Example:

```
                                  .TITLE ASYNCHRONOUS DIRECTORY OPERATION REQUEST EXAMPLE X01.01

                                  .MCALL  .LOOKUP,.SPFUN,.CLOSE,.PRINT,.EXIT

                                  ;DEFINITIONS

  000000

          177754                  ASYREQ  =       -20.    ;ASYNCHRONOUS REQUEST CODE
          000003                  LOOKUP  =       3       ;LOOKUP CODE FOR ASYNCHRONOUS REQUEST
          000004                  ENTER   =       4       ;ENTER CODE FOR ASYNCHRONOUS REQUEST
          000000                  CHAN    =       0       ;USE CHANNEL 0
          000001                  FNF     =       1       ;FILE NOT FOUND ERROR
          000000                  FSN     =       0       ;USE 0 FOR FILE SEQUENCE NUMBER

                                  ;MAGTAPE HANDLER IS ASSUMED TO BE LOADED.

  000000                          START:  .LOOKUP #AREA,#CHAN,#NFSBLK     ;OPEN A CHANNEL FOR THE NEXT REQUEST
  000024  103433                          BCS     LOOKER                  ;ERROR OCCURRED
  000026                                  .SPFUN  #AREA,#CHAN,#ASYREQ,#COMBLK,,#ERRBLK    ;DO A LOOKUP
  000074  103012                          BCC     FOUND                   ;NO ERRORS MEANS FILE WAS FOUND

  000076  022767  000001  000104          CMP     #FNF,ERRBLK     ;FILE NOT FOUND ERROR?
  000104  001411                          BEQ     NOTFIND         ;YES
  000106  012700  000320'                 MOV     #ASYERR,R0      ;NO
  000112  000410                          BR      CLOSE

  000114  012700  000220'         LOOKER: MOV     #LOOERR,R0      ;NFS .LOOKUP ERROR
  000120  000405                          BR      CLOSE

  000122  012700  000264'         FOUND:  MOV     #OK,R0          ;FILE FOUND MESSAGE
  000126  000402                          BR      CLOSE

  000130  012700  000300'         NOTFND: MOV     #NOK,R0         ;FILE NOT FOUND MESSAGE

  000134                          CLOSE:  .PRINT                  ;PRINT MESSAGE POINTED TO BY R0
  000136                                  .CLOSE  #CHAN
  000144                                  .EXIT

                                  ;DATA AREA

  000146                          AREA:   .BLKW   6               ;EMT ARGUMENT AREA
  000162  052140                  NFSBLK: .RAD50  /MT/            ;USE THIS TO OPEN MAGTAPE NON FILE STRUCTURED
  000164  000000  000000  000000          .WORD   0,0,0
  000172  023364  053665  100370 COMBLK:  .RAD50  /FILNAMTYP/     ;THIS IS THE FILE NAME WE'RE LOOKING FOR
  000200  000003                          .WORD   LOOKUP          ;THIS IS THE ASYNCHRONOUS OPERATION CODE FOR LOOKUP
  000202  000000                          .WORD   FSN             ;THIS IS THE FILE SEQUENCE NUMBER FOR THE LOOKUP
  000204  000000  000000                  .WORD   0,0             ;RESERVED (MUST BE ZERO)
  000210  000000  000000  000000 ERRBLK:  .WORD   0,0,0,0         ;PICK UP ERRORS HERE
  000216  000000

                                  ;MESSAGE AREA

  000220     116     117     116 LOOERR:  .ASCIZ  'NON FILE STRUCTURED .LOOKUP FAILED.'
  000264     106     111     114 OK:      .ASCIZ  'FILE FOUND.'
  000300     106     111     114 NOK:     .ASCIZ  'FILE NOT FOUND.'
  000320     101     123     131 ASYERR:  .ASCIZ  'ASYNCHRONOUS REQUEST ERROR.'

          000000'                         .END    START
```

Hardware Handler Functions

The hardware handler functions can be used with or without the file structure module.

1. Issuing hardware handler calls in a magtape file

   The magtape handler is designed to perform two distinct types of access. One type of access is file oriented and it attempts to make the magtape act like a disk; in other words, to make the magtape device be as device independent as possible. The other type of access allows access to the hardware commands such as read, write, space, etc., but the user doesn't have to know whether the magtape is a TM11 or TJU16.

   When accessing magtape using file oriented commands, the handler keeps track of the file sequence number where the tape is positioned. Tape movement during file searches can be optimized.

   When accessing data in a magtape file using the .READx/.WRITx requests, the magtape handler keeps track of the current block number as well as the last block number accessible. The block number argument can be used to simulate a random access device even on .ENTERed files.

The two access methods described above can be combined; that is, it is possible to use hardware handler tape movement commands on a magtape file. However, doing so causes the following to happen.

a. When the first hardware handler command is received, the stored file sequence number and block number information described above are erased and are not reinitialized until a .CLOSE and another file opening command have been performed. Note that the .CLOSE moves and, in the .ENTERed file case, writes the tape no matter what commands have been issued since the file was opened. Also note that the tape will no longer be an ANSI compatible tape. When the file is .CLOSEd, the magtape handler can't write out the size of the file because the file size is lost to the handler. It writes out a zero in its place. The file sequence number field will be correct.

b. The only exception to the above rule is when the user wishes to open the tape as file structured and write data blocks that are not the standard 512 (decimal) byte size that RT-11 magtape .WRITx commands use. The magtape handler keeps track of the number of blocks written and the end-of-file 1 label are correct as long as no commands other than the .SPFUN write command are used. Otherwise, the block size will be lost.

c. It is recommended that the user issue .SPFUN commands to a magtape file only for the case described in b. above.

2. Exception Reporting - Those .SPFUN's that are accepted by the hardware handler report end of file and hard error conditions through byte 52 in the system communication area. Additionally, they use the argument normally used for a block number as a pointer to a four-word error and status block to return qualifying information about exception conditions. When the block number argument is 0, no qualifying information is returned. Note that the contents of these words are undefined when no exception conditions have occurred (carry bit not set). The block is defined as follows:

   Words 1 and 2 are qualifying information.

   Words 3 and 4 are reserved, and must be set to 0.

   a. Qualifying information returned for the end of file condition is as follows:

	Code (Octal)	Condition
Word 1:	1	Tape at end of file only (tape mark detected)
	2	Tape at end of tape only (no tape mark detected)
	3	Tape at end of tape and end of file (tape mark detected)
	4	Tape at beginning of tape (no tape mark detected)

When a tape mark is detected during a spacing operation, the number of blocks not spaced is returned in the second word.

End of tape, tape mark and beginning of tape are returned as an end of file by the hardware handler.

b. Qualifying information returned for the hard error condition is as follows:

	Code (Octal)	Condition
Word 1:	0	No additional information
	1	Tape drive not available
	2	Tape position lost. When this error occurs, the tape should be rewound or backspaced to a known position.
	3	Nonexistent memory accessed
	4	Tape write-locked.
	5	Last block read had more information. The MM handler will return the number of words not read in the second word.
	6	Short block was read (the differences between the number of bytes (not words) requested and the number of bytes read is returned in the second word).

c. The hardware handler issues a hard error if it receives any request other than .LOOKUP (non-file-structured), .CLOSE, or any .SPFUN request not defined for the hardware handler.

d. When running under the XM monitor the blk area for error reporting must be mapped at all times.

3. Read/Write Physical Blocks of Any Size - The hardwarehandler reads and writes blocks of any size. Requests for reading and writing a variable number of words are implemented with two .SPFUN codes.

   a. The .SPFUN request to read a variable number of words in a block has the following form:

```
.SPFUN area,chan,#370,buf,wcnt,blk[,crtn]
```

where: 370	is the function code for a read operation
blk	is the address of a four-word error and status block used for returning the exception conditions.
crtn	is an optional argument that specifies a completion routine is to be entered after the request is executed.

This request returns the following errors. Additional qualifying information for these errors is returned in the first two words of the blk argument block.

Byte 52 error	Qualifying information
EOF (end of file) Value=0	Tape is at end of file only (tape mark detected) if bit 0 is set. Tape is at end of tape only (no tape mark detected) if bit 1 is set. Tape is at end of tape and end of file (tape mark detected) if bits 0,1 are set.
Hard Error (Value=1)	No additional information (Code=0)
	Tape drive is not available (Code=1)
	Tape position lost (Code=2)
	Nonexistent memory accessed (Code=3)
	Short block was read. The difference between the number of words requested and the number of words read is returned in the second word of blk (Code=6).
	The last block read had more information. For the TJU16 the number of words not read is returned in the second word of blk (Code=5).

b. The .SPFUN request to write a variable number of words to a block has the following form:

```
.SPFUN area,chan,#371,buf,wcnt,blk[,crtn]
```

where: 371 is the function code (decimal) for a write operation.

This request returns the following errors. Additional qualifying information for these errors is returned in the first two words of the blk argument block.

Byte 52 Error	Qualifying Information
EOF (end of file) (Value=0)	Tape is at end of tape only if bit 1 is set.
Hard error (Value=1)	No additional information (Code=0) Tape drive not available (Code=1) Tape position lost (Code=2) Nonexistent memory accessed (Code=3) Tape is write locked (Code=4)

NOTE

The TJU16 tape drive can return a hard error if a write request with a word count less than 7 is attempted.

4. Space Forward/Backward - The hardware handler accepts a command that spaces forward or backward block-by-block or until a tape mark is detected. When a tape mark is detected, the handler reports it along with the number of blocks not skipped. These commands can be used to issue a space-to-tape mark command by passing a number greater than the maximum number of blocks on a tape. The tape is left positioned after the tape mark or the last block passed. There are two spacing requests, which have the following forms:

   a. Space forward by block

      ```
      .SPFUN area,chan,#376,,wcnt,blk[,crtn]
      ```

where:	376	is the function code for forward space operation.
	wcnt	is the number of blocks to space past (cannot exceed 65,534).
	crtn	is a completion routine to be entered when the operation is complete.

This request returns the following errors. Additional qualifying information for these errors is returned in the first two words of the blk argument block.

Byte 52 error	Qualifying information
EOF (end of file)	Tape is at end of file only (tape mark detected) if bit 0 is set Tape is at end of tape only (no tape mark detected) if bit 1 is set Tape is at end of tape and end of file (tape mark detected) if bits 0,1 are set

The second word in blk contains the number of blocks requested to be spaced (wcnt) minus the number of blocks spaced if a tape mark is detected. Otherwise its value is not defined.

Hard error	No additional information (Code=0) Tape drive not available (Code=1) Tape position lost (Code=2)

NOTE

Due to hardware restrictions it is recommended that no forward space commands be issued if the reel is positioned past the end of tape marker.

   b. Space backward by block:

      ```
      .SPFUN area,chan,#375,,wcnt,blk[,crtn]
      ```

where:	375	is the function code for a backspace operation.

This request returns the following errors and additional qualifying information is returned in the first two words of the blk argument block.

Byte 52 error	Qualifying information
EOF (end of file)	Tape is at end of file (tape mark detected) if bit 0 is set Tape is at end of tape (no tape mark detected) if bit 1 is set Tape is at end of tape and end of file (tape mark detected) if bit 0,1 are set Tape is at beginning of tape (no tape mark detected) if bit 2 is set

The second word in blk contains the number of blocks requested to be spaced (wcnt) minus the number of blocks actually spaced (including the tape mark) if a tape mark is detected. Otherwise, its value is not defined.

Hard error	No additional information (Code=0) Tape drive not available (Code=1) Tape position lost (Code=2)

5. Rewind - The handler accepts a rewind command, and rewinds the tape drive to the beginning of tape. The handler cannot accept other requests until the rewind operation is complete, but other handlers can be active during tape rewind. The rewind request has the following format:

```
.SPFUN area,chan,#373,,,blk[,crtn]
```

where:	373	is the function code for the rewind operation.
	crtn	is a completion routine to be entered when the operation is complete.

This request returns the following error, and additional qualifying information is returned in the blk argument block.

Byte 52 error	Qualifying information
Hard error	No additional information (Code=0) Tape drive not available (Code=1)

6. Rewind and Go Off Line - This request is the same as rewind except that it takes the tape drive off-line, and then rewinds to the beginning of tape. The handler is free to accept commands after the rewind is initiated. The rewind and go off-line request has the following format:

```
.SPFUN area,chan,#372,,,blk[,crtn]
```

where:	372	is the function code for the rewind and go off-line operation.
	crtn	

This request returns the same error codes and qualifying information as the rewind request.

7. Write With Extended Gap - This request allows writing on tapes with bad spots. This request is identical to the write request except that the function code for write with extended gap operation is 374.

   The errors for this request are identical to those for the write request.

8. Write Tape Mark - The hardware handler accepts a request to write a tape mark. This request has the following format:

   ```
   .SPFUN area,chan,#377,,,blk[,crtn]

   where: 377     is the function code for the write
                  tape mark operation.
   ```

   This request returns the following errors: Additional qualifying information is returned in the first two words of the blk argument block.

