations apply to the mapping of the ISR, fork routine, and enable/disable routine as well as all task data buffers accessed by these routines.

FORTRAN Call:

Not supported

Macro Call:

```
CINT$   vec,base,isr,edir,pri,ast
```

vec = Interrupt vector address -- Must be in the range 60(8) to highest vector specified during SYSGEN, inclusive, and must be a multiple of 4.

base = Virtual base address for kernel APR 5 mapping of the ISR, and enable/disable interrupt routines -- This address is automatically truncated to a 32(10)-word boundary. The "base" argument is ignored in an unmapped system.

isr = Virtual address of the ISR, or 0 to disconnect from the interrupt vector

edir = Virtual address of the enable/disable interrupt routine

pri = Initial priority at which the ISR is to execute -- This is normally equal to the hard-wired interrupt priority, and is expressed in the form n*40, where n is a number in the range 0-7. This form puts the value in bits 5-7 of pri. It is recommended that the programmer make use of the symbols PR4, PR5, PR6, and PR7 for this purpose. These are implemented via the macro HWDDF$ found in [1,1]EXEMC.MLB. Also, the programmer should take care to specify the correct value for this parameter. An incorrect initial priority (for example, specifying PR4 for a device that interrupts at PR5) may result in a system crash.

ast = Virtual address of an AST routine to be entered after the fork-level routine queues an AST

To disconnect from interrupts on a vector, the argument isr is set to 0 and the arguments base, edir, psw, and ast are ignored.

Macro Expansion:

```
CINT$   420,BADR,TADR,EDADR,PR5,ASTADR
.BYTE   129.,7              ;CINT$ MACRO DIC, DPB SIZE = 7 WORDS
.WORD   420                 ;INTERRUPT VECTOR ADDRESS = 420
.WORD   BADR                ;VIRTUAL BASE ADDRESS FOR KERNAL APR
.WORD   IADR                ;VIRTUAL ADDRESS OF THE INTERRUPT
                            ;SERVICE ROUTINE
.WORD   EDADR               ;VIRTUAL ADDRESS OF THE INTERRUPT
                            ;ENABLE/DISABLE ROUTINE
.BYTE   PR5,0               ;INITIAL INTERRUPT SERVICE ROUTINE
                            ;PRIORITY (LOW BYTE).  (HIGH BYTE = 0.)
.WORD   ASTADR              ;VIRTUAL ADDRESS OF AST ROUTINE
```

Local Symbol Definitions:

C.INVE	--	Vector address (2)
C.INBA	--	Base address (2)
C.INIS	--	ISR address (2)
C.INDI	--	Enable/disable interrupt routine address (2)
C.INPS	--	Priority (1)
C.INAS	--	AST address (2)

DSW Return Codes:

IE.UPN	--	An ITB could not be allocated (no pool space).
IE.ITS	--	The function requested is "disconnect" and the task is not the owner of the vector.
IE.PRI	--	Issuing task is not privileged (not applicable in unmapped system).
IE.RSU	--	The specified vector is already in use.
IE.ILV	--	The specified vector is illegal (lower than 60 or higher than highest vector specified during SYSGEN, or not a multiple of 4).
IE.MAP	--	ISR or enable/disable interrupt routine is not within 4K words from the value (base address & 177700).
IE.ADP	--	Part of the DPB is out of the issuing task's address space.
IE.SDP	--	DIC or DPB size is invalid.

Notes:

1. Checkpointable Tasks

   The following points should be noted for checkpointable tasks only:

   When a task connects to an interrupt vector, checkpointing of the task is automatically disabled.

   When a task disconnects from a vector and is not connected to any other vector, checkpointing of the task is automatically enabled, regardless of its state before the first connect, or any change in state while the task was connected.

2. Mapping Considerations

   In an unmapped system, the argument "base" is ignored, and the arguments "isr," "edir," and "ast" are physical addresses.

   In a mapped system, however; the argument "base," after being truncated to a 32(10)-word boundary, is the start of a 4K-word area mapped in kernel APR 5. All code and data in the task that is used by the routines must fall within that area, or a fatal error will occur, probably resulting in a system crash.

Furthermore, the code and data must be either position independent (refer to the PDP-11 MACRO-11 Language Reference Manual for more information on position-independent code) or coded in such a way that the code can execute in APR 5 mapping. When the routines execute, the processor is in kernel mode, and the virtual address space includes all of the Executive, the pool, and the I/O page.

References within the task image must be PC-relative or use a special offset defined below. References outside the task image must be absolute.

The following solutions are possible:

a. Write the ISR, enable/disable interrupt routines, and data in position-independent code.

b. Include the code and data in a common partition, task-build it with absolute addresses in APR 5 (PAR=ISR:120000:20000), and link the task to the common partition.

c. Build the task privileged with APR 5 mapping and use the constant 120000 as argument "base" in the CINT$ directive.

d. When accessing locations within the task image in immediate or absolute addressing mode, use an offset of

```
<120000-<base & 177700>>
```

3. ISR

When the ISR is entered, R5 points to the fork block in the Interrupt Transfer Block (ITB), and R4 is saved and free to be used. Registers R0 through R3 must be saved and restored if used. If one ISR services multiple vectors, the interrupting vector can be identified by the vector address, which is stored at offset X.VEC in the ITB. The following example loads the vector address into R4:

```
MOV X.VEC-X.FORK(R5),R4
```

The ISR either dismisses the interrupt directly by an RTS PC instruction, or calls $FORK2 if the fork routine is to be entered. When calling $FORK2, R5 must point to the fork block in the ITB, and the stack must be in the same state as it was upon entry to the ISR. Note that the call must use absolute addressing: CALL @#$FORK2.

NOTE

On RSX-11-M-PLUS systems, do not put the ISR in a common. Commons can be checkpointed or shuffled independently from the task and the Executive only disables checkpointing and shuffling for the task region.

4. Fork-Level Routine

   The fork-level routine starts immediately after the call to $FORK2. On entry, R4 and R5 are the same as when $FORK2 was called. All registers are free to be used. The first instruction of the fork routine must be CLR @R3, which declares the fork block free.

   The fork-level routine should be entered if servicing the interrupt takes more than 500 microseconds. It must be entered if an AST is to be queued or an event flag is to be set. (Fork level is discussed in greater detail in the <u>RSX-11M Guide to Writing an I/O Driver</u>.)

   An AST is queued by calling the subroutine $QASTC.

   Input: R5 -- pointer to fork block in the ITB

   Output: if AST successfully queued -- Carry bit = 0

   if AST was not specified by CINT$ -- Carry bit = 1

   Registers altered: R0, R1, R2, and R3

   An event flag is set by calling the subroutine $SETF.

   Input: R0 -- event flag number

   R5 -- Task Control Block (TCB) address of task for which flag is to be set -- This is usually, but not necessarily, the task that has connected to the vector. This task's TCB address is found at offset X.TCB in the ITB.

   Output: specified event flag set

   Registers altered: R1 and R2

   Note that absolute addressing must be used when calling these routines (and any other Executive subroutines) from fork level:

   ```
   CALL @#$QASTC

   CALL @#$SETF
   ```

5. Enable/Disable Interrupt Routine

   The purpose of the enable/disable interrupt routine, whose address is included in the directive call, is to allow the user to have a routine automatically called in the following three cases:

   a. When the directive is successfully executed to connect to an interrupt vector (argument isr nonzero) -- The routine is called immediately before return to the task.

   b. When the directive is successfully executed to disconnect from an interrupt vector (argument isr=0).

   c. When the task is aborted or exits with interrupt vectors still connected.

In case a, the routine is called with the Carry bit cleared; in cases b and c, with the Carry bit set. In all three cases, R1 is a pointer to the Interrupt Transfer Block (ITB). Registers R0, R2, and R3 are free to be used; other registers must be returned unmodified. Return is accomplished by means of an RTS PC instruction.

Typically, the routine dispatches to one of two routines, depending on whether the Carry bit is cleared or set. One routine sets interrupt enable and performs any other necessary initialization; the other clears interrupt enable and cleans up.

Note that the ITB contains the vector address, in the event that common code is used for multiple vectors.

6. AST Routine

   The fork routine may queue as AST for the task through a call to the Executive routine $QASTC as described above. When the AST routine is entered (at task level), the top word of the stack contains the vector address and must be popped off the stack before AST exit (ASTX$S).

7. ITB Structure

   The following offsets are defined relative to the start of the ITB:

X.LNK	--	Link word
X.JSR	--	Subroutine call to $INTSC
X.PSW	--	PSW for ISR (low-order byte)
X.ISR	--	ISR address (relocated)
X.FORK	--	Start of fork block
X.REL	--	APR 5 relocation (only in mapped systems)
X.DSI	--	Address of enable/disable interrupt routine (relocated)
X.TCB	--	TCB address of owning task
X.AST	--	Start of AST block
X.VEC	--	Vector address
X.VPC	--	Saved PC from vector
X.LEN	--	Length in bytes of ITB

   The symbols X.LNK through X.TCB are defined locally by the macro ITBDF$, which is included in [1,1]EXEMC.MLB. All symbols are defined globally by [1,1]EXELIB.OLB.

The following programming example illustrates the use of the CINT$ directive:

```
        .TITLE PUNTSK PUNCH ASCII TEXT ON PAPER TAPE PUNCH
;++
;       THIS TASK WILL PUNCH AN ASCII STRING TO THE PAPER TAPE PUNCH
;       USING THE CINT$ DIRECTIVE.
;
;       IT MUST BE BUILT USING THE /PR:0 TASK BUILDER SWITCH.
;       NOTE THAT THIS METHOD ALLOWS A TASK TO BE A FULL 32K
;       WORDS LONG.  IF IT IS NECESSARY TO ACCESS THE I/O PAGE
;       IN OTHER THAN THE ENABLE/DISABLE ROUTINE OR THE ISR
;       THE TASK MUST BE LINKED TO A COMMON BLOCK COVERING
;       THE CORRECT PART OF THE I/O PAGE.
;
;
;       TASK BUILD COMMAND FILE:
;
;       PUNTSK/MM/PR:0/-FP,PUNTSK/-SP/MA=PUNTSK
;       [1,54]RSX11M.STB/SS
;       /
;       GBLDEF=$VECTR:74
;       GBLDEF=$DVCSR:177554
;       UNITS=1
;       ASG=TI:1
;       PAR=GEN:0:40000
;
;
;       IT IS POSSIBLE TO HAVE THIS TASK TYPE ON THE CONSOLE TERMINAL
;       IF THERE IS NO PAPER TAPE PUNCH AVAILABLE.  TO DO THIS THE
;       VECTOR FOR THE CONSOLE OUTPUT MUST APPEAR TO BE UNUSED.  THIS
;       MAY BE DONE BY (ON A TERMINAL OTHER THAN THE CONSOLE!) OPENING
;       THE VECTOR LOCATION (64) AND REPLACING ITS CONTENTS WITH
;       THE VALUE OF '$NS0' AS OBTAINED FROM A MAP OF THE SYSTEM. BE
;       SURE TO REMEMBER THE OLD VALUE OR YOUR CONSOLE WILL BE DEAD
;       UNTIL YOU REBOOT THE SYSTEM.  NOW TASK BUILD USING THE FOLLOWING
;       COMMAND FILE:
;
;       PUNTTY/MM/PR:0,/-FP,PUNTTY/-SP/MA=PUNTSK
;       [1,54]RSX11M.STB/SS
;       /
;       GBLDEF=$VECTR:64
;       GBLDEF=$DVCSR:177564
;       UNITS=1
;       ASG=TI:1
;       PAR=GEN:0:40000
;
;
;       NOTE THAT IN THE ABOVE TWO TKB COMMAND FILES THE FOLLOWING
;       CHANGES MUST BE MADE IN ORDER TO RUN ON AN UNMAPPED SYSTEM:
;
;       1) /MM SHOULD BE CHANGED TO /-MM
;       2) 'PAR=GEN:0:40000' SHOULD BE CHANGED TO
;               'PAR=GEN:40000:40000'
;
;       IN ADDITION, PLACE A SEMI-COLON IN FRONT OF THE SOURCE LINE
;       BELOW THAT DEFINE THE SYMBOL 'M$$MGE'.
;--
.MCALL CINT$, QIOW$, CLEF$S, WTSE$S, EXIT$S, DIR$
;
; LOCAL SYMBOLS
;
```

```
      LUN.TT        =       1         ;LUN FOR TERMINAL I/O
      EFN.TT        =       1         ;EFN FOR TERMINAL I/O
      EFN.WF        =       2         ;EFN TO WAIT FOR PUNCHING TO COMPLETE
      M$$MGE        =       0         ;DEFINE THIS SYMBOL TO RUN ON MAPPED SYSTEM
      ;++
      ;         MACRO TO GENERATE AN ASCII STRING AND A QIO TO OUTPUT
      ;         THE STRING TO THE TERMINAL.
      ;
      ;         MESSG   NAM,STRING
      ;
      ;         WHERE:
      ;
      ;         NAM     IS THE NAME OF THE GENERATED QIO DPB
      ;         STRING  IS THE ASCII STRING TO OUTPUT
      ;
      ;--

                  .MACRO  MESSG   NAM,STRING,?LBL
                  $CHR=0
                  .IRPC   X,<STRING>
                  $CHR=$CHR+1
                  ENDR
                  .ENABL  LSB
LBL:              .ASCII  /STRING/
                  .EVEN
NAM:              QIOW$  IO.WVB,LUN.TT,EFN.TT,,,,<LBL,$CHR,40>
                  .DSABL  LSB
                  .ENDM

                  MESSG  HELLO,<CONNECT TO INTERRUPT TEST>
                  MESSG  CINWRK,<CONNECT TO INTERRUPT WORKS--CHECK THE PAPER TAPE PUNCH>

CINT:             CINT$  $VECTR,$BASE,$PNISR,$PNEDI,PR4

                                        ;CONNECT TO INTERRUPT
                                        ;       VECTOR=$VECTR
                                        ;       BASE.FOR.MAPPING=$BASE
                                        ;       ISR=$PNISR
                                        ;       ENB.DSABL.RTN=$PNEDI
                                        ;       PRIO=PR4

          DISCON:CINT$  $VECTR,0,0      ;DISCONNECT FROM INTERRUPT
                                        ;       VECTOR=74
          ;++
          ;    ENTRY POINT TO THE PUNCH TASK.  THE TASK WILL ANNOUNCE
          ;    ITSELF ON THE INITIATING TERMINAL, CONNECT TO THE
          ;    SPECIFIED VECTOR, OUTPUT THE ASCII STRING, AND THEN
          ;    OUTPUT A MESSAGE THAT IT WAS SUCCESSFUL.  IF THE TASK
          ;    TERMINATES WITH AN I/O TRAP THE CONNECT-TO-INTERRUPT
          ;    DIRECTIVE FAILED, AND R1 WILL CONTAIN THE DSW RETURNED
          ;    IN ORDER TO DIAGNOSE THE ERROR.
          ;--

          $PUNTK::DIR$      #HELLO      ;ANNOUNCE THAT WE ARE HERE
                  DIR$      #CINT       ;CONNECT TO THE PUNCH
                                        ;       THIS CAN BE EITHER THE TERMINAL
                                        ;       OR THE PAPER TAPE PUNCH.
                  BC$       ERR1        ;IF CS THEN DIRECTIVE ERROR
                  WTSE$S    #EFN.WF     ;WAIT FOR PUNCH TO FINISH
                  DIR$      #DISCON     ;DISCONNECT FROM INTERRUPTS
                  DIR$      #CINWRK     ;TELL USER THAT CINT WORKS
                  EXIT$S
```

```
        ERR1:   MOV      #1,R0       ;ERROR # 1
                MOV      $DSW,R1     ;GET THE DSW TO SHOW THE CINT ERROR RETURN
                IOT                  ;DUMP REGISTERS

$BASE;                               ;THIS IS THE BASE OF THE MAPPING USED
                                     ;BY THE EXECUTIVE WHEN MAPPING TO THE
                                     ;'DRIVER'. THIS MAPPING IS REQUIRED
                                     ;ONLY ON MAPPED SYSTEMS; UNMAPPED
                                     ;SYSTEMS DO NOT HAVE THIS PROBLEM.

        ;++
        ;        FOLLOWING IS THE ASCII STRING PUNCHED BY THIS TASK.
        ;--

                .NLIST  BEX
        PUNMSG: .ASCIZ  /ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789!@#$%^&*()_+-=/<15><12>
                 .LIST  BEX
                 .EVEN

        PUNPTR: .WORD   0            ;POINTER INTO PUNMSG FOR ISR
        TSKTCB: .WORD   0            ;TCB ADDRESS OF TASK
        PUNCSR: .WORD   $DVCSR       ;PAPER TAPE PUNCH CSR ADDRESS
        PUNBUF: .WORD   $DVCSR+2     ;PAPER TAPE PUNCH BUFFER ADDRESS
        ;++
        ;    ENABLE/DISABLE ROUTINE.
        ;
        ;    THIS ROUTINE IS CALLED BY THE EXEC ON EITHER A CONNECT OR DISCONNECT
        ;    FROM INTERRUPT VECTOR REQUEST, OR WHEN THE TASK EXITS WITH INTERRUPT
        ;    VECTORS STILL CONNECTED.
        ;
        ;    ENTRY CONDITIONS:
        ;
        ;    C-CLEAR        THIS IS A SUCCESSFUL CONNECT.
        ;    C-SET          THIS IS A DISCONNECT.
        ;
        ;    $TKTCB         THE TCB ADDRESS OF THE CURRENTLY EXECUTING TASK (MEM).
        ;
        ;    ACTION:
        ;
        ;    IF THE C-BIT IS SET WE MERELY DISABLE THE PUNCH AND RETURN.  IF
        ;    THE C-BIT IS CLEAR WE WILL ENABLE THE PUNCH TO INTERRUPT.  THIS
        ;    WILL IMMEDIATELY CAUSE AN INTERRUPT AND THE INTERRUPT SERVICE
        ;    ROUTINE WILL OUTPUT CHARACTERS TO THE PUNCH (ONE PER
        ;    INTERRUPT) UNTIL A ZERO BYTE IS OUTPUT.  THE ISR WILL THEN FORK
        ;    AND SET THE LOCAL EVENT FLAG 'EFN.WF'. THIS WILL THEN CAUSE THE
        ;    TASK PORTION OF THIS TASK TO CONTINUE EXECUTING AND EVENTUALLY
        ;    EXIT.
        ;
        ;--
```

```
$PNEDI::BCS     20$          ;IF CS THEN DISCONNECT
        MOV     @#$TKTCB,TSKTCB ;COPY TASK TCB ADDRESS FOR LATER
                             ;SO WE CAN SET EFN.

        .IF DF M$$MGE        ;MAPPED SYSTEM?

        MOV     #PUNMSG+120000-<$BASE&177700>,PUNPTR ;RELOCATE ADDRESS
                             ;TO APR 5 MAPPING, AND SET UP
                             ;BUFFER POINTER

        .IFF    M$$MGE       ;UNMAPPED SYSTEM?

        MOV     #PUNMSG,PUNPTR ;SET UP BUFFER POINTER

        .ENDC

        BIS     #100,@PUNCSR ;ALLOW INTERRUPTS
        RETURN
                             ;WHEN WE ARE DONE PUNCHING

20$:    BIC     #100,@PUNCSR ;DISABLE INTERRUPTS
        RETURN

        .END
```

CLEF$

5.3.7 Clear Event Flag

The Clear Event Flag directive instructs the system to report an indicated event flag's polarity and then clear it.

FORTRAN Call:

```
     CALL CLREF (efn[,ids])

       efn  =  Event flag number

       ids  =  Directive status
```

Macro Call:

```
     CLEF$   efn

       efn  =  Event flag number
```

Macro Expansion:

```
     CLEF$   52.
     .BYTE   31.,2             ;CLEF$ MACRO DIC, DPB SIZE=2 WORDS
     .WORD   52.               ;EVENT FLAG NUMBER 52.
```

Local Symbol Definitions:

C.LEEF	--	Event flag number (2)

DSW Return Codes:

IS.CLR	--	Successful completion; flag was already clear.
IS.SET	--	Successful completion; flag was set.
IE.IEF	--	Invalid event flag number (EFN<1, or EFN>96, if group global event flags exist for the task's group; or EFN>64 if not).
IE.ADP	--	Part of the DPB is out of the issuing task's address space.
IE.SDP	--	DIC or DPB size is invalid.

5.3.8 Cancel Mark Time Requests

The Cancel Mark Time Requests directive instructs the system to cancel a specific Mark Time Request or all Mark Time requests that have been made by the issuing task.

FORTRAN Call:

```
     CALL CANMT ([efn][,ids])

       efn  =  Event flag number

       ids  =  Directive status
```

Macro Call:

```
     CMKT$  [efn,ast,err]

       err  =  Error routine address

       efn  =  Event flag number

       ast  =  Mark time AST address
```

Macro Expansion:

```
     CMKT$   52.,MRKAST,ERR ;NOTE: THERE ARE TWO IGNORED ARGUMENTS
     .BYTE   27.,3          ;CMKT$ MACRO DIC, DPB SIZE=3 WORDS
     .WORD   52.            ;EVENT FLAG NUMBER 52.
     .WORD   MRKAST         ;ADDRESS OF MARK TIME REQUEST AST ROUTINE
```

NOTE

The above example will cancel only the Mark Time requests that were specified with efn 52 or the AST address MRKAST. If no ast or efn parameters are specified, all Mark Time requests issued by the task are cancelled, and the DPB size will equal 1.

Local Symbol Definitions:

C.MKEF	--	Event flag number (2)
C.MKAE	--	Mark Time Request AST routine address (2)

DSW Return Codes:

IS.SUC	--	Successful completion.
IE.ADP	--	Part of the DPB is out of the issuing task's address space.
IE.SDP	--	DIC or DPB size is invalid.

Notes:

1. If neither the efn nor ast parameters are specified, all Mark Time Requests issued by the task are canceled. In addition, the DPB size will be one word. (When either the efn and/or ast parameters are specified, the DPB size will be three words.)

2. If both efn and ast parameters are specified (and nonzero), only Mark Time Requests issued by the task specifying either that event flag or AST address are canceled.

3. If only one efn or ast parameter is specified (and nonzero), only Mark Time Requests issued by the task specifying the event flag or AST address are canceled.

4. If the specified event flag is a group global, then the use count for the event flag's group is run down when a Mark Time request is canceled.

5.3.9 Connect

The Connect directive synchronizes the task issuing the directive with the exit or emit status of another task (offspring) that is already active. Execution of this directive queues an Offspring Control Block (OCB) to the offspring task, and increments the issuing task's rundown count (contained in the issuing task's Task Control Block). The rundown count is maintained to indicate the combined total number of tasks presently connected as offspring tasks and the total number of virtual terminals the task has created. The exit AST routine is called when the offspring exits or emits status with the address of the associated exit status block on the stack. This directive should not be issued to connect to Command Line Interpreter (CLI) tasks; it is illegal to connect to a CLI task.

FORTRAN Call:

```
CALL CNCT (rtname,[iefn],[iast],[iesb],[iparm][,ids])
```

rtname	=	A single-precision, floating-point variable containing the offspring task name in Radix-50 format.
iefn	=	Event flag to be set when the offspring task exits or emits status
iast	=	Name of an AST routine to be called when the offspring task exits or emits status
iesb	=	Name of an 8-word status block to be written when the offspring task exits or emits status

Word 0 -- Offspring task exit status

Word 1 -- TKTN abort code

Word 2-7 -- Reserved

NOTE

The exit status block defaults to one word. To use the 8-word exit status block, you must specify the logical or of the symbol SP.WX8 and the event flag number in the iefn parameter above.

iparm	=	Name of a word to receive the status block address when an AST occurs
ids	=	Integer to receive the Directive Status Word

Macro Call:

```
CNCT$   tname,[efn],[east],[esb]
```

tname	=	Name (RAD50) of the offspring task to be connected
efn	=	The event flag to be cleared on issuance and set when the offspring task exits or emits status

east	=	Address of an AST routine to be called when the offspring task exits or emits status
esb	=	Address of an 8-word status block to be written when the offspring task exits or emit status

word 0 -- Offspring task exit status

word 1 -- TKTN abort code

word 2-7 -- Reserved

NOTE

The exit status block defaults to one word. To use the 8-word exit status block, you must specify the logical or of the symbol SP.WX8 and the event flag number in the efn parameter above.

Macro Expansion:

```
     CNCT$      ALPHA,1,CONAST,STBUF
     .BYTE      143.,6          ;CNCT$ MACRO DIC, DPB SIZE=6 WORDS
     .RAD50     ALPHA           ;OFFSPRING TASK NAME
     .BYTE      1               ;EVENT FLAG NO = 1
     .BYTE      16.             ;EXIT STATUS BLOCK CONSTANT
     .WORD      CONAST          ;AST ROUTINE ADDRESS
     .WORD      STBUF           ;EXIT STATUS BLOCK ADDRESS
```

Local Symbol Definitions:

C.NCTN	--	Task name (4)
C.NCEF	--	Event flag (2)
C.NCEA	--	AST routine address (2)
C.NCES	--	Exit status block address (2)

DSW Return Codes:

IS.SUC	--	Successful completion.
IE.UPN	--	Insufficient dynamic memory to allocate an offspring control block.
IE.INS	--	The specified task was a command line interpreter.
IE.ACT	--	The specified task was not active.
IE.IEF	--	Invalid event flag number (EFN<0, or EFN>96 if group global event flags exist for the task's group; or EFN>64 if not).
IE.ADP	--	Part of the DPB or exit status block is not in the issuing task's address space.
IE.SDP	--	DIC or DPB size is invalid.

Notes:

1. If the specified event flag is a group global, the use count for the event flag's group is incremented to prevent premature elimination of the event flags. The use count is run down when:

   - The connected task returns status

   - The issuing task exits before status is returned

2. Do not change the virtual mapping of the exit status block while the connection is in effect. Doing so may cause obscure errors since the exit status block is always returned to the virtual address specified regardless of the physical address to which it is mapped.

CPCR$

5.3.10 Checkpoint Common Region

This directive instructs the system to force the specified Read/Write common region to be checkpointed. This directive stops all the tasks that are mapped to the common region, writes the common region out to the disk and then unstops the tasks.

The issuing task must be privileged (PR:0).

The issuing task must be attached to the specified common region.

If the issuing task is mapped to the specified common region, it is blocked. Any task (including the issuing task) is also blocked if it maps to the common region while the checkpoint is in progress. If the task was built with the /COMMON= qualifier, the task will be blocked when it issues this directive. If the task becomes attached by means of the Attach Region directive, it is not blocked unless it issues a Map directive.

You can use this directive to preserve changes made to a memory resident common region. When a region is checkpointed, it is copied to its own image on the disk and not to the checkpoint file. Therefore, any update to the memory resident copy of the common region becomes permanent.

FORTRAN Call:

```
     CALL CPCR(name[,ids])

          name = Name  (in  RAD50)  of  the  common  region  to  be
                 checkpointed

          ids  = Directive Status
```

Macro Call:

```
     CPCR$   name

          name = Name of the common region to be checkpointed
```

Macro Expansion:

```
     CPCR$   NAME
     .BYTE   205.,3          ;DIC =205., DPB SIZE= 3 WORDS
     .RAD50  /NAME/
```

Local Symbol Definitions:

```
     C.PCR   --  Name of common region
```

DSW Return Codes:

IE.SUC	--	Successful completion.
IE.PRI	--	Privilege violation.
IE.NSP	--	The specified common region does not exist.
IE.ITS	--	I/O in progress to the specified region.
IE.ADP	--	Part of the DPB is out of the issuing task's address space.
IE.SDP	--	DIC or DPB size is invalid.

CRAW$

5.3.11 Create Address Window

The Create Address Window directive creates a new virtual address window by allocating a window block from the header of the issuing task and establishing its virtual address base and size. (Space for the window block has to be reserved at task-build time by means of the WNDWS keyword. See the RSX-11M/M-PLUS Task Builder Manual.) Execution of this directive unmaps and then eliminates any existing windows that overlap the specified range of virtual addresses. If the window is successfully created, the Executive returns an 8-bit window ID to the task.

The 8-bit window ID returned to the task is a number from 1 to 15 (1 to 23 on RSX-11M-PLUS systems), which is an index to the window block in the task's header. The window block describes the created address window.

If WS.SIS in the window status word is set, the Executive creates the window in supervisor-mode I-space. Program control can subsequently be transferred to supervisor-mode I-space upon issuing a Supervisor Call directive.

If WS.UDS in the window status word is set, the Executive creates the window in user-mode D-space.

If WS.MAP in the window status word is set, the Executive proceeds to map the window according to the Window Definition Block input parameters.

A task can specify any length for the mapping assignment that is less than or equal to both the window size specified when the window was created, and the length remaining between the specified offset within the region and the end of the region.

If W.NLEN is set to 0, the length defaults to either the window size or the length remaining in the region, whichever is smaller. (Because the Executive returns the actual length mapped as an output parameter, the task must clear that offset before issuing the directive each time it wants to default the length of the map.)

The values that can be assigned to W.NOFF depend on the setting of bit WS.64B in the window status word (W.NSTS):

- If WS.64B = 0, the offset specified in W.NOFF must represent a multiple of 256 words (512 bytes). Because the value of W.NOFF is expressed in units of 32-word blocks, the value must be a multiple of 8.

- If WS.64B = 1, the task can align on 32-word boundaries; the programmer can therefore specify any offset within the region.

NOTE

Applications dependent on 32-word or 64-byte alignment (WS.64B = 1) may not be compatible with future RSX emulators. To avoid future incompatibility, programmers should write applications adaptable to either alignment requirement. The bit setting of WS.64B could be a parameter chosen at assembly time (by means of a prefix file), at task-build time (as input to the GBLDEF option), or at run time (by means of command input).

FORTRAN Call:

```
CALL CRAW (iwdb[,ids])
```

iwdb	=	An 8-word integer array containing a window definition block (see Section 3.5.2.2)
ids	=	Directive status

Macro Call:

```
CRAW$   wdb
```

wdb = Window Definition Block address

Macro Expansion:

```
CRAW$   WDBADR
.BYTE   117.,2    ;CRAW$ MACRO DIC, DPB SIZE=2 WORDS
.WORD   WDBADR    ;WDB ADDRESS
```

Window Definition Block Parameters:

Input parameters:

Array Element	Offset		
iwdb(1), bits 8-15	W.NAPR	--	Base APR of the address window to be created.
iwdb(3)	W.NSIZ	--	Desired size, in 32-word blocks, of the address window.
iwdb(4)	W.NRID	--	ID of the region to which the new window is to be mapped, or 0 for task region (to be specified only if WS.MAP=1).
iwdb(5)	W.NOFF	--	Offset in 32-word blocks from the start of the region at which the window is to start mapping (to be specified only if WS.MAP=1). Note that if WS.64B in the window status word equals 0, the value specified must be a multiple of 8.

iwdb(6)	W.NLEN	-- Length in 32-word blocks to be mapped, or 0 if the length is to default to either the size of the window or the space remaining in the region, whichever is smaller (to be specified only if WS.MAP=1).
iwdb(7)	W.NSTS	-- Bit settings[1] in the window status word:

Bit	Definition
WS.MAP	1 if the new window is to be mapped
WS.WRT	1 if the mapping assignment is to occur with write access
WS.64B	0 for 256-word (512-byte) alignment; or 1 for 32-word (64-byte) alignment

Output parameters

iwdb(1), bits 0-7	W.NID	-- ID assigned to the window
iwdb(2)	W.NBAS	-- Virtual address base of the new window
iwdb(6)	W.NLEN	-- Length, in 32-word blocks, actually mapped by the window
iwdb(7)	W.NSTS	-- Bit settings[1] in the window status word:

Bit	Definition (if bit=1)
WS.CRW	Address window was successfully created.
WS.UNM	At least one window was unmapped.
WS.ELW	At least one window was eliminated.
WS.RRF	Reference was successfully received.
WS.NBP	Do not bypass the cache (for multiprocessor systems).
WS.RES	Map only if resident.
WS.NAT	Create attachment descriptor only if necessary (for Send By Reference directives).

1. If you are a FORTRAN programmer, refer to Section 3.5.2 to determine the bit values represented by the symbolic names described.

Bit	Definition (if bit=1)
WS.64B	Define the task's permitted alignment boundaries -- 0 for 256-word (512-byte) alignment; or 1 for 32-word (64-byte) alignment.
WS.MAP	Window is to be mapped.
WS.RCX	Exit if no references to receive.
WS.SIS	Create window in supervisor I-space.
WS.UDS	Create window in user D-space.
WS.DEL	Send with delete access.
WS.EXT	Send with extend access.
WS.WRT	Send with write access or map with write access.
WS.RED	Send with read access.

Local Symbol Definitions:

C.RABA	-- Window definition block address (2)

DSW Return Codes:

IS.SUC	-- Successful completion.
IE.HWR	-- Directive failed in mapping storage because region has incurred a parity error.
IE.PRI	-- Requested access denied at mapping stage.
IE.NVR	-- Invalid region ID.
IE.ALG	-- Task specified either an invalid base APR and window size combination, or an invalid region offset and length combination in the mapping assignment; or WS.64B = 0 and the value of W.NOFF is not a multiple of 8.
IE.WOV	-- No window blocks available in task's header.
IE.ADP	-- Part of the DPB or WDB is out of the issuing task's address space.
IE.SDP	-- DIC or DPB size is invalid.

CRGF$

5.3.12 Create Group Global Event Flags

The Create Group Global Event Flags directive creates a Group Global Event Flag Control Block (GFB) and links it into the GFB list. If a GFB for the specified group is not present when the directive is issued, the Executive creates the GFB data structure with all event flags initialized to zero. If a GFB is present when the directive is issued, the Executive uses the present GFB and the event flags are not initialized. However, if the GFB is marked for delete (by a previously issued Eliminate Group Global Event Flags directive), the Executive clears the GS.DEL bit (see Section 5.3.20).

If the specified group code matches the group code of the issuing task's protection UIC (H.CUIC+1), this directive increments the access count for the event flags. This locks the event flags so they cannot be eliminated by another task that is sharing them. The issuing task can explicitly unlock the event flags with an Unlock Group Global Event Flags directive or an Eliminate Group Global Event Flags directive. The Executive automatically unlocks the event flags when the task exits if necessary. Note that a task may not lock the event flags more than once in succession. Any attempt to lock event flags that are already locked will return the IE.RSU error code.

FORTRAN Call:

```
     CALL CRGF ([group][,ids])
```

group	=	Group number for the flags to be created. Only privileged tasks can specify group numbers other than the issuing task's group UIC. If not specified, the task's protection UIC (H.CUIC+1) in the task's header is used.
ids	=	Integer to receive the Directive Status Word.

Macro Call:

```
     CRGF$    [group]
```

group	=	Group number for the flags to be created. Only privileged tasks can specify group numbers other than the issuing task's group UIC. If not specified, the task's protection UIC (H.CUIC+1) in the task's header is used.

Macro Expansion:

```
     CRGF$      4
     .BYTE      157.,2          ;CRGF$ MACRO DIC, DPB SIZE=2 WORDS
     .WORD      4               ;GROUP 4 GLOBAL EVENT FLAGS
```

Local Symbol Definitions:

```
     C.RGRP -- Group Number (2)
```

DSW Return Codes:

IS.SUC	-- Successful completion.
IE.UPN	-- Insufficient dynamic storage.
IE.PRI	-- Privilege violation.
IE.IUI	-- Invalid group.
IE.RSU	-- Event flags already exist or are already locked.
IE.APD	-- Part of the DPB is out of the issuing task's address space.
IE.DIC	-- DIC or DPB size is invalid.

Note:

A privileged task may specify group numbers other than the group UIC of the issuing task. However, the task can only lock the event flags created for its own group. This directive does not return an error if it does not lock the event flags.

CRRG$

5.3.13 Create Region

The Create Region directive creates a dynamic region in a system-controlled partition and optionally attaches it to the issuing task.

If RS.ATT is set in the region status word, the Executive attempts to attach the task to the newly created region. If no region name has been specified, the user's program must set RS.ATT (see the description of the Attach Region directive).

By default, the Executive automatically marks a dynamically created region for deletion when the last task detaches from it. To override this default condition, set RS.NDL in the region status word as an input parameter. Be careful in considering to override the delete-on-last-detach option. An error within a program can cause the system to lock by leaving no free space in a system-controlled partition.

If the region is not given a name, the Executive ignores the state of RS.NDL. All unnamed regions are deleted when the last task detaches from them.

Named regions in RSX-11M-PLUS systems are put in the Common Block Directory (CBD). However, memory is not allocated until the Executive maps a task to the region.

The Executive returns an error if there is not enough space to accommodate the region in the specified partition. See Notes below.

FORTRAN Call:

```
     CALL CRRG (irdb[,ids])

       irdb  =  An 8-word integer array containing a Region Definition
                Block (see Section 3.5.1.2)

       ids   =  Directive status
```

Macro Call:

```
     CRRG$   rdb

       rdb  =  Region Definition Block address
```

Macro Expansion:

```
     CRRG$   RDBADR
     .BYTE   55.,2        ;CRRG$ MACRO DIC, DPB SIZE = 2 WORDS
     .WORD   RDBADR       ;RDB ADDRESS
```

Region Definition Block Parameters:

Input parameters:

Array Element	Offset	
irdb(2)	R.GSIZ	-- Size, in 32-word blocks, of the region to be created

irdb(3)(4)	R.GNAM	--	Name of the region to be created, or 0 for no name
irdb(5)(6)	R.GPAR	--	Name of the system-controlled partition in which the region is to be allocated, or 0 for the partition in which the task is running
irdb(7)	R.GSTS	--	Bit settings[1] in the region status word:

Bit	**Definition (if bit=1)**
RS.CRR	Region was successfully created.
RS.UNM	At least one window was unmapped on a detach.
RS.MDL	Mark region for deletion on last detach.
RS.NDL	The region should not be deleted on last detach.
RS.ATT	Created region should be attached.
RS.NEX	Created region is not extendible.
RS.RED	Read access is desired on attach.
RS.WRT	Write access is desired on attach.
RS.EXT	Extend access is desired on attach.
RS.DEL	Delete access is desired on attach.

irdb(8)	R.GPRO	--	Protection word for the region (DEWR,DEWR,DEWR,DEWR)

Output parameters

irdb(1)	R.GID	--	ID assigned to the created region (returned if RS.ATT=1)
irdb(2)	R.GSIZ	--	Size in 32-word blocks of the attached region (returned if RS.ATT=1)
irdb(7)	R.GSTS	--	Bit settings[1] in the region status word:

Bit	**Definition**
RS.CRR	1 if the region was successfully created

1. If you are a FORTRAN programmer, refer to Section 3.5.1 to define the bit values represented by the symbolic names described.

Local Symbol Definitions:

C.RRBA	--	Region Definition Block address (2)

DSW Return Codes:

IS.SUC	--	Successful completion.
IE.UPN	--	A Partition Control Block (PCB) or an attachment descriptor could not be allocated, or the partition was not large enough to accommodate the region, or there is currently not enough continuous space in the partition to accommodate the region.
IE.HWR	--	The directive failed in the attachment stage because a region parity error was detected.
IE.PRI	--	Attach failed because desired access was not allowed.
IE.PNS	--	Specified partition in which the region was to be allocated does not exist; or no region name was specified and RS.ATT = 0.
IE.ADP	--	Part of the DPB or RDB is out of issuing task's address space.
IE.SDP	--	DIC or RDB size is invalid.

Notes:

1. The Executive does not return an error if the named region already exists. In this case, the Executive clears the RS.CRR bit in the status word R.GSTS. If RS.ATT has been set, the Executive attempts to attach the already existing named region to the issuing task.

2. The protection word (see R.GPRO above) has the same format as that of the file system protection word. There are four categories, and the access for each category is coded into four bits. From low order to high order, the categories follow this order: system, owner, group, world. The access code bits within each category are arranged (from low order to high order) as follows: read, write, extend, delete. A bit that is set indicates that the corresponding access is denied.

   The issuing task's UIC is the created region's owner UIC.

   In order to prevent the creation of common blocks that are not easily deleted, the system and owner categories are always forced to have delete access, regardless of the value actually specified in the protection word.

5.3.14 Create Virtual Terminal

The Create Virtual Terminal directive creates a virtual terminal for use by a parent task in communicating with its offspring tasks. When the offspring task issues a read or write to its TI: terminal, the request is sent to the parent task through the virtual terminal. For example, when the Batch Processor invokes a task, it communicates with that task through a virtual terminal rather than a physical terminal.

This directive creates a Device Control Block (DCB) and a Unit Control Block (UCB) for each virtual terminal unit, and links the unit to the device list. Each newly created virtual terminal unit is assigned the lowest available virtual terminal unit number.

Only a single copy of the Status Control Block (SCB) is required. The data structure for Virtual Terminal Unit 0 (VT0:) is used as a template for these dynamically created data structures. Therefore, VT0: is never assigned as a virtual terminal unit number.

On successful completion of this directive, the assigned VT: unit number is returned in the DSW with the Carry bit clear. The task must save this number if this virtual terminal is to be referenced in another directive.

A rundown count is maintained in the issuing task's TCB to indicate the total (current) number of virtual terminals the task has created and the number of connected offspring tasks. This count is reduced when an Eliminate Virtual Terminal directive is issued specifying this VT: unit.

The input and output AST routines for the virtual terminal unit are entered with the following three words on the stack:

SP+04	--	Third parameter word (VFC) of offspring request
SP+02	--	Byte count of offspring request
SP+00	--	Virtual terminal unit number (low byte); I/O subfunction code of offspring request (high byte)

The attach and detach AST routines are entered with the following three words on the stack:

SP+04	--	Second word of offspring task name (0 if detach AST)
SP+02	--	First word of offspring task name (0 if detach AST)
SP+00	--	Virtual terminal unit number (low byte); I/O subfunction code of offspring request (high byte)

Note that the detach AST routine is entered with 0 in both task name words on the stack. The AST routine must remove the three words from the stack before it issues an AST Service Exit directive.

Parent tasks can service each offspring input or output request with a corresponding output or input request to the correct virtual device unit. For example, where Macro-11 has been activated as an offspring task of the Batch Processor with a TI: of VT3:

1. Macro-11 issues an IO.RVB or IO.RLB to TI: for its first input line. The virtual terminal driver queues the read request internally and effects an AST in the Batch Processor at the virtual address "iast" with the unit number 3 and the byte count from Macro-11's I/O request on the stack.

2. In its AST routine, the Batch Processor retrieves an input line for Macro-11 from the Batch stream and specifies this line in a QIO directive to a LUN assigned to VT3: with an IO.WVB or IO.WLB function, a byte count of the line, and the status to be returned (such as IS.CR).

3. The virtual terminal driver reads the line from the Batch Processor's buffer, writes the line to Macro-11's buffer, and then signals I/O completion for both I/O requests.

