Quelo™

LINKER AND OBJECT LIBRARIAN MANUAL

TABLE OF CONTENTS

Contents.

SECTION 1
Hands On

Introduction:

The linker program is used to bring together separately assembled modules to make up one large program. Code is relocated to absolute memory addresses and references between modules are resolved. Individual modules lose their identity in the linking process. Operation of the linker is controlled by a link specification file. The end result of linking is a HEX load module containing load address information and the data to be loaded.

The object librarian program is used to bring together separately assembled modules into one file for later linking. No relocation or linking is performed. Each module in a library file retains its identity separate from all others. Operation of the object librarian is controlled by a library specification file. Object libraries are a convenience feature and are not required for use of the assembler package.

The linker program will be demonstrated using sample files which are supplied on your distribution disk.

In addition to this manual, there may also be some supplemental documentation. This may be in printed form and/or in a text file on your distribution disk. Look for a file named "READ.ME" on the disk.

IMPORTANT — Read the supplemental information first. It may contain corrections or additions to the manuals. It will contain information specific to your operating system.

Working Disk Preparation:

Do NOT try the samples directly on your distribution disk. First copy the contents of the distribution disk to a working disk and then store your distribution disk in a safe place. If you should damage your distribution disk, a replacement may be had from Quelo for the cost of materials, shipping and handling.

Program Execution:

The command used for program startup may vary from one operating environment to another. In this manual, commands suitable for CP/M-80 (CP/M 1.4, 2.2, 3.0, MP/M x.x, etc.) will be used for illustration purposes. These commands will also be applicable to the 8086 and 68000 versions of CP/M and MP/M and also MSDOS and PCDOS on the IBM PC. For other environments, the program startup commands may vary, especially in the area of input/output file specification. Supplementary documentation should be consulted for the specific operating environment.

Execution Date and Time:

When available, the current date and time are placed in the listing file produced by the program. The source of the date and time is system dependent and is covered in the supplemental documentation.

Running the Linker:

The simplest command line for linking consists of the linker name followed by the name of the link specification file. In this case some options will also be included in the command line. Note that the files LT30.A68, LT31.A68 and LT32.A68 must first be assembled to produce LT30.LTX, LT31.LTX and LT32.LTX, respectively. The discussion of this example covers the linker reports: module summary, symbol table, cross-reference and memory load map.

Enter the Following Commands:

A68K -L LT30	; assembly command
A68K -L LT31	; assembly command
A68K -L LT32	; assembly command
Read assembler source,	LT3x.A68
Produce listings,	LT3x.PRN
Produce object files,	LT3x.LTX
QLINK -LISX LT33	; link command
Read link specification,	LT33.LNK
Read object files,	LT30.LTX
	LT31.LTX
	LT32.LTX
Produce listing file,	LT33.LST
Produce hex load file,	LT33.HEX
Produce symbol file,	LT33.SYM
QSYM -IBM LT33	; report command
Read symbol file,	LT33.SYM
Produce report file,	LT33.RPT

The "L" option instructs the assembler and linker to retain lower case in symbols. The "I", "S" and "X" options instruct the linker to place module summary, symbol table and cross-reference data in the ".SYM" output file. The "I", "B", and "M" options instruct the symbol report program to generate three reports: module summary (IDNT directive information), combined symbol table and cross-reference, and a memory load map.

Examine the listing file, LT33.LST. It should appear similar to that provided in Appendix A, with the possibility of minor differences due to changes in the listing format or due to editing necessary to make the listing fit the basic format of this manual.

The IDNT directive establishes a name for the program. This name is placed in the "S0" header record of the hex load file. The LINK directives determine which modules are to be linked and the order in which they are linked. The position of the LINK directives in the link specification do NOT have any bearing on where code from the modules is positioned in memory. Code is located by section alone, unless it was assembled under an ORG directive, in which case the position of the code was determined at the time of assembly.

The linker ORG directive, like the assembler ORG directive, is used to establish an absolute memory location. However, the linker SECTION directive is unlike the assembler SECTION directive in that it only reserves space for the specified relocatable section or sections. The assembler SECTION directive selects one of the 16 relocatable program counters.

To the linker, "SECTION 2" means reserve space for all of the code for section 2 from all of the modules to be linked. In this example, modules LT31 and LT32 both contribute code to section 2. This can be seen in the module listings and the memory load map in Appendix A. Note in the load map that the order of the section 2 blocks from these two modules was determined from the order of the LINK directives in the link specification.

Section 0 was located at address $4000. Section 4 was located immediately following the revision list data (to be discussed later). Section 3 was located immediately following section 4, etc. To reiterate, code is primarily located by the use of ORG and SECTION directives and the order of the LINK directives affects module code location within a section.

The REVLIST directive causes the linker to place module name, version and revision information into the program being constructed. REVLIST may be considered to represent a 17th relocatable section whose data comes from the IDNT directives of all the modules being linked. With this data included in the final program, there can be no doubt as to the program configuration.

Examine the hex load file, LT33.HEX. The sample shown in Appendix A has been edited to include commentary information. Data in the hex file h are generated as modules are linked, so that in general the hex record load addresses will not be in ascending sequence.

The END directive has a symbol in the operand field. This specifies the value of that symbol as the starting point of the program. The address of this symbol appears in the "S9" end-of-file HEX record. The linker also generates an internal symbol "<<<<<<<<" which appears in the symbol table and load map to indicate the starting point of the program.

For the sake of experiment, try changing the order of LINK and SECTION directives and observe the effect on the load map.

Using the Linker:

Section 2 describes the overall operation of the linker.

Section 3 of this manual elaborates on the previous discussion.

Section 4 of this manual describes each of the linker directives in detail.

Section 5 of this manual covers the HEX file utilities. SPLIT is used to split a Motorola S-Record HEX file into separate Intel HEX files for even and odd address bytes for 8 bit wide ROMS in systems with a 16 bit wide data bus. IMAGE is used to create a memory image binary file from a HEX load module. IMAGE would be used to create an executable file for CP/M-68K for instance.

Section 6 of this manual describes the object librarian and its use.

Appendix A contains listings referenced in the preceding discussion.

Appendix B covers the restrictions and limitations on the linker.

Appendix C describes the relocatable object format.

Appendix D describes the various HEX load formats available from the linker.

Appendix E explains the linker error messages.

SECTION 2
Linker Overview

Introduction:

Although substantially different in appearance from the Motorola linker, QLINK performs a very similar function. Code from each of the modules being linked is grouped by relocatable section. Each of the 16 relocatable sections may be independently located in memory. This facilitates separation of RAM and ROM spaces and separation of code, initialized data and uninitialized data spaces.

The link specification is patterned after assembly language source code. Although the linker does not assemble machine instructions, it does process such directives as ORG, DC, DCB, DS, EQU, IFEQ, etc., the same as the assembler. Symbols may be defined by the SET and EQU directives or as labels. All symbols defined in the link specification will be public and may be referenced as externals in the modules being linked. Expressions used as operands to the linker directives may contain references to symbols defined as public in the various modules being linked.

This scheme provides considerable flexibility and a language already familiar to the programmer, rather than the cryptic commands found in most linkers. The link specification may even be generated using the macro pre-processor, M68K. The minimal link specification would require LINK directives to select the modules for linking and an ORG directive to locate the program in memory.

The linker does not generate a memory image of the final program and does not load the program in memory. The IMAGE program may be used to create a memory image of the program. As object modules are processed, absolute HEX object records are generated directly. Consequently, the HEX object records will not be arranged in ascending load address order. This is not a problem, but just a statement of fact. Memory space is required only for symbol table storage. No HEX object code is generated for uninitialized space.

The Linker End Product:

The linker produces a HEX object file of Motorola S-records. Optionally, Intel style hex records and Mostek hex records may be had. See Appendix D for descriptions of these formats.

The linker also produces a listing file comparable to that of the assembler. Command line options may be used to write symbol table data and cross-reference data to a separate file for subsequent generation of module list, symbol table, cross-reference and memory map reports.

The Linking Process:

Since object code is accumulated by section, the linker must interrogate all object modules for section sizes before it can allocate memory and begin the linking process. Therefore, the first order of business is to determine what modules are to be linked. The LINK directive is used to specify object files for loading.

The smallest unit that can be loaded by the linker is known as a module. A module is the relocatable-linkable object code produced by one execution of the assembler. A library is a collection of modules gathered together by the object librarian program. One application for a library is to keep a collection of utility modules which might be useful in a number of different programs. Another application is to collect all modules for a given program in one file.

Several formats are available for LINK directive operands. The type of the specified file (module or library) and the specific operand syntax used determine what action the linker takes. The following actions are available:

- Load a module file.
- Load a module file only if it is needed to satisfy undefined symbols.
- Load all modules in a library file. This would be useful in the case of a library of modules for a given program.
- Load modules from a library file according to a list of specified module names. Library files have a table of module names at the beginning to facilitate searches. Associated with each module name is an offset to the location of the module in the library file.
- Load modules from a library file only if they are needed to satisfy undefined symbols. This action is usually known as a library search and would typically be used for a library of utility modules. Library files have a table of globally defined symbols at the beginning to facilitate searches. Associated with each symbol is an offset to the location of the module in which the symbol is defined. The undefined symbol search process repeats each time another module is selected from the library. This iterative search process eliminates the need for careful ordering of the modules in a library and insures that all required modules will be loaded.

Refer to Section 4, Linker Directives, for the LINK directive syntax. Note that LINK directives are processed as encountered in the link specification file. If a new module is loaded after a library search and the new module needs a module (not loaded) from the library previously searched, it will be necessary to use another LINK directive to initiate another library search. Thus, the order of appearance of link directives may be important in a particular application.

During the process of determining which modules are to be linked, a symbol table is constructed from the symbols for each module and the section sizes from each module are accumulated. The order in which the modules will be linked is the same as the order of appearance of the LINK directives. In the case of modules included from library searches, the order is unspecified.

At the end of this first phase of linking the total size for each of the 16 relocatable sections will be known. Given this information, the linker can then allocate memory based on the ORG and SECTION directives in the link specification. At this point, the symbols will be of type constant, absolute address or relocatable address.

The next phase of linking is to process the link specification in two passes, corresponding to the two passes of the assembler. The first pass is needed to deal with forward referenced symbols. References in the link specification to symbols defined in the modules being linked are permitted, but references to relocatable symbols are restricted. For instance, in "DCB exp1,exp2", exp1 (the block size) may not be relocatable but exp2 may be relocatable. No expression that could affect memory allocation may reference relocatable symbols.

At the end of this first pass, all relocatable symbols are assigned their final absolute memory addresses. The second pass is then used to output hex code for such directives as DC, DCB and possibly, REVLIST.

At the end of this second pass, the final phase of linking begins. It is this phase that processes the code portions of the modules being linked. External references are resolved and absolute object code is output in the form of hex records for subsequent loading into the target 68000 system.

Memory Allocation:

The ORG directive is used to establish absolute addresses. As in the assembler, it assigns a value to the program counter.

SECTION directives follow ORG directives to specify where the section is to be located in memory. The program counter is incremented by the size of the particular section specified in the directive. A section may be made to immediately follow another section by not preceding the SECTION directive with an ORG directive or by specifying several sections with a single SECTION directive.

The link specification may include DS directives to reserve space. DC and DCB may be used to create data in 68000 byte order. DB, DW, DL and DUP may be used to create data in Z80 byte order. Labels may also appear in the link specification. Such labels will have a type of ''absolute address'' and will be treated as any other global symbol.

SET and EQU directives are also available for defining symbols in the link specification. The IFxx and ENDC directives, as in conditional assembly, may be used for conditional linking.

All symbols defined in the link specification are available when the object code linking process actually begins. Therefore, external references in the various object modules may be satisfied by symbols defined in the link specification.

In Summary:

The linker performs the following agenda:

Initial Phase.
- Process LINK directives.
- Build list of modules to be linked.
- Build symbol table from modules.
- Accumulate relocatable section sizes.

Middle Phase.
- Ignore LINK directives.
- Pass 1.
  - Process ORG, SECTION, DS, DC, DCB, IDNT, REVLIST, SET, EQU and IFxx directives.
  - Allocate memory.
  - Add to symbol table.
  - Relocate symbols to absolute addresses.
- Pass 2.
  - Generate code for DC, DCB and possibly the REVLIST directive.

Final Phase.
- Read object modules.
- Resolve external references.
- Output HEX load file.
- Output module, symbol table and cross-reference information if requested.

SECTION 3
Linker Usage

Command Line to Invoke the Linker:

QLINK <filename>	; short form
QLINK =<filename>	; intermediate
QLINK <hex>,<listing>,<symbol>=<link spec>	; full form

-------------------------------- I/O Specification Equivalence --------------------------------

XYZ	XYZ.HEX,XYZ.LST,XYZ.SYM=XYZ.LNK
,CON:=XYZ	NUL:,CON:,NUL:=XYZ.LNK
=XYZ	XYZ.HEX,NUL:,NUL:=XYZ.LNK
XYZ.OBJ=XYZ.SRC	XYZ.OBJ,NUL:,NUL:=XYZ.SRC
,CON:=CON:	NUL:,CON:,NUL:=CON:
B:,C:,D:=A:X	B:X.HEX,C:X.LST,D:X.SYM=A:X.LNK
.OBJ,.PRN=X	X.OBJ,X.PRN,NUL:=X.LNK

Note that these examples apply to "CP/M like" environments. Supplementary documentation deals with issues specific to the operating system under which QLINK is being used.

