LINK

Relocating Loader Manual

RTE-A.1 • RTE-6/VM

HEWLETT PACKARD

HEWLETT-PACKARD COMPANY
Data Systems Division
11000 Wolfe Road
Cupertino, California 95014

MANUAL PART NO. 92077-90009
Printed in U.S.A. July 1982
E0782

PRINTING HISTORY

The Printing History below identifies the Edition of this Manual and any Updates that are included. Periodically, Update packages are distributed which contain replacement pages to be merged into the manual, including an updated copy of this Printing History page. Also, the update may contain write-in instructions.

Each reprinting of this manual will incorporate all past Updates, however, no new information will be added. Thus, the reprinted copy will be identical in content to prior printings of the same edition with its user-inserted update information. New editions of this manual will contain new information, as well as all Updates.

To determine what software manual edition and update is compatible with your current software revision code, refer to the appropriate Software Numbering Catalog, Software Product Catalog, or Diagnostic Configurator Manual.

Second Edition ..... Jul 1982

NOTICE

The information contained in this document is subject to change without notice.

HEWLETT-PACKARD MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Hewlett-Packard shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance or use of this material.

Hewlett-Packard assumes no responsibility for the use or reliability of its software on equipment that is not furnished by Hewlett-Packard.

This document contains proprietary information which is protected by copyright. All rights are reserved. No part of this document may be photocopied, reproduced or translated to another program language without the prior written consent of Hewlett-Packard Company.

Copyright © 1982 by HEWLETT-PACKARD COMPANY

HP Computer Museum
www.hpmuseum.net

For research and education purposes only.

Preface

This manual is a tutorial guide and reference for users of the RTE-A.1 or RTE-6/VM LINK Relocating Loader. It is assumed that the user knows how to edit and compile programs in the appropriate operating system. For more information about loading programs in RTE-6/VM, refer to the RTE-6/VM Loader Reference manual (part no. 92084-90008), which provides information on the other loader programs (LOADR and MLLDR).

Chapter 1 provides a description of what LINK is, what it does, and how it fits into the HP 1000 system program development cycle. A sample loader session is given to illustrate how LINK works. At the end of this chapter, a comparison among the RTE loaders available is given. The differences and the purpose of each loader are described.

Chapter 2 describes the LINK program commands. The command syntax and a brief description for each command are provided. The commands are given in alphabetical order. This chapter may be used as a reference section.

Chapter 3 is a functional description of the LINK features and of the various processes involved. A special section is included to show how to optimize the performance of the LINK loader by means of merging and indexing libraries. A sample load map is also provided and discussed.

Chapter 4 is the installation guide. It provides the necessary information on how to load LINK on your system.

Appendix A provides a description of the LINK error messages.

Table of Contents

List of Illustrations

List of Tables

Chapter 1
Introduction

This chapter provides an overview for users who have little experience in loading programs using the HP loader LINK. It describes the loading process and how LINK fits into the HP 1000 system program development cycle. A sample load session is given to demonstrate some of the LINK commands and the common loading procedure.

LINK is one of several loader programs available in the HP 1000 computer system. The differences and purposes of each of these loader programs are given at the end of this chapter. Throughout this manual, the term loader refers to the LINK program.

The operating environment for LINK is the RTE-A.1 or the RTE-6/VM operating system. Unless stated otherwise, all text applies to both systems.

What is LINK?

LINK is a loader for the HP 1000. You use it to prepare your programs for execution, after they have been compiled or assembled. LINK processes the binary relocatable output of the compiler or assembler, combining it with other subroutines to produce the executable program. (The term "loader" is synonymous with the terms "linker" and "linkage editor" used by some other computer systems.)

Loading programs is an essential part of the program development cycle, whether you use Pascal, FORTRAN, Macro, or other compiler or Assembler. It occurs between the compilation step and the test/debug step in the cycle. (Note that LINK is not involved when you use the BASIC Interpreter. An interpreter does not produce relocatable code.) The steps you follow in program development using LINK on the HP 1000 computer system are listed below:

HP 1000 Program Development Cycle

1. Plan your program.
2. Create or modify the files containing the source code for your program by using EDIT.
3. Compile or assemble the source code using the appropriate language processor: FORTRAN, Pascal, Macro, etc.
4. Make your program ready to run by using LINK.
5. Debug and test your program, using Symbolic Debug for FORTRAN or Macro programs.
6. Go back to step 2 above to correct problems and continue development.

The result of Step 4 is that the relocatable output of the compiler has been made ready to run and has been placed in a program file. Step 4 serves two main functions in preparing your program for execution. First, LINK makes sure that all of your program will have a place in memory when your program runs. LINK creates a type-6 file that is an image of the program in memory. Second, LINK appends all subroutines that your program will need to run, e.g., subroutines to do input/output or math subroutines; it searches libraries to find these subroutines.

Other functions performed by LINK are specifying the amount of memory to give the program while it is running, whether or not the program will use virtual memory when it runs, and the use of symbolic debugging. The LINK functions are explained in detail in Chapter 3.

What is a Library?

A library is a relocatable file containing a number of subroutines, sometimes including an index of where each subroutine is in the file. HP supplies a large number of libraries for the HP 1000, and many users develop their own libraries. LINK searches through libraries to include only those subroutines needed by the particular program it is loading.

Some libraries are used by almost all programs, so they are usually designated by the system manager at generation time as standard or "system" libraries. System libraries are automatically searched by LINK without any special direction by the user.

Other libraries are needed only with a particular subsystem, so they are not designated as system libraries. LINK must be explicitly instructed about these libraries if it is to find any subroutines contained in them. Usually, inform LINK to search these specialized libraries before searching system libraries, as the subroutines in the specialized libraries may call subroutines in the system library.

Indexing Libraries — The LINK Indexer

Searching libraries is often the most time-consuming part of loading a program and this time can be reduced by indexing the libraries. Indexing a library is a means of showing what modules are included in the library, where to find each module, and is a recommended procedure.

LINK uses indexes produced by a program called LINDX. LINDX is a LINK indexer program that reads in a specified library, then copies it to a new file with an index included in the library. There are some cases where LINDX will make the difference between whether or not LINK thinks your program will want a subroutine in a library; this is discussed later in Chapter 3.

Sample LINK Session

The following is an example session in which a short program is loaded. Some common LINK commands used to load a program are demonstrated. Detailed descriptions of these commands are contained in Chapter 2. Note that this example does not include special topics such as symbolic debugging or virtual memory.

The following is a description of the source files that make up the example program:

&MAIN - calls subroutine INIT, CALC, and QUIT. These subroutines are in files other than &MAIN.

&INIT - calls library subroutines OPEN and CREAT.

&CALC - calls utility function PLOT from the user library $UTIL. (LINK treats functions as subroutines.)

&QUIT - calls library subroutine CLOSE.

&UTIL - source for library $UTIL. Contains a number of subroutines, including function PLOT needed by CALC above.

At this point, assume that each of the source files has been compiled into relocatable files %MAIN, %INIT, %CALC, %QUIT, and $UTIL. (Relocatable names start with a "%", and library names start with a "$", following the RTE file naming convention.) LINK uses these files to prepare the program for execution. User inputs in the following example are underlined.

```
:RU,LINK                          (run LINK from File Manager)
link REV.2226 Use ? for help      (message from LINK)
link: re,%main
link: re,%init
link: re,%calc
link: re,%quit
link:
```

In this example, LINK has prompted for input five times. LINK was directed to read each of the four relocatables that make up our sample program. The RE (relocate) command tells LINK that all of the subroutines in the named file are needed. As LINK processes each RE command, it reports the symbol (module) names (not shown in the above example).

At the end of the above command sequence the routines MAIN, INIT, CALC, and QUIT are loaded. (Actually, LINK has stored them into a temporary file until it is sure that it can find all of the subroutines it needs. Then it loads everything into the program file.) Now LINK has to find the subroutines (OPEN, CREAT, CLOSE, and PLOT) that are in libraries. The DI (display) command displays the missing subroutines:

```
link: di
  Undefined symbols:
   CLOSE   OPEN   CREAT   PLOT
```

The DI command is for information only, and has no effect on the loading process other than to display undefined subroutines. Note that LINK calls these subroutines "symbols". If a symbol is undefined, it means that LINK has not found it yet.