4. Similarly, if Macro-11 needs to print an error message, it does so with an IO.WVB or IO.WLB to TI:. The virtual terminal driver queues the write request internally and effects an AST in the Batch Processor at the virtual address "oast" with the unit number 3, the byte count, and the VFC from Macro-11's I/O request on the stack.

5. In its output AST routine, the Batch Processor issues an IO.RVB or IO.RLB to retrieve the line by means of the virtual terminal driver. The Batch Processor may then output this line to its log file. The third word on the AST stack in the Batch output AST routine is the vertical format character, telling Batch what type of carriage control is expected for the output line. This word would be ignored in the input AST routine.

The virtual terminal driver does not interpret or modify transferred bytes, I/O subfunction codes, or vertical format characters. However, this driver does automatically truncate offspring I/O requests to the maximum byte count specified in the "mlen" parameter, notifying neither the parent nor offspring task. The actual number of bytes transferred on each request is equal to the smaller of the byte counts specified in the offspring and parent I/O requests. The total number of bytes transferred is returned in the corresponding I/O status blocks. Note that offspring tasks can receive "mlen" in the fourth characteristics word when a Get LUN Information directive is issued.

Intermediate buffering in secondary pool, when enabled by the parent task, is performed on offspring input and output requests when the offspring task is checkpointable. Offspring tasks, therefore, may be stopped and checkpointed. If the parent task is stopped and checkpointed when the offspring task issues an I/O request, the resulting AST brings the parent task to an unstopped state from which it may return to memory to service the I/O request. Upon exit from the AST routine, the parent task is again stopped. This mode of operation allows the parent and offspring tasks to share the same physical memory, even while the parent task services the terminal I/O requests for the offspring task. Whenever, for any reason, the virtual terminal driver determines that is should not use intermediate buffering, offspring tasks are locked in memory when I/O requests are issued, and transfers occur directly between parent and offspring buffers.

The intermediate buffering of offspring I/O requests can normally be enabled and disabled by the parent task by the IO.STC function, as described below. An exception to this exists for virtual terminals created with a "mlen" parameter greater than a system-wide maximum specified at Sysgen time. (Sysgen does not allow this maximum to be greater than 512.) If a Create Virtual Terminal directive is specified with a "mlen" parameter greater than the system-wide maximum, the parameter is accepted, but intermediate buffering for the created virtual terminal unit is automatically disabled. Furthermore, intermediate buffering for that unit cannot be enabled by the parent task by the IO.STC function.

Parent tasks specify the first word of the I/O completion status for the offspring request in the third word of the QIO DPB. For example, consider an offspring input request for 10 characters or more that is honored with a write logical of 10 characters and IS.CR in the third parameter word. The second word of the I/O status would be set to 10 and 10 characters would be transferred. Another example is when a parent task issues a read request to satisfy a write request that was issued by the offspring task. To notify the offspring task that its write request was satisfied the parent task would specify IS.SUC in the third parameter word.

A special I/O function, IO.STC, returns status to an offspring task without a data transfer. The parameter word format for the IO.STC function is as follows:

- Word 0 with bit 0 set indicates that status is being returned.
- Word 0 with bit 1 clear, if the virtual terminal is in full-duplex mode, indicates that status is being returned for an offspring read request.
- Word 0 with bit 1 set, if the virtual terminal is in full-duplex mode, indicates that status is being returned for an offspring write request.

NOTE

If the virtual terminal is in half-duplex mode, bit 1 is ignored.

- Word 1 is the second word of I/O return status.
- Word 2 is the first word of I/O return status.

The status words are reversed in order to be similar to the format in which status must be passed back in a parent read or write function to an offspring task. The IO.STC function must be used to return status when no transfer is desired, since a byte count of 0 is not allowed in an IO.RLB or IO.WLB (read logical block and write logical block operations, respectively). For example, IE.EOF (write end-of-file tape mark), would normally be returned with IO.STC.

Note that it is important to specify an I/O completion status for all parent read and write requests that satisfy corresponding requests from the offspring task. If a return status is not specified, it defaults to zero. A zero indicates that the I/O is still pending (IS.PND). This causes the offspring task to hang if it examines the I/O status block to determine whether the I/O is completed.

In addition to returning status, the IO.STC function has an additional purpose. It can enable or disable intermediate buffering of I/O requests; note that a task cannot perform both IO.STC functions in the same I/O request. If bit 0 of the first parameter word in IO.STC is clear, bit 1 in this word is interpreted as a disable buffering flag:

- If bit 0 is clear and bit 1 is set, intermediate buffering of offspring I/O is disabled.
- If bit 0 is clear and bit 1 is clear, buffering is enabled.

Buffering cannot be enabled on a virtual terminal unit that has been created with an "mlen" parameter greater than the system-wide maximum specified at Sysgen time. An attempt to do both results in an error return of IE.IFC.

The only tasks that can assign LUNs to a virtual terminal unit are:

- The task that created the virtual terminal unit
- That task's offspring task(s), whose TI: is the virtual terminal unit

Attachment of a virtual terminal unit by an offspring task prevents the dequeuing of I/O requests to that unit from other offspring tasks; parent I/O requests are always serviced.

Both parent and offspring tasks can specify the I/O functions IO.GTS, SF.GMC, and SF.SMC. However, SF.GMC and SF.SMC support only a limited number of terminal characteristics for virtual terminals. Please refer to the RSX-11M/M-PLUS I/O Drivers Reference Manual for a list of valid characteristics.

Note that the parent task is not notified when the offspring issues any of the above directives.

When an offspring task issues a read-with-prompt request (IO.RPR), the virtual terminal driver separates the request into an IO.WLB request and an IO.RLB request. The parent task cannot issue an IO.RPR.

When a virtual terminal is in half-duplex mode, the virtual terminal driver handles only one offspring request at a time. For example, if the offspring task issues a read request and then issues a write request without waiting for the read to be completed, the driver queues the write request to be processed when the read is completed.

The parent task may issue an SF.SMC function to set the virtual terminal to full-duplex mode. In full-duplex mode, the write request in the previous example would be processed even if the previous read was not yet completed. If the parent task is at AST state, it will not receive notification of the I/O request.

Both parent and offspring tasks can issue an SF.GMC request to determine the mode of the virtual terminal. However, only the parent task can change the mode (using SF.SMC).

FORTRAN Call:

```
     CALL CRVT ([iiast],[ioast],[iaast],[imlen],iparm,[ids])

        iiast  =  AST address at which input requests from offspring
                  tasks are serviced

        ioast  =  AST address at which output requests from offspring
                  tasks are serviced

        iaast  =  AST address at which the parent task may be notified
                  of the completion of successful offspring attach and
                  detach requests to the virtual terminal unit
```

NOTE

At least one of the above optional parameters should be specified; otherwise, the virtual terminal created is treated as the null device.

imlen = Maximum buffer length allowed for offspring I/O requests

iparm = Address of 3-word buffer to receive information from the stack when an AST occurs

ids = Integer to receive the directive status word containing the virtual terminal number

Macro Call:

```
     CRVT$    [iast],[oast],[aast],[mlen]
```

iast = AST address at which input requests from offspring tasks are serviced. If iast=0, offspring input requests are rejected with IE.IFC returned.

oast = AST address at which output requests from offspring tasks are serviced. If oast=0, offspring output requests are rejected with IE.IFC returned.

aast = AST address at which the parent task may be notified of the completion of successful offspring attach and detach requests to the virtual terminal unit. If aast=0, no notification of offspring attach/detach is returned to the parent task.

NOTE

At least one of the above optional parameters should be specified; otherwise, the virtual terminal created is treated as the null device.

mlen = Maximum buffer length (bytes) allowed for offspring I/O requests (default and maximum values for this parameter are SYSGEN options).

Macro Expansion:

```
     CRVT$     IASTRU,OASTRU,PAST,20.
     .BYTE     149.,5          ;CRVT$ MACRO DIC, DPB SIZE=5 WORDS
     .WORD     IASTRU          ;INPUT REQUEST AST ROUTINE ADDRESS
     .WORD     OASTRU          ;OUTPUT REQUEST AST ROUTINE ADDRESS
     .WORD     PAST            ;SUCCESSFUL VT ATTACH NOTIFICATION AST
                               ;ROUTINE ADDRESS
     .WORD     20.             ;MAXIMUM BUFFER LENGTH = 20.BYTES
```

Local Symbol Definitions:

C.RVIA -- Input request AST routine address (2)

C.RVOA -- Output request AST routine address (2)

C.RVAA -- VT attach notification AST routine address (2)

C.RVML -- Maximum buffer length (2)

DSW Return Codes:

unit	--	Successful completion results in the return of the unit number of the created virtual terminal unit with the C bit clear.
IE.UPN	--	Insufficient dynamic memory to allocate the virtual terminal device unit data structures.
IE.HWR	--	Virtual terminal device driver not resident.
IE.ADP	--	Part of the DPB is out of the issuing task's address space.
IE.SDP	--	DIC or DPB size is invalid.

5.3.15 Cancel Time Based Initiation Requests

The Cancel Time Based Initiation Requests directive instructs the system to cancel all time-synchronized initiation requests for a specified task, regardless of the source of each request. These requests result from a Run directive, or from any of the time-synchronized variations of the MCR or DCL Run command.

In a multiuser protection system, a nonprivileged task can cancel time-based initiation requests only for a task with the same TI:.

FORTRAN Call:

```
     CALL CANALL (tsk[,ids])

       tsk  =  Task name

       ids  =  Directive status
```

Macro Call:

```
     CSRQ$   tsk

       tsk  =  Scheduled (target) task name
```

Macro Expansion:

```
     CSRQ$   ALPHA
     .BYTE   25.,3             ;CSRQ$ MACRO DIC, DPB SIZE=3 WORDS
     .RAD50  /ALPHA/           ;TASK "ALPHA"
```

Local Symbol Definitions:

C.SRTN -- Target task name (4)

DSW Return Codes:

IS.SUC	--	Successful completion.
IE.INS	--	Task is not installed.
IE.PRI	--	The issuing task is not privileged and is attempting to cancel requests made by another task.
IE.ADP	--	Part of the DPB is out of the issuing task's address space.
IE.SDP	--	DIC or DPB size is invalid.

Note:

If you specify an error routine address when using the $C or $S macro form, then you must include a null argument for RSX-11D compatibility. For example:

```
        CSRQ$S      #TNAME,,ERR      ;CANCEL REQUESTS FOR "ALPHA"
          .           .
          .           .
          .           .
        TNAME: .RAD50  /ALPHA/
```

DECL$

5.3.16 Declare Significant Event ($S Form Recommended)

The Declare Significant Event directive instructs the system to declare a significant event.

Declaration of a significant event causes the Executive to scan the Active Task List from the beginning, searching for the highest priority task that is ready to run. Use this directive with discretion to avoid excessive scanning overhead.

FORTRAN Call:

```
CALL DECLAR ([,ids])

  ids  =  Directive status
```

Macro Call:

```
DECL$S  [,err]

  err  =  Error routine address
```

Macro Expansion:

```
DECL$S   ,ERR           ;NOTE:  THERE IS ONE IGNORED ARGUMENT
MOV      (PC)+,-(SP)    ;PUSH DPB ONTO THE STACK
.BYTE    35.,1          ;DECL$S MACRO DIC, DPB SIZE=1 WORD
EMT      377            ;TRAP TO THE EXECUTIVE
BCC      .+6            ;BRANCH IF DIRECTIVE SUCCESSFUL
JSR      PC,ERR         ;OTHERWISE, CALL ROUTINE "ERR"
```

Local Symbol Definitions:

None

DSW Return Codes:

IS.SUC	--	Successful completion.
IE.ADP	--	Part of the DPB is out of the issuing task's address space.
IE.SDP	--	DIC or DPB size is invalid.

Note:

The $S form of the macro is recommended because this directive requires only a 1-word DPB.

5.3.17 Disable (or Inhibit) AST Recognition ($S Form Recommended)

The Disable (or Inhibit) AST Recognition directive instructs the system to disable recognition of ASTs for the issuing task. The ASTs are queued as they occur and are effected when the task reenables AST recognition. There is an implied disable AST recognition directive whenever an AST service routine is executing. When a task's execution is started, AST recognition is enabled. See Notes below.

FORTRAN Call:

```
CALL DSASTR  [(ids)]
   or
CALL INASTR  [(ids)]

  ids  =  Directive status
```

Macro Call:

```
DSAR$S  [err]

  err  =  Error routine address
```

Macro Expansion:

```
DSAR$S   ERR
MOV      (PC)+,-(SP)        ;PUSH DPB ONTO THE STACK
.BYTE    99.,1              ;DSAR$S MACRO DIC, DPB SIZE=1 WORD
EMT      377                ;TRAP TO THE EXECUTIVE
BCC      .+6                ;BRANCH IF DIRECTIVE SUCCESSFUL
JSR      PC,ERR             ;OTHERWISE, CALL ROUTINE "ERR"
```

Local Symbol Definitions:

None

DSW Return Codes:

IS.SUC	--	Successful completion.
IE.ITS	--	AST recognition is already disabled.
IE.ADP	--	Part of the DPB is out of the issuing task's address space.
IE.SDP	--	DIC or DPB size is invalid.

Notes:

1. This directive disables only the recognition of ASTs; the Executive still queues the ASTs. They are queued FIFO and will occur in that order when the task reenables AST recognition.

2. The FORTRAN calls, DSASTR (or INASTR) and ENASTR (see Section 5.3.24), exist solely to control the possible jump to the PWRUP routine (power-up). FORTRAN is not designed to link to a system's trapping mechanism. The PWRUP routine is strictly controlled by the system, which both accepts the trap and subsequently dismisses it. The FORTRAN program is notified by a jump to PWRUP but must use DSASTR (or INASTR) and ENASTR to ensure the integrity of FORTRAN data structures, most importantly the stack, during power-up processing.

3. Because this directive requires only a 1-word DPB, the $S form of the macro is recommended. It requires less space and executes with the same speed as that of the DIR$ macro.

5.3.18 Disable Checkpointing ($S Form Recommended)

The Disable Checkpointing directive instructs the system to disable the checkpointability of a task that has been installed as a checkpointable task. Only the affected task can issue this directive. A task cannot disable the ability of another task to be checkpointed.

FORTRAN Call:

```
CALL DISCKP [(ids)]

  ids  =  Directive status
```

Macro Call:

```
DSCP$S  [err]

  err  =  Error routine address
```

Macro Expansion:

```
DSCP$S   ERR
MOV      (PC)+,-(SP)       ;PUSH DPB ONTO THE STACK
.BYTE    95.,1             ;DSCP$S MACRO DIC, DPB SIZE=1 WORD
EMT      377               ;TRAP TO THE EXECUTIVE
BCC      .+6               ;BRANCH IF DIRECTIVE SUCCESSFUL
JSR      PC,ERR            ;OTHERWISE, CALL ROUTINE "ERR"
```

Local Symbol Definitions:

None

DSW Return Codes:

IS.SUC	--	Successful completion.
IE.ITS	--	Task checkpointing is already disabled.
IE.CKP	--	Issuing task is not checkpointable.
IE.ADP	--	Part of the DPB is out of the issuing task's address space.
IE.SDP	--	DIC or DPB size is invalid.

Notes:

1. When a checkpointable task's execution is started, checkpointing is enabled (that is, the task can be checkpointed).

2. Because this directive requires only a 1-word DPB, the $S form of the macro is recommended. It requires less space and executes with the same speed as that of the DIR$ macro.

DTRG$

5.3.19 Detach Region

The Detach Region directive detaches the issuing task from a specified, previously attached region. Any of the task's windows that are currently mapped to the region are automatically unmapped.

If RS.MDL is set in the region status word when the directive is issued, the task marks the region for deletion on the last detach. A task must be attached with delete access to mark a region for deletion.

FORTRAN Call:

```
     CALL DTRG (irdb[,ids])

       irdb  =  An 8-word integer array containing a Region Definition
                Block (see Section 3.5.1.2)

       ids   =  Directive status
```

Macro Call:

```
     DTRG$      rdb

       rdb  =  Region Definition Block address
```

Macro Expansion:

```
     DTRG$   RDBADR
     .BYTE   59.,2     ;DTRG$ MACRO DIC, DPB SIZE=2 WORDS
     .WORD   RDBADR    ;RDB ADDRESS
```

Region Definition Block Parameters:

Input parameters:

Array Element	Offset		
irdb(1)	R.GID	--	ID of the region to be detached
irdb(7)	R.GSTS	--	Bit settings[1] in the region status word:

Bit	Definition
RS.MDL	1 if the region should be marked for deletion when the last task detaches from it

---

1. If you are a FORTRAN programmer, refer to Section 3.5.1 to determine the bit values represented by the symbolic names described.

Output parameters:

irdb(7) R.GSTS -- Bit settings[1] in the region status word:

Bit	Definition
RS.UNM	1 if any windows were unmapped

Local Symbol Definitions:

D.TRBA -- Region Definition Block address (2)

DSW Return Codes:

IS.SUC -- Successful completion.

IE.PRI -- The task, which is not attached with delete access, has attempted to mark the region for deletion on the last detach, or the task has outstanding I/O (not necessarily to this region on RSX-11M systems only).

IE.NVR -- The task specified an invalid region ID or attempted to detach region 0 (its own task region).

IE.ADP -- Part of the DPD or RDB is out of the issuing task's address space.

IE.SDP -- DIC or DPB size is invalid.

1. If you are a FORTRAN programmer, refer to Section 3.5.1 to determine the bit values represented by the symbolic names described.

ELAW$

5.3.20 Eliminate Address Window

The Eliminate Address Window directive deletes an existing address window, unmapping it first if necessary. Subsequent use of the eliminated window's ID is invalid.

FORTRAN Call:

```
     CALL ELAW (iwdb[,ids])

       iwdb  =  An 8-word integer array containing a Window Definition
                Block (see Section 3.5.2.2)

       ids   =  Directive status
```

Macro Call:

```
     ELAW$   wdb

       wdb  =  Window Definition Block address
```

Macro Expansion:

```
     ELAW$   WDBADR
     .BYTE   119.,2          ;ELAW$ MACRO DIC, DPB SIZE=2 WORDS
     .WORD   WDBADR          ;WDB ADDRESS
```

Window Definition Block Parameters:

Input parameters:

Array Element	Offset		
iwdb(1) bits 0-7	W.NID	--	ID of the address window to be eliminated

Output parameters

iwdb(7)   W.NSTS   --   Bit settings[1] in the window status word:

Bit	Definition
WS.ELW	1 if the address window was successfully eliminated
WS.UNM	1 if the address window was unmapped

1. If you are a FORTRAN programmer, refer to Section 3.5.2 to determine the bit values represented by the symbolic names described.

Local Symbol Definitions:

E.LABA	--	Window Definition Block address (2)

DSW Return Codes:

IS.SUC	--	Successful completion.
IE.NVW	--	Invalid address window ID.
IE.ADP	--	Part of the DPB or WDB is out of the issuing task's address space.
IE.SDP	--	DIC or DPB size is invalid.

ELGF$

5.3.21 Eliminate Group Global Event Flags

The Eliminate Group Global Event Flags directive marks group-global event flags for deletion. If no tasks in this group are using the group-global event flags (the use count for this group maintained by the Executive in G.CNT is 0), the Group Global Event Flags Control Block (GFB) is immediately unlinked and deallocated. If tasks are using flags in this group, the Executive marks the flags for deletion (GS.DEL is set to 1) and the GFB is eliminated when no remaining tasks are using the flags in this group; however, if a Create Group Global Event Flags directive is issued before the flags are eliminated, the Executive clears GS.DEL.

If the specified group code matches the group code of the issuing task's protection UIC and the event flags are locked by this task (by a previous Create Group Global Event Flags directive), this directive unlocks the event flags by decrementing the access count. Note that a task may not unlock the event flags more than once in succession. Any attempt to unlock event flags that are already unlocked will return the IE.RSU error code.

FORTRAN Call:

```
     CALL ELGF ([group][,ids])
```

group	=	Group number of flags to be eliminated. Only privileged tasks can specify group numbers other than the issuing task's group UIC. If not specified, the task's protection UIC (H.CUIC+1) in the task's header is used.
ids	=	Integer to receive the Directive Status Word.

Macro Call:

```
     ELGF$    [group]
```

group	=	Group number of flags to be eliminated. Only privileged tasks can specify group numbers other than the issuing task's group UIC. If not specified, the task's protection UIC (H.CUIC+1) in the task's header is used.

Macro Expansion:

```
     ELGF$      303
     .BYTE      159.,2            ;ELGF$ MACRO DIC, DPB SIZE=2 WORDS
     .WORD      303               ;GROUP NUMBER 303 FLAGS
```

Local Symbol Definitions:

E.LGRP -- Group number (2)

DSW Return Codes:

IS.SUC --	Successful completion.
IE.PRI --	Privilege violation.
IE.IUI --	Invalid group (group>377 octal).
IE.IEF --	Group is not found.
IE.RSU --	Event flags are already marked for deletion.
IE.ADP --	Part of the DPB is out of the issuing task's address space.
IE.DIC --	DIC or DPB size is invalid.

ELVT$

5.3.22 Eliminate Virtual Terminal

The Eliminate Virtual Terminal directive causes the specified virtual terminal unit data structures to be marked for deallocation and eventually to be unlinked from the device list and deallocated. This directive can only be issued by the task that created the virtual terminal device unit. Any active nonprivileged tasks are aborted whose TI: device units are the virtual terminal being deallocated. TKTN messages reporting the abort of these tasks in this instance are directed to CO:. Any LUNs assigned by the issuing task, or by any offspring task being aborted, are deassigned.

A rundown count is maintained in the TCB of each parent task. This count reflects the total number of outstanding virtual terminal units the task has created, plus the number of connected (offspring) tasks. A series of ELVT$ directives are issued when a parent task, which has not eliminated virtual terminals it has created, exits. The virtual terminal data structures continue to exist until the last task exits whose TI: is the virtual terminal unit and until all CLI commands for that unit have been processed.

FORTRAN Call:

```
     CALL ELVT (iunum[,ids])

        iunum  =  Virtual terminal unit number

        ids    =  Integer to receive the Directive Status Word
```

Macro Call:

```
     ELVT$   unum

        unum   =  Unit number of the virtual terminal to be eliminated.
                  The task must provide this parameter after the
                  virtual terminal is created (see Note).
```

Macro Expansion:

```
     ELVT$      0
     .BYTE      151.,2          ;ELVT$ MACRO DIC, DPB SIZE=2 WORDS
     .WORD      0               ;VIRTUAL TERMINAL UNIT NUMBER
```

Local Symbol Definitions:

```
     E.LVNM  --  VT unit number (2)
```

DSW Return Codes:

```
     IS.SUC  --  Successful completion.

     IE.IDU  --  The specified virtual terminal unit does not exist or
                 it was not created by the issuing task.

     IE.ADP  --  Part of the DPB is out of the issuing task's address
                 space.

     IE.SDP  --  DIC or DPB size is invalid.
```

Note:

The actual virtual terminal unit number is not known until after the virtual terminal is actually created (that is, after successfully completing a Create Virtual Terminal directive). The Create Virtual Terminal directive DSW contains the actual virtual terminal unit number for use in the Eliminate Virtual Terminal directive. Thus, the task must save DSWs for all virtual terminals it creates, and later eliminate them using the Eliminate Virtual Terminal Directive.

EMST$

5.3.23 Emit Status

The Emit Status directive returns the specified 16-bit quantity to the specified connected task. It possibly sets an event flag or declares an AST if previously specified by the connected task in a Send, Request And Connect, a Spawn, or a Connect directive. If the specified task is multiply connected to the task issuing this directive, the first (oldest) Offspring control Block (OCB) in the queue is used to return status. If no task name is specified, this action is taken for all tasks that are connected to the issuing task at that time. In any case, whenever status is emitted to one or more tasks, those tasks no longer remain connected to the task issuing the Emit Status directive.

FORTRAN Call:

```
     CALL EMST ([rtname],status[,ids])
```

rtname	=	Name of a task connected to the issuing task to which the status is to be emitted
status	=	A 16-bit quantity to be returned to the connected task
ids	=	Integer to receive the Directive Status Word

Macro Call:

```
     EMST$    [tname],status
```

tname	=	Name of a task connected to the issuing task to which the status is to be emitted
status	=	16-bit quantity to be returned to the connected task

Macro Expansion:

```
     EMST$      ALPHA,STWD
     .BYTE      147.,4          ;EMST$ MACRO DIC, DPB SIZE=4 WORDS
     .RAD50     ALPHA           ;NAME OF CONNECTED TASK TO RECEIVE STATUS
     .WORD      STWD            ;VALUE OF STATUS TO BE RETURNED
```

Local Symbol Definitions:

E.MSTN	--	Task name (4)
E.MSST	--	Status to be returned (2)

DSW Return Codes:

IS.SUC	--	Successful completion.
IE.ITS	--	The specified task is not connected to the issuing task.
IE.ADP	--	Part of the DPB is out of the issuing task's address space.
IE.SDP	--	DIC or DPB size is invalid.

5.3.24 Enable AST Recognition ($S Form Recommended)

The Enable AST Recognition directive instructs the system to recognize ASTs for the issuing task; that is, the directive nullifies a Disable AST Recognition directive. ASTs that were queued while recognition was disabled are effected at issuance. When a task's execution is started, AST recognition is enabled.

FORTRAN Call:

```
CALL ENASTR  [(ids)]

  ids  =  Directive status
```

Macro Call:

```
ENAR$S  [err]

  err  =  Error routine address
```

Macro Expansion:

```
ENAR$S  ERR
MOV     (PC)+,-(SP)        ;PUSH DPB ONTO THE STACK
.BYTE   101.,1             ;ENAR$S MACRO DIC, DPB SIZE=1 WORD
EMT     377                ;TRAP TO THE EXECUTIVE
BCC     .+6                ;BRANCH IF DIRECTIVE SUCCESSFUL
JSR     PC,ERR             ;OTHERWISE, CALL ROUTINE "ERR"
```

Local Symbol Definitions:

None

DSW Return Codes:

IS.SUC	-- Successful completion.
IE.ITS	-- AST recognition is not disabled.
IE.ADP	-- Part of the DPB is out of the issuing task's address space.
IE.SDP	-- DIC or DPB size is invalid.

Notes:

1. Because this directive requires only a 1-word DPB, the $S form of the macro is recommended. It requires less space and executes with the same speed as that of the DIR$ macro.

2. The FORTRAN calls DSASTR (or INASTR) (see Section 5.3.17) and ENASTR exist solely to control the jump to the PWRUP routine (power-up). FORTRAN is not designed to link to a system's trapping mechanism. The PWRUP routine is strictly controlled by the system. It is the system that both accepts the trap and subsequently dismisses it. The FORTRAN program is notified by a jump to PWRUP but must use DSASTR (or INASTR) and ENASTR to ensure the integrity of FORTRAN data structures, most importantly the stack, during power-up processing.

ENCP$S

5.3.25 Enable Checkpointing ($S Form Recommended)

The Enable Checkpointing directive instructs the system to make the issuing task checkpointable after its checkpointability has been disabled; that is, the directive nullifies a DSCP$S directive. This directive cannot be used to enable checkpointing of a task that was built noncheckpointable.

FORTRAN Call:

```
     CALL ENACKP [(ids)]

       ids  =  Directive status
```

Macro Call:

```
     ENCP$S  [err]

       err  =  Error routine address
```

Macro Expansion:

```
     ENCP$S   ERR
     MOV      (PC)+,-(SP)       ;PUSH DPB ONTO THE STACK
     .BYTE    97.,1             ;ENCP$S MACRO DIC, DPB SIZE=1 WORD
     EMT      377               ;TRAP TO THE EXECUTIVE
     BCC      .+6               ;BRANCH IF DIRECTIVE SUCCESSFUL
     JSR      PC,ERR            ;OTHERWISE, CALL ROUTINE "ERR"
```

Local Symbol Definitions:

None

DSW Return Codes:

IS.SUC	--	Successful completion.
IE.ITS	--	Checkpointing is not disabled or task is connected to an interrupt vector.
IE.ADP	--	Part of the DPB is out of the issuing task's address space.
IE.SDP	--	DIC or DPB size is invalid.

Note:

Because this directive requires only a 1-word DPB, the $S form of the macro is recommended. It requires less space and executes with the same speed as that of the DIR$ macro.

5.3.26 Exit If

The Exit If directive instructs the system to terminate the execution of the issuing task if, and only if, an indicated event flag is not set. The Executive returns control to the issuing task if the specified event flag is set. See Notes below.

FORTRAN Call:

```
     CALL EXITIF (efn[,ids])

       efn  =  Event flag number

       ids  =  Directive status
```

Macro Call:

```
     EXIF$    efn

       efn  =  Event flag number
```

Macro Expansion:

```
     EXIF$  52.
     .BYTE   53.,2          ;EXIF$ MACRO DIC, DPB SIZE=2 WORDS
     .WORD   52.            ;EVENT FLAG NUMBER 52.
```

Local Symbol Definitions:

E.XFEF -- Event flag number (2)

DSW Return Codes:

IS.SET --	Indicated EFN set; task did not exit.
IE.IEF --	Invalid event flag number (EFN<1, or EFN>96, if group global event flags exist for the task's group; or EFN>64 if not).
IE.ADP --	Part of the DPB is out of the issuing task's address space.
IE.SDP --	DIC or DPB size is invalid.

Notes:

1. The Exit If directive is useful in avoiding a possible race condition that can occur between two tasks communicating by means of the Send and Receive directives. The race condition occurs when one task executes a Receive directive and finds its receive queue empty; but before the task can exit, the other task sends it a message. The message is lost because the Executive flushed the receiver task's receive queue when it decided to exit. This condition can be avoided if the sending task specifies a common event flag in the Send directive and the receiving task executes an Exit If specifying the same common event flag. If the event flag is set, the Exit If directive will return control to the issuing task, signaling that something has been sent.

2. A FORTRAN program that issues the Exit If call must first close all files by issuing Close calls. See the _IAS/RSX-11 FORTRAN IV or FORTRAN IV-PLUS User's Guide_ for instructions on how to ensure that such files are closed properly if the task exits. To avoid the time overhead involved in the closing and reopening of files, the task should first issue the appropriate test or clear event flag directive. If the directive status word indicates that the flag was not set, then the task can close all files and issue the call to Exit If.

3. On Exit, the Executive frees task resources. In particular, the Executive:

   - Detaches all attached devices
   - Flushes the AST queue and despecifies all specified ASTs
   - Flushes the receive and receive-by-reference queues
   - Flushes the clock queue for any outstanding Mark Time requests for the task
   - Closes all open files (files open for write access are locked)
   - Detaches all attached regions, except in the case of a fixed task
   - Runs down the task's I/O
   - Deaccesses the group global event flags for the task's group
   - Disconnects from interrupts
   - Flushes all outstanding CLI command buffers for the task
   - Breaks the connection with any offspring tasks
   - Returns a success status (EX$SUC) to any parent tasks
   - Marks for deallocation all virtual terminal units the task has created (see Section 5.2)
   - Frees the task's memory if the exiting task was not fixed

4. If the task exits, the Executive declares a significant event.

5.3.27 Task Exit ($S Form Recommended)

The Task Exit directive instructs the system to terminate the execution of the issuing task.

FORTRAN Call:

See Note 5 below.

Macro Call:

```
EXIT$S  [err]

  err  =  Error routine address
```

Macro Expansion:

```
EXIT$S  ERR
MOV     (PC)+,-(SP)      ;PUSH DPB ONTO THE STACK
.BYTE   51.,1            ;EXIT$S MACRO DIC, DPB SIZE=1 WORD
EMT     377              ;TRAP TO THE EXECUTIVE
JSR     PC,ERR           ;CALL ROUTINE "ERR"
```

Local Symbol Definitions:

None

DSW Return Codes:

IE.ADP	--	Part of the DPB is out of the issuing task's address space.
IE.SDP	--	DIC or DPB size is invalid.

Notes:

1. A return to the task occurs if, and only if, the directive is rejected. Therefore, no Branch on Carry Clear instruction is generated if an error routine address is given, since the return will only occur with carry set.

2. Exit causes a significant event to be declared.

3. On Exit, the Executive frees task resources. In particular, the Executive:

   - Detaches all attached devices
   - Flushes the AST queue and despecifies all specified ASTs
   - Flushes the receive and receive-by-reference queues
   - Flushes the clock queue for any outstanding Mark Time requests for the task
   - Closes all open files (files open for write access are locked)

- Detaches all attached regions, except in the case of a fixed task, where no detaching occurs
- Runs down the task's I/O
- Deaccesses the group global event flags for the task's group
- Disconnects from interrupts
- Flushes all outstanding CLI command buffers for the task
- Breaks the connection with any offspring tasks
- Returns a success code (EX$SUC) to any parent task
- Marks for deallocation all virtual terminal units the task has created (see Section 5.2)
- Frees the task's memory if the exiting task was not fixed

4. Because this directive requires only a 1-word DPB, the $S form of the macro is recommended. It requires less space and executes with the same speed as that of the DIR$ macro.

5. You can terminate FORTRAN tasks with the STOP statement or with CALL EXIT. CALL EXIT is a FORTRAN OTS routine that closes open files and performs other cleanup before it issues an EXIT$S directive (or an EXST$ directive in FORTRAN IV-PLUS). FORTRAN tasks that terminate with the STOP statement result in a message being displayed on the task's TI:. This message includes task name (as it appears in the Active Task List), the statement causing the task to stop, and an optional character string specified in the STOP statement. Tasks that terminate with CALL EXIT do not display a termination message. For example, a FORTRAN task containing the following statement:

```
20   STOP 'THIS FORTRAN TASK'
```

   exits with the following message displayed on the tasks TI: (TT37 in this example):

```
TT37 -- STOP THIS FORTRAN TASK
```

5.3.28 Exit With Status

The Exit With Status directive causes the issuing task to exit, passing a 16-bit status back to all tasks connected (by the Spawn, Connect, or Send, Request And Connect directive). If the issuing task has no connected tasks, then the directive simply performs a Task Exit. No format of the status word is enforced by the Executive; format conventions are a function of the cooperation between parent and offspring tasks. However, if an offspring task aborts for any reason, a status of EX$SEV is returned to the parent task. This value is interpreted as a "severe error" by Batch processors. Furthermore, if a task performs a normal exit with other tasks connected to it, a status of EX$SUC (successful completion) is returned to all connected tasks.

FORTRAN Call:

```
     CALL EXST (istat)

       istat    =  A 16-bit quantity to be returned to parent task
```

Macro Call:

```
     EXST$   status

       status   =  A 16-bit quantity to be returned to parent task
```

Macro Expansion:

```
     EXST$      STWD
     .BYTE      29.,2          ;EXST$ MACRO DIC, DPB SIZE=2 WORDS
     .WORD      STWD           ;VALUE OF STATUS TO BE RETURNED
```

Local Symbol Definitions:

E.XSTS -- Value of status to be returned (2)

DSW Return Codes:

No status is returned if the directive is successfully completed since the directive causes the issuing task to exit.

IE.ADP -- Part of the DPB is out of the issuing task's address space.

IE.SDP -- DIC or DPB size is invalid.

Notes:

1. The executive does the following to free a task's resources on Exit:

   - Detaches all attached devices
   - Flushes the AST queue and despecifies all specified ASTs
   - Flushes the Receive and Receive-by-reference queues
   - Flushes the clock queue for any outstanding Mark Time requests for the task
   - Closes all open files (files open for write access are locked)
   - Detaches all attached regions except in the case of a fixed task
   - Runs down the task's I/O
   - Deaccesses the group global event flags for the task's group
   - Disconnects from interrupts
   - Flushes all outstanding CLI command buffers for the task
   - Breaks the connection with any offspring tasks
   - Returns the specified exit status to any parent tasks
   - Marks for deallocation all the virtual terminal units that the task has created
   - Frees the task's memory if the exiting task was not fixed

2. If the task exits, the executive declares a significant event.

5.3.29 Extend Task

The Extend Task directive instructs the system to modify the size of the issuing task by a positive or negative increment of 32-word blocks. If the directive does not specify an increment value or specifies an increment value of zero, the Executive makes the issuing task's size equal to its installed size. The issuing task must be running in a system-controlled partition and cannot have any outstanding I/O when it issues the directive. The task must also be checkpointable to increase its size; if necessary, the Executive checkpoints the task, and then returns the task to memory with its size modified as directed.

In a system that supports the memory management directives, the Executive does not change any current mapping assignments if the task has memory-resident overlays. However, if the task does not have memory-resident overlays, the Executive attempts to modify, by the specified number of 32-word blocks, the mapping of the task to its task region.

If the issuing task is checkpointable but has no preallocated checkpoint space available, a positive increment may require dynamic memory and extra space in a checkpoint file sufficient to contain the task.

There are several constraints on the size to which a task can extend itself using the Extend directive:

- No task can extend itself beyond the maximum size set by the MCR command SET /MAXEXT or DCL command SET EXTENSION_LIMIT or the size of the partition in which it is running. (See the _RSX-11M/M-PLUS MCR Operations Manual_ or the _RSX-11M/M-PLUS Command Language Manual_.)
- A task that does not have memory-resident overlays cannot extend itself beyond 32K minus 32 words.
- A task that has preallocated checkpoint space in its task image file cannot extend itself beyond its installed size.
- A task that has memory-resident overlays cannot reduce its size below the highest window in the task partition.

FORTRAN Call:

```
CALL EXTTSK ([inc][,ids])
```

inc	=	A positive or negative number equal to the number of 32-word blocks by which the task size is to be extended or reduced
ids	=	Directive status

Macro Call:

```
     EXTK$   [inc]

       inc  =  A positive or negative number equal to  the  number  of
               32-word blocks by which the task size is to be extended
               or reduced
```

Macro Expansion:

```
     EXTK$   40
     .BYTE   89.,3             ;EXTK$ MACRO DIC, DPB SIZE=3 WORDS
     .WORD   40                ;EXTEND INCREMENT, 40(8) BLOCKS (1K
                               ;WORDS)
     .WORD   0                 ;RESERVED WORD
```

Local Symbol Definitions:

E.XTIN	-- Extend increment (2)

DSW Return Codes:

IS.SUC	-- Successful completion.
IE.UPN	-- Insufficient dynamic memory, or insufficient space in a checkpoint file.
IE.ITS	-- The issuing task is not running in a system controlled partition.
IE.ALG	-- The issuing task attempted to reduce its size to less than the size of its task header; or the task tried to increase its size beyond 32K words or beyond the maximum set by the MCR SET /MAXEXT command or the DCL SET EXTENSION_LIMIT command; or the task tried to increase its size to the extent that one virtual address window would overlap another; or the task has memory-resident overlays and it attempted to reduce its size below the highest window mapped to the task partition.
IE.RSU	-- Other tasks are attached to this task partition.
IE.IOP	-- I/O is in progress for this task partition.
IE.CKP	-- The issuing task is not checkpointable and specified a positive integer.
IE.NSW	-- Attempt to extend to larger than installed size (when checkpoint space is allocated in the task).
IE.ADP	-- Part of the DPB is out of the issuing task's address space.
IE.SDP	-- DIC or DPB size is invalid.

5.3.30 Get Command for Command Interpreter

The Get Command for Command Interpreter directive instructs the system to retrieve a command buffer for a Command Line Interpreter (CLI) task and copy it to a buffer in the task's address space. Information about the issuing terminal can also be returned to the CLI task.

Only CLI tasks can issue this directive.

FORTRAN Call:

```
CALL GTCMCI (icbf,icbfl,[iibuf],[iibfl],[iaddr],[incp][,ids])
```

icbf = Name of a byte array to receive the command

icbfl = Integer containing the size of the icbf array in bytes

iibuf = Name of an integer array to receive the optional information buffer

iibfl = Name of an integer containing the length of the optional information buffer. If you specify a length shorter than the information buffer, as much information as will fit in the length you specified is returned.

iaddr = Name of an integer that contains the address in pool of the command desired. This address was obtained by a previous call to GTCMCI with GC.CND specified.

incp = Name of an integer containing a bit mask indicating the action to take if there is no command queued:

Bit	Octal Value	Definition
GC.CCS	000	Return with carry set (default).
GC.CEX	001	Force CLI to exit instead of returning.
GC.CST	002	Force CLI to stop instead of returning.
GC.CND	200	Copy command into buffer but do not dequeue it from the list.

You must specify these as decimal values in your FORTRAN program.

ids = Integer to receive the Directive Status Word

Macro Call:

```
GCCI$ cbuf,cbfl,[ibuf],[ibfl],[addr],[ncp]
```

cbuf = Address of buffer to receive command string

cbfl	= Length of buffer. Maximum buffer size is 91. for RSX-11M and 266. for RSX-11M-PLUS.
ibuf	= Address of buffer to receive information on the issuing terminal
ibfl	= Length of buffer to receive information
addr	= Address of command. This address is returned in G.CCCA of the information buffer if GC.CND is specified in the ncp argument. If this argument is nonzero then only the command with the address specified by this argument is copied and/or dequeued. Note that this address is only filled in if the command is not dequeued.
ncp	= Action to take if no command buffer present:

Bit	Octal Value	Definition
GC.CCS	000	Return with carry set (default).
GC.CEX	001	Force CLI to exit instead of returning.
GC.CST	002	Force CLI to stop instead of returning.
GC.CND	200	Copy command into buffer but do not dequeue it from the list.

NOTE

GC.CND can be supplied with one of the other options, for example, GC.CND!GC.CEX.

Command Buffer Format:

G.CCDV	--	ASCII device name of issuing terminal (2)
G.CCCT	--	Number of characters (1)
G.CCUN	--	Octal unit number of issuing terminal (1)
G.CCCL	--	Number of characters in command line (2)
G.CCFL	--	Flags (1)

The values returned in the flag byte, G.CCFL, are:

Flag	Value	Definition
GC.CNL =	1	Null command line
GC.CTE =	2	Prompt from a task exit

G.CCTC -- Terminator (1)

G.CCBF -- Command text in ASCII (80 byte on RSX-11M systems, 256 bytes on RSX-11M-PLUS systems)

Information Buffer Format:

The format of the information buffer in the CLI address task space is:

G.CCW2 -- U.CW2 of issuing terminal (2)

G.CCPT -- Name of parent task (if any) (4)

G.CCOA -- Address of offspring control block from parent (2)

G.CCPU -- Login UIC of issuing terminal (2)

G.CCCU -- Current UIC of issuing terminal (2)

G.CCCA -- Address of command, if not dequeued (2)

Macro expansion:

```
GCCI$   CBUF,CBFL,IBUF,IBFL,ADDR,NCP
.BYTE   127.,7.             ;DIC= 127., DPB SIZE=7 WORDS
.BYTE   NCP                 ;ACTION TO TAKE IF NO COMMAND QUEUED
.BYTE   0
.WORD   ADDR                ;ADDRESS OF COMMAND
.WORD   CBUF                ;COMMAND BUFFER ADDRESS
.WORD   CBFL                ;COMMAND BUFFER LENGTH
.WORD   IBUF                ;INFORMATION BUFFER ADDRESS
.WORD   IBFL                ;INFORMATION BUFFER LENGTH
```

Local Symbol Definitions:

G.CCNC -- Action if no command queued (2)

G.CCAD -- Address of command to be returned (2)

G.CCBA -- Address of command buffer (2)

G.CCBL -- Length of task's command buffer (2)

G.CCIA -- Address of optional information buffer (2)

G.CCIL -- Length of optional information buffer (2)

DSW Return Codes:

IE.AST -- The stop-on-no-command option was set and the directive was issued from AST state.

