Chapter 14 Command references

This chapter describes references necessary for each command. Command references are listed in the alphabetical order.
This section describes how to read the command references. The references include the following items.

[Name]	This shows a command name. Following a hyphen (-), a brief explanation of the command is put.
[Syntax]	A syntax to cal up the command is described. Options enclosed by [ ] can be omitted. Additionally, | shows that either one is selected.
[Comment]	This describes how the command functions.
[Option]	Options applicable to the command are listed.
[Example]	Examples of the actual command usage are described.
[File]	Files are shown that the command creates or refers to.
[Environmental variable]	Environmental variables to be used by the command are shown.
[End code]	If the command returns values to the environment, these values and their meanings are described.
[Reference]	Items closely related to the command are shown.
[Diagnosis message]	Error and warning messages, which are detected by the command, are listed.
[Bug]	Cautions for use of the command are described.

CPP (1)

[Name]

cpp - preprocessor

[Syntax]

cpp [-B] [-c] [-D*NAME*][=*value*]] [-I*directory*] [-j[X]] [-o*file*] [-s] [-w] [-y[*N*]] [*inputfile…*]

[Comment]

The ccp is a C preprocessor and the first path of the compiler. The ccp processes preprocessor commands, such as #define, #include, and #if, and writes the result to the standard output. The output destination can be changed by -o option.

If *inputfile* is given, the ccp processes it as an input. It is also possible to specify multiple input files. If no *inputfile* is specified, the standard input is used.

[Option]

-B	This option makes use of a C++ comment starting with // enabled.
-c	This option enables nesting of comments.
-D*NAME*[=*value*]	This option defines a macro called NAME. If "=*value*" is omitted, =1 is used.
-I*directory*	This option searches the directory "*directory*" for files to be included. It is also possible to specify more than one this option.
-j[X]	This option is used to select which language is applied to error message display.

For MS-DOS and Windows

-j0	Error messages are displayed in English.
-j, -j1	Error messages are displayed in Japanese. "Shift-JIS" is used for Japanese character codes.

For UNIX

-j0	Error messages are displayed in English.
-j, -j1,-je	Error messages are displayed in Japanese. "EUC" is used for Japanese character codes.
-js	Error messages are displayed in Japanese. "Shift-JIS" is used for Japanese character codes.

In either OS, if this option is omitted, this option becomes as -j0 is specified.

`-o`*`file`* This option writes the output to “`file`”.

`-s` This option does not output line number information starting with # to the output.

`-w` This option makes warning message display disabled.

`-y[N]` This option specifies handling of the extended character code set. Since the `ccp` normally handles data as shift JIS code strings, it cannot handle a character string in a system other than JIS shift code system (for example a heart mark in IBM-PC) in correct manner if it appears. To solve this problem, the following option must be specified.

- `-y` All character codes are handled as 1-byte characters.
- `-y0` All character codes are handled as Japanese character codes (shift JIS code).
- `-y1` All character codes are handled as Chinese and Taiwanese codes.
- `-y2` All character codes are handled as Korean codes.

If no option is specified, the command functions as `-y0` is specified.

[Example]

```
cpp  -I/usr/include  -DANSI  -otest.out  test.c
```

In the above example, `test.c` is processed and the result is output to test.out. Files to be included are found in the directory “`/usr/include`”. Additionally, the macro ANSI is defined to 1.

[End code]

If the `ccp` is completed successfully, 0 is returned. If an error is detected, 1 is returned. If a warning occurs, 2 is returned.

KMMAKE (1)

[Name]

kmmake - program maintenance

[Syntax]

kmmake [-f file] [-i] [-k] [-n] [-r] [-t] [NAME = value…] [target…]

[Comment]

The kmmake executes commands specified in the makefile to update one or more target. If -f option is not specified, commands are read from file "makefile" in the current directory.