Byte 52 error	Qualifying information
EOF (end of file)	End of tape is detected if bit 1 is set.
Hard error	No additional information (Code=0) Tape drive not available (Code=1) Tape position lost (Code=2) Tape is write locked (Code=4)

9. Error Recovery Algorithm - Any errors detected during spacing operations cause the recovery attempt to be aborted and a hard (position) error is reported.

   a. Read Error Recovery - The hardware handler performs the following algorithm if a read parity error is detected.

      1. Backspaces over the block and rereads. When unsuccessful it is repeated until five read commands have failed.

      2. Backspaces five blocks, spaces forward four blocks, then reads the record.

      3. This entire sequence (steps 1 and 2) is repeated eight times or until the block is read successfully.

   b. Write Error Recovery - The hardware handler performs the following algorithm upon detection of a read after write parity error.

      1. Backspaces over one block.

      2. Erases three inches of tape and rewrites the block. In no case is an attempt made to rewrite the block over the bad spot, since, even if successful, the block could be marginal and cause problems at a later time.

      3. If the read after write still fails, the entire sequence (steps 1 and 2) are repeated. When 25 feet of erased tape have been written, a hard error is given.

10. Non-File-Structured .LOOKUP Request - The hardware handler accepts a non-file-structured .LOOKUP request. This function is necessary to open a channel to the device before any I/O operations can be executed. It causes the hardware handler to mark the drive busy so that no other channel can be opened to that drive until a .CLOSE is performed. This request has the following form:

    .LOOKUP area,chan,dblk,seqnum

    where: seqnum is an argument that specifies whether the tape is to be rewound or not. When this argument is 0, the tape is rewound. When this argument is -1, the tape is not rewound.

    This request returns the following errors.

Byte 52 code	Meaning
0 or 1	Not meaningful for this request.
2	Device in use. The drive being accessed is already attached to another channel.
3	Tape drive not available.
4	Illegal argument detected. The file name was not 0 or the seqnum had an argument that was not 0 or -1.

11. .CLOSE Request - The hardware handler accepts the .CLOSE request and causes the handler to mark the drive as available. This request has the following form:

    .CLOSE chan

12. SET Commands - The hardware handler accepts SET commands to set the track number, density and parity of the tape drive. These commands are fully described in Chapter 4 of the RT-11 System User's Guide.

13. Non-File-Structured .WRITx Request - The hardware handler accepts .WRITx requests that write a variable number of words to a block on tape. The block number field is ignored. This request has the following form:

    .WRITx area,chan,buf,wcnt[,,crtn]

    This request returns the following errors. Note that no additional qualifying information is available.

Byte 52 error	Meaning
EOF (end of file) (Value=0)	The end of tape marker has been sensed.
Hard error (Value=1)	This can mean any of the error conditions listed for the file-structured write request.

14. Non-File-Structured .READx Request - This request reads a variable number of words from a block on tape. It ignores the end of tape marker and only reports end of file when a tape mark is read. The block number field is ignored. The request has the following form:

```
.READx area,chan,buf,wcnt[,,crtn]
```

This request returns the following errors. Note that there is no additional qualifying information available.

Byte 52 error	Meaning
EOF (end of file) (Value=0)	Only reported if a tape mark is read. The end of tape marker will not cause end of file.
Hard error (Value=1)	This can mean any of the error conditions listed for the file-structured read request.

Writing Tapes On Other PDP-11 Operating Systems To Be Read By RT-11

RT-11 can read files written on other computer systems that support the DIGITAL standard (ANSI) for labels. Below are a few examples of how to write ANSI tapes on some common DIGITAL PDP11 operating systems. Keep in mind that there are other factors involved besides just the label and format compatibility. These include density, parity and number of tracks written on the tape.

Writing Tapes on RSTS/E

RSTS/E supports two types of magtape formats, DOS-11 and ANSI. In the following examples, dd represents the magtape handler name, either MM or MT. In order to ensure that an ANSI file structure is written, be sure to issue the following command:

ASSIGN ddn:.ANSI	(Allocates the device to the job and ensures that an ANSI file structure is used)
RUN $PIP ddn:/ZE/VID:xxxxxx	(PIP is used to initialize the tape; xxxxxx is the volume ID)
Really zero ddn:? Yes	(PIP prompts before initializing the tape)
PIP ddn:=FARQUA.MAC,VEG.TEC	(PIP is used to copy files to the tape
DEASSIGN ddn:	(Deallocates the device)

Writing Tapes on RSX-11/M

RSX-11/M needs the following commands to access a magtape.

ALL ddn:	(Allocates a drive)
INIT ddn:RT11	(Initializes the tape and gives the name "RT11" as the volume identifier)
MOU ddn:RT11	(Mounts the tape volume)
PIP ddn:=[13,10]F11PRE.MAC,ALLOC.MAC	(Copies files to the tape)
DMO ddn:RT11	(Dismounts the tape volume)
DEA ddn:	(Deassigns the drive)

Writing Tapes on RSX-11/D and IAS

INIT ddn:RT11	(Initializes the tape and gives the name "RT11" as the volume identifier)
MOU ddn:RT11	(Mounts a tape volume)

```
(For RSX-11/D use the PIP program to write files to the tape)
(For IAS use the COPY command)
DMO ddn:RT11      (Dismounts the tape volume)
```

The above examples are intended only as examples. For more complete information on the above systems consult the appropriate documentation.

The contents of files written under the RSX-11 and IAS systems do not necessarily correspond to those types of data files under RT-11. For example, under RT-11 text files consist of stream ASCII data (carriage return and line feed characters are imbedded in the text) whereas the other systems just mentioned use a different type of character storage. The user is urged to pay special attention to the contents of the files he wishes to transfer.

When writing files to be read under RT-11, the only block size the RT-11 PIP program reads is 512(decimal) characters/block. However, the RT-11 DIR program produces a directory for any compatible tape.

1.4.8.2 **Cassette Tape Handler (CT)** - The CT handler can operate in two modes: hardware mode and software mode. These names refer to the type of operation that can be performed on the device at a given time. Software mode is the normal mode of operation used when accessing the device through any of the RT-11 system programs. In software mode, the handler automatically attends to file headers and uses a fixed record length of 64 words to transfer data.

Hardware mode allows the user to read or write any format desired, using any record size. In this mode, the word count is taken as the physical record size.

When the handlers are initially loaded by either the .FETCH programmed request or the LOAD command, only software functions are permitted. To switch from software to hardware mode, either a rewind or a non-file-structured .LOOKUP must be performed. (A non-file-structured .LOOKUP is a .LOOKUP in which the first word of the file name is null.)

In software mode, the following functions are permitted:

.ENTER	- Open new file for output
.LOOKUP	- Open existing file for input and/or output
.DELETE	- Delete an existing file on the specified device.
.CLOSE	- Close a file that was opened with .ENTER or .LOOKUP
.READ/.WRITE	- Perform data transfer requests

In .ENTER, .LOOKUP, and .DELETE an optional file count parameter can be specified. Its meaning is as follows:

Count Argument	Meaning
=0	A rewind is done before the operation.
>0	No rewind is done. The value of the count is taken as a limit of how many files to look at before performing the operation (for example, a count of 2 looks at two files at most. A count of 1 looks at only the next file).

<0	A rewind is done. The absolute value of the switch is then used as the limit.

If the file indicated in the request is located before the limit is exhausted, the search succeeds at that point.

As an example, consider:

```
        .LOOKUP #AREA,#0,#PTR,#5
        BCS A1
           .
           .
AREA:   .BLKW 10.
PTR:    .RAD50 /CT0/
        .RAD50 /EXAMPLMAC/
```

In this case, the file count argument is +5, indicating that no rewind is to be done and that CT0 is to be searched for the indicated file (EXAMPL.MAC). If the file is not found after four files have been skipped, or if an end-of-tape occurs in that space, the search is stopped, and the tape is positioned either at the end of tape (EOT) or at the start of the fifth file. If the named file is found within the five files, the tape is positioned at its start. If the end of tape is encountered first, an error is generated.

As another example:

```
.LOOKUP #AREA,#0,#PTR,#-5
```

This performs a rewind, and then uses a file count of five in the same way the previous example does.

Handler Functions - The cassette handler performs the following functions:

1. .LOOKUP Request

   If the file name (or the first word of the file name) is null, the operation is considered to be a non-file-structured .LOOKUP. This operation puts the handler into hardware mode. A rewind is automatically done in this case.

   If the file name is not null, the handler tries to find the indicated file. .LOOKUP uses the optional file count as illustrated above. Only software functions are allowed.

2. .DELETE Request

   .DELETE eliminates a file of the designated name from the device. .DELETE also uses the file count argument, and can thus do a delete of a numbered file as well as a delete by name. When a file is deleted, an unused space is created there. However, it is not possible to reclaim that space, as it is when the device is random access. The unused spot remains until the volume is re-initialized and rewritten. If a file name is not present, a non-file-structured .DELETE is performed and the tape is zeroed.

3. .ENTER Request

   The .ENTER request creates a new file of the designated name on the device. This request uses the optional file count, and can thus enter a file by name or by number. If enter by name is done, the handler deletes any files of the same name.

If enter by number is done, the indicated number of files is skipped, and the tape is positioned at the start of the next file.

NOTE

> Care must be exercised in performing numbered .ENTERs, as it is possible to enter a file in the middle of existing files and thus destroy any files from the next file to the end of the tape.
>
> It is also possible to create more than one file with the same name, since .ENTER only deletes files of the same name it sees while passing down the tape. If an .ENTER is done with a count greater than 0, no rewind is performed before the file is entered. If a file of the same name is present at an earlier spot on the tape, the handler cannot delete it. A non-file-structured .ENTER performs the same function as a non-file-structured .LOOKUP but does not rewind the tape. Since both functions allow writing to the tape without regard to the tape's file structure, they should be used with care on a file-structured tape.

4. .CLOSE Requests

   .CLOSE terminates operations to a file on cassette and resets the handler to allow more .LOOKUPs, .ENTERs, or .DELETEs. If a .CLOSE is not performed on an entered file, the end of tape label will be missing and no new files can be created on that volume. In this case, the last file on the tape must be rewritten and closed to create a valid volume.

5. .READ/.WRITE Requests

   .READ and .WRITE requests are unique in that they can be done either in hardware or software mode. In software mode (file opened with .LOOKUP or .ENTER), records are written in a fixed size (64 words). The word count specified in the operation is translated to the correct number of records. On a .READ, the user buffer is filled with zeroes if the word count exceeds the amount of data available.

   Following is a discussion of how the various parameters for .READ/.WRITE are used.

   a. Block Number

      Only sequential operations are performed. If the block number is 0, the cassette is rewound to the start of the file. Any other block number is disregarded.

b. Word Count

If the word count is 0, the following conditions are possible:

If the block number is non-zero, the operation is actually a file name seek. The block number is interpreted as the file count argument, as discussed in the example of .LOOKUP. The buffer address should point to the Radix-50 equivalents of the device and file to be located. This feature essentially allows an asynchronous .LOOKUP to be performed. The standard .LOOKUP request does not return control to the user program until the tape is positioned properly, whereas this asynchronous version returns control immediately and interrupts when the file is positioned.

The user can then do a synchronous, positively numbered .LOOKUP to the file just positioned, thus avoiding a long synchronous search of the tape.

If the block number is 0, a cyclical redundancy check error occurs.

Following is a description of the allowed hardware mode functions for the handler, as well as examples of how to call them. In general, special functions are called by using the .SPFUN request; examples of usage follow each function. The special functions require a channel number as an argument. The channel must initially be opened with a non-file-structured .LOOKUP, which places the handler in hardware mode.

The general form of the .SPFUN request is:

```
     .SPFUN  area,chan,func,buf,wcnt,blk,crtn
```

where:

func is the function code to be performed.

The request format is:

```
R0 area:  32 chan

          blk
          buf
          wcnt
          func 377
          crtn
```

Cassette Special Functions

1. Rewind (Code = 373) - This request rewinds the tape to load point. This puts the unit in hardware mode in the same manner as a non-file-structured .LOOKUP where any of the other functions can be done. Unless a completion routine is specified, control does not return to the user until the rewind completes. This request has the following form:

   ```
   .SPFUN   area,#0,#373,#0,#0,#0,crtn
   ```

   where: crtn is a completion routine to be entered when the operation is complete. The other arguments are not required.

2. Last File (Code = 377) - This request rewinds the cassette and positions it immediately before the sentinel file (logical end-of-tape). The request form is the same as for rewind except that code 377 is used.

