Table 4-1 (cont.)
Summary of SYSF4 Subprograms

Function Call	Section	Purpose
RT-11 Services		
CHAIN	4.3.2	Chains to another program (in the background job only).
*DEVICE	4.3.6	Specifies actions to be taken on normal or abnormal program termination, such as turning off interrupt enable on foreign devices, etc.
GTJB	4.3.10	Returns the parameters of this job.
IDSTAT	4.3.22	Returns the status of the specified device.
IFETCH	4.3.29	Loads a device handler into memory.
IQSET	4.3.37	Expands the size of the RT-11 monitor queue from the free space managed by the FORTRAN system.
ISPFN ISPFNC ISPFNF ISPFNW	4.3.47	Issues special function requests to various handlers, such as magtape. The four modes correspond to the IWRITE, IWRITC, IWRITF, the IWRITW modes.
*ITLOCK	4.3.50	Indicates whether the USR is currently in use by another job and performs a LOCK if the USR is available.
LOCK	4.3.69	Makes the RT-11 monitor User Service Routine (USR) permanently resident until an UNLOCK function is executed. A portion of the user's program is swapped out to make room for the USR if necessary.
RCHAIN	4.3.86	Allows a program to access variables passed across a chain.
RCTRLO	4.3.87	Enables output to the terminal by cancelling the effect of a previously typed CTRL/O, if any.
*RESUME	4.3.89	Causes the main program execution of a job to resume where it was suspended by a SUSPND function call.
SCCA	4.3.90	Intercepts a CTRL/C command initiated at the console terminal.

* FB and XM monitors only.

(continued on next page)

Table 4-1 (cont.)
Summary of SYSF4 Subprograms

Function Call	Section	Purpose
RT-11 Services (cont.)		
SETCMD	4.3.99	Passes command lines to the keyboard monitor to be executed after the program exits.
*SUSPND	4.3.97	Suspends main program execution of the running job; completion routines continue to execute.
UNLOCK	4.3.102	Releases the USR if a LOCK was performed; the user program is swapped in if required.
INTEGER*4 Support Functions		
AJFLT	4.3.1	Converts a specified INTEGER*4 value to REAL*4 and returns the result as the function value.
DJFLT	4.3.7	Converts a specified INTEGER*4 value to REAL*8 and returns the result as the function value.
IAJFLT	4.3.13	Converts a specified INTEGER*4 value to REAL*4 and stores the result.
IDJFLT	4.3.21	Converts a specified INTEGER*4 value to REAL*8 and stores the result.
IJCVT	4.3.28	Converts a specified INTEGER*4 value to INTEGER*2.
JADD	4.3.57	Computes the sum of two INTEGER*4 values.
JAFIX	4.3.58	Converts a REAL*4 value to INTEGER*4.
JCMP	4.3.59	Compares two INTEGER*4 values and returns an INTEGER*2 value that reflects the signed comparison result.
JDFIX	4.3.60	Converts a REAL*8 value to INTEGER*4.
JDIV	4.3.61	Computes the quotient and remainder of two INTEGER*4 values.
JICVT	4.3.62	Converts an INTEGER*2 value to INTEGER*4.
JJCVT	4.3.63	Converts the two-word internal time format to INTEGER*4 format, and vice versa.

* FB and XM monitors only.

(continued on next page)

Table 4-1 (Cont.)
Summary of SYSF4 Sbuprograms

Function Call	Section	Purpose
INTEGER*4 Support Functions (cont.)		
JMOV	4.3.64	Assigns an INTEGER*4 value to a variable.
JMUL	4.3.65	Computes the product of two INTEGER*4 values.
JSUB	4.3.66	Computes the difference between two INTEGER*4 values.
Character String Functions		
CONCAT	4.3.4	Concatenates two variable-length strings.
GETSTR	4.3.8	Reads a character string from a specified FORTRAN logical unit.
INDEX	4.3.30	Returns the location in one string of the first occurrence of another string .
INSERT	4.3.31	Replaces a portion of one string with another string.
ISCOMP	4.3.91	Compares two character strings.
IVERIF	4.3.103	Indicates whether characters in one string appear in another.
LEN	4.3.68	Returns the number of characters in a specified string.
PUTSTR	4.3.83	Writes a variable-length character string on a specified FORTRAN logical unit.
REPEAT	4.3.88	Concatenates a specified string with itself to provide an indicated number of copies and stores the resultant string.
SCOMP	4.3.91	Compares two character strings.
SCOPY	4.3.92	Copies a character string from one array to another.
STRPAD	4.3.95	Pads a variable-length string on the right with blanks to create a new string of a specified length.
SUBSTR	4.3.96	Copies a substring from a specified string.
TRANSL	4.3.100	Replaces one string with another after performing character modification.

* FB and XM monitors only.

(continued on next page)

Table 4-1 (cont.)
Summary of SYSF4 Subprograms

Function Call	Section	Purpose
Character String Functions (cont.)		
TRIM	4.3.101	Removes trailing blanks from a character string.
VERIFY	4.3.103	Indicates whether characters in one string appear in another.
Radix-50 Conversion Operations		
IRAD50	4.3.38	Converts ASCII characters to Radix-50, returning the number of characters converted.
R50ASC	4.3.84	Converts Radix-50 characters to ASCII.
RAD50	4.3.85	Converts six ASCII characters, returning a REAL*4 result that is the 2-word Radix-50 value.
Miscellaneous Services		
IADDR	4.3.12	Obtains the memory address of a specified entity.
IGETSP	4.3.27	Returns the address and size (in words) of free space obtained from the FORTRAN system.
INTSET	4.3.32	Establishes a specified FORTRAN subroutine as an interrupt service routine at a specified priority.
IPEEK	4.3.33	Returns the value of a word located at a specified absolute memory address.
IPEEKB	4.3.34	Returns the value of a byte located at a specified byte address.
IPOKE	4.3.35	Stores an integer value in an absolute memory location.
IPOKEB	4.3.36	Stores an integer value in a specified byte location.
ISPY	4.3.48	Returns the integer value of the word located at a specified offset from the beginning of the RT-11 resident monitor.

* FB and XM monitors only.

Routines requiring the USR (see Section 2.3) differ between the SJ and FB monitors. (The USR is always resident in the XM monitor.) The following functions require the use of the USR:

```
CLOSEC
GETSTR (only if first I/O operation on logical unit)
ICDFN  (single job only)
GETLIN
ICSI
IDELET
IDSTAT
IENTER
IFETCH
IQSET
IRENAM
ITLOCK (only if USR is not in use by the other job)
LOCK   (only if USR is in a swapping state)
LOOKUP
PUTSTR (only if first I/O operation on logical unit)
```

Certain requests require a queue element taken from the same list as the I/O queue elements. These are:

```
IRCVD/IRCVDC/IRCVDF/IRCVDW
IREAD/IREADC/IREADF/IREADW
ISCHED
ISDAT/ISDATC/ISDATF/ISDATW
ISLEEP
ISPFN/ISPFNC/ISPFNF/ISPFNW
ITIMER
ITWAIT
IUNTIL
IWRITC/IWRITE/IWRITF/IWRITW
MRKT
MWAIT
```

4.2.1 Completion Routines

Completion routines are subprograms that execute asynchronously with a main program. A completion routine is scheduled to run as soon as possible after the event for which it has been waiting has completed (such as the completion of an I/O transfer, or the lapsing of a specified time interval). All completion routines of the current job have higher priority than other parts of the job; therefore, once a completion routine becomes runnable because of its associated event, it interrupts execution of the job and continues to execute until it relinquishes control. See Figure 1-2, in Chapter 1.

Completion routines are handled differently in the SJ and the FB monitors. In SJ, completion routines are totally asynchronous and can interrupt one another. In FB (and XM), completion routines do not interrupt each other but are queued and made to wait until the correct job is running. (For further information on completion routines, see Sections 2.2.8 and 4.1.4).

A FORTRAN completion routine can have a maximum of two arguments:

```
SUBROUTINE crtn [(iarg1,iarg2)]
```

where: iarg1 is equivalent to R0 on entry to an assembly language completion routine.

iarg2	is equivalent to R1 on entry to an assembly language completion routine.

If an error occurs in a completion routine or in a subroutine at completion level, the error handler traces back normally through to the original interruption of the main program. Thus the traceback is shown as though the completion routine were called from the main program and lets the user know where the main program was executing if a fatal error occurs.

Certain restrictions apply to completion routines (those routines that are activated by the following calls:)

INTSET
IRCVDC
IRCVDF
IREADC
IREADF
ISCHED
ISDATC
ISDATF
ISPFNC
ISPFNF
ITIMER
IWRITC
IWRITF
MRKT

These restrictions are:

1. The first subroutine call that references a FORTRAN completion routine must be issued from the main program.

2. No channels can be allocated (by calls to IGETC) or freed (by calls to IFREEC) from a completion routine. Channels to be used by completion routines should be allocated and placed in a COMMON block for use by the routine.

3. The completion routine cannot perform any call that requires the use of the USR, such as LOOKUP and IENTER. See Section 4.2 for a list of SYSF4 functions that call the USR.

4. Files to be operated upon in completion routines must be opened and closed by the main program. There are, however, no restrictions on the input or output operations that can be performed in the completion routine. If many files must be made available to the completion routine, they can be opened by the main program and saved for later use (without tying up RT-11 channels) by the ISAVES call. The completion routine can later make them available by reattaching the file to a channel with an IREOPN call.

5. FORTRAN subprograms are reusable but not reentrant. A given subprogram can be used many times as a completion routine or as a routine in the main program, but a subprogram executing as main program code does not work properly if it is interrupted at the completion level. This restriction applies to all subprograms that can be invoked at the completion level and can be active at the same time in the main program.

6. Only one completion function should be active at any time under the single-job monitor (see Section 4.1.4).

7. Assembly language completion routines must be exited via an RTS PC.

8. FORTRAN completion routines must be exited by execution of a RETURN or END statement in the subroutine.

4.2.2 Channel-Oriented Operations

An RT-11 channel being used for input/output with SYSF4 must be allocated in one of the following two ways:

1. The channel is allocated and marked in use to the FORTRAN I/O system by a call to IGETC and is later freed by a call to IFREEC.

2. An ICDFN call is issued to define more channels (up to 256). All channels numbered greater than 17 (octal) can be freely used by the programmer; the FORTRAN I/O system uses only channels 0 through 17 (octal).

Channels must be allocated in the main program routine or its subprograms, not in routines that are activated as the result of I/O completion events or ISCHED or ITIMER calls.

4.2.3 INTEGER*4 Support Functions

INTEGER*4 variables are allocated two words of storage. INTEGER*4 values are stored in two's complement representation. The first word (lower address) contains the low-order part of the value, and the second word (higher address) contains the sign and the high-order part of the value. The range of numbers supported is -2^31+1 to 2^31-1.

Note that this format differs from the 2-word internal time format that stores the high-order part of the value in the first word and the low-order part in the second. The JJCVT function (Section 4.3.63) is provided for conversion between the two internal formats.