We know that PLOT is in $UTIL, so we can tell LINK to search that library with the SE (search) command:

```
link: se,$util
```

This command instructs LINK to search the library file $UTIL for any of the undefined subroutines. It will find PLOT in this library. Note that if subroutines OPEN, CLOSE or CREAT were in this library, then LINK would get them also. This can be a consideration when you have two or more subroutines with the same names but in different libraries; LINK takes the first one it finds.

Now we have told LINK about all of the files that we provided. The missing routines OPEN, CLOSE and CREAT are in system libraries. If not, the library files must be provided and searched, as with $UTIL above. After providing all the necessary information, we can tell LINK to finish processing our program with the EN (end) command. It will begin to search the system libraries, finding OPEN, CLOSE and CREAT. After finding all of the necessary subroutines, LINK loads them all into a type-6 program file, and then terminates. (Note that in RTE-6/VM, this is different from LOADR. With LOADR, the File Manager SP command is used to create the type-6 file.)

If we enter only EN, LINK tries to determine the name of the program based on the program statement (or NAM statement in Assembler) in our source code. In this case it will use the name MAIN, and will put the output file on the first cartridge available. If we want the program name to be EXAMP, and put it on cartridge 50, we need to enter:

```
link: en,examp::50
```

Here is the LINK command sequence used thus far:

```
re,%main
re,%init
re,%calc
re,%quit
se,$util
en
```

Most programs can be loaded by using the RE, SE, and EN commands. More sophisticated processing can be done with other LINK commands. All LINK commands are described in Chapter 2.

Our example has shown using LINK interactively. Programs may be loaded many times during its development, so you could prepare a command file containing the required LINK commands. These loader command files are named with a leading "#" by convention, for example:

```
:RU,LINK,#MAIN          (run non-interactively using LINK command
                           file #MAIN)
```

You may also enter LINK commands in the LINK runstring. This is discussed in detail in Chapter 2.

Comparison of HP Loaders

This section compares the loaders available on the HP 1000 computer system. On RTE-6/VM, these are LINK, LOADR, and MLLDR; on RTE-A.1, only LINK is available. This section aids in determining which loader to select when loading your program. The features of each of the loaders are summarized in Table 1-1.

Table 1-1. Comparison of RTE Loaders

FEATURE	LINK	LOADR	MLLDR
General Purpose Usage?	yes	yes	no
Speed	fast	slow	slow
Symbolic Debugging	yes	no	no
Type of Output	Type-6 file	Temp Prog (SP needed)	Temp Prog (SP needed)
Permanent Programs	no	yes	yes
Use of Scratch Files	yes	no	no
EXEC 8 Allowed	no (SEGLD)	yes	no
Segments in Program File	yes	no	yes
Indexing of Libraries	yes (LINDX)	no	yes (INDXR)
Multi-Level Segmentation	no	no	yes

LOADR

LOADR is a general purpose loader used to load user programs on RTE-6/VM. It offers approximately the same features as LINK. It does not support the symbolic debugger, but can produce "permanent" programs. (A permanent program is one whose ID segment cannot be removed with the File Manager OF command.) LOADR loads programs into special program areas on the system discs (LU 2 and 3), rather than directly into type-6 files as LINK does. This means that you must use the File Manager SP command to obtain a type-6 file on a disc cartridge. LOADR requires less temporary disc space to load programs than LINK, but is much slower. LOADR builds the system entry point file ($SYENT) needed by MLLDR and LOADR. Refer to the RTE-6/VM Loader Reference Manual for more information on LOADR.

MLLDR

MLLDR is a special-purpose loader used to load programs in a multi-level segmentation format. It is best suited for loading very large programs that normally would not fit on an RTE system with any other loader. Refer to the RTE-6/VM Loader Reference Manual for more information on MLLDR.

Chapter 2
LINK Commands

This chapter contains descriptions of the LINK commands. These commands may be entered interactively, in the LINK runstring, or in a command file.

LINK Syntax Conventions

Throughout this manual, the convention of representing optional parameters with square brackets [], and variable parameters with angle brackets <>, is used. All file names may have a maximum of six characters, including the prefix.

You may specify only one each of the following file descriptor prefixes:

' or "	List file descriptor ('LKST::YL or "LKST::YL)
^	Snap file descriptor (^LKSTO::YL)
#	Command file descriptor (#LKSTC::YL)
alpha	Output file descriptor (any letter - FLKST)

LINK does not impose limits on the number of file descriptors or commands defined with the following prefixes:

%	Relocatable file descriptor (%LKST::YL)
$	Library file descriptor ($LKST::YL)
+<command>	LINK commands in runstring only. (When entered interactively command formats follow the RTE-6/VM or RTE-A.1 operating system conventions.)

Commands and file descriptors may be entered in either upper- or lower-case characters.

Running LINK Interactively

To run the loader interactively from your terminal, enter

```
:RU,LINK
```

LINK will respond with the prompt sequence

```
link: REV.2226 Use ? for help
link:
```

You may now specify files and enter LINK commands one at a time in response to each LINK program prompt. Enter EN to end the link-command sequence. The sample loader session given in Chapter 1 shows a simple command sequence.

LINK Runstring

Files and special commands can be included in the LINK program runstring up to a maximum of 72 characters, including all delimiters and the RU,LINK, call.

Either commas or blanks can be used to delimit commands and file descriptors (the RTE-6/VM and RTE-A.1 operating system conventions must be used in delimiting file descriptor subparameters). Note, however, that commas must be used as delimiters for the RU,LINK, command entries. As an example

```
:ru,link,%job $joblb job1::cr 'job $jobl2
```

Here, relocatable file %job is relocated using $joblb and $jobl2 as libraries. The output type-6 file is job1, and the list (map) file is 'job. The snapshot file will be defaulted to snap::0 in RTE-A.1 or snap.6 in RTE-6/VM. Subparameters after "link" need not be in any order.

The special commands that can be used in the runstring are:

+CR:<crn>Specify scratch disc cartridge. Default is the top cartridge.

+DE Symbolic Debug operation.

+DP Inhibit purging of duplicate type-6 file.

+EB Specify extended background program (RTE-6/VM only).

+EC Echo commands, all input commands are placed in list file.

+LC Labeled System Common accessed (RTE-A.1 only).

+MA Display load map on terminal.

+RO Reorder modules to reduce base page links.

+SC Blank System Common accessed.

+SY:<sys>Specify target system: sys=6 for RTE-6/VM
sys=A for RTE-A.1

+SZ:<nn> Size program to nn pages. nn = 2 through 32

The colon is required when the SY and SZ commands are used in a runstring, but a comma is required when these commands are used in a command file or interactively from a terminal. The special commands can be entered in any order, and require only the first two characters.

In the following example, LINK relocates %JOB using the default system snapshot file SNAP on RTE-A.1, or SNAP.6 on RTE-6/VM. It also uses System Labeled Common in RTE-A.1 (or SSGA in RTE-6/VM) and turns on the reorder option.

```
:RU,LINK,%JOB +LC +ro
```

The following example loads the four segments of LKST from cartridge L1 (%LKSTA through %LKSTD), listing to file 'LKST on cartridge DL. Special command +EC echoes the LINK commands to the list file. Special command SZ:32 sizes the program to 32 pages. The output file will default to the first program name specified in the PROGRAM statement (or Assembler NAM) of the source program, and will be stored on the first available cartridge. The example is the maximum string (72 characters) you can enter. Any characters beyond 72, (counting the RU command) will be truncated. (Note that blanks are used as delimiters in the runstring.)

```
:RU,LINK,'LKST::DL +EC +SZ:32 %LKSTA::L1 %LKSTB::L1 %LKSTC::L1 %LKSTD::L1
```

The following example loads relocatable file %LCDE, overwriting output file LCDE on T6 if it exists or creating the file if it does not. Special command +LC accesses System Labeled Common. The loader uses library $LCDE for searches. (Note that commas are used as delimiters in the runstring.)

```
:RU,LINK,+LC,%LCDE::GR,$LCDE::GR,LCDE::T6
```

LINK Command Files

To run LINK using a command file, the command file name must be specified in the runstring. All runstring options are processed before the command file is read, regardless of their order. (Note that the command file name must begin with the # prefix.)

```
:RU,LINK,#<command file>
```

In the following command file, the existing EMATS file on cartridge DL will be overlayed by EMATS (unless it has a security code) by specifying the output file name as a parameter of the END command. If the output file had been specified using the 'OU' command (rather than 'EN'), an existing EMATS will not be overlayed; LINK would issue an error 146 message "Program namr already exists", and exit. File ^SNP.W is the snapshot file, and the library to search for external references is $VMALB. VMA size is 1400 pages, and working-set size is 100 pages.

```
li $vmalb
ws 100
vs 1400
re %emats
sn ^snp.w
en emats::dl
```