[Option]

-d	This option displays the debug information.
-f file	This option specifies the name of a file, from which kmmake commands are read. If this option is not specified, the file "makefile" in the current directory is read. If "-" is specified in "file", commands are input from the standard input.
-i	This option disregards the return code from the executed command. The kmmake functions in the same manner as that ".IGNORE" is specified.
-k	This option creates as many targets as possible without exiting of the command if an error is detected while the command is being executed.
-n	This option does not execute the command, but displays commands, which may be executed. Use of this option makes it possible to check which commands are executed before starting execution of the commands.
-r	This option makes the rules for omission of the kmmake invalid. That is, the makedef is not read.
-s	The kmmake normally displays commands currently being executed. However, if this option is specified, commands are not displayed. The kmmake functions in the same manner as that ".SILENT" is specified.
-t	This option updates only time stamps for the target files without execution of commands.

[Example]

```
kmmake -f makeit all
```

The above example reads the dependent relationship from the file "makeit" in the current directory and creates target "all".

`kmmake -n`
The above example analyzes the dependent relationship, but executes no commands, and displays commands scheduled to be executed.

`kmmake -f b:filemake CFLAG = -DDEBUG`
The above example reads the dependent relationship from the file "`b:filemake`" instead of `makefile`. Additionally, the macro named `CFLAG` is defined in "`-DDEBUG`".

[File]

`makedef`	This file describes the default rules.
`makefile`	This file describes the `kmmake` commands. If -f option is not specified, `makefile` in the current directory is used instead.

LCC86 (1)

[Name]

lcc86 - compiler driver

[Syntax]

lcc86 [@file] [-a [file]] [-B] [-c] [-cn] [-cs | -cu] [-cv] [-DNAME [=value]] [-E] [-g] [-h] [-Idir] [-j[X]] [-kX] [-llibrary] [-Ldir] [-m] [-mX] [-mslink | -nomslink] [-nN] [-noconf] [-O[0]] [-ofile] [-P] [-p] [-S[c]] [-Tdir] [-v[X]] [-w[X]] [-Xdir] [-y[X]] [-z] sourcefile…

[Comment]

The lcc86 calls up the C compiler, assembler, and linker to convert the C source programs into those in the executable format.

For source files, C language files with an extension of ".c", assembly language files with an extension of ".a86 (r86)", ".p86 (r86)", or ".asm (MASM)", and object files with an extension of ".obj" can be specified. More than one file can be written and they are linked together.

The lcc86 first compiles or assembles these files to make object files. If the extension of an assembly language file is ".a86" or ".p86", it is assembled by the r86 after the ccp is executed. If the extension is ".asm", the assembly language file is assembled by the MASM.

If no error is found, all objects are linked to create one executable file (with an extension of ".exe"). The same name as that of the first source file is put on this ".exe" file. (The name of a file to be created can be changed using -o option.)

If an error is found, the command is executed until objects are created, but linking is not made. Since the error message is output to the standard output, it is possible to output it to a file using redirect ">" of the standard output.

For options that need to specify a directory name, either putting or not putting "/" (or "\") at the end may be accepted. Please also note that a letter in the capital is regarded as a different letter when it is used in the small letter.

The lcc86 analyzes character strings given in command lines in sequential order. If a character string is an option, relevant process is executed. If it is not an option, it is thought as a source file name, and then compilation is executed using options specified at that time. For example,

```
lcc86 -c a.c -S b.c -p
```

if the above command is given, the command is executed until assembling to create an object file named “a.obj” since -c is specified for a.c. On the contrary, since -S is valid when compiling b.c, the file “b.c” is compiled to assembly language file “b.a86”.

When the lcc86 recognizes “-p” specified at the end, codes for the profiler are not generated since both are already compiled. As described above, the order in which options and source files are specified must always checked carefully.

[Option]

@file	This option reads command line parameters from the file “file”. Options and source file names divided by a space or LF code are written in “file”. These options can be placed halfway or twice or more as shown in the following example. `lcc86 -o foo.exe @files @libs -m` However, nesting is not allowed. That is, @ option cannot be used in “*file*”.
-a[*file*]	This option specifies “*file*” as the initialization routine. This file option must exist in a directory specified by -L option. Additionally, -a and file must be written continually without spaces. If “*file*” is blank (only -a is specified), it is considered that there is no initialization routine.
-B	This option makes use of a C++ comment starting with // enabled. However, note that use of // comment lowers the portability.
-c	This option does not execute the link after creating object files. If there is only one file to be compiled, it is possible to specify an object file name using *-o* option.
-cn	This option makes nesting of comments enabled.
-cs, -cu	-cs option specifies that the char type is signed. -cu option specifies that the char type is unsigned. If this option is omitted, the char type is unsigned.
-cv	This option creates variables allocated to the class DATA in declaration order. If this option is omitted, variables are sorted and created in alphabetical sort.
-DNAME [=value]	This option defines a macro named NAME. This option functions in the same manner as that the following line is placed at the beginning of the C language source file. `#define NAME value` However, “=value” is omitted, it functions in the same manner as described below. #define NAME 1

`-E` This option executes only the preprocessor `ccp`. The result is output to the standard output or a file specified by -o option. The output includes the preprocessor line number information.