Integer and real arguments to subprograms are indicated in the following manner in this chapter.

i =	INTEGER*2 arguments
j =	INTEGER*4 arguments
a =	REAL*4 arguments
d =	REAL*8 arguments

When the DATA statement is used to initialize INTEGER*4 variables, it must specify both the low- and high-order parts. The following example only initializes the first word.

```
INTEGER*4 J
DATA J/3/
```

The correct way to initialize an INTEGER*4 variable to a constant (such as, 3) is shown below:

```
INTEGER*4 J
INTEGER*2 I(2)
EQUIVALENCE (J,I)
DATA I/3,0/       !INITIALIZE J TO 3
```

If initializing an INTEGER*4 variable to a negative value (such as -4), the high-order (second word) part must be the continuation of the two's complement of the low-order part. For example:

```
        INTEGER*4 J
        INTEGER*2 I(2)
        EQUIVALENCE (J,I)
        DATA I/-4,-1/      !INITIALIZE J TO -4
```

The following form is suitable for INTEGER*4 arguments to subprograms:

```
        INTEGER*2 J(2)
        DATA J/3,0/        !LOW-ORDER,HIGH-ORDER
```

4.2.4 Character String Functions

The SYSF4 character string functions and routines provide variable-length string support for RT-11 FORTRAN. SYSF4 calls are provided to perform the following character string operations:

- Read character strings from a specified FORTRAN logical unit (GETSTR).
- Write character strings to a specified FORTRAN logical unit (PUTSTR).
- Concatenate variable-length strings (CONCAT).
- Return the position of one string in another (INDEX).
- Insert one string into another (INSERT).
- Return the length of a string (LEN).
- Repeat a character string (REPEAT).
- Compare two strings (SCOMP).
- Copy a character string (SCOPY).
- Pad a string with rightmost blanks (STRPAD).
- Copy a substring from a string (SUBSTR).
- Perform character modification (TRANSL).
- Remove trailing blanks (TRIM).
- Verify the presence of characters in a string (VERIFY).

Strings are stored in LOGICAL*1 arrays that are defined and dimensioned by the FORTRAN programmer. Strings are stored in these arrays as one character per array element plus a zero element to indicate the current end of the string (ASCIZ format).

The length of a string can vary at execution time, ranging from zero characters in length to one less than the size of the array that stores the string. The maximum size of any string is 32767 characters. Strings can contain any of the 7-bit ASCII characters except null (0), since the null character is used to mark the end of the string. Bit 7 of each character must be cleared (0); therefore, the valid characters are those whose decimal representations range from 1 to 127, inclusive.

The ASCII code used in this string package is the same as that employed by FORTRAN for A-type FORMAT items, ENCODE/DECODE strings, and object-time FORMAT strings. ASCIZ strings in the form used by these routines are generated by the FORTRAN compiler whenever quoted strings are used as arguments in the CALL statement. Note that a null string (a string containing no characters) can be represented in FORTRAN by a variable or constant of any type that contains the value zero, or by a LOGICAL variable or constant with the .FALSE. value.

The SYSF4 user should ensure that a string never overflows the array that contains it by being aware of the length of the string result produced by each routine. In many routines where the resultant string length can vary or is difficult to determine, an optional integer argument can be specified to the subroutine to limit the length. In the sections describing the character string routines, this argument is called len. The length of an output string is limited to the value specified for len plus one (for the null terminator); therefore the array receiving the result must be at least len plus one elements in size.

The optional argument err can be included when len is specified. Err is a logical variable that should be initialized by the FORTRAN program to the .FALSE. value. If a string function is given the arguments len and err, and len is actually used to limit the length of the string result, then err is set to the .TRUE. value. If len is not used to truncate the string, err is unchanged; that is, it remains .FALSE..

Arguments len and err are normally optional arguments. The argument len can appear alone; however, len must appear if err is specified. The err argument should be used for GETSTR and PUTSTR.

Several routines use the concept of character position. Each character in a string is assigned a position number that is one greater than the position of the character immediately to its left. The first character in a string is in position one.

4.2.4.1 Allocating Character String Variables

- A one-dimensional LOGICAL*1 array can be used to contain a single string whose length can vary from zero characters to one fewer than the dimensioned length of the array. For example:

```
     LOGICAL*1 A(45)      !ALLOCATE SPACE FOR STRING VARIABLE A
```

The preceding example allows array A to be used as a string variable that can contain a string of 44 or fewer characters. Similarly, a two-dimensional LOGICAL*1 array can be used to contain a one-dimensional array of strings. Each string in the array can have a length up to one less than the first dimension of the LOGICAL*1 array. There can be as many strings as the number specified for the second dimension of the LOGICAL*1 array. For example:

```
     LOGICAL*1 W(21,10)      !ALLOCATE AN ARRAY OF STRINGS
```

The preceding example creates a string array W that has ten string elements, each of which can contain up to 20 characters. String I in array W is referenced in subroutine or function calls as W(1,I).

A two-dimensional string array can be allocated. For example:

```
     LOGICAL*1 T(14,5,7)      !ALLOCATE A 5 BY 7 STRING ARRAY
```

In the preceding example, each string in array T can vary in length to a maximum of 13 characters. String I,J of the array can be referenced as T(1,I,J). Note that T is the same as T(1,1,1). This dimensioning process can be continued to create string arrays of up to six dimensions (represented by LOGICAL*1 arrays of up to seven dimensions).

4.2.4.2 **Passing Strings to Subprograms** - The LOGICAL*1 arrays that contain strings can be placed in a COMMON block and referenced by any or all routines with a similar COMMON declaration. However, care should be taken when a LOGICAL*1 array is placed in a COMMON block, for if such an array has an odd length, it causes all succeeding variables in the COMMON block to be assigned odd addresses.

A LOGICAL*1 array has an odd length only if the product of its dimensions is odd. For example:

```
     LOGICAL*1 B(10,7)       !(10*7)=70; EVEN LENGTH
     LOGICAL*1 H(21)         !21 IS ODD; ODD LENGTH
```

If odd length arrays are to be placed in a COMMON block, they should either be placed at the end of the block or they should be paired to result in an effective even length. For example:

```
     COMMON A1,A2,A3(10),H(21)          !PLACE ODD-SIZED ARRAY AT END

          or

     COMMON A1,A2,H(21),H1(7),A3(10)   !PAIR ODD-SIZED ARRAYS H AND H1
```

Note that these cautions apply only to LOGICAL*1 variables and arrays.

The second method of passing strings to subprograms is through arguments and formal parameters. A single string can be passed by using its array name as an argument. For example:

```
     LOGICAL*1 A(21)      !STRING VARIABLE "A", 20 CHARACTERS MAXIMUM
     CALL SUBR(A)         !PASS STRING A TO SUBROUTINE SUBR
```

If the maximum length of a string argument is unknown in a subroutine or function, or if the routine is used to handle many different length strings, the dummy argument in the routine should be declared as a LOGICAL*1 array with a dimension of one, such as LOGICAL*1 ARG(1). In this case, the string routines correctly determine the length of ARG whenever it is used, but it is not possible to determine the maximum size string that can be stored in ARG. If a multi-dimensional array of strings is passed to a routine, it must be declared in the called program with the same dimensions as were specified in the calling program.

NOTE

> The length argument specified in many of the character string functions refers to the maximum length of the string excluding the necessary null byte terminator. The length of the LOGICAL*1 array to receive the string must be at least one greater than the length argument.

4.2.4.3 **Using Quoted-String Literals** - Quoted-strings can be used as arguments to any of the string routines that are invoked by functions or the CALL statement. They can be used for routines invoked as functions. The following example compares the string in the array NAME to the constant string DOE, JOHN and sets the value of the integer variable M accordingly.

```
     CALL SCOMP(NAME,'DOE, JOHN',M)
```

4.3 LIBRARY FUNCTIONS AND SUBROUTINES

This section presents all SYSF4 functions and subroutines in alphabetical order. To reference these subprograms by usage, see Table 4-1.

4.3.1 AJFLT

The AJFLT function converts an INTEGER*4 value to a REAL*4 value and returns that result as the function value.

Form: a = AJFLT (jsrc)

where:	jsrc	is the INTEGER*4 variable to be converted.

Function Results:

The function result is the REAL*4 value that is the result of the operation.

Example:

The following example converts the INTEGER*4 value contained in JVAL to single precision (REAL*4), multiplies it by 3.5, and stores the result in VALUE.

```
REAL*4 VALUE,AJFLT
INTEGER*4 JVAL
.
.
.
VALUE=AJFLT(JVAL)*3.5
```

4.3.2 CHAIN

The CHAIN subroutine allows a background program (or any program in the single-job system) to transfer control directly to another background program, passing it specified information. CHAIN cannot be called from a completion or interrupt routine. CHAIN does not close any of the FORTRAN logical units. When CHAINing to any other program, the user should explicitly close the opened logical units with calls to the CLOSE routine. Any routines specified in a FORTRAN USEREX library call are not executed if a CHAIN is accomplished.

Form: CALL CHAIN (dblk,var,wcnt)

where:	dblk	is the address of a four-word Radix-50 descriptor of the file specification for the program to be run. (See the PDP-11 FORTRAN Language Reference Manual for the format of the file specification.)

var	is the first variable (must start on a word boundary) in a sequence of variables with increasing memory addresses to be passed between programs in the chain parameter area (absolute locations 510 up to 700). A single array or a COMMON block (or portion of a COMMON block) is a suitable sequence of variables.
wcnt	is a word count (up to 60 words) specifying the number of words (beginning at var) to be passed to the called program. If no words are passed, then a word count of 0 is supplied.

If the size of the chain parameter area is insufficient, it can be increased by specifying the /B (or /BOTTOM) option to LINK for both the program executing the CHAIN call and the program receiving control.

The data passed can be accessed through a call to the RCHAIN routine. For more information on chaining to other programs, see Section 2.4.2.

Errors:

None.

Example:

The following example transfers control from the main program to PROG.SAV, on DT0, passing it variables.

```
REAL* PROGNM(2)                          !RAD50 FOR PROGRAM NAME
COMMON /BLK1/ A,B,C,D                    !DATA TO BE PASSED
DATA PROGNM/2RDTOPROG....SAV/
.
.
.
CALL CHAIN(PROGNM,A,8)                   !RUN DTO:PROG.SAV
CHAIN(PROGN,,0)                          !IF NO DATA PASSED
```

CLOSEC

4.3.3 CLOSEC

The CLOSEC subroutine terminates activity on the specified channel and frees it for use in another operation. The handler for the associated device must be in memory. CLOSEC cannot be called from a completion or interrupt routine.

Form: CALL CLOSEC (chan)

where:	chan	is the channel number to be closed. This argument must be located so that the USR cannot swap over it.

A CLOSEC or PURGE must eventually be issued for any channel opened for either input or output. A CLOSEC call specifying a channel that is not open is ignored.

A CLOSEC performed on a file that was opened via an IENTER causes the device directory to be updated to make that file permanent. If the device associated with the specified channel already contains a file with the same name and type, the old copy is deleted when the new file is made permanent. A CLOSEC on a file opened via LOOKUP does not require any directory operations.

When an entered file is CLOSECed, its permanent length reflects the highest block of the file written since the file was entered; for example, if the highest block written is block number 0, the file is given a length of 1; if the file was never written, it is given a length of 0. If this length is less than the size of the area allocated at IENTER time, the unused blocks are reclaimed as an empty area on the device.

Errors:

CLOSEC does not generate any errors. If the device handler for the operation is not in memory, a fatal monitor error is generated.

Example:

The following example creates and processes a 56-block file.

```
        REAL*4 DBLK(2)
        DATA DBLK/6RSYONEW,6RFILDAT/
        DATA ISIZE/56/
        .
        .
        .
        ICHAN=IGETC()
        IF(ICHAN.LT.0) GOTO 100
        IF(IENTER(ICHAN,DBLK,ISIZE)-1) 10,110,120
10      .
        .
        .
        CALL CLOSEC(ICHAN)
        CALL IFREEC(ICHAN)
        CALL EXIT
100     STOP 'NO AVAILABLE CHANNELS'
110     STOP 'CHANNEL ALREADY IN USE'
120     STOP 'NOT ENOUGH ROOM ON DEVICE'
        END
```

CONCAT

4.3.4 CONCAT

The CONCAT subroutine is used to concatenate character strings.

Form: CALL CONCAT (a,b,out[,len[,err]])

where:	a	is the array containing the left string.
	b	is the array containing the right string.
	out	is the array into which the concatenated result is placed. This array must be at least one element longer than the maximum length of the resultant string (that is, one greater than the value of len, if specified).

len	is the integer number of characters representing the maximum length of the output string. The effect of len is to truncate the output string to a given length, if necessary.
err	is the logical error flag set if the output string is truncated to the length specified by len.

