FLEX™ Linking Loader

Table of Contents Page

1. INTRODUCTION

1.1. Preface

This manual describes the use and operation of the FLEX Linking Loader. It is assumed the reader is familiar with the operation of the FLEX 6809 Relocating Assembler, and is comfortable with the concepts of relocation and linkage editing.

Throughout the manual a couple of notational conventions are used which are explained here. Angle brackets ('<' and '>') are often used to enclose the description of a single item even though the description may require several words. Square brackets ('[' and ']') are used to enclose an optional item.

1.2. Terminology

absolute module
: A subprogram assembled by the relocating assembler which makes use of the "ABS" and "ORG" directives. Addresses in these modules are bound to absolute locations at assembly time and cannot be relocated by the linking loader.

file
: A FLEX file containing one or more object-code modules.

file name
: The name of a FLEX file.

loading
: The placement of instructions and data into memory in preparation for execution. This preparation includes linking (the matching of symbolic references and definitions), and relocation of symbols and address expressions.

module
: A general term for either an absolute or relocatable module which has been assembled using the 6809 relocating assembler.

module name
: The name given to a module by the programmer by using the "name" directive of the relocating assembler. If the "name" directive was not used, the module name is the same as the FLEX file name in which it is contained. Therefore, several modules may have the same name. A relocatable output module of the loader may be given a name by use of the "N" option.

relocatable module
: A subprogram assembled by the relocating assembler which does not contain an "ABS" directive. Addresses in a relocatable object-code module are not bound to absolute locations at assembly time.

1.3. Linking Loader Input

Technical System Consultants Inc.'s Linking Loader will accept as input independently assembled, relocatable and/or absolute modules.

Relocatable object-code is generated by the FLEX relocating assembler "RELASMB" in such a way that addresses are not bound to absolute locations at assembly time; this binding of the address fields will be accomplished by the Linking Loader. "LLOAD" binds the addresses at the time the object-code segments are combined to produce an executable program. The binding or adjustment of the address fields is termed "relocation". Relocation is necessary when an instruction expects an absolute address as an operand. The address field of this instruction must be increased by a "relocation constant". The relocation constant is the address at which the module is loaded for execution.

Address fields which do not require relocation are absolute addresses (their values remain the same regardless of the position of the object-code segment in memory). Since the loader does not have access to the source text, it cannot determine if an address field is absolute or relocatable. In fact, it cannot distinguish addresses from data or opcodes. Therefore, the assembler must indicate to the loader the address fields which require relocation. This communication is accomplished through "relocation records" which are appended to the object-code produced by the assembler. Such a file is called a "relocatable module".

Absolute object-code modules, on the other hand, have had all of its address fields bound to absolute locations at assembly time. These modules will not be relocated by the linking loader; rather, they will be loaded where they were "ORGed".

1.4. Linking Loader Output

As output, "LLOAD" produces an object-code module, a load map, a module map, and a global symbol table. The object-code module can be either a relocatable module or an executable program. A relocatable module produced by the loader cannot be distinguished from a relocatable module produced by "RELASMB". Only the loader, however, can transform multiple relocatable modules into an executable program. If absolute modules are included as input to the loader, the output must be executable or an error will result. An executable program cannot be distinguished from programs produced by the absolute assembler "ASMB".

2. INVOKING THE LOADER

The Linking loader accepts as input previously assembled object-code modules and produces as output:

1) A link edited, relocatable, object-code module or
2) a link edited, relocated, executable program.

The command line necessary to invoke the linking loader is as follows:

+++LLOAD <input file list> [+options]

where:

the three plus signs are FLEX's ready prompt, and "LLOAD" is the name of the linking loader command file.

"input file list" is a list of one or more FLEX file names, separated by blanks, which contain the modules you wish to load. The modules will be loaded in the order specified.

