CHAPTER 8

USING THE PROGRAMMING SUPPORT UTILITIES

This chapter describes each of the programming support utilities packaged with the C cross compiler targeting **Z80/HD64180.** The following utilities are available:

hex80	translates object module format
lby	builds and maintains libraries
lm	correlates lines among input files
lord80	orders library routines
pr	prints source files
rel80	examines objects modules
toprom	automatic data initialization
unhex	retranslates object module formats

The assembler is described in Chapter 5. The linker is described in Chapter 6. Support for debugging is described in Chapter 7.

The description of each utility tells you what tasks it can perform, the command line options it accepts, and how you use it to perform some commonly required operations. At the end of the chapter are a series of examples that show you how to combine the programming support utilities to perform more complex operations.

The hex80 Utility

You use the **hex80** utility to translate executable images produced by **lnk80** to one of several hexadecimal interchange formats. These formats are: Intel standard hex format, Motorola S-record format, Tektronix standard hex format, and Tektronix extended hex format. You can also use **hex80** to override text and data biases in an executable image or to output only a portion of the executable.

The executable image is read from the input file <file>. If no <file> is specified, or if the filename specified is -, the file **xeq.80** is read.

Command Line Options

hex80 accepts the following command line options, each of which is described in detail below:

```
hex80 -[c## db## dr h m* n*^ o* r## s2 s tb## tk
tx +## #] <file>
```

-c## output only ## bytes.

-db## when processing a standard object file as input, override the object module data bias with ##.

-dr output text records as record type 0x81, and data records as record type 0x82, for use with the Digital Research **gencmd** utility under CP/M-86. This option is used only with Intel standard hex format.

-h do not produce start record for Intel standard hex format, or S0 record for Motorola S-record format.

-m* insert the string *, instead of <file>, into the Motorola S0 record. This option is used only with Motorola S-record format.

-n* output only segments whose name is equal to the string *. Up to ten different names may be specified on the command line. If there are several segments with the same name, they will be all produced.

-o* write output module to file *. The default is STDOUT.

-r## interpret the input file as a "raw" binary image and not as an object file. Output is produced as described below, in either format, except that the starting address of the module is specified by the long integer ##.

-s2 produce Motorola S-record format, using only S2 records as described below.

-s produce Motorola S-record format.

-tb## when processing a standard object file as input, override the object module text bias with ##.

-tk produce Tektronix standard hex format.

-tx produce Tektronix extended hex format.

+## start output with the ##th byte. You use this option in combination with the -# option described below. For example, the sequence -4 +3 produces bytes 3, 7, 11, 15, ...; the sequence -2 +0 produces all even bytes; -2 +1 outputs all odd bytes. The default is +0, which outputs all bytes.

-# output every #th byte. You use this option in combination with the +# option described above. A value of -2 outputs every other byte, -4 outputs every fourth byte; 1 is the default, which outputs all bytes.

At most one of the options **-dr, -s, -s2, -tk,** or **-tx** may be specified concurrently, except that **-s2** and **-s** may be specified together, and **-s2** implies the **-s** option. If no format is specified, the default is Intel standard hex format. If **-r##** is specified, **-db##** or **-tb##** may not be used. The default is to treat input as a standard object file.

When the input file is a standard object file, the load offset of the first record output for its text or data segment is determined as follows. If an offset is explicitly specified for that segment (with **-db##** or **-tb##**), it is used. If all bytes of the input file are being output, the appropriate bias is used from the object file header. Otherwise, a load offset of zero is used. Then, the value specified with **+##** is added to the chosen offset to obtain the first offset actually output.

Intel Standard Hex Format

A file in Intel hex format consists of the following records, in the order listed:

1) a '$' character alone on a line to indicate the end of the (nonexistent) symbol table. If **-h** is specified, this line is omitted.

2) data records for the text segment, if any. These represent 32 image bytes per line, possibly terminated by a shorter line.

3) data records for the data segment, if any, in the same format as the text segment records.

4) extended address records, if any. These records are output only if the load address exceeds 0xffff, and may occur at any point in the records that are output.

5) an end record specifying the start address.