The following command file loads the four segments of %LKST (A through D). The result is identical to that of the LKST runstring given in the previous section. The equivalent list, echo, size, and relocate commands are included.

```
LL,'LKST::DL
ECHO
SZ,32
RE,%LKSTA::L1
RE,%LKSTB::L1
RE,%LKSTC::L1
RE,%LKSTD::L1
EN
```

The following command file is equivalent to the LCDE runstring given in the previous section, except that the command file will not overwrite the output file if it exists, since the name of the output file is given in an OUTPUT command. Other specifications are the same: Labeled System Common is specified, the library is $LCDE on cartridge GR, and the output file is LCDE on cartridge T6.

```
LC
re %lcde::gr
ou,lcde::t6
se,$lcde::gr
en
```

This sample command sequence relinks two programs to different RTE-A.1 systems. Program USERA, size 28, is linked to a system described by snap file ^SNP.W. Program USERB, size 14, is linked to a system described by snap file ^SNP.A. (This command-file structure is valid for RTE-A.1 systems only.)

```
SN,^SNP.W
SZ,28
LK,USERA
SN,^SNP.A
SZ,14
LK,USERB
EN
```

The following command file uses the reorder (RO) command. The only advantage to the RO command is that it reduces the number of base page links required. It is used here since, without the RO command, this program will not load because of base page overflow (error 125: Ran out of base page space). The command file defaults the output file descriptor, letting LINK construct it from the NAM record of the first main module.

```
RO
RE,%OVBSE
EN
```

Using File Manager Global Parameters

During program development, you can use a transfer file to load and run the application program after each change to the program. By using File Manager global parameters, you can write one transfer file for all occasions. (Refer to the RTE-6/VM Terminal User's Reference Manual or RTE-A.1 Operator's Guide for details of FMGR globals use in transfer files.)

When the loader completes, it places information into the global parameter area and indicates whether an error occurred.

For a successful program load, the global parameter area contains the following:

```
1P    2P    3P    4P    5P

PR    OG    nn    xx    ^^
```

where:

PR OG nn	is the name of the program just loaded
xx	is undefined
^^	represents blanks inserted by LINK to overwrite any previous contents of 5P

For an unsuccessful program load, the global parameter area contains the following:

```
1P    2P    3P    4P    5P

ec    xx    xx    xx     0
```

where:

ec	represents the fatal error code (see Appendix A).
xx	is undefined
0	in 5P indicates that an error has occurred

Descriptions of LINK Commands

LINK commands can be used interactively at the terminal keyboard, or can be included in a command file. Only the first two letters of each command are required; more characters can be used if desired for clarity. In the syntax description, brackets indicate optional parameters and angle brackets indicate a variable parameter. Commands that can be used in the LINK runstring are shown with a + sign in the syntax.

Note that LINK will accept commands entered with the RTE-6/VM loader OP (opcode) command. For example, OP,EB is the same as EB. Only one option is allowed per entry; if "OP,EBDB" is entered, DB will be ignored.

ABORT

Command Syntax: AB

Description: Aborts LINK immediately.

ASSIGN Partition

Command Syntax: AS,<partition nn>

nn = partition number in the range of 1 through 1023. Defaults to 0.

Description: Specifies that the program will reside in the specified partition. A partition number of 0 overrides any number previously specified.

BG (Background)

Command Syntax: BG

Description: (RTE-6/VM only.) Specifies background (type 3) program.

DBUGR

Command Syntax: DB

Description: Appends DBUGR subroutine to the program being relocated.

USER NOTE:

On RTE-A.1, when a program is relinked, this option will cause LINK to attempt to set the program's primary entry point to DBUGR if DBUGR was originally relocated with the program. On RTE-6/VM, segment names must not be specified when setting breakpoints. Only the wildcard segment (["A]) may be used.

DEBUG

Command Syntax: DE (+DE in runstring)

Description: Prepares LINK for use with the Symbolic Debug/1000 Program.

USER NOTE:

LINK will create a file named @<program name> that contains debug information. LINK will overwrite an existing debug information file with the same name but will not overwrite other disc files with the same name. The debug information file will be placed on the same cartridge as the program file.

DISPLAY

Command Syntax: DI

Description: Places a list of the current undefined externals in the list device or list file. Undefined externals in the main and current segment are listed. Note that the undefined externals are not displayed to the terminal if you are using a list file.

USER NOTE:

Use the Relocate, Search, or the End command to satisfy undefined externals. These commands cause a default search of the system library as defined in your snapshot file.

DP (Don't Purge)

Command Syntax: +DP

Description: Valid in LINK runstring only. Inhibits purging of any existing type-6 output files. This command can be overridden by the EN command.

EB (Extended Background)

Command Syntax: EB (+EB in runstring)

Description: (RTE-6/VM only.) Specifies that the program to be loaded is an extended background program. It can be up to 32 pages. If the program accesses System Common, it is automatically loaded as a large background program.

ECHO

Command Syntax: EC (+EC in runstring)

Description: Causes input commands from a file to be echoed on the list device as they are encountered, which makes it useful for debugging loader command files.

EMA (Extended Memory Access)

Command Syntax: EM,<size>

size = number of pages in the range of 2 through 1022

Description: In most cases, LINK knows if EMA is necessary. However, this command is needed if the program is segmented, the main and none of the segments use EMA, but a library routine called from a segment uses (local) EMA. The size parameter specifies a minimum amount of EMA for the program.

END

Command Syntax: EN[,<file descriptor>]

Description: End of command input. This signals LINK to end the command mode and complete the loading process. If you include a file descriptor, the specified file will be used as the output file (and override any OUTPUT command issued). If the file already exists as a type-6 file, LINK will purge it before creating the new file. This command also overrides the +DP command.

FORCE

Command Syntax: FO

Description: Force loads a program and/or a program segment. Undefined externals are set to zero and ignored, but they are still listed to the list device or list file. Your program will run as long as it does not have a call to an undefined external, but results are unpredicatable if the symbol is referenced during execution. This command is useful when running programs during the development stages when you want to test specific portions that do not call undefined externals.

LB (Large Background)

Command Syntax: LB

Description: (RTE-6/VM only.) Specifies that the program to be loaded is a large background program.

LC (Labeled System Common)

Command Syntax: LC (+LC in runstring)

Description: (RTE-A.1 only) Specifies that any references to Labeled System Common by the program will be satisfied. This command is necessary if your program will reference Labeled System Common.

LIBRARY

Command Syntax: LI,<file descriptor>

Description: Defines a library file that LINK searches immediately preceding the search of the snapshot and system libraries. The file is searched until any backward references within the file are satisfied if the file is indexed, otherwise it is only searched once. A maximum of 10 libraries can be defined, and each requires a separate LI command.

LK (Relink)

Command Syntax: LK,<file descriptor>

Description: (RTE-A.1 only.) Relinks the type-6 program file according to the snapshot file specified (with the SN command). The program file is relinked in place. Programs that use System Common cannot be relinked if the Common area size and starting location of the new system does not match that of the Common used in the original relocation of the type-6 program file. In this case the program must be relocated again to cause correct linking to the new System Common.

Any symbols that should be relinked but are not defined by the current snapshot file are forced to 0. These symbols are listed to the list file as undefined externals. In this case, results are unpredictable.

This command takes the current priority setting and size setting and DBUGR option and applies these to the program to be relinked. (No other loader options are affected by relinking.)

LL (List Option)

Command Syntax: LL,<namr>

namr = LU number or file descriptor

Description: Specifies the LU number of the list device or file descriptor if the listing is to go to a file. If a file descriptor is specified, the file must not exist unless its name begins with a single quote (') or double quote (").

NC (No Common)

Command Syntax: NC

Description: Specifies no System Common is to be used. This command is included for compatibility with other loaders.

OUTPUT

Command Syntax: OU,<file descriptor>

Description: Specifies the name and all subparameters needed for the type-6, memory image output file. If the OUTPUT or END command is not used to define the output file, LINK will create an output file with the main program's name on the first cartridge available. If the file specified exists, and you want to overwrite the file with a new version, include a file descriptor in the END command.

PRIORITY

Command Syntax: PR,<nn>

nn = a program priority number in the range of 1 through 32767. Defaults to 99.