Command Line Invoked Options:

The command line may consist of several items, one of which must be the I/O specification. Command line items are separated by spaces, implying that the I/O specification may not have embedded spaces. Items beginning with a minus sign are interpreted as option selections.

The "-B" option has the same effect as selecting both the "-S" and "-X" options.

The "-E" option causes local labels from linked modules to be retained in the symbol table for inclusion in the memory map. These labels will not otherwise be used by the linker.

The "-Hx" option selects the format for the HEX object output. The "x" may be "S" for Motorola S-records, "I" for Intel HEX records or "M" for Mostek HEX records.

The "-I" option causes module summary information (from IDNT directives) to be written to disk for subsequent processing by QSYM. Whether or not this file is actually written depends on the input/output specification in the command line. The same disk file is used for the "I", "S" and "X" options. From this information, the QSYM program produces a list of all modules linked including module name, version, revision and description.

The "-L" option causes the linker to distinguish between upper and lower case letters in user-defined symbols. Upper and lower case letters will still be considered the same in directives.

The "-S" option causes the symbol table to be written to disk for subsequent processing by QSYM. Whether or not this file is actually written depends on the input/output specification in the command line. The same disk file is used for the "I", "S" and "X" options. The QSYM program formats a symbol table report and/or a cross reference report and/or a memory map.

The "-T" option causes truncation of long symbols to 8 characters, rather than the usual 31 characters.

The "-V" option is used to put an ASCII formfeed character at the beginning of the listing output.

The "-X" option causes cross reference data to be written to disk for subsequent processing by the QSYM program. Whether or not this file is actually written depends on the input/output specification in the command line. The same disk file is used for the "I", "S" and "X" options.

Command Line Examples

"QLINK -S XYZ" specifies that a symbol file is to be produced, that the link specification file is "XYZ.LNK", that the listing file is "XYZ.LST", that the hex object file is "XYZ.HEX" and that the symbol file is "XYZ.SYM".

"QLINK -LE XYZ"specifies the same link specification, listing and hex object files as the previous example, but specifies the "L" and "E" options. In this case there is no symbol file since the "I", "S" and "X" options are all absent. Note that the options may be concatenated following a single minus sign. "QLINK -L XYZ -E" would produce the same effect.

"QLINK ,CON: = XYZ"suppresses the object and symbol files, sends the listing to the console and specifies "XYZ.LNK" as the link specification file. Here, the console is the "standard output" device of the C programming language.

"QLINK ,CON: = CON:" suppresses the object and symbol files and uses the console for both source input and listing output ("standard input" and "standard output" in C). This means that the linker may be used interactively, with certain limitations: The linker makes three passes over the link specification input. So, unless the same information is entered for each pass, errors will probably result, especially in the area of user-defined symbols. Also, the listing is produced during pass 2, so that no feedback will be seen until the first "END" directive is entered.

General Syntax Rules:

The link specification syntax is field oriented, with spaces and/or tabs serving as field delimiters. Spaces and tabs will be referred to as blanks in the following discussion. The implication of this is that, except for strings and character constants, blanks may not appear in expressions.

The label field normally begins in column 1. For labels beginning in column 1, a terminating colon is optional. For labels preceded by blanks, a terminating colon is required in order to prevent the linker from confusing the label field with the operation field.

The second field is the operation field, where linker directives appear. If a symbol is present in the label field, a colon or blank or both must precede the operation field, if the operation field is present. Note that an asterisk may appear in the operation field to indicate that the rest of the line is commentary (i.e. in this context, the asterisk may be considered a linker directive which means "comment").

In another context the asterisk may be considered a symbol to represent the program counter in expressions. And last, but not least, the asterisk serves as the multiply operator in expressions. This triple duty for the asterisk is consistent with the assembler, A68K. A semicolon may be used in place of the asterisk to indicate comments.

Linker directives are described with examples in Section 4, Linker Directives. Although some directives are partially explained in this Section, Section 4 should be referenced for complete details.

A label may appear on a line by itself. Anything following the label field will be assumed to be a directive, unless it is begun with an asterisk to indicate commentary. An asterisk as the first non-blank character on a line indicates that the entire line is commentary.

The third field is for any operands of directives. For those directives not requiring operands, anything appearing in the operand field will be treated as commentary. The fourth field is always treated as commentary.

User-Defined Symbols:

Symbols begin with a letter, a period or an underscore. Additional characters may be letters, digits, periods, dollar signs and underscores. Upper and lower case letters are not distinguished, unless the "-L" option is in effect. Symbols may retain up to 31 significant characters. If the "-T" option is in effect, long symbols are truncated to 8 characters.

Symbols in the link specification may represent constants or absolute addresses. The implications of this symbol typing are discussed under Expression Evaluation.

Symbols may be defined by use in the label field of certain directives. In the case of the SET and EQU directives, the symbol value and the symbol type are derived from the expression in the operand field of the directive. The expression may NOT contain forward references, but may contain references to non-relocatable global symbols defined in the modules being linked.

Note that a size designation may be appended to the SET and EQU directives to govern the way character constants appearing in the expression are handled. Character constants are left justified according to size and zero filled. This issue is further discussed under Constants.

The IDNT directive defines a symbol as the name of the module being assembled. The symbol itself appears in the label field. See Software Configuration Tracking later in this section for the purpose of IDNT.

Other symbols appearing in the label field are assigned the type and current value of the program counter. From now on such symbols will be referred to as labels. Labels are used to reference specific points in a program or data area and have a type of "absolute address".

The symbol table includes symbol type information and an indication as to how the symbol was defined (SET, EQU, label, etc.). All symbols defined in the link specification are global, and must not have the same name as symbols made public by the XDEF assembler directive in the modules being linked.

Program Counters:

Program counters are selected and initialized by the OFFSET and ORG directives. The OFFSET program counter is of type "constant". The ORG program counter is of type "absolute address". An asterisk may be used as a symbol in expressions to represent the current value of the currently selected program counter.

OFFSET is similar to ORG, only no object code related features may be used ("SECTION", "REVLIST", "DC" and "DCB"). "DS" is used to increment the offset counter. OFFSET is handy for generating offset labels for use in the d(An) and d(An,Ri) addressing modes available in the assembler.

All sections and all constants of size word and size long (except when defined by the Z80 byte order directives) are required to reside at an even address. QLINK forces such items to an even address by insertion of a zero byte when necessary. Note that "DS 0" may be used to force an even address. If the program counter is already at an even address, no action is taken.

When a label appears on the same line as a directive, value assignment is postponed until after even address adjustment occurs. This assures that a word or long data item label falls on an even address. However, there will be no even address adjustment when a label appears on a line by itself. If the program counter value is odd, a lone label will be assigned that odd value. The label would then not be useful if the next line contained a word or long data item. QLINK flags this as an error.

Constants:

Constants, other than decimal, are preceded by a special character to indicate the format of the constant as follows:

character	- '		
hex	- $	binary	- %
octal	- @	decimal	- 0 thru 9

Character constants are enclosed in single quotes. Two adjacent single quotes are used to force one quote into a constant. The number of characters that may appear in such a constant is dependent on the context in which the constant is used. For instance, the character constant in "DC.B 'x' + 128" is limited to 8 bits, or one character. Note that 'x' in this example is a character constant and NOT a string because of the " + 128". Strings have unlimited length. In DC.W the character constant would be limited to 16 bits or two characters. Likewise, in DC.L the limit would be 32 bits or 4 characters.

One or two characters are left justified in 16 bits and three or four characters are left justified in 32 bits. For DC.B, one character is NOT left justified in a word. More characters than the limit results in an error message.

The Z80 byte order directives, DB-DW-DL-DUP, do NOT left justify character constants.

In the expressions for the SET, EQU and IFxx conditional assembly directives, the character constant justification is in a word for two or less characters. For three or four characters, the justification is in a long word. For more than four characters, an error message is produced. A size designation may be appended to SET, EQU and IFxx to force justification to byte, word or long sizes.

Strings:

Strings are delimited by single quotes. Two adjacent single quotes are used to force one quote into a string. For instance '''' is a string containing one single quote mark. Strings may be used as operands to the DC directive. Such strings are padded with zeros if the size designated with the DC directive is word or long. Thus, the total number of bytes generated for the string will be some multiple of 2 or 4 for word and long sizes respectively.

No padding takes place for the byte size. The default size for the DC directive is word (DC.W). The Z80 byte order directives, DB-DW-DL-DUP, do NOT pad strings with zeros for boundary alignment.

Operators for Expressions:

+	add	
-	subtract	
*	signed multiply	
/	signed divide	
%	signed modulus	
&	bitwise AND	
| !	bitwise OR (both allowed)	mm
^	bitwise exclusive OR	
<<	logical shift left (zero fill)	
>>	logical shift right (zero fill)	
-	unary minus (two's complement)	
~	unary NOT (one's complement)	

mm - "!" allowed for consistency with the Motorola assembler

Operator Precedence:

unary not, minus	— highest precedence
shift left, right	
and, or, exclusive or	
multiply, divide, modulus	
add, subtract	— lowest precedence

Precedence may be altered in the usual way by using parentheses.

Expression Evaluation:

Except within character constants, embedded blanks are not allowed in expressions. Blanks serve as field delimiters. Expressions have a type attribute to facilitate evaluation and error detection. By the time expression evaluation takes place, all relocatable symbols from the various modules being linked have been assigned absolute addresses.

Symbol and expression "type" attributes are:

c - constant
a - absolute address
r - relocatable address

Expression Result Type Determination:

The type and class attributes are processed separately. The following table is used to determine the result type for each binary operation. A minus sign indicates an error situation and a result type of c.

operand types left	operand types right	result types other	result types -	result types +
c	c	c	c	c
c	a	–	–	a
c	r	–	–	r
a	c	–	a	a
a	a	–	c	–
a	r	–	c	–
r	c	–	r	r
r	a	–	c	–
r	r	–	c	–

The following table is used to determine the result type for each unary operation. A minus sign indicates an error situation and a result type of c.

operand type	result types ~	result types –
c	c	c
a	–	–
r	–	–

If it is necessary to use operators other than + and - with absolute addresses, base symbols can be subtracted to convert to a constant value. An absolute base symbol would be defined by following "ORG 0" with a label, "BASE".

Expression Usage Restrictions:

Expressions may appear as parameters to various linker directives. In some cases, expressions may not contain forward references. References to relocatable symbols in the modules being linked will result in error messages in most expression applications. The exceptions are in the DC and DCB directive data operands and in the starting address operand of the END directive. Expressions will also be found in the object modules being linked, but in a binary, postfix format. The following table summarizes the restrictions:

Exp. Use	**Fwd. Ref.**	**Valid Type**	**Value Range**
DB,DW,DL,	ok	c,a,r,	unsigned 32, 16, 8
DC,DCB	ok	c,a,r	unsigned 32, 16, 8
DCB size	no	c	0 to 32767 *
DS	no	c	0 to 32767 *
DUP size	no	c	0 to 32767 *
DUPB,W,L	ok	c,a,r	unsigned 32, 16, 8
END	no	c,a,r	
EQU	no	c,a	
IDNT	ok	c	0 to 32767 *
IFxx	no	c,a	
LLEN	no	c	79 to 132
OFFSET	no	c,a	
ORG	no	c,a	
PLEN	no	c	24 to 120
SECTION	no	c	0 to 15
SET	no	c,a	
SPC	no	c	0 to 120

* Negative result treated as 0

Expression Result Truncation:

Truncation from 32 bit to 8 and 16 bit quantities is required in various situations. For example: "DC.B PDQ+X2" requires unsigned truncation of the result of the expression, "PDQ+X2", to 8 bits.

Signed truncation to byte and word objects requires the expression value to be in the ranges (-128..127) and (-32768..32767) respectively. Signed truncation applies to the 68000 absolute short addressing mode and to the displacements associated with other addressing modes.

Unsigned truncation to byte and word objects requires the expression value to be in the ranges (-128..255) and (-32768..65535) respectively. Unsigned truncation applies to immediate operands and data for the DC and DCB directives.

Conditional Linking:

Conditional linking is implemented via the IFEQ, IFNE, IFLT, IFLE, IFGT, IFGE and ENDC arithmetic test directives. The value of a single expression is tested for the condition specified. For example, the IFLT test result is "true" if the expression value is less than zero. Note that the expression may not contain forward references or relocatable symbols.

Character constants appearing in the expression are handled as described previously under Constants. For example, "IFNE.W" will cause any character constants in the expression to be left justified in 16 bits, with zero filling.

