Link-Z

A Z-80 based Linking Loader

August, 1986

Created by
MICROCode Consulting
www.microcodeconsulting.com

CP/M 2.2 is a registered trademark of Digital Research, Incorporated. Z-80 is a registered trademark of Zilog, Incorporated. NSC-800 is a trademark of National Semiconductor Corporation. WordStar is a trademark of MicroPro International. Macro-80, Link-80, and Lib-80 are registered trademarks of MicroSoft Corporation.

DISCLAIMER

MICROCode Consulting makes no representations or warranties with respect to the contents hereof and specifically disclaims any implied warranties of merchantibility or fitness for any particular purpose. Further, MICROCode Consulting reserves the right to revise this publication and to make changes from time to time in the content hereof without obligation of MICROCode Consulting to notify any person of such revision or changes.

TABLE OF CONTENTS

Link-Z is a part of the Z development system. Other programs that comprise the Z development system are Macro-Z and Lib-Z.

A. INTRODUCTION

Link-Z (with an executable named LZ.COM) is a powerful linking loader for any Z-80 (or NSC-800) based CP/M, QP/M or any other CP/M-compatible system. A linker generates an executable file (.COM) from one or more relocatable modules and libraries while resolving external symbols to ensure proper operation. “External symbols” are names or global values used in one module which are defined in another module. The most common usage of this feature is by a compiler: the user generates the specific program, compiles it, and links it along with the associated program library containing the run-time routines. It is also useful for assemblers allowing for a library of common routines which were previously generated, saving re-assembly every time the same routine is needed.

The Link-Z linker has a rich set of commands and is three times faster than MicroSoft's Link-80. A brief list of features include:

- RAM-based linkage
- One pass operation
- Global labels up to 7 characters in length are accepted
- Nearly 100% compatible with MicroSoft Link-80
- Number of COMMON areas limited only by available memory
- Fast
- Accepts a symbol table input (.SYM) for overlay generation
- Generate executable file (.COM), hex file (.HEX), and/or SID/ZSID/DCON compatible alphabetized symbol table (.SYM)
- Set global labels from keyboard
- Supports math operations on external symbols
- Optional generation of relocator for object code that executes at greater than 100H.

If you are familiar with MicroSoft's Link-80, you can use Link-Z immediately as Link-Z commands are a superset of the MicroSoft command structure.

B. Restrictions

Although few, there are some restrictions under Link-Z.

- Only 600 global symbols and COMMON areas allowed
- Does not support MicroSoft COBOL overlays
- Size of linkable file limited to memory available (approximately 8k more than Link-80; 52k on a 60k system)

C. Link-Z Operation

To execute Link-Z, type

```
LZ
```

and the prompt "*" will appear after printing the sign-on message. At this time, a command line is entered consisting of one or more commands separated by commas:

```
cmd1,cmd2,....cmdN
```

A 'cmd' can either be a filename of a module to load, a switch command, or a switch command which includes a filename. A filename may also have a drive and/or user designator preceding it. For example,

```
HIS,B:HERS,A7:OURS,11;MINE
```

loads, in order, the file `HIS` from the current disk, `HERS.REL` from drive B:, and `OURS.REL` from drive A: user 7 , and `MINE.REL` from user 11. Note that LZ supports the QP/M construct of either a colon or a semi-colon to delineate the drive and/or user prefix.

Link-Z also accepts commands on the execution line as in

```
LZ HIS,B;HERS,A7;OURS,11:MINE
```

which has the same result. Regardless of whether any commands were entered on the Link-Z execution command line, Link-Z will print the loading statistics and return to the prompt "*" *unless* an exit switch command was given (/E, /G, or /Q).

In actuality, the link process can be quite complex. However, it can be treated as four separate sections:

- segment address declaration (if any),
- module loading (.REL and .SYM),
- searching libraries (if necessary),
- and writing the output file(s).

In addition, there are some special switch commands that allow

- listing of (un)defined symbols and statistics,
- drive reset (for changing disks),
- immediate program execution,
- assignment of global symbol values,
- resetting Link-Z,
- and aborting Link-Z.

These latter commands can be executed at any time, and will be discussed first.