"options" is a list of options which must start with a plus sign ("+") and may not contain any embedded spaces. More than one list of options may be specified, but each list must start with a plus sign. Some of the options are single characters while others require an argument. Those that are single letters may be grouped together; for example: +PD. Those that require arguments may either stand alone or be the last of a group of options; for example: +PMA=100, where the "A=100" is an option with an argument. The equal sign is not required in options with an argument. Therefore, "+A=100" is equivalent to "+A100".

Following is a detailed description of each of the valid options.

+A=addr
The binary output of the linking loader is to be executable, and its beginning load address (in hex) is to be "addr". If this option is not selected, the binary output will be relocatable object-code.

+B Do not create a binary file on disk. This is useful when link-editing a program to check for errors before the final program is completed or when obtaining a Global Symbol Table.

+C The output file is to have an extension of CMD. If the "B" option was selected, the "C" option is ignored.

+D Do not print the date at the top of each page when the "P" option is specified. If the "P" option is not specified, the "D" option is ignored.

+F Each file specified in "input file list" is a text file containing a list of one or more file names of object-code modules. The file names must be on separate lines.

+G=go-time command line
After the program has been link-edited and relocated, execution control is passed to the program, starting at the address specified by the transfer address. The "G" option must be the last option specified on the calling command line.

+I Include all internal symbols in the symbol table for symbolic debugging. If the "A" option is also specified, the "I" option has no effect. If the "I" option is not specified, only global symbols are included in the relocatable module.

+L=<library file name> A maximum of five libraries may be specified by repeated use of the "L" option. Libraries are only searched when an executable output program is specified (by using the "A" option). In the following example, an effort is made to resolve externals not found in the user's modules by searching the libraries LIB1 and LIB2.

```
+++LLOAD ECHO.REL +A +L=LIB1 +L=LIB2
```

See the LIBRARIES chapter (chapter 3) for more information concerning the formation and use of libraries.

+M Print the Load and Module Maps. The Load Map provides information about the type of output file (relocatable, executable or none) produced, the length of the resulting object-code module and the transfer address. The Module Map describes the load address and object-code length of each input module.

+N=<module name>
The "module name" is given to the output module of the loader in a manner similar to the "NAME" directive of the relocating assembler. Since the loader does not propogate the module names of the relocatable input modules to the output module, the "N" option must be used to assign a name to a module. If this option is not used, the module name will default to the name of the file in which it is contained. Only relocatable modules can receive module names, so "N" is ignored if "A" is specified. The name is limited to a maximum of 8 characters.

+P Selects pagination of the printed output. The date (if the "D" option is not specified) and a page number are included at the top of each page.

+O=file-spec
Allows specification of an output binary file name. If the "O" option is not specified, the output file name will be "OUTPUT" on the work drive. The extension, if not specified, will be forced to .BIN if the "A=" option was specified, or to .REL if the "A=" option was not specified. Use of the "C" option overrides the above defaults. If a file by this name already exists on the specified drive, a request will be issued to delete the old file.

+S Select printing of the Global Symbol Table. If specified, the linking loader will print each global symbol and its address.

+Y This option overrides the prompt for deleting an existing binary file. In other words, if the "Y" option is specified, an existing binary file with the same name as the one to be created will be automatically deleted without a prompt.

+U Do not print the "unresolved external" message when producing a relocatable output module.

+Z Force zero code suppression. When the loader is being used to produce executable output, this option will cause the deletion of any continuous groups of 16 or more zero bytes. If only relocatable output is being produced this option will have no effect. Although this option has many uses, a good example use of the option is to save disk space when storing compiled FORTRAN programs with large arrays. If a FORTRAN program declares a large array, the assembly code produced by the compiler will reserve space for that array by zeroing out the necessary area. This zeroed area will be included with the binary code and stored on the disk. If some large arrays are being created in the FORTRAN program this could be very wasteful of disk space. The Z option will delete large zeroed out areas, thus making it possible to store more on a disk. One caution must be made when using this Z option and then running the FORTRAN program produced by it. If the program assumes that the arrays are already zeroed out, then using the Z option could cause problems. One way around this is to make sure in the FORTRAN program that no variable is used without having a value assigned to it. Another way is to write a short utility (see appendix A) that will zero all memory, and then execute the FORTRAN program. The former is the recommended approach since it is good programming practice to make assignments or initializations to all variables before using them.