IE.PRI -- Task is not a CLI.

IE.RSU -- The issuing task has a group global context active and next command to be received would have caused the task's protection group to have changed.

IE.ITS --	No command was queued for the CLI and the directive was issued with the return-with-carry-set option.
IS.CLR --	Returned with carry clear when the CLI was unstopped due to command arrival, after having been stopped by a GCII$ with the stop-on-no-command-option set.
IE.ADP --	DPB, send buffer or information buffer was outside the task's address space, or the information buffer was shorter than nine bytes.
IE.SDP --	DIC and DPB size is invalid.

Notes:

1. The number of characters returned (G.CCCT) could be less than the number of characters in the command (G.CCCL) if the length of the command buffer in the task, as specified by cbfl argument, is smaller than the actual command line. If there is sufficient room, a carriage return is placed at the end of the command line returned at G.CCBF in the command buffer inside the task to ease parsing.

2. If a command is successfully returned, the protection and default UICs for the issuing task are changed by this directive to match the UICs of the originating terminal. These UICs are returned in words G.CCPU and G.CCCU of the optional information buffer.

5.3.31 Get Command Interpreter Information

The Get Command Interpreter Information directive instructs the system to fill a buffer with information about a specified CLI or the CLI associated with a given terminal. A task must be privileged in order to issue this directive for any terminal other than its own TI:, or a CLI to which its TI: is not set.

FORTRAN Call:

```
CALL GETCII (ibuf,ibfl,[icli],[idev],[iunit][,ids])
```

ibuf	=	Name of an integer array to receive the CLI information
ibfl	=	Length in bytes of the integer array to receive the CLI information
icli	=	Name of a two-word array element containing the Radix-50 name of the CLI
idev	=	Name of an integer containing the ASCII name of terminal (default = TI:)
iunit	=	Name of an integer containing the Octal unit number of terminal
ids	=	Directive status

MACRO Call:

```
GCII$ buf,bufl,[cli],[dev,unit]
```

buf	=	Address of buffer to receive information
bufl	=	Length of information buffer
cli	=	Name in Radix-50 of the CLI that information is requested on
dev	=	ASCII name of terminal whose CLI should be used (default = TI:)
unit	=	Octal unit number of terminal

Information Buffer Format:

G.CICL -- Name of CLI

G.CICS -- Bit settings in the CLI status word:

Bit	Value	Definition
CP.NUL=	1	Pass empty command lines to CLI.
CP.MSG=	2	CLI wants system messages.
CP.LGO=	4	CLI wants commands from logged-off terminals.
CP.DSB=	10	CLI is disabled. (Note that MCR does not check this bit.)

Bit	Value	Definition
CP.PRV=	20	User must be privileged to set terminal to this CLI.
CP.SGL=	40	Don't handle continuations. Always set on RSX-11M systems.
CP.NIO=	100	MCR..., HEL, BYE do no I/O to terminal; HEL, BYE also do not set CLI, and so forth.
CP.RST=	200	Restricted access; only this CLI task can set a terminal to this CLI.
CP.EXT=	400	Pass task exit prompt requests to CLI.

G.CITK	--	Name of task serving as CLI
G.CIW2	--	Terminal's U.CW2
G.CIPU	--	Terminal's protection UIC
G.CICU	--	Terminal's current UIC
G.CIDP	--	CLI default prompt string (16-byte block. First byte is length of string.)

Macro Expansion:

```
GCII$ buf,bufl,cli,dev,unit

.BYTE   173,7           ;DIC to be supplied.  DPB SIZE=5
.WORD   buf             ;ADDRESS OF BUFFER
.WORD   bufl            ;LENGTH OF BUFFER
.RAD50  /cli/           ;RAD50 NAME OF CLI
.ASCII  /dev/           ;ASCII NAME OF TERMINAL
.WORD   unit            ;TERMINAL UNIT NUMBER
```

DSW Returns:

IE.MAP	--	Both a terminal and a CLI were specified.
IE.INS	--	Specified CLI does not exist.
IE.IDU	--	Specified device was not a terminal or does not exist.
IE.PRI	--	Nonprivileged task attempted to get information on a CLI other than it's own.
IE.ADP	--	Part of the DPB or buffer was out of the issuing task's address space.
IE.SDP	--	DIC or DPB size is invalid.

Notes:

1. If the buffer is not long enough to contain all the information, the data that does not fit will not be supplied. No indication of this is returned to the issuing task. The buffer is filled from left to right.

2. You may not specify both a CLI and a terminal. If the cli argument is present, the dev and unit arguments must be zero.

5.3.32 Get LUN Information

The Get LUN Information directive instructs the system to fill a 6-word buffer with information about a physical device unit to which a LUN is assigned. If requests to the physical device unit have been redirected to another unit, the information returned will describe the effective assignment.

FORTRAN Call:

```
CALL GETLUN (lun,dat[,ids])
```

lun = Logical unit number

dat = A 6-word integer array to receive LUN information

ids = Directive status

Macro Call:

```
GLUN$   lun,buf
```

lun = Logical unit number

buf = Address of 6-word buffer that will receive the LUN information

Buffer Format:

Word 0 -- Name of assigned device

Word _ -- Unit number of assigned device and flags byte (flags byte equals 200 if the device driver is resident or 0 if the driver is not loaded)

Word 2 -- First device characteristics word:

- Bit 0 -- Record-oriented device (DV.REC,1=yes) [FD.REC][1]
- Bit 1 -- Carriage-control device (DV.CCL,1=yes) [FD.CCL]
- Bit 2 -- Terminal device (DV.TTY,1=yes) [FD.TTY]
- Bit 3 -- Directory (file-structured) device (DV.DIR,1=yes) [FD.DIR]
- Bit 4 -- Single directory device (DV.SDI,1=yes) [FD.SDI]
- Bit 5 -- Sequential device (DV.SQD,1=yes) [FD.SQD]
- Bit 6 -- Mass storage device (DV.MSD,1=yes)
- Bit 7 -- User-mode diagnostics supported (DV.UMD,1=yes)

---

1. Bits with associated symbols have the symbols shown in square brackets. These symbols can be defined for use by a task by means of the FCSBT$ macro. See the IAS/RSX-11 I/O Operations Reference Manual.

Bit 8 --	Device supports extended 22-bit UNIBUS controller (DV.EXT,1=yes)
Bit 9 --	Unit software write-locked (DV.SWL,1=yes)
Bit 10 --	Input spooled device (DV.ISP,1=yes)
Bit 11 --	Output spooled device (DV.OSP,1=yes)
Bit 12 --	Pseudo device (DV.PSE,1=yes)
Bit 13 --	Device mountable as a communications channel (DV.COM,1=yes)
Bit 14 --	Device mountable as a Files-11 device (DV.F11,1=yes)
Bit 15 --	Device mountable (DV.MNT,1=yes)

Word 3 --	Second device characteristics word
Word 4 --	Third device characteristics word (words 3 and 4 are device driver specific)
Word 5 --	Fourth device characteristics word (normally buffer-size as specified in the MCR or DCL SET/BUFF command)

Macro Expansion:

```
GLUN$   7,LUNBUF
.BYTE   5,3             ;GLUN$ MACRO DIC, DPB SIZE=3 WORDS
.WORD   7               ;LOGICAL UNIT NUMBER 7
.WORD   LUNBUF          ;ADDRESS OF 6-WORD BUFFER
```

Local Symbol Definitions:

G.LULU --	Logical unit number (2)
G.LUBA --	Buffer address (2)

The following offsets are assigned relative to the start of the LUN information buffer:

G.LUNA --	Device name (2)
G.LUNU --	Device unit number (1)
G.LUFB --	Flags byte (1)
G.LUCW --	Four device characteristics words (8)

DSW Return Codes:

IS.SUC --	Successful completion.
IE.ULN --	Unassigned LUN.
IE.ILU --	Invalid logical unit number.
IE.ADP --	Part of the DPB or buffer is out of the issuing task's address space.
IE.SDP --	DIC or DPB size is invalid.

Note:

If a spooled device is found in the redirection chain and the issuing task is not the despooler, the LUN information returned by the Executive is as follows:

Word 0	--	Name of assigned (spooled) device
Word 1	--	Unit number of assigned spooled device and flags byte
Word 2	--	Logical OR of the first device characteristics word for the intermediate device and the output spool bit (spooled device first characteristics word, bit 11)
Word 3	--	Spooled device fourth device characteristics word
Word 4	--	Not defined
Word 5	--	Intermediate device standard device buffer size

GMCR$

5.3.33 Get MCR Command Line

The Get MCR Command Line directive instructs the system to transfer an 80-byte command line to the issuing task.

When a task is installed with a task name of "...tsk" or "tskTn", where "tsk" consists of three alphanumeric characters and n is an octal terminal number, the MCR dispatcher requests the task's execution when a user issues the command

```
>tsk command-line
```

from terminal number n. A task invoked in this manner must execute a call to Get MCR Command Line, which results in the entire "command line" following the prompt being placed into an 80-byte command line buffer. (The MCR dispatcher is described in the RSX-11M/M-PLUS MCR Operations Manual.)

FORTRAN Call:

```
CALL GETMCR (buf[,ids])
```

buf	=	An 80-byte array to receive command line
ids	=	Directive status

Macro Call:

```
GMCR$
```

Macro Expansion:

```
GMCR$
.BYTE   127.,41.        ;GMCR$ MACRO DIC, DPB SIZE=41.  WORDS
.BLKW   40.             ;80.  CHARACTER MCR COMMAND LINE BUFFER
```

Local Symbol Definitions:

G.MCRB -- MCR line buffer (80)

DSW Return Codes:

+n -- Successful completion; n is the number of data bytes transferred (excluding the termination character). The termination character is, however, in the buffer.

IE.AST -- No MCR command line exists for the issuing task; that is, the task was not requested by a command line as follows:

```
>tsk command-string
```

or the task has already issued the Get MCR Command Line directive.

IE.ADP -- Part of the DPB is out of the issuing task's address space.

IE.SDP -- DIC or DPB size is invalid.

Notes:

1. The GMCR$S form of the macro is not supplied, since the DPB receives the actual command line.

2. The system processes all lines to:

   - Convert tabs to a single space
   - Convert multiple spaces to a single space
   - Convert lowercase to uppercase
   - Remove all trailing blanks

   The terminator (<CR> or <ESC>) is the last character in the line.

3. On RSX-11M-PLUS systems, if the character before the terminator is a hyphen, there is at least one continuation line present. Therefore, you must issue another GMCR$ directive to obtain the rest of the command line.

GMCX$

5.3.34 Get Mapping Context

The Get Mapping Context directive causes the Executive to return a description of the current window-to-region mapping assignments. The returned description is in a form that enables the user to restore the mapping context through a series of Create Address Window directives (see Section 5.3.11). The macro argument specifies the address of a vector that contains one Window Definition Block (WDB) for each window block allocated in the task's header, plus a terminator word.

For each window block in the task's header, the Executive sets up a WDB in the vector as follows:

1. If the window block is unused (that is, if it does not correspond to an existing address window), the Executive does not record any information about that block in a WDB. Instead, the Executive uses the WDB to record information about the first block encountered that corresponds to an existing window. In this way, unused window blocks are ignored in the mapping context description returned by the Executive.

2. If a window block describes an existing unmapped address window, the Executive fills in the offsets W.NID, W.NAPR, W.NBAS, and W.NSIZ with information sufficient to re-create the window. The window status word W.NSTS is cleared.

3. If a window block describes an existing mapped window, the Executive fills in the offsets W.NAPR, W.NBAS, W.NSIZ, W.NRID, W.NOFF, W.NLEN, and W.NSTS with information sufficient to create and map the address window. WS.MAP is set in the status word (W.NSTS) and, if the window is mapped with write access, the bit WS.WRT is set as well.

Note that in no case does the Executive modify W.NSRB.

The terminator word, which follows the last WDB filled in, is a word equal to the negative of the total number of window blocks in the task's header. It is thereby possible to issue a TST or TSTB instruction to detect the last WDB used in the vector. The terminating word can also be used to determine the number of window blocks built into the task's header.

When Create Address Window directives are used to restore the mapping context, there is no guarantee that the same address window IDs will be used. The user must therefore be careful to use the latest window IDs returned from the Create Address Window directives.

FORTRAN Call:

```
     CALL GMCX (imcx[,ids])
```

imcx	=	An integer array to receive the mapping context. The size of the array is 8*n+1 where n is the number of window blocks in the task's header. The maximum size is 8*8+1=65 words on RSX-11M systems. The maximum size is 8*24+1=193 on RSX-11M-PLUS systems.
ids	=	Directive status.

Macro Call:

```
     GMCX$   wvec

       wvec  =  The address of a vector of n Window Definition Blocks,
                followed by a terminator word; n is the number of
                window blocks in the task's header.
```

Macro Expansion:

```
     GMCX$   VECADR
     .BYTE   113.,2              ;GMCX$ MACRO DIC, DPB SIZE=2 WORDS
     .WORD   VECADR              ;WDB VECTOR ADDRESS
```

Window Definition Block Parameters:

Input parameters:

None

Output parameters:

Array Element	**Offset**		
iwdb(1) bits 0-7	W.NID	--	ID of address window
iwdb(1) bits 8-15	W.NAPR	--	Base APR of the window
iwdb(2)	W.NBAS	--	Base virtual address of the window
iwdb(3)	W.NSIZ	--	Size, in 32-word blocks, of the window
iwdb(4)	W.NRID	--	ID of the mapped region, or no change if the window is unmapped
iwdb(5)	W.NOFF	--	Offset, in 32-word blocks, from the start of the region at which mapping begins, or no change if the window is unmapped
iwdb(6)	W.NLEN	--	Length, in 32-word blocks, of the area currently mapped within the region, or no change if the window is unmapped
iwdb(7)	W.NSTS	--	Bit settings[1] in the window status word (all 0 if the window is not mapped):

Bit	**Definition**
WS.MAP	1 if the window is mapped
WS.WRT	1 if the window is mapped with write access
WS.SIS	1 if the window is mapped in supervisor-mode instruction space

1. If you are a FORTRAN programmer, refer to Section 3.5.2 to determine the bit values represented by the symbolic names described.

Bit	Definition
WS.UDS	1 if the window is mapped in user-mode data space
WS.NBP	1 if the window was created with cache bypass disabled (on multiprocessor systems only)
WS.RCX	1 if cache bypass has been enabled for the current mapping of the window (on multiprocessor systems only)

Note that the length mapped (W.NLEN) can be less than the size of the window (W.NSIZ) if the area from W.NOFF to the end of the partition is smaller than the window size.

Local Symbol Definitions:

G.MCVA -- Address of the vector (wvec) containing the window definition blocks and terminator word (2)

DSW Return Codes:

IS.SUC -- Successful completion.

IE.ADP -- Address check of the DPB or the vector (wvec) failed.

IE.SDP -- DIC or DPB size is invalid.

Note:

Due to the use of the WS.RCX to indicate cache-bypass state, you may need to do additional manipulation of the WDB before you issue a CRAW$ or MAP$ directive. (On multiprocessor systems only.)

5.3.35 Get Partition Parameters

The Get Partition Parameters directive instructs the system to fill an indicated 3-word buffer with partition parameters. If a partition is not specified, the partition of the issuing task is assumed.

FORTRAN Call:

```
     CALL GETPAR ([prt],buf[,ids])

       prt  =  Partition name

       buf  =  A 3-word integer array to receive partition parameters

       ids  =  Directive status
```

Macro Call:

```
     GPRT$    [prt],buf

       prt  =  Partition name

       buf  =  Address of a 3-word buffer
```

Buffer Format:

Word 0 --	Partition physical base address expressed as a multiple of 32 words (partitions are always aligned on 32-word boundaries). Therefore, a partition starting at 40000(8) will have 400(8) returned in this word.
Word 1 --	Partition size expressed as a multiple of 32 words.
Word 2 --	Partition flags word. This word is returned equal to 0 to indicate a system-controlled partition, or equal to 1 to indicate a user-controlled partition.

Macro Expansion:

```
     GPRT$   ALPHA,DATBUF
     .BYTE   65.,4              ;GPRT$ DIC, DPB SIZE=4 WORDS
     .RAD50  /ALPHA/            ;PARTITION "ALPHA"
     .WORD   DATBUF             ;ADDRESS OF 3-WORD BUFFER
```

Local Symbol Definitions:

G.PRPN --	Partition name (4)
G.PRBA --	Buffer address (2)

The following offsets are assigned relative to the start of the partition parameters buffer:

G.PRPB --	Partition physical base address expressed as an absolute 32-word block number (2)
G.PRPS --	Partition size expressed as a multiple of 32-word blocks (2)
G.PRFW --	Partition flags word (2)

DSW Return Codes:

Successful completion is indicated by a cleared Carry bit, and the starting address of the partition is returned in the DSW. In unmapped systems, the address is physical. In mapped systems, the returned address is virtual and is always zero if it is not the task partition. Unsuccessful completion is indicated by a set Carry bit and one of the following codes in the DSW:

IE.INS --	Specified partition not in system.
IE.ADP --	Part of the DPB or buffer is out of the issuing task's address space.
IE.SDP --	DIC or DPB size is invalid.

Notes:

1. For Executives that support the memory management directives, a variation of this directive exists called Get Region Parameters (see Section 5.3.36). When the first word of the 2-word partition name is 0, the Executive interprets the second word of the partition name as a region ID. If the 2-word name is 0,0, it refers to the task region of the issuing task.

2. Omission of the partition-name argument returns parameters for the issuing task's unnamed subpartition, not for the system-controlled partition.

5.3.36 Get Region Parameters

The Get Region Parameters directive instructs the Executive to fill an indicated 3-word buffer with region parameters. If a region is not specified, the task region of the issuing task is assumed.

This directive is a variation of the Get Partition Parameters directive (see Section 5.3.35) for Executives that support the memory management directives.

FORTRAN Call:

```
CALL GETREG ([rid],buf[,ids])
```

rid	=	Region id
buf	=	A 3-word integer array to receive region parameters
ids	=	Directive status

Macro Call:

```
GREG$   [rid],buf
```

rid	=	Region ID
buf	=	Address of a 3-word buffer

Buffer Format:

Word 0	--	Region base address expressed as a multiple of 32 words (regions are always aligned on 32-word boundaries). Thus, a region starting at 1000(8) will have 10(8) returned in this word.
Word 1	--	Region size expressed as a multiple of 32 words.
Word 2	--	Region flags word. This word is returned equal to 0 if the region resides in a system-controlled partition, or equal to 1 if the region resides in a user-controlled partition.

Macro Expansion:

```
GREG$   RID,DATBUF
.BYTE   65.,4           ;GREG$ MACRO DIC, DPB SIZE=4 WORDS
.WORD   0               ;WORD THAT DISTINGUISHES GREG$
                        ;FROM GPRT$
.WORD   RID             ;REGION ID
.WORD   DATBUF          ;ADDRESS OF 3-WORD BUFFER
```

Local Symbol Definitions:

G.RGID	--	Region ID (2)
G.RGBA	--	Buffer address

The following offsets are assigned relative to the start of the region parameters buffer:

G.RGRB	--	Region base address expressed as an absolute 32-word block number (2)
G.RGRS	--	Region size expressed as a multiple of 32-word blocks (2)
G.RGFW	--	Region flags word (2)

DSW Return Codes:

Successful completion is indicated by carry clear, and the starting address of the region is returned in the DSW. In unmapped systems, the returned address is physical. In mapped systems, the returned address is virtual and is always zero if it is not the task region. Unsuccessful completion is indicated by carry set and one of the following codes in the DSW:

IE.NVR	--	Invalid region ID.
IE.ADP	--	Part of the DPB or buffer is out of the issuing task's address space.
IE.SDP	--	DIC or DPB size is invalid.

5.3.37 Get Sense Switches ($S Form Recommended)

The Get Sense Switches directive instructs the system to obtain the contents of the console switch register and store it in the issuing task's Directive Status word.

FORTRAN Call:

```
     CALL READSW (isw)

       isw  =  Integer to receive the console switch settings
```

The following FORTRAN call allows a program to read the state of a single switch:

```
     CALL SSWTCH    (ibt,ist)

       ibt  =  The switch to be tested (0 to 15)

       ist  =  Test results where:

               1  =  switch on

               2  =  switch off
```

Macro Call:

```
     GSSW$S  [err]

       err  =  Error routine address
```

Macro Expansion:

```
     GSSW$S  ERR
     MOV     (PC)+,-(SP)      ;PUSH DPB ONTO THE STACK
     .BYTE   125.,1           ;GSSW$S MACRO DIC, DPB SIZE=1 WORD
     EMT     377              ;TRAP TO THE EXECUTIVE
     BCC     .+6              ;BRANCH IF DIRECTIVE SUCCESSFUL
     JSR     PC,ERR           ;OTHERWISE, CALL ROUTINE "ERR"
```

Local Symbol Definitions:

None

DSW Return Codes:

Successful completion is indicated by carry clear, and the contents of the console switch register are returned in the DSW. Unsuccessful completion is indicated by carry set and one of the following codes in the DSW:

IE.ADP -- Part of the DPB is out of the issuing task's address space.

IE.SDP -- DIC or DPB size is invalid.

Note:

1. Because this directive requires only a 1-word DPB, the $S form of the macro is recommended. It requires less space and executes with the same speed as that of the DIR$ macro.

2. On multiprocessor systems, the value returned is that of the virtual switch register maintained by the MCR SWR command.

5.3.38 Get Time Parameters

The Get Time Parameters directive instructs the system to fill an indicated 8-word buffer with the current time parameters. All time parameters are delivered as binary numbers. The value ranges (in decimal) are shown in the table below.

FORTRAN Call:

```
     CALL GETTIM (ibfp[,ids])

       ibfp = An 8-word integer array
```

Macro Call:

```
     GTIM$   buf

       buf  =  Address of 8-word buffer
```

Buffer Format:

Word 0	--	Year (since 1900)
Word 1	--	Month (1-12)
Word 2	--	Day (1-31)
Word 3	--	Hour (0-23)
Word 4	--	Minute (0-59)
Word 5	--	Second (0-59)
Word 6	--	Tick of second (depends on the frequency of the clock)
Word 7	--	Ticks per second (depends on the frequency of the clock)

Macro Expansion:

```
     GTIM$   DATBUF
     .BYTE   61.,2             ;GTIM$ DIC, DPB SIZE=2 WORDS
     .WORD   DATBUF            ;ADDRESS OF 8.-WORD BUFFER
```

Local Symbol Definitions:

```
     G.TIBA  --  Buffer address (2)
```

The following offsets are assigned relative to the start of the time parameters buffer:

G.TIYR	--	Year (2)
G.TIMO	--	Month (2)
G.TIDA	--	Day (2)
G.TIHR	--	Hour (2)
G.TIMI	--	Minute (2)
G.TISC	--	Second (2)
G.TICT	--	Clock Tick of Second (2)
G.TICP	--	Clock Ticks per Second (2)

DSW Return Codes:

IS.SUC	--	Successful completion.
IE.ADP	--	Part of the DPB or buffer is out of the issuing task's address space.
IE.SDP	--	DIC or DPB size is invalid.

Note:

The format of the time buffer is compatible with that of the buffers used with the Set System Time directive.

5.3.39 Get Task Parameters

The Get Task Parameters directive instructs the system to fill an indicated 16-word buffer with parameters relating to the issuing task.

FORTRAN Call:

```
     CALL GETTSK (buf[,ids])

       buf  =  A 16-word integer array to receive the task parameters

       ids  =  Directive status
```

Macro Call:

```
     GTSK$  buf

       buf  =  Address of a 16-word buffer
```

Buffer Format:

Word 0	--	Issuing task's name in Radix-50 (first half)
Word 1	--	Issuing task's name in Radix-50 (second half)
Word 2	--	Partition name in Radix-50 (first half)
Word 3	--	Partition name in Radix-50 (second half)
Word 4	--	Undefined in RSX-11M/M-PLUS (this word exists for RSX-11D compatibility)
Word 5	--	Undefined in RSX-11M/M-PLUS (this word exists for RSX-11D compatibility)
Word 6	--	Run priority
Word 7	--	User identification code (UIC) of issuing task (in a multiuser protection system, the task's default UIC)[1]
Word 10	--	Number of logical I/O units (LUNs)
Word 11	--	Undefined in RSX-11M/M-PLUS (this word exists for RSX-11D compatibility)
Word 12	--	Undefined in RSX-11M/M-PLUS (this word exists for RSX-11D compatibility)
Word 13	--	(Address of task SST vector tables)[2]

1. See note in RQST$ description (Section 5.3.54) on contents of words 07 and 17.

2. Words 13 and 14 will contain valid data if word 14 is not zero. If word 14 is zero, the contents of word 13 are meaningless.

Word 14	--	(Size of task SST vector table in words)[2]
Word 15	--	Size (in bytes) either of task's address window 0 in mapped systems, or of task's partition in unmapped system (equivalent to partition size)
Word 16	--	System on which task is running:

0	for RSX-11D
1	for RSX-11M
2	for RSX-11S
3	for IAS
4	for RSTS
5	for VAX/VMS
6	for RSX-11M-PLUS
7	for RT11 single Job Monitor
10	for RT11 Foreground/Background and Extended Memory Monitor

Word 17	--	Protection UIC (in multiuser system, the log-in UIC)[1]

Macro Expansion:

```
     GTSK$   DATBUF
     .BYTE   63.,2          ;GTSK$ DIC, DPB=2-WORDS
     .WORD   DATBUF         ;ADDRESS OF 16-WORD BUFFER
```

Local Symbol Definitions

G.TSBA	--	Buffer address (2)

The following offsets are assigned relative to the task parameter buffer:

G.TSTN	--	Task name (4)
G.TSPN	--	Partition name (4)
G.TSPR	--	Priority (2)
G.TSGC	--	UIC group code (1)
G.TSPC	--	UIC member code (1)
G.TSNL	--	Number of logical units (2)
G.TSVA	--	Task's SST vector address (2)
G.TSVL	--	Task's SST vector length in words (2)
G.TSTS	--	Task size (2)
G.TSSY	--	System on which task is running (2)
G.TSDU	--	Protection UIC (2)

---

1. See note in RQST$ description (Section 5.3.54) on contents of words 07 and 17.

2. Words 13 and 14 will contain data if word 14 is not zero. If word 14 is zero, the contents of word 13 are meaningless.

DSW Return Codes:

IS.SUC	--	Successful completion.
IE.ADP	--	Part of the DPB or buffer is out of the issuing task's address space.
IE.SDP	--	DIC or DPB is invalid.

MAP$

5.3.40 **Map Address Window**

The Map Address Window directive maps an existing window to an attached region. The mapping begins at a specified offset from the start of the region. If the window is already mapped elsewhere, the Executive unmaps it before carrying out the mapping assignment described in the directive.

For the mapping asssignment, a task can specify any length that is less than or equal to both:

- The window size specified when the window was created
- The length remaining between the specified offset within the region and the end of the region

A task must be attached with write access to a region in order to map to it with write access. To map to a region with read-only access, the task must be attached with either read or write access.

If W.NLEN is set to 0, the length defaults to either the window size or the length remaining in the region, whichever is smaller. (Since the Executive returns the actual length mapped as an output parameter, the task must clear that parameter in the WDB before issuing the directive each time it wants to default the length of the map.)

The values that can be assigned to W.NOFF depend on the setting of bit WS.64B in the window status word (W.NSTS):

- If WS.64B = 0, the offset specified in W.NOFF must represent a multiple of 256 words (512 bytes). Because the value of W.NOFF is expressed in units of 32-word blocks, the value must be a multiple of 8.
- If WS.64B = 1, the task can align on 32-word boundaries; the programmer can therefore specify any offset within the region.

NOTE

Applications dependent on 32-word or 64-byte alignment (WS.64B = 1) may not be compatible with future implementations of RSX emulators. Therefore, programmers should write applications adaptable to either alignment requirement. The bit setting of WS.64B could be a parameter chosen at assembly time (by means of a prefix file), at task-build time (as input to the GBLDEF option), or at run time (by means of command input).

FORTRAN Call:

```
CALL MAP (iwdb[,ids])
```

iwdb	=	An 8-word integer array containing a Window Definition Block (see Section 3.5.2.2)
ids	=	Directive status

Macro Call:

```
     MAP$   wdb

       wdb  =  Window Definition Block address
```

Macro Expansion:

```
     MAP$    WDBADR
     .BYTE   121.,2            ;MAP$ MACRO DIC, DPB SIZE=2 WORDS
     .WORD   WDBADR            ;WDB ADDRESS
```

Window Definition Block Parameters:

Input parameters:

Array Element	Offset		
iwdb(1) bits 0-7	W.NID	--	ID of the window to be mapped.
iwdb(4)	W.NRID	--	ID of the region to which the window is to be mapped, or 0 if the task region is to be mapped.
iwdb(5)	W.NOFF	--	Offset, in 32-word blocks, within the region at which mapping is to begin. Note that if WS.64B in the window status word equals 0, the value specified must be a multiple of 8.
iwdb(6)	W.NLEN	--	Length, in 32-word blocks, within the region to be mapped, or 0 if the length is to default to either the size of the window or the space remaining in the region from the specified offset, whichever is smaller.
iwdb(7)	W.NSTS	--	Bit settings[1] in the window status word:

Bit	Definition
WS.WRT	1 if write access is desired
WS.64B	0 for 256-word (512-byte) alignment, or 1 for 32-word (64-byte) alignment

Output parameters:

iwdb(6)	W.NLEN	--	Length of the area within the region actually mapped by the window
iwdb(7)	W.NSTS	--	Bit settings[1] in the window status word:

WS.UNM -- 1 if the window was unmapped first

---

1. If you are a FORTRAN programmer, refer to Section 3.5.2 to determine the bit values represented by the symbolic names described.

Local Symbol Definitions:

M.APBA	--	Window Definition Block address (2)

DSW Return Codes:

IS.SUC	--	Successful completion.
IE.PRI	--	Privilege violation.
IE.NVR	--	Invalid region ID.
IE.NVW	--	Invalid address window ID.
IE.ALG	--	Task specified an invalid region offset and length combination in the Window Definition Block parameters; or WS.64B = 0 and the value of W.NOFF is not a multiple of 8.
IE.HWR	--	Region had a parity error or a load failure.
IE.ITS	--	WS.RES was set and region is not resident.
IE.ADP	--	Part of the DPB or WDB is out of the issuing task's address space.
IE.SDP	--	DIC or DPB size is invalid.

Notes:

1. When the Map Address Window directive is issued, the task may be blocked until the region is loaded.

2. Bit WS.RES in word W.NSTS of the Window Definition Block, when set, specifies that the region should be mapped only if the region is resident.

5.3.41 Mark Time

The Mark Time directive instructs the system to declare a significant event after an indicated time interval. The interval begins when the task issues the directive; however, task execution continues during the interval. If an event flag is specified, the flag is cleared when the directive is issued, and set when the significant event occurs. If an AST entry point address is specified, an AST (see Section 2.3.3) occurs at the time of the significant event. When the AST occurs, the task's PS, PC, directive status, Wait For mask words, and the event flag number specified in the directive are pushed onto the issuing task's stack. If neither an event flag number nor an AST service entry point is specified, the significant event still occurs after the indicated time interval. See Notes below.

FORTRAN Calls:

```
     CALL MARK (efn,tmg,tnt[,ids])

       efn  =  Event flag number

       tmg  =  Time interval magnitude (see Note 5)

       tnt  =  Time interval unit (see Note 5)

       ids  =  Directive status
```

The ISA standard call for delaying a task for a specified time interval is also provided:

```
     CALL WAIT (tmg,tnt[,ids])

       tmg  =  Time interval magnitude (see Note 5 below)

       tnt  =  Time interval unit (see Note 5 below)

       ids  =  Directive status
```

Macro Call:

```
     MRKT$   [efn],tmg,tnt[,ast]

       efn  =  Event flag number

       tmg  =  Time interval magnitude (see last Note below)

       tnt  =  Time interval unit (see last Note below)

       ast  =  AST entry point address
```

Macro Expansion:

```
     MRKT$    52.,30.,2,MRKAST
     .BYTE    23.,5              ;MRKT$ MACRO DIC, DPB SIZE=5 WORDS
     .WORD    52.                ;EVENT FLAG NUMBER 52.
     .WORD    30.                ;TIME MAGNITUDE=30.
     .WORD    2                  ;TIME UNIT=SECONDS
     .WORD    MRKAST             ;ADDRESS OF MARK TIME AST ROUTINE
```

Local Symbol Definitions:

M.KTEF	-- Event flag (2)
M.KTMG	-- Time magnitude (2)
M.KTUN	-- Time unit (2)
M.KTAE	-- AST entry point address (2)

DSW Return Codes:

For CALL MARK and MRKT$:

IS.SUC	-- Successful completion.
IE.UPN	-- Insufficient dynamic memory.
IE.ITI	-- Invalid time parameter.
IE.IEF	-- Invalid event flag number (EFN<0, or EFN>96 if group global event flags exist for the task's group; or EFN>64 if not).
IE.ADP	-- Part of the DPB is out of the issuing task's address space.
IE.SDP	-- DIC or DPB size is invalid.

For CALL WAIT:

RSX-11M/M-PLUS provides the following positive error codes to be returned for ISA calls:

1	-- Successful completion
2	-- Insufficient dynamic storage
3	-- Specified task not installed
94	-- Invalid time parameters
98	-- Invalid event flag number
99	-- Part of DPB out of task's range
100	-- DIC or DPB size invalid

Notes:

1. Mark Time requires dynamic memory for the clock queue entry.

2. If an AST entry point address is specified, the AST service routine is entered with the task's stack in the following state:

   ```
   SP+10 - Event flag mask word[1]
   SP+06 - PS of task prior to AST
   SP+04 - PC of task prior to AST
   SP+02 - DSW of task prior to AST
   SP+00 - Event flag number or zero (if none was
           specified in the Mark Time directive)
   ```

   The event flag number must be removed from the task's stack before an AST Service Exit directive (see Section 6.3.4) is executed.

3. If the directive is rejected, the specified event flag is not guaranteed to be cleared or set. Consequently, if the task indiscriminately executes a Wait For directive and the Mark Time directive is rejected, the task may wait indefinitely. Care should always be taken to ensure that the directive was successfully completed.

4. If a task issues a Mark Time directive that specifies a common or group global event flag and then exits before the indicated time has elapsed, the event flag is not set.

5. The Executive returns the code IE.ITI (or 94) in the Directive Status Word if the directive specifies an invalid time parameter. The time parameter consists of two components: the time interval magnitude and the time interval unit, represented by the arguments tmg and tnt, respectively.

   A legal magnitude value (tmg) is related to the value assigned to the time interval unit (tnt). The unit values are encoded as follows:

   For an ISA FORTRAN call (CALL WAIT):

   0 = Ticks. A tick occurs for each clock interrupt and is dependent on the type of clock installed in the system.

   For a line frequency clock, the tick rate is either 50 or 60 per second, corresponding to the power-line frequency.

   For a programmable clock, a maximum of 1000 ticks per second is available (the exact rate is determined at system generation time).

---

1. The event flag mask word preserves the Wait For conditions of a task prior to AST entry. A task can, after an AST, return to a Wait For state. Because these flags and the other stack data are in the user task, they can be modified. Such modification is strongly discouraged, however, since the task can easily fault on obscure conditions. For example, clearing the mask word results in a permanent Wait For State.

1 = Milliseconds. The subroutine converts the specified magnitude to the equivalent number of system clock ticks. On systems with line frequency clocks, millisecond Mark Time requests can only be approximations.

For all other FORTRAN and macro calls:

1 = Ticks. See definition of ticks above.

For both types of FORTRAN calls and all macro calls:

2 = Seconds

3 = Minutes

4 = Hours

The magnitude (tmg) is the number of units to be clocked. The following list describes the magnitude values that are valid for each type of unit. In no case can the value of tmg exceed 24 hours. The list applies to both FORTRAN and macro calls.

If tnt = 0, 1, or 2, tmg can be any positive value with a maximum of 15 bits.

If tnt = 3, tmg can have a maximum value of 1440(10).

If tnt = 4, tmg can have a maximum value of 24(10).

6. If the specified event flag is a group global, the use count for the event flag's group is incremented to prevent premature elimination of event flags. The use count is run down when:

- The Mark Time event occurs.
- The Mark Time event is cancelled.
- The issuing task exits with the Mark Time event still on the clock queue.

7. The minimum time interval is one tick. If you specify a time interval of zero, it will be converted to one tick.

5.3.42 Map Supervisor D-Space

The Map Supervisor D-Space directive allows the issuing task to change the mapping of its supervisor-mode D-space APRs. This directive also provides information about the current mapping of the task's supervisor-mode D-space APRs.

Tasks that do not use data space execute with instruction and data space overmapped. Tasks in which the Task Builder has separated instruction and data space are mapped separately (instruction and data space are not overmapped). The overall mapping structure for these tasks is as follows:

Window 0	Root I-space.
Window 1	Task header, stack and root D-space.
Window 2	I-space of the read-only section if a multiuser task. Memory resident overlays if not a multiuser task.
Window 3	D-space of the read-only section if a multiuser task. Memory resident overlays if not a multiuser task.
Window 4	Memory resident overlays.
.	.
.	.
.	.

When supervisor-mode library code is executing, the supervisor-mode I-space APRs map supervisor-mode instruction space. However, the supervisor-mode D-space APRs normally map user-mode data space. Code that resides in a supervisor-mode library can include data (such as error messages) within its own instruction space. The Map Supervisor D-Space directive allows such code to use the supervisor-mode D-Space APRs to map locations in supervisor-mode instruction space that contain data.

The Map Supervisor D-Space directive allows the issuing task to specify a 7-bit mask that determines the mapping of supervisor-mode D-space APRs. The mask value contains one bit for each APR starting with APR 1. The bits control the value stored in the supervisor mapping control byte in the task header (H.SMAP).

This mask is stored in the high byte of the parameter. The low byte of the parameter is ignored. Since the high bit of the PSW may be set, the PSW is returned in the low byte. The mask is returned in the high byte. Note that although there are eight APRs, the mask is only seven bits since APR 0 may not be changed. The mask position in the parameter is identical to the DSW return.

To provide for the case when a supervisor-mode library is being used by some tasks as a user-mode library, this directive does not change the task's mapping when it is issued from user mode. However, the DSW is still returned.

When the directive is successfully executed, the DSW provides information about the task's current mapping and mode. Specifying a negative mask value causes the directive to return information rather than change the mapping.

FORTRAN Call:

Not supported

Macro Call:

```
MSDS$   mask
```

mask = A 7-bit mask with one bit corresponding to each supervisor-mode D-space APR. If the bit is set, the APR is mapped to supervisor-mode I-space. If the bit is clear, the APR is mapped to user-mode D-space. The 7 bits are specified in bits 8 through 14 of the mask word.

Macro Expansion:

```
MSDS$ mask
.BYTE   201.,1          ;DIC=201.  DPB SIZE= 1 WORD
```

DSW Return Codes:

IE.ADP --	Part of the DPB is out of the issuing task's address space.
IE.SDP --	DIC or DPB size is invalid.

Notes:

1. When including data in a supervisor-mode library, the library may not overmap APR 0 with the supervisor-mode library. The executive assumes it has access to the task's DSW regardless of the mode from which a directive is issued. Data must therefore be placed near the end of the library, or mapped through a memory resident overlay to force its mapping into some APR other than 0.

2. In the following example, a supervisor-mode library routine changes its mapping in order to access an error message (which is data).

```
MESSAG: .ASCIZ          / ERROR IN INPUT DATA/
TST     (R0)            ; CHECK SOME PIECE OF USER DATA
BPL     10$             ; IF PLUS OK

MSDS$S  #100000         ; GET CURRENT STATUS OF MAPPING
MOV     $DSW,R0         ;
MOV     R0,-(SP)        ; SAVE CURRENT STATE FOR RESTORE
                        ; OF MAPPING STATE

BIS     #400,R0         ; UPDATE MASK TO MAP APR1 TO SUPER
                        ; MODE
MSDS$S  R0              ; MAP TO SUPER I SPACE

MOV     #MESSAG,R1      ; POINT TO ERROR MESSAGE
                        ; WHICH IS DATA
CALL    ERROR           ; ERROR IS A SUBROUTINE THAT
                        ; HAS LOCAL ERROR MESSAGES IN
                        ; A SUPERVISOR MODE LIBRARY

MOV     (SP)+,R0        ; GET OLD MAPPING STATUS
MSDS$S  R0              ; RESTORE OLD MAPPING STATUS

RETURN                  ; BACK TO USER
```

MVTS$

5.3.43 Move To/From User/Supervisor I/D-Space

The Move To/From User/Supervisor I/D-Space directive instructs the system to fetch data from a specified location in user-mode or supervisor-mode I-or D-space, or to write the specified value in the specified location in the specified type of address space.

This directive allows you to access a single word of I-space as data without creating a D-space window. This function is primarily intended for use by debugging aids. Use of this directive in production code is not recommended since the directive is not optimized for performance.

FORTRAN Call:

Not supported.

Macro Call:

```
MVTS$   action,addr,val
                   buff
```

action = One of the following:

```
MV.TUI -- Move to user I-space
MV.TUD -- Move to user D-space
MV.TSI -- Move to supervisor I-space
MV.TSD -- Move to supervisor D-space
MV.FUI -- Move from user I-space
MV.FUD -- Move from user D-space
MV.FSI -- Move from supervisor I-space
MV.FSD -- Move from supervisor D-space
```

addr = Address of the location in the task

buff = Buffer to receive the value fetched, for the move from operations

val = Value to be stored in the location, for the move to operations

Macro Expansion:

```
MVTS$   action,addr,val
.BYTE   203.,4          ;DIC 203., DPB SIZE= 4 WORDS
.WORD   action          ;THE OPERATION TO BE PERFORMED
.WORD   addr            ;ADDRESS OF THE TASK LOCATION
.WORD   val             ;VALUE TO BE WRITTEN (OR BUFFER IF MOVE FROM)
```

Local Symbol Definitions:

M.VTAC -- Action code

M.VTAD -- Address of location in I/D space to be moved to or from

M.VTBF -- Buffer address or
or
M.VTVA Value to be moved

DSW Return Codes:

IE.SDP --	DIC or DPB size is invalid.
IE.ADP --	Part of the DPB is out of the issuing task's address space; the specified address is not mapped; or the buffer or the target address is not in the issuing task's address space.
IE.PRI --	The issuing task does not have write access to the target address.

