Notes:

1. This function can be cancelled at a later time by an ICMKT function call.

2. If the system is busy, the actual time interval at which the completion routine is run can be greater than the time interval requested.

3. FORTRAN subroutines can periodically reschedule themselves by issuing ISCHED or ITIMER calls.

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

For more information on scheduling completion routines, see Section 4.2.1 and the assembly language .MRKT request, Section 2.4.

Errors:

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

Example:

```
        INTEGER*2 AREA(4)
        EXTERNAL WATCHD
        .
        .
        .
C       IF THE CODE FOLLOWING ITIMER DOES NOT REACH THE ICMKT CALL
C       IN 12 MINUTES, WATCH DOG COMPLETION ROUTINE WILL BE
C       ENTERED WITH ID OF 3
C
        CALL ITIMER(0,12,0,0,AREA,3,WATCHD)
        .
        .
        .
        CALL ICMKT(3,AREA)
        .
        .
        .
        END
        SUBROUTINE WATCHD(ID)
C
C       THIS IS CALLED AFTER 12 MINUTES
        .
        .
        .
        RETURN
        END
```

4.3.50 ITLOCK (FB and XM Only)

The ITLOCK function is used in an FB or XM system to attempt to gain ownership of the USR. It is similiar to LOCK (Section 4.3.69) in that if successful, the user job returns with the USR in memory. However, if a job attempts to LOCK the USR while the other job is using it, the requesting job is suspended until the USR is free. With ITLOCK, if the USR is not available, control returns immediately and the lock failure is indicated. ITLOCK cannot be called from a completion or interrupt routine.

Form: i = ITLOCK()

For further information on gaining ownership of the USR, see the assembly language .TLOCK request, Section 2.4.

Errors:

i	= 0	Normal return.
	= 1	USR is already in use by another job.

Example:

```
        IF(ITLOCK().NE.0) GOTO 100      !GOTO 100 IF USR BUSY
```

ITTINR

4.3.51 ITTINR

The ITTINR function transfers a character from the console terminal to the user program. If no characters are available, return is made with an error flag set.

Form: i = ITTINR()

If the function result (i) is less than 0 when execution of the ITTINR function is complete, it indicates that no character was available; the user has not yet typed a valid line. Under the FB or XM monitor, ITTINR does not return a result of less than zero unless bit 6 of the job status word was on when the request was issued.

There are two modes of doing console terminal input. The mode is governed by bit 12 of the job status word (JSW). The JSW is at octal location 44. If bit 12 equals 0, normal I/O is performed. In this mode, the following conditions apply:

1. The monitor echoes all characters typed.
2. CTRL/U and RUBOUT perform line deletion and character deletion, respectively.
3. A carriage return, line feed, CTRL/Z, or CTRL/C must be struck before characters on the current line are available to the program. When one of these is typed, characters on the line typed are passed one by one to the user program. Both carriage return and line feed are passed to the program.

If bit 12 equals 1, the console is in special mode. The effects are:

1. The monitor does not echo characters typed except for CTRL/C and CTRL/O.
2. CTRL/U and RUBOUT do not perform special functions.
3. Characters are immediately available to the program.
4. No ALTMODE conversion is done.

In special mode, the user program must echo the characters desired. However, CTRL/C and CTRL/O are acted on by the monitor in the usual way. Bits 12 and 14 in the JSW must be set by the user program if special console mode or lower case characters are desired (see the example under Section 4.3.35). These bits are cleared when control returns to RT-11.

NOTE

> To set and/or clear bits in the JSW, do an IPEEK and then an IPOKE. In special terminal mode (JSW bit 12 set), normal FORTRAN formatted I/O from the console is undefined.

In the FB or XM monitor, CTRL/F and CTRL/B are not affected by the setting of bit 12. The monitor always acts on these characters if the SET TT FB command is in effect.

Under the FB or XM monitor, if a terminal input request is made and no character is available, job execution is suspended until a character is ready. If a program really requires execution to continue and ITTINR to return a result of less than zero, it must turn on bit 6 of the JSW before the ITTINR. Bit 6 is cleared when a program terminates.

NOTE

> If a foreground job has characters in the TT output buffer, they are not output under the following conditions:
>
> (1) If a background job is doing output to the console TT, the foreground job cannot output characters from its buffer until the background job outputs a line feed character. This can be troublesome if the console device is a graphics terminal, and the background job is doing graphic output without sending any line feeds.
>
> (2) If no background job is running (that is, KMON is in control of background), the foreground job cannot output its characters until the user types a carriage return or a line feed. In the former case, KMON gets control again and locks out foreground output as soon as the foreground output buffer is empty.

Function Results:

i >0	Normal return; character read.
<0	Error return; no character available.

Example:

```
ICHAR=ITTINR()              !READ A CHARACTER FROM THE CONSOLE
IF(ICHAR.LT.0) GOTO 100     !CHARACTER NOT AVAILABLE
```

ITTOUR

4.3.52 ITTOUR

The ITTOUR function transfers a character from the user program to the console terminal if there is room for the character in the monitor buffer. If it is not currently possible to output a character, an error flag is returned.

Form: i = ITTOUR (char)

where:	char	is the character to be output, right-justified in the integer (can be LOGICAL*1 entity if desired).

If the function result (i) is 1 when execution of the ITTOUR function is complete, it indicates that there is no room in the buffer and that no character was output. Under the FB or XM monitor, ITTOUR normally does not return a result of 1. Instead, the job is blocked until room is available in the output buffer. If a job really requires execution to continue and a result of 1 to be returned, it must turn on bit 6 of the JSW (location 44) before issuing the request.

The ITTINR and ITTOUR have been supplied as a help to those users who do not wish to suspend program execution until a console operation is complete. With these modes of I/O, if a no-character or no-room condition occurs, the user program can continue processing and try the operation again at a later time.

Errors:

i = 0	Normal return; character was output.
= 1	Error return; ring buffer is full.

Example:

```
      DO 20 I=1,5
10    IF(ITTOUR("007).NE.0) GOTO 10      !RING BELL 5 TIMES
20    CONTINUE
```

ITWAIT

4.3.53 ITWAIT (FB and XM Only)

The ITWAIT function suspends the main program execution of the current job for a specified time interval. All completion routines continue to execute.

Form: i = ITWAIT (itime)

where:	itime	is the two-word internal format time interval.
		itime (1) is the high-order time.
		itime (2) is the low-order time.

Notes:

1. ITWAIT 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 actual time interval for which execution is suspended can be greater than the time interval specified.

Errors:

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

Example:

```
        INTEGER*2 TIME(2)
        .
        .
        .
        CALL ITWAIT(TIME)      !WAIT FOR TIME TIME
```

IUNTIL

4.3.54 IUNTIL (FB and XM Only)

The IUNTIL function suspends main program execution of the job until the time of day specified. All completion routines continue to run.

Form: i = IUNTIL (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. IUNTIL 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 actual time of day that the program resumes execution can be later than that requested.

Errors:

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

Example:

```
    C   TAKE A LUNCH BREAK
        CALL IUNTIL(13,0,0,0)      !START UP AGAIN AT 1 P.M.
```

IWAIT

4.3.55 IWAIT

The IWAIT function suspends execution of the main program until all input/output operations on the specified channel are complete. This function is used with IREAD, IWRITE, and ISPFN calls. Completion routines continue to execute.

Form: i = IWAIT (chan)

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

For further information on suspending execution of the main program, see the assembly language .WAIT request, Section 2.4.

Errors:

i = 0	Normal return.
= 1	Channel specified is not open.
= 2	Hardware error occurred during the previous I/O operation on this channel.

Example:

```
        IF(IWAIT(ICHAN).NE.0) CALL IOERR(4)
```

IWRITC/IWRITE/IWRITF/IWRITW

4.3.56 IWRITC/IWRITE/IWRITF/IWRITW

These functions transfer a specified number of words from memory to the specified channel. The IWRITE functions require queue elements; this should be considered when the IQSET function (Section 4.3.37) is executed.

IWRITC

The IWRITC function transfers a specified number of words from memory to the specified channel. The request is queued and control returns to the user program. When the transfer is complete, the specified assembly language routine (crtn) is entered as an asynchronous completion routine.

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

where:	wcnt	is the relative integer number of words to be transferred.
	buff	is the array to be used as the output buffer.
	blk	is the relative integer block number of the file to be written. 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.
crtn	is the name of the assembly language routine to be activated upon completion of the transfer. This name must be specified in an EXTERNAL statement in the FORTRAN routine that issues the IWRITC call.

NOTE

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

Errors:

i = n	Normal return; n equals the number of words written, rounded to a multiple of 256 (0 for non-file-structured writes).

NOTE

If the word count returned is less than that requested, an implied end-of-file has occurred although the normal return is indicated.

= -1	Attempt to write past end-of-file; no more space is available in the file.
= -2	Hardware error occurred.
= -3	Channel specified is not open.

Example:

```
INTEGER*2 IBUF(256)
EXTERNAL CRTN
.
.
.
ICODE=IWRITC(256,IBUF,IBLK,ICHAN,CRTN)
```

IWRITE

The IWRITE function transfers a specified number of words from memory to the specified channel. Control returns to the user program immediately after the request is queued. No special action is taken upon completion of the operation.

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

where:	wcnt	is the integer number of words to be transferred.
	buff	is the array to be used as the output buffer.

blk	is the integer block number of the file to be written. 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 the errors under IWRITC.

Example:

See the example under IREAD, Section 4.3.40.

IWRITF

The IWRITF function transfers a number of words from memory to the specified channel. The transfer request is queued an: control returns to the user program. When the operation is complete, the specified FORTRAN subprogram (crtn) is entered as an asynchronous completion routine (see Section 4.2.1).

Form: i = IWRITF (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 output buffer.
	blk	is the integer block number of the file to be written. 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 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 the FORTRAN routine to be activated upon completion of the transfer. This name must be specified in an EXTERNAL statement in the FORTRAN routine that issues the IWRITF 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:

See the errors under IWRITC.

Example:

See the example under IREADF, Section 4.3.40.

IWRITW

The IWRITW function transfers a specified number of words from memory to the specified channel. Control returns to the user program when the transfer is complete.

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

where:	wcnt	is the integer number of words to be transferred.
	buff	is the array to be used as the output buffer.
	blk	is the integer block number of the file to be written. 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 the errors under IWRITC.

Example:

See the example under IREADW, Section 4.3.40.

4.3.57 JADD

The JADD function computes the sum of two INTEGER*4 values.

