DEC-10-ULKMA-A-D

DECsystem-10

# LINK-10

# PROGRAMMER'S REFERENCE MANUAL

This document reflects the software as of Version 1.

1st Printing May 1973

LINK-10

TABLE OF CONTENTS

LINK-10

LINK-10

FOREWORD

This manual is the reference document on the DECsystem-10 Linking Loader, LINK-10. It is aimed at the intermediate-level applications programmer and contains complete documentation on LINK-10, including descriptions of the LINK Item Types generated by the DECsystem-10 Translators.

Chapter 1 is an introduction to LINK-10 and describes the two methods of initializing the Linking-Loader. Chapter 2 discusses the automatic use of LINK-10 through the COMPIL-class commands, and Chapter 3 discusses the direct use of LINK-10 through the R LINK system command. LINK-10 switches are described in alphabetical order in Chapter 4. LINK-10 messages and examples appear in Chapters 5 and 6, respectively. The Appendices and Glossary contain supplementary information.

A beginning user of LINK-10 can benefit from this manual by reading Chapters 1 and 2, whereas an advanced user would be more interested in Chapters 3 and 4. A user who has been employing the LOADER program will find Appendix B a valuable aid in the transition to the LINK-10 program.

LINK-10

Input to LINK-1Ø

# CHAPTER 1

# INTRODUCTION TO LINK-10

LINK-10, the DECsystem-10 Linking Loader, is the system utility program that merges independently-translated modules of a user's program into a single module. Its main function is to prepare and link this input with other modules required by the user into a form that can be executed by the operating system.

## 1.1 INPUT TO LINK-10

LINK-10 accepts as its primary input the output from the DECsystem-10 translators in order to produce an executable version of the user's program. This output, known as object modules, is in the form of binary files which contain the user's programs and additional information generated by the translators. This additional information is necessary for linking separately-translated modules, for debugging, and for generating auxilary output such as map, log, and save files.

### 1.1.1 Relocatable Code

Most object modules contain relocatable code so that the module's position in core can be determined by LINK-10. Relocatable code is a benefit to both the user and the system. The user benefits because he is able to code all of his modules without regard to where they will be located in core. He need not be concerned with the location where one module ends and another one begins. The system benefits because a module written in relocatable code can be placed anywhere in core memory. When moving the relocatable object modules into the areas of

core memory at which they will be executed, LINK-10 adjusts all relocatable addresses in the modules into actual machine locations. In reality, LINK-10 places the modules in a user virtual address space (refer to the Glossary) and the operating system, as it schedules the usage of the system, transfers the modules to and from core memory. However, for simplicity, the user virtual address space is referred to as core memory in the remainder of the manual.

### 1.1.2 Symbols and Libraries

In addition to relocating and loading the user's object modules, LINK-10 is also responsible for linking these modules with other modules required for execution. Linkages among modules are provided through the use of symbols. By including symbols in his programs, the user is delaying the assignment of actual values until load time. This method of assigning values is advantageous because:

- It allows the user to change only the definition of the symbol instead of changing every occurrence of the value, and
- Only the module containing the definition of the symbol must be retranslated when a change occurs. Since other modules using the symbol are bound to it at load time, they do not have to be retranslated.

Although a user can define and use a symbol entirely within a single module, he usually refers to additional symbols that are defined in other modules. It is these modules that must be linked to the user's program for execution. In most cases, these required modules are contained in a library of relocatable binary programs. Modules within a library can either be created and translated previously by the user

or be part of the system's repertoire of programs. For instance, most higher-level languages have associated with them a library containing commonly-used mathematical, input/output, and data conversion routines. The user refers to modules in the library via symbols in his program and these symbols are then linked to the proper location in the library modules themselves. By linking these symbols and loading the required modules, LINK-10 provides communication between independently-translated modules and library routines.

In order to satisfy any undefined symbols, the required system libraries are usually searched after all loading specified by the user has been performed. However, the user can indicate that libraries be searched at a particular point in the loading procedure by specifying the appropriate switch to LINK-10 (refer to /SEARCH and /SYSLIB in Chapter 4). When LINK-10 processes the switch, the indicated libraries are searched and the required modules are loaded. The user also has the option of specifying by name which modules he wants (or does not want) loaded from a library or of inhibiting the search of the library altogether.

## 1.2 OUTPUT FROM LINK-10

When LINK-10 has performed the tasks of loading the user's object modules in core, bringing in and linking any other modules required for execution, and adjusting all the addresses, there is in core an executable version of the user's program. This executable version is the primary output of LINK-10. Since the loaded program at this point reflects the state of the user's core memory, it is usually referred

to as his core image. Having arrived at this state, the user can request LINK-10 to either:

- Transfer control to the core image for immediate execution (using the EXECUTE or START system commands, or the /DEBUG, /EXECUTE, or /TEST switches in LINK-10), or
- Output the core image to a device for storage (using the SAVE or SSAVE system commands, or the /SAVE or /SSAVE switches in LINK-10) in order to avoid the loading procedure in the future.

If the complete, loaded program is saved on a device in core image form, it can be brought into core and executed at a later time (using the GET and RUN system commands). The loading process does not have to be repeated since the results of all of LINK-10's actions are contained in the core image. However, if the user wishes to revise the modules that made up his core image, he must once again use LINK-10.

While the primary output of LINK-10 is the executable version of the user's program, the user can request auxilary output from LINK-10 in the form of map, log, save, symbol, and expanded core image files (XPN files). This additional output is not automatically generated by LINK-10 and the user must include the appropriate switches to obtain this output (refer to Chapter 4 for a description of the switches). This output is for the user's convenience when debugging his program.

## 1.3 OVERLAY FACILITY

LINK-10 will have an overlay facility to be used when the total core required by a program is more than the core available to the user. The user then organizes his program so that only some parts of the

program are required in core at any one time and the remaining parts are transferred in and out of core. During execution, these transferred parts are brought into core as required. The part brought into core overlays the part currently in that area. Because these parts of the program reside in the same area of core at different times, the amount of core required for the entire program is reduced.

## 1.4 MISCELLANEOUS FEATURES

LINK-10 has a large number of options in order that the user can gain precise control over the loading process. The user can set various loading parameters and can control the loading of symbols and modules. By setting switches in his input command strings to LINK-10, he can specify the core size of LINK-10 modules, the start address of modules, the size of the symbol table, the messages that he will see on his terminal or in his log file, and the severity level and verbosity of the messages. He can control the loading of modules by specifying the modules that should be loaded and the files that should be searched for symbol definitions. He has control over the number of segments to be allowed and the segment into which the symbol table will be placed.

The user has control over file specifications that LINK-10 examines to determine device names and filenames. He can accept the LINK-10 defaults for components in a file specification or he can set his own defaults which will be used automatically when he omits a component from his command string. He can also position devices, allocate space and assign protections to output files, and clear directories of DECtapes.

Some options available to the user are interactive. In the process of producing a core image, LINK-10 attempts to satisfy all requests for symbols defined in other modules and allows the user to interactively ask for a list of undefined symbols during the loading procedure. The user then has the opportunity to define them without reloading.

## 1.5 INITIALIZATION OF LINK-10

LINK-10 is initialized by the user in one of two ways:

- Automatically through the use of the LOAD, EXECUTE, or DEBUG system commands. This is the most common usage of LINK-10.
- Directly through the use of the R LINK system command. This is recommended for very large and relatively complex loading procedures.

### 1.5.1 Using LINK-10 Automatically

LINK-10 is automatically initiated when the user issues one of the system commands LOAD, EXECUTE, or DEBUG. These commands are known as COMPIL-class commands because they use the COMPIL program to control the actions of DECsystem-10 translators and LINK-10. COMPIL's job is to accept the command string typed by the user, interpret it, and construct and pass new command strings to various system programs, including the translators and LINK-10. This action taken by COMPIL is a convenience to the user since it saves him from typing the command strings to LINK-10. Once the command string to COMPIL is processed, the user does not interactively communicate with the translators or LINK-10. LINK-10 processes the appropriate command strings passed to

it by COMPIL and supplies intelligent defaults for any parameters not specified by the user. If LINK-10 obtains an error condition, it terminates the load and returns control to the operating system for further instructions. Otherwise, it loads the program and, depending on the COMPIL-class command used, either exits or starts the loaded program. Refer to Chapter 2 for the descriptions and use of the COMPIL-class commands.

In general, the extremely fine control of the loading process that is provided by manually running LINK-10 is not required for the average user because the COMPIL program supplies reasonable defaults to LINK-10.

### 1.5.2 Using LINK-10 Directly

Direct use of LINK-10 is useful for those who are developing large and complex programs, loading from devices other than disk, manipulating symbol tables for complex debugging situations, and performing segment manipulations.

The user runs LINK-10 directly by using the system command R LINK. LINK-10 responds with an asterisk which indicates that the user can type his input as a series of specifications which are to be used in the loading process. LINK-10 accepts input until the user specifies the exit condition; at which point it finishes all of its tasks and exits or begins the program, as specified by the user.

This method of running LINK-10 gives the user access to its full capability. The user does not have to accept LINK-10's default

conditions, but can supply his own set of defaults. He can interactively monitor the loading process by setting internal parameters, requesting values of particular items, specifying modules and files to be loaded, and controlling the format and contents of output files. Refer to Chapter 3 for the description of the LINK-10 command string, and Chapter 4 for the switches used when directly running LINK-10.

LINK-10

Automatic Use of LINK-1Ø

# CHAPTER 2

# AUTOMATIC USE OF LINK-10

The user causes LINK-10 to be run automatically whenever he types the LOAD, EXECUTE, and DEBUG system commands. These commands accept a simple command string format and are converted internally to a series of more complex command strings that are directly processed by various system programs, including language translators and LINK-10. The aforementioned commands are used to compile, load, and execute programs, to obtain output in the form of maps, to search files in library search mode, and to invoke the various debugging aids. The following paragraphs describe each of these system commands.

NOTE

> The information in this chapter is a subset of the material available on the LOAD, EXECUTE, and DEBUG commands. The subset presented here assumes that the source files have previously been translated, and thus only the switches directly applicable to loading the binary files are listed. Complete reference documentation on the COMPIL-class commands, their valid command formats, and all available switches can be obtained from the appropriate command descriptions in DECsystem-10 OPERATING SYSTEM COMMANDS, DEC-10-MRDC-D, located in the DECsystem-10 SOFTWARE NOTEBOOKS and in the DECsystem-10 USERS HANDBOOK, DEC-10-NGZB-D.

The LOAD command translates the user-specified source files into relocatable object modules (if necessary) and loads these object modules to form a core image. This command does not cause execution of the resulting core image. After completion of this command, the user can either execute his program (START system command) or save the core image (SAVE or SSAVE system command) for future execution.

LINK-10

Automatic Use of LINK-1Ø

The EXECUTE commmand translates the user-specified source files (if necessary), loads the object modules into a core image, and, in addition, begins execution of the program. The action of this command is the same as that of the LOAD command followed by the START system command.

The DEBUG command translates the user-specified source files (if necessary), loads the object modules into a core image, and prepares for debugging by additionally loading a system debugging program. Usually this debugging program is loaded first, followed by the user's program and other information required by the debugging program (e.g., the symbol table). However, when COBOL programs are being loaded, COBDDT (the COBOL debugging program) is loaded after the user's program. Upon completion of loading, control is transferred to the debugging program, rather than the user's program, so that the user can check out his program by examining and modifying the contents of locations. This examination and modification can occur both before program execution begins and during execution if the user specifies breakpoints in the program at which execution is to be suspended.

The debugging program can be COBDDT, MANTIS, or DDT, depending on the first source file in the command string. If the first file is a COBOL file, COBDDT (the COBOL debugging program) is loaded. If the first file is a FORTRAN source file, MANTIS (the FORTRAN debugging program) is loaded. (Note that MANTIS is under the control of an assembly conditional switch which is normally off. Therefore, the installation must turn this assembly switch on for the loading of MANTIS.) If the

LINK-10
General Command Format

first file is any other file, DDT (the Dynamic Debugging Technique) is loaded. When the first file has previously been compiled (i.e., the file has an extension of .REL, meaning relocatable binary object module), COMPIL does not determine the type of source file from which it came so DDT is loaded with the binary files. In this case, if the user desires COBDDT or MANTIS, he must explicitly specify this debugging program via the appropriate switch (refer to the /COBOL and /MANTIS switches in DECsystem-10 OPERATING SYSTEM COMMANDS).

## 2.1 GENERAL COMMAND FORMAT

The LOAD, EXECUTE, and DEBUG system commands have the same general command format. They all accept a list of file specifications.

LOAD output file spec = concatenated input file specs
EXECUTE output file spec = concatenated input file specs
DEBUG output file spec = concatenated input file specs

An input or output file specification consists of a device name, a filename with or without a filename extension, and a directory enclosed in square brackets. Only one output file specification can be given on the left of each equals sign, but any number of input file specifications can occur on the right. Input file specifications are separated from each other by commas or plus signs. If commas are used, the translator produces separate relocatable object modules for each output file. If plus signs are used, the input files separated by plus signs will be translated into a single relocatable object module. Plus signs must be used when a collection of files must be

concatenated to produce an acceptable module as input to a translator. The sequence of "output file spec = concatenated input file specs" can be given repeatedly in a command string by separating each sequence with a comma.

The output file specification and the equals sign can be omitted, in which case the object module is placed in the user's default directory on the disk with a name derived from the source file and the extension .REL. The filename given to the output file depends upon the form of the user's input file specifications. If the user has only one input file, the output file is given the name of the input file. If the user has more than one input file and the files are separated by commas, the name of each output file is the name of the corresponding input file. If the user has plus signs separating the file specifications, the name given to the output file is the name of the last input file in the series of files separated by plus signs.

## 2.2 COMPIL SWITCHES

Switches can be included on the LOAD, EXECUTE, and DEBUG command strings to direct LINK-10 in its processing. These switches are used to generate listings, to create libraries, to search user libraries, and to obtain loader maps. Each switch is preceded by a slash and can be either temporary or permanent. A temporary switch applies only to the file immediately preceding it. Characters (including spaces or commas) cannot separate the filename and the switch. A permanent switch applies to all files following it until modified by a subsequent switch. It is separated from the file it precedes by a space or a comma.

LINK-10

COMPIL Switches

LINK-10 switches described in Chapter 3 can be passed on the COMPIL-class command strings by preceding the switch specification with a % character instead of a / character. Following the % character is the LINK-10 switch specification preceded and followed by a delimiter. The delimiter can be any character; however, the user must be careful that the character he uses does not have a specific meaning to the COMPIL program. For example, the @ character indicates an indirect command file, and the semicolon causes the remainder of the line to be treated as a comment and thus ignored. The recommended delimiter is a single or double quote character. The beginning and ending delimiter must be the same character. A LINK-10 switch specification consists of the switch name and optionally a keyword and a value. The items in the specification are separated by colons. (Refer to Chapter 4 for the formats of the individual LINK-10 switches.) Note that LOADER switches (those beginning with a % but without enclosing delimiters) are illegal when passed to LINK-10. As an aid to users, a warning message is printed if the LINK-10 switch delimiter is one that could be interpreted as a LOADER switch (e.g.,A-Z,a-z,0-9,&, and -).

Since the first function of each of these three commands is to determine if the source files need translating (i.e., compiling or assembling), there are many switches that pertain to the translating process. The purpose of this manual is to describe the use of LINK-10 and switches pertaining to the translation of the source file are not

included. All switches that can be placed on the command string are described in DECsystem-10 OPERATING SYSTEM COMMANDS.

NOTE

Since currently there are two linking-loaders on the DECsystem-10, the user must indicate the desired loader when using the LOAD, EXECUTE, or DEBUG command. At the present time, the LOADER program is the default case, and the user must include the /LINK switch to indicate that he wishes to use the LINK-10 program. (The setting of the LOADER program as the default is a system parameter that can be changed by individual installations.) In the future, LINK-10 will become the standard default.

Table 2-1

COMPIL Switches Pertaining to Loading

| | |
|---|---|
| /DDT | Loads DDT regardless of the extension of the first file in the command string. This is a permanent switch in that it applies to all subsequent files regardless of its position in the command string. |
| /FOROTS | Loads the file with FOROTS (the new FORTRAN object time system) instead of FORSE. This switch affects FORTRAN files only. |
| /FORSE | Loads the file with FORSE (the old FORTRAN object time system) instead of FOROTS. This switch affects FORTRAN files only. |

LINK-10

Table 2-1 (Cont)

COMPIL Switches Pertaining to Loading

| | |
|---|---|
| /LIBRARY | The action is identical to that of the /SEARCH switch. The use of the /SEARCH switch is recommended since it is the complement of /NOSEARCH. |
| /LINK | Causes the files to be loaded by the LINK-10 program instead of the LOADER program. If used, this switch must be placed before any file specifications (either implied or explicit) since the COMPIL program may have to generate load-control switches. |
| /LMAP | Produces a loader map during the loading process (same action as /MAP) containing the local symbols. |
| /LOADER | Causes the file to be loaded by the LOADER program instead of the LINK-10 program. Since this is the current default action, this switch is needed only if the installation has specified LINK-10 as the default linking-loader. In a future release, LINK-10 will become the standard default. |
| /MAP | Produces a load map during the loading process. The map does not contain local symbols. When this switch is encountered, a loader map is requested from LINK-10. After |

Table 2-1 (Cont)

COMPIL Switches Pertaining to Loading

| | |
|---|---|
| | the library search of the default system libraries, the map is written in the user's disk area with the filename specified by the user (e.g., /MAP:dev:file.ext[directory]) or the default filename (e.g., the name of the last program seen with a start address or nnnLNK.MAP (where nnn is the user's job number) if there is no such program). This switch is an exception to the permanent switch rule in that it causes only one map to be produced even though it may appear as a permanent switch. |
| /NOSEARCH | Loads all routines of the file whether the routines are referenced or not. Since this is the default action, this switch is used only to turn off library search mode (/LIBRARY or /SEARCH). This switch is not equivalent to the /NOSYSLIB switch of LINK-10, which does not search any libraries, including the default system libraries. The /NOSEARCH default is to search the default system libraries. |
| /SEARCH | Loads the files in library search mode. This mode causes a module in a special library file to be loaded only if one or more of its |

Table 2-1 (Cont)
COMPIL Switches Pertaining to Loading

| | |
|---|---|
| | declared entry symbols satisfies an undefined global request. The default system libraries are always searched regardless of the state of this switch. |

## 2.3 SPECIFYING DISK AREAS OTHER THAN SYS

When translating his source files, the user has the option of selecting the disk area from which the language translator is obtained. The disk areas are [1,3] for OLD, [1,4] for SYS, [1,5] for NEW, and the user's area for DSK and are specified by the switches /OLD, /SYS, /NEW, and /SELF, respectively. (These four switches are described in DECsystem-10 OPERATING SYSTEM COMMANDS.) For example, if the user is translating his source files with a FORTRAN compiler that is on the OLD disk area of [1,3], he gives the following command string:

```
COMPILE/OLD FILEA.F4,FILEB.F4,FILEC.F4
```

The FORTRAN compiler is then obtained from area [1,3].

The first disk area seen in the command string is also the area from which LINK-10 is obtained. Thus, in the command string:

```
LOAD /LINK /OLD FILEA.F4,FILEB.F4,FILEC.F4
```

not only is the FORTRAN compiler obtained from OLD, but also the LINK-10 linking-loader. If LINK-10 is not found on the specified area, then the SYS disk area of [1,4] is searched. However, if the first disk area seen is the user's area (as indicated by the /SELF switch), only the areas specified in the user's job search list, which may include a user library (LIB), are searched. The searching does

not continue onto the NEW, OLD, and SYS areas. Thus, a user who is using a copy of a translator in his disk area but who does not have a copy of LINK-10 in that area must use two disk area specifications. For example,

```
LOAD /LINK /SYS /SELF FILEA.FOR,FILEB.FOR,FILEC.FOR
```

LINK-10 is obtained from the SYS disk area and the FORTRAN compiler from the user's disk area. Since SYS will be searched for LINK-10 on all disk specifications other than SELF, the user needs to specify two disk areas only when he is using a translator from his area.

## 2.4 SAVE AND SSAVE SYSTEM COMMANDS

After loading is completed, the loaded program may be written onto an output device so that it can be executed at some future date without rerunning LINK-10. The SAVE and SSAVE system commands output the core image onto the specified device as one or two files. If the SAVE command is used, the program will be nonsharable when it is later loaded into core. When the SSAVE command is used, the high segment (if any) of the program will be sharable when the program is loaded. The general command format of the two commands is the same:

```
SAVE dev:file.ext[directory]core

SSAVE dev:file.ext[directory]core
```

where

dev: is the name of the device on which to write the saved file. If omitted, DSK: is assumed.

file is the name of the saved file. If omitted, the job's current name is used. This name is set by the last R, RUN, GET, SAVE, or SSAVE system command, the last command which ran a

program (e.g., DIRECT), or the last SETNAM UUO.

.ext is the extension of the low segment file. If omitted, the following extensions are assigned:

If the program has one segment, the extension .SAV is assigned.

If the program has two segments, the low segment file has the extension .LOW, and the high segment file has the extension .HGH when a SAVE command is used and the extension .SHR when a SSAVE command is used.

[directory] is the area in which to save the file. If omitted, the user's default directory is used.

core is the amount of core in which to save the program. If omitted, the minimum required is assigned.

Refer to DECsystem-10 OPERATING SYSTEM COMMANDS for complete descriptions on the SAVE and SSAVE commands.

## 2.5 COMPIL Examples

In the following example, the user is translating, loading, and executing a MACRO program. The /LINK switch requests that the LINK-10 linking loader be used instead of the LOADER.

```
.EXECUTE /LINK SIMPLE,MAC )
MACRO: SIMPLE
LINK: LOADING
[EXECUTION]
THIS IS A VERY SIMPLE TWO-SEGMENT MACRO PROGRAM,

EXIT
```

LINK-10

COMPIL Examples

In the example below, the user is compiling, loading, and executing three COBOL programs. The /MAP:PROGMP.MAP switch requests the generation of a map file with the name PROGMP.MAP.

```
.EXECUTE /LINK /MAP:PROGMP FILA,FILB,FILC )
COBOL: CBS08A [FILA,CBL]
COBOL: CBS08B [FILB,CBL]
COBOL: CBS08C [FILC,CBL]
LINK:  LOADING
[EXECUTION]
RUNNING CBS08A
RUNNING CBS08B
RUNNING CBS08C

EXIT
```

The map file is now on the user's disk area. He can print the file with the following command:

```
.PRINT PROGMP.MAP )
TOTAL OF 3 BLOCKS IN LPT REQUEST
```

The following is a listing of the map file generated.

LINK-10

COMPIL Examples

```
                LINK-10 SYMBOL MAP OF   PROGMP          PAGE 1
     PRODUCED BY LINK-10 VERSION (32) ON  2-APR-73 AT 8:28:10

LOW  SEGMENT STARTS AT      0 ENDS AT    2416 LENGTH     2416 = 2K
       STARTING ADDRESS IS  1250, LOCATED IN PROGRAM CBS00A

                ***************

JOBDAT-INITIAL-SYMBOLS

        ZERO LENGTH MODULE

                ***************

LIBOL-STATIC-AREA
     LOW  SEGMENT STARTS AT     140 ENDS AT    1200 LENGTH    1040 (OCTAL), 544 (DECIMAL)

        .COMM.          140     COMMON          LENGTH          544     (DECIMAL)

                ***************

CBS00A  FROM DSK:FILA.REL[27,235]       CREATED BY COBOL ON  2-APR-73 AT 8:24:00
     LOW  SEGMENT STARTS AT    1200 ENDS AT    1355 LENGTH     155 (OCTAL), 109 (DECIMAL)

        CBS00A          1270    ENTRY   POINT   RELOCATABLE

                ***************

CBS00B  FROM DSK:FILB.REL[27,235]       CREATED BY COBOL ON  2-APR-73 AT 8:24:00
     LOW  SEGMENT STARTS AT    1355 ENDS AT    2040 LENGTH     463 (OCTAL), 307 (DECIMAL)

        CBS00B          1464    ENTRY   POINT   RELOCATABLE

                ***************
```

LINK-10

COMPIL Examples

```
CBS08C  FROM DSK:FILC REL[27,235]     CREATED BY COBOL ON  2-APR-73 AT  8:25:00
        LOW  SEGMENT STARTS AT   2040 ENDS AT   2407 LENGTH   347 (OCTAL); 231 (DECIMAL)

        CBS08C          2140    ENTRY   POINT   RELOCATABLE

                ***************

TRACED  FROM SYS:LIBOL.REL[1,4] CREATED ON 23-MAR-73 AT 16:40:00      7 (OCTAL); 7 (DECIMAL)
        LOW  SEGMENT STARTS AT   2407 ENDS AT   2416 LENGTH

        BTRAC,          2412    ENTRY   POINT   RELOCATABLE
        CBDDT,          2413    ENTRY   POINT   RELOCATABLE
        PTFLG,          2414    GLOBAL  SYMBOL  RELOCATABLE
        TRACE,          2407    ENTRY   POINT   RELOCATABLE
        TRPD,           2412    ENTRY   POINT   RELOCATABLE
        TRPOP,          2412    ENTRY   POINT   RELOCATABLE

                ***************

        [END OF LINK-10 MAP OF  PROGMP]
```

LINK-10

Summary

## 2.6 SUMMARY

The LOAD, EXECUTE, and DEBUG system commands, along with the switches described in Table 2-1, are sufficient for loading and executing most programs. The user can load separately-compiled programs and debugging programs, obtain maps, search files in a library search mode, and execute the program. To produce a saved file of his core image, the user can employ the system commands SAVE and SSAVE. More complex loading procedures can be performed by directly using LINK-10, as described in Chapter 3.