Data records each begin with a ':' character and consist of pairs of hexadecimal digits, each pair representing the numerical value of a byte. The last pair is a checksum such that the numerical sum of all the bytes represented on the line, modulo 256, is zero. The bytes on a data record line are:

a) the number of image bytes on the line.

b) two bytes for the load offset of the first image byte. The offset is written more significant byte first so that it reads correctly as a four-digit hexadecimal number.

c) a one byte constant "00".

d) the image bytes in increasing order of address.

e) the checksum byte.

Extended address records each begin with a ':' character and are written as digit pairs with a trailing checksum. An extended address record defines the value of a segment base address (SBA) which remains in effect until another extended address record is encountered. The starting value of the SBA initially defaults to zero.

The absolute memory address of a content byte in subsequent data records is obtained by adding the SBA times sixteen, plus the load offset of the data record, plus the index of the byte in the data record. **hex80** will never produce a data record whose load offset plus any data byte index exceeds 64K, thus avoiding any wraparound problems on programs that read this form of output.

The bytes on an extended address record line are:

a) a one byte constant "02" for the number of image bytes on the line.

b) a two byte constant "0000" for the load offset of the first image byte.

c) a one byte constant "02" for the record type.

d) two bytes for the SBA, which is an absolute memory address divided by sixteen. The address is written more significant byte first so that it reads correctly as a four-digit hexadecimal number.

e) the checksum byte.

An end record also begins with a ':' character and is written as digit pairs with a trailing checksum. Its format is:

a) a one byte constant "00".

b) two bytes for the start address, in the same format as the load offset in a data record. The start address used is the first load offset output for the file.

c) a one byte constant "01".

d) the checksum byte.

Motorola S-Record Format

A file in Motorola S-record format is a series of records each containing the following fields:

<S field><count><addr><data bytes><checksum>

All information is represented as pairs of hexadecimal digits, each pair representing the numerical value of a byte.

<S field> determines the interpretation of the remainder of the line; valid S fields are "S0", "S1", "S2", "S8", "S9". <count> indicates the number of bytes represented in the rest of the line, so the total number of characters in the line is <count> * 2 + 4.

<addr> indicates the byte address of the first data byte in the data field. S0 records have two zero bytes as their address field; S1 and S2 records have <addr> fields of two and three bytes in length, respectively; S9 and S8 records have <addr> fields of two and three bytes in length, respectively, and contain no data bytes. <addr> is represented most significant byte first. If an <addr> value is small enough to fit in two bytes, and if the **-s2** option is not specified, S1 and S9 records will be output, instead of the longer S2 and S8 records.

The S0 record contains the name of the input file, formatted as data bytes. If input was from **xeq.80**, XEQ is used as the name. If the **-h** option is specified, the S0 record is omitted. S1 and S2 records represent text or data segment bytes to be loaded. They normally contain 32 image bytes, output in increasing order of address; the last record of each segment may be shorter. The text segment is output first, followed by the data segment. S9 records contain only a two-byte start address in their <addr> field; S8 records contain a three-byte address. The start address used is the first load offset output for the file.

<checksum> is a single byte value such that the numerical sum of all the bytes represented on the line (except the S field), taken modulo 256, is 255 (0xFF).

Tektronix Standard Hex Format

A file in Tektronix standard hex format consists of the following records, in the order listed:

1) data records for the text segment, if any. These represent 30 image bytes per line, possibly terminated by a shorter line.

2) data records for the data segment, if any, in the same format as the text segment records.

3) an end record specifying the start address.

Data records each begin with a '/' character and consist of pairs of hexadecimal digits, with each pair representing the numerical value of a byte. The bytes on a data record line are:

a) two bytes for the load offset of the first image byte. The offset is written more significant byte first so that it reads correctly as a four-digit hexadecimal number.

b) a byte count, which is the number of image bytes on the line.

c) a checksum byte calculated by taking the numerical sum of all the characters in the load address and the byte count, modulo 256.

d) the image bytes in increasing order of address.

e) a checksum byte for the image bytes calculated by taking the numerical sum of all the characters in the image bytes represented on the line modulo 256.

An end record also begins with a '/' character and is written as digit pairs with a trailing checksum. Its format is:

a) two bytes for the start address, in the same format as the load offset in a data record. The start address used is the first load offset output for the file.

b) a one byte constant "00".

c) the checksum byte calculated by taking the numerical sum of all the characters in the start address and the one byte constant, modulo 256.