Form: i = JADD (jopr1,jopr2,jres)

where:	jopr1	is an INTEGER*4 variable.
	jopr2	is an INTEGER*4 variable.
	jres	is an INTEGER*4 variable that receives the sum of jopr1 and jopr2.

Function Results:

i = -2	An overflow occurred while computing the result.
= -1	Normal return; the result is negative.
= 0	Normal return; the result is zero.
= 1	Normal return; the result is positive.

Example:

```
        INTEGER*4 JOP1,JOP2,JRES
        .
        .
        .
        IF(JADD(JOP1,JOP2,JRES).EQ.-2) GOTO 100
```

JAFIX

4.3.58 JAFIX

The JAFIX function converts a REAL*4 value to INTEGER*4.

Form: i = JAFIX (asrc,jres)

where:		
	asrc	is a REAL*4 variable, constant, or expression to be converted to INTEGER*4.
	jres	is an INTEGER*4 variable that is to contain the result of the conversion.

Function Results:

i = -2	An overflow occurred while computing the result.
= -1	Normal return; the result is negative.
= 0	Normal return; the result is zero.
= 1	Normal return; the result is positive.

Example:

```
      INTEGER*4 JOP1
C     READ A LARGE INTEGER FROM THE TERMINAL
      ACCEPT 99,A
99    FORMAT (F15.0)
      IF(JAFIX(A,JOP1).EQ.-2) GOTO 100
      .
      .
      .
```

JCMP

4.3.59 JCMP

The JCMP function compares two INTEGER*4 values and returns an INTEGER*2 value that reflects the signed comparison result.

Form: i = JCMP (jopr1,jopr2)

where:		
	jopr1	is the INTEGER*4 variable or array element that is the first operand in the comparison.
	jopr2	is the INTEGER*4 variable or array element that is the second operand in the comparison.

Function Result:

```
i = -1    If joprl is less than jopr2
  = 0     If joprl is equal to jopr2
  = 1     If joprl is greater than jopr2
```

Example:

```
          INTEGER*4 JOPX,JOPY
          .
          .
          .
          IF(JCMP(JOPX,JOPY)) 10,20,30
```

JDFIX

4.3.60 JDFIX

The JDFIX function converts a REAL*8 (DOUBLE PRECISION) value to INTEGER*4.

Form: i = JDFIX (dsrc,jres)

where:	dsrc	is a REAL*8 variable, constant, or expression to be converted to INTEGER*4.
	jres	is an INTEGER*4 variable to contain the conversion result.

Function Results:

```
i = -2       An overflow occurred while computing the result.
  = -1       Normal return;  the result is negative.
  = 0        Normal return;  the result is zero.
  = 1        Normal return;  the result is positive.
```

Example:

```
          INTEGER*4 JNUM
          REAL*8 DPNUM
          .
          .
          .
     20   TYPE 98
     98   FORMAT(' ENTER POSITIVE INTEGER')
          ACCEPT 99,DPNUM
     99   FORMAT(F20.0)
          IF(JDFIX(DPNUM,JNUM).LT.0) GOTO 20
          .
          .
          .
```

JDIV

4.3.61 JDIV

The JDIV function computes the quotient of two INTEGER*4 values.

Form: i = JDIV (joprl,jopr2,jres[,jrem]) where: joprl is an INTEGER*4 variable that is the dividend of the operation.

jopr2	is an INTEGER*4 variable that is divisor of joprl.
jres	is an INTEGER*4 variable that receives the quotient of the operation (that is, jres=joprl/jopr2).
jrem	is an INTEGER*4 variable that receives the remainder of the operation. The sign is the same as that for joprl.

Function Results:

i = -3	An attempt was made to divide by 0.
= -2	(not used)
= -1	Normal return; the quotient is negative.
= 0	Normal return; the quotient is 0.
= 1	Normal return; the quotient is positive.

Example:

```
INTEGER*4 JN1,JN2,JQUO
.
.
.
CALL JDIV(JN1,JN2,JQUO)
.
.
.
```

JICVT

4.3.62 **JICVT**

The JICVT function converts a specified INTEGER*2 value to INTEGER*4.

Form: i = JICVT (isrc,jres)

Where:	isrc	is the INTEGER*2 quantity to be converted.
	jres	is the INTEGER*4 variable or array element to receive the result.

Function Results:

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 JVAL
CALL JICVT(478,JVAL)      !FORM A 32-BIT CONSTANT
```

4.3.63 JJCVT

The JJCVT function interchanges words of an INTEGER*4 value to form an internal format time or vice versa. This procedure is necessary when the INTEGER*4 variable is to be used as an argument in a timer-support function such as ITWAIT. When a two-word internal format time is specified to a function such as ITWAIT, it must have the high-order time as the first word and the low-order time as the second word.

Form: CALL JJCVT (jsrc)

where:	jsrc	is the INTEGER*4 variable whose contents are to be interchanged.

Errors:

None.

Example:

```
        INTEGER*4 TIME
        .
        .
        .
        CALL GTIM(TIME)      !GET TIME OF DAY
        CALL JJCVT(TIME)     !TURN IT INTO INTEGER*4 FORMAT
```

4.3.64 JMOV

The JMOV function assigns the value of an INTEGER*4 variable to another INTEGER*4 variable and returns the sign of the value moved.

Form: i = JMOV (jsrc,jdest)

where:	jsrc	is the INTEGER*4 variable whose contents are to be moved.
	jdest	is the INTEGER*4 variable that is the target of the assignment.

Function Result:

The value of the function is an INTEGER*2 value that represents the sign of the result as follows:

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

Example:

The JMOV function allows an INTEGER*4 quantity to be compared with 0 by using it in a logical IF statement. For example:

```
     INTEGER*4 INT1
     .
     .
     .
     IF(JMOV(INT1,INT1)) 300,100,300 !GO TO STMT 300 IF INT1 IS NOT 0
```

JMUL

4.3.65 JMUL

The JMUL function computes the product of two INTEGER*4 values.

Form: i = JMUL (joprl,jopr2,jres)

where	joprl	is an INTEGER*4 variable that is the multiplicand.
	jopr2	is an INTEGER*4 variable that is the multiplier.
	jres	is an INTEGER*4 variable that receives the product of the operation.

Function Results:

i = -2	An overflow occurred while computing the result.
= -1	Normal return; the product is negative.
= 0	Normal return; the product is 0.
= 1	Normal return; the product is positive.

Example:

```
        INTEGER*4 J1,J2,JRES
        .
        .
        .
        IF(JMUL(J1,J2,JRES)+1) 100,10,20
C       GOTO 100 IF OVERFLOW
C       GOTO 10 IF RESULT IS NEGATIVE
C       GOTO 20 IF RESULT IS POSITIVE OR ZERO
```

JSUB

4.3.66 JSUB

The JSUB function computes the difference between two INTEGER*4 values.

Form: i = JSUB (joprl,jopr2,jres)

where:	joprl	is an INTEGER*4 variable that is the minuend of the operation.

jopr2	is an INTEGER*4 variable that is the subtrahend of the operation.
jres	is an INTEGER*4 variable that is to receive the difference between iopr1 and iopr2 (that is, jres=jopr1-jopr2).

Function Results:

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

Example:

```
        INTEGER*4 JOP1,JOP2,J3
        .
        .
        .
        CALL JSUB(JOP1,JOP2,J3)
```

JTIME

4.3.67 JTIME

The JTIME subroutine converts the time specified to the internal two-word format time.

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

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 for 60-cycle clocks; 1/50 of a second for 50-cycle clocks).
	time	is the two-word area to receive the internal format time: time(1) is the high-order time. time(2) is the low-order time.

Errors:

None.

Example:

```
        INTEGER*4 J1
        .
        .
        .
C       CONVERT 3 HRS, 7 MIN, 23 SECONDS TO INTEGER *4 VALUE
        CALL JTIME(3,7,23,0,J1)
        CALL JJCVT(J1)
```

LEN

4.3.68 LEN

The LEN function returns the number of characters currently in the string contained in a specified array. This number is computed as the number of characters preceeding the first null byte encountered. If the specified array contains a null string, a value of 0 is returned.

Form: i = LEN (a)

where:	a	specifies the array containing the string.

Errors:

None.

Example:

```
        LOGICAL*1 STRNG(73)
        .
        .
        .
        TYPE 99,(STRNG(I),I=1,LEN(STRNG))
   99   FORMAT('0',132A1)
```

LOCK

4.3.69 LOCK

The LOCK subroutine is issued to keep the USR in memory for a series of operations. The USR (User Service Routine) is the section of the RT-11 system that performs various file management functions.

If all the conditions that cause swapping are satisfied, a portion of the user program is written out to the disk file SWAP.SYS and the USR is loaded. Otherwise, the USR in memory is used, and no swapping occurs. The USR is not released until an UNLOCK (see Section 4.3.102) is given. (Note that in an FB system, calling the CSI can also perform an implicit UNLOCK.) A program that has many USR requests to make can LOCK the USR in memory, make all the requests, and then UNLOCK the USR; no time is spent doing unnecessary swapping.

In an FB or XM environment, a LOCK inhibits the other job from using the USR. Thus, the USR should be locked only for as long as necessary.

NOTE

Foreground jobs perform a LOCK when they require the USR. This can cause the USR to be unavailable for other jobs for a considerable period of time. The USR is not reentrant and it cannot be shared by other jobs. Only one job has use of the USR at a time and other jobs requiring it must queue up for it. This fact should be considered for systems requiring concurrent foreground and background jobs. This is particularly true when magtape and/or cassette are active.

The USR does file operations, and these operations require a sequential search of the tape for magtape and cassette. This could lock out the foreground job for a long time while the background job does a tape operation. The programmer should keep this in mind when designing such systems. The FB and XM monitors supply the ITLOCK routine, which permits the foreground job to check for the availability of the USR.

Form: CALL LOCK

Note that the LOCK routine reduces time spent in file handling by eliminating the swapping of the USR. If the USR is currently resident, LOCK involves no I/O. (The USR is always resident in XM.) After a LOCK has been executed, the UNLOCK routine must be executed to release the USR from memory. The LOCK/UNLOCK routines are complementary and must be matched. That is, if three LOCKs are issued, at least three UNLOCKs must be done, otherwise the USR is not released. More UNLOCKs than LOCKs can occur without error; the extra UNLOCKs are ignored.

Notes:

1. It is vital that the LOCK call not come from within the area into which the USR will be swapped. If this should occur, the return from the USR request would not be to the user program, but to the USR itself, since the LOCK function causes part of the user program to be saved on disk and replaced in memory by the USR. Furthermore, subroutines, variables, and arrays in the area where the USR is swapping should not be referenced while the USR is LOCKed in memory.

2. Once a LOCK has been performed, it is not advisable for the program to destroy the area the USR is in, even though no further use of the USR is required. This causes unpredictable results when an UNLOCK is done.

3. LOCK cannot be called from a completion or interrupt routine.

4. If a SET USR NOSWAP command has been issued, LOCK and UNLOCK do not cause the USR to swap. However, in FB, LOCK still inhibits the other job from using the USR and UNLOCK allows the other job access to the USR.

5. The USR cannot accept argument lists, such as device file name specifications, located in the area into which it has been locked.

Errors:

None.

LOOKUP

4.3.70 LOOKUP

The LOOKUP function associates a specified channel with a device and/or file for the purpose of performing I/O operations. The channel used is then busy until one of the following functions is executed.

CLOSEC
ISAVES
PURGE

Form: i = LOOKUP (chan,dblk[,count])

where:		
	chan	is the integer specification for the RT-11 channel to be associated with the file.
	dblk	is the four-word area specifying the Radix-50 file descriptor. Note that unpredictable results occur if the USR swaps over this four-word area.
	count	is an optional argument used for the cassette handler. This argument defaults to 0.

NOTE

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

The handler for the selected device must be in memory for a LOOKUP. If the first word of the file name in dblk is 0 and the device is a file-structured device, absolute block 0 of the device is designated as the beginning of the file. This technique, called a non-file-structured LOOKUP, allows I/O to any physical block on the device. If a file name is specified for a device that is not file-structured (such as LP:FILE.TYP), the name is ignored.

NOTE

Since non-file-structured LOOKUPs allow I/O to any physical block on the device, the user must be aware that, in this mode, it is possible to overwrite the RT-11 device directory, thus destroying all file information on the device.

Errors:

i = n	Normal return; n equals the number of blocks in the file (0 for non-file-structured lookups on a cassette and magtape).
= -1	Channel specified is already open.
= -2	File specified was not found on the device.
-3	Device in use
-4	Tape drive is not available

Example:

```
INTEGER*2 DBLK(4)
DATA DBLK/3RDK0,3RFTN,3R44 ,3RDAT/
.
.
.
ICHAN=IGETC()
IF(ICHAN.LT.0) STOP 'NO CHANNEL'
IF(IFETCH(DBLK).NE.0) STOP 'BAD FETCH'
IF(LOOKUP(ICHAN,DBLK).LT.0) STOP 'BAD LOOKUP'
.
.
.
CALL CLOSEC(ICHAN)
CALL IFREEC(ICHAN)
.
.
.
```

MRKT

4.3.71 MRKT

The MRKT function schedules an assembly language completion routine to be entered after a specified time interval has elapsed. Support for MRKT in SJ requires timer support.

Form: i = MRKT (id,crtn,time)

where:

id	is an integer identification number to be passed to the routine being scheduled.
crtn	is the name of the assembly language routine to be entered when the time interval elapses. This name must be specified in an EXTERNAL statement in the FORTRAN routine that issues the MRKT call.
time	is the two-word internal format time interval; when this interval elapses, the routine is entered. If considered as a two-element INTEGER*2 array:

time (1) is the high-order time.

time (2) is the low-order time.

Notes:

1. MRKT requires a queue element; this should be considered when the IQSET function (Section 4.3.33) is executed.

2. If the system is busy, the time interval that elapses before the completion routine is run can be greater than that requested.

For further information on scheduling completion routines, see the assembly language .MRKT request, Section 2.4.22.

Errors:

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

Example:

```
INTEGER*2 TINT(2)
EXTERNAL ARTN
.
.
.
CALL MRKT(4,ARTN,TINT)
```

MTATCH

4.3.72 **MTATCH (FB and XM Only)**

The MTATCH subroutine attaches a terminal for exclusive use by the requesting job. This operation must be performed before any job can use a terminal with multi-terminal programmed requests.

Form: i = MTATCH (unit[,addr])

where:	addr	is the optional address of an asynchronous terminal status word. Omit this argument if the asynchronous terminal status word is not required. The asynchronous terminal status word is a SYSGEN option.
	unit	is the logical unit number of the terminal (lun).

Errors:

i = 0	Normal return
= 3	Non-existent unit number
= 5	Unit attached by another job
= 6	In XM monitor, the optional status word address is not in a valid user virtual address space.

4.3.73 MTDTCH (FB and XM Only)

The MTDTCH subroutine is the complement of the MTATCH subroutine. Its function is to detach a terminal from a particular job, and make it available for other jobs.

Form: i = MTDTCH(unit)

where:	unit	is the logical unit number of the terminal to be detached (lun).

Errors:

i = 0	Normal return
= 2	Illegal unit number. Terminal is not attached.
= 3	Non-existent unit number.

4.3.74 MTGET (FB and XM Only)

The MTGET subroutine furnishes the user with information about a specific terminal in a multi-terminal system.

Form: i = MTGET (unit,addr)

where:	addr	is a four-word area to receive the status information. The area is a four-element INTEGER*2 array. See Section 2.4.36 for area format.
	unit	is the unit number of the line and terminal whose status is desired.

Errors:

i = 0	Normal return
= 2	Unit not attached
= 3	Non-existent unit number
= 6	In XM monitor, the terminal status buffer address is outside legal program limits.

4.3.75 MTIN (FB and XM Only)

The MTIN subroutine transfers characters from a specified terminal to the user program. This subroutine is a multi-terminal form of ITTINR. If no characters are available, a flag is set to indicate an error upon return from the subroutine. If no character count argument is specified, one character is transferred.

Form: i = MTIN (unit,char[,chrcnt])

where:		
	unit	is the unit number of the terminal.
	char	is the variable to contain the characters read in from the terminal indicated by the unit number.
	chrcnt	is an optional argument that indicates the number of characters to be read.

Errors:

i = 0	Normal return
= 1	No input available
= 2	Unit not attached
= 3	Non-existent unit number

MTOUT

4.3.76 **MTOUT (FB and XM Only)**

The MTOUT subroutine transfers characters to a specified terminal. This subroutine is a multi-terminal form of ITTOU. If no room is available in the output ring buffer, a flag is set to indicate an error upon return from the subroutine. If no character count argument is specified, one character is transferred.

Form: i = MTOUT (unit,char[,chrcnt])

where:		
	unit	is the unit number of the terminal.
	char	is the variable containing the characters to be output, right-justified in the integer (can be LOGICAL*1 if desired).
	chrcnt	is an optional argument that indicates the number of characters to be output.

Errors:

i = 0	Normal return
i = 1	No room in output ring buffer.
i = 2	Unit not attached
i = 3	Non-existent unit number

MTPRNT

4.3.77 **MTPRNT (FB and XM Only)**

The MTPRNT subroutine operates the same as the PRINT subroutine (Section 4.3.81) in a multi-terminal environment. It allows output to be printed at the console terminal (see Section 2.4 for more details)

Form: i = MTPRNT (unit,string)

where:	string	is the character string to be printed. Note that all quoted literals used in FORTRAN subroutine calls are in ASCIZ format. All character strings produced by the SYSF4 string handling package are also in the ASCIZ format.
	unit	is the unit number associated with the terminal.

Errors:

i = 0	Normal return
i = 2	Unit not attached
i = 3	Non-existent unit number

MTRCTO

4.3.78 **MTRCTO (FB and XM Only)**

The MTRCTO subroutine operates the same as the .RCTRLO programmed request in a multi-terminal environment. This request resets the CTRL/O command originated at the console terminal.

Form: i = MTRCTO(unit)

where:	unit	is the unit number associated with the terminal.

Errors:

i = 0	Normal return
= 2	Unit not attached
= 3	Non-existent unit number

MTSET

4.3.79 **MTSET (FB and XM Only)**

The MTSET subroutine allows the user program to set terminal and line characteristics. (See .MTSET program request in Chapter 2 for more details.)

Form: i = MTSET (unit,addr)

where:	addr	is a four-word area to pass the status information. The area is a four-element INTEGER*2 array. See Section 2.4 for area format.
	unit	is the unit number of the line and terminal whose characteristics are to be changed.

Errors:

i = 0	Normal return
= 2	Unit not attached
= 3	Non-existent unit number

= 6 In the XM monitor, the terminal status buffer address is outside legal program limits.

Example:

```
      PROGRAM MULTEM
C     MULTEM.FOR SYSF4 TEST FOR MULTI-TERMINAL ROUTINES
C
      DIMENSION IADDR(2,4)            !(IUNIT,STATUS WD)
      LOGICAL*1 PROMPT(8),ISTRNG(134)
      DATA PROMPT/'S','T','R','I','N','G','>',"200/
C
      CALL PRINT ('THE FOLLOWING NUMBERS ACTIVATE CERTAIN FUNCTIONS')
      CALL PRINT ('1 = MTSET')
      CALL PRINT ('2 = MTGET')
      CALL PRINT ('3 = MTIN')
      CALL PRINT ('4 = MTOUT')
      CALL PRINT ('5 = MTRCTO')
      CALL PRINT ('6 = MTATCH')
      CALL PRINT ('7 = MTDTCH')
      CALL PRINT ('8 = MTPRNT')
5     TYPE 19
      ACCEPT 8,IFUN                   !GET FUNCTION TO DO
      GOTO (100,200,300,400,500,600,700,800),IFUN
C
100   TYPE 109
109   FORMAT ('$SETUP TERMINAL STATUS BLOCK,STATUS WORD 1 ? ')
      ACCEPT 9,IADDR(IUNIT,1)
      IADDR(IUNIT,2) = 0
      TYPE 129
129   FORMAT ('$FILLER CHARACTER ? ')
      ACCEPT 9,J
      TYPE 139
139   FORMAT ('$NUMBER OF FILLERS ? ')
      ACCEPT 8,I
      IADDR(IUNIT,3)=I*256 + J
      TYPE 149
149   FORMAT ('$CARRIAGE WIDTH ? ')
      ACCEPT 8,J
      TYPE 159
159   FORMAT ('$STATE BYTE ? ')
      ACCEPT 8,I
      IADDR(IUNIT,4) = I*256 + J
      IERR = MTSET (IUNIT, IADDR(IUNIT,1))
      GOTO 999
C
200   IERR = MTGET (IUNIT, IADDR(IUNIT,1))
      TYPE 209,(IADDR(IUNIT,I),I=1,4)
209   FORMAT (' 4 WORD STATUS BLK IS:',4O6)
      GOTO 999
C
300   IERR = MTIN (IUNIT, ICHAR)
305   TYPE 309,ICHAR
309   FORMAT (' ICHAR=',A1)
      GOTO 999
C
400   IERR = MTOUT (IUNIT, ICHAR)
      GOTO 305
C
500   IERR = MTRCTO (IUNIT)
      GOTO 999
C
600   TYPE 609
```