Description: Sets the priority of the program. This command will override the priority set in the NAM or PROGRAM statement in Macro Assembler, FORTRAN or Pascal.

Refer to your operating system Terminal User's and Programmer's Reference manuals for effects of program priorities on system performance.

PS (Page Align Segments)

Command Syntax: PS

Description: Specifies that segments should start at a page boundary. This command is useful if you want to conserve base-page links.

RC (Reverse Common)

Command Syntax: RC (+RC in runstring)

Description: (RTE-6/VM only.) Reverse common. Background programs will then have their common placed in the real-time common area. Real-time programs will have their common placed in the background common area.

RELOCATE

Command Syntax: RE,<file descriptor>

Description: Loads the specified file as part of the program. The file may contain a program, program segments, or subroutines.

RO (Reorder)

Command Syntax: RO (+RO in runstring)

Description: Specifies that the modules will be reordered in an attempt to reduce the number of base page links required by the program.

RS (Reset)

Command Syntax: RS

Description: (RTE-A.1 only). This option resets the DBUGR option. If a program is relinked, this option resets the program primary entry point to the main rather than DBUGR.

RT (Real Time)

Command Syntax: RT

Description: (RTE-6/VM only.) Specifies real time (type 2) program.

SC (System Common)

Command Syntax: SC (+SC in runstring)

Description: Specifies that any blank common referenced by the program is placed in Blank System Common.

SEARCH

Command Syntax: SE[,<file descriptor>]

Description: Omitting the file descriptor causes a search of the snapshot and system library files in an attempt to satisfy undefined external references. If the file descriptor is specified, the loader searches that particular file as a library. All backward references within the library will be satisfied if the library is indexed. A warning (error 150) is issued when an unindexed library is searched.

SHAREABLE EMA

Command Syntax: SH,<label>[,<partition nn>]

label = name of the partition (maximum of six characters)

Partition nn (used in RTE-A.1 only) specifies a reserved partition number in the range of 1 through 1023.

Description: Specifies that the EMA declared resides in the specified shareable EMA partition.

Shareable EMA label files are not supported by LINK. Instead, EMA common blocks must be declared in a separate module (block data subprogram in FORTRAN) and relocated first in all programs that will share the EMA area.

SNAPSHOT

Command Syntax: SN[,<file descriptor>]

Description: Defines or displays the snapshot file. If the SNAPSHOT command is not specified before loading begins, the loader defaults to the file SNAP in RTE-A.1 and SNAP.6 in RTE-6/VM. If the file descriptor parameter is not specified with the SNAP command, the snapshot name is listed to the list device or list file.

SSGA (Subsystem Global Area)

Command Syntax: SS

Description: (RTE-6/VM only.) This command is included for compatibility with other RTE loaders that require this command. LINK automatically provides access to SSGA for those programs which access any point located in SSGA.

SYSTEM

Command Syntax: SY,<sys> (+SY:<sys> in runstring)

sys = 6 for RTE-6/VM or
A for RTE-A.1

Description: This command specifies whether the LINK output file is to be executed in an RTE-A.1 or RTE-6/VM operating system. This command can be used to change the default target system set at LINK load time. The command must be given in the form shown: SY,sys interactively or in a command file, and +SY:sys in the runstring.

SZ (Size)

Command Syntax: SZ[,size] (+SZ:nn in runstring)

size = number of pages in the range of 2 to 32.

Description: This option specifies the number of physical memory pages required to run the program being relocated. This option allows the program to require space beyond its code area if necessary. If nn is omitted, the current size value is printed to the list file. The size of System Common is not counted in the size specification of the program. If nn is larger than the number of pages available, the maximum number of available pages will be used. The command must be given in the form shown: SZ,sys interactively or in a command file, and SZ:sys in the runstring.

VM (Virtual Memory)

Command Syntax: VM

Description: Specifies access to Virtual Memory Area. Sets the default VS and WS sizes.

VS (Virtual Memory Size)

Command Syntax: VS,<size>

size = number of pages in the range of 32 through 65536. Default is 8192 pages.

Description: Specifies size of virtual memory. The maximum size in RTE-6/VM is 65536 pages. The maximum size in RTE-A.1 depends on the size of the disc, typically 8000 pages.

USER NOTE:

Although LINK allows 65536 pages of virtual space, the system on which the program runs would need a large backing store file, larger than any file supported on RTE-A.1. Refer to your RTE-A.1 Operator's Guide and Programmer's Reference manuals for details.

WS (Work Size of VMA)

Command Syntax: WS,<size>

size = number of pages in the range of 2 through 1022. Defaults to 32.

Description: Specifies working set size of VMA. Note that the size does NOT include the one-page PTE (Page Table Entry) used by the system.

* (Comment)

Command Syntax: *

Description: Denotes a comment line. When the asterisk (*) is the first of an entry line, LINK ignores the entire line.

Chapter 3
Functional Description

This chapter describes some of the LINK features and functions useful for the more experienced users. A sample load map is provided at the end of this chapter to illustrate some of the LINK features.

Program Areas

Figure 3-1 shows a memory map of the memory image program output of the loader. Figure 3-2 shows the memory map if the program specifies Virtual Memory Access (VMA) and Extended Memory Access (EMA). These maps apply only to programs that do not access common. On RTE-6/VM, the EB command must be given to start the program at 2000B.