LINK-10

Using LINK-1Ø

# CHAPTER 3

# USING LINK-10

The user runs LINK-10 directly by issuing the system command

R LINK

LINK-10 responds with an asterisk at which point the user types in his command strings. The LINK-10 program interprets all of the input typed by the user up to the end of the command string. A command string is defined as a series of characters terminated by a carriage return-line feed. A carriage return-line feed is generated when the user depresses the RETURN key on his terminal. The RETURN key is represented in this manual by the symbol ). If the user needs to continue a command string on another line, he can place a hyphen as the last non-blank, non-comment character before the carriage return-line feed. Continuation lines are considered part of the current command string, and the current string is not considered terminated until a carriage return-line feed is seen without a preceding hyphen. Comments may be added to any line by preceding the comment with a semicolon. Trailing spaces and tabs (including those before comments) are always ignored.

When the command string is terminated, LINK-10 processes the data in the command string by performing the actions specified by the user. This usually entails setting relevant internal conditions and storing information for later use. Each command string is completely scanned and processed before LINK-10 accepts a new one. After scanning and

processing the current command string, LINK-10 returns with another asterisk signifying its readiness to accept more input. The program accepts command string input until the user gives the exit condition switch (/GO) indicating that LINK-10 is to finish all loading tasks. At this point control is either returned to the operating system or given to the loaded program for execution, depending upon the preceding command strings.

## 3.1 LINK-10 COMMAND STRINGS

Command strings to LINK-10 contain a series of input and/or output file specifications and non-conflicting switches to direct the loading process. The general command string format is as follows:

*output specifications=input specifications

Any number of specifications can be included in the command string by separating each specification from other specifications with a comma. Although the equals sign is not required, it is recommended that the user include it so that he can distinguish his output specifications from his input ones. If the user does not include an equals sign, he must use a comma to separate the specifications. The input and output specifications are then distinguished by the type of switch associated with the specification, and the specifications can appear in any order (e.g., input specifications can precede output specifications).

An input or output specification consists of a file specification and switches appearing before and/or after the file specification. A file specification is in the form

LINK-10

Command Strings

dev:file.ext[directory]

and the individual switches that can be used in the command string are described in Chapter 4.

When items in a file specification are missing, LINK-10 has a set of initial values to be used as defaults. On input specifications, the default values assumed for missing items in a file specification are as follows:

| | |
|---|---|
| Device | DSK: |
| Filename | A blank filename |
| Extension | .REL |
| Directory | The user's default directory |

On output specifications, the default values are as follows:

| | |
|---|---|
| Device | DSK: |
| Filename | Name of the last program containing a start address. If there is no program with a start address, the name nnnLNK, where nnn is the user's job number, is used. |
| Extension | Dependent on the type of output file requested via switches. |

| | |
|---|---|
| Log file | .LOG |
| Map file | .MAP |
| Saved file | .SHR,.HGH,.SAV,.LOW |
| Symbol file | .SYM |
| Expanded save file | .XPN |

| | |
|---|---|
| Directory | The user's default directory. |

These defaults are applied just prior to initializing the device and opening the file, and are used only if the user has not given values for items in a file specification. The initial LINK-10 defaults for items in a file specification are used only when a value for the item

does not appear in the command string or until the value is seen if it is after the beginning of the string.

If a component of a file specification is given before the filename, it remains in effect until changed by a value given subsequently by the user for the same component or until the end of the command string. For example, a user can specify a device name at the beginning of the string and not have to repeat the device name for each specification if he is using the same device for all specifications in the command string. However, once the device name is changed, the new name is used as the default device for the reaminder of the command string.

As another example, the user can specify an extension and a directory to be used by issuing a command string such as

```
*.BIN[10,7]DSKB:FIL1,DSKC:FIL2.REL[10,20],DSKA:FIL3 ↵
```

The extension .BIN and the directory [10,7] are used for any specifications that do not include an extension or directory. The above command string is equivalent to

```
*DSKB:FIL1.BIN[10,7],DSKC:FIL2.REL[10,20],DSKA:FIL3.REL[10,7] ↵
```

## 3.2 CHANGING DEFAULTS

The /DEFAULT switch is used to change the initial values that are assumed when the user does not include a component of a file specification in his command string. The values specified with this

switch remain in effect for the entire load unless changed by another /DEFAULT switch. The form of the /DEFAULT switch is as follows:

components of file specification /DEFAULT:keyword

where

components of file specification are the components which the user wants as his default components.

keyword is either INPUT or OUTPUT to change the default components for the input or output specifications, respectively. If this argument is omitted, INPUT is assumed.

For example, the following specification

DSKB: .BIN[10,20]/DEFAULT

changes the values to be used as defaults for the input specifications to be DSKB: for the device, .BIN for the extension, and [10,20] for the directory.

NOTE

Because the extensions for output files depend upon the types of file being requested, the user cannot change the output extensions. Any attempt to do so is ignored.

## 3.3 LINK-10 SWITCH ALGORITHMS

LINK-10 allows the user to request various loading parameters via switches in the command string. Switches are used to specify output files, to set defaults, to control the loading of programs, to set values, to format maps and symbol tables, to request values of symbols, and to position devices. Some switches merely change the status of LINK-10 by setting internal values; others request immediate

action to be taken.

LINK-10 has several categories of switches with a specific algorithm for the handling of each category. These categories are:

- Device Switches
- File Dependent Switches
- Output Switches
- Immediate Action Switches
- Delayed Action Switches
- Switches that create implicit file specifications

### 3.3.1 Device Switches

Switches in this category (e.g., /SKIP, /REWIND) affect the device within an input or output specification. The switch is in effect after the device is initialized and, depending on its position, either before or after the file is read or written. If the switch appears before the filename, the appropriate action is taken before the file is processed, and if it appears after the filename, action is taken after the file is processed. Switches in this category apply only to the current input or output specification and do not carry over to subsequent devices. In other words, once the requested action is performed, it is not performed again unless another device switch is given.

For example, the following specification may be given by the user:

```
/SKIP:2 MTA1:MYFILE/UNLOAD,
```

LINK-10

Switch Algorithms

After the magnetic tape is initialized, LINK-10 skips forward over two files (/SKIP:2), reads the file called MYFILE, and after reading the file, rewinds and unloads the tape (/UNLOAD).

### 3.3.2 File Dependent Switches

Switches belonging to this category (e.g.,/NOLOCAL, /SEARCH) modify the loading or the contents of a file. These switches are either temporary or permanent in nature. A temporary switch applies only to the file specification immediately preceding it. An intervening comma cannot separate the file specification and the switch. A permanent switch appears before the file specification and applies to all file specifications following it until modified by a subsequent switch or until the end of the current command string is reached. (Remember that continuation lines are considered part of the current command string). This means that permanent file-dependent switches, unlike device switches, continue to apply to following specifications (i.e., the action requested by the switch is not terminated at the comma which separates specifications).

For example, the following specifications may be issued by the user:

```
,/NOLOCAL DTA3:MAIN1,MAIN2,MYLIB/SEARCH,
```

Two files, MAIN1 and MAIN2, are loaded in their entirety from DTA3 without their local symbols. The file MYLIB is searched and parts of it are loaded only if required (i.e., they are required to satisfy any undefined symbol requests); if needed, they are also loaded without local symbols.

LINK-10

Switch Algorithms

### 3.3.3 Output Switches

Switches in this category (e.g.,/MAP, /LOG, /SAVE) initialize the output devices and create the output files. Each output specification must contain one of these switches because LINK-10 does not create output files unless explicitly requested to do so. Each switch represents a specific type of output file and is used with a file specification to indicate the device and filename of the file. Only one output switch can be used with each output specification. If the switch is the only item appearing in the output specification, the device name and filename are taken from the previous specification or from the LINK-10 defaults for output.

For example, if the user desires a saved file and a map file on DSKB: and both with the name OUTPUT, he can issue the following specifications:

```
DSKB:OUTPUT/SAVE,/MAP=
```

The two files will have the same filename (OUTPUT) but, by default, the extensions will be different (refer to Paragraph 3.1). The comma separating the two switches is required to indicate that two output files are desired. If the user is satisfied with accepting the LINK-10 defaults for output specifications, he can give the following

```
/SAVE,/MAP=
```

LINK-10

Switch Algorithms

NOTE

Although the /LOG switch is considered an output switch, it is handled in a slightly different fashion from the remaining output switches. By assigning a device the logical name LOG before initializing LINK-10, the user receives the log file on the device assigned as LOG, even if he does not include the /LOG switch in his command string. The filename associated with the log file is nnnLNK.LOG, where nnn is the user's job number. The /LOG switch can then be used in the LINK-10 command string to change the filename of the log file. For example,

```
.ASSIGN DSKC:LOG:
.R LINK
*DSKC:MYLOG/LOG
```

renames the log file on DSKC: from nnnLNK.LOG to MYLOG.LOG. If the logical device is not assigned, then the building of the log file begins when the /LOG switch is seen. This results in the initialization timings not being included in the file.

### 3.3.4 Immediate Action Switches

Switches in this category (e.g., /UNDEF, /VALUE, /NOINITIAL, /NOSYM) are processed by LINK-10 as soon as they are seen. These switches are divided into two types:

- Those that request typeout from LINK-10.
- Those that change the status of the loading procedure.

Type-out switches (e.g., /UNDEF) request information from LINK-10 and are not dependent upon a particular specification. For this reason, they can appear anywhere in the command string but are usually on a command line by themselves because the user is interactively requesting information to determine if he may have forgotten to specify needed parameters. After processing the switch (i.e., at the

end of the command string), LINK-10 returns the requested information immediately. Once the information is returned to the user, the switch is cleared.

Status changing switches (e.g.,/NOINITIAL, /NOSYM) are related to the entire loading procedure and not to an individual specification. They are placed in the command string at the point at which the user wants the action to be performed. Once the action has been taken, it is in effect for the entire loading process and cannot be overridden. For example, once the user gives the /NOSYM switch to notify LINK-10 not to generate a local symbol table, he cannot, in the same load, give a switch to LINK-10 to nullify this action.

### 3.3.5 Delayed Action Switches

Switches in this category (e.g., /MAXCOR,/HASHSIZE) are used to change operational parameters of LINK-10 to the specified values. When the switch is seen, LINK-10 accepts the value but does not use it until it is needed. For example, there is a preset value for the maximum core LINK-10 can occupy during loading. Use of the /MAXCOR switch changes this value immediately but LINK-10 does not examine the value until it needs to expand its core size.

### 3.3.6 Switches that Create Implicit File Specifications

Switches in this category (e.g., /DEBUG, /SYSLIB) cause LINK-10 to create one or more input file specifications for programs that must be loaded along with the user's program and to set various other switches related to the implicitly specified file. As an example, the /DEBUG

switch indicates that a debugging program is to be loaded and that subsequent modules are to be loaded with local symbols, unless otherwise specified by the user. If one of these switches appears before the file specification, the program implied by the switch is loaded before the current file. If the switch is after the file specification, the program is loaded after the current file. Once the program implied by the switch is loaded, the switch is cleared.

## 3.4 LINK-10 SWITCHES

Switches to LINK-10 have one of the following forms:

```
/switch
/switch:arg
/switch:(arg,...,arg)
/switch:value
/switch:arg:value
/switch:(arg:value,...,arg:value)
```

where

| | |
|---|---|
| /switch | is the name of the desired switch. This name can be truncated to a unique abbreviation. The first six characters of the name are sufficient to ensure uniqueness. |
| arg | is a keyword or a symbol name. Keywords can be truncated to a unique abbreviation. |
| value | is either a decimal or octal number. An octal value can be used with a switch that accepts decimal values by preceding the octal value with a number sign (#). |
| : | is the separator between components in a switch specification and must be present if more than one item is given. |
| ( ) | are used to enclose multiple keywords and/or values to a switch. They are required if more than one argument appears with the switch. |

LINK-10

LINK-1Ø Switches

NOTE

For the first release of LINK-10, multiple keywords cannot be specified in a single switch specification. This means that the user must issue a switch specification for each desired keyword (e.g., /CONTENTS:LOCAL /CONTENTS: RELOCATABLE). This restriction will be removed in a later release of LINK-10.

Each switch specification must be terminated with a space; however, spaces cannot appear within a switch specification (i.e., between the slash and the end of the value).

Table 3-1 briefly describes the switches that can be used on the LINK-10 command string, and Chapter 4 contains the complete descriptions of the switches in alphabetical order.

Table 3-1

LINK-10 Switches

| Switch | Meaning |
|---|---|
| /BACKSPACE | Spaces backwards over the specified number of files. |
| /COMMON | Allocates a COMMON area. |
| /CONTENTS | Specifies the types of symbols to be output in a map. |
| /CORE | Specifies LINK-10's initial low segment size. |
| /COUNTER | Lists the relocation counters and their values. |
| /CPU | Specifies the processor on which the program will run. |
| /DATA | Loads defined constant data. This switch is not implemented in Version 1. |
| /DEBUG or /D | Loads and specifies execution of a debugging program. |
| /DEFAULT | Changes default values for missing components in a file specification. |
| /DEFINE | Assigns values to undefined global symbols interactively. |
| /ENTRY | Lists library search symbols. |
| /ERRORLEVEL | Selectively suppresses messages to the terminal. |
| /ESTIMATE | Allocates disk space for an output file. |
| /EXCLUDE | Inhibits the loading of specified modules. |
| /EXECUTE or /E | Specifies execution of the program upon completion of loading. |

LINK-10

Table 3-1 (Cont.)

LINK-10 Switches

| Switch | Meaning |
|---|---|
| /FOROTS | Loads FOROTS, if required, during default system library searching. |
| /FORSE | Loads FORSE, if required, during default system library searching. |
| /FRECOR | Specifies the amount of free core guaranteed after each expansion. |
| /GO or /G | Terminates the loading progress. |
| /HASHSIZE | Specifies the size of the global symbol table. |
| /INCLUDE | Forces the loading of specified modules from a library. |
| /LOCALS or /L | Loads with local symbols. |
| /LOG | Causes a log file to be generated. |
| /LOGLEVEL | Suppresses messages to the log file. |
| /MAP or /M | Causes a map file to be generated. |
| /MAXCOR | Specifies LINK-10's maximum low segment core size. |
| /MPSORT | Sorts the symbol table for output to the map file. |
| /MTAPE | Performs magnetic tape functions. |
| /NOINITIAL | Clears the initial global symbol tables. |
| /NOLOCAL or /N | Loads without local symbols. |
| /NOSEARCH | Turns off user library search mode. |

Table 3-1 (Cont.)
LINK-10 Switches

| Switch | Meaning |
|---|---|
| /NOSTART | Ignores starting addresses. |
| /NOSYMBOL | Inhibits the generation of a symbol table in core. |
| /NOSYSLIB | Prevents a search of the default system libraries. |
| /OTS | Indicates the segment for the object time system. |
| /PATCHSIZE | Allocates patch space. |
| /REQUIRE | Generates global requests for the specified symbols. |
| /REWIND | Rewinds the DECtape or magnetic tape. |
| /RUNCOR | Assigns the initial low segment core size for the program. |
| /RUNAME | Assigns the program name. |
| /SAVE | Causes a saved file to be generated. |
| /SEARCH or /S | Turns on user library search mode. |
| /SEGMENT | Specifies the segment in which to load the modules. |
| /SET | Defines the values of a relocation counter. |
| /SEVERITY | Defines the fatality level of errors. |
| /SKIP | Spaces forward on a magnetic tape. |
| /SSAVE | Causes a sharable saved file to be generated. |
| /START | Specifies the start address of a program. |
| /SYMBOL | Causes a symbol file to be generated. |

LINK-10

Table 3-1 (Cont.)
LINK-10 Switches

| Switch | Meaning |
|---|---|
| /SYMSEG | Moves the symbol table to the specified segment. |
| /SYSLIB | Performs a search of the default system libraries. |
| /SYSORT | Sorts the symbol table for output to the symbol file. |
| /TEST | Loads a debugging program. |
| /UNDEFINED or /U | Types undefined global symbols on the terminal. |
| /UNLOAD | Rewinds and unloads the DECtape or magnetic tape. |
| /VERBOSITY | Specifies the amount of text to be printed for a message. |
| /VALUE | Lists the current values of the specified global symbols. |
| /XPN | Creates or saves the expanded core image file. |
| /ZERO | Clears the specified DECtape directory. |

LINK-10

Switches

# CHAPTER 4

# LINK-10 SWITCHES

## /BACKSPACE

### Function

The /BACKSPACE switch is used to space backwards over the specified number of files. This switch has an effect only on tape devices and is ignored for non-tape devices.

### Switch Format

/BACKSPACE:n

n is a decimal number representing the number of files to backspace over. If n is omitted, n=1 is assumed.

### Category of Switch

Device Switch (refer to Paragraph 3.3.1)

### Examples

,MTA0:/BACK:3,

Backspace MTA0 by three files.

LINK-10

Switches

/COMMON

Function

The /COMMON switch is used to allocate an area of storage of the specified size before loading any more code. An array of storage (a COMMON area) is reserved into which data can be placed in order that it may be shared by several programs and routines. Because the FORTRAN language contains a statement that reserves space for a COMMON area, this switch is used to reserve COMMON arrays when loading non-FORTRAN programs or to allocate a different size area than given via the COMMON statement in a FORTRAN program. However, if this switch is used to allocate a larger size area of the same name as that given in the FORTRAN program, the switch specification must be given before the FORTRAN program is loaded.

The name of each labeled area of COMMON storage is defined as an internal symbol whose value is the address of the first word of the COMMON area. These symbols may be used by other programs as external symbols.

Switch Format

/COMMON:name:n

Name is the symbolic name of up to six SIXBIT characters of the COMMON area. Blank COMMON is designated with the symbolic name ".COMM.".

LINK-10

Switches

n is a decimal number representing the size of the area in words.

Restrictions

Although various modules may redefine COMMON areas of the same name, the size of a COMMON area cannot be increased during the loading process. Therefore, the largest definition of a given COMMON area must be loaded first. Any attempt to increase the size of a COMMON area by redefinition will result in a fatal error. This applies to both modules defining COMMON areas and the /COMMON switch.

Category of Switch

Immediate Action Switch (refer to Paragraph 3.3.4)

Examples

/COMMON:.COMM.:1000

Allocate blank COMMON to be 1000 words.

LINK-10

Switches

/CONTENTS

Function

The /CONTENTS switch gives the user control over the contents of the map file by allowing him to specify the types of symbols to be included in the file. Each symbol is marked as to its type by the translator that processed the module containing the symbol. Some symbols may be of more than one type. For example, a symbol may be both a global symbol and a relocatable symbol. To insure the inclusion of such a symbol in the map file, the user must specify both the GLOBAL and the RELOCATABLE keywords in the /CONTENTS switch.

Each specification of the /CONTENTS switch is cumulative; keywords set by the first specification are not automatically cleared by the second specification. If the user desires to clear a keyword set in a previous specification, he must explicitly specify its complement.

NOTE

This switch does not produce a map file. The user must specify the /MAP switch on an output specification in order to obtain the file. Unless the /MAP is given, the /CONTENTS switch has no meaning and is ignored.

Switch Format

/CONTENTS:keyword

/CONTENTS:(keyword,. . ., keyword)

LINK-10

Switches

Keywords are as follows:

| | |
|---|---|
| ABSOLUTE | include all absolute symbols (usually flags, accumulators, and masks). Complement of NOABSOLUTE. |
| ALL | include all symbols. Complement of NONE. |
| COMMON | include all COMMON symbols. Complement of NOCOMMON. |
| DEFAULT | include the symbols according to LINK-10's default setting, that is: COMMON, GLOBAL, ENTRY, ABSOLUTE, RELOCATABLE, NOLOCAL, and NOZERO. This keyword is used to reset the /CONTENTS switch to the original default setting. |
| ENTRY | include all entry name symbols. Complement of NOENTRY. |
| GLOBAL | include all global symbols including COMMON and ENTRY symbols unless these symbols are suppressed with the NOCOMMON and NOENTRY keywords. Complement of NOGLOBAL. |
| LOCALS | include all local symbols. Complement of NOLOCAL. |
| NOABSOLUTE | do not include absolute symbols (i.e., turn off the condition corresponding to absolute symbols). Complement of ABSOLUTE. |
| NOCOMMON | do not include COMMON symbols. Complement of COMMON. |
| NOENTRY | do not include entry name symbols. Complement of ENTRY. |
| NOGLOBAL | do not include global symbols including COMMON and ENTRY symbols unless these symbols are requested with the COMMON and ENTRY keywords. Complement of GLOBAL. |
| NOLOCAL | do not include local symbols. Complement of LOCALS. |
| NONE | do not include any symbols of any kind. However, header information is still output in the map. Complement of ALL. |
| NORELOCATABLE | do not include relocatable symbols. Complement of RELOCATABLE. |

LINK-10

Switches

| | |
|---|---|
| NOZERO | do not include symbols from zero length programs. Complement of ZERO. |
| RELOCATABLE | include symbols that are relocatable (usually addresses). Complement of NORELOCATABLE. |
| ZERO | include symbols from zero length modules (usually parameter files). A zero length module is one which defines symbols but generates no code. Complement of NOZERO. |

If the /CONTENTS switch is not specified, the default setting is COMMON, GLOBAL, ENTRY, RELOCATABLE, ABSOLUTE, NOLOCAL, and NOZERO. When the user specifies a keyword, the keyword is either added to the default setting or deleted from the default setting. For example, if the user issues the /CONTENTS:ZERO switch, the condition for symbols in zero length programs is added to the default setting. However, the keywords ALL, NONE, and DEFAULT reset the default setting to their respective meanings.

Category of Switch
---

Delayed Action Switch (refer to Paragraph 3.3.5)

Examples
---

/CONTENT:ZERO,/CON:LOCAL,

Include in the map local symbols and symbols from zero length modules, in addition to the types of symbols in LINK-10's default setting.

LINK-10

Switches

/CORE

Function

The /CORE switch is used to specify the initial size of LINK-10's low segment. Generally, this size is less than or equal to MAXCOR (the maximum size of LINK-10's low segment). If the size specified in the /CORE switch is greater than MAXCOR, the core will be assigned. However, the next time LINK-10 needs to expand core, the size will be reduced to MAXCOR.

Switch Format

/CORE:n

n is a decimal number that represents the initial low segment core size for LINK-10. An octal value can be given by preceding it with a number sign (#). N is expressed in units of 1024 words or 512 words (a page) by following the number with K or P respectively. If K or P is omitted, K (1024 words) is assumed.

Category of Switch

Immediate Action Switch (refer to Paragraph 3.3.4)

Examples

/CORE:17K

Specify 17K words as the initial size of LINK-10's low segment.

LINK-10

Switches

/COUNTER

Function

The /COUNTER switch is used to output to the terminal the relocation counters, their initial and current values, and for undefined counters, the length of code depending on them. When a relocation counter is not known, a count of the amount of core used by the counter is kept so that loading can be resolved. Code depending on the counter is stored on the disk until the counter is defined.

Although LINK-10 is designed to handle an indefinite number of relocation counters to provide efficient program construction, the first release of LINK-10 only uses two relocation counters, the low segment counter (.LOW.) and the high segment counter (.HIGH.). These counters are listed in a map file with their initial and final values.

Switch Format

/COUNTER

Category of Switch

Immediate Action Typeout Switch (refer to Paragraph 3.3.4)

LINK-10

Switches

Examples

/COUNTER

| RELOCATION COUNTER | INITIAL VALUE | CURRENT VALUE |
|---|---|---|
| .LOW. | 0 | 140 |
| .HIGH. | 400000 | 400010 |

LINK-10

Switches

/CPU

Function

The /CPU switch is used to indicate the central processor on which the program will run once it has been loaded.

Switch Format

/CPU:keyword

Keyword is either KA10 or KI10. If the keyword is omitted, KA10 is assumed. If the /CPU switch is omitted, the machine on which the program is loaded is assumed.

Category of Switch

Delayed Action Switch (refer to Paragraph 3.3.5)

Examples

/CPU:KI10

Run the program on the KI10 processor.

LINK-10

Switches

/DEBUG

Function

The /DEBUG switch is used to load a debugging program and to specify that execution of the load will begin at the normal start address of the debugging program instead of the user's program. The debugging programs available are DDT, MANTIS, and COBDDT. This switch does not cause termination of the loading procedure, the /GO switch is needed for termination. The /EXECUTE switch is not used for execution when the /DEBUG switch is given.

The /DEBUG switch turns on the load with local symbols mode and causes it to be in effect for the remainder of the load unless overridden by the /NOLOCALS switch. However, since the /NOLOCALS switch is file dependent, it is cleared at the end of the command string in which it appears and local symbols mode is reinstated. Note that the /LOCALS switch is also file dependent; therefore, the use of the /LOCALS switch and the implicit use of the /LOCALS switch in the /DEBUG switch context have different results (i.e., the /LOCALS switch is cleared at the end of the command string and the load with local symbols mode implied by the /DEBUG switch is not).

The /DEBUG switch does not cause the local symbols of the debugging program to be loaded, regardless of the state of the /LOCALS switch.