Tektronix Extended Hex Format

A file in Tektronix extended hex format consists of the following records, in the order listed:

1) data records for the text segment, if any. These represent 32 image bytes per line, possibly terminated by a shorter line.

2) data records for the data segment, if any, in the same format as the text segment records.

3) an end record specifying the start address.

Data character records each begin with a '%' character and consist of pairs of hexadecimal digits, each pair representing the numerical value of a byte. The exceptions to this pairing rule are the record type, which is a single character, and the variable length load offset, which may or may not be digit pairs depending on the value of the load offset. The bytes on a data record line are:

a) a record length byte, which is the number of characters in the record, not including the leading '%' or the end-of-line character.

b) a record type character, **6** for data records.

c) a checksum byte calculated by taking the numerical sum of all the characters in the record length, record type and image bytes, modulo 256.

d) a variable length load offset of the first image byte. The length can be from two to nine characters, where the first character specifies the number of characters which follow. The offset is written more significant byte first so that it reads correctly as a four-digit hexadecimal number.

e) the image bytes in increasing order of address.

An end record also begins with a **%** character and is written as digit pairs, except for the the record type, which is a single character, and the variable length start address, which may or may not be digit pairs depending on the value of the start address. Its format is:

a) a record length byte.

b) a record type character, **8** for end records.

c) a checksum byte.

d) a variable length start address, in the same format as the load offset in a data record. The start address used is the first load offset output for the file.

Return Status

hex80 returns success if no error messages are printed; that is, if all records are valid and all reads and writes succeed. Otherwise it returns failure.

Examples

The file **hello.c**, consisting of:

```
    char *p {"hello world"};
```

when compiled produces the following Intel standard hex format:

```
hex80 hello.o
$
:0E000000020068656C6C6F20776F726C640094
:00000001FF
```

and the following Intel extended hex format:

```
hex80 -db0xfff6 hello.o
$
:020000020FFFEE
:0E000600020068656C6C6F20776F726C64008E
:00000001FF
```

and the following Motorola S-record format:

```
hex80 -s hello.o
S00A000068656C6C6F2E6F44
S1110000020068656C6C6F20776F726C640090
S9030000FC
```

and the following Tektronix standard hex format:

```
hex80 -tk hello.o
/00000E0E020068656C6C6F20776F726C64009E
/00000000
```

and the following Tektronix extended hex format:

```
hex80 -tx hello.o
%236AA10020068656C6C6F20776F726C6400
%0781010
```

The lby Utility

lby builds and maintains object module libraries in either standard Whitesmiths library format or the format used by UNIX/System III, UNIX/V6, or UNIX/V7. **lby** can also be used to collect arbitrary files in one place. <lfile> is the name of an existing library file or, in the case of replace or create operations, the name of the library to be constructed.

Command Line Options

lby accepts the following command line options, each of which is described in detail below:

```
lby -[c d i p r t v3 v6 v7 v x] <lfile> <files>
```

-c create a library containing <files>, in the order specified. Any existing <lfile> of the same name is removed before the new one is created.

-d delete from the library the zero or more files in <files>.

-i take <files> from STDIN instead of the command line. Any <files> listed on the command line are ignored.

-p print named files to STDOUT. If no files are named, all files are printed.

-r in an existing library, replace the zero or more files in <files>. If no library <lfile> exists, create a library containing <files>, in the order specified. The files in <files> not present in the library are appended to it, in the order specified.

-t list the files in the library in the order they occur. If <files> are specified, then only the files named and present are listed.

-v3 create the output library in UNIX/System III format, VAX-11 byte order. This option is meaningful only in combination with the -c or -r options.

-v6 create the output library in UNIX/V6 format. This option is meaningful only in combination with the -c or -r options.

-v7 create the output library in UNIX/V7 format, PDP-11 byte order. This option is meaningful only in combination with the **-c** or **-r** options. Its use is mandatory when specifying the **-t** and **-x** options. UNIX/System III format is assumed if this option is not specified.

-v be verbose. The name of each object file operated on is output, preceded by a one-character code indicating its status: **c** for files retained unmodified, **d** for those deleted, **r** for those replaced, and **a** for those appended to <lfile>. If **-v** is used in combination with **-t**, each object file is listed, followed by its length in bytes.