```
609   FORMAT ('$IUNIT ? ')
      ACCEPT 8,IUNIT
      TYPE 619
619   FORMAT ('$ASYNCHRONOUS WORD ? ')
      ACCEPT 9,IASYN
      IERR = MTATCH (IUNIT, IASYN)
      GOTO 999
C
700   IERR = MTDTCH (IUNIT)
      GOTO 999
C
800   CALL GTLIN (ISTRNG,PROMPT)
      IERR = MTPRNT (IUNIT, ISTRNG)
999   TYPE 998,IERR
      GOTO 5
C
8     FORMAT (I4)
9     FORMAT (O6)
19    FORMAT ('$FUNCTION ? ')
998   FORMAT (' IERR =',I4)
      END
```

MWAIT

4.3.80 MWAIT (FB and XM Only)

The MWAIT subroutine suspends main program execution of the current job until all messages sent to or from the other job have been transmitted or received. It provides a means for ensuring that a required message has been processed. MWAIT is used primarily in conjunction with the IRCVD and ISDAT calls, where no action is taken when a message transmission is completed. This subroutine requires a queue element; this should be considered when the IQSET function (Section 4.3.37) is executed.

Form: CALL MWAIT

Errors:

None.

Example:

See the example under ISDAT, Section 4.3.45.

PRINT

4.3.81 PRINT

The PRINT subroutine causes output (from a specified string) to be printed at the console terminal. This routine can be used to print messages from completion routines without using the FORTRAN formatted I/O system. Control returns to the user program after all characers have been placed in the output buffer.

The string to be printed can be terminated with either a null (0) byte or a 200 (octal) byte. If the null (ASCIZ) format is used, the output is automatically followed by a carriage return/line feed pair (octal

15 and 12). If a 200 byte terminates the string, no carriage return/line feed pair is generated.

In the FB monitor, a change in the job that is controlling terminal output is indicated by a B> or F>. Any text following the message has been printed by the job indicated (foreground or background) until another B> or F> is printed. When PRINT is used by the foreground job, the message appears immediately, regardless of the state of the background job. Thus, for urgent messages, PRINT should be used rather than ITTOUR.

Form: CALL PRINT (string)

where:	string	is the string to be printed. Note that all quoted literals used in FORTRAN subroutine calls are in ASCIZ format as are all strings produced by the SYSF4 string handling package.

Errors:

None.

Example:

```
        CALL PRINT ('THE COFFEE IS READY')
```

PURGE

4.3.82 PURGE

The PURGE subroutine is used to deactivate a channel without performing an ISAVES or a CLOSEC. Any tentative file currently associated with the channel is not made permanent. This subroutine is useful for keeping ENTERed (IENTER or .ENTER) files from becoming permanent directory entries.

Form: CALL PURGE (chan)

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

Errors:

None.

Example:

See the example under IENTER, Section 4.3.22.

PUTSTR

4.3.83 PUTSTR

The PUTSTR subroutine writes a variable-length character string to a specified FORTRAN logical unit. PUTSTR can be used in main program routines or in completion routines but not in both in the same program at the same time. If PUTSTR is used in a completion routine, it must not be the first I/O operation on the specified logical unit.

Form: CALL PUTSTR (lun,in,char,err)

where:	lun	is the integer specification of the FORTRAN logical unit number to which the string is to be written.
	in	is the array containing the string to be written.
	char	is an ASCII character that is appended to the beginning of the string before it is output. If 0, the first character of in is the first character of the record. This character is used primarily for carriage control purposes (see Section 4.3.13).
	err	is a LOGICAL*1 variable that is .TRUE for an error condition and .FALSE for a no error condition.

Errors:

ERR = -1	End-of-file for write operation
-2	Hardware error for write operation

Example:

```
        LOGICAL*1 STRNG(81)
        .
        .
        .
        CALL PUTSTR(7,STRNG,'0')  !OUTPUT STRING WITH DOUBLE SPACING
```

R50ASC

4.3.84 R50ASC

The R50ASC subroutine converts a specified number of Radix-50 characters to ASCII.

Form: CALL R50ASC (icnt,input,output)

where:	icnt	is the integer number of ASCII characters to be produced.
	input	is the area from which words of Radix-50 values to be converted are taken. Note that (icnt+2)/3 words are read for conversion.
	output	is the area into which the ASCII characters are stored.

Errors:

If an input word contains illegal Radix-50 codes (that is, if the input word is greater (unsigned) than 174777(octal)), the routine outputs question marks for the value.

Example:

```
        REAL*8 NAME
        LOGICAL*1 OUTP(12)
        .
        .
        .
        CALL R50ASC(12,NAME,OUTP)
```

RAD50

4.3.85 RAD50

The RAD50 function provides a method of encoding RT-11 file descriptors in Radix-50 notation. The RAD50 function converts six ASCII characters from the specified area, returning a REAL*4 result that is the two-word Radix-50 value.

Form: a = RAD50 (input)

where:	input	is the area from which the ASCII input characters are taken.

The RAD50 call:

```
     A = RAD50 (LINE)
```

is exactly equivalent to the IRAD50 call:

```
     CALL IRAD50 (6,LINE,A)
```

Function Results:

The two-word Radix-50 value is returned as the function result.

RCHAIN

4.3.86 RCHAIN

The RCHAIN subroutine allows a program to determine whether it has been chained to and to access variables passed across a chain. If RCHAIN is used, it must be used in the first executable FORTRAN statement in a program. RCHAIN cannot be called from a completion or interrupt routine.

Form: CALL RCHAIN (flag,var,wcnt)

where:	flag	is an integer variable that is set to -1 if the program has been chained to; otherwise, it is 0.
	var	is the first variable in a sequence of variables with increasing memory addresses to receive the information passed across the chain (see Section 4.3.2).

wcnt — is the number of words to be moved from the chain parameter area to the area specified by var. RCHAIN moves wcnt words into the area beginning at var.

Errors:

None.

Example:

```
INTEGER*2 PARMS(50)
CALL RCHAIN(IFLAG,PARMS,50)
IF(IFLAG) GOTO 10      !GOTO 10 IF CHAINED TO
.
.
.
```

RCTRLO

4.3.87 RCTRLO

The RCTRLO subroutine resets the effect of any console terminal CTRL/O command that was typed. After an RCTRLO call, any output directed to the console terminal prints until another CTRL/O is typed.

Form: CALL RCTRLO

Errors:

None.

Example:

```
CALL RCTRLO
CALL PRINT ('THE REACTOR IS ABOUT TO BLOW UP')
```

REPEAT

4.3.88 REPEAT

The REPEAT subroutine concatenates a specified string with itself to produce the indicated number of copies. REPEAT places the resulting string in a specified array.

Form: CALL REPEAT (in,out,i[,len[,err]])

where:	in	is the array containing the string to be repeated.
	out	is the array into which the resultant string is placed. This array must be at least one element longer than the value of len, if specified.
	i	is the integer number of times to repeat the string.

len	is the integer number representing the maximum length of the output string.
err	is the logical error flag set if the output string is truncated to the length specified by len.

Input and output strings can specify the same array only if the repeat count (i) is 1 or 0. When the repeat count is 1, this routine is the equivalent of SCOPY; when the repeat count is 0, out is replaced by 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:

```
        LOGICAL*1 SIN(21),SOUT(101)
        .
        .
        .
        CALL REPEAT(SIN,SOUT,5)
```

RESUME

4.3.89 RESUME (FB and XM Only)

The RESUME subroutine allows a job to resume execution of the main program. A RESUME call is normally issued from an asynchronous FORTRAN routine entered on I/O completion or because of a schedule request. (See Section 4.3.97 for more information.)

Form: CALL RESUME

Errors:

None.

Example:

See the example under SUSPND, Section 4.3.97.

SCCA

4.3.90 SCCA

The SCCA subroutine provides a CTRL/C intercept to perform the following functions:

1. Inhibit a CTRL/C abort

2. Indicate that a CTRL/C command is active

3. Distinguish between single and double CTRL/C commands

Form: CALL SCCA [(iflag)]

where:	iflag	is an integer terminal status word that must be tested and cleared to determine if two CTRL/Cs were typed at the console terminal. The iflag must be an INTEGER*2 variable (not LOGICAL*1).

When a CTRL/C is typed, the SCCA subroutine, having been previously called, makes the CTRL/C command inactive and places it in the input ring buffer. While residing in the buffer, the inactive command can be read as a valid character by the program. The program must test and clear the iflag to determine if two CTRL/C commands were typed consecutively. The iflag is set to non-zero when two CTRL/Cs are typed together. It is the responsibility of the program to abort itself, if appropriate, on an input of CTRL/C from the terminal. The SCCA subroutine with no argument disables the CTRL/C intercept.

Errors:

None

Example:

```
      PROGRAM SCCA
C     SCCA.FOR SYSF4 TEST FOR SCCA
C
      CALL PRINT ('PROGRAM HAS STARTED, TYPE')
      IFLAG=0
      CALL SCCA (IFLAG)
10    I = ITTINR()               !GET A CHARACTER
      IF (I .NE. 3) GOTO 10
C     A CTRL/C WAS TYPED
      CALL PRINT ('A CTRL/C WAS TYPED')
      IF (IFLAG .EQ. 0) GOTO 10
      CALL PRINT ('A DOUBLE CTRL/C WAS TYPED')
      TYPE 19,IFLAG
19    FORMAT (' IFLAG = ',O6,/)
      CALL SCCA                  !DISABLE CTRL/C INTERCEPT
      CALL PRINT ('TYPE A CTRL/C TO EXIT')
20    GOTO 20                    !LOOP UNTIL CTRL/C TYPED
      END
```

4.3.91 SCOMP

The SCOMP routine compares two character strings and returns the integer result of the comparison.

Form: CALL SCOMP (a,b,i)

or

i = ISCOMP (a,b)

where:	a	is the array containing the first string.
	b	is the array containing the second string.

i	is the integer variable that receives the result of the comparison.

The strings are compared left to right, one character at a time, using the collating sequence specified by the ASCII codes for each character. If the two strings are not equal, the absolute value of variable i (or the result of the function ISCOMP) is the character position of the first inequality found in scanning left to right. Strings are terminated by a null (0) character.

If the strings are not the same length, the shorter one is treated as if it were padded on the right with blanks to the length of the other string. A null string argument is equivalent to a string containing only blanks.

Function Result:

i <0	if a is less than b
=0	if a is equal to b
>0	if a is greater than b

Example:

```
LOGICAL*1 INSTR(81)
.
.
.
CALL GETSTR(5,INSTR,80)
CALL SCOMP('YES',INSTR,IVAL)
IF(IVAL) GOTO 10      !IF INPUT STRING IS NOT YES GOTO 10
```

SCOPY

4.3.92 SCOPY

The SCOPY routine copies a character string from one array to another. Copying stops either when a null (0) character is encountered or when a specified number of characters have been moved.

Form: CALL SCOPY (in,out[,len[,err]])

where:	in	is the array containing the string to be copied.
	out	is the array to receive the copied string. This array must be at least one element longer than the value of len, if specified.
	len	is the integer number 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 a logical variable that receives the error indication if the output string was truncated to the length specified by len.

The input (in) and output (out) arguments can specify the same array. The string previously contained in the output array is lost when this subroutine is called.

Errors:

Error conditions are indicated by err, if specified. If err is given and the output string was truncated to the length specified by len, then err is set to .TRUE.; otherwise, err is unchanged.

Example:

SCOPY is useful for initializing strings to a constant value, for example:

```
          LOGICAL*1 STRING(80)
          CALL SCOPY('THIS IS THE INITIAL VALUE',STRING)
```

SECNDS

4.3.93 SECNDS

The SECNDS function returns the current system time, in seconds past midnight, minus the value of a specified argument. Thus, SECNDS can be used to calculate elapsed time. The value returned is single-precision floating point (REAL*4).

Form: a = SECNDS (atime)

where:	atime	is a REAL*4 variable, constant, or expression whose value is subtracted from the current time of day to form the result.

Notes:

This function does floating-point arithmetic. Elapsed time can also be calculated by using the GTIM call and the INTEGER*4 support functions.

Function Result:

The function result (a) is the REAL*4 value returned. Example:

```
C     START OF TIMED SEQUENCE
      T1=SECNDS(0.)
C
C     CODE TO BE TIMED GOES HERE
C
      DELTA=SECNDS(T1)     !DELTA IS ELAPSED TIME
```

SETCMD

4.3.94 SETCMD

The SETCMD routine allows a user program to pass a command line to the keyboard monitor to be executed after the program exits. The command lines are passed to the chain information area (500-777(octal)) and stored beginning at location 512(octal). No check is made to determine if the string extends into the stack space. For this reason, the command line should be short and the subroutine call should be made in the main program unit near the end of the program just before completion. If several commands are desired to be executed, an indirect command file that contains many command lines should be used.

The following monitor commands are disallowed if the SETCMD feature is used.

1. REENTER
2. START
3. CLOSE

Form: CALL SETCMD (string)

where: string is a keyboard monitor command line in ASCIZ format with no embedded carriage returns or line feeds.

Errors

None

Example:

```
LOGICAL*1 INPUT(134),PROMPT(8)
DATAPROMPT/'P','R','O','M','P','T','>',"200/
CALL GTLIN (INPUT,PROMPT)
CALL SETCMD (INPUT)
END
```

NOTE

Set USR NOSWAP or specify /NOSWAP with the COMPILE, FORTRAN, or EXECUTE command.

STRPAD

4.3.95 STRPAD

The STRPAD routine pads a character string with rightmost blanks until that string is a specified length. This padding is done in place; the result string is contained in its original array. If the present length of the string is greater than or equal to the specified length, no padding occurs.

Form: CALL STRPAD (a,i[,err])

where:	a	is the string to be padded.
	i	is the integer length of the desired result string.
	err	is the logical error flag that is set to .TRUE. if the string specified by a exceeds the value of i in length.

Errors:

Error conditions are indicated by err, if specified. If err is given and the string indicated is longer than i characters, err is set to .TRUE.; otherwise, the value of err is unchanged.

Example:

This routine is especially useful for preparing strings to be output in A-type FORMAT fields. For example:

```
          LOGICAL*1 STR(81)
          .
          .
          .
          CALL STRPAD(STR,80)        !ASSURE 80 VALID CHARACTERS
          PRINT 100,(STR(I),I=1,80) !PRINT STRING OF 80 CHARACTERS
     100  FORMAT(80A1)
```

SUBSTR

4.3.96 SUBSTR

The SUBSTR routine copies a substring from a specified position in a character string. If desired, the substring can then be placed in the same array as the string from which it was taken.

Form: CALL SUBSTR (in,out,i[,len])

where:	in	is the array from which the substring is taken.
	out	is the array to contain the substring result. This array must be one element longer than len, if specified.
	i	is the integer character position in the input string of the first character of the desired substring.
	len	is the integer number of characters representing the maximum length of the substring.

If a maximum length (len) is not given, the substring contains all characters to the right of character position i in array in. If len is equal to zero, out is replaced by the null string. The old contents of array out are lost when this routine is called.

Errors:

None.

4.3.97 SUSPND (FB and XM Only)

The SUSPND subroutine suspends main program execution of the current job and allows only completion routines (for I/O and scheduling requests) to run.

Form: CALL SUSPND

Notes:

1. The monitor maintains a suspension counter for each job. This count is decremented by SUSPND and incremented by RESUME (see Section 4.3.81). A job will actually be suspended only if this counter is negative. Thus, if a RESUME is issued before a SUSPND, the latter function will return immediately.

2. A program must issue an equal number of SUSPNDs and RESUMEs.

3. A RESUME subroutine call from the main program or from a completion routine increments the suspension counter.

4. A SUSPND subroutine call from a completion routine decrements the suspension counter but does not suspend the main program. If a completion routine does a SUSPND, the main program continues until it also issues a SUSPND, at which time it is suspended and requires two RESUMEs to proceed.

5. Because SUSPND and RESUME are used to simulate an ITWAIT (see Section 4.3.45) in the monitor, a RESUME issued from a completion routine and not matched by a previously executed SUSPND can cause the main program execution to continue past a timed wait before the entire time interval has elapsed.

For further information on suspending main program execution of the current job, see the assembly language .SPND request, Section 2.4.

Errors:

None.

Example:

```
      INTEGER IAREA(4)
      COMMON /RDBLK/ IBUF(256)
      EXTERNAL RDFIN
      .
      .
      .
      IF(IREADF(256,IBUF,IBLK,ICHAN,IAREA,RDFIN).NE.0) GOTO 1000
C     GOTO 1000 FOR ANY TYPE OF ERROR
C
C     DO OVERLAPPED PROCESSING
      .
      .
      .
      CALL SUSPND      !SYNCHRONIZE WITH COMPLETION ROUTINE
      .
      .
      .
      END
```

```
          SUBROUTINE RDFIN(IARG1,IARG2)
          COMMON /RDBLK/ IBUF(256)
          .
          .
          .
          CALL RESUME      !CONTINUE MAIN PROGRAM
          .
          .
          .
          END
```

TIMASC

4.3.98 TIMASC

The TIMASC subroutine converts a two-word internal format time into an ASCII string of the form:

hh:mm:ss

where:	hh	is the two-digit hours indication
	mm	is the two-digit minutes indication
	ss	is the two-digit seconds indication

Form: CALL TIMASC (itime,strng)

where:	itime	is the two-word internal format time to be converted. itime (1) is the high-order time. itime (2) is the low-order time.
	strng	is the eight-element array to contain the ASCII time.

Errors:

None.

Example:

The following example determines the amount of time until 5 p.m. and prints it.

```
          INTEGER*4 J1,J2,J3
          LOGICAL*1 STRNG(8)
          .
          .
          .
          CALL JTIME(17,0,0,0,J1)
          CALL GTIM(J2)
          CALL JJCVT(J1)
          CALL JJCVT(J2)
          CALL JSUB(J1,J2,J3)
          CALL JJCVT(J3)
          CALL TIMASC(J3,STRNG)
          TYPE 99,(STRNG(I),I=1,8)
     99   FORMAT(' IT IS ',8A1,' TILL 5 P.M.')
          .
          .
          .
```

TIME

4.3.99 TIME

The TIME subroutine returns the current system time of day as an eight-character ASCII string of the form:

hh:mm:ss

where:	hh	is the two-digit hours indication
	mm	is the two-digit minutes indication
	ss	is the two-digit seconds indication

Form: CALL TIME (strng)

where:	strng	is the eight-element array to receive the ASCII time.

Notes:

A 24-hour clock is used (for example, 1:00 p.m. is represented as 13:00:00). The DATE and IDATE subroutines are available as part of FORTRAN IV system routines.

Errors:

None.

Example:

```
        LOGICAL*1 STRNG(8)
        .
        .
        .
        CALL TIME(STRNG)
        TYPE 99,(STRNG(I),I=1,8)
   99   FORMAT (' IT IS NOW ',8A1)
```

TRANSL

4.3.100 TRANSL

The TRANSL routine performs character translation on a specified string. The TRANSL routine requires approximately 64 words on the R6 stack for its execution. This space should be considered when allocating stack space.

Form: CALL TRANSL (in,out,r[,p])

where:	in	is the array containing the input string.
	out	is the array to receive the translated string.

r	is the array containing the replacement string.
p	is the array containing the characters in in to be translated.

The string specified by array out is replaced by the string specified by array in, modified by the character translation process specified by arrays r and p. If any character position in in contains a character that appears in the string specified by p, it is replaced in out by the corresponding character from string r. If the array p is omitted, it is assumed to be the 127 seven-bit ASCII characters arranged in ascending order, beginning with the character whose ASCII code is 001. If strings r and p are given and differ in length, the longer string is truncated to the length of the shorter. If a character appears more than once in string p, only the last occurrence is significant. A character can appear any number of times in string r.

Errors:

None.

Examples:

The following example causes the string in array A to be copied to array B. All periods within A become minus signs, and all question marks become exclamation points.

```
     CALL TRANSL(A,B,'-!','.?')
```

The following is an example of TRANSL being used to format character data.

