RSX-11M
Executive Reference Manual

Order No. AA-2544D-TC

digital

First Printing, November 1974
Revised: September 1975
November 1976
December 1977

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this document.

The software described in this document is furnished under a license and may only be used or copied in accordance with the terms of such license.

No responsibility is assumed for the use or reliability of software on equipment that is not supplied by DIGITAL or its affiliated companies.

The postage-prepaid READER'S COMMENTS form on the last page of this document requests the user's critical evaluation to assist us in preparing future documentation.

The following are trademarks of Digital Equipment Corporation:

DIGITAL	DECsystem-10	MASSBUS
DEC	DECtape	OMNIBUS
PDP	DIBOL	OS/8
DECUS	EDUSYSTEM	PHA
UNIBUS	FLIP CHIP	RSTS
COMPUTER LABS	FOCAL	RSX
COMTEX	INDAC	TYPESET-8
DDT	LAB-8	TYPESET-11
DECCOMM	DECSYSTEM-20	TMS-11
ASSIST-11	RTS-8	ITPS-10

RSX-11M

Executive Reference Manual

Order No. AA-2544D-TC

RSX-11M Version 3.1

To order additional copies of this document, contact the Software Distribution Center, Digital Equipment Corporation, Maynard, Massachusetts 01754

digital equipment corporation · maynard, massachusetts

First Printing, November 1974
Revised: September 1975
November 1976
December 1977

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this document.

The software described in this document is furnished under a license and may only be used or copied in accordance with the terms of such license.

No responsibility is assumed for the use or reliability of software on equipment that is not supplied by DIGITAL or its affiliated companies.

The postage-prepaid READER'S COMMENTS form on the last page of this document requests the user's critical evaluation to assist us in preparing future documentation.

The following are trademarks of Digital Equipment Corporation:

DIGITAL	DECsystem-10	MASSBUS
DEC	DECtape	OMNIBUS
PDP	DIBOL	OS/8
DECUS	EDUSYSTEM	PHA
UNIBUS	FLIP CHIP	RSTS
COMPUTER LABS	FOCAL	RSX
COMTEX	INDAC	TYPESET-8
DDT	LAB-8	TYPESET-11
DECCOMM	DECSYSTEM-20	TMS-11
ASSIST-11	RTS-8	ITPS-10

CONTENTS

FIGURES

TABLES

PREFACE

0.1 MANUAL OBJECTIVES AND READER ASSUMPTIONS

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

0.2 STRUCTURE OF THE DOCUMENT

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

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

Chapter 3 introduces the concept of extended logical address space and describes the associated memory management directives.

Chapter 4 contains a short summary of all directives, listed according to category. The summary is followed by the detailed directive specifications. The specifications are arranged alphabetically according to macro call.

Appendix A contains abbreviated specifications of all the directives (directive name, FORTRAN call, and macro call only), arranged alphabetically according to macro call.

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

0.3 ASSOCIATED DOCUMENTS

The following manuals are prerequisite sources of information for readers of this manual:

RSX-11M Task Builder Reference Manual

IAS/RSX-11 MACRO-11 Reference Manual

PDP-11 FORTRAN Language Reference Manual

Other documents related to the contents of this manual are described briefly in the RSX-11M/RSX-11S Documentation Directory, Order No. AA-2593D-TC. The directory defines the intended readership of each manual in the RSX-11M/RSX-11S set and provides a brief synopsis of each manual's contents.

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-11M features that can be selected during system generation. The discussion of these features always assumes that the features have been generated for the system. See the RSX-11M System Generation Reference Manual for a list of optional features.

1.1 INTRODUCTION

A system directive is a request from a task to the Executive to perform an indicated operation. The programmer uses the directives to control the execution and interaction of tasks. The MACRO-11 programmer usually issues directives in the form of macros defined in the system macro library. The FORTRAN programmer issues system directives in the form of calls to subroutines contained in the system object module library.

System directives enable tasks to perform functions such as the following:

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

Directives are implemented via 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. The Executive aborts the task unless the task has specified that it wants to receive control when such traps occur. Note that RSX-11M reserves EMT 370 and above for possible use as special system traps in the future.

A MACRO-11 programmer should use the system directives supplied in the system macro library for directive calls, rather than hand-coding calls to directives. The programmer then needs only to reassemble the program to incorporate any changes in the directive specifications.

Sections 1.2, 1.3, and 1.6 are directed to 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 for that language.

1.2 DIRECTIVE PROCESSING

There are four steps in the processing of a system directive:

1. The user task issues a directive with arguments that are mainly for the creation of 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 from the directive (or a DIR$ macro).

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 from 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 odd, thus the Executive can 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. The Executive also clears or sets the Carry bit in the Processor Status word (PS) to indicate acceptance or rejection, respectively, of the directive. The Directive Status Word (DSW), addressed symbolically as $DSW, is set to indicate a more specific cause for acceptance or rejection of the directive.* The DSW usually has a value of +1 for acceptance and a range of negative values for rejection (exceptions are success return codes for the directives CLEF$, SETF$, and GPRT$, among others). RSX-11M associates DSW values with symbols, using mnemonics that report either successful

---

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

completion or the cause of an error (see Section 1.3). (The ISA FORTRAN calls CALL START and CALL WAIT are exceptions; ISA requires positive numeric error codes. See Sections 4.3.39 and 4.3.30 for details.) The detailed return values are listed 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:

1. Detaches all attached devices
2. Flushes the Asynchronous System Trap (AST) queue (ASTs are described in Chapter 2 of this manual)
3. Flushes the clock queues for outstanding Mark Time requests for the task (see Section 4.3.30)
4. Flushes the receive-data and receive-by-reference queues
5. Closes all open files (files open for write access are locked)
6. Cancels all outstanding I/O
7. Detaches all attached regions, except in the case of a fixed task in a system that supports the memory management directives, where no detaching takes place (see Section 3.3.2)
8. Frees the task's memory if the task is 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 WAITFOR directive corresponding to a previously issued MARK TIME directive that the Executive has rejected. Care should always be taken to ensure that a directive has been completed successfully.

1.3 ERROR RETURNS

As stated above, RSX-11M associates 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 4 contains specific, directive-related interpretations of the error codes.

1.4 USING THE DIRECTIVE MACROS

To issue a directive, a task supplies the system with a directive code and parameters (the DPB), and issues an EMT 377 instruction.

The DPB can be created in two ways:

1. To adapt to the requirements of reentrant code --

   The reentrant method allows for the creation of the DPB on the stack at run time (see Section 1.4.1.3, which describes the $S form of directive).

2. To adapt to code that does not have reentrant requirements --

   The non-reentrant method allows for the creation of the DPB 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).

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

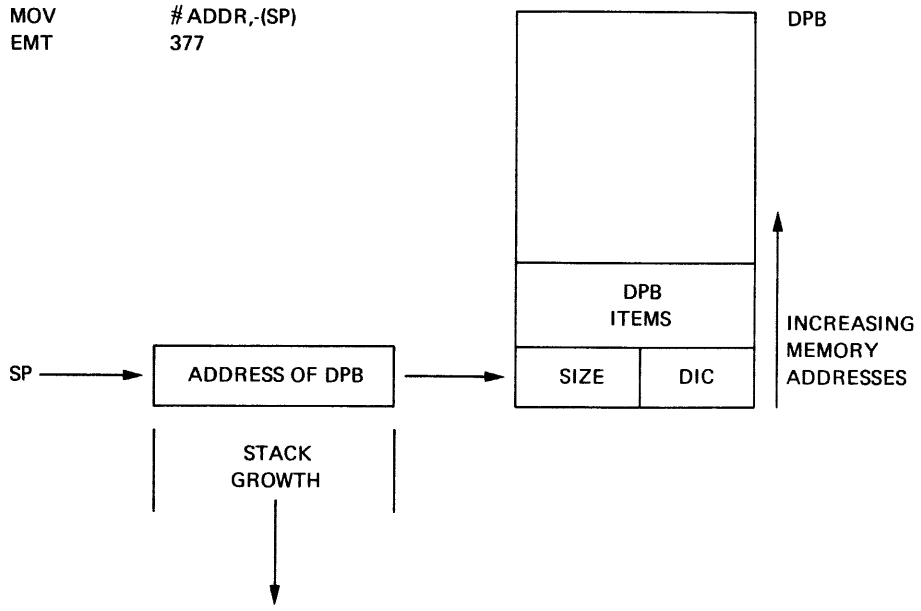


Figure 1-1 Directive Parameter Block (DPB) Pointer on the Stack

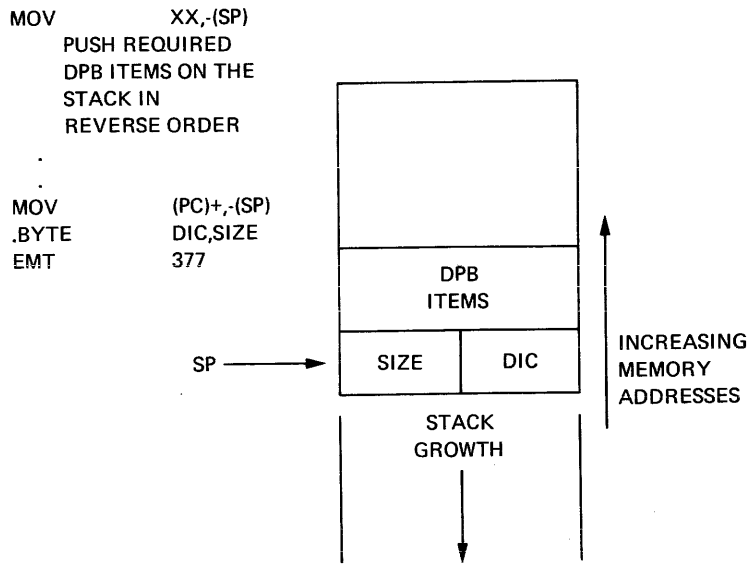


Figure 1-2 Directive Parameter Block (DPB) on the Stack

1.4.1 Macro Name Conventions

To use system directives, a MACRO-11 programmer includes directive macro calls in programs. The macros for the RSX-11M directives are contained in the System Macro Library (LB:[1,1]RSXMAC.SML). To make the macros available to a program, the programmer issues the .MCALL assembler directive. 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 (omission of the optional letter) is useful for a directive operation that is to be issued several times from different locations in a non-reentrant program segment. 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.

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). The programmer can alter individual parameters in the DPB. This might be done, for example, if the directive is to be used many times with varying parameters.

1.4.1.2 **$C Form** - Programmers should use the $C form when a directive is to be issued only once, and the program segment does not need to be reentrant. 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 DPB address $$$ is redefined by the new value of the assembler's location counter each time an additional $C directive is issued.)

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

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 macro call, the parameters required for DPB construction must be valid expressions to be used in MACRO-11 data storage instructions (such as .BYTE, .WORD, and .RAD50). (This is not true for the PSECT argument or 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 both code to push a DPB onto the stack, and 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 the Stack Pointer should not be used to address the parameters.* (As above, the error routine argument is an address for a JSR instruction.)

1.4.2 The DIR$ Macro

The DIR$ macro allows the programmer 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.