`-g` This option embeds the debug information (line number information) in the object. This is used by Microsoft's SYMDEB.

`-h` This option generates codes to inspect the stack overflow. When this option is used, the execution speed of the created program becomes slow. If the stack overflow is detected in the created program, a return code of 9 is returned and the execution is stopped.

`-I`*`dir`* This option adds "`dir`" to the list of directories where files specified by `#include` statement are searched for. Multiple `-I` options can be specified. At this time, all directories specified are searched.

`-j[X]` This option is used to select which language is applied to error message display.

For MS-DOS and Windows

`-j0`	Error messages are displayed in English.
`-j, -j1`	Error messages are displayed in Japanese. "Shift-JIS" is used for Japanese character codes.

For UNIX

`-j0`	Error messages are displayed in English.
`-j, -j1,-je`	Error messages are displayed in Japanese. "EUC" is used for Japanese character codes.
`-js`	Error messages are displayed in Japanese. "Shift-JIS" is used for Japanese character codes.

In either OS, if this option is omitted, this option becomes as `-j0` is specified.

`-kX` This option transfers option X to the linker. For example, to set a stack size, the following command is specified using this option.

```
lcc86  foo.c -k"-s 5000"
```

In the above example, foo.c is compiled and the stack size is set to 0x5000 bytes (-s 5000 is an option of lld). In this example, -s 5000 is enclosed by " ", but use of ' ' is recommended in UNIX.

`-l`*`library`* This option scans the library "*`library`*" during linking. The file "*`library`*" is considered as that it is in a directory specified by `-L` option. Additionally, when an extension of "`.obj`" is identified, it is possible to connect object files in the library directory. If the linker lld encounters this option, it immediately searches for unsolved symbol. Therefore, the order of writing options has the

meaning. Note that desired results are not obtained even though libraries are specified preceding the object files.

`-L`*`dir`* This option adds "`dir`" to the list of directories where a library is searched for. At this time, there must be S, P, D, and L sub-directories corresponding to each memory size. Actual libraries must be put in these sub-directories.

`-m` This option creates a linkage map file. No line number information is output to the map file.

`-m`X This option specifies a memory model. Any of s, p, d, and l is specified for X. If this option is not specified, S model is selected (-ms).

`-mslink, -nomslink`
`-mslink` option declares that MS-LINK is used as a linker. `-nonmslink` option declares that the standard `lld` is used as a linker. This option must be specified preceding `-k` option. If this option is not specified, the `lld` is used as a linker. This option is invalid in the UNIX version.

`-n`N This option sets the upper limit for the number of continuously detected errors to N (N is in decimal). If the number of detected errors exceeds the set level, compilation of that file is stopped. The default setting of N is 20.

`-noconf` This option makes the configuration file invalid. This option must be written at the beginning of the command line. If this option is written at other position, an error may result.

`-O[0]` When `-O` is specified, the compiler performs the optimization. When `-O0` is specified, the compiler does not perform the optimization. If this option is not specified, the optimization is not made (`-O0`). In the current version, only optimization of jump commands by the assembler r86 is controlled by `-O0` option.

`-o` *`file`* This option specifies the output file name to "*file*". Normally, this file name is a name of the executable file. However, if -c option is specified, the object file name is set to the executable file name. If *-s* option is specified, the assembly file name is set to the executable file name.

`-P` This option executes only the preprocessor `ccp`. The result is output to a file with an extension of "`.i`". The output does not include the preprocessor line number information.

`-p` This option generates codes for the profiler. The profiler is a utility that checks the program execution status and reports how many times each program part is executed.