   ```
   .SPFUN  area,#0,#377,#0,#0,#0[,crtn]
   ```

3. Last Block (Code = 376) - This request rewinds one record.

   ```
   .SPFUN  area,#0,#376,#0,#0,#0[,crtn]
   ```

4. Next File (Code = 375) - This request spaces the cassette forward to the next file.

   ```
   .SPFUN  area,#0,#375,#0,#0,#0[,crtn]
   ```

5. Next Block (Code = 374) - This request spaces the cassette forward by one record.

   ```
   .SPFUN  area,#0,#374,#0,#0,#0[,crtn]
   ```

6. Write File Gap (Code = 372) - This request terminates a file written by the user program when in hardware mode.

   Sample Macro Call:

   ```
   .SPFUN  area,#0,#372,#0,#0,#0
   ```

   This writes a file gap synchronously, while:

   ```
   .SPFUN  area,#0,#372,#0,#0,#0,#1
   ```

   or

   ```
   .SPFUN  area,#0,#372,#0,#0,#0,crtn
   ```

   performs asynchronous write file gap operations.

Cassette End-of-File Detection - Since cassette is a sequential device, the handler for this device cannot know in advance the number of blocks in a particular file, and thus cannot determine if a particular read request is attempting to read past the end of file. User programs can use the following procedures to determine if the handler has encountered end of file in either software or hardware mode.

In software mode, if end of file is encountered during a read and some data is read the cassette handler will zero fill the rest of the buffer and return to the caller. The next read attempted on that channel returns with the carry bit set and with the error byte (absolute location 52) set to indicate an attempt to read past end-of-file.

In hardware mode, the cassette handler does not report end of file as it does in software mode. The best way that user programs can determine if a cassette read has encountered a file gap is to check the device status registers after each hardware mode read is complete.

Example:

```
        TACS=177500                     ;TA11 CONTROL AND STATUS REGISTER
        TAEOF=4000                      ;EOF BIT IN TACS
        TAEOT=20000                     ;EOT BIT IN TACS
            .
            .
            .
        .READW  #AREA,#CHNL,#BUFF,#400,BLKNUM  ;READ FROM CT
        BCS     EMTERR                  ;TEST ERRORS
        TST     @#TACS                  ;ERROR BIT SET IN TACS?
        BPL     NOERR                   ;IF PL - NO
        BIT     #TAEOF,@#TACS           ;YES - WAS IT END OF FILE?
        BNE     EOF                     ;IF NE - YES
            .
            .
            .
EOF:                                    ;CASSETTE END OF FILE ENCOUNTERED
            .
            .
```

If desired, both the EOF and EOT bits could be checked:

```
        BIT     #MTSEOF+MTSEOT,@#MTS ;MT EOF OR EOT?
```

or

```
        BIT     #TAEOF+TAEOT,@#TACS ;CT EOF OR EOT?
```

1.4.8.3 **Diskette Handlers (DX,DY)** - The .SPFUN request permits reading and writing of absolute sectors on the diskettes. The DY handler accepts an additional .SPFUN request to determine the size, in 256-word blocks, of the volume mounted in a particular unit. On double density diskettes, sectors are 128 words long. RT-11 normally reads and writes them in groups of two sectors. On single density diskettes, sectors are 64 words long. RT-11 normally reads and writes them in groups of four sectors. Sectors can be accessed individually through the .SPFUN request. The format of the request is as follows:

```
.SPFUN area,chan,code,buf,wcnt,blk,crtn
```

where:

code is the function to be performed:

377	Read physical sector
376	Write physical sector
375	Write physical sector with deleted data mark
374	unused
373	(DY only) determine device size, in 256-word blocks, of a particular volume

buf for functions 377, 376, 375:
is the location of a 129-word buffer (for double density diskettes) or a 65-word buffer (for single density diskettes). The first word of the buffer, the flag word, is normally set to 0.

If the first word is set to 1, a read on a physical sector containing a deleted data mark is indicated. The actual data area of the buffer extends from the second word to the end of the buffer.

for function 373:
buf is the location of a one-word buffer in which the size of the volume in the specified unit is returned. (For single density diskettes, 494 (decimal) is returned. For double density diskettes, 988 (decimal) is returned.)

wcnt for functions 377, 376, 375:
is the absolute track number, 0 through 76, to be read or written.

for function 373:
wcnt is unused and should be set to 1.

blk for functions 377, 376, 375:
is the absolute sector number, 0 through 26, to be read or written.

for function 373:
blk is unused and should be set to C.

The diskette should be opened with a non-file-structured .LOOKUP. Note also that the buf, wcnt, and blk arguments have different meanings when used with diskettes.

Sample Macro Call:

```
     .SPFUN  #RDLIST,#1,#377,#BUFF,#0,#7,#0
                                      ;PERFORM A
                                      ;SYNCHRONOUS SECTOR READ
                                      ;FROM TRACK 0, SECTOR 7
                                      ;INTO THE 65-WORD AREA BUFF
```

Each DX and DY handler can support two controllers, and each controller supports two drives. For example, if the RX01 handler is SYSGENed to support two controllers, it will support four devices: DX0, DX1, DX2 and DX3. DX0 and DX1 are drives 0 and 1 of the standard diskette at vector 264 and CSR 177170. DX2 and DX3 are drives 0 and 1 of the other controller. Note that only one I/O process can be active at one time even though there are two controllers. There is no overlapped I/O to the handler.

1.4.8.4 **Card Reader Handler (CR)** - The card reader handler can transfer data either as ASCII characters in DEC 026 or DEC 029 card codes (see Table 1-3) or as column images controlled by the SET command. In ASCII mode (SET CR NOIMAGE), invalid punch combinations are decoded as the error character 134(octal)--backslash. In IMAGE mode, no punch combination is invalid; each column is read as 12 bits of data right-justified in one word of the input buffer. The handler continues reading until the transfer word count is satisfied or until a standard end-of-file card is encountered (12-11-0-1-6-7-8-9 punch in column 1; the rest of the card is arbitrary). On end-of-file, the buffer is filled with zeroes and the request terminates successfully; the next input request from the card reader gives an end-of-file error. Note that if the transfer count is satisfied at a point that is not a card boundary, the next request continues from the middle of the card with no loss of information. If the input hopper is emptied before the transfer request is complete, the handler hangs until the hopper is reloaded and the "START" button on the reader is pressed again. The transfer then continues until completion or until another hopper empty condition exists. End-of-file is not reported on the hopper empty condition. The handler hangs if the hopper empties during the transfer regardless of the status of the SET CR HANG/NO

HANG option. No special action is required to use the card reader handler with the CM 11 mark sense card reader. The program should be aware of the fact that mark sense cards can contain less than 80 characters. Note also that when the CR handler is set to CRLF or TRIM and is reading in IMAGE mode, unpredictable results can occur.

Table 1-3
DEC 026/DEC 029 Card Code Conversions

Zone	Digit	Octal	Character	Name
none	none	040		SPACE
	1	061	1	digit 1
	2	062	2	digit 2
	3	063	3	digit 3
	4	064	4	digit 4
	5	065	5	digit 5
	6	066	6	digit 6
	7	067	7	digit 7
12 (DEC 029)	none	046	&	ampersand
(DEC 026)		053	+	plus sign
	1	101	A	upper case A
	2	102	B	upper case B
	3	103	C	upper case C
	4	104	D	upper case D
	5	105	E	upper case E
	6	106	F	upper case F
	7	107	G	upper case G
11	none	055	-	minus sign
	1	112	J	upper case J
	2	113	K	upper case K
	3	114	L	upper case L
	4	115	M	upper case M
	5	116	N	upper case N
	6	117	O	upper case O
	7	107	P	upper case P
0	none	060	0	digit 0
	1	057	/	slash
	2	123	S	upper case S
	3	124	T	upper case T
	4	125	U	upper case U
	5	126	V	upper case V
	6	127	W	upper case W
	7	130	X	upper case X
8	none	70	8	digit 8
	1	140	`	accent grave
(DEC 029)	2	072	:	colon
(DEC 026)		137	_	backarrow (underscore)
(DEC 029)	3	043	#	number sign
(DEC 026)		075	=	equal sign
	4	100	@	commercial "at"
(DEC 029)	5	047	'	single quote
(DEC 026)		136	^	uparrow (circumflex)

(continued on next page)

Table 1-3 (Cont.)
DEC 026/DEC 029 Card Code Conversions

Zone	Digit	Octal	Character	Name
(DEC 029)	6	075	=	equal sign
(DEC 026)		047	'	single quote
(DEC 029)	7	042	"	double quote
(DEC 026)		134	\	backslash
9	none	071	9	digit 9
	2	026	CTRL/V	SYN
	7	004	CTRL/D	EOT
12-11	none	174	|	vertical bar
	1	152	j	lower-case J
	2	153	k	lower-case K
	3	154	l	lower-case L
	4	155	m	lower-case M
	5	156	n	lower-case N
	6	157	o	lower-case O
	7	160	p	lower-case P
12-0	none	173	{	open brace
	1	141	a	lower-case A
	2	142	b	lower-case B
	3	143	c	lower-case C
	4	144	d	lower-case D
	5	145	e	lower-case E
	6	146	f	lower-case F
	7	147	g	lower-case G
12-8	none	110	H	upper-case H
(DEC 029)	2	133	[	open square bracket
(DEC 026)		077	?	question mark
	3	056	.	period
(DEC 029)	4	074	<	open angle bracket
(DEC 026)		051	)	close parenthesis
(DEC 029)	5	050	(	open parenthesis
(DEC 026)		135	]	close square bracket
(DEC 029)	6	053	+	plus sign
(DEC 026)		074	<	open angle bracket
	7	041	!	exclamation mark
12-9	none	111	I	upper-case I
	1	001	CTRL/A	SOH
	2	002	CTRL/B	STX
	3	003	CTRL/C	ETX
	5	011	CTRL/I	HT
	7	177		DEL
11-0	none	175	}	close brace
	1	176	~	tilde
	2	163	s	lower-case S
	3	164	t	lower-case T
	4	165	u	lower-case U
	5	166	v	lower-case V
	6	167	w	lower-case W
	7	170	x	lower-case X

(continued on next page)

Table 1-3 (Cont.)
DEC 026/DEC 029 Card Code Conversions

Zone	Digit	Octal	Character	Name
11-8				
	none	121	Q	upper-case Q
(DEC 029)	2	135	]	close square bracket
(DEC 026)		072	:	colon
	3	044	$	currency symbol
	4	052	*	asterisk
(DEC 029)	5	051	)	close parenthesis
(DEC 026)		133	[	open square bracket
(DEC 029)	6	073	;	semi-colon
(DEC 026)		076	>	close angle bracket
(DEC 029)	7	136	^	uparrow (circumflex)
(DEC 026)		046	&	ampersand
11-9				
	none	122	R	upper-case R
	1	021	CTRL/Q	DC1
	2	022	CTRL/R	DC2
	3	023	CTRL/S	DC3
	6	010	CTRL/H	BS
0-8				
	null	131	Y	upper-case Y
(DEC 029)	2	134	\	backslash
(DEC 026)		073	;	semi-colon
	3	054	,	comma
(DEC 029)	4	045	%	percent sign
(DEC 026)		050	(	open parenthesis
(DEC 029)	5	137	_	backarrow (underscore)
(DEC 026)		042	"	double quote
(DEC 029)	6	076	>	close angle bracket
(DEC 026)		043	#	number sign
(DEC 029)	7	077	?	question mark
(DEC 026)		045	%	percent sign
0-9				
	null	132	Z	upper-case Z
	5	012	CTRL/J	LF
	6	027	CTRL/W	ETB
	7	033		ESC
9-8				
	4	024	CTRL/T	DC4
	5	025	CTRL/U	NAK
	7	032	CTRL/Z	SUB
12-9-8				
	3	013	CTRL/K	VT
	4	014	CTRL/L	FF
	5	015	CTRL/M	CR
	6	016	CTRL/N	SO
	7	017	CTRL/O	SI
11-9-8				
	none	030	CTRL/X	CAN
	1	031	CTRL/Y	EM
	4	034	CTRL/\	FS
	5	035	CTRL/]	GS
	6	036	CTRL/^	RS
	7	037	CTRL/_	US

(continued on next page)

Table 1-3 (Cont.)
DEC 026/DEC 029 Card Code Conversions

Zone	Digit	Octal	Character	Name
0-9-8				
	5	005	CTRL/E	ENQ
	6	006	CTRL/F	ACK
	7	007	CTRL/G	BEL
12-0-8	none	150	h	lower-case H
12-0-9	none	151	i	lower-case I
12-11-8	none	161	q	lower-case Q
12-11-9	none	162	r	lower-case R
11-0-8	none	171	y	lower-case Y
11-0-9	none	172	z	lower-case Z
12-11-9-8				
	1	020	CTRL/P	DLE
12-0-9-8				
	1	000		NUL

1.4.8.5 **High-Speed Paper Tape Reader/Punch** (PC) - RT-11 provides support of the PR11 High Speed Reader and the PC11 High Speed Reader/Punch through the PC handler. The PC handler distributed with the system supports both the paper tape reader and punch. A handler supporting only the paper tape reader can be created during SYSGEN. The PC handler does not print an ^ on the terminal when it is entered for input the first time, as did the PR handler for earlier releases of RT-11. The tape must be in the reader when the command is issued, or an input error occurs. This prohibits any two-pass operations from being done using PC as an input device. For example, linking and assembling from PC does not work; an input error occurs when the second pass is initiated. The correct procedure is to transfer the paper tape to disk or DECtape, and then perform the operation on the transferred file.

