Part Three

80/RL R & L Tools
Reference Guide

Table of Contents

1. Introduction

This portion of the Experts-PL/M™ manual describes the use of the 80/RL™ Relocation and Linkage Tools. These are a collection of utilities that may be used to combine and manipulate object files to produce absolute object modules suitable for loading and running on an Intel 8080/8085 or a Zilog Z80 system. Although examples are given, the manual is not intended as a tutorial – it assumes at least some familiarity with linkers.

Several chapters describe the independent use of the R & L tools; another describes some of the concerted uses of the tools; still others contain examples and other miscellaneous information.

1.1 THE 80/RL TOOLS

The 80/RL tools are:

80/LINK™	Combines multiple object modules and libraries into a single relocatable object module.
80/LOC™	Converts a single relocatable object module into an absolute object module.
80/THEX™	Converts a single relocatable object module into an absolute object module in either Tektronix LAS format or extended Tekhex format.
80/MAP™	Produces an address map of one or more object modules.
80/STRIP™	Deletes public and debugging dictionary information from one or more object modules.
80/HEX™	Converts an object module from Intel MCS-80/85 Relocatable Object Module Format to absolute hexadecimal form.
80/DSOBJ™	Converts an object module from Intel MCS-80/85 Relocatable Object Module Format to a convenient, human-readable form.
80/CROBJ™	Converts the display form of an object module, as produced by 80/DSOBJ, to Intel MCS-80/85 Relocatable Object Module Format.
80/LIBCR™	Creates a library of object modules in a form to be searched by 80/LINK.
80/LIBLS™	Provides a listing of information about a library created by 80/LIBCR.

These tools operate under the VAX/VMS™, UNIX™, and PC-DOS operating systems.

1.2 INVOKING THE 80/RL TOOLS

Under the UNIX and PC-DOS operating systems, all tools are invoked in a uniform manner using UNIX conventions. Under VAX/VMS, some tools are invoked using VMS conventions and some are invoked using UNIX conventions.

1.2.1 Invocation Considerations for UNIX and PC-DOS

The method for invoking each of the 80/RL tools is described in the chapter devoted to that tool. However, certain invocation features under UNIX and PC-DOS are common to all of the tools and are described in this section.

1.2.1.1 Argument Files

Any command line argument may have the form

```
@argfile
```

where *argfile* is a file containing more arguments. This is particularly useful in cases where more arguments are required than will fit on the original command line.

1.2.1.2 Redirecting the Standard Error File

Error messages are written on the standard error file, which is usually the display screen. This may be changed by using a command line (or argument file) argument of the form

```
^errfile
```

where *errfile* is the name of the file to receive error messages. If the argument has the form

```
^^errfile
```

the messages will be appended to the file.

1.2.2 Invocation Considerations for VAX/VMS

The 80/LINK, 80/LOC, and 80/MAP tools are invoked using VMS-style conventions. The others use the UNIX style. When using the UNIX style under VMS, be sure to enclose argument strings in double-quote marks (") if they contain any uppercase characters that should remain as uppercase.

2. Using 80/LINK: UNIX and PC-DOS

The 80/LINK linking tool is invoked using UNIX conventions under UNIX and PC-DOS and using VMS conventions under VMS. This chapter describes the use of 80/LINK under UNIX and PC-DOS. See Chapter 3 for a discussion of its use under VMS.

Under UNIX and PC-DOS, 80/LINK is invoked as

```
80link [-v] [-o [output]] [-p] file [[-p] file] ...
```

which combines a list of 8080 object files and generates a single object file from them. Every module header record turns into a module ancestor record. Any library named as an input file is searched for modules which satisfy outstanding external references; such modules are incorporated into the output file.

If at least one main module is present in the modules combined, then the resulting module is also a main module – its starting address and name are the same as that of the first main module encountered. If a main module is not present, the resulting module is a subroutine module – it has no starting address, and its name is that of the first module encountered.

The following switches are recognized:

-p The next object file is scanned only for absolute public definitions. Such publics are used to resolve external references, but are not written to the output file.

-o The next argument is used as the output file name (rather than c.out, the default). The output file name may be so specified only once.

-v Each unresolved external is named (the default action is to report only that unresolved externals are present).

-Xt*aaa* Specifies, as *aaa*, the *prefix* to be used on all temporary file names, instead of the default "\tmp\" under PC-DOS or "/usr/tmp/" under UNIX. As an example, "-Xt./" will cause temporary files to be created in the current directory (i.e., the one in use when 80link is invoked).

2.1 SIMPLE LINKING

At its simplest, the linker produces a file containing a single linked object module from a number of simple input object files:

```
80link obj1.q obj2.q ... objn.q
```

leaves the linked output module of the n object files in c.out. Each of the input files must be either a legal object file, or an 80/DS library. The linked output can be left in any desired file by using the -o switch followed by the name of the desired output file (two separate arguments to the linker). The linked output cannot be sent to standard output.

2.2 OUTPUT MODULE CHARACTERISTICS

If none of the linker's input object files contains a main program module, then the resulting module is not a main program, and its module name is the same as that of the first module in the first input object file. If at least one of the input object modules is a main program module, then the resulting module is a main program, its starting address is the starting address of the first main program encountered, and its module name is the module name of the first main program encountered (if more than one of the input object modules is a main program, an error message is also generated).

2.3 USING LIBRARIES

When the linker encounters an 80/DS library, it does the following for each member of the library in turn:

- It compares the publics of the member against the outstanding externals.
- If a match is found, the current member is immediately linked in: its publics resolve any outstanding externals, and any of its externals which are not currently satisfied are *immediately* added to the outstanding external pool.

If, when the linker reaches the end of the library, at least one member has been linked in on the current pass through the library, it repeats the process from the first member of the library.

Library search is actually performed on a library dictionary, not on the individual library members, so that library search is relatively fast.

If no public within a library appears in more than one member, then the order of members within the library is immaterial, even if library members reference publics in other members of the same library. The extraction of members from a library containing multiply-defined publics is described in Section 15.4.

The linker cannot extract a specific member from a library. Therefore, a library member with no publics can never be used, and is, therefore, wasted.

The linker views each COMMON block as a public. Therefore, a library member which references a COMMON com will be extracted from the library if an external com currently exists, *and will resolve that external.* Only one library member can be extracted in this fashion for any given COMMON block name. The linker has no way of differentiating between library members which *reference* COMMON blocks (like subroutines) and those which *define* COMMON blocks (like BLOCK DATA subprograms).

2.4 RESOLVING AN EXTERNAL REFERENCE TO A FIXED ADDRESS

The linker's -p switch followed by a file name (two separate arguments to the linker) causes bound publics from the file to be used in resolving any outstanding external references (such a file is called a "publics-only" file). This can be used to resolve external references between two programs which cannot be physically linked together into a single module, such as a program run as a root and a separately-loaded overlay. Using a root and overlay as an example, it is then unnecessary either to know the physical addresses of things in the root when writing the code for the overlay, or to generate a set of object files, each of which has nothing but a single public declaration at an absolute address.

A library may not be used as a publics-only file.

If the linker is invoked by

```
80link oa.q ob.q -p root x.lib -o overlay
```

the following should be noted:

1. No data records from the publics-only file (*root*) are incorporated into the output (*overlay*).
2. No public symbol definitions from the publics-only file (*root*) appear in the output (*overlay*). If *unref* is a public in root referenced in both *ob.q* and *xb.q*,

   ```
   80link xa.q xb.q -p overlay x.lib
   ```

   will cause an "unresolved externals" warning message; *overlay*'s knowledge of *unref* is entirely private.
3. Only *bound* (absolute) publics from the publics-only file (*root*) are used; the -p switch causes relocatable publics in the file so specified to be ignored.
4. If a bound public in the publics-only file (*root*) and a relocatable public in a normal input file (*oa.q* or *ob.q*) have the same name, the public from the publics-only file is ignored; no "duplicate public" error is provoked.

A more complete example appears in Section 13.1.1.

2.5 COMMON BLOCK NAMES AND NORMAL PUBLICS

The linker assumes that a named COMMON block resolves references to an external of the same name. Thus, no public symbol may have the same name as a named COMMON block. Any program may reference COMMON blocks by declaring them to be external structures. The definition of a COMMON block as an external structure does not affect the size of the COMMON block.

2.6 SEGMENT SIZES

Each segment has a size strictly less than 64K (65536) associated with it. When the linker combines partial segments from two input files, the length is determined as follows:

segment	*combined segment size*
code, data	Modified summing: the size of the resulting segment is at least the sum of the sizes of the same segments in the input files (see Section 2.8 for an explanation).

stack — Summing: the size of the resulting segment is the sum of the sizes of the same segments in the input files.

memory — Overlay: the size of the resulting segment is the maximum of the sizes of the same segments in the input files.

unnamed COMMON, named COMMON — Overlay: the size of the resulting segment is the maximum of the sizes of the same segments in the input files.

If the sum of the sizes of the partial *code*, *data*, or *stack* segments is greater than (64K - 1), the linker issues an error message and sets the size of the resulting segment to (64K - 1).

The linker does not propagate a segment of zero length which has no inter- or intra-segment references to it.

2.7 SEGMENT ALIGNMENT

Each segment has an alignment which tells the binder the next suitable address for the segment: the next available byte (byte alignment); the next available byte whose address is a multiple of 256 (page alignment); or the next available byte which will ensure that the segment will not span a multiple of 256 (inpage alignment).

The linker determines the alignment of the combined segment produced from two partial segments. For the *code* and *data* segments, the combination of two partial segments is byte-aligned if both are byte-aligned; inpage-aligned if both are inpage-aligned and the sum of their lengths is no more than 256 bytes; and page-aligned otherwise. A gap may be generated if the second partial segment (the "new" segment being combined with an "existing" segment) is not byte-aligned (see Section 2.8). For all other relocatable segments, the combination of two partial segments is byte-aligned if both are byte-aligned, and page-aligned otherwise.

2.8 INVISIBLE GAPS

The linker can generate unreported (invisible) gaps in the code and data segments. This causes the length of the combined segment to be greater than the sum of sizes of the partial segments. Although the description is given for the data segment, it is equally valid for the code segment.

Let two data segments be data1 and data2, of length (a = 256*b + c) and (x = 256*y + z), respectively. The table indicates the intersegment gap produced for all possible alignments of data1 and data2:

data1, data2	*result*
byte, byte	No gap is produced.
page, byte	No gap is produced.
inpage, byte	No gap is produced.
byte, page	No gap is produced if *c* = 0; otherwise, the gap is (256 - *c*) bytes long.
page, page	No gap is produced if *c* = 0; otherwise, the gap is (256 - *c*) bytes long.
inpage, page	No gap is produced if *c* = 0; otherwise, the gap is (256 - *c*) bytes long.

byte, inpage	No gap is produced if $x \leq (256 - c)$; otherwise, the gap is (256 - c) bytes long.
page, inpage	No gap is produced if $x \leq (256 - c)$; otherwise, the gap is (256 - c) bytes long.
inpage, inpage	No gap is produced if $x \leq (256 - c)$ (the resulting segment is inpage-aligned); otherwise, the gap is (256 - c) bytes long (the resulting segment is page-aligned).

3. Using 80/LINK: VMS

The 80/LINK linking tool is invoked using UNIX conventions under UNIX and PC-DOS and using VMS conventions under VMS. This chapter describes the use of 80/LINK under VMS. See Chapter 2 for a discussion of its use under UNIX and PC-DOS.

Under VMS, 80/LINK is invoked by:

```
80LINK [options] file-spec[,file-spec...]
Qualifiers:                 Defaults:
/ARGS="options"
/[NO]OUTPUT[=file-spec]      /OUTPUT
/PUBLICS_ONLY
/[NO]VERBOSE                 /VERBOSE
```

By default, 80LINK will process the specified files (with default file type of "Q80") and place the resulting linked object module in a file with the same name as the first input file and a file type of "L80".