5.3.44 Queue I/O Request

The Queue I/O Request directive instructs the system to place an I/O request for an indicated physical device unit into a queue of priority-ordered requests for that device unit. The physical device unit is specified as a logical unit number (LUN) assigned to the device.

The Executive declares a significant event when the I/O transfer completes. If the directive call specifies an event flag, the Executive clears the flag when the request is queued and sets the flag when the significant event occurs.

The I/O status block is also cleared when the request is queued and is set to the final I/O status when the I/O request is complete. If an AST service routine entry point address is specified, the AST occurs upon I/O completion, and the task's Wait For mask word, PS, PC, DSW, and the address of the I/O status block are pushed onto the task's stack.

The description below deals solely with the Executive directive; the device-dependent information can be found in the *RSX-11M/M-PLUS I/O Drivers Reference Manual*. See Notes below.

FORTRAN Call:

```
     CALL QIO (fnc,lun,[efn],[pri],[isb],[prl][,ids])
```

fnc	=	I/O function code[1]
lun	=	Logical unit number
efn	=	Event flag number
pri	=	Priority; ignored, but must be present
isb	=	A 2-word integer array to receive final I/O status
prl	=	A 6-word integer array containing device-dependent parameters to be placed in parameter words 1 through 6 of the DPB. Fill in this array by using the GETADR routine (see Section 1.5.1.4).
ids	=	Directive status

Macro Call:

```
     QIO$   fnc,lun,[efn],[pri],[isb],[ast],[prl]
```

fnc	=	I/O function code[1]
lun	=	Logical unit number

1. I/O function code definitions are included in the *RSX-11M/M-PLUS I/O Drivers Reference Manual*.

efn	=	Event flag number
pri	=	Priority; ignored, but must be present
isb	=	Address of I/O status block
ast	=	Address of entry point of AST service routine
prl	=	Parameter list of the form <P1,...P6>

Macro Expansion:

```
QIO$    IO.RVB,7,52.,,IOSTAT,IOAST,<IOBUFR,512.>
.BYTE   1,12.             ;QIO$ MACRO DIC, DPB SIZE=12
.WORD   IO.RVB            ;FUNCTION=READ VIRTUAL BLOCK
.WORD   7                 ;LOGICAL UNIT NUMBER 7
.BYTE   52.,0             ;EFN 52., PRIORITY IGNORED
.WORD   IOSTAT            ;ADDRESS OF 2-WORD I/O STATUS BLOCK
.WORD   IOAST             ;ADDRESS OF I/O AST ROUTINE
.WORD   IOBUFR            ;ADDRESS OF DATA BUFFER
.WORD   512.              ;BYTE COUNT=512.
.WORD   0                 ;ADDITIONAL PARAMETERS...
.WORD   0                 ;...NOT USED IN...
.WORD   0                 ;...THIS PARTICULAR...
.WORD   0                 ;...INVOCATION OF QUEUE I/O
```

Local Symbol Definitions:

Q.IOFN	--	I/O function code (2)
Q.IOLU	--	Logical unit number (2)
Q.IOEF	--	Event flag number (1)
Q.IOPR	--	Priority (1)
Q.IOSB	--	Address of I/O status block (2)
Q.IOAE	--	Address of I/O done AST entry point (2)
Q.IOPL	--	Parameter list (6 words) (12)

DSW Return Codes:

IS.SUC	--	Successful completion.
IE.UPN	--	Insufficient dynamic memory.
IE.ULN	--	Unassigned LUN.
IE.HWR	--	Device driver not loaded.
IE.PRI	--	Task other than despooler attempted a write logical block operation.
IE.ILU	--	Invalid LUN.
IE.IEF	--	Invalid event flag number (EFN<0, or EFN>96 if group global event flags exist for the task's group; or EFN>64 if not).
IE.ADP	--	Part of the DPB or I/O status block is out of the issuing task's address space.
IE.SDP	--	DIC or DPB size is invalid.

Notes:

1. If the directive call specifies an AST entry point address, the task enters the AST service routine with its stack in the following state:

   ```
   SP+10 - Event flag mask word
   SP+06 - PS of task prior to AST
   SP+04 - PC of task prior to AST
   SP+02 - DSW of task prior to AST

   SP+00 - Address of I/O status block, or zero, if none
           was specified in the QIO directive
   ```

   The address of the I/O status block, which is a trap-dependent parameter, must be removed from the task's stack before an AST Service Exit directive (see Section 5.3.4) is executed.

2. If the directive is rejected, the specified event flag is not guaranteed to be cleared or set. Consequently, if the task indiscriminately executes a Wait For or Stop For directive and the QIO directive is rejected, the task may wait indefinitely. Care should always be taken to ensure that the directive was successfully completed.

3. Tasks (or regions on RSX-11M-PLUS systems) cannot normally be checkpointed with I/O outstanding for two reasons:

   - If the QIO directive results in a data transfer, the data transfers directly to or from the user-specified buffer.
   - If an I/O status block address is specified, the directive status is returned directly to the I/O status block.

   The Executive waits until a task has no outstanding I/O before initiating checkpointing in all cases except the one described below.

   In systems that support buffered I/O, drivers that buffer I/O check for the following conditions for a task:

   - That the task is checkpointable
   - That checkpointing is enabled

   If those two conditions are met, the driver and/or the Executive buffers the I/O request internally and the task is checkpointable with this outstanding I/O. If the task also entered a Wait For state when the I/O was issued (see the QIOW$ directive) or subsequently enters a Wait For state, the task is stopped. Any competing task waiting to be loaded into the partition can checkpoint the stopped task, regardless of priority. If the stopped task is checkpointed, the executive does not bring it back into memory until the stopped state is terminated by completion of buffered I/O or satisfaction of the Wait For condition.

   Not all drivers buffer I/O requests. The terminal driver is an example of one that does.

4. A privileged task on RSX-11M and any task on RSX-11M-PLUS that is linked to a common (read-only) area can issue QIO write requests to that area.

5. If the specified event flag is a group global, the use count for the event flag's group is incremented to prevent premature elimination of the event flags. The use count is run down when:

   - The I/O is completed.

   - The I/O is killed by reassigning the specified LUN with the ALUN$ directive.

   - The I/O is killed by issuing the IO.KIL function for the specified LUN.

   - The task exits before I/O is completed.

QIOW$

5.3.45 Queue I/O Request and Wait

The Queue I/O Request And Wait directive is identical to the Queue I/O Request in all but one aspect. If the Wait variation of the directive specifies an event flag, the Executive automatically effects a Wait For Single Event Flag directive. If an event flag is not specified, however, the Executive treats the directive as if it were a simple Queue I/O Request.

The following description lists the FORTRAN and macro calls with the associated parameters, as well as the macro expansion. Consult the description of Queue I/O Request for a definition of the parameters, the local symbol definitions, the DSW return codes, and explanatory notes.

FORTRAN Call:

```
CALL WTQIO (fnc,lun,[efn],[pri],[isb],[prl][,ids])
```

fnc	=	I/O function code[1]
lun	=	Logical unit number
efn	=	Event flag number
pri	=	Priority; ignored, but must be present
isb	=	A 2-word integer array to receive final I/O status
prl	=	A 6-word integer array containing device-dependent parameters to be placed in parameter words 1 through 6 of the DPB
ids	=	Directive status

Macro Call:

```
QIOW$   fnc,lun,[efn],[pri],[isb],[ast][,prl]
```

fnc	=	I/O function code[1]
lun	=	Logical unit number
efn	=	Event flag number
pri	=	Priority; ignored, but must be present
isb	=	Address of I/O status block
ast	=	Address of entry point of AST service routine
prl	=	Parameter list of the form <P1,...P6>

1. I/O function codes are defined in the _RSX-11M/M-PLUS I/O Drivers Reference Manual_.

Macro Expansion:

```
     QIOW$    IO.RVB,7,52.,,IOSTAT,IOAST,<IOBUFR,512.>
     .BYTE    3,12.              ;QIO$ MACRO DIC, DPB SIZE=12.
     .WORD    IO.RVB             ;FUNCTION=READ VIRTUAL BLOCK
     .WORD    7                  ;LOGICAL UNIT NUMBER 7
     .BYTE    52.,0              ;EFN 52., PRIORITY IGNORED
     .WORD    IOSTAT             ;ADDRESS OF 2-WORD I/O STATUS BLOCK
     .WORD    IOAST              ;ADDRESS OF I/O AST ROUTINE
     .WORD    IOBUFR             ;ADDRESS OF DATA BUFFER
     .WORD    512.               ;BYTE COUNT=512.
     .WORD    0                  ;ADDITIONAL PARAMETERS...
     .WORD    0                  ;...NOT USED IN...
     .WORD    0                  ;...THIS PARTICULAR...
     .WORD    0                  ;...INVOCATION OF QUEUE I/O
```

RCST$

5.3.46 Receive Data Or Stop

The Receive Data Or Stop directive instructs the system to dequeue a 13-word data block for the issuing task; the data block was queued for the task with a Send Data Directive or a Send, Request and Connect directive.

A 2-word task name of the sender (in Radix-50 format) and the 13-word data block are returned in an indicated 15-word buffer. The task name is contained in the first two words of the buffer.

If no data has been sent, the issuing task is stopped. In this case, the sender task is expected to issue an Unstop directive after sending data. A success status code of IS.SUC indicates that a packet has been received. A success status code of IS.SET indicates that the task was stopped and has been unstopped. The directive must then be reissued to retrieve the packet.

When a slave task issues the Receive Data or Stop directive, it assumes the UIC (if it has no outstanding group global context) and TI: terminal of the task that sent the data.

FORTRAN Call:

```
CALL RCST ([rtname],ibuf[,ids])
```

rtname	=	Sender task name (if not specified, data may be received from any task.)
ibuf	=	Address of 15-word buffer to receive the sender task name and data
ids	=	Integer to receive the directive status word

Macro Call:

```
RCST$    [tname],buf
```

tname	=	Sender task name (If not specified, data may be received from any task.)
buf	=	Address of 15-word buffer to receive the sender task name and data

Macro Expansion:

```
RCST$     ALPHA,TSKBUF
.BYTE     139.,4         ;RCST$ MACRO DIC, DPB SIZE=4 WORDS
.RAD50    ALPHA          ;DATA SENDER TASK NAME
.WORD     TSKBUF         ;BUFFER ADDRESS
```

Local Symbol Definitions:

R.CSTN -- Task name (4)

R.CSBF -- Buffer address (2)

DSW Return Codes:

IS.SUC	--	Successful completion.
IS.SET	--	No data was received and task was stopped (note that the task must be Unstopped before it can see this status).
IE.RSU	--	The issuing task is a slave task with a group global context active, and the next packet received would have changed the task's group number.
IE.AST	--	The issuing task is at AST state.
IE.ADP	--	Part of the DPB is out of the issuing task's address space.
IE.SDP	--	DIC or DPB size is invalid.

Note:

In RSX-11M-PLUS systems that support variable send and receive directives (secondary pool support SYSGEN option), the Receive Data Or Stop directive is treated as a 13. word Variable Receive Data Or Stop directive (see Section 5.3.88).

RCVD$

5.3.47 Receive Data

The Receive Data directive instructs the system to dequeue a 13-word data block for the issuing task; the data block has been queued (FIFO) for the task by a Send Data Directive.

A 2-word sender task name (in Radix-50 form) and the 13-word data block are returned in an indicated 15-word buffer, with the task name in the first two words.

When a slave task issues the Receive Data directive, it assumes the UIC (if it has no outstanding group global event flag context) and TI: terminal of the task that sent the data.

FORTRAN Call:

```
CALL RECEIV ([tsk],buf[,,ids])
```

tsk	=	Sender task name (If not specified, data may be received from any task.)
buf	=	A 15-word integer array for received data
ids	=	Directive status

Macro Call:

```
RCVD$   [tsk],buf
```

tsk	=	Sender task name (If not specified, data may be received from any task.)
buf	=	Address of 15-word buffer

Macro Expansion:

```
RCVD$     ALPHA,DATBUF      ;TASK NAME AND BUFFER ADDRESS
.BYTE     75.,4             ;RCVD$ MACRO DIC, DPB SIZE=4 WORDS
.RAD50    /ALPHA/           ;SENDER TASK NAME
.WORD     DATBUF            ;ADDRESS OF 15.-WORD BUFFER
```

Local Symbol Definitions:

R.VDTN	--	Sender task name (4)
R.VDBA	--	Buffer address (2)

DSW Return Codes:

IS.SUC	--	Successful completion.
IE.ITS	--	No data currently queued.
IE.RSU	--	The issuing task is a slave task with a group global context active, and the next packet to be received would have changed the task's group number.
IE.ADP	--	Part of the DPB or buffer is out of the issuing task's address space.
IE.SDP	--	DIC or DPB size is invalid.

Notes:

1. In RSX-11M-PLUS systems that support variable send and receive directives (secondary pool support SYSGEN option), the Receive Data directive is treated as a 13. word Variable Receive Data directive (see Section 5.3.87).

2. If the sending task specifies a common or group global event flag in the Send Data directive, the receiving task may use that event flag for synchronization. However, between the time that the receiver issues this directive and the time the receiver issues it's next instruction, the sender can send data and set the event flag. If the next instruction is an Exit directive, any data sent during this time will be lost because the Executive flushes the task's receive list as part of exit processing. Therefore, use the Exit If directive or the Receive Data or Exit directive in order to avoid the race condition.

RCVX$

5.3.48 Receive Data Or Exit

The Receive Data Or Exit directive instructs the system to dequeue a 13-word data block for the issuing task; the data block has been queued (FIFO) for the task by a Send Data directive.

A 2-word sender task name (in Radix-50 form) and the 13-word data block are returned in an indicated 15-word buffer, with the task name in the first two words.

If no data has been sent, a task exit occurs. To prevent the possible loss of Send packets, the user should not rely on I/O rundown to take care of any outstanding I/O or open files; the task should assume this responsibility.

When a slave task issues the Receive Data Or Exit directive, it assumes the UIC (if it has no outstanding group global event flag context) and TI: terminal of the task that sent the data. See Notes below.

FORTRAN Call:

```
     CALL RECOEX ([tsk],buf[,,ids])

       tsk  =  Sender task name (If not specified, data may be
               received from any task.)

       buf  =  A 15-word integer array for received data

       ids  =  Directive status
```

Macro Call:

```
     RCVX$    [tsk],buf

       tsk  =  Sender task name (If not specified, data may be
               received from any task.)

       buf  =  Address of 15-word buffer
```

Macro Expansion:

```
     RCVX$     ALPHA,DATBUF     ;TASK NAME AND BUFFER ADDRESS
     .BYTE     77.,4            ;RCVX$ MACRO DIC, DPB SIZE=4 WORDS
     .RAD50    /ALPHA/          ;SENDER TASK NAME
     .WORD     DATBUF           ;ADDRESS OF 15.-WORD BUFFER
```

Local Symbol Definitions:

```
     R.VXTN  =  Sender task name (4) R.VXBA  =  Buffer address (2)
```

DSW Return Codes:

IS.SUC	-- Successful completion.
IE.RSU	-- The issuing task is a slave task with a group global context active, and the next packet to be received would have changed the task's group number.
IE.ADP	-- Part of the DPB or buffer is out of the issuing task's address space.
IE.SDP	-- DIC or DPB size is invalid.

Notes:

1. A FORTRAN program that issues the RECOEX call must first close all files by issuing CLOSE calls. See the IAS/RSX-11 FORTRAN IV or the FORTRAN IV-PLUS User's Guide for instructions concerning how to ensure that such files are closed properly if the task exits.

   To avoid the time overhead involved in the closing and reopening of files, the task should first issue the RECEIV call. If the directive status indicates that no data were received, then the task can close all files and issue the call to RECOEX. The following example illustrates the same overhead saving in MACRO:

```
RCVBUF: .BLKW    15.              ; Receive buffer

START:  RCVX$C   ,RCVBUF          ; Attempt to receive message
        CALL     OPEN             ; Call user subroutine to open files.
PROC:            .
                 .
        Process  packet of data
                 .
                 .
        RCVD$C   ,RCVBUF          ; Attempt to receive another message
        BCC      PROC             ; If CC successful receive
        CALL     CLOSE            ; Call user subroutine to close files
                                  ; and prepare for possible task exit
        JMP      START            ; Make one last attempt at receiving
```

2. If no data have been sent, that is, if no Send Data directive has been issued, the task exits. Send packets may be lost if a task exits with outstanding I/O or open files (see third paragraph of this section).

3. The Receive Data Or Exit directive is useful in avoiding a possible race condition that can occur between two tasks communicating by the Send and Receive directives. The race condition occurs when one task executes a Receive directive and finds its receive queue empty; but before the task can exit, the other task sends it a message. The message is lost because the Executive flushes the receiver task's receive queue when it exits. This condition can be avoided by the receiving task's executing a Receive Data Or Exit directive. If the receive queue is found to be empty, a task exit occurs before the other task can send any data; thus, no loss of data can occur.

4. On Exit, the Executive frees task resources. In particular, the Executive:

   - Detaches all attached devices
   - Flushes the AST queue and despecifies all specified ASTs
   - Flushes the receive and receive-by-reference queues
   - Flushes the clock queue for outstanding Mark Time requests for the task
   - Closes all open files (files open for write access are locked)
   - Detaches all attached regions except in the case of a fixed task, where no detaching occurs
   - Runs down the task's I/O
   - Deaccesses the group global event flags for the task's group
   - Disconnects from interrupts
   - Flushes all outstanding CLI command buffers for the task
   - Returns a success status (EX$SUC) to any parent tasks
   - In RSX-11M-PLUS systems, marks for deallocation all virtual terminal units the task has created
   - Breaks the connection with any offspring tasks
   - Frees the task's memory if the exiting task was not fixed

5. If the task exits, the Executive declares a significant event.

6. In RSX-11M-PLUS systems that support variable send and receive directives (secondary pool support SYSGEN option), the Receive Data Or Exit directive is treated as a 13-word Variable Receive Data Or Exit directive (see Section 5.3.89).

5.3.49 Read All Event Flags

The Read All Event Flags directive inscructs the system to read all 64 event flags for the issuing task and record their polarity in a 64-bit (4-word) buffer.

NOTE

> This directive does not return group-global event flags (event flags 65 - 96).

FORTRAN Call:

A FORTRAN task can read only one event flag. The call is:

```
CALL READEF (efn[,ids])

  efn  =  Event flag number

  ids  =  Directive status
```

The Executive returns the status codes IS.SET (+02) and IS.CLR (00) for FORTRAN calls in order to report event flag polarity.

Macro Call:

```
RDAF$   buf
```

Buffer Format:

```
Word  0  --  Task Local Flags 1-16

Word  1  --  Task Local Flags 17-32

Word  2  --  Task Common Flags 33-48

Word  3  --  Task Common Flags 49-64
```

Macro Expansion:

```
RDAF$     FLGBUF
.BYTE     39.,2          ;RDAF$ MACRO DIC, DPB SIZE=2 WORDS
.WORD     FLGBUF         ;ADDRESS OF 4-WORD BUFFER
```

Local Symbol Definitions:

R.DABA -- Buffer address (2)

DSW Return Codes:

IS.SUC	--	Successful completion.
IE.ADP	--	Part of the DPB or buffer is out of the issuing task's address space.
IE.SDP	--	DIC or DPB size is invalid.

RDEF$

5.3.50 Read Event Flag

The Read Event Flag directive tests an indicated event flag and reports its polarity in the DSW.

FORTRAN Call:

```
     CALL READEF (iefn[,ids])

       iefn  =  Integer containing an Event Flag Number

       ids   =  Integer variable to receive the Directive Status Word
```

Macro Call:

```
     RDEF$ efn

       efn  =  Event flag number
```

Macro Expansion:

```
     RDEF$  6
     .BYTE  37.,2
     .WORD  6
```

Local Symbol Definitions:

The following symbol is locally defined with its assigned value equal to the byte offset from the start of the DPB to the DPB element:

R.DEEF -- Event flag number (length 2 bytes)

DSW Return Codes:

IS.CLR --	Flag was clear.
IS.SET --	Flag was set.
IE.IEF --	Invalid event flag number (event flag number <1 or >96.).
IE.ADP --	Part of DPB is out of issuing task's address space.
IE.SDP --	DIC or DPB size is invalid.

5.3.51 Read Extended Event Flags

The Read Extended Event Flags directive instructs the system to read all local, common, and group-global event flags for the issuing task and record their polarity in a 96-bit (6-word) buffer.

FORTRAN Call:

A FORTRAN task can read only one event flag. The call is:

CALL READEF (efn[,ids])

efn	=	Event flag number
ids	=	Directive status

The Executive returns the status codes IS.SET (+02) and IS.CLR (00) for FORTRAN calls to report event flag polarity.

Macro Call:

```
RDXF$   buf
```

Buffer Format:

Word 0	--	Task Local Flags 1-16
Word 1	--	Task Local Flags 17-32
Word 2	--	Task Common Flags 33-48
Word 3	--	Task Common Flags 49-64
Word 4	--	Task Group-Global Flags 65-80
Word 5	--	Task Group-Global Flags 81-96

Macro Expansion:

```
RDXF$     FLGBUF
.BYTE     39.,3          ;RDXF$ MACRO DIC, DPB SIZE=3 WORDS
.WORD     FLGBUF         ;ADDRESS OF 6-WORD BUFFER
```

Local Symbol Definitions:

R.DABA	--	Buffer address (2)

DSW Return Codes:

IS.SUC	--	Successful completion.
IS.CLR	--	Group-global event flags do not exist. Words 4 and 5 of the buffer contain 0.
IE.ADP	--	Part of the DPB or buffer is out of the issuing task's address space.
IE.SDP	--	DIC or DPB size is invalid.

RMAF$S

5.3.52 Remove Affinity ($S Form Recommended)

The Remove Affinity directive removes the task's CPU affinity that was previously established by issuing a Set Affinity directive. Note that only the $S form is available for this directive.

FORTRAN Call:

```
     CALL RMAF [(ids)]

       ids    =  Integer to receive Directive Status Word
```

Macro Call:

```
     RMAF$S
```

Macro Expansion:

```
     RMAF$S
     MOV       (PC)+,-(SP)     ;PUSH DPB ONTO THE STACK
     .BYTE     163.,1          ;RMAF$S MACRO DIC, DPB SIZE=1 WORD
     EMT       377             ;TRAP TO EXECUTIVE
```

Local Symbol Definitions:

None

DSW Return Codes:

IS.SUC	--	Successful completion.
IE.ADP	--	Part of the DPB is out of the issuing task's address space.
IE.SDP	--	DIC or DPB size is invalid.
IE.ITS	--	Task installed with affinity.

Note:

A task that is installed with task affinity must not issue this directive. Any attempt to do so results in an IE.ITS error returned.

5.3.53 Request and Pass Offspring Information

This directive instructs the system to request the specified task, and to chain to it by passing any or all of the parent connections from the issuing task to the requested task. Optionally, the directive can pass a command line to the requested task. Only a privileged or CLI task may specify the UIC and TI: of the requested task.

FORTRAN Call:

```
     CALL RPOI (tname,[iugc],[iumc],[iparen],[ibuf],[ibfl],[isc],
               [idnam],[iunit],[itask],[ocbad][,ids])
```

tname	=	Name of an array containing the actual name (in RAD50) of the task to be requested and optionally chained to.
iugc	=	Name of an integer containing the group code number for the UIC of the requested target chain task.
iumc	=	Name of the integer containing the member code number for the UIC of the requested target chain task.
iparen	=	Name of an array (or I*4 integer) containing the RAD50 name of the parent task. This is returned in the information buffer of the GTCMCI subroutine.
ibuf	=	Name of an array that contains the command line text for the chained task.
ibfl	=	Name of an integer that contains the number of bytes in the command in the ibuf array.
isc	=	Flag byte controlling the actions of this directive request when executed. The bit definitions of this byte (only the low order byte of the integer specified in the call is ever used) are as follows:

RP.OEX = 128.	Force this task to exit on successful execution of the RPOI directive.
RP.OAL = 1	Pass all of this task's connections to the requested task. (The default is none.)

NOTE

You cannot pass all connections if the target task is a CLI task.

RP.ONX = 2	Pass the first connection in the queue if there is one.

idnam = Name of an integer containing the ASCII device name of the requested task's TI:

iunit = Name of an integer containing the unit number of the requested tasks TI: device.

itask = Name of an array which contains the RAD50 name the requested task is to run under. On RSX-11M systems, this argument is valid only if the issuing task is a CLI task.

On RSX-11M-PLUS systems, any task may specify a new name for the requested task. However, the requested (specified in the tname parameter) task must be installed in the ...tsk format and must not be a CLI task.

ocbad = Name of an integer containing the pool address of the parent OCB. This value may only be obtained in the information buffer of the GTCMCI subroutine, which only a CLI can issue, so therefore, only a CLI can specify this argument.

ids = Name of an integer to receive the directive status word

Macro Call:

```
RPOI$ tname,,,[ugc],[umc],[parent],[bufadr],[buflen],[sc],[dnam],
     [unit],[task],[ocbad]
```

tname = Name of task to be chained to

ugc = Group code for UIC of the requested task

umc = Member code for UIC of the requested task

parent = Name of issuing task's parent task whose connection is to be passed. If not specified, all connections are passed.

bufadr = Address of buffer to be given to the requested task

buflen = Length of buffer to be given to requested task

sc = Flags byte:

RP.OEX	-- (200)	Force issuing task to exit
RP.OAL	-- (1)	Pass all connections (Default is none.)

NOTE

You cannot pass all connections if the target task is a CLI task.

RP.ONX	-- (2)	Pass the first connection in the queue, if there is one.

dnam	=	ASCII device name for TI:
unit	=	Unit number of task TI:
task	=	RAD50 name that the requested task is to run under. On RSX-11M systems, this parameter is only valid if the issuing task is a CLI task. On RSX-11M-PLUS systems, any task may specify a new name for the requested task. However, the requested (specified in the tname parameter) task must be installed in the ...tsk format and must not be a CLI task.
ocbad	=	Address of OCB to pass (CLIs only)

Local Symbol Definitions:

R.POTK	--	RAD50 name of task to be chained to (2)
R.POUM	--	UIC member code (1)
R.POUG	--	UIC group code
R.POPT	--	Name of parent whose OCB should be passed (4)
R.POOA	--	Address of OCB to pass (CLIs only) (2)
R.POBF	--	Address of command buffer (2)
R.POBL	--	Length of command (2)
R.POUN	--	Unit number of task TI: (1)
R.POSC	--	Flags byte (1)
R.PODV	--	ASCII device name for TI: (2)
R.POTN	--	RAD50 name of task to be started (4)

Macro Expansion:

```
RPOI$   tname,,,ugc,umc,ptsk,buf,buflen,sc,dev,unit,task,ocbad
.BYTE   11,16           ;DIC 11 DPB SIZE = 16. words
.RAD50  /tname/         ;NAME OF TASK TO CHAIN TO
.BLKW   3               ;RESERVED
.BYTE   umc             ;UIC MEMBER CODE
.BYTE   ugc             ;UIC GROUP CODE
.RAD50  ptsk            :NAME OF TASK WHOSE OCB SHOULD BE PASSED
.WORD   ocbad           ;ADDRESS OF OCB
.WORD   buf             ;ADDRESS OF BUFFER TO SEND
.WORD   buflen          ;LENGTH OF BUFFER
.ASCII  /dev/           ;ASCII NAME OF TI: OF REQUESTED TASK
.BYTE   unit            ;UNIT NUMBER OF TI: DEVICE
.BYTE   sc              ;PASS BUFFER AS SEND PACKET OR COMMAND
                        ;CODE
```

DSW Return Codes:

IE.UPN	--	Insufficient dynamic memory to allocate an offspring control block, command line buffer, task control block, or partition control block.
IE.INS	--	The specified task was not installed, or it was a CLI but no command line was specified.
IE.ACT	--	The specified task was already active and it was not a command line interpreter.
IE.IDU	--	The specified virtual terminal unit does not exist or was not created by the issuing task.
IE.ITS	--	A task that is not a CLI specified a CLI only parameter or specified passing all connections to a CLI.
IE.NVR	--	There is no offspring control block from the specified parent task.
IE.ALG	--	A CLI specified a parent name and an offspring control block address that did not describe the same connection, or either a parent name or an offspring control block address was specified and the pass all connections flag or the pass next connection flag was set.
IE.PNS	--	The task control block cannot be created in the same partition as its prototype.
IE.ADP	--	Part of the DPB, exit status block, or command line is out of the issuing task's address space.
IE.SDP	--	DIC or DPB size is invalid.

5.3.54 Request Task

The Request Task directive instructs the system to activate a task. The task is activated and subsequently runs contingent upon priority and memory availability. The Request Task directive is the basic mechanism used by running tasks to initiate other installed (dormant) tasks. The Request Task directive is a frequently used subset of the Run directive. See Notes below.

FORTRAN Call:

```
     CALL REQUES (tsk,[opt][,ids])

       tsk  =  Task name

       opt  =  A 4-word integer array

               opt(1)  =  Partition name  first  half;   ignored,  but
                          must be present

               opt(2)  =  Partition name second  half;   ignored,  but
                          must be present

               opt(3)  =  Priority;  ignored, but must be present

               opt(4)  =  User Identification Code

       ids  =  Directive status
```

Macro Call:

```
     RQST$   tsk,[prt],[pri][,ugc,umc]

       tsk  =  Task name

       prt  =  Partition name;  ignored, but must be present

       pri  =  Priority;  ignored, but must be present

       ugc  =  UIC group code

       umc  =  UIC member code
```

Macro Expansion:

```
     RQST$      ALPHA,,,20,10
     .BYTE      11.,7           ;RQST$ MACRO DIC, DPB SIZE=7 WORDS
     .RAD50     /ALPHA/         ;TASK "ALPHA"
     .WORD      0,0             ;PARTITION IGNORED
     .WORD      0               ;PRIORITY IGNORED
     .BYTE      10,20           ;UIC UNDER WHICH TO RUN TASK
```

Local Symbol Definitions:

R.QSTN	--	Task name (4)
R.QSPN	--	Partition name (4)
R.QSPR	--	Priority (2)
R.QSGC	--	UIC group (1)
R.QSPC	--	UIC member (1)

DSW Return Codes:

IS.SUC	--	Successful completion.
IE.UPN	--	Insufficient dynamic memory.
IE.INS	--	Task is not installed.
IE.ACT	--	Task is already active.
IE.ADP	--	Part of the DPB is out of the issuing task's address space.
IE.SDP	--	DIC or DPB size is invalid.

Notes:

1. The requested task must be installed in the system.

2. If the partition in which a requested task is to run is already occupied, the Executive places the task in a queue of tasks waiting for that partition. The requested task then runs, depending on priority, and resource availability, when the partition is free. Another possibility is that checkpointing may occur. If the current occupant(s) of the partition is checkpointable, has checkpointing enabled, and is of lower priority than the requested task, it is written to disk when its current outstanding I/O completes; the requested task is then read into the partition.

3. Successful completion means that the task has been declared active, not that the task is actually running.

4. The requested task acquires the same TI: terminal assignment as that of the requesting task.

5. The requested task always runs at the priority specified in its task header.

6. A task that executes in a system-controlled partition requires dynamic memory for the partition control block used to describe its memory requirements.

7. In a system that does not support multiuser protection, a task can be requested under any UIC, regardless of the UIC of the requesting task. If no UIC is specified in the request, the system uses the UIC from the task's header, which was specified at task-build time.

8. In a system that supports multiuser protection, each active task has two UICs: a protection UIC and a default UIC. These are both returned when a task issues a Get Task Parameters directive (GTSK$). The UICs are used in the following ways:

   - The protection UIC determines the task's access rights for opening files and attaching to regions. When a task attempts to open a file, the system compares the task's protection UIC against the protection mask of the specified UFD; the comparison determines whether the task is to be considered for system, owner, group, or world access.

   - The default UIC is used by the File Control Subroutines (FCS) to determine the default UFD when a file-open operation does not specify a UIC. (The default UIC has no significance when a task attaches to a region.)

   In a multiuser protection system, each terminal also has a protection UIC and a default UIC. If a terminal is nonprivileged, the protection UIC is the log-on UIC, and the default UIC is the UIC specified in the last SET /UIC command issued. If no SET /UIC command has been issued, the default UIC is equal to the log-on UIC. If the terminal is privileged, both the protection and the default UICs are equal either to the UIC specified in the last SET /UIC command or to the log-on UIC if a SET /UIC command has not been issued.

   The system establishes a task's UICs when the task is activated. In general, when the MCR Dispatcher or the MCR Run command activates a task, the task assumes the protection and default UICs of the issuing terminal. However, if the user specifies the /UIC keyword to the MCR or DCL Install or Run command, the specified UIC becomes the default UIC for the activated task; and if the issuing terminal is privileged, the specified UIC becomes the activated task's protection UIC as well.

   The system establishes UICs in the same manner when one task issues a Request directive to activate another task. The protection and default UICs of the issuing task generally become the corresponding UICs of the requested task. However, if a nonprivileged task specifies a UIC in a Request directive, the specified UIC becomes only the default UIC for the requested task. If a privileged task specifies a UIC in a Request directive, the specified UIC becomes both the protection and default UIC for the requested task.

RREF$

5.3.55 Receive By Reference

The Receive By Reference directive requests the Executive to dequeue the next packet in the receive-by-reference queue of the issuing (receiver) task. Optionally, the task will exit if there are no packets in the queue. The directive may also specify that the Executive proceed to map the region referred to.

If successful, the directive declares a significant event.

Each reference in the task's receive-by-reference queue represents a separate attachment to a region. If a task has multiple references to a given region, it is attached to that region the corresponding number of times. Because region attachment requires system dynamic memory, the receiver task should detach from any region that it was already attached to in order to prevent depletion of the memory pool. That is, the task needs to be attached to a given region only once.

If the Executive does not find a packet in the queue, and the task has set WS.RCX in the window status word (W.NSTS), the task exits. If WS.RCX is not set, the Executive returns the DSW code IE.ITS.

If the Executive finds a packet, it writes the information provided to the corresponding words in the Window Definition Block. This information provides sufficient information to map the reference, according to the sender task's specifications, with a previously created address window.

If the address of a 10-word receive buffer has been specified (W.NSRB in the Window Definition Block), then the sender task name and the eight additional words passed by the sender task (if any) are placed in the specified buffer. If the sender task did not pass on any additional information, the Executive writes in the sender task name and eight words of zero.

If the WS.MAP bit in the window status word has been set to 1, the Executive transfers control to the Map Address Window directive (see Section 6.3.40) to attempt to map the reference.

When a task that has unreceived packets in its receive-by-reference queue exits or is removed, the Executive removes the packets from the queue and deallocates them. Any related flags are not set.

FORTRAN Call:

```
     CALL RREF (iwdb,[isrb][,ids])
```

iwdb	=	An 8-word integer array containing a Window Definition Block (see Section 3.5.2.2)
isrb	=	A 10-word integer array to be used as the receive buffer. If the call omits this parameter, the contents of iwdb(8) are unchanged.
ids	=	Directive status

Macro Call:

```
     RREF$  wdb
```

wdb	=	Window Definition Block address

Macro Expansion:

```
     RREF$    WDBADR
     .BYTE    81.,2         ;RREF$ MACRO DIC, DPB SIZE=2 WORDS
     .WORD    WDBADR        ;WDB ADDRESS
```

Window Definition Block Parameters:

Input parameters:

Array Element	Offset		
iwdb(1) bits 0-7	W.NID	--	ID of an existing window if region is to be mapped
iwdb(7)	W.NSTS	--	Bit settings[1] in the window status word:
iwdb(8)	W.NSRB	--	Optional address of a 10-word buffer, to contain the sender task name and additional information

Bit settings for iwdb(7) W.NSTS (input):

Bit	Definition
WS.MAP	1 if received reference is to be mapped
WS.RCX	1 if task exit desired when no packet is found in the queue

Output parameters:

iwbd(4)	W.NRID	--	Region ID (pointer to attachment description)
iwdb(5)	W.NOFF	--	Offset word specified by sender task
iwdb(6)	W.NLEN	--	Length word specified by sender task
iwdb(7)	W.NSTS	--	Bit settings[1] in the window status word:

Bit	Definition
WS.RED	1 if attached with read access
WS.WRT	1 if attached with write access
WS.EXT	1 if attached with extend access
WS.DEL	1 if attached with delete access
WS.RRF	1 if receive was successful

The Executive clears the remaining bits.

1. If you are a FORTRAN programmer, refer to Section 3.5.2 to determine the bit values represented by the symbolic names described.

Local Symbol Definitions:

R.REBA	--	Window definition block address (2)

DSW Return Codes:

IS.SUC	--	Successful completion.
IS.HWR	--	Region has incurred a parity error.
IE.ITS	--	No packet found in the receive-by-reference queue.
IE.ADP	--	Address check of the DPB, WDB, or the receive buffer (W.NSRB) failed.
IE.SDP	--	DIC or DPB size is invalid.

5.3.56 Resume Task

The Resume Task directive instructs the system to resume the execution of a task that has issued a Suspend directive.

FORTRAN Call:

```
CALL RESUME (tsk[,ids])

  tsk  =  Task name

  ids  =  Directive status
```

Macro Call:

```
RSUM$   tsk

  tsk  =  Task name
```

Macro Expansion:

```
RSUM$    ALPHA
.BYTE    47.,3             ;RSUM$ MACRO DIC, DPB SIZE=3 WORDS
.RAD50   /ALPHA/           ;TASK "ALPHA"
```

Local Symbol Definitions:

R.SUTN	--	Task name (4)

DSW Return Codes:

IS.SUC	--	Successful completion.
IE.INS	--	Task is not installed.
IE.ACT	--	Task is not active.
IE.ITS	--	Task is not suspended.
IE.ADP	--	Part of the DPB is out of the issuing task's address space.
IE.SDP	--	DIC or DPB size is invalid.

RUN$

5.3.57 Run Task

The Run Task directive causes a task to be requested at a specified future time, and optionally to be requested periodically. The schedule time is specified in terms of delta time from issuance. If the smg, rmg, and rnt parameters are omitted, Run is the same as Request except that:

1. Run causes the task to become active one clock tick after the directive is issued.

2. The system always sets the TI: device for the requested task, to CO:.

See Notes below.

FORTRAN Call:

```
     CALL RUN (tsk,[opt],[smg],snt,[rmg],[rnt][,ids])

       tsk  =  Task name

       opt  =  A 4-word integer array

               opt(1)  =  Partition name first half; ignored, but
                          must be present

               opt(2)  =  Partition name second half; ignored,
                          but must be present

               opt(3)  =  Priority; ignored, but must be present

               opt(4)  =  User Identification Code

       smg  =  Schedule delta magnitude

       snt  =  Schedule delta unit (either 1, 2, 3, or 4)

       rmg  =  Reschedule interval magnitude

       rnt  =  Reschedule interval unit

       ids  =  Directive status
```

The ISA standard call for initiating a task is also provided:

```
     CALL START(tsk,smg,snt[,ids])

       tsk  =  Task name

       smg  =  Schedule delta magnitude

       snt  =  Schedule delta unit (either 0, 1, 2, 3, or 4)

       ids  =  Directive status
```

Macro Call:

```
     RUN$    tsk,[prt],[pri],[ugc],[umc],[smg],snt[,rmg,rnt]

       tsk  =  Task name

       prt  =  Partition name;  ignored, but must be present

       pri  =  Priority;  ignored, but must be present

       ugc  =  UIC group code

       umc  =  UIC member code

       smg  =  Schedule delta magnitude

       snt  =  Schedule delta unit (either 1, 2, 3, or 4)

       rmg  =  Reschedule interval magnitude

       rnt  =  Reschedule interval unit
```

Macro Expansion:

```
     RUN$      ALPHA,,,20,10,20.,3,10.,3
     BYTE      17.,11.           ;RUN$ MACRO DIC, DPB SIZE=11.  WORDS
     .RAD50    /ALPHA/           ;TASK "ALPHA"
     .WORD     0,0               ;PARTITION IGNORED
     .WORD     0                 ;PRIORITY IGNORED
     .BYTE     10,20             ;UIC TO RUN TASK UNDER
     .WORD     20.               ;SCHEDULE MAGNITUDE=20
     .WORD     3                 ;SCH. DELTA TIME UNIT=MINUTE (=3)
     .WORD     10.               ;RESCH. INTERVAL MAGNITUDE=10.
     .WORD     3                 ;RESCH. INTERVAL UNIT=MINUTE (=3)
```

Local Symbol Definitions:

```
     R.UNTN  --  Task name (4)

     R.UNPN  --  Partition name (4)

     R.UNPR  --  Priority (2)

     R.UNGC  --  UIC group code (1)

     R.UNPC  --  UIC member code (1)

     R.UNSM  --  Schedule magnitude (2)

     R.UNSU  --  Schedule unit (2)

     R.UNRM  --  Reschedule magnitude (2)

     R.UNRU  --  Reschedule unit (2)
```

DSW Return Codes:

For CALL RUN and RUN$:

IS.SUC	-- Successful completion.
IE.UPN	-- Insufficient dynamic memory.
IE.ACT	-- Multiuser task name specified.
IE.INS	-- Task is not installed.
IE.PRI	-- Nonprivileged task specified a UIC other than its own.
IE.ITI	-- Invalid time parameter.
IE.ADP	-- Part of the DPB is out of the issuing task's address space.
IE.SDP	-- DIC or DPB size is invalid.

For CALL START:

RSX-11M/M-PLUS provides the following positive error codes to be returned for ISA calls:

2	-- Insufficient dynamic storage.
3	-- Specified task not installed.
94	-- Invalid time parameter.
98	-- Invalid event flag number.
99	-- Part of DPB is out of task's address space.
100	-- DIC or DPB size is invalid.

Notes:

1. In a multiuser protection system, a nonprivileged task cannot specify a UIC that is not equal to its own protection UIC. (See Note 8, Section 5.3.54, for a definition of the protection UIC.) A privileged task can specify any UIC.

2. In a system that does not support multiuser protection, a task may be run under any UIC, regardless of the UIC of the requesting task. If no UIC is specified in the request, the Executive uses the default UIC from the requested task's header. The priority is always that specified in the requested task's Task Control Block.

3. The target task must be installed in the system.

4. If there is not enough room in the partition in which a requested task is to run, the Executive places the task in a queue of tasks waiting for that partition. The requested task will then run, depending on priority and resource availability, when the partition is free. Another possibility is that checkpointing will occur. If the current occupant(s) of the partition is checkpointable, has checkpointing enabled, is of lower priority than the requested task, or is stopped for terminal input, it will be written to disk when its current outstanding I/O completes. The requested task will then be read into the partition.