C-1. Listing of (un)defined symbols and statistics

There are two commands which control listing of symbols and statistics: /M (Mapping) and /U (Unresolved map). The Mapping command lists the values of all global symbols, both defined and undefined, as well as the load statistics. The Unresolved map is similar to the Mapping command except that defined symbols are not listed. The following is an example output from a /M command.

```
 EXT1 0140 EXT2 3020 EXT3 0317

-UDF1 0144 -UDF2 0271

       2 undefined symbols

Library requests:  LIB1 LIB2

Program 0100 0320       <  544>
Data    2000 200E       <   14>
Common  3000 312F       <   47>

[   97 sectors, 49 pages]
43735 bytes free
```

The first part of the listing contains the names of the **defined global symbols**. The defined symbols are listed, three per line, with a blank ' ' preceding the symbol followed by its defined value. A blank line separates the defined and undefined symbols, if any. **Undefined symbols** are preceded by a '-' and followed by the internal chain location; the number of undefined symbols is also displayed.

Following the symbol listing are **pending library requests**, if any. These requests are *only* executed when an output file command is given *and* there are one or more undefined symbols. Next, the load statistics for each **independent segment** is shown. An “independent segment” is one that has been assigned a specific base address with the proper switch command. (Refer to the Segment Address Declaration section for more detail.) Finally, the **size (in sectors and pages)** of the loaded program is shown as well as **available memory**.

If you had issued a /U command instead of a /M command, the above display would be identical with one exception: defined symbols would not be displayed.

In addition to the two switches just discussed, if a module load of a .REL file was the last command on the command line, Link-Z performs an automatic /U command.

C-2. Drive Reset

Unlike MicroSoft's Link-80, Link-Z does not perform drive reset automatically. Rather, the switch command /R (Reset drives) must be explicitly given. The /R command can occur by itself or be part of any other command. For example, one could load a module from a new disk by

```
/R
```

followed by

```
NEWMODL
```

An easier way is

```
NEWMODL/R
```

Regardless of where the /R switch appears in a given command, a Drive Reset is *always* the first operation performed. For example, to save the file `WORK.COM` and `WORK.SYM` on a new disk in drive C and exit, you could enter

```
C;WORK/N/Y/E/R
```

Before any files are created, the disk system is reset.

C-3. Immediate Program Execution

After your program has been linked, you can save the loaded code or execute it. The /G (Go) command executes the code at the start address which, if not explicitly defined, is the base address of the program. If you wish to define a starting address, you can enter the start address after the /G and precede it with a (semi-)colon; the starting address can either be a hex value or a defined symbol location. The three possibilities are

`/G`	which executes the code at the start address
`/G:<hex>`	which executes the code at the designated address
`/G:<sym>`	which executes the code at the symbol location

C-4. Assignment of Global Symbol Values

At any time during or before the linking process, you can arbitrarily assign a value to a symbol. The format is

```
/O:<sym>=<hex>
```

as in

```
/O:MYSYM=155
```

which assigns (creates) `MYSYM` and sets its value to 155H. If <sym> does not exist, it will be created. If <sym> exists, a value will only be assigned if <sym> is undefined. Otherwise, the assignment is an error.

C-5. Resetting Link-Z

At any time during the link process you can start over by entering the /Z (Zap) command. Zapping takes place immediately upon recognition of the /Z command and all remaining commands are discarded. For example, if you entered

```
OURFILE,LIBY/S,/Z,OUTFIL/N/Y/R/X/E
```

then Link-Z would load `OURFILE`, search `LIBY` for unresolved symbols, and then reset Link-Z. The remaining commands on the line are discarded.

After execution of the /Z command , a new sign-on message is given and Link-Z prompts the user "*" for input.

C-6. Aborting Link-Z

Although a CTRL+C is a common means to abort a program, Link-Z also accepts the /Q command and is identical to pressing a CTRL+C at the start of a command line. However, if you are running under some sort of batch mode, control characters such as CTRL+C may not be accepted. /Q is an alternative way to perform the abort function.

C-7. Segment Address Declaration

Before linkage of modules and/or libraries begins, assignment of the segment addresses should be considered. If you are generating a normal .COM file, it is unlikely that the segment addresses would have any effect. Since Link-Z handles this "normal" situation safely and automatically, you may wish to skip this step.

There are three basic segment types: **program**, **data**, and **COMMON** areas. In general, "program" is thought of as the specific program code (e.g. as in ROM code that cannot modify itself), "data" is thought of as the variable space (e.g. the ROM's scratch RAM area), and "COMMON", which is similar to data, serves an additional purpose for higher level languages. Each of these can be assigned a distinct address or not assigned any address, as desired. When a segment is not assigned any address, it allocates space from the data segment. The sole exception is if **program** is defined and **data** is not, in which case the data segment is automatically appended *after* the program segment.

There are four commands for setting the segment addresses as shown at the top of the next page. Although only colon is used in the format examples, semi-colon is also accepted.