```
          LOGICAL*1 STRING(27),RESULT(27),PATRN(27)
     C    SET UP THE STRING TO BE REFORMATTED
     C
          CALL SCOPY('THE HORN BLOWS AT MIDNIGHT',STRING)
     C
     C    00000000011111111112222222
     C    12345678901234567890123456
     C    THE HORN BLOWS AT MIDNIGHT
     C    NOW SET UP PATRN TO CONTAIN THE FOLLOWING PATTERN:
     C    16,17,18,19,20,21,22,23,24,25,26,15,1,2,3,4,5,6,7,8,9,10,11,12,13,14,0
     C
          DO 10 I=16,26
     10   PATRN(I-15)=I
          PATRN(12)=15
          DO 20 I=1,14
     20   PATRN(I+12)=I
          PATRN(27)=0
     C
     C    THE FOLLOWING CALL TO TRANSL REARRANGES THE CHARACTERS OF
     C    THE INPUT STRING TO THE ORDER SPECIFIED BY PATRN:
     C
          CALL TRANSL(PATRN,RESULT,STRING)
     C
     C    RESULT NOW CONTAINS THE STRING 'AT MIDNIGHT THE HORN BLOWS'
     C    IN GENERAL, THIS METHOD CAN BE USED TO FORMAT INPUT STRINGS
     C    OF UP TO 127 CHARACTERS. THE RESULTANT STRING WILL BE
     C    AS LONG AS THE PATTERN STRING (AS IN THE ABOVE EXAMPLE).
```

TRIM

4.3.101 TRIM

The TRIM routine shortens a specified character string by removing all trailing blanks. A trailing blank is a blank that has no non-blanks to its right. If the specified string contains all blank characters, it is replaced by the null string. If the specified string has no trailing blanks, it is unchanged.

Form: CALL TRIM (a)

where:	a	is the array containing the string to be trimmed.

Errors:

None.

Example:

```
          LOGICAL*1 STRING(81)
          ACCEPT 100,(STRING(I),I=1,80)
    100   FORMAT(80A1)
          CALL SCOPY(STRING,STRING,80)        !MAKE ASCIZ
          CALL TRIM(STRING)                   !TRIM TRAILING BLANKS
```

UNLOCK

4.3.102 UNLOCK

The UNLOCK subroutine releases the User Service Routine (USR) from memory if it was placed there by the LOCK routine. If the LOCK required a swap, the UNLOCK loads the user program back into memory. If the USR does not require swapping, the UNLOCK involves no I/O. The USR is always resident in XM.

Form: CALL UNLOCK

Notes:

1. It is important that at least as many UNLOCKS are given as LOCKs. If more LOCKs were done, the USR remains locked in memory. It is not harmful to give more UNLOCKS than are required; those that are extra are ignored.

2. When running two jobs in the FB system, use the LOCK/UNLOCK pairs only when absolutely necessary. If one job LOCKs the USR, the other job cannot use the USR until it is UNLOCKed. Thus, the USR should not be LOCKed unnecessarily, as this may degrade performance in some cases.

3. In an FB system, calling the CSI (ICSI) with input coming from the console terminal performs an implicit UNLOCK.

For further information on releasing the USR from memory, see the assembly language .LOCK/.UNLOCK requests, Section 2.4.

Errors:

None.

Example:

```
        .
        .
        .
C       GET READY TO DO MANY USR OPERATIONS
        CALL LOCK      !DISABLE USR SWAPPING
C       PERFORM THE USR CALLS
        .
        .
        .
C       FREE THE USR
        CALL UNLOCK
        .
        .
        .
```

VERIFY

4.3.103 VERIFY

The VERIFY routine determines whether each character of a specified string occurs anywhere in another string. If a character does not exist in the string being examined, VERIFY returns its character position in string b. If all characters exist, VERIFY returns a 0.

Form: CALL VERIFY (a,b,i)

or

i = IVERIF (a,b)

where:		
	a	is the array containing the string to be scanned.
	b	is the array containing the string of characters to be accepted in a.
	i	is the integer result of the verification.

Function Result:

i = 0	if all characters of a exist in b; also if a is a null string.
= n	where n is the character position of the first character in a that does not appear in b; if b is a null string and a is not, i equals 1.

Example:

The following example accepts a one- to five-digit unsigned decimal number and returns its value.

```
          LOGICAL*1 INSTR(81)
          .
          .
          .
          CALL VERIFY(INSTR(IPOS),'0123456789',I)
          IF(I.EQ.1) STOP 'NUMBER MISSING'
          IF(I.EQ.0) I=LEN(INSTR)-IPOS+1
          IF(I.GT.5) STOP 'TOO MANY DIGITS'
          NUM=IVALUE(INSTR(IPOS),I)
          .
          .
          .
          END
          FUNCTION IVALUE(ARRAY,I)
          LOGICAL*1 ARRAY(1)
          DECODE(I,99,ARRAY) IVALUE
     99   FORMAT(I5)
          END
```

APPENDIX A

DISPLAY FILE HANDLER

This appendix describes the assembly language support provided under RT-11 for the VT11 graphic display hardware systems.

The following manuals are suggested for additional reference:

GT40/GT42 User's Guide
EK-GT40-OP-002

GT44 User's Guide
EK-GT44-OP-001

VT11 Graphic Display Processor
EK-VT11-TM-001

DECGRAPHIC-11 GT Series Reference Card
EH-02784-73

DECGraphic-11 FORTRAN Reference Manual
DEC-11-GFRMA-A-D

BASIC-11 Graphics Extensions User's Guide
DEC-11-LBGEA-A-D

A.1 DESCRIPTION

The graphics display terminals have hardware configurations that include a display processor and CRT (cathode ray tube) display. All systems are equipped with light pens and hardware character and vector generators, and are capable of high-quality graphics. The Display File Handler supports this graphics hardware at the assembly language level under the RT-11 monitor.

A.1.1 Assembly Language Display Support

The Display File Handler is not an RT-11 device handler, since it does not use the I/O structure of the RT-11 monitor. For example, it is not possible to use a utility program to transfer a text file to the display through the Display File Handler. Rather, the Display File Handler provides the graphics programmer the means for the display of graphics files and the easy management of the display processor. Included in its capabilities are such services as interrupt handling, light pen support, tracking object, and starting and stopping of the display processor.

The Display File Handler manages the display processor by means of a base segment (called VTBASE) which contains interrupt handlers, an internal display file and some pointers and flags. The display processor cycles through the internal display file; any user graphics files to be displayed are accessed by display subroutine calls from the Handler's display file. In this way, the Display File Handler exerts control over the display processor, relieving the assembly language user of the task.

Through the Display File Handler, the programmer can insert and remove calls to display files from the Handler's internal display file. Up to two user files may be inserted at one time, and that number may be increased by re-assembling the Handler. Any user file inserted for display may be blanked (the subroutine call to it bypassed) and unblanked by macro calls to the Display File Handler.

Since the Handler treats all user display files as graphics subroutines to its internal display file, a display processor subroutine call is required. This is implemented with software, using the display stop instruction, and is available for user programs. This instruction and several other extended instructions implemented with the display stop instruction are described in Section A.3.

The facilities of the Display File Handler are accessed through a file of macro definitions (VTMAC) which generate calls to a set of subroutines in VTLIB. VTMAC's call protocol is similar to that of the RT-11 macros. The expansion of the macros is shown in Section A.6. VTMAC also contains, for convenience in programming, the set of recommended display processor instruction mnemonics and their values. The mnemonics are listed in Section A.7 and are used in the examples throughout this appendix.

VTCAL1 through VTCAL4 are the set of subroutines which service the VTMAC calls. They include functions for display file and display processor management. These are described in detail in Section A.2. VTCAL1 through VTCAL4 are distributed, along with the base segment VTBASE, as a file of five object modules called VTHDLR.OBJ. VTHDLR is built into the graphics library VTLIB by using the monitor LIBRARY command. Section A.4.2 shows an example.

A.1.2 Monitor Display Support

The RT-11 monitor, under Version 03, directly supports the display as a console device. A keyboard monitor command, GT ON (GT OFF) permits the selection of the display as console device. Selection results in the allocation of approximately 1.25K words of memory for text buffer and code. The buffer holds approximately 2000 characters.

The text display includes a blinking cursor to indicate the position in the text where a character is added. The cursor initially appears at the top left corner of the text area. As lines are added to the text the cursor moves down the screen. When the maximum number of lines are on the screen, the top line is deleted from the text buffer when the line feed terminating a new line is received. This causes the appearance of "scrolling", as the text disappears off the top of the display.

When the maximum number of characters have been inserted in the text buffer, the scroller logic deletes a line from the top of the screen to make room for additional characters. Text may appear to move (scroll) off the top of the screen while the cursor is in the middle of a line.

The Display File Handler can operate simultaneously with the scroller program, permitting graphic displays and monitor dialogue to appear on the screen at the same time. It does this by inserting its internal display file into the display processor loop through the text buffer. However, the following should be noted. Under the SJ Monitor, if a program using the display for graphics is running with the scroller in use (that is, GT ON is in effect), and the program does a soft exit (.EXIT with R0 not equal to 0) with the display stopped, the display remains stopped until a CTRL/C is typed at the keyboard.

This can be recognized by failure of the monitor to echo on the screen when expected. If the scroller text display disappears after a program exit, always type CTRL/C to restore. If CTRL/C fails to restore the display, the running program probably has an error.

Four scroller control characters provide the user with the capability of halting the scroller, advancing the scrolling in page sections, and printing hard copy from the scroller.

NOTE

The scroller logic does not limit the length of a line, but the length of text lines affects the number of lines which may be displayed, since the text buffer is finite. As text lines become longer, the scroller logic may delete extra lines to make room for new text, temporarily decreasing the number of lines displayed.

A.2 DESCRIPTION OF GRAPHICS MACROS

The facilities of the Display File Handler are accessed through a set of macros, contained in VTMAC, which generate assembly language calls to the Handler at assembly time. The calls take the form of subroutine calls to the subroutines in VTLIB. Arguments are passed to the subroutines through register 0 and, in the case of the .TRACK call, through both register 0 and the stack.

This call convention is similar to Version 1 RT-11 I/O macro calls, except that the subroutine call instruction is used instead of the EMT instruction. If a macro requires an argument but none is specified, it is assumed that the address of the argument has already been placed in register 0. The programmer should not assume that R0 is preserved through the call.

A.2.1 .BLANK

The .BLANK request temporarily blanks the user display file specified in the request. It does this by by-passing the call to the user display file, which prevents the display processor from cycling through the user file, effectively blanking it. This effect can later be cancelled by the .RESTR request, which restores the user file. When the call returns, the user is assured the display processor is not in the file that was blanked.

Macro Call: .BLANK faddr

where:	faddr	is the address of the user display file to be blanked.

Errors:

No error is returned. If the file specified was not found in the Handler file or has already been blanked, the request is ignored.

A.2.2 .CLEAR

The .CLEAR request initializes the Display File Handler, clearing out any calls to user display files and resetting all of the internal flags and pointers.

After initialization with .LNKRT (Section A.2.4), the .CLEAR request can be used any time in a program to clear the display and to reset pointers. All calls to user files are deleted and all pointers to status buffers are reset. They must be re-inserted if they are to be used again.

Macro Call: .CLEAR

Errors:

None.

Example:

This example uses a .CLEAR request to initialize the Handler then later uses the .CLEAR to re-initialize the display. The first .CLEAR is used for the case when a program may be restarted after a CTRL C or other exit.