-x extract the files in <files> that are present in the library into discrete files with the same names. If no <files> are specified, all files in the library are extracted.

At most one of the options **-[c d p r t x]** may be specified concurrently. If none of these is specified, the **-t** option is assumed. Similarly, at most one of the options **-[v3 v6 v7]** should be present. If none of these is present, standard Whitesmiths library file format is assumed when a new library is created by passing the **-r** or **-c** options. An existing library is processed in its proper mode, except for UNIX/V7 libraries, for which the option **-v7** must be specified.

If a filename in the list <files> contains any of the characters ':', ']', or '/', the filename actually recorded or searched for in the library is formed from the longest filename suffix that does not contain a ':', ']', or '/'.

The Whitesmiths standard library format consists of a two byte header having the value 0177565, written less significant byte first, and followed by zero or more entries. Each entry consists of a 14 byte name, NUL-padded, followed by a two byte unsigned file size, also stored less significant byte first, followed by the file contents proper. If a name begins with a NUL byte, it is taken as the end of the library file.

Note that this differs in several small ways from UNIX/V6 format, which has a header of 0177555, an eight byte name, six bytes of miscellaneous UNIX-specific file attributes, and a two byte file size. In addition, a file of odd length is followed by a NUL-padding byte in UNIX format, while no padding is used in Whitesmiths standard library format.

UNIX/System III and UNIX/V7 formats are characterized by a header of 0177545, a 14 byte name, eight bytes of UNIX-specific file attributes, and a four byte size. The size is in VAX-11 native byte order for UNIX/System III and in PDP-11 native byte order for UNIX/V7. Odd length files are also padded to an even boundary.

Return Status

lby returns success if no problems are encountered. Otherwise it returns failure. After most failures, an error message is printed to STDERR and the library file is not modified. Output from the **-t** option, and verbose remarks, are written to STDOUT.

Examples

To build a library and check its contents:

```
lby -c libc one.o two.o three.o
lby -tv libc
```

Special Usage Considerations

If all modules are deleted from a library, a file of zero length remains.

Modifying UNIX/System III, UNIX/V6, or UNIX/V7 format files causes all attributes of all file entries to be zeroed.

lby doesn't check for files too large to be properly represented in a library (i.e. files longer than 65,534 bytes). Problems here stem from the fact that the file length code in Whitesmiths and UNIX/V6 formats is only two bytes long. To process large files, use the **-v3** option to specify UNIX System III format, which has a 4 byte length code.

The lm Utility

lm has features which allow it to be used by the C cross compiler in combination with the **pr** utility to generate merged C and assembly language source listings. **lm** examines each of the files it accepts as input for commands to itself.

lm writes to STDOUT a copy of the lines contained in the input files specified by <files>, replacing any occurrence of the sequence

#line= <filename>:<linenumber>

with the appropriate line or range of lines (from the last line printed to the specified line) from the "source" file named filename. A "source" file is any file referred to by **#line** and **#error** sequences embedded in one or more files specified on the **lm** command line. If that line has already been printed, no replacement occurs. If more than one <files> is specified on the **lm** command line, each is initially scanned, in order of appearance on the command line, for the sequence described. The order in which the files are further scanned depends primarily on line number orderings. If no files are specified, or if '-' appears as a filename, then input is expected from STDIN. At most one copy of a given filename/linerange combination will be inserted into a given output.

The sequence

#error <program> <filename>:<linenumber>'\t'<text>
is also recognized by **lm,** which replaces it with the appropriate line or range of lines from the specified source file (**'\t'** refers to an ASCII tab character). The **#error** sequence will cause **lm** to output the string **~~e** to the output file the first time that a **#error** sequence is encountered. The #error sequence also forces **lm** to assert that all lines up to <linenumber> have been copied to STDOUT. This sequence is also recopied to the output rather than replaced, (as **#line=** directives normally are). (**Note:** This only occurs the first time the original sequence of lines is encountered among the files named.)

By default (i.e. if -**err** is not specified), **lm** reports any error diagnostics immediately (either from the input file or generated internally) and changes its return status to failure.

Command Line Options

lm accepts the following command line options, each of which is described in detail below:

lm -[c* err e* f l o* s t w#] <files>

-c* Force each "source" line copied to STDOUT to be prefixed by the string '*' as a comment delimiter.