`-S[C]` -S option creates an assembly language file (`.a86`) for each C language file. Assembling and linking are not made. -SC option embeds C source codes in the created assembly language file in

the comment format. Created files are not assembled by the MS-DOS standard assembler “MASM”. At this time, the r86 supplied with this package must be used.

`-T`*`dir`* This option creates a compiler work file in the directory “*`dir`*”. If a high-speed access device, such as RAM disk is provided, specifying this directory may greatly improve the compilation speed. If this option is not specified, the current directory is used.

`-v[X]` This option specifies how the display functions during compilation.

- `-v0` Nothing is displayed.
- `-v, -v1` Commands executed internally are displayed.
- `-v2` In addition to -v1, names of functions currently being processed are displayed during code generation.

If this option is not specified, the command functions in the same manner as that -v0 is specified.

`-w[X]` This option specifies the handling method, if a warning is detected.

- `-w, -w0` No warning message is displayed and compilation is continued.
- `-w1` Warning message is displayed and compilation is continued.
- `-w2` Warning message is displayed and compilation is stopped.
- `-wi` Warning message is controlled, which reported if undeclared function is used.

If this option is not specified, the command functions in the same manner as that `-w1` is specified. If an error is detected, the compilation is stopped regardless of designation of this option.

`-X`*`dir`* This option searches the directory “*`dir`*” for compiler internal commands (cf, cg86, or etc.). If this option is not specified, a directory specified in the environmental variable “PATH” is searched.

`-y[N]` This option specifies handling of the extended character code set. Since the lcc86 normally handles the contents of character strings as shift JIS code strings, it cannot directly handle a character string in a system other than JIS shift code system (for example a heart mark in IBM-PC) in correct manner if it appears. To solve this problem, the following option must be specified.

- `-y` All character codes are handled as 1-byte characters.
- `-y0` All character codes are handled as Japanese character codes (shift JIS code).
- `-y1` All character codes are handled as Chinese and Taiwanese codes.
- `-y2` All character codes are handled as Korean codes.

If no option is specified, the command functions in the same manner as that `-y0` is specified.

`-z` This option displays commands to be executed. The commands are not executed. This option is used to check whether or not the environmental settings are correct.

[Example]

```
lcc86 -DDEBUG -T/tmp test.c
```

In this example, macro DEBUG is defined to `1` and `test.c` is compiled. A temporary file created by the compiler is placed in the directory /temp.

```
lcc86 -ml -o foo.exe main .c sub.c -lmathlib
```

In this example, `main.c` and `sub.c` are compiled with a large model. They are gathered together by the linker and the name of an execution file is "`foo.exe`". Additionally, `mathlib.lib` is scanned during linking.

[File]

MS-DOS and Windows version

`_lcc86`

Environmental settings necessary for the compiler are described in _lcc86. If the lcc86 does not find ".lcc86" in the current directory, it searches directories defined in the environmental variable PATH in order.

UNIX version

`.lcc86, options.lcc86`

Environmental settings necessary for the compiler are described in "`.lcc86`" and "`options.lcc86`". The lcc86 sequentially searches the current directory and directories specified in environmental variable Home for `.lcc86`. Finally, the `lcc86` searches `/user/local/lib/lsic/options.lcc86`.

[End code]

0 is returned if the compilation is completed successfully. If an error is detected, 1 is returned. If a warning is detected, subsequent operation depends on -w option. 1 is returned if -w2 is specified, otherwise 0 is returned.

[Name]

libr86 - library compiler

[Syntax]

libr86 [-l] [-n] [-o *file*] [-s] [-x *lcc*] [*lcc86 option …*] *sourcefile*

[Comment]

The libr86 is a type of compiler driver that disassembles the gathered library module and combines them with compile and library files.

The name of a source code file in the library is specified for "sourcefile". Any name can be used, but normally an extension of ".l" is put.

The libr86 reads the input file and cuts each module. After that, it compiles and combines the compilation result with the library file. To perform actual compilation, the lcc86 is called up for execution. Modules are combined by the librarian oar. Therefore, it is absolutely necessary to put these commands in the executable directory.

The work file is made in the current directory and the library file is also created in the current directory with a name of "libr.out" put.