LINK-10

Switches

Switch Format

/DEBUG:keyword

Keyword is one of the following: COBDDT, COBOL, DDT, FORTRAN, MACRO, MANTIS. When a compiler or the assembler is specified, the debugging aid associated with that translator is used. For example, if MACRO is specified, the loading of DDT is implied. If the keyword is omitted, DDT is assumed.

Category of Switch

Creates an implicit file specification (refer to Paragraph 3.3.6)

Examples

```
,/DEBUG:DDT DTA3:FILEA.MAC,
```

LINK-10

Switches

/DEFAULT

Function

The /DEFAULT switch is used to change LINK-10's initially-assumed values for components missing in a file specification. A file specification is in the form dev:file.ext[directory]. The initial defaults for input specifications are

DSK:.REL [user's default directory]

and for output specifications are

DSK:name of main program.ext dependent on type of output file [user's default directory].

Thus, the user cannot change the extensions of output files, and any attempt to do so is ignored.

Values specified via the /DEFAULT switch are in effect for the entire loading process or until the user issues another /DEFAULT switch.

Switch Format

/DEFAULT:keyword

Keyword is either INPUT or OUTPUT to specify default conditions for input and output specifications, respectively. If the keyword argument is omitted, INPUT is assumed.

Category of Switch

Immediate Action Switch (refer to Paragraph 3.3.4)

LINK-10

Switches

Examples
---

DSK:MAIN,/DEFAULT .BIN[10,7],

Load the file MAIN.REL from the user's default directory of the disk and then change the input defaults to load .BIN files from the [10,7] area of the disk.

LINK-10

Switches

/DEFINE

Function

The /DEFINE switch is used interactively by the user to assign values to undefined global symbols in order to satisfy global requests before LINK-10 terminates the load with undefined symbols. The user can employ the /UNDEF switch to obtain a list of the undefined symbols and then use the /DEFINE switch to satisfy the requests for these symbols.

Switch Formats

/DEFINE:symbol:value

/DEFINE:(symbol:value, . . .,symbol:value)

Symbol is the name of the symbol to be defined. If the name given is one of an already-defined symbol, the user receives an error message.

Value is the decimal number to be associated with the symbol. An octal value can be given by preceding it with a number sign (#).

LINK-10

Switches

Category of Switch

Immediate Action Switch (refer to Paragraph 3.3.4)

Examples

```
*/UNDEF)
1 UNDEFINED SYMBOL
NOW 400123
*/DEFINE:NOW:897)
*/DEFINE:OCT:#1234)
```

LINK-10

Switches

/ENTRY

Function

The /ENTRY switch is used to type out all library search symbols (i.e., entry points) that have been loaded up to the time the switch is given. These symbols are recognized by a specific condition set in the first word of the symbol by the translator that processed the module containing the symbol. The user defines symbols as library search symbols with an ENTRY statement in a MACRO-10 or BLISS-10 module, with a SUBROUTINE, FUNCTION, or ENTRY statement in a FORTRAN module, or with a SUBROUTINE statement in a COBOL module.

This switch is useful for the future overlay facility of LINK-10.

Switch Format

/ENTRY

Category of Switch

Immediate Action Typeout Switch (refer to Paragraph 3.3.4)

Examples

```
*/ENTRY)
Library Search Symbols
SQRT.  3456
```

LINK-10

Switches

/ERRORLEVEL

Function

The /ERRORLEVEL switch is used to selectively suppress LINK-10 messages to the user's terminal. Associated with each message is a decimal number from 0 to 31 called the message level. Via this switch, the user can decide that messages with a message level less than or equal to a specific number are not to be output to his terminal. A user would normally want to suppress informative messages rather than error messages. The higher the message level, the more serious the message. Refer to Chapter 5 for the message level of each LINK-10 message.

Switch Format

/ERRORLEVEL:n

n is a decimal number from 0 to 30. Messages with a message level less than or equal to n will not be output to the terminal. Note that a message with a level of 31 cannot be suppressed. If this switch, or the value of the switch, is omitted, informative messages are suppressed.

Category of Switch

Delayed Action Switch (refer to Paragraph 3.3.5)

Examples

/ERRORLEVEL:10

LINK-10

Switches

/ESTIMATE

Function

The /ESTIMATE switch is used to reserve disk space for an output file and must be associated with an output specification. Because each occurrence of the switch allocates space for only one file, the user must issue an /ESTIMATE switch for each file that needs space reserved.

This switch is not required for space allocation for an output file, but its use can both help the user stay within his quota allotment and reduce the number of (RIB) pointers associated with the file.

Switch Format

/ESTIMATE:n

n is a decimal number representing the estimated number of blocks of 128 words of the output file. A warning message is given if LINK-10 fails to allocate the requested size.

If this switch is omitted, or if an insufficient estimate is given, space is allocated automatically as needed.

LINK-10

Switches

Category of Switch
---

Output Switch (refer to Paragraph 3.3.3)

Examples
---

DSKC:OUTPUT/MAP/ESTIMATE:50,/SAVE/ESTIMATE:200,

Allocate 50 blocks for the map file and 200 blocks for the save file.

LINK-10

Switches

/EXCLUDE

Function

The /EXCLUDE switch is used to inhibit the loading of certain modules in a file when loading the file in the current mode (either search or nonsearch mode). This switch is useful when the user is searching a library file and definitely knows he does not want certan modules, even though his program may reference the names of these modules. For example, if a library file has several modules with the same library search symbols (e.g., as in dummy routines) and the user wants to load a module other than the first one, he can use this switch to prevent the loading of the modules not desired. Another use of the /EXCLUDE switch is to satisfy global symbol definitions during library searching by excluding the modules that would cause multiply-defined symbols.

Switch Formats

/EXCLUDE:symbol

/EXCLUDE:(symbol, . . ., symbol)

Symbol is the name of the module.

LINK-10

Switches

Category of Switch
---

File Dependent Switch (refer to Paragraph 3.3.2)

Examples
---

,/SEARCH LIBFIL.REL/EXCLUDE:(MOD1,MOD2),

Search the file LIBFIL as a library but do not load the modules MOD1 and MOD2 from the file, even if they are referenced.

LINK-10

Switches

/EXECUTE

Function

The /EXECUTE switch is used to specify that the loaded program is to be started at the normal entry point (i.e., the start address) upon completion of loading. This switch does not cause the termination of loading; the /GO switch is needed to terminate loading.

The /EXECUTE and /DEBUG switches cannot be used together because one switch specifies execution of the user's program (/EXECUTE) and the other switch specifies execution of the debugging program (/DEBUG).

Switch Format

/EXECUTE

Category of Switch

Delayed Action Switch (refer to Paragraph 3.3.5)

Examples

/EXE

LINK-10

Switches

/FOROTS

Function

The /FOROTS switch is used to specify the object time system FOROTS, instead of FORSE, for use with FORTRAN programs. FOROTS is then loaded, if required, when LINK-10 searches the default system libraries.

Switch Format

/FOROTS

Category of Switch

Creates an implicit file specification (refer to Paragraph 3.3.6)

Examples

,/FOROTS DSK:MAIN,SUB1,

LINK-10

Switches

/FORSE

Function

The /FORSE switch is used to specify the object time system FORSE, instead of FOROTS, for use with FORTRAN programs. FORSE is then loaded, if required, when LINK-10 searches the default system libraries.

Switch Format

/FORSE

Category of Switch

Creates an implicit file specification (refer to Paragraph 3.3.6)

Examples

```
,DSK:MAIN.F4/FORSE,
```

LINK-10

Switches

/FRECOR

Function

The /FRECOR switch guarantees that the specified amount of free core will remain after LINK-10 expands specific areas in its low segment. Since LINK-10's default amount of free core is 2K, users do not need this switch when loading most modules. However, when the modules being loaded are quite large (e.g., monitor modules), a larger amount of FRECOR will result in a faster loading process because LINK-10 will not have to move areas around in core as often.

During the loading procedure, LINK-10 has five areas that can be expanded beyond their initial sizes. These areas are: the user's low segment code area (LC), the user's high segment code area (HC), the local symbol table area (LS), the fixup area (FX), and the global symbol table area (GS). Each area has a lower boundary, a maximum upper boundary, and an actual upper boundary. LINK-10 tries to maintain space between the actual upper boundary and the maximum upper boundary at all times. However, as the loading procedure progresses, LINK-10 may have to expand an area to accomodate the user's input. If the sum of the amount of free core between the actual upper boundary and the maximum upper boundary for all areas minus the size required for the expansion is less than FRECOR, core is expanded to an amount large enough to maintain FRECOR. If the required size of the low segment becomes greater than MAXCOR (the user specified limit) or CORMAX

LINK-10

Switches

(the system limit) allows, no further expansion is attempted and core is obtained from the free space recovered by shuffling areas. When all of the free space has been obtained, some or all of the above-mentioned areas must overflow to the disk. Note that free core is not maintained when areas overflow to the disk.

Switch Format

/FRECOR:n

n is a decimal number representing the number of words of free core rounded to the next 128-word multiple. If this switch, or the value of this switch, is omitted, 2K words is assumed.

Category of Switch

Delayed Action Switch (refer to Paragraph 3.3.5)

Examples

/FRECOR:3K

LINK-10

Switches

/GO

Function

The /GO switch is used to terminate the loading process and is the only termination switch available. When LINK-10 executes the /GO switch, it finishes loading the current specification, searches default libraries (if this action has not been suppressed with the /NOSYSLIB switch), produces the requested output files, and either exits to the monitor or runs the core image produced depending upon the switches appearing in the input command strings. If the /DEBUG switch has been specified, execution begins at the normal start address of the appropriate debugging program. If the /EXECUTE or /TEST switch has been specified, execution begins at the normal start address of the user's program. If one of these switches has not been specified, LINK-10 exits to the monitor.

Switch Format

/GO

Category of Switch

Immediate Action Switch (refer to Paragraph 3.3.4)

Example

```
/GO
EXIT
.
```

LINK-10

Switches

/HASHSIZE

Function

The /HASHSIZE switch is used to specify the initial size of the global symbol table. LINK-10 uses the lowest prime number in its internal list that is greater than or equal to the given value as the hashsize for the symbol table. This switch can be employed by a user who knows before loading that the number of global symbols used by his program is going to be quite large. By setting the hashsize of the symbol table to a larger number, the user can save LINK-10 time and space that would be used in expanding the hash table. When the user receives the message REHASHING GLOBAL SYMBOL TABLE on a load, it serves as an indication that he should use the /HASHSIZE switch at the beginning of subsequent loads of the same programs. Refer to the LINK-10 Design Specification for the hashing technique used in symbol tables.

Switch Format

/HASHSIZE:n

n is a decimal number representing the estimated hashsize of the global symbol table. A recommended hashsize is a number 1/3 larger than the total number of global symbols in the load. The default size (initially 127) is an assembly parameter.

LINK-10

Switches

Category of Switch
---

Delayed Action Switch (refer to Paragraph 3.3.5)

Examples
---

/HAS:1000

LINK-10 uses the prime number 1021.

LINK-10

Switches

/INCLUDE

Function
---

The /INCLUDE switch is used, when loading a file in search mode, to force the loading of specified modules in that file whether or not the user's program references them. For example, if the user does not have a global request for a desired module, he can use this switch to cause that module to be loaded.

Although the /INCLUDE switch is implemented in Version 1, its primary use is for the overlay facility in order to call a module.

Switch Format
---

/INCLUDE:symbol

/INCLUDE:(symbol, . . ., symbol)

Symbol is the module name of the desired module.

Category of Switch
---

File Dependent Switch (refer to Paragraph 3.3.2)

Examples
---

,SYS:LIB40/INCLUDE:(SIN,COS,TAN),

Search the library LIB40, but always load the modules SIN, COS, and TAN.

LINK-10

Switches

/LOCALS

Function

The /LOCALS switch is used to load local symbols with the specified programs. Local symbols are not processed by LINK-10, but are useful to the user when debugging.

This switch does not cause local symbols to be saved as part of the core image requested by the /SAVE or /SSAVE switch. The /SYMSEG switch or an entry in the JOBDAT location .JBDDT is required if local symbols are to remain in core.

Switch Format

/LOCALS

Category of Switch

File Dependent Switch (refer to Paragraph 3.3.2)

Examples

```
,MYFILE,/LOCAL MYDATA,MYSUB,MYLIB,
```

Load local symbols with the programs MYDATA, MYSUB, and MYLIB.

LINK-10

Switches

/LOG

Function

The /LOG switch is used to specify an output log file into which LINK-10 places information that is useful for the user when he is debugging his program. This file is a report of LINK-10's progress in loading the user's program because the actions taken by LINK-10 are shown. The times at which these actions took place are also indicated.

This switch is not required to obtain a log file if the user assigns a device the logical name LOG before running LINK-10. Then all log information will be recorded in a file on this assigned device. The file is named nnnLNK.LOG where nnn is the user's job number. In this case, the /LOG switch merely causes the file to be renamed to the user's specifications.

If the user does not assign a device the logical name LOG prior to running LINK-10, he must use the /LOG switch in order to obtain a log file. However, any times and messages output before the /LOG switch is seen in the command string will not appear in the log file.

Switch Format

file specification/LOG

File specification is in the form dev:file.ext[directory] to specify the device and name associated with the log file. The

LINK-10

Switches

default file specification is DSK:name of main program.LOG [user's default directory]. The user's terminal may be specified as the log device.

Category of Switch

Output Switch (refer to Paragraph 3.3.3)

Examples

DSKB:MYLOG/LOG

Create a log file on DSKB: with the name MYLOG.

LINK-10

Switches

/LOGLEVEL

Function

The /LOGLEVEL switch is used to suppress LINK-10 messages to the user's log file. This switch permits the user to set the level of messages that are to appear in the log file. Refer to the /ERRORLEVEL switch and Chapter 5.

If the log file is output to the user's terminal (i.e., the log device is the user's terminal), the messages output are determined by the lower of the arguments specified in the /ERRORLEVEL and /LOGLEVEL switches. The user would rarely set the log device as the terminal because the /ERRORLEVEL switch with a low number allows him to obtain all messages on the terminal.

Switch Format

/LOGLEVEL:n

n is a decimal number from 0 to 30. Messages with a message level less than or equal to n will not be output to the log file. The user cannot suppress messages with a level of 31. If this switch, or the value of the switch is omitted, a message level of 0 is assumed (i.e., all messages are output to the log file).

LINK-10

Switches

Category of Switch
---

Delayed Action Switch (refer to Paragraph 3.3.5)

Examples
---

/LOGLEVEL:5

Do not output any message to the log file with a message level less than or equal to 5.

LINK-10

Switches

/MAP

Function
---

The /MAP switch is used to specify an output map file which consists of the types of symbols requested by the user with the /CONTENTS switch. The map file is useful to the user when he is debugging his program because it lists the symbols used by his program along with their values. Header information (e.g., relocation counters with their lengths and starting addresses) is also included in the map.

Switch Format
---

file specification/MAP:keyword

File specification is in the form dev:file.ext [directory] and specifies the device and name associated with the map file. The default specification is DSK:name of main program.MAP[user's default directory].

Keyword is one of the following:

END to produce a map file at the end of loading.

ERROR to produce a map file of the code loaded if a fatal error occurs (i.e.,an error from which LINK-10 cannot recover).

NOW to produce a map file at the time this keyword is seen. The map contains all of the information up to and including the last file loaded. Default libraries will not be searched unless specified. This keyword is normally used during debugging to determine how the load is progressing.

If the /MAP switch is not issued by the user, no map file will be

generated. If the switch is given, but the keyword is omitted, the keyword END is assumed.

Category of Switch
---

Output Switch. Also, /MAP:NOW is an immediate action switch.

Examples
---

DSKB:MYMAP/MAP

Specify a map file on DSKB: with the name MYMAP.

LINK-10

Switches

/MAXCOR

Function

The /MAXCOR switch is used to specify the maximum amount of core LINK-10 may use as its low segment while loading. LINK-10 will expand to this size if required and then will overflow to the disk, rather than expanding in core, when it reaches the maximum core size allowed. When LINK-10 must overflow to the disk, it writes out part or all of the symbol area, the low code area, and/or the high core area in order that loading can continue. If the current amount of core used is greater than the size specified by the user, the next time LINK-10 requests more core, the size will decrease to the amount specified by the user and the remaining code will overflow to the disk. If the amount specified by the user is less than the minimum amount required by LINK-10, he receives a warning message indicating the amount required. He should then respecify the switch with a larger argument.

Switch Format

/MAXCOR:n

n is a decimal number that represents the maximum low segment core size for LINK-10. An octal value can be given by preceeding it with a number sign (#). N is expressed in units of 1024 words or 512 words (a page) by following the number with K or P respectively. If K or P is omitted, K (1024 words) is assumed.

LINK-10

Switches

The default size is all of available user core. The minimum size is dependent upon the code already loaded.

Category of Switch
---

Delayed Action Switch (refer to Paragraph 3.3.5)

Examples
---

/MAXCOR:30K

Allow LINK-10 to expand its low segment to 30K before overflowing to the disk.

LINK-10

Switches

/MPSORT

Function
---

The /MPSORT switch is used to arrange the symbol table for output to the map file in the order most convenient to the user.

Switch Format
---

/MPSORT:keyword

Keyword is one of the following:

UNSORTED to print the symbols in the order in which they are placed in the symbol table. This keyword is the default.

ALPHABETICAL to arrange the symbol table in alphabetical order for each module or for each block in a block-structured module.

NUMERICAL to arrange the symbol table in numerical order according to the values of the symbols for each module.

NOTE
For the first release of LINK-10, UNSORTED is the only keyword implemented. The other keywords listed above are ignored and a warning message is output.

Category of Switch
---

Delayed Action Switch (refer to Paragraph 3.3.5)

Examples
---

MYMAP/MAP/MPSORT:UNSORTED

Specify a map file with the name MYMAP and print the symbols in the order in which they appear in the symbol table.

LINK-10

Switches

/MTAPE

Function
---

The /MTAPE switch allows the user to perform magnetic tape functions such as rewind, backspace, and skip. If this switch is given in an input specification, the action is performed immediately. However, when the switch is part of an output specification, the action requested is not performed until the output device has been initialized.

Switch Format
---

/MTAPE:keyword

Keyword is one of the following:

| | |
|---|---|
| MTWAT | to wait for spacing and I/O to finish, |
| MTREW | to rewind the tape to load point. |
| MTEOF | to write an EOF, |
| MTSKR | to skip one record. |
| MTBSR | to backspace one record. |
| MTEOT | to space to the logical end-of-tape. |
| MTUNL | to rewind and unload the tape. |
| MTBLK | to write 3 inches of blank tape. |
| MTSKF | to skip one file. |
| MTBSF | to backspace one file. |
| MTDEC | to initialize for Digital-compatible 9-channel tape. |

LINK-10

Switches

MTIND to initialize for industry-compatible 9-channel tape.

Category of Switch

Device Switch (refer to Paragraph 3.3.1)

Examples

MTAØ:/MTAPE:MTEOT/MAP

Output the map file to MTAØ: after spacing to the logical end of tape (i.e., to the first free block).

LINK-10
Switches

/NOINITIAL

Function

The /NOINITIAL switch is used to clear LINK-10's initial global symbol table. This initial global symbol table consists of the .JBxxx symbols in JOBDAT. (Refer to DECsystem-10 Monitor Calls for a description of JOBDAT.) This switch is normally employed when the user is loading LINK-10 itself (in order to get the latest copy of JOBDAT), when the user wants to load a private copy of JOBDAT in order to use new values, or when the user is loading a program (for the purpose of creating a core image file) that will eventually run as an exec mode program (e.g., the monitor, diagnostics, a bootstrap loader). This switch must appear before the first file specification in the command string or else the initial LINK-10 global symbol table (JOBDAT) will be loaded. If the /NOINITIAL switch is specified, JOBDAT will be searched when the default system libraries are searched.

Switch Format

/NOINITIAL

If this switch is omitted, LINK-10's internal JOBDAT area symbols are used as the initial global symbol table.

LINK-10

Switches

Category of Switch

Immediate Action Switch (refer to Paragraph 3.3.4)

Examples

/NOINITIAL,COMMON,COMDEV,COMMOD,TOPS10/SEARCH/GO

Load the monitor without LINK-10's initial global symbol table.

/NOINITIAL,DTBOOT,EDDT/GO

Load the exec mode program without LINK-10's initial global symbol table.

LINK-10

Switches

/NOLOCAL

Function
---

The /NOLOCAL switch is used to load the programs without their local symbols. This is the default action.

Switch Format
---

/NOLOCAL

Category of Switch
---

File Dependent Switch (refer to Paragraph 3.3.2)

Examples
---

/LOCAL FIRST,SECOND,THIRD,FOURTH/NOLOCAL

Load the programs FIRST, SECOND, and THIRD with their local symbols and load the program FOURTH without its local symbols.

LINK-10
Switches

/NOSEARCH

Function

The /NOSEARCH switch is used to turn off library search mode (i.e., to always load the entire indicated file or files whether or not the files are required). The files are not searched to determine if they are needed. This switch is normally used after a /SEARCH switch has set library search mode. This is the default action.

Switch Format

/NOSEARCH

Category of Switch

File Dependent Switch (refer to Paragraph 3.3.2)

Examples

```
PARTA,/SEARCH LIBMAC,LIBCBL,LIBFOR,/NOSEARCH PARTB,PARTC
```

The files LIBMAC, LIBCBL, and LIBFOR are searched as libraries. The files PARTA, PARTB, and PARTC are loaded in their entirety.

LINK-10

Switches

/NOSTART

Function

The /NOSTART switch indicates to LINK-10 to ignore all start addresses in the binary input programs. The start address for the current program is not changed.

Switch Format

/NOSTART

If this switch is omitted and more than one start address is encountered, the last one seen is used.

Category of Switch

File Dependent Switch (refer to Paragraph 3.3.2)

Examples

MAIN1,/NOSTART MAIN2,MAIN3

Start addresses are ignored in files MAIN2 and MAIN3.

LINK-10

Switches

/NOSYMBOL

Function

The /NOSYMBOL switch signals LINK-10 not to construct a table of the symbols used by the user's program. This switch affects the speed of loading in that LINK-10 is not required to spend time in generating a symbol table for the user. If this switch is given, the user is not able to obtain output symbol files or output map files containing symbol listings. A map file can be obtained,however, with header information only.

Switch Format

/NOSYMBOL

Category of Switch

Immediate Action Switch (refer to Paragraph 3.3.4)

Examples

/NOSYM

LINK-10

Switches

/NOSYSLIB

Function
---

The /NOSYSLIB switch is used to inhibit the searching of one or more of the system libraries upon completion of the loading process. The system libraries required by the loaded modules are usually searched at the end of the load in order to satisfy undefined global requests. These libraries are LIBOL for COBOL modules, FORLIB for FORTRAN-10 modules, LIB40 for F40 modules, and ALGLIB for ALGOL modules.

Switch Format
---

/NOSYSLIB:keyword

/NOSYSLIB:(keyword, . . .,keyword)

Keyword is one or more of the following:

| | |
|---|---|
| ALGOL | to suppress the searching of ALGLIB. |
| BCPL | to suppress the searching of BCPLIB (not supported by DEC). |
| COBOL | to suppress the searching of LIBOL. |
| FORTRAN | to suppress the searching of FORLIB. |
| F40 | to suppress the searching of LIB40. |
| NELIAC | to suppress the searching of LIBNEL (not supported by DEC). |

If the keyword is omitted, the searching of all system libraries is suppressed.

LINK-10

Switches

Category of Switch

Delayed Action Switch (refer to Paragraph 3.3.5)

Examples

/NOSYSLIB:ALGOL/NOSYSLIB:COBOL

Do not search ALGLIB and LIBOL.

/NOSYSLIB

Do not search any system libraries.

LINK-10

Switches

/OTS

Function

The /OTS switch is used to indicate the segment into which the appropriate object time system is to be loaded.

Switch Format

/OTS:keyword

Keyword is one of the following:

DEFAULT to load the object time system into the segment specified by its code. FORTRAN, NELIAC, and ALGOL specify the high segment. This keyword is used to reset to normal conditions after specifying a /OTS switch with either the HIGH or LOW keywords.

LOW to load the object time system into the low segment.

HIGH to load the object time system into the high segment.

If this switch, or the value of this switch, is omitted, the default action is to load the object time system into the high segment unless either:

Code already exists in the high segment and /SEGMENT:HIGH is not set, or

The user has specified the /SEGMENT:LOW switch.

In these two cases, the object time system is loaded into the low segment.

LINK-10

Switches

Category of Switch
---

Delayed Action Switch (refer to Paragraph 3.3.5)

Examples
---

FILA.REL/SYSLIB/OTS:HIGH

Load the required object time system into the high segment.

LINK-10

Switches

/PATCHSIZE

Function

The /PATCHSIZE switch is used to allocate space between the top of the loaded code and the bottom of the symbol table. This space is then used for new symbols defined by the user with DDT and/or for patching. Note that when the user defines symbols with DDT, each symbol will occupy two words. The space is allocated in either the high or low segment, depending upon the placement of the symbol table as specified with the /SYMSEG switch. The default is to place the symbol table in the low segment.

Switch Format

/PATCHSIZE:n

n is a decimal number representing the number of words to be allocated as patching space. An octal value can be given by preceding it with a number sign (#). A global symbol, PAT.., is defined to be equal to the first location in the patching system.