-err Pass error diagnostics from input file to output file, add error diagnostics generated to output file, and report success. Only fatal errors (such as being unable to write an output file due to lack of space) will be reported and cause the return status to change to failure.

-e* Force each **#error** line copied to STDOUT to be prefixed by '*' as a comment delimiter.

-f For use with a foreign assembler, this option suppresses outputting of the string **~~e** as described above, counts the number of **#error** lines, and outputs the number of high-level errors at the end of the listing.

-l Number each "source" line copied to STDOUT. Each "source" file is numbered independently and the line number appears after the comment string (which may be null).

-o* Send output to the file '*' rather than to STDOUT.

-s Strip all **#error** and excess **#line=** sequences from the files **lm** accepts as input.

-t Strip only the excess **#line=** sequences from the files **lm** accepts as input.

-w# If source output line must wrap around, start new line at column #.

Return Status

lm returns success if it can open its output file and all source files, or if the -err option is specified.

Examples

To get a listing of C interspersed with assembly language:

```
c -dlistcs +o echo.c
```

Special Usage Considerations

The line ranges are restricted to be monotonically increasing in value between files.

The lord80 Utility

You use the **lord80** utility to organize your object module libraries. **lord80** reads in a list of module names (with associated interdependencies) from STDIN, and outputs to STDOUT a topologically sorted list of module names such that, if at all possible, no module depends on an earlier module in the list. Alternatively, **lord80** can be used (via the **-o*** option described below) to create a library directory that will contain size, location, and interdependency information about the modules. This allows libraries to be built using a set of modules that have circular dependencies.

Each module is introduced by a line containing its name, followed by a colon. Subsequent lines are interpreted as either:

DEFs - things defined by the module, or

REFs - things referred to by the module.

Anything not recognizable as a module, a REF, or a DEF is considered undefined and will cause **lord80** to emit warning messages.

REFs and DEFs have the syntax given by one or more format specifiers entered as options on the command line. Each character of the format specifier must match the corresponding character at the beginning of an input line; a **?** will match any character except newline. If all the characters of the format specifier match up, then the rest of the input line is taken as a REF or DEF name. Thus, the format option "**-d0x????D** " (note the trailing space after the **D** in all cases) would identify as a valid DEF any line beginning with **0x**, four arbitrary characters, and a **'D'** followed by a space, so that the input line **0x3ff0D _inbuf** would be taken as a DEF named **_inbuf**.

Command Line Options

lord80 accepts the following command line options, each of which is described in detail below:

```
lord80 -[c* d*^ i o* r*^ s]
```

-c* prepend the string * to the output stream. Using **-c*** implies the use of the -s option as well. Each module name output is preceded by a space; the output stream is terminated with a newline. Therefore, **lord80** can be used to build a command line.

-d* use the string * as a format for DEFs. Up to ten **-d*** options may be specified.

-i ignore anything that is not a DEF or a REF. The default is to emit a warning about any line not recognizable as a DEF or REF.

-o* create a library directory with the string '*' as a name. The default name is **dir.**

-r* use the string '*' as a format for REFs. Up to ten **-r*** options may be specified.

-s suppress output of DEFs and REFs; output only module names in order.

If no **-d** options are specified, **lord80** uses the following default DEF formats (note the trailing space after each):

```
    "0x????????B "
    "0x????????D "
    "0x????????G "
    "0x????????T "
    "0x????B "
    "0x????D "
    "0x????G "
    "0x????T "
```

If no **-r** options are specified, **lord80** uses the default REF formats (note the trailing space after both):

```
    "0x????????U "
    "0x????U "
```

All the default REF and DEF format specifiers are compatible with the hexadecimal output of **rel80.**

In general, **lord80** makes rearrangements only when necessary, so an ordered set of modules should pass through **lord80** unchanged. When creating a directory, the order of the modules is never changed. If the **-o*** directory option is not specified, and if there are circular dependencies among the modules, **lord80** writes the name of the file that begins the unsorted input list to STDERR, followed by the message "not completely sorted". **lord80's** output in this case is a partially sorted list.

RETURNS

lord80 returns success if no error messages are printed. Otherwise it returns failure.