[Option]

-l	This option uses Microsoft's LIB instead of oar as a librarian.
-n	This option does not execute the compilation and creation of library, and displays only commands.
-o *file*	This option sets the name of an output library file to "*file*". If this option is not specified, the output file name is "libr.out".
-s	The libr86 displays lcc86 and oar commands, which are being executed internally, on the console. If this option is specified, they are not displayed.
-x *lcc*	This option uses *lcc* as a compiler driver. If this option is not specified, lcc86 is used as a compiler driver.

All options starting with "-" other than those shown above are transferred to the lcc86.

[Example]

```
libr86  -ms  libfile.l
```

The above example makes a small model library from "libfile.l". -ms is an option of the lcc86 that specifies a small model. Library file "libr.out" is created in the current directory.

[File]

`libr.out`

This is a library file created by the libr86 if -o option is not specified.

[End code]

0 is returned if the process is completed successfully.

[Reference]

`lcc86 (1), oar (1)`

[Name]

lld - link loader

[Syntax]

lld [@*file*] [-F*form*] [-I[X]] [-L*dir*] [-l*file*] [-g] [-M] [-o *file*] [-s *hex*] [-T[*seg*]] *hex*] *objfile*…

[Comment]

The lld connects files in Microsoft Object Module Format to create an EXE format file in the executable, core image, extended Intel HEX, or Motorola S2 format.

[Option]

@*file* This option reads command line parameters from "file".

-F*form* This option sets the output file format to "form". "form" must be any of the following.

e	MS-EXE format
c	COM format
h	Extended Intel HEX format
s	Motorola S2 format

If above option is not specified, the format is determined based on an extension of the output file.

.exe	e
.com	c
.hex	h

-I[X] This controls whether or not upper and lower case letters of the name are distinct.

-I, -I1	Upper and lower case letters are not distinct.
-I0	Upper and lower case letters are distinct.

If the above option is not specified, the command functions in the same manner as that -I0 is specified.

-L*dir* This option specifies a directory where library files exist.

-l*file* This option scans the library "file". ".lib" can be omitted.

-g This option creates line number information. When -g is specified, -M is specified automatically.

-M This option creates a map file.

-o *file* This option sets the output file name to "file". If this option is not specified, .exe file with the same base name as that of the object file specified first is created.

-s *hex* This option sets the stack size to "hex". A hexadecimal number is specified for "hex".

`-T` *`hex`* This sets the start address of the first segment to "`hex`". A hexadecimal number is specified for "`hex`".

`-T`*`seg hex`* This option specifies the start address of the segment "`seg`" to "`hex`". A hexadecimal number is specified for "`hex`".

[Example]

```
lld -o  foo.exe  foo.obj  bar.obj
```

This example combines object modules `foo.obj` and `bar.obj` to create `foo.exe`.

```
lld  foo.obj  bar.obj
```

This example functions in the same manner as described in the above example.

```
lld -o  foo.exe  bar.obj  zot.obj  -L/use/lib/c  -lclib
```

This example links `bar.obj` and `zot.obj`, and searches the library `/usr/lib/c/clib.lib` to make `foo.exe`.

```
lld -Fh -o  foo.hex  bar.obj  zot.obj
```

This example connects `bar.obj` and `zot.obj` to make extended Intel HEX file named `foo.hex`.

[End code]

0 is returned if the link is completed successfully. 1 is returned if an error is detected.

[Diagnosis message]

`Address already given: `*`segment`*

-T option is specified twice or more in the same segment "*`segment`*" or -T option is specified even though the address is specified by the SEGMENT command.

`file``: Bad module type`

"*`file`*" is not an object module created by the r86. The `lld` cannot handle this file correctly.

file: Conflicting class name: segment

Segments with the same name on multiple modules are declared with different classes.

file: Conflicting combine type: segment

Segments with the same name on multiple modules are declared with different combine types.

file: Conflicting group definition: segment

Segments with the same name on multiple modules belong to different groups.

file: Data before 0x100 (May be ORG 100H missing)

Program or data is found at a position before 0x100 while the COM file is being created. Don't you forget ORG 100H?

file: Device full

Disk is full while the file is being written.

file: Invalid object module: reason

Object module has an unexpected structure. The following may be the cause.