EXAMPLES

1. Link-edit the file "DOING" from drive 1 and call the output file "DONE" which should also end up on drive 1. The code should be executable binary output. Therefore the output file should have a ".CMD" extension.

```
LLOAD DOING.1 +O=1.DONE +CA=0000
```

   This would link edit the file "DOING" from drive 1 and put the output into "DONE.CMD" (+C option) also on drive 1, which is executable binary code whose beginning load address is 0000 hex.

2. Get all the information possible about a link-edit attempt of file "DOING" but do not produce any binary output.

```
LLOAD DOING +BIMS
```

   There will be no binary output from this run (+B option), but all internal symbols will be included in the symbol table (+I option), the Load Map and Module Map will be printed (+M option), and the Global Symbol Table is also printed (+S option). This might be done, if the user was trying to track down some error in a program.

3. LIBRARIES

3.1. Introduction

The linking loader can search up to five libraries when there are externals which cannot be resolved from the user's modules.

A library is a special collection of relocatable modules. When an external cannot be resolved from the user's modules, the libraries are searched in an effort to resolve it. The linking loader will search the libraries in the order specified on the command line. The search for an external can be summarized as follows:

1) Can the external be resolved from the user's modules?
2) Can it be found in the user specified libraries?

When an external is resolved from a module contained in a library, the module is loaded and is then considered a "user" module. Therefore, library modules can reference other library modules.

3.2. Library Generation

The "LIB-GEN" utility is used to create new libraries and update existing libraries. All modules in a library must have a name. The name is assigned to a module by the "NAME" pseudo-op in the relocating assembler or by the "N" option of the linking loader. It is the responsibility of the programmer to ensure that all modules in a library have names. "LIB-GEN" will not accept a module without a name.

The "LIB-GEN" utility is called with a command of the following general form:

```
LIB-GEN O=<old>,N=<new>,U=<updates>,<options>,<deletions>
```

The arguments may be specified in any order.

The argument "O=<old>" specifies the name of an existing library file. The library file must have been created previously by "LIB-GEN". If "LIB-GEN" is being called to create a new library then this should be omitted.

The argument "N=<new>" specifies the name of the new library. If this file already exists, it will be deleted before the new library is written. This argument is not needed when updating an existing library. In this case, "LIB-GEN" will put the new library in a scratch file, delete the old library file, and rename the scratch file, giving it the name of the old library. It is not permitted to omit both the "O=<old>" and "N=<new>" arguments.

The argument "U=<updates>" specifies the name of a file containing modules to add to the library, replacing existing modules in the library if necessary. Up to nine files may be specified by repeating the "U=" argument. See the examples below.

As "LIB-GEN" runs, it produces a report, describing the action that it has taken for each module in the library. The report includes the module name and the file from which it was read (the old library or one of the update files). The options are used to eliminate or shorten this report. If the option "+L" is specified, no report will be produced. If the option "+A" is specified, the report will only contain information about those modules that were replaced, added, or deleted. No information about modules copied from the old library will be given.

The "<deletions>" argument is a list of module names that are to be deleted from the old library. The names may be separated by commas or spaces. If a name is specified that cannot be found in the old library, a warning message is issued. If the "+L" option was specified, no warning is issued.

EXAMPLES

1. Create a new library with the name "BINLIB" containing Modules from the files "ONE", "TWO", and "THREE".

   ```
   LIB-GEN N=BINLIB U=ONE U=TWO U=THREE
   ```

   Since a new library is being created, the "O=<old>" argument was omitted. Note that the "U=" argument was repeated for each update file.