If this switch, or the value of this switch, is omitted, the default allocation is 64 (decimal) or 100 (octal) words.

LINK-10

Switches

Category of Switch

Delayed Action Switch (refer to Paragraph 3.3.5)

Examples

/SYMSEG:HIGH/PATCHSIZE:200

Load the symbol table into the high segment and allocate 200 words between the loaded code and the symbol table.

LINK-10

Switches

/REQUIRE

Function

The /REQUIRE switch is used to generate global requests for the indicated symbols. Thus, this switch can be used to load library modules out of their normal loading sequence or to force the loading of modules for overlays.

The /REQUIRE switch is used to load a module by specifying one or more of its library search symbols (entry points), whereas the /INCLUDE switch is used to load a module by specifying its name. Thus, the /REQUIRE switch is useful when the user knows the function he wants loaded (e.g., SQRT), but does not know the name of the module containing that function.

Switch Format

/REQUIRE:symbol

/REQUIRE:(symbol, . . .,symbol)

Symbol is the SIXBIT symbol name for which the user wants a global request generated.

Category of Switch

Immediate Action Switch (refer to Paragraph 3.3.4)

Examples

/REQUIRE:NAME

Generate a global request for the symbol called NAME.

LINK-10

Switches

/REWIND

Function

The /REWIND switch is used to rewind the current input or output device. The device associated with this switch must be a DECtape or magnetic tape. If the device is not a tape device, the switch is ignored.

Switch Format

/REWIND

Category of Switch

Device Switch (refer to Paragraph 3.3.1)

Examples

,/REWIND MTA0:,

LINK-10

Switches

/RUNCOR

Function
---

The /RUNCOR switch is used to specify the amount of core to be assigned to the low segment of the program when it is executed. The effect of this switch is identical to that produced when the program is run by the system run commands (R or RUN) with the given core argument.

Switch Format
---

/RUNCOR:n

n is a decimal number that represents the amount of core to be used as the initial core size for the program when obtained with the GET system command. An octal value can be given by preceeding it with a number sign (#). N is expressed in units of 1024 words or 512 words (a page) by following the number with K or P respectively. If K or P is omitted, K (1024 words) is assumed. If n is omitted or is less than the amount required, the number of blocks required by the core image area is assumed.

Category of Switch
---

Delayed Action Switch (refer to Paragraph 3.3.5)

Examples
---

/RUNCOR:50P

LINK-10

Switches

/RUNAME

Function

The /RUNAME switch is used to assign the name to the program that is to be used while the program is running. This name is stored in a job-associated table in the Monitor and is used by the SYSTAT program and the VERSION system command. This switch affects high segment programs only.

Switch Format

/RUNAME:symbol

Symbol is the name to be assigned to the program. Only the first six characters specified are used. If this switch is omitted, the default name is the name of the module with the last start address. If there is no module containing a start address, the name used is nnnLNK, where nnn is the user's job number.

Category of Switch

Delayed Action Switch (refer to Paragraph 3.3.5)

Examples

/RUNAME:PRIV,MYPROG/SSAVE

Save the file with the name MYPROG (i.e., MYPROG.SHR), but the program is run with the name PRIV.

LINK-10

Switches

/SAVE

Function

The /SAVE switch is used to define an output save file which will contain the core image generated by LINK-10. The core image is saved as one or two files: a low segment file and/or a high segment file. After the core image is saved on the specified output device, it can later be brought into core and executed as a non-sharable program (by using the RUN or GET system commands) without rerunning LINK-10.

Before writing low segment files (i.e., files with extensions .SAV or .LOW), LINK-10 compresses the core image by eliminating all zero blocks. High segment files are not compressed. This action is known as zero-compression and is used to save space on the storage device. The resulting zero-compressed file is, in essence, identical to the one produced by the SAVE system command.

Switch Format

file specification/SAVE:n

File specification is in the form dev:file[directory] and specifies the device and name associated with the save file. The default specification is:

DSK:name of main program.[user's default directory]

LINK-10

Switches

User-supplied extensions are ignored and the extension given to the file depends on the number of segments saved. If there is only one segment, the extension .SAV is used. If there are two segments, the extension .LOW is used for the low segment and .HGH for the high segment.

N is a decimal number that represents the amount of core (sum of high and low segments) in which the program is later to be run. An octal value can be given by preceding it with a number sign (#). N is expressed in units of 1024 words or 512 words (a page) by following the number with K or P respectively. If K or P is omitted, K (1024 words) is assumed.

If the /SAVE is not used, a save file will not be generated. If the switch is given but the core argument is omitted, the minimum core required by the core image is used.

Category of Switch

Output Switch (refer to Paragraph 3.3.3)

Examples

```
DTA3:MYPROG/SAVE:4K=
```

Define a save file on DTA3: with the name MYPROG. The program will be run in 4K.

LINK-10

Switches

/SEARCH

Function
---

The /SEARCH switch is used to turn on library search mode (i.e., to search specified files in order to load only those modules of the file that are required to satisfy undefined global requests). The user gives this switch to search either library files that he may have created or ones that are not part of the required system libraries. The /NOSEARCH switch is used to turn off library search mode. The required system libraries are still searched unless the user has inhibited the searching with the /NOSYSLIB switch.

Switch Format
---

/SEARCH

Category of Switch
---

File Dependent Switch (refer to Paragraph 3.3.2)

Examples
---

```
PARTA,/SEARCH LIBMAC,LIBCBL,LIBFOR,/NOSEARCH PARTB,PARTC
```

The files LIBMAC, LIBCBL, and LIBFOR are searched as libraries. The files PARTA, PARTB, and PARTC are loaded in their entirety.

LINK-10

Switches

/SEGMENT

Function
---

The /SEGMENT switch is used to indicate to LINK-10 the segment into which to load the input modules.

Switch Format
---

/SEGMENT:keyword

Keyword is one of the following:

DEFAULT to follow the specifications in the program. The typical case is to load pure code into the high segment and impure code into the low segment. This keyword is used to reset to normal conditions after specifying a /SEGMENT switch with either the HIGH or LOW keywords.

LOW to load code into the low segment.

HIGH to load code into the high segment, even if the code is impure.

If this switch, or the value of the switch, is omitted, high segment code is loaded into the high segment and low segment code into the low segment.

Category of Switch
---

File Dependent Switch (refer to Paragraph 3.3.2)

Examples
---

/SEGMENT:LOW TESTPRG,ANSWER,ROUTIN/SEGMENT:HIGH,

Load the modules TESTPRG and ANSWER into the low segment and the module ROUTIN into the high segment.

LINK-10

Switches

/SET

Function

The /SET switch is used to set the value of a relocation counter to a specified number. Although LINK-10 will handle many relocation counters, in the first release only two relocation counters are implemented: the counter for the low segment (.LOW.) which begins at zero, and the counter for the high segment (.HIGH.) which begins at location 400000 or the end of the low segment, whichever is greater. Other counters can be set, but they are currently not used by LINK-10.

Switch Format

/SET:symbol:n

Symbol is the name of the relocation counter.

n is an octal number representing the value of the counter. For the first release of LINK-10, only two relocation counters can usefully be given, .LOW. and .HIGH.

Category of Switch

Immediate Action Switch (refer to Paragraph 3.3.4)

Examples

```
.SET:.LOW.:1000,/SET:.HIGH.:400000
```

LINK-10

Switches

/SEVERITY

Function

The /SEVERITY switch specifies to LINK-10 the level at which messages are to be considered fatal. Associated with each message is a decimal number from 0 to 31 called the severity level. With this switch, the user can specify that messages with a severity level less than or equal to a specific number are not to cause his job to be terminated. Any message with a severity level above the specified number will cause his job to abort.

Switch Format

/SEVERITY:n

n is a decimal number from 0 to 30. LINK-10 messages with a severity level above n will cause a user's job to be aborted. Even though the highest severity level is 31, the user cannot indicate that a message with this severity level is to be considered non-fatal. If this switch, or the value of the switch, is omitted, a fatal error for a timesharing job is one whose severity level is greater than 24 (decimal), and for a batch job, one whose level is greater than 16 (decimal).

Category of Switch

Delayed Action Switch (refer to Paragraph 3.3.5)

Examples

/SEVERITY:30

LINK-10

Switches

/SKIP

Function

The /SKIP switch is used to space forward over the specified number of input or output files. This switch is implemented for magnetic tape only and is ignored if it is given for any other device.

Switch Format

/SKIP:n

n is a decimal number representing the number of files to skip over.

Category of Switch

Device Switch (refer to Paragraph 3.3.1)

Examples

/SKIP:4 MTA3:

LINK-10

Switches

/SSAVE

Function

The /SSAVE switch is used to define an output save file which will contain the core image produced by LINK-10. It is similar to the /SAVE switch except that the high segment will be sharable when it is brought into core and executed. The saved file produced by this switch is the same as the one produced by the SSAVE system command. Refer to the /SAVE switch.

Switch Format

file specification/SSAVE:n

Arguments are the same as for the /SAVE switch except for the following difference: when there are two segments, the extension .LOW is assumed for the low segment and .SHR for the high segment.

Category of Switch

Output Switch (refer to Paragraph 3.3.3)

Examples

DTA:SHRPRG/SSAVE,

Define a sharable save file with the name SHRPRG on the user's DECtape. The minimum core required by the core image is assigned.

LINK-10

Switches

/START

Function

The /START switch is used to specify the start address of the loaded program or to allow a program to specify its own start address. When a start address is specified, all subsequent start addresses are ignored. This is the default action.

Switch Format

/START:n

n is either of the following:

an octal number preceded by a number sign (#) representing the starting address of the program, or

a SIXBIT global symbol whose value is the start address. The global symbol specified must be defined.

If n is omitted, LINK-10 does not change the current start address but will accept all start addresses from the following modules (i.e., the action is to turn off a /NOSTART switch setting).

Category of Switch

File Dependent Switch (refer to Paragraph 3.3.2)

Examples

,MAINPG/START,/NOSTART PROG1,PROG2,

Use the start address in MAINPG and ignore the start addresses in PROG1 and PROG2.

LINK-10

Switches

/SYMBOL

Function

The /SYMBOL switch is used to specify an output symbol file which will consist of local symbols (if loaded), information stored in the local symbol table, such as module names and lengths, and global symbols sorted for DDT.

Via keywords, the user can specify that the symbol file is to be either in radix-50 representation or in triplet format. These two symbol table formats can be distinguished from each other in several ways:

1. The first word of the radix-50 symbol table is always negative. The first word of the triplet symbol table is always zero.

2. The listing of each radix-50 symbol requires two words; the first word is the symbol name in radix-50 representation, and the second word is the value.

3. The listing of each triplet symbol requires three words; the first one contains flags, the second is the symbol name in SIXBIT, and the third is the value.

This switch is useful when DDT is not loaded with the user's program because it guarantees that the symbols will be available. Note that if the user issues the /NOSYMBOL switch in the command string, he is not able to obtain the output symbol file.

LINK-10

Switches

Switch Format

file specification/SYMBOL:keyword

File specification is in the form dev:file[directory] and specifies the device and name associated with the symbol file. The default specification is

DSK:name of main program .SYM[user's default directory]

If there is no main program, the filename nnnLNK, where nnn is the user's job number, is used.

Keyword is one of the following:

RADIX-50 to obtain the symbols in radix-50 representation.

TRIPLET to obtain the symbols in triplet format.

If the /SYMBOL switch is not issued by the user, no output symbol file will be generated. If the keyword is omitted, RADIX-50 is assumed.

Category of Switch

Output Switch (refer to Paragraph 3.3.3)

Examples

DSKB:SYMFIL[20,235]/SYMBOL,

Define a symbol file with the name SYMFIL on the [20,235] area of DSKB:. The symbols will be output in the RADIX-50 format.

LINK-10

Switches

/SYMSEG

Function

The /SYMSEG switch causes symbols to be loaded with the program and indicates the segment into which the symbol table is to be placed. With this switch, the user insures that his program when loaded with DDT will run in as much core as is available without overwriting the symbol table. Loading DDT or setting the JOBDAT location .JBDDT to a non-zero value also causes the symbols to be loaded.

Switch Format

/SYMSEG:keyword

Keyword is one of the following:

DEFAULT to move the symbol table from its current position at the top of core to the first free location after the patching space. The JOBDAT location .JBFF, which points to the first free location, is adjusted to point to the first free location after the symbol table. This keyword is used to reset to the normal action after invoking the /SYMSEG switch with either the HIGH or LOW keywords.

HIGH to place the symbol table into the high segment.

LOW to place the symbol table into the low segment.

If the switch, or the value of the switch, is omitted, the symbol table is moved from its current position in the segment to the first free location in that segment. The first free location is determined after the allocation of space (default allocation is 64 decimal or 100 octal words) for patching of symbols. A global symbol, PAT.., is defined to be equal to the first location in

the patching space.

Category of Switch
---

Delayed Action Switch (refer to Paragraph 3.3.5)

Examples
---

/SYMSEG:HIGH

LINK-10

Switches

/SYSLIB

Function

The /SYSLIB switch forces the system libraries to be searched in order to satisfy any undefined global requests. LINK-10 examines the main program first and, depending on the compiler used, searches the appropriate library (e.g., an ALGOL main program causes ALGLIB to be loaded). Then LINK-10 looks at any remaining programs and searches the relevant libraries.

A system library is not automatically searched unless its corresponding compiler-produced code has been loaded. This means that a user must explicitly request a system library when he is not loading the corresponding compiler-produced code for that library. For example, if the user is loading only MACRO-10 programs and he wants the LIB40 library searched, he must specify it in the switch format; LIB40 is not automatically searched unless F40 code has been loaded.

The normal action taken by LINK-10 is to search all required libraries at the end of the loading procedure; however, this switch without any keywords causes the libraries to be searched at the time the switch is given. If keywords are specified on the switch, the searcing of the indicated libraries occurs at the end of the loading procedure or on a subsequent /SYSLIB switch with no arguments, whichever occurs first.

LINK-10

Switches

Switch Format

/SYSLIB:keyword

/SYSLIB:(keyword, . . .,keyword)

Keyword is one of the following:

| | |
|---|---|
| ALGOL | to search ALGLIB |
| BCPL | to search BCPLIB (not supported by DEC) |
| COBOL | to search LIBOL |
| FORTRAN | to search FORLIB |
| F40 | to search LIB40 or FORLIB. The library searched depends upon the /FOROTS or /FORSE switch, if given, or on the default FORTRAN library, which is normally FORLIB, if neither switch is given. |
| NELIAC | to search LIBNEL (not supported by DEC) |

If the keyword is omitted, only the libraries for which corresponding compiler-produced code has been loaded will be searched.

Category of Switch

Creates an implicit file specification (refer to Paragraph 3.3.6)

Examples

/SYSLIB

LINK-10

Switches

/SYSORT

Function

The /SYSORT switch is used to arrange the symbol table for output to the symbol file into the order most convenient to the user.

Switch Format

/SYSORT:keyword

Keyword is one of the following:

UNSORTED to leave the symbols in the order in which they are placed in the symbol table. This is the default.

ALPHABETICAL to arrange the symbol table in alphabetical order for each module or for each block in a block-structured module.

NUMERICAL to arrange the symbol table in numerical order for each module according to the values of the symbols.

NOTE

For the first release of LINK-10, UNSORTED is the only keyword implemented. The other keywords described above are accepted but LINK-10's action is the same as that taken with the UNSORTED keyword.

Category of Switch

Delayed Action Switch (refer to Paragraph 3.3.5)

Examples

/SYSORT:UNSORTED

LINK-10

Switches

/TEST

Function

The /TEST switch is used to load a debugging program and to specify execution of the user's program. Thus, it is similar to the /DEBUG switch except that it specifies execution of the user's program instead of the debugging program. This switch does not cause termination of the loading; the /GO switch is required to terminate laoding.

Switch Format

/TEST:keyword

Keyword is one of the following: COBDDT, COBOL, DDT, FORTRAN, MACRO, MANTIS. When a compiler or the assembler is specified, the debugging aid associated with that translator is used (e.g., if MACRO is specified, the debugging program DDT is loaded).

Category of Switch

Creates an implicit file specification (refer to Paragraph 3.3.6)

Examples

,MAIN1,/TEST:COBOL DATPRG,DATA,TEST,

LINK-10

Switches

/UNDEFINED

Function

The /UNDEFINED switch is used to type all undefined global requests on the user's terminal. The user can employ this switch to determine the undefined symbols and then use the /DEFINE switch to satisfy the requests for these symbols. Thus, the user can interactively satisfy requests before LINK-10 terminates the load with undefined symbols.

Switch Format

/UNDEFINED

Category of Switch

Immediate Action Switch (refer to Paragraph 3.3.4)

Examples

```
*/UNDEF )
1 UNDEFINED SYMBOL
NAME     400100
```

400100 is a word in the chain of fixups depending on the symbol.

LINK-10

Switches

/UNLOAD

Function

The /UNLOAD switch is used to rewind and unload the current input or output device. The device associated with this switch must be a DECtape or a magnetic tape; the switch is ignored for non-tape devices.

Switch Format

/UNLOAD

Category of Switch

Device Switch; however, the action of this switch is always performed after the file is processed regardless of its position in the specification (refer to Paragraph 3.3.1)

Examples

```
,/REWIND DTA3:FILNAM/UNLOAD,
```

LINK-10

Switches

/VALUE

Function

The /VALUE switch allows the user to interactively type in the names of global symbols in order to find out their current values. The output given to the user consists of the requested symbol, its current value, and its status. The status can be one of: DEFINED (i.e., in the symbol table with its final value), UNKNOWN (i.e., not in the symbol table), UNDEFINED (i.e., in the symbol table as undefined), COMMON (i.e., in the symbol table and defined as COMMON).

Switch Format

/VALUE:symbol

/VALUE:(symbol, . . .,symbol)

Symbol is the name of the symbol in ASCII.

Category of Switch

Immediate Action Switch (refer to Paragraph 3.3.4)

Examples

```
*/VALUE:(TAG1,START)

TAG1     400010     DEFINED
START      0        UNDEFINED
```

The symbol TAG1 is defined to be the value 400010, and the symbol START is undefined.

LINK-10

Switches

/VERBOSITY

Function

The /VERBOSITY switch gives the user control over the amount of text transmitted to both his terminal and his log file whenever he receives a message from LINK-10. Associated with each message is a verbosity indicating the amount of text contained in the message. A verbosity of SHORT indicates that the message consists only of a 3-letter code (e.g., STC). A message with a verbosity of MEDIUM consists of the 3-letter code and one line that explains the code (e.g., STC Symbol Table Completed). A message with a verbosity of LONG consists of the 3-letter code, the one line of explanation, plus a more detailed explanation of the message. Thus, the user can specify via this switch the amount of explanation output to his terminal and log file.

LINK-10 has the following feature to aid users receiving fatal messages (i.e., ones preceded by ?). If the user receives a fatal message but has not indicated that he wants to see the detailed explanations (i.e., verbosity LONG), he can give the CONTINUE system command after he receives the message. LINK-10 then types out the remainder of the message (if there is more information available) on the user's terminal. This additional information is not included in the user's log file nor is the job continuable after the message is output.

LINK-10

Switches

Switch Format

/VERBOSITY:keyword

Keyword is one of the following:

| | |
|---|---|
| SHORT | 3-letter code only. |
| MEDIUM | 3-letter code and a one-line explanation. |
| LONG | 3-letter code, a one-line explanation, and a detailed explanation. |

The default value is MEDIUM if this switch, or the keyword to the switch, is omitted.

If the user specifies a verbosity greater than the one available for the message, the specified keyword is ignored for that message and only the available text is output. For example, if the user specifies MEDIUM as the verbosity but the message only has a 3-letter code available (i.e., SHORT), only the 3-letter code will be output because there is no additional information available for that message.

Category of Switch

Delayed Action Switch (refer to Paragraph 3.3.5)

Examples

/VER:SHORT

LINK-10

Switches

/XPN

Function

The /XPN switch is used to create or save on the disk the expanded core image file (XPN file) of the low segment. If the program has not been loaded onto the disk, this switch causes the file to be created with the name specified by the user. If the program has been loaded onto the disk, the file already exists, but with the name nnnLLC.TMP where nnn is the user's job number. Since this extension indicates a temporary file, the expanded file is normally deleted upon the completion of LINK-10's processing. Thus, in this case, the /XPN switch is used to rename the file with the .XPN extension, so that it will not be deleted.

Switch Format

file specification /XPN

File specification is in the form dev:file[directory] and specifies the device and name to be associated with the expanded core image file. The default specification is

DSK:name of main program.XPN[user's default directory]

If there is no main program, the filename nnnLNK, where nnn is the user's job number, is used.

**LINK-10**

**Switches**

**Category of Switch**

**Output Switch (refer to Paragraph 3.3.3)**

**Example**

**DSKC:XPNFIL[20,270]/XPN**

Save the expanded core image file on the [20,270] area of DSKC: and with the name XPNFIL.

LINK-10

Switches

/ZERO

Function

The /ZERO switch is used to clear the directory of the associated DECtape. The directory is always cleared before the file is written, regardless of the switch's position in the current specification. This switch is ignored for all non-DECtape devices.

Switch Format

file specification/ZERO

File specification is an output specification.

Category of Switch

Output Switch (refer to Paragraph 3.3.3)

Examples

```
DTA3:MYPROG/SAVE/ZERO
```

LINK-10

Messages

# CHAPTER 5

# LINK-10 MESSAGES

The following table of LINK-10 messages consists of four columns: CODE, LVL, SEV, and MESSAGE. The leftmost column (CODE) contains a 3-letter code, which represents a terse, abbreviated form of the message. The user can indicate, via the /VERBOSITY:SHORT switch, that he desires only this code to be output whenever he receives a LINK-10 message. Refer to the /VERBOSITY switch in Chapter 4 for additional information.

The second column of each message (LVL) indicates the message level associated with that message. The message level is the factor that determines if the message is to be output. Normally, informative messages are suppressed to the user's terminal and all messages are output to the log file, if the user has designated one. However, the user can override this action with the /ERRORLEVEL and /LOGLEVEL switches. These switches accept a decimal number and indicate to LINK-10 that messages with a message level less than or equal to the specified number are not to be output to the user's terminal (/ERRORLEVEL) or to his log file (/LOGLEVEL). Messages with a message level greater than the specified number will be output. The two switches are independent if the user's log file is not being output to his terminal. That is, he can have one set of messages printed on his terminal and another set listed in his log file. When the device for the log file is the user's terminal, only one set of messages is output. This set is the one generated by the lower argument in either

the /ERRORLEVEL or /LOGLEVEL switch.

There are currently representations for three message levels:

%I message level 1 (informative)

%W message level 10 (warning)

%F message level 31 (fatal)

Refer to the /ERRORLEVEL and /LOGLEVEL switches in Chapter 4 for additional information.

The third column (SEV) contains the severity level associated with each message. The severity level is the point at which LINK-10 considers a message to be fatal (i.e., one which will terminate the load). The predefined LINK-10 severity levels can be overridden by the user via the /SEVERITY switch. This switch accepts a decimal number and indicates to LINK-10 that messages with a severity level less than or equal to the specified number are not to be considered fatal. Messages with a severity level greater than the specified number will cause the load to be terminated. (Note that messages with a severity level of 31 are always fatal and that the user cannot override the action taken with these messages.) If the user does not give a /SEVERITY switch, or does not give an argument to the switch, a severity level of 24 is considered fatal for a timesharing job and a severity level of 16 is considered fatal for a batch job.

LINK-10

Messages

Currently the representations for the severity levels are as follows:

%I severity level 1. The message is enclosed in square brackets (informative).

%W severity level 10. The message is preceded by a percent sign (warning).

%E severity level 30. The message is preceded by a percent sign and followed by a line requesting the user to re-edit the current file specification, if he wishes. This option is available only to a time-sharing user (editing).

%F severity level 31. The message is preceded by a question mark (fatal).

Refer to the /SEVERITY switch in Chapter 4 for additional information.

The rightmost column (MESSAGE) contains a more detailed explanation of the message than the one appearing in the CODE column. This message, along with the three-letter code, is normally output. However, the user can override this action with the /VERBOSITY switch. Refer to the /VERBOSITY switch in Chapter 4 for further information.

LINK-10

Messages

| CODE | LVL | SEV | MESSAGE |
|---|---|---|---|
| ANC | %F | %F | ADDRESS NOT IN CORE (1)<br><br>LINK-10 expected a particular user address to be in core, but it is not there. This is a LINK-10 internal error. |
| AZW | %F | %F | ALLOCATING ZERO WORDS (1)<br><br>LINK-10's space allocator was called with a request for zero words. This is an internal error in LINK-10. |
| CEF | %F | %F | CORE EXPANSION FAILED (1)<br><br>All attempts to obtain more core, including writing files onto disk, have failed. |
| CLF | %I | %I | CLOSING LOG FILE, CONTINUING ON [file specification]<br><br>This message occurs when the user changes the device on which the log file is being written. The log file is closed on the first device and the remainder of the file is written on the second device. |
| CMF | %F | %F | COBOL MODULE MUST BE LOADED FIRST<br><br>The COBOL-produced file must be the first file loaded when loading COBOL modules. COBDDT, the COBOL debugging program, or any other modules, such as a MACRO routine, cannot be the first file in the command string. The user should begin loading again and place the COBOL main program or routine as the first file in the command string. |

---

(1) This message is not expected to occur. If it does, please notify your Software Specialist or send a Software Performance Report (SPR) to DEC.