Examples

To create an ordered library of object modules:

```
rel80 file1.o file2.o file3.o | lord80 -s | lby -ci libx.a
```

To create a directory-controlled library of object modules:

```
rel80 -du +size *.o | lord80 -so libxdir
lby -c libx.a libxdir file1.o file2.o file3.o
```

The pr Utility

You use **pr** to obtain a listing of your C and assembly language source files on your host system. **pr** also has features which allow it to be used by the C cross compiler in combination with the **lm** listing merger utility to generate more readable listings. **pr** examines each line of the files printed for special commands to itself.

Standard Output Format

pr prints to STDOUT the contents of the files in the list <files>, adding a title and empty lines for page breaks, and padding to an integral number of pages. **pr** understands three basic kinds of embedded commands, each of which must begin with an "attention sequence" (described below).

If you do not specify any <files>, **pr** takes input from STDIN. A filename of - also causes STDIN to be read, at the point in the list of files where the - appears. Each page **pr** prints has a five line heading containing two empty lines, a title line, a subtitle line, and another empty line, along with a five line footer.

The standard title consists of the last modification date of the file being output (where available), its name, and the current page number. When you specify a title or when STDIN is the file being printed, the current date and time are used instead of a modification date. The standard subtitle is an empty line.

The default attention sequence is ~~ (two ASCII "tilde" characters). It can be changed via the **-attn*** option, as described below. A line on which only the attention sequence appears (~~ if the default is in effect) is taken as a request to begin a new page (a newpage command). If a line begins with the attention sequence followed by a **1**, (~~1 if the default is in effect) everything between the command sequence and the next ASCII newline character is taken as a new title for the page header, to be printed at the top of the next page printed. If a line begins with the attention sequence followed by a **2**, (~~2 if the default is in effect) everything between the command sequence and the next ASCII newline character is taken as a new subtitle, to be printed below the header at the top of the next page printed. If a line begins with the attention sequence followed by an **e**, (~~e if the default is in effect), and the **-err** option is specified, **pr** will by default create a file called **nolink.e**. This default can be overridden by setting the environment variable CERRFILE to the new file name.

Command Line Options

pr accepts the following command line options, each of which is described in detail below:

```
pr -[attn* c# err f# h it# l# m n ot# p s? t* w# +## ##]
<files>
```

-attn* set the attention sequence to *. The default attention sequence is ~~.

-c# print each file in # columns. The default is 1 column.

-err pass error diagnostics from input file to output file, add error diagnostics generated to output file, and report success. Only fatal errors (such as not being able to write an output file due to lack of space) will be reported and cause the error status to change to failure.

-f# set the number of the first page printed to #.

-h suppress headings and footings on output.

-it# space incoming tabs (tabs in the input files) at intervals of #. The default is 8 spaces between tabstops (4 spaces between tabstops under IDRIS). The **-it0** option suppresses input detabbing.

-l# print pages # lines in length. The default is 66 lines per page.

-m print all files simultaneously, each in a separate column.

-n number the output lines.

-ot# entab the output assuming a tabstop every # columns. The default is 0, meaning that output is not entabbed unless otherwise specified.

-p print one page of output and pause, waiting for input from STDIN.

-s? use a single **?** to separate multiple columns of output, instead of whitespace. **?** always matches the next argument character, if any, or a NUL character.

When this option is specified, entabbing is suppressed.

-t*
: use *** as the filename field of the heading title. Note that any ~~? sequences encountered by **pr** will override this option.

-w#
: specifies a page width of *#* positions. The default is 72.

+##
: start output of each file at page *##*. The default is to start output at page 1.

-##
: stop output of each file after page *##*. The default is huge.

At most one of the options **-c#** and **-m** may be specified concurrently.

By default (i.e. if the **-err** option is not specified), **pr** reports any error diagnostics immediately (either from the input file or generated internally) and changes the return status to failure.

Return Status

pr returns success if no error messages are written to STDERR, or if the **-err** option is specified.