2. Update the library named "BINLIB", adding or replacing records from the file "NEW". Produce an abbreviated report.

   ```
   LIB-GEN O=BINLIB U=NEW +A
   ```

   Since no new library was specified, the new library will be given the name of the old library.

3. Update the library named "BINLIB", deleting the modules named "DIAGONAL" and "TRANSPOSE". Also add new modules from the file "XYZ" and write the new library in the file "NEWLIB".

   ```
   LIB-GEN O=BINLIB U=XYZ N=NEWLIB TRANSPOSE DIAGONAL
   ```

4. MEMORY ASSIGNMENT

4.1. Absolute Modules

The use of the "ABS" directive signals the relocating assembler to produce absolute object-code. Addresses in absolute modules are bound to absolute locations by the assembler by employing the "ORG" directive. Absolute modules cannot be relocated by the loader. In addition to not being "relocatable", absolute modules are not executable because they are not FLEX binary files. The relocating assembler does not have the capability of producing executble object-code; only the absolute assembler and the linking loader have this ability.

4.2. Relocatable Modules

If the "ABS" directive is ommitted from the assembler source code, relocatable object-code will be produced. Relocatable modules do not have their addresses bound to absolute locations by the relocating assembler. This task of relocation is passed on to the loader. The linking loader can also produce a relocatable module by combining one or more relocatable (not absolute) modules and not using the "A" option. Use of the "A" option causes-the linking loader to produce an executable program.

Common blocks are not combined with other modules when the loader produces a relocatable module. Instead, common blocks retain their identity as separate modules and are appended to the resulting relocatable output module. Common areas wil] be linked with the other modules only when producing an executable program.

Relocatable modules can be given module names by the use of the "NAME" directive of the relocating assembler. This name is used when printing the module map. If no name was given to a module by use of the "NAME" directive, the name of the file in which it is contained is printed. When producing a relocatable output module, the linking loader does not propagate any of these module names to the output module. To assign the output module a name, use the "N" option when invoking the loader.

4. 3. Executable Programs

When loading modules to produce an executable program, it is guaranteed that the user modules are loaded first in the order specified on the command line. Common areas are loaded after the last module specified on the command line. Libraries are loaded after the last common block, or after the last user module on the command line if there are no common blocks.

In this type of file, the binary is stored in a record format where each record has it own load address and contains the object-code of one module. The user can specify the initial program load address (IPLA) of the first record (module) by use of the "A" option. Successive modules are loaded in the order specified on the command line and immediately follow the previous module. In other words, the program is loaded continuously starting at IPLA.

The following memory map illustrates how the individual modules are placed in relation to other modules. The module numbers are the order in which they appear on the command line; "m" is the last module specified. Common blocks 1-x and library modules 1-n which are loaded to complete the program are also represented.

```
     IPLA --> module 1
              module 2
                 .
                 .
                 .
              module m
              common 1
              common 2
                 .
                 .
                 .
              common x
              library 1
              library 2
                 .
                 .
                 .
              library n
```

5. LOAD AND MODULE MAP

5.1. Load Map

The "M" option controls the printing of the module and load maps. When selected, the load map will provide information as to the type of output produced, the length of the resulting output object-code module, the number of input modules, and the transfer address.

5.2. Module Map

Use of the "M" option also selects printing of the module map. The module map describes the load addresses and object-code length for each of the input modules. The load addresses for each of the segments (text, data, and bss) take on different meanings depending on the type of output file produced.