On input, the PC handler zero fills the buffer when no more tape is available to read. On the next read request to the PC handler, the end-of-file bit in byte 52 is set and the C bit is set on return from the I/O completion.

1.4.8.6 **Console Terminal Handler (TT)** - The console terminal can be used as a peripheral device by using the TT handler. Note that:

1. An ^ is typed when the handler is ready for input.

2. CTRL/Z can be used to specify the end of input to TT. No carriage return is required after the CTRL/Z. If CTRL/Z is not typed, the TT handler accepts characters until the word count of the input request is satisfied.

3. CTRL/O, struck while output is directed to TT, causes an entire output buffer (all characters currently queued) to be ignored.

4. A single CTRL/C struck while typing input to TT causes a return to the monitor. If output is directed to TT, a double CTRL/C is required to return to the monitor if FB is running. If the SJ monitor is running, only a single CTRL/C is required to terminate output.

5. The TT handler can be in use for only one job (foreground or background) at a time, and for only one function (input or output) at a time. The terminal communication for the job not using TT is not affected at all.

6. The user can type ahead to TT; characters are obtained from the input ring buffer before the keyboard is referenced. The terminating CTRL/Z can also be typed ahead.

7. If the mainline code of a job is using TT for input, and a completion routine does a .TTYIN, typed characters are passed unpredictably to the .TTYIN and TT. Therefore, this practice should be avoided.

8. If a job sends data to TT for output and then does a .TTYOUT or a .PRINT, the output from the latter is delayed until the handler completes its transfer. If a TT output operation is started when the monitor's terminal output ring buffer is not empty (before the print-ahead is complete), the handler completes the transfer operation before the buffer contents are printed.

9. The TT handler does not interface to terminals other than the assigned console terminal in a multi-terminal system.

1.4.8.7 **RK06/07 Disk Handler (DM)** - The RT-11 RK06/07 handler has some features that are not standard for most RT-11 handlers. Among these non-standard features are the following:

1. Support of bad block replacement.

2. .SPFUN requests to read and write absolute blocks on disk.

3. .SPFUN request to initialize the bad block replacement table.

4. .SPFUN error return information.

5. .SPFUN request to determine the size of a volume mounted in a particular device unit. (The RK06 and RK07 disks share the same controller and handler. The RK07 has twice as many blocks as the RK06 volume.)

These features are discussed further in the following sections.

1. Bad block replacement - The last cylinder of the RK06 and RK07 disks is used for bad block replacement and error information. RT-11 supports a maximum of 32 bad blocks on these disks. The bad block information is stored in block 1 on track 0, cylinder 0, of the disk. The replacement blocks are stored on tracks 0 and 1 of the last cylinder. A bad block replacement table is created in block 1 of the disk by the DUP utility program when the disk is initialized. When a bad block is encountered and the table is not present in the handler from the same volume, the DM handler reads a replacement table from block 1 of the disk and stores it in the handler.

When a bad sector error (BSE) or header validity error (HVRC) is detected during a read or write, the DM handler replaces the bad block with a good block from the replacement tracks. The bad block replacement feature of RT-11 requires blocks 0 through 5 and tracks 0 and 1 of the last cylinder to be good. This procedure causes an I/O delay since the read/write heads must move from their present position on the disk to the replacement area, and back again.

If this I/O delay cannot be tolerated, the disk can be initialized without bad block replacement. In this case, bad blocks are covered by .BAD files. Neither the bad blocks nor the replacement tracks will be accessed. The advantage of using bad block replacement is that the entire disk appears to be good. If .BAD files are used instead, the disk becomes fragmented around the bad blocks.

Only BSE and HVRC errors trigger the DM handler's bad block replacement mechanism. If a bad block develops that is not a BSE or HVRC error, the disk must be reformatted to have this new block included in the replacement mechanism. Reformatting should detect the new bad block, mark it so that it generates a BSE or HVRC error, and add the block number to the bad block information on the disk. The disk should then be initialized to add the bad block to the replacement table.

2. .SPFUN Requests - The RK06/07 handler accepts the .SPFUN request with the following function codes:

   377 - for a read operation
   376 - for a write operation
   374 - for initializing the bad block replacement table in the handler.
   373 - for determining the size, in 256-word blocks, of a particular volume.

   The format of the .SPFUN request is the same as explained in Chapter 2 except as follows: for function codes 377 and 376, the buffer size for reads and writes must be one word larger than required for the data. The first word of the buffer contains the error information returned from the .SPFUN request. This information is returned for a .SPFUN read or write, and the data transferred follows the error information. The error codes and information are as follows:

Code	Meaning
100000	If the I/O operation is successful
100200	If a bad block is detected (BSE error)
100001	If an ECC error is corrected
100002	If an error recovered on retry
100004	If an error recovered through an offset retry
100010	If an error recovered after recalibration
1774xx	If an error did not recover

For function code 374, the buf, wcnt, and blk arguments should be 0. For function code 373, buf is a one-word buffer where the size of the specified volume in 256-word blocks is returned. The wcnt argument should be 1 and the blk argument should be 0.

1.4.8.8 **Null Handler (NL)**

- The null handler can accept all read/write requests. On output operations this handler acts as a data sink. When NL is called, it returns immediately to the monitor indicating that the output is complete. The NL handler returns no errors and causes no interrupts. On input operations NL returns an end-of-file indication for all requests and no data is transferred. Hence, the contents of the input buffer are unchanged.

1.4.8.9 **RL01 Disk Handler (DL)**

- The RL01 disk handler includes the following special features:

1. .SPFUN requests to read and write absolute blocks on the disk (without invoking the bad block replacement scheme).
2. Support of automatic bad block replacement.
3. .SPFUN request to initialize the bad block replacement table.
4. .SPFUN request to determine the size of a volume mounted in a particular device unit.

The .SPFUN requests are as follows:

377 -	for a read operation
376 -	for a write operation
374 -	for initializing the bad block replacement table in the handler
373 -	for determining the size, in 256-word blocks, of a particular volume

Unlike the DM handler, the read and write .SPFUN requests for the DL handler do not return an error status in the first word of the buffer.

See the description of the .SPFUN programmed request in Chapter 2 for details on the special functions.

Bad block replacement for the RL01 is similar to the bad block support for the RK06 and RK07. However, the RL01 device generates neither the bad sector error (BSE) nor the header validity error (HVRC). Therefore, the handler must check the bad block replacement table for each I/O transfer. Since the table is always in memory as part of the DL handler, the I/O delay is not significant.

The last track of the RL01 disk contains a table of the bad sectors that were discovered during manufacture of the disk. The ten blocks preceding this table (the last ten blocks in the second-to-last track) are set aside for bad block replacements. The maximum number of bad blocks, ten, is defined in the handler.

As with the RK06 and RK07, the user determines at initialization time whether to cover bad blocks with .BAD files or to create a replacement table for them and substitute good blocks during I/O transfers. The advantage of using bad block replacement is that it makes a disk with some bad blocks appear to have none. On the other hand, covering bad blocks with .BAD files fragments the disk. Because RT-11 files must be stored in contiguous blocks, this fragmentation limits the size of the largest file that can be stored.

If the /REPLACE option is specified during initialization of an RL01 disk, DUP scans the disk for bad blocks. It merges the scan information with the manufacturing bad sector table, allocates a replacement for each bad block, and writes a table of the bad blocks and their replacements in the first 20 words of block 1 of the disk. Block 1 is a table of two-word entries. The first word is the block number of a bad block; the second word is its allocated replacement. The last entry in the table is a zero word. The entries in the table are in order by ascending bad block number. A sample table is as follows:

Bad block	12	Word 0
Its replacement	10210	
	37	Word 2
	10211	
	553	Word 4
	10212	
End of list	0	Word 6

The handler contains space to hold a resident copy of the bad block table for each unit. The amount of space allocated is defined by the SYSGEN conditional DL$UN, which is the number of RL01 units to be supported. The value defaults to two if it is not defined. The handler reads the disk copy of the table into its resident area under the following three conditions:

1. If a request is passed to the handler and the table for that unit has not been read since the handler was loaded into memory.

2. If a request is passed to the handler and the handler detects Volume Check drive status. This status indicates that the drive spun down and spun up again, which means that the disk was probably changed.

3. If a .SPFUN 374 request is passed to the handler. This special function is used by DUP when it initializes the disk table to ensure that the handler has a valid resident copy.

1.5 MULTI-TERMINAL SUPPORT

The multi-terminal device handler supports from one to sixteen terminals. It is a SYSGEN option for FB and XM monitors that is integrated into the resident monitor (RMON) and console terminal service.

The multi-terminal service provides eight programmed requests as follows (see Chapter 2 for additional details):

Sub-Code	Request	Operation
0	.MTSET	Set terminal characteristics
1	.MTGET	Get terminal characteristics
2	.MTIN	Input characters from terminal
3	.MTOUT	Output characters to terminal

Sub-Code	Request	Operation
4	.MTRCTO	Reset CTRL/O flag
5	.MTATCH	Attach a terminal
6	.MTDTCH	Detach a terminal
7	.MTPRN	Print a line

Errors are returned in the error byte, location 52, as follows:

Error Codes	Meanings
0	No character in buffer (MTIN). No room in buffer (MTOUT).
1	Illegal unit number. The job did not attach it.
2	Non-existent unit number.
3	Illegal request - sub-code out of range.
4	Attempt to attach or detach a unit that is already attached to another job.
5	Buffer or status block is outside legal addressing range (XM monitor only).

The number and types of interfaces must be declared at SYSGEN time, then logical unit numbers (lun) are assigned to identify the terminals. Lun's are assigned in the following order:

1. hardware console interface (a local DL11)
2. other local mode DL11's
3. remote DL11 interfaces
4. local DZ11 lines
5. remote DZ11 lines

A unit control block, which associates a lun with a specific interface, is set up for each terminal. Terminals are referenced by the logical unit numbers. For example, logical unit number 0 is the default console lun and is assigned to the hardware console interface. The .TTYIN, .TTYOUT, .PRINT, .CSIGEN, .CSISPC, .GTLIN requests, and all TT references use the console; no TT support is provided for terminals other than the console. Hence, an .MTIN or .MTOUT executed with 0 as the logical unit number is directed to the console terminal. However, the terminal that the system uses as the console can be changed by the SET command as follows (provided that the terminal is a local DL11):

```
SET TT CONSOL=n
```

where:

n is a decimal value from 0 to 15 that indicates the logical unit number of the terminal to be used as the new console.

For example, the following command assigns terminal number 3, which is a local DL11, to the system hardware console interface:

```
     SET TT CONSOL=3
```

After this command is issued, .TTYIN, .TTYOUT, .PRINT, and any other requests directed to the console terminal will use terminal number 3.

The foreground and background jobs can either share a single console or they can have separate consoles. If a console is shared, only one job can attach it. Only the owner of the shared console can issue multi-terminal programmed requests to the terminal, but both jobs can issue .TTYIN, .TTYOUT, .CSIGEN, .CSISPC, .GTLIN, and .PRINT requests.

All other terminals must be attached by the job before they can be referenced and used. When the terminal becomes attached, it is dedicated to the job that issued the attach request except when the console must be shared by foreground and background jobs. The foreground job can have a separate console with a different lun assigned to it. This lun will be the default value for the .TTYIN, .TTYOUT, .CSIGEN, .CSISPC, .GTLIN, and .PRINT programmed requests. The assigning of the separate console is performed at load time by the FRUN option /T:lun. The separate console is not the primary system console and can only be considered an auxiliary console since KMON cannot communicate with it. This auxiliary foreground console must also be a local DL11 terminal interface and cannot be changed by the SET TT CONSOL command.

When a terminal is attached to a job, it remains attached until it is detached by a .MDTCH programmed request (see Chapter 2 for details), or until the job exits or is aborted. If the terminal is detached through a programmed request, the output in process at the terminal is allowed to finish before the terminal is detached. If the terminal is detached by aborting the job, the output is terminated and the terminal is detached immediately.

When a terminal is attached to a job, it has the following default characteristics:

80.	character column width
CRLF$	option enabled (generates LF after RET)
PAGE$	option enabled (XON/XOFF enabled)

These defaults can be changed by the .MTSET request.

An asynchronous terminal status (ATS) option is available and can be selected at SYSGEN time. This option provides the job with updated status of the terminal and modem. When the terminal is attached, the job can supply a status word that is updated as changes in the terminal status occur. The status bits and their meanings are as follows:

AS.CTC	100000	bit 15	Double CTRL/C struck
AS.INP	40000	bit 14	Input is available
AS.OUT	20000	bit 13	Output buffer empty
AS.CAR	200	bit 7	Carrier present (remote only)

The AS.CTC bit is set if a double CTRL/C is struck on any terminal except the job's console terminal. If a double CTRL/C is struck on the job's console terminal, the job is aborted unless an .SCCA request has been issued. In this case, bit 15 of the terminal status word is set. This bit must be reset by the job before further processing.