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

Macro Call: DIR$ adr,err

adr and err are optional

adr is 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 is the address of the error return (see Section 1.4.3). If this error return is not specified, an error simply sets the C-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. 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
```

The $ form of directive macro does not accept an error address argument.

---

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

1.4.4 Symbolic Offsets

Most system directive macro calls generate local symbolic offsets. The symbols are unique to each directive and each is assigned an index value corresponding to the number of bytes into the DPB that a given DPB element is located.

Because the offsets are defined symbolically, the programmer who must refer to or modify DPB elements can do so 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 DPBs longer than one word generate local offsets. All informational directives (see Chapter 4, Table 4-2) 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.

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 PSECT.

```
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 PSECT $DPB$$ both a DPB and 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 provides an extensive set of subroutines for use in FORTRAN programs, to perform RSX-11M system directive operations.

The directive descriptions in Chapter 4 describe the FORTRAN subroutine calls, as well as the macro calls.

The FORTRAN subroutines fall into three basic groups:

1. Subroutines based on the Instrument Standard of America (ISA) Standard ISA 62.1 -- These subroutines are included in the subroutine descriptions associated with the macro calls. See Chapter 4.

2. Subroutines designed to use and control specific process control interface devices supplied by DIGITAL and supported by the RSX-11M operating system

3. Subroutines for performing RSX-11M 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 of 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 of 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. To use one of these routines, the programmer includes 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 routine is user-defined. If a routine is not user-defined, the Task Builder automatically searches for it in the system object module library. If the routine 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, programmers 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 4.

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 variable of type REAL that represents the task name as Radix-50 code. 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 named 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)
```

Task names may also be defined 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 of 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 the programmer specifies the /I4 option switch when compiling a program, particular care must be taken to ensure that all integer 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, programmers should use the GETADR subroutine described below.

Calling Sequence:

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

ipm	is an array of dimension n.
arg1,...argn	are 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.

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 the programmer to specify a buffer address in the CALL QIO directive (see Section 4.3.31).

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 4 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
AST SERVICE EXIT	ASTX$S	Not available
ATTACH REGION	ATRG$	CALL ATRG
CONNECT TO INTERRUPT VECTOR	CINT$	Not available
CLEAR EVENT FLAG	CLEF$	CALL CLREF
CANCEL MARK TIME REQUESTS	CMKT$S	CALL CANMT
CANCEL TIME BASED INITIATION REQUESTS	CSRQ$	CALL CANALL
CREATE ADDRESS WINDOW	CRAW$	CALL CRAW
CREATE REGION	CRRG$	CALL CRRG
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
ENABLE AST RECOGNITION	ENAR$S	CALL ENASTR
ENABLE CHECKPOINTING	ENCP$S	CALL ENACKP
EXITIF	EXIF$	CALL EXITIF
TASK EXIT	EXIT$S	CALL EXIT
EXTEND TASK	EXTK$	CALL EXTTSK
GET LUN INFORMATION	GLUN$	CALL GETLUN
GET MAPPING CONTEXT	GMCX$	CALL GMCX
GET MCR COMMAND LINE	GMCR$	CALL GETMCR

(continued on next page)

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

Directive	Macro Call	FORTRAN Subroutine
GET PARTITION PARAMETERS	GPRT$	CALL GETPAR
GET REGION PARAMETERS	GREG$	CALL GETREG
GET SENSE SWITCHES	GSSW$S	CALL READSW CALL SSWTCH
GET TIME PARAMETERS	GTIM$	Several subroutines available (see the appropriate FORTRAN User's Guide)
GET TASK PARAMETERS	GTSK$	CALL GETTSK
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$	Only a single event flag can be read by a FORTRAN task: CALL READEF
RECEIVE DATA	RCVD$	CALL RECEIV
RECEIVE DATA OR EXIT	RCVX$	CALL RECOEX
RECEIVE BY REFERENCE	RREF$	CALL RREF
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
SET EVENT FLAG	SETF$	CALL SETEF
SPECIFY FLOATING POINT EXCEPTION AST	SFPA$	Not available

(continued on next page)

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

Directive	Macro Call	FORTRAN Subroutine
SPECIFY POWER RECOVERY AST	SPRA$	EXTERNAL SUBNAM CALL PWRUP (SUBNAM) (to establish an AST) CALL PWRUP (to remove an AST)
SPECIFY RECEIVE DATA AST	SRDA$	Not available
SPECIFY RECEIVE BY REFERENCE AST	SRRA$	Not available
SPECIFY SST VECTOR TABLE FOR DEBUGGING AID	SVDB$	Not available
SUSPEND	SPND$S	CALL SUSPND
SPECIFY SST VECTOR TABLE FOR TASK	SVTK$	Not available
UNMAP ADDRESS WINDOW	UMAP$	CALL UNMAP
WAIT FOR LOGICAL OR OF EVENT FLAGS	WTLO$	CALL WFLOR
WAIT FOR SIGNIFICANT EVENT	WSIG$S	CALL WFSNE
WAIT FOR SINGLE EVENT FLAG	WTSE$	CALL WAITFR

1.5.3 Error Conditions

Each subroutine call includes an optional argument (ids). When a programmer specifies 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 4.3.30 and 4.3.39).

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

1. SYSTEM 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).

2. SYSTEM DIRECTIVE: INVALID EVENT FLAG NUMBER
   This message indicates that an event flag number in a call to WFLOR (WAIT FOR LOGICAL "OR" OF EVENT FLAGS) was not in the range 1 to 64 (OTS error number 101).

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 -- dormant and active. The active state has two substates -- ready-to-run and blocked.

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 Operator's Procedures Manual.) Once a task has been installed, it is either dormant or active. These states are defined as follows:

1. Dormant -- Immediately following the Monitor Console Routine's processing of an INStall command, 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 (that is, neither a RQST$ nor RUN$ macro, nor an MCR RUN command, has been issued for it).

2. Active -- A task is active from the time it is requested until the time it exits. The request is either an issuance of the RQST$ or RUN$ macro, or an MCR RUN command issued by an operator from a terminal. An active task is eligible for scheduling, whereas a dormant task is not.

   An active task can be in one of two substates, ready-to-run or blocked.

   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.

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
- An MCR 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 WAITFOR condition
- The Executive checkpoints a task out of memory
- A checkpointable task issues a terminal input request*

---

* Only in systems that support the checkpointing of tasks during terminal input.

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

- A RSUM$ directive issued by another task
- An MCR RESUME command
- A WAITFOR condition is satisfied
- The Executive reads a checkpointed task into memory
- Terminal input for a checkpointable task completes*

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

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

1.6.2 Removing an Installed Task

To remove an installed task from the system, the user issues the MCR command REMOVE from a privileged terminal. Refer to the RSX-11M Operator's Procedures Manual.

* Only in systems that support the checkpointing of tasks during terminal input.

CHAPTER 2

SIGNIFICANT EVENTS AND SYSTEM TRAPS

This chapter introduces the concept of significant events and describes the ways in which a programmer can make use of event flags and synchronous and asynchronous system traps.

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. 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 4.3.40)
- The execution of a SEND BY REFERENCE or a RECEIVE BY REFERENCE directive (see Section 4.3.37)
- The execution of an ALTER PRIORITY directive (see Section 4.3.2)
- The removal of an entry from the clock queue (e.g., 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 4.3.11)
- The execution of the round-robin scheduling algorithm at the end of a round-robin scheduling interval

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.) When a task requests a system operation (such as an I/O transfer), the task may associate an event flag with the completion of the operation. When the event occurs, the Executive sets the specified flag. Section 2.2.1 describes in several examples how tasks can use event flags to coordinate task execution.

Sixty-four event flags are available to enable tasks to distinguish one event from another. Each event flag has a corresponding unique Event Flag Number (EFN). The first 32 (1-32) flags are unique to each task and are set or cleared as a result of that task's operation. The second 32 flags (33-64) are common to all tasks and are therefore called 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.

Tasks can use the common flags for intertask communication or their own local event flags internally. The setting, clearing, and testing of local flags can be performed by using SET EVENT FLAG (SETF$), CLEAR EVENT FLAG (CLEF$), and READ ALL EVENT FLAGS (RDAF$) directives.

Programmers must take great care when setting or clearing event flags, especially common 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. The implicit handling of event flags substantially reduces errors caused by multiple setting and clearing of event flags.

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 WAITFOR 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 WAITFOR directive. When Task A's SEND DATA directive has caused the Executive to set flag 42 and to cause a significant event, Task B issues a RECEIVE DATA directive because its WAITFOR condition has been satisfied.

Example 3

A task contains a QUEUE I/O REQUEST and an associated WAITFOR directive, which both 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 a WAITFOR directive that specifies the same local event flag, 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 WAITFOR directive. The local event flag used in this manner ensures that the task does not attempt to 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 WAITFOR 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 WAITFOR directive before the local flag has been set (that is, before the time interval has elapsed) the Executive blocks the task. The task then resumes when the Executive sets the flag.

Specifying an event flag does not imply that a WAITFOR directive must be issued. Event flag testing can be performed at any time. The purpose of a WAITFOR directive is to stop task execution until an indicated significant event occurs. Hence, it is not necessary to issue a WAITFOR directive immediately following a QUEUE I/O REQUEST or a MARK TIME directive.

If a task issues a WAITFOR 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.

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 distinct 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 previous instructions are repeated. For example, an illegal instruction causes an SST.

- Asynchronous System Traps (ASTs) -- ASTs detect significant events that occur asynchronously to the task's execution. That is, the task has no direct control over the precise time that the event occurs. 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 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:

1. Illegal instructions
2. Instructions with invalid addresses
3. Trap instructions
4. 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, only the debugging aid will receive the SST.

SST routines must always be reentrant because 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) and Program Counter (PC) onto the task's stack. The SST returns control to the task by issuing an RTI or RTT instruction. Note that the task's general purpose registers R0-R6 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. For example, 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. However, programmers should remember that any operations performed by the routine (such as the modification of 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. The user specifies 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:

```
WD. 00 -- Odd or nonexistent memory address error -- (Also, on
          some PDP-11 processors (e.g., PDP-11/45), an illegal
          instruction traps here rather than through word 04.)
WD. 01 -- Memory protect violation
WD. 02 -- T-bit trap or execution of a BPT instruction
WD. 03 -- Execution of an IOT instruction
WD. 04 -- Execution of a reserved instruction
WD. 05 -- Execution of a non-RSX EMT instruction
WD. 06 -- Execution of a TRAP instruction
WD. 07 -- Synchronous floating point exception
```

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.

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)*
SP+02 -- Virtual PC of the faulting instruction (SR2)*
SP+00 -- Instruction backup register (SR1)*
```

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 (usually by means of an RTI or RTT instruction).

* For details of SR0, SR1, and SR2, see the memory management unit section of the appropriate PDP-11 Processor Handbook.

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, a task can associate an AST with the completion of an I/O operation. When the AST informs the task that the event has occurred, the task can service the event and then 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. This capability enables the user to 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.

In contrast to the execution of an SST routine, which is indistinguishable 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.

The following notes describe general characteristics and uses of ASTs:

- If an AST occurs while the related task is executing, the task is interrupted in order to execute the AST service routine.

- If an AST occurs while another AST is being processed, the Executive queues the latest AST (First-In-First-Out or FIFO) and 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 when an associated AST occurs, the task remains suspended after the AST routine has been executed, except that the suspended task can be explicitly resumed either by the AST service routine itself, or by another task (the MCR RESUME command, for example).

- If an AST occurs while the related task is waiting for an event flag setting (a WAITFOR directive), the task continues to wait after execution of the AST service routine until the AST service routine itself or another task sets the appropriate event flag.