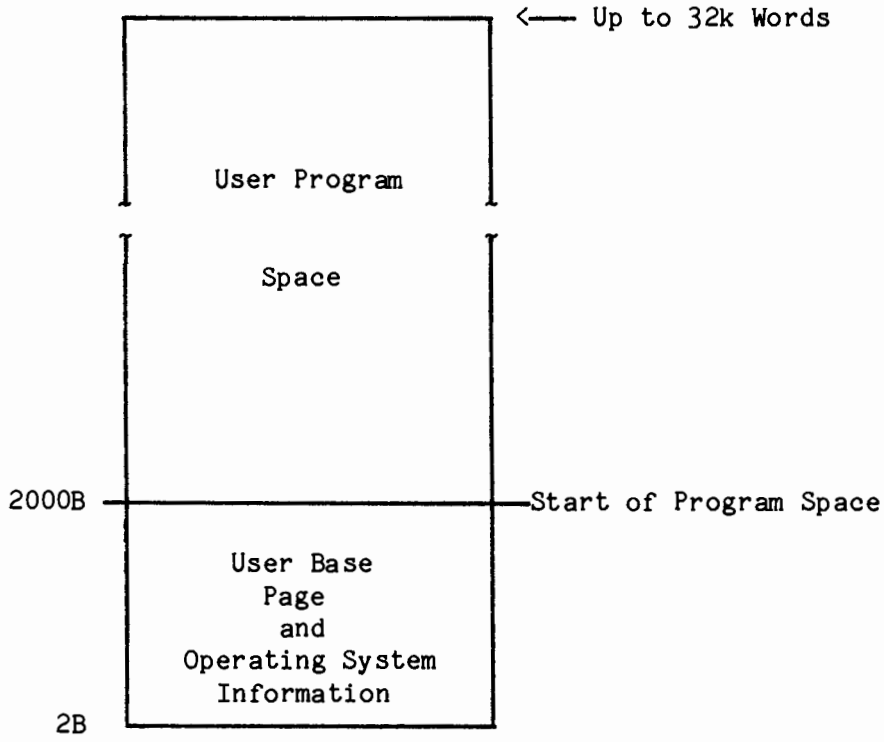

Figure 3-1. Program Memory Map

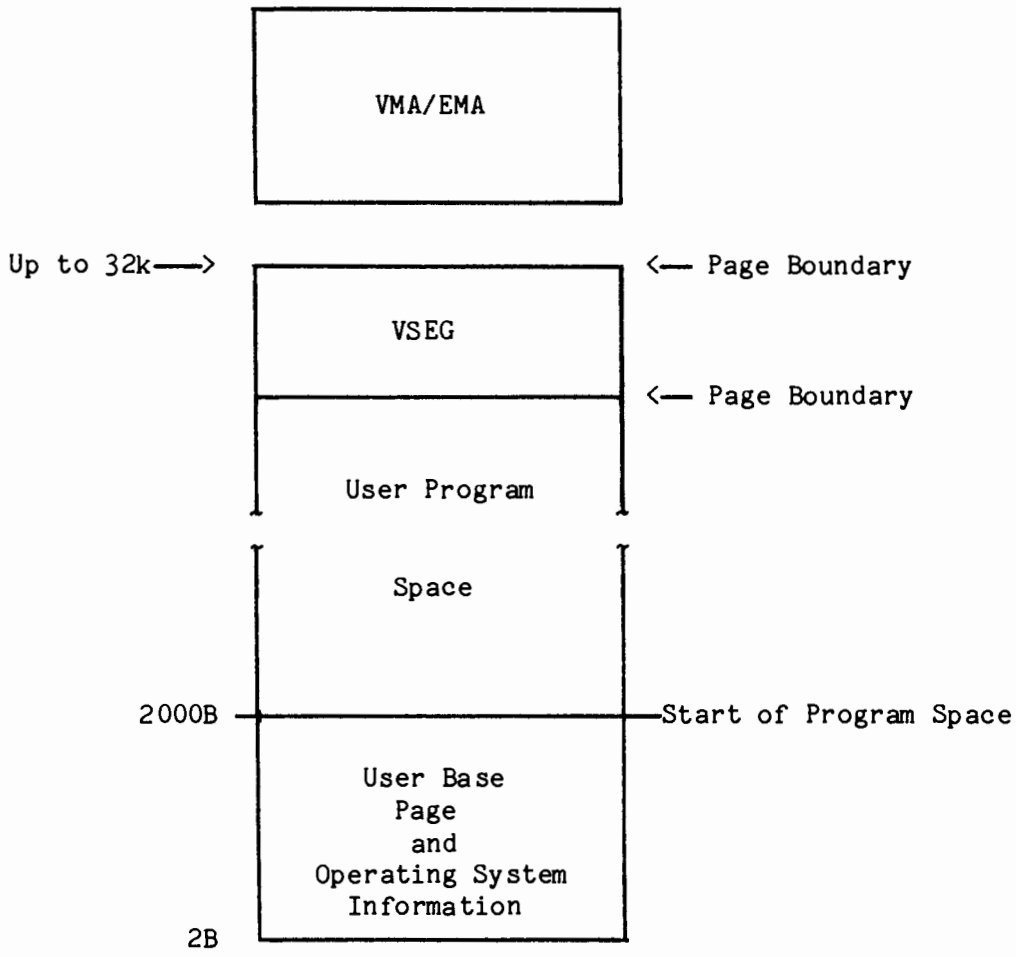

Figure 3-2. Program Memory Map with VMA/EMA

Segmented Programs

NOTE

RTE-6/VM users who wish to run segmented programs loaded with LINK should use the subroutine SEGLD to invoke those segments, not EXEC(8) calls. The routines COR.A, and COR.B cannot be used. However, you will see COR.A loaded with your program, but it will not be called. It is used by SEGLD for programs loaded with LOADR. The five optional SEGLD parameters cannot be used with LINK in RTE-6/VM.

In RTE-A.1, EXEC(8) calls can be used to invoke the segments. However, SEGLD must be used for new programs to be debugged with the symbolic Debug program.

A segmented program consists of a main program and one or more program segments. The segments can overlay each other as they are executed, thus resulting in a considerable savings in memory.

When program segments are to be linked to a main program, a block of memory for segment information is allocated by LINK for each program segment just after the main program. This block is approximately 10 words per segment. Figure 3-3 shows the program memory map for a segmented program.

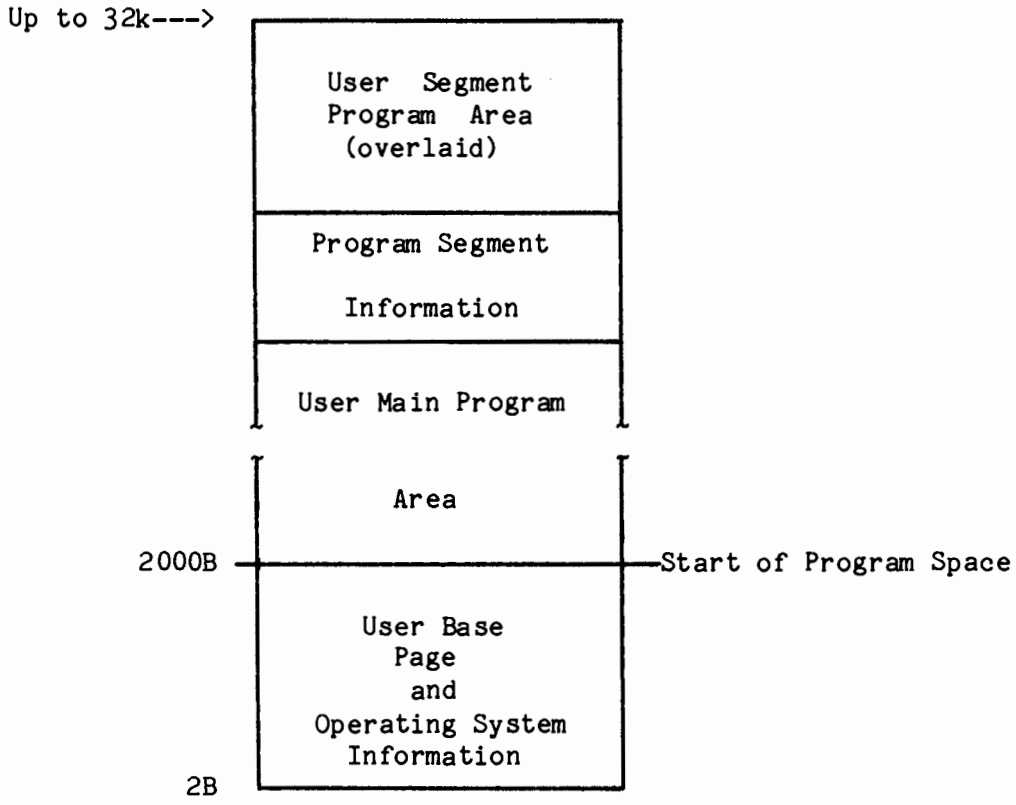

Figure 3-3. Program Memory Map with Segmentation

Snapshot File

To specify the name of the snapshot file to be referenced, use the SNAPSHOT command, or specify the snapshot file in the runstring. If this command is omitted, the loader will look for a snapshot file named SNAP in RTE-A.1 or SNAP.6 in RTE-6/VM.

In RTE-A.1, the system snapshot is a file output by the system generator containing a memory map of the operating system (defined system libraries, boundaries, and entry points). All RTE-A.1 systems have a unique snapshot file, which is required by the LINK loader to reference operating system related information.

In RTE-6/VM, a snapshot file is a system library index file created by LINDX. This file specifies a list of library files to be searched and contains the index information for the system library created by the RTE-6/VM Generator. If the system library index file does not exist, LINDX must be used to create one.

Reducing Base Page Links

The number of links on the base page can be reduced by reordering the sequence in which the modules are relocated. Reordering can be achieved by use of the RORDER command. This is useful when the base page link area overflows. There is no current page linking. In RTE-6/VM, a CP command (a LOADR command) is interpreted by LINK as the RO command. For more base page discussion, refer to the System Design Manual in RTE-A.1 or to the RTE-6/VM Loader Reference Manual.

Common Allocation

System Common is an area of memory that is accessible to more than one program; therefore, information can be stored in this area by one program, then used later by other programs in a predetermined manner. To use System Common, you must so instruct the loader by using the appropriate command. Refer to your operating system manuals for more detail of common usage.

Appending DBUGR

The DBUGR subroutine is a utility distributed with the operating system. With DBUGR, you can examine and modify memory, examine and modify registers, set breakpoints, and trace instruction execution. DBUGR can be appended to a program automatically at load time by entering the DBUGR command in the interactive mode or in the loader command file. This command instructs the loader to append DBUGR to the type-6 file output.

In RTE-A.1, note that DBUGR is a module located in the system library, $SYSLB; therefore, when requesting DBUGR, the library must either be defined in the snapshot file, or must be searched at load time.

Do not confuse DBUGR with DEBUG/1000, a symbolic debugger that is not appended to your code space. Refer to the Symbolic Debug manual (part no. 92860-90001) for the symbolic debugging procedure.

Library Files

A library file is a set of relocatable modules in one type-5 file. You can use the utility MERGE to combine individual relocatable modules into a library. The MERGE program is described in the Utility Programs Manual.

At generation time, you can specify files that are to be used by the loader as system library files with the system generator. If system library files have been specified at generation time, then the loader automatically searches these files to satisfy undefined external references during a program load. Bear in mind, however, that any system library files named at generation time (or specified in the snapshot file for RTE-6/VM) must be available to the loader when it executes, whether run on RTE-A.1 or RTE-6/VM.