3.0.1 Invocation Options

/ARGS="options"

Allows the use of UNIX-style options. Note that the option list should be enclosed in double-quote marks (") to preserve the case of the options.

/OUTPUT[=file-spec]
/NOOUTPUT

Controls whether the linker produces an output object module. By default, the linker produces an object module that has the same file name as the first input file and a file type of "L80".

/PUBLICS_ONLY

Identifies files to be scanned only for absolute public definitions. These files are used only to resolve external references. They are not written to the output file. This qualifier is specified *only* on particular files, not on the command. Its effect is *only* for that file.

/VERBOSE
/NOVERBOSE

Indicates that the linker should identify all unresolved externals. The default is /VERBOSE.

3.1 SIMPLE LINKING

At its simplest, the linker produces a file containing a single linked object module from a number of simple input object files:

```
80link obj1,obj2,...,objn
```

links *obj1.q80*, *obj2.q80*,... and leaves the linked output module of the n object files in *obj1.l80*. Each of the input files must be either a legal object file, or an 80/DS library. The linked output can be left in any desired file by using the /OUTPUT qualifier as in

```
80link/output=mtest a1,a2
```

which links *a1.q80* and *a2.q80* and leaves the result in *mtest.l80*.

3.2 OUTPUT MODULE CHARACTERISTICS

If none of the linker's input object files contains a main program module, then the resulting module is not a main program, and its module name is the same as that of the first module in the first input object file. If at least one of the input object modules is a main program module, then the resulting module is a main program, its starting address is the starting address of the first main program encountered, and its module name is the module name of the first main program encountered (if more than one of the input object modules is a main program, an error message is also generated).

3.3 USING LIBRARIES

When the linker encounters an 80/DS library, it does the following for each member of the library in turn:

- It compares the publics of the member against the outstanding externals.
- If a match is found, the current member is immediately linked in: its publics resolve any outstanding externals, and any of its externals which are not currently satisfied are *immediately* added to the outstanding external pool.

If, when the linker reaches the end of the library, at least one member has been linked in on the current pass through the library, it repeats the process from the first member of the library.

Library search is actually performed on a library dictionary, not on the individual library members, so that library search is relatively fast.

If no public within a library appears in more than one member, then the order of members within the library is immaterial, even if library members reference publics in other members of the same library. The extraction of members from a library containing multiply-defined publics is described in Section 15.4.

The linker cannot extract a specific member from a library. Therefore, a library member with no publics can never be used, and is, therefore, wasted.

The linker views each COMMON block as a public. Therefore, a library member which references a COMMON *com* will be extracted from the library if an external

com currently exists, *and will resolve that external.* Only one library member can be extracted in this fashion for any given COMMON block name. The linker has no way of differentiating between library members which *reference* COMMON blocks (like subroutines) and those which *define* COMMON blocks (like BLOCK DATA subprograms).

3.4 RESOLVING AN EXTERNAL REFERENCE TO A FIXED ADDRESS

The linker's /PUBLICS_ONLY qualifier following an input file name causes bound publics from the file to be used in resolving any outstanding external references (such a file is called a "publics-only" file). This can be used to resolve external references between two programs which cannot be physically linked together into a single module, such as a program run as a root and a separately-loaded overlay. Using a root and overlay as an example, it is then unnecessary either to know the physical addresses of things in the root when writing the code for the overlay, or to generate a set of object files, each of which has nothing but a single public declaration at an absolute address.

A library may not be used as a publics-only file.

If the linker is invoked by

```
80link/output=overlay oa,ob,root/publics,x.lib
```

the following should be noted:

1. No data records from the publics-only file (*root*) are incorporated into the output (*overlay*).
2. No public symbol definitions from the publics-only file (*root*) appear in the output (*overlay*). If *unref* is a public in root referenced in both *ob.q80* and *xb.q80*,

   ```
   80link xa,xb,overlay.180/publics,x.lib
   ```

   will cause an "unresolved externals" warning message; *overlay*'s knowledge of *unref* is entirely private.
3. Only *bound* (absolute) publics from the publics-only file (*root*) are used; the /PUBLICS_ONLY qualifier causes relocatable publics in the file so specified to be ignored.
4. If a bound public in the publics-only file (*root*) and a relocatable public in a normal input file (*oa.q80* or *ob.q80*) have the same name, the public from the publics-only file is ignored; no "duplicate public" error is provoked.

A more complete example appears in Section 13.1.1.

3.5 COMMON BLOCK NAMES AND NORMAL PUBLICS

The linker assumes that a named COMMON block resolves references to an external of the same name. Thus, no public symbol may have the same name as a named COMMON block. Any program may reference COMMON blocks by declaring them to be external structures. The definition of a COMMON block as an external structure does not affect the size of the COMMON block.

3.6 SEGMENT SIZES

Each segment has a size strictly less than 64K (65536) associated with it. When the linker combines partial segments from two input files, the length is determined as follows:

segment	*combined segment size*
code, data	Modified summing: the size of the resulting segment is at least the sum of the sizes of the same segments in the input files (see Section 3.8 for an explanation).
stack	Summing: the size of the resulting segment is the sum of the sizes of the same segments in the input files.
memory	Overlay: the size of the resulting segment is the maximum of the sizes of the same segments in the input files.
unnamed COMMON, named COMMON	Overlay: the size of the resulting segment is the maximum of the sizes of the same segments in the input files.

If the sum of the sizes of the partial *code*, *data*, or *stack* segments is greater than (64K - 1], the linker issues an error message and sets the size of the resulting segment to (64K - 1).

The linker does not propagate a segment of zero length which has no inter- or intra-segment references to it.

3.7 SEGMENT ALIGNMENT

Each segment has an alignment which tells the binder the next suitable address for the segment: the next available byte (byte alignment); the next available byte whose address is a multiple of 256 (page alignment); or the next available byte which will ensure that the segment will not span a multiple of 256 (inpage alignment).

The linker determines the alignment of the combined segment produced from two partial segments. For the *code* and *data* segments, the combination of two partial segments is byte-aligned if both are byte-aligned; inpage-aligned if both are inpage-aligned and the sum of their lengths is no more than 256 bytes; and page-aligned otherwise. A gap may be generated if the second partial segment (the "new" segment being combined with an "existing" segment) is not byte-aligned (see Section 3.8). For all other relocatable segments, the combination of two partial segments is byte-aligned if both are byte-aligned, and page-aligned otherwise.

3.8 INVISIBLE GAPS

The linker can generate unreported (invisible) gaps in the code and data segments. This causes the length of the combined segment to be greater than the sum of sizes of the partial segments. Although the description is given for the data segment, it is equally valid for the code segment.

Let two data segments be data1 and data2, of length ($a = 256*b + c$) and ($x = 256*y + z$), respectively. The table indicates the intersegment gap produced for all possible alignments of data1 and data2:

data1, data2	result
byte, byte	No gap is produced.
page, byte	No gap is produced.
inpage, byte	No gap is produced.
byte, page	No gap is produced if c = 0; otherwise, the gap is (256 - c) bytes long.
page, page	No gap is produced if c = 0; otherwise, the gap is (256 - c) bytes long.
inpage, page	No gap is produced if c = 0; otherwise, the gap is (256 - c) bytes long.
byte, inpage	No gap is produced if x ≤ (256 - c); otherwise, the gap is (256 - c) bytes long.
page, inpage	No gap is produced if x ≤ (256 - c); otherwise, the gap is (256 - c) bytes long.
inpage, inpage	No gap is produced if x ≤ (256 - c) (the resulting segment is inpage-aligned); otherwise, the gap is (256 - c) bytes long (the resulting segment is page-aligned).

4. Using 80/LOC: UNIX and PC-DOS

The 80/LOC binder tool is invoked using UNIX conventions under UNIX and PC-DOS and using VMS conventions under VMS. This chapter describes the use of 80/LOC under UNIX and PC-DOS. See Chapter 5 for a discussion of its use under VMS.

Under UNIX and PC-DOS, 80/LOC is invoked as

```
80loc [-aaln] [-c] [-lloc] [-m map] [-o file]
      [-sseg] [file]
```

which assigns physical addresses to logical addresses in an 8080 object module. The files "c.out" and "d.out" in the current directory are used as the default input and output files, respectively. At most one input file may be processed.

The default segment order is

1. CODE
2. STACK
3. any named COMMON blocks, in an unspecified order
4. unnamed (blank) COMMON
5. DATA
6. MEMORY

The next available address is initially 0.

The following switches are recognized; all numbers are entered in 80/PL notation (⟨number⟩[BOQDH]).

-sseg The indicated segment is assigned the next available address (possibly adjusted for the segment's intrinsic alignment). All segments mentioned in -s switches are assigned addresses before segments not so mentioned are assigned addresses. Common block names must be enclosed in virgules (//), so that unnamed common has the segment name //, and a common block named XYZ has the segment name /XYZ/. The pseudo-segment names COMMONS and ALLOTHER correspond to all common blocks and all segments not otherwise mentioned, respectively, in the default order mentioned previously.

-sseg=lth, -sseg+lth, or -sseg-lth
: The indicated segment is assigned the next available address. Additionally, its length is modified. The first form causes the segment's length to be simply set to lth; the other forms cause the segment length given in the input file to be incremented (+) or decremented (-) by lth (but never before 0 or past 0FFFFH).

-aaln
: If aln is greater than 1, and the next available address is not divisible by aln, the next available address is advanced to the next multiple of aln.

-lloc
: The next available address will be loc.

-lseg, -lseg+loc, or -lseg-loc
: The next available address will be the address of the indicated segment (which must have already appeared in the argument list, either explicitly or implicitly), optionally incremented or decremented by loc. The pseudo-segment MAX may be used to retrieve the high-water mark of all currently-allocated memory.

-m
: A memory map is requested. If the next argument is -, then the map is sent to standard output; otherwise, the next argument is used as the name of the map file.

-o
: If this switch is the last argument, no bound output is produced. Otherwise, the next argument is used as the name of the output file (rather than d.out).

-c
: The data records are compacted – that is, they are sorted by address, with adjacent records potentially collapsed into one. This requires another pass through the input file. The default is random (uncompacted) data records.

-B*string*
: Prepend *string* to the name of each 80/LOC phase before executing it, thus allowing alternate versions of 80/LOC to be executed.

-Xt*aaa*
: Specifies, as *aaa*, the *prefix* to be used on all temporary file names, instead of the default "\tmp\" under PC-DOS or "/usr/tmp/" under UNIX. As an example, "-Xt./" will cause temporary files to be created in the current directory (i.e., the one in use when 80PC is invoked).

4.1 SIMPLE ADDRESS BINDING

At its simplest, the binder binds a single object module to physical (absolute) addresses. The command

```
80loc
```

binds the object module in *c.out* to physical addresses, and leaves the bound object module in *d.out*.

The input file may be set to any desired file by giving a file name:

```
80loc obj.lnk
```

binds the file *obj.lnk* rather than c.out. The bound output can be left in any desired file with the -o switch:

```
80loc -o run
```

binds c.out and leaves the result in run rather than d.out.

```
80loc -o
```

binds c.out but does not save the result anywhere. The bound output cannot be sent to standard output.

4.2 SWITCH MODIFIERS

The actual switches and the legal forms for their modifiers are described elsewhere; the following general comments apply to the binder's switch modifiers:

1. Switch modifiers are case-insensitive. However, symbolic modifiers will be given in lower case and literal modifiers will be given in upper case in the following sections.
2. Numbers in switch modifiers are entered in 80/PL notation, and are always collected mod 65536.
3. Whenever a segment is referenced in a switch modifier, the segment's name (not its numeric id) is required.
4. The pseudo-segments MAX, COMMONS and ALLOTHER may be used in place of normal segments in some situations. These are described with the appropriate switches.
5. The name of a COMMON block must be enclosed in virgules (//). Unnamed COMMON has the segment name //, the COMMON block named plotx has the segment name /plotx/, and the COMMON block named data has the segment name /data/; /data/ is not the data segment (*data* is).