The string in array a immediately followed on the right by the string in array b and a terminating null character replaces the string in array out. Any combination of string arguments is allowed so long as b and out do not specify the same array. Concatenation stops either when a null character is detected in b or when the number of characters specified by len has been moved.

If either the left or right string is a null string, the other string is copied to out. If both are null strings, then out is set to a null string. The old contents of out are lost when this routine is called.

Errors:

Error conditions are indicated by err, if specified. If err is given and the output string would have been longer than len characters, then err is set to .TRUE.; otherwise, err is unchanged.

Example:

The following example concatenates the string in array STR and the string in array IN and stores the resultant string in array OUT. OUT cannot be larger than 29 characters.

```
LOGICAL*1 IN(30),OUT(30),STR(7)
.
.
.
CALL CONCAT(STR,IN,OUT,29)
```

CVTTIM

4.3.5 CVTTIM

The CVTTIM subroutine converts a two-word internal format time to hours, minutes, seconds, and ticks.

Form: CALL CVTTIM (time,hrs,min,sec,tick)

where:	time	is the two-word internal format time to be converted. If time is considered as a two-element INTEGER*2 array, then: time (1) is the high-order time. time (2) is the low-order time.
	hrs	is the integer number of hours.
	min	is the integer number of minutes.
	sec	is the integer number of seconds.

tick	is the integer number of ticks (1/60 of a second for 60-cycle clocks; 1/50 of a second for 50-cycle clocks).

Errors:

None.

Example:

```
     INTEGER*4 ITIME
     .
     .
     .
     CALL GTIM(ITIME)                     !GET CURRENT TIME-OF-DAY
     CALL CVTTIM(ITIME,IHRS,IMIN,ISEC,ITCK)
     IF(IHRS.GE.12) GOTO 100              !TIME FOR LUNCH
```

DEVICE

4.3.6 DEVICE (FB and XM Only)

The DEVICE subroutine allows the user to set up a list of addresses to be loaded with specified values when the program is terminated. If a job terminates or is aborted with a CTRL/C from the terminal, this list is picked up by the system and the appropriate addresses are set up with the corresponding values.

This function is primarily designed to allow user programs to load device registers with necessary values. In particular, it is used to turn off a device's interrupt enable bit when the program servicing the device terminates.

Only one address list can be active at any given time; hence, if multiple DEVICE calls are issued, only the last one has any effect. The list must not be modified by the FORTRAN program after the DEVICE call has been issued, and the list must not be located in an overlay or an area over which the USR swaps.

The second argument of the call (link) provides support for a linked list of tables. The link argument is optional and causes the first word of the list to be processed as the link word.

Form: CALL DEVICE (ilist[,link])

where:	ilist	is an integer array containing address/value pairs, terminated by a zero word. On program termination, each value is moved to the corresponding address.
	link	an optional argument that is any value to indicate a linked list table is to be used.
		If the linked list form is used the first word of the array is the link list pointer.

For more information on loading values into device registers, see the assembly language .DEVICE request, Section 2.4.11.

Errors:

None.

Example:

```
     INTEGER*2 IDR11(3)              !DEVICE ARRAY SPEC
     DATA IDR11(1)/"167770/          !DR11 CSR ADDRESS (OCTAL)
     DATA IDR11(2)/0/                !VALUE TO CLEAR INTERRUPT ENABLE
     DATA IDR11(3)/0/                !AND END-OF-LIST FLAG
     CALL DEVICE(IDR11)              !SET UP FOR ABORT
```

DJFLT

4.3.7 DJFLT

The DJFLT function converts an INTEGER*4 value into a REAL*8 (DOUBLE PRECISION) value and returns that result as the function value.

Form: d = DJFLT (jsrc)

where:	jsrc	specifies the INTEGER*4 variable which is to be converted.

Notes:

If DJFLT is used, it must be explicitly defined (REAL*8 DJFLT) or implicitly defined (IMPLICIT REAL*8 (D)) in the FORTRAN program. If this is not done, its type is assumed to be REAL*4 (single precision).

Function Results:

The function result is the REAL*8 value that is the result of the operation.

Example:

```
          INTEGER*4 JVAL
          REAL*8 DJFLT,D
          .
          .
          .
          D=DJFLT(JVAL)
```

GETSTR

4.3.8 GETSTR

The GETSTR subroutine reads a formatted ASCII record from a specified FORTRAN logical unit into a specified array. The data is truncated (trailing blanks removed) and a null byte is inserted at the end to form a character string.

GETSTR can be used in main program routines or in completion routines but cannot be used in both at the same time. If GETSTR is used in a completion routine, it cannot be the first I/O operation on the specified logical unit.

Form: CALL GETSTR (lun,out,len,err)

where: lun	is the integer FORTRAN logical unit number of a formatted sequential file from which the string is to be read.
out	is the array to receive the string; this array must be one element longer than len.
len	is the integer number representing the maximum length of the string to be input.
err	is the LOGICAL*1 error flag that is set to .TRUE if an error occurred. If an error did not occur, it is .FALSE.

Errors:

Error conditions are indicated by err. If err is .TRUE, the values returned are as follows:

ERR = -1 End of file for a read operation
ERR = -2 Hard error for a read operation
ERR = -3 More than len bytes were contained in a record.

Example:

The following example reads a string of up to 80 characters from logical unit 5 into the array STRING.

```
        LOGICAL*1 STRING(81),ERR
        .
        .
        .
        CALL GETSTR(5,STRING,80,ERR)
```

GTIM

4.3.9 GTIM

The GTIM subroutine allows user programs to access the current time of day. The time is returned in two words and is given in terms of clock ticks past midnight. If the system does not have a line clock, a value of 0 is returned. If an RT-11 monitor TIME command has not been entered, the value returned is the time elapsed since the system was bootstrapped, rather than the time of day.

Form: CALL GTIM (itime)

where: itime is the two-word area to receive the time of day.

The high-order time is returned in the first word, the low-order time in the second word. The SYSF4 routine CVTTIM (Section 4.3.5) can be used to convert the time into hours, minutes, seconds and ticks. CVTTIM performs the conversion based on the monitor configuration word for 50- or 60-cycle clocks (see Section 2.2.6). Under an FB or XM monitor, the time-of-day is automatically reset after 24:00 when a GTIM is executed; under the single-job monitor, it is not.

Errors:

None.

Example:

```
        INTEGER*4 JTIME
        .
        .
        .
        CALL GTIM(JTIME)
```

GTJB

4.3.10 GTJB

The GTJB subroutine passes certain job parameters back to the user program.

Form: CALL GTJB (addr)

where: addr is an eight-word area to receive the job parameters. This area, considered as an eight-element INTEGER*2 array, has the following format:

addr(1)	job number. (0=background, 2=foreground)
addr(2)	high memory limit
addr(3)	low memory limit
addr(4)	beginning of I/O channel space
addr(5)-addr(8)	reserved for future use

For more information on passing job parameters, see the assembly language .GTJB request, Section 2.4.

Errors:

None.

Example:

```
        INTEGER*2 PARAMS(8)
        CALL GTJB(PARAMS)
        IF(PARAMS(1).EQ.0) TYPE 99
    99  FORMAT (' THIS IS THE BACKGROUND JOB')
```

GTLIN

4.3.11 GTLIN

The GTLIN subroutine requires the USR. It transfers a line of input from the console terminal or an active indirect command file to the user program. This request is used to get information from the user, and it allows the program to operate through indirect files. The maximum size of the input line is 80 characters.

Form: CALL GTLIN (result[,prompt])

where: result — is the array receiving the string. This LOGICAL*1 array contains a maximum of 80 characters plus 0 as the end indicator.

prompt — is an optional prompt string to be printed before getting the input line. The string format is the same as that used by the PRINT subroutine.

Errors:

None

IADDR

4.3.12 IADDR

The IADDR function returns the 16-bit absolute memory address of its argument as the integer function value.

Form: i = IADDR (arg)

where: arg — is the variable, constant, or expression whose memory address is to be obtained.

Errors:

None.

Example:

IADDR can be used to find the address of an assembly language global area. For example:

```
        EXTERNAL CAREA
        J=IADDR(CAREA)
```

IAJFLT

4.3.13 IAJFLT

The IAJFLT function converts an INTEGER*4 value to a REAL*4 value and stores the result.

Form: i = IAJFLT (jsrc,ares)

where: jsrc — is the INTEGER*4 variable to be converted.

ares — is the REAL*4 variable or array element to receive the converted value.

Function Results:

The function result indicates the following:

i = -2	Significant digits were lost during the conversion.
= -1	Normal return; the result is negative.
= 0	Normal return; the result is 0.
= 1	Normal return; the result is positive.

Example:

```
        INTEGER*4 JVAL
        REAL*4 RESULT
        .
        .
        .
        IF(IAJFLT(JVAL,RESULT).EQ.-2) TYPE 99
99      FORMAT (' OVERFLOW IN INTEGER*4 TO REAL CONVERSION')
```

IASIGN

4.3.14 IASIGN

The IASIGN function sets information in the FORTRAN logical unit table (overriding the defaults) so that the specified information is used when the FORTRAN Object Time System (OTS) opens the logical unit. This function can be used with ICSI (see Section 4.3.18) to allow a FORTRAN program to accept a standard CSI input specification. IASIGN must be called before the unit is opened; that is, before any READ, WRITE, PRINT, TYPE, or ACCEPT statements are executed that reference the logical unit.

Form: i = IASIGN (lun,idev[,ifiltyp[,isize[,itype]]])

where:	lun	is an INTEGER*2 variable, constant, or expression specifying the FORTRAN logical unit for which information is being specified.
	idev	is a one-word Radix-50 device name; this can be the first word of an ICSI input or output file specification.
	ifiltyp	is a three-word Radix-50 file name and file type; this can be words 2 through 4 of an ICSI input or output file specification.
	isize	is the length (in blocks) to allocate for an output file; this can be the fifth word of an ICSI output specification. If 0, the larger of either one-half the largest empty segment or the entire second largest empty segement is allocated (see Section 2.4). If the value specified for length is -1, the entire largest empty segment is allocated.

itype is an integer value determining the optional attributes to be assigned to the file. This value is obtained by adding the values that correspond to the desired operations:

1	use double buffering for output
2	open the file as a temporary file
4	Force a LOOKUP on an existing file during the first I/O operation (otherwise, the first FORTRAN I/O operation determines how the file is opened). For example, if the next I/O operation is a write, an IENTER is performed on the specified logical unit. A read causes a LOOKUP.
8	expand carriage control information (see Notes below)
16	do not expand carriage control information
32	file is read-only

Notes:

Expanded carriage control information applies only to formatted output files and means that the first character of each record is used as a carriage control character when processing a write operation to the given logical unit. The first character is removed from the record and converted to the appropriate ASCII characters to simulate the requested carriage control.

If carriage control information is not expanded, the first character of each record is unmodified and the FORTRAN OTS outputs a line feed, followed by the record, followed by a carriage return.

If carriage control is unspecified, the FORTRAN OTS sends expanded carriage control information to the terminal and line printer and sends unexpanded carriage control information to all other devices and files. See the PDP-11 FORTRAN Language Reference Manual for further carriage control information.

Function Results:

i = 0	Normal return.
<> 0	The specified logical unit is already in use or there is no space for another logical unit association.