The AS.INP bit is set if input is available (a line of characters in normal mode or a single character in special mode). The bit is cleared when the characters are read.

The AS.OUT bit is set when the output ring buffer is empty (when the last character is printed). It is cleared when characters remain to be printed.

The AS.CAR bit is set when a remote line is answered. It is cleared when a remote line hangs up or drops a carrier.

All of the bits discussed in the previous section indicate significant events have occurred when they are set. These bits are set and cleared by the multi-terminal service, except AS.CTC, which must be cleared by the program when tested.

1.6 ERROR LOGGING

The error logging process keeps a statistical record of all I/O operations on devices that are supported by this feature. In addition to the statistics, the error logging process also detects and stores any errors that occur during the I/O operations. The following statistics for each supported device are recorded:

1. number of read successes
2. number of write successes
3. number of hard errors (unrecoverable errors)
4. number of soft errors (recoverable errors)

The following statistics for memory and cache are recorded:

1. number of memory parity errors
2. number of cache memory errors

In addition to the statistics listed above, the following information is retained if an error occurs in memory:

1. error sequence number
2. PC
3. PS
4. memory parity registers
5. cache error registers

The following information is retained if an error occurs on a supported peripheral device:

1. error sequence number
2. unit number
3. device ID (from $STAT)
4. queue element block number

5. queue element buffer address
6. queue element word count
7. device hardware registers
8. total retry count
9. retry countdown

1.6.1 The Error Logging Subsystem

The error logging process is implemented through an error logging subsystem consisting of four programs written in MACRO and FORTRAN (see Figure 1-4 and Table 1-4).

Table 1-4
Error Logging Subsystem Components

Program	Language	Function
Error Log Handler (EL)	MACRO-11	Reads and stores system errors and successful I/O operations for all supported devices.
Error Log Utility (ERRUTL)	MACRO-11	Creates a disk file (ERRTMP.SYS), writes out the data collected by the EL handler to the file ERRTMP.SYS, and queries for number of errors in EL.
Error log file Formatter (PSE)	MACRO-11	Formats the file produced by ERRUTL into a standard error log file named ERROR.DAT.
Error Summary/ Report Generator (SYE)	FORTRAN IV	Analyzes and writes out the contents of the standard error log file to a hard-copy or visual display device.

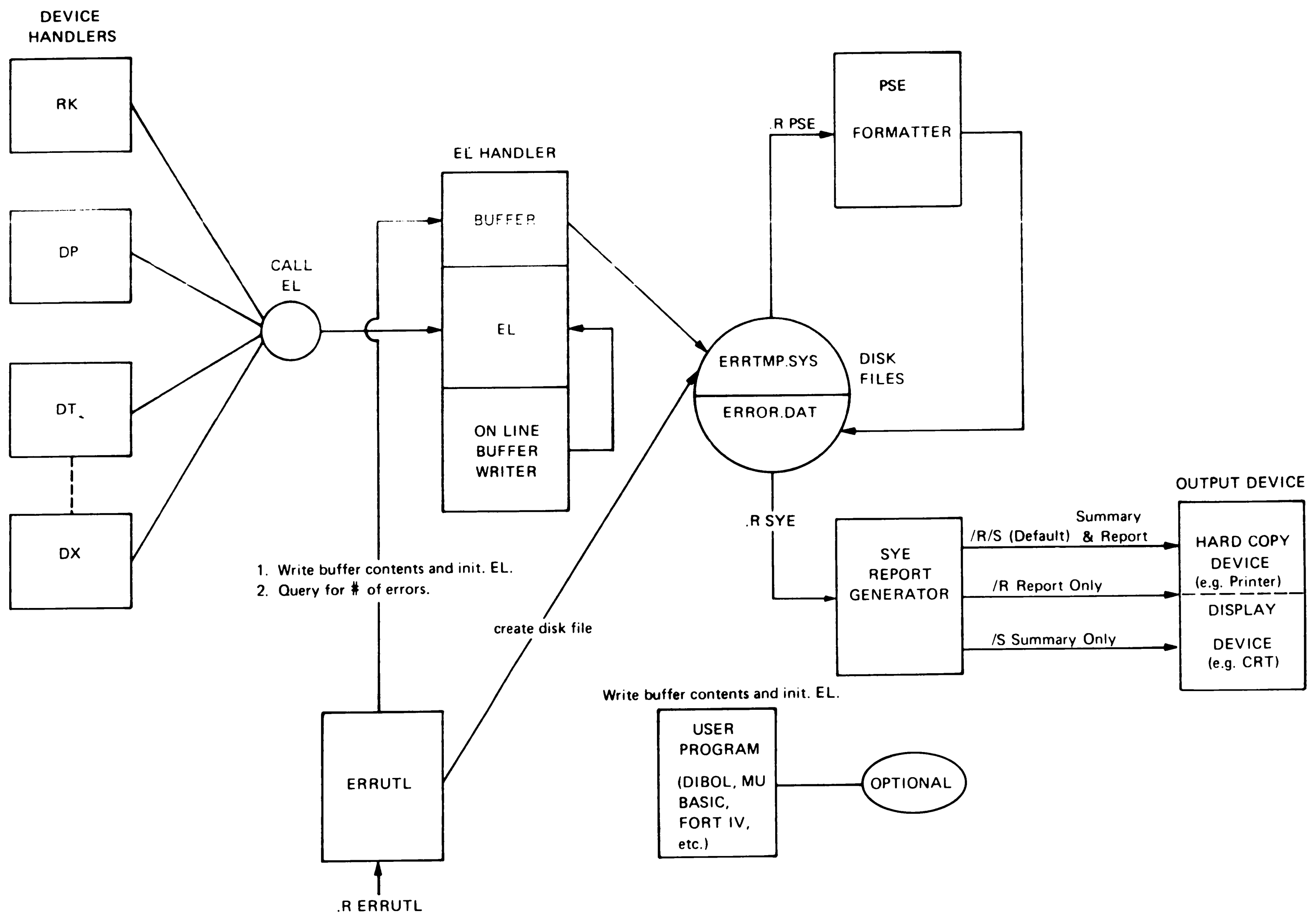

DEVICE HANDLERS
RK
DP
DT
DX
CALL EL
EL HANDLER
BUFFER
EL
ON LINE BUFFER WRITER
1. Write buffer contents and init. EL.
2. Query for # of errors.
create disk file
ERRUTL
.R ERRUTL
.R PSE
PSE FORMATTER
ERRTMP.SYS
ERROR.DAT
DISK FILES
.R SYE
SYE REPORT GENERATOR
/R/S (Default) Summary & Report
/R Report Only
/S Summary Only
OUTPUT DEVICE
HARD COPY DEVICE (e.g. Printer)
DISPLAY DEVICE (e.g. CRT)
Write buffer contents and init. EL.
USER PROGRAM (DIBOL, MU BASIC, FORT IV, etc.)
OPTIONAL

Figure 1-4 Error Logging Subsystem Functional Block Diagram

1.6.1.1 **The Error Log (EL) Handler** - The RT-11 EL handler is a MACRO-11 program that reads and stores errors and statistics and I/O operations. It consists of the following parts:

1. Information and pointer area
2. Buffer initializer
3. On-line memory-to-file routine
4. Statistics and error collector
5. Statistics buffer
6. Error log buffer

The functions for the various parts of the EL handler are discussed in the following section:

1. The information and pointer area consists of the following:
    a. An error buffer overflow counter containing the number of free words in the error log buffer.
    b. An offset pointer containing the byte offset to the statistics buffer from the EL load address.
    c. An offset pointer containing the byte offset to the error log buffer from the EL load address.
    d. The sequence number of the next error to be logged. If the value is equal to 1, it indicates that no error has been logged.
2. Buffer Initializer - This section of the EL handler is called to initialize the error log buffer. The error log buffer can be initialized in two ways:
    a. As a ring buffer to save the newest data.
    b. As a sequential buffer to save the oldest data.
3. On-Line Memory-To-File Routine - This part of the EL handler allows the user programs or system programs (such as a multi-user language system) to write the statistics buffer's and error log buffer's contents to the disk-resident error log file (ERRTMP.SYS). The program, however, must provide a channel and queue element to accomplish the write operation. In addition to this program-controlled method of writing the buffers' contents to the ERRTMP.SYS file, facilities are provided for accomplishing the same thing manually through a system utility program (ERRUTL).
4. Statistics and Error Collection - This section of the EL handler logs the read/write statistics and detects and stores the error information. The information is retained in two separate buffers until they are written to the disk-resident error log file (ERRTMP.SYS).
5. The statistics buffer stores information on I/O operations for all supported devices since the last time that the buffer was initialized. The information contained in this buffer is as follows:
    a. The number of successful read/write operations.

   b. The number of hard and soft I/O errors. A soft error is defined as one that recovered or corrected itself. A hard error is defined as one that did not recover or correct itself and was reported back to the program.

   c. The number of cache and memory parity errors.

6. The error buffer stores information on each hard or soft device error and parity or cache errors. The information contained in this buffer for a hard or soft device error is as follows:

   a. The error sequence number and error type

   b. The unit and device identification

   c. The device's block address

   d. The memory buffer address

   e. The word count

   f. The retry count

   g. The number and content of all pertinent hardware registers

   The information contained in the error buffer for a cache or memory parity error is as follows:

   a. The error sequence number and error type

   b. The PC and PS

   c. The memory parity registers or cache error registers

1.6.1.2 **The Error Utility Program (ERRUTL)** - ERRUTL is a utility program that performs the following operations:

1. Creates the system's error log file, ERRTMP.SYS.

2. Writes the error buffer and statistics buffer to the ERRTMP.SYS file.

3. Allows the operator to query the number of errors in the error buffer.

4. Initializes the EL handler after writing the buffer contents or after creating the ERRTMP.SYS file upon an operator's request.

Table 1-5 summarizes the commands that ERRUTL accepts.

1.6.1.3 **Data Format Converter (PSE)** - PSE is a system program that performs the following operations:

1. Determines if the ERROR.DAT file exists on the system disk.

2. Creates the ERROR.DAT file if it does not already exist.

3. Reads the ERRTMP.SYS file and converts the RT-11 specific records to equivalent records in a DIGITAL standard error logging format in the ERROR.DAT file.

1.6.1.4 **Report Generator (SYE)** - SYE is a system program that performs the following operations:

1. Formats the ERROR.DAT file data into an error report, or summary, or both.

2. Writes out the formatted data to a display, hard-copy, or other device.

1.6.2 Using the Error Logging Subsystem

The error logging subsystem is useful in providing a history of system performance that can be used to determine if specific devices are becoming unreliable. However, the use of this subsystem does present some restrictions to the overall RT-11 operating system. Among these restrictions is the additional overhead on all I/O transfers whether an error occurred or not. The additional overhead on I/O transfers will be noticed only on time-dependent processes. In addition to the increased amount of time, some memory space must be permanently given up due to the increased size of the monitor and the presence of the EL handler. Presently, the EL handler occupies a minimum of slightly less than 1K words of memory.

Error logging is not included in the distributed RT-11 monitors. A system generation must be performed to enable error logging. See the RT-11 System Generation Manual for details.

1.6.2.1 **Loading the EL Handler** - The first thing that must be done to use the error logging subsystem is to load the EL handler. The EL handler is loaded and unloaded by the standard RT-11 LOAD and UNLOAD commands.

To load the handler, type the following command in response to the monitor's prompt (.). All commands must be terminated by a carriage return.

```
.LOAD EL (RET)
```

It is desirable to have the EL handler loaded before other handlers are loaded. This practice allows other handlers to be loaded and then released, thus returning the memory space back to the system.

1.6.2.2 **Using ERRUTL** - After the EL handler is loaded, ERRUTL must be used to create the error log file (ERRTMP.SYS) on the designated device. To invoke the ERRUTL program, the user should type the following command in response to the keyboard monitor's prompt (.). All commands must be terminated by a carriage return.

```
.R ERRUTL (RET)
*
```

The asterisk indicates that ERRUTL is ready to accept a command. Commands to ERRUTL are of the form:

```
device-name:[/options]
```

where:

device-name	is the RT-11 physical device code for the output device for the file ERRTMP.SYS. The format for the device code is: ddn where dd is the two-character RT-11 device mnemonic. n is the device unit number.
options	represents one or more command options from Table 1-5.