5.2.1. The Module Map of a Relocatable Module

When producing a relocatable module, both the relocating assembler and the linking loader do not "bind" or tie addresses to absolute locations; they are made relative to a base. The following example assembled by the relocating assembler will illustrate this point.

```
 1                                  EXT    PDATA
 2
 3 + 0000 8E   0010     START       LDX    #MSG1      POINT TO 1ST MESSAGE
 4 +>0003 BD   0009                 JSR    SUB1       PRINT IT
 5 + 0006 8E   001A                 LDX    #M1SG2     POINT TO 2ND MESSAGE
 6 + 0009 BF   002D     SUB1        STX    MSGADDR    SAVE MESSAGE ADDRESS
 7 X 000C BD   0000                 JSR    PDATA      PRINT A MESSAGE
 8   000F 39                        RTS               ALL DONE
 9
10   0010 4D 45 53 53   MSG1        FCC    'MESSAGE 1',0
11   0014 4D 45 53 53   MSG2        FCC    'MESSAGE 2',0
12
13   0024 00   000000               RMBB   9
14   002D 00   00         MSGADDR   RMB    2          MSSGE ADDR SAVE AREA
15
16 +                               END    START
```

This example shows that the address of the first instruction of 'PDATA' starts at '0000'. When this module is linked with other modules, each of the addresses will be added to a base to give the address for the output file.

As an example, three modules called PDATA, MAIN and CNTCH were asssembled separately. Each was assembled with a starting address of '0000'. The three modules were then linked to produce a relocatable module and the following load and module maps.

```
Errors Detected.
  CNTCH in 1.MAIN.REL, unresolved.
  PDATA in 1.MAIN.REL, unresolved.

Load Map.
  No output file produced.
  Program length: 0056
  Transfer address: 0000

Module Map

 MODULE NAME      ADDR    SIZE

1.MAIN    .REL    0000    0009
1.CNTCH   .REL    0009    001E
1.TEST    .REL    0027    002F
```

The program length is the sum of the lengths of all the modules. The transfer address is seen to be in the module "MAIN".

5.2.2. The Module Map of an Absolute Program

When creating an absolute, executable program, the user can specify the initial program load address (IPLA), and all address expressions are relocated relative to the IPLA. The same modules used in the relocatable example above were linked with an IPLA of $100 to produce the following load map:

```
Load Map
  Absolute file produced.
  Program length:  0056
  Transfer  addr:  0100

Module Map

 MODULE NAME     ADDR  SIZE

1.MAIN   .REL    0100  0009
1.CNTCH  .REL    0109  001E
1.TEST   .REL    0127  002F
```

The transfer address has been set to the IPLA and execution will begin with the module "MAIN".

6. MISCELLANEOUS

6.1. Transfer Address

A transfer address is the location at which execution is to start when the program is invoked. Use of the 'END' directive in the relocating assembler can be used to indicate a transfer address.

Only one relocatable module to be included in a program should contain a transfer address. If more than one module has a transfer address, the linking loader will accept the first one encountered and ignore all others.

7. ERROR MESSAGES

Absolute module table overflow
: The number of absolute input modules cannot exceed 50. Combine several of the modules into one and re-assemble.

Attempt to redefine entry point <entry point name>
: Two entry points (global symbols) exist with the same name. Entry point names must be distinct. One of the globals will have to be renamed, its module re-assembled, and linked.

Can't go. Command line overflow
: When the "g" option is specified, the linking loader builds a multiple command line: the first command is the linking loader invocation as supplied by the user, and the second is the invocation of the executable output file of the linking loader. Since a command line is limited to 128 characters, it is conceivable that both commands may not fit. If the "go-time" command cannot be shortened, it will be necessary to invoke the executable output file separately after the linking process has completed.

Error in load address
: The address following the "A" option is not a valid hexadecimal address. See the "A" option for complete specifications.

<entry point name> in <module name> unresolved.
: An unresolved external has been detected by the loader. If the output of the loader is to be a relocatable module, this condition is probably expected and can be considered a warning. Use of the "u" option will suppress this warning message. However, when producing an executable program, this is an error.

<file name> - contains assembly errors.
: An input file to the loader contained errors when assembled. Correct the errors in the source code and re-assemble.