```
                   BR RSTRT

         EX1:      BIS #20000,@#44      ;SET REENTER BIT IN JSW
         RSTRT:    .UNLNK               ;CLEARS LINK FLAG FOR RESTART
                   .LNKRT               ;SET UP VECTORS, START DISPLAY
                   .CLEAR               ;INITIALIZE HANDLER
                   .INSRT #FILE1        ;DISPLAY A PICTURE
         1$:       .TTYIN               ;WAIT FOR A KEY STRIKE
                   CMPB #12,R0          ;LINE FEED?
                   BNE 1$               ;NO, LOOP
                   .CLEAR               ;YES, CLEAR DISPLAY
                   .INSRT #FILE2        ;DISPLAY NEW PICTURE
                        .
                        .
                        .
         FILE1:    POINT             ;AT POINT (0,500)
                   0
                   500
                   LONGV             ;DRAW A LINE
                   500!INTX          ;TO (500,500)
                   0
                   DRET
                   0

         FILE2:    POINT             ;AT POINT (500,0)
                   500
                   0
                   LONGV             ;DRAW A LINE
                   0!INTX            ;TO (500,500)
                   500
                   DRET
                   0

                   .END EX1
```

A.2.3 .INSRT

The .INSRT request inserts a call to the user display file specified in the request into the Display File Handler's internal display file. .INSRT causes the display processor to cycle through the user file as a subroutine to the internal file. The handler permits two user files at one time. The call inserted in the handler looks like the following:

```
                DJSR        ;DISPLAY SUBROUTINE
                .+4         ;RETURN ADDRESS
                .faddr      ;SUBROUTINE ADDRESS
```

The call to the user file is removed by replacing its address with the address of a null display file. The user file is blanked by replacing the DJSR with a DJMP instruction, bypassing the user file.

Macro Call: .INSRT faddr

where:	faddr	is the address of the user display file to be inserted.

Errors:

The .INSRT request returns with the C bit set if there was an error in processing the request. An error occurs only when the Handler's display file is full and cannot accept another file. If the user file specified exists, the request is not processed. Two display files with the same starting address cannot be inserted.

Example:

See the examples in Sections A.2.2 and A.2.4.

A.2.4 .LNKRT

The .LNKRT request sets up the display interrupt vectors and possibly links the Display File Handler to the scroll text buffer in the RT-11 monitor. It must be the first call to the Handler, and is used whether or not the RT-11 monitor is using the display for console output (i.e., the KMON command GT ON has been entered).

The .LNKRT request used with the Version 03 RT-11 monitor enables a display application program to determine the environment in which it is operating. Error codes are provided for the situations where there is no display hardware present on the system or the display hardware is already being used by another task (e.g., a foreground job in the foreground/background version).

The existence of the monitor scroller and the size of the Handler's subpicture stack are also returned to the caller. If a previous call to .LNKRT was made without a subsequent .UNLNK, the .LNKRT call is ignored and an error code is returned.

Macro Call: .LNKRT

Errors:

Error codes are returned in R0, with the N condition bit set.

Code	Meaning
-1	No VT11 display hardware is present on this system.
-2	VT11 hardware is presently in use.
-3	Handler has already been linked.

On completion of a successful .LNKRT request, R0 will contain the display subroutine stack size, indicating the depth to which display subroutines may be nested. The N bit will be zero.

If the RT-11 monitor scroll text buffer was not in memory at the time of the .LNKRT, the C bit will be returned set. The KMON commands GT ON and GT OFF cannot be issued while a task is using the display.

Example:

```
START:    .LNKRT                    ;LINK TO MONITOR
          BMI       ERROR           ;ERROR DOING LINK
          BCS       CONT            ;NO SCROLL IF C SET
          .SCROL    #SBUF           ;ADJUST SCROLL PARAMETERS
CONT:     .INSRT    #FILE1          ;DISPLAY A PICTURE
1$:       .TTYIN                    ;WAIT FOR KEY STRIKE
          CMPB      #12,R0          ;LINE FEED?
          BNE       1$              ;NO, LOOP
          .UNLNK                    ;YES, UNLINK AND EXIT
          .EXIT

SBUF:     .BYTE     5               ;LINE COUNT OF 5
          .BYTE     7               ;INTENSITY 7 (SCALE OF 1-8)
          .WORD     1000            ;POSITION OF TOP LINE

FILE1:    POINT                     ;AT POINT (500,500)
          500
          500
          CHAR                      ;DISPLAY SOME TEXT
          .ASCII   /FILE1 THIS IS FILE1.  TYPE CR TO EXIT/
          .EVEN
          DRET
          0

ERROR:    Error routine
```

A.2.5 .LPEN

The .LPEN request transfers the address of a light pen status data buffer to VTBASE. Once the buffer pointer has been passed to the Handler, the light pen interrupt handler in VTBASE will transfer display processor status data to the buffer, depending on the state of the buffer flag.

The buffer must have seven contiguous words of storage. The first word is the buffer flag, and it is initially cleared (set to zero) by the .LPEN request. When a light pen interrupt occurs, the interrupt handler transfers status data to the buffer and then sets the buffer flag non-zero. The program can loop on the buffer flag when waiting for a light pen hit (although doing this will tie up the processor; in a foreground/background environment, timed waits would be more desirable). No further data transfers take place, despite the occurrence of numerous light pen interrupts, until the buffer flag is again cleared to zero. This permits the program to process the data before it is destroyed by another interrupt.

The buffer structure looks like this:

```
        Buffer Flag
        Name
        Subpicture Tag
        Display Program Counter (DPC)
        Display Status Register (DSR)
        X Status Register (XSR)
        Y Status Register (YSR)
```

The Name value is the contents of the software Name Register (described in A.3.5) at the time of interrupt. The Tag value is the tag of the subpicture being displayed at the time of interrupt. The last four data items are the contents of the display processor status registers at the time of interrupt. They are described in detail in Table A-1.

Macro Call: .LPEN baddr

where:	baddr	is the address of the 7-word light pen status data buffer.

Errors:

None.

If a .LPEN was already issued and a buffer specified, the new buffer address replaces the previous buffer address. Only one light pen buffer can be in use at a time.

Example:

```
                .INSRT      #LFILE      ;DISPLAY LFILE
                .LPEN       #LBUF       ;SET UP LPEN BUFFER
        LOOP:   TST         LBUF        ;TEST LBUF FLAG, WHICH
                BEQ         LOOP        ;WILL BE SET NON-ZERO
                                        ;ON LIGHT PEN HIT.
        ;PROCESS DATA IN LBUF HERE.
                                        ;DATA IN LBUF
                CLR         LBUF        ;CLEAR THE BUFFER FLAG
                                        ;PERMITTING ANOTHER "HIT"
                BR          LOOP        ;GO WAIT FOR IT
```

```
LBUF:       .BLKW     7          ;SEVEN WORD LPEN BUFFER
LFILE:
                 .
                 .
```

Table A-1
Description of Display Status Words

Bits	Significance
DISPLAY PROGRAM COUNTER (DPC=172000)	
0-15	Address of display processor program counter at time of interrupt.
DISPLAY STATUS REGISTER (DSR=172002)	
0-1	Line Type
2	Spare
3	Blink
4	Italics
5	Edge Indicator
6	Shift Out
7	Light Pen Flag
8-10	Intensity
11-14	Mode
15	Stop Flag
X STATUS REGISTER (XSR=172004)	
0-9	X Position
10-15	Graphplot Increment
Y STATUS REGISTER (YSR=172006)	
0-9	Y Position
10-15	Character Register

A.2.6 .NAME

The .NAME request has been added to the Version 03 Display File Handler. The contents of the name register are now stacked when a subpicture call is made. When a light pen interrupt occurs, the contents of the name register stack may be recovered if the user program has supplied the address of a buffer through the **.NAME** request.

The buffer must have a size equal to the stack depth (default is 10) plus one word for the flag. When the .NAME request is entered, the address of the buffer is passed to the Handler and the first word (the flag word) is cleared. When a light pen hit occurs, the stack's contents are transferred and the flag is set non-zero.

Macro Call: .NAME baddr

where:	baddr	is the address of the name register buffer.

Errors:

None.

If a .NAME request has been previously issued, the new buffer address replaces the previous buffer address.

A.2.7 .REMOV

The .REMOV request removes the call to a user display file previously inserted in the handler's display file by the .INSRT request. All reference to the user file is removed, unlike the .BLANK request, which merely bypasses the call while leaving it intact.

Macro Call: .REMOV faddr

where:	faddr	is the address of the display file to be removed.

Errors:

No errors are returned. If the file address given cannot be found, the request is ignored.

A.2.8 .RESTR

The .RESTR request restores a user display file that was previously blanked by a .BLANK request. It removes the by-pass of the call to the user file, so that the display processor once again cycles through the user file.

Macro Call: .RESTR faddr

where:	faddr	is the address of the user file that is to be restored to view.

Errors:

No errors are returned. If the file specified cannot be found, the request is ignored.

A.2.9 .SCROL

This request is used to modify the appearance of the Display Monitor's text display. The .SCROL request permits the programmer to change the maximum line count, intensity and the position of the top line of text of the scroller. The request passes the address of a two-word buffer which contains the parameter specifications. The first byte is the line count, the second byte is the intensity, and the second word is the Y position. Line count, intensity and Y position must all be octal numbers. The intensity may be any number from 0 to 7, ranging from dimmest to brightest. (If an intensity of 0 is specified, the scroller text will be almost unnoticeable at a BRIGHTNESS knob setting less than one-half). The scroller parameter change is temporary, since an .UNLNK or CTRL/C restores the previous values.

Macro Call: .SCROL baddr

where:	baddr	is the address of the two-word scroll parameters buffer.

Errors:

No errors are returned. No checking is done on the values of the parameters. A zero argument is interpreted to mean that the parameter value is not to be changed. A negative argument causes the default parameter value to be restored.

Example:

```
                .SCROL  #SCBUF  ;ADJUST SCROLL PARAMETERS
                .
                .
                .