- If an AST occurs for a checkpointed task, the Executive queues the AST (FIFO), and then activates it when the task returns to direct competition for processor resources. Powerfail recovery ASTs are an exception, however. The Executive does not activate powerfail recovery ASTs that occurred for a task while the task was checkpointed.

  When a task is checkpointed back into memory, the Executive issues an AST for the task if its receive queue contains one or more entries. This practice prevents checkpointed tasks from losing receive ASTs.

- An optional RSX-11M feature allows the checkpointing of tasks during terminal input. When this feature is included, the Executive stops the execution of a checkpointable task when the terminal driver receives an input request for the task. The task resumes execution when the terminal input has finished. A stopped task can execute an AST service routine if an AST occurs; but the task remains stopped after the routine finishes unless the terminal input has finished in the meantime. Note, however, that an AST routine itself can reactivate the stopped task by issuing an I/O Kill function for the task's terminal input request.

- 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.

- Two directives, DISABLE AST RECOGNITION and ENABLE AST RECOGNITION, allow ASTs to be queued 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 WAITFOR 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 WAITFOR 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-R6 are not saved. If the routine makes use of them, it must save and restore them itself.

The WAITFOR 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 WAITFOR SINGLE EVENT FLAG or WAITFOR LOGICAL "OR" OF EVENT FLAGS 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 WAITFOR mask word is meaningless if the task has not issued either type of WAITFOR directive.

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 4.2.4). When the AST service routine exits, it returns control to one of two places -- another AST or the original task.

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

1. 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

2. 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

3. 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. If the task specifies one of these directives, an AST will occur when a message or reference is sent to the task. An AST also occurs when a task has at least one item in the receive queue when the task is checkpointed into or initially loaded into memory. 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

4. 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).

5. 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)

CHAPTER 3

MEMORY MANAGEMENT DIRECTIVES

This chapter discusses the concepts of extended logical address space, regions, and virtual address windows. The chapter also introduces the related memory management directives.

3.1 ADDRESSING CAPABILITIES OF AN RSX-11M TASK

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. To avoid limiting the size of a task to its addressing capability, RSX-11M allows it to be overlaid. An overlaid task is divided into segments -- a single root segment, which is always in memory, and any number of 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 task segments are not in memory, they reside on disk. When resident task segments cannot exceed 32K words, a task requiring large amounts of data must access disk-based data that cannot fit into memory with what is already there. In addition, transmission of large amounts of data between tasks is only practical via 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 always 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 memory requirements.

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

- 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 below).

- Logical Address Space -- A task's logical address space is the total amount of physical memory to which the task has access rights. The task can divide its logical address space into various areas called regions (see Section 3.3 below). Each region occupies a continuous block of memory.

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. 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.

3.2 VIRTUAL ADDRESS WINDOWS

In order to manipulate the mapping of virtual addresses to various logical areas, the user must first divide a task's 32K of virtual address space into segments. These segments are called virtual address windows. Each window encompasses a continuous 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 (as discussed below, window 0 is not available to the user). The size of each window can range from a minimum of 32 words to a maximum of 32K minus 32 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. When linking the task, the programmer specifies the required number of window blocks to be set up by the Task Builder (see the RSX-11M Task Builder Reference Manual). The number of blocks should equal the maximum number of windows that will exist concurrently while the task is running.

A window's identification is a number from 0 to 7, which is an index to the window's corresponding window block. The address window identified by 0 is the window that always maps the task's header and root 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.

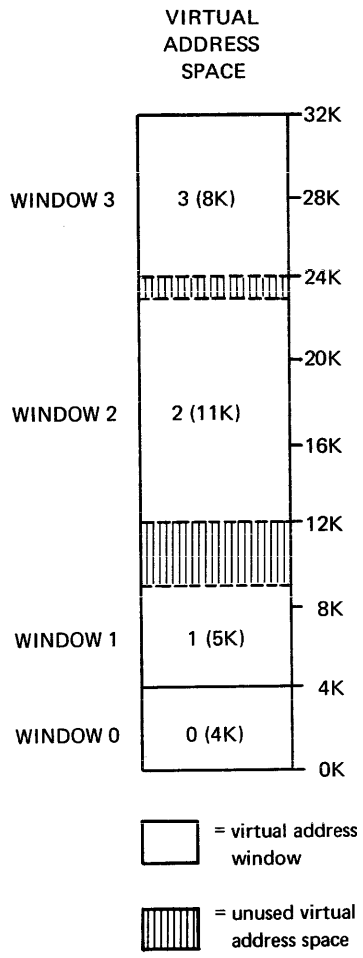


Figure 3-1 Virtual Address Windows

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, the address does not point anywhere. 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.

Once a task has defined the necessary windows and regions, the task 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.

NOTE

It is currently possible for a task with outstanding I/O to unmap from a region (although it cannot detach from any -- see Section 3.3.2). Because this feature may be impossible to support in future releases of the system, it is recommended that users consider carefully before designing an application that is based on this capability.

3.3 REGIONS

The current window-to-region mapping context determines the 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 region:

- Task Region -- The task region is a continuous block of memory in which the task runs.
- Static Common Region -- A static common region is an area defined by an operator at run time or at system generation time, such as a global common area.
- Dynamic Region -- A dynamic region is a region created dynamically at run time by issuing the memory management directives.

Tasks refer to a region by means of a region ID returned to the task by the Executive. Region ID 0 always refers to a task's task region. 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. (A task's logical address space can enlarge or contract dynamically.) The header and root segment are always part of the task region. Since a region occupies a continuous 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 the first two figures.

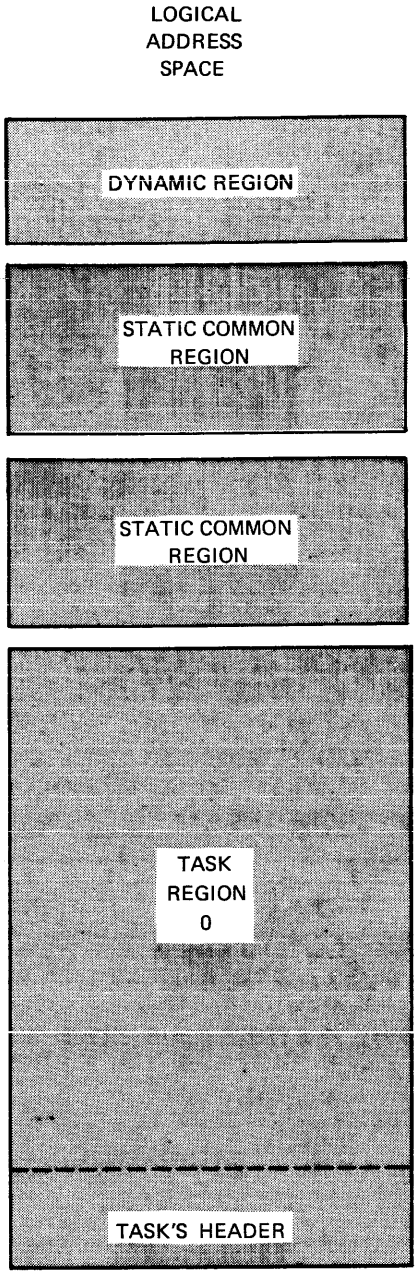


Figure 3-2 Region Definition Block

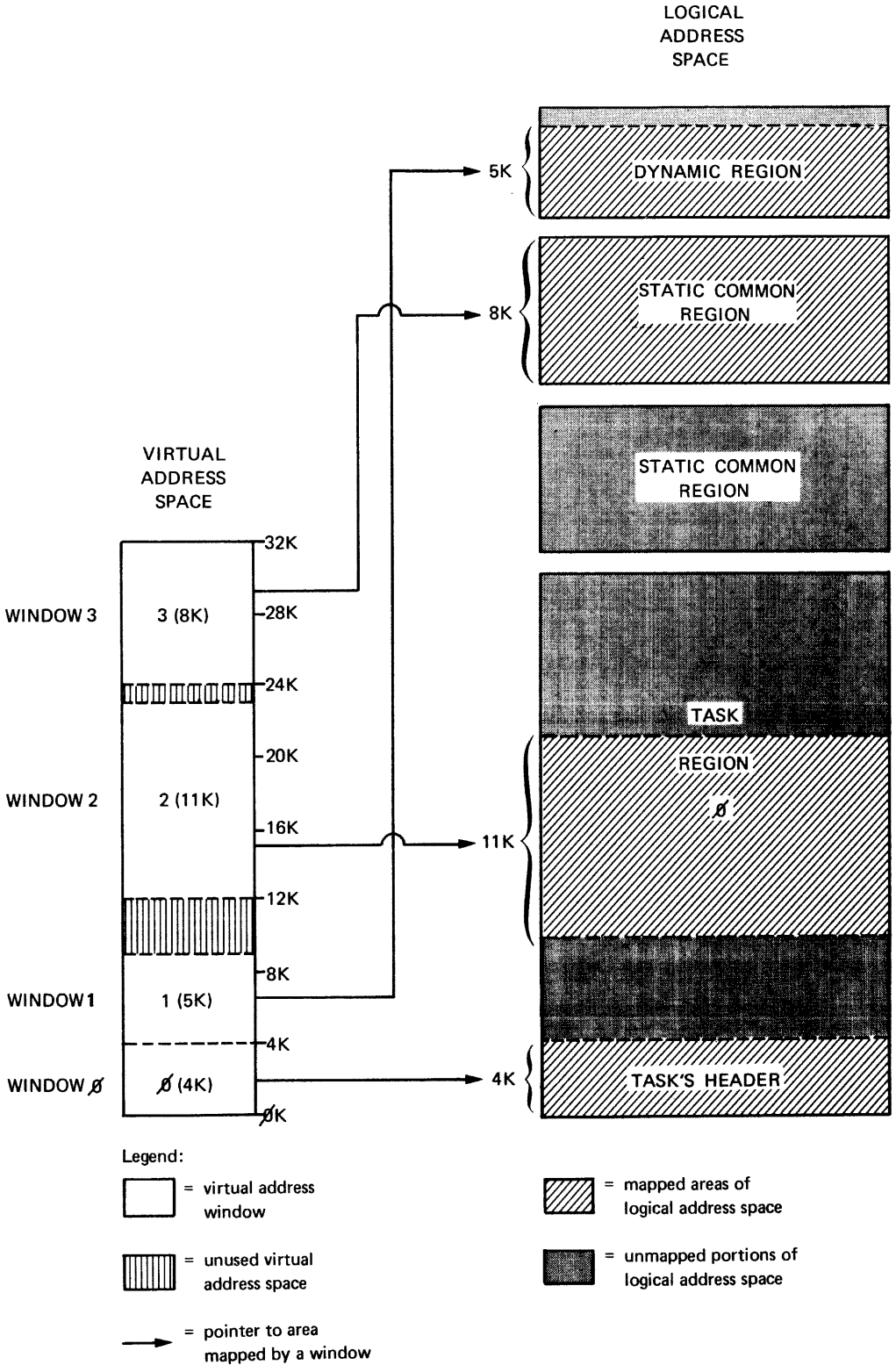


Figure 3-3 Mapping Windows to Regions

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. Tasks can map to required routines at run time, rather than link to them at task-build time.

3.3.2 Attaching to Regions

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

1. All regions that are linked to a task at task-build time are automatically attached.

2. A task can issue a directive to attach itself to a named static common region or a named dynamic region.

3. A task can request the Executive to attach any region within its own logical address space (other than its task region) to another specified 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.)