Examples

To number the output lines of a file:

```
pr -n file1
```

```
Tue Jan 12 15:15:56 1988  file1  Page 1


    1       file 1 line 1
    2       file 1 line 2
    3       file 1 line 3
    4       file 1 line 4
    5       file 1 last
```

To add a special filename field to the heading title:

```
pr -t"pr OUTPUT" file2
```

```
Tue Jan 12 15:24:44 1988 pr OUTPUT  Page 1

file 2 line 1
file 2 line 2
file 2 line 3
file 2 line 4
file 2 line 5
file 2 line 6
file 2 line 7
file 2 line 8
file 2 line 9
file 2 last
```

The rel80 Utility

You use **rel80** to inspect library directories and relocatable object files in standard format. Such files may have been output by an assembler, output as a directory by **lord80**, combined by **lnk80**, or archived by **lby.** **rel80** can be used to check the size and configuration of relocatable object files or to output information from their symbol tables.

Command Line Options

rel80 accepts the following options, each of which is described in detail below.

```
rel80 -[a +dec +dir d g +in i o +seg +size s t u v] <files>
```

<u><files></u> specifies zero or more files, which must be in relocatable format, library directory format, standard library format, UNIX/V6 or UNIX/V7 library format, or UNIX/System III library format. If more than one file (or a library) is specified, then the name of each separate file or module precedes any information output about it. Each name is followed by a colon and a newline. If -s is specified, a line of totals is also output. If no <u><files></u> are specified, or if '-' is encountered on the command line, **xeq.80** is used as the input file.

-a sort output in alphabetical order by symbol name. Use of the **-d** option below is implied.

+dec output symbol values and segment sizes in decimal. The default is hexadecimal.

+dir be verbose about processing library directories. The default behavior is to mention only that a directory was encountered.

-d output all defined symbols in each file, one symbol per line. Each line contains the value of the symbol, a "relocation code" indicating what the value is relative to, and the symbol name. Values are output as the number of digits needed to represent an integer on the target machine. The meaning of the relocation codes in the output are: 'T' for text relative, 'D' for data relative, 'B' for bss relative, 'A' for absolute, 'G' for debug relative, and '?' for anything **rel80** doesn't recognize. Lowercase letters are used to represent local symbols; uppercase is used for globals.

-g print global symbols only.

+in take <u><files></u> from STDIN and append them to the command line.

-i print all global symbols, with the interval in bytes between successive symbols shown in each value field. Specifying -i implies the use of the **-d**, **-u**, and **-v** options as well.

-o output symbol values and segment sizes in octal. The default is decimal.

+seg output the bias, offset, and size of each segment in a multisegment object file.

+size display the file size, in decimal. This allows **rel80** to produce all the information **lord80** needs to create a library directory. For this purpose, **rel80** must be invoked with the options **-hdu** and **+size.** As mentioned below, **-s** must not be used when creating a directory.

-s display the sizes (in decimal) of the text segment, the data segment, the debug segment, the bss segment, and the space reserved for the runtime stack plus heap, followed by the sum of all their sizes. If the output of **rel80** is to be passed to **lord80** to create a library directory or sort the object modules, the **-s** option must not be used.

-t list type information for this file. For each object file the data output is as follows: the size of an integer on the target machine, the target machine byte order, whether even byte boundaries are enforced by the hardware, whether the text segment byte order is reversed from that of the data segment, and the maximum number of characters permitted in an external name. If the file is a library, the type of library is output and the information above is specified for each module in the library.

-u list all undefined symbols in each file. If **-d** is also specified, each undefined symbol is listed with the code **'U'** for symbols that are undefined. The value of each symbol, if non-zero, is the space to be reserved for it at load time if it is not explicitly defined.

-v sort by value within segments; implies the use of the **-d** option above. Symbols of equal value are sorted alphabetically. Absolute symbols are presented first, followed by text relative, data relative, and bss relative symbols.

If none of the symbol specification options **-[d g u]** or the information request options **-[+dir +seg +size s t]** are specified, the default is **-[d u]**; i.e., all symbols are listed in the order that they are encountered in the symbol table. If more than one of the options **-[d +size s t u]** is selected, then type information is output first, followed by segment sizes, followed by the symbol list specified with **-d** or **-u**.

Return Status

rel80 returns success if no diagnostics are produced (i.e. if all reads are successful and all file formats are valid).