5. Successful completion means the task has been made active; it does not mean that the task is actually running.

6. Time Intervals

   The Executive returns the code IE.ITI in the DSW if the directive specifies an invalid time parameter. A time parameter consists of two components: the time interval magnitude, and the time interval unit.

   A legal magnitude value (smg or rmg) is related to the value assigned to the time interval unit snt or rnt. The unit values are encoded as follows:

   For an ISA FORTRAN call (CALL START):

   0 = Ticks -- A tick occurs for each clock interrupt and is dependent on the type of clock installed in the system.

   For a line frequency clock, the tick rate is either 50 or 60 per second, corresponding to the power-line frequency.

   For a programmable clock, a maximum of 1000 ticks per second is available (the exact rate is determined during system generation).

   1 = Milliseconds -- The subroutine converts the specified magnitude to the equivalent number of system clock ticks.

   For all other FORTRAN and all macro calls:

   1 = Ticks -- See definition of ticks above.

   For both types of FORTRAN calls and all macro calls:

   2 = Seconds

   3 = Minutes

   4 = Hours

The magnitude is the number of units to be clocked. The following list describes the magnitude values that are valid for each type of unit. In no case can the magnitude exceed 24 hours. The list applies to both FORTRAN and macro calls.

If unit = 0, 1, or 2, the magnitude can be any positive value with a maximum of 15 bits.

If unit = 3, the magnitude can have a maximum value of 1440(10).

If unit = 4, the magnitude can have a maximum value of 24(10).

7. The schedule delta time is the difference in time from the issuance of the RUN$ directive to the time the task is to be run. This time may be specified in the range from one clock tick to 24 hours.

8. The reschedule interval is the difference in time from task initiation to the time the task is to be reinitiated. If this time interval elapses and the task is still active, no reinitiation request will be issued. However, a new reschedule interval will be started. The Executive will continually try to start a task, wait for the specified time interval, and then restart the task. This process continues until a CSRQ$ (Cancel Time Based Initiation Requests) directive or an MCR or DCL Cancel command is issued.

9. Run requires dynamic memory for the clock queue entry used to start the task after the specified delta time. If the task is to run in a system-controlled partition, further dynamic memory is required for the task's dynamically allocated partition control block (PCB).

10. If optional rescheduling is not desired, then the macro call should omit the arguments rmg and rnt.

5.3.58 Specify Command Arrival AST

This directive instructs the system to enable or disable command arrival ASTs for the issuing CLI task. If command arrival ASTs are enabled, the executive transfers control to a specified address when commands have been queued to the CLI.

Only CLI tasks can use this AST.

The format of the stack when the AST routine is entered is as follows:

```
SP+10 - zero since no event flags are involved
SP+06 - PS of task prior to AST
SP+04 - PC of task prior to AST
SP+02 - DSW of task prior to AST
SP+00 - address of command buffer just queued
```

The AST routine must remove the command buffer address from the stack before issuing an ASTX$ directive.

The command buffer address may be used when issuing a GCCI$ directive.

FORTRAN Call:

Not supported.

Macro Call:

```
SCAA$  [ast]
```

ast = AST service routine entry point. Omitting this parameter disables command arrival ASTs for the issuing task until the directive is respecified.

Macro Expansion:

```
SCAA$   ast
.BYTE   173.,2          ;DIC = 173 , DPB SIZE = 2 WORDS
.WORD   ast             ;ADDRESS OF AST ROUTINE
```

Local Symbol Definitions:

S.CAAE -- Address of AST routine

DSW Return Codes:

IE.ITS --	ASTs are already not desired.
IE.AST --	Directive issued from AST state.
IE.PRV --	Issuing task is not a CLI.
IE.UPN --	Insufficient dynamic memory.
IE.ADP --	Part of the DPB was out of the issuing task's address space.
IE.SDP --	DIC or DPB size is invalid.

SCAL$S

5.3.59 Supervisor Call ($S Form Recommended)

The Supervisor Call directive is issued by a task in user mode or supervisor mode to call a supervisor-mode library routine. Returning to the user mode from supervisor mode routines entered with the SCAL$S directive (Macro form) is effected by a completion routine that is executed in supervisor mode. Note that only the $S form is available for this directive.

NOTE

> We strongly suggest using the taskbuilder to resolve references to supervisor mode routines rather than explicitly using the SCAL$S directive. Doing so allows you to take advantage of any new techniques implemented in future releases automatically.

FORTRAN Call:

Not Supported

Macro Call:

```
SCAL$S  saddr,caddr

  saddr  =  Address of the called supervisor-mode routine

  caddr  =  Address of the completion routine for return to the
            caller

  err    =  Address of error routine
```

Macro Expansion:

```
SCAL$S  SRAD,CRAD,ERR
MOV     CRAD,-(SP)
MOV     SRAD,-(SP)
MOV     (PC)+,-(SP)
.BYTE   155.,3
EMT     ^O<377>
BCC     .+6
CALL    ERR
```

Local Symbol Definitions:

None

DSW Return Codes:

IS.SUC	--	Successful completion.
IE.ADP	--	Part of the DPB is out of the issuing task's address space.
IE.SDP	--	DIC or DPB size is invalid.

Note:

This directive transfers control to the specified routine in supervisor mode with all registers preserved and with the following stack:

```
                                  +------------------------------+
Supervisor stack pointer -------->| Completion routine address   |
                                  +------------------------------+
                                  | PC+2 of Supervisor Call      |
                                  +------------------------------+
                                  | PS of Supervisor Call        |
                                  +------------------------------+
User stack pointer
```

The stack, as shown, represents the stack content immediately after issuing the Supervisor Call directive. The user stack pointer is not guaranteed to remain valid.

The supervisor stack is the user stack with three words pushed onto it. It is mapped in Supervisor Data Space along with the rest of the user mode mapping. Previous mode bits are set to the caller's mode. This is normally user mode, but it may be supervisor mode.

If there is insufficient stack space for the three words, the issuing task is aborted.

SCLI$

5.3.60 Set Command Line Interpreter

The Set Command Line Interpreter directive instructs the system to set up the specified CLI as the CLI for the indicated terminal. The issuing task must be privileged or a CLI.

If the restricted access flag (CP.RST) in the CLI status word is set, the issuing CLI task is the only CLI task that can set a terminal to that CLI.

FORTRAN Call:

```
     CALL SETCLI (icli,idev,iunit[,ids])
```

icli	=	Name of a two-word array element containing the name of the CLI the terminal is to be set to
idev	=	Name of an integer containing the ASCII name of the terminal to be set (default = TI:)
iunit	=	Name of an integer containing the unit number of terminal
ids	=	Directive status

Macro Call:

```
     SCLI$ cli,[dev],[unit]
```

cli	=	Name of the CLI the terminal is to be set to
dev	=	ASCII name of the terminal to be set (default = TI:)
unit	=	Unit number of terminal

Local Symbol Definitions:

S.CIDV	--	ASCII name of the terminal whose CLI is to be set
S.CIUN	--	Octal unit number of terminal
S.CICN	--	RAD50 name of the CLI that the terminal is to be set to

Macro Expansion:

```
     SCLI$   cli,dev,unit
     .BYTE   173.,5           ;DIC 173. DPB SIZE = 5 WORDS
     .RAD50  /cli/            ;CLI NAME
     .ASCII  /dev/            ;ASCII NAME OF TERMINAL TO BE SET
     .WORD   unit             ;UNIT NUMBER
```

DSW Returns:

IE.PRI	--	Task not privileged or not a CLI. If CP.RST was set, task was not the CLI itself.
IE.IDU	--	Device not a terminal or does not exist.
IE.INS	--	Specified CLI does not exist.
IE.UPN	--	Insufficient dynamic memory.
IE.ADP	--	Part of the DPB was out of the issuing task's address space.
IE.SDP	--	DIC or DPB length is invalid.

SDAT$

5.3.61 **Send Data**

The Send Data directive instructs the system to declare a significant event and to queue (FIFO) a 13-word block of data for a task to receive.

NOTE

> When a local event flag is specified, the indicated event flag is set for the sending task; a significant event is always declared.

FORTRAN Call:

```
     CALL SEND (tsk,buf,[efn][,ids])

       tsk  =  Task name

       buf  =  13-word integer array of data to be sent

       efn  =  Event flag number

       ids  =  Directive status
```

Macro Call:

```
     SDAT$    tsk,buf[,efn]

       tsk  =  Task name

       buf  =  Address of 13-word data buffer

       efn  =  Event flag number
```

Macro Expansion:

```
     SDAT$     ALPHA,DATBUF,52.
     .BYTE     71.,5      ;SDAT$ MACRO DIC, DPB SIZE=5 WORDS
     .RAD50    /ALPHA/    ;RECEIVER TASK NAME
     .WORD     DATBUF     ;ADDRESS OF 13.-WORD BUFFER
     .WORD     52.        ;EVENT FLAG NUMBER 52.
```

Local Symbol Definitions:

```
     S.DATN  --  Task name (4)

     S.DABA  --  Buffer address (2)

     S.DAEF  --  Event flag number (2)
```

DSW Return Codes:

IS.SUC --	Successful completion.
IE.INS --	Receiver task is not installed.
IE.UPN --	Isufficient dynamic memory.
IE.IEF --	Invalid event flag number (EFN<0, or EFN>96 if group global event flags exist for the task's group; or EFN>64 if not).
IE.ADP --	Part of the DPB or data block is out of the issuing task's address space.
IE.SDP --	DIC or DPB size is invalid.

Notes:

1. Send Data requires dynamic memory.

2. If the directive specifies a local event flag, the flag is local to the sender (issuing) task. RSX-11M does not allow one task to set or clear a flag that is local to another task.

   Normally, the event flag is used to trigger the receiver task into some action. For this purpose, the event flag must be common (33 through 64) or group global (65 through 96) rather than local. (Refer to the descriptions of the Receive Data directive and the Exit IF directive.)

3. In RSX-11M-PLUS systems the Send Data directive is treated as a 13-word Variable Send Data directive (see Section 5.3.90).

5.3.62 Send, Request and Connect

The Send, Request And Connect directive performs a Send Data to the specified task, Requests the task if it is not already active, and then Connects to the task. The receiver task normally returns status by an Emit Status or Exit With Status directive.

FORTRAN Call:

```
     CALL SDRC (rtname, ibuf,[iefn],[iast],[iesb],[iparm][,ids])
```

rtname	=	Target task name of the offspring task to be connected
ibuf	=	Name of 13-word send buffer
iefn	=	Event flag to be set when the offspring task exits or emits status
iast	=	Name of an AST routine to be called when the offspring task exits or emits status
iesb	=	Name of an 8-word status block to be written when the offspring task exits or emits status

Word 0	-- Offspring task exit status
Word 1	-- TKTN abort code
Word 2-7	-- Reserved

NOTE

The exit status block defaults to 1 one word. To use the 8-word exit status block, you must specify the logical or of the symbol SP.WXB and the event flag number in the iefn parameter above.

iparm	=	Name of a word to receive the status block address when an AST occurs
ids	=	Integer to receive the Directive Status Word

Macro Call:

```
     SDRC$    tname,buf,[efn],[east],[esb]
```

tname	=	Target task name of the offspring task to be connected
buf	=	Address of 13-word send buffer
efn	=	The event flag to be cleared on issuance and set when the offspring task exits or emits status
east	=	Address of an AST routine to be called when the offspring task exits or emits status

esb = Address of an 8-word status block to be written when the offspring task exits or emits status

Word 0 -- Offspring task exit status

Word 1 -- TKTN abort code

Word 2-7 -- Reserved

NOTE

The exit status block defaults to 1 one word. To use the 8-word exit status block, you must specify the logical or of the symbol SP.WXB and the event flag number in the efn parameter above.

Macro Expansion:

```
SDRC$    ALPHA,BUFFR,2,SDRCTR,STBLK
.BYTE    141.,7          ;SDRC$ MACRO DIC, DPB SIZE=7 WORDS
.RAD50   ALPHA           ;TARGET TASK NAME
.WORD    BUFFR           ;SEND BUFFER ADDRESS
.BYTE    2               ;EVENT FLAG NUMBER = 2
.BYTE    16.             ;EXIT STATUS BLOCK CONSTANT
.WORD    SDRCTR          ;ADDRESS OF AST ROUTINE
.WORD    STBLK           ;ADDRESS OF STATUS BLOCK
```

Local Symbol Definitions:

S.DRTN -- Task name (4)

S.DRBF -- Buffer address (2)

S.DREF -- Event flag (2)

S.DREA -- AST routine address (2)

S.DRES -- Status block address (2)

DSW Return Codes:

IS.SUC -- Successful completion.

IE.UPN -- Insufficient dynamic memory to allocate a send packet, Offspring Control Block, Task Control Block, or Partition Control Block.

IE.INS -- The specified task is an ACP or has the no-send attribute.

IE.IEF -- An invalid event flag number was specified (EFN < 0 or EFN > 96 if group global event flags exist for the task. EFN > 64 if not.).

IE.ADP -- Part of the DPB or exit status block is not in the issuing task's address space.

IE.SDP -- DIC or DPB size is invalid.

Notes:

1. If the specified event flag is group global, the use count for the event flag's group is incremented to prevent premature elimination of the event flags. The use count is run down when:

   - Status is returned from the connected task.

   - The issuing task exits before status is returned.

2. The virtual mapping of the exit status block should not be changed while the connection is in effect. Doing so may result in obscure errors.

3. If the directive is rejected, the state of the specified event flag is indeterminate.

5.3.63 Send Data Request and Pass Offspring Control Block

This directive instructs the system to send a send data packet for the specified task, chain to the requested task, and request it if it is not already active.

FORTRAN Call:

```
CALL SDRP(task,ibuf,[ibfl],[iefn],[iflag],[iparen],[iocbad]
          [,ids])
```

task = Name of an array (REAL,INTEGER,I*4) that contains the RAD50 name of the target task

ibuf = Name of an integer array containing the data to be sent

ibfl = Name of an integer containing the number of words (integers) in the array to be sent. On RSX-11M systems, this argument must always be 13 or must be defaulted. On RSX-11M-PLUS systems this argument may be in the range of 1 to 255. On either system if this argument is not specified, a default value of 13. is assumed.

iefn = Name of an integer containing the number of the event flag that is to be set when this directive is executed successfully.

iflag = Name of an integer containing the flag bits controlling the execution of this directive. They are defined as follows:

SD.REX = 128.	Force this task to exit upon successful execution of this directive
SD.RAL = 1	Pass all connections to the requested task (default is pass none). If you specify this flag, do not specify the parent task name.

NOTE

The target task may not be a CLI task.

SD.RNX = 2	Pass the first connection in the queue, if there is one, to the requested task. If you specify this flag, do not specify the parent task name.

iparen = Name of an array containing the RAD50 name of the parent task whose connection should be passed to the target task. The name of the parent task was returned in the information buffer of the GTCMCI subroutine.

iocbad	=	Name of an integer containing the pool address of the OCB to pass. This value was returned in the information buffer of the GTCMCI subroutine. Only CLI tasks may specify this parameter.
ids	=	Name of an integer to receive the contents of the Directive Status Word.

Macro Call:

```
SDRP$ task,bufadr,[buflen],[efn],[flag],[parent],[ocbad]
```

task = Name of task to be chained to

bufadr = Address of buffer to be given to the requested task

buflen = Length of buffer to be given to requested task

efn = Event flag

flag = Flags byte controlling the execution of this directive. The flag bits are defined as follows:

SD.REX	= 128.	Force this task to exit upon successful completion of this directive.
SD.RAL	= 1	Pass all connections to the requested task (default is pass none). If you specify this flag, do not specify the parent task name.

NOTE

The target task may not be a CLI task.

SD.RNX	= 2	Pass the first connection in the queue, if there is one, to the requested task. If you specify this flag, do not specify the parent task name.

parent = Name of issuing task's parent task whose connection is to be passed. If not specified, all connections or no connections are passed depending on the flag byte.

ocbad = Address of OCB to pass (CLI tasks only)

Macro Expansion:

```
SDRP$ task,bufadr,[buflen],[efn],[flag],[parent],[ocbad]

.BYTE   141.,9.         ;DIC = 141, DPB LENGTH =9 WORDS
.RAD50  /task/          ;TASK NAME IN RADIX-50
.WORD   BUFADR          ;BUFFER ADDRESS
.BYTE   EFN,FLAG        ;EVENT FLAG, FLAGS BYTE
.WORD   BUFLEN          ;BUFFER LENGTH
.RAD50  /PARENT/        ;PARENT TASK NAME
.WORD   OCBAD           ;ADDRESS OF OCB
```

Local Symbol Definitions:

S.DRTK --	RAD50 name of task to be chained to
S.DRAD --	Send data buffer address
S.DREF --	Event flag
S.DRFL --	Flags byte:

SD.REX -- (200)	Force task to exit (task issuing directive)
SD.RAL -- (1)	Pass all connections to the requested task (default is pass none). If you specify this flag, do not specify the parent task name.
SD.RNX -- (2)	Pass the first connection in the queue, if there is one, to the requested task. If you specify this flag, do not specify the parent task name.

S.DRBL --	Length of send data packet (Always 13 words for RSX-11M, up to 255 words for RSX-11M-PLUS)
S.DRPT --	Name of parent whose OCB should be passed
S.DROA --	Address of OCB to pass (CLIs only)

DSW Return Codes:

IE.ITS --	A task that is not a CLI specified a CLI only parameter, or attempted to pass all connections to a CLI.
IE.NVR --	No offspring control block from specified parent.
IE.ALG --	A CLI specified a parent name and an offspring control block address that did not describe the same connection, or either a parent name or an OCB address was specified and the pass all connections flag was set.
IE.IBS --	Length of send packet is illegal. On RSX-11M systems the send packet must be 13. bytes long. On RSX-11M-PLUS systems the send packet may be up to 255. bytes long.
IE.UPN --	Insufficient dynamic memory to allocate a send packet, offspring control block, task control block, or partition control block.
IE.INS --	The specified task is an ACP or has the no-send attribute.
IE.IEF --	An invalid event flag number was specified (EFN<0 or EFN >96 if group global event flags exist. EFN >64 if not).
IE.ADP --	Part of the DPB or exit status block is out of the issuing task's address space.
IE.SDP --	DIC or DPB size is invalid.

Notes:

1. If the directive is rejected, the state of the specified event flag is indeterminate.

2. If the specified event flag is group global, the use count for the event flag's group is incremented to prevent premature elimination of the event flags. The use count is run down when:

   - Status is returned from the connected tasks

   - The issuing task exits before status is returned.

5.3.64 Set Event Flag

The Set Event Flag directive instructs the system to set an indicated event flag, reporting the flag's polarity before setting.

FORTRAN Call:

```
     CALL SETEF (efn[,ids])

       efn  =  Event flag number

       ids  =  Directive status
```

Macro Call:

```
     SETF$    efn

       efn  =  Event flag number
```

Macro Expansion:

```
     SETF$     52.
     .BYTE     33.,2          ;SETF$ MACRO DIC, DPB SIZE=2 WORDS
     .WORD     52.            ;EVENT FLAG NUMBER 52.
```

Local Symbol Definitions:

S.ETEF	--	Event flag number (2)

DSW Return Codes:

IS.CLR	--	Flag was clear.
IS.SET	--	Flag was already set.
IE.IEF	--	Invalid event flag number (EFN<1, or EFN>96 if group global event flags exist for the task's group; or EFN>64 if not).
IE.ADP	--	Part of the DPB is out of the issuing task's address space.
IE.SDP	--	DIC or DPB size is invalid.

Note:

Set Event Flag does not declare a significant event; it merely sets the specified flag.

SFPA$

5.3.65 Specify Floating Point Processor Exception AST

The Specify Floating Point Processor Exception AST directive instructs the system to record one of the two following cases:

- Floating Point Processor exception ASTs for the issuing task are desired, and the Executive is to transfer control to a specified address when such an AST occurs for the task.
- Floating Point Processor exception ASTs for the issuing task are no longer desired.

When an AST service routine entry point address is specified, future Floating Point Processor exception ASTs will occur for the issuing task, and control will be transferred to the indicated location at the time of the AST's occurrence. When an AST service entry point address is not specified, future Floating Point Processor exception ASTs will not occur until the task issues a directive that specifies an AST entry point. See Notes below.

FORTRAN Call:

Not supported

Macro Call:

```
SFPA$    [ast]

  ast  =  AST service routine entry point address
```

Macro Expansion:

```
SFPA$     FLTAST
.BYTE     111.,2           ;SFPA$ MACRO DIC, DPB SIZE=2 WORDS
.WORD     FLTAST           ;ADDRESS OF FLOATING POINT AST
```

Local Symbol Definitions:

S.FPAE -- AST entry address (2)

DSW Return Codes:

IS.SUC	--	Successful completion.
IE.UPN	--	Insufficient dynamic memory.
IE.ITS	--	AST entry point address is already unspecified or task was built without floating-point support (FP switch not specified in Task Builder .TSK file specification).
IE.AST	--	Directive was issued from an AST service routine, or ASTs are disabled.
IE.ADP	--	Part of the DPB is out of the issuing task's address space.
IE.SDP	--	DIC or DPB size is invalid.

Notes:

1. A Specify Floating Point Processor Exception AST requires dynamic memory.

2. The Executive queues Floating Point Processor exception ASTs when a Floating Point Processor exception trap occurs for the task. No future ASTs of this kind will be queued for the task until the first one queued has actually been effected.

3. The Floating Point Processor exception AST service routine is entered with the task stack in the following state:

   ```
   SP+12 - Event flag mask word
   SP+10 - PS of task prior to AST
   SP+06 - PC of task prior to AST
   SP+04 - DSW of task prior to AST
   SP+02 - Floating exception code
   SP+00 - Floating exception address
   ```

   The task must remove the floating exception code and address from the task's stack before an AST Service Exit (see Section 5.3.4) directive is executed.

4. This directive cannot be issued either from an AST service routine or when ASTs are disabled.

5. This directive applies only to the Floating Point Processor.

SMSG$

5.3.66 Send Message

The Send Message directive instructs the system to create and send a formatted data packet to a system-defined target task. The only valid target for the Send Message directive is Error Logger, and the Error Logging formatted data packet is an Error Log packet. The task that issues the SMSG$ directive must be privileged. The valid system defined target and identifier are:

TARGET IDENTIFIER	CODE
Error Logging	SM.SER

FORTRAN Call:

```
CALL SMSG (itgt,ibuf,ibufl,iprm,iprml,ids)
```

itgt	=	The name of the integer containing the target object (currently only SM.SER is defined)
ibuf	=	The name of an integer array containing the data to be inserted into the formatted data packet.
ibufl	=	The name of an integer containing the length of the ibuf array.
iprm	=	The name of an integer array containing any additional parameters.
iprml	=	The name of an integer containing the number of parameters in the iprm array.
ids	=	The name of an optional integer to receive the directive status.

MACRO Call:

```
SMSG$ tgt,buf,len,<pri,...,prn>
```

tgt	=	Target identifier
buf	=	Address of optional data buffer
len	=	Length in bytes of optional data buffer
pri,...,prn	=	Target-specific parameter list:

Parameter list for Error Logging

```
SMSG$ SM.SER,buf,len,<typ,sub,lun,msk>
```

typ	=	Error Log packet type code
sub	=	Error Log packet subtype code
lun	=	Logical unit number of device
msk	=	Control mask word

The directive creates an Error Log packet of the specified type and subtype codes. If you specify a LUN, the directive also records information about the device to which the LUN refers. The control mask word sets flags to zero I/O and error counts on the device specified, as shown below:

Control mask word flag:

SM.ZER -- Zeroes device I/O and error counts for device specified by LUN

The directive also creates the following subpackets and places them in the Error Log packet in the order listed below:

1. Header Subpacket - The header subpacket, which contains the type and subtype codes, the time stamp, and system identification, is always recorded.

2. Task Subpacket - The task subpacket, which identifies the task that issued the directive, is always recorded.

3. Device Subpacket - The device subpacket, which identifies the device, is recorded if the directive specifies a LUN argument.

4. Data Subpacket - The data subpacket is recorded if the directive specifies an address and length of an optional data buffer.

Macro Expansion (with Error Logging target)

```
SMSG$   SM.ERR  DATBUF,DATLEN,<PR1,PR2,PR3,PR4>
.BYTE   DIC,8.  ;SMSG$ MACRO DIC, DPB SIZE=8 WORDS
.WORD   SM.ERR  ;TARGET IDENTIFIER - ERROR LOGGING
.WORD   DATBUF  ;DATA BUFFER ADDRESS
.WORD   DATLEN  ;DATA BUFFER LENGTH
.WORD   PR1     ;PARAMETER  1
.WORD   PR2     ;PARAMETER  2
.WORD   PR3     ;PARAMETER  3
.WORD   PR4     ;PARAMETER  4
```

Local Symbol Definitions:

S.MTGT -- Target identifier (2)

S.MDBA -- Buffer address (2)

S.MDBL -- Buffer length (2)

S.MPRL -- Parameter list

S.MERR -- Error Log Target Identifier

DSW Return Codes:

IS.SUC -- Successful completion.

IE.ILU -- Invalid LUN (Error Log target only).

IE.SDP -- DIC or DPB size is invalid.

IE.ULN	--	Unassigned LUN (Error Log target only).
IE.UPN	--	Insufficient dynamic memory.
IE.INS	--	Target task is not installed.
IE.ITS	--	Invalid target identifier or invalid control mask.
IE.ADP	--	Part of the DPB or data buffer is out of the issuing task's address space.

5.3.67 Send Next Command

The Send Next command directive allows a task that is servicing a CLI command to inform the system that the command execution is complete. This normally happens automatically when the task exits, and this directive is not necessary if the task will exit when it completes the command. It is intended for tasks that do not exit at this point.

The task of concern here is the final task involved in the command processing. For example, a CLI that passes the command to another task using the RPOI$ or SDRP$ directives and exits need not issue an SNXC$ directive. If the CLI were to do all the processing necessary for a command, not pass it to another task, and go on to the next command, it would have to issue an SNXC$ directive.

Issuing this directive causes a prompt request to be generated if one would have occurred on task exit, and will cause the terminal driver to send the next command to the dispatcher if the terminal is in serial execution mode.

A nonprivileged task may specify only its TI:. A privileged task or a CLI task may specify any terminal. If no terminal is specified, the default is the issuing task's TI:.

FORTRAN Call:

```
     CALL SNXC([dnam][,iunit][,ids])
```

dnam	=	Device name (ASCII). If not specified, TI: is used.
iunit	=	Unit number of the terminal from which the command is to be sent.
ids	=	Integer to receive the directive status word.

Macro Call:

```
     SNXC$   [dnam][,unum]
```

dnam	=	Device name (ASCII). If not specified, TI: is used.
unum	=	Unit number of the terminal from which the command is to be sent.

Macro Expansion:

```
     SNXC$   TT,3
     .BYTE     127.,3     ;SNXC$ MACRO DIC, DPB SIZE=3 WORDS
     .ASCII    /TT/       ;ASCII DEVICE NAME
     .BYTE     3,0        ;UNIT NUMBER IS 3
```

DSW Return Codes:

IE.IDU	--	The specified device does not exist or is not a terminal.
IE.PRI	--	Nonprivileged task specified a terminal other than its own TI:.
IE.ADP	--	Part of the DPB is out of the issuing task's address space.
IE.SDP	--	DIC or DPB size is invalid.

SPEA$

5.3.68 Specify Parity Error AST

The Specify Parity Error AST directive enables a task to specify an AST service routine to be entered if a hardware parity error occurs. If an AST address is not specified, any previously specified parity error AST is canceled. Upon entering the AST service routine, the stack contains the following information:

```
     SP+62 -- Event flag mask word
     SP+60 -- PS of task prior to AST
     SP+56 -- PC of task prior to AST
     SP+54 -- Task's directive status word
     SP+52 --
     SP+50 --
     SP+46 --
     SP+44 --
     SP+42 --
     SP+40 --
     SP+36 --
     SP+34 --                  Contents of memory parity CSRs
     SP+32 --                  (hardware-dependent information)
     SP+30 --
     SP+26 --
     SP+24 --
     SP+22 --
     SP+20 --
     SP+16 --
     SP+14 --
     SP+12 -- Contents of cache control register
     SP+10 -- Contents of memory system error register
     SP+06 -- Contents of high error address register
     SP+04 -- Contents of low error address register
     SP+02 -- Processor identification (single processor system=0)
     SP+00 -- Number of bytes to add to SP to clean the stack (52)
```

FORTRAN Call:

Not supported

Macro Call:

```
     SPEA$   [ast]

       ast  =  AST service routine entry point address
```

Macro Expansion:

```
     SPEA$      PTYERR
     .BYTE      165.,2          ;SPEA$ MACRO DIC, DPB SIZE=2 WORDS
     .WORD      PTYERR          ;PARITY ERROR AST ROUTINE ADDRESS
```

Local Symbol Definitions:

```
     S.PEAE  --  Parity error AST routine address (2)
```

DSW Return Codes:

IS.SUC	--	Successful completion.
IE.UPN	--	Insufficient dynamic storage.
IE.ITS	--	ASTs already not desired.
IE.AST	--	Directive was issued from an AST service routine, or ASTs are disabled.
IE.ADP	--	Part of the DPB is out of the issuing task's address space.
IE.SDP	--	DIC or DPB size is invalid.

5.3.69 Suspend ($S Form Recommended)

The Suspend directive instructs the system to suspend the execution of the issuing task. A task can suspend only itself, not another task. The task can be restarted either by a Resume directive, or by an MCR or DCL Resume command.

FORTRAN Call:

```
     CALL SUSPND   [(ids)]

       ids  =  Directive status
```

Macro Call:

```
     SPND$S   [err]

       err  =  Error routine address
```

Macro Expansion:

```
     SPND$S   ERR
     MOV      (PC)+,-(SP)       ;PUSH DPB ONTO THE STACK
     .BYTE    45.,1             ;SPND$S MACRO DIC, DPB SIZE=1 WORD
     EMT      377               ;TRAP TO THE EXECUTIVE
     BCC      .+6               ;BRANCH IF DIRECTIVE SUCCESSFUL
     JSR      PC,ERR            ;OTHERWISE, CALL ROUTINE "ERR"
```

Local Symbol Definitions:

None

DSW Return Codes:

IS.SPD	--	Successful completion (task was suspended).
IE.ADP	--	Part of the DPB is out of the issuing task's address space.
IE.SDP	--	DIC or DPB size is invalid.

Notes:

1. A suspended task retains control of the system resources allocated to it. The Executive makes no attempt to free these resources until a task exits.

2. A suspended task is eligible for checkpointing unless it is fixed or declared to be noncheckpointable.

3. Because this directive requires only a 1-word DPB, the $S form of the macro is recommended. It requires less space and executes with the same speed as that of the DIR$ macro.

SPRA$

5.3.70 Specify Power Recovery AST

The Specify Power Recovery AST directive instructs the system to record one of the two following cases:

1. Power recovery ASTs for the issuing task are desired, and control is to be transferred when a powerfail recovery AST occurs.

2. Power recovery ASTs for the issuing task are no longer desired.

When an AST service routine entry point address is specified, future power recovery ASTs will occur for the issuing task, and control will be transferred to the indicated location at the time of the AST's occurrence. When an AST service entry point address is not specified, future power recovery ASTs will not occur until an AST entry point is again specified. See Notes below.

FORTRAN Call:

To establish an AST:

```
     EXTERNAL  sub
     CALL PWRUP (sub)
```

sub = Name of a subroutine to be executed upon power recovery. The PWRUP subroutine will effect a

CALL sub (no arguments).

The subroutine is called as a result of a power recovery AST, and therefore may be controlled at critical points by using DSASTR and ENASTR subroutine calls.

To remove an AST:

```
     CALL PWRUP
```

Macro Call:

```
     SPRA$   [ast]
```

ast = AST service routine entry point address

Macro Expansion:

```
     SPRA$   PWRAST
     .BYTE   109.,2          ;SPRA$ MACRO DIC, DPB SIZE=2 WORDS
     .WORD   PWRAST          ;ADDRESS OF POWER RECOVERY AST
```

Local Symbol Definitions:

```
     S.PRAE  --  AST entry address (2)
```

DSW Return Codes:

IS.SUC	--	Successful completion.
IE.UPN	--	Insufficient dynamic memory.
IE.ITS	--	AST entry point address is already unspecified.
IE.AST	--	Directive was issued from an AST service routine, or, ASTs are disabled.
IE.ADP	--	Part of the DPB is out of the issuing task's address space.
IE.SDP	--	DIC or DPB size is invalid.

Notes:

1. Specify Power Recovery AST requires dynamic memory.

2. The Executive queues power recovery ASTs when the power-up interrupt occurs following a power failure. No future powerfail ASTs will be queued for the task until the first one queued has been effected.

3. The task enters the powerfail AST service routine with the task stack in the following state:

   ```
   SP+06 - Event flag mask word
   SP+04 - PS of task prior to AST
   SP+02 - PC of task prior to AST
   SP+00 - DSW of task prior to AST
   ```

   No trap-dependent parameters accompany a power recovery AST; therefore, the AST Service Exit directive (see Section 5.3.4) can be executed with the stack in the same state as when the AST was entered.

4. This directive cannot be issued either from an AST service routine or when ASTs are disabled.

5. Refer to Chapter 1 for a list of the restrictions on operations that may be performed in a FORTRAN AST routine.

SPWN$

5.3.71 Spawn

The Spawn directive requests a specified task for execution, optionally queuing a command line[1] and establishing the task's TI: as a previously created virtual terminal unit or a physical terminal.

When this directive is issued, an Offspring Control Block (OCB) is queued to the offspring TCB and a rundown count is incremented in the parent task's TCB. The rundown count is used to inform the Executive that the task is a parent task and has one or more offspring tasks and virtual terminal(s); cleanup is necessary if a parent task exits with active offspring tasks. The rundown count is decremented when the spawned task exits. The OCB contains the TCB address as well as sufficient information to effect all of the specified exit events when the offspring task exits.

If a command line is specified, it is buffered in the Executive pool and queued for the offspring task for subsequent retrieval by the offspring task with the Get MCR Command Line directive. Maximum command line length is 79 characters on RSX-11M systems and 255 characters on RSX-11M-PLUS systems.

If an AST address is specified, an exit AST routine is effected when the spawned task exits with the address of the task's exit status block on the stack. The AST routine must remove this word from the stack before issuing the AST Service Exit directive.

Special action is taken if the task being spawned is a Command Line Interpreter (CLI), such as MCR or DCL. In this case, a command line must be specified, and both the OCB and the command line are queued for the interpreter task. MCR and DCL either handle commands directly or dispatch them to another task. In the case of direct execution of the command, the OCB may be used to immediately effect the proper exit conditions and return exit status by an Executive routine. If MCR or DCL dispatch another task, they simply move the OCB from their own OCB queue directly to the OCB queue of the dispatched task. They also queue the command line for the dispatched task as usual. At this point, the situation is exactly the same as if the SPWN$ directive had specified the dispatched task directly. No exit conditions occur until the dispatched task exits.

FORTRAN Call:

```
     CALL SPAWN (rtname,[iugc],[iumc],[iefn],[iast],[iesb],[iparm],
                 [icmlin],[icmlen],[iunit],[dnam][,ids])
```

rtname	=	Name (RAD50) of the offspring task to be spawned.
iugc	=	Group code number for the UIC of the offspring task.
iumc	=	Member code number for the UIC of the offspring task.
iefn	=	Event flag to be set when the offspring task exits or emits status.

1. Command line processing is not available for RSX-11S tasks.

iast	=	Name of an AST routine to be called when the offspring task exits or emits status.
iesb	=	Name of an 8-word status block to be written when the offspring task exits or emits status.

Word 0 -- Offspring task exit status

Word 1 -- TKTN abort code

Word 2-7 -- Reserved

NOTE

The exit status block defaults to one word. To use the 8-word exit status block, you must specify the logical or of the symbol SP.WX8 and the event flag number in the iefn parameter above.

iparm	=	Name of a word to receive the status block address when the AST occurs.
icmlin	=	Name of a command line to be queued for the offspring task.
icmlen	=	Length of the command line (79. characters maximum).
iunit	=	Unit number of terminal to be used as the TI: for the offspring task. If the optional dnam parameter is not specified, this parameter must be the unit number of a virtual terminal created by the issuing task; if a value of 0 is specified, the TI: of the issuing task is propagated. A task must be privileged or must be a CLI task in order to specify a TI: other than the parent task's TI:.
dnam	=	Device name mnemonic. If not specified, the virtual terminal is used as TI:.
ids	=	Integer to receive the directive status word.

Macro Call:

```
SPWN$   tname,,,[ugc],[umc],[efn],[east],[esb],[cmdlin],[cmdlen],
          [unum],[dnam]
```

tname	=	Name (RAD50) of the offspring task to be spawned.
ugc	=	Group code number for the UIC of the offspring task.
umc	=	Member code number for the UIC of the offspring task.
efn	=	The event flag to be cleared on issuance and set when the offspring task exits or emits status.
east	=	Address of an AST routine to be called when the offspring task exits or emits status.

esb = Address of an 8-word status block to be written when the offspring task exits or emits status.

Word 0 -- Offspring task exit status

Word 1 -- TKTN abort code

Word 2-7 -- Reserved

NOTE

The exit status block defaults to one word. To use the 8-word exit status block, you must specify the logical or of the symbol SP.WX8 and the event flag number in the efn parameter above.

cmdlin = Address of a command line to be queued for the offspring task.

cmdlen = Length of the command line (maximum length is 79.).

unum = Unit number of terminal to be used as the TI: for the offspring task. If the optional dnam parameter is not specified, this parameter must be the unit number of a virtual terminal created by the issuing task; if a value of 0 is specified, the TI: of the issuing task is propagated. A task must be privileged or must be a CLI task in order to specify a TI: other than the parent task's TI:.

dnam = Device name mnemonic. If not specified, the virtual terminal is used as TI:.

Macro Expansion:

```
SPWN$   ALPHA,,,3,7,1,ASTRUT,STBLK,CMDLIN,72.,2
.BYTE   11.,13.         ;SPWN$ MACRO DIC, DPB SIZE=13 WORDS
.RAD50  ALPHA           ;NAME OF TASK TO BE SPAWNED
.BLKW   3               ;RESERVED
.BYTE   7,3             ;UMC = 7   UGC = 3
.BYTE   1               ;EVENT FLAG NUMBER = 1
.BYTE   16.             ;EXIT STATUS BLOCK CONSTANT
.WORD   ASTRUT          ;AST ROUTINE ADDRESS
.WORD   STBLK           ;EXIT STATUS BLOCK ADDRESS
.WORD   CMDLIN          ;ADDRESS OF COMMAND LINE
.WORD   72.             ;COMMAND LINE LENGTH = 72. CHARACTERS
.WORD   2               ;VIRTUAL TERMINAL UNIT NUMBER =2
```

NOTE

If a virtual terminal is not specified, one additional parameter (device name) can be added for a hardware terminal name. For example, TT2 (instead of VT2) would have the same macro expansion shown above, plus the following:

```
.ASCII  /TT/    ;ASCII  DEVICE NAME
```

The DPB size will then be 14 words.

Local Symbol Definitions:

S.PWTN	--	Task name (4)
S.PWXX	--	Reserved (6)
S.PWUM	--	User member code (1)
S.PWUG	--	User group code (1)
S.PWEF	--	Event flag number (2)
S.PWEA	--	Exit AST routine address (2)
S.PWES	--	Exit status block address (2)
S.PWCA	--	Command line address (2)
S.PWCL	--	Command line length (2)
S.PWVT	--	Terminal unit number (2)
S.PWDN	--	Device name (2)

DSW Return Codes:

IS.SUC	--	Successful completion.
IE.UPN	--	Insufficient dynamic memory to allocate an offspring control block, command line buffer, task control block, or partition control block.
IE.INS	--	The specified task was not installed, or it was a command line interpreter but no command line was specified.
IE.ACT	--	The specified task was already active and it was not a command line interpreter.
IE.PRI	--	Nonprivileged task attempted to specify an offspring task's TI: to be different from its own.

DSW Return Codes:

IE.IDU	--	The specified virtual terminal unit does not exist, or it was not created by the issuing task, or the specified TI: device is not a terminal.
IE.IEF	--	Invalid event flag number (EFN<0, or EFN>96 if group global event flags exist for the task's group; or EFN>64 if not).
IE.ADP	--	Part of the DPB, exit status block, or command line is out of the issuing task's address space.
IE.SDP	--	DIC or DPB size is invalid.

Notes:

1. If the UIC is defaulted and the offspring task is not a Command Line Interpreter (CLI), that task is requested to run under the UIC of the parent task. If the UIC is defaulted, the offspring task is a CLI, and the CLI passes the specified command line to a dispatched task, the dispatched task will run under the UIC of its TI: terminal. See the notes for the Request Task (RQST$) directive for more information about task UICs.

2. If the specified event flag is group global, then the use count for the event flag's group is incremented to prevent premature elimination of event flags. The use count is run down when:

   - Status is returned from the spawned task.

   - The issuing task exits before status is returned.

3. The virtual mapping of the exit status block should not be changed while the connection is in effect. Doing so may cause obscure errors.

4. The types of operations that a FORTRAN AST routine may perform are extremely limited. Please refer to Chapter 1 for a list of restrictions.