Example:

The following example creates an output file on logical unit 3 using the first output file given to the RT-11 Command String Interpreter (CSI), sets it up for double buffering, creates an input file on logical unit 4 based on the first input file specification given to the RT-11 CSI, and makes it available for read-only access.

```
      INTEGER*2 SPEC(39)
      REAL*4 EXT(2)
      DATA EXT/6RDATDAT,6RDATDAT/        !DEFAULT FILE TYPE IS DAT
      .
      .
      .
10    IF(ICSI(SPEC,TYP,,,0).NE.0) GOTO 10
C
C     DO NOT ACCEPT ANY SWITCHES
C
      CALL IASIGN(3,SPEC(1),SPEC(2),SPEC(5),1)
      CALL IASIGN(4,SPEC(16),SPEC(17),0,32)
```

ICDFN

4.3.15 ICDFN

The ICDFN function increases the number of input/output channels. Note that ICDFN defines new channels; the previously-defined channels are not used. Thus, an ICDFN for 20 channels (while the 16 original channels are defined) causes only 20 I/O channels to exist; the space for the original 16 is unused. The space for the new channel area is allocated out of the free space managed by the FORTRAN system.

Form: i = ICDFN (num)

where:	num	is the integer number of channels to be allocated. The number of channels must be greater than 16 and can be a maximum of 256. SYSF4 can use all new channels greater than 16 without a call to IGETC; the FORTRAN system input/output uses only the first 16 channels. This argument must be positioned so that the USR cannot swap over it.

Notes:

1. ICDFN cannot be issued from a completion or interrupt routine.
2. It is recommended that the ICDFN function be used at the beginning of the main program before any I/O operations are initiated.
3. If ICDFN is executed more than once, a completely new set of channels is created each time ICDFN is called.
4. ICDFN requires that extra memory space be allocated to foreground programs (see Section 4.1.4).

Function Results:

i = 0	Normal return.
= 1	An attempt was made to allocate fewer channels than already exist.
= 2	Not enough free space is available for the channel area.

Example:

```
      IF(ICDFN(24).NE.0) STOP 'NOT ENOUGH MEMORY'
```

4.3.16 ICHCPY (FB and XM Only)

The ICHCPY function opens a channel for input, logically connecting it to a file that is currently open by another job for either input or output. This function can be used by either the foreground or the background. An ICHCPY must be done before the first read or write for the given channel.

Form: i = ICHCPY (chan,ochan)

where:	chan	is the channel the job will use to read the data.
	ochan	is the channel number of the other job that is to be copied.

Notes:

1. If the other job's channel was opened via an IENTER function or a .ENTER programmed request to create a file, the copier's channel indicates a file that extends to the highest block that the creator of the file had written at the time the ICHCPY was executed.

2. A channel that is open on a sequential-access device should not be copied, because buffer requests can become intermixed.

3. A program can write to a file (that is being created by the other job) on a copied channel just as it could if it were the creator. When the copier's channel is closed, however, no directory update takes place.

Errors:

i = 0	Normal return.
= 1	Other job does not exist or does not have the specified channel (ochan) open.
= 2	Channel (chan) is already open.

4.3.17 ICMKT

The ICMKT function causes one or more scheduling requests (made by an ISCHED, ITIMER or MRKT routine) to be cancelled. Support for ICMKT in SJ also requires timer support.

Form: i = ICMKT (id,time)

where:	id	is the identification integer of the request to be cancelled. If id is equal to 0, all scheduling requests are cancelled.
	time	is the name of a 2-word area in which the monitor returns the amount of time remaining in the cancelled request.

For further information on cancelling scheduling requests, see the assembly language .CMKT request, Section 2.4.

Errors:

i = 0	Normal return.
= 1	id was not equal to 0 and no schedule request with that identification could be found.

Example:

```
        INTEGER*4 J
        .
        .
        .
        CALL ICMKT(0,J)      !ABORT ALL TIMER REQUESTS NOW
        .
        .
        .
        END
```

ICSI

4.3.18 ICSI

The ICSI function calls the RT-11 Command String Interpreter in special mode to parse a command string and return file descriptors and options to the program. In this mode, the CSI does not perform any handler IFETCHes, CLOSECs, IENTERs, or LOOKUPs. An optional argument (cstring) provides ICSI with the capability of returning the original command string. This argument is allowed only when the input is from the console terminal. ICSI cannot be called from a completion or interrupt routine.

Form: i = ICSI (filspc,deftyp,[cstring],[option],x)

where: filspc is the 39-word area to receive the file specifications. The format of this area (considered as a 39-element INTEGER*2 array) is:

filspc(1)- filspc(4)	output file number 1 specification
filspc(5)	output file number 1 length
filspc(6)- filspc(9)	output file number 2 specification
filspc(10)	output file number 2 length
filspc(11)- filspc(14)	output file number 3 specification
filspc(15)	output file number 3 length
filspc(16)- filspc(19)	input file number 1 specification
filspc(20)- filspc(23)	input file number 2 specification
filspc(24)- filspc(27)	input file number 3 specification

filspc(28)- filspc(31)	input file number 4 specification
filspc(32)- filspc(35)	input file number 5 specification
filspc(36)- filspc(39)	input file number 6 specification

deftyp is the table of Radix-50 default file types to be assumed when a file is specified without a file type.

deftyp(1) is the default for all input file types.
deftyp(2) is the default file type for output file number 1.
deftyp(3) is the default file type for output file number 2.
deftyp(4) is the default file type for output file number 3.

cstring is the area that contains the ASCIZ command string to be interpreted; the string must end in a zero byte. If the argument is omitted, the system prints the prompt character (*) at the terminal and accepts a command string.

option is the name of an INTEGER*2 array dimensioned (4,n) where n represents the number of options defined to the program. This argument must be present if the value specified for "x" is non-zero. This array has the following format for the nth option described by the array.

option(1,n) is the one-character ASCII name of the option.
option(2,n) is set by the routine to 0, if the option did not occur; to 1, if the option occurred without a value; to 2, if the option occurred with a value.
option(3,n) is set to the file number on which the option is specified.
option(4,n) is set to the specified value if option(2,n) is equal to 2.

x is the number of options defined in the array "option".

Notes:

The array "option" must be set up to contain the names of the valid options. For example, use the following to set up names for five options:

```
INTEGER*2 SW(4,5)
DATA SW(1,1)/'S'/,SW(1,2)/'M'/,SW(1,3)/'I'/
DATA SW(1,4)/'L'/,SW(1,5)/'E'/
```

Multiple occurrences of the same option are supported by allocating an entry in the option array for each occurrence of the option. Each time the option occurs in the option array, the next unused entry for the named option is used.

The arguments of ICSI must be positioned so that the USR cannot swap over them.

For more information on calling the Command String Interpreter, see the assembly language .CSISPC request, Section 2.4.

Errors:

i = 0	Normal return.
= 1	Illegal command line; no data was returned.
= 2	An illegal device specification occurred in the string.
= 3	An illegal option was specified, or a given option was specified more times than were allowed for in the option array.

Example:

The following example causes the program to loop until a valid command is typed at the console terminal.

```
        INTEGER*2 SPEC(39)
        REAL*4 EXT(2)
        DATA EXT/6RDATDAT,6RDATDAT/
        .
        .
        .
10      TYPE 99
99      FORMAT (' ENTER VALID CSI STRING WITH NO OPTIONS')
        IF(ICSI(SPEC,EXT,,,0).NE.0) GOTO 10
```

ICSTAT

4.3.19 ICSTAT (FB and XM Only)

The ICSTAT function furnishes the user with information about a channel. It is supported only in the FB or XM environment; no information is returned when operating under the single-job monitor.

Form: i = ICSTAT (chan,addr)

where:

chan	is the channel whose status is desired.
addr	is a six-word area to receive the status information. The area, as a six-element INTEGER*2 array, has the following format.

addr(1)	channel status word (see Section 2.4)
addr(2)	starting absolute block number of file on this channel
addr(3)	length of file
addr(4)	highest block number written since file was opened (see Section 2.4)

addr(5)	unit number of device with which this channel is associated
addr(6)	Radix-50 of device name with which the channel is associated

Errors:

i = 0	Normal return.
= 1	Channel specified is not open.

Example:

The following example obtains channel status information about channel I.

```
        INTEGER*2 AREA(6)
        I=7
        IF(ICSTAT(I,AREA).NE.0) TYPE 99,I
 99     FORMAT(1X,'CHANNEL',I4,'IS NOT OPEN')
```

4.3.20 IDELET

The IDELET function deletes a named file from an indicated device. Since this routine passes information to the USR, provisions must be made to prevent information required by the USR from being swapped. This is accomplished by moving all parameters to the stack and pointing to these parameters in the program request. IDELET cannot be issued from a completion or interrupt routine.

Form: i = IDELET (chan,dblk[,seqnum])

where:

chan	is the channel to be used for the delete operation.
dblk	is the four-word Radix-50 specification (dev:filnam.typ) for the file to be deleted.
seqnum	file number for cassette operations: if this argument is blank, a value of 0 is assumed. For magtape operation, it describes a file sequence number that can have the following values:

Value	Meaning
-1	For LOOKUP or IDELET, this value suppresses rewinding and searching for a file name from the current tape position. Note that if the position is unknown, the handler executes a positioning algorithm that involves backspacing until an end-of-file label is found. The user should not use any other value since all other negative values are reserved for future use.

0	For LOOKUP or IDELET, this value rewinds the magtape and spaces forward until the file name is found. For .ENTER it rewinds the magtape and spaces forward until the file name is found or until the logical end of tape is detected. If the file name is found, it is deleted and tape search continues.
n	Where n is any positive number. This value positions the magtape at file sequence number n. If the file represented by the file sequence number is greater than two files away from the beginning of tape, a rewind is performed. If not, the tape is backspaced to the file.

NOTE

The arguments of IDELET must be located so that the USR cannot swap over them.

The specified channel is left inactive when the IDELET is complete. IDELET requires that the handler to be used be resident (via an IFETCH call) at the time the IDELET is issued. If it is not, a monitor error occurs.

For further information on deleting files, see the assembly language .DELETE request, Section 2.4.

Errors:

i = 0	Normal return.
= 1	Channel specified is already open.
= 2	File specified was not found.
= 3	Device is use

Example:

The following example deletes a file named FTN5.DAT from SY0.

```
        REAL*4 FILNAM(2)
        DATA FILNAM/6RSYOFTN,6R5  DAT/
        .
        .
        .
        I=IGETC()
        IF(I.LT.0) STOP 'NO CHANNEL'
        CALL IDELET(I,FILNAM)
        CALL IFREEC(I)
```

IDJFLT

4.3.21 IDJFLT

The IDJFLT function converts an INTEGER*4 value into a REAL*8 (DOUBLE PRECISION) value and stores the result.

Form: i = IDJFLT (jsrc,dres)

where:	jsrc	specifies the INTEGER*4 variable that is to be converted.
	dres	specifies the REAL*8 (or DOUBLE PRECISION) variable to receive the converted value.

Function Results:

The function result indicates the following:

i = -1	Normal return; the result is negative.
= 0	Normal return; the result is 0.
= 1	Normal return; the result is positive.

Example:

```
        INTEGER*4 JJ
        REAL*8 DJ
        .
        .
        .
        IF(IDJFLT(JJ,DJ).LE.0) TYPE 99
   99   FORMAT (' VALUE IS NOT POSITIVE')
```

IDSTAT

4.3.22 IDSTAT

The IDSTAT function is used to obtain information about a particular device. IDSTAT cannot be issued from a completion or interrupt routine.

Form: i = IDSTAT (devnam,cblk)

where:	devnam	is the Radix-50 device name.
	cblk	is the four-word area used to store the status information. The area, as a four-element INTEGER*2 array, has the following format:

cblk(1)	device status word (see Section 2.4.13)
cblk(2)	size of handler in bytes
cblk(3)	entry point of handler (non-zero implies that the handler is in memory)
cblk(4)	size of the device (in 256-word blocks) for block-replaceable devices; zero for sequential-access devices

NOTE

The arguments of IDSTAT must be positioned so that the USR cannot swap over them.