`/A:<hex>`	assigns all segments to the same area
`/P:<hex>`	assigns the base of the Program segment
`/D:<hex>`	assigns the base of the Data segment
`/C:<hex>`	assigns the base of the COMMON segment

where <hex> is a hexadecimal address value.

Upon program start-up (or after the /Z command), Link-Z sets all segment addresses to 0103H. The /M command shows the values:

```
Program 0103 0103       <    0>
```

This means that any relocatable code will start at 0103H. Further, the absence of a Data or COMMON summary means that all segments are using the same memory. This does *not* mean they are *sharing* the same memory, e.g. their segments are overlapping and in conflict. Rather, Link-Z allocates memory as needed for each segment in the specific order of: Program, COMMON, and finally, Data, with each region allocated in a non-overlapping fashion.

For example, assume you are loading a module called `SAMP1` with 60H bytes of program, 30H bytes of data, and 20H bytes of COMMON. After the load, the linked module will look internally like

0103-0162	Allocated to Program for this module
0163-0182	Allocated to COMMON for this module
0183-01B2	Allocated to Data for this module

Obviously, the space for all three segments must be known before linkage starts. Link-Z checks that this information is given at the start of any REL-module before code is loaded.

Since the segments are not distinct, the boundaries between the segments is lost once the module is loaded. After loading the above program with no segment addresses assigned, /M shows

```
Program 0103 01B3       <  176>
```

When the next module is loaded, the memory will again be allocated in a similar fashion starting from 01B3H. Once the program is linked, it will consist of Program, COMMON, and Data areas all interleaved. (Picture a layer cake of Program, COMMON, and Data of the first module followed by Program, COMMON , and Data of the second module etc.) And, for a normal executable writing a ROM to execute at 0 with data and COMMON at F000H, you would enter:

```
* /P;0/D;F000
```

A /M command would show

```
Program 0000 0000       <    0>
Data    F000 F000       <    0>
```

Since no COMMON area was assigned, COMMON "borrows" memory from the Data segment. After loading SAMP1, Link-Z would show

```
Program 0000 0060       <   96>
Data    F000 F050       <   80>
```

The program segment was loaded by itself, the data segment used the Data area, and the COMMON segment allocated memory from the Data area.

NOTE: Assignment of the various segment addresses can only occur at the start of Link-Z. Once *any* code has been loaded, the segments cannot be assigned a new location.

The /A command is somewhat different than the other segment switches. The /A command is *not* the same as a succession of /P, /D, and /C commands. Rather, /A assigns the Program segment to a new value and forces the Data and COMMON segments to allocate memory after the Program segment (effectively restoring the layer cake approach). In contrast, a sequence of /P, /D , and /C commands that specify the same base address instructs Link-Z to load all segments at the *same* address. As code from a each segment is loaded at a given address, the previous code at that location is destroyed. Link-Z checks carefully to prevent this from happening.

C-8. Module Loading

Module loading loads the designated code from a MicroSoft compatible .REL file or a symbol table (.SYM) file. As discussed earlier, loading of a .REL file simply requires the name of the file with an optional drive and/or user designator as in

```
HISFILE
B3;HERFILE
```

Link-Z automatically attaches the .REL extension to the file and reads in the code. Link-Z assigns a location to the incoming code, checks for load errors, adds any global symbols to the symbol table, and notes any library requests. If a module load of a .REL file is the last (or only) command on the command line, Link-Z prints the loading statistics and awaits further commands.

A module can also consist of just a symbol table. In this case, a /L (Load .SYM) switch is attached to the filename. For example, if the files on the previous page were symbol files, then you would enter

```
HISFILE/L
C:HERFILE/L
```

Link-Z automatically appends the .SYM extension for /L commands. Loading a symbol table is useful when generating overlays that require the location of routines in the root module(s). Link-Z does not print the loader statistics after loading a symbol file.

C-9. Searching Libraries

Once all modules are loaded, it may be necessary to search code **libraries** for unresolved symbols (usually run-time routines). "Libraries" are normal .REL files that consist of a number of separate program modules. For example, these could be a varied set of math routines, language functions, subroutines, and data block modules for a BASIC Compiler. The advantages of having often used modules in a single file is that: (1) the size is much smaller than a number of separate files spread on disk, (2) searching a single file is much faster than loading a number of files, and (3) you do not need to worry about which modules are necessary as the linker will figure that out for you.

Library searches are only performed if there are one or more unresolved global symbols. If all symbols are defined, Link-Z cannot be forced to search a library.