The following program illustrates the use of the FORTRAN callable SPAWN routine and the mechanism for handling ASTs from a FORTRAN program:

```
      P R O G R A M   S P W A S T
C
C This program illustrates the use of the FORTRAN callable
C SPAWN routine and the use of a FORTRAN subprogram at AST state.
C This example keeps "ITMAX" tasks active at any point in time
C without having several copies of each utility installed under
C different names. The input file consists of single line commands
C of up to 45 characters in length which invoke tasks in the system
C library UIC. The first three characters of the input command line
C are the name of the task to be invoked(ie: MAC).  The output file
C consists of a log file containing the command lines and the exit status
C of the program invoked.
C
```

```
C The above is accomplished as follows:
C
C       A command is read from the input file "CMDFIL.CMD" which has the
C form "NAM COMMAND", where NAM is the name of the task and COMMAND is the
C command to be passed to this task. This input command line is transformed
C into an MCR "RUN" command line such as
C       "RUN $MAC/TASK=TSK#/EST=NO/CMD="COMAND".
C where # is a number assigned by this task so that the target task name
C is both known and unique. The MCR dispatcher (MCR...) is spawned with this
C transformed command line, which in turn causes the MCR... task to dispatch
C a copy of ...MCR under the name MCRTnn to execute this command.  When
C this copy of ...MCR exits, an exit AST is serviced by this task which
C issues a "CONNECT" to the target task "TSK#". This method introduces a timing
C window such that the target task could exit before the CONNECT is made. In
C this case, an error message is written to the log file indicating that
C exit status could not be returned due to a connect failure.
C
C This non-privileged FORTRAN IV-PLUS program is compiled and
C built as follows:
C
C MCR>F4P SPWAST,SPWAST/-SP=SPWAST
C MCR>TKB SPWAST/FP,SPWAST=SPWAST,LB:[1,1]F4POTS/LB
C
C
C Define data structures
C
C  The following variables are kept on a per active "invoked task" basis
C  For lack of a better name, each respective entry is called task
C  inforamation block.
C
C  IESTAT(8,XXX)        IEXSAD(XXX)     ISTAT(XXX)      ICMDLN(45,XXX)
C

        PARAMETER ITMAX=3

        COMMON /KOM1/IESTAT(8,ITMAX),IEXSAD(ITMAX),ISTAT(ITMAX),IPARM,RTNAME(2)
        COMMON /KOM2/THISTK(16)
        COMMON /COMMAN/ICMDLN(45,ITMAX)


        INTEGER IESTAT  !exit status array for each task
        INTEGER IEXSAD  !array containing the address of each task's iestat
        INTEGER ISTAT   !array containing the status (active vs free) of
C                        each task information block.
        INTEGER IPARM   !contains address of IESTAT at AST state
        INTEGER RTNAME  !contains the RAD50 name of the target task to be
C                       !connected to at AST state
        INTEGER THISTK
        BYTE ICMDLN     !saved input command line per task
C
C  Local input buffer variables
C
        DIMENSION INPCOM(3)
        DIMENSION INPBUF(45)
        EQUIVALENCE (INPBUF(1),INPCOM(1))

        BYTE INPBUF     !INPUT BUFFER
        BYTE INPCOM     !COMPONENT NAME FIELD OF INPBUF
```

```
C
C  Local variables for SPAWN call
C
        EXTERNAL EXTAST         !define the name of the AST routine externally

        DIMENSION CMDLIN(79)    !maximum command line passed to is 79. bytes

        BYTE CMDLIN             !actual command line passed to MCR...
        INTEGER*4 DSPNAM        !variable containing RAD50 task name of MCR...

        DATA    DSPNAM/6RMCR.../!fill in name of ...MCR at compile time
C
C  Local control variables
C
        INTEGER ITCNT   !count of number of free task information blocks
        LOGICAL EOF     !flag indicating EOF detected on command input file
C
C  Misc. local variables
C
        INTEGER IDSW    !integer to contain directive status
C
C Open files
C
        OPEN (UNIT=1,TYPE='OLD',READONLY,NAME='CMDFIL.CMD')
        OPEN (UNIT=2,TYPE='NEW',CARRIAGECONTROL='FORTRAN',NAME='CMDFIL.LOG')
C
C Initialize Variables
C
        ITCNT=ITMAX+1   !set current count of available task info blocks
        EOF=.FALSE.     !reset EOF flag

        CALL IRAD50(3,'TSK',RTNAME(1)) !setup first half of target task name
        CALL GETTSK(THISTK(1))         !determine this task's name so that
C                                       STOPing and UNSTOPing may be done
C
C Initialize the IEXSAD array such that each entry contains the address
C of the exit status block which has the corresponding index. This is
C necessary so that the correct exit status block may be determined at AST
C state.
C
        DO 5 I=1,ITMAX
        CALL GETADR(IEXSAD(I),IESTAT(1,I))
5       CONTINUE

C
C Read a command line from the input file and initialize a free task info
C block.
C

10      READ (1,900,END=30)I,INPBUF     !read input command line
        ITCNT=ITCNT-1                   !one less free block
        DO 20 K=1,ITMAX                 !search for the free block
        IF (ISTAT(K) .NE. 0) GOTO 20    !IF NE, block is in use
        ISTAT(K)=1              !ELSE found one, mark it in use
        DO 15 J=1,I             !save command line for output later
        ICMDLN(J,K)=INPBUF(J)
15      CONTINUE
        DO 16 J=I+1,45          !pad saved command line with spaces
        ICMDLN(J,K)="40
16      CONTINUE
        GOTO 40         !exit search loop
20      CONTINUE
30      EOF=.TRUE.      !set EOF flag
        GOTO 55         !continue to log exit status of what's currently
C                       !active
```

```
C
C Construct the actual command line specified in the SPAWN call
C
C  Write saved command line to TI: so that any MCR "RUN" error messages
C  have context.

40      WRITE(5,710)(ICMDLN(J,K),J=1,45)
710     FORMAT(1X,45A1)

        ENCODE(I+35,800,CMDLIN)INPCOM,K,(INPBUF(J),J=1,I)
800     FORMAT('RUN $',3A1,'/TASK=TSK',I1,'/EST=NO/CMD="',45A1)
        CMDLIN(I+32)="42         !add terminating quote
        CMDLIN(I+33)="15         !and terminator.
C
C Spawn MCR... with the command line such as:
C
C       "RUN $MAC/TASK=TSK1/EST=NO/CMD="MAC TEST1=TEST1""
C
C At this point the second half of the RAD50 target task name is calculated
C so that the first exit AST may issue a connect after ...MCR exits.

        RTNAME(2)=40*40*(30+K)  !calculate second half of RAD50 taskname

C Spawn the MCR dispatcher with the constructed command line. The dispatcher
C will then spawn a copy of ...MCR which will in turn process the "RUN" command.

45      CALL SPAWN(DSPNAM,,,1,EXTAST,IESTAT(1,K),IPARM,CMDLIN,I+33,0,,IDSW)

C  An error could be received from the SPAWN call. This could be due to a
C  variety of reasons, such as the task file specified was not found or there
C  was insufficient system resources at the time the executive directive
C  was issued. Only the IE.RSU errors will be recovered by waiting for
C  a significant event and reissuing the call to SPAWN.

        IF(IDSW+1) 50,52,54             !check directive status returned
C
C Spawn error
C
50      IESTAT(1,K)=5                   !if mi, uncorrectable error mark status
        IESTAT(2,K)=IDSW                !save directive status returned for log
        ISTAT(K)=3                      !indicate status present
        GOTO 60                         !go write error to log file and cleanup
C
C Spawn error due to insufficient resources
c
52      CALL WFSNE                      !wait for significant event
        GOTO 45                         !reissue SPAWN
C
C Spawn successful, wait till ...MCR exits and first AST has been serviced.
C
54      CALL WAITFR(1)                  !wait for ...MCR to exit
C
C Do not STOP if connect failed, just process task info block and continue..
C
        IF(IESTAT(1,K) .EQ. 6) GOTO 60  !exit status code of 6 indicates
C                                        connect failure..
C
C At this point a check is made to determine whether this task has
C completed its quest. If there is no more input and all task information
C blocks are free, then exit processing will be performed.
C
```

```
55      IF(EOF .AND. (ITCNT .EQ. ITMAX+1)) GOTO 500
C
C Next, if all the task information blocks are being used, or if there
C is no more input to process, this task is stopped so as to lower its
C priority effectively to zero. This task will once again wake up when
C the connect AST unstops this task.
C
        IF(ITCNT .EQ. 1 .OR. (EOF)) CALL STOP
C
C Scan all the task information blocks to process task information blocks
C which are now waiting for cleanup and log file processing.
C

60      DO 70 K=1,ITMAX                 !search task information blocks for
C                                        the task(s) which exited
        IF (ISTAT(K) .NE. 3) GOTO 70    !if eq, then offspring task connect AST
C                                        has not occurred for this task
        WRITE (2,901) (ICMDLN(J,K),J=1,45)      !write cmdlin to log file
        GOTO (62,63,64,61,65,66,67),(IESTAT(1,K) .AND. "377)+1 !decode exit status
61      WRITE (2,902)(IESTAT(1,K) .AND. "377) !unknown exit status
        GOTO 68
62      WRITE (2,903)                           !EX$WAR -- warning
C                                               !or none returned
        GOTO 68
63      WRITE (2,904)                           !EX$SUC -- success
        GOTO 68
64      WRITE (2,905)                           !EX$ERR -- error
        GOTO 68
65      WRITE (2,906)                           !EX$SEV -- severe error
        GOTO 68
66      WRITE (2,907)IESTAT(2,K)                !internal -- SPAWN failure
        GOTO 68
67      WRITE (2,908)IESTAT(2,K)                !internal -- CONNECT failure
68      ISTAT(K)=0                              !free up task info block
        IESTAT(1,K)=0                           !initialize exit status
        ITCNT=ITCNT+1                           !adjust free task info block ct
70      CONTINUE
        GOTO 10

900     FORMAT(Q,45A1)
901     FORMAT('$',45A1)
902     FORMAT('+','Unknown exit status =',I3)
903     FORMAT('+','<< Warning')
904     FORMAT('+','<< Success')
905     FORMAT('+','<< Error')
906     FORMAT('+','<< Severe error')
907     FORMAT('+','<< Spawn error, DSW =',I3)
908     FORMAT('+','<< Connect error, DSW =',I3)

C
C Exit cleanly by closing all files
C
500     CLOSE (UNIT=1)          !close input  file on LUN 1
        CLOSE (UNIT=2)          !close output file on LUN 2
        CALL EXIT               !exit

        END
        S U B R O U T I N E   E X T A S T

        PARAMETER ITMAX=3
        COMMON /KOM1/IESTAT(8,ITMAX),IEXSAD(ITMAX),ISTAT(ITMAX),IPARM,RTNAME(2)
        COMMON /KOM2/THISTK(16)

        INTEGER IESTAT  !exit status array for each task
        INTEGER IEXSAD  !array containing the address of each task's IESTAT
        INTEGER ISTAT   !array containing the status (active vs free) of
C                        each task information block.
        INTEGER IPARM   !contains address of IESTAT at AST state
        INTEGER RTNAME  !contains the RAD50 name of the target task to be
C                       !connected to at AST state
        INTEGER THISTK

        EXTERNAL TSKEXT
C
C Using IPARM, which contains the address of the exit status block array,
C find the task information block, by comparing this with the address of each
C exit status block array.(contained in IEXSAD)
C
        DO 10 I=1,ITMAX
        IF (IEXSAD(I) .EQ. IPARM) GOTO 20 !found the task info block
10      CONTINUE
        GOTO 30
20      ISTAT(I)=2                        !indicate ...MCR has exited
C
C Try to connect to the target task
C
        CALL CNCT(RTNAME(1),2,TSKEXT,IESTAT(1,I),IPARM,IDSW)
        IF(IDSW .EQ. 1) GOTO 30           !IF EQ, then successful connect
        IESTAT(1,I)=6                     !else pass connect failed status
        IESTAT(2,I)=IDSW
        ISTAT(I)=3                        !mark task info block as done.
30      RETURN    !return from AST state (returns to internal AST handler)

        END


        S U B R O U T I N E   T S K E X T

        PARAMETER ITMAX=3
        COMMON /KOM1/IESTAT(8,ITMAX),IEXSAD(ITMAX),ISTAT(ITMAX),IPARM,RTNAME(2)
        COMMON /KOM2/THISTK(16)
        INTEGER IESTAT  !exit status array for each task
        INTEGER IEXSAD  !array containing the address of each task's IESTAT
        INTEGER ISTAT   !array containing the status (active vs free) of
C                        each task information block.
        INTEGER IPARM   !contains address of IESTAT at AST state
        INTEGER RTNAME  !contains the RAD50 name of the target task to be
C                       !connected to at AST state
        INTEGER THISTK  !This task's name (so that an UNSTOP may be performed)
C
C Find exit status block
C
        DO 10 I=1,ITMAX
        IF (IEXSAD(I) .EQ. IPARM) GOTO 20 !found the task info block
10      CONTINUE
        GOTO 30
20      ISTAT(I)=3                !indicate AST has been serviced
        CALL USTP(THISTK)         !UNSTOP  this task
30      RETURN    !return from AST state (returns to internal AST handler)

        END
```

SRDA$

5.3.72 Specify Receive Data AST

The Specify Receive Data AST directive instructs the system to record one of the following two cases:

- Receive data ASTs for the issuing task are desired, and the Executive transfers control to a specified address when data has been placed in the task's receive queue
- Receive data ASTs for the issuing task are no longer desired.

When the directive specifies an AST service routine entry point, receive data ASTs for the task will subsequently occur whenever data has been placed in the task's receive queue; the Executive will transfer control to the specified address.

When the directive omits an entry point address, the Executive disables receive data ASTs for the issuing task. Receive data ASTs will not occur until the task issues another Specify Receive Data AST directive that specifies an entry point address. See Notes below.

FORTRAN Call:

Neither the FORTRAN language nor the ISA standard permits direct linking to system trapping mechanisms; therefore, this directive is not available to FORTRAN tasks.

Macro Call:

```
SRDA$   [ast]

  ast  =  AST service routine entry point address
```

Macro Expansion:

```
SRDA$   RECAST
.BYTE   107.,2          ;SRDA$ MACRO DIC, DPB SIZE=2 WORDS
.WORD   RECAST          ;ADDRESS OF RECEIVE AST
```

Local Symbol Definitions:

S.RDAE -- AST entry address (2)

DSW Return Codes:

IS.SUC -- Successful completion.

IE.UPN -- Insufficient dynamic memory.

IE.ITS -- AST entry point address is already unspecified.

IE.AST -- Directive was issued from an AST service routine, or ASTs are disabled.

IE.ADP -- Part of the DPB is out of the issuing task's address space.

IE.SDP -- DIC or DPB size is invalid.

Notes:

1. A Specify Receive Data AST requires dynamic memory.

2. The Executive queues receive data ASTs when a message is sent to the task. No future receive data ASTs will be queued for the task until the first one queued has been effected.

3. The task enters the recieve data AST service routine with the task stack in the following state:

   ```
   SP+06 - Event flag mask word
   SP+04 - PS of task prior to AST
   SP+02 - PC of task prior to AST
   SP+00 - DSW of task prior to AST
   ```

   No trap-dependent parameters accompany a receive data AST; therefore, the AST Service Exit directive (see Section 5.3.4) must be executed with the stack in the same state as when the AST was effected.

4. This directive cannot be issued either from an AST service routine or when ASTs are disabled.

5.3.73 Specify Requested Exit AST Directive - SREA$ or SREX$

The Specify Requested Exit AST directive allows the task issuing the directive to specify the AST service routine to be entered if an attempt is made to abort the task by a directive or MCR or DCL ABO command. This allows a task to enter a routine for clean-up instead of abruptly aborting.

If an AST address is not specified, any previously specified exit AST is canceled.

Privileged tasks enter the specified AST routine each time an abort is issued. However, subsequent exit ASTs will not be queued until the first exit AST has occurred.

Nonprivileged tasks enter the specified AST routine only once. Subsequent attempts to abort the task will actually abort the task.

SREX$ is the preferred form of this directive. The differences are explained in Notes 1 and 2 below.

FORTRAN Call:

```
     CALL SREA(ast[,ids])

       ast     = Name of the externally declared AST subroutine

       ids     = Name of an optional integer to receive the  Directive
                 Status Word

     CALL SREX(ast,ipblk,ipblkl,[dummy][,ids])

       ast     = Name of the externally declared AST subroutine

       ipblk   = Name of an integer array to receive the
                 trap-dependent parameters

       ipblkl  = Number of parameters to be returned into the ipblk
                 array.

       dummy   = Reserved for future use

       ids     = Name of an optional integer to receive the  Directive
                 Status Word
```

Macro Call:

```
     SREA$   [ast]

     SREX$   [ast][,dummy]

       ast   = AST service routine entry point address
       dummy = Reserved for future expansion
```

Macro Expansion:

```
     SREA$   REQAST
     .BYTE   167.,2             ;SREA$ MACRO DIC, DPB SIZE=2 WORDS
     .WORD   REQAST             ;EXIT AST ROUTINE ADDRESS

     SREX$   REQAST
     .BYTE   167.,3             ;SREX$ MACRO DIC, DPB SIZE=3 WORDS
     .WORD   REQAST             ;EXIT AST ROUTINE ADDRESS
     .WORD   0                  ;RESERVED FOR FUTURE EXPANSION
```

NOTE

The DPB length for the SREA$ form of the directive is two words. For the SREX$ form of the directive, it is three words.

Local Symbol Definitions:

S.REAE -- Exit AST routine address (2)

DSW Return Codes:

IS.SUC -- Successful completion.

IE.UPN -- Insufficient dynamic storage.

IE.AST -- Directive was issued from an AST service routine, or ASTs are disabled.

IE.ITS -- ASTs already not desired, or nonprivileged task attempted to respecify or cancel the AST after one had already occurred.

IE.ADP -- Part of the DPB is out of the issuing task's address space.

IE.SDP -- DIC or DPB size is invalid.

Notes:

1. The SREX$ form of the directive is recommended for tasks that wish to handle all privileged and nonprivileged abort attempts that do not violate multiuser protection checks. The issuing task can use the information returned on the stack for this version of the directive to decide how to handle the abort attempt.

   After specifying a requested exit AST using the SREX$ form of the directive, the issuing task will enter the AST service routine if any attempt is made to abort the task. On systems with multiuser protection, nonprivileged abort attempts must originate from the same TI: as that of the issuing task.

   When the AST service routine is entered and the AST has been specified using the SREX$ version of the directive, the task's stack is in the following state:

   ```
   SP+12 - Event flag mask word
   SP+10 - PS of task prior to AST
   SP+06 - PC of task prior to AST
   SP+04 - DSW of task prior to AST
   SP+02 - Trap-dependent parameter
   SP+00 - Number of bytes to add to SP to clean stack (4)
   ```

   The trap-dependent parameter is formatted as follows:

   ```
   Bit 0 = 0 if the abort attempt was privileged
         = 1 if the abort attempt was nonprivileged

   Bit 1 = 0 if the ABRT$ directive was issued
         = 1 if the MCR or DCL abort command was used

   Bits 2-15 are reserved for future use
   ```

   The task must remove the trap-dependent parameters from the stack before an AST Service Exit directive is executed. The recommended method is to add the value stored in SP+00 to SP. This is also the only recommended way to access the non-trap-dependent parameters on the stack.

2. The SREA$ form of the directive is recommended for privileged tasks that do not wish abort attempts from a nonprivileged user's MCR or DCL abort command to be allowed, and do not otherwise care about the nature of the abort attempt. It is also recommended for any nonprivileged tasks that simply do not care about the nature of the abort attempt.

   After specifying a requested exit AST using the SREA$ form of the directive, privileged tasks will enter the AST service routine if any of the following abort attempts is made:

   - Any privileged ABRT$ directive or privileged MCR or DCL abort command

   - Any nonprivileged ABRT$ directive on systems without multiuser protection

   - Any nonprivileged ABRT$ directive from the same TI: on systems with multiuser protection

Nonprivileged tasks will enter the AST service routine for all of the abort attempts listed above, plus the following:

- Any nonprivileged MCR or DCL abort command on systems without multiuser protection
- Any nonprivileged MCR or DCL abort command from the same TI: on systems with multiuser protection

When the AST service routine is entered, the task's stack is in the following state:

```
SP+06 - Event flag mask word
SP+04 - PS of task prior to AST
SP+02 - PC of task prior to AST
SP+00 - DSW of task prior to AST
```

No trap-dependent parameters accompany an AST specified by SREA$; therefore, the AST Service Exit directive can be executed with the stack in the same state as when the AST was entered.

3. The event flag mask word at the bottom of the stack preserves the Wait For conditions of a task prior to AST entry. A task can, after an AST, return to a Wait For state. Because these flags and other stack data are in the user task, they can be modified. However, modifying the stack data may cause unpredictable results. Therefore, such modification is not recommended.

4. If an SREX$ requested exit AST is not specified for a task, it is impossible to abort a privileged task from a nonprivileged terminal using either MCR or DCL on systems with multiuser protection.

5. The two forms of this directive should not be mixed in the same code, since the stack format and the trap-dependent parameters differ. Any mismatch between the form of the directive and the AST routine will have unpredictable results.

6. Please see Chapter 1 for a list of restrictions on operations that can be performed in a FORTRAN AST routine.

SREF$

5.3.74 Send by Reference

The Send By Reference directive inserts a packet containing a reference to a region into the receive-by-reference queue of a specified (receiver) task. The Executive automatically attaches the receiver task for each Send By Reference directive issued by the task to the specified region (the region identified in W.NRID of the Window Definition Block). The attachment occurs even if the receiver task is already attached to the region, unless bit WS.NAT in W.NSTS of the Window Definition Block is set. The successful execution of this directive causes a significant event to occur.

The send packet contains:

- A pointer to the created attachment descriptor, which becomes the region ID to be used by the receiver task
- The offset and length words specified in W.NOFF and W.NLEN of the Window Definition Block (which the Executive passes without checking)
- The receiver task's permitted access to the region, contained in the window status word W.NSTS
- The sender task name
- Optionally, the address of an 8-word buffer that contains additional information (If the packet does not include a buffer address, the Executive sends 8 words of 0.)

The receiver task automatically has access to the entire region as specified in W.NSTS. The sender task must be attached to the region with at least the same types of access. By setting all the bits in W.NSTS to 0, the receiver task can default the permitted access to that of the sender task.

If the directive specifies an event flag, the Executive sets the flag in the sender task -- when the receiver task acknowledges the reference -- by issuing the Receive By Reference directive (see Section 5.3.55). When the sender task exits, the system searches for any unreceived references that specify event flags, and prevents any invalid attempts to set the flags. The references themselves remain in the receiver task's receive-by-reference queues.

FORTRAN Call:

```
CALL SREF (tsk,[efn],iwdb,[isrb][,ids])
```

tsk	=	A single-precision, floating-point variable containing the name of the receiving task in Radix-50 format
efn	=	Event flag number
iwdb	=	An 8-word integer array containing a Window Definition Block (see Section 3.5.2.2)
isrb	=	An 8-word integer array containing additional information (If specified, the address of isrb is placed in iwdb(8). If isrb is omitted, the contents of iwdb(8) remain unchanged.)
ids	=	Directive status

Macro Call:

```
     SREF$    task,wdb[,efn]

       task   =  Name of the receiver task

       wdb    =  Window Definition Block address

       efn    =  Event flag number
```

Macro Expansion:

```
     SREF$    ALPHA,WDBADR,48.
     .BYTE    69.,5             ;SREF$ MACRO DIC, DPB SIZE=5 WORDS
     .RAD50   /ALPHA/           ;RECEIVER TASK NAME
     .WORD    48.               ;EVENT FLAG NUMBER
     .WORD    WDBADR            ;WDB ADDRESS
```

Window Definition Block Parameters:

Input parameters:

Array Element	Offset		
iwdb(4)	W.NRID	--	ID of the region to be sent by reference
iwdb(5)	W.NOFF	--	Offset word, passed without checking
iwdb(6)	W.NLEN	--	Length word, passed without checking
iwdb(7)	W.NSTS	--	Bit settings[1] in window status word (the receiver task's permitted access):

Bit	Definition
WS.RED	1 if read access is permitted
WS.WRT	1 if write access is permitted
WS.EXT	1 if extend access is permitted
WS.DEL	1 if delete access is permitted

iwdb(8)	W.NSRB	--	Optional address of an 8-word buffer containing additional information

Output parameters

None

Local Symbol Definitions:

S.RETN	--	Receiver task name (4)
S.REBA	--	Window Definition Block base address (2)
S.REEF	--	Event flag number (2)

1. If you are a FORTRAN programmer, refer to Section 3.5.2 to determine the bit values represented by the symbolic names described.

DSW Return Codes:

IS.SUC --	Successful completion.
IE.UPN --	A send packet or an attachment descriptor could not be allocated.
IE.INS --	The sender task attempted to send a reference to an Ancillary Control Processor (ACP) task, or task not installed.
IE.PRI --	Specified access not allowed to sender task itself.
IE.NVR --	Invalid region ID.
IE.IEF --	Invalid event flag number (EFN<0, or EFN>96 if group global event flags exist for the task; or EFN>64 if not).
IE.HWR --	Region had load failure or parity error.
IE.ADP --	The address check of the DPB, the WDB, or the send buffer failed.
IE.SDP --	DIC or DPB size is invalid.

Notes:

1. For the user's convenience, the ordering of the SREF$ macro arguments does not directly correspond to the format of the DPB. The arguments have been arranged so that the optional argument (efn) is at the end of the macro call. This arrangement is also compatible with the SDAT$ macro.

2. Because region attachment requires system dynamic memory, the receiver task should detach from any region to which it was already attached, in order to prevent depletion of the memory pool. That is, the task needs to be attached to a given region only once.

3. If the specified event flag is group global, then the use count for the event flag's group is incremented to prevent premature elimination of the event flags. The use count is run down when:

   - The packet is received.

   - The issuing task exits before the packet is received.

5.3.75 Specify Receive-by-Reference AST

The Specify Receive-By-Reference AST directive instructs the system to record one of the following two cases:

- Receive-by-reference ASTs for the issuing task are desired, and the Executive transfers control to a specified address when such an AST occurs.
- Receive-by-reference ASTs for the issuing task are no longer desired.

When the directive specifies an AST service routine entry point, receive-by-reference ASTs for the task will occur. The Executive will transfer control to the specified address.

When the directive omits an entry point address, the Executive stops the occurrence of receive-by-reference ASTs for the issuing task. Receive-by-reference ASTs will not occur until the task issues another Specify Receive-By-Reference AST directive that specifies an entry point address. See Notes below.

FORTRAN Call:

Neither the FORTRAN language nor the ISA standard permits direct linking to system trapping mechanisms; therefore, this directive is not available to FORTRAN tasks.

Macro Call:

```
SRRA$   [ast]

  ast  =  AST service routine entry point address (0)
```

Macro Expansion:

```
SRRA$   RECAST
.BYTE   21.,2              ;SRRA$ MACRO DIC, DPB SIZE=2 WORDS
.WORD   RECAST             ;ADDRESS OF RECEIVE AST
```

Local Symbol Definitions:

S.RRAE -- AST entry address (2)

DSW Return Codes:

IS.SUC -- Successful completion.

IE.UPN -- Insufficient dynamic memory.

IE.ITS -- AST entry point address is already unspecified.

IE.AST -- Directive was issued from an AST service routine, or ASTs are disabled.

IE.ADP -- Part of the DPB is out of the issuing task's address space.

IE.SDP -- DIC or DPB size is invalid.

Notes:

1. Specify Receive-By-Reference AST requires dynamic memory.

2. The Executive queues receive-by-reference ASTs when a message is sent to the task. Future receive-by-reference ASTs will not be queued for the task until the first one queued has been effected.

3. The task enters the receive-by-reference AST service routine with the task stack in the following state:

   ```
   SP+06 - Event flag mask word
   SP+04 - PS of task prior to AST
   SP+02 - PC of task prior to AST
   SP+00 - DSW of task prior to AST
   ```

   No trap-dependent parameters accompany a receive-by-reference AST; therefore, the AST Service Exit directive (see Section 5.3.4) must be executed with the stack in the same state as when the AST was effected.

4. This directive cannot be issued either from an AST service routine or when ASTs are disabled.

STAF$

5.3.76 Set Affinity

The Set Affinity directive can be issued by a task to select which CPU and UNIBUS run(s) to use during task execution. This directive can only be issued for RSX-11M-PLUS tasks that are executed on PDP-11 multiprocessor systems.

Task CPU/UNIBUS affinity enables a task to select which CPU and UNIBUS run(s) to use for task execution when running on PDP-11 multiprocessor systems. The programmer must be completely aware of the particular system hardware configuration in which the task will be executed before using these directives.

Task CPU/UNIBUS affinity can be established at three possible times:

1. When the task is installed
2. When the task is mapped into a device partition (which must have CPU/UNIBUS run affinity previously established)
3. When set by the Set Affinity directive

When issued, the Set Affinity directive produces an affinity mask word that defines task CPU/UNIBUS affinity. One bit in the word is set to select one CPU on which the task will be run. One or more of 12 additional bits can be set to select one or more UNIBUS runs for peripheral device use during task execution.

Two directives support task affinity, as follows:

- Set Affinity - This directive accepts parameters that define the CPU and UNIBUS run mask for task execution. At assembly time a 1-word mask is created consisting of the logical OR of all the parameters.
- Remove Affinity - This directive removes task CPU/UNIBUS affinity previously established by a Set Affinity directive.

A 1-word CPU/UNIBUS affinity mask defines directive parameters. Parameters enable specification of one of four (maximum) CPUs and one or more of twelve (maximum) UNIBUS runs. The affinity mask word consists of the logical OR of all the parameters. Only one parameter (cp or ub) is required. Directive parameters are assembled to produce the mask word bit values shown as follows:

Directive Parameter	Mask Word Function	Assembled Bit Value
CPA	Select CPU "A"	1
CPB	Select CPU "B"	2
CPC	Select CPU "C"	4
CPD	Select CPU "D"	10
UBE	Select UNIBUS run "E"	20
UBF	Select UNIBUS run "F"	40
UBH	Select UNIBUS run "H"	100
UBJ	Select UNIBUS run "J"	200

Directive Parameter	Mask Word Function	Assembled Bit Value
UBK	Select UNIBUS run "K"	400
UBL	Select UNIBUS run "L"	1000
UBM	Select UNIBUS run "M"	2000
UBN	Select UNIBUS run "N"	4000
UBP	Select UNIBUS run "P"	10000
UBR	Select UNIBUS run "R"	20000
UBS	Select UNIBUS run "S"	40000
UBT	Select UNIBUS run "T"	100000

FORTRAN Call:

```
CALL STAF (iaff[,ids])
```

iaff	=	Affinity mask word
ids	=	Integer to receive Directive Status Word

Macro Call:

```
STAF$    [cp!ub!ub...]
```

cp	=	CPU selected (A through D, as previously listed)
ub	=	UNIBUS run selected (E through T, as previously listed)

Macro Expansion:

```
STAF$   CPB!UBF!UBJ
.BYTE   161.,2          ;STAF$ MACRO DIC, DPB SIZE=2 WORDS
.WORD   242             ; AFFINITY MASK WORD ('OR' OF PARAMETERS)
```

Local Symbol Definitions:

S.AFAF -- Affinity mask word (2)

DSW Return Codes:

IS.SUC -- Successful completion.

IE.ITS -- Task installed with affinity .

IE.ADP -- Part of the DPB is out of the issuing task's address space.

IE.SDP -- DIC or DPB size is invalid.

Notes:

1. A task that is installed with task affinity must not issue this directive. Any attempt to do so results in an IE.ITS error returned.

2. If this directive is issued with parameters that prevent the task from running, an IE.ITS error is returned.

5.3.77 Set System Time Directive

The Set System Time directive instructs the system to set the system's internal time to the specified time parameters. Optionally, the Set System Time directive returns the system's current internal time to the issuing task before setting it to the specified values.

All time parameters must be specified as binary numbers.

A task must be privileged to issue this directive.

When this directive changes the system time by a specified amount, it also effectively changes the time of anything resident on the clock queue by the same amount. Thus, the time synchronization of events is maintained.

FORTRAN Call:

```
     CALL SETTIM (ibufn[,ibufp[,ids]])

       ibufn  =  An 8-word integer array -- new time specification
                 buffer

       ibufp  =  An 8-word integer array -- previous time buffer

       ids    =  Directive status
```

Macro Call:

```
     STIM$ bufn,[bufp]

       bufn  =  Address of 8-word new time specification buffer

       bufp  =  Address of 8-word buffer to receive the
                previous system time parameters
```

Buffer Format:

Word 0	--	Year (since 1900).
Word 1	--	Month (1-12).
Word 2	--	Day (1-n, where n is the highest day possible for the given month and year).
Word 3	--	Hour (0-23).
Word 4	--	Minute (0-59).
Word 5	--	Second (0-59).
Word 6	--	Tick of second (0-n, where n is the frequency of the system clock minus one). If the next parameter (ticks per second) is defaulted, this parameter is ignored.
Word 7	--	Ticks per second (must be defaulted or must match the frequency of the system clock). This parameter is used to verify the intended granularity of the "tick of second" parameter.

NOTE

If any of the specified new time parameters are defaulted (equal to -1), the corresponding previous system time parameters will remain unchanged and will be substituted for the defaulted parameters during argument validation.

Macro Expansion:

```
     STIM$   NEWTIM,OLDTIM
     .BYTE   61.,3            ;STIM$ DIC, DPB SIZE=3 WORDS
     .WORD   NEWTIM           ;ADDRESS OF 8.-WORD INPUT BUFFER
     .WORD   OLDTIM           ;ADDRESS OF 8.-WORD OUTPUT BUFFER
```

Local Symbol Definitions:

S.TIBA -- Input buffer address (2)

S.TIBO -- Output buffer address (2)

The following offsets are assigned relative to the start of each time parameters buffer:

S.TIYR -- Year (2)

S.TIMO -- Month (2)

S.TIDA -- Day (2)

S.TIHR -- Hour (2)

S.TIMI -- Minute (2)

S.TICS -- Second (2)

S.TICT -- Clock tick of second (2)

S.TICP -- Clock ticks per second (2)

DSW Return codes:

IS.SUC -- Successful completion.

IE.PRI -- The issuing task is not privileged.

IE.ITI -- One of the specified time parameters is out of range, or both the tick-of-second parameter and the ticks-per-second parameter were specified and the ticks-per-second parameter does not match the system's clock frequency. The system time at the moment the directive is issued (returned in the second buffer) can be useful in determining the cause of the fault if any of the specified time parameters were defaulted.

IE.ADP -- Part of the DPB or one of the buffers is out of the issuing task's address space.

IE.SDP -- DIC or DPB size is invalid.

Notes:

1. Execution of this directive generates an Error Log packet and sends it to the Error Logging subsystem.

2. On an RSX-11M-PLUS system with accounting active, this directive causes an accounting transaction that records both the old and new time.

3. The highest clock frequency supported by the operating system is 1000 hz for a programmable clock. Note that as the clock frequency approaches this value, the maximum resolution for this directive becomes more time critical. The accuracy of this directive depends upon the elapsed time between the moment that a new system time is specified and the time that the directive actually traps to the Executive.

4. The buffers used in this directive are compatible with those of the Get Time Parameters (GTIM$) directive.

5. The second buffer (previous time) is only filled in if the directive was successfully executed or if it was rejected with an error code of IE.ITI.

STLO$

5.3.78 Stop For Logical OR Of Event Flags

The Stop For Logical OR Of Event Flags directive instructs the system to stop the issuing task until the Executive sets one or more of the indicated event flags from one of the following groups:

GR 0	--	Local flags 1-16
GR 1	--	Local flags 17-32
GR 2	--	Common flags 33-48
GR 3	--	Common flags 49-64
GR 4	--	Group global flags 65-80
GR 5	--	Group global flags 81-96

The task does not stop itself if any of the indicated flags are already set when the task issues the directive. This directive cannot be issued at AST state.

A task that is stopped for one or more event flags can only become unstopped by setting the specified event flag; it cannot become unstopped with the Unstop directive or the MCR Unstop or DCL START command.

FORTRAN Call:

```
CALL STLOR (iefl,ief2,ief3, ...  ief(n))

  iefl ... ief(n)  =  List of event flag numbers
```

Macro Call:

```
STLO$   grp, msk

  grp  =  Desired group of event flags

  msk  =  A 16-bit mask word
```

Macro Expansion:

```
STLO$   1,47
.BYTE   137.,3          ;STLO$ MACRO DIC, DPB SIZE=3 WORDS
.WORD   1               ;GROUP 1 FLAGS (FLAGS 17-32)
.WORD   47              ;MASK WORD = 47 (FLAGS 17, 18, 19, 22)
```

Local Symbol Definitions:

S.TLGR	--	Group flags (2)
S.TLMS	--	Mask word (2)

DSW Return Codes:

IS.SUC	--	Successful completion.
IE.AST	--	The issuing task is at AST state.
IE.IEF	--	An event flag group other than 0 thru 5 was specified, or the event flag mask word is zero.
IE.ADP	--	Part of the DPB is out of the issuing task's address space.
IE.SDP	--	DIC or DPB size is invalid.

Notes:

1. There is a one-to-one correspondence between bits in the mask word and the event flags in the specified group. That is, if group 1 were specified (as in the above macro expansion example), bit 0 in the mask word would correspond to event flag 17, bit 1 to event flag 18, and so forth.

2. The Executive does not arbitrarily clear event flags when Stop For Logical OR Of Event Flags conditions are met. Some directives (Queue I/O Request, for example) implicitly clear a flag; otherwise, they must be explicitly cleared by a Clear Event Flag directive.

3. The argument list specified in the FORTRAN call must contain only event flag numbers that lie within one event flag group. If event flag numbers are specified that lie in more than one group, or if an invalid event flag number is specified, a fatal FORTRAN error is generated.

4. Tasks stopped for event flag conditions cannot be unstopped by issuing the Unstop directive; tasks stopped in this manner can only be unstopped by meeting event flag conditions.

5. The grp operand must always be of the form n regardless of the macro form used. In all other macro calls, numeric or address values for $S form macros have the form:

   ```
   #n
   ```

   For STLO$S this form of the grp argument would be:

   ```
   n
   ```

6. If the specified event flag group is group global, the group's use count is incremented to prevent premature elimination of the event flags. The use count is run down when:

   - The Stop For condition is satisfied.
   - The issuing task exits before the Stop For condition is satisfied.

STOP$S

5.3.79 Stop ($S Form Recommended)

The Stop directive stops the issuing task. This directive cannot be issued at AST state. A task stopped in this manner can only be unstopped by: another task that issues an Unstop directive directed to this task; this task issuing an Unstop directive at AST state; or the MCR Unstop or DCL START command.

FORTRAN Call:

```
     CALL STOP ([ids])

       ids    =  Integer to receive the directive status word
```

Macro Call:

```
     STOP$S
```

Macro Expansion:

```
     STOP$S
     MOV      (PC)+,-(SP)        ;PUSH DPB ONTO THE STACK
     .BYTE    131.,1             ;STOP$ MACRO DIC, DPB SIZE=1 WORD
     EMT      377                ;TRAP TO THE EXECUTIVE
```

Local Symbol Definitions:

None

DSW Return Codes:

IS.SET	--	Successful completion.
IE.AST	--	The issuing task is at AST state.
IE.ADP	--	Part of the DPB is out of the issuing task's address space.
IE.SDP	--	DIC or DPB size is invalid.

5.3.80 Stop For Single Event Flag

The Stop For Single Event Flag directive instructs the system to stop the issuing task until the specified event flag is set. If the flag is set at issuance, the task is not stopped. This directive can not be issued at the AST state.

A task that is stopped for one or more event flags can only become unstopped by setting the specified event flag; it cannot become unstopped by the Unstop directive or the MCR Unstop or DCL START command.

FORTRAN Call:

```
     CALL STOPFR (iefn[,ids])

       iefn  =  Event flag number

       ids   =  Integer to receive directive status word
```

Macro Call:

```
     STSE$   efn

       efn  =  Event flag number
```

Macro Expansion:

```
     STSE$    7
     .BYTE    135.,2           ;STSE$ MACRO DIC, DPB SIZE=2 WORDS
     .WORD    7                ;LOCAL EVENT FLAG NUMBER = 7
```

Local Symbol Definitions:

S.TSEF -- Event flag number (2)

DSW Return Codes:

IS.SUC	-- Successful completion.
IE.AST	-- The issuing task is at AST state.
IE.IEF	-- Invalid event flag number (EFN<1, or EFN>96 if group global event flags exist for the task's group; or EFN>64 if not).
IE.ADP	-- Part of the DPB is out of the issuing task's address space.
IE.SDP	-- DIC or DPB size is invalid.

Note:

If the specified event flag is group global, the use count for the event flag's group is incremented to prevent premature elimination of event flags. The use count is run down when:

- The Stop For condition is satisfied.
- The issuing task exits before the Stop For condition is satisfied.

SVDB$

5.3.81 Specify SST Vector Table For Debugging Aid

The Specify SST Vector Table For Debugging Aid directive instructs the system to record the address of a table of SST service routine entry points for use by an intratask debugging aid (ODT, for example).

To deassign the vector table, omit the parameters adr and len from the macro call.

Whenever an SST service routine entry is specified in both the table used by the task and the table used by a debugging aid, the trap occurs for the debugging aid, not for the task.

FORTRAN Call:

Neither the FORTRAN language nor the ISA standard permits direct linking to system trapping mechanisms; therefore, this directive is not available to FORTRAN tasks.

Macro Call:

```
SVDB$   [adr][,len]
```

adr	=	Address of SST vector table
len	=	Length of (that is, number of entries in) the table in words

The vector table has the following format:

Word 0	--	Odd address of nonexistent memory error
Word 1	--	Memory protect violation
Word 2	--	T-bit trap or execution of a BPT instruction
Word 3	--	Execution of an IOT instruction
Word 4	--	Execution of a reserved instruction
Word 5	--	Execution of a non-RSX EMT instruction
Word 6	--	Execution of a TRAP instruction
Word 7	--	PDP-11/40 floating-point exception

A 0 entry in the table indicates that the task does not want to process the corresponding SST.

Macro Expansion:

```
SVDB$   SSTTBL,4
.BYTE   103.,3          ;SVDB$ MACRO DIC, DPB SIZE=3 WORDS
.WORD   SSTTBL          ;ADDRESS OF SST TABLE
.WORD   4               ;SST TABLE LENGTH=4 WORDS
```

Local Symbol Definitions:

S.VDTA	--	Table address (2)
S.VDTL	--	Table length (2)

DSW Return Codes:

IS.SUC	--	Successful completion.
IE.ADP	--	Part of the DPB or table is out of the issuing task's address space.
IE.SDP	--	DIC or DPB size is invalid.

SVTK$

5.3.82 Specify SST Vector Table For Task

The Specify SST Vector Table For Task directive instructs the system to record the address of a table of SST service routine entry points for use by the issuing task.

To deassign the vector table, omit the parameters adr and len from the macro call.

Whenever an SST service routine entry is specified in both the table used by the task and the table used by a debugging aid, the trap occurs for the debugging aid, not for the task.

FORTRAN Call:

> Neither the FORTRAN language nor the ISA standard permits direct linking to system trapping mechanism; therefore, this directive is not available to FORTRAN tasks.

Macro Call:

```
     SVTK$    [adr][,len]

       adr  =  Address of SST vector table

       len  =  Length of (that is, number of entries in) the table  in
               words
```

The vector table has the following format:

```
     Word  0  --  Odd address of nonexistent memory error

     Word  1  --  Memory protect violation

     Word  2  --  T-bit trap or execution of a BPT instruction

     Word  3  --  Execution of an IOT instruction

     Word  4  --  Execution of a reserved instruction

     Word  5  --  Execution of a non-RSX EMT instruction

     Word  6  --  Execution of a TRAP instruction

     Word  7  --  PDP-11/40 floating-point exception
```

A 0 entry in the table indicates that the task does not want to process the corresponding SST.

Macro Expansion:

```
     SVTK$    SSTTBL,4
     .BYTE    105.,3              ;SVTK$ MACRO DIC, DPB SIZE=3 WORDS
     .WORD    SSTTBL              ;ADDRESS OF SST TABLE
     .WORD    4                   ;SET TABLE LENGTH=4 WORDS
```

Local Symbol Definitions:

S.VTTA	--	Table address (2)
S.VTTL	--	Table length (2)

DSW Return Codes:

IS.SUC	--	Successful completion.
IE.ADP	--	Part of the DPB or table is out of the issuing task's address space.
IE.SDP	--	DIC or DPB size is invalid.