4.3 SEGMENT PLACEMENT

By default, segments are assigned physical addresses beginning at 0, in nonoverlapping but adjacent memory (subject to segment alignment constraints – see Section 2.7).

The binder's -l switch modifies the next address available for a segment. Its uses are described in the table below; n is a number entered in 80/PL notation.

modifier	result
n	The next available address is set to n mod 65536.
segment	The next available address is set to the address of segment. If the segment is not the pseudo-segment MAX, it must have already appeared in the list of switches (forward references are disallowed); the address of MAX is considered to be higher than all currently-allocated addresses. MAX is useful when different segments have been physically overlaid, or when previous segments have been placed at absolute addresses in random order and an arbitrary (but nonconflicting) address is desired. MAX may appear any number of times; its value may be different each time that it appears, in a nondecreasing sequence.
segment+n	The next available address is set to the maximum of 65535 and (the address of segment + n). The segment must have already appeared in the list of switches, or else be MAX.

segment-n The next available address is set to the minimum of 0 and (the address of *segment* - *n*). The *segment* must have already appeared in the list of switches, or else be MAX.

4.4 SEGMENT ORDER

By default, the binder binds segments to physical addresses in the order

1. CODE
2. STACK
3. named COMMON blocks, in an unspecified order
4. unnamed (blank) COMMON
5. DATA
6. MEMORY

The binder's -s switch modifies this order; it has the form -ssegment. All segments mentioned in -s switches are assigned addresses before unmentioned segments are assigned addresses; unmentioned segments are assigned addresses in the default order. The next available address may be modified (by the -l switch) between two -s switches; the switches are processed in left-to-right order.

The pseudo-segment COMMONS causes all named COMMON blocks which are not mentioned in any other -s switch to be assigned addresses; unnamed (blank) COMMON is not affected. All -s switches – not just the preceding switches – are scanned for COMMON block names if -sCOMMONS appears. Named COMMON blocks selected in this way are assigned addresses in an unspecified order.

The pseudo-segment ALLOTHER causes all segments which are not mentioned in any other -s switch to be assigned addresses, in their default order. All -s switches – not just the preceding switches – are scanned for segment and pseudo-segment names if -sALLOTHER appears.

4.5 SEGMENT LENGTH MANAGEMENT

Every relocatable segment has a length associated with it in the module header. This length is managed by the linker when object modules are combined (see Section 2.6). The binder's -s switch can further modify the length of a segment, as described in the table below; *n* is a number entered in 80/PL notation.

modifier	**result**
segment=n	The length of *segment* is set to exactly *n*.
segment+n	The length of *segment* is set to the maximum of 65535 and (the length of *segment* in the input file + *n*).
segment-n	The length of *segment* is set to the minimum of 0 and (the length of *segment* in the input file - *n*).

If a segment appears in more than one -s switch, its *first* appearance defines its position, and its *last* explicitly-given length (if any) defines its length.

4.6 ALIGNMENT OF THE NEXT AVAILABLE LOCATION

Every relocatable segment has an alignment associated with it (see Section 2.7). The -a switch causes the binder to perform some alignment of the next available address; it takes a numeric modifier *n*.

If the modifier of the -a switch is 0 or 1, the next available address is left unchanged. Otherwise, if the next available address is not already a multiple of *n*, it is advanced to the next larger multiple of *n*. This alignment does not override the alignment of the next segment as given in the module header: it is impossible to assign a page-aligned segment an address which is an arbitrary multiple of 7, for instance (Section 15.5 has examples of the -a switch).

4.7 SEGMENT MAPS

The binder can produce a map of every segment and every previously-bound data record which it encounters, the execution start address of the module, any embedded address gaps, and any segment or previously-bound data record overlaps. The -m switch requests that such a map be produced; the next argument indicates the file to which the map is to be sent. The map is sent to standard output if - is used as the map file name.

A sample map might look like

```
00040H 00245H CODE
00285H 0007BH *GAP
00300H 00080H //
00300H 0048CH DATA
00300H 00080H *OVERLAP
0078CH 00038H STACK
007C4H 0F03CH *GAP
0F800H 00800H ABSOLUTE
0004FH        START ADDRESS
```

The first column gives the beginning address of the item, in 80/PL hexadecimal notation. The second column gives the length of the item, in 80/PL hexadecimal notation (the start address has no length). The third column gives the name of the item:

ABSOLUTE	a previously-bound data record
*GAP	a gap between the preceding and following items; no data records appear for these addresses
*OVERLAP	the previous two items overlapped in the indicated region
START ADDRESS	the execution start address of the module; this line is only present if the input object module is a main program
segment	a program segment or COMMON block

Except for the execution start address (which is always the last line, if present), the lines are in order of the items' beginning addresses.

4.8 DATA RECORD CONCATENATION

By default, the binder does not coalesce data records in the input object module: there are exactly as many data records in the bound output as there are in the (unbound) input. The -c switch causes two data records which do not refer to externals and which refer to adjacent addresses to be coalesced into a single data record. This causes a (slight) decrease in the size of the bound output, and may (favorably) affect the time it takes for other programs to process the bound output.

5. Using 80/LOC: VMS

The 80/LOC linking tool is invoked using UNIX conventions under UNIX and PC-DOS and using VMS conventions under VMS. This chapter describes the use of 80/LOC under VMS. See Chapter 4 for a discussion of its use under UNIX and PC-DOS.

Under VMS, 80/LOC is invoked by:

```
80LOC [options] file-spec
Qualifiers:                 Defaults:
/ARGS="options"
/[NO]COMPACT                 /NOCOMPACT
/[NO]MAP[=file-spec]         /NOMAP
/MEMORY=(options)
/[NO]OUTPUT[=file-spec]      /OUTPUT
```

By default, 80LOC will process the specified file (with default file type of "L80") and place the resulting absolute object module in a file with the same name and a file type of "B80".

5.0.1 Invocation Options

/ARGS="options"

Allows the use of UNIX-style options. Note that the option list should be enclosed in double-quote marks (") to preserve the case of the options.

/COMPACT
/NOCOMPACT

Controls whether 80/LOC compacts the output data records. With /COMPACT specified, the data records are sorted by address, collapsing adjacent data records into one. This requires an extra pass through the input file. The default is /NOCOMPACT.

/MAP[=file-spec]
/NOMAP

Controls whether a memory map is produced. If requested, the name defaults to that of the input file with a file type of "BMP". The default is /NOMAP.

/MEMORY=(options)

Controls the memory layout of the absolute output module. The next available address is initially 0. The default segment order is: CODE, STACK, named COMMON, unnamed COMMON, DATA, and MEMORY.

The following options may be used to override the defaults. All numbers are entered in 80/PL notation (⟨number⟩[BOQDH]). A plus sign (+) may be used where "#" is indicated, but it must be quoted because of DCL requirements. The keywords may be abbreviated as long as they are distinguishable.

SEGMENT:name or name=n or name#n or name-n

The indicated segment is asigned the next available address, adjusted for the segment's intrinsic alignment. All segments so mentioned are assigned addresses before all other segments. Common block names should be formed as: |name| and ||. The pseudo segment names COMMONS and ALLOTHERS are also available.

Optionally, the segment's length may be modified. It may be set to n (=), incremented by n (+ or #), or decremented by n (-).

ALIGNMENT:n

The next available address is (if it is not already) aligned to a multiple of n.

LOCATION:n

The next available address is set to n.

LOCATION:name or name#n or name-n

The next available address will be that of the named segment (which must have already been mentioned), optionally incremented (+ or #) or decremented (-) by n. The pseudo-segment MAX is the high water mark of currently allocated memory.

/OUTPUT[=file-spec]
/NOOUTPUT

Controls whether 80/LOC produces an output absolute module. By default, 80/LOC produces an absolute module that has the same file name as the input file and a file type of "B80".

5.1 SIMPLE ADDRESS BINDING

At its simplest, the binder binds a single object module to physical (absolute) addresses. The command

```
80loc test
```

binds the object module in *test.l80* to physical addresses and leaves the bound result in *test.b80*.

The bound output can be left in any desired file with the /OUTPUT qualifier

```
80loc/output=run test
```

binds *test.l80* and leaves the result in *run.b80* rather than *test.b80*.

5.2 QUALIFIER MODIFIERS

The actual qualifiers and the legal forms for their modifiers are described elsewhere; the following general comments apply to the binder's switch modifiers:

1. Qualifier modifiers are case-insensitive. However, symbolic modifiers will be given in lower case and literal modifiers will be given in upper case in the following sections.
2. Numbers are entered in 80/PL notation, and are always collected mod 65536.
3. Whenever a segment is referenced in a switch modifier, the segment's name (not its numeric id) is required.
4. The pseudo-segments MAX, COMMONS and ALLOTHER may be used in place of normal segments in some situations. These are described with the appropriate switches.
5. The name of a COMMON block must be enclosed in vertical bars (||). Unnamed COMMON has the segment name ||, the COMMON block named *plotx* has the segment name |*plotx*|, and the COMMON block named *data* has the segment name |*data*|; |*data*| is not the data segment (*data* is).

5.3 SEGMENT PLACEMENT

By default, segments are assigned physical addresses beginning at 0, in nonoverlapping but adjacent memory (subject to segment alignment constraints – see Section 3.7).

The binder's LOCATION keyword to the /MEMORY qualifier modifies the next address available for a segment. Its uses are described in the table below; *n* is a number entered in 80/PL notation.

modifier	result
n	The next available address is set to *n* mod 65536.
segment	The next available address is set to the address of *segment*. If the *segment* is not the pseudo-segment MAX, it must have already appeared in the list of switches (forward references are disallowed); the address of MAX is considered to be higher than all currently-allocated addresses. MAX is useful when different segments have been physically overlaid, or when previous segments have been placed at absolute addresses in random order and an arbitrary (but nonconflicting) address is desired. MAX may appear any number of times; its value may be different each time that it appears, in a nondecreasing sequence.
segment#n	The next available address is set to the maximum of 65535 and (the address of *segment* + *n*). The *segment* must have already appeared in the list of switches, or else be MAX.
segment-n	The next available address is set to the minimum of 0 and (the address of *segment* - *n*). The *segment* must have already appeared in the list of switches, or else be MAX.

5.4 SEGMENT ORDER

By default, the binder binds segments to physical addresses in the order

1. CODE
2. STACK
3. named COMMON blocks, in an unspecified order
4. unnamed (blank) COMMON
5. DATA
6. MEMORY

The binder's SEGMENT keyword to the /MEMORY qualifier modifies this order; it has the form SEGMENT:name. All segments specified by in SEGMENT keywords are assigned addresses before unmentioned segments are assigned addresses; unmentioned segments are assigned addresses in the default order. The next available address may be modified (by the LOCATION keyword) between two SEGMENT keywords; the keywords are processed in left-to-right order.

The pseudo-segment COMMONS causes all named COMMON blocks which are not specified in any other SEGMENT option to be assigned addresses; unnamed (blank) COMMON is not affected. All SEGMENT keywords – not just the preceding switches – are scanned for COMMON block names if SEGMENT:COMMONS appears. Named COMMON blocks selected in this way are assigned addresses in an unspecified order.

The pseudo-segment ALLOTHER causes all segments which are not specified in any other SEGMENT keyword to be assigned addresses, in their default order. All SEGMENT keywords – not just the preceding ones – are scanned for segment and pseudo-segment names if SEGMENT:ALLOTHER appears.

5.5 SEGMENT LENGTH MANAGEMENT

Every relocatable segment has a length associated with it in the module header. This length is managed by the linker when object modules are combined (see Section 3.6). The binder's SEGMENT keyword to the /MEMORY qualifier can further modify the length of a segment, as described in the table below; *n* is a number entered in 80/PL notation.

modifier	result
segment=n	The length of *segment* is set to exactly *n*.
segment#n	The length of *segment* is set to the maximum of 65535 and (the length of *segment* in the input file + *n*).
segment-n	The length of *segment* is set to the minimum of 0 and (the length of *segment* in the input file - *n*).