        SCBUF:  .BYTE 5         ;DECREASE #LINES TO 5.
                .BYTE 0         ;LEAVE INTENSITY UNCHANGED.
                .WORD 300       ;TOP LINE AT Y=300.
```

A.2.10 .START

The .START request starts the display processor if it was stopped by a .STOP directive. If the display processor is running, it is stopped first, then restarted. In either case, the subpicture stack is cleared and the display processor is started at the top of the handler's internal display file.

Macro Call: .START

Errors:

None.

A.2.11 .STAT

The .STAT request transfers the address of a seven-word status buffer to the display stop interrupt routine in VTBASE. Once the transfer has been made, display processor status data is transferred to the buffer by the display stop interrupt routine in VTBASE whenever a .DSTAT or .DHALT instruction is encountered (see Sections A.3.3 and A.3.4). The transfer is made only when the buffer flag is clear (zero). After the transfer is made, the buffer flag is set non-zero and the .DSTAT or .DHALT instruction is replaced by a .DNOP (Display NOP) instruction.

The status buffer must be a seven-word, contiguous block of memory. Its contents are the same as the light pen status buffer. For a detailed description of the buffer and an explanation of the status words, see Section A.2.5 and Table A-1.

Macro Call: .STAT baddr

where:	baddr	is the address of the status buffer receiving the data.

Errors:

No errors are indicated. If a buffer was previously set up, the new buffer address is replaced as the old buffer address.

A.2.12 .STOP

The .STOP request "stops" the display processor. It actually effects a stop by preventing the DPU from cycling through any user display files. It is useful for stopping the display during modification of a display file, a risky task when the display processor is running. However, a .BLANK could be equally useful for this purpose, since the .BLANK request does not return until the display processor has been removed from the user display file being blanked.

Macro Call: .STOP

Errors:

None.

NOTE

Since the display processor must cycle through the text buffer in the Display Monitor in order for console output to be processed, the text buffer remains visible after a .STOP request is processed, but all user files disappear.

A.2.13 .SYNC/.NOSYN

The .SYNC and .NOSYN requests provide program access to the power line synchronization feature of the display processor. The .SYNC request enables synchronization and the .NOSYN request disables it (the default case).

Synchronization is achieved by stopping the display and restarting it when the power line frequency reaches a trigger point, e.g., a peak or zero-crossing. Synchronization has the effect of fixing the display refresh time. This may be useful in some cases where small amounts of material are displayed but the amount frequently changes, causing changes in intensity. In most cases, however, using synchronization increases flicker.

```
Macro Calls:  .SYNC
              .NOSYN
```

Errors:

None.

A.2.14 .TRACK

The .TRACK request causes the tracking object to appear on the display CRT at the position specified in the request. The tracking object is a diamond-shaped display figure which is light-pen sensitive. If the light pen is placed over the tracking object and then moved, the tracking object follows the light pen, trying to center itself on the pen.

The tracking object first appears at a position specified in a two-word buffer whose address was supplied with the .TRACK request. As the tracking object moves to keep centered on the light pen, the new center position is returned to the buffer. A new set of X and Y values is returned for each light pen interrupt.

The tracking object cannot be lost by moving it off the visible portion of the display CRT. When the edge flag is set, indicating a side of the tracking object is crossing the edge of the display area, the tracking object stops until moved toward the center. To remove the tracking object from the screen, repeat the .TRACK request without arguments.

The .TRACK request may also include the address of a completion routine as the second argument. If a .TRACK completion routine is specified, the light pen interrupt handler passes control to the completion routine at interrupt level. The completion routine is called as a subroutine and the exit statement must be an RTS PC. The completion routine must also preserve any registers it may use.

Macro Call: .TRACK baddr, croutine

where:	baddr	is the address of the two-word buffer containing the X and Y position for the track object.
	croutine	is the address of the completion routine.

Errors:

None.

Example:

See Section A.10.

A.2.15 .UNLNK

The .UNLNK request is used before exiting from a program. In the case where the scroller is present, .UNLNK breaks the link, established by .LNKRT, between the Display File Handler's internal display file and the scroll file in the Display Monitor. The display processor is started cycling in the scroll text buffer, and no further graphics may be done until the link is established again. In the case where no scroller exists, the display processor is simply left stopped.

Macro Call: .UNLNK

Errors:

No errors are returned. An internal link flag is checked to determine if the link exists. If it does not exist, the request is ignored.

A.3 EXTENDED DISPLAY INSTRUCTIONS

The Display File Handler offers the assembly language graphics programmer an extended display processor instruction set, implemented in software through the use of the Load Status Register A (LSRA) instruction. The extended instruction set includes: subroutine call, subroutine return, display status return, display halt, and load name register.

A.3.1 DJSR Subroutine Call Instruction

The DJSR instruction (octal code is 173400) simulates a display subroutine call instruction by using the display stop instruction (LSRA instruction with interrupt bits set). The display stop interrupt handler interprets the non-zero word following the DJSR as the subroutine return address, and the second word following the DJSR as the address of the subroutine to be called. The instruction sequence is:

```
DJSR
Return address
Subroutine address
```

Example:

To call a subroutine SQUARE:

```
POINT          ;POSITION BEAM
100            ;AT (100,100)
100
DJSR           ;THEN CALL SUBROUTINE
.+4
SQUARE         ;TO DRAW A SQUARE
DRET
0
```

The use of the return address preceding the subroutine address offers several advantages. For example, the BASIC-11 graphics software uses the return address to branch around subpicture tag data stored following the subpicture address. This structure is described in Section A.5.3. In addition, a subroutine may be temporarily bypassed by replacing the DJSR code with a DJMP instruction, without the need to stop the display processor to make the by-pass.

The address of the return address is stacked by the display stop interrupt handler on an internal subpicture stack. The stack depth is conditionalized and has a default depth of 10. If the stack bottom is reached, the display stop interrupt handler attempts to protect the system by rejecting additional subroutine calls. In that case, the portions of the display exceeding the legal stack depth will not be displayed.

A.3.2 DRET Subroutine Return Instruction

The DRET instruction provides the means for returning from a display file subroutine. It uses the same octal code as DJSR, but with a single argument of zero. The DRET instruction causes the display stop interrupt handler to pop its subpicture stack and fetch the subroutine return address.

Example:

```
          SQUARE:  LONGV            ;DRAW A SQUARE
                   100!INTX
                   0
                   0!INTX
                   100
                   100!INTX!MINUSX
                   0
                   0!INTX
                   100!MINUSX
                   DRET             ;RETURN FROM SUBPICTURE
                   0
```

A.3.3 DSTAT Display Status Instruction

The DSTAT instruction (octal code is 173420) uses the LSRA instruction to produce a display stop interrupt, causing the display stop interrupt handler to return display status data to a seven-word user status buffer. The status buffer must first have been set up with a .STAT macro call (if not, the DSTAT is ignored and the display is resumed). The first word of the buffer is set non-zero to indicate the transfer has taken place, and the DSTAT is replaced with a DNOP (display NOP). The first word is the buffer flag and the next six words contain name register contents, current subpicture tag, display program counter, display status register, display X register, and display Y register. After transfer of status data, the display is resumed.

A.3.4 DHALT Display Halt Instruction

The DHALT instruction (octal code is 173500) operates similarly to the DSTAT instruction. The difference between the two instructions is that the DHALT instruction leaves the display processor stopped when exiting from the interrupt. A status data transfer takes place provided the buffer was initialized with a .STAT call. If not, the DHALT is ignored.

Example:

```
                   .STAT     #SBUF           ;INIT BUFFER
                   MOV       #DHALT,STPLOC   ;INSERT DHALT
                   .INSRT    #DFILE          ;DISPLAY THE PICTURE
```

```
1$:       TST       SBUF          ;DHALT PROCESSED?
          BEQ       1$            ;NO, WAIT
           .
           .
SBUF:     .BLKW 7                 ;STATUS BUFFER
DFILE:    POINT                   ;POSITION NEAR TOP OF 12" TUBE
          .WORD     500,1350
          LONGV                   ;DRAW A LINE, MAYBE OVER EDGE
          .WORD     0,400         ;IF IT IS A 12" SCOPE.
STPLOC:   DNOP                    ;STATUS WILL BE RETURNED AT TH
          DRET
          0
```

A.3.5 DNAME Load Name Register Instruction

The Display File Handler provides a name register capability through the use of the display stop interrupt. When a DNAME instruction (octal code is 173520) is encountered, a display stop interrupt is generated. The display stop handler stores the argument following the DNAME instruction in an internal software register called the "name register". The current name register contents are returned whenever a DSTAT or DHALT is encountered, and more importantly, whenever a light pen interrupt occurs. The use of a "name" (with a valid range from 1 to 77777) enables the programmer to label each element of the display file with a unique name, permitting the easy identification of the particular display element selected by the light pen.

The name register contents are stacked on a subpicture call and restored on return from the subpicture.

Example:

The SQUARE subroutine with "named" sides.

```
SQUARE:   DNAME               ;NAME IS
          10                  ;10
          LONGV               ;DRAW A SIDE
          100!INTX
          0
          DNAME               ;THIS SIDE IS NAMED
          11                  ;11
          0!INTX              ;STILL IN LONG VECTOR MODE
          100
          DNAME
          12
          100!INTX!MINUSX
          0
          DNAME
          13
          0!INTX
          100!MINUSX
          DRET                ;RETURN FROM SUBPICTURE
          0
```

A.4 USING THE DISPLAY FILE HANDLER

Graphics programs which intend to use the Display File Handler for display processor management can be written in MACRO assembly language. The display code portions of the program may use the mnemonics described in Section A.7. Calls to the Handler should have the format described in Section A.6.

The Display File Handler is supplied in two pieces, a file of MACRO definitions and a library containing the Display File Handler modules.

MACRO Definition File:	VTMAC.MAC	
Display File Handler:	VTLIB.OBJ	(consisting of:)
		VTBASE.OBJ
		VTCAL1.OBJ
		VTCAL2.OBJ
		VTCAL3.OBJ
		VTCAL4.OBJ

A.4.1 Assembling Graphics Programs

To assemble a graphics program using the display processor mnemonics or the Display Handler macro calls, the file VTMAC.MAC must be assembled with the program, and must precede the program in the assembler command string.

Example:

Assume PICTUR.MAC is a user graphics program to be assembled. An assembler command string would look like this:

```
          MACRO VTMAC+PICTUR/OBJECT
```

A.4.2 Linking Graphics Programs

Once assembled with VTMAC, the graphics program must be linked with the Display File Handler, which is supplied as a single concatenated object module,VTHDLR.OBJ. The Handler may optionally be built as a library, following the directions in A.8.5. The advantage of using the library when linking is that the Linker will select from the library only those modules actually used. Linking with VTHDLR.OBJ results in all modules being included in the link.

To link a user program called PICTUR.OBJ using the concatenated object module supplied with RT-11:

```
          LINK PICTUR,VTHDLR
```

To link a program called PICTUR.OBJ using the VTLIB library built by