3.3.3 Region Protection

A task cannnot indiscriminately attach to any region. The following criteria determine how tasks can attach to regions outside their logical address space:

- 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.

- When a task creates a dynamic region, it may or may not give that region a name. If the dynamic region is named, any task can map to it as long as it knows the name and there is no protection violation. If a dynamic region is unnamed, a task can map to the region only if the task that created the dynamic region issues a SEND BY REFERENCE directive addressed to the requesting task.

- Any task can issue a SEND BY REFERENCE directive to attach any region (except the task region) to another specific task. 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 as long as there is no protection violation.

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 system-controlled partition and optionally attaches the issuing task to it. (See Section 4.3.10.)

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 4.3.5.)

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 4.3.15.)

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, if necessary, and then eliminated. (See Section 4.3.8.)

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 4.3.16.)

3.4.6 MAP ADDRESS WINDOW Directive (MAP$)

The MAP ADDRESS WINDOW directive maps an existing window to an attached region beginning at a specified offset from the start of the region, and going 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 4.3.30.)

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 4.3.51.)

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 4.3.47.)

3.4.9 RECEIVE BY REFERENCE Directive (RREF$)

The RECEIVE BY REFERENCE directive requests the Executive to select the next packet from the receive-by-reference queue of the issuing task, and 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 4.3.38.)

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 by a series of CREATE ADDRESS WINDOW directives. (See Section 4.3.24.)

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 4.3.26.)

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, and
- 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), and
- 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. Values assigned by the task to offsets within an RDB or a WDB 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 defined bits are:

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.
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 extendible.
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.

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 other bits are never modified by the Executive.

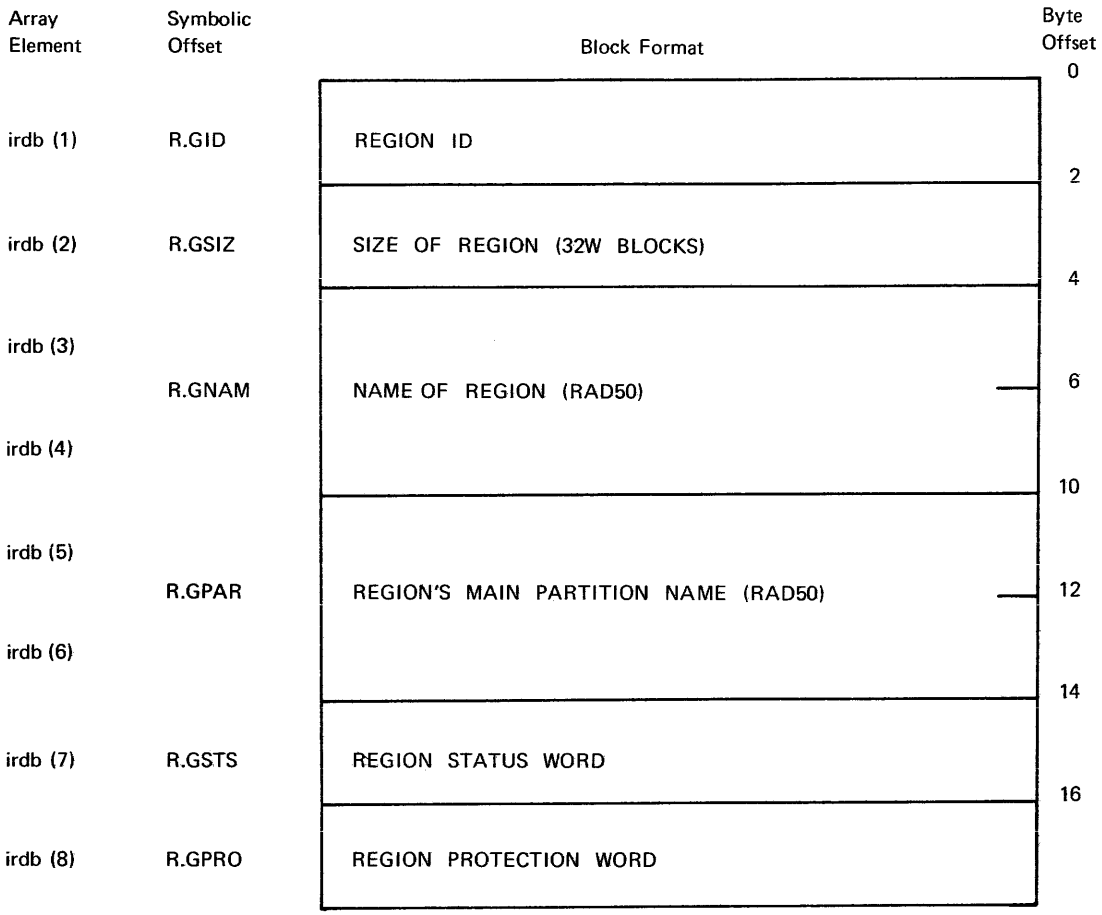


Figure 3-4 Region Definition Block

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$, the programmer need only specify RDBBK$ in a module that creates an RDB. The format of the call to RDBBK$ is:

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

where

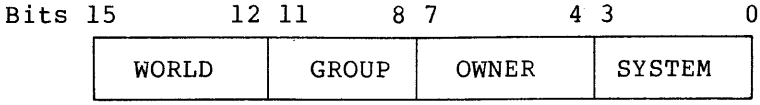
```
siz = the region size in 32-word blocks
nam = the region name (RAD50)
par = the name of the partition in which to create the region
      (RAD50)
sts = the region status word bit definitions
pro = the region's default protection word
```

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

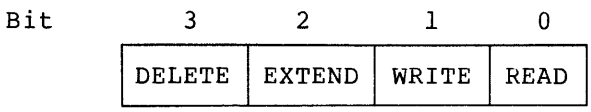
```
<bitl[!...!bitn]>
```

where bit is 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:



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



A bit value of zero (0) indicates that the specified type of access is to be allowed; a bit value of one (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 the macro call expanded above, 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 1110111000000000, grants all types of access to system and owner tasks (0000), and read access only, to group and world tasks (1110).

If the CREATE REGION directive is successful, the Executive will return to the issuing task a region ID value in the symbolic offset R.GID, and will set the defined bit RS.CRR in the status word R.GSTS.

3.5.1.2 **Using FORTRAN to Generate an RDB** - FORTRAN programmers 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 paragraph immediately below)
irdb(8)	Region protection code

The FORTRAN programmer modifies the region status word, irdb(7), by setting or clearing the appropriate bits. See the list above 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 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 offsets. One of the offsets 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 defined bits are:

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.64B=400	Defines the task's permitted alignment boundaries -- 0 for 256-word (512-byte) alignment, 1 for 32-word (64-byte) alignment.

Bit	Definition (Cont.)
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.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.

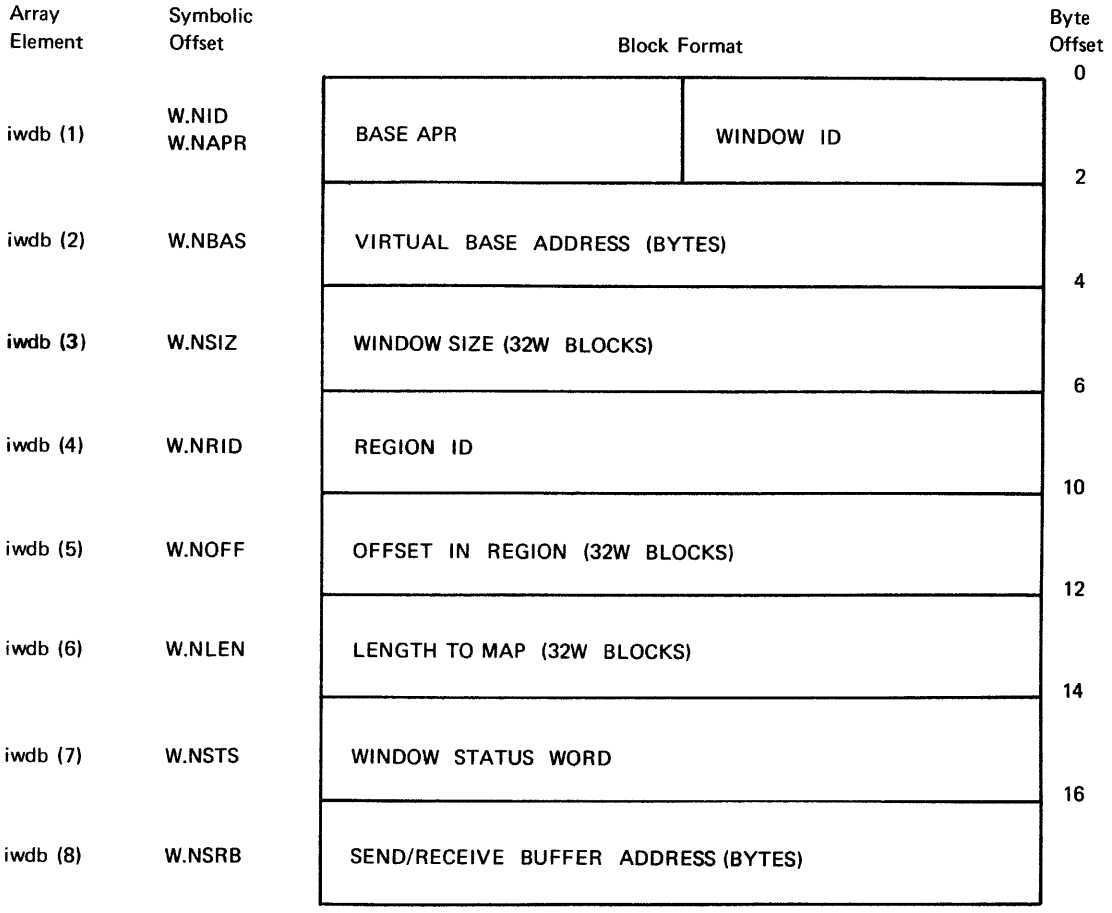


Figure 3-5 Window Definition Block

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. The Executive can then set any of these bits after completing the directive operation, to tell the task what actions were taken. The other bits are never modified by the Executive.