If a segment appears in more than one SEGMENT keyword, its *first* appearance defines its position, and its *last* explicitly-given length (if any) defines its length.

5.6 ALIGNMENT OF THE NEXT AVAILABLE LOCATION

Every relocatable segment has an alignment associated with it (see Section 3.7). The ALIGNMENT keyword causes the binder to perform some alignment of the next available address; it takes a numeric modifier n.

If the modifier of the keyword is 0 or 1, the next available address is left unchanged. Otherwise, if the next available address is not already a multiple of n, it is advanced to the next larger multiple of n. This alignment does not override the alignment of the next segment as given in the module header: it is impossible to assign a page-aligned segment an address which is an arbitrary multiple of 7, for instance (Section 16.5 has examples of the ALIGNMENT keyword).

5.7 SEGMENT MAPS

The binder can produce a map of every segment and every previously-bound data record which it encounters, the execution start address of the module, any embedded address gaps, and any segment or previously-bound data record overlaps. The /MAP qualifier requests that such a map be produced and optionally provides a file into which the map is to be written. If a file name is not given, the map wil be written to the a file with the same name as the input file with a file type of "BMP".

A sample map might look like

```
00040H 00245H CODE
00285H 0007BH *GAP
00300H 00080H //
00300H 0048CH DATA
00300H 00080H *OVERLAP
0078CH 00038H STACK
007C4H 0F03CH *GAP
0F800H 00800H ABSOLUTE
0004FH        START ADDRESS
```

The first column gives the beginning address of the item, in 80/PL hexadecimal notation. The second column gives the length of the item, in 80/PL hexadecimal notation (the start address has no length). The third column gives the name of the item:

ABSOLUTE	a previously-bound data record
*GAP	a gap between the preceding and following items; no data records appear for these addresses
*OVERLAP	the previous two items overlapped in the indicated region
START ADDRESS	the execution start address of the module; this line is only present if the input object module is a main program
segment	a program segment or COMMON block

Except for the execution start address (which is always the last line, if present), the lines are in order of the items' beginning addresses.

5.8 DATA RECORD CONCATENATION

By default, the binder does not coalesce data records in the input object module: there are exactly as many data records in the bound output as there are in the (unbound) input. The /COMPACT qualifier causes two data records which do not refer to externals and which refer to adjacent addresses to be coalesced into a single data record. This causes a (slight) decrease in the size of the bound output, and may (favorably) affect the time it takes for other programs to process the bound output.

6. Using 80/THEX

Some versions of 80/DS support a utility known as *80/THEX*. It is invoked by

```
80thex [-aaln] [-lloc] [-m map] [-o file] [-sseg]
       [-tterm] [-wwidth] [-fformat] [-xchip] [file]
```

under UNIX and PC-DOS. Under VMS, the invocation is the same except that the command name is "THEX80". 80/THEX assigns physical addresses to logical addresses in an object module and produces an absolute object module in either Tektronix LAS format or extended Tekhex format. The files c.out and e.out in the current directory are used as the default input and output files, respectively. At most one input file will be processed.

The default segment order is

1. CODE
2. STACK
3. any named COMMON blocks, in an unspecified order
4. unnamed (blank) COMMON
5. DATA
6. MEMORY

The next available address is initially 0.

Names longer than 16 characters are converted to a 16-character form by retaining the first 6 and the last 10 characters of the name (true for either output format).

The following apply if the output object is extended Tekhex. The segment A$*modulename* is created to hold symbols which are already absolute (which belong to no segment); this segment always has zero length and begins at address 0.

The following apply if the output object is LAS format. The segment A$*modulename* will contain all symbols and text found before the first ancestor record. The segment A.*oldmodule* will contain all symbols and text found after the ancestor record for the module *oldmodule*. These segments always have length 64K and begin at address 0. Note that the prefixes are different for the module and the ancestor cases.

The following switches are recognized; all numbers are entered in 80/PL notation (⟨number⟩[BOQDH]).

-sseg The indicated segment is assigned the next available address (possibly adjusted for the segment's intrinsic alignment). All segments mentioned in -s switches are assigned addresses before segments not so mentioned are assigned addresses. Common block names must be enclosed in virgules (//), so that unnamed common has the segment name //, and a common block named XYZ has the segment name /XYZ/. The pseudo-segment names COMMONS and ALLOTHER correspond to all common blocks and all segments not otherwise mentioned, respectively, in the default order mentioned previously.

-sseg=lth, -sseg+lth, or -sseg-lth
The indicated segment is assigned the next available address. Additionally, its length is modified. The first form causes the segment's length to be simply set to *lth*; the other forms cause the segment length given in the input file to be incremented (+) or decremented (-) by *lth* (but never before 0 or past 0FFFFH).

-aaln If *aln* is greater than 1, and the next available address is not divisible by *aln*, the next available address is advanced to the next multiple of *aln*.

-lloc The next available address will be *loc*.

-lseg, -lseg+loc, or -lseg-loc
The next available address will be the address of the indicated segment (which must have already appeared in the argument list, either explicitly or implicitly), optionally incremented or decremented by *loc*. The pseudo-segment MAX may be used to retrieve the high-water mark of all currently-allocated memory.

-m A memory map is requested. If the next argument is -, then the map is sent to standard output; otherwise, the next argument is used as the name of the map file.

-o If this switch is the last argument, no bound output is produced. Otherwise, the next argument is used as the name of the output file (rather than e.out).

-tterminator Change the Tekhex end-of-line sequence from \n (the default) to *terminator*; *terminator* may not be null. Standard C language escape sequences are supported. For example '\r' means 'carriage return', '\n' means 'new line' and '\013' means 'the character with octal value 13'.

-wwidth Change the maximum number of characters in extended Tekhex records from 256 (the default) to *width*. Extended Tekhex records are only used for symbol table information. The width may not be set smaller than 60 nor larger than 256.

-fformat Change the output file format to LAS format (if *format* is L), or extended Tekhex (if *format* is T). LAS format is the default.

-xchip Change the target processor information to 8080/8085 (if *chip* is 8080 or 8085), or Z80 (if *chip* is Z80 or z80). 8080/8085 is the default. This switch has an observable effect only when LAS format output has been selected.

6.1 CONSIDERATIONS

80/THEX functions similarly to 80/LOC. Each reads an input object module in the Intel MCS-80/85 Relocatable Object Module format and assigns physical addresses to the logical addresses of its input. Controls common to the two processors function identically. Note that the 80/LOC "compact" control is absent from 80/THEX (but see Section 15.8), and the 80/THEX "terminator", "width", "format" and "chip" controls are absent from 80/LOC.

The primary difference between the two processors is the output object module format. 80/LOC produces an output object module in the Intel MCS-80/85 Relocatable Object Module format, which can be further processed by 80/DS tools. 80/THEX produces an output object module in either the Tektronix LAS format or the Tektronix extended Tekhex format, neither of which can be further processed by 80/DS tools.

7. Using 80/MAP: UNIX and PC-DOS

The 80/MAP map generator is invoked using UNIX conventions under UNIX and PC-DOS and using VMS conventions under VMS. This chapter describes the use of 80/MAP under UNIX and PC-DOS. See Chapter 8 for a discussion of its use under VMS.

The map generator produces a single symbolic map (symbol table) for a collection of arbitrary object files. It can produce such a map in a format suitable for machine searching ("grep" format), or in a format more suited to casual perusal ("simple" format). The map may reflect the order of the symbolic information in the object file, or may be sorted by name or address.

80/MAP is invoked by

```
80map [-aatoms] [-ftype] [-s[sort]]
      [-ttab] [name] ...
```

If no object files are named, the standard input is mapped. The map is written to standard output.

The default map has the format

```
address   file   module   type   item
```

An optional, simpler format has lines with the format

```
address   type   item
```

By default, the entries in each line are separated by tab characters.

The most significant byte of any address is the segment of that address. Thus, absolute addresses look like 00XXXX; relocatable addresses in the CODE segment look like 01XXXX; and similarly for the other segments. All addresses are given in hexadecimal.

By default, all external, local symbol, public symbol and line number information goes into the map.

The following switches are recognized:

-ftype — Specify a map format. If *type* is "g", generate the default ("grep") format map; if *type* is "s", generate the simpler format map.

-stype — Sort the map. The map is sorted by name, sorted by address, or unsorted ("sorted" by order of appearance), as *type* is "n", "a", or "u". The default is

an unsorted map. The sorting method may not be changed from sorted to unsorted (or conversely) after an input file has been encountered. If more than one sorting method is given, the last one encountered takes effect ("-sa file1 -sn file2" produces a map sorted by name). When sorting by name, all items with no address (ancestors, externals, files and modules) collate low to all symbols (public and local), which collate low to all line numbers.

-aatoms Specify the atoms (types of items) in the map. The possible atoms are

a	module ancestor records (simple format only)
e	external symbols
f	file names (simple format only)
l	line numbers
m	module name records (simple format only)
p	public symbols
s	local symbols
w	all of the above (the world)
-	the following atoms are deleted from the map
+	the following atoms are added to the map

The default is +w- (everything goes into the map; subsequent atom types are deleted from the map).

-ttab Change the tab character to "tab".

7.1 SIMPLE MAP GENERATION

The simplest map generation produces the symbolic map of a single object file:

```
80map obj.q
```

sends the default-format symbolic map of obj.q to standard output. More than one object file name may be given, resulting in a single "super-map". If no object file name is given, the symbolic map of standard input is generated.

7.2 MAP FORMATS

By default, the map generator produces a machine-searchable map; this format can also be specifically requested by the -f switch with the modifier g. Lines in such a map have the format

```
address   file   module   type   item
```

A simple map is produced by the -f switch with the modifier s. Lines in such a map have the format

```
address   type   item
```

The same information is present for either format, although a simple format map will have more lines than its corresponding default format map. The map format may not be changed after an object file has been processed.

The entries in each line in either format are separated by tab characters. The tab character is initially the ASCII tab; it can be changed to the first character of the modifier of a -t switch.

The *address* portion of a line gives the address of the *item* as a six-digit hexadecimal number. The first two digits of an *address* give a numeric segment id; the remaining four digits give an offset within that segment. Thus, an absolute *item* has an address in the range 000000 to 00ffff; a code segment relocatable *item*, in the range 010000 to 01ffff; and similarly for the other segments. External names, file names, module names and ancestor names never have *addresses* associated with them.

The *file* and *module* are the names of the file and module in which the *item* reside.

The *type*'s and the map format in which they may appear are

type	**description**
extern	external symbol name (g, s)
line	line number (g, s)
public	public symbol name (g, s)
symbol	local symbol name (g, s)
ancest	the ancestor module for the subsequent items (s)
file	the file containing the subsequent items (s)
module	the module containing the subsequent items (s)

The *item*'s are either names or line numbers.

7.3 SORTED MAP GENERATION

The -s switch causes the symbolic map to be sorted as described in the table:

modifier	**sorting method**
a	sort the map by address; *type*'s without an address collate low to *type*'s with an address; line numbers collate high to symbols
n	sort the map by name (*item*)
u	produce an unsorted map (order of appearance in the input file)

The map may not be changed from sorted to unsorted, or unsorted to sorted, after an object file has been processed; otherwise, the last sorting method encountered takes effect for the entire map.

7.4 MODIFYING THE CONTENTS OF A MAP

The various types of items (atoms) which are actually selected for the map can be specified by the modifiers to the -a switch, as given in the table:

modifier	atom selected
a	module ancestor names (ignored in default-format maps)
e	external names
f	file names (ignored in default-format maps)
l	line numbers
m	module names (ignored in default-format maps)
p	public symbol names
s	local symbol names
w	all of the set a, e, f, l, m, p, s (the world)
-	subsequent atoms in -a switch modifiers are deleted from the map
+	subsequent atoms in -a switch modifiers are added to the map

The default setting is *+w-*: everything is sent to the map; alphabetic modifiers to any later -a switch are deleted from the map. Each -a switch affects object files until a new -a switch is encountered. Note that the -a switch is cumulative.