<file name> - illegal file specification
: The file name is not a valid FLEX file specification.

<file name> - illegal input file.
: The file name mentioned is not a relocatable module as built by the relocating assembler or the loader.

<file name> invalid library specified.
: The file name mentioned is not a valid library as built by the library generator program. All libraries must be created using the library generator.

<file name> - not found
The file specified could not be located.

FLEX ERROR #<error>
A FLEX system error has been detected. Consult the FLEX manual for definitions of the errors.

'Go' denied.
In order for the loader to execute the user's program, all of the following conditions must hold: an output file be produced and must be executable, the program must have a transfer address, and the program must be error free.

'Go' pending. Warnings detected. Still go (y or n)?
Warnings were detected by the loader while preparing a program for execution. The operator must answer with "Y" or "y" if these warnings are to be ignored and execution initiated; any other response will cause execution to be aborted.

Insufficient memory
There is insufficient user RAM to support the linking loader. The minimum amount of RAM capable of comfortably supporting the linking loader is 24K.

Linking error at $<address> relative to <module name>
When adding the address of an external to a field-being linked, the carry bit was set. "Module name" is the offending input module, and "address" is the offset of the link field from the base of the module. This message is merely a warning to the user that an error condition may have occurred. In some cases, this message would be expected. Consider the following module:

```
                                    NAME    EXTTEST
                                    EXT     EXTRN
X 0000 8E   FFFF     START          LDX     #EXTRN-1
X 0003 C6   FF                     LDB     #EXTRN+255
                                    END
```

The loader would report the following if the address of EXTRN. was non-zero:

```
Errors Detected.
  Linking error at $0001 relative to EXTTEST
  Linking error at $0004 relative to EXTTEST
```

This first message is informing us that the sum of the address of EXTRN and -1 cannot be held in 2 bytes (the carry bit was set). However, the address loaded into the X register when this instruction is executed would be exactly as we would expect: one less than the address of EXTRN. A similar situation would occur when we force one-byte linking as in the second instruction of the

above module: the sum of the least significant byte of the address of EXTRN (if non-zero) and 255 will not fit into one byte (the carry bit was set).

Module name table overflow
: All available space in the module name table has been exhausted. Link a subset of the input modules to produce a larger relocatable module.

<module name> not loaded. Memory conflict with loader
: There is insufficient memory available to hold the entire module. To correct this situation, break the module source code into several smaller modules and assemble separately. This message can also be generated when the "G" option is specified, and an absolute module was to be loaded over the loader. In this situation, create an executable program without using the "G" option and invoke it directly from FLEX.

Symbol table overflow
: No more room exists in the global symbol table to insert symbols. The symbol table maximum capacity is 341 symbols. Symbols which are not required to be global should be made internal in order to free up more room in the table.

Too many Common Blocks.
: The number of unique common modules cannot exceed 50.

Too many input modules.
: The number of input relocatable, absolute, and common modules exceeds 100. Link a subset of these modules and produce a larger relocatable module. This single large module can then be linked with the remaining ones.

Too many libraries specified.
: More than five libraries were specified on the command line. See the chapter LIBRARIES for further information.

Unknown option ignored - <option>
: An invalid option was specified on the command line. Consult "INVOKING THE LOADER" chapter for valid options.

8. APPENDIX A

The following is a short utility that will zero memory. It is useful when the +Z option of the loader was used but the program assumes its variables have been initialized to zero. The source for this utility is 6809 assembler code.

```
*
*   This command zeros all of RAM memory
*     syntax: ++ zmem
*
          ABS
          ORG    $C100
MEMEND    EQU    $CC2B
FLEX      EQU    $CD03
*
Z_MEM     LDX    #0          set starting address
          LDD    #0          set fill value
10        STD    ,X++        fill memory
          CMPX   MEMEND      end of memory?
          BLO    10B
          JMP    FLEX        all done exit
*
          END    Z_MEM
```