If a program you are loading requires modules from a library that has not been declared at generation time, you can direct the loader to search the library file by specifying the file descriptor in a loader SEARCH or LIBRARY command. Again, the library file must be available on the disc when you run the loader.

Under RTE-6/VM, additional library files may be specified for automatic searches when the system library is indexed with the LINDX program. Note that libraries indexed with INDXR cannot be used with LINK as indexed files; these are used for MLLDR only. They can be used as unindexed libraries.

High Speed Loading

For high-speed performance using LINK, always merge and index library files into large indexed files. Use the MERGE program (Utilities Manual) to merge the files, then use the LINDX program (described later in this section) to index the library. This is not a required procedure but is recommended because it can result in a significant reduction in load time when there are many externals to be satisfied.

The way you store your libraries on disc can also affect loader speed. The following are the rules for creating libraries that LINK can read rapidly:

1. Libraries should be indexed.

2. Libraries that can be merged should be.

3. File sizes should be in multiples of 24 blocks.

4. Files should not have extents. If a file has extents, each extent should be a multiple of 24 blocks.

5. Snapshot files should follow rules 3 and 4.

The following sections provide more background on how library searches impact loader speed.

Merging and Indexing Libraries

If you do not merge and index your libraries, the loader must conduct repeated searches of all the libraries you declare until all external references are satisfied or determined not to be in the library. Since these searches include backward references from the external modules themselves, the search time can be appreciable.

Furthermore, there is a difference in the way the loader searches system libraries and user libraries. If you specify the libraries when you generate your system, the loader will search these system libraries repeatedly to satisfy external references. For user libraries, however, you must instruct the loader to repeat the search until all external references are either satisfied or determined not to be in the library.

If you index your libraries, the loader finds all backward references within one library, but it must recheck other indexed libraries for any externals that remain unsatisfied. The recheck is automatic for system libraries, but not for user libraries. Although this is faster than searching each library, the load will be faster still if you merge the libraries before indexing. For example, for the RTE-A.1 primary system, all system libraries are merged together under the name $BIGLB, then the generator LIBRARY command specifies $BIGLB for the location of externals.

When you generate a new system, merge any additional libraries into $BIGLB, then use LINDX to index the merged library. This gives the loader an index in which it can find all references from your new subroutines to the system subroutines originally supplied in $BIGLB.

The same principles apply when you write external modules for your application programs. Merge the external modules into one library, then index the library to eliminate repeated searches for backward references. The procedure is particularly beneficial for non-system libraries, because repeat searches of these libraries are not automatic. The loader searches system libraries last; therefore, references from your non-system subroutines to subroutines in the system library are automatic.

You can keep copies of your individual libraries for convenience in reassembling or editing the set. For example, the RTE-A.1 primary system includes copies of $FMP, %DECAR, $HPIB, $SYSLB, $MLIB1, and $MLIB2, which are the individual libraries that make up $BIGLB on the primary system.

When you run MERGE, the resulting merged library is not indexed, even if the individual libraries that form the input to MERGE are indexed. After merging, you must run LINDX on the merged file in order to prevent the loader from taking the time to search the library.

Indexing Library Files

Searching library files can be made much faster by indexing. A library file can be indexed by using the utility LINDX:


```
:RU,LINDX,<input file>,<output file>
```

<input file> is the file to be indexed.

<output file> is the indexed library. It must not be the same as <input file>.

Searching an indexed library is equivalent to multiple searching of an unindexed library (assuming no duplicate entry points in the library). In RTE-6/VM, files indexed by INDXR will not be considered indexed by LINK, although the files can still be searched as unindexed libraries.

To prepare the RTE-6/VM system library index file, run LINDX as follows:

```
:RU,LINDX,SYSTEM,SNAP.6::<CRN>,<library files>
```

SYSTEM is a keyword flag to LINDX to search the system libraries.

SNAP.6::<CRN> is the parameter used to create the snapshot file. It must be on a disc cartridge (CRN) accessible to all LINK users.

<library files> are files to be searched automatically before searching the system libraries defined at generation. The maximum number of libraries is 10.

Storing Library Files

The way a library is stored on disc also significantly affects loader speed. The following is from a directory listing of the same library stored on disc in four different ways under four different names. From the slowest to the fastest, the names are:

```
$WORST $BETTR $BTRYT $BEST
```

Because of the way they are stored, $BEST is the fastest to read and $WORST is the slowest. The directory list describes the way the files are stored:

```
NAME   TYPE   SIZE/LU    OPEN TO

$WORST 00005 00001 BLKS
$WORST 00005 00001 BLKS +001
$WORST 00005 00001 BLKS +002
    :   (with 46 extents)
$WORST 00005 00001 BLKS +046
$BETTR 00005 00047 BLKS
$BTRYT 00005 00024 BLKS
$BTRYT 00005 00024 BLKS +001
$BEST  00005 00048 BLKS
```

LINK will be able to read $BEST more than twenty times as fast as $WORST (when LINK is sized to 32 pages). It will be able to read $BEST many times faster than $BETTR. This is because files without extents are faster to read than files with extents, and files (and extents) that are multiples of the internal buffer are also faster to read.

The following FMGR commands would create $BEST from $WORST:

```
CR,$BEST::L1:5:48
DU,$WORST::L1,$BEST::L1
```

Specifying CRN or LU in search and relocate commands will speed up the load.

Search Sequence for Undefined Externals

In the program relocating process, LINK accepts binary relocatable code from an FMP file on disc or from an input device (LU), and outputs to the disc a memory image type 6 file ready to be copied into memory and executed. The final step of loading is attempting to satisfy all undefined externals. The files are searched in the following sequence:

1. User libraries are searched in order given. Each library is searched to resolve all backward references <u>if it is indexed</u>.

2. Each system library is searched to resolve backward references. Backward references between libraries are also resolved.

The sequence described above has several implications for you to keep in mind when preparing a segmented program input file for the loader:

a. A subroutine called by more than one segment must be relocated with the main, or with each segment that calls it, or must reside in a library to be searched.

b. Subroutines loaded with the main are not loaded with program segments.

c. Any subroutines relocated before the first program segment are loaded with the main program.

d. Any subroutines relocated after the first segment and before the second segment are loaded with the first segment.

e. Similarly, any subroutines relocated between segments are loaded with the preceding segment, and subroutines relocated after the last segment are loaded with the last segment.

Subroutines must be relocated with the segments (or the main) that they belong in, or in a library to be searched.

NOTE

If you have used other RTE loaders, this may not be the search sequence to which you are accustomed. The most important differences are that subroutines are loaded with the segment and libraries cannot be catenated to the end of the segmented program file.

If the relocatable file you are loading was produced for another RTE system and contains a segmented relocatable program, you may have to change it if one or both of the following are true:

a. Subroutines called from a segment are not relocated directly after that segment, nor in the main, nor in a library. That is, the relocatable file resides with the relocatable for another segment or in a library appended to the file to be loaded.

b. Subroutines called from the main are not relocated in a segment or directly after the main. That is, the relocatable resides in a library appended to the file to be loaded.

The most important class of file that meets these requirements is a segmented program with libraries catenated to the end of the file. Such files have to be changed in some combination of the following ways so they can be loaded by LINK:

a. Each subroutine called by more than one segment must have a copy immediately following each calling segment in the file, or must reside in a library searched by LINK, or must reside in the main.

b. Any subroutine called by the main must reside directly after the main or in a segment or in a library searched by Link. Note that subroutines residing in a segment will be loaded with that segment and not with the main.

The Relinking Process

The loader has the ability to relink an RTE-A.1 program file to the system snapshot file of a system other than that used to load the program the first time. This capability speeds the loading of programs that seldom change during system maintenance. Whenever possible, LINK creates transportable type-6 files for RTE-6/VM. Refer to the RTE-6/VM System Manager's Reference Manual for more information on transportable type-6 files.

In RTE-A.1, programs that are loaded without System Common can always be relinked. Programs that use Labeled or Blank System Common can only be relinked if the common area is unchanged between the two systems involved. If the RPLs change from one system to another RTE-A.1 system, relinks is allowed. However, it may result in an unimplemented instruction interrupt at execution time, if the RPL in question is not provided by the system executing the program.

The unimplemented instruction interrupt can be avoided by reloading and using the software equivalent of the RPLs if the program is to be moved to another system with different RPLs.