8. Using 80/MAP: VMS

The 80/MAP map generator is invoked using UNIX conventions under UNIX and PC-DOS and using VMS conventions under VMS. This chapter describes the use of 80/MAP under VMS. See Chapter 7 for a discussion of its use under UNIX and PC-DOS.

The map generator produces a single symbolic map (symbol table) for a collection of arbitrary object files. It can produce such a map in a format suitable for machine searching or in a format more suited to casual perusal. The map may reflect the order of the symbolic information in the object file, or may be sorted by name or address.

80/MAP is invoked by

```
80MAP [options] file-spec[,file-spec...]
Command Qualifiers:     Defaults:
/ARGS="options"
/[NO]BRIEF                /NOBRIEF
/ITEMS=(items)
/MAP=file-spec
/SEPARATOR=character
/[NO]SORT                 /NOSORT
```

One or more input files are processed to produce one map output for each. The default file type for input files is "B80". The map is placed in a file with the name of the first input file and a file type of "MAP".

The default map has the format

```
address   file   module   type   item
```

An optional, simpler format has lines with the format

```
address   type   item
```

By default, the entries in each line are separated by tab characters.

The most significant byte of any address is the segment of that address. Thus, absolute addresses look like 00XXXX; relocatable addresses in the CODE segment look like 01XXXX; and similarly for the other segments. All addresses are given in hexadecimal.

By default, all external, local symbol, public symbol and line number information goes into the map.

The processing of the map can be controlled by the following qualifiers:

/ARGS="options"

Allows UNIX-style arguments (see Chapter 7) to be used. The option list should be enclosed in double-quote marks (") in order to preserve the case of the options.

/BRIEF
/NOBRIEF

Controls the format of the map. By default, /NOBRIEF is in effect and the full map is produced. The /BRIEF qualifier produces the simpler format.

ITEMS=(items)

Controls what items will appear in the map. By default, all items appear. Any items specified are cummulative, so specifying /ITEMS=PUBLICS will have no effect (it just adds PUBLICS to the default list which already contains PUBLICS). If only the publics are desired, specify /ITEMS=(NONE,PUBLICS). To get everything but publics, specify /ITEMS=NOPUBLICS. The possible items are:

ANCESTORS	module ancestor records (effective for /BRIEF only)
EXTERNALS	external symbols
FILE_NAMES	file names (effective for /BRIEF only)
LINE_NUMBERS	line numbers
MODULE_NAMES	module name records (effective for /BRIEF only)
PUBLICS	public symbols
SYMBOLS	local symbols
ALL	all of the above
NONE	none of the above

This qualifier may be used both on the command and on individual files. On individual files, it overrides anything specified on the command and starts over again from the default of ALL.

/MAP=file-spec

May be used to change the name of the map file. It may not be negated nor supplied without a *file-spec*. The name portion of the *file-spec* defaults to the input file name. The file type defaults to "MAP".

/SEPARATOR=character

May be used to change the character used to separate the fields of the map. The default is "horizontal tab". If a character is used which appears in any of the fields themselves, a sort will produce unpredictable results.

/SORT=type
/NOSORT

Controls how the map file is sorted. By default, the map is unsorted (sorted by order of appearance). This may be changed to a sort by either NAME or ADDRESS. When sorting by name, all items with no address (ancestors, externals, files, and modules) collate low to all symbols (public and local), which collate low to all line numbers. Beware of changing the separator character when sorting; the results are unpredictable if it appears anywhere other than as the separator.

8.1 SIMPLE MAP GENERATION

The simplest map generation produces the symbolic map of a single object file:

```
80map obj
```

places the default-format symbolic map of *obj.b80* to *obj.map*. More than one object file name may be given, resulting in a single *super-map*.

8.2 MAP FORMATS

By default, the map generator produces a machine-searchable map; this format can also be specifically requested by the /NOBRIEF qualifier. Lines in such a map have the format

```
address   file   module   type   item
```

A simple map is produced by the /BRIEF qualifier. Lines in such a map have the format

```
address   type   item
```

The same information is present for either format, although a simple format map will have more lines than its corresponding default format map. The map format may not be changed after an object file has been processed.

The entries in each line in either format are separated by tab characters. The tab character is initially the ASCII tab; it can be changed by the /SEPARATOR qualifier.

The *address* portion of a line gives the address of the *item* as a six-digit hexadecimal number. The first two digits of an *address* give a numeric segment id; the remaining four digits give an offset within that segment. Thus, an absolute *item* has an address in the range 000000 to 00ffff; a code segment relocatable *item*, in the range 010000 to 01ffff; and similarly for the other segments. External names, file names, module names and ancestor names never have *addresses* associated with them.

The *file* and *module* are the names of the file and module in which the *item* reside.

The *type*'s and the map format in which they may appear are

type	description
extern	external symbol name (both)
line	line number (both)
public	public symbol name (both)
symbol	local symbol name (both)
ancest	the ancestor module for the subsequent items (brief)

file	the file containing the subsequent items (brief)
module	the module containing the subsequent items (brief)

The *item*'s are either names or line numbers.

8.3 SORTED MAP GENERATION

The /SORT qualifier causes the symbolic map to be sorted as described in the table:

modifier	**sorting method**
ADDRESS	sort the map by address; *type*'s without an address collate low to *type*'s with an address; line numbers collate high to symbols
NAME	sort the map by name (*item*)

The map may not be changed from sorted to unsorted, or unsorted to sorted, after an object file has been processed; otherwise, the last sorting method encountered takes effect for the entire map.

8.4 MODIFYING THE CONTENTS OF A MAP

The various types of items which are actually selected for the map can be specified by the modifiers to the /ITEMS qualifier, as given in the table:

modifier	**atom selected**
ANCESTORS	module ancestor names (ignored in /NOBRIEF format maps)
EXTERNALS	external names
FILE_NAMES	file names (ignored in /NOBRIEF format maps)
LINE_NUMBERS	line numbers
MODULE_NAMES	module names (ignored in /NOBRIEF format maps)
PUBLICS	public symbol names
SYMBOLS	local symbol names
ALL	all of the above
NONE	none of the above

The default setting is *ALL*: everything is sent to the map;

9. Using 80/STRIP

The stripper removes public symbols, local symbols, line numbers, and ancestor module names from arbitrary object files. This reduces the size of the object file. It is commonly performed only on fully linked and bound object files. 80/STRIP is invoked by

```
80strip [-lLpPsS] name [[-lLpPsS] name ...]
```

under UNIX and PC-DOS. It is invoked the same way under VMS except that the command name is "STRIP80". The switches are defined in Section 9.2 and examples are given in Section 15.10 and Section 15.11.

9.1 SIMPLE OBJECT FILE STRIPPING

The simplest use of the stripper removes all possible information – public symbols, local symbols, line numbers, and ancestor module names – from a single object file:

```
80strip obj1.q
```

replaces *obj1.q* by its stripped form (the original file is lost except under VMS). Multiple object files may be stripped at once:

```
80strip obj1.q obj2.q obj3.q
```

replaces each of the named object files by its stripped form.

9.2 RESTRICTED OBJECT FILE STRIPPING

Any of the strippable information classes (except ancestor module names) may be retained independently, under the control of six switches:

switch	action
-l	line numbers and ancestor module names are retained
-p	public symbols are retained
-s	local symbols and ancestor module names are retained
-L	line numbers are removed; ancestor module names are removed if local symbols are also removed
-P	public symbols are removed

-S local symbols are removed; ancestor module names are removed if line numbers are also removed

The switches are cumulative. The default setting is -LPS.

Note under VMS, arguments must be enclosed in double-quote marks if they contain any upper-case characters.

10. Using 80/HEX

The absolute hex object file producer produces an absolute hexadecimal form of an object file from the object file itself. It is invoked by

```
80hex [name]
```

under UNIX and PC-DOS. It is invoked the same way under VMS except that the command name is "HEX80".

10.1 ABSOLUTE HEX OBJECT FILE PRODUCTION

The invocation

```
80hex
```

reads an object file from standard input, and

```
80hex obj1.q
```

reads an object file from *obj1*.q. The absolute hex form is always sent to standard output. Exactly one object file is read and processed. There are no switches.

10.2 RESTRICTIONS

The input to 80/HEX must be an *absolute* (bound) object file. If the file contains any relocatable data records, external references, inter- or intra-segment references, or a relocatable start address, a fatal error is provoked.

The 80/RL Relocation and Linkage Tools do not further process absolute hexadecimal object files.

10.3 OUTPUT FILE FORMAT

Output records are entirely in ASCII. There are only two kinds of output record: data and end. Each record begins with a colon, is terminated with a newline, and contains no embedded blanks.

A data record looks like

```
:10500000000102030405060708090A0B0C0D0E0F28\n
```

The first two digits give the number of data bytes, n, in hex – here, 10h or 16. The next four digits give the address to which the first data byte is bound – here, 5000h. The next

two digits are always *00*. The next *2n* digits give the data bytes themselves, in hex – here, the byte integers from 0 to 15 (00h to 0Fh). The last two digits are a checksum.

An end record looks like

```
:0052370176\n
```

The first two digits are always *00* (there are no data bytes in an end record). The next four digits give the starting address of the object module, in hex – in this case, 5237h. The next two digits are always *01*. The last two digits are a checksum.

10.4 USE WITH VMS

Since the output of 80/HEX is placed on SYS$OUTPUT, it may be convenient to define a command procedure such as

```
$delete 'p1'.hex;*
$define/user sys$output 'p1'.hex
$hex80 'p1'.q80
```

which will take it's input from a file with a file type of "q80" and place the output in a file with the same name but with a file type of "hex".

11. Using 80/CROBJ and 80/DSOBJ

The 80/CROBJ object file creator produces an object file from a human-readable form of the object file (creation source). The 80/DSOBJ object file displayer inverts the process, producing creation source from an object file.

11.1 CREATION SOURCE FORMAT

Creation source is essentially a byte-by-byte ASCII hexadecimal form of the desired object file. The only items in the resulting object file which are not given in ASCII hex are

- default (processor-calculated) record lengths
- default (processor-calculated) checksums
- object file information which has the form of a *name*: a one-byte count followed by that many (printable) characters

Other symbolic information may be included in creation source, but such information is considered to be a comment, and is not processed: there are no symbolic forms for the different record types, segment names, or anything else. Comment information cannot contain (,), *, ', or *0* (digit zero).

The object file creator will always correctly process the output of the object file displayer, which may be used as a pattern for user-generated creation source.

11.1.1 Creation Source Elements

The basic elements of creation source are "whitespace", "numbers", and "names".

11.1.1.1 Whitespace

Whitespace is an arbitrary-length (nonzero) sequence of blanks, tabs and newlines.

11.1.1.2 Numbers

A number is a string of 2n+1 hex digits, n > 0. The first digit must be zero. Every succeeding pair of digits defines a byte datum. Bytes are given in 8080 memory order: the number 001F0 represents the two sequential bytes (01, F0h) or the 8080 word (F001h), not the 8080 word (01F0h). A number must be followed by either whitespace, (, or).

11.1.1.3 Names

A name is an item of the form

```
name-length name-characters
```

The name-length is a byte, so there may be from 0-255 name-characters in a name (the total length of the construct is from 1-256 bytes). In creation source, a name may be written as

```
'name-characters'
```

where the name-characters are taken from the set of the twenty-six letters, the ten decimal digits, and the special characters @ and ?.