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$, the programmer need only specify WDBBK$ in a module that generates a WDB. The format of the call to WDBBK$ is:

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

where

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 region to be mapped, in 32-word blocks.
sts = the window status word bit definitions
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]>
```

where bit is 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 the length remaining in the region, whichever is smaller (because the macro call defaulted the len argument).
- Return values to the symbolic offsets 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** - FORTRAN programmers 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 paragraph immediately below)
iwdb(8)	Address of send/receive buffer

The FORTRAN programmer modifies the window status word, iwdb(7), by setting or clearing the appropriate bits. See the list above in Section 3.5.2 that describes the defined bits. The bit values are listed alongside the symbolic offsets.

Notes:

- 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.

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 4 describes which offsets and settings are relevant for each memory management directive. The values assigned by the task are called input parameters; 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 5 or 6 APRs to this mapping. A privileged task can issue memory management directives to remap any number of these APRs to regions. Programmers should take great care when using the directives in this way. 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.

CHAPTER 4

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 (e.g., 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, the DIR$ macro can be used by the programmer to execute a directive if the directive has a predefined DPB. See Sections 1.4.1.1 and 1.4.2 for further details.

4.1 DIRECTIVE CATEGORIES

For ease of reference, the directive descriptions are presented alphabetically in Section 4.3 according to the directive macro calls. This section, however, groups the directives by function and gives the number of the section that describes each directive in detail. The directives are grouped into the following seven categories:

1. Task Execution Control Directives
2. Task Status Control Directives
3. Informational Directives
4. Event-associated Directives
5. Trap-associated Directives
6. I/O and Intertask Communications Related Directives
7. Memory Management Directives

4.1.1 Task Execution Control Directives

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

Macro	Section	Directive Name
ABRT$	4.3.1	ABORT TASK
CSRQ$	4.3.11	CANCEL TIME BASED INITIATION REQUESTS
EXIT$S	4.3.20	TASK EXIT ($S form recommended)
EXTK$	4.3.21	EXTEND TASK
RQST$	4.3.37	REQUEST TASK
RSUM$	4.3.39	RESUME TASK
RUN$	4.3.40	RUN TASK
SPND$S	4.3.44	SUSPEND ($S form recommended)

4.1.2 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	Section	Directive Name
ALTP$	4.3.2	ALTER PRIORITY
DSCP$S	4.3.14	DISABLE CHECKPOINTING ($S form recommended)
ENCP$S	4.3.18	ENABLE CHECKPOINTING ($S form recommended)

4.1.3 Informational Directives

Several informational directives provide the issuing task with data retained by the system. These directives provide the time of day, the task parameters, the console switch settings, and partition or region parameters. The directives are:

Macro	Section	Directive Name
GPRT$	4.3.25	GET PARTITION PARAMETERS
GREG$	4.3.26	GET REGION PARAMETERS
GSSW$S	4.3.27	GET SENSE SWITCHES ($S form recommended)
GTIM$	4.3.28	GET TIME PARAMETERS
GTSK$	4.3.29	GET TASK PARAMETERS

4.1.4 Event-Associated Directives

The event and event flag directives are the means provided in the system for inter- and intra-task synchronization and signalling. These directives must be used carefully since software faults resulting from erroneous signalling and synchronization are often obscure and difficult to isolate. These directives are:

Macro	Section	Directive Name
CLEF$	4.3.7	CLEAR EVENT FLAG
CMKT$S	4.3.8	CANCEL MARK-TIME REQUESTS ($S form recommended)
DECL$S	4.3.12	DECLARE SIGNIFICANT EVENT ($S form recommended)
EXIF$	4.3.19	EXITIF
MRKT$	4.3.31	MARK TIME
RDAF$	4.3.36	READ ALL EVENT FLAGS
SETF$	4.3.42	SET EVENT FLAG
WSIG$S	4.3.52	WAIT FOR SIGNIFICANT EVENT ($S form recommended)
WTLO$	4.3.53	WAIT FOR LOGICAL "OR" OF EVENT FLAGS
WTSE$	4.3.54	WAIT FOR SINGLE EVENT FLAG

4.1.5 Trap-Associated Directives

The trap-associated directives provide the user with the same facilities inherent in the PDP-11 hardware trap system. They allow transfers of control (software interrupts) to the executing tasks. These directives are:

Macro	Section	Directive Name
ASTX$S	4.3.4	AST SERVICE EXIT ($S form recommended)
DSAR$S	4.3.13	DISABLE AST RECOGNITION ($S form recommended)
ENAR$S	4.3.17	ENABLE AST RECOGNITION ($S form recommended)
IHAR$S	4.3.13	INHIBIT AST RECOGNITION ($S form recommended)
SFPA$	4.3.43	SPECIFY FLOATING POINT PROCESSOR EXCEPTION AST
SPRA$	4.3.45	SPECIFY POWER RECOVERY AST
SRDA$	4.3.46	SPECIFY RECEIVE DATA AST
SRRA$	4.3.48	SPECIFY RECEIVE BY REFERENCE AST
SVDB$	4.3.49	SPECIFY SST VECTOR TABLE FOR DEBUGGING AID
SVTK$	4.3.50	SPECIFY SST VECTOR TABLE FOR TASK

4.1.6 I/O and Intertask Communications-Related Directives

The I/O and 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	Section	Directive Name
ALUN$	4.3.3	ASSIGN LUN
CINT$	4.3.6	CONNECT TO INTERRUPT VECTOR
GLUN$	4.3.22	GET LUN INFORMATION
GMCR$	4.3.23	GET MCR COMMAND LINE
QIO$	4.3.32	QUEUE I/O REQUEST
QIOW$	4.3.33	QUEUE I/O REQUEST AND WAIT
RCVD$	4.3.34	RECEIVE DATA
RCVX$	4.3.35	RECEIVE DATA OR EXIT
SDAT$	4.3.41	SEND DATA

4.1.7 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	Section	Directive Name
ATRG$	4.3.5	ATTACH REGION
CRAW$	4.3.9	CREATE ADDRESS WINDOW
CRRG$	4.3.10	CREATE REGION
DTRG$	4.3.15	DETACH REGION
ELAW$	4.3.16	ELIMINATE ADDRESS WINDOW
GMCX$	4.3.24	GET MAPPING CONTEXT
MAP$	4.3.30	MAP ADDRESS WINDOW
RREF$	4.3.38	RECEIVE BY REFERENCE
SREF$	4.3.47	SEND BY REFERENCE
UMAP$	4.3.51	UNMAP ADDRESS WINDOW

4.2 DIRECTIVE CONVENTIONS

Programmers using system directives should adhere to the following conventions:

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 1 to 6 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 2 characters long and are represented by one word in 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 RSX-11M compatibility with RSX-11D.

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 64(10). Numbers from 1 to 32(10) denote local flags. Numbers from 33 to 64 denote common flags.

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

4.3 SYSTEM DIRECTIVE DESCRIPTIONS

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

Name:

The function of the directive is described.

FORTRAN Call:

The FORTRAN subroutine call is shown, and each parameter is defined.

Macro Call:

The macro call is shown, each parameter is defined, and the defaults for optional parameters are given 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 are required for compatibility with RSX-11D.

Macro Expansion:

The $ form of the macro is expanded in most of the directive descriptions. Where the $S form is recommended for a directive, the expansion for that form is shown instead. Expansions for all three forms and for the DIR$ macro are illustrated in Section 1.4.5.

Definition Block Parameters:

These parameters are given only in the memory management directive descriptions. 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. These symbols are listed. 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 4 bytes.

DSW Return Code:

All valid return codes are listed.

Notes:

The notes presented with some directive descriptions expand on the function, use, and/or consequences of using the directives. Users should always read the notes carefully to ensure proper use of these directives.

ABRT$

4.3.1 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. A termination notification is displayed, 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 via a RUN$ directive or via an MCR RUN command that specifies one or more time parameters

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 itself).

FORTRAN Call:

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

  tsk  =  Task name
  ids  =  Directive status
```

Macro Call:

```
ABRT$   tsk

  tsk  =  Task name
```

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 is not installed
IE.ACT  --  Task is 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
```

Note:

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

    1. Detaches all attached devices
    2. Flushes the AST queue
    3. Flushes the receive and receive-by-reference queue
    4. Flushes the clock queue for outstanding Mark Time requests for the task
    5. Closes all open files (files open for write access are locked)
    6. Detaches all attached regions except in the case of a fixed task, where no detaching occurs
    7. Runs down the task's I/O
    8. Frees the task's memory if the aborted task was not fixed

ALTP$

4.3.2 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 task must be privileged to issue the ALTER PRIORITY directive.

FORTRAN Call:

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

  tsk  = Active task name
  ipri = 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 DPB out of the issuing task's address space
IE.SDP  --  DIC or DPB size is invalid
```

4.3.3 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 MCR ASN command in the RSX-11M Operator's Procedures Manual). The Executive first searches the logical assignment table for a device name match. If a match is found in the logical assignment table, the physical device unit associated with the matching entry is assigned to the logical unit. Otherwise, the Executive then 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 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

Note:

- 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.

4.3.4 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:

- 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.)

- 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
        .
        .
        QIO$C   IO.WVB,2,,,IOSB,ASTSER,<BUFFER,60.,40>
        .
        .                       ;PROCESS & WAIT
        .
        EXIT$S                  ;EXIT TO EXECUTIVE

;
; AST SERVICE ROUTINE
;

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

- The task can alter its return state 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 this alteration should be exercised with extreme caution since AST service routine bugs are difficult to isolate.

- 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.

4.3.5 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 desired access is granted. 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.

The directive can also be used 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. A programmer 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* in the region status word (specifying desired access to the region):

* FORTRAN programmers should refer to Section 3.5.1 to determine the bit values represented by the symbolic names described.

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:

Array Element	Offset	
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 -- The specified region name does not exist
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

4.3.6 CONNECT TO INTERRUPT VECTOR

The CONNECT TO INTERRUPT VECTOR directive provides for a task the capability of processing 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 is the execution of about 10 instructions before entry into the ISR, and 10 instructions after exit from the ISR. A mechanism is provided for transfer of control from the ISR to task-level code either via an asynchronous system trap (AST) or a local event flag.

After a task has connected to an interrupt vector, it has the capability of processing interrupts on three different levels:

- interrupt level
- fork level
- task level

The task level may be subdivided into:

- AST level
- non-AST level

1. Interrupt Level

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

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

2. Fork Level

   The fork level routine executes at priority 0 and therefore has more time to do further processing. 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 condenses the information further. 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, to be able to use the CINT$ directive, a task must be built privileged. However, it is legal to use the /PR:0 switch to the Task Builder to have "unprivileged mapping," i.e., 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
```

Argument descriptions:

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.

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,IADR,EDADR,PR5,ASTADR
.BYTE   129.,7.
.WORD   420
.WORD   BADR
.WORD   IADR
.WORD   EDADR
.BYTE   PR5,0
.WORD   ASTADR
```

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:

- 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.

- Mapping Considerations

In an unmapped system, the argument "base" is ignored, and the arguments "isr," "edir," and "ast" require no further explanation.

In a mapped system, however, it must be understood how the Executive maps the ISR and enable/disable interrupt routine when they are called. 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 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:

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

2. 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.

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

4. Use an offset of

   <120000-<base & 177700>>

   when accessing locations within the task image in immediate or absolute addressing mode.

- 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 via 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.

- 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 RSX-11M Guide to Writing an I/O Driver.)

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
```

- 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:

1. 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.

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

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

In case #1, the routine is called with the Carry bit cleared; in cases #2 and #3, 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 case common code is used for multiple vectors.

- AST Routine

  The fork routine may queue an AST for the task via 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).

- 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.

4.3.7 CLEAR EVENT FLAG

The CLEAR EVENT FLAG directive instructs the system to clear an indicated event flag and report the flag's polarity before clearing.

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>64 or EFN<1)
IE.ADP  --  Part of the DPB is out of the issuing task's
            address space
IE.SDP  --  DIC or DPB size is invalid
```

CMKT$S

4.3.8 CANCEL MARK TIME REQUESTS ($S form recommended)

The CANCEL MARK TIME REQUESTS directive instructs the system to cancel all MARK TIME requests that have been made by the issuing task.