Unlimited nesting is supported for conditional linking. Each IFxx directive must be matched by a corresponding ENDC. When an IFxx directive test result is "false", all subsequent source input lines will be ignored until the matching ENDC is encountered. All intervening IFxx and ENDC directives are recognized to make the nesting work correctly.

Additional information on the conditional linking directives may be found in Section 4, Linker Directives.

Relocation and Linking Considerations:

Items to be resolved by the linker may be byte, word and long objects. All objects to be resolved are evaluated in 32 bit arithmetic by the linker. Byte and word objects are then truncated to 8 and 16 bits respectively. The truncation may be signed or unsigned, depending on the nature of the object.

None of the 16 relocatable sections are assigned any special purpose. The user may allocate any section for code, data, ROM, etc. The linker groups all of the code for a given section together. The final location for the code of each section is established in the link specification by the combined use of ORG and SECTION directives.

Symbols declared external (XREF) to a module must be defined and tagged as global (XDEF) in another module. Also, all symbols defined in the link specification are global and symbols defined in some module being linked may be referenced in the link specification. When there is no global definition for a symbol which is declared external, the linker will produce an error message. A given symbol may be tagged global in only one place. More than one global definition of a symbol will result in an error message.

Note that all symbols defined in the link specification are automatically global and may be referenced by any module. A given symbol may be declared external in more than one module. When symbol type information is specified in an external declaration, the type must agree with that specified in the definition of the global symbol. Otherwise, an error message will result.

All global symbols must be unique in a given program link. Two global symbols of the same name, even if defined in different sections, will be treated as duplicate global definitions, and an error message will result. Named relocatable sections (COMMON) are NOT supported in QLINK at the present time.

Software Configuration Tracking:

The IDNT directive in A68K and QLINK and the REVLIST directive in QLINK may be used together to insert module version and revision information into your final program. That way, inspection of the program memory image can reveal exactly which modules were used to make up the program. That information includes the module name and the module version and revision numbers.

For this feature to be useful, you must remember to change the version and/or revision number in the IDNT directive each time a module is modified. This information is passed on to the relocatable object code (linker text). Essentially, the revision list may be considered a special 17th relocatable section. When the REVLIST directive is omitted from the link specification, the revision information will not appear in the final object code.

The sample program on your distribution disk (LT30.A68, LT31.A68, LT32.A68 and LT33.LNK) displays its own revision list at the terminal of an ERG 68000 system. The terminal character output routine found in LT32.A68 may be modified for your particular hardware or operating system. The ORG directive in LT33.LNK may be changed to locate the program at a convenient place for your system. Each of LT30, LT31 and LT32 are to be assembled by A68K. The object modules are then to be linked using LT33 as the specification to QLINK.

Try these commands:

```
A68K LT30
A68K LT31
A68K LT32
QLINK LT33
```

Macros:

Macros can be used to construct link specification files. Macros are implemented via a separate pre-processor program. This is necessary due to the limited memory space available on CP/M-80 systems. The symbol table is memory resident, occupying the TPA space from the end of the linker program to the beginning of BDOS, less stack and file control table space. The macro pre-processor is discussed in detail in the assembler manual in Section 7, The Macro Pre-Processor.

Linker Error Detection:

Various situations can produce error messages. In some cases a single error may actually produce more than one message. Invalid characters in symbols will confuse the parser, so that error messages will vary, depending on the context in which the symbol is found. The error message may be for invalid operands, invalid expression or both. Error messages are listed and explained in Appendix E.

A period without a size may follow the mnemonic for a directive which has an associated size. This permits null macro parameter substitutions without causing errors. If parameter zero in "DC.\0 XYZ" is null, no error will be generated. The size for the directive will be the default as usual.

Invalid operands for directives will produce error messages.

Syntax errors may produce a variety of error messages, depending on the situation.

The expression evaluator places restrictions on the types of operands that are valid for each operator. Invalid types will cause error messages. Embedded blanks or other syntax errors will also produce error messages. Character constants with too many characters will result in error messages.

Expressions are used in many different contexts, many of which place restrictions on the nature of the expression. Forward references are taken into account for error checking. Expression value truncation to 8 or 16 bits may produce errors if the value is out of range.

Linker Error Message Content:

All error messages will include a message and "ERROR" or "WARNING" or a blank field for extensions of a previous error message.

During processing of the link specification file, error messages will contain the name of the current input file (normal input or "include" input), the listing line number and the line number of the current input file. This style of error message can appear during the first part of linker phase 1 (specification pass 0) or during linker phase 2 (specification pass 2).

During processing of an object module for symbol information in linker phase 1, error messages will include the module name. Such messages will appear in the listing file before the actual listing and will be caused by such things as duplicate symbols or discrepancies between the definition and use of a symbol.

During processing of an object module for relocation and linking in phase 3, error messages will include the module name and address information. If the code was assembled under the ORG directive, only the absolute address will be given. If the code was assembled under a SECTION directive, the section number will be given with the relative offset from the beginning of the section for the module. The absolute address to which the code was relocated will also be given.

With the module name and the address information, the offending source code can easily be found in the assembly listing for the module. For instance, if a BSR.L instruction references an external symbol which ends up located more than 32K from the location of the branch instruction, a "Word signed truncation" error message will result. In the module listing simply scan the address column to find the address given in the error message. In this case the error message address will be the address of the BSR instruction plus two.

SECTION 4
Linker Directives

Introduction:

Linker directives perform a variety of functions, from generating object code to controlling the linking process. For instance, the DC directive is used to place data in the resulting object code. The EQU directive is used to assign a value to a user-defined symbol. The IFEQ directive is used to enable or disable processing of the link specification lines which follow the directive.

Some directives control various aspects of the link listing. The OPT directive is used to make selections from a variety of listing options.

The syntax for each directive is described in this section. Examples are given for each of those directives having an operand field. Blanks in the syntax descriptions are not significant. Required white space for field delimiters is represented by a <s>. Other notations used in the syntax descriptions are:

<s>	represents one or more spaces or tabs
<olabel>	represents optional label
<rlabel>	represents required label
-------	represents no label allowed
<.z>	represents optional .B, .W or .L

Data Generating Directives:

Directives are provided to generate data in Z80 or 68000 byte order. No boundary alignment is performed for Z80 directives, and character constants in expressions are right justified.

68000	Z80
-----	-----
DC.B	DB
DC.W	DW
DC.L	DL
DCB.B	DUPB
DCB.W	DUPW
DCB.L	DUPL

DC — Define constant (68000 byte order).

<olabel> DC<.z> <s> <list of data items>

This directive provides the means for assembling data into a program. <list of data items> is a list of data items separated by commas. The data item size may be BYTE, WORD or LONG, with the default being WORD. A data item may be a string of characters enclosed in single quotes or a data expression. In the case of a string, the size controls the number of zeros padded at the end of the string to round out a word or long word.

Note that there is a difference between 'abcdef' and 'ab' + 128. The first case is a string which may have any number of characters. The second case is a data expression, in which 'ab' is a character constant. The handling of character constants is discussed in detail in Section 3, Linker Usage - Constants.

The data expression may contain forward references and may contain relocatable symbols as well as a constant and absolute symbols. The data expression is subject to unsigned truncation for BYTE and WORD sizes. Truncation is discussed in Section 3, Linker Usage - Expression Result Truncation.

Examples:

```
DC.B    'Hello Quelo'
DC      'Hello'                     * 48 65 6C 6C 6F 00
DC      'AB'+128                    * 41 C2
DC      MAIN
DC.L    PROC0,PROC1,PROC2,PROC3
DC.B    31,28,31,30,31,30,31,31,30,31,30,31
```

DB — Define byte (Z80 byte order).

<olabel> DB <s> <list of data items>

See the paragraph on data generating directives at the beginning of this section.

DCB — Define constant block (initialized space).

<olabel> DCB<.z> <s> <count expr> , <data expr>

This directive provides the means for assembling a block of identical data items into a program. <count expr> is an expression that specifies the number of data items. The count may not exceed 32767. A negative count or a count greater than 32767 is treated as a count of zero. The count expression may NOT contain forward references and may NOT have a relocatable value. Such cases will be flagged as errors.

<data expr> is an expression that provides the data value. The data value size may be BYTE, WORD or LONG, with the default being WORD. The data expression may contain forward references and may contain relocatable symbols as well as a constant and absolute symbols. The handling of character constants is discussed in detail in Section 3, Linker Usage - Constants. The data expression is subject to unsigned truncation for BYTE and WORD sizes. Truncation is discussed in Section 3, Linker Usage - Expression Result Truncation.

Examples:

```
DCB     3,0         * three words of zeros
DCB.B   25,' '      * 25 ASCII spaces
DCB.L   10,ERROR
```

DL — Define long word (Z80 byte order).

<olabel> DL <s> <list of data items>

See the paragraph on data generating directives at the beginning of this section.

DS — Define storage (uninitialized space).

<olabel> DS<.z> <s> <count expr>

This directive is used to reserve uninitialized space in a program. <count expr> is an expression that specifies the count of space items. A space item is BYTE, WORD or LONG, with the default being WORD. The count of space items may not exceed 32767. A negative count or a count greater than 32767 is treated as a count of zero. The count expression may NOT contain forward references and may NOT have a relocatable value. Such cases will be flagged as errors.

Examples:

DS	16	* 16 words (32 bytes)
DS.B	BUFFER.SIZE	
DS.L	7	

DUPB — Define block of bytes.

<olabel> DUPB <s> <list of data items>

DUPL — Define block of long words.

<olabel> DUPL <s> <list of data items>

DUPW — Define block of words.

<olabel> DUPW <s> <list of data items>

See the paragraph on data generating directives at the beginning of this section.

DW — Define word (Z80 byte order).

<olabel> DW <s> <list of data items>

See the paragraph on data generating directives at the beginning of this section.

END — End of link specification.

------- END <optional expression>

The linker ignores any lines following the END directive. If the END directive is in an "included" file, the remainder of the main input file, as well as the included file, will NOT be processed. The <optional expression> must include a leading space if present.

The expression establishes the entry point for the program being linked. The linker generates an internal symbol, "<<<<<<<<", which appears in the symbol table and load map reports to indicate the program starting address. The value of this expression appears in the HEX file end record address field.

If an entry point is also specified in one of the modules being linked, the linker will produce an error message. Refer to the assembler END directive description. If the symbol is omitted, it will be necessary to precede any comment with an asterisk.

Examples:

END		* no symbol given
END	MAIN.PROG	no * necessary

ENDC — End of IFxx conditional assembly.

------- ENDC

This directive terminates the IFEQ, IFNE, IFGT, IFLT, IFGE and IFLE conditional assembly directives.

EQU — Assign a permanent value to a symbol.

<rlabel> EQU<.z> <s> <expression>

This directive assigns a permanent value to the user symbol <rlabel>. The expression may NOT contain forward references and may NOT have a relocatable value. The symbol value may NOT be defined more than once in the link specification and the value must be the same for passes 1 and 2. The symbol will be globally defined, and must not conflict with public symbols defined in the modules being linked.

Note that a size designation may be specified with the directive. This enables the user to control the justification of character constants in expression evaluation. The default size is WORD for two or fewer characters and LONG for more than two characters.

The handling of character constants is discussed in detail in Section 3, Linker Usage -Constants. Justification is to the left with zero filling. A constant of more than four characters will be flagged as an error.

Examples:

ABC	EQU	'a'	* 00006100
ABC	EQU	'ab'	* 00006162
ABC	EQU	'abc'	* 61626300
ABC	EQU.B	'a'	* 00000061
ABC	EQU.W	'abc'	* error
ABC	EQU.L	'ab'	* 61620000
XYZ	EQU	0	
XYZ	EQU	BUFFERS*BUFFER.SIZE	

IDNT — Identify module.

<rlabel> IDNT <s> <ver> , <rev> <description>

This directive establishes the program identification, where <rlabel> is the program name, <ver> is the version number of the program and <rev> is the program revision number. These two numbers may range from 0 to 32767. <description> is the optional comment field and serves to give a brief description of the module.

This directive provides the means for software configuration tracking. The REVLIST directive in the linker will cause the program and module names and their version and revision numbers to be placed in the final program produced. Thus, the program configuration can be discovered by direct inspection of the memory image of the program.

Examples:

UTIL	IDNT	1,3	Utility routines

IFxx — Beginning of conditional link specification.

------- IF<condition><.z> <s> <expression>

These directives, IFEQ, IFNE, IFGT, IFLT, IFGE and IFLE, begin conditional link specification. Note that the linker does NOT support IFC and IFNC, the string compare version of conditional linking. However, these IFC and IFNC are supported in the macro pre-processor program. These conditional linking directives are terminated by the ENDC directive. Conditional linking structures may be nested and each IFxx must have a matching ENDC.

<condition> specifies the test to be applied to the value of <expression>. The expression may NOT contain forward references and may NOT have a relocatable value. IFEQ is true only if the value of the expression is zero. IFLE is true only if the value of the expression is less than or equal to zero. The others have similar interpretations.

Note that a size designation, <.z>, may be specified with the directive. This enables the user to control the justification of character constants in expression evaluation. The default size is WORD for two or fewer characters and LONG for more than two characters.