11.1.2 Creation Source Records

Creation source takes the form of a sequence of records. A record has the form

```
( record-type record-length record-body ) checksum
```

The first character of a record is an open parenthesis. The record type is given by a number. The record length (the length of the record body and checksum) may be an asterisk or a number. A number gives the actual record length; an asterisk is a default indicator – the correct length is not important, and is no more than 1025. The record body may contain an arbitrary collection of names and numbers, and is followed by a close paren. The checksum may be an asterisk or a number. A number gives the actual checksum; an asterisk is a default indicator – the actual value is not important.

11.2 OBJECT FILE CREATION

80/CROBJ is invoked as

```
80crobj [-cC] [file]
```

under UNIX and PC-DOS. It is invoked the same way under VMS except that the command name is "CROBJ80". The invocation

```
80crobj
```

reads creation source from standard input, and

```
80crobj obj1.Q
```

reads creation source from *obj1.Q*. The object file is always written to standard output. Exactly one creation source file is read and processed.

By default, the object file creator promotes lower-case letters in names to upper-case; although there is no casual use of the creator which requires it, the -c switch turns off case promotion.

If the creation source file contains a numeric record length or a numeric checksum, that numeric value is written to the object file; if it does not match the actual record length or checksum, a nonfatal error message is sent to standard error.

The object file creator knows nothing about legal record types or specific record formats; every record type is considered legal.

11.3 OBJECT FILE DISPLAY

80/DSOBJ is invoked as

```
80dsobj [-cClL] [name]
```

under UNIX and PC-DOS. It is invoked the same way under VMS except that the command name is "DSOBJ80". The invocation

```
80dsobj
```

reads an object file from standard input, and

```
80dsobj obj1.q
```

reads an object file from *obj1.q*. The creation source is always written to standard output. Exactly one object file is read and processed. The object file may not be an 80/DS library. The creation source lines produced are no longer than 80 characters (plus a newline).

By default, the object file displayer gives record lengths and checksums as numeric values; the -l switch causes any record length no greater than 1025 to be given as an asterisk (a default indicator), and the -c switch causes each checksum to be given as an asterisk (a default indicator).

Numbers are displayed in groups of no more than four bytes (a leading zero and up to eight digits). Names longer than 62 characters are displayed as a one-byte numeric length followed by the proper number of numeric data bytes, rather than as quoted strings. Name characters and hexadecimal digits are given in upper-case.

The record type, a comment giving a mnemonic for the record type, the record length, and any unrepeated fields in a record are displayed on one line; the first repeated field starts on a new indented line. If the record type has no repeated fields, the checksum is given on the same line with the record type; otherwise, it is given on the same line as the last repeated field.

The object file displayer considers any record which does not have the form of a legal and recognized 8080 object record to be in error. It places a specific warning comment in the creation source near the error, and the warning comment

```
<<<<<< previous record had error >>>>>>
```

is inserted between the record in error and the next record. If the displayer encounters a relocatable or fixed-up data record with a record length greater than 1025, it places the warning comment

```
<<<<<< data record is too long >>>>>>
```

between the data record and the next record. If any records provoke warnings, a count of such records is sent to standard error after the input file has been processed. Fatal error messages are sent only to standard error, and are preceded by the ordinal of the record provoking the error.

11.4 SPECIAL VMS CONSIDERATIONS

Since these utilities take their input from SYS$INPUT and place their output on SYS$OUTPUT, it may be convenient to define a command procedure such as

```
$delete 'p1'.dso;*
$define/user sys$output 'p1'.dso
$define/user sys$input 'p1'.q80
$dsobj80
```

which will run 80/DSOBJ, taking it's input from a file with a file type of "q80" and place the output in a file with the same name but with a file type of "dso".

12. Using the Library Tools

Two tools exist for library management: the library creator, 80/LIBCR, and the library lister, 80/LIBLS.

12.1 SIMPLE LIBRARY CREATION

80/LIBCR is invoked as

```
80libcr [-f input] libname [objname ...]
```

under UNIX and PC-DOS. It is invoked the same way under VMS except that the command name is "LIBCR80".

The simplest use of the library creator generates a library from a list of (object) file names:

```
80libcr x.lib a1.q a2.q ... an.q
```

generates the library x.*lib* from the *n* object files. If no object file names are given, an empty library is created. If any of the object files is actually an Intel-format library, or is the concatenation of several object files, as might be produced by

```
cat >cat.q b1.q b2.q ... bn.q
```

under UNIX, the library creator considers each module in such an object file a separate input file.

12.2 MANY-MEMBER LIBRARY CREATION

The library creator recognizes a -f option. This option directs the library creator to use the next argument as the name of a file containing the names of object files to be added to the library (an auxiliary input file). If the next argument is '-', standard input is read for object file names. If this option is present, it must precede all other file names. Any object files named in the invocation are processed before the files named in the auxiliary input file.

12.3 SIMPLE LIBRARY LISTING

80/LIBLS is invoked as

```
80libls [-px] [libname ...]
```

under UNIX and PC-DOS. It is invoked the same way under VMS except that the command name is "LIBLS80".

The simplest use of the library lister generates an indented listing of the module names in an 80/PL library:

```
80libls x.lib
```

sends a list of the module names in x.*lib* to standard output. The -p switch additionally causes the public names associated with each module to be sent to standard output. The -x switch selects an alternate, unindented format for the list.

If no library names are given, the lister reads library names from the standard input.

12.4 LISTING FORMATS

The default listing format for a library looks like

```
library-name-1
    module-name-1
        public-name-a
        public-name-b
    module-name-2
        public-name-c
```

although the public names do not appear by default. The alternate, unindented listing format looks like

```
library-name-1:module-name-1:public-name-a
library-name-1:module-name-1:public-name-b
library-name-1:module-name-2:public-name-c
```

although, again, the public names (and their associated colons) do not appear by default.

13. Using the Tools Together

This chapter describes some of the ways in which the linker, binder, and other tools may be used together (sometimes more than once) to perform various useful tasks.

13.1 OVERLAYS

The construction of overlaid programs requires somewhat more work than the construction of an equal number of non-overlaid programs. Overlaid programs which do not reference the MEMORY segment are considered "simple". If the MEMORY segment must be referenced, either the overlay object modules must not reference MEMORY by the normal compiler or assembler keywords, or else every overlay must be sent through the binder twice. All three cases are described.

13.1.1 Making a Simple Overlay

The following procedure is used to construct a root and a number of overlays (the linker and binder invocations may be found in Section 15.14)

1. Link and bind the pieces of the root, producing a first approximation to the root. Send the output of the linker to a file other than *c.out*, send the output of the binder to a file other than *d.out*, and obtain a segment map from the binder. Unresolved external references to the overlays may still be present. If necessary, adjust the size of the stack to allow for the maximum stack usage by any overlay. Call the address at which the segment map indicates the overlays may be loaded next.
2. Link the pieces of the first overlay with the (bound) publics of the root (by using the linker -p option, or the /PUBLICS_ONLY qualifier for VMS, on the root), and bind the linked output to an address no less than next. The resulting overlay has had all external references to things in the root satisfied. No unresolved externals should be present.
3. Repeat the previous step for every other overlay.
4. Link the previously-linked (but unbound) root with the (bound) publics of all the overlays (by using the linker -p option, or the /PUBLICS_ONLY qualifier for VMS, on each of the overlays) and bind the linked output. The resulting root has had all external references to things in the overlays satisfied (the root's entry points to the different overlays must be unique publics). No unresolved externals should be present.

In general, producing a root and its n immediately-subordinate overlays requires n+1

invocations of the linker and binder. If an overlay itself has overlays, the process is continued, always from the root outward.

13.1.2 Making an Overlay Which References MEMORY Indirectly

A public variable *true$memory$base* is initialized to the address of the memory segment by the root. The overlays base an array *true$memory* on *true$memory$base*. Construct the overlay as in Section 13.1.1, but with the following modifications. Obtain a segment map from the binder for each overlay. From these maps, determine the lowest address for the memory segment which will not be destroyed by any of the overlays; call this address *mem*. Then when binding the root for the last time, place the MEMORY segment at *mem* rather than at its default location. The linker and binder invocations to do this may be found in Section 15.15.

13.1.3 Making an Overlay Which References MEMORY Directly

Both the root and the overlays reference the memory segment in the normal manner. Construct the overlay as in Section 13.1.2, but with the following modifications. When linking the overlays, send the output of the linker to a file other than c.out. Determine *mem*, and then re-bind each overlay, this time placing the memory segment at *mem* rather than at its default location. The linker and binder invocations to do this may be found in Section 15.16.

13.2 REMOVING SPECIFIC PUBLIC DEFINITIONS

Consider a library member which is itself a complicated program. Suppose the member is constructed by linking together three object modules. These three object modules communicate with each other via public symbols. The potential exists for conflict between these "private" public symbols and the public symbols in any other file to which the library member is linked. If the linked module is run through the stripper, the entry point name or names are lost; if nothing is done, duplicate public definitions will be reported.

The solution is to use the object file displayer, an editor, and the object file creator.

- Link the library member in the normal fashion.
- Generate creation source for the linked output with the object file displayer; standard output must be redirected, and default checksums and record lengths should be requested.
- Edit the creation source to remove the "private" public definitions.
- Finally, reconstruct an object file from the edited creation source with the object file generator.

This procedure may also be useful in some overlay schemes.

14. Common Topics

This chapter discusses various topics which are common to many of the 80/RL relocation and linking tools.

14.1 FUNCTIONS

This section describes some of the things each of the relocation and linkage tools can and cannot do.

14.1.1 The Functions of 80/LINK

- It produces one output object file from one or more input object files or modules.
- It produces a segment in the output object module for every distinctly-named segment in the input object files. Four "normal" segments, one unnamed COMMON segment, and at most 249 named COMMON segments are allowed. For the purposes of this discussion, a COMMON segment is equivalent to a COMMON block.
- It combines like-named segments from the input modules into single segments (conceptually – it adjusts the relative addresses of the like-named segments properly, but does not reorder the data records).
- It attempts to convert external references into intra- or inter-segment references. If it cannot resolve all external references, it generates a warning message.
- It maintains information about the ancestry of any line numbers and local symbols in the input modules.
- It does *not* generate a symbol table or a segment map.
- It does *not* indicate what modules were used in generating the final linked module.
- It does *not* allow the module name of the final linked module to be changed arbitrarily.
- It does *not* allow arbitrary library members to be specifically included in the final linked module.

14.1.2 The Functions of 80/LOC and 80/THEX

- Each binds the relocatable segments, symbols, and line numbers of a single object module to physical addresses (assigns physical addresses).
- Each resolves inter- and intra-segment references into absolute references.

- Each issues a warning if external references remain in the object module.
- Each can assign physical addresses to segments in an arbitrary order.
- Each can modify the length of any segment.
- Each can produce a memory map of the relocatable segments and previously-bound (absolute) data records.
- Each can coalesce data records referencing adjacent memory into a single data record.
- Each generates a symbol table.
- Neither removes public symbol, local symbol, or line number information.

14.1.3 The Functions of 80/MAP

- It produces a map of the symbolic information in an object module (the object module need not be absolute).
- It can sort the map so produced, either by address or by symbol.
- It can delete entire classes of symbolic information from the map.

14.1.4 The Functions of 80/STRIP

- It removes entire classes of symbolic information from an object module (the object module need not be absolute).

14.2 DEFAULT INPUTS AND OUTPUTS

The table below shows the default inputs and outputs of the various tools under UNIX and PC-DOS. It is not meaningful for VMS.