ULGF$

5.3.83 Unlock Group Global Event Flags ($S Form Recommended)

The Unlock Group Global Event Flags directive instructs the Executive to decrement the use count of the group global event flags for the issuing task's protection group UIC (H.CUIC+1). This unlocks flags that were locked by the Create Group Global Event Flags directive.

A task may only unlock the event flags once before locking them again.

The group global event flags are eliminated if the following two conditions are satisfied:

- The use count in the group global event flag control block (GFB) is zero after this directive is issued.
- The GFB is marked for deletion.

FORTRAN Call:

```
CALL ULGF ([ids])

  ids  =  Directive status
```

Macro Call:

```
ULGF$S   [err]

  err  =  Error routine address
```

Macro Expansion:

```
ULGF$S  ERR
MOV     (PC)+,-(SP)                ;PUSH DPB ONTO THE STACK
.BYTE   159.,1                     ;DIC=159., DPB SIZE= 1 WORD
EMT     377                        ;TRAP TO THE EXECUTIVE
BCC     .+6                        ;BRANCH IF DIRECTIVE SUCCESSFUL
JSR     PC,ERR                     ;OTHERWISE, CALL ROUTINE "ERR"
```

Local Symbol Definitions:

None.

DSW Return Codes:

IS.SUC	--	Successful completion.
IE.RSU	--	Event flags already unlocked from the issuing task.
IE.ADP	--	Part of the DPB is out of the issuing task's address space.
IE.SDP	--	DIC or DPB size is invalid.

5.3.84 Unmap Address Window

The Unmap Address Window directive unmaps a specified window. After the window has been unmapped, references to the corresponding virtual addresses are invalid and cause a processor trap to occur.

FORTRAN Call:

```
     CALL UNMAP (iwdb[,ids])

       iwdb  =  An 8-word integer array containing a Window Definition
                Block (see Section 3.5.2.2)

       ids   =  Directive status
```

Macro Call:

```
     UMAP$   wdb

       wdb  =  Window Definition Block address
```

Macro Expansion:

```
     UMAP$   WDBADR
     .BYTE   123.,2           ;UMAP$ MACRO DIC, DPB SIZE=2 WORDS
     .WORD   WDBADR           ;WDB ADDRESS
```

Window Definition Block Parameters:

Input parameters:

Array Element	Offset	
iwdb(1) bits 0-7	W.NID	-- ID of the window to be unmapped

Output parameters

iwdb(7) W.NSTS -- Bit settings[1] in the window status word:

Bit	Definition
WS.UNM	1 if the window was successfully unmapped

Local Symbol Definitions:

U.MABA -- Window Definition Block address (2)

1. If you are a FORTRAN programmer, refer to Section 3.5.2 to determine the bit values represented by the symbolic names described.

DSW Return Codes:

IS.SUC	--	Successful completion.
IE.ITS	--	The specified address window is not mapped.
IE.NVW	--	Invalid address window ID.
IE.ADP	--	DPB or WDB out of range.
IE.SDP	--	DIC or DPB size is invalid.

5.3.85 Unstop Task

The Unstop Task directive unstops the specified task that has stopped itself by either the Stop or the Receive Data Or Stop directive. It does not unstop tasks stopped for event flag(s) or tasks stopped for buffered I/O. If the Unstop directive is issued to a task previously stopped by means of the Stop or Receive Or Stop directive while at task state, and the task is presently at AST state, the task only becomes unstopped when it returns to task state.

It is considered the responsibility of the unstopped task to determine if it has been validly unstopped.

The Unstop directive does not cause a significant event.

FORTRAN Call:

```
     CALL USTP (rtname[,ids])

       rtname  =  Name of task to be unstopped

       ids     =  Integer to receive directive status information
```

Macro Call:

```
     USTP$   tname

       tname  =  Name of task to be unstopped
```

Macro Expansion:

```
     USTP$   ALPHA
     .BYTE   133.,3            ;USTP$ MACRO DIC, DPB SIZE=3 WORDS
     .RAD50  /ALPHA/           ;NAME OF TASK TO BE UNSTOPPED
```

Local Symbol Definitions:

U.STTN	--	Task name (4)

DSW Return Codes:

IS.SUC	--	Successful completion.
IE.INS	--	The specified task is not installed in the system.
IE.ACT	--	The specified task is not active.
IE.ITS	--	The specified task is not stopped, or it is stopped for event flag(s) or buffered I/O.
IE.ADP	--	Part of the DPB is out of the issuing task's address space.
IE.SDP	--	DIC or DPB size is invalid.

VRCD$

5.3.86 Variable Receive Data

The Variable Receive Data directive instructs the system to dequeue a variable-length data block for the issuing task; the data block has been queued (FIFO) for the task by a Variable Send Data directive. When a sender task is specified, only data sent by the specified task is received.

Buffer size can be 256. words maximum. If no buffer size is specified, the buffer size is 13. words. If a buffer size greater than 256. is specified, an IE.IBS error is returned.

A 2-word sender task name (in Radix-50 form) and the data block are returned in the specified buffer, with the task name in the first two words. For this reason, the storage you allocate within the buffer should be two words greater than the size of the data portion of the message specified in the directive.

Variable-length data blocks are transferred from the sending task to the receiving task by means of buffers in the secondary pool.

FORTRAN Call:

```
CALL VRCD ([task],bufadr,[buflen][,ids])
```

task	=	Sender task name
buf	=	Address of buffer to receive the sender task name and data
buflen	=	Length of buffer
ids	=	Integer to receive the directive status word.

If the directive was successful, it returns the number of words transferred into the user buffer. If the directive execution encountered an error, it returns the error code in the ids parameter.

Any error return of the form IE.XXX is a negative word value. If the status is positive, the value of the status word is the number of words transferred including the taskname. For example, if you specify a buffer size of 13 in the VRCD$ call, the value returned in the directive status word is 15 (13 words of data plus the two words needed to return the taskname).

Macro Call:

```
VRCD$     [task],bufadr[,buflen]
```

task	=	Sender task name
bufadr	=	Buffer address
buflen	=	Buffer size in words

Macro Expansion:

```
     VRCD$      SNDTSK,DATBUF,BUFSIZ
     .BYTE      75.,6             ;VRCD$ MACRO DIC, DPB SIZE=6 WORDS
     .RAD50     /SNDTSK/          ;SENDER TASK NAME
     .WORD      DATBUF            ;ADDRESS OF DATA BUFFER
     .WORD      BUFSIZ            ;BUFFER SIZE
```

Local Symbol Definitions:

R.VDTN	--	Sender task name (4)
R.VDBA	--	Buffer address (2)
R.VDBL	--	Buffer length (2)
R.VDTI	--	Reserved (2)

DSW Return Codes:

IS.SUC	--	Successful completion.
IE.INS	--	Specified task not installed.
IE.ITS	--	No data in task's receive queue.
IE.RBS	--	Receive buffer is too small.
IE.IBS	--	Invalid buffer size specified (greater than 255.).
IE.ADP	--	Part of the DPB or buffer is out of the issuing task's address space.
IE.SDP	--	DIC or DPB size is invalid.

VRCS$

5.3.87 Variable Receive Data Or Stop

The Variable Receive Data Or Stop directive instructs the system to dequeue a variable-length data block for the issuing task; the data block has been queued (FIFO) for the task by a Variable Send Data directive. If there is no such packet to be dequeued, the issuing task is stopped. In this case, another task (the sender task) is expected to issue an Unstop directive after sending the data. When stopped in this manner, the directive status returned is IS.SET, indicating that the task was stopped and that no data has been received; however, since the task must be unstopped in order to see this status, the task can now reissue the Variable Receive Data Or Stop directive to actually receive the data packet.

When a sender task is specified, only data sent by the specified task is received.

Buffer size can be 256. words maximum. If no buffer size is specified, the buffer size is 13. words. If a buffer size greater than 256. is specified, an IE.IBS error is returned.

A 2-word sender task name (in Radix-50 form) and the data block are returned in the specified buffer, with the task name in the first 2 words. For this reason, the storage you allocate within the buffer should be two words greater than the size of the data portion of the message specified in the directive.

Variable-length data blocks are transferred from the sending task to the receiving task by means of buffers in the secondary pool.

FORTRAN Call:

```
CALL VRCS ([task],bufadr,[buflen][,ids])
```

task	=	Sender task name
buf	=	Address of buffer to receive the sender task name and data
buflen	=	Length of buffer
ids	=	Integer to receive the directive status word

If the directive was successful, it returns the number of words transferred into the user buffer. If the directive execution encountered an error, it returns the error code in the ids parameter.

Any error return of the form IE.XXX is a negative word value. If the status is positive, the value of the status word is the number of words transferred including the taskname. For example, if you specify a buffer size of 13 in the VRCS$ call, the value returned in the directive status word is 15 (13 words of data plus the two words needed to return the taskname).

Macro Call:

```
     VRCS$      [task],bufadr[,buflen]

       task   =  Sender task name

       bufadr =  Buffer address

       buflen =  Buffer size in words
```

Macro Expansion:

```
     VRCS$      SNDTSK,DATBUF,BUFSIZ
     .BYTE      139.,6           ;VRCS$ MACRO DIC, DPB SIZE=6 WORDS
     .RAD50     /SNDTSK/         ;SENDER TASK NAME
     .WORD      DATBUF           ;ADDRESS OF DATA BUFFER
     .WORD      BUFSIZ           ;BUFFER SIZE
```

Local Symbol Definitions:

R.VSTN	--	Sender task name (4)
R.VSBA	--	Buffer address (2)
R.VSBL	--	Buffer length (2)
R.VSTI	--	Reserved (2)

DSW Return Codes:

IS.SUC	--	Successful completion.
IE.INS	--	Specified task not installed.
IE.RBS	--	Receive buffer is too small.
IE.IBS	--	Invalid buffer size specified (greater than 255.).
IE.ADP	--	Part of the DPB or buffer is out of the issuing task's address space.
IE.SDP	--	DIC or DPB size is invalid.

VRCX$

5.3.88 Variable Receive Data Or Exit

The Variable Receive Data Or Exit directive instructs the system to dequeue a variable-length data block for the issuing task; the data block has been queued (FIFO) for the task by a Variable Send Data directive. When a sender task is specified, only data sent by the specified task is received.

A 2-word sender task name (in Radix-50 form) and the data block are returned in the specified buffer, with the task name in the first 2 words. For this reason, the storage you allocate within the buffer should be two words greater than the size of the data portion of the message specified in the directive.

If no data has been sent, a task exit occurs. To prevent the possible loss of send data packets, the user should not rely on I/O rundown to take care of any outstanding I/O or open files; the task should assume this responsibility.

Buffer size can be 256. words maximum. If no buffer size is specified, the buffer size is 13. words. If a buffer size greater than 256. is specified, an IE.IBS error is returned.

Variable-length data blocks are transferred from the sending task to the receiving task by means of buffers in the secondary pool.

FORTRAN Call:

```
CALL VRCX ([task],bufadr,[buflen][,ids])
```

task	=	Sender task name
buf	=	Address of buffer to receive the sender task name and data
buflen	=	Length of buffer
ids	=	Integer to receive the directive status word

If the directive was successful, it returns the number of words transferred into the user buffer. If the directive execution encountered an error, it returns the error code in the ids parameter.

Any error return of the form IE.XXX is a negative word value. If the status is positive, the value of the status word is the number of words transferred including the taskname. For example, if you specify a buffer size of 13 in the VRCX$ call, the value returned in the directive status word is 15 (13 words of data plus the two words needed to return the taskname).

Macro Call:

```
VRCX$     [task],bufadr[,buflen]
```

task	=	Sender task name
bufadr	=	Buffer address
buflen	=	Buffer size in words

Macro Expansion:

```
VRCX$     SNDTSK,DATBUF,BUFSIZ
.BYTE     77.,6               ;VRCX$ MACRO DIC, DPB SIZE=6 WORDS
.RAD50    /SNDTSK/            ;SENDER TASK NAME
.WORD     DATBUF              ;ADDRESS OF DATA BUFFER
.WORD     BUFSIZ              ;BUFFER SIZE
```

Local Symbol Definitions:

R.VXTN	--	Sender task name (4)
R.VXBA	--	Buffer address (2)
R.VXBL	--	Buffer length (2)
R.VXTI	--	Reserved (2)

DSW Return Codes:

IS.SUC	--	Successful completion.
IE.INS	--	Specified task not installed.
IE.RBS	--	Receive buffer is too small.
IE.IBS	--	Invalid buffer size specified (greater than 255.).
IE.ADP	--	Part of the DPB or buffer is out of the issuing task's address space.
IE.SDP	--	DIC or DPB size is invalid.

VSDA$

5.3.89 Variable Send Data

The Variable Send Data directive instructs the system to queue a variable-length data block for the specified task to receive.

Buffer size can be 256. words maximum. If no buffer size is specified, the buffer size is 13. words. If a buffer size greater than 256. is specified, an IE.IBS error is returned.

When an event flag is specified, a significant event is declared if the directive is successfully executed, and the indicated event flag is set for the sending task.

Variable-length data blocks are transferred from the sending task to the receiving task by buffers in the secondary pool.

FORTRAN Call:

```
     CALL VSDA (task,bufadr,[buflen],[efn][,ids])
```

task	=	Receiver task name
buf	=	Address of buffer to receive the receiver task name and data
buflen	=	Length of buffer
efn	=	Event flag number
ids	=	Integer to receive the directive status word

Macro Call:

```
     VSDA$    task,bufadr[,buflen][,efn]
```

task	=	Receiver task name
bufadr	=	Buffer address
buflen	=	Buffer size in words

Macro Expansion:

```
     VSDA$    RECTSK,DATBUF,BUFSIZ,4
     .BYTE    71.,8           ;VSDA$ MACRO DIC, DPB SIZE=8 WORDS
     .RAD50   /RECTSK/        ;RECEIVER TASK NAME
     .WORD    DATBUF          ;ADDRESS OF DATA BUFFER
     .WORD    4               ;EVENT FLAG 4
     .WORD    BUFSIZ          ;BUFFER SIZE
```

Local Symbol Definitions:

S.DATN	--	Sender task name (4)
S.DABA	--	Buffer address (2)
S.DAEF	--	Event flag number (2)
S.DABL	--	Buffer length (2)
S.DATI	--	Reserved (2)

DSW Return Codes:

IS.SUC	--	Successful completion.
IE.UPN	--	Insufficient dynamic storage.
IE.INS	--	Specified task not installed.
IE.IBS	--	Invalid buffer size specified (greater than 255.).
IE.IEF	--	Invalid event flag number (EFN<0 or EFN>96).
IE.ADP	--	Part of the DPB or buffer is out of the issuing task's address space.
IE.SDP	--	DIC or DPB size is invalid.

VSRC$

5.3.90 Variable Send, Request and Connect

The Variable Send, Request and Connect directive performs a Variable Send Data to the specified task, requests the task if it is not already active, and then connects to the task. The receiver task normally returns status by the Emit Status or the Exit With Status directive.

Buffer size can be 256. words maximum. If no buffer size is specified, the buffer size is 13. words. If a buffer size greater than 256. is specified, an IE.IBS error is returned.

FORTRAN Call:

```
     CALL VSRC (rtname, ibuf,[ibuflen],[iefn],[iast],[iesb],[iparm][,ids])
```

rtname	=	Target task name of the offspring task to be connected
ibuf	=	Name of send buffer
ibuflen	=	Length of the buffer
iefn	=	Event flag to be set when the offspring task exits or emits status
iast	=	Name of an AST routine to be called when the offspring task exits or emits status
iesb	=	Name of an 8-word status block to be written when the offspring task exits or emits status

```
Word    0 -- Offspring task exit status

Word    1 -- TKTN abort code

Word  2-7 -- Reserved
```

NOTE

The exit status block defaults to one word. To use the 8-word exit status block, you must specify the logical or of the symbol SP.WX8 and the event flag number in the iefn parameter above.

iparm	=	Name of a word to receive the status block address when an AST occurs
ids	=	Integer to receive the Directive Status Word

Macro Call:

```
     VSRC$    tname,buf[,buflen],[efn],[east],esb]
```

tname	=	Target task name of the offspring task to be connected
buf	=	Address of send buffer
buflen	=	Length of buffer
efn	=	The event flag to be cleared on issuance and set when the offspring task exits or emits status
east	=	Address of an AST routine to be called when the offspring task exits or emits status
esb	=	Address of an 8-word status block to be written when the offspring task exits or emits status

Word 0 -- Offspring task exit status

Word 1 -- TKTN abort code

Word 2-7 -- Reserved

NOTE

The exit status block defaults to 1 one word. To use the 8-word exit status block, you must specify the logical or of the symbol SP.WX8 and the event flag number in the efn parameter above.

Macro Expansion:

```
     VSRC$    ALPHA,BUFFR,BUFSIZE,2,SDRCTR,STBLK
     .BYTE    141.,8         ;VSRC$ MACRO DIC, DPB SIZE=8 WORDS
     .RAD50   /ALPHA/        ;TARGET TASK NAME
     .WORD    BUFFR          ;SEND BUFFER ADDRESS
     .BYTE    2              ;EVENT FLAG NUMBER = 2
     .BYTE    16.            ;EXIT STATUS BLOCK CONSTANT
     .WORD    BUFSIZE        ;LENGTH OF BUFFER IN BYTES
     .WORD    SDRCTR         ;ADDRESS OF AST ROUTINE
     .WORD    STBLK          ;ADDRESS OF STATUS BLOCK
```

Local Symbol Definitions:

S.DRTN -- Task name (4)

S.DRBF -- Buffer address (2)

S.DREF -- Event flag (2)

S.DREA -- AST routine address (2)

S.DRES -- Status block address (2)

DSW Return Codes:

IS.SUC --	Successful completion.
IE.UPN --	Insufficient dynamic memory to allocate a send packet, Offspring Control Block, Task Control Block, or Partition Control Block.
IE.INS --	The specified task is an ACP or has the no-send attribute.
IE.IEF --	An invalid event flag number was specified (EFN<0 or EFN>96 if group global event flags exist. EFN>64 if not.).
IE.ADP --	Part of the DPB or exit status block is not in the issuing task's address space.
IE.SDP --	DIC or DPB size is invalid.

Notes:

1. If the specified event flag is group global, the use count for the event flag's group is incremented to prevent premature elimination of the event flags. The use count is run down when:

   - Status is returned from the connected task.
   - The issuing task exits before status is returned.

2. Changing the virtual mapping of the exit status block while the connection is in effect may result in obscure errors.

5.3.91 Wait For Significant Event ($S Form Recommended)

The Wait For Significant Event directive is used to suspend the execution of the issuing task until the next significant event occurs. It is an especially effective way to block a task that cannot continue because of a lack of dynamic memory, since significant events occurring throughout the system often result in the release of dynamic memory. The execution of a Wait For Significant Event directive does not itself constitute a significant event.

FORTRAN Call:

```
CALL WFSNE
```

Macro Call:

```
WSIG$S  [err]

  err  =  Error routine address
```

Macro Expansion:

```
WSIG$S   ERR
MOV      (PC)+,-(SP)        ;PUSH DPB ONTO THE STACK
.BYTE    49.,1              ;WSIG$S MACRO DIC, DPB SIZE=1 WORD
EMT      377                ;TRAP TO THE EXECUTIVE
BCC      .+6                ;BRANCH IF DIRECTIVE SUCCESSFUL
JSR      PC,ERR             ;OTHERWISE, CALL ROUTINE "ERR"
```

Local Symbol Definitions:

None

DSW Return Codes:

IS.SUC	--	Successful completion.
IE.ADP	--	Part of the DPB is out of the issuing task's address space.
IE.SDP	--	DIC or DPB size is invalid.

Notes:

1. If a directive is rejected for lack of dynamic memory, this directive is the only technique available for blocking task execution until dynamic memory may again be available.

2. The wait state induced by this directive is satisfied by the first significant event to occur after the directive has been issued. The significant event that occurs may or may not be related to the issuing task.

3. Because this directive requires only a 1-word DPB, the $S form of the macro is recommended. It requires less space and executes with the same speed as that of the DIR$ macro.

4. Significant events include the following:

   - I/O completion
   - Task exit
   - Execution of a Send Data directive
   - Execution of a Send Data, Request and Pass OCB directive
   - Execution of a Send, Request and Connect directive
   - Execution of a Send By Reference directive or a Receive by Reference directive
   - Execution of an Alter Priority directive
   - Removal of an entry from the clock queue (for instance, resulting from the execution of a Mark Time directive or the issuance of a rescheduling request)
   - Execution of a Declare Significant Event directive
   - Execution of the round-robin scheduling algorithm at the end of a round-robin scheduling interval
   - Execution of an Exit, an Exit with Status, or Emit Status directive

5.3.92 Wait For Logical OR Of Event Flags

The Wait For Logical OR Of Event Flags directive instructs the system to block the execution of the issuing task until the Executive sets the indicated event flags from one of the following groups:

GR 0	--	Flags 1-16
GR 1	--	Flags 17-32
GR 2	--	Flags 33-48
GR 3	--	Flags 49-64
GR 4	--	Flags 65-80
GR 5	--	Flags 81-96

The task does not block itself if any of the indicated flags are already set when the task issues the directive. See Notes below.

FORTRAN Call:

```
CALL WFLOR (efn1,efn2,...efnn)
```

efn = List of event flag numbers taken as the set of flags to be specified in the directive

Macro Call:

```
WTLO$   grp,msk
```

grp = Desired group of event flags

msk = A 16-bit flag mask word

Macro Expansion:

```
WTLO$   2,160003
.BYTE   43.,3               ;WTLO$ MACRO DIC, DPB SIZE=3 WORDS
.WORD   2                   ;FLAGS SET NUMBER 2 (FLAGS 33:48.)
.WORD   160003              ;EVENT FLAGS 33,34,46,47 AND 48.
```

Local Symbol Definitions:

None

DSW Return Codes:

IS.SUC -- Successful completion.

IE.IEF -- No event flag specified in the mask word or flag group indicator other than 0, 1, 2, 3, 4, or 5.

IE.ADP -- Part of the DPB is out of the issuing task's address space.

IE.SDP -- DIC or DPB size is invalid.

Notes:

1. There is a one-to-one correspondence between bits in the mask word and the event flags in the specified group. That is, if group 1 were specified, then bit 0 in the mask word would correspond to event flag 17, bit 1 to event flag 18, and so forth.

2. The Executive does not arbitrarily clear event flags when Wait For conditions are met. Some directives (Queue I/O Request, for example) implicitly clear a flag; otherwise, they must be explicitly cleared by a Clear Event Flag directive.

3. The grp operand must always be of the form n regardless of the macro form used. In all other macro calls, numeric or address values for $S form macros have the form:

   ```
   #n
   ```

   For WTLO$S this form of the grp argument would be:

   ```
   n
   ```

4. The argument list specified in the FORTRAN call must contain only event flag numbers that lie within one event flag group. If event flag numbers are specified that lie in more than one group, or if an invalid event flag number is specified, a fatal FORTRAN error is generated.

5. If the issuing task has outstanding buffered I/O when it enters the Wait For state, it will be stopped. When the task is in a stopped state, it can be checkpointed by any other task regardless of priority. The task is unstopped when:

   - The outstanding buffered I/O completes.
   - The Wait For condition is satisfied.

6. If the specified group of event flags is group global, the group's use count is incremented to prevent premature elimination of the event flags. The use count is run down when:

   - The Wait For condition is satisfied.
   - The issuing task exits before the Wait For condition is satisfied.

5.3.93 Wait For Single Event Flag

The Wait For Single Event Flag directive instructs the system to block the execution of the issuing task until the indicated event flag is set. If the flag is set at issuance, task execution is not blocked.

FORTRAN Call:

```
     CALL WAITFR (efn[,ids])

       efn  =  Event flag number

       ids  =  Directive status
```

Macro Call:

```
     WTSE$   efn

       efn  =  Event flag number
```

Macro Expansion:

```
     WTSE$   52.
     .BYTE   41.,2            ;WTSE$ MACRO DIC, DPB SIZE=2 WORDS
     .WORD   52.              ;EVENT FLAG NUMBER 52.
```

Local Symbol Definitions:

W.TSEF -- Event flag number (2)

DSW Return Codes:

IS.SUC	--	Successful completion.
IE.IEF	--	Invalid event flag number (EFN<1, or EFN>96 if group global event flags exist for the task's group; or EFN>64 if not).
IE.ADP	--	Part of the DPB is out of the issuing task's address space.
IE.SDP	--	DIC or DPB size is invalid.

Notes:

1. If the issuing task has outstanding buffered I/O when it enters the Wait For state, it will be stopped. When the task is in a stopped state, it can be checkpointed by any other task regardless of priority. The task is unstopped when:
   - The outstanding buffered I/O completes.
   - The Wait For condition is satisfied.
2. If the specified event flag is group global, the group's use count is incremented to prevent premature elimination of event flags. The use count is run down when:
   - The Wait For condition is satisfied.
   - The issuing task exits before the Wait For condition is satisfied.

APPENDIX A

DIRECTIVE SUMMARY - ALPHABETICAL ORDER BY MACRO CALL

Abort Task **ABRT$**

FORTRAN Call:

```
     CALL ABORT (tsk[,ids])

       tsk  =  Task name to be aborted (RAD50)

       ids  =  Directive status
```

Macro Call:

```
     ABRT$   tsk

       tsk  =  Task name to be aborted (RAD50)
```

Alter Priority **ALTP$**

FORTRAN Call:

```
     CALL ALTPRI ([tsk],[ipri][,ids])

       tsk  =  Active task name

       ipri =  A 1-word integer value equal to the new priority,  from
               1 to 250 (decimal)

       ids  =  Directive status
```

Macro Call:

```
     ALTP$   [tsk][,pri]

       tsk  =  Active task name

       pri  =  New priority, from 1 to 250 (decimal)
```

Assign LUN **ALUN$**

FORTRAN Call:

```
     CALL ASNLUN (lun,dev,unt[,ids])

       lun  =  Logical unit number

       dev  =  Device name (format:  1A2)

       unt  =  Device unit number

       ids  =  Directive status
```

Macro Call:

```
     ALUN$   lun,dev,unt

       lun  =  Logical unit number

       dev  =  Device name (two characters)

       unt  =  Device unit number
```

AST Service Exit ($S form recommended) **ASTX$S**

FORTRAN Call:

Neither the FORTRAN language nor the ISA standard permits direct linking to system-trapping mechanisms; therefore, this directive is not available to FORTRAN tasks.

Macro Call:

```
     ASTX$S  [err]

       err  =  Error routine address
```

Attach Region **ATRG$**

FORTRAN Call:

```
     CALL ATRG (irdb[,ids])

       irdb =  An 8-word integer array containing a Region  Definition
               Block (see Section 3.5.1.2)

       ids  =  Directive status
```

Macro Call:

```
     ATRG$   rdb

       rdb  =  Region Definition Block address
```

Connect To Interrupt Vector **CINT$**

FORTRAN Call:

Not supported

Macro Call:

```
     CINT$   vec,base,isr,edir,pri,ast

       vec  =  Interrupt vector address -- Must be in the range  60(8)
               to  highest  vector specified during SYSGEN, inclusive,
               and must be a multiple of 4

       base =  Virtual base address for kernel APR 5  mapping  of  the
               ISR, and enable/disable interrupt routines

       isr  =  Virtual address of the ISR, or 0 to disconnect from the
               interrupt vector

       edir =  Virtual address of the enable/disable interrupt routine

       pri  =  Initial priority at which the ISR is to execute

       ast  =  Virtual address of an AST routine to be  entered  after
               the fork-level routine queues an AST
```

Clear Event Flag **CLEF$**

FORTRAN Call:

```
     CALL CLREF (efn[,ids])

       efn  =  Event flag number

       ids  =  Directive status
```

Macro Call:

```
     CLEF$  efn

       efn  =  Event flag number
```

Cancel Mark Time Requests **CMKT$**

FORTRAN Call:

```
     CALL CANMT ([efn][,ids])

       efn  =  Event flag number

       ids  =  Directive status
```

Macro Call:

```
     CMKT$  [efn,ast,err]

       efn  =  Event flag number

       ast  =  Mark time AST address

       err  =  Error routine address
```

Connect **CNCT$**

FORTRAN Call:

```
     CALL CNCT (rtname,[iefn],[iast],[iesb],[iparm][,ids])

       rtname  =  Name (RAD50) of the offspring task to be connected

       iefn    =  Event flag to be set when the offspring task exits
                  or emits status

       iast    =  Name of an AST routine to be called when the
                  offspring task exits or emits status

       iesb    =  Name of an 8-word status block to be written when
                  the offspring task exits or emits status

                     Word   0 -- Offspring task exit status

                     Word 1-7 -- Reserved

       iparm   =  Name of a word to receive the status block address
                  when an AST occurs

       ids     =  Integer to receive the Directive Status Word
```

Macro Call:

```
     CNCT$   tname, [efn],[east],[esb]

       tname   =  Name (RAD50) of the offspring task to be connected

       efn     =  The event flag to be cleared on issuance and set
                  when the offspring task exits or emits status

       east    =  Address of an AST routine to be called when the
                  offspring task exits or emits status

       esb     =  Address of an 8-word status block to be written when
                  the offspring task exits or emits status

                     Word   0 -- Offspring task exit status

                     Word 1-7 -- Reserved
```

Checkpoint Common Region **CPCR$**

FORTRAN Call:

```
     CALL CPCR   (name[,ids])

       name  =  Name (in RAD50) of the common region to be
                checkpointed

       ids   =  Directive status
```

Macro Call:

```
     CPCR$ name

       name  =  Name of the common region to be checkpointed
```

Create Address Window **CRAW$**

FORTRAN Call:

```
     CALL CRAW (iwdb[,ids])

       iwdb =  An 8-word integer array containing a Window  Definition
               Block (see Section 3.5.2.2)

       ids  =  Directive status
```

Macro Call:

```
     CRAW$   wdb

       wdb  =  Window Definition Block address
```

Create Group Global Event Flags **CRGF$**

FORTRAN Call:

```
     CALL CRGF ([group][,ids])

       group  =  Group number for the flags  to  be  created - If  not
                 specified,  the  task's  protection UIC (H.CUIC+1) in
                 the task's header is used

       ids    =  Integer to receive the Directive Status Word
```

Macro Call:

```
     CRGF$   [group]

       group  =  Group number for the flags  to  be  created - If  not
                 specified,  the  task's  protection UIC (H.CUIC+1) in
                 the task's header is used
```

Create Region **CRRG$**

FORTRAN Call:

```
     CALL CRRG (irdb[,ids])

       irdb =  An 8-word integer array containing a Region  Definition
               Block (see Section 3.5.1.2)

       ids  =  Directive status
```

Macro Call:

```
     CRRG$   rdb

       rdb  =  Region Definition Block address
```

Create Virtual Terminal **CRVT$**

FORTRAN Call:

```
    CALL CRVT ([iiast],[ioast],[iaast],[imlen],iparm[,ids])

      iiast   =  AST address at which input requests from offspring
                 tasks are serviced

      ioast   =  AST address at which output requests from offspring
                 tasks are serviced

      iaast   =  AST address at which the parent task may be notified
                 of the completion of successful offspring attach and
                 detach requests to the virtual terminal unit

      imlen   =  Maximum buffer length allowed for offspring I/O
                 requests

      iparm   =  Address of 3-word buffer to receive information from
                 the stack when an AST occurs

      ids     =  Integer to receive the Directive Status Word
```

Macro Call:

```
    CRVT$   [iast],[oast],[aast],[mlen]

      iast    =  AST address at which input requests from offspring
                 tasks are serviced

      oast    =  AST address at which output requests from offspring
                 tasks are serviced

      aast    =  AST address at which the parent task may be notified
                 of the completion of successful offspring attach and
                 detach requests to the virtual terminal unit (If
                 this parameter is not specified, no notification of
                 attaches and detaches are returned to the parent
                 task.)

      mlen    =  Maximum buffer length allowed for offspring I/O
                 requests
```

Cancel Time Based Initiation Requests **CSRQ$**

FORTRAN Call:

```
    CALL CANALL (tsk[,ids])

      tsk  =  Task name

      ids  =  Directive status
```

Macro Call:

```
    CSRQ$  tsk

      tsk  =  Task name
```

Declare Significant Event ($S form recommended) DECL$S

FORTRAN Call:

```
     CALL DECLAR ([,ids])

       ids  =  Directive status
```

Macro Call:

```
     DECL$S  [,err]

       err  =  Error routine address
```

Disable AST Recognition ($S form recommended) DSAR$S

FORTRAN Call:

```
     CALL DSASTR [(ids)]

       ids  =  Directive status
```

Macro Call:

```
     DSAR$S  [err]

       err  =  Error routine address
```

Disable Checkpointing ($S form recommended) DSCP$S

FORTRAN Call:

```
     CALL DISCKP [(ids)]

  ids  =  Directive status
```

Macro Call:

```
     DSCP$S  [err]

       err  =  Error routine address
```

Detach Region DTRG$

FORTRAN Call:

```
     CALL DTRG (irdb[,ids])

       irdb =  An 8-word integer array containing a Region  Definition
               Block (see Section 3.5.1.2)

       ids  =  Directive status
```

Macro Call:

```
     DTRG$   rdb

       rdb  =  Region Definition Block address
```

Eliminate Address Window **ELAW$**

FORTRAN Call:

```
CALL ELAW (iwdb[,ids])

  iwdbi    =  An 8-word integer array containing a Window
              Definition Block (see Section 3.5.2.2)

  ids      =  Directive status
```

Macro Call:

```
ELAW$   wdb

  wdb      =  Window Definition Block address
```

Eliminate Group Global Event Flags **ELGF$**

FORTRAN Call:

```
CALL ELGF ([group][,ids])

  group  =  Group number of flags to be eliminated

  ids    =  Integer to receive the Directive Status Word
```

Macro Call:

```
ELGF$    [group]

  group  =  Group number of flags to be eliminated
```

Eliminate Virtual Terminal **ELVT$**

FORTRAN Call:

```
CALL ELVT (iunum[,ids])

  iunum   =  Virtual terminal unit number

  ids     =  Integer to receive the Directive Status Word
```

Macro Call:

```
ELVT$   unum

  unum    =  Unit number of the virtual terminal to be eliminated
```

Emit Status **EMST$**

FORTRAN Call:

```
CALL EMST ([rtname],istat[,ids])

  rtname  =  Name of task connected to issuing task to which the
             status is to be emitted

  istat   =  A 16-bit quantity to be returned to the connected
             task

  ids     =  Integer to receive the Directive Status Word
```

Macro Call:

```
     EMST$    [tname],status

       tname    =  Name of a task connected to the issuing task to
                   which the status is to be emitted

       status   =  A 16-bit quantity to be returned to the connected
                   task
```

Enable AST Recognition ($S form recommended) **ENAR$S**

FORTRAN Call:

```
     CALL ENASTR [(ids)]

       ids  =  Directive status
```

Macro Call:

```
     ENAR$S  [err]

       err  =  Error routine address
```

Enable Checkpointing ($S form recommended) **ENCP$S**

FORTRAN Call:

```
     CALL ENACKP [(ids)]

        ids  =  Directive status
```

Macro Call:

```
     ENCP$S  [err]

       err  =  Error routine address
```

Exit If **EXIF$**

FORTRAN Call:

```
     CALL EXITIF (efn[,ids])

       efn  =  Event flag number

       ids  =  Directive status
```

Macro Call:

```
     EXIF$  efn

       efn  =  Event flag number
```

Task Exit ($S form recommended) EXIT$S

FORTRAN Call:

Fortran tasks that terminate with the STOP statement result in a message that includes task name, a statement causing the task to stop, and an optional character string specified in the STOP statement. CALL EXIT terminates with the message STOP THIS FORTRAN TASK.

Macro Call:

```
EXIT$S  [err]

  err  =  Error routine address
```

Exit With Status EXST$

FORTRAN Call:

```
CALL EXST (istat)

  istat   =  A 16-bit quantity to be returned to parent task
```

Macro Call:

```
EXST$   status

  status  =  A 16-bit quantity to be returned to parent task
```

Extend Task EXTK$

FORTRAN Call:

```
CALL EXTTSK ([inc][,ids])

  inc  =  A positive or negative number equal to the number of
          32-word blocks by which the task size is to be extended
          or reduced (If omitted, task size defaults to installed
          task size.)

  ids  =  Directive status
```

Macro Call:

```
EXTK$   [inc]

  inc  =  A positive or negative number equal to the number of
          32-word blocks by which the task is to be extended or
          reduced (If omitted, task size defaults to installed
          task size.)
```

Get Command for Command Interpreter GCCI$

Fortran Call:

```
CALL GTCMCI (icbf,icbfl,[iibuf],[iibfl],[iaddr],[incp][,ids])

  icbf  = Name of a byte to receive the command
```

icbfl = Integer containing the size of the icbf array in bytes

iibuf = Name of an integer containing the length of the optional information buffer

iaddr = Name of an integer that contains the address in pool of the command desired (This address was obtained by a previous call to GTCMCI with GC.CND specified.)

incp = Name of an integer containing a value indicating the action to take if there is no command queued

ids = Integer to receive the directive status word

Macro Call:

```
GCCI$ cbuf,cbfl,[ibuf],[ibfl],[addr],[ncp]
```

cbuf = Address of buffer to receive command string

cbfl = Length of buffer. Maximum buffer size is 84. for RSX-11M and 259. for RSX-11M-PLUS.

ibuf = Address of buffer to receive information on the issuing terminal

ibfl = Length of buffer to receive information

addr = Address of command

ncp = Action to take if no command buffer is present

GC.CCS (000) -- Return with carry set (default)

GC.CEX (001) -- Force CLI to exit instead of returning

GC.CST (002) -- Force CLI to stop instead of returning

GC.CND (200) -- Copy command into buffer but do not dequeue it from the list

Get Command Interpreter Information GCII$

FORTRAN Call:

```
CALL GETCII (ibuf,ibfl,[icli],[idev],[iunit][,ids]
```

ibuf = Name of an integer array to receive the CLI information

ibfl = Length in bytes of the integer array to receive the CLI information

icli = Name of a 2-word array element containing the RAD50 name of the CLI

idev = Name of an integer containing the ASCII name of terminal (default = TI:)

iunit = Name of an integer containing the octal unit number of terminal

ids = Directive status

Macro Call:

GCII$ buf,bufl,cli,[dev],[unit]

buf	=	Address of buffer to receive information
bufl	=	Length of information buffer
cli	=	Name in RAD50 of the CLI that information is requested on
dev	=	ASCII name of terminal whose CLI should be used
unit	=	Octal unit number of terminal

Get LUN Information **GLUN$**

FORTRAN Call:

CALL GETLUN (lun,dat[,ids])

lun	=	Logical unit number
dat	=	A 6-word integer array to receive LUN information
ids	=	Directive status

Macro Call:

GLUN$ lun,buf

lun	=	Logical unit number
buf	=	Address of 6-word buffer that will receive the LUN information

Get MCR Command Line **GMCR$**

FORTRAN Call:

CALL GETMCR (buf[,ids])

buf	=	An 80-byte array to receive command line
ids	=	Directive status

Macro Call:

GMCR$

Get Mapping Context **GMCX$**

FORTRAN Call:

```
     CALL GMCX (imcx[,ids])

       imcx =  An integer array to receive the mapping  context.   The
               size  of  the  array is 8*n+1, where n is the number of
               window blocks in the task's header (The maximum size is
               8*8+1=65  on  RSX-11M  systems.   The  maximum  size is
               8*24+1=193 on RSX-11M-PLUS systems.)

       ids  =  Directive status
```

Macro Call:

```
     GMCX$   wvec

       wvec =  The address of a vector of n Window Definition  Blocks;
               n is the number of window blocks in the task's header.
```

Get Partition Parameters **GPRT$**

FORTRAN Call:

```
     CALL GETPAR ([prt],buf[,ids])

       prt  =  Partition name

       buf  =  A 3-word integer array to receive partition parameters

       ids  =  Directive status
```

Macro Call:

```
     GPRT$  [prt],buf

       prt  =  Partition name

       buf  =  Address of 3-word buffer
```

Get Region Parameters **GREG$**

FORTRAN Call:

```
     CALL GETREG ([rid],buf[,ids])

       rid  =  Region id

       buf  =  A 3-word integer array to receive region parameters

       ids  =  Directive status
```

Macro Call:

```
     GREG$   [rid],buf

       rid  =  Region ID

       buf  =  Address of 3-word buffer
```

Get Sense Switches ($S form recommended) **GSSW$S**

FORTRAN Call:

```
     CALL READSW (isw)

       isw  =  Integer to receive the console switch settings
```

The following FORTRAN call allows a program to read the state of a single switch:

```
     CALL SWITCH (ibt,ist)

       ibt  =  The switch to be tested (0 to 15)

       ist  =  Test results where:

                    1  =  switch on

                    2  =  switch off
```

Macro Call:

```
     GSSW$S  [err]

       err  =  Error routine address
```

Get Time Parameters **GTIM$**

FORTRAN Call:

```
     CALL GETTIM (ibfl[,ids])

       ibfl  = An 8-word integer array

       ids   = Directive status
```

Macro Call:

```
     GTIM$   buf

       buf  =  Address of 8-word buffer
```

Get Task Parameters **GTSK$**

FORTRAN Call:

```
     CALL GETTSK (buf[,ids])

       buf  =  A 16-word integer array to receive the task parameters

       ids  =  Directive status
```

Macro Call:

```
     GTSK$   buf

       buf  =  Address of 16-word buffer
```

Inhibit AST Recognition ($S form recommended) **IHAR$S**

FORTRAN Call:

```
     CALL INASTR  [(ids)]

       ids  =  Directive status
```

Macro Call:

```
     IHAR$S  [err]

       err  =  Error routine address
```

Map Address Window **MAP$**

FORTRAN Call:

```
     CALL MAP (iwdb[,ids])

       iwdb =  An 8-word integer array containing a Window  Definition
               Block (see Section 3.5.2.2)

       ids  =  Directive status
```

Macro Call:

```
     MAP$   wdb

       wdb  =  Window Definition Block address
```

Mark Time **MRKT$**

FORTRAN Call:

```
     CALL MARK (efn,tmg,tnt[,ids])

       efn  =  Event flag number

       tmg  =  Time interval magnitude

       tnt  =  Time interval unit

       ids  =  Directive status
```

The ISA standard call for delaying a task for a specified time interval is also included:

```
     CALL WAIT (tmg,tnt,ids)

       tmg  =  Time interval magnitude

       tnt  =  Time interval unit

       ids  =  Directive status
```

Macro Call:

```
     MRKT$    [efn],tmg,tnt[,ast]

       efn  =  Event flag number

       tmg  =  Time interval magnitude

       tnt  =  Time interval unit

       ast  =  AST entry point address
```

Map Supervisor D-Space to Supervisor I-Space **MSDS$**

FORTRAN Call:

Not supported.