IDSTAT looks for the device specified by devnam and, if found, returns four words of status in cblk. Errors:

i = 0	Normal return.
= 1	Device not found in monitor tables.

Example:

The following example determines whether the line printer handler is in memory. If it is not, the program stops and prints a message to indicate that the handler must be loaded.

```
REAL*4 IDNAM
INTEGER*2 CBLK(4)
DATA IDNAM/3RLP /
DATA CBLK/4*0/
CALL IDSTAT(IDNAM,CBLK)
IF(CBLK(3).EQ.0) STOP 'LOAD THE LP HANDLER AND RERUN'
```

IENTER

4.3.23 IENTER

The IENTER function allocates space on the specified device and creates a tentative directory entry for the named file. If a file of the same name already exists on the specified device, it is not deleted until the tentative entry is made permanent by CLOSEC. The file is attached to the channel number specified.

Form: i = IENTER (chan,dblk,length[,seqnum])

where:

chan	is the integer specification for the RT-11 channel to be associated with the file.
dblk	is the four-word Radix-50 descriptor of the file to be operated upon.
length	is the integer number of blocks to be allocated for the file. If 0, the larger of either one-half the largest empty segment or the entire second largest empty segment is allocated (see Section 2.4.14). If the value specified for length is -1, the entire largest empty segment is allocated.
seqnum	file number for cassette. If this argument is blank, a value of 0 is assumed. For magtape, it describes a file sequence number that can have the following values: -1 - means space to the logical-end-of-tape and enter file 0 - means rewind the magtape and space forward until the file name is found or the logical-end-of-tape is detected. If file name is found, delete it and continue tape search.

n - means position magtape at file sequence number n. If the file represented by the file sequence number is greater than two files away from beginning of tape, then a rewind is performed. If not, the tape is backspaced to the file.

Notes:

1. IENTER cannot be issued from a completion or interrupt routine.

2. IENTER requires that the appropriate device handler be in memory.

3. The arguments of IENTER must be positioned so that the USR does not swap over them.

For further information on creating tentative directory entries, see the assembly language .ENTER request, Section 2.4.

Errors:

i = n	Normal return; number of blocks actually allocated (n = 0 for nonfile-structured IENTER).
= -1	Channel (chan) is already in use.
= -2	In a fixed-length request, no space greater than or equal to length was found.
= -3	Device in use
= -4	File sequence number not found

Example:

The following example allocates a channel for file TEMP.TMP on Y0. If no channel is available, the program prints a message and halts.

```
      REAL*4 DBLK(2)
      DATA DBLK/6RSYOTEM,6RP  TMP/
      ICHAN=IGETC()
      IF(ICHAN.LT.0) STOP 'NO AVAILABLE CHANNEL'
C
C     CREATE TEMPORARY WORK FILE
C
      IF(IENTER(ICHAN,DBLK,20).LT.0) STOP 'ENTER FAILURE'
      .
      .
      .
      CALL PURGE(ICHAN)
      CALL IFREEC(ICHAN)
```

IFETCH

4.3.24 IFETCH

The IFETCH function loads a device handler into memory from the system device, making the device available for input/output operations. The handler is loaded into the free area managed by the FORTRAN system. Once the handler is loaded, it cannot be released and the memory in which it resides cannot be reclaimed. IFETCH cannot be issued from a completion or interrupt routine.

Form: i = IFETCH (devnam)

where: devnam	is the one-word Radix-50 name of the device for which the handler is desired. This argument can be the first word of an ICSI input or output file specification. This argument must be positioned so that the USR cannot swap over it.

For further information on loading device handlers into memory, see the assembly language .FETCH request, Section 2.4.16.

Errors:

i = 0	Normal return.
= 1	Device name specified does not exist.
= 2	Not enough room exists to load the handler.
= 3	No handler for the specified device exists on the system device.

Example:

The following example requests the DX1 handler to be loaded into memory; execution stops if the handler cannot be loaded.

```
        REAL*4 IDNAM
        DATA IDNAM/3RDX1/
        .
        .
        .
        IF (IFETCH(IDNAM).NE.0) STOP 'FATAL ERROR FETCHING HANDLER'
```

IFREEC

4.3.25 IFREEC

The IFREEC function returns a specified RT-11 channel to the available pool of channels. Before IFREEC is called, the specified channel must be closed or deactivated with a CLOSEC (see Section 4.3.3) or a PURGE (see Section 4.3.66) call. IFREEC cannot be called from a completion or interrupt routine. IFREEC calls must be issued only for channels that have been successfully allocated by IGETC calls; otherwise, the results are unpredictable.

Form: i = IFREEC (chan)

where: chan	is the integer number of the channel to be freed.

Errors:

i = 0	Normal return.
= 1	Specified channel is not currently allocated.

Example:

See the example under IGETC, (Section 4.3.26).

4.3.26 IGETC

The IGETC function allocates an RT-11 channel (in the range 0-17 octal) and marks it in use for the FORTRAN I/O system. IGETC cannot be issued from a completion or interrupt routine.

Form: i = IGETC()

Function Results:

i = -1	No channels are available.
= n	Channel n has been allocated.

Example:

```
ICHAN=IGETC()               !ALLOCATE CHANNEL
IF(ICHAN.LT.0) STOP 'CANNOT ALLOCATE CHANNEL'
.
.
.
CALL IFREEC(ICHAN)          !FREE IT WHEN THROUGH
.
.
.
END
```

4.3.27 IGETSP

The IGETSP subroutine returns the address and size (number of words) of free space obtained from the FORTRAN system. When this space is obtained, it is allocated for the duration of the program.

Form: i = IGETSP (min,max,iaddr)

where:	min	is the minimum space to be obtained without an error indicating that the desired amount of space is not available.
	max	is the maximum space to be obtained without an error indicating that the desired amount of space is not available.
	iaddr	is the integer specifying the address of the start of the free space (buffer).

Function Results:

i = -1	Error: not enough free space is available to meet the minimum requirements; no allocation was taken from the FORTRAN system free space.
i = n	is the actual size allocated whose value is min .GE. n .LE. max.

The size (min, max, n) is specified in words. Extreme caution should be exercised to avoid using all of the free space allocated by the FORTRAN system. If the FORTRAN system runs out of dynamic free space, fatal errors (Error 29, 30, 42, etc.) occur.

IJCVT

4.3.28 IJCVT

The IJCVT function converts an INTEGER*4 value to INTEGER*2 format. If ires is not specified, the result returned is the INTEGER*2 value of jsrc. If ires is specified, the result is stored there.

Form: i = IJCVT (jsrc[,ires])

where:	jsrc	specifies the INTEGER*4 variable or array element whose value is to be converted.
	ires	specifies the INTEGER*2 entity to receive the conversion result.

Function results if ires is specified:

i = -2	An overflow occurred during conversion.
= -1	Normal return; the result is negative.
= 0	Normal return; the result is 0.
= 1	Normal return; the result is positive.

Example:

```
        INTEGER*4 JVAL
        INTEGER*2 IVAL
        .
        .
        .
        IF(IJCVT(JVAL,IVAL).EQ.-2) TYPE 99
99      FORMAT(' NUMBER TOO LARGE IN IJCVT CONVERSION')
```

ILUN

4.3.29 ILUN

The ILUN function returns the RT-11 channel number with which a FORTRAN logical unit is associated.

Form: i = ILUN (lun)

where:	lun	is an integer expression whose value is a FORTRAN logical unit number in the range 1-99.

Function Results:

i = -1	Logical unit is not open.
= -2	Logical unit is opened to console terminal.
= +n	RT-11 channel number n is associated with lun.

Example:

```
    PRINT 99
99  FORMAT(' PRINT DEFAULTS TO LOGICAL UNIT 6, WHICH FURTHER DEFAULTS TO LP:')
    LUNRT=ILUN(6)                  !WHICH RT-11 CHANNEL IS RECEIVING I/O?
```

INDEX

4.3.30 INDEX

The INDEX subroutine searches a string for the occurrence of another string and returns the character position of the first occurrence of that string.

Form: CALL INDEX (a,pattrn,[i],m)

or

m = INDEX (a,pattrn[,i])

where:		
	a	is the array containing the string to be searched.
	pattrn	is the string being sought.
	i	is the integer starting character position of the search in a. If i is omitted, a is searched beginning at the first character position.
	m	is the integer result of the search; m equals the starting character position of pattrn in a, if found; otherwise it is 0.

Errors:

None.

Example:

The following example searches the array STRING for the first occurrence of strings EFG and XYZ and searches the string ABCABCABC for the occurrence of string ABC after position 5.

```
CALL SCOPY('ABCDEFGHI',STRING)      !INITIALIZE STRING
CALL INDEX(STRING,'EFG',,M)         !M=5
CALL INDEX(STRING,'XYZ',,N)         !N=0
CALL INDEX('ABCABCABC','ABC',5,L)   !L=7
```

INSERT

4.3.31 INSERT

The INSERT subroutine replaces a portion of one string with another string.

Form: CALL INSERT (in,out,i[,m])

Example:

```
     EXTERNAL CLKSUB                      !SUBR TO HANDLE KW11-P CLOCK
     .
     .
     .
     I=INTSET('104,6,0,CLKSUB)            !ATTACH ROUTINE
     IF (I.NE.0) GOTO 100                 !BRANCH IF ERROR
     .
     .
     .
     END
     SUBROUTINE CLKSUB(ID)
     .
     .
     .
     END
```

IPEEK

4.3.33 IPEEK

The IPEEK function returns the contents of the word located at a specified absolute 16-bit memory address. This function can be used to examine device registers or any location in memory.

Form: i = IPEEK (iaddr)

where:	iaddr	is the integer specification of the absolute address to be examined. If this argument is not an even value, a trap results.

Function Result:

The function result (i) is set to the value of the word examined.

Example:

```
     ISWIT = IPEEK('177570)               !GET VALUE OF CONSOLE SWITCHES
```

IPEEKB

4.3.34 IPEEKB

The IPEEKB subroutine returns the contents of a byte located at a specified absolute byte address. Since this routine operates in a byte mode, the address supplied can be even or odd. This subroutine can be used to examine device registers or any byte in memory.

Form: i = IPEEKB (iaddr)

where:	iaddr	is the integer specification of the absolute byte address to be examined. Unlike the IPEEK subroutine, the IPEEKB subroutine allows odd addresses.

Function Result:

The function result (i) is set to the value of the byte examined.

Example:

```
     IERR = IPEEKB('53)  !Get error byte
```

IPOKE

4.3.35 IPOKE

The IPOKE subroutine stores a specified 16-bit integer value into a specified absolute memory location. This subroutine can be used to store values in device registers.

Form: CALL IPOKE (iaddr,ivalue)

where:	iaddr	is the integer specification of the absolute address to be modified. If this argument is not an even value, a trap results.
	ivalue	is the integer value to be stored in the given address (iaddr).

Errors:

None.

Example:

The following example displays the value of IVAL in the console display register.

```
          CALL IPOKE('177570,IVAL)
```

To set bit 12 in the JSW without zeroing any other bits in the JSW, use the following procedure.

```
          CALL IPOKE('44,'10000.OR.IPEEK('44))
```

IPOKEB

4.3.36 IPOKEB

The IPOKEB subroutine stores a specified eight-bit integer value into a specified byte location. Since this routine operates in a byte mode, the address supplied can be even or odd. This subroutine can be used to store values in device registers.

Form: CALL IPOKEB (iaddr,ivalue)

where:	iaddr	is the integer specification of the absolute address to be modified. Unlike the IPOKE subroutine, the IPOKEB subroutine allows odd addresses.
	ivalue	is the integer value to be stored in the given address specified by the iaddr argument.

Errors:

None

Example: (see section 4.3.35)

IQSET

4.3.37 IQSET

The IQSET function is used to make the RT-11 queue larger (that is, to add available elements to the queue). These elements are allocated out of the free space managed by the FORTRAN system. IQSET cannot be called from a completion or interrupt routine.