Relinking is made possible because attached to the program type-6 file is a table containing the addresses of all the system reference locations that were resolved by the snapshot file during program relocation. When a program is to be relinked, the table is read and the symbols required by the program are redefined by the snap of the new system. The references to the system in the program are then patched to the new locations specified by the new snap.

References to memory resident symbols in the system are relinked. References to absolute symbols, RPLs, and common symbols are not relinked.

The relink routine only has the ability to patch full word references to the system. This restriction prevents the use of MRG references to the system. (MRG is the memory reference group of Macro Assembler instructions. Examples are the LDA, STA and JSB commands.) System Common references are not restricted to non-MRG instructions; because of this, they cannot be relinked if System Common is changed.

The relink routine is unable to correct for a different use of RPLs or absolute symbols between one generation and another. This is posted as a warning because the program will still function if it does not use the RPLs or absolute symbols that are changed.

A program moving from a system without a particular RPL or ABS to another system that has that RPL or ABS will always work. Conversely, the program may do an unimplemented instruction interrupt for the RPL. Thus, for maximum transportability of the program file, the program should be loaded with the software implementations of all RPLs and should not use any system absolute symbols.

If the program contains references to system entries that are not defined in the new system, these references are forced to 0 and the following error is printed, "(127) There are undefined externals", which is followed by the symbols involved. Each time the snapshot file is searched and undefined externals are found, the symbols are printed out and are separated by a blank line from previous undefined externals. Since the snapshot file is searched several times on relink, the same symbol name may be reported as undefined for more than one pass of the snapshot file.

Sample Load Map

A sample LINK session in the RTE-6/VM operating system is used to illustrate some of the features of LINK. A program called EMATS that has two segments, EMA1 and EMA2, is being loaded. Each segment has a subroutine called SEG1SUB and SEG2SUB. Lines have been deleted from the listing to display only the pertinent sections of the program.

The load map allows the user to see what modules are loaded with the program, where each module is, and the address of each module. This information is useful if the system aborts the program and displays the address of the offending instruction.

As each module to be loaded is read, the module name is printed to the screen or list file, with up to ten names per line. This occurs in the example after the "re,%emats" entry. Each of the module names is printed out. After the "end" command is entered, LINK searches libraries and displays the names of the required modules it finds. A sample display format is shown below:

```
(name)(start address)(length)  (comment)
.EIO.       34323        43. 24998-1X329 REV.2140 810422
```

The module name is .EIO. with a starting address of 34323. The number 43 is the decimal representation of the length of the module in words. LINK uses the period at the end of all numbers to indicate decimal representation. The remainder of the line is reserved for comments supplied by the person who wrote the source code.

LINK concludes the load map with information showing the program name, size, and the operating system in which it can be executed. In the following example load map, LINK displays that this program is seven pages long using 64 pages of EMA. This means that this program needs seven pages to hold all the code generated and 64 pages to hold the EMA data required. Therefore, this program needs a total of 72 pages (7+64+1) for execution. The extra page is needed because every EMA program has one page of overhead.

The sample load map is shown below.

```
:ru,link
link: ro
link: li,$mlib1
link: li,$mlib2
link: re,%emats
   EMATS
   EMA1  SEG1SUB
   EMA2  SEG2SUB

link: end
 (150) $MLIB1 is not an indexed library - search proceeding
  .EIO.    .IIO.    .FMIN    .FMCN    .FMCV    .FMFP    .FMIO    .FMUI    .FMUR
  .FMO?    .FMER    .FMGB    .OPN?    .FION    .UFMP
 (150) $MLIB2 is not an indexed library - search proceeding
  .ENTR    .IOER    PAU.E    ERO.E
  LOGLU    PNAME    REIO     IFBRK    $EMA$    $SWP$    L$PTE   VMAST
  SEGLD    .LWAS    COR.A    IDGET
  CNUMD    $CVT3    SEGLD    .LWAS    COR.A    IDGET
purging old file
Load map:
  EMATS       34012   201.  EMA test for LINK <820716.1541>
  .EIO.       34323    43. 24998-1X329 REV.2140 810422
                :
           ( lines deleted )
                :
  IFBRK       37667    22. 92084-1X078 REV.2121 800129
  $EMA$       41377    95.       :
  $SWP$       41536   116.       :
  L$PTE       37715    17.       :
  VMAST       41722    39.       :
Segment EMA1
  EMA1        44000     5.
  SEG1SUB     44005    33.
  .EXIT       44046    44. 24998-1X320 REV.2101 800731
                :
           ( lines deleted )
                :
  .LWAS       45775     1. 92084-1X411 REV.2121 810717
  COR.A       46000    17. 92084-1X009 REV.2121 770621
  IDGET       46021    51. 92084-1X029 REV.2121 790314
Segment EMA2
  EMA2        44000     5.
  SEG2SUB     44005    11.
                :
           ( lines deleted )
                :
  COR.A       44456     7. 92084-1X009 REV.2121 770621
  IDGET       44477    51. 92084-1X029 REV.2121 790314

Main            34000 - 43777    4096. words
Segment EMA1    44000 - 46103    1092. words
Segment EMA2    44000 - 44561     370. words
```

```
Program EMATS::LD ready; 7 pages, 64 pages EMA
Runnable only on an RTE-6   system
```

LINK Features Discussion

This section contains features of the LINK loader that may cause some confusion for users of other RTE-6/VM loaders. These areas are given in the form of questions and answers. It covers only those areas considered to be most useful.

In the sample load map, why do the addresses start at 34012? What happened to the memory area between 2000 and 34011?

> On RTE-6/VM, the addresses between 2000 and 34000 are used by the system in this example. The program does not need them unless you use special system features. Starting program space at 2000 is obtained with the EB command.

Why are the addresses not in order?

> The addresses are often in order. However, in this case, the RO command was used which caused LINK to reorder the modules in an attempt to reduce base page links. LINK displays the order it processes the modules, not their locations, resulting in an unordered list.

What is the significance of the segments starting at 44000?

> Address 44000 is a page boundary. Segments will always start on the next page boundary following the main when the RO command is used.

With segments starting at a page boundary, is the space between the end of the main and the beginning of the segments wasted?

> Yes. Some memory is unused on almost every page. However, the amount lost is small and is almost always less than previous loaders used with current page linking.

Why is there an overlap of the two segment addresses?

> The segments use the same address spaces. They are mutually exclusive. When one segment is memory, the other cannot be in memory. When the user program calls SEGLD (segment loader), the code of the appropriate segment is brought in and overlays whatever is in memory at those addresses. Each segment always has the same starting address as any other segment in that program.

Why isn't there any free memory displayed in the load map?

Free memory is the memory left between the end of user program and the end of the partition. One cannot access this memory unless the SZ command has been entered. LINK therefore will not report it unless this command is used.

What does the message "Main pass 1; <cr> to proceed" mean?

Often during LINK execution, the system break command is entered, accidentally or intentionally. LINK halts and shows where your program is stopped and what pass. The carriage return can be used to continue LINK operation. Any other character will abort LINK. (This message does not, in fact, appear in the sample load map.)

Chapter 4
Installation Guide

This chapter describes how to load LINK on either an RTE-A.1 or an RTE-6/VM operating system. On RTE-A.1, LINK is supplied as a type-6 file on the Primary System. LINK is used to load new copies of LINK.

On RTE-6/VM, the LOADR program is used to load LINK. LINK should not be generated into the operating system. One common means of loading LINK is to use LOADR to load LINK and then use LINK to load LINK again. This provides a single type-6 file with the segments included for subsequent use. If you do not load LINK with itself, you will have to SP and OF the segments as usual with LOADR. Using LINK to load itself is the recommended method of loading.

RTE-A.1 Installation

The following loader command file loads LINK in the RTE-A.1 operating system:

```
*  LINK loader command file for RTE-A.1
*
SZ,32
*  Default output to run on RTE-A.1
RE,%LNKDA
*  Modules required for running on RTE-A.1
RE,%LNKRA
*  Relocate LINK
RE,%LINKA
RE,%LINKB
RE,%LINKC
RE,%LINKD
EN
```

To load LINDX on RTE-A.1, use the following loader command file:

```
*  LINDX loader command file for RTE-A.1
*
SZ,32
*  Module required for running on RTE-A.1
RE,%LNKRA
*  Relocate LINDX
RE,%LINDX
EN
```

RTE-6/VM Installation

The following loader command file loads LINK in the RTE-6/VM operating system:

```
*  LINK loader command file for RTE-6/VM
*
OP,EB
SZ,32
*  Module required for defaulting output to RTE-6/VM
RE,%LNKD6
*  Module required for running on RTE-6/VM
RE,%LNKR6
*  Relocate LINK
RE,%LINKA
RE,%LINKB
RE,%LINKC
RE,%LINKD
EN
```

If you wish to default your output to run on RTE-A.1, replace RE,%LNKD6 with RE,%LNKDA in the above command file.