Library searches are initiated from two sources: module request(s) and the user. If a module has a Request Library Search command in it, the name of the requested library file is saved internally. If the user wishes a file to be searched, the filename with an optional drive and/or user specifier is entered followed by a /S (Search Library) switch as in

```
MYLIB/S
```

Again, if all symbols are resolved, Link-Z will ignore the command.

Note that module and user search commands are treated differently. A user request is executed *immediately* whereas a module request is *saved* until an output command is entered. At that time, Link-Z performs all module Requested Library Searches, in order, until either all undefined globals are resolved or the library list is exhausted. Note that Link-Z only has room for eight library requests; additional requests will result in a nasty message, but will otherwise be ignored.

C-10. Writing the output file(s)

When instructed to write an output file, whether it be a .COM (executable object code), .HEX (Intel hex format) , or .SYM (symbol table) or any combination, Link-Z first performs a little housekeeping first. If there are any unresolved symbols and pending library requests, Link-Z conducts a search of the requested library files until either all symbols have been defined or the list is exhausted. Next, Link-Z performs any external offset requests and extension link operations in the loaded relocatable code.

Finally, Link-Z checks for the three special symbols: $MEMRY, $$PROG, and $$DATA. If $MEMRY is found, Link-Z loads the first "free" memory location into the program at the location pointed to $MEMRY; this value is extremely useful for programs which must know where free memory starts (such as a "sbrk" function in C). ("Free" memory is the first location above the program which is NOT used; all memory locations from $MEMRY to the top of the TPA are not occupied by the loaded program or data.)

The remaining locations, $$PROG and $$DATA (if present), are filled with the Program and Data base addresses, respectively. This could be used to indicate the lowest memory location occupied by the program; however, since an executable .COM file ***must*** start at 100H anyway, these two symbols are less useful. Presumably, MicroSoft introduced them for upward compatibility with the 8088/8086 assemblers where this would be more important.

Once the housekeeping operations are completed, Link-Z displays any remaining unresolved symbols and load statistics. The output files requested are then written. Switch commands for the output files are a filename followed by one or more of:

`/N`	which generates a .COM file
`/X`	which generates a .HEX file
`/H`	which also generates a .HEX file
`/Y`	which generates a .SYM file

Any or all of the switches may be used in a single command; however, all output files use the same primary name with only the file extensions different.

Since output files are usually the last operation performed, there is a switch command which informs Link-Z to exit when done. The /E (End link) command can be attached to any command, but is more commonly attached to an output command. /E also accepts an execution address identical to the /G command described earlier in Immediate Program Execution.

Some example cases are:

Example 1: `LZ /P;100,BASE1,MYPRG,MYSIM/L,MYLIB/S,B1:MYPRG/N/Y/E`

Link-Z is executed, the Program segment is set to 0100H, modules `BASE1.REL` and `MYPRG.REL` are loaded, symbol table `MYSIM.SYM` is loaded, `MYLIB.REL` is searched for any undefined symbols, and finally, `B1:MYPRG.COM` and `B1:MYPRG.SYM` are created. Link-Z exits.

Example 2: `LZ /P:100/D;2000,HIS,C:HERS,M0;OURS/X/Y/E:SYM1`

Link-Z is executed, the Program segment is set to 0100H and the Data segment is set to 2000H, `HIS.REL` and `C:HERS.REL` are loaded, and finally, `M0:OURS.HEX` and `M:OURS.SYM` are written with the starting address set to the value of symbol SYM1. Link-Z exits.

Example 3: `LZ BASIC1/R,BASLIB/S,BASIC1/N/E`

Link-Z is executed, a drive reset is performed, `BASIC1.REL` is loaded, `BASLIB.REL` is searched for any undefined symbols, and `BASIC1.COM` is written. Link-Z exits.

There is a special case to consider when a .COM file being saved is not based at 0100H. For programs which are based below 0100H, Link-Z displays

```
Origin below 0100H - writing .CIM file
```

and completes the write of a .CIM instead. For programs based above 010DH, Link-Z inquires

```
Origin above 010DH - install relocator?  (Y/N):
```

If you answer "Y" to the question, a relocator is installed which moves the program to the proper location and then JuMPs to the execution address. Since the finished file is executable, Link-Z writes a .COM file in this case.

If you answer "N" to the relocation question, Link-Z assumes it is an overlay or some other special program and informs

```
Origin above 010DH - writing .CIM file
```

No additional code is included.