In the table, *stdin* is standard input, *stdout* is standard output.

program	default input	default output	restrictions
80/CROBJ	stdin	stdout	can't redefine output
80/DSOBJ	stdin	stdout	can't redefine output
80/HEX	stdin	stdout	can't redefine output
80/LIBCR	(none)	(none)	can't use stdin or stdout
80/LIBLS	(none)	stdout	can't redefine output
80/LINK	(none)	c.out	can't use stdin or stdout
80/LOC	c.out	d.out	can't use stdin or stdout
80/MAP	stdin	stdout	can't redefine output
80/STRIP	(none)	(input file)	can't use stdin or stdout

14.3 DEFINITIONS

This section defines a few of the terms that are used in discussing the various relocation and linking tools.

absolute	Bound to a physical address or physical addresses; generally synonymous with *bound*.
absolute data	Data which refers to specific physical 8080 addresses.
absolute module	An object module which has been bound to physical addresses (either by the method in which it was created, or by virtue of having been passed through the binder)
absolute public	A public which has a physical 8080 address associated with it.
absolute segment	A segment which has physical 8080 addresses associated with it. An absolute segment no longer has a name associated with it in an object file.
binder	A program which binds (associates) object module segments to physical addresses; 80/LOC.
bound	Generally synonymous with *absolute*.
creation source	A primitive readable form for an object file.
creator	A program which produces an object file from creation source; 80/CROBJ.
debug information	Local symbols, line numbers, and ancestor module names. These are never required for relocation and linkage, but may be useful when debugging an executable program.
displayer	A program which produces creation source from an object file; 80/DSOBJ.
linker	A program which produces a single object module from a collection of object modules, maintaining the inter-module references in a consistent format; 80/LINK.
map	A human-readable association of symbolic and address information.
map generator	A program which displays the association between symbols and addresses (physical or relocatable); 80/MAP.
partial segment	A relocatable segment from a single input object module. To combine two partial segments means to combine the like-named segments from two input object modules.
publics-only file	A file which is used only for the absolute public definitions which it contains.
stripper	A program which removes classes of information from a file; 80/STRIP.
switch modifier	The modifier to a switch is just the remainder of the switch argument. Switch modifiers may be *simple* (contain only alphabetic

and numeric characters) or *compound* (contain special characters, such as +, -, =).

15. Examples: UNIX and PC-DOS

This chapter presents a number of examples that demonstrate the use of the 80/RL relocation and linking tools under the UNIX and PC-DOS operating systems. See Chapter 16 for a description of their use under VMS.

15.1 SIMPLE DEFAULT LINK

```
80link obj1.q obj2.q ... objn.q
```

Each file is an object file or a library. The output is left in *c.out*.

15.2 SIMPLE REDIRECTED LINK

```
80link obj1.q -o ovl.lnk obj2.q ... objn.q
```

Each file *obj1.q*, *obj2.q*, ... is an object file or a library. The output is left in *ovl.lnk*.

```
80link obj1.q obj2.q ... objn.q -o
```

Fatal error: the linker requires a file name for the -o switch.

```
80link -o x obj1.q -o y obj2.q ... objn.q
```

Fatal error: the linker -o switch may be used only once.

15.3 DEFAULT LINK USING BOUND PUBLICS

```
80link x1.q x2.q x3.q -p a x4.q x.lib
```

Bound (absolute) publics in *a* resolve externals in *x1.q*, *x2.q*, and *x3.q* (but not in *x4.q* or *x.lib*).

```
80link -p b x1.q x2.q x3.q x4.q x.lib
```

The object file *b* is searched for absolute publics, but any found are ignored (don't do this).

15.4 LIBRARIES WITH MULTIPLY-DEFINED PUBLICS

Suppose the library *lib1* has these members in the indicated order:

1. *M1a* has the public *a*
2. *M1b* has the external *a* and the public *b*

3. *M1c* has the public *a*

M1c contains a default definition of *a*. If both *a* and *b* are external when *lib1* is encountered, then *M1a* and *M1b* will be extracted. If *b* is external and *a* is neither public nor external when *lib1* is encountered, then *M1b* and *M1c* will be extracted.

Suppose the library *lib2* has these members in the indicated order:

1. *M2a* has the public *c*
2. *M2b* has the publics *c* and *d*
3. *M2c* has the public *c*

It is not possible to extract *M2c*. If *d* is external and *c* is neither public nor external when *lib2* is encountered, then *M2b* will be extracted, and no error will be noted. If *d* is external and *c* is public when *lib2* is encountered, then *M2b* will be extracted, and a "duplicate public" error will be noted (the definition of *c* in *M2b* is not used to resolve any external references to *c*, even ones in modules provided to the linker after *lib2*). If both *c* and *d* are external when *lib2* is encountered, then *M2a* and *M2b* will be extracted, and a "duplicate public" error will be noted (again, the definition of *c* in *M2b* is not used to resolve any external references to *c*).

15.5 SPECIFYING ALIGNMENT

Suppose the object file *c.out* contains the following segments:

1. byte-aligned CODE segment, length 150 bytes (96h)
2. page-aligned DATA segment, length 193 bytes (0c1h)
3. inpage-aligned STACK segment, length 48 bytes (30h)

Invoking the binder with and without the -a switch causes segments to be bound as follows:

```
80loc -15000h
```

CODE is bound to 05000h, DATA to 05100h, STACK to 051c1h;

```
80loc -15030h -scode -a20h
```

CODE is bound to 05030h, DATA to 05100h (not 050e0h), STACK to 051c1h;

```
80loc -15000h -sallother -a10h -sstack
```

CODE is bound to 05000h, DATA to 05100h, STACK to 051d0h;

```
80loc -15000h -sallother -a20h -sstack
```

CODE is bound to 05000h, DATA to 05100h, STACK to 05200h (not 051e0h).

15.6 ADJUSTING THE SIZE OF A SEGMENT

```
80loc -15000h -scode -sdata -sstack=100
```

sets the size of the stack to 100 (decimal) bytes (typically, the linked stack size – the sum of the sizes of the stack segments in the linker's input – is far larger than actually required). Note that the last switch defines both the size *and the position* of the stack.

15.7 SORTED AND COMPACTED BOUND OUTPUT

```
80loc -c -15000h
```

The input file is c.out and the output file is d.out. Segments are bound in their default order, beginning at location 5000h. Data records which reference unresolved external symbols are written to the output file before data records which do not reference unresolved external symbols. Data records which do not reference unresolved external symbols appear in their physical address order (rather than their order of appearance in the input file), and compacted. That is, a data record which contains data for physical addresses 5000h through 5043h, and a data record which contains data for physical addresses 5044h through 5077h will be converted into a single data record which contains data for physical addresses 5000h through 5077h.

15.8 SORTED AND COMPACTED BOUND OUTPUT, TEKTRONIX FORMAT

The 80thex utility may be used, in conjunction with 80loc, to produce compacted, sorted output in Tetronix format.

```
80loc -c -15000h
80thex d.out
```

The first command is exactly the example in Section 15.7. The second command reads the (absolute) object file d.out (produced by the first command), and converts it to Tektronix LAS object format in the file e.out. Data records will retain the same relative ordering, although a single long data record in d.out may be converted into several shorter (but adjacent) data records in e.out.

15.9 SYMBOLIC MAP EXAMPLES

```
80map root
```

An unsorted machine-searchable symbolic map of root is sent to standard output.

```
80map ovl1 ovl2 ovl3
```

An unsorted machine-searchable symbolic map of the three overlays is sent to standard output. Each line in the map indicates the file and module which produced it.

```
80map -fs ovl1 ovl2 ovl3
```

An unsorted simple symbolic map of the three overlays is sent to standard output. One line is produced whenever the file, module, or ancestor module name changes. An automatic (machine) search of standard output will not indicate the file or module in which a symbol or line number resides.

```
80map -fs -sa -aafm runme
```

A simple symbolic map of runme, sorted by address, is sent to standard output. Only line numbers and public, local and external symbols appear in the map.

```
80map -sa -aafm this
80map -sa this
```

produce identical machine-searchable symbolic maps of *this*, sorted by address.

```
80map -sn -aw+l this -a-l+s that -a-s+p tother -sa
```

produces a machine-searchable symbolic map of the local symbols from *this*, the line numbers from *that*, and the public symbols from *tother*. The entire map is sorted by address (not by name; the request for sorting by address overrides the request for sorting by name).

```
80map this -fs that
```

Fatal error: The map format may not be changed after a file is processed.

```
80map this -sa that
```

Fatal error: The map may not be changed from unsorted (the default) to sorted after a file is processed.

15.10 DEFAULT STRIP

```
80strip obj1.q obj2.q obj3.q
```

removes all public symbol, local symbol, line number, and ancestor module name information from the three object files *obj1.q*, *obj2.q*, and *obj3.q*. Note that none of the three files may later be used to resolve external references.

15.11 RESTRICTED STRIP

```
80strip -p big.q
```

removes all local symbol, line number, and ancestor module name information from the object file *big.q*; public symbols are not removed. The file may later be used to resolve external references.

```
80strip -p a.q -l b.q -Ps c.q -LS d.q
```

The four files are affected as indicated:

file	*information removed*
a.q	local symbol, line number, ancestor module name
b.q	local symbol (ancestor module names are not removed)
c.q	public symbol
d.q	public symbol, local symbol, line number, ancestor module name

15.12 MANY-MEMBER LIBRARY CREATION

Suppose the object files *a1.q*, *a2.q*, ... , *a200.q* must be placed in the library *big.lib*. Suppose further that the text file *aux.in* contains the lines

```
a1.q a2.q a3.q a4.q
a5.q
a6.q a7.q
...
a198.q a199.q a200.q
```

Then either of the invocations

```
80libcr big.lib a1.q a2.q a3.q ... a199.q a200.q
```

or

```
80libcr -f aux.in big.lib
```

will create the desired library.

15.13 LIBRARY LISTING

```
80libls x.lib y.lib
```

Lines of the form

```
x.lib
    first
    second
y.lib
    organic
    inorganic
    physical
```

are sent to standard output;

```
80libls -px x.lib y.lib
```

sends lines of the form

```
x.lib:first:a
x.lib:first:b
x.lib:second:hippo
x.lib:second:rhino
y.lib:organic:saturated
y.lib:organic:unsaturated
y.lib:inorganic:metals
y.lib:inorganic:nonmetals
y.lib:physical:classical
```

to standard output.

15.14 SIMPLE OVERLAY

```
80link ra.q rb.q rc.q x.lib -o root.lnk
80loc root.lnk -15000h -o root -m root.map
: Assume root.map indicates that
:       next is 8000h
80link oa.q ob.q -p root x.lib
80loc -18000h -o overlay1
80link pa.q pb.q -p root x.lib
80loc -18000h -o overlay2
...
80link root.lnk -p overlay1 -p overlay2 ...
80loc -15000h -o root
```

15.15 OVERLAY REFERENCING MEMORY INDIRECTLY

```
80link ra.q rb.q rc.q x.lib -o root.lnk
80loc root.lnk -15000h -o root -m root.map
: Assume root.map indicates that
:       next is 8000h.
80link oa.q ob.q -p root x.lib
80loc -18000h -o overlay1 -m ov1.map
80link pa.q pb.q -p root x.lib
80loc -18000h -o overlay2 -m ov2.map
...
: Assume ov1.map, ov2.map, ...
:       indicate that mem is 9600h.
80link root.lnk -p overlay1 -p overlay2 ...
80loc -15000h -sallother -19600h -smemory -o root
```

15.16 OVERLAY REFERENCING MEMORY DIRECTLY

```
80link ra.q rb.q rc.q x.lib -o root.lnk
80loc root.lnk -15000h -o root -m root.map
: Assume root.map indicates that next is 8000h.
80link oa.q ob.q -p root x.lib -o ov1.lnk
80loc -18000h -o overlay1 -m ov1.map
80link pa.q pb.q -p root x.lib
80loc -18000h -o overlay2 -m ov2.map
...
: Assume ov1.map, ov2.map, ... indicate
:       that mem is 9600h.
80loc ov1.lnk -18000h -sallother -19600h
        -smemory -o overlay1
80loc ov2.lnk -18000h -sallother -19600h
        -smemory -o overlay2
...
80link root.lnk -p overlay1 -p overlay2 ...
80loc -15000h -sallother -19600h -smemory -o root
```

15.17 OVERLAYING TWO SEGMENTS

Two segments can be overlaid by absolute physical addresses

```
80loc -15000h -s/com1/ -15000h -s/com2/ -lmax
```

or by symbolic addresses

```
80loc -15000h -s/com1/ -l/com1/ -s/com2/ -lmax
```

Each of the two examples causes the two COMMON blocks *com1* and *com2* to use the same memory. The MAX pseudo-segment guarantees that later segments do not overlap the end of *com1*. The second method is much more useful than the first whenever segment order is at least partially fixed, but only one segment address is actually important:

```
80loc -15000h -sdata -sstack -s/com1/ -l/com1/
        -s/com2/ -lmax -scommons
```

In every example, an overlap indication would be sent to the segment map, if requested.