LINK-10

Messages

| CODE | LVL | SEV | MESSAGE |
|---|---|---|---|
| CNW | %F | %F | CODE NOT YET WRITTEN AT [label] (1)<br><br>The user attempted a feature that is not yet implemented. This is an internal error in LINK-10. |
| CSF | %I | %I | CREATING SAV FILE<br><br>LINK-10 is generating the requested save file by running the core image through a zero compressor routine in order to produce a SAV format file. |
| DNS | %I | %I | DEVICE NOT SPECIFIED FOR /switch<br><br>A device switch, such as /REWIND or /BACKSPACE, has been given, but there is no device to be associated with it. The switch is ignored. This occurs when the user does not give a device name in the specification containing the switch or has not specified a device name in the current line. (Remember that devices are cleared at the end of the line.) LINK-10's default device DSK does not apply to device switches nor does a device specified in a /DEFAULT switch apply. The user should respecify the command line and include the appropriate device name with the switch. |

---

(1) This message is not expected to occur. If it does, please notify your Software Specialist or send a Software Performance Report (SPR) to DEC.

LINK-10

Messages

| CODE | LVL | SEV | MESSAGE |
|---|---|---|---|
| DRC | %W | %W | DECREASING RELOCATION COUNTER [symbol] FROM [value] TO [value]<br><br>The user is reducing the size of an already defined relocation counter via the /SET switch. The new value is accepted. The user should be extremely careful when he does this because code previously loaded under the old relocation counter may be overwritten. This practice of reducing counters is dangerous unless the user knows exactly where modules are loaded. |
| DSO | %F | %F | DATA STATEMENT OVERFLOW (1)<br><br>Incorrect code has been generated by the F40 compiler. |
| DUZ | %F | %F | DECREASING UNDEFINED SYMBOL COUNT BELOW ZERO (1)<br><br>On an internal check of the counter for undefined symbols, LINK-10 determined that the counter was negative. This is an internal error. |
| EID | %F | %F | ERROR ON INPUT DEVICE STATUS (xxxxxx) FOR [file specification]<br><br>A read error has occurred on the input device. Use of the device is terminated and the file is released. The status is represented by the right half of the file status word. Refer to DECsystem-10 Monitor Calls for the explanation of the file status bits. |

---

(1) This message is not expected to occur. If it does, please notify your Software Specialist or send a Software Performance Report (SPR) to DEC.

LINK-10

Messages

| CODE | LVL | SEV | MESSAGE |
|---|---|---|---|
| {ELC, EHC, ELS, EFX, EGS} | %F | %F | ERROR CREATING OVERFLOW FILE FOR AREA {LC, HC, LS, FX, GS} |
| | | | LINK-10 could not make the named file on the disk (LC=user's low segment code, HC=user's high segment code, LS=local symbol table, FX=fixup area, and GS=global symbol table). The user could be over quota, or the disk could be full or have errors. |
| EMS | %I | %I | END OF MAP SEGMENT |
| | | | Notification that the LINK-10 module LNKMAP has completed the writing of the map file. The map is now closed. |
| ESN | %F | %F | EXTENDED SYMBOL NOT EXPECTED (1) |
| | | | The code to handle symbols longer than six characters has not been completed. This code will be available in a future release. |
| EXP | %I | %I | EXPANDING LOW SEGMENT TO [n] K |
| | | | LINK-10 needs more core and is expanding to the specified amount. In future loads of the same programs, the user can run LINK-10 more efficiently by requesting this amount of core at the beginning of the load with the /CORE switch. |

---

(1) This message is not expected to occur. If it does, please notify your Software Specialist or send a Software Performance Report (SPR) to DEC.

LINK-10

Messages

| CODE | LVL | SEV | MESSAGE |
|---|---|---|---|
| EXS | %I | %I | EXIT SEGMENT |

LINK-10 is entering the completion stages of the loading process. These stages include the creation of save and symbol files and, if required, the execution of the core image.

| | | | |
|---|---|---|---|
| FCD | %F | %F | FORTRAN CONFUSED ABOUT DATA STATEMENTS (1) |

Incorrect code was generated by the F40 compiler for a data statement in the form

```
DATA A(I),I=1,4/1,2,3,4/
```

as opposed to a data statement in the form

```
DATA (A(I),I=1,4)/1,2,3,4/
```

| | | | |
|---|---|---|---|
| FCF | %I | %I | FINAL CODE FIXUPS |

LINK-10 is now reading the low and/or high segment overflow files backwards in order to do all remaining code fixups. This process may cause considerable disk overhead. Note that the message occurs only if the load was too large to fit entirely in core.

| | | | |
|---|---|---|---|
| FIA | %F | %F | CANNOT MIX KI10 AND KA10 FORTRAN-10 COMPILED CODE |

The FORTRAN-10 compiler generates different output for the KA10 and the KI10 processors (e.g., double precision code) and the user cannot load this mixture. He should decide which processor he wants to use and then recompile the appropriate programs.

---

(1) This message is not expected to occur. If it does, please notify your Software Specialist or send a Software Performance Report (SPR) to DEC.

LINK-10

Messages

| CODE | LVL | SEV | MESSAGE |
|---|---|---|---|
| FIN | %I | %I | LINK-10 FINISHED |

LINK-10 has completed its task of loading the user's program and other required programs. Control is either returned to the monitor or given to the user's program for execution.

| | | | |
|---|---|---|---|
| FON | %F | %F | CANNOT MIX F40 AND FORTRAN-10 COMPILED CODE |

Output from the F40 and FORTRAN-10 compilers cannot be used together in the same load. The user should decide which compiler he wants and then recompile the appropriate program with that compiler.

| | | | |
|---|---|---|---|
| (FEE / FRE) | %F | %F | (ENTER / RENAME) ERROR (0) ILLEGAL FILENAME FOR [file specification] |

One of the following conditions occurred:

1. The filename given was illegal.
2. When updating a file, the filename given did not match the file to be updated.
3. The RENAME UUO following a LOOKUP UUO failed.

| | | | |
|---|---|---|---|
| (FLE / GSE) | %F | %E | (LOOKUP / GETSEG) ERROR (0) FILE WAS NOT FOUND |

The file requested by the user was not found. The user should respecify the correct filename.

LINK-10

Messages

| CODE | LVL | SEV | MESSAGE |
|---|---|---|---|
| {FEE, FLE, FRE, GSE} | %F | %E | {ENTER, LOOKUP, RENAME, GETSEG} ERROR (1) NO DIRECTORY FOR PROJECT-PROGRAMMER NUMBER FOR [file specification] |

The UFD does not exist on the named file structure, or the project-programmer number given was incorrect.

| | | | |
|---|---|---|---|
| {FEE, FLE, FRE, GSE} | %F | %E | {ENTER, LOOKUP, RENAME, GETSEG} ERROR (2) PROTECTION FAILURE FOR [file specification] |

The user does not have the correct privileges to access the named file.

| | | | |
|---|---|---|---|
| FRE | %F | %E | ENTER ERROR (2) DIRECTORY FULL |

The directory on the DECtape has no room for the file.

| | | | |
|---|---|---|---|
| {FEE, FLE, FRE, GSE} | %F | %F | {ENTER, LOOKUP, RENAME, GETSEG} ERROR (3) FILE WAS BEING MODIFIED FOR [file specification] |

Another user is currently modifying the named file. The user should try accessing the file later.

| | | | |
|---|---|---|---|
| {FEE, FLE, FRE, GSE} | %F | %F | {ENTER, LOOKUP, RENAME, GETSEG} ERROR (4) RENAME FILENAME ALREADY EXISTS FOR [file specification] (1) |

The specified filename already exists, or a different filename was given on the ENTER UUO following a LOOKUP UUO.

---

(1) This message is not expected to occur. If it does, please notify your Software Specialist or send a Software Performance Report (SPR) to DEC.

LINK-10

Messages

| CODE | LVL | SEV | MESSAGE |
|---|---|---|---|
| FEE<br>FLE<br>FRE<br>GSE | %F | %F | ENTER<br>LOOKUP<br>RENAME<br>GETSEG ERROR (5) ILLEGAL SEQUENCE OF UUOS FOR [file specification] (1) |

The user specified an illegal sequence of monitor calls, UUOs, (e.g., a RENAME without a preceding LOOKUP or ENTER, or a LOOKUP after an ENTER).

| CODE | LVL | SEV | MESSAGE |
|---|---|---|---|
| FEE<br>FLE<br>FRE<br>GSE | %F | %F | ENTER<br>LOOKUP<br>RENAME<br>GETSEG ERROR (6) BAD UFD OR BAD RIB FOR [file specification] (1) |

One of the following conditions occurred:

1. Transmission, device, or data error occurred while attempting to read the UFD or RIB.

2. A hardware-detected device or data error was detected while reading the UFD RIB or UFD data block.

3. A software-detected data inconsistency error was detected while reading the UFD RIB or file RIB.

| CODE | LVL | SEV | MESSAGE |
|---|---|---|---|
| FEE<br>FLE<br>FRE<br>GSE | %F | %F | ENTER<br>LOOKUP<br>RENAME<br>GETSEG ERROR (7) NOT A SAV FILE FOR [file specification] (1) |

The named file is not a core image file. This message can never occur and is included only for completeness of the LOOKUP, ENTER, and RENAME error codes.

---

(1) This message is not expected to occur. If it does, please notify your Software Specialist or send a Software Performance Report (SPR) to DEC.

LINK-10

Messages

| CODE | LVL | SEV | MESSAGE |
|---|---|---|---|
| {FEE, FLE, FRE, GSE} | %F | %F | {ENTER, LOOKUP, RENAME, GETSEG} ERROR (10) NOT ENOUGH CORE FOR [file specification] (1) |

The system cannot supply enough core to use as buffers or to read in a program. This message can never occur and is included only for completeness of the LOOKUP, ENTER, and RENAME error codes.

| | | | |
|---|---|---|---|
| {FEE, FLE, FRE, GSE} | %F | %F | {ENTER, LOOKUP, RENAME, GETSEG} ERROR (11) DEVICE NOT AVAILABLE FOR [file specification] (1) |

The device indicated by the user is currently not available. This message can never occur and is included only for completeness of the LOOKUP, ENTER and RENAME error codes.

| | | | |
|---|---|---|---|
| {FEE, FLE, FRE, GSE} | %F | %F | {ENTER, LOOKUP, RENAME, GETSEG} ERROR (12) NO SUCH DEVICE FOR [file specification] (1) |

The device specified by the user does not exist. This message can never occur and is included only for completeness of the LOOKUP, ENTER, and RENAME error codes.

---

(1) This message is not expected to occur. If it does, please notify your Software Specialist or send a Software Performance Report (SPR) to DEC.

LINK-10

Messages

| CODE | LVL | SEV | MESSAGE |
|---|---|---|---|
| FEE<br>FLE<br>FRE<br>GSE | %F | %F | ENTER<br>LOOKUP<br>RENAME<br>GETSEG ERROR (13) NOT TWO RELOC REG CAPABILITY FOR [file specification] (1) |

The machine does not have a two-register relocation capability. This message can never occur and is included only for completeness of the LOOKUP, ENTER and RENAME error codes.

| CODE | LVL | SEV | MESSAGE |
|---|---|---|---|
| FEE<br>FLE<br>FRE<br>GSE | %F | %F | ENTER<br>LOOKUP<br>RENAME<br>GETSEG ERROR (14) NO ROOM OR QUOTA EXCEEDED FOR [file specification] |

There is no room on the file structure for the named file, or the user's quota on the file structure would be exceeded if the file were placed on the structure.

| CODE | LVL | SEV | MESSAGE |
|---|---|---|---|
| FEE<br>FLE<br>FRE<br>GSE | %F | %F | ENTER<br>LOOKUP<br>RENAME<br>GETSEG ERROR (15) WRITE LOCK ERROR FOR [file specification] |

The user cannot write on the specified device because it is write-locked.

| CODE | LVL | SEV | MESSAGE |
|---|---|---|---|
| FEE<br>FLE<br>FRE<br>GSE | %F | %F | ENTER<br>LOOKUP<br>RENAME<br>GETSEG ERROR (16) NOT ENOUGH MONITOR TABLE SPACE FOR [file specification] |

There is not enough table space in the monitor's (FILSER) 4-word blocks for the specified file. The user should try running the job at a later time.

---

(1) This message is not expected to occur. If it does, please notify your Software Specialist or send a Software Performance Report (SPR) to DEC.

LINK-10

Messages

| CODE | LVL | SEV | MESSAGE |
|---|---|---|---|
| {FEE, FLE, FRE, GSE} | %W | %W | {ENTER, LOOKUP, RENAME, GETSEG} ERROR (17) PARTIAL ALLOCATION ONLY FOR [file specification] |

Because of the user's quota or the available space on the device, the total number of blocks requested could not be allocated and a partial allocation was given.

| | | | |
|---|---|---|---|
| {FEE, FLE, FRE, GSE} | %F | %F | {ENTER, LOOKUP, RENAME, GETSEG} ERROR (20) BLOCK NOT FREE ON ALLOCATION FOR [file specification] (1) |

The block required by LINK-10 is not available for allocation. This message can never occur and is included only for completeness of the LOOKUP, ENTER, and RENAME error codes.

| | | | |
|---|---|---|---|
| {FEE, FLE, FRE, GSE} | %F | %F | {ENTER, LOOKUP, RENAME, GETSEG} ERROR (21) CAN'T SUPERSEDE (ENTER) AN EXISTING DIRECTORY FOR [file specification] (1) |

The user attempted to supersede an existing directory. This message can never occur and is included only for completeness of the LOOKUP, ENTER, and RENAME error codes.

---

(1) This message is not expected to occur. If it does, please notify your Software Specialist or send a Software Performance Report (SPR) to DEC.

LINK-10

Messages

| CODE | LVL | SEV | MESSAGE |
|---|---|---|---|
| {FEE, FLE, FRE, GSE} | %F | %F | {ENTER, LOOKUP, RENAME, GETSEG} ERROR (22) CAN'T DELETE (RENAME) A NON-EMPTY DIRECTORY FOR [file specification] (1) |

The user attempted to delete a directory that was not empty. This message can never occur and is included only for completeness of the LOOKUP, ENTER, and RENAME error codes.

| CODE | LVL | SEV | MESSAGE |
|---|---|---|---|
| {FEE, FLE, FRE, GSE} | %F | %F | {ENTER, LOOKUP, RENAME, GETSEG} ERROR (23) SFD NOT FOUND FOR [file specification] |

The required sub-file directory in the specified path was not found.

| CODE | LVL | SEV | MESSAGE |
|---|---|---|---|
| {FEE, FLE, FRE, GSE} | %F | %F | {ENTER, LOOKUP, RENAME, GETSEG} ERROR (24) SEARCH LIST EMPTY FOR [file specification] |

A LOOKUP and ENTER UUO was performed on generic device DSK and the search list is empty.

| CODE | LVL | SEV | MESSAGE |
|---|---|---|---|
| {FEE, FLE, FRE, GSE} | %F | %F | {ENTER, LOOKUP, RENAME, GETSEG} ERROR (25) SFD NEST LEVEL TOO DEEP FOR [file specification] (1) |

An attempt was made to create a subfile directory nested deeper than the maximum level allowed. This message can never occur and is included only for completeness of the LOOKUP, ENTER, and RENAME error codes.

---

(1) This message is not expected to occur. If it does, please notify your Software Specialist or send a Software Performance Report (SPR) to DEC.

LINK-10

Messages

| CODE | LVL | SEV | MESSAGE |
|---|---|---|---|
| FEE<br>FLE<br>FRE<br>GSE | %F | %F | ENTER<br>LOOKUP<br>RENAME<br>GETSEG ERROR (26) NO-CREATE ON FOR ALL SEARCH LIST FOR [file specification] |

No file structure in the job's search list has both the no-create bit and the write-lock bit equal to zero and has the UFD or SFD specified by the default or explicit path.

| CODE | LVL | SEV | MESSAGE |
|---|---|---|---|
| FEE<br>FLE<br>FRE<br>GSE | %F | %F | ENTER<br>LOOKUP<br>RENAME<br>GETSEG ERROR (27) SEGMENT NOT ON SWAP SPACE FOR [file specification] (1) |

A GETSEG UUO was issued from a locked low segment to a high segment which is not a dormant, active, or idle segment. This message can never occur and is included only for completeness of the LOOKUP, ENTER, and RENAME error codes.

| CODE | LVL | SEV | MESSAGE |
|---|---|---|---|
| FEE<br>FLE<br>FRE<br>GSE | %F | %F | ENTER<br>LOOKUP<br>RENAME<br>GETSEG ERROR (nn) UNKNOWN CAUSE FOR [file specification] (1) |

This message indicates that a LOOKUP, ENTER, or RENAME error occurred which was larger in number than the errors LINK-10 knows about.

---

(1) This message is not expected to occur. If it does, please notify your Software Specialist or send a Software Performance Report (SPR) to DEC.

LINK-10

Messages

| CODE | LVL | SEV | MESSAGE |
|---|---|---|---|
| HSL | %F | %F | ATTEMPT TO SET HIGH SEGMENT ORIGIN TOO LOW |
| | | | The user is trying to set the beginning of the high segment below 400,000 or below the end of the low segment, whichever is larger. This can be the result of a /SET:.HIGH. switch with a value less than 400,000. If this is the case, the switch is ignored and the user should again specify the /SET:.HIGH. switch with a valid argument. This message can also occur when the low segment is greater than 400,000 and a module being loaded is requesting the high segment to start at 400,000. The user can either give a /SET switch or retranslate the module. |
| HSO | %W | %W | ATTEMPT TO CHANGE HIGH SEGMENT ORIGIN FROM [value] TO [value] |
| | | | The user is attempting to change the starting address of the high segment. The specified value is ignored. The cause may be that the user gave a /SET:.HIGH.: value switch which does not agree with the LINK item type 3, or that two LINK item type 3's have different origins. The user should recompile the incorrect files. |
| HTL | %F | %F | SYMBOL HASH TABLE TOO LARGE (1) |
| | | | The user has more global symbols than can fit in the maximum hash table (about 25K in size) LINK-10 can generate. Possible action is to increase the maximum allowable size of the hash table. |

---

(1) This message is not expected to occur. If it does, please notify your Software Specialist or send a Software Performance Report (SPR) to DEC.

LINK-10

Messages

| CODE | LVL | SEV | MESSAGE |
|---|---|---|---|
| {I4D / I4S / I4T} | %F | %F | ILLEGAL F40 {DATA CODE / SUB-BLOCK / TABLE ENTRY} (xxxxxx) (1) |
| | | | Incorrect code was produced by the F40 compiler. |
| IBC | %F | %F | ATTEMPT TO INCREASE SIZE OF BLANK COMMON |
| | | | An attempt was made to expand the blank COMMON area. Once a COMMON area is defined, the size cannot be expanded. The user should load the module with the largest blank COMMON area first or specify the larger area with the /COMMON switch before loading either module. |
| ICI | %F | %F | INSUFFICIENT CORE TO INITIALIZE LINK-10 |
| | | | There is not enough core in the system to initialize LINK-10. |
| IDM | %F | %E | ILLEGAL DATA MODE FOR DEVICE |
| | | | The data mode specified for a device is illegal, such as dump mode for the terminal (e.g., TTY:/SAVE). The user should respecify the correct device. |
| IFD | %F | %F | INIT FAILURE FOR DEVICE [dev] |
| | | | The OPEN or INIT UUO failed for the specified device. The device could be in use by another user. |

---

(1) This message is not expected to occur. If it does, please notify your Software Specialist or send a Software Performance Report (SPR) to DEC.

LINK-10

Messages

| CODE | LVL | SEV | MESSAGE |
|---|---|---|---|
| {ILC, IHC, ILS, IFX, IGS} | %F | %F | ERROR INPUTTING AREA {LC, HC, LS, FX, GS} - STATUS (xxxxxx)<br><br>An error occurred while reading in the named area (LC=user's low segment code, HC=user's high segment code, LS=local symbol table, FX=fixup area, and GS=global symbol table). The status is represented by the right half of the file status word. Refer to DECsystem-10 Monitor Calls for the explanation of the file status bits. |
| ILI | %F | %F | ILLEGAL LINK ITEM TYPE (xxxxxx) ON [file specification]<br><br>The input file either was generated by a translator that LINK-10 does not recognize (e.g., a non-supported translator) or is not in proper binary format (e.g., is an ASCII or SAV file). |
| IMA | %I | %I | INCREMENTAL MAPS NOT YET AVAILABLE<br><br>The INCREMENTAL keyword for the /MAP switch is not implemented. The switch is ignored. |
| INS | %F | %F | I/O DATA BLOCK NOT SET UP (1)<br><br>LINK-10 attempted to do I/O (LOOKUP, ENTER UUOs) for a channel that has not been set up. This is an internal LINK-10 error. |

---

(1) This message is not expected to occur. If it does, please notify your Software Specialist or send a Software Performance Report (SPR) to DEC.

LINK-10

Messages

| CODE | LVL | SEV | MESSAGE |
|---|---|---|---|
| IPO | %F | %F | INVALID POLISH OPERATOR (1)<br><br>An incorrect link item type 11 was seen. This is an internal LINK-10 error. |
| ISD | %F | %F | INCONSISTENT SYMBOL DEFINITION FOR [symbol]<br><br>An already-defined symbol has been given a second "partial" definition. The user should examine the usage of the named symbol. |
| ISO | %F | %F | INCORRECT STORE OPERATOR (1)<br><br>An incorrect link item type 11 was seen. This is an internal LINK-10 error. |
| ISP | %F | %F | INCORRECT SYMBOL POINTER (1)<br><br>The current symbol pointer does not point to a valid symbol triplet. This can occur if an extended symbol does not terminate properly. This is an internal LINK-10 error. |
| IST | %F | %F | INCONSISTENCY IN SWITCH TABLE (1)<br><br>An internal error occurred in the switch tables built by the SCAN module. |
| IVC | %F | %F | INDEX VALIDATION CHECK FAILED AT [address] (1)<br><br>The range checking of LINK-10's internal tables and arrays failed. The address output is the point in the appropriate LINK-10 segment at which this occurred. |

(1) This message is not expected to occur. If it does, please notify your Software Specialist or send a Software Performance Report (SPR) to DEC.

LINK-10

Message

| CODE | LVL | SEV | MESSAGE |
|---|---|---|---|
| LDS | %I | %I | LOAD SEGMENT<br><br>Indication that the LINK-10 module LNKLOD has started its processing. |
| LIM | %I | %I | LINK-10 INITIALIZATION<br><br>LINK-10 has begun its processing of the user's input. |
| LIT | %F | %F | LINK ITEM TYPE (xxxxxx) TOO SHORT FOR [file specification]<br><br>An error occurred in the named link item. This could result from incorrect output generated by a translator (e.g., no argument is seen on an END block when one is required). The user should retranslate the module. |
| LMN | %I | %I | LOADING MODULE [name]<br><br>LINK-10 is in the process of loading the named module. |
| MDS | %W | %W | MULTIPLY-DEFINED GLOBAL SYMBOL [symbol] IN MODULE [name] DEFINED VALUE = [value], THIS VALUE = [value]<br><br>The user has given an existing global symbol a value different from its original one. The second occurrence of the global symbol is in the named module. The currently defined value is used. The user should change the name of the symbol or reassemble one of the files with the correct parameters. |
| MNS | %I | %I | MAP SORTING NOT YET IMPLEMENTED<br><br>Alphabetic and numeric sorting of the map file is not yet implemented. The symbols appear in the order in which they were placed in the symbol table. |

LINK-10

Messages

| CODE | LVL | SEV | MESSAGE |
|---|---|---|---|
| MOV | %I | %I | MOVING LOW SEGMENT TO EXPAND AREA[area] |

This message indicates that LINK-10 is making inefficient use of core. In future loads of the same programs, the user should allocate more core to LINK-10 at the beginning of the load. Area is one of the following: LC=user's low segment code, HC=user's high segment code, LS=local symbol table, FX=fixup area, and GS=global symbol table.

| | | | |
|---|---|---|---|
| MPS | %I | %I | MAP SEGMENT |

Indication that the LINK-10 module LNKMAP has begun to write a map file.

| | | | |
|---|---|---|---|
| MSS | %W | %W | MAXCOR SET TOO SMALL, INCREASING TO nK |

The current value of MAXCOR is too small for LINK-10 to operate. In future loads of this program, the user can save LINK-10 time by setting MAXCOR to this new expanded size at the beginning of the load.

| | | | |
|---|---|---|---|
| MTS | %W | %W | MAXCOR TOO SMALL, AT LEAST nK IS REQUIRED |

The user specified the /MAXCOR switch with an argument that is below the minimum size LINK-10 requires as its low segment. The switch is ignored. The minimum size is dependent upon the code already loaded. The user should respecify the switch.

| | | | |
|---|---|---|---|
| NCL | %W | %W | NOT ENOUGH CORE TO LOAD JOB, SAVED AS [file specification] |

The user's program was too large to load into core. Thus, LINK-10 created a saved file on disk and cleared user core. The user can perform a GET or RUN operation on the program to load it into core. If the core image is still too big, the user can either employ a bigger machine or obtain a larger core limit (e.g., increase CORMAX).