For programs which are based between 0100H and 010DH, Link-Z fills in the intermediate range with NOPs and writes the file unless an execution point other than the base has been specified. (The "base" is defined as the lowest address where code was loaded.) In this case, Link-Z inserts a JMP instruction at 0100H which jumps to the designated address. If the base address is less than 103H and there is a specified execution point, Link-Z displays

```
Invalid execution address (xxxxH)
```

and aborts because there is no room to put the JMP instruction.

D. Command Line Options

Below is a description of the command line options. Certain abbreviations are used to indicate the syntax. Any value in square brackets [] is optional. The abbreviations are as follows:

<hex>	A 1 to 4 digit hexadecimal number
<name>	A 1 to 8 character filename
<symbol>	A 1 to 8 character symbol name
<val>	Either <name> or <hex>

D-1. /A:<hex>

The **/A** command sets the loading base for Program, Data, and COMMON segments to the <*hex*> value. Essentially, this command resets any Data or COMMON base and performs a /P command. A **/A** is *not* the same as a succession of **/P**, **/D**, and **/C** commands. Once loading has started, segment bases *cannot* be changed.

D-2. /C:<hex>

This command sets the loading base for the COMMON segment to the <*hex*> value entered. No other segments may use the COMMON area. Once loading has started, the COMMON segment base *cannot* be changed.

D-3. /D:<hex>

This command sets the loading base for the Data segment to the <*hex*> value entered. If a separate declaration has *not* been made for the COMMON segment, any COMMONs encountered allocate memory from the Data segment. Once loading has started, the Data segment base *cannot* be changed.

D-4. /E[:<val>]

The **/E** command indicates the end of the link and can be appended to any other file operation command (**/N**, **/H**, **/Y**, or **/X**). If you wish to enter or change the current execution address, one may be entered if preceded by a colon. If a symbol rather than a hex value is used for <*val*>, the symbol must both exist and have its address defined.

D-5. /G[:<val>]

The **/G** command is identical to the **/E** command with the exception that the linked code is relocated to its starting point and executed. If you wish to enter or change the current execution address, one may be entered if preceded by a colon as described in **/E** above.

D-6. <name>/H

Save the named Intel format .HEX file. This switch can be used with the **/N**, **/X**, and **/Y** output file switches, although only one filename can be entered. (Each generates a different extension.)

D-7. <name>/L

The named .SYM file is loaded with the **/L** command. This command is helpful for generating nested overlays which use routines in the root segments. No load of any root code is necessary, just its symbol table. Link-Z automatically appends the .SYM extension to the filename.

D-8. /M

List the symbol table (both defined and undefined) and print the load statistics.

D-9. <name>/N

Save the named .COM (executable) file. This switch can be used with the **/H**, **/X**, and **/Y** output file switches, although only one filename can be entered. (Each generates a different extension.)

D-10. /O:<sym>=<hex>

Origin the symbol at the designated location. <*sym*> may either be a new symbol or an undefined symbol.

D-11. /P:<hex>

This command sets the loading base for the Program segment to the <*hex*> value entered. If a separate declaration has *not* been made for either the Data segment, COMMON segment or both, any Data/COMMON/both encountered allocate memory from the Program segment. Once loading has started, the Program segment base cannot be changed.

D-12. /Q

Quit linker and return to operating system.

D-13. /R

Reset drives. This switch can be used in conjunction with another command(s) to reset the drive *before* executing the other command(s).

D-14. <name>/S

Search the named library for undefined symbols, if any. Link-Z automatically appends the .REL extension.

D-15. /U

List the undefined symbols and the load statistics.

D-16. <name>/X

Save the named Intel format .HEX file. Identical to the **/H** command, this switch can be used with the **/N** and **/Y** output file switches, although only one filename can be entered. (Each generates a different extension.)

D-17. <name>/Y

Save the named symbol table (.SYM) file. This switch can be used with the **/N**, **/X**, and **/H** output file switches, although only one filename can be entered. (Each generates a different extension.)

D-18. /Z

Resets the linker to its initial state.

E. Error Messages

During the linking process, Link-Z constantly checks for any errors which impact the generation of a correct file. There are three types of errors: **data entry**, **warning**, and **fatal errors**.

A **data entry error** is indicated by a question mark (**?**) followed by the error description; the input buffer is discarded and Link-Z prompts for a new command.

A **warning error** is indicated by a message surrounded by percent signs (**%**). Link-Z continues operation at the peril of generating some invalid data dependent on the type of warning.

An **uncorrectable (fatal) error** is considered unrecoverable which means that there is either no way or no reason to continue operation. Link-Z aborts and returns to the operating system.

A summary and description of each error is included here.

Allocating sizes after load started

Either a Program, Data or Common size was (re)defined after loading of object code had started. This is confusing and illegal. Link-Z aborts.