FORTRAN Call:

```
CALL CANMT ([,ids])

  ids  =  Directive status
```

Macro Call:

```
CMKT$S  [,,err]

  err  =  Error routine address
```

Macro Expansion:

```
CMKT$S  ,,ERR           ;NOTE: THERE ARE TWO IGNORED ARGUMENTS
MOV     (PC)+,-(SP)     ;PUSH DPB ONTO THE STACK
.BYTE   27.,1           ;CMKT$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:

- 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.

4.3.9 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 Task Builder Reference Manual.) Any existing windows that overlap the specified range of virtual addresses are unmapped, if necessary, and then eliminated. 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 7, which is an index to the window block in the task's header. The window block describes the created address window.

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 software products. 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* in the window status word:

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

* FORTRAN programmers should refer to Section 3.5.2 to determine the bit values represented by the symbolic names described.

Output parameters:

Array Element	Offset	
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* in the window status word:

WS.CRW -- 1 if the address window was successfully created
WS.ELW -- 1 if any address windows were eliminated
WS.UNM -- 1 if any address windows were unmapped

Local Symbol Definitions:

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

DSW Return Codes:

IS.SUC -- Successful completion
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

* FORTRAN programmers should refer to Section 3.5.2 to determine the bit values represented by the symbolic names described.

CRRG$

4.3.10 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 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, the user can set RS.NDL in the region status word as an input parameter. Note that programmers should be careful in considering overriding 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.

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* in the region status word:
		RS.NDL -- 1 if the region should not be deleted on last detach
		RS.ATT -- 1 if created region should be attached
		RS.RED -- 1 if read access is desired on attach
		RS.WRT -- 1 if write access is desired on attach
		RS.EXT -- 1 if extend access is desired on attach
		RS.DEL -- 1 if delete access is desired on attach
irdb(8)	R.GPRO	-- Protection word for the region (DEWR,DEWR,DEWR,DEWR)

Output parameters:

Array Element	Offset	
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* in region status word:
		RS.CRR -- 1 if region was successfully created

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.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:

- The Executive does not return an error if the named region has already been created. 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.

---

* FORTRAN programmers should refer to Section 3.5.1 to determine the bit values represented by the symbolic names described.

- 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.

4.3.11 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 RUN command.

In a multiuser protection system, a task must be privileged to cancel time-based initiation requests for a task other than itself.

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 the programmer specifies an error routine address when using the $C or $S macro form, then a null argument must be included for RSX-11D compatibility. For example:

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

DECL$S

4.3.12 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 System Task Directory from the beginning, searching for the highest priority task that is ready to run. This directive should be used 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:

- 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.

DSAR$S or IHAR$S

4.3.13 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 will be effected when the task enables AST recognition. There is an implied AST disable recognition directive whenever an AST service routine is executing. When a task's execution is started, AST recognition is not disabled.

See Notes below.

FORTRAN Call:

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

ids = Directive status

Macro Call:

```
DSAR$S  [err]
   or
IHAR$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:

- Only the recognition of ASTs is disabled; the Executive still queues the ASTs. They are queued FIFO and will occur in that order when the task reenables AST recognition.
- 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.

- The FORTRAN calls, DSASTR (or INASTR) and ENASTR (see Section 4.3.17) 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. 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.

4.3.14 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. This directive can be issued only by the task that is to be affected. 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:

- When a checkpointable task's execution is started, checkpointing is not disabled (i.e., the task can be checkpointed).
- 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.

DTRG$

4.3.15 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* in the region status word:
		RS.MDL -- 1 if the region should be marked for deletion when the last task detaches from it

Output parameters:

Array Element	Offset	
irdb(7)	R.GSTS	-- Bit settings* in the region status word:
		RS.UNM -- 1 if any windows were unmapped

---

* FORTRAN programmers should refer to Section 3.5.1 to determine the bit values represented by the symbolic names described.

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.
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

ELAW$

4.3.16 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 = A window definition block composed of an 8-word integer
         array (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* in the window status word:
		WS.ELW -- 1 if the address window was successfully eliminated
		WS.UNM -- 1 if the address window was unmapped

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
```

* FORTRAN programmers should refer to Section 3.5.2 to determine the bit values represented by the symbolic names described.

4.3.17 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 have been 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:

- 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.

- The FORTRAN calls DSASTR (or INASTR) (see Section 4.3.13) 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 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.

ENCP$S

4.3.18 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.

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
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 $S form of the macro is recommended. It requires less space and executes with the same speed as the DIR$ macro.

4.3.19 EXITIF

The EXITIF 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>64 or EFN<1)
IE.ADP  --  Part of the DPB is out of the issuing task's
            address space
IE.SDP  --  DIC or DPB size is invalid
```

Notes:

- The EXITIF directive is useful in avoiding a possible race condition that can occur between two tasks communicating via 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 EXITIF specifying the same common event flag. If the event flag is set, the EXITIF directive will return control to the issuing task, signalling that something has been sent.

- A FORTRAN program that issues the EXITIF 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 EXITIF.

- On EXIT, the Executive frees task resources. In particular, the Executive:

  1. Detaches all attached devices
  2. Flushes the AST queue
  3. Flushes the receive and receive-by-reference queues
  4. Flushes the clock queue for any outstanding Mark Time requests for the task
  5. Closes all open files (files open for write access are locked)
  6. Detaches all attached tasks, except in the case of a fixed task in a system that supports the memory management directives
  7. Runs down the task's I/O
  8. Frees the task's memory if the exiting task was not fixed

- If the task exits, the Executive declares a significant event.

4.3.20 TASK EXIT ($S form recommended)

The TASK EXIT directive instructs the system to terminate the execution of the issuing task.

FORTRAN Call:

```
STOP
 or
CALL EXIT
```

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:

- 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.
- EXIT causes a significant event.
- On EXIT, the Executive frees task resources. In particular, the Executive:

  1. Detaches all attached devices
  2. Flushes the AST queue
  3. Flushes the receive and receive-by-reference queues
  4. Flushes the clock queue for any outstanding Mark Time requests for the task
  5. Closes all open files (files open for write access are locked)
  6. Detaches all attached regions, except in the case of a fixed task, where no detaching occurs

7. Runs down the task's I/O

8. Frees the task's memory if the exiting task was not fixed

- 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.

4.3.21 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, 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 it 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, 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 TASK directive:

- No task can extend itself beyond the maximum size set by the MCR command SET /MAXEXT or the size of the partition in which it is running. (See the RSX-11M Operator's Procedures 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.

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; or
the issuing task is not checkpointable and specified a positive increment; or
the issuing task has preallocated checkpoint space in its task image and has attempted to extend its size beyond its installed size; or
the issuing task had outstanding I/O when it issued the directive; or
task has memory-resident overlays and is attempting to reduce its size.

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 task tried to increase its size to the extent that one virtual address window would overlap another.

IE.ADP -- Part of the DPB is out of the issuing task's address space.

IE.SDP -- DIC or DPB size is invalid.

GLUN$

4.3.22 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  =  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:

WD. 00 -- Name of assigned device

WD. 01 -- 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.)

WD. 02 -- First device characteristics word:

```
Bit 0     --  Record-oriented device (1=yes) [FD.REC]*
Bit 1     --  Carriage-control device (1=yes)[FD.CCL]
Bit 2     --  Terminal device (1=Yes)[FD.TTY]
Bit 3     --  Directory (file-structured) device (1=yes)[FD.DIR]
Bit 4     --  Single directory device (1=yes)[FD.SDI]
Bit 5     --  Sequential device (1=yes)[FD.SQD]
Bit 6     --  Mass-bus device (1=yes)
Bit 7     --  User-mode diagnostics supported
Bit 8     --  Reserved
Bit 9     --  Unit software write locked (1=yes)
Bits 10-11    Reserved
Bit 12    --  Pseudo device (1=yes)
Bit 13    --  Device mountable as a communications channel
              (1=yes)
Bit 14    --  Device mountable as a Files-11 device (1=Yes)
Bit 15    --  Device mountable (1=yes)
```

WD. 03 -- Second device characteristics word

WD. 04 -- Third device characteristics word (Words 3 and 4 are device driver specific)

WD. 05 -- Standard device buffer size

* Bits with associated symbols have the symbols shown in square brackets. These symbols can be defined for use by a task via the FCSBT$ macro. See the IAS/RSX-11 I/O Operations Reference Manual.

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

4.3.23 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 "command line" being placed into an 80-byte command line buffer. (The MCR dispatcher is described in the RSX-11M Operator's Procedures Manual.)

FORTRAN Call:

```
CALL GETMCR (buf[,ids])

  buf  =  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:

- The GMCR$S form of the macro is not supplied, since the DPB receives the actual command line.

- The system processes all lines to:

  1. Convert tabs to a single space
  2. Convert multiple spaces to a single space
  3. Convert lower case to upper case
  4. Remove all trailing blanks

  The terminator (<CR> or <ESC>) is the last character in the line.

4.3.24 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 described by a series of CREATE ADDRESS WINDOW directives (see Section 4.3.9). 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 recreate 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 GCMX (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.

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 in each window definition block:

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* in the window status word (all 0 if the window is not mapped): WS.MAP -- 1 if the window is mapped WS.WRT -- 1 if the window is mapped with write access

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

* FORTRAN programmers should refer to Section 3.5.2 to determine the bit values represented by the symbolic names described.

4.3.25 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  =  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
```

The buffer has the following format:

WD. 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.

WD. 1 -- Partition size expressed as a multiple of 32 words.

WD. 2 -- Partition flags word. This word is returned equal to zero 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 returned address is physical. In mapped systems the returned address is virtual and always 0. 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:

- A variation of this directive exists for Executives that support the memory management directives. The variation is GET REGION PARAMETERS (see Section 4.3.26). 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.

- Omission of the partition-name argument returns parameters for the issuing task's unnamed subpartition, not for the system-controlled partition.

4.3.26 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 4.3.25) for Executives that support the memory management directives.

FORTRAN Call:

```
CALL GETREG ([rid],buf[,ids])

  rid = Region id
  buf = 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:

WD.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.

WD.1 -- Region size expressed as a multiple of 32-words.

WD.2 -- Region flags word. This word is returned equal to zero 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 always 0. 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

4.3.27 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:

- 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.

GTIM$

4.3.28 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:

FORTRAN provides several subroutines for obtaining the time in a number of formats. See the IAS/RSX-11M FORTRAN IV or the FORTRAN IV-PLUS User's Guide.

Macro Call:

```
GTIM$   buf
```

buf = Address of 8-word buffer

The buffer has the following format:

```
WD. 0  --  Year (since 1900)
WD. 1  --  Month (1-12)
WD. 2  --  Day (1-31)
WD. 3  --  Hour (0-23)
WD. 4  --  Minute (0-59)
WD. 5  --  Second (0-59)
WD. 6  --  Tick of second (depends on the frequency of the
           clock)
WD. 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
```

4.3.29 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  =  16-word integer array to receive the task parameters
  ids  =  Directive status
```

Macro Call:

```
GTSK$  buf

  buf  =  Address of a 16-word buffer
```

The buffer has the following format:

```
WD. 00  --  Issuing task's name in Radix-50 (first half)
WD. 01  --  Issuing task's name in Radix-50 (second half)
WD. 02  --  Partition name in Radix-50 (first half)
WD. 03  --  Partition name in Radix-50 (second half)
WD. 04  --  Undefined in RSX-11M -- This word exists
            for RSX-11D compatibility.
WD. 05  --  Undefined in RSX-11M -- This word exists
            for RSX-11D compatibility.
WD. 06  --  Run priority
WD. 07  --  User identification code (UIC) of issuing task
            (in a multiuser protection system, the task's default
            UIC)**
WD. 10  --  Number of logical I/O units (LUNs)
WD. 11  --  Undefined in RSX-11M -- This word exists for
            RSX-11D compatibility.
WD. 12  --  Undefined in RSX-11M -- This word exists for
            RSX-11D compatibility.
WD. 13  --  (Address of task SST vector tables)*
WD. 14  --  (Size of task SST vector table in words)*
WD. 15  --  Size (in bytes) either of task's address window
            0 in mapped systems, or of task's partition in
            unmapped systems (equivalent to partition size)
WD. 16  --  System on which task is running:
                0 for RSX-11D
                1 for RSX-11M
                2 for RSX-11S
                3 for IAS
WD. 17  --  Protection UIC (in a multiuser system, the log-in
            UIC)**
```

---

* 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.