LINK-10

Messages

| CODE | LVL | SEV | MESSAGE |
|---|---|---|---|
| NCX | %W | %I | NOT ENOUGH CORE TO LOAD AND EXECUTE JOB, WILL RUN FROM [file specification]<br><br>The user's program was too large to load into core and LINK-10 created a saved file on disk. It automatically executes the program by performing a RUN UUO. However, the saved file remains on disk and the user must delete it, if he wishes. |
| NED | %F | %E | NON-EXISTENT DEVICE [dev]:<br><br>The user has specified a device that does not exist in the system. The user can re-edit the input files to correct the device name or type control-C to abort the load. |
| NYI | %W | %W | NOT YET IMPLEMENTED - /switch<br><br>The user issued a switch that is not implemented in this version of LINK-10. |
| {OLC, OHC, OLS, OFX, OGS} | %F | %F | ERROR OUTPUTTING AREA {LC, HC, LS, FX, GS} -STATUS (xxxxxx)<br><br>An error occurred while writing out the named area (LC=user's low segment code, HC=user's high segment code, LS=local symbol table , FX=fixup area, and GS=global symbol table). The status is represented by the right half of the file status word. Refer to DECsystem-10 Monitor Calls for the explanation of the file status bits. |

LINK-10

Messages

| CODE | LVL | SEV | MESSAGE |
|---|---|---|---|
| {OEL, OEM, OES, OEX} | %W | %W | OUTPUT ERROR ON {LOG, MAP, SYMBOL, XPN} FILE. FILE CLOSED. JOB CONTINUING - STATUS [xxxxxx] |

An error has occurred on the output file. The output file is closed at the end of the last data that was successfully output. The status is represented by the right half of the file status word. Refer to DECsystem-10 Monitor Calls for the explanation of the file status bits.

| | | | |
|---|---|---|---|
| OFN | %F | %F | OLD FORTRAN (F40) MODULE NOT AVAILABLE |

The standard released version of LINK-10 includes the LNKF40 module that loads F40 code. However, the installation has removed it by loading a dummy version of LNKF40 and thus LINK-10 is unable to handle F40 compiler output. The user should request his installation to load a version of LINK-10 with the real LNKF40 module.

| | | | |
|---|---|---|---|
| OMN | %F | %F | OBSOLETE MONITOR WILL NOT SUPPORT LINK-10 |

LINK-10 requires a monitor that contains the DEVSIZ UUO.

| | | | |
|---|---|---|---|
| {PLC, PHC, PLS, PFX, PGS} | %I | %I | AREA {LC, HC, LS, FX, GS} OVERFLOWING TO DSK |

The job is too large to fit into the allowed core and the named area is being moved to disk (LC=user low segment code, HC=user high segment code, LS=local symbol table, FX=fixup area, and GS=global symbol table).

LINK-10

Messages

| CODE | LVL | SEV | MESSAGE |
|---|---|---|---|
| PSF | %F | %F | POLISH SYMBOL FIXUPS NOT YET IMPLEMENTED<br><br>The requested feature is not yet available. |
| RCF | %F | %F | RELOCATION COUNTER TABLE FULL<br><br>The relocation counter table is a fixed length and cannot be expanded in the current version of LINK-10. This restriction will be eliminated in a future release. |
| RED | %I | %I | REDUCING LOW SEGMENT TO [n] K<br><br>LINK-10's internal tables have been deleted and core has been reclaimed. This message occurs near the end of loading. |
| RGS | %I | %I | REHASHING GLOBAL SYMBOL TABLE FROM [old size] TO [new size]<br><br>LINK-10 is expanding the global symbol table either to the next prime number as requested by the user (via /HASHSIZE) or to its next expansion of about 50%. In future loads of this program, the user can save LINK-10 time by setting the hash table to this new expanded size at the beginning of the load. |
| SIF | %F | %F | SYMBOL INSERT FAILURE, NON-ZERO HOLE FOUND (1)<br><br>An internal LINK-10 error. LINK-10's hashing algorithm failed. A symbol already exists in the location in which LINK-10 needs to place the new symbol. The error may disappear if the user loads the files in a different order. |

---

(1) This message is not expected to occur. If it does, please notify your Software Specialist or send a Software Performance Report (SPR) to DEC.

LINK-10

Messages

| CODE | LVL | SEV | MESSAGE |
|---|---|---|---|
| SFU | %I | %I | SYMBOL TABLE FOULED UP (1) |

An internal LINK-10 inconsistency. LINK-10 cannot locate the TITLE triplets in order to store the lengths of the control sections. The loading process continues. Any maps requested by the user will not contain the lengths of the control sections.

| | | | |
|---|---|---|---|
| SNC | %F | %F | SYMBOL [symbol] ALREADY DEFINED, BUT NOT AS COMMON |

The user has defined a non-COMMON symbol with the same name as a COMMON symbol. The user should decide which symbol definition he wants. If he uses the COMMON definition, the COMMON area should be loaded first.

| | | | |
|---|---|---|---|
| SNL | %I | %I | SCANNING NEW COMMAND LINE |

LINK-10 has completed the scanning and processing of the current command line and is ready to accept the input on the next line.

| | | | |
|---|---|---|---|
| SOE | %F | %F | SAVE FILE OUTPUT ERROR - STATUS (xxxxxx) |

An error has occurred on the save file. The file is closed at the end of the last data that was successfully output. The status is represented by the right half of the file status word. Refer to DECsystem-10 Monitor Calls for the explanation of the file status bits.

| | | | |
|---|---|---|---|
| SSN | %I | %I | SYMBOL TABLE SORTING NOT YET IMPLEMENTED |

Alphabetic and numeric sorting of the symbol table is not yet implemented. The symbols appear in the order in which they were placed in the symbol table.

LINK-10

Messages

| CODE | LVL | SEV | MESSAGE |
|---|---|---|---|
| SST | %I | %I | SORTING SYMBOL TABLE |

LINK-10 is arranging the symbol table in the order specified by the user via the /SYSORT switch, and if required, is converting the symbols from the new to old format as indicated on the /SYMSEG, /SYMBOL, or /DEBUG switch.

| | | | |
|---|---|---|---|
| STC | %I | %I | SYMBOL TABLE COMPLETED |

The symbol table has bee.. sorted and moved according to the user's request via the /SYMSEG, /SYMBOL, or /DEBUG switch.

| | | | |
|---|---|---|---|
| T13 | %F | %F | LVAR (TYPE 13) CODE NOT IMPLEMENTED |

LINK item type 13 (LVAR) is not implemented in LINK-10 nor supported by DEC. The TWOSEG pseudo-op in the MACRO-10 language should be used.

| | | | |
|---|---|---|---|
| TDS | %W | %W | TOO LATE TO DELETE INITIAL SYMBOLS |

The /NOINITIAL switch was placed in the command string after the first file specification. Because this switch was not first in the command string, LINK-10's initial symbol table was loaded.

| | | | |
|---|---|---|---|
| TEC | %F | %F | TRYING TO EXPAND COMMON |

An attempt was made to expand a COMMON area. The largest occurrence of the COMMON area of a given name must be linked first. Once defined, the size cannot be expanded although new COMMON areas of different names can be defined. The user should load the largest occurrence first.

LINK-10

Messages

| CODE | LVL | SEV | MESSAGE |
|---|---|---|---|
| TSO | %F | %F | CANNOT LOAD TWO SEGMENT MODULE INTO ONE SEGMENT<br><br>The user attempted to force two segments into one segment via the /SEGMENT switch. However, the binary file does not contain the necessary information (i.e., the length of the high segment) in LINK item type 3. This situation is usually caused by a one-pass compiler (e.g.,ALGOL). |
| URC | %I | %I | UNKNOWN RADIX-50 SYMBOL CODE<br><br>Bits 0-3 of the first word of the link item contain an unknown symbol code. Either the translator is generating incorrect code or the binary file is bad. The user should recompile the file. |
| USA | %W | %W | UNDEFINED STARTING ADDRESS<br><br>The user has given a global symbol as the start address and the symbol is currently undefined. The user should load the module that defines the symbol. |

LINK-10

Examples

## CHAPTER 6

## LINK-10 EXAMPLES

### EXAMPLE 1 Loading and Executing COBOL Programs

The following files are on the user's disk area:

```
.DIRECT )

FILA     CBL  1    <055>          6-FEB-73    DSKB:  [27,235]
FILB     CBL  2    <055>          6-FEB-73
FILC     CBL  1    <055>          6-FEB-73
006LNK   LOG  1    <055>         28-FEB-73
SIMPLE   MAC  1    <055>         28-FEB-73

  TOTAL OF 6 BLOCKS IN 5 FILES ON DSKB: [27,235]
```

In the command string shown below, the user is automatically compiling, loading, and executing the programs and generating a map. The /CONT:ZERO switch is passed to LINK.

```
.EXECUTE /LINK/MAP FILA,FILB,FILC%'CONT:ZERO' )
COBOL:    CBS08A    [FILA,CBL]
COBOL:    CBS08B    [FILB,CBL]
COBOL:    CBS08C    [FILC,CBL]
LINK:     LOADING
[EXECUTION]
RUNNING CBS08A
RUNNING CBS08B
RUNNING CBS08C

EXIT
```

In the followng command sequences the user is compiling the files and then directly loading and executing them through LINK-10.

LINK-10
Examples

```
.COM FILA,FILB,FILC )
COBOL]     CBS08A     [FILA,CBL]
COBOL]     CBS08B     [FILB,CBL]
COBOL]     CBS08C     [FILC,CBL]

EXIT

.R LINK )

*FILA,FILB,FILC,/MAP/CONT:ZERO/EXECUTE/GO )
[EXECUTION]
RUNNING CBS08A
RUNNING CBS08B
RUNNING CBS08C

EXIT
```

EXAMPLE 2 Loading and Executing a MACRO Program

The user assembles the following MACRO program:

```
.COMPILE SIMPLE.MAC )
MACRO] SIMPLE


EXIT
```

In the following command sequences, the user loads the MACRO program, interactively requests a listing of the relocation counters, library search symbols, and undefined global symbols, and then executes the program.

```
.R  LINK )

*SIMPLE )
*/COUNTER )
RELOCATION COUNTER        INITIAL VALUE     CURRENT VALUE (OCTAL)
.LOW.                          0                 140
.HIGH.                    400000            400025
*/ENTRY )
NO LIBRARY SEARCH SYMBOLS (ENTRY POINTS)
*/UNDEFINE )

NO UNDEFINED GLOBAL SYMBOLS
*/EXECUTE/GO )
```

LINK-10

Examples

```
[EXECUTION]
THIS IS A VERY SIMPLE TWO-SEGMENT MACRO PROGRAM.

EXIT

.
```

EXAMPLE 3 Loading COBOL Programs and Creating a Saved File

In the following example, the user is individually loading each file and requesting a listing of undefined global symbols after each file is loaded. He also is requesting the searching of the default system libraries. After searching has been performed, the user creates a saved file and executes the core image.

```
.R  LINK )
*FILA/U )

6 UNDEFINED GLOBAL SYMBOLS
BTRAC,     1212
TRACE,     1277
TRPD,      1214
TRPOP,     1213
CBS08B     1327
CBDDT,     1260
*FILB/U )

6 UNDEFINED GLOBAL SYMBOLS
BTRAC,     1367
TRACE,     1473
TRPD,      1371
CBS08C     1615
TRPOP,     1370
CBDDT,     1454
*FILC/U )

5 UNDEFINED GLOBAL SYMBOLS
BTRAC,     2052
TRACE,     2147
TRPD,      2054
TRPOP,     2053
CBDDT,     2130
*/SYSLIB/U )

NO UNDEFINED GLOBAL SYMBOLS
```

LINK-10

Examples

```
*FILE/SAV/EXECUTE/GO )
[EXECUTION]
RUNNING CBS08A
RUNNING CBS08B
RUNNING CBS08C

EXIT

.DIR *.SAV )

FILE     SAV     5  <055>  23-APR-73     DSKC!   [27,235]


.
```

Example 4 Loading LINK-10

The Command File

```
/NOINITIAL /LOGLEVEL:1 DSK:LINK/MAP /CONTINOABS =/RUNAME:LINK=
/HASHSIZE:1000/TEST:DDT/SYMSEG:H,LNKEXO,SCAN,HELPER=
,LNKINI,/NOSTART LNKSCN,LNKWLD,LNKFIO,LNKLOD,LNKOLD,LNKNEW,LNKF40=
,LNKCST,LNKCOR,LNKLOG,LNKERR,LNKMAP,LNKXIT,LNKSUB/SEARCH/GO
```

Running LINK-10 With the Terminal as the Log Device

```
      LINK-10 LOG file          23-Apr-73
8158:11   1   1 LIM LINK-10 initialization
8158:16   1   1 EXP Expanding low segment to 14P
8158:16   1   1 MOV Moving low segment to expand area DY
8158:16   1   1 LMN Loading module UDDT
8158:16   1   1 MOV Moving low segment to expand area GS
8158:17   1   1 EXP Expanding low segment to 18P
8158:17   1   1 MOV Moving low segment to expand area LC
8158:17   1   1 MOV Moving low segment to expand area LC
8158:17   1   1 LMN Loading module LNKEXO
8158:17   1   1 EXP Expanding low segment to 22P
8158:17   1   1 MOV Moving low segment to expand area LC
8158:18   1   1 LMN Loading module SCNDCL
8158:18   1   1 LMN Loading module ,SCAN
8158:18   1   1 MOV Moving low segment to expand area HC
8158:18   1   1 MOV Moving low segment to expand area HC
8158:18   1   1 MOV Moving low segment to expand area HC
8158:18   1   1 EXP Expanding low segment to 26P
8158:18   1   1 MOV Moving low segment to expand area LS
8158:18   1   1 MOV Moving low segment to expand area LS
8158:18   1   1 LMN Loading module ,TOUTS
```

LINK-10

Examples

```
8158|18  1  1 MOV Moving low segment to expand area HC
8158|18  1  1 LMN Loading module ,CNTDT
8158|18  1  1 EXP Expanding low segment to 30P
8158|18  1  1 MOV Moving low segment to expand area LS
8158|18  1  1 LMN Loading module ,SAVE
8158|18  1  1 LMN Loading module HELPER
8158|19  1  1 LMN Loading module LINK
8158|19  1  1 MOV Moving low segment to expand area HC
8158|19  1  1 EXP Expanding low segment to 34P
8158|19  1  1 MOV Moving low segment to expand area LS
8158|19  1  1 LMN Loading module LNKSCN
8158|19  1  1 MOV Moving low segment to expand area HC
8158|19  1  1 LMN Loading module LNKWLD
8158|19  1  1 MOV Moving low segment to expand area HC
8158|20  1  1 EXP Expanding low segment to 38P
8158|20  1  1 MOV Moving low segment to expand area HC
8158|20  1  1 MOV Moving low segment to expand area HC
8158|20  1  1 EXP Expanding low segment to 42P
8158|20  1  1 MOV Moving low segment to expand area LS
8158|21  1  1 LMN Loading module LNKFIO
8158|21  1  1 MOV Moving low segment to expand area LS
8158|21  1  1 LMN Loading module LNKLOD
8158|21  1  1 MOV Moving low segment to expand area HC
8158|21  1  1 EXP Expanding low segment to 46P
8158|21  1  1 MOV Moving low segment to expand area HC
8158|21  1  1 MOV Moving low segment to expand area LS
8158|22  1  1 LMN Loading module LNKOLD
8158|22  1  1 EXP Expanding low segment to 50P
```

•

•

•

```
8158|25  1  1 PLS Area LS overflowing to DSK
8158|28  1  1 MOV Moving low segment to expand area LS
8158|28  1  1 LMN Loading module LNKMAP
8158|28  1  1 LMN Loading module LNKXIT
8158|29  1  1 LMN Loading module LNKPRM
8158|29  1  1 LMN Loading module ,TSUBS
8158|29  1  1 LMN Loading module ,INSUB
8158|29  1  1 LMN Loading module JOBDAT
8158|30  1  1 MPS MAP segment
8158|30  1  1 MOV Moving low segment to expand area LS
8158|32  1  1 EMS End of MAP segment
8158|32  1  1 EXS EXIT segment
8158|33  1  1 SST Sorting symbol table
8158|34  1  1 EXP Expanding low segment to 67P
8158|34  1  1 MOV Moving low segment to expand area HC
8158|34  1  1 STC Symmbol table completed
8158|36  1  1 FIN LINK-10 finished
      [END OF LOG FILE]
```

## The Map File

```
                LINK-10 symbol map of   LINK    version (33)          page 1
        Produced by LINK-10 version (33) on 23-Apr-73 at  8158130

Low  segment starts at       0 ends at    5247 length      5247 = 6P
High segment starts at  400000 ends at  445460 length     45460 = 36P
        Start address is 405045, located in program LINK

                ***************

UDDT    from SYS:DDT,REL[1,5]   created on 10-Mar-73 at  0100100
        Low  segment starts at     140 ends at    4555 length     4415 (octal), 2317 (decimal)

                DDT         140     Entry  point    Relocatable
                DDTEND     4555     Global symbol    Relocatable
                $1R        4426     Global symbol    Relocatable
                $2B        4431     Global symbol    Relocatable
                $3B        4434     Global symbol    Relocatable
                $4B        4437     Global symbol    Relocatable
                $5B        4442     Global symbol    Relocatable
                $6B        4445     Global symbol    Relocatable
                $7B        4450     Global symbol    Relocatable
                $8B        4453     Global symbol    Relocatable
                $I         4422     Global symbol    Relocatable
                $M         4425     Global symbol    Relocatable

                ***************

LNKEXO  from DSK:LNKEXO,REL[11,131]     created on 22-Apr-73 at  9:44:00
        High segment starts at  400010 ends at  400020 length       10 (octal), 8 (decimal)

                ***************

SCNDCL  from DSK:SCAN,REL[11,131]       created on  1-Nov-72 at  0100100

        Zero length module

                ***************
```

LINK-10

Examples

```
,SCAN   from DSK:SCAN.REL[11,1311]      created on  1-Nov-72 at  01:00:00
        Low  segment starts at    4555 ends at    5245 length     470 (octal),  312 (decimal)
        High segment starts at  400020 ends at  404224 length    4204 (octal), 2180 (decimal)

E,FMI    400731  Global symbol  Relocatable
E,FMO    400720  Global symbol  Relocatable
E,SVNG   402102  Global symbol  Relocatable
E,SVTL   402074  Global symbol  Relocatable
E,UKK    402037  Global symbol  Relocatable
F,NAM      5117  Global symbol  Relocatable
,ALDON   400700  Global symbol  Relocatable
,CLRBF   403735  Global symbol  Relocatable
,DATIC   402166  Global symbol  Relocatable
,DATIF   402136  Global symbol  Relocatable
,DATIG   402137  Global symbol  Relocatable
,DATIM   402165  Global symbol  Relocatable
,DATIP   402152  Global symbol  Relocatable

                    • • •

Index to LINK-10 symbol map of LINK  version (33)          page 15

Name    Page    Name    Page    Name    Page    Name    Page

HELPER    4     LNKF40   11     LNKOLD    9     UUDT      1
JOBDAT   14     LNKF10    7     LNKPRM   14     ,GNTDT    3
LINK      4     LNKCOD    8     LNKSCN    5     ,INSUB   14
LNKCOR   11     LNKLOG   13     LNKWLD    6     ,SAVE    13
LNKCST   11     LNKMAP   13     LNKXIT   13     ,SCAN    15
LNKERR   13     LNKNEW   10     SCNDCL    1     ,TOUTS
LNKEXO    1

[End of LINK-10 map of LINK]
```

LINK-10

Examples

Example 5 Loading the Monitor

The Command File

```
/NOINITIAL /LOGLEVEL:1 /HASH:7000 TOPS10/SAVE=
,TOPS10/MAP = /LOCALS /MAXCOR:200K COMMON,COMDEV,COMMOD=
,TOPS10/SEARCH /NOSYSLIBRARY /GO
```

The Log File

```
        LINK-10 LOG file          23-Apr-73
9:04:12   1   1 LIM LINK-10 Initialization
9:04:19   1   1 EXP Expanding low segment to 14P
9:04:19   1   1 MOV Moving low segment to expand area DY
9:04:19   1   1 LMN Loading module COMMON
9:04:19   1   1 MOV Moving low segment to expand area LC
9:04:19   1   1 EXP Expanding low segment to 28P
9:04:19   1   1 EXP Expanding low segment to 32P
9:04:19   1   1 MOV Moving low segment to expand area LC
9:04:19   1   1 MOV Moving low segment to expand area LC
9:04:19   1   1 MOV Moving low segment to expand area LC
9:04:20   1   1 EXP Expanding low segment to 36P
9:04:20   1   1 MOV Moving low segment to expand area LS
9:04:20   1   1 EXP Expanding low segment to 40P
9:04:20   1   1 MOV Moving low segment to expand area LS
9:04:20   1   1 EXP Expanding low segment to 44P
9:04:20   1   1 MOV Moving low segment to expand area LS
9:04:21   1   1 EXP Expanding low segment to 48P
9:04:21   1   1 MOV Moving low segment to expand area LS
9:04:21   1   1 EXP Expanding low segment to 52P
9:04:21   1   1 MOV Moving low segment to expand area LS
9:04:22   1   1 MOV Moving low segment to expand area LS
9:04:22   1   1 LMN Loading module COMDEV
9:04:22   1   1 EXP Expanding low segment to 56P
9:04:22   1   1 MOV Moving low segment to expand area LC
9:04:22   1   1 MOV Moving low segment to expand area LC
9:04:22   1   1 MOV Moving low segment to expand area LC
9:04:23   1   1 EXP Expanding low segment to 60P
9:04:23   1   1 MOV Moving low segment to expand area LS
9:04:23   1   1 MOV Moving low segment to expand area LS
9:04:23   1   1 LMN Loading module COMMOD
9:04:23   1   1 EXP Expanding low segment to 61P
9:04:23   1   1 MOV Moving low segment to expand area LC
9:04:23   1   1 EXP Expanding low segment to 62P
```

LINK-10

Examples

```
9|04|23   1   1 MOV Moving low segment to expand area LC
9|04|23   1   1 EXP Expanding low segment to 63P
9|04|23   1   1 MOV Moving low segment to expand area LC
9|04|23   1   1 MOV Moving low segment to expand area LC
9|04|24   1   1 MOV Moving low segment to expand area LC
9|04|24   1   1 MOV Moving low segment to expand area LS
9|04|24   1   1 MOV Moving low segment to expand area LS
9|04|24   1   1 MOV Moving low segment to expand area LS
9|04|24   1   1 MOV Moving low segment to expand area GS
9|04|24   1   1 MOV Moving low segment to expand area LS
9|04|24   1   1 MOV Moving low segment to expand area LS
9|04|24   1   1 MOV Moving low segment to expand area GS
9|04|24   1   1 MOV Moving low segment to expand area LS
9|04|24   1   1 MOV Moving low segment to expand area LS
9|04|24   1   1 PLS Area LS overflowing to DSK
9|04|25   1   1 MOV Moving low segment to expand area GS
9|04|27   1   1 MOV Moving low segment to expand area LS
9|04|27   1   1 MOV Moving low segment to expand area LS
9|04|27   1   1 MOV Moving low segment to expand area LS
9|04|27   1   1 MOV Moving low segment to expand area GS
9|04|27   1   1 LMN Loading module EJBDAT
9|04|27   1   1 MOV Moving low segment to expand area LS
9|04|27   1   1 LMN Loading module FILFND
9|04|28   1   1 MOV Moving low segment to expand area LS
9|04|28   1   1 MOV Moving low segment to expand area LS
9|04|28   1   1 MOV Moving low segment to expand area GS
```

•
•
•

```
9|04|56   1   1 MOV Moving low segment to expand area LS
9|04|57   1   1 MOV Moving low segment to expand area LS
9|04|57   1   1 MOV Moving low segment to expand area LS
9|04|57   1   1 LMN Loading module ONCE
9|04|57   1   1 MOV Moving low segment to expand area LS
9|04|57   1   1 MOV Moving low segment to expand area LC
9|04|57   1   1 MOV Moving low segment to expand area LC
9|04|57   1   1 MOV Moving low segment to expand area LC
9|04|58   1   1 MOV Moving low segment to expand area GS
9|04|58   1   1 MOV Moving low segment to expand area LS
9|04|58   1   1 MOV Moving low segment to expand area LS
9|04|58   1   1 MOV Moving low segment to expand area LS
9|04|58   1   1 FCF Final core fixups
9|05|02   1   1 MPS MAP segment
9|05|02   1   1 MOV Moving low segment to expand area LS
9|05|18   1   1 EMS End of MAP segment
9|05|29   1   1 EXS EXIT segment
9|05|31   1   1 SST Sorting symbol table
9|05|35   1   1 STC Symmbol table completed
9|05|37   1   1 CSF Creating SAV file
9|05|51   1   1 FIN LINK-10 finished
        [END OF LOG FILE]
```