Macro Call:

```
     MSDS$    mask

       mask  = A 7-bit mask with one bit corresponding to each APR.
               If the bit is set, the APR is mapped to supervisor-mode
               I-space. If the bit is clear the APR is mapped to
               user-mode D-space. The 7 bits are specified in bits 8
               through 14 of the mask word.
```

Move to/from User/Supervisor I/D-Space **MVTS$**

FORTRAN Call:

Not supported.

Macro Call:

```
     MVTS$    action,addr,val
                          buff

       action  =  One of the following:

                    MV.TUI -- Move to user I-space
                    MV.TUD -- Move to user D-space
                    MV.TSI -- Move to supervisor I-space
                    MV.TSD -- Move to supervisor D-space
                    MV.FUI -- Move from user I-space
                    MV.FUD -- Move from user D-space
                    MV.FSI -- Move from supervisor I-space
                    MV.FSD -- Move from supervisor D-space

       addr    =  Address of the location in the task

       buf     =  Buffer to receive the value fetched, for the move
                  from operations

       val     =  Value to be stored in the location, for the move to
                  operations
```

Queue I/O Request **QIO$**

FORTRAN Call:

```
CALL QIO (fnc,lun,[efn],[pri],[isb],[prl][,ids])
```

fuc	=	I/O function code
lun	=	Logical unit number
efn	=	Event flag number
pri	=	Priority; ignored, but must be present
isb	=	A 2-word integer array to receive final I/O status
prl	=	A 6-word integer array containing device-dependent parameters to be placed in parameter words 1 through 6 of the Directive Parameter Block (DPB). Fill in this array by using the GETADR routine (see Section 1.5.1.4).
ids	=	Directive status

Macro Call:

```
QIO$    fnc,lun,[efn],[pri],[isb],[ast],[prl]
```

fnc	=	I/O function code
lun	=	Logical unit number
efn	=	Event flag number
pri	=	Priority; ignored, but must be present
isb	=	Address of I/O status block
ast	=	Address of AST service routine entry point
prl	=	Parameter list of the form <p1,...p6>

Queue I/O Request And Wait **QIOW$**

FORTRAN Call:

```
CALL WTQIO (fnc,lun,[efn],[pri],[isb],[prl][,ids])
```

fnc	=	I/O function code
lun	=	Logical unit number
efn	=	Event flag number
pri	=	Priority; ignored, but must be present
isb	=	A 2-word integer array to receive final I/O status
prl	=	A 6-word integer array containing device dependent parameters to be placed in parameter words 1 through 6 of the DPB
ids	=	Directive status

Macro Call:

```
QIOW$   fnc,lun,[efn],[pri],[isb],[ast][,prl]
```

fnc	=	I/O function code
lun	=	Logical unit number
efn	=	Event flag number
pri	=	Priority; ignored, but must be present
isb	=	Address of I/O status block
ast	=	Address of AST service routine entry point
prl	=	Parameter list of the form <pl,...p6>

Receive Data Or Stop **RCST$**

FORTRAN Call:

```
CALL RCST ([rtname],ibuf[,ids])
```

rtname	=	Sender task name (If not specified, data may be received from any task.)
ibuf	=	Address of 15-word buffer to receive the sender task name and data
ids	=	Integer to receive the Directive Status Word

Macro Call:

```
RCST$   [tname],buf
```

tname	=	Sender Task name (If not specified, data may be received from any task.)
buf	=	Address of a 15-word buffer to receive the sender task name and data

Receive Data **RCVD$**

FORTRAN Call:

```
CALL RECEIV ([tsk],buf[,,ids])
```

tsk	=	Sender task name (If not specified, data may be received from any task.)
buf	=	A 15-word integer array for received data
ids	=	Directive status

Macro Call:

```
RCVD$  [tsk],buf
```

tsk	=	Sender task name (If not specified, data may be received from any task.)
buf	=	Address of 15-word buffer

Receive Data Or Exit **RCVX$**

FORTRAN Call:

CALL RECOEX ([tsk],buf[,,ids])

tsk	=	Sender task name (If not specified, data may be received from any task.)
buf	=	A 15-word integer array for received data
ids	=	Directive status

Macro Call:

RCVX$ [tsk],buf

tsk	=	Sender task name (If not specified, data may be received from any task.)
buf	=	Address of 15-word buffer

Read All Event Flags **RDAF$**

FORTRAN Call:

A FORTRAN task can only read a single event flag. The call is:

CALL READEF (efn[,ids])

efn	=	Event flag number (1-64.)
ids	=	Directive status

Macro Call:

RDAF$ buf

buf	=	Address of 4-word buffer

Read Event Flag **RDEF$**

FORTRAN Call:

CALL READEF (iefn[,ids])

iefn	=	Integer containing an event flag number
ids	=	Integer variable to receive the Directive Status Word

Macro Call:

RDEF$ efn

efn	=	Event flag number

Read Extended Event Flags **RDXF$**

FORTRAN Call:

A FORTRAN task can read only a single event flag. The call is:

```
CALL READEF (efn[,ids])

  efn  =  Event flag number (1-96.)

  ids  =  Directive status
```

Macro Call:

```
RDXF$  buf

  buf  =  Address of 6-word buffer
```

Remove Affinity ($S form recommended) **RMAF$S**

FORTRAN Call:

```
CALL RMAF  [(ids)]

  ids =  Integer to receive the Directive Status Word
```

Macro Call:

```
RMAF$S
```

Request and Pass Offspring Information **RPOI$**

FORTRAN Call:

```
CALL RPOI (tname,[iugc],[iumc],[iparen],[ibuf],[ibfl],[isc],
           [itask],[ocbad][,ids])
```

tname	= An array containing the actual name of the task to be requested and optionally chained to
iugc	= Integer containing the group code number for the UIC of the requested target chain task
iumc	= Integer containing the member code number for the UIC of the requested target chain task
iparen	= Array (or I*4 integer) containing the RAD50 name of the parent task (This is returned in the information buffer of the GTCMCI subroutine.)
ibuf	= Array that contains the command line text for the chained task.

isc	= Flag byte controlling the actions of this directive request when executed. The bit definitions of this byte are as follows:

RP.OEX = 128.	Force this task to exit on successful execution of the RPOI directive.
RP.OAL = 1	Pass all of this task's OCBs to the requested task. (Default is none.)

idnam	= Integer containing the ASCII device name of the requested tasks TI:
iunit	= Integer containing the unit number of the requested tasks TI: device
itask	= Array which contains the RAD50 name the requested task is to run under. (Valid only for CLIs.)
ocbad	= Integer containing the internal pool address of the parent OCB (Only a CLI can specify this argument because the value can only be obtained in the information buffer of the GTCMCI subroutine.)
ids	= Integer to receive the directive status word

Macro Call:

```
RPOI$
tname,,,,[ugc],[umc],[parent],[bufadr],[buflen],[sc],[dnam],
      [unit],[tas]k,[ocba]d
```

tname	= Name of task to be chained to
ugc	= Group code for UIC of the requested task
umc	= Member code for UIC of the requested task
parent	= Name of issuing task's parent task whose OCB is to be passed. If not specified, all OCB's are passed.
bufadr	= Address of buffer to be given to the requested task
buflen	= Length of buffer to be given to requested task
sc	= Flags byte:

```
RP.OEX -- (200) Force issuing task to exit
RP.OAL -- (1)   Pass all OCBs
```

dnam	= ASCII device name for TI:
unit	= Unit number of task TI:
task	= RAD50 name of task to be started
ocbad	= Address of OCB to pass (CLIs only)

Request Task **RQST$**

FORTRAN Call:

```
     CALL REQUES (tsk,[opt][,ids])

       tsk  =  Task name

       opt  =  A 4-word integer array:

                  opt(1)  =  Partition name first half;  ignored,  but
                             must be present

                  opt(2)  =  Partition name second half;  ignored, but
                             must be present

                  opt(3)  =  Priority;  ignored, but must be present

                  opt(4)  =  User Identification Code

       ids  =  Directive status
```

Macro Call:

```
     RQST$  tsk,[prt],[pri][,ugc,umc]

       tsk  =  Task name

       prt  =  Partition name;  ignored, but must be present

       pri  =  Priority;  ignored, but must be present

       ugc  =  UIC group code

       umc  =  UIC member code
```

Receive By Reference **RREF$**

FORTRAN Call:

```
     CALL RREF (iwdb,[isrb][,ids])

       iwdb =  An 8-word integer array containing a Window  Definition
               Block (see Section 3.5.2.2)

       isrb =  A 10-word integer array  to  be  used  as  the  receive
               buffer

       ids  =  Directive status
```

Macro Call:

```
     RREF$   wdb

       wdb  =  Window Definition Block
```

Resume Task **RSUM$**

FORTRAN Call:

```
     CALL RESUME (tsk[,ids])

       tsk  =  Task name

       ids  =  Directive status
```

Macro Call:

```
     RSUM$  tsk

       tsk  =  Task name
```

Run Task **RUN$**

FORTRAN Call:

```
     CALL RUN (tsk,[opt],[smg],snt,[rmg],[rnt][,ids])

       tsk  =  Task name

       opt  =  A 4-word integer array:

                  opt(1)  =  Partition name first half;  ignored,  but
                             must be present

                  opt(2)  =  Partition name second half;  ignored, but
                             must be present

                  opt(3)  =  Priority;  ignored, but must be present

                  opt(4)  =  User Identification Code

       smg  =  Schedule delta magnitude

       snt  =  Schedule delta unit

       rmg  =  Reschedule interval magnitude

       rnt  =  Reschedule interval unit

       ids  =  Directive status
```

The ISA standard call for initiating a task is also included:

```
     CALL START (tsk,smg,snt[,ids])

       tsk  =  Task name

       smg  =  Schedule delta magnitude

       snt  =  Schedule delta unit

       ids  =  Directive status
```

Macro Call:

```
     RUN$   tsk,[prt],[pri],[ugc],[umc],[smg],snt[,rmg,rnt]

       tsk  =  Task name

       prt  =  Partition name;  ignored, but must be present

       pri  =  Priority;  ignored, but must be present

       ugc  =  UIC group code

       umc  =  UIC member code

       smg  =  Schedule delta magnitude

       snt  =  Schedule delta unit

       rmg  =  Reschedule interval magnitude

       rnt  =  Reschedule interval unit
```

Supervisor Call ($$ form recommended) **SCAL$S**

FORTRAN Call:

Not supported

Macro Call:

```
     SCAL$S   saddr,caddr

       saddr   =  Address of the called supervisor-mode routine

       caddr   =  Address of the completion routine for return to  the
                  caller
```

Set Command Line Interpreter **SCLI$**

FORTRAN Call:

```
     CALL SETCLI (icli,idev,iunit[,ids])

       icli  = A two word array element containing the name of the CLI
               to which the terminal is to be set

       idev  = Integer containing the ASCII name of the terminal to be
               set (default = TI:)

       iunit = Integer containing the unit number of terminal

       ids   = Directive status
```

Macro Call:

```
     SCLI$ cli,[dev],[unit]

       cli    =  Name of the CLI to which the terminal is to be set

       dev    =  ASCII name of the terminal to be set (default = TI:)

       unit   =  Unit number of terminal
```

Send Data **SDAT$**

FORTRAN Call:

```
     CALL SEND (tsk,buf,[efn][,ids])

       tsk  =  Task name

       buf  =  A 13-word integer array of data to be sent

       efn  =  Event flag number

       ids  =  Directive status
```

Macro Call:

```
     SDAT$  tsk,buf[,efn]

       tsk  =  Task name

       buf  =  Address of 13-word data buffer

       efn  =  Event flag number
```

Send, Request And Connect **SDRC$**

FORTRAN Call:

```
     CALL SDRC (rtname,ibuf,[iefn],[iast],[iesb],[iparm][,ids])

       rtname  =  Target task name of the offspring task to be
                  connected

       ibuf    =  Name of 13-word send buffer

       iefn    =  Event flag to be set when the offspring task exits
                  or emits status

       iast    =  Name of an AST routine to be called when the
                  offspring task exits or emits status

       iesb    =  Name of an 8-word status block to be written when
                  the offspring task exits or emits status

                      Word   0 -- Offspring task exit status

                      Word 1-7 -- Reserved

       iparm   =  Name of a word to receive the status block address
                  when an AST occurs

       ids     =  Integer to receive the Directive Status Word
```

Macro Call:

```
     SDRC$   tname,buf,[efn],[east],[esb]

       tname  =  Target task name of the offspring task to be
                 connected

       buf    =  Address of a 13-word send buffer
```

efn = The event flag to be cleared on issuance and when the offspring task exits or emit status

east = Address of an AST routine to be called when the offspring task exits or emits status

esb = Address of a 8-word status block to be written when the offspring task exits or emits status

Word 0 -- Offspring task exit status

Word 1-7 -- Reserved

Send Data Request and Pass Offspring Control Block **SDRP$**

FORTRAN Call:

```
CALL SDRP (task,ibuf,[ibfl],[iefn],[iflag],[iparen],
           [iocbad][,ids])
```

task = Name of an array (REAL, INTEGER, I*4) that contains the RAD50 name of target task

ibuf = Integer array containing data to be sent

ibfl = Integer containing number of words (integers) in the array to be sent (On RSX-11M systems, this argument must be 13., and on RSX-11M-PLUS systems, this argument may be in the range of 1 to 255.) (Default = 13.)

iefn = Integer containing the number of the event flag to be set when this directive is executed successfully

iflag = Integer containing flags bits controlling the execution. They are defined as follows:

SD.REX = 128.	Force this task to exit upon successful execution
SD.RAL = 1	Pass all OCBs

iparen = Name of array containing the RAD50 name of the parent task whose OCB should be passed to the target task

iocbad = Name of an integer containing internal pool address of the OCB to pass

ids = Integer to receive the contents of the Directive Status Word

Macro Call:

```
SDRP$ task,bufadr,[buflen],[efn],[flag],[parent],[ocbad]
```

task = Name of task to be chained to

bufadr = Address of buffer to be given to the requested task

buflen = Length of buffer to be given to requested task

efn	=	Event flag
flag	=	Flags byte (Force exit, pass all OCB's)
parent	=	Name of issuing task's parent task whose OCB is to be passed
ocbad	=	Address of OCB to pass (CLI's only)

Set Event Flag **SETF$**

FORTRAN Call:

```
CALL SETEF (efn[,ids])
```

efn	=	Event flag number
ids	=	Directive status

Macro Call:

```
SETF$   efn
```

efn	=	Event flag number

Specify Floating Point Exception AST **SEPA$**

FORTRAN Call:

Not supported

Macro Call:

```
SFPA$   [ast]
```

ast	=	AST service routine entry point address

Send Message **SMSG$**

FORTRAN Call:

```
CALL SMSG (itgt,ibuf,ibufl,iprm,iprml,ids)
```

itgt	=	Integer containing the target object
ibuf	=	Integer array containing the data to be inserted into the formatted data packet
ibufl	=	Integer containing length of the ibuf array
iprm	=	Integer array containing any additional parameters
iprml	=	Integer containing the number of parameters in the iprm array
ids	=	Optional integer to receive the directive status

Macro Call:

```
     SMSG$ tgt,buf,len,<pri,...,prn>

       tgt           =  Target identifier

       buf           =  Address of optional data buffer

       len           =  Length in bytes of optional data buffer

       pri,...,prn   =  Target-specific parameter list:

     Parameter list for Error Logging

        SMSG$ SM.SER,buf,len,typ,sub,lun,mask>

          typ  = Error Log packet code

          sub  = Error Log packet subtype code

          lun  = Logical unit number of device

          msk  = Control mask word
```

Send Next Command **SNXC$**

FORTRAN Call:

```
     CALL SNXC ([dnam][,iunit][,ids])

       dnam  = Device name (ASCII).  If not specified, TI:  is used.

       iunit = Unit number of the terminal from which the  command  is
               to be sent.

       ids   = Integer to receive the directive status word.
```

Macro Call:

```
     SNXC$ [dnam][,unum]

       dnam  = Device name (ASCII).  If not specified, TI:  is used.

       unum  = Unit number of the terminal from which the  command  is
               to be sent.
```

Specify Parity Error AST **SPEA$**

FORTRAN Call:

     Not supported

Macro Call:

```
     SPEA$ [ast]

       ast  =  AST services routine entry point address
```

Suspend ($S form recommended) **SPND$S**

FORTRAN Call:

```
     CALL SUSPND   [(ids)]

       ids  = Directive status
```

Macro Call:

```
     SPND$S  [err]

       err  =  Error routine address
```

Specify Power Recovery AST **SPRA$**

FORTRAN Call:

```
     EXTERNAL sub

     CALL PWRUP (sub)

       sub  =  Name of a subroutine to be executed upon power
               recovery.  The PWRUP subroutine will effect the
               following:

                  CALL sub (no arguments)

               The subroutine is called as a result of a power
               recovery AST, and therefore the subroutine can be
               controlled at critical points by using the DSASTR (or
               INASTR) and ENASTR subroutine calls.
```

To Remove an AST:

```
     CALL PWRUP
```

Macro Call:

```
     SPRA$  [ast]

       ast  =  AST service routine entry point address
```

Spawn **SPWN$**

FORTRAN Call:

```
     CALL SPAWN (rtname,[iugc],[iumc],[iefn],[iast],[iesb],[iparm],
                [icmlin],[icmlen],[iunit],[dnam][,ids])

       rtname  =  Name (RAD50) of the offspring task to be spawned

       iugc    =  Group code number for the UIC of the offspring task

       iumc    =  Member code number for the UIC of the offspring task

       iefn    =  Event flag to be set when the offspring task exits
                  or emits status

       iast    =  Name of an AST routine to be called when the
                  offspring task exits or emits status
```

iesb	=	Name of an 8-word status block to be written when the offspring task exits or emits status Word 0 -- Offspring task exit status Word 1-7 -- Reserved
iparm	=	Name of a word to receive the status block address when the AST occurs
icmlin	=	Name of a command line to be queued for the offspring task
icmlen	=	Length of the command line (79. characters maximum)
iunit	=	Unit number of terminal to be used as the TI: for the offspring task (If the optional dnam parameter is not specified, this parameter must be the unit number of a virtual terminal created by the issuing task; if a value of 0 is specified, the TI: of the issuing task is propagated.)
dnam	=	Device name mnemonic (If not specified, the virtual terminal is used as TI:.)
ids	=	Integer to receive the Directive Status Word

Macro Call:

```
SPWN$   tname,,,[ugc],[umc],[efn],[east],[esb],[cmdlin],[cmdlen]
             ,[unum],[dnam]
```

tname	=	Name (RAD50) of the offspring task to be spawned
ugc	=	Group code number for the UIC of the offspring task
umc	=	Member code number for the UIC of the offspring task
efn	=	The event flag to be cleared on issuance and set when the offspring task exits or emits status
east	=	Address of an AST routine to be called when the offspring task exits or emits status
esb	=	Address of an 8-word status block to be written when the offspring task exits or emits status Word 0 -- Offspring task exit status Word 1-7 -- Reserved
cmdlin	=	Address of a command line to be queued for the offspring task
cmdlen	=	Length of the command line (maximum length is 79.)
unum	=	Unit number of terminal to be used as the TI: for the offspring task (If the optional dnam parameter is not specified, this parameter must be the unit number of a virtual terminal created by the issuing task; if a value of 0 is specified, the TI: of the issuing task is propagated.)

dnam = Device name mnemonic (If not specified, the virtual terminal is used as TI:.)

NOTE

1. If neither unum nor dnam is specified, the TI: of the issuing task is propagated.

2. If only unum is specified, TI: is a virtual terminal.

Specify Receive Data AST — SRDA$

FORTRAN Call:

Not supported

Macro Call:

```
SRDA$   [ast]

  ast  =  AST service routine entry point address
```

Specify Requested Exit AST — SREA$ SREX$

FORTRAN Call:

```
CALL SREA (ast[,ids])

  ast = Name of the externally declared AST subroutine

  ids = Name of an optional  integer  to  receive  the  Directive
        Status Word

CALL SREX (ast,ipblk,ipblkl,[dummy][,ids])

  ast    = Name of the externally declared AST subroutine

  ipblk  = Name of an integer array to receive the  trap-dependent
           parameters

  ipblkl = Number of parameters to be  returned  into  the  ipblk
           array

  dummy  = Reserved for future use

  ids    = Name of an optional integer to  receive  the  Directive
           Status Word
```

Macro Call:

```
SREA$ [ast]

SREX$ [ast][,dummy]

  ast   = AST service routine entry point address

  dummy = Reserved for future expansion
```

Send By Reference **SREF$**

FORTRAN Call:

CALL SREF (tsk,[efn],iwdb,[isrb][,ids])

tsk	=	Receiver task name
efn	=	Event flag number
iwdb	=	An 8-word integer array containing a Window Definition Block (see Section 3.5.2.2)
isrb	=	An 8-word integer array containing additional information
ids	=	Directive status

Macro Call:

SREF$ task,wdb[,efn]

task	=	Receiver task name
wdb	=	Window Definition Block address
efn	=	Event flag number

Specify Receive-By-Reference AST **SRRA$**

FORTRAN Call:

Not supported

Macro Call:

SRRA$ [ast]

ast	=	AST service routine entry point address

Set Affinity **STAF$**

FORTRAN Call:

CALL STAF (iaff[,ids])

iaff	=	Affinity mask word
ids	=	Integer to receive Directive Status Word

Macro Call:

STAF$ [cp!ub!ub...]

cp	=	CPU selected (A through D)
ub	=	UNIBUS run(s) selected (E through T)

Set System Time Directive **STIM$**

FORTRAN Call:

```
     CALL SETTIM (ibufn[,ibufp][,ids])

       ibufn = An 8-word integer array, new time specification buffer

       ibufp = An 8-word integer array, previous time buffer

       ids   = Directive status
```

Macro Call:

```
     STIM$ bufn,[bufp]

       bufn  = Address of 8-word new time specification buffer

       bufp  = Address of 8-word buffer to receive the previous system
               time parameters
```

Stop For Logical OR Of Event Flags **STLO$**

FORTRAN Call:

```
     CALL STLOR (iefl,ief2,ief3, ...  ief(n))

       iefl ...  ief(n)  =  List of event flag numbers
```

Macro Call:

```
     STLO$   grp, msk

       grp  =  Desired group of event flags

       msk  =  A 16-bit mask word
```

Stop ($S form recommended) **STOP$S**

FORTRAN Call:

```
     CALL STOP ([ids])

       ids  =  Integer to receive the Directive Status Word
```

Macro Call:

```
     STOP$S
```

Stop For Single Event Flag **STSE$**

FORTRAN Call:

```
     CALL STOPFR (iefn[,ids])

       iefn =  Event flag number

       ids  =  Integer to receive Directive Status Word
```

Macro Call:

```
     STSE$   efn

       efn  =  Event flag number
```

Specify SST Vector Table For Debugging Aid **SVDB$**

FORTRAN Call:

Not supported

Macro Call:

```
     SVDB$  [adr][,len]

       adr  =  Address of SST vector table

       len  =  Length of (that is, number of entries in) table in
               words
```

Specify SST Vector Table For Task **SVTK$**

FORTRAN Call:

Not supported

Macro Call:

```
     SVTK$  [adr][,len]

       adr  =  Address of SST vector table

       len  =  Length of (that is, number of entries in) table in
               words
```

Unlock Group Global Event Flags ($S form recommended) **ULGF$S**

FORTRAN Call:

```
     CALL ULGF ([ids])

       ids = Directive status
```

Macro Call:

```
     ULGF$S [,err]

       err = Error routine address
```

Unmap Address Window **UNMAP$**

FORTRAN Call:

```
     CALL UNMAP (iwdb[,ids])

       iwdb =  An 8-word integer array containing a Window Definition
               Block (see Section 3.5.2.2)

       ids  =  Directive status
```

Macro Call:

```
     UMAP$   wdb

       wdb  =  Window Definition Block address
```

Unstop TASK **USTP$**

FORTRAN Call:

```
     CALL USTP (rtname[,ids])

       rtname  =  Name of task to be unstopped

       ids     =  Integer to receive directive status information
```

Macro Call:

```
     USTP$   tname

       tname   =  Name of task to be unstopped
```

Variable Receive Data **VRCD$**

FORTRAN Call:

```
     CALL VRCD ([task],bufadr,[buflen][,ids])

       task    =  Sender task name

       bufadr  =  Address of buffer to receive the  sender  task  name
                  and data

       buflen  =  Length of buffer

       ids     =  Integer to receive the Directive Status Word
```

Macro Call:

```
     VRCD$  [task],bufadr[,buflen]

       task    =  Sender task name

       bufadr  =  Buffer address

       buflen  =  Buffer size in words
```

Variable Receive Data Or Stop **VRCS$**

FORTRAN Call:

```
     CALL VRCS ([task],bufadr,[buflen][,ids])

       task    =  Sender task name

       buf     =  Address of buffer to receive the  sender  task  name
                  and data

       buflen  =  Length of buffer

       ids     =  Integer to receive the Directive Status Word
```

Macro Call:

```
     VRCS$       [task],bufadr[,buflen]

       task    =  Sender task name

       bufadr  =  Buffer address

       buflen  =  Buffer size in words
```

Variable Receive Data Or Exit **VRCX$**

FORTRAN Call:

```
     CALL VRCX ([task],bufadr,[buflen][,ids])

       task    =  Sender task name

       bufadr  =  Address of buffer to receive the sender task name
                  and data

       buflen  =  Length of buffer

       ids     =  Integer to receive the Directive Status Word
```

Macro Call:

```
     VRCX$       [task],bufadr[,buflen]

       task    =  Sender task name

       bufadr  =  Buffer address

       buflen  =  Buffer size in words
```

Variable Send Data **VSDA$**

FORTRAN Call:

```
     CALL VSDA ([task],bufadr,[buflen],[efn][,ids])

       task    =  Receiver task name

       bufadr  =  Address of buffer to receive the sender task name
                  and data

       buflen  =  Length of buffer

       efn     =  Event flag number

       ids     =  Integer to receive the Directive Status Word
```

Macro Call:

```
     VSDA$       [task],bufadr,[buflen][,efn]

       task    =  Receiver task name

       bufadr  =  Buffer address

       buflen  =  Buffer size in words

       efn     =  Event flag number
```

Variable Send, Request and Connect **VSRC$**

FORTRAN Call:

```
CALL VSRC (rtname,ibuf,[ibuflen],[iefn],[iast],[iesb],[iparm][,ids])
```

rtname =	Target task name of the offspring task to be connected
ibuf =	Name of 13-word send buffer
ibuflen =	Length of buffer
iefn =	Event flag to be set when the offspring task exits or emits status
iast =	Name of an AST routine to be called when the offspring task exits or emits status
iesb =	Name of an 8-word status block to be written when the offspring task exits or emits status Word 0 -- Offspring task exit status Word 1-7 -- Reserved
iparm =	Name of a word to receive the status block address when an AST occurs
ids =	Integer to receive the Directive Status Word

Macro Call:

```
VSRC$    tname,buf[,buflen],efn,east,esb
```

tname =	Target task name of the offspring task to be connected
buf =	Address of a 13-word send buffer
buflen =	Length of buffer
efn =	The event flag to be cleared on issuance and set when the offspring task exits or emits status
east =	Address of an AST routine to be called when the offspring task exits or emits status
esb =	Address of a 8-word status block to be written when the offspring task exits or emits status Word 0 -- Offspring task exit status Word 1-7 -- Reserved

Wait For Significant Event ($S form recommended) **WSIG$S**

FORTRAN Call:

```
     CALL WFSNE
```

Macro Call:

```
     WSIG$S  [err]

       err  =  Error routine address
```

Wait For Logical OR Of Event Flags **WTLO$**

FORTRAN Call:

```
     CALL WFLOR (efn1,efn2,...efnn)

       efn  =  List of event flag numbers taken as the set of flags to
               be specified in the directive
```

Macro Call:

```
     WTLO$   grp,msk

       grp  =  Desired group of event flags

       msk  =  A 16-bit octal mask word
```

Wait For Single Event Flag **WTSE$**

FORTRAN Call:

```
     CALL WAITFR (efn[,ids])

       efn  =  Event flag number

       ids  =  Directive status
```

Macro Call:

```
     WTSE$  efn

       efn  =  Event flag number
```

APPENDIX B

STANDARD ERROR CODES

The symbols listed below are associated with the directive status codes returned by the RSX-11M/M-PLUS Executive. They are determined (by default) at task-build time. To include these in a MACRO-11 program, use the following two lines of code:

```
        .MCALL  DRERR$
        DRERR$
```

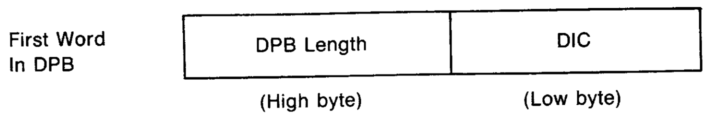
```
;
; STANDARD ERROR CODES RETURNED BY DIRECTIVES IN THE DIRECTIVE STATUS
; WORD
;
        IS.CLR  +00     EVENT FLAG WAS CLEAR
        IS.SUC  +01     OPERATION COMPLETE, SUCCESS
        IS.SET  +02     EVENT FLAG WAS SET
;
;
;
        IE.UPN  -01.    INSUFFICIENT DYNAMIC STORAGE
        IE.INS  -02.    SPECIFIED TASK NOT INSTALLED
        IE.UNS  -04.    INSUFFICIENT DYNAMIC STORAGE FOR SEND
        IE.ULN  -05.    UNASSIGNED LUN
        IE.HWR  -06.    DEVICE DRIVER NOT RESIDENT
        IE.ACT  -07.    TASK NOT ACTIVE
        IE.ITS  -08.    DIRECTIVE INCONSISTENT WITH TASK STATE
        IE.FIX  -09.    TASK ALREADY FIXED/UNFIXED
        IE.CKP  -10.    ISSUING TASK NOT CHECKPOINTABLE
        IE.TCH  -11.    TASK IS CHECKPOINTABLE
        IE.RBS  -15.    RECEIVE BUFFER TOO SMALL
        IE.PRI  -16.    PRIVILEGE VIOLATION
        IE.RSU  -17.    SPECIFIED VECTOR ALREADY IN USE
        IE.NSW  -18.    NO SWAP SPACE AVAILABLE
        IE.ILV  -19.    SPECIFIED VECTOR ILLEGAL
;
;
;
        IE.AST  -80.    DIRECTIVE ISSUED/NOT ISSUED FROM AST
        IE.MAP  -81.    ISR OR ENABLE/DISABLE INTERRUPT ROUTINE
                        NOT WITHIN 4K WORDS FROM VALUE OF
                        BASE ADDRESS & 177700
```

IE.IOP	-83.	WINDOW HAS I/O IN PROGRESS
IE.ALG	-84.	ALIGNMENT ERROR
IE.WOV	-85.	ADDRESS WINDOW ALLOCATION OVERFLOW
IE.NVR	-86.	INVALID REGION ID
IE.NVW	-87.	INVALID ADDRESS WINDOW ID
IE.ITP	-88.	INVALID TI PARAMETER
IE.IBS	-89.	INVALID SEND BUFFER SIZE (>255.)
IE.LNL	-90.	LUN LOCKED IN USE
IE.IUI	-91.	INVALID UIC
IE.IDU	-92.	INVALID DEVICE OR UNIT
IE.ITI	-93.	INVALID TIME PARAMETERS
IE.PNS	-94.	PARTITION/REGION NOT IN SYSTEM
IE.IPR	-95.	INVALID PRIORITY (>250.)
IE.ILU	-96.	INVALID LUN
IE.IEF	-97.	INVALID EVENT FLAG NUMBER
IE.ADP	-98.	PART OF DPB OUT OF USER'S SPACE
IE.SDP	-99.	DIC OR DPB SIZE INVALID

APPENDIX C

DIRECTIVE IDENTIFICATION CODES

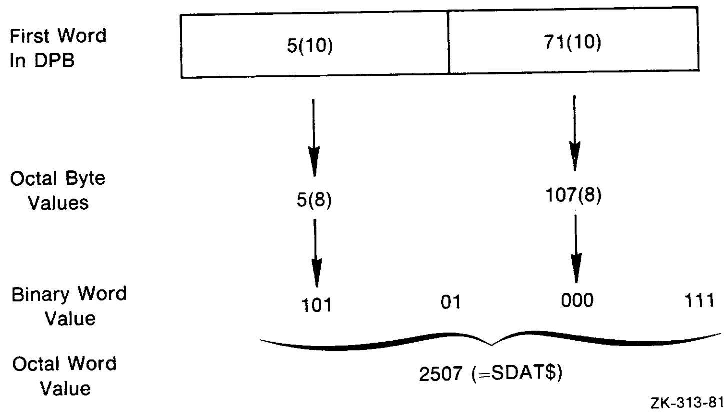
Directive Identification Codes (DICs) are used to identify each directive. The DIC appears in the low byte of the first (or only) word in the Directive Parameter Block (DPB). The DPB length (in words) appears in the high byte of the first DPB word. Thus, both bytes make up the word format shown below:



ZK-312-81

The remainder of this appendix contains a listing of directives arranged in numerical sequence, according to the octal value for the first DPB word. In addition, the DIC and DPB lengths are included as decimal values as they appear in Chapter 5.

This list can be used as a software debugging aid to quickly identify directives based on the octal value of the first word in a DPB. An example for the SDAT$ directive is provided below, illustrating the manner in which the octal value is obtained:



ZK-313-81

DIRECTIVE IDENTIFICATION CODES

Octal Value For DPB First Word	Directive (Macro Call)	Decimal DIC	Values For DPB Length
433	CMKT$	27.	1.
443	DECL$S	35.	1.
455	SPND$S	45.	1.
461	WSIG$S	49.	1.
463	EXIT$S	51.	1.
537	DSCP$S	95.	1.
541	ENCP$S	97.	1.
543	DSAR$S or IHAR$S	99.	1.
545	ENAR$S	101.	1.
563	ASTX$S	115.	1.
575	GSSW$S	125.	1.
603	STOP$S	131.	1.
637	ULGF$S	159.	1.
643	RMAF$S	163.	1.
1015	STAF$	13.	2.
1025	SRRA$	21.	2.
1035	EXST$	29.	2.
1037	CLEF$	31.	2.
1041	SETF$	33.	2.
1047	RDAF$	39.	2.
1051	WTSE$	41.	2.
1065	EXIF$	53.	2.
1067	CRRG$	55.	2.
1071	ATRG$	57.	2.
1073	DTRG$	59.	2.
1075	GTIM$	61.	2.
1077	GTSK$	63.	2.
1121	RREF$	81.	2.
1153	SRDA$	107.	2.
1155	SPRA$	109.	2.
1157	SFPA$	111.	2.
1161	GMCX$	113.	2.
1165	CRAW$	117.	2.
1171	MAP$	121.	2.
1173	UMAP$	123.	2.
1207	STSE$	135.	2.
1227	ELVT$	151.	2.
1235	CRGF$	157.	2.
1237	ELGF$	159.	2.
1241	STAF$	161.	2.
1245	SPEA$	165.	2.
1247	SREA$	167.	2.
1405	GLUN$	5.	3.
1431	CSRQ$	25.	3.
1433	CMKT$	27.	3.
1447	RDXF$	39.	3.
1453	WTLO$	43.	3.
1457	RSUM$	47.	3.
1475	STIM$	61.	3.
1523	ABRT$	83.	3.
1531	EXTK$	89.	3.
1547	SVDB$	103.	3.
1551	SVTK$	105.	3.
1605	USTP$	133.	3.

DIRECTIVE IDENTIFICATION CODES

Octal Value For DPB First Word	Directive (Macro Call)	Decimal DIC	Values For DPB Length
1611	STLO$	137.	3.
1617	CNCT$	143.	3.
1633	SCAL$S	155.	3.
1647	SREX$	167.	3.
2007	ALUN$	7.	4.
2011	ALTP$	9.	4.
2101	GPRT$ or GREG$	65.	4.
2113	RCVD$	75.	4.
2115	RCVX$	77.	4.
2213	RCST$	139.	4.
2223	EMST$	147.	4.
2427	MRKT$	23.	5.
2505	SREF$	69.	5.
2507	SDAT$	71.	5.
2625	CRVT$	149.	5.
3113	VRCD$	75.	6.
3115	VRCX$	77.	6.
3213	VRCS$	139.	6.
3413	RQST$	11.	7.
3601	CINT$	129.	7.
3615	SDRC$	141.	7.
4107	VSDA$	71.	8.
5421	RUN$	17.	11.
6001	QIO$	1.	12.
6003	QIOW$	3.	12.
6413	SPWN$	11.	13.
7013	SPWN$	11.	14.
24577	GMCR$	127.	41.

APPENDIX D

RSX-11 SYSGEN SELECTION OF EXECUTIVE DIRECTIVES

The following list contains all Executive directive macro calls described in this manual and means of selection at SYSGEN time. Those directives not available for specific RSX-11 systems are noted as N/A. Directives that are SYSGEN options are noted as O. The number in parentheses after the O refers to the SYSGEN options at the end of the list. Directives that are standard (not SYSGEN options) are indicated by an asterisk (*).

Directive Macro Call	**System Type**		
	RSX-11S	**RSX-11M**	**RSX-11M-PLUS**
ABRT$	*	*	*
ALTP$	O (1)	O (1)	*
ALUN$	*	*	*
ASTX$S	O (2)	O (2)	*
ATRG$	O (3)	O (3)	*
CINT$	O (1)	O (1)	*
CLEF$	*	*	*
CMKT$	*	*	*
CNCT$	O (4)	O (4)	*
CPCR$	N/A	N/A	*
CRAW$	O (3)	O (3)	*
CRGF$	O (5)	O (5)	*
CRRG$	O (3)	O (3)	*
CRVT$	N/A	N/A	O (6)
CSRQ$	*	*	*
DECL$S	*	*	*
DSAR$S or IHAR$S	O (2)	O (2)	*
DSCP$S	N/A	O (7)	*
DTRG$	O (3)	O (3)	*
ELAW$	O (3)	O (3)	*
ELGF$	O (5)	O (5)	*
ELVT$	N/A	N/A	O (6)
EMST$	O (4)	O (4)	*
ENAR$S	O (2)	O (2)	*
ENCP$S	N/A	O (7)	*
EXIF$	*	*	*
EXIT$S	*	*	*
EXST$	O (4)	O (4)	*
EXTK$	O (1)	O (1)	*
GCCI$	N/A	O (15)	O (15)
GCII$	N/A	O (15)	O (15)
GLUN$	*	*	*
GMCR$	N/A	*	*
GMCX$	O (3)	O (3)	*
GPRT$	O (1)	O (1)	*
GREG$	O (3)	O (3)	*
GSSW$S	O (1)	O (1)	*
GTIM$	*	*	*

RSX-11 SYSGEN SELECTION OF EXECUTIVE DIRECTIVES

Directive Macro Call	System Type RSX-11S	RSX-11M	RSX-11M-PLUS
GTSK$	O (1)	O (1)	*
MAP$	O (3)	O (3)	*
MRKT$	*	*	*
MSDS$	N/A	N/A	O (16)
MVTS$	N/A	N/A	O (16)
QIO$	*	*	*
QIOW$	*	*	*
RCST$	O (13,14)	O (13,14)	*
RCVD$	O (14)	O (14)	*
RCVX$	O (14)	O (14)	*
RDAF$	*	*	*
RDEF$	N/A	N/A	*
RDXF$	O (5)	O (5)	*
RMAF$S	N/A	N/A	O (8)
RPOI$	O (1,4)	O (1,4)	*
RQST$	*	*	*
RREF$	O (1,3)	O (1,3)	*
RSUM$	*	*	*
RUN$	*	*	*
SCAA$	N/A	O (15)	O (15)
SCAL$S	N/A	N/A	O (9)
SCLI$	N/A	O (15)	O (15)
SDAT$	O (14)	O (14)	*
SDRC$	O (4,14)	O (4,14)	*
SDRP$	N/A	O (14,15)	*
SETF$	*	*	*
SFPA$	O (2, 10)	O (2, 10)	O (10)
SMSG$	N/A	O (12)	*
SNXC$	N/A	N/A	*
SPEA$	N/A	N/A	*
SPND$S	*	*	*
SPRA$	O (2, 11)	O (2, 11)	*
SPWN$	O (4)	O (4)	*
SRDA$	O (2, 14)	O (2, 14)	*
SREA$	O (1,2)	O (1,2)	*
SREF$	O (1,3)	O (1,3)	*
SREX$	O (1,2)	O (1,2)	*
SRRA$	O (1, 2, 3)	O (1, 2, 3)	*
SRRC$	N/A	N/A	*
STAF$	N/A	N/A	O (8)
STIM$	O (1)	O (1)	*
STLO$	O (13)	O (13)	*
STOP$S	O (13)	O (13)	*
STSE$	O (13)	O (13)	*
SVDB$	*	*	*
SVTK$	*	*	*
ULGF$S	O (5)	O (5)	*
UMAP$	O (3)	O (3)	*
USTP$	O (13)	O (13)	*
VRCD$	N/A	N/A	*
VRCS$	N/A	N/A	*
VRCX$	N/A	N/A	*
VSDA$	N/A	N/A	*
VSRC$	N/A	N/A	*
WSIG$S	*	*	*
WTLO$	*	*	*
WTSE$	*	*	*

RSX-11 SYSGEN SELECTION OF EXECUTIVE DIRECTIVES

SYSGEN Options:

1. Specific Executive directive support
2. AST support
3. Memory management directives
4. Parent/offspring tasking support
5. Group-global event flag support
6. Virtual terminal support
7. Checkpointing support
8. Multiprocessor support
9. Supervisor-mode library support
10. Floating Point Processor support
11. Powerfail recovery support
12. Error Logging support
13. Stop bit synchronization support
14. Send/receive support
15. Alternate CLI support
16. D-space support

INDEX

INDEX

INDEX

RSX-11M/M-PLUS
Executive
Reference Manual
Order No. AA-L675A-TC

READER'S COMMENTS

NOTE: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. If you require a written reply and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

Did you find errors in this manual? If so, specify the error and the page number.

Please indicate the type of user/reader that you most nearly represent.

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Other (please specify) ____________

Name ____________ Date ____________

Organization ____________

Street ____________

City ____________ State ____________ Zip Code or Country ____________

Do Not Tear - Fold Here and Tape

digital

No Postage
Necessary
if Mailed in the
United States

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO.33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

BSSG PUBLICATIONS ZK1-3/J35
DIGITAL EQUIPMENT CORPORATION
110 SPIT BROOK ROAD
NASHUA, NEW HAMPSHIRE 03061

Do Not Tear - Fold Here

Cut Along Dotted Line