The handling of character constants is discussed in detail in Section 3, Linker Usage -Constants. Justification is to the left with zero filling. A constant of more than four characters will be flagged as an error.

Examples:

```
IFEQ    0       * always true
IFNE    0       * always false
IFGT    PDQ     * true if PDQ > 0
IFLE    PDQ     * true if PDQ <= 0
```

INCLUDE — Include another file.

------- INCLUDE <s> <file specification>

This directive causes link specification input to be temporarily switched over to another file, specified by <file specification>. When the end of the include file is reached, the input is switched back to the original source file at the next line following the INCLUDE directive. The exact nature of <file specification> is operating system dependent and is discussed in a supplemental document specific to the system on which the linker is to be used.

If an included file contains an INCLUDE directive, a warning message will be generated and the input will be switched to the new include file. None of the remaining lines of the first include file will be processed. The end of an include file always returns to the original source file. Thus, nesting of INCLUDEs is not supported but chaining is.

Examples:

```
INCLUDE   SYSDEF.A68
```

LINK — Specify items to be linked.

------- LINK <s> <item> , <item> , ...

This directive is used to specify items to be linked. An item may be a module, an entire library or selected modules from a library. Module selection may be made explicit or may be based on what is actually needed to satisfy references to undefined symbols. The type of file (module or library) and the syntax used to specify the link item determine the action taken by the linker. Variations for <item> follow.

<module file> causes a module to be loaded.

<module file> () causes a module to be loaded only if needed to satisfy references to undefined symbols.

<library file> causes all modules in the library file to be loaded.

<library file> (<module list>) causes all modules listed in <module list> to be loaded from the library file. <module list> is a series of module names separated by commas. Module names are established by the IDNT directive in the assembler source code.

<library file> () causes the library to be searched. Only those modules needed to satisfy references to undefined symbols will be loaded from the library.

Library and module file name syntax is operating system dependent.

Examples:

```
LINK    MODULE1
LINK    MODULE2()
LINK    LIBRARY1
LINK    LIBRARY2(MOD7,MOD8,MOD9)
LINK    LIBRARY3()
LINK    M1,M2,M3(),L1(M4,M5),L2()
```

LIST — Enable listing.

------- LIST

This directive is used to cancel the effect of the NOLIST directive. The source line containing this directive appears in the listing.

LLEN — Set the maximum listing line length.

------- LLEN <s> <max length>

This directive sets the maximum line length for the listing. The range limits of <max length> are 79 and 132.

Examples:

```
        LLEN        79          * screen size
```

MSG — Message to console.

------- MSG <s> <message>

This directive sends the <message> information to the console. It may be used to monitor the progress of long linker runs or to signal the operator that console input is required when the line following the MSG directive has an "INCLUDE CON:" directive. Note that "INCLUDE CON:" is applicable to CP/M-like systems, but may not be applicable to other operating environments. Since the linker makes three passes over the link specification file, it will be necessary to enter information for the INCLUDE three times.

Example:

```
        MSG Enter "BASE EQU ???"
```

NOLIST — Suppress listing.

------- NOLIST

------- NOL

These directives suppress listing output until a LIST directive is processed. Listing output is enabled as the default. The source line containing these directives appears in the listing.

NOPAGE — Disable vertical page formatting.

------- NOPAGE

This directive suppresses page titles and top and bottom margins. The effect is a continuous, unbroken listing.

OFFSET — Begin an "offset" label defining section.

------- OFFSET <s> <expression>

This directive selects and initializes a special "program counter", whose value type is "constant", as opposed to "absolute address". The value of <expression> is used to initialize the program counter. NO data producing directives may be used under the OFFSET directive. OFFSET and ORG directives may be intermingled freely in a link specification. The default program counter at the beginning of linking is ORG 0.

Typically, labels and DS directives would be used following OFFSET to assign constant values to symbols. This provides a convenient means for defining data structures.

Examples:

```
        OFFSET 0
        OFFSET FCB.BASE
```

OPT — Specify one or more options.

------- OPT <s> <option list>

<option list> is a list of option selection mnemonics, separated by commas. Options CL and MEX are used to control the presence of conditional assembly and macro expansions in the listing.

CEX	use multiple lines for object code if necessary	- default
CL	print conditional assembly	- default
MC	print macro calls	- default
MD	print macro definitions	- default
MEX	print macro expansions	
NOCEX	disable CEX	
NOCL	disable CL	
NOMC	disable MC	
NOMD	disable MD	
NOMEX	disable MEX	- default

Examples:

```
OPT     NOCL
OPT     MC,MD     * defaults
```

ORG — Begin absolute memory allocation.

------- ORG <s> <expression>

This directive initializes the absolute "program counter", whose value type is "absolute address", as opposed to "constant" (OFFSET). The value of <expression> is used to initialize the program counter. OFFSET and ORG directives may be intermingled freely in an assembly. The default program counter at the beginning of linking is ORG 0.

Examples:

```
ORG     $800
ORG     BASE
```

OUTPUT — Specify HEX output format.

OUTPUT <s> <format>

This directive may be used to specify the HEX output file format as follows:

I	Intel HEX
INTEL	Intel HEX
M	Mostek HEX
MOSTEK	Mostek HEX
MOTOROLA	Motorola S-records
S	Motorola S-records

The default format is Motorola S-records. "MOT" followed by anything will get S-records. Otherwise, only the first letter is actually used to determine format. This directive takes priority over the command line HEX format specification, so the command line "-H" option will be ignored if this directive appears in the link specification.

Examples:

```
OUTPUT  M
OUTPUT  I
```

PAGE — Begin new listing page.

------- PAGE

This directive begins a new page. If the NOPAGE directive has been used, the PAGE directive will have no effect. Also, if the NOLIST directive is in effect, the PAGE directive will have no effect. If the NOPAGE directive is not in effect, the appropriate number of blank lines will be sent to the listing to fill out a page. The vertical page format may be set using the PLEN directive.

Also, if the NOPAGE directive is not in effect, the new page will begin with a top margin. The top margin will include a title if the TTL directive has been used. The PAGE directive source line does NOT appear in the listing.

PLEN — Establish listing vertical format.

------- PLEN <s> <page length>,<top>,<bottom>

This directive may be used to control the listing page length and top and bottom margins. The default page length is 66 lines. The default top margin is 0 lines. The default bottom margin is 6 lines. That leaves 60 lines for the actual listing information, including 4 lines of page heading. The operand expressions may NOT contain forward references and may NOT have a relocatable value.

The range limits of page length, top margin and bottom margin are (24..120), (0..8) and (-8..8) respectively. When a negative bottom margin is indicated, page breaks will be accomplished with blank lines and the absolute value of the number given will determine the bottom margin. Otherwise, ASCII formfeed characters are used for page breaks. Vertical page formatting has no meaning when the NOPAGE directive is in effect.

This directive should only be used near the beginning of the link specification file, otherwise the linker may have difficulty compensating for the change in vertical page format. It does NOT have to be the first line, because that is typically where the TTL directive will be used.

Example:

```
        PLEN      66,0,6       * the default
```

REVLIST — Select revision list option.

------- REVLIST

This directive, like the SECTION directive, reserves space for a special section to contain module revision information. This software configuration information will appear in the final program output by the linker. If the REVLIST directive is not used, no space will be reserved and no configuration information will be placed in the final program. See Section 3, Linker Usage, under Software Configuration Control.

SECTION — Assign sections to absolute address.

------- SECTION <s> <section number list>

<section number list> is a list of <section number> items, separated by commas. <section number> may be an expression that evaluates to 0 through 15. This directive is unlike the SECTION directive in the assembler. In the link specification, the function is to reserve space for the specified section or sections at the current location of the program counter, established by a preceding ORG directive.

Thus, "SECTION 3" is equivalent to "DS <section 3 size>", except that the DS directive would not communicate the necessary section number to the linker. <section 3 size> refers to the accumulated sizes of the section 3 portions of all modules being linked. Even though code for various sections may be intermingled at assembly, the linker will keep the code for each section separate in the final program. This scheme permits generating programs for RAM-ROM environments.

Sections not explicitly allocated will be assigned in numerical sequence following everything else in the link specification.

Examples:

```
          SECTION   3         * locate section 3
          SECTION   DSEG      * DSEG defined by EQU somewhere
          SECTION   2,5,1     * location order is 2,5,1
```

SET — Assign a non-permanent value to a symbol.

<rlabel> SET<.z> <s> <expression>

This directive assigns a value to the user symbol <rlabel>. The expression may NOT contain forward references and may NOT have a relocatable value. The symbol value may be defined more than once in a program. The symbol will be globally defined, and must not conflict with public symbols defined in the modules being linked.

Note that a size designation may be specified with the directive. This enables the user to control the justification of character constants in expression evaluation. The default size is WORD for two or less characters and LONG for more than two characters.

The handling of character constants is discussed in detail in Section 3, Linker Usage -Constants. Justification is to the left with zero filling. A constant of more than four characters will be flagged as an error.

Examples:

```
ABC       SET       'ab'                * see EQU for string examples
XYZ       SET       45
XYZ       SET       2*(XX-YY/5)|$800
XYZ       SET       XYZ+1
```

SPC — Blank lines to listing.

------- SPC <s> <blank line count>

If the NOLIST directive is in effect, the SPC directive will have no effect. This directive causes a number of blank lines to appear in the listing. The range limit of <blank line count> is 0 through the maximum listing lines per page. The count expression may NOT contain forward references and may NOT have a relocatable value. The SPC directive does not appear in the listing.

Examples:

```
          SPC       3         * 3 blank lines to listing
          SPC       NSP
```

TTL — Establish page title information.

------- TTL <s> <title string>

The title string is limited to 60 characters and appears at the top of each successive listing page until another title directive takes effect. The PLEN directive determines the size of a page. The title string begins with the first non-blank character and carries through to the end of the line. The string is NOT enclosed in quotes. For the title to appear on the first listing page, no printable source lines may precede the TTL direcive.

Example:

```
          TTL       Gizmo Program - 9/8/83
```

SECTION 5
HEX Utilities

The SPLIT Program:

This program is used to split a HEX file of Motorola S-records into separate even and odd address data HEX files of Intel hex records. This separation of data is required for programming 8 bit wide ROMs to be used in systems having a 16 bit wide data bus. The program prompts the user for starting and ending addresses. All data outside that address range will be discarded.

The starting address must be even and the ending address must be odd.

All addresses in the specified range for which no data is present in the input file will have a value of zero. The first byte in the even output file will be the byte at the starting address. The first byte in the odd output file will be the byte at the starting address plus one.

The first address in both of the output files will be zero, as the addresses in the output files are relative to the beginning of the specified range. The program keeps track of the highest address used within the specified range and does NOT output data for any address beyond that.

The command line to run the program is as follows:

```
SPLIT XYZ                              ; short form
SPLIT XYZ.EVN,XYZ.ODD=XYZ.HEX          ; long form
```

The two command lines above are equivalent. The long form permits giving the output files different names if desired.

The IMAGE Program:

The purpose of the IMAGE program is to transform a HEX file of Motorola S-records into a memory image binary file. This might be used to create an executable file to run on CP/M-68K, provided that header information is included at the beginning of the program.

The entire HEX file is read once to determine the size of the memory image. If the image is too large for the available working memory, the HEX file will be processed in multiple passes to create the output file. If the memory space available to IMAGE is sufficient, only a second pass will be required.

The command line to run the program is as follows:

```
IMAGE XYZ                              ; short form
IMAGE XYZ.IMG=XYZ.HEX                  ; long form
```

The two command lines above are equivalent. The long form permits giving the output file a different name if desired.

SECTION 6
The Object Librarian

Object Library Benefits:

The Object Librarian provides the means for creating and maintaining a library (collection) of relocatable object modules in a single file. There are several benefits to keeping modules in a library. For one, utility subroutines used by a number of different programs may be collected in a library. Such a library would then be searched by the linker for modules required to complete the linking process.

Another benefit is a reduced burden on the directory of the computer operating system. For instance, collecting all of the modules for a given program into a library reduces the directory usage from many file names to the single file name of the library.

Librarian Features:

The librarian is controlled by a library specification input. This input may come from a file prepared in advance or from the console for interactive use. The librarian produces a listing output which contains the specification input, error messages and other information. This output may go to a file, the console or the listing device. If the output is directed to a file and the input is from the console, the output will also go to the console. It is important to keep in mind that QLIB maintains internal tables for module names and global symbols. The ADD and DELETE commands operate on these tables and do not affect the files on the disk. The contents of the module table at the time the END command is processed determines the contents of the new library to be constructed.

Means are provided for creating a new library, updating an existing library and copying selected modules from a library into individual files. Library contents may be reported by creating a file for use by QSYM. Module summary, symbol table and cross-reference reports are available.

Object Librarian Demonstration:

The Object Librarian will be demonstrated using sample files which are supplied on your distribution disk.