** See note in RQST$ description (Section 4.3.37) on contents of words 07 and 17.

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)

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

4.3.30 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 assignment, a task can specify any length 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.

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 software products. 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 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* in the window status word: 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:

Array Element	Offset	
iwdb(6)	W.NLEN	-- Length of the area within the region actually mapped by the window
iwdb(7)	W.NSTS	-- Bit settings* in the window status word: WS.UNM -- 1 if the window was unmapped first

Local Symbol Definitions:

M.APBA -- Window definition block address (2)

---

* FORTRAN programmers should 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.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.ADP -- Part of the DPB or WDB is out of the issuing task's address space.
IE.SDP -- DIC or DPB size is invalid

MRKT$

4.3.31 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, WAITFOR 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 last Note below)
  tnt  =  Time interval unit (see last Note below)
  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 last Note below)
  tnt  =  Time interval unit (see last Note 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 (>64 or <0)
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 provides the following positive error codes to be returned for ISA calls:

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:

- MARK TIME requires dynamic memory for the clock queue entry.
- 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*
  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 4.3.4) is executed.
- If the directive is rejected, the specified event flag is not guaranteed to be cleared or set. Consequently, if the task indiscriminately executes a WAITFOR 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.
- If a task issues a MARK TIME directive that specifies a common event flag and then exits before the indicated time has elapsed, the event flag is not set.

---

* The event flag mask word preserves the WAITFOR conditions of a task prior to AST entry. A task can, after an AST, return to a WAITFOR state. Because these flags and the other stack data are in the user task, they can be modified. Such modification is strongly discouraged, since, if done inappropriately, the task may fault on obscure conditions.

- 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 = Milliseconds. The subroutine converts the specified magnitude to the equivalent number of system clock ticks.

  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).

4.3.32 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).

The device drivers declare 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 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 WAITFOR mask word, PS, PC, DSW (directive status), 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 I/O Drivers Reference Manual.

See Notes below.

FORTRAN Call:

```
CALL QIO (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  =  2-word integer array to receive final I/O status
prl  =  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:

```
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>
```

* I/O function code definitions are included in the RSX-11M I/O Drivers Reference Manual.

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.ILU  --  Invalid LUN
IE.IEF  --  Invalid event flag number (>64 or <0)
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:

- 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 4.3.4) is executed.

- If the directive is rejected, the specified event flag is not guaranteed to be cleared or set. Consequently, if the task indiscriminately executes a WAITFOR 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.

- Tasks cannot be checkpointed with QIO outstanding for two reasons:

    1. If the QIO directive results in a data transfer, the data transfers directly to or from the user-specified buffer.

    2. 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 the checkpointing of tasks during terminal input, the terminal driver checks for the following conditions when the driver dequeues an input request for a task:

    - That the task is checkpointable
    - That checkpointing is enabled
    - That the task is not executing an AST routine
    - That ASTs are enabled

  If the four conditions exist, the Executive immediately stops the task's execution. 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 its terminal input has completed. While the task is stopped, the terminal driver buffers the task's terminal input.

QIOW$

4.3.33 QUEUE I/O REQUEST AND WAIT

The QUEUE I/O REQUEST AND WAIT directive is identical to 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*
  lun = Logical unit number
  efn = Event flag number
  pri = Priority; ignored, but must be present
  isb = 2-word integer array to receive final I/O status
  prl = 6-word integer array containing device-dependent
        parameters to be placed in parameter words 1 through 6 of
        the directive parameter block (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 <P1,...P6>
```

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
```

* I/O function codes are defined in the RSX-11M I/O Drivers Reference Manual.

4.3.34 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 via 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.

In a system that supports multiuser protection, a task can be installed as a slave by the keyword /SLV=YES. (See the RSX-11M Operator's Procedures Manual.) When a slave task issues the RECEIVE DATA directive, it assumes the UIC and TI terminal of the task that sent the data.

FORTRAN Call:

```
CALL RECEIV (tsk,buf[,,ids])

  tsk  =  Sender task name
  buf  =  15-word integer array for received data
  ids  =  Directive status
```

Macro Call:

```
RCVD$   tsk,buf

  tsk  =  Sender task name
  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.ADP  --  Part of the DPB or buffer is out of the issuing
            task's address space
IE.SDP  --  DIC or DPB size is invalid
```

RCVX$

4.3.35 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 via 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.

In a system that supports multiuser protection, a task can be installed as a slave by the keyword /SLV=YES. (See the RSX-11M Operator's Procedures Manual.) When a slave task issues the RECEIVE DATA OR EXIT directive, it assumes the UIC and TI terminal of the task that sent the data.

See Notes below.

Fortran Call:

```
CALL RECOEX (tsk,buf[,,ids])

  tsk  =  Sender task name
  buf  =  15-word integer array for received data
  ids  =  Directive status
```

Macro Call:

```
RCVX$   tsk,buf

  tsk  =  Sender task name
  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.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:

- 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 was received, then the task can close all files and issue the call to RECOEX.

- If no data has 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 directive description).

- The RECEIVE DATA OR EXIT directive is useful in avoiding a possible race condition that can occur between two tasks communicating via 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 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.

- On EXIT, the Executive frees task resources. In particular, the Executive:

  1. Detaches all attached devices
  2. Flushes the AST queue
  3. Flushes the receive and receive-by-reference queues
  4. Flushes the clock queue for outstanding Mark Time requests for the task
  5. Closes all open files (files open for write access are locked)
  6. Detaches all attached regions except in the case of a fixed task, where no detaching occurs
  7. Runs down the task's I/O
  8. Frees the task's memory if the exiting task was not fixed

- If the task exits, the Executive declares a significant event.

RDAF$

4.3.36 READ ALL EVENT FLAGS

The READ ALL EVENT FLAGS directive instructs the system to read all 64 event flags for the issuing task and record their polarity in a 64-bit (4-word) buffer.

FORTRAN Call:

Only one event flag can be read by a FORTRAN task. 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:

```
RDAF$   buf
```

The buffer has the following format:

```
WD. 00  --  Task Local Flags 1-16
WD. 01  --  Task Local Flags 17-32
WD. 02  --  Task Common Flags 33-48
WD. 03  --  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
```

4.3.37 REQUEST

The REQUEST directive instructs the system to activate a task. The task is activated and subsequently runs contingent upon priority and memory availability. REQUEST is the basic mechanism used by running tasks for initiating other installed (dormant) tasks. REQUEST is a frequently used subset of the RUN directive.

See Notes below.

FORTRAN Call:

```
CALL REQUES (tsk,[opt][,ids])

  tsk  =  Task name
  opt  =  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.OSTN  --  Task name (4)
R.QSPN  --  Partition name (4)
R.OSPR  --  Priority (2)
R.OSGC  --  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:

- The requested task must be installed in the system.
- 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.
- Successful completion means that the task has been declared active, not that the task is actually running.
- The requested task acquires the same TI terminal assignment as that of the requesting task.
- The requested task always runs at the priority specified in its task header.
- A task that executes in a system-controlled partition requires dynamic memory for the partition control block used to describe its memory requirements.
- 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.
- 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$).

  1. 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.

  2. 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 to be 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 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$

4.3.38 RECEIVE BY REFERENCE

The RECEIVE BY REFERENCE directive causes 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 causes 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 (see Section 4.3.47), 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 4.3.30) 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* in the window status word: 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
iwdb(8)	W.NSRB	-- Optional address of a 10-word buffer, to contain the sender task name and additional information

Output parameters:

Array Element	Offset	
iwdb(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* in the window status word: 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.

Local Symbol Definitions:

R.REBA -- Window definition block address (2)

DSW Return Codes:

IS.SUC -- Successful completion
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

---

* FORTRAN programmers should refer to Section 3.5.2 to determine the bit values represented by the symbolic names described.

RSUM$

4.3.39 RESUME

The RESUME 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.PRI  --  Task not privileged (multiuser protection
            systems only)
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
```

4.3.40 RUN

The RUN 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, and

2. the system always sets the TI: (Terminal Input) 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  =  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.INS  --  Task is not installed
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 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 out of task's address space
100 --  DIC or DPB size invalid
```

Notes:

- In a multiuser protection system, a nonprivileged task cannot specify a UIC that is not equal to its own protection UIC. (See the last note in the description of the REQUEST directive for a definition of the protection UIC.) A privileged task can specify any UIC.

- 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.

- The target task must be installed in the system.

- 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.

- Successful completion means the task has been made active; it does not mean that the task is actually running.

- 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 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).

- 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.

- 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 Cancel command is issued.

- 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).

- If optional rescheduling is not desired, then the macro call should omit the arguments rmg and rnt.

4.3.41 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. 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  --  Insufficient dynamic memory
IE.IEF  --  Invalid event flag number (EFN.GT.64 or EFN.LT.0)
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:

- SEND DATA requires dynamic memory.
- 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.

SETF$

4.3.42 SET EVENT FLAG

The SET EVENT FLAG directive instructs the system to set an indicated event flag and report 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.GT.64 or EFN.LT.1)
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.

4.3.43 SPECIFY FLOATING POINT PROCESSOR EXCEPTION AST

The SPECIFY FLOATING POINT PROCESSOR EXCEPTION AST directive instructs the system to record either:

- that floating point processor exception ASTs for the issuing task are desired, and that the Executive is to transfer control to a specified address when such an AST occurs for the task, or
- that 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:

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:

```
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
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:

- SPECIFY FLOATING POINT PROCESSOR EXCEPTION AST requires dynamic memory.

- The Executive queues floating point processor exception ASTs when a floating point processor exception trap occurs for the task. No future floating point processor exception ASTs will be queued for the task until the first one queued has actually been effected.

- 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 4.3.4) directive is executed.

- This directive cannot be issued from an AST service routine or when ASTs are disabled.

- This directive applies only to the Floating Point Processor.

4.3.44 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 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:

- A suspended task retains control of the system resources allocated to it. The Executive makes no attempt to free these resources. When a task exits, the Executive frees the task's resources.
- A suspended task is eligible for checkpointing unless it is fixed or declared to be non-checkpointable.
- 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.

SPRA$

4.3.45 SPECIFY POWER RECOVERY AST

The SPECIFY POWER RECOVERY AST directive instructs the system to record either:

1. that power recovery ASTs for the issuing task are desired, and that control is to be transferred when a powerfail recovery AST occurs, or

2. that 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:

- SPECIFY POWER RECOVERY AST requires dynamic memory.

- 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 actually been effected.

- 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 4.3.4) can be executed with the stack in the same state as when the AST was entered.

- If a power recovery AST entry point is specified by a checkpointable task and the power fails while the task is checkpointed, the Executive does not effect or queue the AST. Therefore, when it is essential that a task be notified of a power failure, the task should disable checkpointing.

- This directive cannot be issued from an AST service routine or when ASTs are disabled.