Bad chain

Internal consistency check during chaining revealed a bad link. Can occur when either the file is bad or PROG, DATA, and/or COMMON areas are overlapping. Link-Z aborts.

Bad character

You entered an invalid character on the command line. The line is ignored.

Bad file

The file being read is either not a .REL-format file or is damaged. Link-Z aborts.

Bad library file

This error will only occur if the .REL-format file is corrupt. (Object code was detected before entry symbols.) Link-Z aborts.

Bad .SYM file

The .SYM file being read via the /Load command is invalid. Link-Z aborts.

Invalid execution address

You specified a starting address which requires a JuMP instruction at the start of the program. However, there is not enough room to put the instruction - move the base address to 103H or higher.

Cannot reset PROG/DATA/COMM after load

Once any loading is done, the base of the PROG, DATA, and COMMON areas cannot be changed.

Command

Link-Z does not recognize the command given.

Data/COMMON overlap

The Data and COMMON areas have collided potentially destroying object code. Change the load address used in **/D** or **/C** or both. Link-Z displays the load statistics and aborts.

Disk full

Occurs during a file write to indicate a full directory or data area on your disk. Change the disk and /Reset.

Drive/user

The command line parser detected what appears to be a filename with an invalid drive and/or user specification.

Duplicate symbol

A duplicate symbol has been detected and ignored. Note that although Link-Z continues (to help you find other duplicate symbols), you should not use the output. Go back and change the symbol name in the source file or fix the conflict another way.

Error during assembly/compilation

The program which generated this relocable file found one or more errors in the source code (which was reported in the relocatable file.). This error only applies to a relocatable files generated by a MICROCode Consulting assembler or compiler.

Execution address conflict

Link-Z detected a valid execution address when one has already been defined. Link-Z aborts.

Extension link symbol not found

During evaluation of a extension link expression, a symbol was used which either does not exist or is not defined.

File could not be created

The output file could not be created. Usually occurs when directory is full. Change disks and /Reset.

File does not exist

Self explanatory.

Hex number

The command line parser detected what appears to be a bad hexadecimal number. Only 4 hex digits (0-9, A-F) are allowed in a hex number.

Invalid link extension

Link-Z detected an invalid extension link item. It could be due to the MicroSoft COBOL overlay (which Link-Z does not support) or non-standard extension link. Link-Z aborts.

Library queue full

Request Library Search has been detected with a full library request queue. Link-Z ignores the request.

Link error

This message can occur for any number of reasons. The most common is a file so weird that Link-Z gets lost. Another time this could appear is during evaluation of pending external offsets and link expressions. Again, if something is wrong, Link-Z prints this message and aborts.

Name or number too long

The command line parser detected a name exceeding eight characters or a number exceeding four characters.

One name

The command line parser detected two names where only one is allowed.

Out of memory

You are trying to link a program that is bigger than available memory. Link-Z aborts.

Program/COMMON overlap

The Program and COMMON areas have collided potentially destroying object code. Change the load address used in **/P** or **/C** or both. Link-Z displays the load statistics and aborts.

Program/Data overlap

The Program and Data areas have collided potentially destroying object code. Change the load address used in **/P** or **/D** or both. Link-Z displays the load statistics and aborts.

Redefined base address after load started

Once object code loading has begun, it is illegal to redefine the low (base) address. This will occur if an absolute segment (ASEG) with an address lower than the Program base address is detected ***after*** loading has started.

Stack overflow

An extension link math expression is too complex. Go back to the source code and simplify any complicated expressions involving external values.

Symbol name or hex value

The command line parser was looking for, but did not find, a valid symbol or a hex number.

Symbol not a COMMON

The symbol referenced by Select COMMON Block is not defined as a COMMON area. Link-Z aborts.

Symbol previously defined

This error occurs during the load of .SYM file (/L command) when a matching symbol in Link-Z has already been defined. Link-Z exits the symbol load routine.

Symbol table overflow

Your linked program exceeds 600 symbols. Reduce the number of global symbols in your source file(s) and try again. Link-Z aborts.

Symbol undefined

You entered a symbol on the command line which is valid, but is not currently defined. The **/G**o and **/E**nd commands can only use a defined symbol (or a hex value).

Undefined COMMON

A COMMON area was selected that has not been defined. Link-Z aborts.

Z80?

Your computer is not a Z80-based system. You cannot use Link-Z. Return the package immediately. Link-Z aborts.