LINK-10

Examples

## The Map File

```
                LINK-10 symbol map of   TOPS10  version (50534)          page 1
      Produced by LINK-10 version (33) on 23-APR-73 at  9105102

Low  segment starts at      0 ends at  132776 length   132776 = 91P
        Start address is 106037, located in program SYSINI

              **************

COMMON  from DSKICOMMON,REL[11,131]     created on 12-Dec-72 at 10140100
        Low  segment starts at    140 ends at   7357 length    7217 (octal), 3727 (decimal)

A                0   Global symbol   Non-Relocatable Suppressed
A0.NOC       77000   Global symbol   Non-Relocatable Suppressed
A00VER       50534   Global symbol   Non-Relocatable Suppressed
A10500           0   Global symbol   Non-Relocatable Suppressed
ABSTAB         410   Global symbol   Non-Relocatable
ADVEVM        5023   Global symbol   Non-Relocatable Suppressed
AL               1   Global symbol   Non-Relocatable Suppressed
ALLJSP        5023   Global symbol   Non-Relocatable Suppressed
ANYRUN        5021   Global symbol   Non-Relocatable Suppressed
AP0BCK        6520   Global symbol   Non-Relocatable Suppressed
AP0CHL        5574   Global symbol   Non-Relocatable Suppressed
AP0CHN           3   Global symbol   Non-Relocatable Suppressed
AP0INT        6510   Global symbol   Non-Relocatable
AP0JEN        5606   Global symbol   Non-Relocatable Suppressed
AP0NUL      433553   Global symbol   Non-Relocatable Suppressed
AP0PDP        5645   Global symbol   Non-Relocatable Suppressed
AP0RET        5604   Global symbol   Non-Relocatable Suppressed
AP0RST      633553   Global symbol   Non-Relocatable Suppressed
AP0SAC        5625   Global symbol   Non-Relocatable Suppressed
AP0SAV        5576   Global symbol   Non-Relocatable Suppressed
APR0SN         240   Global symbol   Non-Relocatable Suppressed
APR1SN           0   Global symbol   Non-Relocatable Suppressed
APRSN          240   Global symbol   Non-Relocatable Suppressed
APRSTS        4625   Global symbol   Non-Relocatable
ASSCON      400000   Global symbol   Non-Relocatable Suppressed
ASSPRG      200000   Global symbol   Non-Relocatable Suppressed
AUAVAL        7140   Global symbol   Non-Relocatable Suppressed
AUQ              4   Global symbol   Non-Relocatable
AUUSER        7153   Global symbol   Non-Relocatable
AVALTB        7140   Global symbol   Non-Relocatable
```

**LINK-10**

Examples

```
AVTBMQ   7133  Global symbol  Non-Relocatable Suppressed
BATMAX   4475  Global symbol  Non-Relocatable
BATMIN   4476  Global symbol  Non-Relocatable
BATNUM   4501  Global symbol  Non-Relocatable
BIGHOL   2411  Global symbol  Non-Relocatable
BISTDV   1355  Global symbol  Non-Relocatable Suppressed
BLKSPK      3  Global symbol  Non-Relocatable Suppressed
BLKSPP      3  Global symbol  Non-Relocatable Suppressed
BNXMTS   4717  Global symbol  Non-Relocatable
BOOTWD     22  Global symbol  Non-Relocatable
BOOTXT   5004  Global symbol  Non-Relocatable Suppressed
BTHN        1  Global symbol  Non-Relocatable Suppressed
BYTTAB   1300  Global symbol  Non-Relocatable Suppressed
C0DPN       0  Global symbol  Non-Relocatable Suppressed
C0FHN       1  Global symbol  Non-Relocatable Suppressed

                    • • •

Index to LINK-10 symbol map of TOPS10 version (50534)          page 84

Name    Page    Name    Page    Name    Page    Name    Page

BTHINT    55    DTASKN    63    MEMCON    66    REMOLX    68
COPSER    55    EDDT      79    MTYSER    66    RTTRP     69
CORSRX    55    EJBOAT    47    ONCE      82    SCHED1    69
CLOCK1    56    ERRCON    63    ONCMOD    80    SCNSEK    70
COMCON    58    FHXKON    54    PATCH     78    SEGCON    74
COMDEV    27    FILFND    47    PLTSER    67    SWPSER    75
COMMOD    32    FILIO     50    PTPSER    67    SYSCHK    80
COMMON     1    FILUUO    52    PTRSER    67    SYSINI    78
CORE1     61    KALOCK    65    PTYSER    67    TMPUUO    75
DATMAN    62    KASER     64    REPSTR    81    UUOCON    75
DPXKON    54    LPTSER    65

[End of LINK-10 map of TOPS10]
```

LINK-10

Item Types

# APPENDIX A

## LINK Item Types

Input to LINK-10 is in the form of relocatable binary (.REL) files. Each .REL file is composed of link items of varying lengths. Each link item contains a specific type of information for LINK-10. The first word of these items is a header word containing, in the left half, an octal code for the item type and, in the right half, usually the number of words in the item. For item types 0-37, the count of words does not include overhead words (i.e., relocation words); for item types 1000-1777, the count does include these words. The format of the remaining words depends upon the individual link item. The link items are as follows:

| Link Item Type | Use |
|---|---|
| 0-37 | Reserved for DEC |
| 40-77 | Reserved for customers |
| 100-377 | Reserved for DEC |
| 400 | FORTRAN-IV (F40) marker block |
| 401 | FORTRAN-IV (F40) with MANTIS information |
| 402-777 | Reserved for customers |
| 1000-1777 | Reserved for DEC (not used in the first release of LINK-10) |
| 2000-3777 | Reserved for customers |
| 4000-777777 | Reserved to avoid conflict with ASCII text |

LINK-10

Item Types Ø-37

## A.1 Link Item Types 0-37

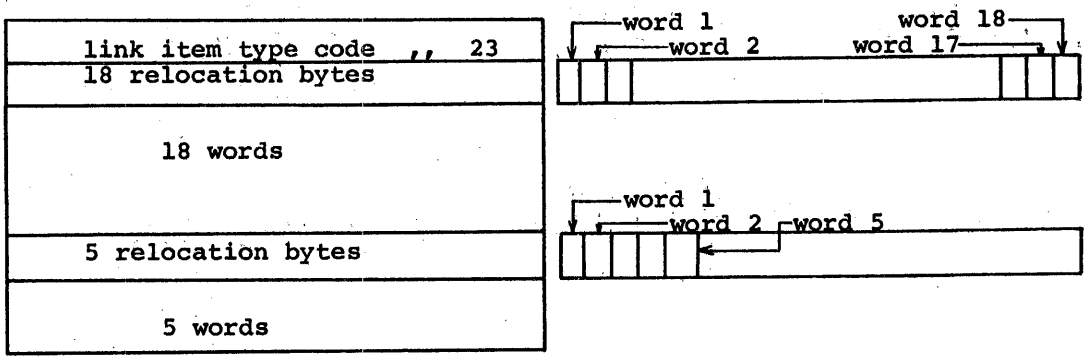
Link items in this range are the LOADER program block types and all have an identical format. The first word of the item, the header word, contains the item type in the left half and the count of data words in the right half. Following the header word is a relocation word containing up to 18 2-bit bytes which specify the relocation bits for the 18 words or less that follow. The relocation bits are left-justified and have the following meanings:

| Byte Value | Meaning |
|---|---|
| 0 | Do not relocate |
| 1 | Relocate right half of word |
| 2 | Relocate left half of word |
| 3 | Relocate both halves of word |

Following the relocation word are up to 18 words of the item. If there are more than 18 words in the item, there is another word of relocation bytes for the next 18 words. The relocation words are not included in the count of data words appearing in the left half of the header word. Thus, an item with a word count of 23 decimal would be as follows:

LINK-10

Item Types Ø-37




A.1.1 Link Item Type 0

This item type is ignored by LINK-10 and therefore can be used to store information not required by it. Totally null words look like this item type.

A.1.2 Link Item Type 1 CODE

This item type contains code and data. The first data word specifies the beginning address into which the item is to be loaded. The remaining words of the item are loaded into contiguous locations starting at that address. All words, including the load address, are relocated as specified by the relocation bytes.

If bit 0 of the first data word is 1, the word is assumed to be a Radix-50 symbol. The load address is then the value of this symbol plus the next word. Thus, in this case, there is one less actual data word than is indicated by the count in the header word.

### A.1.3 Link Item Type 2 SYMBOLS

This item type consists of symbols, with each symbol occupying two words. The first word of each symbol contains 4 bits of code (bits 0-3) and 32 bits of the Radix-50 representation of the symbol (bits 4-35). The second word is the value of the symbol.

The code bits are as follows:

00 This symbol is a program name. It is entered into the symbol table by a link item type 6, not an item type 2. (This code should never happen.)

04 This symbol is a global definition. Its value is available to other programs. Two global symbols with the same name but different values cause an error message.

10 This symbol is a local symbol and is not loaded unless the user requests the loading of local symbols. Local symbols of the same name can occur in different modules without causng an error, even though the values may be different.

14 This symbol is a block name and is used by translators that are block structured. This symbol is not loaded unless the user requests loading of local symbols.

44 Same as code 04, with the addition that the global symbol is suppressed to DDT typeout.

50 Same as code 10, with the addition that the local symbol is suppressed to DDT typeout.

60 This symbol is a global request.

If bit 0 of the second word in the pair is 0, then bits 18-35 contain the address of the first word in a chain of requests for the global. In each request, the right half of the second word contains the address of the next request. The chain is terminated when the right half of the second word contains zero.

If bit 0 of the second word in the pair is 1, the request involves additive global processing. When bit 2 of this word is 0, bits 18-35 contain an address of a word of code. The right half of the value of the symbol requested is added to the left or right half of this word of code according to the following rule:

If bit 1 of the second word in the pair is 1, the add

is to the left half.
If bit 1 of the second word is 0, the add is to the right half.

The result is stored back into the word of code. (Note that there is no full word add; that result must be accomplished by a left and a right add.)

When bit 2 of the second word is 1, bits 3-35 contain the Radix-50 representation of a second symbol, whose value depends upon the global being requested. The second symbol must be the last symbol defined before the global request or else it will be treated as a local symbol and no action will occur, unless local symbols are being loaded. When the value of the requested global is determined, it is added to the right half of the value of the second symbol if bit 1 of the second word is 0, or to the left half if bit 1 is 1. Since the actual value of the symbol is not determined until the definition of the global upon which it depends, the code bits of the symbol indicate that the value of the symbol will change and cannot be used to satisfy requests until the symbol is fully defined.

### A.1.4 Link Item Type 3 HISEG

This item type indicates to LINK-10 that code is to be loaded into the high segment. This item type has either one or two data words. The right half of the first data word is the initial address in the high segment (usually 400000). When the left half of the data word is zero, subsequent CODE items are assumed to have been produced by the HISEG pseudo-op in MACRO-10. This means that the addresses are relative to zero but are to be placed into the high segment. When the left half of the first data word is negative (i.e., greater than the right half) , subsequent CODE items have been generated by the TWOSEG pseudo-op in MACRO-10. This requires that addresses greater than the right half be placed into the high segment and addresses less than the right half be placed into the low segment. The left half is interpreted as the high segment break (i.e., the first free location

after the high segment) with the maximum length of the high segment being the difference between the left and right halves of the word. One-pass translators that cannot determine the high segment break should set the left half of the data word equal to the right half.

If there is a second data word (e.g., as in FORTRAN-10), the right half of this word is the low segment origin (usually 0) and the left half is the low segment program break.

### A.1.5 Link Item Type 4 ENTRY

This item type is the entry item and must be the first item in a .REL file if the .REL file is to be loaded in a library search. It consists of a list of Radix-50 symbols which are separated every 18 words by a relocation word of zeroes. When LINK-10 is in library search mode, each global symbol in the list is checked against the undefined global requests for the load. If one or more matches occur, the following module is loaded. If a match does not occur, the module is ignored. If LINK-10 is not in library search mode, this checking of undefined global requests is not performed.

The entry items are stored. If the module is not loaded, these items are ignored. If the module is loaded, the entry items are scanned again and the entry point bit is turned on for the corresponding symbol in the symbol table.

### A.1.6 LINK Item Type 5 END

This item type is the end item and is the last link item in the .REL

file. It contains two words whose meanings depend on whether the file contains two segments or one. If the file has two segments, the first word is the high segment break and the second word is the low segment break. If the file has only one segment, the first word is the first free location above the program (this word is relocatable) and the second word is the highest absolute address seen, if higher than location 137.

### A.1.7 Link Item Type 6 NAME

This item is the name item and must appear before any type 2 link item (SYMBOL), The item has one or two data words. The first word is the program name in Radix-50 symbol format. The second word, if it appears, contains in bits 6-17 a code for the translator that produced the binary file, and in the right half the length of blank COMMON. (FORTRAN programs use both named and blank COMMON. COBOL uses blank COMMON to indicate the length of LIBOL's static area. Thus, the length has meaning for FORTRAN and COBOL programs.) The octal codes (bits 6-17) for the various translators are as follows:

| Octal Code | Translator |
|---|---|
| 0 | UNKNOWN |
| 1 | F40 |
| 2 | COBOL |
| 3 | ALGOL-60 |
| 4 | NELIAC |
| 5 | PL/1 |
| 6 | BLISS-10 |
| 7 | SAIL |
| 10 | FORTRAN-10 |
| 11 | MACRO |
| 12 | FAIL |

Bits 0-5 of the second word indicate the processor on which the program will execute. If the value of these bits is 0, the program will execute on either processor; if the value is 1, the program will execute only on the KA10 processor; and if the value is 2, the program will execute only on the KI10 processor. Remaining values are reserved for the future.

### A.1.8 Link Item Type 7 START ADDRESS

This item type contains in the right half of the data word the address at which execution of the program is to begin. The start address for a relocatable program may be relocated by means of the relocation bits. The last link item of this type encountered by LINK-10 is the one used, unless LINK-10 is ignoring start addresses (indicated by the user via switches). If the program is not to specify a start address, no item of this type should be included.

### A.1.9 Link Item Type 10 INTERNAL REQUEST

This item type is provided for one-pass language translators when internal symbols are used before they are defined. The item type consists of a series of data words where each word represents one request. Each data word has a value in the right half and a pointer to the last request in the chain of requests for that value in the left half. Each quantity may be relocatable. The symbols are chained in a manner similar to the global requests which have bit 0 in the second word of each pair equal to zero (i.e., the value is substituted in the right half of each location in the chain). However, if a data

word is -1, then the next data word indicates a chained request to the left half of the word specified rather than the right half.

### A.1.10 Link Item Type 11 POLISH

This item type is provided for Polish fixups involving arithmetic and logical operations on relocatable or externally-defined quantities. Each item contains only one Polish string. The data words in each item are a series of half-words consisting of operators and operands followed by store operators and store addresses. The operators and operands are as follows:

| | |
|---|---|
| 0 | The next half word is an operand. |
| 1 | The next two half-words form a 36-bit operand. |
| 2 | The next two half-words form a Radix-50 symbol which is a global request. The operand is the value of the global. |
| 3 | Add. |
| 4 | Subtract. |
| 5 | Multiply. |
| 6 | Divide. |
| 7 | Logical AND. |
| 10 | Logical OR. |
| 11 | Left shift. |
| 12 | Logical XOR. |
| 13 | One's complement (not). |
| 14 | Two's complement (negative). |

The store operators are as follows:

| 18 bit value | |
|---|---|
| -1 | Right half chained fixup (777777). |
| -2 | Left half chained fixup (777776). |
| -3 | Full word chained fixup (777775). The entire word pointed to is replaced and the old right half points to the next full word. |

The half word following the store operator is used as the address of the first element in the chain.

### A.1.11 Link Item Type 12 LINK

Data words in this item type occur in pairs. The first word of the pair is a link number and the second word is an address. There are 20 (octal) links numbered from 1 to 20. When LINK-10 is initialized, the value of each link is set to zero. Each time a specific link is seen, the current value of the link is stored in the address specified by the second word of the word pair, and the specified address becomes the new value of the link. If the number of the link seen is negative, the address is saved as the end of the link . At the end of loading, the current value for each link is stored in the address indicated by the end of each link. If the end of the link is 0, no storing is done.

### A.1.12 Link Item Type 13 LVAR

This item type is used in LVAR fixups and is not currently handled by LINK-10. It is not supported by DEC and is not needed because the TWOSEG pseudo-op is superior. The first data word is the location of a variable area in the low segment. The second data word is the length of the area needed. The low segment relocation counter is incremented by the area needed. Data words after the first two data words occur in pairs. If bit 2 of the first word of the pair is zero, then the second word contains, in its left half, the address of a fixup chain, and in the right half, the relative location in the variable area to use for this fixup. The chaining occurs with the right half of the words if bit 0 of the first word is 0; otherwise, chaining occurs with the left half of the words.

LINK-10

Item Types Ø-37

If bit 2 of the first word of the pair is one, then the pair is used to make a symbol table fixup. The right half of the first word is the value of the fixup. The second word is the Radix-50 representation for the symbol.

A.1.13 Link Item Type 14 INDEX

This item type is produced by FUDGE2 to identify an index to LINK-10. The index is a list of all entry points (Link item type 4) in a library .REL file with pointers to the beginning of the individual modules. The index is 200 octal words long and if there are more entries in the library than will fit in 200 words, other item types 14 are created to contain the remainder of the entries. Each index is divided into sub-items of various lengths. The sub-items do not include the relocation word normally found in entry items of a library. Each sub-item has a header word with the word count in the right half and the link item type 4 in the left half. Following this header is the list of Radix-50 entry symbols. After the list of entries, there is a pointer to the individual module within the library file. The right half of the pointer is the block number of the module, and the left half is the word count within the block for the start of the module. The last word of the index item type contains a -1 in the left half to signal the end of the index item and the block number of the next index item in the right half. If LINK-10 is not in library search mode, index items are ignored.

### A.1.14 Link Item Type 15 ALGOL

This item type is the special ALGOL OWN item. The first data word is the length of the OWN area to be allocated in the low segment. The remaining words are chained with the right half of the OWN fixups.

### A.1.15 Link Item Type 16 REQUEST LOAD

This item type is produced by the SAIL compiler and is used to request the loading of programs. Thus, a .REL file can request libraries and other files to be loaded, thereby keeping the command string to LINK-10 simple. LINK-10 maintains a table for the names of libraries to be loaded and another table for the names of standard relocatable binary files to be loaded. When a new file is requested by link item type 16 or 17, LINK-10 searches the appropriate table to verify that the file has not already been specified. If it has not been specified, an entry is made in the appropriate table. After all files in the LINK-10 command string have been loaded, the files specified in the two tables are loaded. The relocatable binary files are loaded first; the libraries are loaded last.

The data words in this link item type appear in triplets. The first word contains the filename in SIXBIT (the extension of .REL is assumed). The second word is the UFD number in binary, and the third word is the SIXBIT name of the device containing the file.

### A.1.16 Link Item Type 17 REQUEST LIBRARY

This item type is the same as item type 16 except that the specified

files are loaded only if they are needed to satisfy global requests. That is, the files are loaded in library search mode. The data words are identical to those in item type 16.

### A.1.17 Link Item Type 20 COMMON ALLOCATION

This item type is used to allocate named COMMON areas. The relocation word must be present, but the bits should be zero. The data words are grouped in pairs, where the first word contains the Radix-50 symbol for the name of the COMMON area and the second word contains the length of the area required by this program.

This item type causes LINK-10 to search for the specified COMMON area to determine if it has been previously loaded. If it has, the length given in this item type must be less than or equal to the length already allocated. Thus, the first program that defines a COMMON area also defines the maximum size of that COMMON area. No subsequent program can expand this particular area, although COMMON areas of different names can be defined.

If the specified COMMON area has not been loaded, the symbol name is given the current low segment relocation value, and the length of the area is added to the low segment relocation counter.

### A.1.18 Link Item Type 21 SPARSE DATA

This item type is used to load data into arrays when link item type 1 is inefficient for this purpose. The data words are grouped in sub-items and each sub-item is treated in the same manner as link item

type 1. The first word of each sub-item contains in the left half a count of the number of data words in the sub-item, and in the right half the beginning address into which the data words are to be loaded. The remaining words of each sub-item are the data words.

If bit 0 of the first word of a sub-item is 1, the first word is assumed to be a Radix-50 symbol. The left half of the second word is the count of data words and the right half contains an offset. The load address is then the value of the symbol plus the offset.

### A.1.19 Link Item Types 22-36

These item types are not yet defined and return an error message if used.

### A.1.20 Link Item Type 37 DEBUG

This item type is used for the debugging symbol table for COBDDT (the COBOL debugging program). If debugging is requested in local symbol mode, the data from this item type is loaded in the same manner as the data from link item type 1. If local symbols are not required, this item type is ignored.

## A.2 Link Item Type 400 FORTRAN (F40)

This item type is output by the old one-pass FORTRAN-IV complier (F40). It does not contain a word count, relocation words, or data words. It contains only the one word indicating the item type code.

LINK-10

Item Type 401

## A.3 Link Item Type 401 FORTRAN (F40)

This item type is similar to link item type 400 and in addition it indicates that the file contains MANTIS debugging information.

## A.4 Link Item Types 1000-1777

Link items in this range do not have identical formats. There is a general pattern in that the first word of each item contains an item type number in the left half and a word count in the right half. However, unlike link item types 0-37, the word count of item types 1000-1777 is a count of all following words including overhead words (relocation words). The structure of the relocation words depends upon the link item; there may be any number of relocation bits from 1 to 18 per half or full word. Link items that do not need relocation do not have relocation words. These item types are not used in the first release of LINK-10.

### A.4.1 Link Item Type 1000

This item type is ignored by LINK-10 and thus can be used to store information not required by it.

### A.4.2 Link Item Type 1001 ENTRY

This item type is the simple entry item and consists of a list of SIXBIT symbols. Each data word contains one left-adjusted symbol which can be a maximum of six characters in length. There are no relocation words, thus distinguishing this item type from item type 4. However, the two item types are used in the same manner.

### A.4.3 Link Item Type 1002 LONG ENTRY

This item type contains one extended symbol (i.e., the symbol contains more than six characters) in SIXBIT, which is tested to determine if it is required as an entry point. This link item type is used in the same manner as link item type 1001.

### A.4.4 Link Item Type 1003 NAME

This item type contains information about the file and the translator that produced it. The information in this item is stored in the symbol table and can be output on a map listing.

The data words occur in triplets. The left half of the first word of each triplet contains flag bits for that triplet and the right half is unused. The first triplet of data (the primary triplet) contains the program name in SIXBIT in the second word. This program name is taken from the TITLE statement in a MACRO-10 program. If the program name is longer than six characters, one or more triplets follow containing the remaining characters of the name. Triplets following the program name are identified by the flag bits in the first word of each triplet. The triplet after the name triplets contains the low segment relocation counter in the second word and the high segment relocation counter in the third word. The next triplet has, in the second word, the SIXBIT name of the translator that produced the file and in the third word, the version number of the translator. This version number is taken from location 137. The following triplet contains the compilation date and time obtained from the LOOKUP UUO block in the

second word, and in the third word, a default code for the translator used, in case LINK-10 could not determine the translator from the information in the previous triplet. The default translator codes are listed in Paragraph A.1.7. The next triplet contains in the second word, the name of the device on which the source file is stored, and in the third word, the SIXBIT filename of the source file. The information in the next triplet is the source filename extension in the second word and the name of the UFD containing the source file in the third word. The next triplet contains sub-file directory information. The following triplet contains the version number of the source file as obtained by the translator that processed the file. The information in the last triplet is interpreted as ASCII text and is stored in the format in which it is given.

More than one NAME link item may be seen per module for programs made from several source files. The program and compiler name triplets must be the same in the the NAME link items, but the source filename and any remaining triplets can be different.

### A.4.5 Link Item Type 1004 RELOCATION

This item type consists of groups of words (usually pairs) without any relocation words. The first data word of the item type contains the total number of relocation groups in the item in order that sufficient space can be allocated. The first word of each relocation group has a relocation level in the left half and the count of the number of words in the relocation counter name in the right half. The remaining words in each group are the relocation counters. The relocation level is

the position in the table of relocation counters, such that for any word needing relocation, the value of the relocation byte will receive the correct constant for addition.

If a relocation counter is not yet defined (or a complex Polish expression not yet resolved), it must be placed in the undefined table, and its slot in the relocation tables is marked as undefined. All code referring to the undefined counter is stored in the fixup area or on the disk. In other words, if the location into which code is to be loaded is not yet defined, all the code under the relocation counter must be placed in the fixup table or on the disk. Link item type 1004 can appear anywhere and must be used whenever a new relocation counter is used. The standard name for the low segment relocation counter is .LOW. and the standard for the high segment counter is .HIGH.. These counters normally occupy positions 1 and 2 in the table of relocation counters.

### A.4.6 Link Item Type 1005

This item type is undefined and reserved for future definition.

### A.4.7 Link Item Type 1006 START

This item type contains the start addresses for the program. It consists of a relocation word with 4-bit bytes for full word relocation, followed by the list of relocatable start addresses in order of their use. These addresses are used or ignored depending on the switches given by the user. Currently, only one start address per program is recognized.

LINK-10

Item Types 1000-1777

### A.4.8 Link Item Type 1007 START

This item type is used for additional start addresses or external symbolic start addresses. The link item is divided into groups of words for each start address. The first word of each group contains flag bits in the left half and the count of the number of words in the group in the right half. Currently, bit 0 is the only flag bit. If this bit is 1, a Polish expression follows; if it is 0, a symbol follows. This item type does not include relocation words.

### A.4.9 Link Item Types 1010-1017 CODE

The link items in the range 1010-1017 are similar except for the size of the relocation byte. The most general case uses 18 bits per half word, but this method consumes too much space for simple programs. Item type 1010 has a byte size of 2 bits, thereby allowing three relocation counters and absolute code. Relocation occurs only on the right half of the word and is positive; the left half is considered absolute. Since in most programs the code consists of constants in the left half (op-codes, indexes, ACs) and relocatable addresses or constants in the right half, this item type should be sufficient for most programs.