Table 1-5
ERRUTL Options

Option	Meaning
/C[:s]	Creates ERRTMP.SYS. The argument, s, specifies the file size, in records. The default size is 20 records. One record accommodates the full contents of the EL handler's error and statistics buffers. ERRTMP.SYS need not be created more than once.
/N	Saves the most recent errors. When the buffer becomes full, the old data is replaced with the most recent errors. The default operation is to save the oldest errors. If the default is chosen, errors that occur after the buffer becomes full are lost. This default can be changed at SYSGEN time.
/Q	Queries the EL handler for the number of errors currently in the buffer.
/W	Writes the contents of the buffer to ERRTMP.SYS and empties the buffer.

The following example creates ERRTMP.SYS on device DM1:. It will contain 50 records of the most recent errors.

```
*DM1:/C:50./N (RET)
```

The user should type (CTRL C) to exit from ERRUTL.

NOTE

If ERRTMP.SYS is written by a program other than ERRUTL, the file ERRTMP.SYS must be created each time the system is bootstrapped.

Another function of the ERRUTL program is to query the EL handler for the number of errors currently stored in the error buffer. Once the amount of data stored in the EL handler is known, ERRUTL can be used to write the contents of the error and statistics buffers to ERRTMP.SYS.

The /Q option should be used to query the EL handler, as this example shows:

```
     .R ERRUTL RET
     */Q RET
```

The current sequence number and the number of bytes remaining in the error buffer print on the console terminal. The error sequence numbers begin with 1 and end with the number of errors in a full buffer, plus 1. An error sequence number of 1 indicates that the buffer is empty.

The /W option is used to write the buffer contents to ERRTMP.SYS. Once ERRUTL is invoked, the format is as follows:

```
     *device-name:/W
```

where:

device-name	represents the device on which ERRTMP.SYS is stored.

The buffers in the EL handler are written to the specified device and re-initialized as a result of this procedure. The EL handler is returned to the state it was in when it was first loaded.

The following command sequence, for example, causes the error and statistics buffers to be written to DM0:ERRTMP.SYS.

```
     *DM0:/W RET
```

1.6.2.3 **Converting the Error Log File to a FORTRAN Data File** - A system program (PSE) is used to convert the error log file (ERRTMP.SYS) to a FORTRAN IV-compatible ERROR.DAT permanent disk file. When the ERRTMP.SYS file is full, or when the user desires to get a listing of the data even if the file is not full, the PSE program can be used to read the ERRTMP.SYS file and convert it to FORTRAN-readable records. Then another system program (SYE, discussed in the next section) is run on the ERROR.DAT file to generate a hard-copy report or display. The ERROR.DAT file is much larger than the ERRTMP.SYS file. Thus, the ERROR.DAT file can accumulate several ERRTMP.SYS files to provide a history of processor errors.

To invoke the PSE program, the user should type the following command in response to the keyboard monitor's prompt (.). All commands must be terminated by a carriage return.

```
     .R PSE RET
     *
```

The asterisk indicates that PSE is ready to accept a command. Commands to PSE are of the form:

```
     output-filespec[/option]=input-device:
```

where:

output-filespec	represents the device, file name, and file type for the FORTRAN file, in the format: `ddn:filnam.typ[size]` where dd is the two-character RT-11 device mnemonic. n is the device unit number. filnam is the six-character file name. typ is the three-character file type. size is an optional argument that specifies the size in blocks of the output file. The default is 60 (decimal) blocks. The default is DK:ERROR.DAT.
option	represents a command option from Table 1-6.
input-device	represents the input device. The default is DK:.

Table 1-6
PSE Options

Option	Meaning
/x[:size]	Causes PSE to change the size of the existing output file. If no size argument is specified, the existing output file is doubled. If a size is specified, the existing output file is increased by that number (decimal) of blocks.

When PSE is invoked, it must first determine the state of the ERROR.DAT file. If no current ERROR.DAT file exists, a new file must be created. Under this condition, the user is requested to input the number of blocks for the new file. If the ERROR.DAT file does exist, PSE examines the file to ensure that there is enough space for the records to be added.

The output file size can be changed at the program level in the following manner:

1. PSE determines if the output file size is large enough to hold the new input records.

   a. If the file size is sufficient, processing continues.

b. If the file size is insufficient, the following error message and prompts are printed out at the console:

```
?PSE-F-OUTPUT FILE TOO SMALL

MUST DELETE RECORDS FROM:  MM:DD:YY

OK TO DELETE:  Y OR N?
```

2. If N followed by a carriage return is entered, PSE prompts for further input, and no records are deleted.

3. If Y followed by a carriage return is entered, records from the specified days (month: day: year) are deleted from the file and processing continues.

The records to be deleted are displayed at the terminal, and PSE can be aborted if the operator wishes to retain the old records.

Once the file space has been examined, the formatting operation begins. As records are formatted and added to the ERROR.DAT file, they are deleted from the ERRTMP.SYS disk file. At the end of the operation ERRTMP.SYS is left in its original null state and is available to receive new data from the EL handler's buffers again. If no further error logging is required, the operator can completely delete the ERRTMP.SYS file.

The PSE program creates one error record for each memory or device error in the buffer. This record contains a header field, a register field and a program field.

In addition to the error record, PSE creates one statistics record for each unit in use during the time span encompassed by this buffer of error data. PSE also creates a statistics record for memory and cache systems. This record contains a header field and a statistics field. Each record contains an error sequence number starting with the next sequential error number after the last one in the buffer. If no error occurred for a device during the time span of the buffer, only the statistics record is generated for read/write operations.

1.6.2.4 **Generating the Error Report** - The last operation and the end objective of the error logging subsystem is to generate the error report. This function is accomplished by using a system program named SYE. SYE formats the ERROR.DAT file into an error report, an error summary, or both. After the data is formatted into the desired type of report, SYE writes out the file to a printer, visual display or other device.

NOTE

Before running the SYE program, the user must make certain that the system date and time are current by executing the DATE and TIME commands. If changes are necessary, enter the following commands in response to the monitor's prompt (.):

```
.DATE  dd-mmm-yy (RET)
.TIME  hh:mm:ss (RET)
```

To invoke the SYE program, the user should type the following command in response to the keyboard monitor's prompt (.). All commands must be terminated by a carriage return.

```
.R SYE (RET)
*
```

The asterisk indicates that SYE is ready to accept a command. Commands to SYE are of the form:

```
output-filespec=input-filespec[/options]
```

where:

output-filespec	represents the device, file name, and file type that are destination for the error report. If no file specification is given, the default device is LP:. If a file name and file type are specified, the default device is DK:. The default file name and file type are ERROR.LST.
input-filespec	represents the device, file name, and file type for the input file. The default is DK:ERROR.DAT.
/options	represents the valid options for SYE. If no options are specified, SYE prints both an error report and an error summary. Table 1-7 lists the options for SYE.

Table 1-7
SYE Options

Option	Meaning
/R	Generates the error report.
/S	Generates the error summary.
default (when no option specified)	Generates both the error report and the error summary.

An error report is a listing of the error types for each supported device. The format of this report is very similar to the format of the error log file. The error summary is a tally or summation of all errors contained in the error log file. The output is selected by user-specified options included in the command string.

1.6.2.5 **Error Logging Example** - The following commented listing is a sample error log run. Following the commands are actual reports produced by SYE, including detailed descriptions of them.

```
RT-11SJ    V03-02
?KMON-F-Command file not found
```

The RT-11 system is bootstrapped.

```
.TIME 15:40:30

.DATE 14-FEB-78
```

The date and time are entered.

```
.LOAD EL
```

The EL handler is made resident in memory.

```
.R ERRUTL
*RK0:/C
```

The file ERRTMP.SYS is created on RK0: since it did not already exist.

The error logger has been initialized and is ready to log errors when the user proceeds with regular system operation.

If the user has not altered the application software to automatically dump the memory error buffer (see Section 1.6.3), ERRUTL must be queried to determine whether or not the buffer is full.

```
.R ERRUTL
*/Q

SEQUENCE NUMBER=2
WORDS LEFT=243
```

ERRUTL is queried for the number of errors in the memory error buffer.

```
*RK0:/W
```

The memory error buffer is written to RK0:.

```
.R PSE
*RK0:ERROR.DAT=RK0:
*^C
```

PSE is invoked to convert the MACRO records to standard FORTRAN-IV records. (Note that FORTRAN-IV is not required to obtain error logging support).

```
.R SYE
*LP:=/R/S
?SYE-I-     2.   Pages
*^C
```

SYE is invoked to format the ERROR.DAT file into an error report and a summary, and to output it to the line printer. SYE then prints an informational message specifying the number of pages printed.

The device error report is printed first.

```
A SYE V03-01 SYSTEM ERROR REPORT COMPILED AT 14-FEB-78 15:44:09    PAGE      1.

  ************************************************************************
B DISK DEVICE ERROR
C LOGGED 14-FEB-78 15:41:31
D ENTRY NUMBER      1.
  ************************************************************************

  UNIT IDENTIFICATION:
      E UNIT  PHYSICAL *          1
      F TYPE                   RX01

  SOFTWARE STATUS INFORMATION:
      G RETRIED                   8.
      H NON-RECOVERED

  DEVICE INFORMATION:
        REGISTERS:
             I RXCS            100140
             J RXDB            000000
             K RXES            000120
        ADDRESS AT ERROR:
             L BLOCK                           6.
      M TRANSFER SIZE IN BYTES:             1000
      N PHYSICAL BUFFER ADDRESS START:       640.
```

The report is interpreted as follows:

Line	Meaning
A	SYE version identification and report title; includes date and time report was generated.
B	Describes the type of error.
C	Date and time the error occurred.
D	Entry number for the particular error.
E	The unit number of the device in error.
F	The type of device in error.
G	The number of times that RT-11 retried the operation in error.
H	Indicates whether or not RT-11's error retry procedure corrected the error.
I through K	Show the device/controller registers at the time of the error. The first column lists the register mnemonics. The second column lists the contents of the registers. The register information on retry operations is not logged.
L	The logical block number at the start of the transfer.
M	The amount of data being transferred to the buffer of the program incurring the error.
N	The starting address of the buffer in the program incurring the error.

The device statistics report is printed next.

```
  ****************************************************************************
A DEVICE STATISTICS
B LOGGED 14-FEB-78 15:41:31
C ENTRY NUMBER     2.
  ****************************************************************************

  UNIT IDENTIFICATION:
       D UNIT  PHYSICAL #           0
       E TYPE                     RK03/RK05/RK05F

  DEVICE STATISTICS FOR THIS UNIT:
       F # SOFT ERRORS:                          0.
       G # HARD ERRORS:                          0.
       H # OF READ SUCCESSES:                   22.
       I # OF WRITE SUCCESSES:                   0.


  ****************************************************************************
  DEVICE STATISTICS
  LOGGED 14-FEB-78 15:41:31
  ENTRY NUMBER     3.
  ****************************************************************************

  UNIT IDENTIFICATION:
         UNIT  PHYSICAL #           1
         TYPE                     RX01

  DEVICE STATISTICS FOR THIS UNIT:
         # SOFT ERRORS:                          0.
         # HARD ERRORS:                          1.
         # OF READ SUCCESSES:                    0.
         # OF WRITE SUCCESSES:                   0.
```

The report is interpreted as follows:

Line	Meaning
A	Shows that device statistics follow.
B	Date and time the statistics were logged.
C	The entry number of the error in the error log.
D	The unit number for the device in error.
E	The type of device in error.
F	The number of soft errors that were logged for the device.
G	The number of hard errors that were logged for the device.
H	The number of successful reads that were performed without retries for the device.
I	The number of successful writes that were performed without retries for the device.

The summary report is printed last.

```
SYE V03-01 SYSTEM ERROR REPORT COMPILED AT 14-FEB-78 15:44:34    PAGE     2.

          SUMMARY REPORT

A REPORT FILE ENVIRONMENT
       B INPUT FILE                DK :ERROR .DAT
       C OUTPUT FILE               LP :ERROR .LST
       D SWITCHES                  /R/S
       E DATE OF FIRST ENTRY       14-FEB-78 15:41:31
       F DATE OF LAST ENTRY        14-FEB-78 15:41:31
G ENTRIES PROCESSED                          3.
H ENTRIES MISSING                            0.
I UNKNOWN ENTRY TYPES ENCOUNTERED            0.
J FIELD FORMAT ERRORS ENCOUNTERED            0.
K UNKNOWN DEVICES ENCOUNTERED                0.
L DEVICE STATISTICS PROCESSED                2.
M MEMORY STATISTICS PROCESSED                0.
N DEVICE ERRORS PROCESSED                    1.
O PARITY ERRORS PROCESSED                    0.
       P -MEMORY                             0.
       Q -CACHE                              0.
       R -UNKNOWN                            0.

  SYSTEM ERROR REPORT SUMMARY

S RK03/RK05/RK05F        UNIT #   0
T SOFT                        0.
U HARD                        0.
V READ SUCCESSES             22.
W WRITE SUCCESSES             0.

  SYSTEM ERROR REPORT SUMMARY

  RX01                   UNIT #   1
  SOFT                        0.
  HARD                        1.
  READ SUCCESSES              0.
  WRITE SUCCESSES             0.
```

The summary report is interpreted as follows:

Line	Meaning
A through F	Describe the SYE input and output files, and the date and time of the first and last entries in the input file.
G	The number of error entries formatted in the report.
H	The number of errors missed because the occurrence of another error prevented a previous one from being logged.
I	The number of unknown errors encountered by SYE. An unknown error is any entry that the current version of SYE cannot format. This situation can occur if an old version of SYE has been run.
J	The number of times that the input file encountered a data structure error (field format error). Such encounters can indicate that the wrong version of the pre-formatter PSE was used.
K	The number of entries that referred to a device not supported by SYE. Such an entry can be encountered if an application has implemented error logging on a device that SYE does not recognize.

where: wcnt

=1	initializes the EL handler and configures the buffer to retain the newest errors.
=2	initializes the EL handler and configures the buffer to retain the oldest errors.
=3	returns four words of information in the buffer "buf".