Form: i = IQSET (qleng)

where:	qleng	is the integer number of elements to be added to the queue. This argument must be positioned so that the USR does not swap over it.

All RT-11 I/O transfers are done through a centralized queue management system. If I/O traffic is very heavy and not enough queue elements are available, the program issuing the I/O requests can be suspended until a queue element becomes available. In an FB or XM system, the other job runs while the first program waits for the element. When IQSET is used in a program to be run in the foreground, the FRUN command must be modified to allocate space for the queue elements (see Section 4.1.4).

A general rule to follow is that each program should contain one more queue element than the total number of I/O and timer requests that will be active simultaneously. Timing functions such as ITWAIT and MRKT also cause elements to be used and must be considered when allocating queue elements for a program. Note that if synchronous I/O is done (IREADW/IWRITW, etc.) and no timing functions are done, no additional queue elements need be allocated. Note also that FORTRAN IV allocates four queue elements. See Section 4.2 for a list of SYSF4 calls that use queue elements.

For further information on adding elements to the queue, see the assembly language .QSET request, Section 2.4.

Function Results:

i = 0	Normal return.
= 1	Not enough free space is available for the number of queue elements to be added; no allocation was made.

Example:

```
     IF(IQSET(5).NE.0) STOP 'NOT ENOUGH FREE SPACE FOR QUEUE ELEMENTS'
```

4.3.38 IRAD50

The IRAD50 function converts a specified number of ASCII characters to Radix-50 and returns the number of characters converted. Conversion stops on the first non-Radix-50 character encountered in the input or when the specified number of ASCII characters have been converted.

Form: n = IRAD50 (icnt,input,output)

where:		
	icnt	is the number of ASCII characters to be converted.
	input	is the area from which input characters are taken.
	output	is the area into which Radix-50 words are stored.

Three characters of text are packed into each word of output. The number of output words modified is computed by the expression (in integer words):

(icnt+2)/3

Thus, if a count of 4 is specified, two words of output are written even if only a one-character input string is given as an argument.

Function Results:

The integer number of input characters actually converted (n) is returned as the function result.

Example:

```
        REAL*8 FSPEC
        CALL IRAD50(12,'SYOTEMP  DAT',FSPEC)
```

4.3.39 IRCVD/IRCVDC/IRCVDF/IRCVDW(FB and XM Only)

There are four forms of the receive data function; these are used in conjunction with the ISDAT (send data) functions to allow a general data/message transfer system. The receive data functions issue RT-11 receive data programmed requests (see Section 2.4). These functions require a queue element; this should be considered when the IQSET function (Section 4.3.37) is executed.

IRCVD

The IRCVD function is used to receive data and continue execution. The operation is queued and the issuing job continues execution. At some point when the job must receive the transmitted message, an MWAIT should be executed. This causes the job to be suspended until the message has been received.

Form: i = IRCVD (buff,wcnt)

where:	buff	is the array to be used to buffer the data received. The array must be one word larger than the message to be received because the first word contains the integer number of words actually transmitted when IRCVD is complete.
	wcnt	is the maximum integer number of words that can be received.

Errors:

i = 0	Normal return.
= 1	No other job exists in the system.

Example:

```
        INTEGER*2 MSG(41)
        .
        .
        .
        CALL IRCVD(MSG,40)
        .
        .
        .
        CALL MWAIT
```

IRCVDC

The IRCVDC function receives data and enters an assembly language completion routine when the message is received. The IRCVDC is queued and program execution stays with the issuing job. When the other job sends a message, the completion routine specified is entered.

Form: i = IRCVDC (buff,wcnt,crtn)

where:	buff	is the array to be used to buffer the data received. The array must be one word larger than the message to be received because the first word contains the integer number of words actually transmitted when IRCVDC is complete.
	wcnt	is the maximum integer number of words to be received.
	crtn	is the assembly language completion routine to be entered. This name must be specified in a FORTRAN EXTERNAL statement in the routine that issues the IRCVDC call.

Errors:

i = 0	Normal return.
= 1	No other job exists in the system.

IRCVDF

The IRCVDF function receives data and enters a FORTRAN completion subroutine (see Section 4.2.1) when the message is received. The

IRCVDF is queued and program execution continues with the issuing job. When the other job sends a message, the FORTRAN completion routine specified is entered.

Form: i = IRCVDF (buff,wcnt,area,crtn)

where:	buff	is the array to be used to buffer the data received. The array must be one word larger than the message to be received because the first word contains the integer number of words actually transmitted when IRCVDF is complete.
	wcnt	is the maximum integer number of words to be received.
	area	is a four-word area to be set aside for linkage information. This area must not be modified by the FORTRAN program and the USR must not swap over it. This area can be reclaimed by other FORTRAN completion routines when crtn has been entered.
	crtn	is the FORTRAN completion routine to be entered. This name must be specified in an EXTERNAL statement in the FORTRAN routine that issues the IRCVDF call.

Errors:

i = 0	Normal return.
= 1	No other job exists in the system.

Example:

```
INTEGER*2 MSG(41),AREA(4)
EXTERNAL RMSGRT
.
.
.
CALL IRCVDF(MSG,40,AREA,RMSGRT)
```

IRCVDW

The IRCVDW function is used to receive data and wait. This function queues a message request and suspends the job issuing the request until the other job sends a message. When execution of the issuing job resumes, the message has been received, and the first word of the buffer indicates the number of words transmitted.

Form: i = IRCVDW (buff,wcnt)

where:	buff	is the array to be used to buffer the data received. The array must be one word larger than the message to be received because the first word contains the integer number of words actually transmitted when IRCVDW is complete.
	wcnt	is the maximum integer number of words to be received.

Errors:

i	= 0	Normal return.
	= 1	No other job exists in the system.

Example:

```
        INTEGER*2 MSG(41)
        IF(IRCVDW(MSG,40).NE.0) STOP 'UNEXPECTED ERROR'
```

IREAD/IREADC/IREADF/IREADW

4.3.40 IREAD/IREADC/IREADF/IREADW

SYSF4 provides four modes of I/O: IREAD/IWRITE, IREADC/IWRITC, IREADF/IWRITF, and IREADW/IWRITW. These functions require a queue element; this should be considered when the IQSET function (Section 4.3.37) is executed.

IREAD

The IREAD function transfers a specified number of words from the file (first block of file = 0) associated with the indicated channel into memory. Control returns to the user program immediately after the IREAD function is initiated. No special action is taken when the transfer is completed.

Form: i = IREAD (wcnt,buff,blk,chan)

where:	wcnt	is the relative integer number of words to be transferred.
	buff	is the array to be used as the buffer. This array must contain at least wcnt words.
	blk	is the relative integer block number of the file to be read. The user program normally updates blk before it is used again.
	chan	is the relative integer specification for the RT-11 channel to be used.

NOTE

The blk argument must be updated, if necessary, by the user program. For example, if the program is reading two blocks at a time, blk should be updated by 2.

When the user program needs to access the data read on the specified channel, an IWAIT function should be issued. This ensures that the IREAD operation has been completed. If an error occurred during the transfer, the IWAIT function indicates the error.

Errors:

i = n	Normal return; n equals the number of words read (0 for non-file-structured read, multiple of 256 for file-structured read). For example: If wcnt is a multiple of 256 and less than that number of words remain in the file, n is shortened to the number of words that remain in the file; for example if wcnt is 512 and only 256 words remain, i = 256. If wcnt is not a multiple of 256 and more than wcnt words remain in the file, n is rounded up to the next block; for example, if wcnt is 312 and more than 312 words remain, i = 512, but only 312 are read. If wcnt is not a multiple of 256 and less than wcnt words remain in the file, n equals a multiple of 256 that is the actual number of words being read.
= -1	Attempt to read past end-of-file; no words remain in the file.
= -2	Hardware error occurred on channel.
= -3	Specified channel is not open.

Example:

```
      INTEGER*2 BUFFER(256),RCODE,BLK
      .
      .
      .
      RCODE = IREAD(256,BUFFER,BLK,ICHAN)
      IF(RCODE+1) 1010,1000,10
C     IF NO ERROR, START HERE
10    .
      .
      .
      IF(IWAIT(ICHAN).NE.0) GOTO 1010
      .
      .
      .
1000  CONTINUE
C     END OF FILE PROCESSING
      .
      .
      .
      CALL EXIT            !NORMAL END OF PROGRAM
1010  STOP 'FATAL READ'
      END
```

IREADC

The IREADC function transfers a specified number of words from the indicated channel into memory. Control returns to the user program immediately after the IREADC function is initiated. When the operation is complete, the specified assembly language routine (crtn) is entered as an asynchronous completion routine.

Form: i = IREADC (wcnt,buff,blk,chan,crtn)

where:		
	wcnt	is the integer number of words to be transferred.
	buff	is the array to be used as the buffer. This array must contain at least wcnt words.
	blk	is the integer block number of the file to be read. The user program normally updates blk before it is used again.
	chan	is the integer specification for the RT-11 channel to be used.
	crtn	is the assembly language routine to be activated when the transfer is complete. This name must be specified in an EXTERNAL statement in the FORTRAN routine that issues the IREADC call.

Errors:

See Errors under IREAD.

Example:

```
        INTEGER*2 IBUF(256),RCODE,IBLK
        EXTERNAL RDCMP
        .
        .
        .
        RCODE=IREADC(256,IBUF,IBLK,ICHAN,RDCMP)
```

IREADF

The IREADF function transfers a specified number of words from the indicated channel into memory. Control returns to the user program immediately after the IREADF function is initiated. When the operation is complete, the specified FORTRAN subprogram (crtn) is entered as an asynchronous completion routine (see Section 4.2.1).

Form: i = IREADF (wcnt,buff,blk,chan,area,crtn)

where:		
	wcnt	is the integer number of words to be transferred.
	buff	is the array to be used as the buffer. This array must contain at least wcnt words.
	blk	is the integer block number of the file to be used. The user program normally updates blk before it is used again.
	chan	is the integer specification for the RT-11 channel to be used.
	area	is a four-word area to be set aside for link information; this area must not be modified by the FORTRAN program or swapped over by the USR. This area can be reclaimed by other FORTRAN completion functions when crtn has been activated.

crtn — is the FORTRAN routine to be activated on completion of the transfer. This name must be specified in an EXTERNAL statement in the routine that issues the IREADF call. The subroutine has two arguments:

SUBROUTINE crtn (iargl,iarg2)

iargl	is the channel status word (see Section 2.4.34) for the operation just completed. If bit 0 is set, a hardware error occurred during the transfer.
iarg2	is the octal channel number used for the operation just completed.

Errors:

See Errors under IREAD.

Example:

```
        INTEGER*2 DBLK(4),BUFFER(256),BLKNO
        DATA DBLK/3RDX0,3RINP,3RUT ,3RDAT/,BLKNO/0/
        EXTERNAL RCMPLT
        .
        .
        .
        ICHAN=IGETC()
        IF(ICHAN.LT.0) STOP 'NO CHANNEL AVAILABLE'
        IF(IFETCH(DBLK).NE.0) STOP 'BAD FETCH'
        IF(LOOKUP(ICHAN,DBLK).LT.0) STOP 'BAD LOOKUP'
        .
        .
        .
20      IF(IREADF(256,BUFFER,BLKNO,ICHAN,DBLK,RCMPLT).LT.0) GOTO 100
C       PERFORM OVERLAP PROCESSING
        .
        .
        .
C       SYNCHRONIZER
        CALL IWAIT(ICHAN)   !WAIT FOR COMPLETION ROUTINE TO RUN
        BLKNO=BLKNO+1       !UPDATE BLOCK NUMBER
        GOTO 20
        .
        .
        .
C       END OF FILE PROCESSING
100     CALL CLOSEC(ICHAN)
        CALL IFREEC(ICHAN)
        .
        .
        .
        CALL EXIT
        END
        SUBROUTINE RCMPLT(I,J)
C       THIS IS THE COMPLETION ROUTINE
        .
        .
        .
        RETURN
        END
```

IREADW

The IREADW function transfers a specified number of words from the indicated channel into memory. Control returns to the user program when the transfer is complete or when an error is detected.