15.18 REMOVING "PRIVATE" PUBLICS

It is desired to remove all public symbols except *one*, *two*, and *three* from the object file xx.q.

```
80dsobj -cl xx.q >xx.Q
: edit xx.Q, removing all publics except one,
:       two, and three
80crobj xx.Q >xx.q
```

xx.q has effectively been partially stripped.

16. Examples: VMS

This chapter presents a number of examples that demonstrate the use of the 80/RL relocation and linking tools under the VMS operating system. See Chapter 15 for a description of their use under UNIX and PC-DOS.

16.1 SIMPLE DEFAULT LINK

```
80link obj1,obj2,...,objn
```

Each file is an object file or a library with a file type of "Q80". The output is left in *obj1.l80*.

16.2 SIMPLE REDIRECTED LINK

```
80link/output=ovl obj1,obj2,objn
```

Each file *obj1.q80*, *obj2.q80*, ... is an object file or a library. The output is left in *ovl.l80*.

16.3 DEFAULT LINK USING BOUND PUBLICS

```
80link x1,x2,x3,a/publics,x4,x.lib
```

Bound (absolute) publics in *a.q80* resolve externals in *x1.q80*, *x2.q80*, and *x3.q80* (but not in *x4.q80* or *x.lib*).

```
80link b/publics,x1,x2,x3,x4,x.lib
```

The object file *b* is searched for absolute publics, but any found are ignored (don't do this).

16.4 LIBRARIES WITH MULTIPLY-DEFINED PUBLICS

Suppose the library *lib1* has these members in the indicated order:

1. *M1a* has the public *a*
2. *M1b* has the external *a* and the public *b*
3. *M1c* has the public *a*

M1c contains a default definition of *a*. If both *a* and *b* are external when *lib1* is encountered, then *M1a* and *M1b* will be extracted. If *b* is external and *a* is neither public nor external when *lib1* is encountered, then *M1b* and *M1c* will be extracted.

Suppose the library *lib2* has these members in the indicated order:

1. *M2a* has the public *c*
2. *M2b* has the publics *c* and *d*
3. *M2c* has the public *c*

It is not possible to extract *M2c*. If *d* is external and *c* is neither public nor external when *lib2* is encountered, then *M2b* will be extracted, and no error will be noted. If *d* is external and *c* is public when *lib2* is encountered, then *M2b* will be extracted, and a "duplicate public" error will be noted (the definition of *c* in *M2b* is not used to resolve any external references to *c*, even ones in modules provided to the linker after *lib2*). If both *c* and *d* are external when *lib2* is encountered, then *M2a* and *M2b* will be extracted, and a "duplicate public" error will be noted (again, the definition of *c* in *M2b* is not used to resolve any external references to *c*).

16.5 ALIGNMENT

Suppose the object file *c.l80* contains the following segments:

1. byte-aligned CODE segment, length 150 bytes (96h)
2. page-aligned DATA segment, length 193 bytes (0c1h)
3. inpage-aligned STACK segment, length 48 bytes (30h)

Invoking the binder with and without the alignment option causes segments to be bound as follows:

```
80loc c /MEMORY=LOC:5000h
```

CODE is bound to 05000h, DATA to 05100h, STACK to 051c1h;

```
80loc c /MEMORY=(LOC:5030h,SEG:code,ALIGN:20h)
```

CODE is bound to 05030h, DATA to 05100h (not 050e0h), STACK to 051c1h;

```
80loc c /MEM=(L:5000h,S:allother,A:10h,S:stack)
```

CODE is bound to 05000h, DATA to 05100h, STACK to 051d0h;

```
80loc c /MEM=(L:5000h,S:allother,A:20h,S:stack)
```

CODE is bound to 05000h, DATA to 05100h, STACK to 05200h (not 051e0h).

16.6 ADJUSTING THE SIZE OF A SEGMENT

```
80loc c /MEM=(L:5000h,S:code,S:data,S:stack=100)
```

sets the size of the stack to 100 (decimal) bytes (typically, the linked stack size – the sum of the sizes of the stack segments in the linker's input – is far larger than actually required). Note that the last switch defines both the size and the position of the stack.

16.7 SORTED AND COMPACTED BOUND OUTPUT

```
80loc c /COMPACT /MEMORY=(L:5000h)
```

The input file is *c.l80* and the output file is *c.b80*. Segments are bound in their default order, beginning at location 5000h. Data records which reference unresolved external symbols are written to the output file before data records which do not reference unresolved external symbols. Data records which do not reference unresolved external symbols appear in their physical address order (rather than their order of appearance in the input file), and compacted. That is, a data record which contains data for physical addresses 5000h through 5043h, and a data record which contains data for physical addresses 5044h through 5077h will be converted into a single data record which contains data for physical addresses 5000h through 5077h.

16.8 SYMBOLIC MAP EXAMPLES

```
80map root
```

An unsorted machine-searchable symbolic map of *root* is sent to *root.map*.

```
80map ovl1,ovl2,ovl3
```

An unsorted machine-searchable symbolic map of the three overlays (*ovl1.b80*, *ovl2.b80*, and *ovl3.b80*) is sent to *ovl1.map*. Each line in the map indicates the file and module which produced it.

```
80map /brief ovl1,ovl2,ovl3
```

An unsorted simple symbolic map of the three overlays is sent to *ovl1.map*. One line is produced whenever the file, module, or ancestor module name changes. An automatic (machine) search of the output will not indicate the file or module in which a symbol or line number resides.

```
80map /brief /sort=address runme -
      /items=(noanc,nofil,nomod)
```

A simple symbolic map of *runme.b80*, sorted by address, is sent to standard output. Only line numbers and public, local and external symbols appear in the map.

```
80map /sort=address /items=(noanc,nofil,nomod) this
80map /sort=address this
```

produce identical machine-searchable symbolic maps of *this.b80*, sorted by address.

```
80map /sort=names this/items=(none,line) -
      that/items=(none,sym) -
      tother/items=(none,pub)
```

produces a machine-searchable symbolic map of the local symbols from *this.b80*, the line numbers from *that.b80*, and the public symbols from *tother.b80*. The entire map is sorted by name.

16.9 DEFAULT STRIP

```
strip80 obj1.q obj2.q obj3.q
```

removes all public symbol, local symbol, line number, and ancestor module name information from the three object files *obj1.q*, *obj2.q*, and *obj3.q*. Note that none of the three files may later be used to resolve external references.

16.10 RESTRICTED STRIP

```
strip80 -p big.q
```

removes all local symbol, line number, and ancestor module name information from the object file *big.q*; public symbols are not removed. The file may later be used to resolve external references.

```
strip80 -p a.q -l b.q "-Ps" c.q "-LS" d.q
```

The four files are affected as indicated:

file	*information removed*
a.q	local symbol, line number, ancestor module name
b.q	local symbol (ancestor module names are not removed)
c.q	public symbol
d.q	public symbol, local symbol, line number, ancestor module name

16.11 MANY-MEMBER LIBRARY CREATION

Suppose the object files *a1.q*, *a2.q*, ... , *a200.q* must be placed in the library *big.lib*. Suppose further that the text file *aux.in* contains the lines

```
a1.q a2.q a3.q a4.q
a5.q
a6.q a7.q
...
a198.q a199.q a200.q
```

Then either of the invocations

```
libcr80 big.lib a1.q a2.q a3.q ... a199.q a200.q
```

or

```
libcr80 -f aux.in big.lib
```

will create the desired library.

16.12 LIBRARY LISTING

```
liblst80 x.lib y.lib
```

Lines of the form

```
x.lib
    first
    second
y.lib
    organic
    inorganic
    physical
```

are sent to standard output;

```
liblst80 -px x.lib y.lib
```

sends lines of the form

```
x.lib:first:a
x.lib:first:b
x.lib:second:hippo
x.lib:second:rhino
y.lib:organic:saturated
y.lib:organic:unsaturated
y.lib:inorganic:metals
y.lib:inorganic:nonmetals
y.lib:physical:classical
```

to standard output.

16.13 SIMPLE OVERLAY

```
80link/root.lnk ra.q rb.q rc.q x.lib
80loc/out=root/map=root.map root.lnk /mem=L:5000h
: Assume root.map indicates that
:       next is 8000h 80link oa.q,ob.q,root/pub,x.lib
80loc/out=overlay1 oa /mem=L:8000h
80link pa.q,pb.q,root/pub,x.lib
80loc/out=overlay2 pa /mem=L:8000h -o overlay2
...
80link root.lnk,overlay1/pub,overlay2/pub ...
80loc/out=root root /mem=L:5000h
```

16.14 OVERLAY REFERENCING MEMORY INDIRECTLY

```
80link ra.q,rb.q,rc.q,x.lib /out=root.lnk
80loc root.lnk /mem=L:5000h /out=root /map=root.map
: Assume root.map indicates that
:       next is 8000h.
80link oa.q,ob.q,root/pub,x.lib
80loc oa /map=L:8000h /out=overlay1 /map=ov1.map
80link pa.q,pb.q,root/pub,x.lib
80loc pa /map=L:8000h /out=overlay2 /map=ov2.map
...
: Assume ov1.map, ov2.map, ...
:       indicate that mem is 9600h.
80link root.lnk,overlay1/pub,overlay2/pub ...
80loc root /mem=(L:5000h,S:allother,L:9600h,S:memory) -
        /out=root
```

16.15 OVERLAY REFERENCING MEMORY DIRECTLY

```
80link ra.q.rb.q,rc.q,x.lib /out=root.lnk
80loc root.lnk /mem=L:5000h /out=root /map=root.map
: Assume root.map indicates that next
:       is 8000h.
80link oa.q,ob.q,root/pub,x.lib /out=ov1.lnk
80loc oa /map=L:8000h /out=overlay1 /map=ov1.map
80link pa.q,pb.q,root/pub,x.lib
80loc pa /mem=L:8000h /out=overlay2 /map=ov2.map
...
: Assume ov1.map, ov2.map, ... indicate
:       that mem is 9600h.
80loc ov1.lnk /mem=(L:8000h,S:allother,L:9600h, -
        S:memory) /out=overlay1
80loc ov2.lnk /mem=(L:8000h,S:allother,L:9600h, -
        S:memory) /out=overlay2
...
80link root.lnk,overlay1/pub,overlay2/pub ...
80loc root /mem=(L:5000h,S:allother,L:9600h, -
        S:memory) /out=root
```

16.16 OVERLAYING TWO SEGMENTS

Two segments can be overlaid by absolute physical addresses

```
80loc f /mem=(L:5000h,S:|com1| ,L:5000h,
            S:|com2| ,L:max)
```

or by symbolic addresses

```
80loc f /mem=(L:5000h,S:|com1| ,L:|com1| ,
            S:|com2| ,L:max)
```

Each of the two examples causes the two COMMON blocks *com1* and *com2* to use the

same memory. The MAX pseudo-segment guarantees that later segments do not overlap the end of com1. The second method is much more useful than the first whenever segment order is at least partially fixed, but only one segment address is actually important:

```
80loc file /mem=(L:5000h,S:data,S:stack,S:|com1|,
        L:|com1|,S:|com2|,L:max,Sscommons)
```

In every example, an overlap indication would be sent to the segment map, if requested.

Index