To load LINDX on RTE-6/VM, use the following loader command file:

```
*  LINDX loader command file for RTE-6/VM
*
OP,EB
SZ,32
*
*  Relocate LINDX
*
RE,%LINDX
EN
```

After loading LINDX, it must be run to create the default snapshot file SNAP.6 needed to run LINK. Create the snapshot file as follows:

```
:RU,LINDX,SYSTEM,SNAP.6::<CRN>
```

The snapshot file must be on a disc cartidge accessible to all LINK users. Refer to the Snapshot File and the Indexing Library Files sections in Chapter 3 for more information on LINDX.

SYDNEY TECHNICAL COLLEGE
SCIENTIFIC INSTRUMENTATION CENTRE
DEPT. OF T.A.F.E.

Appendix A
Error Messages

When you run LINK, error messages can come from either LINK or from the file management package (FMP). Refer to your operating system Terminal User's Reference Manual for the FMP error message format.

Error messages from LINK have the following form:

(nnn) <message>

where:

(nnn) is a three-digit number in parentheses, (100) or higher.

<message> is a brief description of the error.

LINK also displays messages in explanation of some FMP errors, particularly with respect to the scratch file. For example:

Scratch file out of room on LU xx

The intermediate scratch file LINK uses overflowed the cartridge (same as FMGR -33). Put a cartridge with more room at the top of the cartridge list and try the load again or use the CR LINK command to control the scratch cartridge selection.

The following list includes an expanded description of each loader error message:

(100) Unexpected eof from command file

More LINK commands are required to load the program than are in the command file.

(101) Size out of range

Size given is not in the required range for this parameter.

(102) Partition out of range

Partition number given is not in the required range.

(103) Too many libraries

The maximum number of libraries is 10.

(104) Can't change snap now

Once the RE command is given, the snapshot file cannot be changed.

(105) Illegal partition number

Partition number given is out of range.

(106) Illegal name

Names must be alphabetic (as parsed by NAMR).

Legal: A324 .DOUG 123C

Not legal: 1234 -DOUG 123B

(107) No modules relocated

The end command was given, but there is nothing to relocate.

(108) Module not relocated

LINK was unable to relocate the given module.

(109) Illegal relocatable

Checksum error in given relocatable; recompile and try again.

(110) Illegal relocatable (Ext)

A relocatable record contains a reference to an undeclared external; correct program, recompile, and try again.

(111) Illegal relocatable (EMA)

Illegal EMA symbol; correct program, recompile, and try again.

(112) Illegal relocatable (MR)

A relocatable record has an illegal MR; correct program, recompile, and try again.

(113) Illegal relocatable (RPL)

Usually an RPL longer than one word; LINK handles only one-word RPLs.

(114) Record ignored

LINK has encountered a relocatable it recognizes but does not use; primarily LOD records.

(115) Too many EMA symbols

EMA symbol table overflow; use fewer EMA symbols. Correct program, recompile, and try again.

(116) Out of table space

Size up LINK as large as possible. It needs more room for an internal table.

(117) Allocate type mismatch

The same symbol has been assigned two different data area allocation types; for example, EMA in one program unit and program-named common in another; types must match.

(118) Symbol table overflow

Too many symbols in program main + current segment. Segment your program and reduce the number of external references.

(119) Illegal EMA record combination.

Two relocatable records have given conflicting EMA directives; usually occurs when mixing old and new relocatables; recompile, generating all new relocatables.

(121) Relocation into system common illegal

Relocation of code or data into the system area by LINK is illegal.

(122) Illegal system reference

An attempt was made to reference a system entry point with a one-word instruction; this will not do what you want.

Examples (assume $CON in system):

```
Legal:      XLA $CON          DEF $CON

Illegal:    LDA $CON          JSB $CON
```

(123) Illegal EMA reference

Relocation of code into EMA is illegal.

(124) Unimplemented relocatable type

Future versions of LINK may accept these, but they currently are not processed by LINK.

(125) Ran out of base page space

Out of space indicates that the problem was detected when relocation of code or data on the base page was being attempted or while a base page link was being created. Use the RO command to decrease the number of base page links required. Try to break up your program into smaller modules. Relocate less code on the base page.

(126) Ran out of workspace

Size up LINK as large as possible to obtain more work space.

(127) There are undefined symbols

The program relocated asked for symbols LINK was unable to find in any of the libraries given.

(128) More common declared than system common available

A program using system common must declare less than or equal to the amount of system common.

(129) Segment base page ent

Base page entries must be defined in the main; relocate the offending module with the main.

(130) Program is too big

Ran out of logical address space; generated code of (main + longest segment + base) is greater than 32k words; segment the program or make it shorter.

(131) Snap file wrong type

Snap file must be of type 3.

(132) Too many system libraries

Maximum number of system libraries in snap file is 64.

(133) No program name

LINK has not been given a name by the user and has not picked up a name from the relocatable; give it a name in the OUTPUT or END command.

(134) Program namr not type 6

The given program namr is not a program type file, therefore LINK will not purge it or overlay it; you must give LINK a different namr or purge the file and reissue the END command.

(135) System has too many base page links

Link has detected an illegal number of base page links for the labeled common area; your snapshot file is probably corrupt.

(136) Bad labeled common links in snap file

LINK has detected inconsistent base page links for the labeled common area; your snapshot file is probably corrupt.

(137) Type 6 file too small

Ran out of space in the type-6 file; when specifying the namr in the END or OUTPUT command, a larger size must be specified. LINK automatically calculates the required number of blocks and issues the message

Try nnnnn Blocks

You can specify the number of blocks as the last subparameter of the file descriptor (NAME::::#blocks) assigned with the OU or EN command.

(138) Corrupt snap file

Either the given file was not a snap file or it has become corrupt.

(139) Duplicate Entry: <name>

Relocation of multiple modules containing identical entry points was attempted; either the same module was relocated twice, or two modules have identical entry points; the modules must be eliminated or the entry point names changed; the entry <name> is supplied.

(140) Illegal program file

Relinking failed because the file given was not a legal type-6 file for this system; the program must be reloaded.

(141) System common change

Relinking failed because either the size of system blank common has changed or labeled common has changed.

(142) RPL change - Program failure possible

This error can only occur during relinking. In spite of the error, the program has been relinked, but there is a discrepancy in the order or the values of RPLs or ABSs. If the program being relinked uses these, it may fail. See the section Relinking in this manual for more information.

(143) Externals were forced - Program failure possible

You instructed LINK to force load the program, and undefined entry points were set to zero.

(144) RPL change and undefined externals.

The program was not relinked because there were undefined system entry points. The program uses system entry points that do not exist on the target system.

(145) EMA partition cannot be the program partition

You have attempted to assign a program and shareable EMA to the same partition. This is not legal.

(146) Program namr already exists

The OU command was used to specify the program file, which prevents automatic purging if the file already exists on the referenced cartridge.

(147) Error loading segment <n>

Unable to load this segment of LINK. LINK probably is corrupt.

(148) Undefined module length

This program was compiled by a compiler not supported on this system. Recompile.

(149) Program RPd to current system

The named program is in use on the current system.

(150) <name> is not an indexed library - search proceeding
<name> was given to LINK as a library, and it is not indexed

LINK does not resolve backward references in non-indexed libraries. Whether or not there are backward references to be resolved, indexing the library produces a faster search.

(151) Too late to relink

You cannot relink after relocation.

NOTE: The following are errors that occur only on RTE-6/VM:

(152) Unknown RTE type

System specified not RTE-A.1 or RTE-6/VM.

(153) Too late to change RTE type

RTE type must be specified at beginning.

(155) SNAP not set up for this system.

SNAP is the file containing index information on an RTE-6/VM system. Refer to the Indexing Library Files section in chapter 3 of this manual.

Index

R

S

MANUAL PART NO. 92077-90009
Printed in U.S.A. July 1982
E0782

HEWLETT-PACKARD COMPANY
Data Systems Division
11000 Wolfe Road
Cupertino, California 95014