Form: i = IREADW (wcnt,buff,blk,chan)

where:		
	wcnt	is the integer number of words to be transferred.
	buff	is the array to be used as the buffer. This array must contain at least wcnt words.
	blk	is the integer block number of the file to be read. The user program normally updates blk before it is used again.
	chan	is the integer specification for the RT-11 channel to be used.

Errors:

See Errors under IREAD.

Example:

```
      INTEGER*2 IBUF(1024)
      .
      .
      .
      ICODE=IREADW(1024,IBUF,IBLK,ICHAN)
      IF(ICODE.EQ.-1) GOTO 100      !END OF FILE PROCESSING AT 100
      IF(ICODE.LT.-1) GOTO 200      !ERROR PROCESSING AT 200
C
C     MODIFY BLOCKS
C
      .
      .
      .
C
C     WRITE THEM OUT
C
      ICODE=IWRITW(1024,IBUF,IBLK,ICHAN)
```

IRENAM

4.3.41 IRENAM

The IRENAM function causes an immediate change of the name of a specified file. An error occurs if the channel specified is already open.

Form: i = IRENAM (chan,dblk)

where:		
	chan	is the integer specification for the RT-11 channel to be used for the operation.

dblk — is the eight-word area specifying the name of the existing file and the new name to be assigned. If considered as an eight-element INTEGER*2 array, dblk has the form:

dblk(1)-dblk(4)	specify the Radix-50 file descriptor for the old file name.
dblk(5)-dblk(8)	specify the Radix-50 file descriptor for the new file name.

NOTE

The arguments of IRENAM must be positioned so that the USR does not swap over them.

If a file already exists with the same name as the new file on the indicated device, it is deleted. The specified channel is left closed when the **IRENAM** is complete. IRENAM requires that the handler to be used be resident at the time the IRENAM is issued. If it is not, a monitor error occurs. The device names specified in the file descriptors must be the same.

For more information on renaming files, see the assembly language **.RENAME** request, Section 2.4.

Errors:

i =	0	Normal return.
=	1	Specified channel is already open.
=	2	Specified file was not found. Example:

```
REAL*8 NAME(2)
DATA NAME/12RDKOFTN2  DAT,12RDKOFTN2  OLD/
.
.
.
ICHAN=IGETC()
IF(ICHAN.LT.0) STOP 'NO CHANNEL'
CALL IRENAM(ICHAN,NAME)      !PRESERVE OLD DATA FILE
CALL IFREEC(ICHAN)
```

IREOPN

4.3.42 IREOPN

The IREOPN function reassociates a specified channel with a file on which an ISAVES was performed. The ISAVES/IREOPN combination is useful when a large number of files must be operated on at one time. Necessary files can be opened with LOOKUP and their status preserved with ISAVES. When data is required from a file, an IREOPN enables the program to read from the file. The IREOPN need not be done on the same channel as the original LOOKUP and ISAVES.

Form: i = IREOPN (chan,cblk)

where:	chan	is the integer specification for the RT-11 channel to be associated with the reopened file. This channel must be initially inactive.
	cblk	is the five-word block where the channel status information was stored by a previous ISAVES. This block, considered as a five-element INTEGER*2 array, has the following format:

cblk(1)	Channel status word (see Section 2.4).
cblk(2)	Starting block number of the file; zero for non-file-structured devices.
cblk(3)	Length of file (in 256-word blocks).
cblk(4)	(Reserved for future use.)
cblk(5)	Two information bytes. Even byte: I/O count of the number of requests made on this channel. Odd byte: unit number of the device associated with the channel.

Errors:

i = 0	Normal return.
= 1	Specified channel is already in use.

Example:

```
INTEGER*2 SAVES(5,10)
DATA ISVPTR/1/
.
.
.
CALL ISAVES(ICHAN,SAVES(1,ISVPTR))
.
.
.
CALL IREOPN(ICHAN,SAVES(1,ISVPTR))
```

ISAVES

4.3.43 ISAVES

The ISAVES function stores five words of channel status information into a user-specified array. These words contain all the information that RT-11 requires to completely define a file. When an ISAVES is finished, the data words are placed in memory and the specified channel is closed and is again available for use. When the saved channel data is required, the IREOPN function (Section 4.3.42) is used.

ISAVES can be used only if a file was opened with a LOOKUP call (see Section 4.3.70). If IENTER was used, ISAVES is illegal and returns an error. Note that ISAVES is not legal on magtape or cassette files.

Form: i = ISAVES (chan,cblk)

where:	chan	is the integer specification for the RT-11 channel whose status is to be saved.
	cblk	is a five-word block into which the channel status information describing the open file is stored. See Section 4.3.42 for the format of this block.

The ISAVES/IREOPN combination is very useful, but care must be exercised when using it. In particular, the following cases should be avoided.

1. If an ISAVES is performed on a file and the same file is then deleted before it is reopened, the space occupied by the file becomes available as an empty space which could then be used by the IENTER function. If this sequence occurs, the contents of the file whose status was supposedly saved changes.

2. Although the handler for the required peripheral need not be in memory for execution of an IREOPN, a fatal error is generated if the handler is not in memory when an IREAD or IWRITE is executed.

Errors:

i = 0	Normal return.
= 1	The specified channel is not currently associated with any file.
= 2	The file was opened with an IENTER call; an ISAVES is illegal.

Example:

```
        INTEGER*2 BLK(5)
        .
        .
        .
        IF(ISAVES(ICHAN,BLK).NE.0) STOP 'ISAVES ERROR'
```

ISCHED

4.3.44 ISCHED

The ISCHED function schedules a specified FORTRAN subroutine to be run as an asynchronous completion routine at a specified time of day. Support for ISCHED in SJ also requires timer support.

Form: i = ISCHED (hrs,min,sec,tick,area,id,crtn)

where:	hrs	is the integer number of hours.
	min	is the integer number of minutes.
	sec	is the integer number of seconds.
	tick	is the integer number of ticks (1/60 of a second on 60-cycle clocks; 1/50 of a second on 50-cycle clocks).

area	is a four-word area that must be provided for link information; this area must never be modified by the FORTRAN program, and the USR must not swap over it. This area can be reclaimed by other FORTRAN completion functions when crtn has been activated.
id	is the identification integer to be passed to the routine being scheduled.
crtn	is the name of the FORTRAN subroutine to be entered at the time of day specified. This name must be specified in an EXTERNAL statement in the FORTRAN routine that issues the ISCHED call. The subroutine has one argument. For example:

```
SUBROUTINE crtn(id)
INTEGER id
```

When the routine is entered, the value of the integer argument is the value specified for id in the appropriate ISCHED call.

Notes:

1. The scheduling request made by this function can be cancelled at a later time by an ICMKT function call.
2. If the system is busy, the actual time of day that the completion routine is run can be greater than the requested time of day.
3. A FORTRAN subroutine can periodically reschedule itself by issuing its own ISCHED or ITIMER calls from within the routine.
4. ISCHED requires a queue element; this should be considered when the IQSET function (Section 4.3.33) is executed.

Errors:

i = 0	Normal return.
= 1	No queue elements available; unable to schedule request.

Example:

```
      INTEGER*2 LINK(4)                  !LINKAGE AREA
      EXTERNAL NOON                      !NAME OF ROUTINE TO RUN
      .
      .
      .
      I=ISCHED(12,0,0,0,LINK,0,NOON)     !RUN SUBR NOON AT 12 PM
      .
      .   (rest of main program)
      .
      END
      SUBROUTINE NOON(ID)
C
C     THIS ROUTINE WILL TERMINATE EXECUTION AT LUNCHTIME,
C     IF THE JOB HAS NOT COMPLETED BY THAT TIME.
C
      STOP     'ABORT JOB -- LUNCHTIME'
      END
```

4.3.45 ISDAT/ISDATC/ISDATF/ISDATW (FB and XM Only)

These functions are used with the IRCVD/IRCVDC/IRCVDF, and IRCVDW calls to allow message transfers under the FB or monitor. Note that the buffer containing the message should not be modified or reused until the message has been received by the other job. These functions require a queue element; this should be considered when the IQSET function (see Section 4.3.37) is executed.

ISDAT

The ISDAT function transfers a specified number of words from one job to the other. Control returns to the user program immediately after the transfer is queued. This call is used with the MWAIT routine (see Section 4.3.80).

Form: i = ISDAT (buff,wcnt)

where:	buff	is the array containing the data to be transferred.
	wcnt	is the integer number of data words to be transferred.

Errors:

i = 0	Normal return.
= 1	No other job currently exists in the system.

Example:

```
      INTEGER*2 MSG(40)
      .
      .
      .
      CALL ISDAT(MSG,40)
      .
      .
      .
      CALL MWAIT
C     PUT NEW MESSAGE IN BUFFER
```

ISDATC

The ISDATC function transfers a specified number of words from one job to another. Control returns to the user program immediately after the transfer is queued. When the other job accepts the message through a receive data request, the specified assembly language routine (crtn) is activated as an asynchronous completion routine.

Form: i = ISDATC (buff,wcnt,crtn)

where:	buff	is the array containing the data to be transferred.

wcnt	is the integer number of data words to be transferred.
crtn	is the name of an assembly language routine to be activated on completion of the transfer. This name must be specified in an EXTERNAL statement in the FORTRAN routine that issues the ISDATC call.

Errors:

i = 0	Normal return.
= 1	No other job currently exists in the system.

Example:

```
INTEGER*2 MSG(40)
EXTERNAL RTN
.
.
.
CALL ISDATC(MSG,40,RTN)
```

ISDATF

The ISDATF function transfers a specified number of words from one job to the other. Control returns to the user program immediately after the transfer is queued and execution continues. When the other job accepts the message through a receive data request, the specified FORTRAN subprogram (crtn) is activated as an asynchronous completion routine (see Section 4.2.1).

Form: i = ISDATF (buff,wcnt,area,crtn)

where:	buff	is the array containing the data to be transferred.
	wcnt	is the integer number of data words to be transferred.
	area	is a four-word area to be set aside for link information; this area must not be modified by the FORTRAN program and the USR must not swap over it. This area can be reclaimed by other FORTRAN completion functions when crtn has been activated.
	crtn	is the name of a FORTRAN routine to be activated on completion of the transfer. This name must be specified in an EXTERNAL statement in the FORTRAN routine that issues the ISDATF call.

Errors:

i = 0	Normal return.
= 1	No other job currently exists in the system.

Example:

```
INTEGER*2 MSG(40),SPOT(4)
EXTERNAL RTN
.
.
.
CALL ISDATF(MSG,40,SPOT,RTN)
```

ISDATW

The ISDATW function transfers a specified number of words from one job to the other. Control returns to the user program when the other job has accepted the data through a receive data request.

Form: i = ISDATW (buff,wcnt)

where:		
	buff	is the array containing the data to be transferred.
	wcnt	is the integer number of data words to be transferred.

Errors:

i = 0	Normal return.
= 1	No other job currently exists in the system.

Example:

```
        INTEGER*2 MSG(40)
        .
        .
        .
        IF (ISDATW(MSG,40).NE.0) STOP 'FOREGROUND JOB NOT RUNNING'
```

ISLEEP

4.3.46 ISLEEP

The ISLEEP function suspends the main program execution of a job for a specified amount of time. The specified time is the sum of hours, minutes, seconds, and ticks specified in the ISLEEP call. All completion routines continue to execute. Support for ISLEEP in SJ also requires timer support.