SRDA$

4.3.46 SPECIFY RECEIVE DATA AST

The SPECIFY RECEIVE DATA AST directive instructs the system to record either:

- that receive data ASTs for the issuing task are desired, and that the Executive transfers control to a specified address when data has been placed in the task's receive queue, or
- that 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:

- SPECIFY RECEIVE DATA AST requires dynamic memory.
- 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 actually been effected.
- The task enters the receive 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 4.3.4) must be executed with the stack in the same state as when the AST was effected.
- This directive cannot be issued from an AST service routine or when ASTs are disabled.
- When a task is checkpointed back into memory, the Executive issues an AST for the task if its receive queue contains an entry. This practice prevents checkpointed tasks from losing receive ASTs.

SREF$

4.3.47 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 receiver task is automatically attached by the Executive, to the region referred to (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. 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. 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 permitted access can be defaulted 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 4.3.38). 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 =  The 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* in window status word (the receiver task's permitted access):
		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)
```

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 ACP
          (Ancillary Control Processor) 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
IE.ADP -- The address check of the DPB, the WDB, or the send
          buffer failed
IE.SDP -- DIC or DPB size is invalid
```

* FORTRAN programmers should refer to Section 3.5.2 to determine the bit values represented by the symbolic names described.

Note:

- 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.

4.3.48 SPECIFY RECEIVE-BY-REFERENCE AST

The SPECIFY RECEIVE-BY-REFERENCE AST directive instructs the system to record either:

- that receive-by-reference ASTs for the issuing task are desired, and that the Executive transfers control to a specified address when such an AST occurs, or
- that 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:

- SPECIFY RECEIVE-BY-REFERENCE AST requires dynamic memory.

- 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 actually been effected.

- 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 4.3.4) must be executed with the stack in the same state as when the AST was effected.

- This directive cannot be issued from an AST service routine or when ASTs are disabled.

- When a task is checkpointed back into memory, the Executive issues an AST for the task if its receive-by-reference queue contains one or more entries. This practice prevents checkpointed tasks from losing receive-by-reference ASTs.

4.3.49 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 intra-task 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:

```
WD.  00  --  Odd address or nonexistent memory error
WD.  01  --  Memory protect violation
WD.  02  --  T-bit trap or execution of a BPT instruction
WD.  03  --  Execution of an IOT instruction
WD.  04  --  Execution of a reserved instruction
WD.  05  --  Execution of a non-RSX EMT instruction
WD.  06  --  Execution of a TRAP instruction
WD.  07  --  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$

4.3.50 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 mechanisms; 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:

```
WD.00  --  Odd address or nonexistent memory error
WD.01  --  Memory protect violation
WD.02  --  T-bit trap or execution of a BPT instruction
WD.03  --  Execution of an IOT instruction
WD.04  --  Execution of a reserved instruction
WD.05  --  Execution of a non-RSX EMT instruction
WD.06  --  Execution of a TRAP instruction
WD.07  --  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
```

4.3.51 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:

Array Element	Offset	
iwdb(7)	W.NSTS	-- Bit settings* in the window status word: WS.UNM -- 1 if the window was successfully unmapped

Local Symbol Definitions:

```
U.MABA -- Window definition block address (2)
```

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
```

---

* FORTRAN programmers should refer to Section 3.5.2 to determine the bit values represented by the symbolic names described.

WSIG$S

4.3.52 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:

- 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.
- 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.
- 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.

- Significant events include the following:

  1. I/O completion
  2. Task exit
  3. The execution of a SEND DATA directive
  4. The execution of a SEND BY REFERENCE or a RECEIVE BY REFERENCE directive
  5. The execution of an ALTER PRIORITY directive
  6. The removal of an entry from the clock queue (e.g., resulting from the execution of a MARK TIME directive or the issuance of a rescheduling request)
  7. The execution of a DECLARE SIGNIFICANT EVENT directive
  8. The execution of the round-robin scheduling algorithm at the end of a round-robin scheduling interval

WTLO$

4.3.53 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
```

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
            set indicator other than 0, 1, 2, or 3
IE.ADP  --  Part of the DPB is out of the issuing task's
            address space
IE.SDP  --  DIC or DPB size is invalid
```

Notes:

- 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.

- The Executive does not arbitrarily clear event flags when WAITFOR 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.

- 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

- 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.

WTSE$

4.3.54 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>64 or EFN<1)
IE.ADP  --  Part of the  DPB is out of the issuing task's
            address space
IE.SDP  --  DIC or DPB size is invalid
```

APPENDIX A

DIRECTIVE SUMMARY - ALPHABETICAL ORDER BY MACRO CALL

ABORT TASK — ABRT$

FORTRAN Call:

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

```
tsk  =  Task name
ids  =  Directive status
```

Macro Call:

```
ABRT$   tsk
```

```
tsk  =  Task name
```

ALTER PRIORITY — ALTP$

FORTRAN Call:

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

```
tsk  =  Active task name
ipri =  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,rsw,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

psw = low-order byte of the Processor Status word to be loaded before entering the ISR

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 ($S form recommended) **CMKT$S**

FORTRAN Call:

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

ids = Directive status

Macro Call:

```
CMKT$S  [,,err]
```

err = Error routine address

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 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
```

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
```

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])
```

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

ENABLE AST RECOGNITION ($S form recommended) **ENAR$S**

FORTRAN Call:

```
CALL ENASTR
```

Macro Call:

```
ENAR$S  [err]
```

err = Error routine address

ENABLE CHECKPOINTING ($S form recommended) **ENCP$S**

FORTRAN Call:

```
CALL ENACKP
```

Macro Call:

```
ENCP$S  [err]
```

err = Error routine address

EXITIF **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:

```
STOP
 or
CALL EXIT
```

Macro Call:

```
EXIT$S  [err]
```

err = Error routine address

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 LUN INFORMATION **GLUN$**

FORTRAN Call:

```
CALL GETLUN (lun,dat[,ids])
```

lun = Logical unit number
dat = 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 = 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.
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 a 3-word buffer
```

GET REGION PARAMETERS **GREG$**

FORTRAN Call:

```
CALL GETREG ([rid],buf[,ids])

  rid  =  Region id
  buf  =  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
```

GET SENSE SWITCHES ($S form recommended) **GSSW$S**

FORTRAN Call:

```
CALL READSW (isw)

  isw  =  Integer to receive the console switch settings
```

Macro Call:

```
GSSW$S  [err]

  err  =  Error routine address
```

GET TIME PARAMETERS **GTIM$**

FORTRAN Call:

FORTRAN provides several subroutines for obtaining the time in a number of formats. See the IAS/RSX-11 FORTRAN-IV User's Guide or the FORTRAN IV-PLUS User's Guide.

Macro Call:

```
GTIM$   buf
```

buf = Address of 8-word buffer

GET TASK PARAMETERS **GTSK$**

FORTRAN Call:

```
CALL GETTSK (buf[,ids])
```

buf = 16-word integer array to receive the task parameters
ids = Directive status

Macro Call:

```
GTSK$   buf
```

buf = Address of a 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
```

QUEUE I/O REQUEST QIO$

FORTRAN Call:

```
CALL QIO (fnc,lun,[efn],[pri],[isb],[prl][,ids])

  fun  =  I/O function code
  lun  =  Logical unit number
  efn  =  Flag number
  pri  =  Priority; ignored, but must be present
  isb  =  2-word integer array to receive final I/O status
  prl  =  6-word integer array containing device-dependent
          parameters to be placed in parameter words 1 through 6
          of the directive parameter block (DPB).
  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  =  2-word integer array to receive final I/O status
  prl  =  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 <P1,...P6>
```

RECEIVE DATA **RCVD$**

FORTRAN Call:

```
CALL RECEIV (tsk,buf[,ids])

  tsk  =  Sender task name
  buf  =  15-word integer array for received data
  ids  =  Directive status
```

Macro Call:

```
RCVD$  tsk,buf

  tsk  =  Sender task name
  buf  =  Address of 15-word buffer
```

RECEIVE DATA OR EXIT **RCVX$**

FORTRAN Call:

```
CALL RECOEX (tsk,buf[,ids])

  tsk  =  Sender task name
  buf  =  15-word integer array for received data
  ids  =  Directive status
```

Macro Call:

```
RCVX$ tsk,buf

  tsk  =  Sender task name
  buf  =  Address of 15-word buffer
```

READ ALL EVENT FLAGS **RDAF$**

FORTRAN Call:

Only a single event flag can be read by a FORTRAN task. The call is:

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

  efn  =  Event flag number
  ids  =  Directive status
```

Macro Call:

```
RDAF$  buf

  buf  =  Address of 4-word buffer
```

REQUEST **RQST$**

FORTRAN Call:

```
CALL REQUES (tsk,[opt][,ids])

  tsk  =  Task name
  opt  =  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 **RSUM$**

FORTRAN Call:

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

  tsk  =  Task name
  ids  =  Directive status
```

Macro Call:

```
RSUM$  tsk

  tsk  =  Task name
```

RUN **RUN$**

FORTRAN Call:

```
CALL RUN (tsk,[opt],[smg],snt,[rmg],[rnt][,ids])

  tsk  =  Task name
  opt  =  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
```

SEND DATA **SDAT$**

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
```

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 **SFPA$**

FORTRAN Call:

Not supported.

Macro Call:

```
SFPA$   [ast]

  ast  =  AST service routine entry point address
```

SUSPEND ($S form recommended) **SPND$S**

FORTRAN Call:

```
CALL SUSPND
```

Macro Call:

```
SPND$S  [err]

  err  =  Error routine address
```

SPECIFY POWER RECOVERY AST **SPRA$**

FORTRAN Call:

```
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.

Macro Call:

```
SPRA$  [ast]
```

ast = AST service routine entry point address

SPECIFY RECEIVE DATA AST **SRDA$**

FORTRAN Call:

Not supported.

Macro Call:

```
SRDA$   [ast]
```

ast = AST service routine entry point address

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
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

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

UNMAP ADDRESS WINDOW **UMAP$**

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

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 is 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 Executive. To include these in a MACRO-11 program, the programmer uses 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.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.CKP  -10.    ISSUING TASK NOT CHECKPOINTABLE
        IE.PRI  -16.    PRIVILEGE VIOLATION
        IE.RSU  -17.    SPECIFIED VECTOR ALREADY IN USE
        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.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.LNL  -90.    LUN LOCKED IN USE
        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
```

RSX-11M
Executive Reference Manual
Order No. AA-2544D-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. Problems with software should be reported on a Software Performance Report (SPR) form. If you require a written reply and are eligible to receive one under SPR service, submit your comments on an SPR form.

Did you find errors in this manual? If so, specify by page.

_____

_____

_____

_____

_____

_____

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

_____

_____

_____

_____

_____

_____

Is there sufficient documentation on associated system programs required for use of the software described in this manual? If not, what material is missing and where should it be placed?

_____

_____

_____

_____

_____

_____

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
- ☐ Non-programmer interested in computer concepts and capabilities

Name _____ Date _____

Organization _____

Street _____

City _____ State _____ Zip Code or Country _____

Please cut along this line.

---------- Fold Here ----------

---------- Do Not Tear - Fold Here and Staple ----------

FIRST CLASS
PERMIT NO. 33
MAYNARD, MASS.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

Postage will be paid by:

digital

Software Documentation
146 Main Street ML5-5/E39
Maynard, Massachusetts 01754

digital

digital equipment corporation

Printed in U.S.A.