Examples

To obtain an alphabetical list of all symbols in a module:

```
rel80 -ha alloc.o
0x0074T _alloc
0x0000U _exit
0x01feT _free
0x00beT _nalloc
0x0000U _sbreak
0x0000U _write
```

The toprom Utility

You use the **toprom** utility to modify executable images produced by **lnk80** in order to produce a romable executable with automatic data area initialization.

toprom first locates a text segment behind which it will add the data image. The default behavior is to select the first text segment in the executable file, but you can override this by marking one text segment by a no init attribute. This is done with the -a option of the linker, which is followed by the numerical value of the attribute. The standard value for a text segment is 5, (executable and readable), so you need to provide this segment with the new value 13 (executable, readable and not initialized). This extra bit will be used by **toprom** to locate the selected text segment.

Then, **toprom** looks in the executable file for initialized data segments (not executable and initialized). Once located, it builds a descriptor containing the starting address and length of each located data segment, and move the descriptor followed by data segments at the end of the selected text segment, without relocating the content of the data segments.

The executable image is read from the input file <file>. If no <file> is specified, or if the filename specified is -, the file **xeq.80** is read.

Command Line Options

toprom accepts the following command line options, each of which is described in detail below:

```
toprom -[o*] <file>
```

-o* write output module to file *. There is no default name.

Descriptor Format

The created descriptor has the following format:

```
    .word start_prom_address    ; starting address of the
                                ; first data image in prom
for each data segment:
    .byte flag                      ; type of data
    .word start_ram_address         ; starting address of data
                                    ; in ram
    .word end_prom_address          ; address of last data byte
                                    ; plus one in prom
after the last segment:
    .byte 0
```

The flag byte is used to detect the end of the descriptor, and also to give a type to the data segment. The actual value is equal to the code of the first letter in the segment name. Refer to the linker documentation to see how to modify the segment name and attribute.

The end address in prom of one segment gives also the starting address in prom of the following segment, if it exists.

In order to use that table, the program has to know where the descriptor is; all the information needed is in it. This has to be done in the linking phase by defining a symbol with the +def option at the correct location, i.e. the end of the selected text segment, or at the end of the first one by default, such as:

```
    +def symbol=__text__
```

symbol will be equal to the first memory location after the previous segment, so it is necessary to put this definition at the end of the modules list of the selected segment. Thus, symbol will be used by the special (provided) startup program to locate the descriptor, and to use its content to copy data from prom to ram.

toprom does not check that there is enough space after the selected text segment to move all the data segments. This can and should be checked after by inspecting segment sizes with the **rel80** utility.

The unhex Utility

You use **unhex** to translate Intel standard hex format or Motorola S-records back to the original object image. The ASCII records are read from the file named <file>. If no <file> is specified, or if the filename specified is '-', STDIN is read.

Command Line Options

unhex accepts the following options, each of which is described in detail below:

```
unhex -[c e o* r## s x] <file>
```

-c do not check checksums. By default the checksum is performed, and errors noted.

-e do not halt on error. By default, errors cause **unhex** to terminate.

-o* write output module to file *. The default is STDOUT.

-r## override the address field with ##.

-s operate silently (without diagnostic output).

-x extract the input file into the name supplied in the Motorola S0 record.

unhex reads STDIN or <file>, converting hex records to binary form. Lines beginning with ':' are assumed to be Intel hex format. Lines beginning with **S0, S1, or S2** are assumed to be Motorola S-record format. Lines beginning with $ or S? (where '?' is any character other than 0, 1, or 2) are ignored. Any other input is considered to be an error.

Each record is checked for a consistent count field and a valid checksum. The count will be incorrect unless the formatted record ends in a single newline. Tabs, carriage returns, and other whitespace are not permitted. A bad count or bad checksum is considered an error.

An error will cause **unhex** to exit unless the **-e** option is set. A message is generated for every error unless the -s option is set. Thus, using **-e** alone simply prints a message on each error, and using -e and -s causes **unhex** to limit its attention to valid records only.

Each address converted causes a seek to the output file unless **-r##** is specified, or the current address does not equal the last address plus the last record size. No seek will be done if the first address is 0, and all addresses are consecutive, or if **-r0** is specified. A seek failure is an error.

Return Status

unhex returns success if it can open, read, and write all files successfully.

Examples

To convert a single file from binary to hexadecimal format and back again:

```
hex80 -o asciifile binaryfile
unhex -o binaryfile asciifile
```

To transfer a number of arbitrary files:

```
hex80 -sr0 file1 >asciifile
hex80 -sr0 file2 >>asciifile
```

and so on...

```
unhex -x -r0 asciifile
file1:
file2:
```