The simplest command line for using the librarian consists of the librarian name followed by the name of the librarian specification file. In this case some options will also be included in the command line. Note that the files LT30.A68, LT31.A68 and LT32.A68 must first be assembled to produce LT30.LTX, LT31.LTX and LT32.LTX, respectively.

Enter the following commands:

```
A68K -L LT30                ; Assembly command.
A68K -L LT31                ; Assembly command.
A68K -L LT32                ; Assembly command.
```

```
Read assembler source,          LT3x.A68
Produce listings,               LT3x.PRN
Produce object files,           LT3x.LTX
```

The following will demonstrate the method for creating a library file using a librarian specification file. Enter the following command.

```
QLIB -SLIX QT33                  ; Librarian command.

Read librarian specification,   QT33.LIB
Create object library file,     LIBRARY.LTX
Add object files to library file, LT3x.LTX
Produce listing file,           QT33.LST
Produce symbol file,            QT33.SYM
```

The "L" option instructs the librarian to retain lower case in the module names used by the librarian commands. The "I", "S" and "X" options instruct the librarian to place module summary, symbol table and cross-reference data in the ".SYM" output file. Enter the following command.

```
QSYM -IB QT33                   ; Report command.

Read symbol file,               QT33.SYM
Produce report file,            QT33.RPT
```

The "I" and "B" options instruct the symbol report program to generate two reports: module summary (IDNT directive information) and combined symbol table and cross-reference.

Examine the listing file, QT33.LST. It should appear similar to that provided in Appendix A, with the possibility of minor differences due to changes in the listing format or due to editing necessary to make the listing fit the basic format of this manual.

The CREATE command selects the librarian operating mode. The IDNT directive establishes a name for the library along with version, revision and description information.

Next is a demonstration of a method to update the library file created above using the Object Librarian interactively.

Enter the following commands:

```
QLIB -L CON:        ; Librarian command.
UPDATE LIBRARY      ; Select the update mode.
MLIST               ; List the module names.
DELETE lt32         ; Delete lt32 module.
                    ; Note: the module name lt32 must be
                    ; entered in lower case because the
                    ; module name appears in lower case within
                    ; the IDNT information found in the source
                    ; file LT32.A68. The "-L" option must also
                    ; be included in the QLIB command line.
MLIST               ; List the module names.
END                 ; End specification phase and begin
                    ; construction of the new library file.
                    ; Note: the library file contents are not
                    ; changed until the "end" command is
                    ; encountered.
```

Next is a demonstration of a method to extract a module from the library created above and copy the module to a disk file.

Enter the following commands:

```
QLIB -L CON:        ; Librarian command.
EXTRACT LIBRARY     ; Select the extract mode.
MLIST               ; List the module names.
COPY lt30,OLDLT30   ; Copy the lt30 module to a file.
                    ; Note: the module name lt30 must be
                    ; entered in lower case because the
                    ; module name appears in lower case within
                    ; the IDNT information found in the source
                    ; file LT30.A68. The "-L" option must
                    ; also be included in the QLIB command line.
END                 ; End specification phase and begin copy
                    ; process.
```

Command Line to Invoke the Librarian:

```
QLIB <filename>                        ; short form
QLIB =<filename>                       ; intermediate
QLIB <listing>,<symbols>=<libspec>     ; full form
```

---------------------------------- I/O Specification Equivalence ----------------------------------

=CON:	CON:,NUL:=CON:
CON:,XYZ=CON:	CON:,XYZ.SYM=CON:
XYZ	XYZ.LST,XYZ.SYM=XYZ.LIB
CON:=XYZ	CON:,NUL:=XYZ.LIB
=XYZ	XYZ.LST,NUL:=XYZ.LIB
XYZ.ABC=XYZ.SRC	XYZ.ABC,NUL:=XYZ.SRC
B:,C:=A:X	B:X.LST,C:X.SYM=A:X.LIB
.ABC,.SYM=X	X.ABC,X.SYM=X.LIB

Note that these examples apply to "CP/M like" environments. Supplementary documentation deals with issues specific to the operating system under which QLIB is being used.

Library and object module file names appear with various librarian commands in the library specification file. The default extension for such file names is ".LTX", an abbreviation for "linker text".

Command Line Invoked Options:

The command line may consist of several items, one of which must be the I/O specification. Command line items are separated by spaces, implying that the I/O specification may not have embedded spaces. Items beginning with a minus sign are interpreted as option selections.

The "-B" option has the same effect as selecting both the "-S" and "-X" options.

The "-I" option causes module summary information (from IDNT directives) to be written to disk for subsequent processing by QSYM. Whether or not this file is actually written depends on the input/output specification in the command line. The same disk file is used for the "I", "S" and "X" options. From this information, the QSYM program produces a list of all library modules including module name, version, revision and description.

The "-L" option causes QSYM to distinguish between upper and lower case letters in module names appearing with the ADD, COPY and DELETE commands.

The "-S" option causes the library symbol information to be written to disk for subsequent processing by QSYM. Whether or not this file is actually written depends on the input/output specification in the command line. The same disk file is used for the "I", "S" and "X" options. The QSYM program formats a symbol table report and/or a cross reference report and/or a module summary report.

The "-T" option causes truncation of long module names to 8 characters, rather than the usual 31 characters. This only affects module names appearing with the ADD, COPY and DELETE commands.

The "-V" option is used to put an ASCII formfeed character at the beginning of the listing output.

The "-X" option causes cross reference data to be written to disk for subsequent processing by the QSYM program. Whether or not this file is actually written depends on the input/output specification in the command line. The same disk file is used for the "I", "S" and "X" options.

Command Line Examples:

"QLIB -S XYZ" specifies that a symbol file is to be produced, that the library specification file is "XYZ.LIB", that the listing file is "XYZ.LST" and that the symbol file is "XYZ.SYM".
"QLIB CON: =XYZ" suppresses the symbol file, sends the listing to the console and specifies "XYZ.LIB" as the library specification file. Here, the console is the "standard output" device of the C programming language.
"QLIB CON: =CON:" suppresses the symbol file and uses the console for both source input and listing output ("standard input" and "standard output" in C). This allows the librarian to be used interactively.

Librarian Usage:

A library specification must begin with the name of the library to be operated upon. For this purpose the CREATE, UPDATE and EXTRACT commands are provided to establish the librarian operating mode and the file name for the "current library". Only one of these commands may be used in a given specification.

The IDNT command is used to establish the new library name, version, revision and description. This command is similar to the linker IDNT directive, only the library name is placed as the first command operand since there is no label field as with the linker directive.

The ADD and DELETE commands are used for manipulation of individual modules under "create" and "update" operating modes.

The COPY command is used to copy a module from a library to a file under the "extract" operating mode.

The MLIST and SLIST commands may be used to determine the current state of the library being operated upon. For instance, one might want to list the global symbols after adding a new module.

The END command terminates the library specification and initiates the actual construction of the new library file in the "create" and "update" operating modes. In the "extract" mode the only effect of the END command is to close the library file. The ADD and DELETE commands only manipulate the symbol and module tables for the library being constructed and the COPY command has no effect on the library file.

No library file changes will be made on disk until an END command is processed or an end-of-file condition is detected for the library specification file.

The QUIT command may be used in place of the END command to exit from the librarian without making any library file changes.

If the specification input is from a file, an end-of-file condition will have the same effect as the END command. If the input is from the console, a control Z may be used in place of END.

The internal structure of a library precludes the inclusion of more than one module defining a given global symbol. A library file begins with a table of modules and a table of all globally defined symbols. Associated with each symbol is information as to the location in the library of the module which defines the global symbol. This structure eliminates the need for the linker to search the entire library file to find specific module names or globally defined symbol names.

Library summary information in the form of a module listing, a symbol table table listing and a cross-reference listing may be had via the QSYM program. This is accomplished via a symbol file created in response to command line options.

Other librarian commands are INCLUDE, LIST, LLEN, MSG, NOL, NOLIST, NOPAGE, PAGE, SPC, PLEN and TTL. These commands operate the same as the linker directives of the same name. Refer to the linker directives for explanations. The only difference is that linker directives are preceded by a label field and librarian commands have no label field.

General Syntax Rules:

Asterisks and semicolons may be used to indicate comments in the specification file. Commands may begin in column 1 or may be preceded by spaces and/or tabs. Some commands may be abbreviated as shown later in a table. The command word (or abbreviation) must be followed by at least one space or tab (represented by <s> below) if additional information is part of the command. Commands may be in upper or lower case.

Module names are subject to the same restrictions imposed by the assembler and linker on symbols.

File name restrictions are operating system dependent.

Command Descriptions:

ADD Add module(s) to the current library.

ADD <s> <add list>

<add list> is a list of items, separated by commas, specifying module(s) for addition to the current library. A list item may have one of the following three possible formats:

<module file> specifies a simple module file, such as that produced by an assembler, for addition to the current library.

<library file> specifies that an entire library file is to be added to the current library.

<library file> (<module list>) specifies individual modules to be added to the current library from a library file. <module list> is a list of module names separated by commas.

Example: ADD mf1,mf2,lf1(mn8,mn3,mn5),lf2.

COPY Copy a module from the current library to the specified file.

COPY <s> <module name> , <module file name>

CREATE Establish a new, empty current library for subsequent ADD and DELETE comands.

CREATE <s> <library file name>

If the file specified by CREATE exists, it will be deleted when the END command is encountered. The IDNT command is required.

DELETE Delete a module from the current library.

DELETE <s> <module name>

END End the library specification phase and construct the new library file.

END

EXTRACT Establish a current library for subsequent COPY commands.

EXTRACT <s> <library file name>

If the file specified by EXTRACT does not exist, an error message will result.

IDNT Identify module.

IDNT <s> <module name>,<ver>,<rev> <s> <description>

This directive establishes module identification, where <module name> is the module name, <ver> is the version number of the module and <rev> is the module revision number. These two numbers may range from 0 to 32767. <description> is the optional comment field and serves to give a brief description of the module.

MLIST List the module names for the current library.

MLIST

QUIT Exit without altering any existing library files.

QUIT

SLIST List the global symbols for the current library.

SLIST

UPDATE Establish a new current library from an existing library for subsequent ADD and DELETE commands.

UPDATE <s> <library file name>

If the file specified by UPDATE does not exist, an error message will result. If it does exist, two things will happen: The module and symbol tables will be initialized from the existing library and the existing library will be renamed to have an extension of ".LTB" when the END command is encountered. The IDNT command is optional. If no IDNT command is given the identification information will be carried over from the old library file and the revision number will automatically be incremented for the new library file.

Command Reference Table:

"create" mode	"update" mode	"extract" mode
CREATE CR	UPDATE UP	EXTRACT EX
IDNT (req.) ID	IDNT (opt.) ID	
ADD A	ADD A	
DELETE D	DELETE D	
		COPY C
MLIST M	MLIST M	MLIST M
SLIST S	SLIST S	SLIST S
END	END	END
QUIT	QUIT	QUIT