1. Extension is made which the lld cannot process.
2. Object module file is corrupted for some reason.
3. Language process system or lld, which creates this object, may have an unknown bug.

file: No such file or directory

File cannot be found.

file: Not a library file

File other than the library file is specified in -l option.

Out of memory

Symbol table overflows.

file (segment + offset): Overflow (base): name

Base address of the segment or symbol "name" cannot be put within 16 bits.

Total program size may exceed 1M bytes.

file (segment + offset): Overflow (byte): name

Value of symbol "name" cannot be put within 8 bits.

file (segment + offset): Overflow (word): name

Value of symbol "name" cannot be put within 16 bits. Segment size may exceed 64 Kbytes.

file: Redefined symbol: name

Symbol “name” defined in the file has already been defined in another file.

file (segment + offset): Too far (different segment): name

Destination symbol of the relative jump/call command does not belong to the same segment/group of that command. (Note: MS-LINK does not report this error. Program which runs in MS-LINK may cause an error in the `llb`. At this time, however, the same executable file as that in MS-LINK is created. If you know the reason, these errors can be disregarded even though we recommend you to correct the source program as much as possible.)

file (segment + offset): Too far (relative byte): name

Destination of the relative jump command is not in a range of -128 - +127.

Too small memory

Memory capacity is insufficient.

file: Undefined symbol: name

Symbol “name” to which the file refers is not defined.

OAR (1)

[Name]

`oar- librarian`

[Syntax]

`oar [-]d|q|r|t|x[cuv] [g N] library member-file…`

[Comment]

The `oar` is a program that controls the library to which the `lld` refers. The oar registers, deletes, picks up, and lists object modules. The library format is compatible with Microsoft's one.

An alphabetic character to be used for control is specified for the first parameter while the name of a library to be controlled is specified for the second parameter. If the extension of the library is not specified, an extension of "`.lib`" is automatically put.

In the "`member-file`", object module file names to be registered or deleted, which are separated by spaces, are specified. If the extension is omitted, an extension of "`.obj`" is automatically put.

The following operations are available.

`d`	This deletes "`member-file`" from "`library`".
`q`	This registers "`member-file`" to "`library`", but always adds it to the end. If the member with the same name already exists, "`member-file`" is not registered to "`library`". Since the processing time of this operation is shorter than that of "`r`", this operation is applicable when the members are registered one-by-one. Additionally, if a file with an extension of "`.lib`" is specified as "`member-file`", that library is connected.
`r`	This registers "`member-file`" to "`library`". If the member with the same name already exists in "`library`", it is overwritten.
`t`	This displays the list of "`library`" contents on the standard output.
`x`	This picks up "`member-file`" from "`library`". If "`member-file`" is not specified, all members are picked up.

[Option]

`@file`	This option reads a command from "`file`" and executes it.
`c`	This option controls whether or not the message "`library: created`" is displayed if a specified library does not exist during execution of r or q command. This option has no meaning if the library already exists. Additionally, if this option is specified for d, t, and x commands, it has no meaning.

g *N* This option specifies a page size of N in decimal. This option is effective only when creating a new library file.

v If this option is specified for d, r, q, and x commands, file names currently being executed are displayed. If this command is specified for t command, detailed information on each object module is displayed.

u When this option is specified for r command, only "`member-files`" having update time not later than that of "`library`" are registered. This option does not give effects on commands other than `r` command.

[Example]

```
oar tv /usr/lib/c/s/doslib.lib
```

The above example displays members in `/user/lib/c/s/doslib.lib`.

```
oar q  mathlib.lib  sin.obj  cos.obj
```

The above example adds sin.obj and cos.obj to mathlib.lib.

```
oar q all.lib  doslib.lib  mathlib.lib  knjlib.lib
```

The above example combines `doslib.lib`, `mathlib.lib`, and `knjlib.lib` to create `all.lib`.

```
oar r mathlib.lib  sin.obj  cos.obj
```

The above example replaces `sin.obj` and `cos.obj` in `mathlib.lib`.

```
oar ru mathlib.lib  *.obj
```

The above example registers `*.obj` files in the current directory, which are more newly created than `mathlib.lib`, to `mathlib.lib`.

```
oar xv romlib.lib
```

The above example picks up all the contents of `romlib.lib` to the current directory. These file names are displayed during execution of the command.

```
oar d romlib.lib  printf
```

The above example deletes `printf` from `romlib.lib`.

```
oar rcvg 64  newlib.lib  *.obj
```

The above example creates new library file `newlib.lib` with a page size of 64 and registers all `.obj` files in the current directory to it.

```
oar qg 64  newlib.lib  oldlib.lib
```

The above example creates a new library file in which the page size of `oldlib.lib` is changed to 64. The new library file name is `newlib.lib`.

```
oar r slib.lib  @list
```

The above example registers members listed in the file “list” to `slib.lib`. As shown in the above example, it is accepted that @file is written in the parameter.