The four words returned in buf for code 3 are as follows:

word 1 = the number of bytes remaining in the error buffer. If this word is equal to 0, no space remains. The buffer is full.

2 = the offset from the load point of the handler to the start of the statistics buffer. Note that a .DSTATUS returns the load address+6.

3 = the offset from the load point of the handler to the start of the error buffer.

4 = the sequence number of the next error. This value is reset to 1 when the EL handler is initialized.

1.6.3.3 **Calling the Error Logger from a Handler** - The error logger can be called from a user-written device handler. EL should be called on every successful transfer. If possible, the handler should distinguish non-recoverable errors (such as write-locked volume, unmounted volume, etc.) and not log them. The error logger should also be called on an initial error and on every retry for that error. Eventual success should be reported by a -1 in the high byte of R4; complete failure should be reported by a 0. The error logger keeps track of both soft (recoverable) and hard (non-recoverable) errors.

When the error logger is called from a handler, the call must be made from fork level. This is because the error logger is not re-entrant. Fork must be used to serialize access to the logger.

The call for the error logger is:

```
     JSR     PC,@$ELPTR
```

where:

$ELPTR	is a pointer to the error logger in the table of pointers constructed by the .DREND macro.

At the time of the call, the error logger requires the following information:

R2	must point to a buffer in the driver that is large enough to temporarily store all the device registers.
R3	the low byte must contain the number of device registers to log; the high byte must contain the maximum retry count.

R4	the high byte must contain the device identification code (extracted from the low byte of the device status word); the low byte must contain a success code:

-1	for a successful transfer
0	for a transfer that has failed completely (the retry count is exhausted)
n	a non-zero retry count for a transfer that failed but is being tried again

R5	must point to the third word of the queue element.

After the error logger is called, R0 through R3 are restored; R4 and R5 are destroyed.

1.6.4 Building the EL Handler

The EL handler is an option that must be SYSGENed into the system along with the fork processor to have a functioning error logging capability. In addition to these software components, the system must contain a disk and at least 16K (words) of memory.

The EL handler contains the following conditional assembly parameters, which are set through a SYSGEN or contained in SYCND.MAC.

1. ERL$B = the error buffer size in 256-word blocks. The default value is 1.
2. ERL$U = the number of specific device units that can be logged. The maximum number is 35 and the default value is 10.
3. ERL$W = the buffer configuration. If set to 1, the newest errors are kept. If not, the default value of 0 is assumed, and the oldest errors are kept. The buffer configuration can be changed by the ERRUTL program when the ERRTMP.SYS file is created.
4. ERL$A = the on-line memory to file routine. If set to 1, it is included. The default value is 0 indicating that the on-line memory to file routine is not included.

The EL handler is assembled and linked as follows:

```
.R MACRO
*EL=SYCND,EL
*^C
.R LINK
*EL.SYS=EL
*^C
.
```

CHAPTER 2

PROGRAMMED REQUESTS

A number of services at the machine language level that the monitor regularly provides to system programs are also available to user-written programs. These include services for file manipulation and command interpretation, and facilities for input and output operations. User programs call these monitor services by means of "programmed requests", which are assembler macro calls written into the user program and interpreted by the monitor at program execution time.

2.0 PROGRAMMED REQUESTS WITH EARLY VERSIONS OF RT-11

Programmed requests were implemented differently in each major release of RT-11. The following sections outline the changes that were made to the programmed requests.

2.0.1 Version 1 Programmed Requests

The earliest programmed requests, such as .READ and .WRITE, were provided with the first release of RT-11. They were designed for a single user, single job environment. As such, they differ significantly from Version 2 and Version 3 programmed requests. Arguments for Version 1 requests were pushed on the stack instead of being stored in an argument list as they are now. The channel number was limited to the range 0 through 17; more channels can be allocated in later versions. Finally, no arguments could be omitted in the macro call.

Programs written for use under Version 1 assemble and execute properly under Version 3 when the ..V1.. macro call is used (see Section 2.3.1.1). The ..V1.. macro call causes all Version 1 programmed requests to expand exactly as they did in Version 1. Version 2 and Version 3 programmed requests expand as they should for Version 2 and Version 3, respectively. However, it is to the user's advantage to convert Version 1 programs so they use the current format for programmed requests. See Section 2.5 for instructions on converting Version 1 macro calls to the current format.

2.0.2 Version 2 Programmed Requests

The second major release of RT-11 brought with it some new programmed requests and a different way of handling arguments for both the new and the pre-existing requests. The new programmed requests reflected

RT-11's ability to run a foreground job as well as a background job; they provided means to suspend and resume the foreground job, and to share messages and data between the two jobs.

The major difference between Version 1 and Version 2 programmed requests is that in Version 2, arguments for the macro calls are stored in an argument list instead of on the stack. Another substantial difference is that arguments can be omitted from the macro calls in Version 2. If the area argument is omitted, the macro assumes that R0 points to a valid argument block. If any of the optional arguments are not present, the macro places a zero in the argument list for the corresponding argument. Version 1 programmed requests were modified to incorporate these changes, and the ..V1.. macro was provided so that Version 1 programs could execute properly under Version 2 without further modification.

Programs written for use under Version 2 assemble and execute properly under Version 3 when the ..V2.. macro call is used (see Section 2.3.1.1). The ..V2.. macro call causes all pre-Version 3 programmed requests to expand in Version 2 format. Version 3 programmed requests, if any, always expand in Version 3 format.

2.0.3 Version 3 (or later) Programmed Requests

The programmed requests for Version 3 provide means for user programs to access regions in extended memory and to use more than one terminal. The chief difference between Version 3 and Version 2 programmed requests is the way in which omitted arguments are handled. In Version 3, blank fields in the macro calls do not cause zeros to be entered into the argument block. In fact, the corresponding argument block entry for the missing field is left untouched.

This change can have a significant impact on user programs. If an argument block within a program is to be used many times for similar calls, a programmer can save instructions by setting up the argument block entries only once (at assembly or run time) and then leaving the corresponding fields blank in the mmacro call.

However, users should keep in mind the fact that zeros are not substituted for missing fields. Programs that make this assumption operate incorrectly and exhibit a wide range of symptoms that can be hard to diagnose. Therefore, the necessary instructions must be written to fill the argument block, if a programmed request is issued with fields left blank in the argument list.

Programmed requests from previous versions were modified to incorporate this change, and the ..V2.. macro was provided so that Version 2 programs could execute properly under Version 3 without further modification.

The macro definitions are included in the file SYSMAC.MAC; Appendix B provides a listing of SYSMAC.MAC.

The FORTRAN programmer should note that the system subroutine library gives him some of the same capability (through FORTRAN) to use the programmed requests that are available to the assembly language programmer and described in this chapter. FORTRAN users should first read this chapter and then read Chapter 4.

2.1 FORMAT OF A PROGRAMMED REQUEST

The basis of a programmed request is the EMT (emulator trap) instruction, used to communicate information to the monitor. When an EMT is executed, control is passed to the monitor, which extracts appropriate information from the EMT instruction and executes the function required. The low-order byte of the EMT instruction contains a code that is interpreted as follows:

Low-Order Byte of EMT	Meaning
377	Reserved; RT-11 ignores this EMT and returns control to the user program immediately.
376	Used internally by the RT-11 monitor; this EMT code should never be used by user programs.
375	Programmed request with several arguments: R0 must point to a list of arguments that designates the specific function.
374	Programmed request with one argument: R0 contains a function code in the high-order byte and a channel number (see Section 2.2.1) or code in the low-order byte.
360-373	Used internally by the RT-11 monitor; these EMT codes should never be used by user programs.
340-357	Programmed request with arguments on the stack and/or in R0.
0-337	Version 1 programmed request. These EMTs use arguments both on the stack and in R0. They are supported for binary compatability with Version 1 programs.

A programmed request consists of a macro call followed, if necessary, by one or more arguments. All programmed requests start with a period (.) to distinguish them from user defined symbols and macros. Arguments supplied to a macro call must be legal assembler expressions since arguments are used as source fields in instructions (such as MOV) when the macros are expanded at assembly time. The following two formats are accepted by the monitor.

```
     Format 1:  .PRGREQ arg1,arg2,...argn

     Format 2:  .PRGREQ area,arg1,arg2,...argn
```

Format 1 contains the argument list arg1 through argn; no argument list pointer is required. Macros of this form generate either an EMT 374 or one of the EMTs 340-357. Certain arguments for this form can be omitted.

In format 2, area is a pointer to the argument block that contains the arguments arg1 through argn. This form always causes an EMT 375 to be generated. Blank fields are permitted; however, if the area argument is empty, the macro assumes that R0 points to a valid argument block (see Section 2.2.3). If any of the fields arg1 to argn are empty, the corresponding entries in the argument list are left untouched. Thus,

```
          .PRGREQ area,a1,a2
```

points R0 to the argument block at area and fills in the first and second arguments, while:

```
          .PRGREQ area
```

points R0 to the block, and fills in the first word (request code) but does not fill in any other arguments.

The call:

```
          .PRGREQ ,al
```

assumes R0 points to the argument block and fills in the al argument, but leaves the a2 argument alone. The call:

```
          .PRGREQ
```

generates only an EMT 375 and assumes that both R0 and the block to which it points are properly set up.

The arguments to RT-11 programmed request macros all serve as the source field of an instruction that moves a value into the argument block or R0. For example:

```
          .PRGREQ  CHAR
```

expands into:

```
          MOV  CHAR,R0
          EMT  374
```

Care should be taken to make certain that the arguments specified are legal source fields and that the address accurately represents the value desired. If the value is a constant or symbolic constant, the immediate addressing mode [#] should be used; if the value is in a register, the register mnemonic [Rn] should be used; if the value is indirectly addressed, the appropriate register convention is necessary [@Rn]; and if the value is in memory, the label of the location whose value is the argument is used.

Following are some examples of both correct and incorrect macro calls. Consider the general request:

```
          .PRGREQ area,argl,...argn
```

A more common way of writing a request of this form is:

```
          .PRGREQ #area,#argl,...#argn
```

In this format, the address of area is put into register 0. Area is the tag that indicates the beginning of the argument block. For example:

```
                .PRGREQ #AREA,#4
                      .
                      .
                      .
          AREA:  .WORD 0,0,0
```

When a direct numerical argument is required, the # causes the correct value to be put into the argument block. For example:

```
          .PRGREQ #area,#4
```

is correct, while:

```
          .PRGREQ #area,4
```

is not. This form interprets the 4 as meaning "move the contents of location 4 into the argument block." Instead, the number 4 itself should be moved into the block.

If the request is written as:

```
          .PRGREQ area,#4
```

it is interpreted as "use the contents of location area as the list pointer", when the address of area is actually desired. This expansion could be used with the following form:

```
                  .PRGREQ LIST1,#4
                     .
                     .
                     .
          LIST1:  .WORD AREA
          AREA:   .WORD 0,0,0
```

In this case, the content of location LIST1 is the address of the argument list. Similarly, this form is correct:

```
                  .PRGREQ LIST1,NUMBER
                         .
                         .
                         .
          LIST1:  .WORD AREA
          NUMBER: .WORD 4
```

In this case, the contents of the locations LIST1 and NUMBER are the argument list pointer and data value, respectively.

NOTE

All registers except R0 are preserved across a programmed request. (In certain cases, R0 contains information passed back by the monitor; however, unless the description of a request indicates that a specific value is returned in R0, the contents of R0 are unpredictable upon return from the request). With the exception of calls to the Command String Interpreter (CSI), the position of the stack pointer is also preserved across a programmed request.

2.2 SYSTEM CONCEPTS

Some basic operational characteristics and concepts of RT-11 are described in the following sections.

2.2.1 Channel Number (chan)

A channel number is a logical identifier for a file or "set of data" used by the RT-11 monitor. It can have a value in the range 0 to 377 (octal)--0 to 255 (decimal). In RT-11, a channel is the logical connection between a channel number and all information that must be maintained between data transfers, such as device and file name. When a file is opened on a particular device, a channel number is assigned to that file. To refer to an open file, it is only necessary to refer to the appropriate channel number for that file.