APPENDIX A
Sample Listings

```
Quelo   ...A68K D5.0L  1/3/85   ...Run on Jan 9, 1985  12:35:31      ...Page  1
A:LT30.LTX , A:LT30.PRN ,  = A:LT30.A68
   ...

                            lt30    idnt    1,3


                            * 1.3   01/09/85  P. Adams     remove nopage and llen
                            * 1.2   05/01/84  R. Curtiss   from  main.a68  1.1
                            *
                            *       Main program for linker testing
                            *


                            *
                            * Entry points
                            *
                                    xdef    main.prog
                            *
                            * External procedures referenced
                            *
                                    xref.s  disp.config
                                    xref.s  line.out
                                    xref.s  cr.lf.out
                            *
                            * External data referenced
                            *
                                    xref.s  sign.on

0'000000                            section 0

                       -- -- -- -- -- -- -- -- -- -- -- -- -- --

0'000000                    main.prog     proc
0'000000  41F8'0000                 lea     sign.on,a0
0'000004  4EB8'0000                 jsr     line.out
0'000008  4EB8'0000                 jsr     cr.lf.out
0'00000C  4EB8'0000                 jsr     disp.config
0'000010  4E75                      rts

0'000012                            end
    0 Errors
```

```
Quelo   ...A68K D5.0L  1/3/85   ...Run on Jan 10, 1985  16:44:49     ...Page  1
A:LT31.LTX , A:LT31.PRN ,  = A:LT31.A68
   ...

                               1.          plen    66,6,6
                               2.
                               3.  lt31    idnt    1,7
                               4.
                               5.
                               6.  * 1.7   01/09/85  P. Adams      remove nopage and llen
                               7.  * 1.6   05/04/84                spelling correction
                               8.  * 1.5                           remove short branches
                               9.  * 1.4   05/01/84  R. Curtiss    from  config.a68  1.3
                              10.  *
                              11.  *       Linker test module
                              12.  *
                              13.
                              14.
                              15.  *
                              16.  * Entry points
                              17.  *
                              18.          xdef    disp.config
                              19.  *
                              20.  * External procedures referenced
                              21.  *
                              22.          xref.s  string.out
                              23.          xref.s  spaces.out
                              24.          xref.s  cr.lf.out
                              25.          xref.s  char.out
                              26.  *
                              27.  * External data referenced
                              28.  *
                              29.          xref.s  revlist
                              30.
1'000000                      31.          section 1

                         -- -- -- -- -- -- -- -- -- -- -- -- -- --

1'000000                      32.  disp.config  proc
                              33.
1'000000  41F8'0000           34.          lea     revlist,a0
                              35.
                              36.          repeat
1'000004  1010                37.            move.b  (a0),d0         end of list check
                              38.
1'000006  6738                39.            break.s if <eq>         loop exit
                              40.
1'000008  2248                41.            move.l  a0,a1           save string origin
1'00000A  4EB8'0000           42.            jsr     string.out      a0 at end of string
                              43.
1'00000E  7020                44.            moveq   #32,d0          32 column field
1'000010  D089                45.            add.l   a1,d0           pad count =
1'000012  9088                46.            sub.l   a0,d0          32 - ( a0 - a1)
1'000014  4EB8'0000           47.            jsr     spaces.out
```

```
Quelo   ...A68K D5.0L. 1/3/85   ...Run on Jan 10, 1985  16:44:49     ...Page  2
A:LT31.LTX , A:LT31.PRN ,  = A:LT31.A68
   ...
1'000018  5288                   add.1   #1,a0          past end of string
1'00001A  3008                   move    a0,d0
1'00001C  COBC 00000001          and.1   #1,d0
1'000022  D1C0                   add.1   d0,a0          word boundary adjust
1'000024  3018                   move    (a0)+,d0       version
1'000026  4EB9'00000000          jsr     hex.word.out
1'00002C  7002                   moveq   #2,d0          two spaces
1'00002E  4EB8'0000              jsr     spaces.out
1'000032  3018                   move    (a0)+,d0       revision
1'000034  4EB9'00000000          jsr     hex.word.out
1'00003A  4EB8'0000              jsr     cr.lf.out
1'00003E  60C4                 endr
1'000040  4EB8'0000            jsr     cr.lf.out
1'000044  4E75                 rts
2'000000                       section 2

                  -- -- -- -- -- -- -- -- -- -- -- -- -- --

2'000000                   hex.word.out  proc
2'000000  2200                 move.1  d0,d1
2'000002  7404                 moveq   #4,d2           repeat count
                               repeat
2'000004  E959                   rol     #4,d1
2'000006  3001                   move    d1,d0
2'000008  C07C 000F              and     #$f,d0         nibble mask
2'00000C  B07C 0009 6F02         if      d0 <gt> #9  then.s
2'000012  5E40                     add     #7,d0          offset to 'A'
                                 endi
2'000014  D03C 0030              add.b   #'0',d0        ASCII bias
2'000018  4EB8'0000              jsr     char.out
2'00001C  5342                   sub     #1,d2
2'00001E  66E4                 until   <eq>
2'000020  4E75                 rts
2'000022                       end
    0 Errors
```

```
Quelo   ...A68K D5.0L  1/3/85   ...Run on Jan 10, 1985  16:46:10    ...Page  1
A:LT32.LTX , A:LT32.PRN ,  = A:LT32.A68
   ...

                             1.         plen    66,6,6
                             2.
                             3. lt32    idnt    2,1
                             4.
                             5.
                             6. * 2.1   01/09/85  P. Adams    remove nopage and llen
                             7. * 2.0                         ' '>>8
                             8. * 1.9                         remove short branches
                             9. * 1.8   05/01/84  R. Curtiss  from  util.a68  1.7
                            10. *
                            11. *       Linker test module
                            12. *
                            13.
                            14. *
                            15. * Entry points
                            16. *
                            17.         xdef    line.out
                            18.         xdef    string.out
                            19.         xdef    spaces.out
                            20.         xdef    cr.lf.out
                            21.         xdef    char.out
                            22.
                            23.
4'000000                    24.         section 4

                        -- -- -- -- -- -- -- -- -- -- -- -- -- --

4'000000                    25. line.out     proc
4'000000  6100 ****         26.         bsr     string.out
4'000004  6100 ****         27.         bsr     cr.lf.out
4'000008  4E75              28.         rts
3'000000                    29.         section 3

                        -- -- -- -- -- -- -- -- -- -- -- -- -- --

3'000000                    30. string.out   proc
                            31.         repeat
3'000000  1018              32.           move.b  (a0)+,d0
3'000002  6700 0008         33.           break   if <eq>        end of string
3'000006  6100 0008         34.           bsr     char.out
3'00000A  60F4              35.         endr
3'00000C  5388              36.         sub.l   #1,a0         a0 at string terminator
3'00000E  4E75              37.         rts
2'000000                    38.         section 2

                        -- -- -- -- -- -- -- -- -- -- -- -- -- --

2'000000                    39. spaces.out   proc
2'000000  3200              40.         move    d0,d1         space count
2'000002  7020              41.         moveq   #' '>>8,d0    space character
```



```
Quelo   ...A68K D5.0L  1/3/85   ...Run on Jan 10, 1985  16:46:10     ...Page  2
A:LT32.LTX , A:LT32.PRN ,  = A:LT32.A68
   ...

                           42.          repeat
2'000004  4A41             43.            tst      d1
2'000006  6F00 000A        44.            break    if <le>          count d1 <= 0
2'00000A  6100 ****        45.            bsr      char.out
2'00000E  5341             46.            sub      #1,d1
2'000010  60F2             47.          endr
2'000012  4E75             48.          rts
1'000000                   49.          section 1

                        -- -- -- -- -- -- -- -- -- -- -- -- -- --

1'000000                   50.  cr.lf.out    proc
1'000000  700D             51.          moveq   #$0d,d0         carriage return
1'000002  6100 ****        52.          bsr     char.out
1'000006  700A             53.          moveq   #$0a,d0         linefeed
1'000008  6100 ****        54.          bsr     char.out
1'00000C  4E75             55.          rts
3'000010                   56.          section 3

                        -- -- -- -- -- -- -- -- -- -- -- -- -- --

3'000010                   57.  char.out    proc
                           58.          repeat
3'000010  1838 FF01        59.            move.b  $ffffff01,d4    get port status
3'000014  C83C 0001        60.            and.b   #1,d4           TxRDY bit mask
3'000018  67F6             61.          until   <ne>
3'00001A  11C0 FF00        62.          move.b  d0,$ffffff00    output data
3'00001E  4E75             63.          rts
                           64.
3'000020                   65.          end
    0 Errors
```

```
Quelo   ...QLINK D2.0J  1/15/85   ...Run on Jan 16, 1985  15:21:55   ...Page  1
A:LT33.HEX , A:LT33.SYM , A:LT33.SYM = A:LT33.LNK
   ...

                                  plen    66,6,6

                          lt331   idnt    1,5               REVLIST demonstration

                          *                                 "lt331" appears in the
                          *                                 S0 record of the HEX
                          *                                 load module


                          * 1.5   01/16/85                  remove llen and nopage
                          * 1.4   05/04/84                  more comments
                          * 1.3   05/01/84  R. Curtiss      from  sample.168  1.2
                          *
                          *       Linker test - link/locate specification
                          *
                          *       0.  Print this file, LT33.LNK, for
                          *           reference.
                          *
                          *       1.  Assemble LT30.A68, LT31.A68 and LT32.A68
                          *           and print the resulting LT30.PRN,
                          *           LT31.PRN and LT32.PRN files.
                          *
                          *               A68K -L LT30
                          *               A68K -L LT31
                          *               A68K -L LT32
                          *
                          *       2.  Link using this link specification file,
                          *           LT33.LNK.  Print the resulting LT33.LST
                          *           and LT33.HEX files.
                          *
                          *               QLINK -SLIX LT33
                          *
                          *       3.  Generate module list, symbol table, cross
                          *           reference and load map.  Print the resulting
                          *           LT33.RPT file.
                          *
                          *               QSYM -IBM LT33
                          *

                                  opt     cex

                                  link    lt32              modules
                                  link    lt30                to
                                  link    lt31               link

00004000                          org     $4000             begin here

00004000                          section 0                 reserve space for
                          *                                 section 0

```

```
Quelo   ...QLINK D2.0J  1/15/85   ...Run on Jan 16, 1985  15:21:55   ...Page  2
A:LT33.HEX , A:LT33.LST , A:LT33.SYM = A:LT33.LNK
   ...

                                 51.
00004012                         52.  sign.on:
00004012  2E 2E 2E 53 61         53.          dc.b    '...Sample program to '
          6D 70 6C 65 20
          70 72 6F 67 72
          61 6D 20 74 6F
          20
00004027  69 6C 6C 75 73         54.          dc.b    'illustrate revision '
          74 72 61 74 65
          20 72 65 76 69
          73 69 6F 6E 20
0000403B  6C 69 73 74 20         55.          dc.b    'list feature',0
          66 65 61 74 75
          72 65 00
00004048  <00000000>             56.          ds      0              insure word boundary
                                 57.
                                 58.
00004048                         59.  revlist:
                                 60.          revlist                rev info all modules
00004070  0000                   61.          dc      0              terminate revlist
                                 62.
                                 63.
00004072                         64.          section 4,3,1,2        reserve space for
                                 65.  *                              these sections in
                                 66.  *                              the order shown
                                 67.
                                 68.
00004400                         69.          org     $4400          all other sections
                                 70.  *                              in case any code
                                 71.  *                              exists for them
                                 72.
                                 73.
00004400                         74.          end     main.prog      start label - see the
     0 Errors
```

LT33 Hex Output

```
S00800006C7433336C45
S12040122E2E2E53616D706C652070726F6772616D20746F20696C6C7573747261F6
S11C402F74652072657669736F6E206C697374206665617475726650024
S105407000004A
S10D40486C7433336C0000010005B2
S10D40526C743332000000002000118
S11D40726100000861000024 4E7510186700000861000008 60F453884E758D
S11740F0320070204A416F00000A6100FF90534160F24E7559
S111409C700D6100FFEC700A6100FFE64E75C6
S113408C1838FF01C83C000167F611C0FF004E75DB
S10D405C6C7433300000000100030F
S115400041F840124EB840724EB8409C4EB840AA4E75D2
S10D40666C7433310000000100070 0
S12040AA41F840481010673822484EB8407C7020D08990884EB840F052883008C040
S12040C7BC00000001D1C030184EB90000410470024EB840F030184EB900004104BA
S10F40E44EB8409C60C44EB8409C4E7521
S120410422007404E9593001C07C000FB07C00096F025E40D03C00304EB8408C539D
S10841214266E44E7546
S9034000BC
```

LT33 Module Summary List

```
Quelo    ...QSYM D2.0A  9/4/84   ...Run on Jan 16, 1985  15:21:55      ...Page  1
A:LT33.RPT = A:LT33.SYM
  ...

  ...68000 Linker      ...Module Summary

Module  Ver  Rev Name         Description
------ ---- ---- ------------ ------------------------------------------
    0.    1    5 1t331        REVLIST demonstration
    1.    2    1 1t32
    2.    1    3 1t30
    3.    1    7 1t31
```

LT33 Symbol Table and Cross Reference

```
Quelo    ...QSYM D2.0A  9/4/84    ...Run on Jan 16, 1985  15:21:55     ...Page  2
A:LT33.RPT = A:LT33.SYM
  ...

  ...68000 Linker       ...Symbol Table  and  Cross Reference

   Value Atr Symbol                   Module number reference/definition(=)
-------- --- --------------------    ---------------------------------------------
00004000 a-I <<<<<<<<                    0=
0000408C 3 P char.out                    1=     3
0000409C 1 P cr.lf.out                   1=     2      3
000040AA 1 P disp.config                 2      3=
00004072 4 P line.out                    1=     2
00004000 0 P main.prog                   0      2=
00004048 @ L revlist                     0=     3
00004012 @ L sign.on                     0=     2
000040F0 2 P spaces.out                  1=     3
0000407C 3 P string.out                  1=     3

  +++++++++++++++   Attribute Legend    +++++++++++++++

Value Type           Def/Ref Flag   By  S,E,L,P = public
-------------------  ------------   ---------------------
   #  constant           def&ref     s  SET directive
                       - def only    e  EQU directive
   M  register mask    ? ref only    r  REG directive
   @  abs address                    l  label
0..F  reloc address                  p  procedure label
   U  unspecified                    x  XREF directive
   a  address                        i  internally def
   %  error            % error       %  error
```