F. Summary of commands

Below is a summary of commands; certain abbreviations are used to indicate the syntax. Any value in brackets [] are optional. The abbreviations are as follows:

<hex>	A 1 to 4 digit hexadecimal number
<name>	A 1 to 8 character filename
<symbol>	A 1 to 8 character symbol name
<val>	Either <name> or <hex>

Command	Description
/A:<hex>	Select loading base for PROG, DATA, and COMMON
/C:<hex>	Select loading base for COMMON area
/D:<hex>	Select loading base for DATA area
/E[:<val>]	End link and generate the designated file(s)
/G[:<val>]	End link and execute program
<name>/H	Save the named .HEX (Intel hex) file
<name>/L	Load the named .SYM table
/M	List the symbol table (both defined and undefined)
<name>/N	Save the named .COM (executable) file
/O:<sym>=<hex>	Origin the symbol at the designated location
/P:<hex>	Select loading base for PROGram area
/Q	Quit linker and return to operating system
/R	Reset drives (allows one to change disks)
<name>/S	Search the named library for undefined symbols
/U	List the undefined symbols
<name>/X	Save the named .HEX (Intel hex) file
<name>/Y	Save the named .SYM (symbol table) file
/Z	Resets the linker to initial state

NOTE: Output files can be generated and closed using a single name and any combination of /N, /X, /Y, /R or /H as in MYFILE/Y/E, MYFILE/X, or the works as in MYFILE/H/R/Y/N/E.

G. Microsoft Relocatable .REL Format

The MicroSoft relocatable (.REL) format is a bit oriented sequential file. Fields extracted may or may not be aligned on byte boundaries, except as indicated by special link item 14.

The first bit of an item indicates its type: '0' indicates an absolute byte and '1' indicates a relocatable type. If the first bit is a '0', the next 8 bits are loaded as an absolute byte. If the first bit is a '1', the next 2 bits indicate the relocatable type:

00	Special link item
01	Program Relative. The next 16 bits are added to the current Program base and loaded as two absolute bytes.
10	Data Relative. The next 16 bits are added to the current Data base and loaded as two absolute bytes.
11	COMMON Relative. The next 16 bits are added to the current COMMON base and loaded as two absolute bytes.

Special link items are selected via the next four bits giving special link the range from 0 to 15 decimal. With the exception of 'End of File' (#15), special link items consist of either an **A-field**, **B-field**, or both. If both fields exist, the A-field precedes the B-field.

An **A-field** is an address field which consists of 18 bits: the first 2 bits indicate the relocation type and the remaining 16 bits define an address. The first 2 bits define the relocatable type similar to that described above with the exception that '00' indicates an absolute address (no offsets are added).

A **B-field** is usually a symbol (except for some extension link operations) which is a variable field length from 11 to 67 bits. The first three bits indicate the symbol length: a value from zero to seven. If you assume there are no zero length symbols, zero can be treated as eight giving lengths from 1 to 8 characters. (NOTE: MicroSoft uses the concept of blank or unnamed COMMONs which are zero length. Link-Z does *not* support zero length symbols and, instead, treats a value of zero as a symbol length of eight.)

Special Link Items

The Special Link description shows the decimal value of the Special Link Item on the left with a description on the right.

Special link items from 0 to 4 only have a B-field.

0	Entry Symbol.	This item is used to inform the linker that the given symbol appears somewhere in this module. Link-Z uses this during library searches (/S) to determine whether to load this module.
1	Select COMMON block.	This selects the previously defined COMMON block for any subsequent relocatable COMMON (11) references.
2	Program Name.	This is the module name and is only used by Link-Z for reference (during error messages).
3	Request Library.	This is the library name to be searched at the end of the link if any unresolved symbols still exist. Link-Z stores up to 8 pending library requests and ignores the rest.
4	Extension Link Item.	Refer to the Extension Link section following this description of Special Link items.

Special link items from 5 to 7 have both an A-field and a B-field.

5	Define COMMON Size.	This item declares a COMMON block and its size. The 2 bit type in the A-field is ignored.
6	Chain External.	An external symbol (B-field) is declared with the head of the address chain defined in the A-field. A chain is a linked list with the value at each address pointing to the next address in the list. An address value of 0000 indicates the end of the chain. When the external value is defined, each address in the chain is replaced by the actual address of the symbol.
7	Define Entry Point.	This item declares a global symbol and its value.

Special link items from 8 to 14 only have an A-field.