R86 (1)

[Name]
r86 - 8086/80186/80286 assembler

[Syntax]
r86 [-m name] [-o outputfile] [-p listfile] [-q] inputfile

[Comment]
The r86 is an assembler for 8086/80186/80286.

Since the r86 aims at assembling of codes generated by the LSI C-86 compiler, it is not so complicate as Microsoft's MASM and is very simple. Additionally, the r86 also has an additional function that automatically converts conditional jump commands, which do not reach, into a long jump command not available in MASM. An input file is specified in `inputfile`. If the extension of the input file is omitted, it is assumed as "`.a86`". The r86 reads the input file and creates `.obj` files with the same name in the same directory. This object file has Microsoft Object Module Format.

[Option]

`-m name`	This option sets the name of a object module to "`name`". If this option is omitted, the module name is "`inputfile`".
`-o file`	This option creates an object in the file "`file`". If this option is omitted, the object file name is "`inputfile.obj`".
`-p file`	This option makes the listing in file "`file`". If this option is omitted, the listing is not made.
`-q`	This option changes all jump commands to long commands to make the assembling time shorter. If this option is omitted, jump commands are assembled into as short commands as possible.

[Example]
r86 -p test.lst test

The above example assembles `test.r86` to create object file `test.obj`. Additionally, the listing is written in `test.lst` at the same time. The module name is test (not `test.a86`) and jump commands are assembled into as short commands as possible.

```
r86 -m  foo -o  bar.obj  -q  zod.a86
```

The above command assembles `zod.a86`. The object file name is `bar.obj`. This example does not optimize jump commands. The module name is `foo`.

[End code]

0 is returned if the assembling is completed. 1 is returned if an error is detected.

[Diagnosis message]

`0 or more than 2 char`

The length of a character constant used in the formula is 0, or 3 or more. A character constant having 3 characters or more is allowed only in the DB command.

`bad number`

The numeric convention is incorrect. For example, a number other than 0 and 1 is used in the binary notation.

`bad operand type`

Operand type is unmatched.

`can't op`

op cannot be used here.

`double defined XXX`

XXX is defined twice or more.

`illegal END`

END command is found at unexpected position.

`illegal align or combine`

Align type or combine type is incorrect. These types may not match with previously declared ones.

`illegal class name XXX`

The class name XXX is incorrect. This class may not match with previous declaration.

`illegal extrn XXX`

The external symbol declaration may not match with the previous declaration.

`illegal group XXX`

The group declaration may not match with the previous declaration.

`illegal operand XXX`
: Operand, use of which is not allowed, is used.

`illegal public XXX`
: The public declaration may not match with the previous declaration.

`illegal type`
: The type is incorrect.

`line detected after END`
: It is not allowed to write lines after END.

`missing quote`
: There are no ‘ and “ that ends character constants.

`missing segment information`
: There is no segment information.

`missing type information`
: There is no type information.

`operand type doesn’t match`
: Operand type is not incorrect.

`operand unmatch`
: Operand is incorrect.

`overflow ESC value`
: Immediate value of the ESC command is too large.

`syntax error`
: A syntax error occurs.

`too long string`
: Character string is too long. The length of a character string must be 127 characters or less.

`undefined symbol XXX`
: XXX is not defined.

`unknown opecode XXX`
: XXX is not a correct command. There are no commands at locations where commands normally exist. Check also that a colon (:) is missing after the label.

`Warning: byte overflow`
: A value of one byte is in a range of -256 - +255.

[Bug]
Even though a large number is specified for the operand of the RS command, the r86 uses only the value of lower 16 bits. Therefore, the maximum size assigned by the RS command is 65535 bytes.