LT33 Load Map

```
Quelo   ...QSYM D2.0A  9/4/84   ...Run on Jan 16, 1985  15:21:55     ...Page  3
A:LT33.RPT = A:LT33.SYM
  ...

  ...68000 Linker       ...Absolute Address Load Map

Section  Address        Module name   Symbol
-------  --------  -----  -----------  ----------------------------------------

0 Start 00004000
0       00004000    2. lt30
a       00004000                       <<<<<<<<
0       00004000                       main.prog
0 End   00004011
        --------
0 Size        12

@       00004012                       sign.on
@       00004048                       revlist

4 Start 00004072
4       00004072    1. lt32
4       00004072                       line.out
4 End   0000407B
        --------
4 Size         A


3 Start 0000407C
3       0000407C    1. lt32
3       0000407C                       string.out
3       0000408C                       char.out
3 End   0000409B
        --------
3 Size        20


1 Start 0000409C
1       0000409C    1. lt32
1       0000409C                       cr.lf.out
1       000040AA    3. lt31
1       000040AA                       disp.config
1 End   000040EF
        --------
1 Size        54


2 Start 000040F0
2       000040F0    1. lt32
2       000040F0                       spaces.out
2       00004104    3. lt31
2 End   00004125
        --------
2 Size        36
```

```
   Quelo   ...QLIB D1.0M  10/25/84    ...Run on Jan 16, 1985  16:23:50   ...Page  1
A:QT33.LST , A:QT33.SYM = A:QT33.LIB
   ...

   1.  * 1.0   01/16/85  P. Adams
   2.  *
   3.  *       QLIB test specification
   4.  *
   5.  *       0.    Print this file, QT33.LIB for reference.
   6.  *
   7.  *       1.    Assemble LT30.A68, LT31.A68, LT32.A68 and print
   8.  *             the resulting LT30.PRN, LT31.PRN, LT32.PRN files.
   9.  *
  10.  *                 A68K -L LT30
  11.  *                 A68K -L LT31
  12.  *                 A68K -L LT32
  13.  *
  14.  *       2.    Run QLIB using this librarian specification file,
  15.  *             QT33.LIB.  Print the resulting LIBRARY.LST file.
  16.  *
  17.  *                 QLIB -SLIX QT33
  18.  *
  19.  *       3.    Generate the librarian module summary and symbol
  20.  *             table report.  Print the resulting QT33.RPT file.
  21.  *
  22.  *                 QSYM -IB QT33
  23.  *
  24.  *
  25.  idnt library,1,0 QLIB test specification
  26.  create library
  27.  add lt30
  28.  add lt31
  29.  add lt32
  30.  mlist
###   Module                           From file
  1.  lt30                             lt30
  2.  lt31                             lt31
  3.  lt32                             lt32
  31.  slist
###   Global symbol                    Defined in module
  1.  char.out                         lt32
  2.  cr.lf.out                        lt32
  3.  disp.config                      lt31
  4.  line.out                         lt32
  5.  main.prog                        lt30
  6.  spaces.out                       lt32
  7.  string.out                       lt32
  32.  end
,1,L,library,1,0,QLIB test specification  <-- New library being constructed
    0 Errors
```

QT33 Module Summary List

```
Quelo   ...QSYM D2.0A  9/4/84   ...Run on Jan 16, 1985  16:23:50     ...Page  1
A:QT33.RPT = A:QT33.SYM
  ...

  ...     Librarian   ...Module Summary

Module  Ver  Rev Name         Description
------ ---- ---- ------------ -----------------------------------------
    0.    1    0 library      QLIB test specification
    1.    1    3 1t30
    2.    1    7 1t31
    3.    2    1 1t32
```

QT33 Symbol Table and Cross Reference

```
Quelo    ...QSYM D2.0A  9/4/84   ...Run on Jan 16, 1985  16:23:50     ...Page  2
A:QT33.RPT = A:QT33.SYM
  ...

  ...      Librarian   ...Symbol Table  and  Cross Reference

   Value Atr Symbol                     Module number reference/definition(=)
-------- --- --------------------      -----------------------------------------
00000010 3 P char.out                        2      3=
00000000 1 P cr.lf.out                       1      2      3=
00000000 1 P disp.config                     1      2=
00000000 4 P line.out                        1      3=
00000000 0 P main.prog                       1=
00000000  ?  revlist                         2
00000000  ?  sign.on                         1
00000000 2 P spaces.out                      2      3=
00000000 3 P string.out                      2      3=

  +++++++++++++++   Attribute Legend   +++++++++++++++

Value Type            Def/Ref Flag   By  S,E,L,P = public
-------------------   ------------   --------------------
   #  constant           def&ref      s  SET directive
                       - def only     e  EQU directive
   M  register mask    ? ref only     r  REG directive
   @  abs address                     l  label
0..F  reloc address                   p  procedure label
   U  unspecified                     x  XREF directive
   a  address                         i  internally def
   %  error            %  error       %  error
```

APPENDIX B
Restrictions and Limitations

Conditional Linking Nesting:

The is no limit to the nesting of conditional assembly blocks.

Expression Arithmetic:

Expression evaluation is accomplished using 32 bit signed 2's complement arithmetic. The right shift is an exception in that zero filling is used instead of sign bit filling. Therefore, it is a logical right shift rather than an arithmetic right shift.

Expression Complexity:

Expression complexity is limited by the space allotted for an operator stack, an operand stack and a result string. The space reserved should be ample for any normal application.

Expression Values:

An expression value is restricted by the context in which the expression is used.

Linkable Objects:

Objects for resolution by the linker may be byte, word or long word in size.

Listing Line Length:

The listing line is limited to 132 characters.

Program Size:

Program size is limited by the file space available on disk and the memory space available for storage of symbols. Disk space and memory space limits are system dependent and are covered in supplementary documentation.

Link Specification Line Length:

Input lines are limited to 252 characters.

Symbol Length:

User defined symbols may be any length, but only the first 31 characters will be significant to the assembler.

Symbol Table Size:

Each user defined symbol uses N+9 bytes of symbol table space, where N is the number of characters in the symbol.

APPENDIX C
Relocatable Object Format (Quelo Linker Text)

Object Format Specification Reference Number:

Each linker text file begins with a reference number to insure that the linker and object librarian do not try to process object format versions for which they are not designed.

Object Header:

The reference number appears in the header which begins each MODULE and each LIBRARY file. Information in the header identifies the file as a MODULE or a LIBRARY. Also, the header contains the module or library name, version number, revision number and description (IDNT directive information). Finally, the creator of the MODULE or LIBRARY is also identified in the header. This creation information includes creator identification (program name, version and date) and creation date.

For user convenience, the entire header is in ASCII text format so that the file can be directly sent to a terminal for quick determination of file type and revision. The text is terminated by a control Z character to indicate end-of-file for display purposes. This will work for the CP/M "type" command but may or may not work for other programs and/or operating systems.

Information following the header is not ASCII text except for things like symbol names. Any information beyond the control Z may confuse your terminal to the point of requiring a reset or power off/on sequence.

The MODULE File:

One execution of the assembler produces one MODULE. The smallest item that can be handled by the linker or object librarian is the MODULE. Only an entire MODULE can be included in linking or can be added to or deleted from a LIBRARY. MODULEs usually contain globally defined symbols and external references to globally defined symbols of other MODULEs.

The LIBRARY File:

A LIBRARY is a collection of modules constructed by the object librarian program. A LIBRARY is a convenience feature for linking and is not required in any way. The order that modules are placed in a LIBRARY is NOT important to the linking process. A given LIBRARY may not contain more than one instance of a globally defined symbol. In other words, two MODULEs in the same LIBRARY may not both globally define the same symbol.

Following the library file header are module name and global symbol name tables. Associated with each name is information which enables direct access to the module body in the library. These tables facilitate library searches and eliminate the need to read through all of the text of all of the modules in the LIBRARY.

Quelo Linker Text Detail:

Following the previously mentioned header, an object MODULE consists of a series of binary records. The first byte of the record indicates record type. Record types include: symbol information, relocatable section summary, constant data for loading, relocatable expressions and end-of-module. The second byte of the record indicates the record length in bytes.

Relocatable expression records are in a postfix (RPN) format and may include all of the usual arithmetic and logical operators. Operands may be variable length constants, relocatable section base references and external symbol references.

The Quelo linker text is described in detail in a separate document. This document is available from Quelo for a nominal cost.

APPENDIX D
HEX Load Formats

Default HEX Format:

The linker normally produces the Motorola HEX format unless the format is changed by command line option or the OUTPUT directive to the linker.

Motorola S-record Format:

The Motorola HEX file format has a one line per record organization.

S0cc0000c1c2c3...cnkk	header record
S1ccaaaab1b2b3...bnkk	data record
S2ccaaaaaab1b2b3...bnkk	data record
S3ccaaaaaaaab1b2b3...bnkk	data record
S7ccaaaaaaaakk	end record
S8ccaaaaaakk	end record
S9ccaaaakk	end record

cc	byte count - includes cc, aa..., kk
aaaa, aaaaaa, aaaaaaaa	16, 24 and 32 bit load addresses or start address
b1, b2, ... bn	data bytes
c1, c2, ... cn	program name characters
kk	checksum such that cc + aa... + b1... + kk == FF

Intel HEX Record Format:

The Intel HEX file format has a one line per record organization.

:ccaaaa00b1b2b3...bnkk	data record
:00aaaa01kk	end record

cc	data byte count
aaaa	load or start address
b1, b2, ... bn	data bytes
kk	checksum such that cc + aa... + b1... + kk == 00

Mostek HEX Record Format:

The Mostek HEX file format consists of lines of 64 characters each. There is no correspondence between lines and records. Records follow one another without breaks.

The last line of the file is padded with ASCII zero characters.

There are four record types:

F0 - module header record
F2 - enumerated data record
F4 - iterated data record
F6 - module end record

Note that the iterated data record described here is not the full format allowed by Mostek. Nested iterated data blocks are never produced by the Quelo linker and therefore are not described.

F0ccccnnc1c2c3...cnzziittllllhhhhkk

F2ccccaaaab1b2b3...bnkk

F4ccccaaaarrrr00b1b2b3...bnkk

F6ccccsssskk

cccc - record byte count including checksum but excluding Fxcccc

zz - address size (two hex digits)
10 - 16 bits (Z80)
20 - 32 bits (68000)

ii - processor id
00 - unknown
01 - 3870
02 - 8080
03 - Z80
04 - reserved
05 - 68000
06 - 68200

tt - module type bits as follows:
000000mt
1.. transfer address in F6
1... main module

kk - checksum such that all bytes in the record from Fx through kk add up to zero

rrrr - replication factor

llll - lowest address to load is 32 bits
(8 hex digits) for the 68000
(4 hex digits for the Z80)

hhhh - highest address to load is 32 bits
(8 hex digits) for the 68000
(4 hex digits for the Z80)

aaaa - load address field is 32 bits
(8 hex digits) for the 68000
(4 hex digits for the Z80)

ssss - transfer address field is 32 bits
(8 hex digits) for the 68000
(4 hex digits for the Z80)

b1, b2, ... bn - data bytes

c1, c2, ... cn - program name characters

nn - number of characters in program name

APPENDIX E
Linker Error Messages

Introduction:

Linker error messages may appear in several different formats, depending on the processing phase at the time the error message is produced. Truncation errors may occur during link specification processing or during object module processing. Note that during object module processing, the error message includes sufficient information to locate the offending line in the assembly listing of the module in question. See Section 3, Linker Usage, under Linker Error Message Content.

Some error messages come from utilities (such as I/O) which are common to all of the programs in the Quelo Assembler Package. These error messages, usually not caused by user error, are documented in the Overview Manual.

Error Message Explanations:

Address reference alignment.

This indicates that a relocatable expression, representing an address, evaluates to an illegal value when resolved by the linker.

Example:

```
        XREF        XXX
        MOVE.W      #3,XXX
```

Since this is a word move operation, the destination address, XXX, must relocate to an even boundary, otherwise the above error message will result. Note that the error message includes sufficient information to locate the offending line in the assembly listing of the module in question.

ALLOC failure. (ABORT)

Unable to allocate sufficient memory for the symbol table. Refer to the operating system supplement for more information. Contact Quelo if the problem cannot be resolved.

Bad postfix operator.

This indicates an error in the object module being processed by the linker. This error should be reported to Quelo.

Bop not implemented.

Binary operator not implemented. This error should be reported to Quelo.

Byte signed truncation.

The expression contains undefined and/or non absolute symbols or the value of the expression is out of the range -128..127.

Byte unsigned truncation.

The expression contains undefined and/or non absolute symbols or the value of the expression is out of the range -128..255.

Character constant too long.

Too many characters in a character constant within an expression. See Section 3, Linker Usage, under Constants.

DS, DCB out of range.

The value of the count expression for one of these directives is out of the range -32768..32767. See Section 4, Linker Directives, under DS or DCB.

Duplicate definition.

Only symbols defined using the SET directive may be redefined, and then only with another SET directive. Permanent value symbols are defined as labels or by using the EQU directive or as global symbols in the modules being linked. See Section 3, Linker Usage, under User-Defined Symbols.

Duplicate program start.

A program start address is specified in more than one module being linked. The optional expression allowed with the assembler and linker END directives specifies the program start address.

Duplicate section definition.

A SECTION directive for a given relocatable section has appeared more than once.

ENDC without IFxx.

An ENDC directive was encountered without a preceding IFEQ, IFNE, IFGT, IFGE, IFLT or IFLE directive.

Expr. stack ufl.

Expression stack underflow. Unless something wrong can be found in the expression, this error should be reported to Quelo.

Expression class-type invalid.

The class-type of an expression is not valid for the context in which the expression is used. See Section 3, Linker Usage, under Expression Usage Restrictions.