Item type 1011 also has 2-bit bytes but has relocation for the left half as well as the right half of the word. This item type allows three relocation counters plus absolute code. Link item type 1011 is used mainly for table generation.

Item type 1012 allows relocation only for the right half of the word (similar to item type 1010) but has a byte size of 4 bits, giving allowances for 15 relocation counters.

Item type 1013 allows relocation for both the left and right halves of the word (similar to item type 1011) but uses a 4-bit byte size.

Item types 1014-1016 are reserved for future use.

Item type 1017 has 18 bits of relocation per half word.

### A.4.10 Link Item Types 1020-1027 SYMBOL

All symbols are in triplet format. The link items in the range 1020-1027 differ only in the size of the relocation b-te. This byte is the same as the byte size for the corresponding CODE item. For example, symbol type 1020 and code type 1010 use 2-bit bytes, symbol type 1022 and code type 1012 use 4-bit bytes, and so forth. The relocation word applies only to the third word of the triplet (the symbol value). Thus, for example, in the case of symbol type 1020, each relocation word is followed by up to 18 triplets rather than 18 words.

### A.4.11 Link Item Type 1030 POLISH

This item type is provided for Polish fixups and consists of operators and operands, including store operators and store operands in pre-fixup form. Each item contains only one Polish string, but may contain many different store pointers. Operators are stored one per half word, and symbols are stored in contiguous half words. Store

LINK-10

Item Types 1000-1777

pointers are in the form of either an address in a halfword or a byte pointer in a full word. Associated with store pointers are store operators that shift the value to the correct field and store operator. The store operator may also point to a symbol that is to be stored in the symbol table.

The operators and operands are as follows:

| | |
|---|---|
| 0 | The next half word is an operand. |
| 1 | The next two half words form a 36-bit operand. |
| 2 | The next two half words form a 36-bit symbol which is a global request. The operand is the value of the global. |
| 3 | The next half word is the count of half words in an extended symbol. The subsequent half words are the symbol. |
| 4 | The next half word is a numeric relocation counter for the program. |
| 5 | The next two half words are a symbolic relocation counter. |
| 6 | The next half word is a count of the number of half words in an extended symbolic relocation counter. The following halfwords are the relocation counter. |
| 7 | The next two half words are a byte pointer to code already loaded. |
| 10-77 | Reserved for future use. |
| 100 | Add |
| 101 | Subtract |
| 102 | Multiply |
| 103 | Divide |
| 104 | Logical AND |
| 105 | Logical OR |
| 106 | Left Shift (LSH) |
| 107 | Logical XOR |
| 110 | One's complement (not) |
| 111 | Two's complement (negate) |
| 112 | Get contents (MOVE) |
| 113 | Reserved for future use |

The store operators are as follows:

| 18 Bit Value | |
|---|---|
| -1 | Right half chained fixup (777777). |
| -2 | Left half chained fixup (777776). |
| -3 | Full word chained fixup (777775). |
| -4 | The next two half words are a byte pointer |

(777774).

-5 The next two half words are an instruction plus an address (ANDM,XORM) (777773).

-6 The next two half words are a symbol and the value is stored in the half words (777772).

-7 The next half word is the count of the number of half words in an extended symbol. The half words following are the extended symbol and the value is stored in these half words (777771).

-10 The next half word is a numeric relocation counter (777770).

-11 The next two half words are a symbolic relocation counter (777767).

-12 The next half word is a count of the number of half words in an extended symbolic relocation counter. The following half words are the counter (777766).

-13 Reserved for future use.

The store operators obtain their arguments from a stack; the first word is usually the value and the second is the memory address. Addresses can be built using other Polish operators. For chained fixups, the half word preceding the store operator is used as the address of the first element in the chain.

### A.4.12 Link Item Type 1031 POLISH

This item type is similar to item type 1030 except that Polish notation in post-fixup form is used. The operators and operands are the same.

### A.4.13 Link Item Types 1032-1033

These item types are reserved for future use.

### A.4.14 Link Item Types 1034-1037 CONDITIONAL

There are three kinds of conditonal loading item types: the Begin

conditional, the End conditional, and the Else conditional. The Begin conditional has a unique number assigned by the translator which is matched with the unique number in the End and Else conditionals. It also contains a conditional operand and operator. The End conditional cancels the conditional loading, updates the relocation counters, and generates the next implicit relocation counter, if it is not explicitly defined by the user, so that following code can be loaded. The Else conditional is the inverse of the condition in the Begin conditional in that code is loaded if the condition is false. The three kinds of condition items can be nested.

A.4.14.1 The Begin Conditional - Link Item Type 1034 - This item type has four relocation bits per half word thereby allowing 15 possible relocation counters. The first data word contains the unique conditional number. If a number is not specified, zero is assumed and LINK-10 matches the Begin with the first End or Else conditional at that level. The second data word contains the conditional operator in the left half and the conditional operand in the right half. The remaining words contain the rest of the operand.

The conditional operators are coded as follows:

| | |
|---|---|
| 0 | null |
| 1 | if zero |
| 2 | if greater than zero |
| 3 | if greater than or equal to zero |
| 4 | if less than zero |
| 5 | if less than or equal to zero |
| 6 | if not equal to zero |
| 7 | if defined |
| 10 | if not defined |
| 11 | if global |
| 12 | if local |

LINK-10

Item Types 1000-1777

The operand is either a symbol or a Polish expression. If the operand cannot be evaluated, the words are stored on the disk. The operands are:

| | |
|---|---|
| 100 | The next two half words contain a SIXBIT symbol. |
| 101 | The next half word is a count of the number of half words in an extended symbol. The following words contain the SIXBIT symbol. |
| 102 | A pre-fixup Polish expression follows (refer to Paragraph A.4.11). |
| 103 | A post-fixup Polish expression follows (refer to Paragraph A.4.12). |

If the condition is met, all code up to an End or Else conditional is loaded. When the condition is not met, the code is not loaded.

A.4.14.2 The Begin Conditional - Link Item Type 1035 - This item type is similar to Link Item Type 1034 except that it has half word relocation per half word.

A.4.14.3 The Else Conditional - Link Item Type 1036 - This item type contains no relocation words and has one data word containing a unique number matching the one in the Begin conditional. If the condition in the Begin conditional is true, the code in the current Else conditional to its matching End conditional or to the next matching Else conditional is ignored. If the condition is not true, the code is loaded.

A.4.14.4 The End Conditional - Link Item Type 1037 - This item type also has no relocation words. The first data word is a unique number matching the one in the Begin conditional. If the condition in the Begin conditional is false and no Else conditional is seen, the End

conditional is ignored. However, if code was loaded, the End conditional is read. The item type contains one data word for each relocation counter used in the same order as specified in the last relocation setting link item. The data words are the highest value of the relocation counter used in the conditionally-loaded code. These values are added to the current values, and to the accumulation of such values, until the final END item type of the REL file.

### A.4.15 Link Item Type 1040 END

This link item marks the end of a link module. It does not contain relocation words but does contain a list of all relocation counters used and their final values. Any conditional code that was loaded plus other overhead items, such as the ALGOL item, are added to the final values. The resulting values are then added to the current values of the relocation counters to obtain the value for the next module. The beginning and ending addresses are stored in the symbol table in order that DDT has the range of the program and that they can be output in a map listing.

### A.4.16 Link Item Type 1041 Special FORTRAN-10 Block

This link item defines a call to a special once-only routine that is to be executed by LINK-10 after all code has been loaded.

### A.4.17 Link Item Type 1042 Program Request

This link item requests the loading of .REL files required for this program. It is similar to link item type 16; however, there are no

relocation words. This item replaces the need for library searches and is useful when loading real and dummy routines because it specifies filenames rather than modules names.

The data appears in groups of four or more words. Each group contains the following words:

Name of the device in SIXBIT containing the file.

Name of the file in SIXBIT.

Extension of the file in SIXBIT in the left half, and the length of the directory in the right half.

UFD in octal.

Remaining words in the group are sub-file directory names in SIXBIT.

The requests are stored until the end of loading and are loaded before the default libraries and requested libraries (link item type 1043). Any number of files can be requested.

### A.4.18 Link Item Type 1043 Library Request

This item type requests the searching of libraries, either in search mode for all unresolved entries or for particular modules. The data is identical to that in item type 1042.

### A.4.19 Link Item Types 1044-1047

These item types are reserved for future use.

LINK-10

Item Types 1000-1777

### A.4.20 Link Item Type 1050 Global Data

This item type contains data that is common to many programs (i.e., constants, argument lists, literals in MARCO-10 language). The global data item consists of two other link items: the relocation setting item (type 1004) and a code item (types 1010-1017). The initial global data item has no relocation words. The first data word is the header of the relocation item and only the relocation actually used should appear in this word; all other entires should be zero. The next data words are the data for the relocation item. Following these data words is a code item with relocation bits and data which may be relocatable or absolute. LINK-10 collects all the global data blocks, compares them, and keeps only one copy of those with the same data and relocation. The global data items are loaded at the end of loading or immediately after a /DATA switch is seen. These items should reduce the size of loaded programs because of pooling of literals.

### A.4.21 Link Item Types Greater Than 3777 ASCII

These items are recognized by the first seven bits being non-zero (i.e., an ASCII character). There is no word count in the item. Termination of the item occurs at a null byte. These items are generated by translators and contain ASCII commands similar to those typed on the user's terminal. Thus they are similar to an indirect file. ASCII items allow the overlay structure to be embedded in the file to simplify the maintaining of large overlay programs.

LINK-10

LOADER and LINK-10 Differences

## APPENDIX B

## LOADER AND LINK-10 DIFFERENCES

This appendix is intended as an aid for users who have been employing the LOADER program and who are now converting to the LINK-10 program. Both programs are linking loaders. Both have the same basic functions of loading and relocating user's object code modules and resolving references among the modules. But LINK-10 is not just an updated version of LOADER. It is a completely new, more sophisticated, and more flexible piece of software. This appendix itemizes the differences between the two programs in order to facilitate conversion to LINK-10.

| LOADER | LINK-10 |
|---|---|
| The default output device is TTY. | The default output device is DSK. |
| The default name of the MAP file is MAP.MAP. | The default name of the MAP file is the name of the last program with a start address. If there is no program with a start address, the default name is nnnLNK.MAP, where nnn is the user's job number. |
| Command files are specified in the form<br>* file @<br>The default extension of the command file is .TMP. | Command files are specified in the form<br>* @ file<br>The default extension of the command file is .CCL. |
| Input and output specifications are separated by a back-arrow (←). Thus, an output file is defined as being on the | Input and output specifications may be separated by an equals sign (=), but this is not required. An output file is |

LINK-10

LOADER and LINK-10 Differences

| | |
|---|---|
| left side of the back-arrow. | specified by giving a file specification followed by an output switch. |
| The only output file produced by LOADER is a map file. | LINK-10 can be instructed to produce map, save, log, symbol, and XPN files. |
| Exit conditions are /G, altmode, and ↑Z. | The only exit condition is /GO. |
| Line terminators (e.g. <carriage return, line feed>) are treated in the same way as commas (i.e., they terminate the specification). File dependent switches remain in effect until overridden by a subsequent switch or until the end of the load. The most recently specified source device remains the default until a new device is specified or until the end of the load. Defaults carry across lines. | LINK-10 has a line oriented scanner. All file-dependent switches are turned off at the end of the line to which they belong. The most recently specified source device remains the default until a new device is specified or until the end of a line is reached. Standard defaults are restored at the beginning of each line. In general, it is best to place all the commands for loading a program on a single line. A hyphen is used as the line continuation character. |
| To load local symbols for FILE1 and FILE2 and then load DDT, the following sequence could be used:<br>`*/S`<br>`*FILE1,FILE2`<br>`*/W/D$` | To load local symbols for FILE1 and FILE2 and then to load DDT, the following sequence is used:<br>`*/LOCALS FILE1,FILE2,`<br>`*/TEST /GO`<br>Note that if the /LOCALS switch had appeared on a line by itself, it would have had no effect. |
| To search FILEA and FILEB in library search mode, the sequence:<br>`*/L`<br>`* FILEA,FILEB` | To search FILEA and FILEB in library search mode, the sequence is:<br>`*/SEARCH FILEA,FILEB`<br>The sequence |

LINK-10

LOADER and LINK-10 Differences

| | |
|---|---|
| could be used. | */SEARCH<br>*FILEA,FILEB<br><br>does not cause FILEA and FILEB to be searched. Instead, they are loaded in their entirety. |
| When performing a search of the default libraries at the end of the load, LOADER makes one pass through all required libraries. In addition, LIB40 is always searched. | LINK-10 performs multiple passes through all required libraries until no undefined symbols remain or until no additional routines have been loaded. In addition, LIB40 is not automatically searched unless it is required by an F40 program. Thus, when loading MACRO programs which utilize routines in LIB40, the user must explicitly request that LIB40 be searched. Also, JOBDAT.REL is not searched unless the /NOINITIAL switch is used. LINK-10 automatically initializes its global symbol table to include JOBDAT symbols. |
| The /D and /T switches load with local symbols. This mode remains in effect until it is turned off with the /W switch, and remains off until another switch which loads local symbols is given. | The /TEST and /DEBUG switches instruct LINK-10 to load all subsequent files with their local symbols. The /NOLOCAL switch can be used to suppress the loading of local symbols. However, since the /NOLOCAL switch is file dependent, it is cleared at the end of the line and load with local symbols mode is reinstated. |

The following table lists each LOADER switch and the LINK-10 switch which performs the nearest equivalent action. Note that there is not always a one-to-one correspondence between the action performed by the LOADER switch and by the LINK-10 switch. Refer to Chapter 4 of the LINK-10 Programmer's Reference Manual for the complete descriptions of the LINK-10 switches.

LINK-10

LOADER and LINK-10 Differences

| LOADER | LINK-10 |
|---|---|
| /A | /CONTENT:ZERO |
| /B | /SYMSEG:LOW |
| /1B | /SYMSEG:HIGH |
| /nnnnB | /PATCHSIZE:nnnn |
| /C | No equivalent switch. LINK-10 does not support the old CHAIN facility. |
| /D | /TEST:DDT or /TEST:MACRO |
| /E | /EXECUTE |
| /F | /SYSLIB |
| /1F | /FORSE |
| /2F | /FOROTS |
| /G | /GO |
| /nnnG | /START:nnn |
| /H | /SEGMENT:LOW |
| /1H | /SEGMENT:HIGH |
| /nnnnH | /SET:.HIGH.:nnnn |
| /-H | /SEGMENT:DEFAULT |
| /I | /NOSTART |
| /J | /START |
| /nK | /RUNCOR:n |
| /-K | No equivalent switch. Use /RUNCOR. |
| /L | /SEARCH |
| /M | /MAP:END |
| /1M | /MAP:END/CONTENT:LOCALS |

LINK-10

LOADER and LINK-10 Differences

| | |
|---|---|
| /N | /NOSEARCH |
| /nnnO | /SET:.LOW.:nnn |
| /P | /NOSYSLIB |
| /Q | /SYSLIB at the end of the command string. |
| /R | No equivalent switch. LINK-10 does not support the old CHAIN facility. |
| /S | /LOCALS |
| /T | /DEBUG:DDT or /DEBUG:MACRO |
| /U | /UNDEFINED |
| /V | /OTS:HIGH |
| /-V | /OTS:LOW |
| /W | /NOLOCALS |
| /X | /CONTENT:NOZERO |
| /Y | /REWIND |
| /Z | /RUN:LINK |

LINK-10

Glossary

# GLOSSARY

## Absolute Address

A fixed location in user virtual address space which cannot be relocated. For example, the high-speed accumulators on the DECsystem-10 occupy locations 0 through 17 (octal) in the user's virtual address space. All modules that reference the accumulators must reference these locations. Thus the addresses 0 through 17 (octal) are absolute addresses.

## Absolute Module

A module whose location counters are set to absolute addresses only.

## Address Binding

The assignment of virtual address space to the physical address space in computer memory. This is automatically performed by the DECsystem-10 monitor and is completely invisible to user programs.

## Assemble

To prepare a machine-language module from a symbolic-language module by substituting the actual numeric operation codes for symbolic operation codes, and absolute or relocatable addresses for symbolic addresses.

LINK-10

Glossary

Assembler

A program which accepts symbolic assembly code and translates it into machine instructions. MACRO-10 is the standard assembler supplied by DEC.

Base Address

An address used as a basis for computing the value of some other address. This computation is usually of the form

final address = base address (+ or -) offset.

Clear

To erase the contents of a location by replacing the contents with blanks or zeroes.

COMMON Area

A section in a program's address space which is set aside for common use by many modules. COMMON is usually set up by modules written in the FORTRAN language. It is used by independently-compiled modules to share the same data locations.

Control Section

A unit of code (instructions and/or data) that is considered an entity and that can be relocated separately at load time without destroying the logic of the program. Control is passed properly from one Control Section to another regardless of their relative positions in user virtual address space. A Control Section is

LINK-10

Glossary

identified by a relocation counter and thus is the smallest unit of code that can be relocated separately.

Default Directory

The directory in which the Monitor searches if a directory specification has not been given by the user. Typically, this is the UFD corresponding to the user's logged-in project-programmer number but it may another UFD or a SFD (sub-file directory).

Directory

A file which contains the names and pointers to other files on the device. The MFD, UFDs, and SFDs are directory files. The MFD is the directory containing all the UFDs. The UFD is the directory containing the files existing in a given project-programmer number area. The SFD is a directory pointed to by a UFD or a higher-level SFD. The SFDs exist as files under the UFD.

External Symbol

A global symbol which is referenced in one module but defined in another module. The EXTERN statement in MACRO-10 is used to declare a symbol external. A subroutine name referenced in a CALL statement in a FORTRAN module is automatically declared external.

LINK-10
Glossary

File

An ordered collection of 36-bit words comprising computer instructions and/or data. A file is stored on a device, such as disk or magnetic tape, and can be of any length, limited only by the available space on the device and the user's maximum space allotment on that device.

File Specification

A list of identifiers which uniquely specify a particular file. A complete file specification consists of: the name of the device on which the file is stored, the name of the file including its extension, and the name of the directory in which the file is contained.

FUDGE2

A system utility program used to update libraries containing one or more modules and to manipulate modules within these libraries.

GET

To transfer a saved program from a file on a device into core memory using a bootstrap program or the Monitor. The GET command places a program into memory. The RUN command performs the same operation and, in addition, starts the program. The GET operation differs from the LOAD operation (refer to LOAD).

LINK-10

Glossary

GLOB

A system utility program used to read libraries and to generate an alphabetical cross-referenced list of all the global symbols encountered. When a program is composed of many modules which communicate via global symbols, it is useful to have an alphabetical list of all global symbols with the names of the modules in which they are defined and referenced.

Global Request

A request to LINK-10 to link a global symbol to a module.

Global Symbol

A symbol that is accessible to modules other than the one in which it is defined. The value of a global symbol is placed in LINK-10's global symbol table when the module containing the symbol definition is loaded.

High Segment

That portion of the user's addressing space, usually beginning at relative location 400000, which generally is used to contain pure code that can be shared by other users. This segment is usually write-protected in order to preserve the data. The user can place information into a high segment with the TWOSEG pseudo-op in MACRO-10. Higher-level language, such as COBOL and FORTRN, also have provisions for loading pure code in the high segment.

LINK-10
Glossary

Initialize

To set counters, switches, or addresses to zero or other starting values at prescribed points in the execution of a computer routine.

Internal Symbol

A global symbol located in the module in which it is defined. In a MACRO-10 program, a symbol is declared internal with the INTERN or ENTRY pseudo-op. These pseudo-ops generate a global definition which is used to satisfy all global requests for the symbol. In FORTRAN programs, internal symbols are generated to match the names of SUBROUTINEs, FUNTIONs, and ENTRYs. An internal symbol is similar to a library search symbol; however, it will not cause a module to be linked in search mode.

Job Data Area (JOBDAT)

The first 140 octal locations of a user's address space. This area provides storage for certain data items used by both the Monitor and the user's program. Refer to the DECsystem-10 Monitor Calls Manual.

Label

A symbolic name used to identify a location in a program.

LINK-10
Glossary

Library

A relocatable binary file containing one or more modules which may be loaded in Library Search Mode. FUDGE2 is a system utility program which enables users to merge and edit a collection of relocatable binary modules into a library file. PIP can also be used to merge relocatable binary modules into a library, but it has no facilities for editing libraries.

Library Search Mode

The mode in which a module (one of many in a library) is loaded only if one or more of its declared entry points satisfy an unresolved global request.

Library Search Symbol (Entry Symbol)

A list of symbols that are matched against unresolved symbols in order to load the appropriate modules. This list is used only in library search mode. A library search symbol is defined by an ENTRY statement in MACRO-10 and BLISS-10 and a SUBROUTINE, FUNCTION, or ENTRY statement in FORTRAN.

Linker

A program that combines many input modules into a single module for loading purposes. Thus, it allows for independent compilations of modules. Typically, it satisfies global references and may combine control sections.

LINK-10

Glossary

Link

To combine independently-translated modules into one module in which all relocation of addresses has been performed relative to that module and all external references to symbols have been resolved based on the definition of internal symbols.

Linking Loader

A program that provides automatic loading, relocation, and linking of compiler and assembler generated object modules.

Load

To produce a core image and/or a saved file from one or more relocatable binary files (REL files) by transforming relocatable addresses to absolute addresses. This operation is not to be confused with the GET operation, which initializes a core image from a saved file (refer to GET).

Local Symbol

A symbol known only to the module in which it is defined. Because it is not accessible to other modules, the same symbol name with different values can appear in more than one module. These modules can be loaded and executed together without conflict. Local symbols are primarily used when debugging modules; symbol conflicts between different modules are resolved by mechanisms in the debugging program.

LINK-10

Glossary

Low Segment

The segment of user virtual address space beginning at zero. The length of the low segment is stored in location .JBREL of the Job Data Area. When writing two-segment programs, it is advisable to place data locations and impure code in the low segment.

Main Program

The module containing the address at which object program execution normally begins.

Module

The smallest entity that can be loaded by LINK-10. It is composed of a collection of control sections. In MACRO-10, the code between the TITLE and END statements represents a module. In FORTRAN, the code between the first statement and the END statement is a module. In COBOL, the code between the IDENTIFICATION DIVISION statement and the last statement is a module.

Module Origin

The first location in user virtual address space of the module.

Object Module

The primary output of an assembler or compiler, which can be linked with other object modules and loaded into a runnable program. This output is composed of the relocatable machine

LINK-10

Glossary

language code for the translated module (i.e., link items), relocation information, and the corresponding symbol table listing the definition and use of symbols within the module.

Object Time System

The collection of modules that supports the compiled code for a particular language. This collection usually includes I/O and trap-handling routines.

Offset

The number of locations relative to zero that a Control Section must be moved before it can be executed.

Operating System

The collection of modules that automatically permits continuous job processing by scheduling and controlling the operation of user and system programs, performing I/O, and allocating resources for efficient use of the hardware.

Physical Address Space

A set of memory locations where information can actually be stored (i.e., core memory) for the purpose of program execution.

Program

A collection of routines which have been linked and loaded to produce a saved file or a core image. These routines typically

consist of a main program and a set of subroutine which may have come from a library.

Pure Code

Code which is never modified in the process of execution. Therefore, it is possible to let many users share the same copy of a program.

REL File

One or more relocatable object modules composed of link items (refer to Appendix A).

Relocatable Address

An address within a module which is specified as an offset from the first location in that module.

Relocatable Control Section

A control section whos addresses have been specified relative to zero. Thus, the control section can be placed into any area of core memory for execution.

Relocation Counter

The number assigned by LINK-10 as the beginning address of a Control Section. This number is assigned in the process of

loading specific Control Sections into a saved file or a core image and is transformed from a relocatable quantity to an absolute quantity.

Relocation Factor

The contents of the relocation counter for a control section. This number is added to every relocatable reference within the Control Section. The relocation factor is determined from the relocatable base address for the control section (usually 0 and 400000) and the actual address in user virtual address space at which the module is being loaded.

Routine

A set of instructions and data for performing one or more specific functions.

Segment

An absolute Control Section.

Source Language Program

The original, untranslated version of a program written in a higher-level language (e.g., FORTAN, COBOL, MACRO). Source programs, when translated, produce object modules as their primary output. A program may exist as a source program, an object module, and a runnable core image.

LINK-10

Glossary

**Symbol**

Any identifier (composed of SIXBIT characters) used to represent a value that may or may not be known at the time of its original use in a source language program. Symbols can appear in source language statements as labels, addresses, operators, and operands.

**Symbol Binding**

To resolve references in one module to symbols which are defined (i.e., are assigned a value) in another module.

**Symbol Table**

A table containing entrie for each symbol defined or used within a module.

**Translate**

To compile or assemble a source program into a machine language program, usually in the form of a (relocatable) object module.

**User Virtual Address Space**

A set of memory addresses within the range of 0 to 256K words. These addresses are mapped into physical core addresses by the paging or relocation-protection hardware when a program is executed. On a KA10 processor, the range of addresses is limited by the amount of physical core available to a given user.

LINK-10

Glossary

User's Program

All of the code running uder control of the Monitor in a user virtual address space of its own.

Zero Length Module

A module containing symbol definitions but no instruction or data words (e.g., JOBDAT). Note that the word "length" in this context refers to the program length of the module after loading.