2.2.2 Device Block (dblk)

A device block is a four-word block of Radix-50 information that specifies a physical device, file name and file type for an RT-11 programmed request. For example, a device block representing the file FILE.TYP on device DK: could be written as:

```
        .RAD50  /DK /
        .RAD50  /FIL/
        .RAD50  /E  /
        .RAD50  /TYP/
```

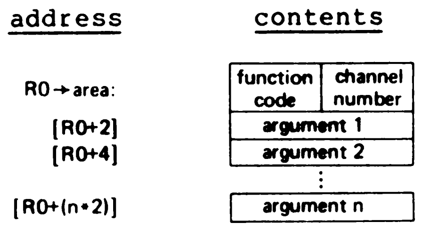
The first word contains the device name, the second and third words contain the file name, and the fourth contains the file type. Device, name, and file type must each be left-justified in the appropriate field. This string could also be written as:

```
        .RAD50  /DK FILE  TYP/
```

Note that spaces must be used to fill out each field. Note also that the colon and period separators do not appear in the actual Radix-50 string. They are used only by the Command String Interpreter to delimit the various fields.

2.2.3 EMT Argument Blocks

Programmed requests that call the monitor via EMT 375 use R0 as a pointer to an argument list. In general, this argument block appears as follows when the EMT instruction is executed:



R0 points to location x. The even (low-order) byte of location x contains the channel number named in the macro call. If no channel number is required, the byte is set to 0. The odd (high-order) byte of x is a code specifying the function to be performed. Locations x+2, x+4, etc., contain arguments to be interpreted. These are described in detail under each request.

Requests that use EMT 374 set up R0 with the channel number in the even byte and the function code in the odd byte. They require no other arguments.

2.2.4 Important Memory Areas

The memory areas for vector addresses, the resident monitor and certain system communication information are particularly important for RT-11's operation. Some addresses in these areas can be used by user programs, but others must not be used under any circumstances.

2.2.4.1 Vector Addresses (0-37 octal, 60-477 octal)

2.2.4.1 Vector Addresses (0-37 octal, 60-477 octal) - Certain areas of memory between 0 and 477 are reserved for use by RT-11. The monitor does not load these locations from the memory image file when it initiates a program. (The monitor RUN command does not load these words, for example.) However, no hardware memory protection is supplied. Therefore programs should never alter the contents of these areas. If they are destroyed by a program, the system must be re-bootstrapped or the program must restore them.

Locations	Contents
0,2	Monitor restart. Executes the .EXIT request and returns control to program. Modifying these locations while using the XM monitor always causes a system crash.
4,6	Time out or bus error trap; RT-11 sets this to point to its internal trap handler.
10,12	Reserved instruction trap; RT-11 sets this to point to its internal trap handler.
30,32	EMT trap vector.
34,36	TRAP instruction vector (in an FB or XM environment this area is loaded by R, RUN, GET and FRUN.
40-51	RT-11 system communication area (this area is loaded by R, RUN and GET).
52-57	RT-11 system communication area (see Section 2.2.4.3, below).
60,62	Console Terminal input interrupt vector.
64,66	Console Terminal output interrupt vector.
100,102	KW11L vector.
104,106	KW11P vector.
160,162	RL01 Disk vector.
200,202	LP11/LS11/LPV11 Line printer vector.
204,206	RF11,RS03/4 vector.
210,212	RK611/RK06, RK07 Disk pack vector.

Locations	Contents
214,216	TC11 vector.
220,222	RK11/RKV11 RK05 Disk vector.
224,226	TJU16,TM11,TS03 Magnetic tape vector.
250,252	KT11 Memory management fault vector.
254,256	RP04/11 Disk pack vector.
260,262	TA11 Cassette vector.
264,266	RX11/RXV11 RX01,RX211/RX2V1 RX02 Diskette vector.
320,322 324,326 330,332	VT11/VS60 Graphics terminal vectors.

2.2.4.2 **Resident Monitor** - Chapter 1 describes the placement of monitor components when the SJ monitor, the FB monitor or the XM monitor is brought into memory; the approximate size of each monitor component and the size of the area available for handlers and user programs is included.

2.2.4.3 **System Communication Area** - RT-11 uses bytes 40-57 to hold information about the program currently executing, as well as certain information used only by the monitor. A description of these bytes follows:

Bytes	Meaning and Use
40,41	Start address of job. When a file is linked to create an RT-11 memory image, this word is set to the starting address of the job. When a foreground program is executed, the FRUN processor relocates this word to contain the actual starting address of the program.
42,43	Initial value of the stack pointer. If it is not set by the user program in an .ASECT, it defaults to 1000 or the top of the .ASECT in the background, whichever is larger. If a foreground program does not specify a stack pointer in this word, a default stack (128 decimal bytes) is allocated by FRUN immediately below the program. The initial stack pointer can also be set by an option of the linker.
44,45	Job Status Word (JSW). Used as a flag word for the monitor. Certain bits are maintained by the monitor exclusively while others may be set or cleared by the user job. Since the currently unassigned bits may be used in future releases of RT-11, user programs should not use these bits for internal flags.

Those bits in the following list marked by an asterisk are bits that can be set by the user job.

Bytes — Meanings and Use

Bit Number	Meaning
15	USR swap bit. (SJ only.) The monitor sets this bit when programs do not require the USR to be swapped. See Section 2.2.5 for details on USR swapping.
*14	Lower-case bit. Disables automatic conversion of lower-case to upper-case when set. EDIT sets this bit when the EL command is typed.
*13	Reenter bit. When set, this bit indicates that the program may be restarted from the terminal with the REENTER command.
*12	Special mode TT bit. When set, this bit indicates that the job is in a "special" keyboard mode of input. Refer to the explanation of the .TTYIN/.TTINR requests for details.
10	Virtual image bit. (XM only.) When set, this bit indicates that the job to be loaded is a virtual image. It must be set in the execute file (with a .ASECT or PATCH) before the program is loaded.
*11	Pass line to KMON bit. If this bit is set when a user program exits, it indicates that the user program is passing a command line to the KMON. The command line is stored in the CHAIN information area (500-776). Refer to the .EXIT example in Section 2.4.15. This bit is not available to foreground jobs under the FB and XM monitors.
9	Overlay Bit. Set (by the linker) if the job uses the linker overlay structure.
8	CHAIN bit. If this bit is set in a job's save image, words 500-776 are loaded from the save file when the job is started even if the job is entered with .CHAIN. (These words are normally used to pass parameters across .CHAINs.) The bit is set when a job is running if and only if the job was actually entered with .CHAIN.

*7	Error halt bit. (SJ only.) When set, this bit indicates a halt on an I/O error. If the user desires to halt when any I/O device error occurs, this bit should be set. (SJ only.)

Bytes	Meaning and Use	
	Bit Number	Meaning
	*6	Inhibit TT wait bit. For use with the FB monitor. When set, this bit inhibits the monitor from entering a console terminal wait state. Refer to the sections concerning .TTYIN/.TTINR and .TTYOUT/.TTOUTR for more information.
	*5	Filter escape sequences. This bit is ignored if bit 4 is not set. Bit 5 is set to specify that escape sequences are to be echoed (if not in special mode), but not passed to the program. If this bit is not set, escape sequences are passed to the user program, but not echoed.
	*4	Process escape sequences. This bit is set to enable any escape sequence support. If this bit is not set, the same support is provided as in version 2C.
	3	Reserved for system use. Users should not attempt to use this bit.
	2-0	Reserved for internal use.
46,47	USR load address. Normally 0, this word can be set to any valid word address in the user's program. If 0, the USR is loaded in the default location through an address contained in offset 266 of RMON. If this value is not 0, the USR is simply loaded at the specified address (address in word location). See Section 2.2.5, Swapping Algorithm, for details of use.	
50,51	High memory address. The monitor maintains the highest address the user program can use in this word. The linker sets it initially to the high limit value. It is modified only by the .SETTOP monitor request.	
52	EMT error code. If a monitor request results in an error, the code number of the error is always returned in byte 52 and the carry bit is set. Each monitor call has its own set of possible errors. It is recommended that the user program refer to byte 52 with absolute addressing rather than relative addressing. For example:	

```
ERRBYT = 52
TSTB ERRBYT          ;RELATIVE ADDRESSING
TSTB @#ERRBYT        ;ABSOLUTE ADDRESSING
```

NOTE

Location 52 must always be addressed as a byte, never as a word, since byte 53 has a different function.

Bytes	Meaning and Use
53	User program error code (USERRB). If a user program encounters errors during execution, it indicates the error by using this byte. The KMON examines this byte when a program terminates. If a significant error is reported by the user program, the KMON can abort any indirect command files in use. This prevents spurious results from occurring if subsequent commands in the indirect file depend on the successful completion of all prior commands.

A program can exit with one of the following states:

- Success
- Warning
- Error
- Severe Error

The program status is successful when the execution of the program is completely free of any errors.

The warning status indicates that warning messages occurred, but the execution of the program is completed. The MACRO assembler sets the warning level bit when it detects errors at assembly time.

The error status indicates that a user error occurred and the execution of the program was not completed. This level is used when the program produces an output file even though the file may contain errors. A compiler can use the error level to indicate that an object file was produced, but the source program contains errors. Under these conditions, execution of the object file will not be successful if the module containing the error is encountered.

The severe error status indicates that the program did not produce any usable output, and any command or operation depending upon this program output will not execute properly. This type of error can result when a resource needed by the program to complete execution is not available -- for example, insufficient memory space to assemble or compile a user program. The user program reports status to RT-11 through byte 53, returning through a hard or soft exit.

The following bits correspond to the four status levels discussed previously.

Bytes	Meaning and Use	
	Bit	**Meaning (if set to 1)**
	7-4	Reserved for future use (should not be set or cleared by program).
	3	Severe error
	2	Error
	1	Warning
	0	Success

Programs should never clear byte 53 and should only set it through a BISB instruction, as in the following example:

```
     USERRB = 53
     SUCCS$ = 1
     WARN$  = 2
     ERROR$ = 4
     SEVER$ = 10
          .
          .
          .
          .
          .
ERROR: BISB  #ERROR$,@#USERRB     ;SET ERROR
                                  ;STATUS
       CLR   R0                   ;HARD EXIT
       .EXIT
```

54,55 Address of the beginning of the resident monitor. RT-11 always loads the monitor into the highest available memory locations; this word points to its first location. It must never be altered by the user. Doing so causes RT-11 to malfunction.

56 Fill character (seven-bit ASCII). Some high-speed terminals require filler (null) characters after printing certain characters. Byte 56 should contain the ASCII seven-bit representation of the character after which fillers are required.

57 Fill count. This byte specifies the number of fill characters that are required. The number of characters is determined by hardware. If bytes 56 and 57 equal 0, no fill is required.

The terminals requiring fill characters are:

Terminal	No. of fills	Word 56 Value
Serial LA30 @ 300 baud	10 after <RET>	5015
Serial LA30 @ 150 baud	4 after <RET>	2015
Serial LA30 @ 110 baud	2 after <RET>	1015
VT05 @ 2400 baud	4 after <LF>	2012
VT05 @ 1200 baud	2 after <LF>	1012
VT05 @ 600 baud	1 after <LF>	412

2.2.5 Swapping Algorithm

Programmed requests are divided into two categories according to whether or not they require the USR to be in memory (see Table 2-2). Any request that requires the USR in memory can also require that a portion of the user program be saved temporarily in the system device swap file (that is, be "swapped out" and stored in the file SWAP.SYS) to provide room for the USR. The USR is then read into memory. In the XM monitor, the USR is always resident, and therefore never swapped. During normal operations, this swapping is invisible to the user. However, it is possible to optimize programs so that they require little or no swapping.

The following items should be considered if a swap operation is necessary:

1. The background job - If a .SETTOP request in a background job specifies an address beyond the point at which the USR normally resides, a swap is required when the USR is called. More details concerning the .SETTOP request are in Section 2.4.3.6.

2. The value of location 46 - If the user either assembles an address into word 46 or moves a value there while the program is running, RT-11 uses the contents of that word as an alternate place to swap the USR. If location 46 is 0, this indicates that the USR will be at its normal location in high memory.

3. Monitor offset 374 - The contents of monitor offset 374 indicates the size of the USR in bytes. This can be useful in planning memory allocation. (See Section 2.2.6.)

NOTES

1. If the USR does not require swapping, the value in location 46 is ignored. Swapping is a relatively time-consuming operation and should be avoided, if possible.

2. A foreground job must always have a value in location 46 unless it is certain that the USR will never be swapped. If the foreground job does not allow space for the USR and a swap is required, a fatal error occurs. (The SET USR NOSWAP command ensures that the USR is always resident.)

3. Care should be taken when specifying an alternate address in location 46. The SJ monitor does not verify the legality of the USR swap address, and if the area to be swapped overlays the resident monitor, the system is destroyed.