Expression type. (WARNING)

During expression evaluation an operator was applied to operand(s) of a type for which the operator is not valid. For instance, multiplying two addresses together. See Section 3, Linker Usage, under Expression Result Type Determination.

Expression undefined.

An expression containing forward references is used in a context where forward references are not permitted. See Section 3, Linker Usage, under Expression Usage Restrictions.

Extra IDNT encountered.

More than one IDNT directive was encountered in the link specification.

I/O specification error. (ABORT)

The input/output specification in the command line was in error. See Section 3, Linker Usage.

IFxx without ENDC.

At the end of link specification processing an IFEQ, IFNE, IFGT, IFGE, IFLT or IFLE directive was left unterminated by an ENDC directive.

Include file not present.

The file specified with the INCLUDE directive could not be opened for reading.

Include file specification error.

The filename specified with the INCLUDE directive was not recognized as being meaningful. Note that filename specifications are operating system dependent.

Invalid directive.

The item in the directive field was not recognized as a linker directive.

Invalid expression.

The parser failed to find a reasonable terminating character when it finished processing the expression. Expected terminating characters are: a blank, a comma, an opening parenthesis, an end-of-line, or possibly others in various situations. Syntax errors might cause an early processing halt in the middle of the expression.

Invalid label.

The user-defined symbol in the label field contains characters not allowed for symbols. See Section 3, Linker Usage, under User-Defined Symbols.

Invalid object record.

Object file error. Please report this to Quelo.

Invalid operand(s).

See Section 4, Linker Directives, for valid directive operands.

Invalid option. (ABORT)

A command line option was not recognized. See Section 3, Linker Usage, under Command Line Invoked Options.

Invalid section number.

The section number specified is out of the range 0 through 15.

Invalid size.

The size designation is not valid for the particular linker directive. Sizes are .B, .W or .L, for byte, word or long respectively. The size designation is appended to a directive mnemonic. See Section 4, Linker Directives, for valid directive sizes.

Label missing.

The directive requires a label. See Section 4, Linker Directives.

Label not permitted.

A label is not permitted with this directive. See Section 4, Linker Directives.

Label on odd address. (WARNING)

The preceding label was assigned an odd address and might possibly be used in references to instructions or to word or long data items. "<label>: DS 0" may be used to insure that the label falls on an even address.

Link file specification error.

The filename specified with the LINK directive was not recognized as being meaningful. Note that filename specifications are operating system dependent.

Link specification file not present.

Input file is missing.

M68K reject. (WARNING)

The line was rejected by M68K. See the Assembler Manual, Appendix F, Macro Preprocessor Error Messages.

Module not found in library.

The LINK directive permits selection of individual modules from a library file. This message results if a selected module is not found. See section 4, Linker Directives, under LINK.

More than one I/O specification. (ABORT)

Only one input/output specification is allowed in the command line.

Nested include encountered. (WARNING)

The remainder of the first include file will never be processed. See Section 4, Linker Directives, under INCLUDE.

No I/O specification. (ABORT)

An input/output specification is always required in the command line.

Not allowed under OFFSET.

A code generating directive such as DC or DCB, was used following the OFFSET directive. See Section 4, Linker Directives, under OFFSET.

Not defined in pass 1.

A definition for a particular symbol was encountered during pass 2, but not during pass 1. This could easily happen when the linker is used interactively with input from the console. Or, it could be caused by an unusual use of conditional linking where some portion of code was linked during one pass and not the other.

Object file error.

Please report this to Quelo.

Obj. seek.

A difficulty was encountered in randomly accessing an object file (module or library). Please report this to Quelo.

Object file missing.

Unable to open an object file (module or library) for reading.

Op().

Please report this to Quelo.

Operand out of range.

See Section 4, Linker Directives, for the ranges of operands.

Operand stack overflow.

The expression is too complex for the allotted operand stack space. Simplify the expression.

Operand stack underflow.

Look for invalid operands following operators in the expression. Operands may be another expression, symbols, constants or an asterisk to represent the program counter.

Operator canceled. (ABORT)

Linker processing was stopped by the operator via the console. The check for operator entry is made prior to reading each link specification line and prior to accessing each object module during linker phase 3.

Operator stack overflow.

The expression is too complex for the allotted operator stack space. Simplify the expression.

Option not implemented.

Pass1 - pass2 value diff.

The value assigned to a symbol differed between pass 1 and pass 2. This could easily happen when the linker is used interactively with input from the console. Or, it could be caused by an unusual use of conditional linking, where some portion of the link specification was linked during one pass and not the other.

PC > 65535.

Program counter is out of range.

ST code.

Object file store code error. Please report this to Quelo.

Symbol not allowed in expression.

The symbol was either not defined as a label, or by the SET, EQU, or XDEF directives.

Symbol reference/definition conflict.

An XREF directive specified a section number for a symbol that was actually defined in a different section.

Symbol table overflow. (ABORT)

Increase the amount of available memory or reduce the number and/or length of global symbols.

Symbol tagged for signed 16 bit value.

A symbol with the attribute of XREF.S has a value outside the range of the low or high 32K block of memory.

Unable to open hex output file.

Unable to open listing file.

Unable to open symbol file.

Unable to open temporary file.

The temporary file contains a summary of modules to be linked. It is written during phase 1 as each LINK directive is processed and is read during phase 3 processing.

Undefined external reference.

A reference to an undefined symbol was encountered in processing a relocatable expression in an object module. Note that the error message includes sufficient information to locate the offending line in the assembly listing of the module in question.

Undefined symbol.

An undefined symbol was referenced in the link specification.

Unknown hex format. (ABORT)

The hex format specified by the "-H" command line option was not recognized. See section 3, Linker Usage, under Command Line Invoked Options.

Unknown output format.

The hex format specified by the OUTPUT directive was not recognized. See section 4, Linker Directives, under OUTPUT.

Uop not implemented.

Unary operator not implemented. This error should be reported to Quelo.

Word signed truncation.

The expression contains undefined and/or non absolute symbols or the value of the expression is out of the range -32768..32767.

Word unsigned truncation.

The expression contains undefined and/or non absolute symbols or the value of the expression is out of the range -32768..65535.

APPENDIX F
SPLIT Error Messages

Introduction:

Some error messages come from utilities (such as I/O) which are common to all of the programs in the Quelo Assembler Package. These error messages, usually not caused by user error, are documented in the Overview Manual.

Error Message Explanations:

ALLOC failure. (ABORT)

Unable to allocate sufficient memory for buffer space. Refer to the operating system supplement for more information. Contact Quelo if the problem cannot be resolved.

Check sum bad.

The checksum computed for a S-record does not match the checksum at the end of the record.

I/O specification error. (ABORT)

The input/output specification in the command line was in error. See Section 5, HEX Utilities, under The SPLIT Program.

Invalid hex digit.

The S-record contains a non-hex character.

Invalid option. (ABORT)

A command line option was not recognized.

More than one I/O specification. (ABORT)

Only one input/output specification is allowed in the command line.

No I/O specification. (ABORT)

An input/output specification is always required in the command line.

Operator canceled. (ABORT)

Split processing was stopped by the operator via the console.

Record has no S in column 1.

Verify that the input file is in the Motorola hex format.

Record type invalid.

The second character in an S-record line is a digit representing the record type. Valid types are: 0, 1, 2, 3, 7, 8 and 9.

S-record file not present. (ABORT)

The input file was not found.

Unable to open even file. (ABORT)

Unable to open the even address data output file.

Unable to open odd file. (ABORT)

Unable to open the odd address data output file.

Unexpected end of record.

The S-record line is shorter than the length implied by the record count field.

APPENDIX G
IMAGE Error Messages

Introduction:

Some error messages come from utilities (such as I/O) which are common to all of the programs in the Quelo Assembler Package. These error messages, usually not caused by user error, are documented in the Overview Manual.

Error Message Explanations:

ALLOC failure. (ABORT)

Unable to allocate sufficient memory for buffer space. Refer to the operating system supplement for more information. Contact Quelo if the problem cannot be resolved.

Check sum bad.

The checksum computed for a S-record does not match the checksum at the end of the record.

Excessively large memory image. (ABORT)

Probably caused by locating data in high and low memory of a 32 bit address space. Examine the linker memory map for load address information.

I/O specification error. (ABORT)

The input/output specification in the command line was in error. See Section 5, HEX Utilities, under The IMAGE Program.

Invalid hex digit.

The S-record contains a non-hex character.

Invalid option. (ABORT)

A command line option was not recognized.

More than one I/O specification. (ABORT)

Only one input/output specification is allowed in the command line.

No data. (ABORT)

There is no S-record data from which to build a memory image.

No I/O specification. (ABORT)

An input/output specification is always required in the command line.

Operator canceled. (ABORT)

Image processing was stopped by the operator via the console.

Record has no S in column 1.

Verify that the input file is in the Motorola hex format.

Record type invalid.

The second character in an S-record line is a digit representing the record type. Valid types are: 0, 1, 2, 3, 7, 8 and 9.

S-record file not present. (ABORT)

The input file was not found.

Unable to open binary output file. (ABORT)

Unable to open error file. (ABORT)

Unable to open the error message output file.

Unexpected end of record.

The S-record line is sorter than the length implied by the record count field.

APPENDIX H
Librarian Error Messages

Introduction:

Some error messages come from utilities (such as I/O) which are common to all of the programs in the Quelo Assembler Package. These error messages, usually not caused by user error, are documented in the Overview Manual.

Error Message Explanations:

ALLOC failure. (ABORT)

Unable to allocate sufficient memory for the symbol table. Refer to the operating system supplement for more information. Contact Quelo if the problem cannot be resolved.

Cannot create temp file. (ABORT)

The librarian uses a temporary file to aid in recovering symbol table space after a module has been deleted from the tables.

Could not add to module table.

Symbol table space is full. No more modules can be added.

Could not add to symbol table.

Symbol table space is full or a particular symbol already exists in another module and may not be duplicated. Therefore symbol table information is incomplete for the last module added to the tables.

Error in writing to temp file. (ABORT)

This is the temporary file used for reclaiming symbol table space after a module has been deleted.

Error reading temp file. (ABORT)

This is the temporary file used for reclaiming symbol table space after a module has been deleted.

Extra IDNT encountered.

More than one IDNT directive was encountered in the library specification.

File specification error.

The filename specified by an INCLUDE, CREATE, UPDATE, EXTRACT, ADD or COPY command is invalid.

I/O specification error. (ABORT)

The input/output specification in the command line was in error.

IDNT command missing.

The IDNT command is required by the CREATE mode.

Include file not present.

The file specified with the INCLUDE command could not be opened for reading.

Include file specification error.

The filename specified with the INCLUDE command was not recognized as being meaningful. Note that filename specifications are operating system dependent.

Int. A. (ABORT)

Internal error. Please report this to Quelo.

Int. B. (ABORT)

Internal error. Please report this to Quelo.

Invalid ADD specification.

See the ADD command syntax description in Section 6, The Object Librarian, under Command Descriptions.

Invalid librarian command.

The item in the command field was not recognized as a librarian command.

Invalid object record.

Object file error. Please report this to Quelo.

Invalid operand(s).

See Section 6, The Object Librarian, under Command Descriptions.

Invalid option. (ABORT)

A command line option was not recognized. See Section 6, The Object Librarian, under Command Line Invoked Options.

Library specification file not present.

Input file is missing.

Module not found in library.

The module specified for a library file by the ADD command was not found.

Module not in table. (WARNING)

The module specified in the DELETE command is not in the module table.

More than one I/O specification. (ABORT)

Only one input/output specification is allowed in the command line.

Nested include encountered. (WARNING)

The remainder of the first include file will never be processed. See Section 4, Linker Directives, under INCLUDE.

No I/O specification. (ABORT)

An input/output specification is always required in the command line.

Not library file.

The file specified in the UPDATE or EXTRACT command is not a library file.

Obj. seek.

A difficulty was encountered in randomly accessing an object file (module or library). Please report this to Quelo.

Object file error.

Please report this to Quelo.

Object file missing.

Unable to open an object file (module or library) for reading.

Operand out of range.

See Section 4, Linker Directives, for the ranges of operands.

Operation mode already set.

A library specification may contain only one of the following commands: CREATE, UPDATE or EXTRACT.

Operation not allowed.

The operation specified does not apply to the selected operating mode. For example: the COPY command cannot be used while in the UPDATE mode.

Operator canceled. (ABORT)

Librarian processing was stopped by the operator via the console. The check for operator entry is made prior to reading each library specification line.

Problems during re-opening temp file. (ABORT)

This is the temporary file used for reclaiming symbol table space.

Replacing module in table. (WARNING)

The module specified with the ADD command is already present in the module table.

Space exhausted. (ABORT)

The symbol table is full. Increase the amount of available memory or reduce the number of modules and/or global symbols in the library.

Unable to open listing file.

Unable to open object file.

This applies to either the new library file being generated or the file specified by the COPY command.

Unable to open symbol file.

Unxpected end of module.

Please report this to Quelo.

INDEX