Form: i = ISLEEP (hrs,min,sec,tick)

where:		
	hrs	is the integer number of hours.
	min	is the integer number of minutes.
	sec	is the integer number of seconds.
	tick	is the integer number of ticks (1/60 of a second on 60-cycle clocks; 1/50 of a second on 50-cycle clocks).

Notes:

1. ISLEEP requires a queue element; this should be considered when the IQSET function (Section 4.3.37) is executed.

2. If the system is busy, the time that execution is suspended may be greater than that specified.

Errors:

i = 0	Normal return.
= 1	No queue element available.

Example:

```
        .
        .
        .
        CALL IQSET(2)
        .
        .
        .
        CALL ISLEEP(0,0,0,4)      !GIVE BACKGROUND JOB SOME TIME
```

ISPFN/ISPFNC/ISPFNF/ISPFNW

4.3.47 ISPFN/ISPFNC/ISPFNF/ISPFNW

These functions are used in conjunction with special functions to various handlers. They provide a means of doing device-dependent functions, such as rewind and backspace, to those devices. If ISPFN function calls are made to any other devices, the function call is ignored. For more information on programming for specific devices, see Section 1.4.7.

To use these functions, the handler must be in memory and a channel associated with a file via a non-file-structured LOOKUP call. These functions require a queue element; this should be considered when the IQSET function (Section 4.3.37) is executed.

ISPFN

The ISPFN function queues the specified operation and immediately returns control to the user program. The IWAIT function can be used to ensure completion of the operation.

Form: i = ISPFN (code,chan[,wcnt,buff,blk])

where:	code	is the integer numeric code of the function to be performed (see Table 4-2).
	chan	is the integer specification for the RT-11 channel to be used for the operation.
	wcnt	is the integer number of data words in the operation.* Default value is 0. In magtape operations, it specifies the number of records to space forward or backward. For a backspace operation (wcnt=0), the tape drive backspaces to a tape mark or to the beginning-of-tape. For a forward space operation (wcnt=0), the tape drive forward spaces to a tape mark or the end-of-tape.
	buff	is the array to be used as the data buffer.* Default value is 0.
	blk	is the integer block number of the file to be operated upon.* Default value is 0.

* These parameters are optional with some ISPFUN calls, depending on the particular function.

When this argument is supplied by magtape, it is the address of a four-word error and status block used for returning the exception conditions. The four words must be initialized to zero.

The error and status block must always be mapped when running in the XM monitor, and the USR must not swap over it. To obtain the address of the error block execute the following instructions:

```
INTEGER*2      ERRADR, ERRBLK(4)
DATA ERRBLK    /0,0,0,0,/
      .
      .
      .
      .
ERRADR = IADDR (ERRBLK) !GET THE ADDRESS OF  THE  4-WORD ERROR BLOCK
ICODE = ISPFN (CODE,ICHAN,WDCT,BUF,ERRADR)
```

Table 4-2
Special Function Codes (Octal)

Function	MT,MM	CT	DX	DM	DY	DL
Read absolute			377	377	377	377
Write absolute			376	376	376	376
Write absolute with deleted data			375		375	
Forward to last file		377				
Forward to last block		376				
Forward to next file		375				
Forward to next block		374				
Rewind to load point	373	373				
Write file gap		372				
Write end-of-file	377					
Forward 1 record	376					
Backspace 1 record	375					
Initialize the bad block replacement table				374		374
Write with extended record gap	374					
Offline	372					
Return volume size				373	373	373

Errors:

i = 0	Normal return.
= 1	Attempt to read or write past end-of-file.
= 2	Hardware error occurred on channel.
= 3	Channel specified is not open.

Example:

```
CALL ISPFN("373,ICHAN)      !REWIND
```

ISPFNC

The ISPFNC function queues the specified operation and immediately returns control to the user program. When the operation is complete, the specified assembly language routine (crtn) is entered as an asynchronous completion routine.

Form: i = ISPFNC (code,chan,wcnt,buff,blk,crtn)

where:	code	is the integer numeric code of the function to be performed (see Table 4-2).
	chan	is the integer specification for the RT-11 channel to be used for the operation.
	wcnt	is the integer number of data words in the operation. This argument must be 0 if not required.
	buff	is the array to be used as the data buffer. This argument must be 0 if not required.
	blk	is the integer block number of the file to be operated upon. This argument must be 0 if not required.

When this argument is supplied by magtape, it is the address of a four-word error and status block used for returning the exception conditions. The four words must be initialized to 0.

The error and status block must always be mapped when running in the XM monitor, and the USR must not swap over it. To obtain the address of the error block execute the following instructions:

```
INTEGER*2       ERRADR, ERRBLK(4)
DATA ERRBLK     /0,0,0,0,/
     .
     .
     .
     .
ERRADR = IADDR (ERRBLK) !GET THE ADDRESS OF  THE  4-WORD ERROR BLOCK
ICODE = ISPFN (CODE,ICHAN,WDCT,BUF,ERRADR)
```

crtn — is the name of an assembly language routine to be activated on completion of the operation. This name must be specified in an EXTERNAL statement in the FORTRAN routine that issues the ISPFNC call.

Errors:

i = 0	Normal return.
= 1	Attempt to read or write past end-of-file.
= 2	Hardware error occurred on channel.
= 3	Channel specified is not open.

ISPFNF

The ISPFNF function queues the specified operation and immediately returns control to the user program. When the operation is complete, the specified FORTRAN subprogram (crtn) is entered as an asynchronous completion routine.

Form: i = ISPFNF (code,chan,wcnt,buff,blk,area,crtn)

where:

code
: is the integer numeric code of the function to be performed (see Table 4-2).

chan
: is the integer specification for the RT-11 channel to be used for the operation.

wcnt
: is the integer number of data words in the operation. This argument must be 0 if not required.

buff
: is the array to be used as the data buffer. This argument must be 0 if not required.

blk
: is the integer block number of the file to be operated upon. This argument must be 0 if not required.

When this argument is supplied by magtape, it is the address of a four-word error and status block used for returning the exception conditions. The four words must be initialized to 0.

The error and status block must always be mapped when running in the XM monitor, and the USR must not swap over it. To obtain the address of the error block execute the following instructions:

```
INTEGER*2      ERRADR, ERRBLK(4)
DATA ERRBLK    /0,0,0,0,/
     .
     .
     .
     .
ERRADR = IADDR (ERRBLK) !GET THE ADDRESS OF  THE  4-WORD ERROR BLOCK
ICODE = ISPFN (CODE,ICHAN,WDCT,BUF,ERRADR)
```

area
: is a four-word area to be set aside for linkage information; this area must not be modified by the FORTRAN program and the USR must not swap over it. This area can be reclaimed by other FORTRAN completion functions when crtn has been activated.

crtn
: is the name of a FORTRAN routine to be activated on completion of the operation. This name must be specified in an EXTERNAL statement in the FORTRAN routine that issues the ISPFNF call. The subroutine has two arguments:

SUBROUTINE crtn (iarg1,iarg2)

iarg1	is the channel status word (see Section 2.4) for the operation just completed. If bit 0 is set, a hardware error occurred during the transfer.
iarg2	is the channel number used for the operation just completed.

Errors:

i = 0	Normal return.
= 1	Attempt to read or write past end-of-file.
= 2	Hardware error occurred on channel.
= 3	Channel specified is not open.

Example:

```
        REAL*4 MTNAME(2),AREA(2)
        DATA MTNAME/3RMT0,0./
        EXTERNAL DONSUB
        .
        .
        .
        I=IGETC()                   !ALLOCATE CHANNEL
        CALL IFETCH(MTNAME)         !FETCH MT HANDLER
        CALL LOOKUP(I,MTNAME)       !NON-FILE-STRUCTURED LOOKUP ON MT0
        IERR=ISPFNF("373,I,0,0,0,AREA,DONSUB)    !REWIND MAGTAPE
        .
        .
        .
        END
        SUBROUTINE DONSUB
C
C       RUNS WHEN MT0 HAS BEEN REWOUND
C
        .
        .
        .
        END
```

ISPFNW

The ISPFNW function queues the specified operation and returns control to the user program when the operation is complete.

Form: i = ISPFNW (code,chan[,wcnt,buff,blk])

where:		
	code	is the integer numeric code of the function to be performed (see Table 4-2).
	chan	is the integer specification for the RT-11 channel to be used for the operation.
	wcnt	is the integer number of data words in the operation.*
	buff	is the array to be used as the data buffer.*
	blk	is the integer block number of the file to be operated upon.*

* These parameters are optional with some ISPFUN calls, depending on the particular function.

When this argument is supplied by magtape, it is the address of a four-word error and status block used for returning the exception conditions. The four words must be initialized to 0.

The error and status block must always be mapped when running in the XM monitor, and the USR must not swap over it. To obtain the address of the error block execute the following instructions:

```
INTEGER*2      ERRADR, ERRBLK(4)
DATA ERRBLK    /0,0,0,0,/
     .
     .
     .
     .
ERRADR = IADDR (ERRBLK) !GET THE ADDRESS OF  THE  4-WORD ERROR BLOCK
ICODE = ISPFN (CODE,ICHAN,WDCT,BUF,ERRADR)
```

Errors:

i = 0	Normal return.
= 1	Attempt to read or write past end-of-file.
= 2	Hardware error occurred on channel.
= 3	Channel specified is not open.

Example:

```
        INTEGER*2 BUF(65),TRACK,SECTOR,DBLK(4)
        DATA DBLK/3RDX0,0,0,0/
        .
        .
        .
        ICHAN=IGETC()
        IF(ICHAN.LT.0) STOP 'NO CHANNEL AVAILABLE'
        IF(LOOKUP(ICHAN,DBLK).LT.0) STOP 'BAD LOOKUP'
        .
        .
        .
C       READ AN ABSOLUTE TRACK AND SECTOR FROM THE FLOPPY
C
        ICODE=ISPFNW('377,ICHAN,TRACK,BUF,SECTOR)
C
C       BUF(1) IS THE DELETED DATA FLAG
C       BUF(2-65) IS THE DATA
```

ISPY

4.3.48 ISPY

The ISPY function returns the integer value of the word at a specified offset from the RT-11 resident monitor. This subroutine uses the .GVAL programmed request to return fixed monitor offsets. (See Section 2.2.6 for information on fixed offset references.)

Form: i = ISPY (ioff)

where: ioff is the offset (from the base of RMON) to be examined.

Function Result:

The function result (i) is set to the value of the word examined.

Example:

```
C
C     BRANCH TO 200 IF RUNNING UNDER FB MONITOR
C
      IF(ISPY("300).AND.1) GOTO 200
C
C     WORD AT OCTAL 300 FROM RMON IS
C     THE CONFIGURATION WORD.
```

ITIMER

4.3.49 ITIMER

The ITIMER function schedules a specified FORTRAN subroutine to be run as an asynchronous completion routine after a specified time interval has elapsed. This request is supported by SJ when the timer support option is included during system generation.

Form: i = ITIMER (hrs,min,sec,tick,area,id,crtn)

where:	hrs	is the integer number of hours.
	min	is the integer number of minutes
	sec	is the integer number of seconds.
	tick	is the integer number of ticks (1/60 of a second on 60-cycle clocks; 1/50 of a second on 50-cycle clocks).
	area	is a four-word area that must be provided for link information; this area must never be modified by the FORTRAN program, and the USR must never swap over it. This area can be reclaimed by other FORTRAN completion functions when crtn has been activated.
	id	is the identification integer to be passed to the routine being scheduled.
	crtn	is the name of the FORTRAN subroutine to be entered when the specified time interval elapses. This name must be specified in an EXTERNAL statement in the FORTRAN routine that references ITIMER. The subroutine has one argument. For example:

```
SUBROUTINE crtn(id)
INTEGER id
```

When the routine is entered, the value of the integer argument is the value specified for id in the appropriate ITIMER call.