8	External – Offset	The A-field value is subtracted from the 16 bit value at the load pointer (the following 16 bits) AFTER all modules have been loaded, globals resolved, etc.
9	External + Offset	The A-field value is added to the 16 bit value at the load pointer (the following 16 bits) AFTER all modules have been loaded, globals resolved, etc.
10	Define Data Size	Defines the size of the Data section of the module. Used by Link-Z to allocate space for the Data segment when shared with the Program segment.
11	Set Load Pointer	The A-field is the new location to load subsequent values.
12	Chain Address	The A-field is the head of a chain (as described in Chain External). Each address in the chain is replaced by the current load address.
13	Define Program Size	The A-field defines the size of the Program segment for this module.
14	End of Module	This item indicates the end of the current module. If the A-field value is not zero (absolute), then it indicates the start address. End of module is used by library searches (/S) to resume search mode after loading the module. NOTE: End of Module forces the input bit sequence to a byte boundary.

The last special link item does not have any fields.

15	End of File	Identical to the EOF (26 decimal, 1AH) character for ASCII files. Link-Z terminates operations on this file.

Extension Link Items

Extension link operations include the original items defined by MicroSoft as well as some useful extensions. Since extension link items are located within a B-field which produces from 1 to 8 characters, the first character indicates the function as follows:

'A'	Arithmetic operation (described below).
'B'	The remaining characters (from one to seven) define an external symbol whose value is pushed onto the expression stack.
'C'	The next three bytes define a 16 bit value to be pushed onto the stack. The lowest order 2 bits of the first byte define the type (absolute, program, data, or COMMON) with the remaining two bytes forming the 16 bit value.
'Z'	This is a MICROCode Consulting extension to indicate that an error occurred during the compilation or assembly process which generated this .REL file.

There are a few other extensions by MicroSoft as well as other assemblers which might be there. However, Link-Z only recognizes the above commands and will abort with an error if it cannot recognize the entry.

Extension Link Arithmetic Operations

For type 'A' extension link items, the next character defines the operation to occur. All math operations occur on an internal expression stack which is constantly checked for overflow and underflow.

Hex Value	Operation	Description
01	Pop Byte	A 16 bit value is popped off the stack and checked for a byte range (high byte is either a 00 or an FFH). The low byte is stored at the current location counter.
02	Pop Word	A 16 bit value is popped off the stack and stored at the current location counter.
03	High Byte	A 16 bit value is popped off the stack and its high byte is pushed back on as a 16-bit value (high 8 bits are zero).
04	Low Byte	Same as High Byte except the low byte is pushed back onto the stack.
05	Bitwise NOT	Same as High Byte except the 1's complement of the popped 16 bit value is pushed back onto the stack.
06	Negation	Same as Bitwise NOT except the 2's complement of the value is used.
07	Subtraction	The first 16 bit value popped off the stack is subtracted from the second 16 bit value popped. The result is pushed back onto the stack.
08	Addition	Same as Subtraction except the values are added.
09	Multiplication (unsigned)	Same as Subtraction except the values are multiplied.
0A	Division (unsigned)	Same as Subtraction with the first value popped being the divisor and the second value popped being the dividend.
0B	Modulus (unsigned)	Same as Division except the remainder is pushed back onto the stack instead of the result.
10	Bitwise AND	Same as Subtraction except the values are logically ANDed. (An extension by SLR Systems.)
11	Bitwise OR	Same as Subtraction except the values are logically ORed. (An extension by SLR Systems.)
12	Bitwise XOR	Same as Subtraction except the values are logically XORed. (An extension by SLR Systems.)
13	Bitwise SHR	Same as Subtraction except the first value popped is the amount to shift right and the second value popped is the value to be shifted. The high order bits are zero filled (logical shift). (An extension by SLR Systems.)
14	Bitwise SHL	Same as Bitwise SHR except the value is shifted left (zero fill). (An extension by SLR Systems.)

No other Extension Link Arithmetic Operations are supported.

H. Intel Hex Format

An Intel standard .HEX file contains one or more records which have the following format:

```
:nnaaaattdd.........ddxx
```

where 'nn' is the number of data bytes
'aaaa' is the starting address
'tt' is the record type (normally 00 , but 01 for the last record)
'dd' are the data bytes starting from address 'aaaa' and working upward in memory
'xx' is the checksum byte. The sum of this byte and all other bytes on the line (except the colon) should add up to zero.

All hexadecimal numbers are two- or four-letter sequences from high-to-low nibble as in 'dd' or 'aaaa'. (A nibble is 1/2 of a byte - 4 bits.) Thus, '10' would be 16 decimal.

Note that every line begins with a colon (:) and ends with a carriage return/line feed sequence.

A record length of zero ('nn') indicates that the execution address follows. 'nn' equal to zero also indicates end of file.